
SmartFusion2/IGLOO2 FPGA

Timing Constraints User's Guide

For Libero SoC v11.7 SP1



Table of Contents

Introduction	3
1 Using Synopsys Design Constraints	4
Object Access	4
Timing Assertions	5
Timing Exceptions	6
2 Timing Constraints and Design Flow	7
Timing Constraints for Synplify Revision 2Pro	7
SCOPE and Using the Forward Annotated SDC	12
Timing Constraints for Timing-Driven Place and Route	15
Improving Placer Performance	21
3 Constraints for SmartFusion2 and IGLOO2 IP Blocks	24
Oscillators	24
Fabric Clock Conditioning Circuit (CCC) for SmartFusion2 and IGLOO2	25
SERDES/DDR Configuration Subsystem (MSS/HPMS FIC_2)	27
CoreResetP False Paths (SmartFusion2 and IGLOO2 Only)	29
High Speed Serial Interface (SERDES) Block	34
4 Constraint Case Studies	37
Source-Synchronous Interface	37
Constraints and Combinational Paths	39
SmartFusion2 MSS and PCIe Design	44
MSS (TBI Interface) to SERDES (SmartFusion2 Only)	48
A Product Support	50
Customer Service	50
Customer Technical Support Center	50
Technical Support	50
Website	50
Contacting the Customer Technical Support Center	50
ITAR Technical Support	51

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 in order to achieve timing closure.

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

For SmartFusion2 and IGLOO2 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 5](#).

If timing performance is not met in the first iteration, you may consider setting additional and more advanced timing constraints in the second and subsequent iterations. See ["Timing Exceptions" on page 6](#).

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 SmartFusion2 and IGLOO2 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 that 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 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

Refer to "[Timing Constraints and Design Flow](#)" on page 7 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.

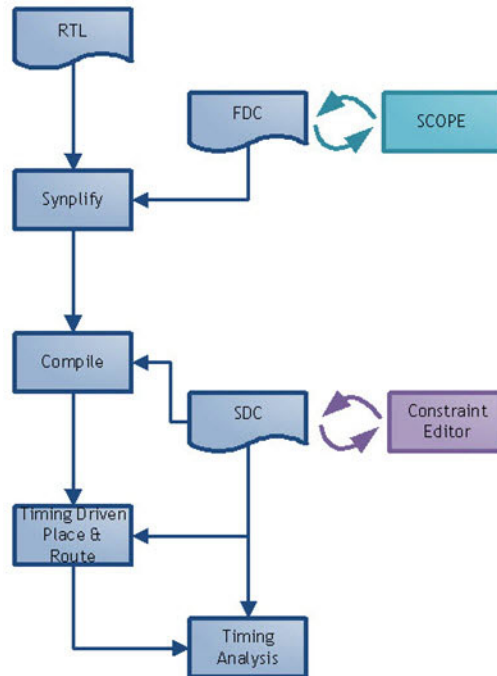


Figure 2-1 • Timing Constraints in the Design Flow

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 SmartFusion2 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).

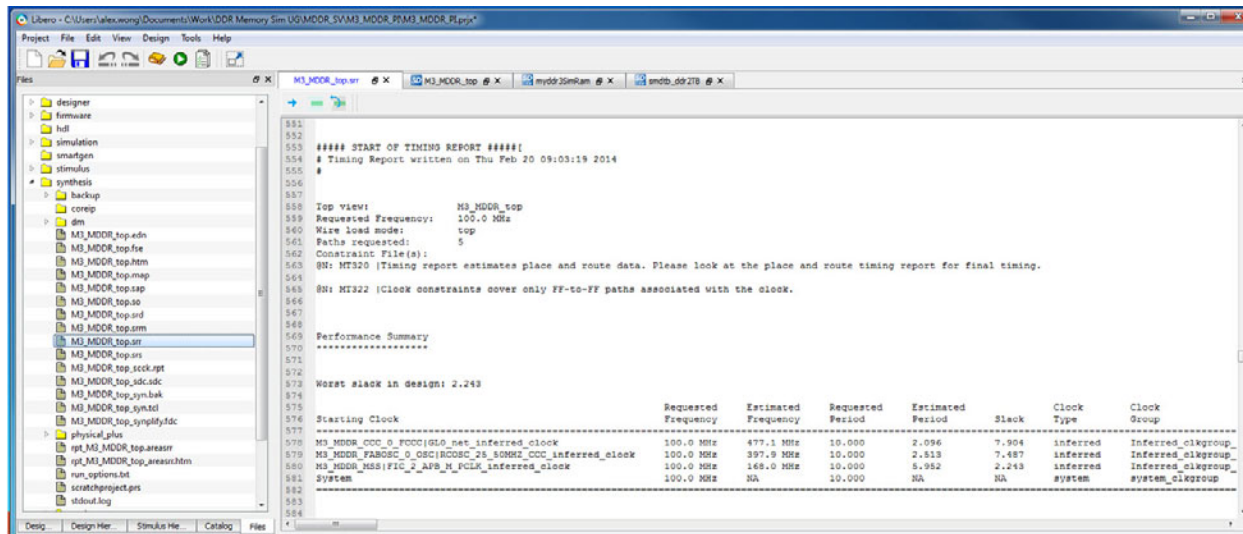


Figure 2-2 • Synplify Pro *.srr File Open in Libero SoC

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 15](#) section 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](#)).

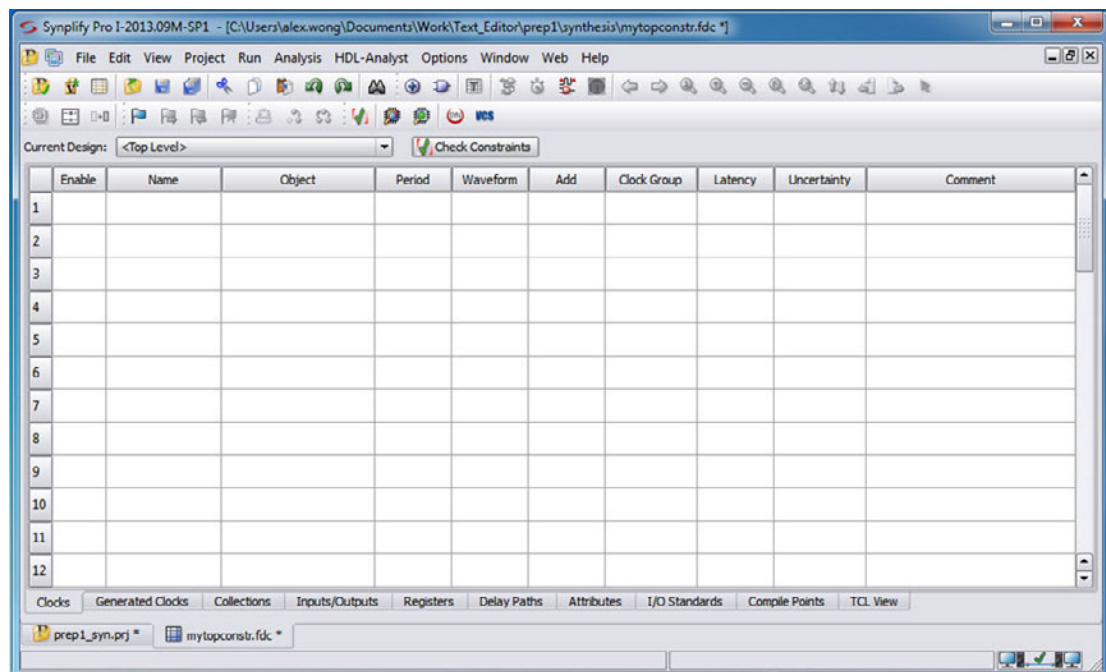


Figure 2-3 • SCOPE Constraints Editor

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: <i>design_name</i>	n/a	Modules
c: <i>clock_name</i>	get_clocks	Clocks
i: <i>instance_name</i>	get_cells	Instances
p: <i>port_name</i>	get_ports	Ports
t: <i>pin_name</i>	get_pins	Hierarchical ports or pins of instantiated cells
b: <i>bus_name</i>	get_pins	Bits of a bus
n: <i>net_name</i>	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](#).

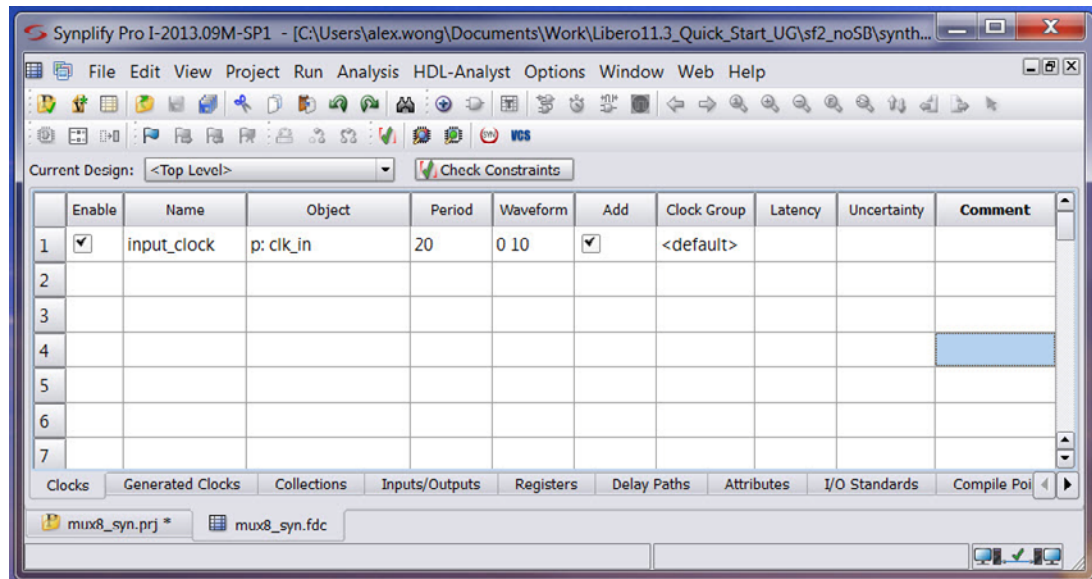


Figure 2-4 • Clocks Tab

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

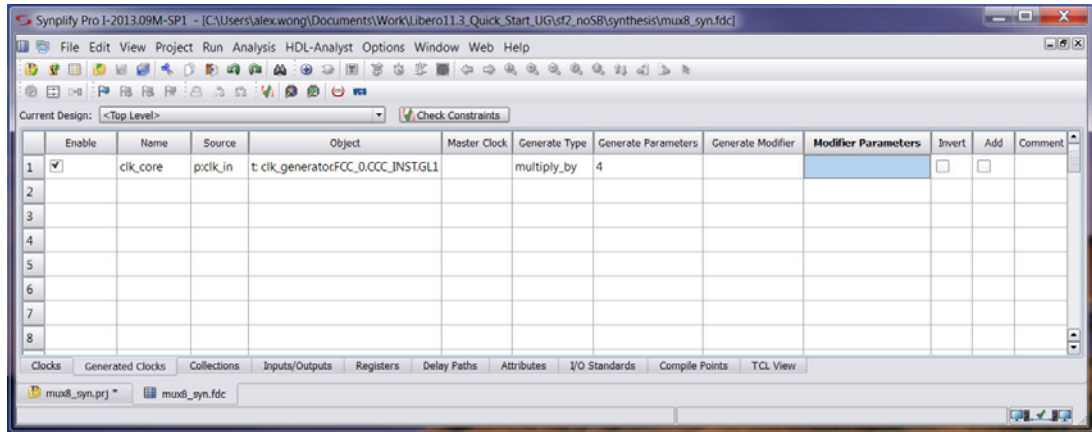


Figure 2-5 • Generated Clocks Tab

For SmartFusion2 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](#)).

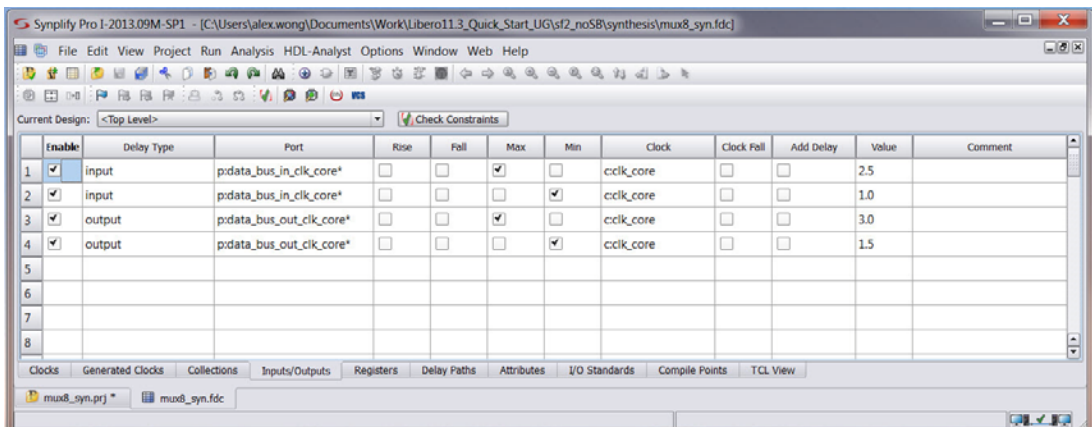


Figure 2-6 • Inputs/Outputs Tab

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](#)).

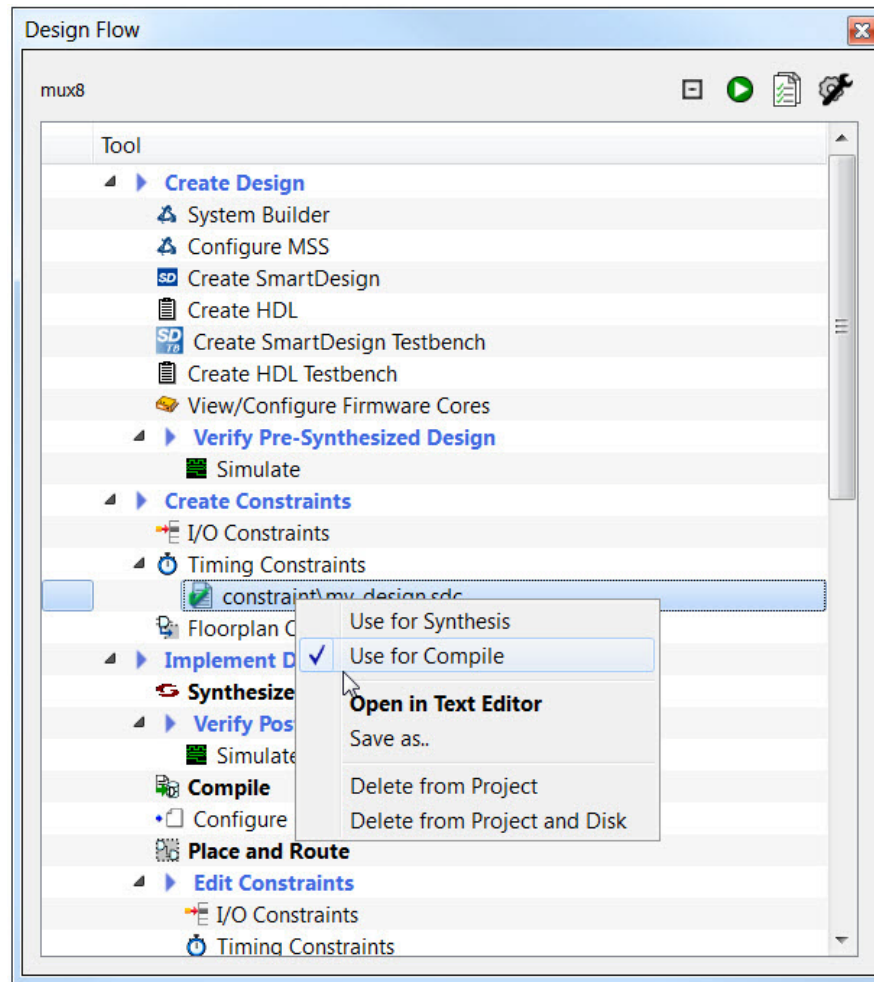


Figure 2-7 • SDC File for Compile

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).

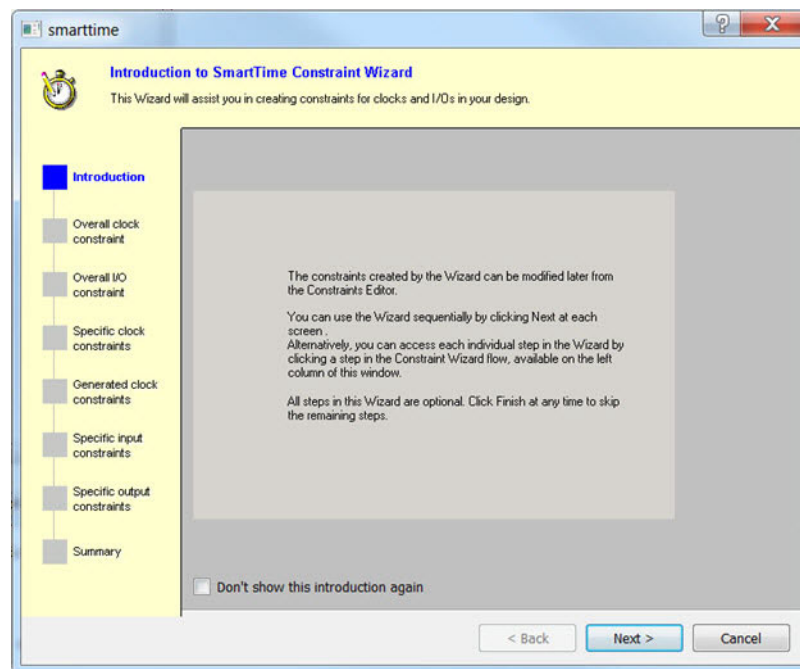


Figure 2-8 • SmartTime Constraint Wizard

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/O's.

The default constraints can be overridden by the Specific constraints for clocks and I/O's.

Clock Constraints

For SmartFusion2/IGLOO2 designs, SmartTime identifies and generates automatic constraints for:

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

Clocks from sources other than the on-chip oscillator or CCC must be defined by you using the Specific clock and Generated clock tabs.

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**).

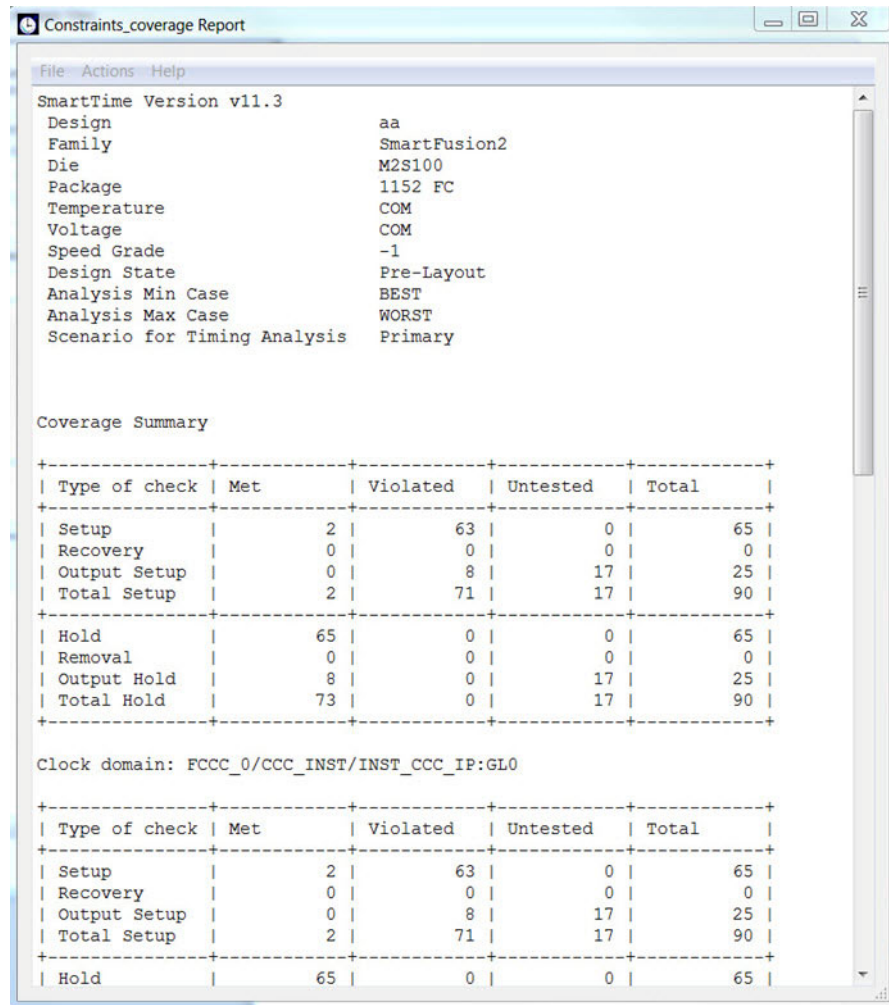


Figure 2-9 • Constraint Coverage Report

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 SmartFusion2.

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 on Primary scenario and select Clone (Figure 2-10).

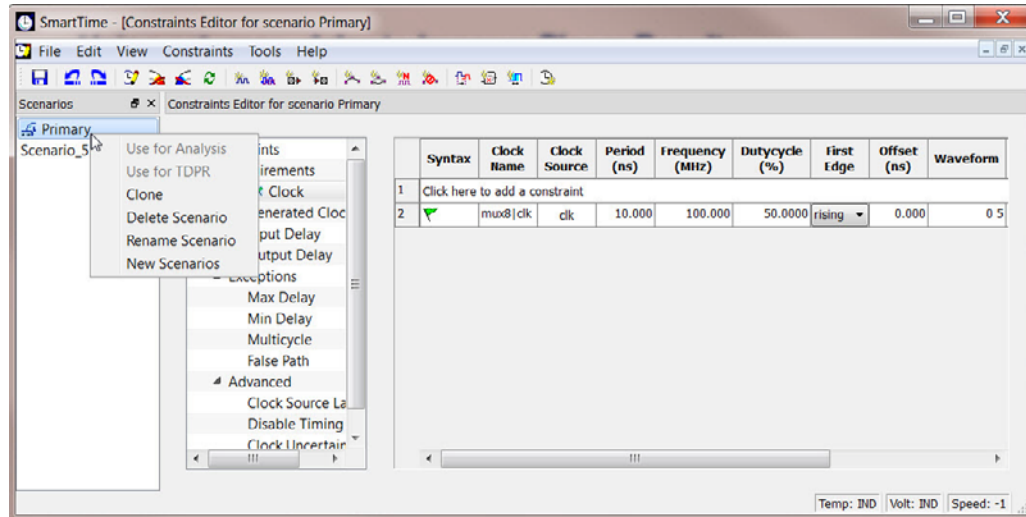


Figure 2-10 • Timing Scenario Clones

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

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

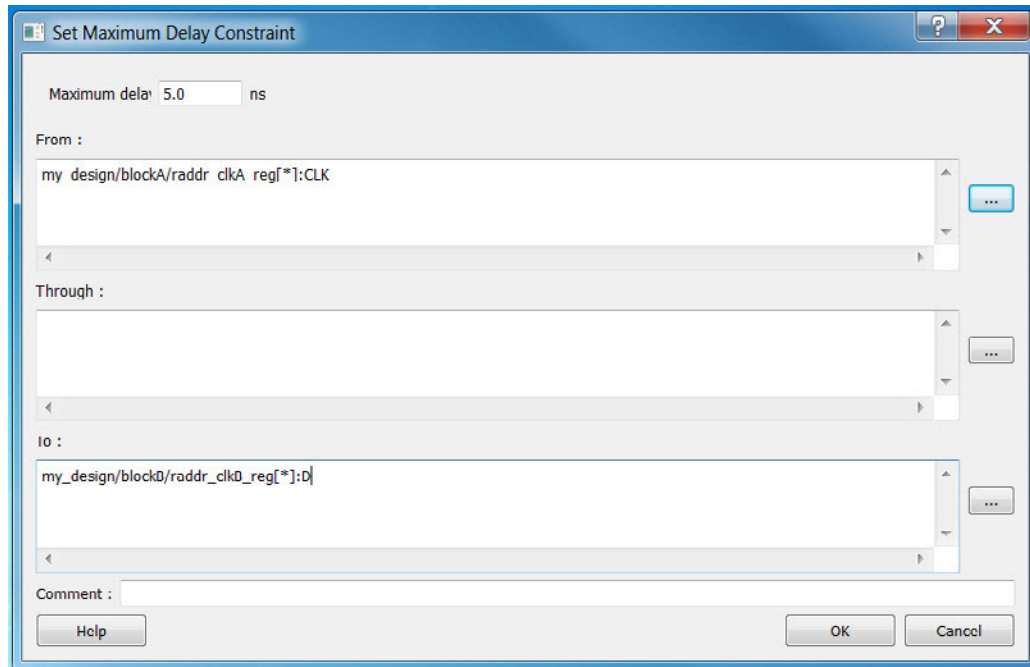


Figure 2-11 • Set_max_delay Constraints in Cloned Scenario

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

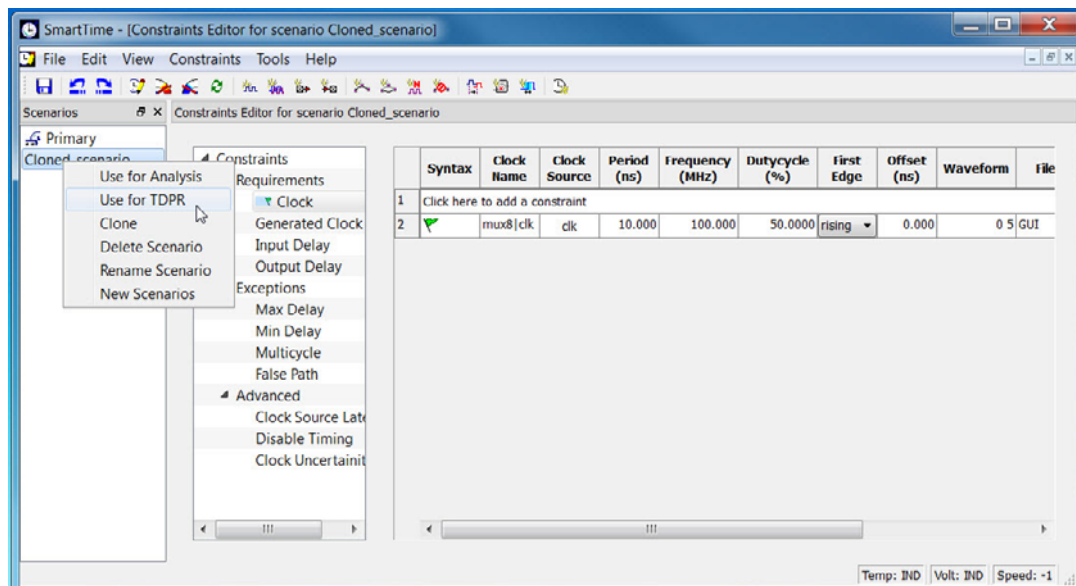


Figure 2-12 • Cloned Scenario for TDPR

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

3 – Constraints for SmartFusion2 and IGLOO2 IP Blocks

This chapter describes the constraint requirements for the following blocks:

- Oscillators
- Fabric Clock Conditioning Circuits (CCC)
- MSS (Microcontroller, SmartFusion2 only)
- High Speed Serial Interface (SERDES)

Oscillators

There are three oscillators available in SmartFusion2:

- External Main Crystal Oscillator that can be configured for frequencies between 32 kHz and 20 MHz.
- 25/50 MHz On-chip RC Oscillator
- 1 MHz On-chip RC Oscillator

The Chip Oscillator Configurator (Figure 3-1) invoked from within Libero enables you to select and configure the oscillator needed in the design. Upon configuration, the configurator generates a block for all oscillators.

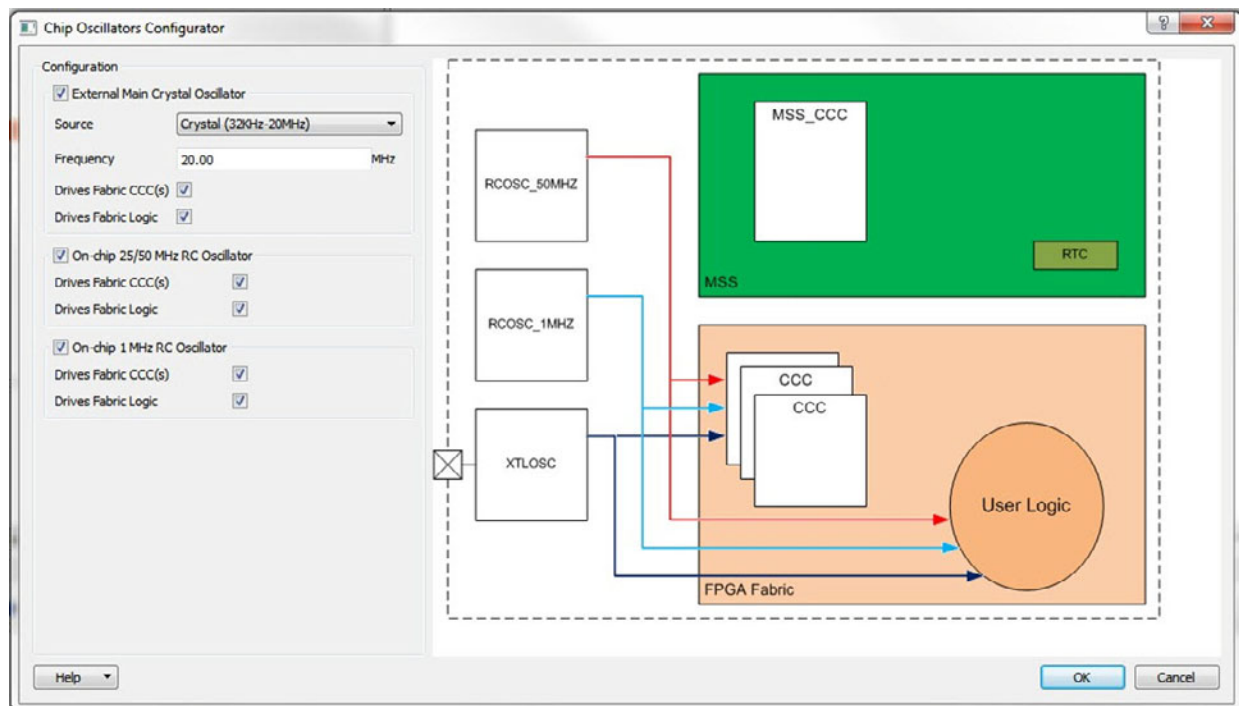


Figure 3-1 • Oscillator Configurator

Depending on the configuration, the oscillator IP provides up to two outputs per clock: one hardwired connection to the CCCs and one routed connection to the FPGA fabric.

Oscillator Synthesis Constraints

You must specify a clock constraint for each oscillator used by the design. The sources of the clock are the output pins of the oscillator. If an oscillator is used by both the CCC and the fabric, its clock constraint will have both outputs as sources.

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

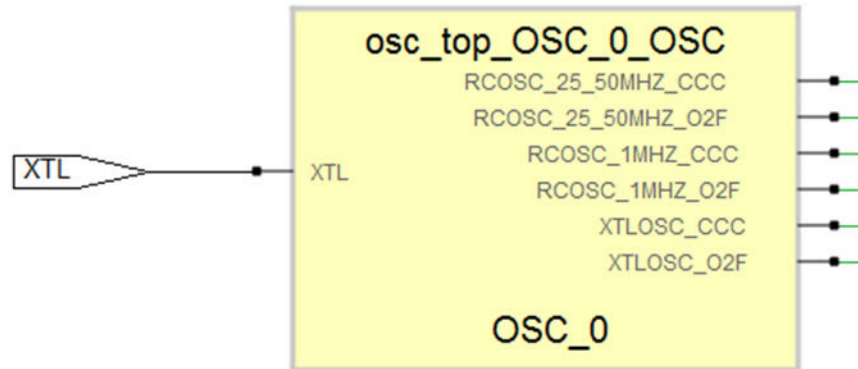


Figure 3-2 • Synthesis View of an Oscillator Block

The following constraints will work for an oscillator with OSC_0 as instance name with all outputs used. The Crystal oscillator is configured for 20 MHz.

```
create_clock -name osc_1MHz -period 1000 \
    [get_pins {OSC_0.RCOSC_1MHZ_O2F OSC_0.RCOSC_1MHZ_CCC}]
create_clock -name osc_50MHz -period 20 \
    [get_pins {OSC_0.RCOSC_25_50MHZ_O2F OSC_0.RCOSC_25_50MHZ_CCC}]
create_clock -name xtal_20MHz -period 50 \
    [get_pins {OSC_0.XTLOSC_O2F OSC_0.XTLOSC_CCC}]
```

Design Created with SystemBuilder

SystemBuilder instantiates an oscillator IP at least to use the 50 MHz clock for reset management (CoreResetP). SystemBuilder can also configure the other oscillators. The constraints needed in this case are similar to the one above. The instance name of the oscillator block depends on the name given to SystemBuilder for the system name. For example, for the system named my_system, the oscillator block will be instantiated as my_system_0.FABOSC_0. The clock constraint for the 50 MHz oscillator used by CoreResetP that should be set for synthesis is:

```
create_clock -name osc_50MHz -period 20 \
    [ get_pins {my_system_0.FABOSC_0.RCOSC_25_50MHZ_O2F}]
```

Oscillator Place and Route Constraints

No other constraints are needed. SmartTime automatically infers clock constraints based on the oscillator configurations.

Fabric Clock Conditioning Circuit (CCC) for SmartFusion2 and IGLOO2

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

Fabric CCC Synthesis Constraints

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

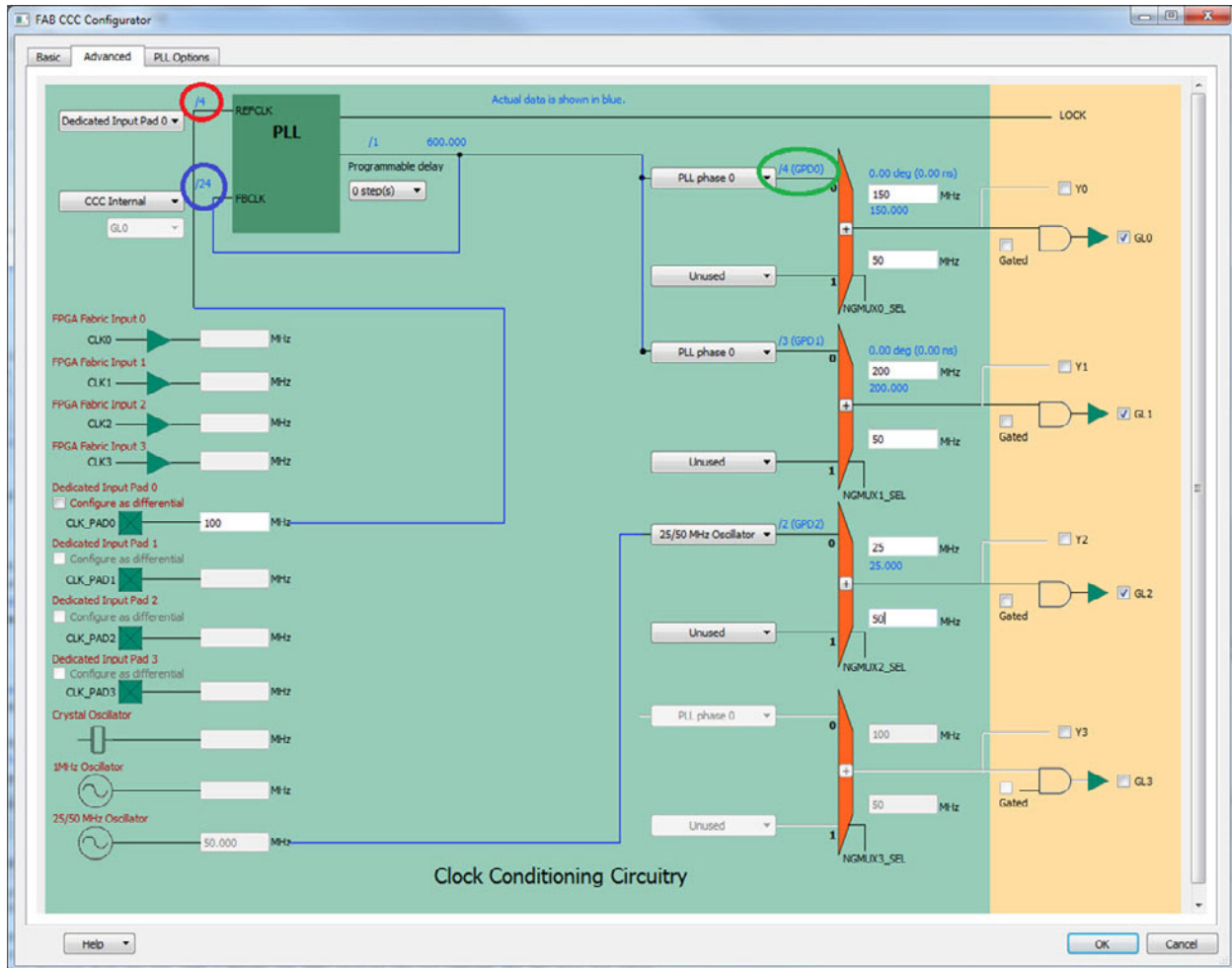


Figure 3-3 • FAB CCC Configurator Advanced Tab

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

- On GL0, a 150 MHz clock generated from the 100 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.

The exact division and multiplication factors can be calculated based on the divider configurations shown in the configurator. The ones 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 FCCC_CLK0_PAD -period 10 [get_pins {FCCC_0.CLK0_PAD}]
create_generated_clock -name clk_150mhz -divide_by 16 -multiply_by 24 \
-source [get_pins {FCCC_0.CLK0_PAD}] \
[get_pins {FCCC_0.GL0}]
create_generated_clock -name clk_200mhz -divide_by 12 -multiply_by 24 \
-source [get_pins {FCCC_0.CLK0_PAD}] \
```

```
[get_pins {FCCC_0.GL1}]
create_generated_clock -name clk_25mhz -divide_by 2 \
-source [get_pins {FCCC_0.OSC}] \
[get_pins {FCCC_0.GL2}]
```

Fabric CCC Place and Route Constraints

You need to add `create_clock` constraints for the input clock:

```
create_clock -name {CLK0_PAD} -period 10.000 -waveform { 0.000 5.000 } {CLK0_PAD}
```

SmartTime automatically infers clock constraints based on the `create_clock` and oscillator configurations.

SERDES/DDR Configuration Subsystem (MSS/HPMS FIC_2)

This section is relevant for designs using the MSS DDR (MDDR), Fabric DDR (FDDR) or SERDES blocks with the MSS FIC_2 interface for initialization. The MSS FIC_2 interface is essentially an APB3-like subsystem which initializes the peripherals at Power Up or on a Chip Level reset. The clock for this sub-system is generated by the MSS FIC_2 block and is defined as ¼ of the MSS (HPMS for IGLOO2) clock.

The following sections describe:

- Creating a clock constraint for FIC_2_APB_M_PCLK.
- Specifying timing requirements for FIC_2 to CoreConfigP interface.

Creating a Clock Constraint for FIC_2_APB_M_PCLK

The following example assumes a Cortex-M3 clocked at 100MHz. The FIC_2_APB_M_PCLK frequency is 25 MHz (¼ of the 100 MHz Cortex-M3 frequency).

Synthesis Timing Constraints

```
create_clock -name {FIC_2_APB_M_PCLK} -period 40 [get_pins \
{mss_system_0.mss_system_MSS_0:FIC_2_APB_M_PCLK}]
```

Note: The period of the clock needs to be four times the period of the MSS/HPMS_CLK.

Note: The name of the pin of the FIC_2_APB_M_PCLK clock is the hierarchical name of the pin in the RTL design.

Place and Route Timing Constraints

```
create_clock -name {FIC_2_APB_M_PCLK} -period 40 [get_pins \
{mss_system_0/mss_system_MSS_0/MSS_ADLIB_INST/INST_MSS_050_IP:CLK_CONFIG_APB}]
```

Note: The period of the clock needs to be four times the period of the MSS/HPMS_CLK.

Note: The name of the pin of the FIC_2_APB_M_PCLK clock is the hierarchical name of the pin in the flattened gate-level netlist created by Compile.

These constraints can also be entered using the SmartTime Constraints Editor. For details, refer to the SmartTime User's Guide (Libero > Help > Reference Manuals > SmartTime User's Guide for SmartFusion2, IGLOO2, and RTG4).

Specifying Timing Requirements for FIC_2 to CoreConfigP Interface

The configuration is performed through the FIC_2 to CoreConfigP interface. This interface has built-in re-timing to eliminate hold violations that may occur on the signals going from the MSS FIC_2 to CoreConfigP.

The following timing requirements are needed to capture the re-timing behavior. They must be manually entered using either an SDC file or the SmartTime Timing Constraints Editor. These constraints are for Static Timing Analysis (STA) and Timing Driven Place and Route (TDPR).

SDC (for Compile - STA and TDPR -)

For all M2S005*/M2GL005* and M2S010*/M2GL010* devices

```
set_min_delay -24 -from [ get_pins {*/INST_MSS_*_IP:CLK_CONFIG_APB} ]
set_max_delay 0 -from [ get_pins {*/INST_MSS_*_IP:CLK_CONFIG_APB} ] \
-through [ get_pins \
{*/INST_MSS_*_IP:PER2_FABRIC_PENABLE */INST_MSS_*_IP:PER2_FABRIC_PSEL} ] \
-to [ get_pins {*/FIC_2_APB_M_PREADY* */state[0]:D} ]
```

For all other devices

```
set_min_delay -24 -from [ get_pins {*/INST_MSS_*_IP:CLK_CONFIG_APB} ]
set_max_delay 0 -from [ get_pins {*/INST_MSS_*_IP:CLK_CONFIG_APB} ] \
-through [ get_pins \
{*/INST_MSS_*_IP:PER2_FABRIC_PENABLE */INST_MSS_*_IP:PER2_FABRIC_PSEL} ] \
-to [ get_pins {*/FIC_2_APB_M_PREADY* */state[0]:D} ]
```

Note: The names of the signals in the constraints use the hierarchical names of the flattened gate-level netlist created by Compile.

These constraints may also be entered using the SmartTime Graphical User Interface, although it is recommended to use an SDC file.

When using the SmartTime GUI, note that negative min-delay values cannot be entered in the min-delay constraint dialog. Only positive values are accepted ([Figure 3-4](#)).

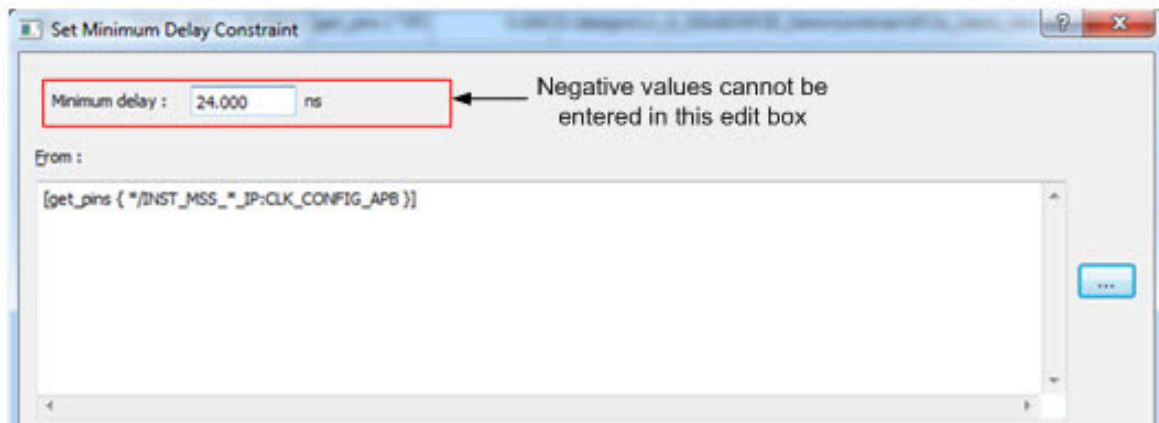
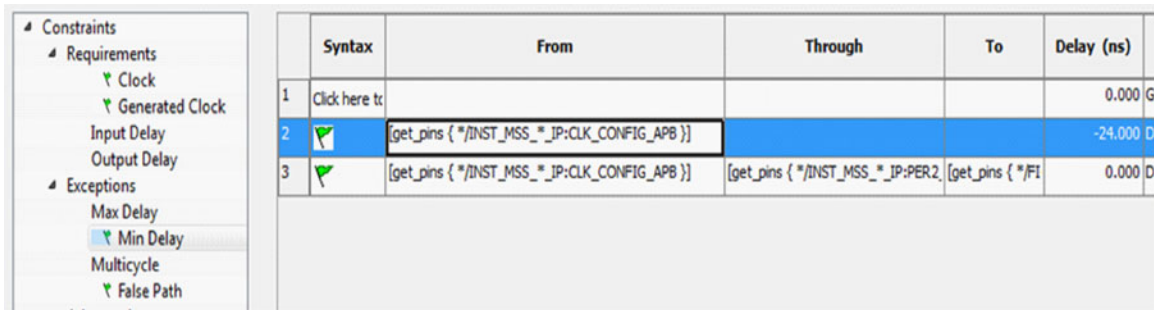


Figure 3-4 • Set Minimum Delay Constraint Dialog Box

You may, however, enter the negative Minimum Delay value in the Min Delay Constraint Table (Figure 3-5).

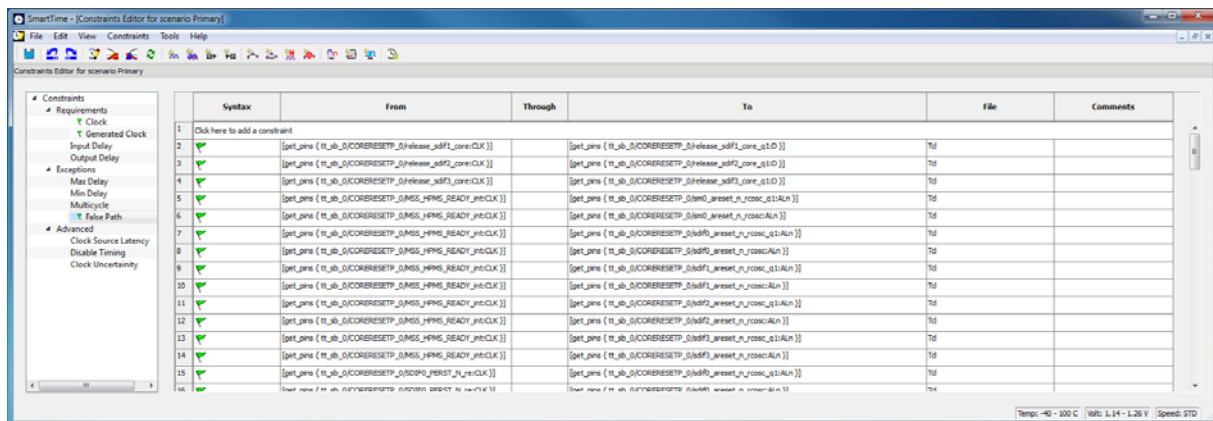


	Syntax	From	Through	To	Delay (ns)
1	Click here to				0.000 D:
2	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1	-24.000 D:
3	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}		[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		0.000 D:

Figure 3-5 • Minimum Delay Constraint Table

CoreResetP False Paths (SmartFusion2 and IGLOO2 Only)

CoreResetP is a soft IP Configurator Core to manage the reset circuitry of your FDDR, MDDR and SERDES IF Blocks. CoreResetP is instantiated by System Builder to handle the reset and initialization of peripherals. Some timing paths inside the CoreResetP block may cause hold time violations to be reported. These are false paths and should be excluded from timing analysis. Libero automatically identifies these paths and sets the false path constraints on them during the Compile phase in the Design Flow. When you open the SmartTime Constraints Editor, you will see the false path constraints in the False Path group of the SmartTime Constraints Editor (**SmartTime > Constraints > Exceptions > False Path**). See Figure 3-6. You do not need to take any further action to exclude these paths from timing analysis..



	Syntax	From	Through	To	File	Comments
1	Click here to add a constraint					
2	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
3	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
4	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
5	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
6	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
7	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
8	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
9	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
10	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
11	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
12	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
13	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
14	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
15	[get_pins { */INST_MSS_*_IP:CLK_CONFIG_APB }}			[get_pins { */INST_MSS_*_IP:PER2, [get_pins { */F1		
16	Total time for all constraints: 0.000 ns					

Figure 3-6 • False Path Constraints in CoreResetP Block

Table 3-1 lists all the potential false paths (Verilog names) inside the CoreResetP Core.

Table 3-1 • Potential False Paths (Verilog Names) inside the CoreResetP Core

From (Source Pin)	To (Sink Pin)
{/ddr_settled:CLK}	{/ddr_settled_q1:D}
{/release_sdif0_core:CLK}	{/release_sdif0_core_q1:D}
{/release_sdif1_core:CLK}	{/release_sdif1_core_q1:D}

Table 3-1 • Potential False Paths (Verilog Names) inside the CoreResetP Core (continued)

From (Source Pin)	To (Sink Pin)
{/release_sdif2_core:CLK}	{/release_sdif2_core_q1:D}
{/release_sdif3_core:CLK}	{/release_sdif3_core_q1:D}
{/MSS_HPMS_READY_int:CLK}	{/sm0_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sm0_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif0_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif0_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif1_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif1_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif2_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif2_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif3_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif3_areset_n_rcosc:ALn}
{/SDIF0_PERST_N_re:CLK}	{/sdif0_areset_n_rcosc_q1:ALn}
{/SDIF0_PERST_N_re:CLK}	{/sdif0_areset_n_rcosc:ALn}
{/SDIF1_PERST_N_re:CLK}	{/sdif1_areset_n_rcosc_q1:ALn}
{/SDIF1_PERST_N_re:CLK}	{/sdif1_areset_n_rcosc:ALn}
{/SDIF2_PERST_N_re:CLK}	{/sdif2_areset_n_rcosc_q1:ALn}
{/SDIF2_PERST_N_re:CLK}	{/sdif2_areset_n_rcosc:ALn}
{/SDIF3_PERST_N_re:CLK}	{/sdif3_areset_n_rcosc_q1:ALn}
{/SDIF3_PERST_N_re:CLK}	{/sdif3_areset_n_rcosc:ALn}
{/count_sdif0_enable:CLK}	{/count_sdif0_enable_q1:D}
{/count_sdif1_enable:CLK}	{/count_sdif1_enable_q1:D}
{/count_sdif2_enable:CLK}	{/count_sdif2_enable_q1:D}
{/count_sdif3_enable:CLK}	{/count_sdif3_enable_q1:D}
{/count_ddr_enable:CLK}	{/count_ddr_enable_q1:D}
{/SDIF0_CORE_RESET_N_0:CLK}	{/genblk2.sdif0_phr/reset_n_q1:ALn}
{/SDIF0_CORE_RESET_N_0:CLK}	{/genblk2.sdif0_phr/reset_n_clk_ltssm:ALn}
{/SDIF1_CORE_RESET_N_0:CLK}	{/genblk3.sdif1_phr/reset_n_q1:ALn}
{/SDIF1_CORE_RESET_N_0:CLK}	{/genblk3.sdif1_phr/reset_n_clk_ltssm:ALn}
{/SDIF2_CORE_RESET_N_0:CLK}	{/genblk4.sdif2_phr/reset_n_q1:ALn}
{/SDIF2_CORE_RESET_N_0:CLK}	{/genblk4.sdif2_phr/reset_n_clk_ltssm:ALn}
{/SDIF3_CORE_RESET_N_0:CLK}	{/genblk5.sdif3_phr/reset_n_q1:ALn}
{/SDIF3_CORE_RESET_N_0:CLK}	{/genblk5.sdif3_phr/reset_n_clk_ltssm:ALn}
{/genblk2.sdif0_phr/hot_reset_n:CLK}	{/genblk2.sdif0_phr/sdif_core_reset_n_q1:ALn}
{/genblk2.sdif0_phr/hot_reset_n:CLK}	{/genblk2.sdif0_phr/sdif_core_reset_n:ALn}
{/genblk3.sdif1_phr/hot_reset_n:CLK}	{/genblk3.sdif1_phr/sdif_core_reset_n_q1:ALn}

Table 3-1 • Potential False Paths (Verilog Names) inside the CoreResetP Core (continued)

From (Source Pin)	To (Sink Pin)
{/genblk3.sdif1_phr/hot_reset_n:CLK}	{/genblk3.sdif1_phr/sdif_core_reset_n:ALn}
{/genblk4.sdif2_phr/hot_reset_n:CLK}	{/genblk4.sdif2_phr/sdif_core_reset_n_q1:ALn}
{/genblk4.sdif2_phr/hot_reset_n:CLK}	{/genblk4.sdif2_phr/sdif_core_reset_n:ALn}
{/genblk5.sdif3_phr/hot_reset_n:CLK}	{/genblk5.sdif3_phr/sdif_core_reset_n_q1:ALn}
{/genblk5.sdif3_phr/hot_reset_n:CLK}	{/genblk5.sdif3_phr/sdif_core_reset_n:ALn}
{* }	{/genblk2.sdif0_phr/ltssm_q1[0]:D}
{* }	{/genblk2.sdif0_phr/ltssm_q1[1]:D}
{* }	{/genblk2.sdif0_phr/ltssm_q1[2]:D}
{* }	{/genblk2.sdif0_phr/ltssm_q1[3]:D}
{* }	{/genblk2.sdif0_phr/ltssm_q1[4]:D}
{* }	{/genblk2.sdif0_phr/psel_q1:D}
{* }	{/genblk2.sdif0_phr/pwrite_q1:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[0]:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[1]:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[2]:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[3]:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[4]:D}
{* }	{/genblk3.sdif1_phr/psel_q1:D}
{* }	{/genblk3.sdif1_phr/pwrite_q1:D}
{* }	{/genblk3.sdif1_phr/ltssm_q1[0]:D}
{* }	{/genblk4.sdif2_phr/ltssm_q1[1]:D}
{* }	{/genblk4.sdif2_phr/ltssm_q1[2]:D}
{* }	{/genblk4.sdif2_phr/ltssm_q1[3]:D}
{* }	{/genblk4.sdif2_phr/ltssm_q1[4]:D}
{* }	{/genblk4.sdif2_phr/psel_q1:D}
{* }	{/genblk4.sdif2_phr/pwrite_q1:D}
{* }	{/genblk5.sdif3_phr/ltssm_q1[0]:D}
{* }	{/genblk5.sdif3_phr/ltssm_q1[1]:D}
{* }	{/genblk5.sdif3_phr/ltssm_q1[2]:D}
{* }	{/genblk5.sdif3_phr/ltssm_q1[3]:D}
{* }	{/genblk5.sdif3_phr/ltssm_q1[4]:D}
{* }	{/genblk5.sdif3_phr/psel_q1:D}
{* }	{/genblk5.sdif3_phr/pwrite_q1:D}

Table 3-2 lists all the potential false paths (VHDL names) inside the CoreResetP Core.

Table 3-2 • Potential False Paths (VHDL Names) inside the CoreResetP Core

From (Source Pin)	To (Sink Pin)
{/ddr_settled:CLK}	{/ddr_settled_q1:D}
{/release_sdif0_core:CLK}	{/release_sdif0_core_q1:D}
{/release_sdif1_core:CLK}	{/release_sdif1_core_q1:D}
{/release_sdif2_core:CLK}	{/release_sdif2_core_q1:D}
{/release_sdif3_core:CLK}	{/release_sdif3_core_q1:D}
{/MSS_HPMS_READY_int:CLK}	{/sm0_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sm0_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif0_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif0_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif1_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif1_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif2_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif2_areset_n_rcosc:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif3_areset_n_rcosc_q1:ALn}
{/MSS_HPMS_READY_int:CLK}	{/sdif3_areset_n_rcosc:ALn}
{/SDIF0_PERST_N_re:CLK}	{/sdif0_areset_n_rcosc_q1:ALn}
{/SDIF0_PERST_N_re:CLK}	{/sdif0_areset_n_rcosc:ALn}
{/SDIF1_PERST_N_re:CLK}	{/sdif1_areset_n_rcosc_q1:ALn}
{/SDIF1_PERST_N_re:CLK}	{/sdif1_areset_n_rcosc:ALn}
{/SDIF2_PERST_N_re:CLK}	{/sdif2_areset_n_rcosc_q1:ALn}
{/SDIF2_PERST_N_re:CLK}	{/sdif2_areset_n_rcosc:ALn}
{/SDIF3_PERST_N_re:CLK}	{/sdif3_areset_n_rcosc_q1:ALn}
{/SDIF3_PERST_N_re:CLK}	{/sdif3_areset_n_rcosc:ALn}
{/count_sdif0_enable:CLK}	{/count_sdif0_enable_q1:D}
{/count_sdif1_enable:CLK}	{/count_sdif1_enable_q1:D}
{/count_sdif2_enable:CLK}	{/count_sdif2_enable_q1:D}
{/count_sdif3_enable:CLK}	{/count_sdif3_enable_q1:D}
{/count_ddr_enable:CLK}	{/count_ddr_enable_q1:D}
{/SDIF0_CORE_RESET_N_0:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/reset_n_q1:ALn}
{/SDIF0_CORE_RESET_N_0:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/reset_n_clk_lts sm:ALn}
{/SDIF1_CORE_RESET_N_0:CLK}	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/reset_n_q1:ALn}
{/SDIF1_CORE_RESET_N_0:CLK}	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/reset_n_clk_lts sm:ALn}

Table 3-2 • Potential False Paths (VHDL Names) inside the CoreResetP Core (continued)

From (Source Pin)	To (Sink Pin)
{/SDIF2_CORE_RESET_N_0:CLK}	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/reset_n_q1:ALn}
{/SDIF2_CORE_RESET_N_0:CLK}	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/reset_n_clk_itsm:ALn}
{/SDIF3_CORE_RESET_N_0:CLK}	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/reset_n_q1:ALn}
{/SDIF3_CORE_RESET_N_0:CLK}	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/reset_n_clk_itsm:ALn}
{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/hot_reset_n:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/sdif_core_reset_n_q1:ALn}
{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/hot_reset_n:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/sdif_core_reset_n:ALn}
{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/hot_reset_n:CLK}	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/sdif_core_reset_n_q1:ALn}
{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/hot_reset_n:CLK}	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/sdif_core_reset_n:ALn}
{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/hot_reset_n:CLK}	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/sdif_core_reset_n_q1:ALn}
{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/hot_reset_n:CLK}	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/sdif_core_reset_n:ALn}
{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/hot_reset_n:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[0]:D}
{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/hot_reset_n:CLK}	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[1]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[0]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[1]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[2]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[3]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/ltssm_q1[4]:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/psel_q1:D}
{* }	{/SDIF0_HR_FIX_INCLUDED.sdif0_phr/pwrite_q1:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/ltssm_q1[0]:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/ltssm_q1[1]:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/ltssm_q1[2]:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/ltssm_q1[3]:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/ltssm_q1[4]:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/psel_q1:D}
{* }	{/SDIF1_HR_FIX_INCLUDED.sdif1_phr/pwrite_q1:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/ltssm_q1[0]:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/ltssm_q1[1]:D}

Table 3-2 • Potential False Paths (VHDL Names) inside the CoreResetP Core (continued)

From (Source Pin)	To (Sink Pin)
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/ltssm_q1[2]:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/ltssm_q1[3]:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/ltssm_q1[4]:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/psel_q1:D}
{* }	{/SDIF2_HR_FIX_INCLUDED.sdif2_phr/pwrite_q1:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/ltssm_q1[0]:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/ltssm_q1[1]:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/ltssm_q1[2]:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/ltssm_q1[3]:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/ltssm_q1[4]:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/psel_q1:D}
{* }	{/SDIF3_HR_FIX_INCLUDED.sdif3_phr/pwrite_q1:D}

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.

XAUI Protocol Mode

In XAUI mode, the SERDESIF block uses four clocks:

- APB_S_CLK for the APB3 configuration bus
- XAUI_MMD_MDC, the MDIO interface clock. In SmartTime, this clock appears as S_AWADDR_HADDR[18] as the physical implementation re-use pins from the AXI/AHBL interface unused in XAUI.
- XAUI_RX_CLK. Received data are synchronized to this clock.
- XAUI_OUT_CLK. Transmitted data are sampled with this clock.

APB_S_CLK and XAUI_MMD_MDC clock must be defined at their source (MSS for APB_S_CLK).

XAUI_RX_CLK and XAUI_OUT_CLK clocks may be defined on the output port of the SERDESIF block. The example below creates these clocks for a SERDESIF block instantiated as XAUI_0.

XAUI Synthesis Constraints

```
create_clock -name XAUI_RX_CLK -period 6.4 \  
    [get_pins {XAUI_0.SERDESIF_INST.EPCS_RXCLK_0}]  
create_clock -name XAUI_OUT_CLK -period 6.4 \  
    [get_pins {XAUI_0.SERDESIF_INST.XAUI_OUT_CLK}]
```

XAUI Place and Route Constraints

```
create_clock -name XAUI_RX_CLK -period 6.4 \  
    [get_pins {XAUI_0/SERDESIF_INST:EPCS_RXCLK_0}]  
create_clock -name XAUI_OUT_CLK -period 6.4 \  
    [get_pins {XAUI_0/SERDESIF_INST:XAUI_OUT_CLK }]
```

EPCS Protocol Mode

In EPCS mode, the SERDESIF can support up to four lanes. Two clocks are generated for each lane: RX and TX clocks. The example below creates these clocks for a SERDESIF block instantiated as EPCS_0 using four lanes.

EPCS Protocol Synthesis Constraints

```
create_clock -name EPCS_0_RX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_RXCLK_0}]  
create_clock -name EPCS_0_TX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_TXCLK_0}]  
create_clock -name EPCS_1_RX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_RXCLK_1}]  
create_clock -name EPCS_1_TX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_TXCLK_1}]  
create_clock -name EPCS_2_RX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_RXCLK[0]}]  
create_clock -name EPCS_2_TX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_TXCLK[0]}]  
create_clock -name EPCS_3_RX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_RXCLK[1]}]  
create_clock -name EPCS_3_TX_CLK -period 8 \  
    [get_pins {EPCS_0.SERDESIF_INST.EPCS_TXCLK[1]}]
```

EPCS Protocol Place and Route Constraints

```
create_clock -name EPCS_0_RX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_RXCLK_0}]  
create_clock -name EPCS_0_TX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_TXCLK_0}]  
create_clock -name EPCS_1_RX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_RXCLK_1}]  
create_clock -name EPCS_1_TX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_TXCLK_1}]  
create_clock -name EPCS_2_RX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_RXCLK[0]}]  
create_clock -name EPCS_2_TX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_TXCLK[0]}]  
create_clock -name EPCS_3_RX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_RXCLK[1]}]  
create_clock -name EPCS_3_TX_CLK -period 8 \  
            [get_pins {EPCS_0/SERDESIF_INST:EPCS_TXCLK[1]}]
```

4 – Constraint Case Studies

This chapter has case studies for:

- Source-Synchronous Interface
- Constraints and Combinational Paths
- SmartFusion2 MSS and PCIe

Source-Synchronous Interface

Source-synchronous interfaces are commonly used for high-speed data transfer. SPI, DDR are standard examples of source-synchronous interfaces. In a source-synchronous interface, the clock used to synchronize the data is provided by one of the actors. [Figure 4-1](#) shows a basic example of a source-synchronous interface with the clock provided by the transmitter.

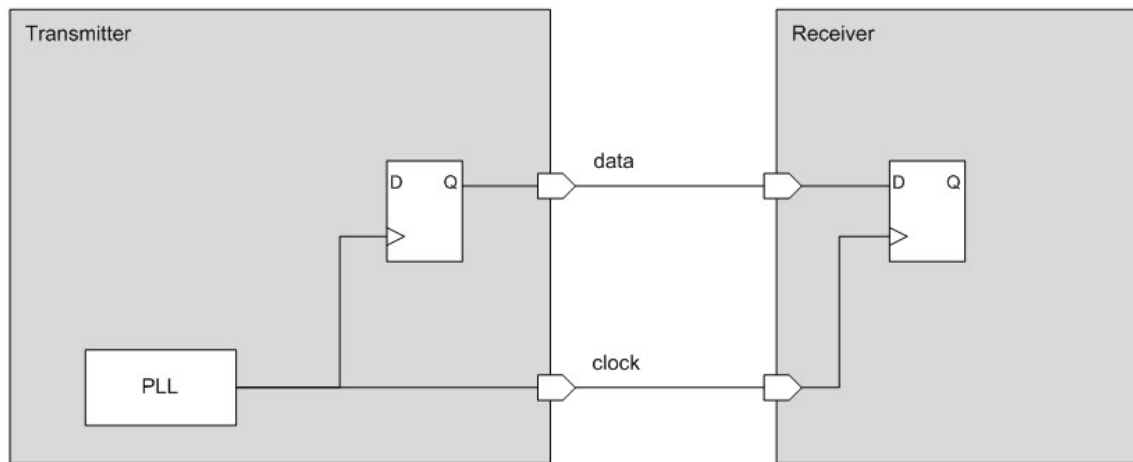


Figure 4-1 • Basic Source-Synchronous Transmitter

Source Synchronous Interface Design Example

The design in Figure 4-2 shows the constraints needed for a source synchronous interface. This design has both input and output data synchronous to an output clock. The clock is generated from an oscillator by a PLL multiplying the clock by two.

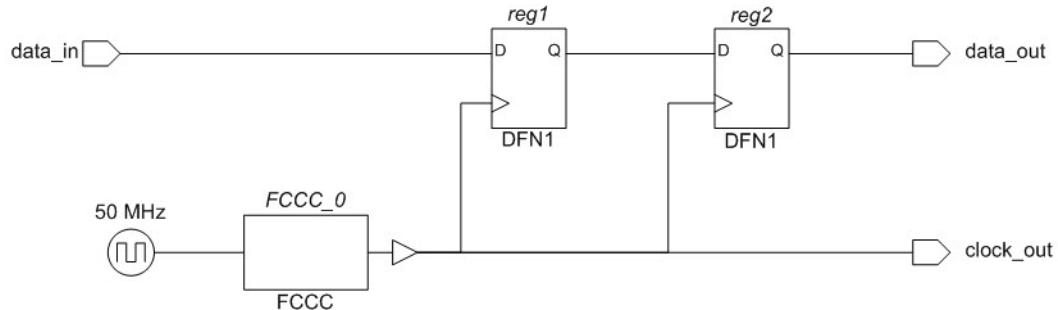


Figure 4-2 • Source-Synchronous Interface Design Example

The following constraints define the three clocks needed for the design:

- One at the output of the oscillator.
- One at the output of the PLL generated from the oscillator.
- One on the output port, copy of the PLL clock, for the source-synchronous interface.

```
create_clock -name OSC_50MHz -period 20 [get_pins {OSC_0.RCOSC_25_50MHZ_CCC}]

create_generated_clock -name PLL_100MHz -multiply_by 2 \
    -source [get_pins {FCCC_0.OSC}] \
    [get_pins {FCCC_0.GL0}]

create_generated_clock -name clock_out -divide_by 1 \
    -source [get_pins {FCCC_0.GL0}] \
    [get_ports {clock_out}]
```

For output data, output delays define the requirements with respect to the output clock. The following constraints specify that the data needs to be valid at the data_out port 2 ns before the clock edge and 0.3 ns after.

```
set_output_delay -max 2 -clock clock_out [get_ports {data_out}]
set_output_delay -min -0.3 -clock clock_out [get_ports {data_out}]
```

For input data, input delays define the requirements with respect to the output clock. The following constraints specify that the external logic will take between 1.3 ns and 3.0 ns to send the data.

```
set_input_delay -max 3.0 -clock clock_out [get_ports {data_in}]
set_input_delay -min 1.3 -clock clock_out [get_ports {data_in}]
```

Place and Route Constraints

The clock constraints for the oscillator and the generated clocks for the CCC are automatically inferred by SmartTime. The user should add the same input and output delay constraints.

Constraints and Combinational Paths

This case below illustrates that constraints (in this case, input delays) set for synchronous paths may impact combinational paths.

In the design shown in [Figure 4-3](#), input data_in is used by both a synchronous path to flip-flop reg1 and a combinational path to data_out.

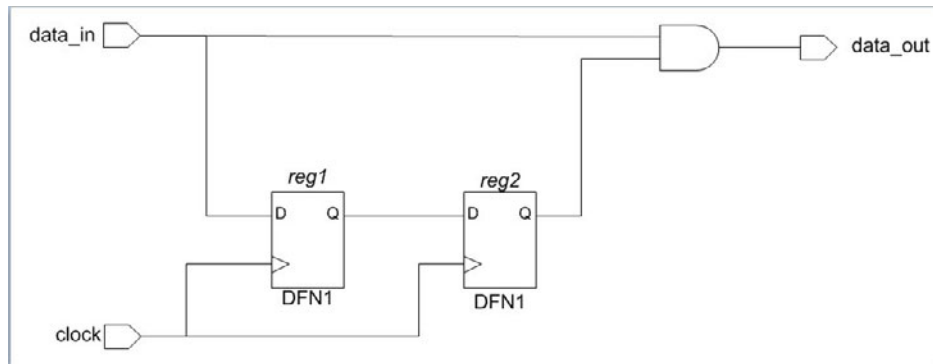


Figure 4-3 • Constraints and Asynchronous Paths Design Example

To specify the requirements for the combinational path, a max delay can be used. For example, to specify that the path between data_in and data_out should be shorter than 7 ns, the following max-delay can be used:

```
set_max_delay 7 -from [get_ports {data_in}] -to [get_ports {data_out}]
```

The timing analysis of this path is shown in [Table 4-1](#).

Table 4-1 • Combinational Path Max Delay Example

From: data_in			
To: data_out			
		data required time	7.000
		data arrival time -	6.745
		slack	0.255
<hr/>			
Data arrival time calculation			
0.000		data_in (r)	
	+	0.000	net: data_in
0.000		data_in_ibuf/U0/U_IOPAD:PAD (r)	
	+	1.802	cell: ADLIB:IOPAD_IN
1.802		data_in_ibuf/U0/U_IOPAD:Y (r)	
	+	0.000	net: data_in_ibuf/U0/YIN1
1.802		data_in_ibuf/U0/U_IOINFF:A (r)	
	+	0.083	cell: ADLIB:IOINFF_BYPASS
1.885		data_in_ibuf/U0/U_IOINFF:Y (r)	
	+	0.485	net: data_in_c
2.370		AND2_0:A (r)	
	+	0.163	cell: ADLIB:CFG2
2.533		AND2_0:Y (r)	
	+	0.692	net: data_out_c
3.225		data_out_obuf/U0/U_IOOUTFF:A (r)	
	+	0.353	cell: ADLIB:IOOUTFF_BYPASS
3.578		data_out_obuf/U0/U_IOOUTFF:Y (r)	
	+	0.000	net: data_out_obuf/U0/DOUT
3.578		data_out_obuf/U0/U_IOPAD:D (r)	
	+	3.167	cell: ADLIB:IOPAD_TRI
6.745		data_out_obuf/U0/U_IOPAD:PAD (r)	
	+	0.000	net: data_out
6.745		data_out (r)	
6.745		data arrival time	
<hr/>			
Data required time calculation			
7.000		data_in (r)	
7.000		data_out (r)	
7.000		data required time	

If constraints for the synchronous paths in the design are added (in this case, a clock and an input delay); it will impact the max-delay.

```
create_clock -name clock -period 10 [get_ports {clock}]
set_input_delay -max 1.2 -clock clock [get_ports {data_in}]
```


This input delay defines the availability of the signal at data_in regardless of the path being analyzed. It will be used in the arrival time calculation of both paths. The timing analysis of the combinational path is shown in [Table 4-2](#). Notice the slack change and the input delay used in the arrival time calculation.

Table 4-2 • Combinational Path Max Delay Example - Slack Change and Input Delay Highlight

From: data_in			
To: data_out			
		data required time	7.000
		data arrival time -	7.945
		slack	-0.945
<hr/>			
Data arrival time calculation			
0.000		clock	
	+	1.200	Input Delay Constraint
1.200		data_in (r)	
	+	0.000	net: data_in
1.200		data_in_ibuf/U0/U_IOPAD:PAD (r)	
	+	1.802	cell: ADLIB:IOPAD_IN
3.002		data_in_ibuf/U0/U_IOPAD:Y (r)	
	+	0.000	net: data_in_ibuf/U0/YIN1
3.002		data_in_ibuf/U0/U_IOINFF:A (r)	
	+	0.083	cell: ADLIB:IOINFF_BYPASS
3.085		data_in_ibuf/U0/U_IOINFF:Y (r)	
	+	0.485	net: data_in_c
3.570		AND2_0:A (r)	
	+	0.163	cell: ADLIB:CFG2
3.733		AND2_0:Y (r)	
	+	0.692	net: data_out_c
4.425		data_out_obuf/U0/U_IOOUTFF:A (r)	
	+	0.353	cell: ADLIB:IOOUTFF_BYPASS
4.778		data_out_obuf/U0/U_IOOUTFF:Y (r)	
	+	0.000	net: data_out_obuf/U0/DOUT
4.778		data_out_obuf/U0/U_IOPAD:D (r)	
	+	3.167	cell: ADLIB:IOPAD_TRI
7.945		data_out_obuf/U0/U_IOPAD:PAD (r)	
	+	0.000	net: data_out
7.945		data_out (r)	
7.945		data arrival time	
<hr/>			
Data required time calculation			
7.000		data_in (r)	
7.000		data_out (r)	
7.000		data required time	

If the actual requirement between data_in and data_out is 7 ns, the max-delay should be 8.2 ns to account for the input-delay. [Table 4-3](#) shows that the slack is restored to its original value.

```
set_max_delay 8.2 ns -from [get_ports {data_in}] -to [get_ports {data_out}]
```

Table 4-3 • Combinational Path Max Delay Example - Slack Restored to Original Value

From: data_in			
To: data_out			
		data required time	8.200
		data arrival time -	7.945
		slack	0.255
<hr/>			
Data arrival time calculation			
0.000		clock	
	+	1.200	Input Delay Constraint
1.200		data_in (r)	
	+	0.000	net: data_in
1.200		data_in_ibuf/U0/U_IOPAD:PAD (r)	
	+	1.802	cell: ADLIB:IOPAD_IN
3.002		data_in_ibuf/U0/U_IOPAD:Y (r)	
	+	0.000	net: data_in_ibuf/U0/YIN1
3.002		data_in_ibuf/U0/U_IOINFF:A (r)	
	+	0.083	cell: ADLIB:IOINFF_BYPASS
3.085		data_in_ibuf/U0/U_IOINFF:Y (r)	
	+	0.485	net: data_in_c
3.570		AND2_0:A (r)	
	+	0.163	cell: ADLIB:CFG2
3.733		AND2_0:Y (r)	
	+	0.692	net: data_out_c
4.425		data_out_obuf/U0/U_IOOUTFF:A (r)	
	+	0.353	cell: ADLIB:IOOUTFF_BYPASS
4.778		data_out_obuf/U0/U_IOOUTFF:Y (r)	
	+	0.000	net: data_out_obuf/U0/DOUT
4.778		data_out_obuf/U0/U_IOPAD:D (r)	
	+	3.167	cell: ADLIB:IOPAD_TRI
7.945		data_out_obuf/U0/U_IOPAD:PAD (r)	
	+	0.000	net: data_out
7.945		data_out (r)	
7.945		data arrival time	
<hr/>			
Data required time calculation			
8.200		data_in (r)	
8.200		data_out (r)	
8.200		data required time	

If the output data will be used synchronously in the external device, it is preferable to use an output-delay (rather than a max-delay) to specify the requirement for the asynchronous path. In our example, the output delay below will create the same 7 ns requirements for data_in to data_out.

```
set_output_delay 1.8 -clock clock [get_ports {data_out}]
```

The timing analysis for the path is shown in [Table 4-4](#).

Table 4-4 • Combinational Path Output Delay Example

From: data_in			
To: data_out			
		data required time	8.200
		data arrival time -	7.945
		slack	0.255
<hr/>			
Data arrival time calculation			
0.000		clock	
	+	1.200	Input Delay Constraint
1.200		data_in (r)	
	+	0.000	net: data_in
1.200		data_in_ibuf/U0/U_IOPAD:PAD (r)	
	+	1.802	cell: ADLIB:IOPAD_IN
3.002		data_in_ibuf/U0/U_IOPAD:Y (r)	
	+	0.000	net: data_in_ibuf/U0/YIN1
3.002		data_in_ibuf/U0/U_IOINFF:A (r)	
	+	0.083	cell: ADLIB:IOINFF_BYPASS
3.085		data_in_ibuf/U0/U_IOINFF:Y (r)	
	+	0.485	net: data_in_c
3.570		AND2_0:A (r)	
	+	0.163	cell: ADLIB:CFG2
3.733		AND2_0:Y (r)	
	+	0.692	net: data_out_c
4.425		data_out_obuf/U0/U_IOOUTFF:A (r)	
	+	0.353	cell: ADLIB:IOOUTFF_BYPASS
4.778		data_out_obuf/U0/U_IOOUTFF:Y (r)	
	+	0.000	net: data_out_obuf/U0/DOUT
4.778		data_out_obuf/U0/U_IOPAD:D (r)	
	+	3.167	cell: ADLIB:IOPAD_TRI
7.945		data_out_obuf/U0/U_IOPAD:PAD (r)	
	+	0.000	net: data_out
7.945		data_out (r)	
7.945		data arrival time	
<hr/>			
Data required time calculation			
10.000		clock	
	+	0.000	Clock source
10.000		clock (r)	
	-	1.800	Output Delay Constraint
8.200		data_out (r)	
8.200		data required time	

SmartFusion2 MSS and PCIe Design

This section uses the design from the PCIe Data Plane Demo Using MSS HPDMA as an example of a typical system created using System Builder. The design creates a DMA between a PCIe host and a DDR3 memory. It does this using a MSS and SERDESIF IP configured as PCIe.

The design documentation can be found in the [SmartFusion2 PCIe MSS HPDMA Demo Guide](#).

Download the design files from the [Microsemi website](#).

SmartFusion2 MSS and PCIe Design Analysis

The block diagram of the demo is shown in [Figure 4-4](#). It shows how MSS, SERDES_IF (configured as a PCIe) and LSRAM are connected using an AHB bus.

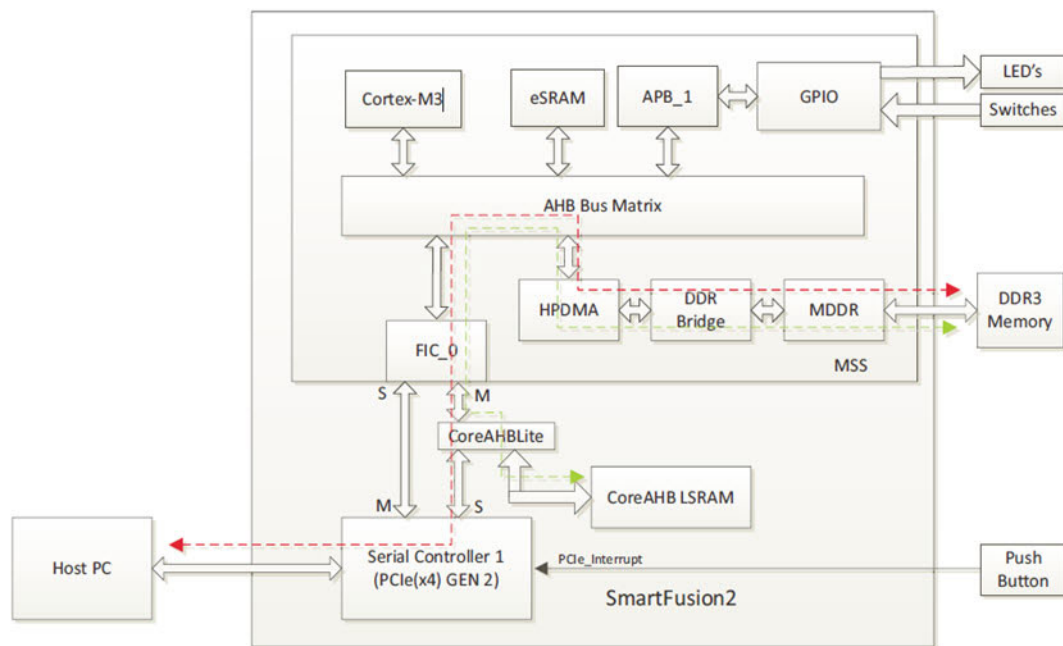


Figure 4-4 • SmartFusion2 MSS and PCIe Block Diagram

In this design, I/Os are used through hard IPs and no constraints are needed. Only clocks must be specified.

The design clocks are configured in System Builder, as shown in Figure 4-5. System Builder also instantiates blocks required for the initialization of the system adding other clocks.

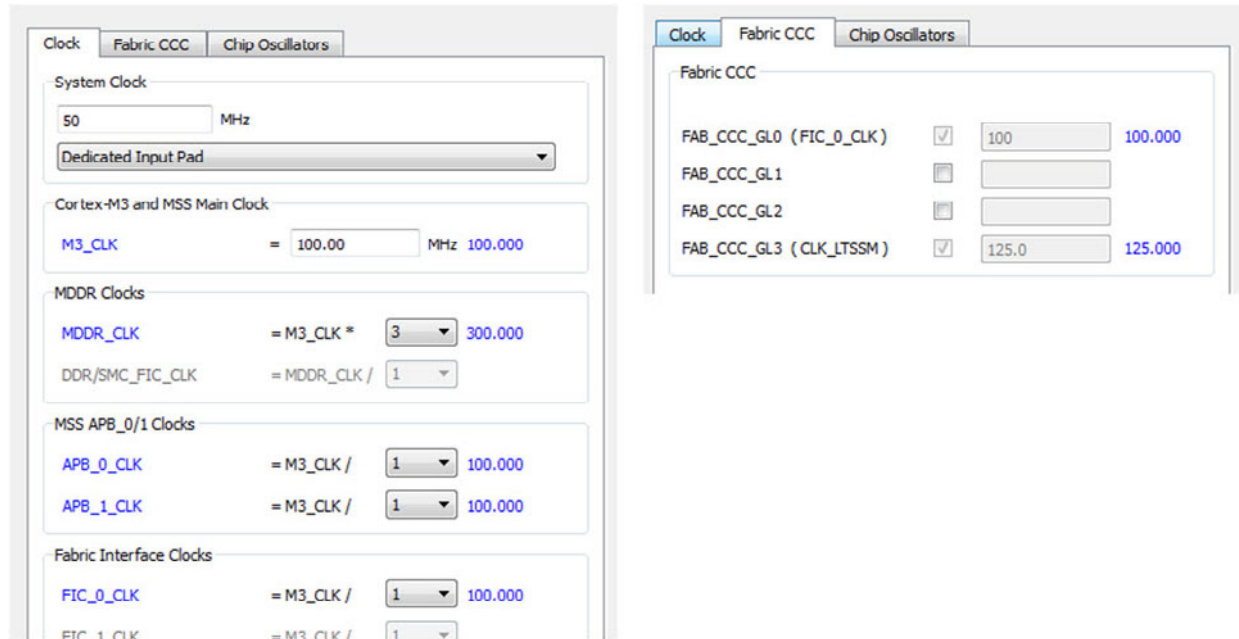


Figure 4-5 • SmartFusion2 MSS and PCIe System Builder Clock Configuration

Figure 4-6 shows the connections of all the clocks in the system.

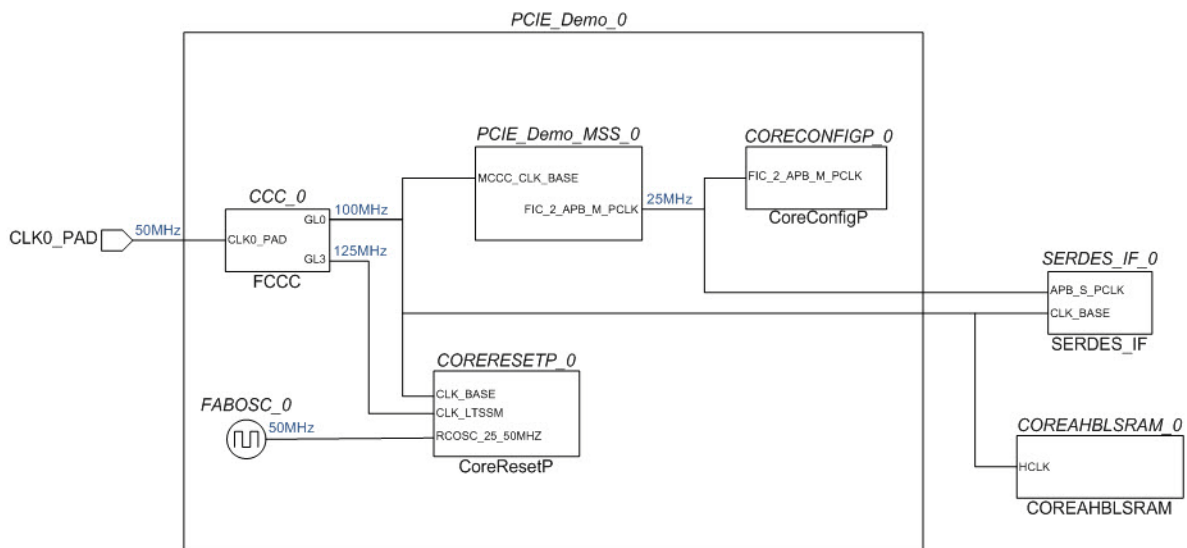


Figure 4-6 • SmartFusion2 MSS and PCIe System Builder Clock Used by the System

The Advanced tab of the CCC configurator provides the details of the CCC configuration used to generate the clocks (Figure 4-7). The 50 MHz clock is first multiplied by 10 by the PLL and then divided by 5 and 4 to generate 100 MHz and 125 MHz, respectively.

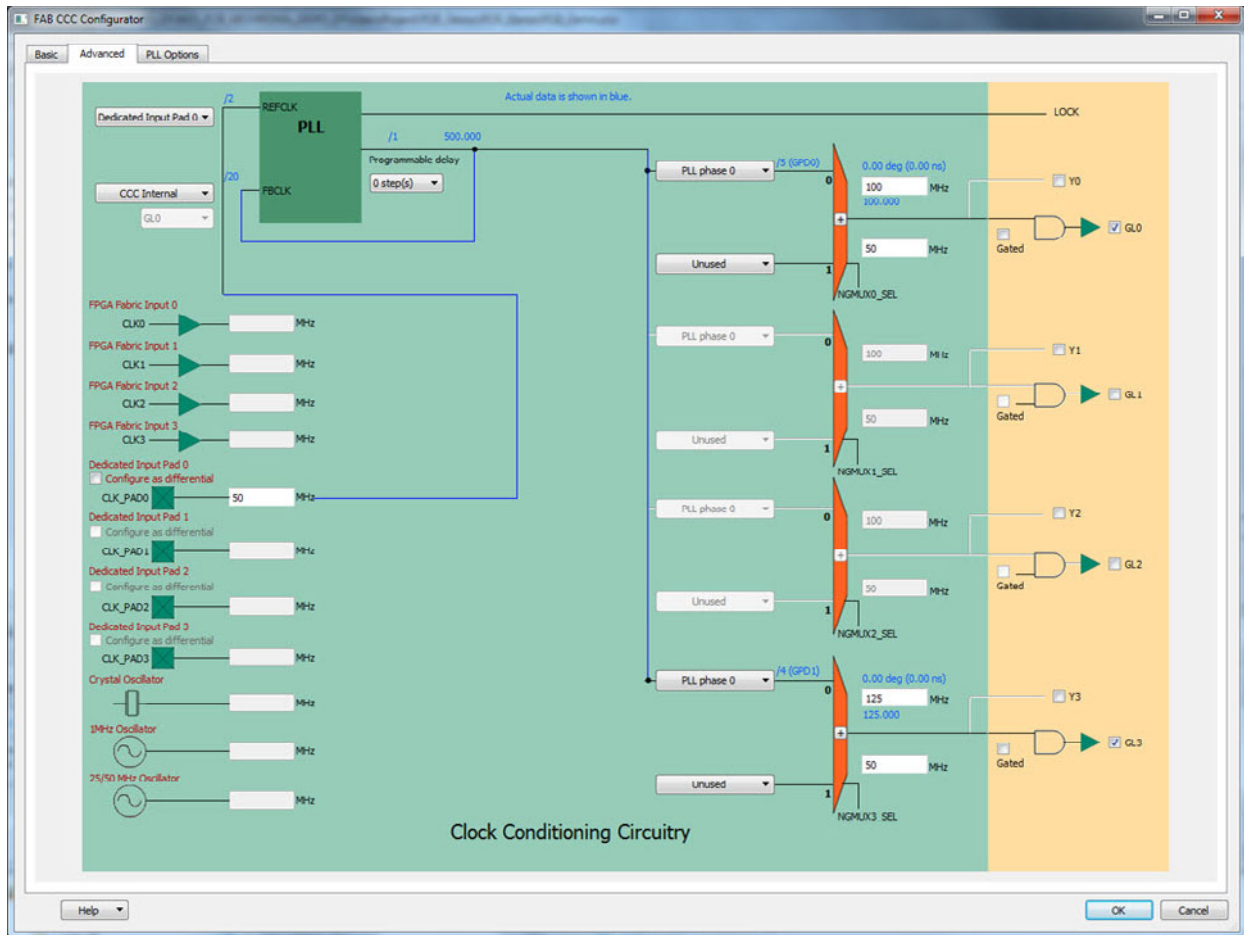


Figure 4-7 • SmartFusion2 MSS and PCIe System Builder CCC Configuration

SmartFusion2 MSS and PCIe Synthesis Constraints

The design uses two clock sources of 50 MHz: one from an input port, the other from an integrated oscillator.

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

create_clock -name OSC_50MHz -period 20 \
  [get_pins {PCIE_Demo_0.FABOSC_0.RCOSC_25_50MHZ_O2F}]
```

The CCC generated 2 clocks from the 50 MHz system clock:

```
FIC_0_CLK at 100 MHz on GL0 for the AHB bus.
CLK_LTSSM at 125 MHz on GL3 used by CoreResetP.
create_generated_clock -name FIC_0_CLK -multiply_by 10 -divide_by 5 \
  -source [get_pins {PCIE_Demo_0.CCC_0.CLK0_PAD}] \
  [get_pins {PCIE_Demo_0.CCC_0.GL0}]

create_generated_clock -name CLK_LTSSM -multiply_by 10 -divide_by 4 \
  -source [get_pins {PCIE_Demo_0.CCC_0.CLK0_PAD}] \
  [get_pins {PCIE_Demo_0.CCC_0.GL3}]
```

Finally, the MSS creates an asynchronous clock for the configuration APB bus of 1/4 of the Cortex-M3 clock or, in this case, 25 MHz.

```
create_clock -name APB_CLK -period 40 \
    [get_pins {PCIE_Demo_0.PCIE_Demo_MSS_0.FIC_2_APB_M_PCLK}]
```

After running synthesis, the clock summary in the synthesis report shows that all clock constraints were taken into account and that no other clocks were inferred.

Clock Summary

Start Clock	Requested Frequency	Requested Period	Clock Type	Clock Group

--				
APB_CLK	50.0 MHz	20.000	declared	default_clkgroup
CLK0_PAD	50.0 MHz	20.000	declared	default_clkgroup
CLK_LTSSM	125.0 MHz	8.000	generated (from CLK0_PAD)	default_clkgroup
FIC_0_CLK	100.0 MHz	10.000	generated (from CLK0_PAD)	default_clkgroup
OSC_50MHz	50.0 MHz	20.000	declared	default_clkgroup
System	1.0 MHz	1000.000	system	system_clkgroup
=====				

If you run synthesis interactively, the same information is provided in the GUI, as shown in [Figure 4-8](#).

Timing Summary			
Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
APB_CLK	50.0 MHz	37.2 MHz	-1.379
CLK0_PAD	50.0 MHz	NA	NA
CLK_LTSSM	125.0 MHz	92.9 MHz	4.816
FIC_0_CLK	100.0 MHz	100.9 MHz	0.089
OSC_50MHz	50.0 MHz	408.6 MHz	9.106
System	100.0 MHz	660.8 MHz	8.487
Detailed report	Timing Report View		

Figure 4-8 • SmartFusion2 MSS and PCIe Clock Summary in Synplify Pro

SmartFusion2 MSS and PCIe Place and Route Constraints

The clocks from the oscillator and the CCC are automatically generated by SmartTime. Only CLK0_PAD and APB_CLK must be specified.

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

create_clock -name APB_CLK -period 40 \
    [get_pins {PCIE_Demo_0/PCIE_Demo_MSS_0/MSS_ADLIB_INST:CLK_CONFIG_APB}]
```

MSS (TBI Interface) to SERDES (SmartFusion2 Only)

When you configure the MSS Ethernet MAC (Right-click **MSS** > **Configure** > double-click **Ethernet**) and select the TBI Interface, the signal path on the MAC_TBI_TCGP[9:0] bus from the MSS MAC to the SERDES in the Fabric is a multicycle path and should be constrained as such (Figure 4-9).

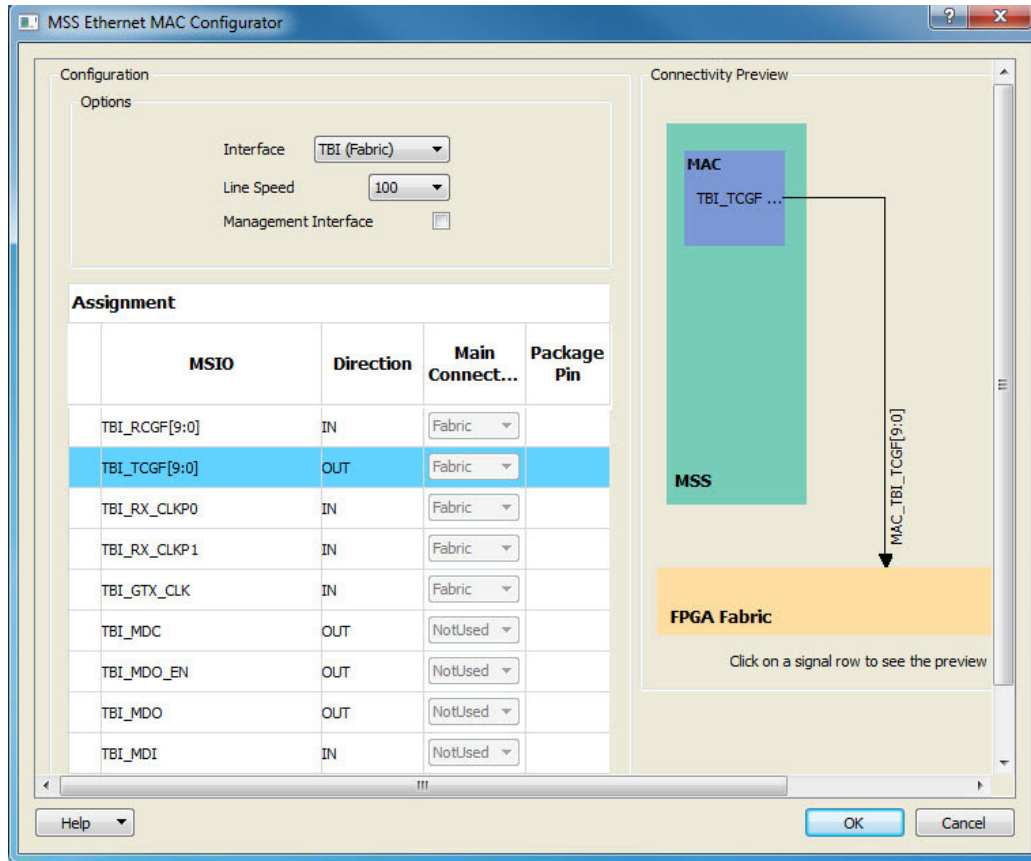


Figure 4-9 • Configuring the MSS Ethernet MAC for the TBI (Fabric) Interface

Figure 4-10 shows the paths between the MSS MAC and the SERDES in the fabric. The path from MAC_TBI_TCGF[9:0] to EPCS_3_TX_DATA[9:0] is a multicycle path.

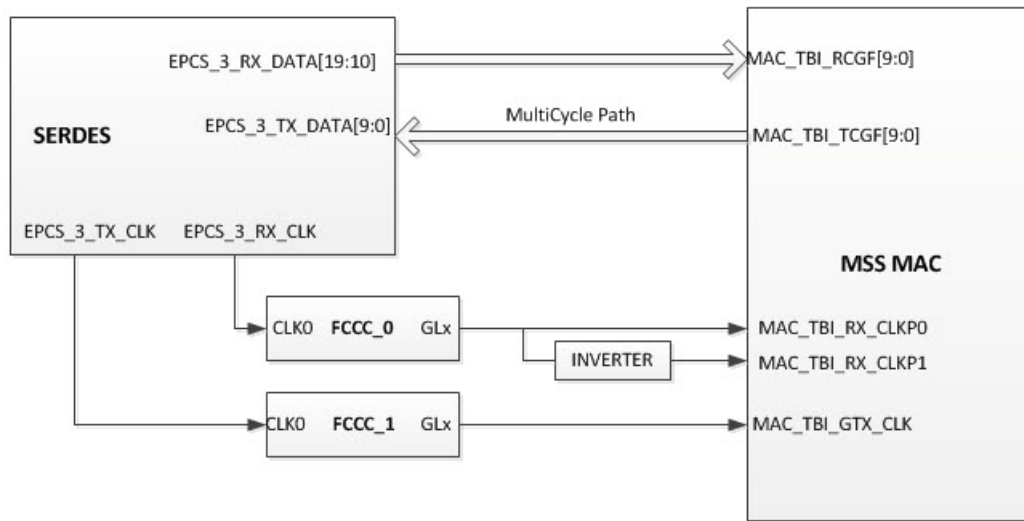


Figure 4-10 • Multicycle Path from MSS MAC to SERDES (Fabric)

Set the multicycle path constraint in an *.SDC file. If you do not, Libero's SmartTime may report set up and/or hold time violations.

```
set_multicycle_path -setup 3 -from { \
  Webserver_TCP_0/Webserver_TCP_MSS_0/MSS_ADLIB_INST/INST_MSS_120_IP:GTX_CLKPF } \
  -to { SERDES_IF_0/SERDESIF_INST/INST_SERDESIF_IP:EPCS_3_TX_DATA* }

set_multicycle_path -hold 0 -from\
  {Webserver_TCP_0/Webserver_TCP_MSS_0/MSS_ADLIB_INST/INST_MSS_120_IP:GTX_CLKPF} \
  -to { SERDES_IF_0/SERDESIF_INST/INST_SERDESIF_IP:EPCS_3_TX_DATA* }
```

Figure 4-11 describes the setup and hold timing check relations.

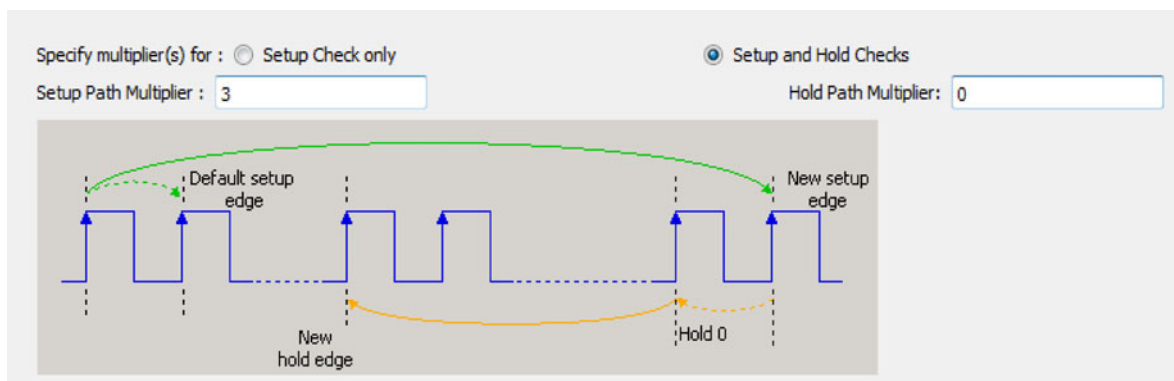


Figure 4-11 • Setup and Hold Timing Check Relations for Multicycle Path

Pass the *.sdc file to Compile (Right-click the *.sdc file and choose **Use for Compile**) and select **Timing-Driven Layout Option (Place and Route > Configure Options)**. Check to make sure that the multicycle path constraints are set when you open the SmartTime Constraints Editor after Layout.

A – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**

From the rest of the world, call **650.318.4460**

Fax, from anywhere in the world, **650.318.8044**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit <http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support>.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group [home page](http://www.microsemi.com/soc), at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to [My Cases](#).

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office.

Visit [About Us](#) for sales office listings and corporate contacts.

Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.



Microsemi Corporate 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

E-mail: sales.support@microsemi.com

©2016 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace 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. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.