
SmartFusion2/IGLOO2 FPGA
Timing Constraints for Enhanced Constraints Flow
User's Guide
For Libero SoC v11.7



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 Pro	7
Timing Constraints for Timing-Driven Place and Route	13
Improving Placer Performance	18
3 Constraints for SmartFusion2 and IGLOO2 IP Blocks	19
Oscillators	19
Fabric Clock Conditioning Circuit (CCC) for SmartFusion2 and IGLOO2	20
SERDES/DDR Configuration Subsystem (MSS/HPMS FIC_2)	22
CoreResetP False Paths (SmartFusion2 and IGLOO2 Only)	23
High Speed Serial Interface (SERDES) Block	24
4 Constraint Case Studies	28
Source-Synchronous Interface	28
Constraints and Combinational Paths	30
SmartFusion2 MSS and PCIe Design	35
MSS (TBI Interface) to SERDES (SmartFusion2 Only)	39
5 Product Support	42
Customer Service	42
Customer Technical Support Center	42
Technical Support	42
Website	42
Contacting the Customer Technical Support Center	42
ITAR Technical Support	43

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

With Libero SoC 11.7 release, Microsemi introduces the Enhanced Constraint Flow to simplify the management of all constraints including SDC timing constraints. In the Enhanced Constraint Flow, timing constraints need only be entered once, and can be applied to Synplify synthesis, Timing-Driven Layout and Timing Verification. Timing constraints for known hardware blocks and IPs (CCC, Oscillator, MSS/HPMS, FDDR, SERDES), can 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 SmartFusion2 and IGLOO2 designs, Microsemi recommends setting SDC 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](#).

Note: This User Guide describes how to set timing constraints for SmartFusion2/IGLOO2 devices in the Enhanced Constraint Flow introduced in Libero 11.7. If you use Libero 11.6 or the Classic Constraint Flow in Libero 11.7, refer to the [SmartFusion2-IGLOO2 FPGA Timing Constraints User's Guide](#).

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

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

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
<code>set_disable_timing</code>	1
<code>set_false_path</code>	2
<code>set_max_delay/set_min_delay</code>	3
<code>set_multicycle_path</code>	4

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

Timing Exceptions	Order of Precedence
<code>set_false_path</code>	1
<code>set_max_delay/set_min_delay</code>	2
<code>set_multicycle_path</code>	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.

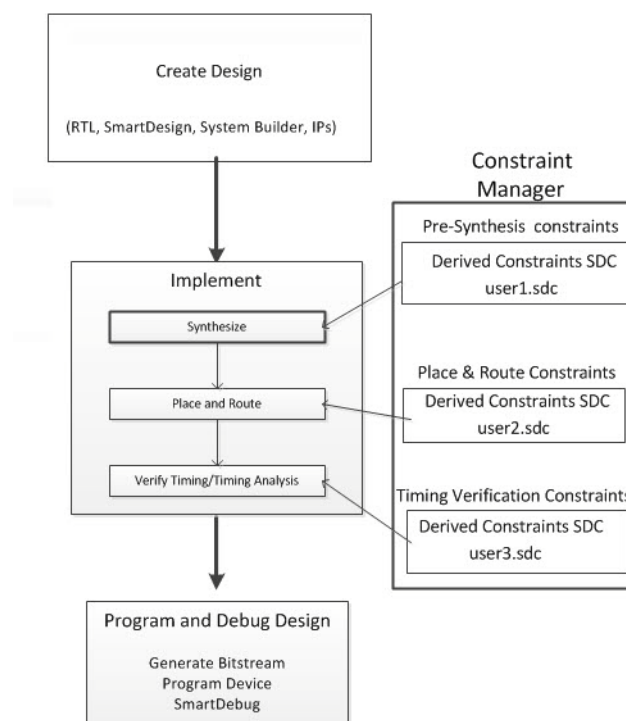


Figure 2-1 • Timing Constraints in the Design Flow

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 (Synplify Pro 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 in one of the following two ways:

- 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.

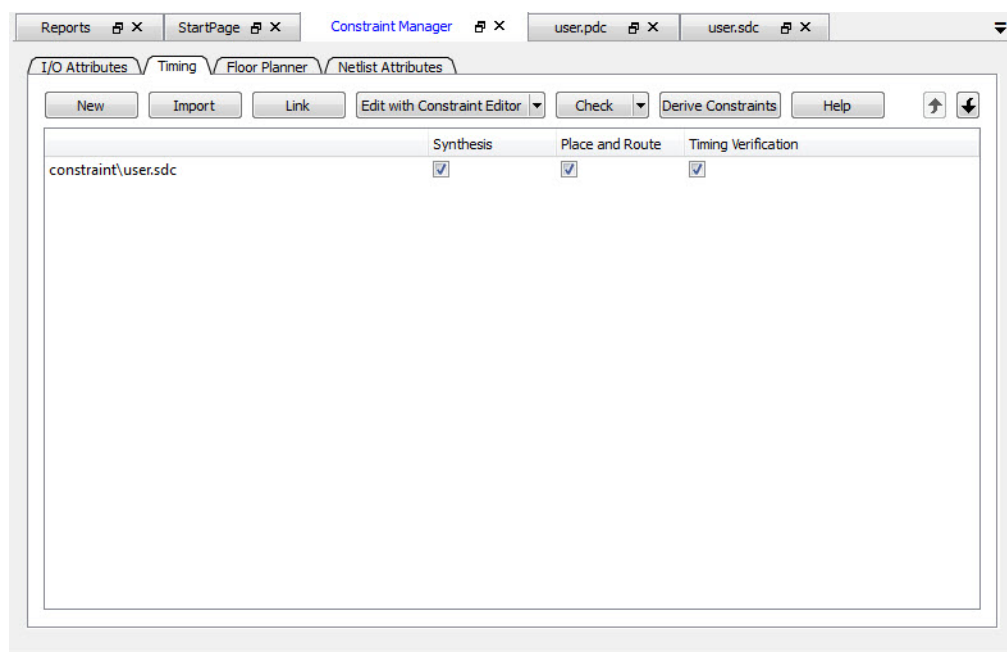


Figure 2-2 • SDC Timing Constraint File Association with Synthesis

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

If the design contains IP blocks such as OSC, CCC, MSS and SERDES, the Libero SoC Enhanced Constraint Flow is capable of deriving SDC timing constraints for the IP blocks. 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. 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. These constraints are essential for board-level 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 • Precedence Order

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).

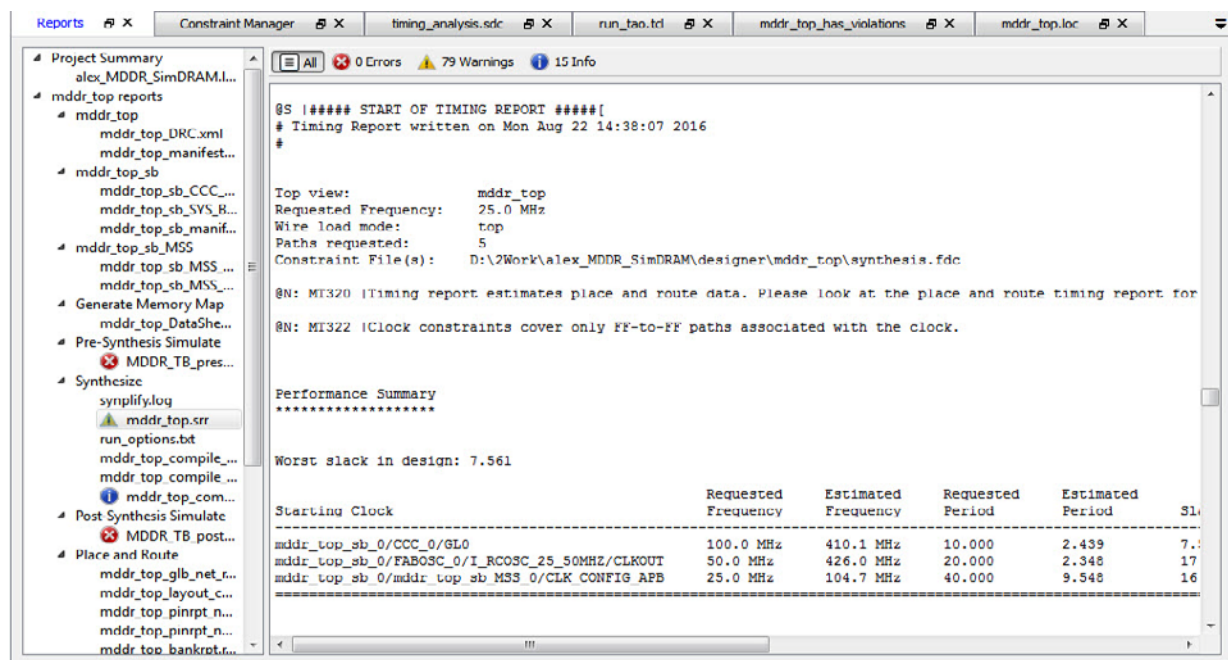


Figure 2-3 • Synplify Pro *.srr File Open in Libero SoC

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 guide 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_ctrl1]

# 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 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 exclusively for Place and Route. To create SDC timing constraints for Place and Route, do one of the following:

- 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.

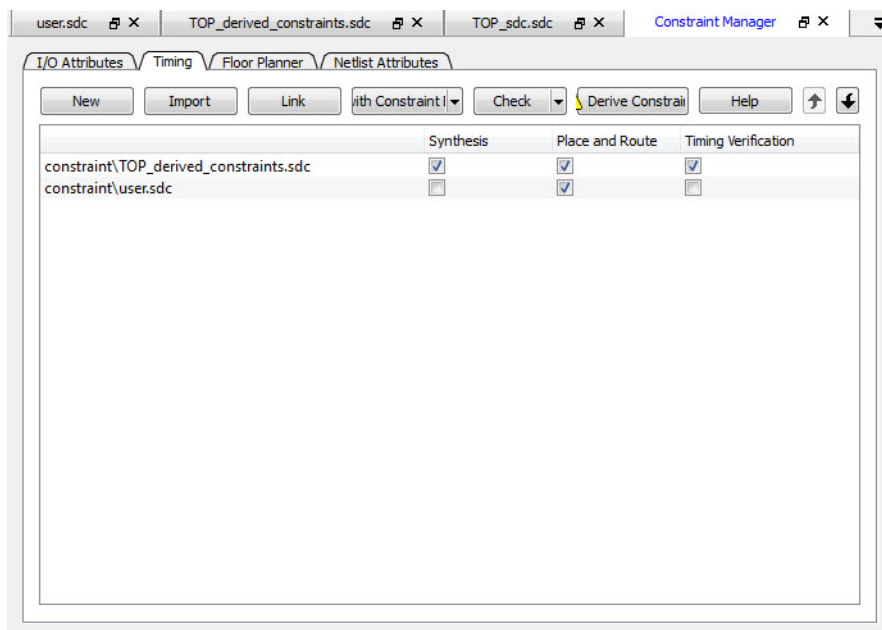


Figure 2-4 • SDC Timing Constraint File Associated with Place and Route

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 SDC 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_group

Limitations

Do not use the SDC object access command [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 [get_pins] construct instead as a workaround.

The following example uses the get_nets command:

```
create_generated_clock -name {sys_clk} -divide_by 1 -source [ get_ports { CLK_40M } ] -
phase 0 [ get_nets { u_PLL/my_pll_0/GL0_net } ]
```

Rewrite the constraint to use get_pins command instead:

```
create_generated_clock -name {sys_clk} -divide_by 1 -source [ get_ports { CLK_40M } ] -
phase 0 [ get_pins { u_PLL/my_pll_0/CCC_INST/GL0 } ]
```

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.

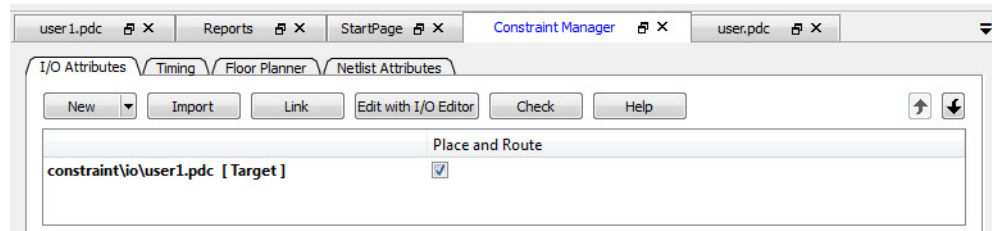


Figure 2-5 • Association of I/O Attribute Constraints

Constraint Coverage and Timing Violations

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 invoke the 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)

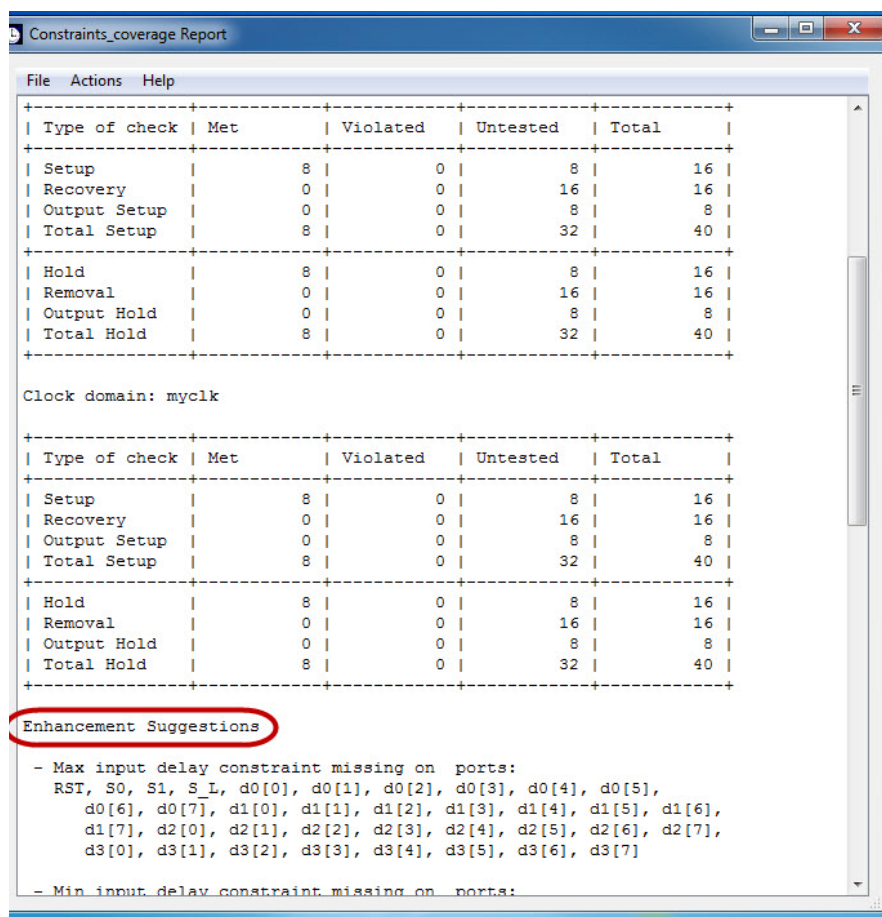


Figure 2-6 • Constraint Coverage Report

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 > 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 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.

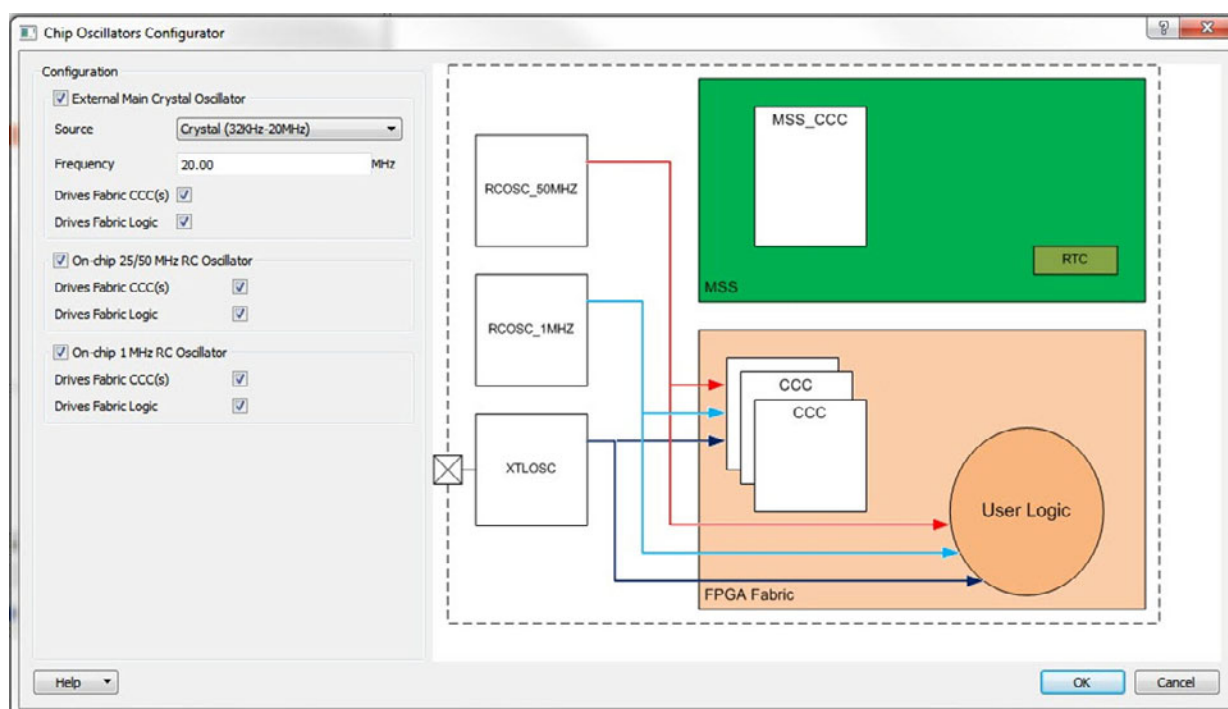


Figure 3-1 • Oscillator Configurator

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 SDC 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.

Libero SoC is able to generate for you the SDC timing constraints for the Oscillator. From the Constraint Manager, generate the clock constraint for the Oscillator (**Constraint Manager > Timing > Derived Constraints**). As an example, for the crystal oscillator configured for 20 MHz, the following constraints are generated and stored in the <root>_derived_constraints.sdc.

```
create_clock -name {M3_MDDR_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT} -period 20\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT } ]  
create_clock -name {M3_MDDR_0/FABOSC_0/I_RCOSC_1MHZ/CLKOUT} -period 1000\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_RCOSC_1MHZ/CLKOUT } ]  
create_clock -name {M3_MDDR_0/FABOSC_0/I_XTLOSC/CLKOUT} -period 50\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_XTLOSC/CLKOUT } ]]
```

To prevent design object name mismatches between the SDC file and the post-synthesis netlist during the layout step, it is recommended that the <root>_derived_constraints.sdc file be passed to Synthesis and the Place and Route tool. To do so, associate the SDC file with Synthesis and Place and Route in the Constraint Manager. See ["Association of SDC Timing Constraint File to Synthesis" on page 8](#) for details.

Design Created with SystemBuilder

SystemBuilder instantiates an oscillator IP 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. The clock constraint is generated and stored in the <root>_derived_constraints.sdc. Associate the <root>_derived_constraints.sdc file with Synthesis and Place and Route. See ["Association of SDC Timing Constraint File to Synthesis" on page 8](#) for details.

```
create_clock -name {M3_MDDR_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT} -period 20\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT } ]  
create_clock -name {M3_MDDR_0/FABOSC_0/I_RCOSC_1MHZ/CLKOUT} -period 1000\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_RCOSC_1MHZ/CLKOUT } ]  
create_clock -name {M3_MDDR_0/FABOSC_0/I_XTLOSC/CLKOUT} -period 50\  
[ get_pins { M3_MDDR_0/FABOSC_0/I_XTLOSC/CLKOUT } ]]
```

Oscillator Place and Route Constraints

No other constraints are needed. To pass the <root>_derived_constraints.sdc file to Place and Route, associate the SDC file with Place and Route in the Constraints Manager. See ["Association of SDC Timing Constraint File to Place and Route" on page 14](#) for details. 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.


```
create_generated_clock -name {M3_MDDR_0/CCC_0/GL0} -multiply_by 24 -divide_by 16\  
-source [ get_pins { M3_MDDR_0/CCC_0/CCC_INST/CLK0_PAD } ] -phase 0\  
[ get_pins { M3_MDDR_0/CCC_0/CCC_INST/GL0 } ]  
create_generated_clock -name {M3_MDDR_0/CCC_0/GL1} -multiply_by 24 -divide_by 12\  
-source [ get_pins { M3_MDDR_0/CCC_0/CCC_INST/CLK0_PAD } ] -phase 0\  
[ get_pins { M3_MDDR_0/CCC_0/CCC_INST/GL1 } ]  
create_generated_clock -name {M3_MDDR_0/CCC_0/GL2} -divide_by 2\  
-source [ get_pins { M3_MDDR_0/CCC_0/CCC_INST/RCOSC_25_50MHZ/CLKOUT } ]\  
[ get_pins { M3_MDDR_0/CCC_0/CCC_INST/GL2 } ]
```

Pass the <root>_derived_constraints.sdc file to Synplify Pro for processing. To pass the <root>_derived_constraints.sdc file to Synplify Pro, associate the SDC file with Synthesis in the Constraint Manager. See "Association of SDC Timing Constraint File to Synthesis" on page 8 for details

Fabric CCC Place and Route Constraints

The same constraints are applicable to Place and Route. Pass the same <root>_derived_constraints.sdc to Place and Route by associating the file with Place and Route in the Constraint Manager. See "Association of SDC Timing Constraint File to Place and Route" on page 14 for details.

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.

Specifying a Clock Constraint for FIC_2_APB_M_PCLK

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

Synthesis Timing Constraints

```
create_clock -name {M3_MDDR_0/M3_MDDR_MSS_0/CLK_CONFIG_APB} -period 40 [ get_pins {  
M3_MDDR_0/M3_MDDR_MSS_0/MSS_ADLIB_INST/CLK_CONFIG_APB } ]
```

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

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

Place and Route Timing Constraints

```
create_clock -name {M3_MDDR_0/M3_MDDR_MSS_0/CLK_CONFIG_APB} -period 40 [ get_pins {  
M3_MDDR_0/M3_MDDR_MSS_0/MSS_ADLIB_INST/CLK_CONFIG_APB } ]
```

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

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

Libero SoC is able to generate for you this constraint and places it in the <root>_derived_constraints.sdc file. After generation of the top level design, go to Timing Tab of the Constraint Manager and click Derive Constraints to generate the constraints (**Constraint Manager > Timing > Derive Constraints**)

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 two timing requirements are needed to capture the re-timing behavior. They are required for Synthesis, Timing Driven Place and Route (TDPR) and Timing Verifications. Libero SoC is able to generate them for you. After generation of the top level design, go to Timing Tab of the Constraint Manager and click Derive Constraints to generate the constraints (**Constraint Manager > Timing > Derive Constraints**). The generated constraints is placed in the <root>_derived_constraints.sdc file. In the Constraint Manager, associate the SDC file to Synthesis, Place and Route and Timing Verifications.

```
set_max_delay 0 -through [ get_nets { M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PSEL\
M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PENABLE } ]\
-to [ get_cells { M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PREADY*\
M3_MDDR_0/CORECONFIGP_0/state[0] } ]
set_min_delay -24 -through [ get_nets { M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PWRITE\
M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PADDR[*]\
M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PWDATA[*]\
M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PSEL\
M3_MDDR_0/CORECONFIGP_0/FIC_2_APB_M_PENABLE } ]
```

Note: The names of the signals in the constraints use the hierarchical names of pre-synthesis netlist

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 when you generate the <root>_derived_constraints.sdc file (**Constraint Manager > Timing > Derive Constraints**).

```
set_false_path -ignore_errors -through [ get_nets {M3_MDDR_0/CORECONFIGP_0/INIT_DONE\
M3_MDDR_0/CORECONFIGP_0/SDIF_RELEASED}]
set_false_path -ignore_errors -through [ get_nets { M3_MDDR_0/CORERESETP_0/DDR_settled\
M3_MDDR_0/CORERESETP_0/count_ddr_enable M3_MDDR_0/CORERESETP_0/release_sdif*_core\
M3_MDDR_0/CORERESETP_0/count_sdif*_enable } ]
set_false_path -ignore_errors -from [ get_cells\
{ M3_MDDR_0/CORERESETP_0/MSS_HPMS_READY_int } ]\
-to [ get_cells { M3_MDDR_0/CORERESETP_0/sm0_areset_n_rcosc\
M3_MDDR_0/CORERESETP_0/sm0_areset_n_rcosc_q1 } ]
set_false_path -ignore_errors -from\
[ get_cells {M3_MDDR_0/CORERESETP_0/MSS_HPMS_READY_int M3_MDDR_0/CORERESETP_0/\
SDIF*_PERST_N_re } ]\
-to [ get_cells { M3_MDDR_0/CORERESETP_0/sdif*_areset_n_rcosc* } ]
set_false_path -ignore_errors -through [get_nets { M3_MDDR_0/CORERESETP_0/CONFIG1_DONE\
M3_MDDR_0/CORERESETP_0/CONFIG2_DONE M3_MDDR_0/CORERESETP_0/SDIF*_PERST_N\
M3_MDDR_0/CORERESETP_0/SDIF*_PSEL M3_MDDR_0/CORERESETP_0/SDIF*_PWRITE\
M3_MDDR_0/CORERESETP_0/SDIF*_PRDATA[*] M3_MDDR_0/CORERESETP_0/SOFT_EXT_RESET_OUT\
M3_MDDR_0/CORERESETP_0/SOFT_RESET_F2M M3_MDDR_0/CORERESETP_0/SOFT_M3_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_MDDR_DDR_AXI_S_CORE_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_FDDR_CORE_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_SDIF*_PHY_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_SDIF*_CORE_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_SDIF0_0_CORE_RESET\
M3_MDDR_0/CORERESETP_0/SOFT_SDIF0_1_CORE_RESET } ]
```

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 3-3. You do not need to take any further action to exclude these paths from timing analysis.

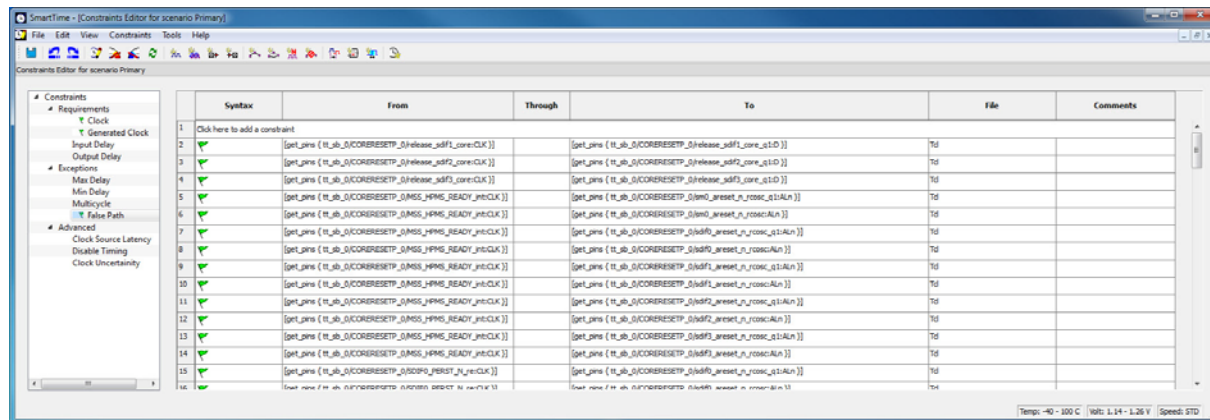


Figure 3-3 • False Path Constraints in CoreResetP Block

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 the SERDES block configured as PCIe mode 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

Libero SoC is able to generate the SDC timing constraints for the XAUI SERDES. From the Constraint Manager, generate the clock constraint for the XAUI SERDES (**Constraint Manager > Timing > Derived Constraints**).

At the prompt click Yes to automatically associate the derived constraints SDC file to Synthesis, Place and Route and Timing Verification.

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.

The generated clock constraints for the XAUI are stored in the <root>_derived_constraints.sdc file:

```
create_clock -name {myxaui_0/myxaui_0/SERDESIF_INST/EPCS_RXCLK_0} -period 6.4\  
[ get_pins { myxaui_0/myxaui_0/SERDESIF_INST/EPCS_RXCLK_0 } ]  
create_clock -name {myxaui_0/myxaui_0/SERDESIF_INST/XAUI_OUT_CLK} -period 6.4\  
[ get_pins { myxaui_0/myxaui_0/SERDESIF_INST/XAUI_OUT_CLK } ]
```

XAUI Place and Route Constraints

The same SDC Timing constraints in the <root>_derived_constraints.sdc file are used for Place and Route as well as for Synthesis.

```
create_clock -name {myxaui_0/myxaui_0/SERDESIF_INST/EPCS_RXCLK_0} -period 6.4\  
[ get_pins { myxaui_0/myxaui_0/SERDESIF_INST/EPCS_RXCLK_0 } ]  
create_clock -name {myxaui_0/myxaui_0/SERDESIF_INST/XAUI_OUT_CLK} -period 6.4\  
[ get_pins { myxaui_0/myxaui_0/SERDESIF_INST/XAUI_OUT_CLK } ]
```

Check the file association in the Constraint Manager. Make sure the <root>_derived_constraints.sdc is associated with Place and Route.

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.

EPCS Protocol Synthesis Constraints

Libero SoC is able to generate/derive the SDC timing constraints for the EPCS SERDES based on the configuration of the SERDES and the component SDC files. From the Constraint Manager generate the clock constraint for the EPCS SERDES (**Constraint Manager > Timing > Derived Constraints**). The generated clock constraints are placed in the file <root>_derived_constraints.sdc.

At the prompt click **Yes** to automatically associate the derived constraints SDC file to Synthesis, Place and Route and Timing Verification.

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.

The example constraints below are the derived clock constraints for a SERDESIF block instantiated as SERDES_IF2_0 using all four lanes with a PHY RefClk Frequency of 125 MHz and a data width of 16.

```
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_0} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_0 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_0} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_0 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_1} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_1 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_1} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_1 } ]
```

```
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[0]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[0] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[0]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[0] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[1]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[1] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[1]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[1] } ]
```

EPCS Protocol Place and Route Constraints

The same constraints in the <root>_derived_constraints.sdc file used for Synthesis are passed to Place and Route.

```
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_0} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_0 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_0} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_0 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_1} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK_1 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_1} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK_1 } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[0]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[0] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[0]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[0] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[1]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_RXCLK[1] } ]  
create_clock -name {SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[1]} -period 8\  
[ get_pins { SERDES_IF2_0/SERDESIF_INST/EPCS_TXCLK[1] } ]
```

Check the file association in the Constraint Manager. Make sure the <root>_derived_constraints.sdc is associated with Place and Route.

4 – Constraint Case Studies

This chapter has case studies for:

- Source-Synchronous Interface
- Constraints and Combinational Paths
- SmartFusion2 MSS and PCIe
- MSS (TBI Interface) to SERDES (SmartFusion2 Only)

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.

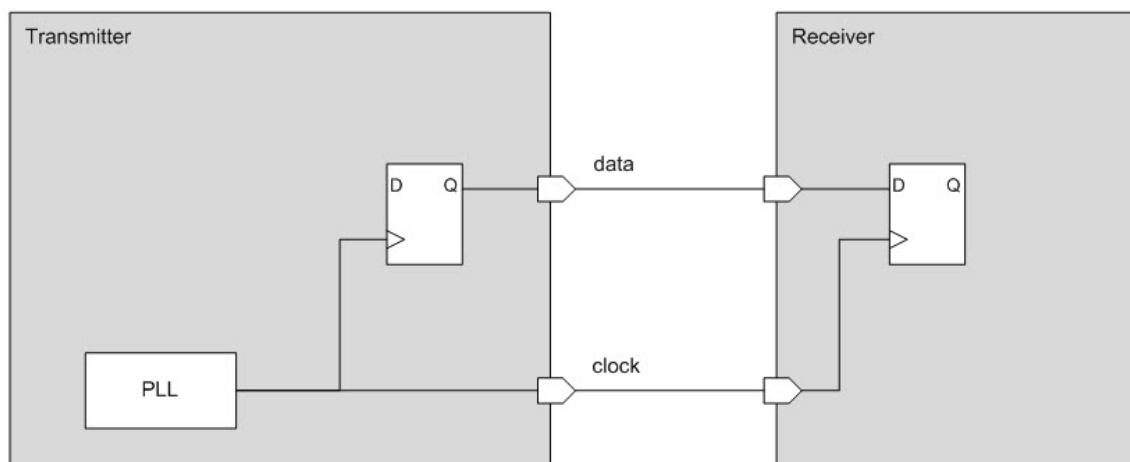


Figure 4-1 • Basic Source-Synchronous Transmitter

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.

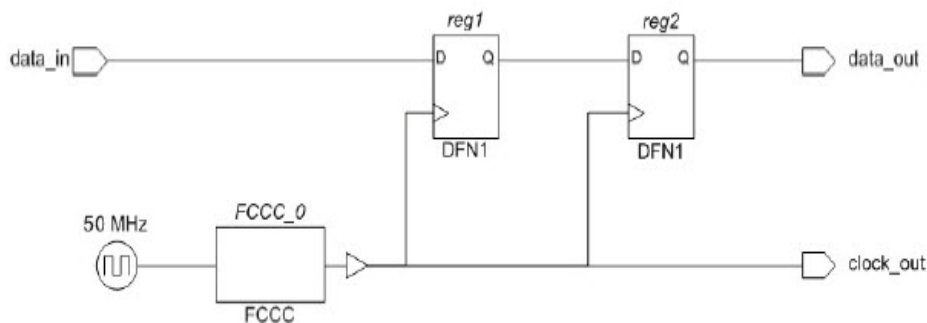


Figure 4-2 • Source-Synchronous Interface Design Example

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

After the top level design generation, Libero SoC is able to derive and generate the clock constraints for the Oscillator and the generated_clock constraints for the CCC. These constraints are placed in the <root>_derived_constraints.sdc file. Associate the <root>_derived_constraints.sdc file to Synthesis and Place and Route.

For the input and output delay constraints, use the Constraint Manager to create the constraints (**Constraint Manager > Timing > Edit with Constraint Editor > Edit Place and Route Constraints**).

In the Constraint Editor that opens, add the Input and Output delay constraints. Save the edits and exit the Constraints Editor. Associate the newly-created SDC timing constraint file to Place and Route.

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`.

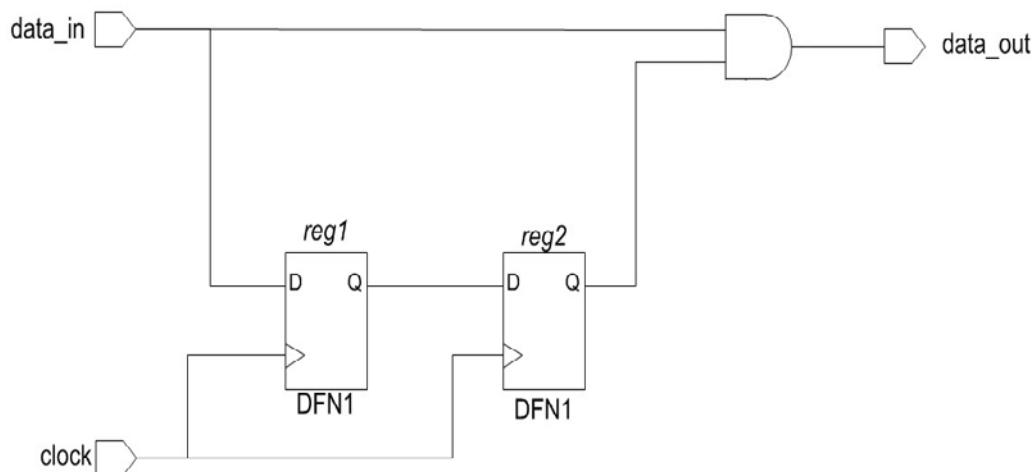


Figure 4-3 • Constraints and Asynchronous Paths Design Example

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
Data arrival time calculation			
0.000		data_in (r)	
	+	net: data_in	
0.000		data_in_ibuf/U0/U_IOPAD:PAD (r)	
	+	cell: ADLIB:IOPAD_IN	
1.802		data_in_ibuf/U0/U_IOPAD:Y (r)	
	+	net: data_in_ibuf/U0/YIN1	
1.802		data_in_ibuf/U0/U_I0INFF:A (r)	
	+	cell: ADLIB:I0INFF_BYPASS	
1.885		data_in_ibuf/U0/U_I0INFF:Y (r)	
	+	net: data_in_c	
2.370		AND2_0:A (r)	
	+	cell: ADLIB:CFG2	
2.533		AND2_0:Y (r)	
	+	net: data_out_c	
3.225		data_out_obuf/U0/U_I0OUTFF:A (r)	
	+	cell: ADLIB:I0OUTFF_BYPASS	
3.578		data_out_obuf/U0/U_I0OUTFF:Y (r)	
	+	net: data_out_obuf/U0/DOUT	
3.578		data_out_obuf/U0/U_IOPAD:D (r)	
	+	cell: ADLIB:IOPAD_TRI	
6.745		data_out_obuf/U0/U_IOPAD:PAD (r)	
	+	net: data_out	
6.745		data_out (r)	
6.745		data arrival time	
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
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	
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
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	
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
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	
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.

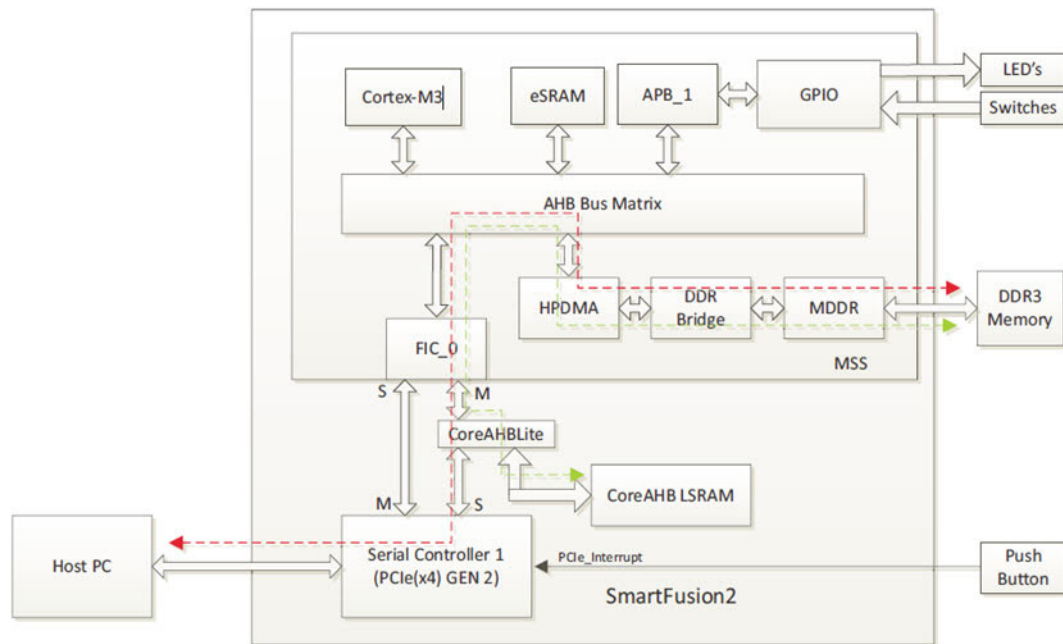


Figure 4-4 • SmartFusion2 MSS and PCIe Block Diagram

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.

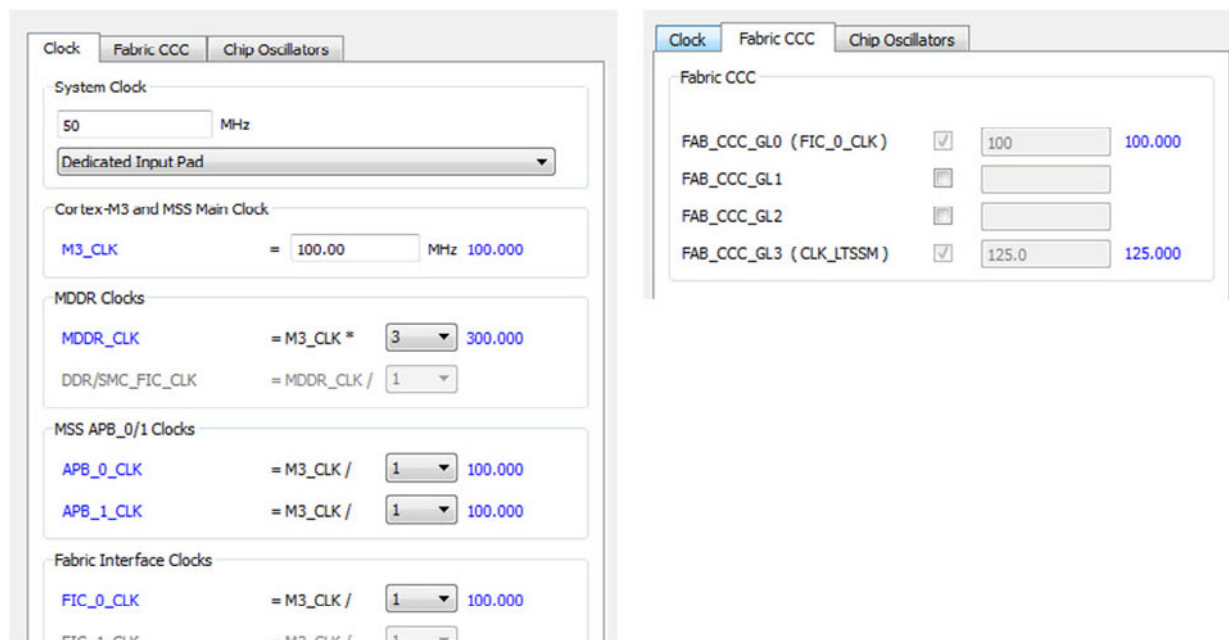


Figure 4-5 • SmartFusion2 MSS and PCIe System Builder Clock Configuration

[Figure 4-6](#) shows the connections of all the clocks in the system.

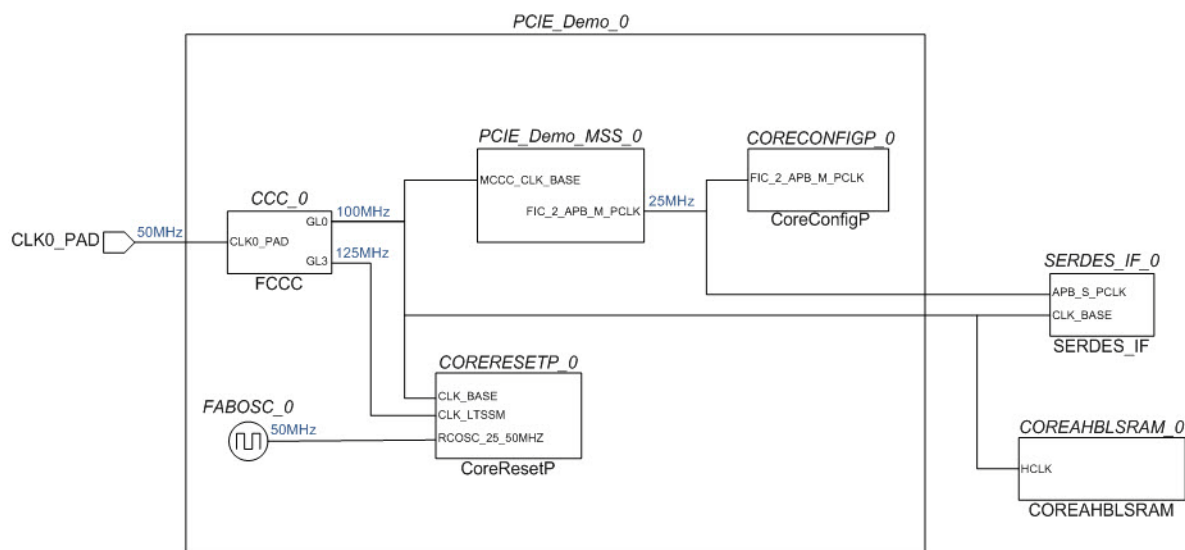


Figure 4-6 • SmartFusion2 MSS and PCIe System Builder Clock Used by the System

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 20 by the PLL and then divided by 10 and 8 to generate the 100 MHz (GL0) and 125 MHz (GL3) respectively.

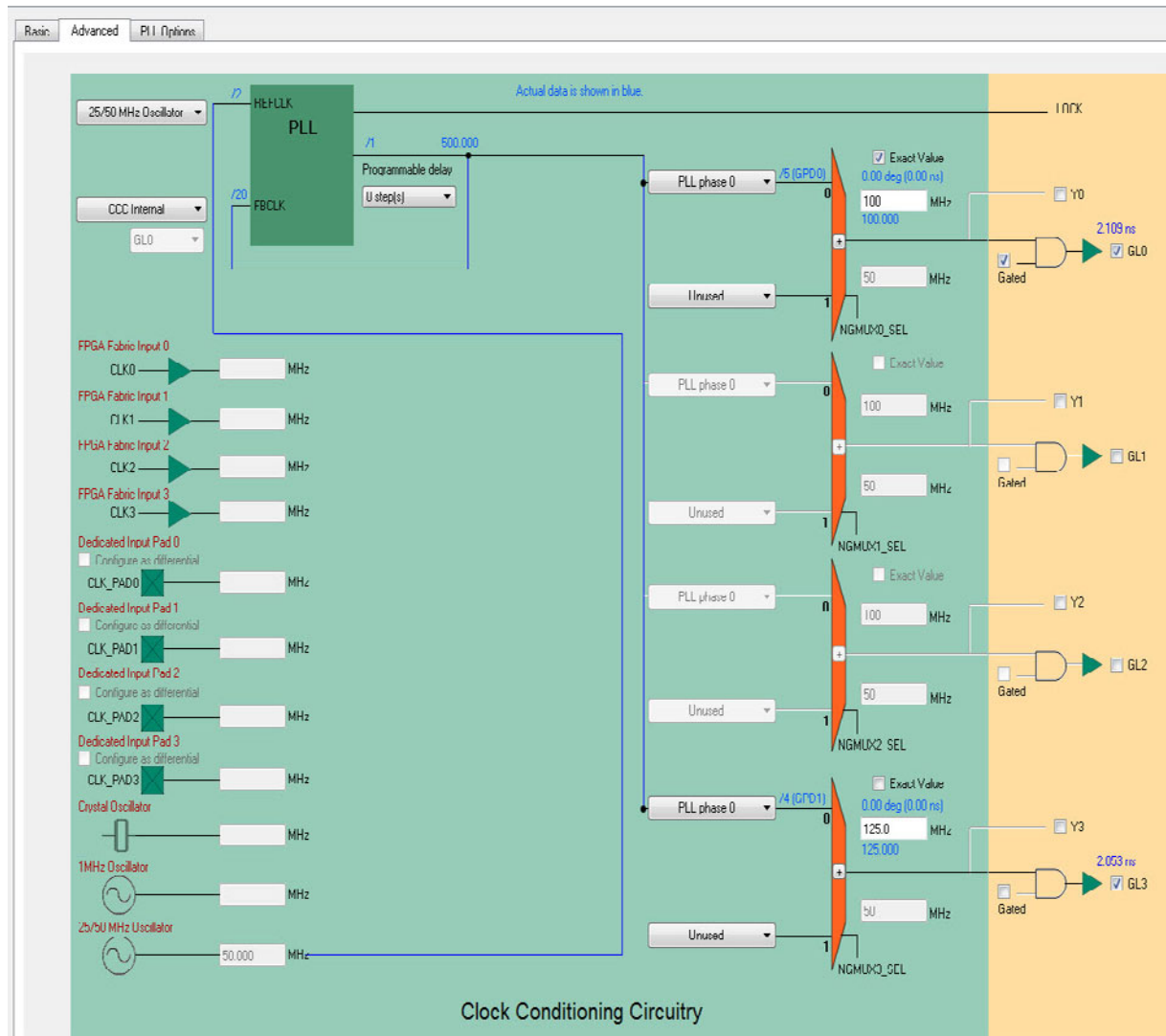


Figure 4-7 • SmartFusion2 MSS and PCIe System Builder CCC Configuration

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.

Libero SoC is able to generate/derive the SDC timing constraints for the clock based on the configuration and the component SDC files.

From the Constraint Manager generate the clock constraint (**Constraint Manager > Timing > Derived Constraints**). The generated clock constraints are placed in the file <root>_derived_constraints.sdc.

```
create_clock -name {PCIE_HPDM0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT} -period 20\
[ get_pins { PCIE_HPDM0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT } ]
```

The CCC generates two clocks from the 50 MHz system clock:

```
create_generated_clock -name {PCIE_HPDMMA_0/CCC_0/GL0} -multiply_by 20 -divide_by 10\
-source [ get_pins { PCIE_HPDMMA_0/CCC_0/CCC_INST/RCOSC_25_50MHZ } ] -phase 0\
[ get_pins { PCIE_HPDMMA_0/CCC_0/CCC_INST/GL0 } ]
create_generated_clock -name {PCIE_HPDMMA_0/CCC_0/GL3} -multiply_by 20\
-divide_by 8 -source [ get_pins { PCIE_HPDMMA_0/CCC_0/CCC_INST/RCOSC_25_50MHZ } ]\
-phase 0 [ get_pins { PCIE_HPDMMA_0/CCC_0/CCC_INST/GL3 } ]
```

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 {PCIE_HPDMMA_0/PCIE_HPDMMA_MSS_0/CLK_CONFIG_APB} -period 40\
[ get_pins { PCIE_HPDMMA_0/PCIE_HPDMMA_MSS_0/MSS_ADLIB_INST/CLK_CONFIG_APB } ]
```

All these clock constraints are placed in the <root>_derived_constraints.sdc file after generation **(Constraint Manager > Timing > Derived Constraints)**.

In the Constraint Manager, associate the file with Synthesis to pass the file to Synplify Pro.

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

```
Finished netlist restructuring (Real Time elapsed 0h:00m:03s; CPU Time elapsed 0h:00m:00s; Memory used current: 159MB peak: 160MB)

@S |Clock Summary
*****

Start      Requested   Requested   Clock
Clock      Frequency   Period      Type
-----
PCIE_HPDMMA_0/CCC_0/GL0      100.0 MHz   10.000      generated (from PCIE_HPDMMA_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT)
PCIE_HPDMMA_0/CCC_0/GL3      125.0 MHz   8.000       generated (from PCIE_HPDMMA_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT)
PCIE_HPDMMA_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT  50.0 MHz   20.000      declared
PCIE_HPDMMA_0/PCIE_HPDMMA_MSS_0/CLK_CONFIG_APB  25.0 MHz   40.000      declared
=====

Finished Pre Mapping Phase.
@N:BN225 : | Writing default property annotation file D:\2Work\M2S_PCIE_MSSHDPMA_DEMO_DF\LiberoProject\PCIE_HPDMMA\synthesis\PCIE_HPDMMA_top.sap.
```

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

SmartFusion2 MSS and PCIe Place and Route Constraints

Associate the <root>_derived_constraints.sdc file to Place and Route in Design Manager to pass the SDC timing constraint file to Place and Route.

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).

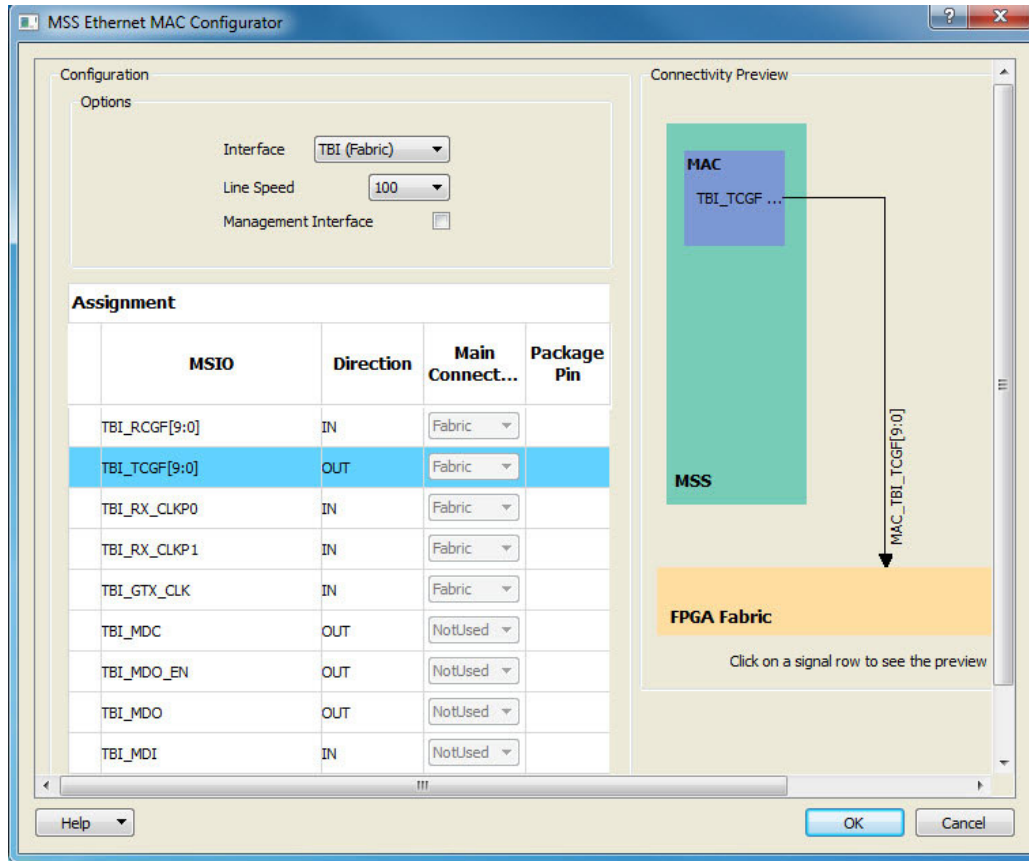


Figure 4-9 • Configuring the MSS Ethernet MAC for the TBI (Fabric) Interface

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.

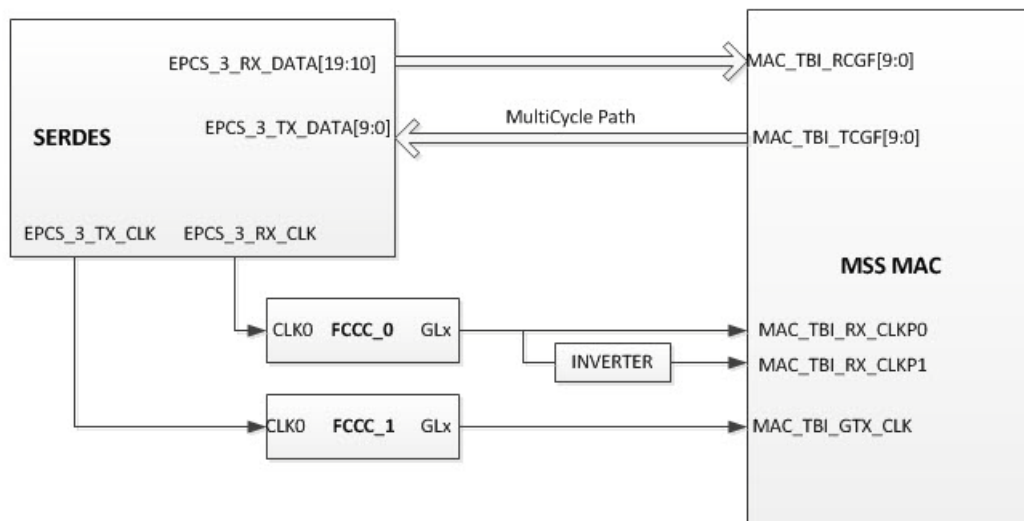


Figure 4-10 • Multicycle Path from MSS MAC to SERDES (Fabric)

Set the multicycle path constraint in an *.SDC file. If you do not, Libero SoC'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.

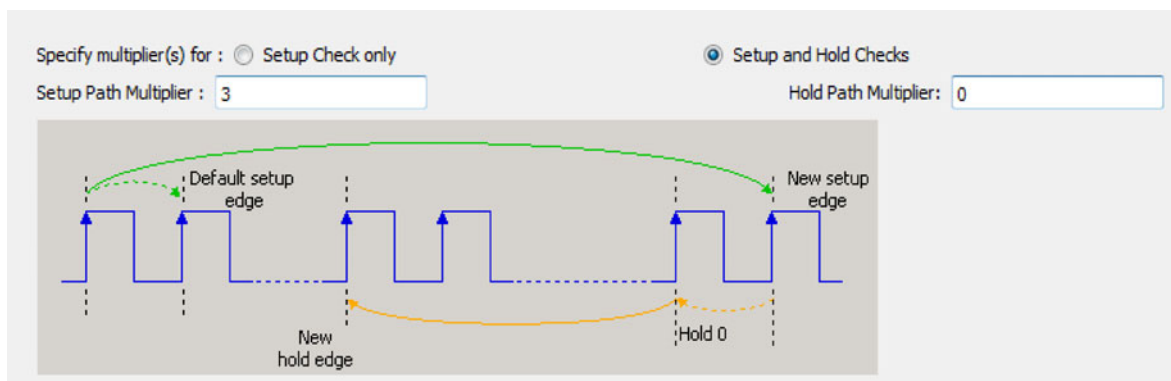


Figure 4-11 • Setup and Hold Timing Check Relations for Multicycle Path

In the Constraint Manager, associate the *.sdc file to Synthesis and Place and Route. The *.sdc file will be used and processed when the tools are run.

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, 408.643.6913

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

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, 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. [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

©2015 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.

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; 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 3,600 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.