Lattice Radiant Timing Constraints Methodology

Application Note

FPGA-AN-02059-1.5

July 2024

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
LDC	Lattice Design Constraints
SDC	Synopsys Design Constraints
FDC	FPGA Design Constraints
PDC	Physical Design Constraints
LSE	Lattice Synthesis Engine
PAR	Place And Route
UDB	Unified Database
SI	Signal Integrity
TCE	Timing Constraints Editor
CPE	Constraints Propagation Engine

1. Introduction

Lattice Radiant® software is the complete design environment for Lattice Semiconductor Field Programmable Gate Arrays (FPGAs). The software includes a comprehensive set of tools for all design tasks, including project management, design entry, simulation, synthesis, place and route, in-system logic analysis, and more.

To ensure the design meets its desired performance goals on the FPGA, it is the responsibility of the users to provide proper timing constraints to the design. The implementation tools in Radiant reads in the constraints provided and works its way to meet the performance requirements. In Radiant, timing constraints are consumed throughout the design implementation flow starting from synthesis to place and route. Physical constraints are only consumed by the back-end flow post-synthesis. Users can still enter the physical constraints using the pre-synthesis constraints file which gets properly passed down to the implementation tools during the flow.

Lattice Radiant supports Lattice Design Constraints (LDC) based on the standard Synopsys™ Design Constraints (SDC) supported by our in-house synthesis tool Lattice Synthesis Engine (LSE). Radiant also supports Synopsys Design Constraints (SDC) file format for both LSE and Synopsys™ Synplify Pro design flows and FPGA Design Constraints (FDC) for Synplify Pro flow. For post-synthesis implementation flow, Radiant supports Physical Design Constraints (PDC) for timing and physical constraints.

This document covers the in-depth timing constraints methodology that is used in Lattice Radiant software.

1.1. Audience

The intended audience for this document includes FPGA design engineers using Lattice Radiant design software. The technical guidelines assume that the readers have some basic knowledge on the SDC constraints usage.

2. Overview

Constraints are consumed throughout the implementation flow starting from Synthesis to Place and Route. The primary goal of the tools is to meet user constraints and performance goals. Figure 2.1 shows an overview of Radiant constraints consumption throughout the implementation flow.

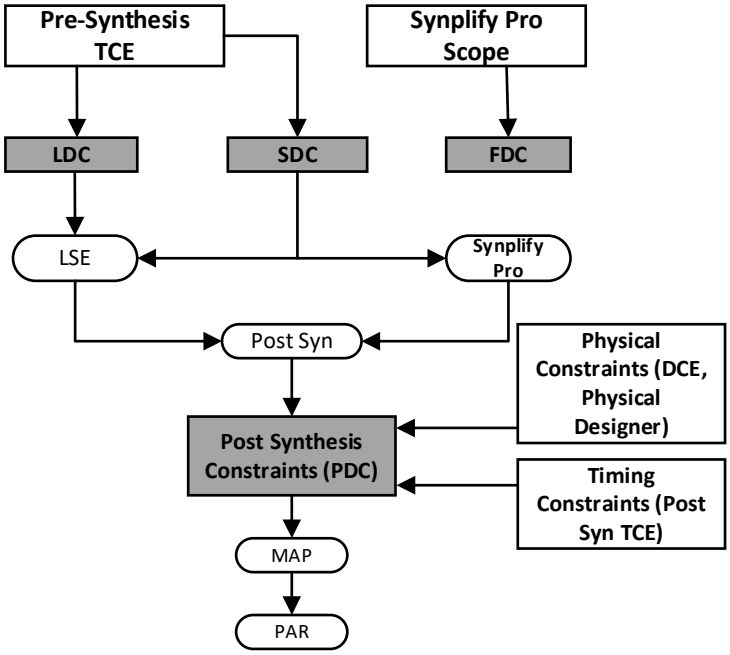


Figure 2.1. Overview of Radiant Constraints Consumption

Radiant supports a subset of standard SDC constraints. The LDC and SDC files allow users to input pre-synthesis timing constraints along with physical constraints and synthesis attributes which will be used for synthesis optimizations and post-synthesis timing analysis. The physical constraints entered in the user LDC or SDC files are not consumed by the synthesis tools. These constraints are written into the database file (.udb) which are then consumed by MAP and PAR engine for implementation during the flow. Users can also enter physical constraints using the PDC file which will be consumed by both MAP and PAR processes. Table 2.1 shows the supported file formats by the implementation tools in Radiant.

Table 2.1. Supported File Formats by the Implementation Tools in Radiant

Constraint type	Implementation Tool
.ldc	LSE
.sdc	LSE & Synplify Pro
.fdc	Synplify Pro
.pdc	MAP & PAR

3. SDC Constraints Supported in Radiant

The section describes the SDC constraints supported in Radiant.

3.1. Clock Constraints

Table 3.1 lists all the clock constraints supported in Radiant.

Table 3.1. Clock Constraints

Constraint	Definition
create_clock	Defines clock constraints for the clocks used in the design
create_generated_clock	Defines generated clocks in the design
set_clock_latency	Specifies source clock latency outside the FPGA
set_clock_uncertainty	Used to over constrain the clock by accounting jitter
set_clock_group	Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis.

3.1.1. create_clock Constraint

The create_clock constraint is used to define primary and virtual clocks in the design. The syntax for the create_clock constraint is:

create_clock -period <period>
[-name <clock name>]
[-waveform <value1 value2>]
[-add]
[get_ports | get_pins | get_nets <source_object>]

Table 3.2. create_clock Constraint Options

Option	Usage
-period	Used to specify clock period
-name	Used to refer the clock in other commands
-waveform	Used to adjust duty cycle and phase
-add	Used to define another clock to an object that has an existing clock

This constraint can be defined using various ways which is described in the Constraints Entry in Radiant section.

Example:

Let us consider the following example design. In Figure 3.1, clkA is a 100 MHz clock that drives the design.

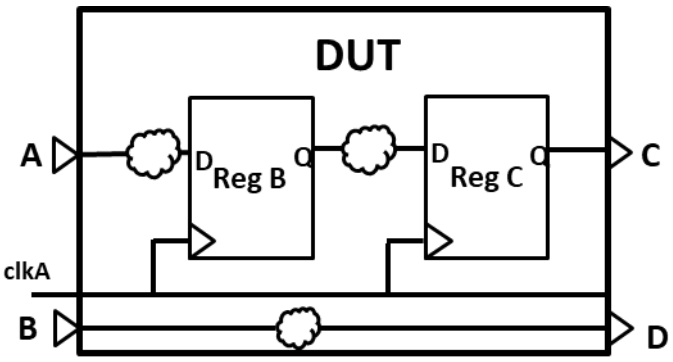


Figure 3.1. DUT

User must define a create_clock constraint at this port clkA. The constraint for clkA with a frequency of 100 MHz with a duty cycle of 50% can be defined as:

create_clock -name clk -period 10 [get_ports clkA]

In general, clocks can be of three clocks:

- **Base clock**: Defined and specified using the create_clock constraint using the clock name and a clock object.
- **Virtual clock**: Defined using the create_clock constraint with a clock name and no object.
  Example: create_clock -name virt_clk -period 10
  This creates a virtual clock called “virt_clk” with a period of 10 ns and is only used for timing analysis purposes.
- **Generated clocks**: These clocks are derived from the base clocks either using PLL or user logic and are defined using the create_generated_clock constraint.

3.1.1.1. Usage Guidelines for create_clock Constraint

- It is always preferred to define clocks on the top-level clock ports.
- If the clocks are defined on the pins/nets, the Radiant timer will not consider the IO buffer delays.

3.1.2. create_generated_clock Constraint

A create_generated_clock constraint is used to define clocks inside the design. For example, a create_generated_clock can be used to define PLL output clocks, clock divider output clocks etc. The syntax for create_generated_clock constraints is:

create_generated_clock -source <clock_source>
[-divide_by <factor>]
[-multiply_by <factor>]
[-duty_cycle <value>] | [-edges <edge specs>]
[-invert]
[-name <clock name>]
[-add]
[-master_clock<clock>]
[get_pins | get_nets <pin/net name>]
<target_object>

Table 3.3. create_generated_clock Constraint Options

Option	Usage
-source	Used to specify the clock source
-divide_by	Used to specify the frequency divide factor
-multiply_by	Used to specify the frequency multiply factor
-duty_cycle	Used to specify the duty cycle (in percentage) if frequency multiplication is used
-edges	Used to specify a list of positive integers that represents the edges from the source clock that are to form the edges of the generated clock
-invert	Inverts the generated clock signal
-add	Used to add the clock to the existing clock
-master_clock	Used to specify the master clock for the generated clock object if there are multiple clocks found on the source object

Example: Let us consider the Figure 3.2. In this example, clk is a 100 MHz input clock to a PLL which generates three clocks 50 MHz, 200 MHz, and 75 MHz.

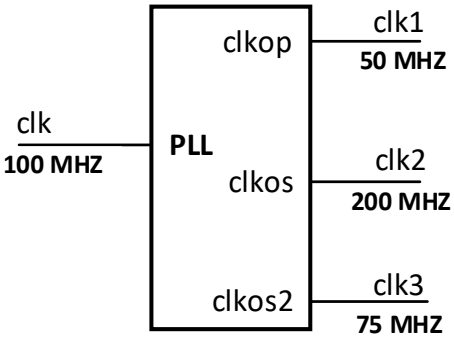


Figure 3.2. PLL with Clocks

To constrain the above example, user may use the following constraints:

- *## Define clock constraint for clk*
  create_clock -name clk -period 10 [get_ports clk]
- *## Define a 50 MHz generated_clock constraint to constrain clk1*
  create_generated_clock -name clk1 -source [get_ports clk] -divide_by 2 [get_pins clkop]
- *## Define a 200 MHz generated_clock constraint to constrain clk2*
  create_generated_clock –name clk2 -source [get_ports clk] -multiply_by 2 [get_pins clkos]
- *## Define a 75 MHz generated_clock constraint to constrain clk3*
  create_generated_clock –name clk3 -source [get_ports clk] -multiply_by 3 -divide_by 4 [get_pins clkos2]

3.1.2.1. Usage Guidelines

- Must define a primary clock source using a create_clock constraint before defining the create_generated_clock constraint.
- Must define a fixed relationship between the primary and secondary clocks.
- If PLL or OSC hard IP is used, the Radiant timer automatically infers the generated clock constraints for the PLL output clocks. Users must define the input clock to the PLL.

3.1.2.2. PLL Constraint Guidelines

Users must follow the following guidelines to constrain the PLL clocks correctly:

- Clock should be defined on the top-level port that feeds the reference clock pin of the PLL using the create_clock constraints. If this is not followed:
  - Insertion delay will be inaccurate.
  - The relationship between PLL output clocks will be lost if clocks are defined at the output of the PLL using create_clock constraint.
- create_generated_clocks could either be defined by the user or preferably be generated by the timing engine at the output pins of the PLL.

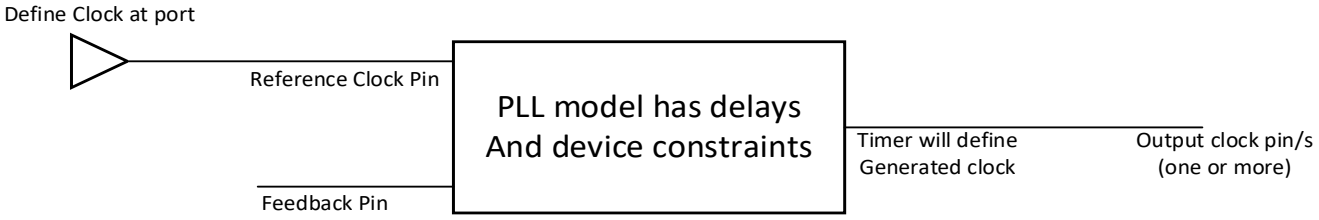


Figure 3.3. PLL Using the create_clock Constraints

3.1.3. set_clock_latency Constraint

The set_clock_latency constraint is used to specify the source clock latency outside the FPGA. This constraint can also be used to specify the delay from virtual clock to actual port arrival, etc. The syntax for set_clock_latency constraint is:

set_clock_latency <value>
-source
[-rise]
[-fall]
[-early | -late]
[get_clocks <clock name>]

Table 3.4. set_clock_latency Constraint Options

Option	Usage
-source	Used to specify the clock source
-rise	Indicates that delay is to apply only to rise clock**. By default, delay is applied to both rise and fall clock latency**
-fall	Indicates that delay is to apply only to fall clock. **By default, delay is applied to both rise and fall clock latency.**
-early	Indicates that delay is to apply only to early clock source latency (Fastest path). **By default, if -source is specified, delay is applied to both late and early clock source latency.**
-late	Indicates that delay is to apply only to late clock source latency (Longest path). **By default, if -source is specified, delay is applied to both late and early clock source latency.**

Example: Let us consider Figure 3.4. Let's assume there is a source clock latency of 0.5 ns outside the FPGA from the clock source before the clock reaches the FPGA boundary.

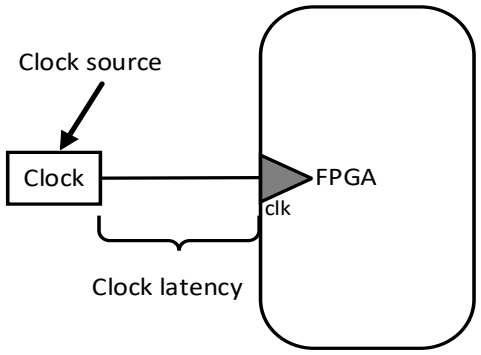


Figure 3.4. Clock Latency

User can use the following constraint to specify the clock latency:

set_clock_latency -source 0.5 [get_clocks clk]

Here's an example to source latency delay of the longest clock path using the using the -late option.

set_clock_latency -source -late 1.0 [get_clocks clk]

Here's an example to source latency delay of the shortest clock path using the using the -early option.

set_clock_latency -source -early 0.5 [get_clocks clk]

3.1.3.1. Usage Guidelines

- The Min and Max delay are used to find the minimum and maximum arrival times even when there is a single path.
- For setup analysis, TA uses the late clock latency for the data arrival path and the early clock latency for the clock arrival path.
- For hold analysis, TA uses the early clock latency for the data arrival time and the late clock latency for the clock arrival time.
- For Nexus devices, the maximum is only for setup calculation.

3.1.4. set_clock_uncertainty Constraint

The set_clock_uncertainty constraint is usually used to over constrain the design and for taking Jitter into account.

Uncertainty also covers all other static/dynamic timing uncertainties too - ->: skew, crosstalk, SI etc. The syntax for set_clock_uncertainty constraint is:

set_clock_uncertainty [-setup]
[-hold]
[-from <clock>]
[-to <clock>]
<uncertainty_value>
[<clock_list>]

Table 3.5. set_clock_uncertainty Constraint Options

Option	Usage
-setup	Indicates that uncertainty applies only to setup checks. **By default, uncertainty applies to both setup and hold checks**
-hold	Indicates that uncertainty applies only to hold checks. **By default, uncertainty applies to both setup and hold checks**

Example:

For Certus-NX according to the datasheet, the output clock cycle-to-cycle jitter is 250ps for clocks ≥ 200 MHz. But for STA, the uncertainty constraint must be half of period jitter which is explained in the net section. The constraint for set_clock_uncertainty for clocks > 200 MHz can be:

set_clock_uncertainty -setup 0.125 [get_clocks <clock>]

3.1.4.1. Usage Guidelines

On chip variations such as jitter may cause clock edges to shift left or right thus shrinking or expanding the clock period. This may cause unintended timing violations on the hardware. If the set _clock_uncertainty constraint is not used, then these violations caused by jitter are not reported by Radiant. There may be functional failures on the hardware because these jitters are not accounted for by timing analysis. If set_clock_uncertainty constraint is added to the clocks, then uncertainties due to jitter will be properly captured for timing analysis. If there are timing violations, Radiant place & route engine may spend additional efforts to resolve timing violations whenever it is possible.

Let's look at the following example of a simple register to register data transfer. Let us assume that the clock period is 5 ns and the data path delay is 4.8 ns between the registers. Thus, we have a positive setup slack of 0.1 ns.

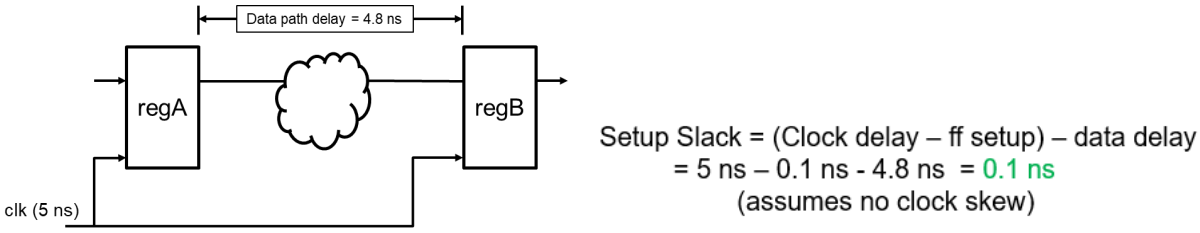


Figure 3.5. Calculating Positive Setup Slack

For synchronous logic, shorter clock periods decrease the setup time margin. Period jitter can be used to calculate the minimum period of a system clock. Thus, minimum clock period = nominal clock period – (period jitter peak-peak)/2 and clock uncertainty would be (Period Jitter Peak-Peak)/2.

Assuming the target device is Certus-NX and we are using an integer-N PLL, the recommended output clock period jitter for clocks ≥ 200 MHz is 0.125 (which is 0.250/2) ns based on the Certus-NX Family Data Sheet (FPGA-DS-02078).

AC Characteristics						
t_{DT}	Output Clock Duty Cycle	—	45	—	55	%
t_{PH}[4]	Output Phase Accuracy	—	–5	—	5	%
t_{OPJIT}[1]	Output Clock Period Jitter	$f_{OUT} \geq 200$ MHz	—	—	250	ps p-p
		$f_{OUT} < 200$ MHz	—	—	0.05	UIPP
	Output Clock Cycle-to-Cycle Jitter	$f_{OUT} \geq 200$ MHz	—	—	250	ps p-p
		$f_{OUT} < 200$ MHz	—	—	0.05	UIPP
	Output Clock Phase Jitter	$f_{PFD} \geq 200$ MHz	—	—	250	ps p-p
		60 MHz ≤ f_{PFD} < 200 MHz	—	—	350	ps p-p
		30 MHz ≤ f_{PFD} < 60 MHz	—	—	450	ps p-p
		18 MHz ≤ f_{PFD} < 30 MHz	—	—	650	ps p-p
	Output Clock Period Jitter (Fractional-N)	$f_{OUT} \geq 200$ MHz	—	—	350	ps p-p
		$f_{OUT} < 200$ MHz	—	—	0.07	UIPP
	Output Clock Cycle-to-Cycle Jitter (Fractional-N)	$f_{OUT} \geq 200$ MHz	—	—	400	ps p-p
		$f_{OUT} < 200$ MHz	—	—	0.08	UIPP

Figure 3.6. AC Characteristics

Considering this uncertainty to the clock clk, our setup slack calculation now changes to,

Setup Slack = Clock delay – ff setup – data delay – uncertainty

= 5 – 0.1 – 4.8 – 0.125

= –0.025 ns

This path may cause functional failures on the board due to timing violations introduced by the PLL induced variations. If uncertainties are added to the clocks, Radiant can catch these timing errors.

- Either PAR puts more effort in meeting the timing depending on the slack
- Or users can follow timing closure techniques to close timing on the failed paths.

When the set_clock_uncertainty constraint is used,

- If no -setup or -hold options are provided, the constraint is applied to both setup and hold analysis.
- If -setup option is used, then the constraint only applies to setup analysis. During setup analysis, the uncertainty is subtracted from the destination clock.

```
Destination Clock Path
 Shown in:  Netlist Analyzer  Physical Designer Placement Mode  Physical Designer Routing Mode
Name                                      Cell/Site Name  Delay Name          Incr       Arrival/Required Time  Fanout
----------------------------------------  --------------  ----------------  ---------  ----------------------  ------
                                                          CONSTRAINT            0.000                   5.000  1
clk                                       count           CLOCK LATENCY         0.000                   5.000  1
clk                                                       NET DELAY             0.000                   5.000  1
clk_ibuf.bb_inst/B->clk_ibuf.bb_inst/O    HPIO_CORE_AA2   PAD2DI_DEL            1.546                   6.546  8
clk_c                                                     NET DELAY             2.935                   9.481  8
{c_Z[1]/CLK    c_Z[0]/CLK}                                CLOCK PIN             0.000                   9.481  1
                                                          Uncertainty         -(0.125)                  9.356
                                                          Common Path Skew      0.048                   9.404
                                                          Setup time          -(-0.066)                 9.470
----------------------------------------  --------------  ----------------  ---------  ----------------------  ------
```

Figure 3.7. Uncertainty is subtracted from Destination Clock Path

- For a single clock timing analysis, use -setup option only as the comparison will be at the same edge and hold uncertainty will be 0.
- There may be cases where uncertainty must be applied to hold analysis also.
- If -hold option is used, then the constraint only applies to hold analysis. During hold analysis, the uncertainty is added to the destination clock.

```
Destination Clock Path
 Shown in:  Netlist Analyzer  Physical Designer Placement Mode  Physical Designer Routing Mode
Name                                      Cell/Site Name  Delay Name          Incr     Arrival/Required Time  Fanout
----------------------------------------  --------------  ----------------  -------  ----------------------  ------
                                                          CONSTRAINT          0.000                   0.000  1
clk                                       count           CLOCK LATENCY       0.000                   0.000  1
clk                                                       NET DELAY           0.000                   0.000  1
clk_ibuf.bb_inst/B->clk_ibuf.bb_inst/O    HPIO_CORE_AA2   PAD2DI_DEL          0.964                   0.964  8
clk_c                                                     NET DELAY           2.280                   3.244  8
{c_Z[13]/CLK    c_Z[12]/CLK}                              CLOCK PIN           0.000                   3.244  1
                                                          Uncertainty         0.125                   3.369
                                                          Common Path Skew   -0.024                   3.345
                                                          Hold time           0.080                   3.425
----------------------------------------  --------------  ----------------  -------  ----------------------  ------
```

Figure 3.8. Uncertainty is added to the Destination Clock Path

- In the case of clock domain crossing paths, only destination clock uncertainty is subtracted/added for setup/hold analysis.

3.1.5. set_clock_group Constraint

The set_clock_group constraint specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis. The syntax for set_clock_groups constraint is:

set_clock_groups -group
<clock_list>
<-logically_exclusive | -physically_exclusive | -asynchronous>

Table 3.6. set_clock_group Constraint Options

Option	Usage
-logically exclusive	Both the clocks (or clock groups) can exist (can be active) simultaneously in the design and the clocks have SI impact on each other's clock domain
-physically exclusive	Both the clocks (or clock groups) cannot exist (cannot be active) simultaneously in the design and the clocks have no SI impact on each other's clock domain
-asynchronous	Specifies that the clock groups are asynchronous to each other (while the Radiant software assume all clocks defined by create_clock and create_generated_clock are synchronous)

Example:

set_clock_groups -asynchronous -group [get_clocks clka_port] -group [get_clocks clkb_port]

3.1.5.1. Usage Guidelines

- This constraint must be used only if the user is sure of the clock exclusiveness.
- If the clocks are not fully exclusive, consider using other clock exception constraints.

3.2. Input/Output Delay Constraints

Table 3.7 lists the input/output delay constraints supported in Radiant.

Table 3.7. Input/output Delay Constraints Supported in Radiant

Constraint	Definition
set_input_delay	Sets input delay on input ports with respect to a clock signal
set_output_delay	Sets output delay on outports ports with respect to a clock signal

3.2.1. set_input_delay Constraint

In Radiant, to constrain the input ports of the FPGA, the set_input_delay constraint must be specified relative to a clock. The set_input_delay constraint is used to specify the external delay of the signal to the fabric input port. The syntax for set_input delay constraint is:

set_input_delay -clock <clock_name>
[-clock_fall]
[-max]
[-min]
[-add_delay]
<delay> <port_list>

Table 3.8. set_input_delay Constraint Options

Option	Usage
-clock_fall	Specifies that the delay is relative to the falling edge of the clock.
-min	Used to specify the best case for the signal at the input port. **Used to perform hold checks.**
-max	Used to specify the worst case for the signal at the input port. **Used to specify Setup checks.**
-add_delay	Used to define more than one input delay on a given port.

Note: When -min or -max are not specified, the same specified value is considered for both setup and hold calculation.

Example:

Let us consider Figure 3.9. In this example, port A is an input signal to the FPGA which is driven by an external design outside the FPGA.

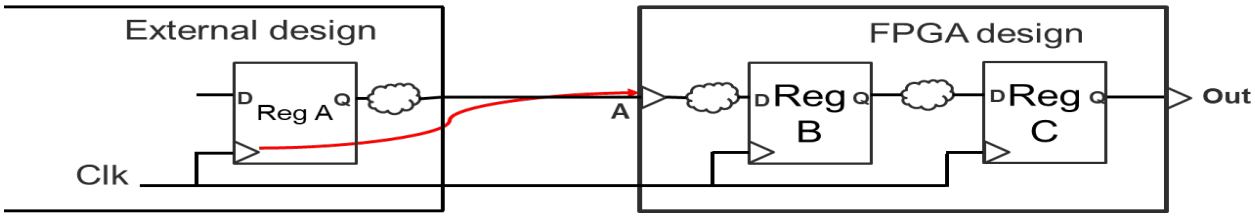


Figure 3.9. FPGA Driven by an External Design Outside the FPGA

Assuming the input clock Clk is 100 MHz and output data from the external design takes 1 ns to reach the FPGA input port A, user can use the following constraints to constraint input port A:

create_clock -name Clk -period 10 [get_ports Clk]

set_input_delay -clock Clk 1 [get_ports A]

Note: If the external clock relationship is unknown, then Clk must be treated as a virtual clock.

3.2.2. set_output_delay Constraint

In Radiant, to constrain the output ports of the FPGA, the set_output_delay constraint must be specified relative to a clock. The syntax for the set_output delay constraint is:

set_output_delay -clock <clock_name>
[-clock_fall]
[-max]
[-min]
[-add_delay] <delay>
<port_list>

Table 3.9. set_output_delay Constraint Options

Option	Usage
-clock_fall	Specifies that the delay is relative to the falling edge of the clock
-max	Used to specify the worst case for the signal at the output port. **Used to perform setup checks**
-min	Used to specify the best case for the signal at the output port. **Used to specify hold checks**
-add_delay	Used to define more than one output delay on a given port

Note: When -min or -max are not specified, the same specified value is considered for both setup and hold calculation

Example 1: Case when external circuit is driven by external clocks

Things to consider:

- Identify relationship between external clock and clock going into FPGA and set a clock latency on the clock input pin.
- Compute the external requirement by figuring out the delay to the clock pin of the external sequential element and the delay from the FPGA output to the external sequential element.
- Define the set_output_delay using the external clock and the external delays.

Consider the circuit in Figure 3.10.

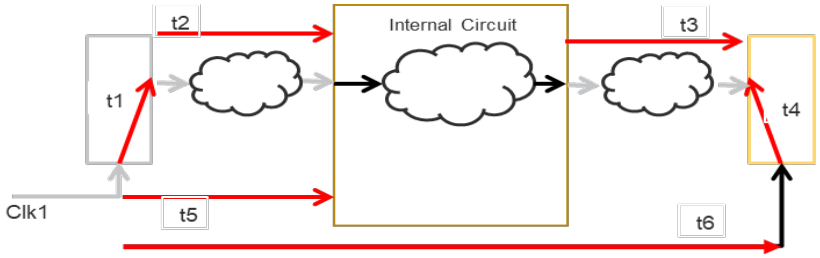


Figure 3.10. External Circuit is Driven by External Clocks

Based on the above points, the following constraints can be applied to the above circuit:

create_clock –period T –name Clk1

create_clock –period T –name Clk2 [get_ports Clk]

set_clock_latency t5 [get_clocks Clk2]

set_input_delay t1 + t2 –clock [get_clocks Clk1] [all_inputs]

create_clock –period T –name Clk3

set_output_delay t3 + t4 – t6 –clock [get_clocks Clk3] [all_outputs]

Example 2: Case when clock from inside the FPGA drives external clock

Things to consider:

- Define a generated clock on the port where the clock comes out.
- Compute external requirements by finding the clock and data delays to the external sequential element.
- Define the set_output_delay constraint using the generated clock and the external delays.

Consider the circuit in Figure 3.11.

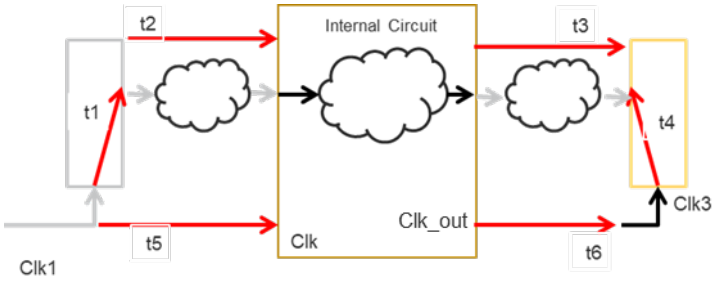


Figure 3.11. Clock from Inside the FPGA Drives External Clock

Based on the above points, the following constraints can be applied to the above circuit:

create_clock –period T –name Clk1

create_clock –period T –name Clk2 [get_ports Clk]

set_clock_latency t5 [get_clocks Clk2]

set_input_delay t1 + t2 –clock [get_clocks Clk1] [all_inputs]

create_generated_clock -name clk3 -source [pll/lscc_pll_inst/gen_no_refclk_mon.u_PLL.PLL_inst/ClkOS] [get_ports Clk_out] *## assuming the clock was generated using a PLL*

set_output_delay t3 + t4 –clock [get_clocks Clk3] [all_outputs]

3.3. Timing Exception Constraints

Table 3.10. Timing Exception Constraints

Constraint	Definition
set_false_path	Used to specify paths that are considered false and excluded from timing analysis
set_max_delay	Specifies the maximum delay for the timing paths
set_min_delay	Specifies the minimum delay for the timing paths
set_multicycle_path	Defines Path that takes multiple clock cycles

3.3.1. set_false_path Constraint

set_false_path constraint is used to specify paths that must be excluded from timing analysis. For example, paths that cross clock domains and the clocks are asynchronous to each other can be excluded from timing analysis by specifying set_false_path constraint between them. The syntax for set_false_path constraint is:

set_false_path [(-from | rise_from | fall_from)<object_list>]
[-through <object_list>]
[(-to | -rise_to | -fall_to) <object_list>]
[-setup]
[-hold]

Table 3.11. set_false_path Constraint Options

Option	Usage
-from	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-to	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-rise_from	Same as -from but the path must rise from the specified object
-fall_from	Same as -from but the path must fall from the specified object
-rise_to	Same as -to but the path must rise to the specified object
-fall_to	Same as -to but the path must fall to the specified object
-through	Specifies false paths through nets
-setup	This option eliminates setup timing analysis for specified timing path
-hold	This option eliminates hold timing analysis for specified timing path

Example: set_false_path -from clk1 -to clk2

3.3.2. set_max_delay Constraint

This constraint is used to specify maximum delay for the timing paths. The syntax for set_max_delay constraint is:

set_max_delay [(-from)<port_object or cell_object>]
[-through <port_object or cell_object>]
[-to <port_object or cell_object>]
-datapath_only
<delay_value>

Table 3.12. set_max_delay Constraint Options

Option	Usage
-from	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-to	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-through	Specifies false paths through nets
-datapath_only	Considers datapath only for timing calculation

3.3.3. set_min_delay Constraint

This constraint is used to specify minimum delay for the timing paths. The syntax for set_min_delay constraint is:

set_min_delay [(-from)<port_object or cell_object>]
[-through port_object or cell_object]
[-to <port_object or cell_object>]
<delay_value>

Table 3.13. set_min_delay Constraint Options

Option	Usage
-from	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-to	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-through	Specifies false paths through nets

3.3.4. set_multicycle_path Constraint

This constraint is used to define paths that take multiple clock cycles. The syntax for set_multicycle_path constraint is

set_multicycle_path <ncycles>
[(-from | rise_from | fall_from)<object_list>]
[-through <object_list>]
[(-to | -rise_to | -fall_to) <object_list>]
[-setup | -hold] [-start | -end]
<delay_value>

Table 3.14. set_multicycle_path Constraint Options

Option	Usage
ncycles	Number of cycles the datapath must have for setup check
-from	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-to	Specifies start points (clocks, ports, pins, or cells) of disabled paths
-through	Specifies false paths through nets

Example: set_multicycle_path -from [get_cells {rd_grey_sync_r[*]}] 2

4. Constraints Entry in Radiant

This section describes the various ways users can enter constraints in the Lattice Radiant design software.

4.1. Using Pre-Synthesis Timing Constraints Editor

The Pre-synthesis Timing Constraints Editor (TCE) can be accessed from the Tools menu. This Pre-Synthesis TCE can be used to enter logical domain timing constraints based on the user design. When Pre-Synthesis TCE is invoked, Radiant compiles the user design and opens a window where users can constrain the design objects such as ports, pins, registers, and nets with various supported timing constraints and attributes. The entered constraints can then be saved into an LDC file for LSE flow or into an SDC file for LSE or Synplify Pro flows. Although there may be some limitations of using the SDC file format, the advantage of using an SDC file format is it is supported by both LSE and Synplify Pro tools. Please refer to Radiant help "Unified Constraints Flow → Known Limitations" for the list of limitations.

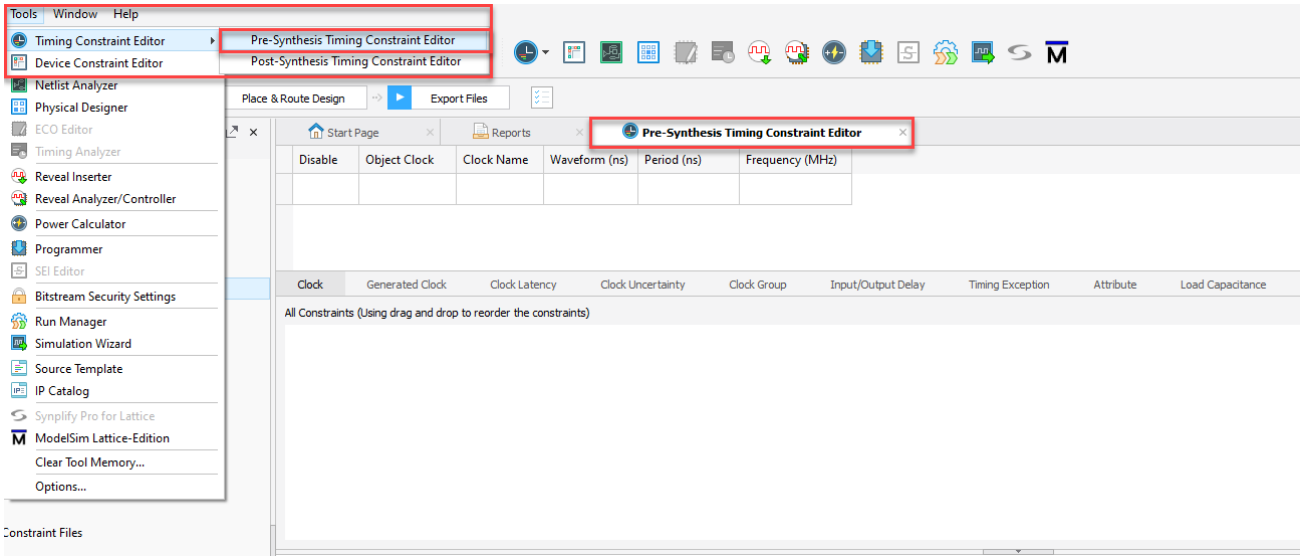


Figure 4.1. Pre-Synthesis Timing Constraints Editor Tools Menu

TCE also has a Design Rule Check (DRC) function which lets the user know if the constraint is incorrect or if the entered design object is not valid. DRC checks are performed two ways. The first one is the Real time DRC checks. These checks are enabled by default and check the validity of the constraint and objects in real time. The second DRC check happens when users save the constraints into an LDC or SDC file. These setting can be changed from the Tools menu: Tools → Options → Tools → Timing Constraints Editor. Here, users can change the behavior of the DRC checks.

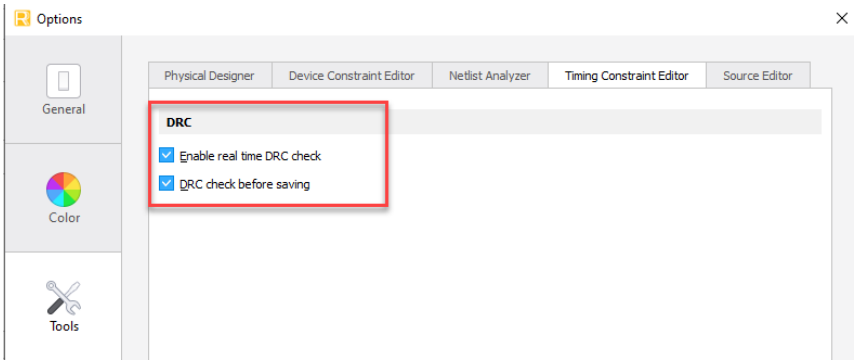


Figure 4.2. DRC Timing Constraints Editor Options

4.2. Using Post-Synthesis Timing Constraints Editor

The Post-synthesis Timing Constraints Editor (TCE) will be available post synthesis and can be accessed from the Tools menu as shown in the figure below. The Post-Synthesis TCE can be used to enter physical domain timing constraints. When Post-Synthesis TCE is invoked, Post-Synthesis TCE reads in the post-synthesis database (*impl_1_syn.udb) file to populate all the post-synthesis netlist objects in the physical domain. The entered constraints can then be saved into a PDC file with an extension of .pdc. This PDC file will then be consumed by MAP and PAR processes.

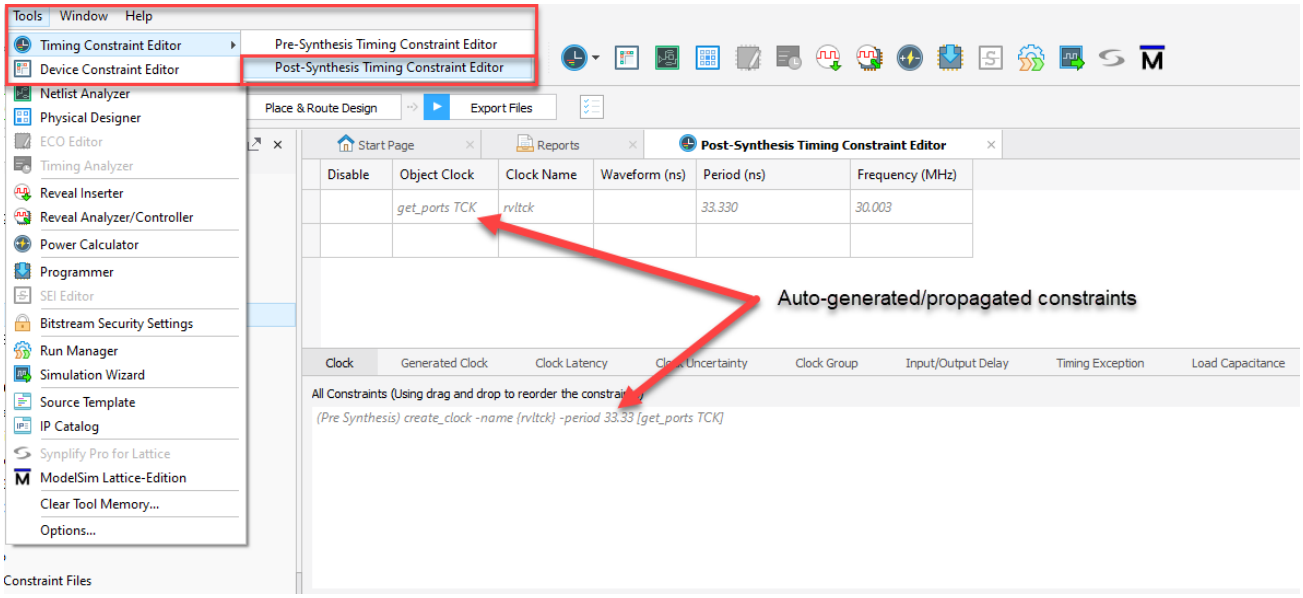


Figure 4.3. Post-synthesis Timing Constraints Editor Tools Menu

The Post-Synthesis TCE also displays auto-derived device constraints and propagated constraints from a pre-synthesis constraints file which will be greyed out. The greyed-out constraints cannot be edited. To edit a greyed-out constraint, copy the constraint by right-clicking on the constraint and choose “copy constraint”. This pastes the constraint on the next line.
Note: The auto generated constraints cannot be removed from the Post-Synthesis TCE.

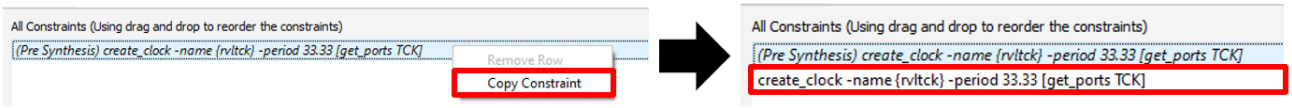


Figure 4.4. To Edit a Greyed-out Constraint

4.3. Using Manual Constraints Entry

Constraints can also be entered manually using the source editor. To create a constraints file, from the File List Window, right-click on the **Pre-synthesis/Post-Synthesis Constraints file → Add → New File**.

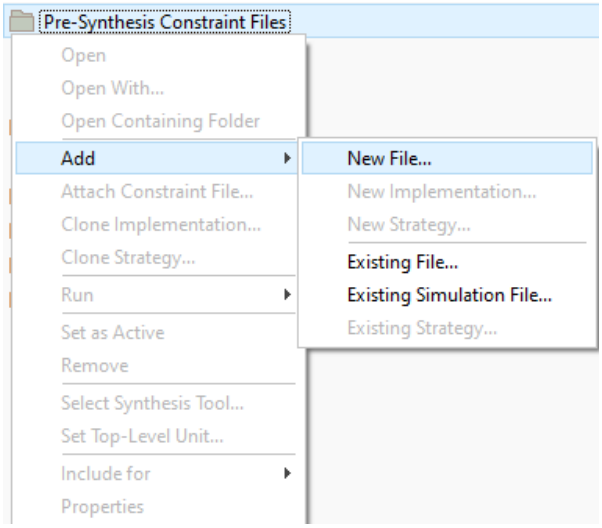


Figure 4.5. Create a Constraints File

Here, users can add LSE Design Constraints File (LDC file) if LSE flow is used, Post-synthesis Constraint Files (PDC file) or Pre-Synthesis-Constraint Files (SDC Files) which support both LSE and Synplify Pro.

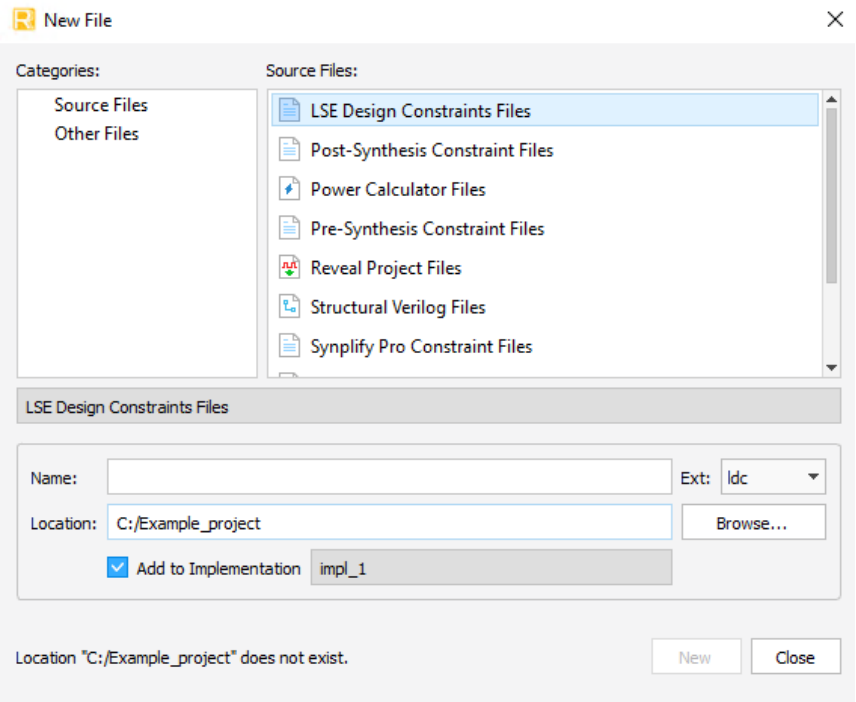


Figure 4.6. Adding LSE Design Constraints File

Apart from this, users can also import an existing LDC, SDC or a PDC file choosing **Add→ Existing File** option from the **File List Window**.

5. Timing Constraints Effect on Implementation Process

Timing Constraints are used for timing driven synthesis and Place and Route (PAR). In Lattice Radiant software tool flow, Synplify Pro, LSE, and PAR engine are all timing driven by default. To maximize the effect of timing driven synthesis, make sure that the LSE strategy option “Optimization Goal” is set to Timing and Synplify Pro strategy option “Area” unchecked.

This section describes the effect of timing constraints on the implementation process.

5.1. Timing Constraints Effect on Synthesis

In Radiant, both LSE and Synplify Pro flows are timing driven. Synthesis tools use user provided pre-synthesis timing constraints to optimize the circuit during synthesis process. The timing driven behavior is turned on by default in Radiant. This behavior can be changed to area driven using the strategy options.

Consider the below example. In this circuit, let us assume that there is a critical path in the logic driven by the signal X. Now, since the synthesis tools are timing driven, they apply timing driven optimization techniques as much as possible to help eliminate the critical paths wherever possible. This circuit preserves all the logic and eliminates the critical path.

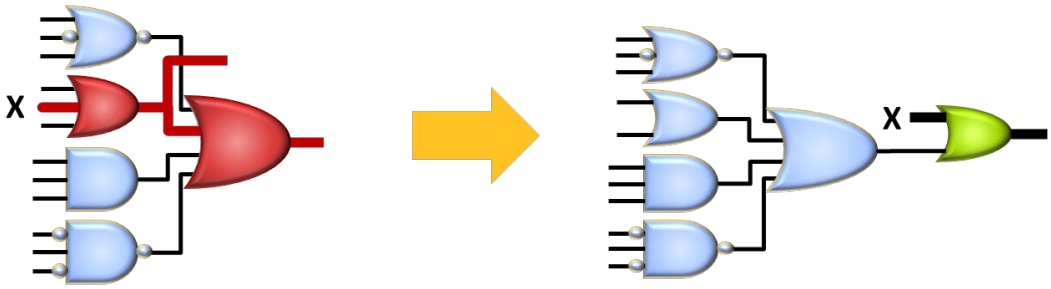


Figure 5.1. Timing Constraints Circuit

If users do not provide any pre-synthesis constraints, synthesis uses a default of 200 MHz for Nexus and Avant devices for all the clocks in the design and will try to optimize the performance of the design for this frequency. This default frequency can be changed using the synthesis strategy options for both LSE and Synplify Pro.

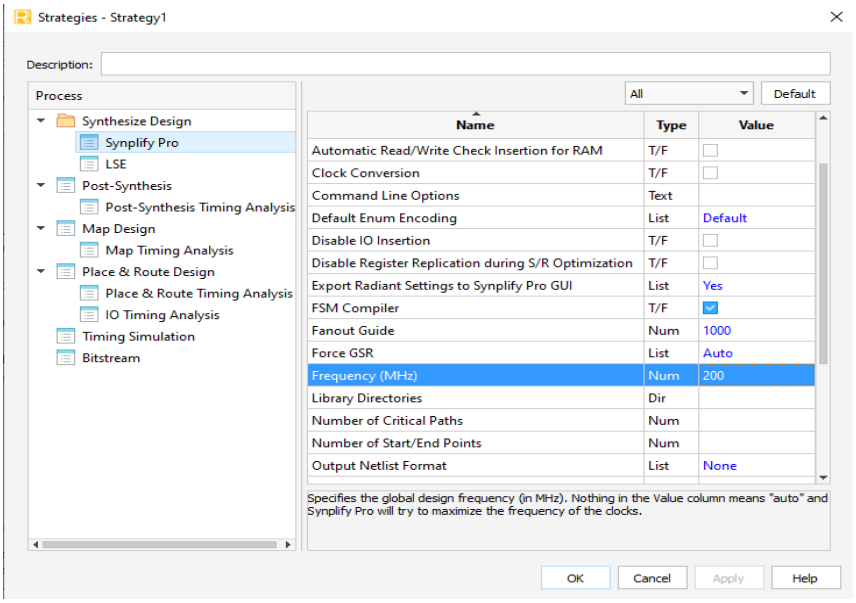


Figure 5.2. Synthesis Strategy Options for Both LSE and Synplify Pro

One of the advantages of using pre-synthesis constraints file is to over constraining synthesis to meet the desired fmax after implementation. Since synthesis is timing driven, the user can constrain synthesis tightly by specifying a faster clock and relax it during PAR to meet the timing requirement.

5.2. Timing Constraints Effect on MAP

In Radiant, timing constraints do not have any effect on MAP optimizations. For Nexus devices, Radiant MAPPER packs LUTs and direct flip flop loads during MAP. MAP also converts all the timing constraints into SLICE level constraints and saves it into the MAP UDB which then goes as an input to Place and Route.

5.3. Timing Constraint Effect on Place and Route

Radiant Place and Route engine is timing driven by default. Place and Route engine uses timing constraints provided by the user to optimally place and route the design to meet user performance goals. The Place and Route engine uses advanced optimization techniques to try its best to correct any hold time violations from the user design. Please note that the timing constraints provided by the users also affect the Place and route run time. The tighter the constraints, the longer the run-time. The default behavior of timing driven place and route can be disabled using the Place and Route Design strategy options.

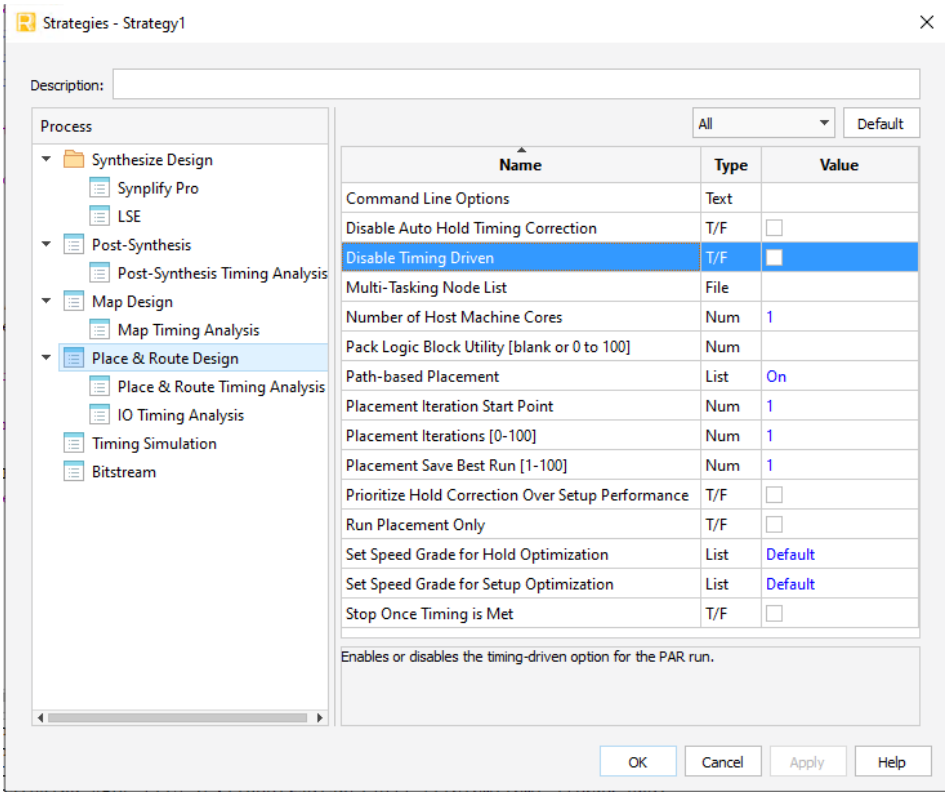


Figure 5.3. Place and Route Design Strategy Options

6. User Constraints Storage During the Implementation Process

The constraints provided by the users are processed in each of the implementation stages and are stored in the respective unified design database (UDB) files. Table 6.1 provides the information on the contents stored in the UDB after each of the implementation stage.

Table 6.1. Contents stored in UDB

Processed Stage	Constraints Storage	Contents
Synthesis	Post Synthesis UDB	Logical Netlist + Timing Constraints
MAP	MAP UDB	Physical Netlist + Timing Constraints
PAR	PAR UDB	Physical Netlist + Routing + Timing Constraints

7. Constraints Propagation Engine

Constraints Propagation Engine is a feature in Radiant SW tool designed to propagate sub-hierarchical constraints and to resolve conflicting constraints between user constraints and IP constraints. Constraints Propagation Engine or CPE compiles all input constraints from multiple .ldc flies from IP and user constraints and creates an effective .ldc file which will be consumed by the synthesis tools. CPE executes right before synthesis begins and requires no user action. Please note that CPE only executes when a user design includes a .ipx with ldc/sdc files present. CPE flow does not support .fdc files.

7.1. CPE Rules

Constraints propagation Engine is executed only when there is an IP with ldc constraints file is instantiated in the user design. CPE does not resolve conflicts within user constraints. User constraints conflicts are handled by Radiant timer. Constraints resolution rules apply only if there is a conflict between user constraints and IP constraints.

Rule 1: Create_clock constraint on an IP port will be ignored

- **Constraint example 1 on an IP port:** create_clock -name {clk_ip_a} –period 10 [get_ports clk]
- **Constraint example 2 on an IP port**: create_clock -name {clk_ip_b} –period 10 [get_ports clk]
- **Resolution**: IP level create clock constraint is Ignored.
- **Reason**: Clocks should always be defined on the top-level ports
  - Clocks on the input side of an IP are clocks that could potentially drive other circuits. In addition, such clocks could give rise to incorrect slack calculations at input ports, output ports and inter-clock paths if defined on the IP.
- **User Action**: Re- define the clock constraint on the top module ports.
- **Example**: create_clock -name clk -period 10 [get_clocks clk_in]

Rule 2: Input/output delay constraint on an IP port will be ignored

- **Constraint**: set_input_delay -clock [get_clock virt_clk] 9 [get_ports in1]
- **Resolution**: Ignored.
- **Reason**: IP Level input delay is not propagated.
- **User Action**: Re- define the constraint at the top-level input ports.
- **Example**: set_input_delay -clock [get_clock virt_clk] 9 [get_ports in_top1]

Note*:* If input/output delay constraints on IP ports come with pads (IO Buffers), only then the constraint will be propagated.

Rule 3: set_clock_groups constraints will be ignored

- **Constraint:** set_clock_groups [get_clocks clk] -group [get_clocks clk2]
- **Resolution**: Ignored.
- **Reason**: clock group constraints may be hazardous if these clocks are used in other parts of the design which need to be timed.

Rule 4: set clock latency constraint on an IP clock will be ignored

- **Constraint**: set_clock_latency 3 -source [get_clocks clk]
- **Resolution**: Ignored
- **Reason**: Clock latency constraint is not propagated.
- **User Action**: Re- define the constraint on the top-level clock.

Table 7.1 provides information on the CPE rules for ignored constraints.

Table 7.1. CPE Rules for Ignored Constraints

Constraint Input	Source Module	Resolution Status	Rule
create_clock -name {clk_ip_a} –period 10 [get_ports clk]	IP	Ignored	Clocks on the input side of an IP are clocks that could potentially drive other circuits. In addition, such clocks could give rise to incorrect slack calculations at input ports, output ports and inter-clock paths if defined on the IP.
create_clock -name {clk_ip_b} –period 10 [get_ports clk]	IP	Ignored	
set_input_delay –clock [get_clock sysclk] 9 [get_ports in_1]	IP	Ignored	IP Level input delay not propagated. Re-define at top level input port.
set_clock_groups [get_clocks clk] –group [get_clocks clk2]	IP	Ignored	Apps working with IP team to replace all Lattice IP constraints containing set_clock_groups constraint with set_false_path constraint.
set_clock_latency 3 –source [get_clocks clk]	IP	Ignored	Clock latency constraint is not propagated.

Table 7.2 provides information on the CPE rule for resolved constraints.

Table 7.2. CPE Rules for Resolved Constraints

Constraint Input	Source Module	Resolution Status	Constraint Output	Comments
create_clock –name {clk_top} –period 5 [get_ports clk_in]	Top	Resolved	create_clock –name {clk_top} –period 5 [get_ports gclk]	Constraint preserved
create_generated_clock –divide_by 2 –source [get_ports clkb] [get_pins b_out]	IP	Resolved	create_generated_clock –divide_by 2 –source [get_pins IP_B/clkb] [get_pins IP_B/b_out]	Propagated and name adjusted.
set_multicycle_path 2 –from [get_pins ff1/Q] –to [get_pins ff2/D]	IP	Resolved	set_multicycle_path 2 –from [get_pins instA/ff1/Q] –to [get_pins instA/ff2/D]	Propagated and name adjusted, since it does not involve clocks.
set_clock_uncertainty 2 [get_clocks internalclk]	IP	Resolved	set_clock_uncertainty 2 [get_clocks IP_C/internalclk]	Internal clock uncertainty still accepted.
set_max_delay –from [get_ports b_in] –to [get_ports b_out] 5	IP	Resolved	set_max_delay –from [get_pins IP_B/b_in] –to [get_pins IP_B/b_out] 5	Max and min delay always propagated.
set_false_path –from [get_ports b_in] –to [get_ports b_out]	IP	Resolved	set_false_path –from [get_pins IP_B/b_in] –to [get_pins IP_B/b_out]	False path propagated when clocks not used.

7.2. CPE Usage and Recommendation

- If the user has an IP instantiated in the design, run through the synthesis flow.
- Check IP constraints that are propagated by CPE using CPE generated Output files (next slide).
- Refer to the "Timing Constraints Resolution Summary" → User Action to "Keep the Constraint" section on Radiant help.

Table 7.3. Timing Constraint Resolution Summary

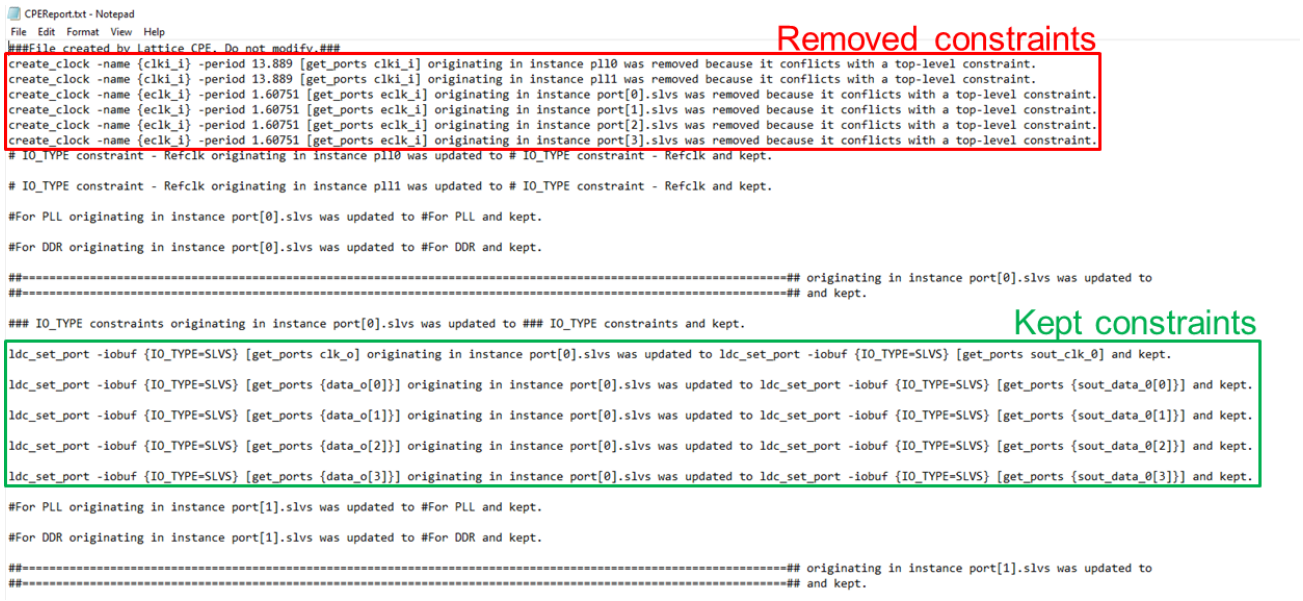
S.No.	Constraint Input	Source Module	Resolution Status	Constraint Output	Resolution Method	User Action to Keep the Constraint
1	create clock -name {clk_top} -period 5 [get_ports gclk]	Top	Resolved	Create_clock -name {clk_top} -period 5 [get_ports gclk]	Constraint preserved	No action needed.
2	create_clock -name {clk_ip_a} -period 10 [get_ports clk]	IP_A	Ignored	Constraint #1	Conflict with Constraint #1	Confirm satisfaction with top-level clock.
3	create_clock -name {clk_ip_b} -period 10 [get_ports clkb]	IP_B	Ignored	Defined on IP input Port	Ignore. Warn user	Redefine IP-level clock on appropriate top-level port.
4	create_generated_clock -divide_by 2 -source [get_ports clkb] [get_pinsb_out]	IP_B	Resolved	create_generated_clock -divide_by 2 -source [get_pins IP B/clkb] [get_pins IP B/b_out]	Propagated and name adjusted	No action needed.
5	set_input_delay -clock [get_clock sysclk] 9 [get_ports b_in]	IP_B	Ignored	Removed	IP-Level input delay not propagated	Redefine at top-level input port.
6	set_clock_groups [get_clocks_clk] -group [get_clocks clk2]	IP_B	Ignored	Removed	At least one clock is not internal	Translate to set_false_path and use appropriate user/custom IP-level objects.
7	set_max_delay -from [get_ports b_in] -to [get_ports b_out] 5	IP_B	Resolved	set_max_delay -from [get_pins IP_B/b_in] -to [get_pins IP_B/b_out] 5	Max and Min delay always propagated	No action needed.

7.3. CPE Output Files

The Constraints Propagation Engine generates some output files. This section provides information on the output files generated by CPE.

7.3.1. CPEreport.txt File

CPE generates a CPEReport.txt file which contains information on the removed and propagated constraints. This file can be found in the project implementation directory. Figure 7.1 gives an example of a CPEreport.txt file.




```
###File created by Lattice CPE. Do not modify.###
create_clock -name {clki_i} -period 13.889 [get_ports clki_i] originating in instance pll0 was removed because it conflicts with a top-level constraint.
create_clock -name {clki_i} -period 13.889 [get_ports clki_i] originating in instance pll1 was removed because it conflicts with a top-level constraint.
create_clock -name {eclk_i} -period 1.60751 [get_ports eclk_i] originating in instance port[0].slvs was removed because it conflicts with a top-level constraint.
create_clock -name {eclk_i} -period 1.60751 [get_ports eclk_i] originating in instance port[1].slvs was removed because it conflicts with a top-level constraint.
create_clock -name {eclk_i} -period 1.60751 [get_ports eclk_i] originating in instance port[2].slvs was removed because it conflicts with a top-level constraint.
create_clock -name {eclk_i} -period 1.60751 [get_ports eclk_i] originating in instance port[3].slvs was removed because it conflicts with a top-level constraint.
# IO_TYPE constraint - Refclk originating in instance pll0 was updated to # IO_TYPE constraint - Refclk and kept.

# IO_TYPE constraint - Refclk originating in instance pll1 was updated to # IO_TYPE constraint - Refclk and kept.

#For PLL originating in instance port[0].slvs was updated to #For PLL and kept.

#For DDR originating in instance port[0].slvs was updated to #For DDR and kept.

##=============================================================================================================## originating in instance port[0].slvs was updated to
##=============================================================================================================## and kept.

### IO_TYPE constraints originating in instance port[0].slvs was updated to ### IO_TYPE constraints and kept.

ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports clk_o] originating in instance port[0].slvs was updated to ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports sout_clk_0] and kept.

ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {data_o[0]}] originating in instance port[0].slvs was updated to ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {sout_data_0[0]}] and kept.

ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {data_o[1]}] originating in instance port[0].slvs was updated to ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {sout_data_0[1]}] and kept.

ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {data_o[2]}] originating in instance port[0].slvs was updated to ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {sout_data_0[2]}] and kept.

ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {data_o[3]}] originating in instance port[0].slvs was updated to ldc_set_port -iobuf {IO_TYPE=SLVS} [get_ports {sout_data_0[3]}] and kept.

#For PLL originating in instance port[1].slvs was updated to #For PLL and kept.

#For DDR originating in instance port[1].slvs was updated to #For DDR and kept.

##=============================================================================================================## originating in instance port[1].slvs was updated to
##=============================================================================================================## and kept.
```

Figure 7.1. Example of a CPEreport.txt File

7.3.2. CPE Generated .ldc File

CPE generates a .ldc file which is an effective constraints file after constraints propagation and resolution. This file is consumed by synthesis tools for synthesis. This file can be used to check propagated constraints. This file is also located in the project implementation directory. The file name for this .ldc file ends with *_impl_1_cpe.ldc. Figure 7.2 gives an example of a CPE generated .ldc file.


```
###File created by Lattice CPE. Do not modify.###
set_hierarchy_separator /
#create_clock -name {clk_byte_fr_i} -period 21.739 [get_pins {IMXcam_rx_I1/clk_byte_fr_i }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/IMXcam_rx/constraints/IMXcam_rx.ldc
create_clock -name {IMXcam_rx_I1_clk_byte_hs_o_c} -period 21.739 [get_nets {IMXcam_rx_I1/clk_byte_hs_o_c }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/IMXcam_rx/constraints/IMXcam_rx.ldc
create_clock -name {IMXcam_rx_I1_clk_byte_o_c} -period 21.739 [get_nets {IMXcam_rx_I1/clk_byte_o_c }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/IMXcam_rx/constraints/IMXcam
#create_clock -name {clk_byte_i} -period 21.739 [get_pins {b2p_I1/clk_byte_i }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
#create_clock -name {clk_pixel_i} -period 6.793 [get_pins {b2p_I1/clk_pixel_i }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
create_clock -name {int_osc_int_hf_clk_out_o} -period 33.333 [get_pins {int_osc_int/hf_clk_out_o }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/int_osc/constraints/int_osc.ldc
#######
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
# there are multicycle paths within the design other than RAW12, RAW14, RAW16.
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
# to constrain these paths, please copy the constraints below to your post-synthesis constraints file (*.pdc) and modify the hierachy/net_names of the ff. registers - pixcnt*,
wc_pix_sync* and pix_out_cntr*
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
# please check the netlist, or the PAR timing results, for the corrrect path and net names of these registers.
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
# **** NOT APPLICABLE for RAW12, RAW14, RAW16. do NOT copy for RAW12, RAW14, RAW16 !!!
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
set_multicycle_path -setup -from [get_nets {b2p_I1/lscc_byte2pixel_inst/lscc_byte2pixel_core/pixcnt* b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/wc_pix_sync_o*
b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/wc_pix_sync* }] -to [get_nets {b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/pix_out_cntr_o*
b2p_I1/lscc_byte2pixel_inst/lscc_byte2pixel_core/pix_out_cntr* }] 7
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
set_multicycle_path -hold -from [get_nets {b2p_I1/lscc_byte2pixel_inst/lscc_byte2pixel_core/pixcnt* b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/wc_pix_sync_o*
b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/wc_pix_sync* }] -to [get_nets {b2p_I1/lscc_byte2pixel_inst/genblk4.lscc_byte2pixel_core/pix_out_cntr_o*
b2p_I1/lscc_byte2pixel_inst/lscc_byte2pixel_core/pix_out_cntr* }] 6
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/b2p/constraints/b2p.ldc
```

Figure 7.2. Example of a CPE Generated .ldc File

8. Physical Constraints

Physical Constraints can be specified in either SDC/LDC file pre-synthesis or using the PDC file post-synthesis. If a physical constraint is specified in the SDC/LDC file pre-synthesis, the synthesis tools simply parse the constraint to post-synthesis stage. In some cases, synthesis tools may ignore physical constraints added to the SDC/LDC file and hence it is recommended to add the physical constraints post synthesis using the PDC file. Table 8.1 lists all the physical constraints supported in Radiant.

Table 8.1. Physical Constraints Supported in Radiant

Constraint	Definition
ldc_create_group	Used to define a group containing logic.
ldc_create_region	Used to define a rectangular area where a group can be located
ldc_set_location	Used to place a component at a specified location including a site or bank
ldc_create_macro	Used to define an instance in the design as a macro
ldc_create_vref	Used to define a voltage reference
ldc_set_vcc	Used to set the voltage and/or a derate for IO bank or core
ldc_set_port	Used to set attributes to a port
ldc_set_sysconfig	Used to set sys_config attributes
ldc_prohibit	Used to prohibit the use of a site or all sites in a region

8.1. ldc_create_group Constraint

This constraint is used to define a group. A group is a logic cluster containing logic instances, hard components like EBR and/or DSP blocks, PIO that are to be packed together. Groups can be created using the Physical Designer GUI or by entering the constraint manually in the PDC file. It is recommended to create groups post synthesis as synthesis name changes may affect the groups created pre-synthesis.

The syntax for the ldc_create_group constraint is:

ldc_create_group -name <group_name>
[-bbox {height width}]
<object>

Table 8.2. ldc_create_group Constraint Options

Option	Usage
-name	Use to define the name of the group
-bbox	Used to define the size of the bounding box for the group. bbox can be specified using the height and width parameters. The height and width are integers

Example:

To create a group containing a few instances:

ldc_create_group -name my_group [get_cells inst1 inst2 inst3]

Note: Refer to Radiant help for detailed instructions on how to create a group.

8.2. ldc_create_region Constraint

Regions are rectangular blocks. A region can be used to place a group of logic or to prohibit any area from placement. A region can be created using the Physical Designer GUI or by manually entering the constraint into the PDC file. The syntax for creating a region is:

ldc_create_region -name <region_name>
[-site <site_name>]
[-width <width>]
[-height <height]

Table 8.3. ldc_create_region Constraint Options

Option	Usage
-name	Used to specify the group name
-site	Used to specify the row/ column of slice D as reference point
-width	Used to specify the width of the region
-height	Used to specify the height of the region

Example: ldc_create_region -name my_region -site R20C32D -width 20 -height 12

Note: Refer to Radiant help for detailed instructions on how to create regions.

8.3. ldc_set_location Constraint

The ldc_set_location constraint is used to place a component at a specific location. Once this constraint is used, then the location is locked. It means that the component will not be unplaced, moved, swapped, or deleted. This constraint can be applied to top level ports in the design, to place a region or to place a component in a slice or a hard block. Location constraints can be set using the Device Constraints Editor (DCE) or by manually entering the constraint into the PDC file. The syntax for the ldc_set_location constraint is:

```
ldc_set_location  [-site <site_name>] |
                  [-bank <bank_number>] |
                  [-region <region_name]
                  <object>
```

Table 8.4. ldc_set_location Constraint Options

Option	Usage
-site	Used to specify the PIO site of the target device or to specify the row/column of a slice/ hard block
-bank	Used to specify the port bank number
-region	Used to specify the region name

When DCE is used to specify the location constraints, when saved, the constraints get saved to the PDC file.

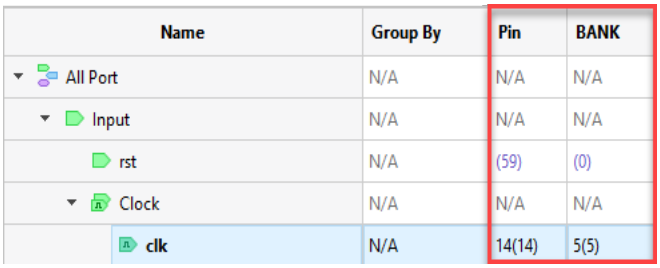


Figure 8.1. DCE location constraints

Example:

- To specify a PIO location to a top-level port:
  ldc_set_location -site 14 [get_ports clk]
- To place a PLL in the upper left corner for Crosslink-NX:
  ldc_set_location -site PLL_ULC [get_cells pll_inst]
- To place regions:
  ldc_set_location -region my_region [ldc_get_groups my_group]
  where ldc_get_groups refers to a previously created group named “my_group”

8.4. ldc_create_macro Constraint

The ldc_create_macro constraint is used to create macros from the user design. A macro can be a soft macro containing logic information, a firm macro containing logic and placement information or a hard macro containing placement and routing information. For instructions on creating macros, please refer to Radiant help.

The syntax for creating a macro is:

ldc_create_macro -name <macro _name>
-use_pio <port_list>
[get_cells <instance_name_with_hierarchy>]

Table 8.5. ldc_create_macro Constraint Options

Option	Usage
-name	Used to define macro name
-use_pio	Used to define the macro ports that are directly connected to the top-level ports. Here port list is a comma separated value.

Example: ldc_create_macro -name my_macro1 -use_pio {in1, in2} [get_cells top/inst1]

Macros can also be created using the pre-synthesis constraints editor. The constraint gets saved into the pre-synthesis SDC/LDC file.

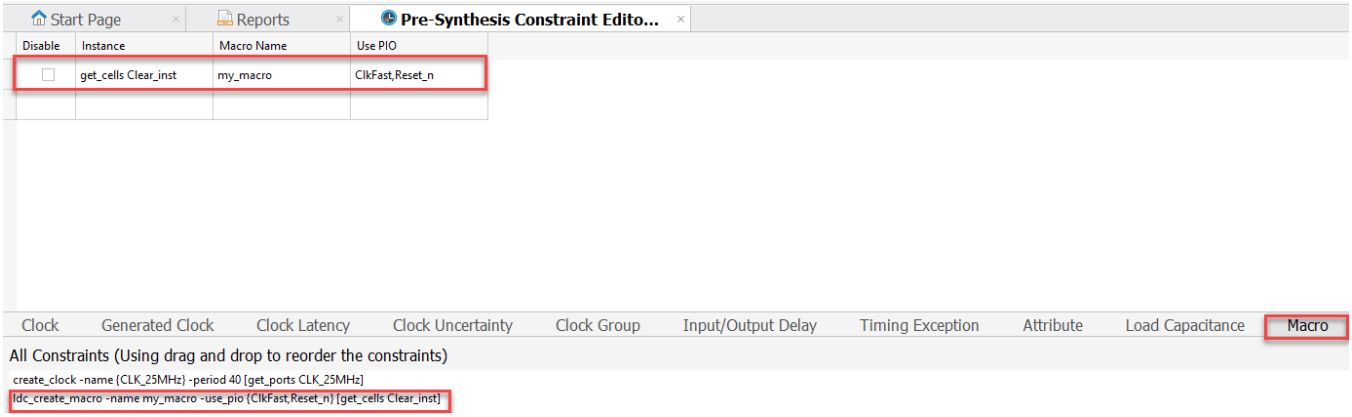


Figure 8.2. Pre-Synthesis Constraint Editor Tools

8.5. ldc_create_vref Constraint

ldc_create_vref constraint is used to define voltage references. Lattice FPGA devices commonly provide two input pins per bank that can serve as on-chip voltage references (Vref) for the I/O types that require them. Device Constraints Editor can be used to name and assign the specific voltage reference pins for these I/O types. If no Vref's are for I/O types are provided, Radiant will automatically create them. However, automatically generated Vrefs will not be added to the PDC file for back annotation. ldc_create_vref can also be manually entered in the PDC file.

The syntax for the ldc_create_vref constraint is:

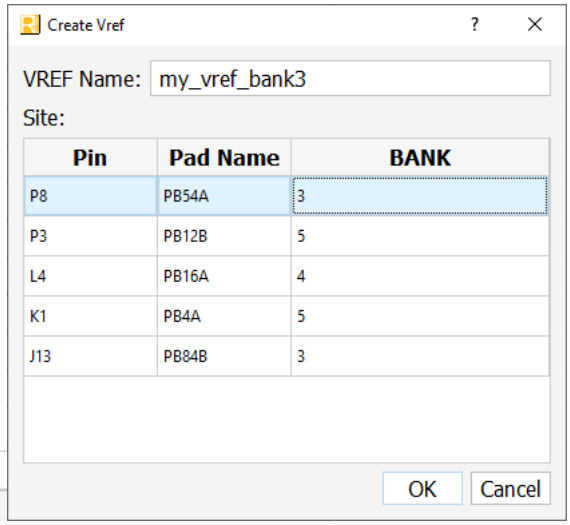
ldc_create_vref -name <vref_name>
-site <site_name>

Table 8.6. ldc_create_vref Constraint Options

Option	Usage
-name	Used to define the Vref name
-site	Used to define the PIO site name

Example: ldc_create_vref -name my_vref_bank3 -site P8

Vref can also be created using the DCE. Go to DCE -> Global -> Vref Locate. Once saved, this constraint will be added to the PDC file.



8.6. ldc_set_vcc Constraint

ldc_set_vcc constraint is used to set the voltage and/or derate for the bank or core. Care must be taken not to create conflicting preferences that locate I/O signals to a bank that are incompatible with the voltage supply designated by the ldc_set_vcc constraint. VCC voltage is automatically set when an I/O signal is assigned to a package pin using the ldc_set_location constraint. The syntax for the ldc_set_vcc constraint is:

ldc_set_vcc -bank <bank_number>|
-core
[-derate <derate_percentage_number>]
[voltage]

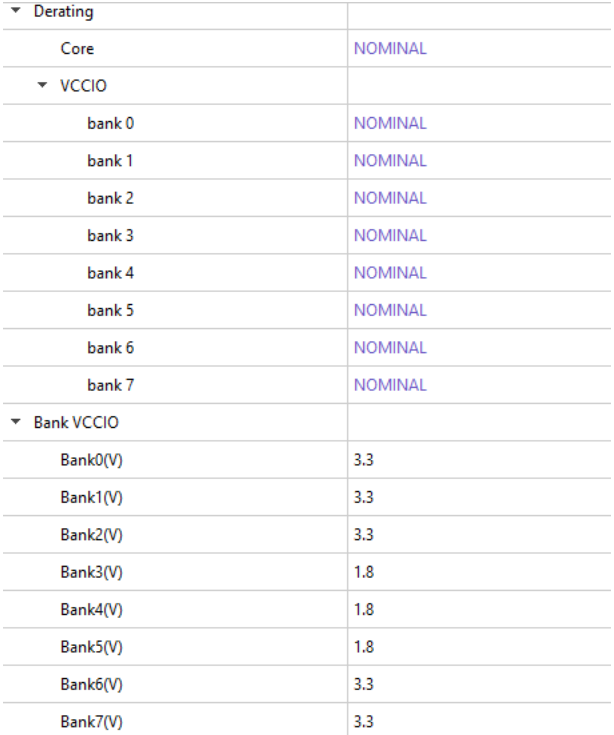
Table 8.7. ldc_set_vcc Constraint Options

Option	Usage
-bank	Used to specify the bank number
-core	Used to specify the voltage to a core
-derate	Used to specify voltage derate percentage
Voltage	Used to specify the compatible voltage value

Example:

- ldc_set_vcc -bank 3 3.3
- ldc_set_vcc -bank 1 -derate -1

VCC can also be set using the DCE. Go to DCE -> Global. The saved constraints will be added to the PDC file.



8.7. ldc_set_port Constraint

ldc_set_port constraint is used to specify attributes to IO ports. IO attributes include IO_TYPE, PULLMODE, OPENDRAIN, DIFFDRIVE, CLAMP, SLEWRATE, TERMINATION, DIFFRESISTOR, GLITCHFILTER, HYSTERESIS, VREF, and VIRTUAL. The syntax for the ldc_set_port constraint is:

ldc_set_port [-iobuf [-vref <vref_name>]] |
[-sso] <key-value list>
<ports>

Table 8.8. ldc_set_port Constraint Options

Option	Usage
-iobuf	Used to specify IOBUF attributes
-vref	Used to specify the voltage reference name. Must be used with -iobuff
-sso	Used to specify SSO key

Example: ldc_set_port -iobuf {PULLMODE=UP} [get_ports {switch_1}]

8.8. ldc_set_sysconfig Constraint

Defines system configuration option settings for the sysCONFIG feature. If you do not specify these settings in the .pdc file, using the Global tab in Device Constraint Editor or manually, some default sysCONFIG constraints will automatically be generated based on the device selection. The SYSCONFIG constraint is available for all Lattice FPGA devices that support the sysCONFIG configuration port. The sysCONFIG port can be a single data line or byte-wide (multiple byte) data port that can support serial and parallel configurations streams. The devices also support daisy chaining.

The syntax for ldc_set_sysconfig is:

ldc_set_sysconfig <key-value_list>
<key-value_list> include SYSCONFIG attributes.

Refer to Radiant help sysCONFIG settings section for details on the valid sysCONFIG options.

Example: ldc_set_sysconfig {JTAG_PORT=ENABLE PROGRAMN_PORT=ENABLE MCCLK_FREQ=56.2 DONE_OD=ON}

8.9. ldc_prohibit Constraint

Used to prohibit the use of a site or all sites in a region.

The syntax for the ldc_prohibit is:

ldc_prohibit -site <site> |
-region <region>

Example: ldc_prohibit -site A12

References

For complete information on Lattice Radiant Project-Based Environment, Design Flow, Implementation Flow and Tasks, as well as on the Simulation Flow, refer to the Lattice Radiant software user guide.

- Certus-NX Family Data Sheet (FPGA-DS-02078)

You may also visit the following web pages at the Lattice website:

- Lattice Radiant Software
- Lattice Synthesis Engine
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.5, July 2024

Section	Change Summary
SDC Constraints Supported in Radiant	Updated the description of Figure 3.5. Calculating Positive Setup Slack.

Revision 1.4, June 2024

Section	Change Summary
All	• Made editorial fixes. • Made changes on *Clock Constraint* values. • Added missing figures and captions.
Disclaimer	Updated this section.
Inclusive Language	Added boilerplate.

SDC Constraints Supported in Radiant	• In the create_clock Constraint section:
	  • Added a new option, -add, in the create_clock syntax and added the new option in Table 3.2. create_clock Constraint Options
	• In the create_generated_clock Constraint section:
	  • Added two new options, -add and -master_clock in the create_clock syntax.
	  • Updated Table 3.3. create_generated_clock Constraint Options.
	• In the set_clock_uncertainty Constraint section:
	  • Updated the description of the example for Table 3.5. set_clock_uncertainty Constraint Options.
	  • Updated the following figures:
	  • Figure 3.6. AC Characteristics – Figure 3.8. Uncertainty is added to the Destination Clock Path.
	• In the set_false_path Constraint section:
	  • Added the two new options, -setup and -hold, in the set_false_path syntax.
	  • Updated Table 3.11. set_false_path Constraint Options.

Revision 1.3, October 2023

Section	Change Summary
All	Corrected typos and grammatical fixes.
Acronyms in This Document	Added *SI* and its acronym *Signal Integrity*.
SDC Constraints Supported in Radiant	• Updated the direction of the outputs outwards in Figure 3.1. DUT. • Replaced the text *CLK* with *Clk* in Input/Output Delay Constraints section.
Physical Constraints	Added this section.

Revision 1.2, July 2023

Section	Change Summary
Clock Constraints	• Changed output clock cycle-to-cycle jitter to 250ps for clocks >=200MHz. • Changed set clock uncertainty setup to 0.25.
References	Added this section.

Revision 1.1, April 2023

Section	Change Summary
All	Updated document type from User Guide to Application Note.

Revision 1.0, January 2023

Section	Change Summary
All	Initial release.

www.latticesemi.com