Lattice Radiant Software Constraints Propagation Engine

Application Note

FPGA-AN-02097-1.0

February 2025

Disclaimers

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

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
CPE	Constraints Propagation Engine
FDC	FPGA Design Constraints
FPGA	Field Programmable Gate Array
IP	Intellectual Property
LDC	Lattice Design Constraints
LSE	Lattice Synthesis Engine
PAR	Place And Route
PDC	Physical Design Constraints
RTL	Register Transfer Level
SI	Signal Integrity
SDC	Synopsys Design Constraints
TCE	Timing Constraints Editor
UDB	Unified Database

1. Introduction

Lattice Radiant™ software is a comprehensive design environment for Lattice Semiconductor FPGAs. It offers a suite of tools for all design stages, from project management, design entry, simulation, synthesis, place and route, in-system logic analysis, and more.

To ensure your design meets its performance goals on the FPGA, you must provide accurate timing constraints to the design. The implementation tool in the Radiant software reads the provided constraints and optimizes your design accordingly. You often define constraints at the top level of your design and include additional constraint files for custom intellectual property (IP) cores or those generated by the Radiant software such as the IP Catalog. However, constraints defined by the module or at IP level may not always have the correct hierarchical names, leading to potential issues during synthesis.

To address this challenge, the Radiant software introduces the Constraints Propagation Engine (CPE). This feature propagates sub-hierarchical constraints and resolves conflicts between your constraints and IP constraints. The CPE compiles input constraints from multiple *.sdc* or *.ldc* files, from both of your IP and user constraints, and creates a unified *.ldc* file for the synthesis tools. It operates seamlessly before synthesis, requiring no manual intervention.

The CPE activates only when your design includes an *.ipx* file with *.sdc* or *.ldc* files, it does not support *.fdc* files. By performing a Design Rule Check (DRC) on all input constraints and generating a new constraint file that supports hierarchical constraints, the CPE ensures that soft IP constraints are prioritized in the top module. This enables cross-IP optimization during logic synthesis, maximizing the impact of your supplied constraints.

This document provides an in-depth look at how the Constraints Propagation Engine works and how it can help you to optimize your FPGA designs.

1.1. Audience

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

2. Constraints Propagation Engine

In the Radiant software, CPE is executed during pre-synthesis and pre-MAP stages. During pre-synthesis stage, CPE is executed right before synthesis to make sure all the required IP constraints are propagated, and any conflicts are resolved on the pre-synthesis netlist objects.

2.1. CPE Rules

The CPE is executed only when an IP with an *.sdc* or *.ldc* constraints file is instantiated in the user design. CPE does not resolve conflicts within user constraints. User constraints conflicts are handled by the Radiant software timer. Constraints resolution rules applies only if there is a conflict between user constraints and IP constraints. The rules are as follow:

- Create_clock constraint on an IP port is 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 must 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 give rise to incorrect slack calculations at input ports, output ports, and inter-clock paths if defined on the IP.
  - User Action: Redefine the clock constraint on the top module ports.
  - Example: create_clock -name clk -period 10 [get_clocks clk_in]
- Input/output delay constraint on an IP port is 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: Redefine 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 is propagated.
- set_clock_groups constraints are ignored.
  - Constraint: set_clock_groups -group [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 that needs to be timed.
- set_clock_latency constraint on an IP clock is ignored.
  - Constraint: set_clock_latency 3 -source [get_clocks clk]
  - Resolution: Ignored.
  - Reason: Clock latency constraint is not propagated.
  - User Action: Redefine the constraint on the top-level clock.

The following table provides information on the CPE rules for ignored constraints.

Table 2.1. 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. Redefine at top-level input port.
set_clock_groups -group [get_clocks clk] -group [get_clocks clk2]	IP	Ignored	Constraint is ignored because at least one clock is not internal. This is hazardous if these clocks are not used in other parts of the designs that need to be timed.
set_clock_latency 3 -source [get_clocks clk]	IP	Ignored	Clock latency constraint is not propagated.

The following table provides information on the CPE rule for resolved constraints.

Table 2.2. CPE Rules for Resolved Constraints

Constraint Input	Source Module	Resolution Status	Constraint Output	Description
create_clock -name {clk_top} -period 5 [get_ports clk_in]	Top	Resolved	create_clock -name {clk_top} -period 5 [get_ports gclk]	Constraint is 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 because 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	Maximum and minimum delay is 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 are not used.

2.2. CPE Usage and Recommendation

The following lists the CPE usage and recommendations:

- If you have an IP instantiated in your design, run through the synthesis flow.
- Check the IP constraints that are propagated by the CPE using the CPE generated output files.
- Refer to the *Timing Constraints Resolution Summary* section, specifically the *User Action to Keep the Constraint* guidance within the Radiant software help documentation.

Table 2.3. Timing Constraint Resolution Summary

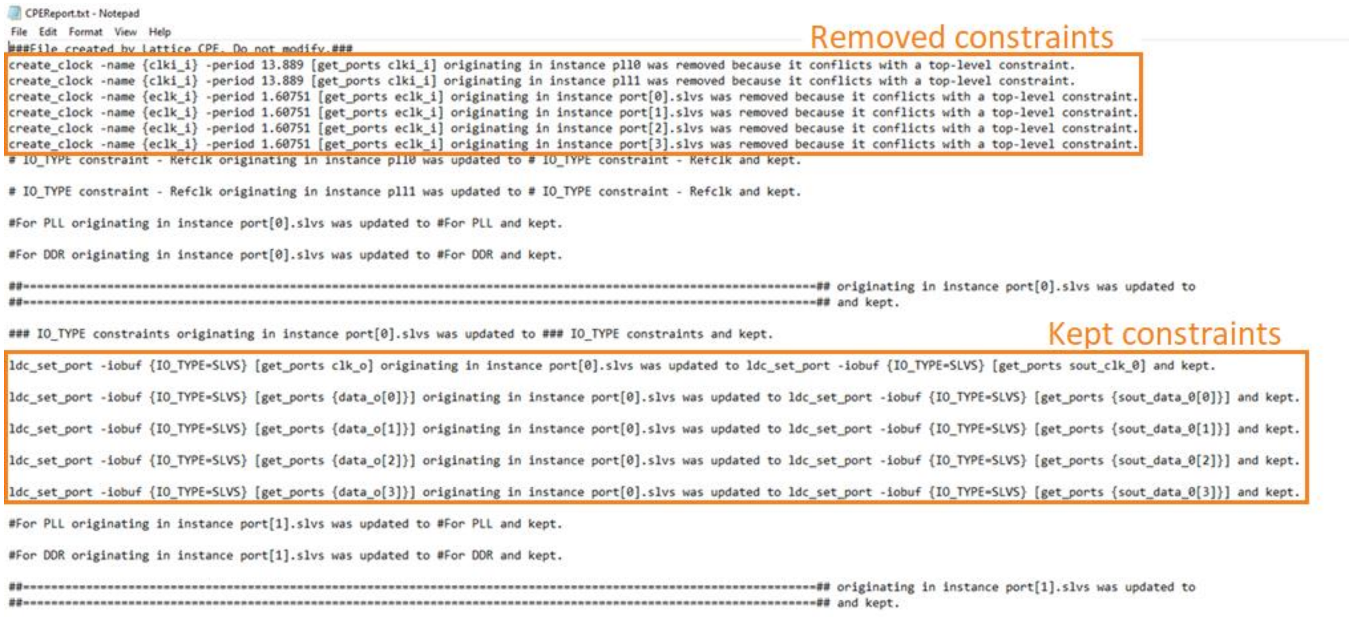
Constraint Input	Source Module	Resolution Status	Constraint Output	Resolution Method	User Action to Keep the Constraint
create clock -name {clk_top} -period 5 [get_ports gclk]	Top	Resolved	Create_clock -name {clk_top} -period 5 [get_ports gclk]	Constraint preserved.	No action needed.
create_clock -name {clk_ip_a} -period 10 [get_ports clk]	IP_A	Ignored	Constraint #1	Conflict with Constraint #1.	Confirm satisfaction with top-level clock.
create_clock -name {clk_ip_b} -period 10 [get_ports clkb]	IP_B	Ignored	Defined on IP input port	Ignore.	Redefine IP-level clock on appropriate top-level port.
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.
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.
set_clock_groups -group [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.
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	Maximum and minimum delay always propagated.	No action needed.

2.3. CPE Output Files

The CPE generates some output files. This section provides information on the output files that are generated by the CPE.

2.3.1. CPEreport.txt file

The CPE generates a *CPEReport.txt* file that contains information on the removed and propagated constraints. This file can be found in the project implementation directory. The following figure shows an example of a *CPEreport.txt* file.

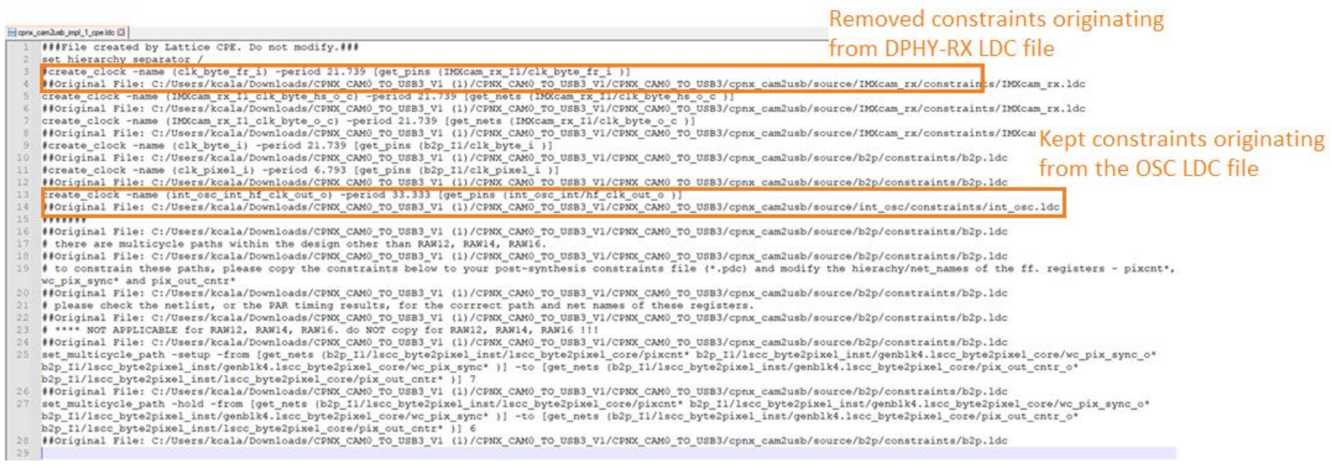


```
###File created by Lattice CPE. Do not modify.###
create_clock -name {clki_i} -period 13.889 [get_ports clki_i] originating in instance pll0 was removed because it conflicts with a top-level constraint.
create_clock -name {clki_i} -period 13.889 [get_ports clki_i] originating in instance pll1 was 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 2.1. CPEReport.txt File

2.3.2. CPE Generated .ldc File

The CPE generates an *.ldc* file that is an effective constraints file after constraints propagation and resolution. This file is consumed by the 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*. The following figure shows an example of a CPE-generated *.ldc* file.


cpnx_cam2usb_impl_1_cpe.ldc
Removed constraints originating from DPHY-RX LDC file
Kept constraints originating from the OSC LDC file

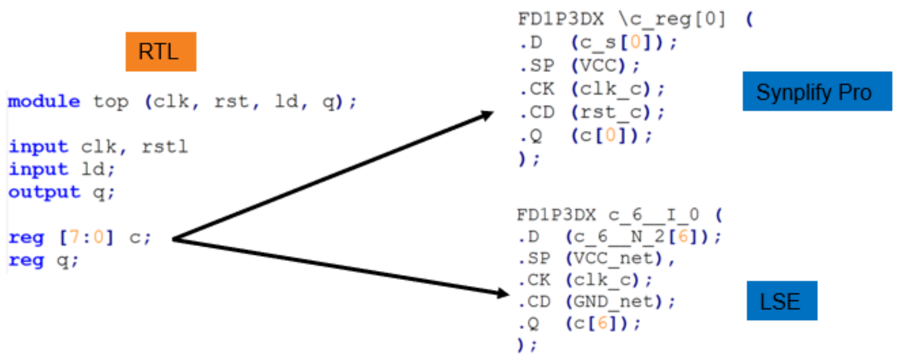
```
###File created by Lattice CPE. Do not modify.###
set hierarchy separator /
#create_clock -name {clk_byte_fr_i} -period 21.739 [get_pins {IMXcam_rx_I1/clk_byte_fr_i }]
##Original File: C:/Users/kcala/Downloads/CPNX_CAM0_TO_USB3_V1 (1)/CPNX_CAM0_TO_USB3_V1/CPNX_CAM0_TO_USB3/cpnx_cam2usb/source/IMXcam_rx/constraints/IMXcam_rx.ldc
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/IMXca
#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 2.2. CPE-generated.ldc File

3. Post-Synthesis Constraint Propagation

Prior to the Radiant software version 2024.1, IP pre-synthesis timing constraints are not automatically propagated, and you must manually copy and paste these constraints into the post-synthesis constraint (*.pdc*) files for them to take effect. This is because object names are changed in the synthesis stage as they undergo transformation from the logical to physical representation. During this stage, the tool also performs optimization to the netlist.

Depending on the synthesis tool used, the resulting object names can be different as shown in Figure 3.1. Object Names in Different Synthesis Tools. In this example, the register *c* in the original RTL file is synthesized to *c_reg* and *c_6__I_0* in the Synplify Pro and LSE tools, respectively.



```
module top (clk, rst, ld, q);

input clk, rstl
input ld;
output q;

reg [7:0] c;
reg q;
```

```
FD1P3DX \c_reg[0] (
.D   (c_s[0]);
.SP  (VCC);
.CK  (clk_c);
.CD  (rst_c);
.Q   (c[0]);
);
```

```
FD1P3DX c_6__I_0 (
.D   (c_6__N_2[6]);
.SP  (VCC_net),
.CK  (clk_c);
.CD  (GND_net);
.Q   (c[6]);
);
```

Figure 3.1. Object Names in Different Synthesis Tools

To address the limitations, the Radiant software versions 2024.1 and later support post-synthesis IP timing constraint propagation. This is achieved through several framework enhancements including:

- Support for IP timing constraints in various compilation design stages and different synthesis tools.
- Support for IP timing constraints in a single file, enabled by the TCL commands and pre-defined Radiant software variables.
- Simplification of constraint file suffixes.

3.1. IP Directory and File Structure

When an IP is generated from the IP Catalog in the Radiant software, a collection of files related to the configured IP are created and organized in the directory as shown below on the machine's disk, where the timing constraint file are organized under the *constraints* folders as a *constraint.sdc* file. Some IPs might still contain *<component_name>.ldc* for backward compatibility.

```
<component_name>
<component_name>.cfg
<component_name>.ipx
component.xml
design.xml
rtl
      <component_name>.v
      <component_name>_bb.v
constraints
      constraint.sdc
      <component_name>.ldc
testbench
      <design testbench files>
<other IP related files>
```

To ensure the timing constraints are effective in a single file, the Radiant software versions 2024.1 and later incorporated an internal TCL interpreter to correctly identify constraints meant for different implementation stages and synthesis tools. The following two variables are added to this interpreter:

- $radiant(stage), where the valid values are *presyn* and *premap*.
- $radiant(synthesis), where the valid values are *lse* and *synpro*.

The following code snippet shows an example of how multiple timing constraints can be written in a single file with the introduction of TCL variables for different stages and synthesis tools.

```
set var 5
if {$radiant(stage) == "presyn"} {
create_clock -period 10 -name myclk [get_ports clk]
}
if {$radiant(synthesis) == "lse"} {
       # LSE
       if {$radiant(stage) == "presyn"} {
              set_max_delay -from [get_cells {c[0]}] $var
} elseif {$radiant(stage) == "premap"} {
              set_max_delay -from [get_cells {c_6__I_0.ff_inst}] [expr $var+20]
       }
} else {
       # synplify
       if {$radiant(stage) == "presyn"} {
              set_max_delay -from [get_cells {c[0]}] 10
} elseif {$radiant(stage) == "premap"} {
              set_max_delay -from [get_cells {c_reg[0].ff_inst}] 25.0
       }
}
set_false_path -from [get_ports {q}]
```

Note that constraints from the previous stage are propagated to the next stage if they are not dropped. Hence, it is unnecessary to duplicate them in multiple stages unless there is a need to overwrite or include additional constraints.

Any constraints that are outside of the if-else statement are evaluated multiple times. In the example above, the *set_false_path* constraint is duplicated for different engines.

Note: It is for the IP developer only – user constraint does not support single constraint file style.

3.2. IP Constraint File Comparison

The following figure highlights the key differences between the LPDDR4 constraint files generated in the Radiant software v2023.1 and the Radiant software v2024.1. The introduction of new framework has led to the creation of a new *constraint.sdc* file within the *Constraint Files* folder.

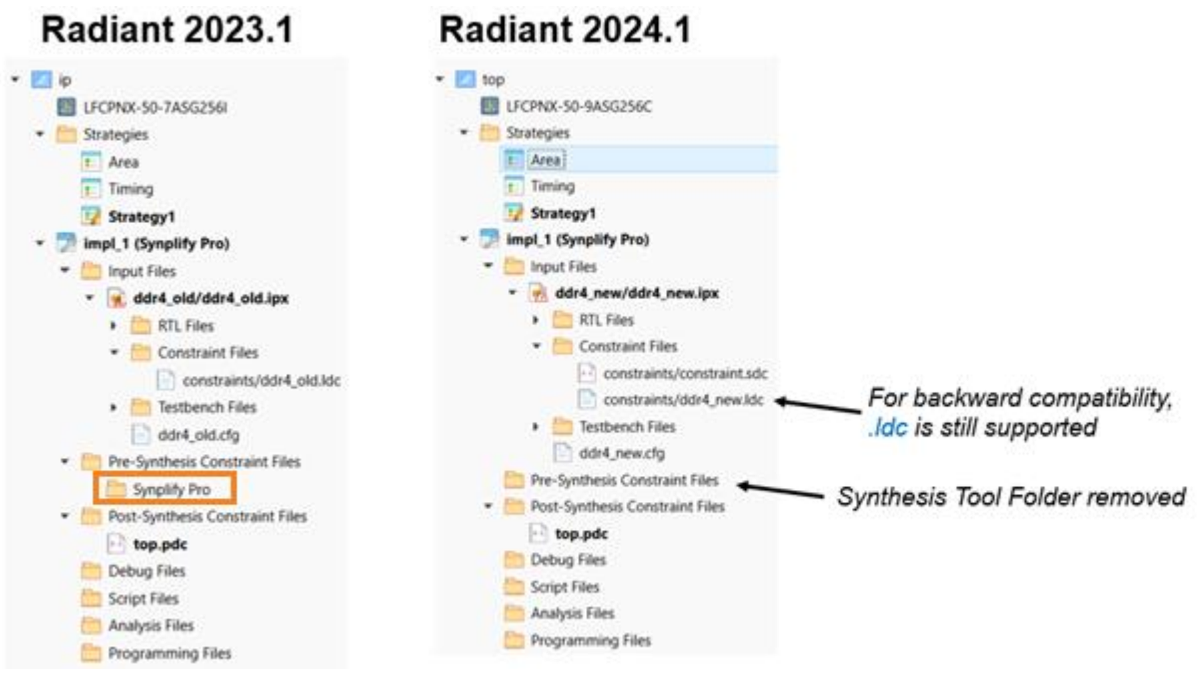


Figure 3.2. IP Constraint File Structure Comparison

4. Summary

The Radiant software provides different mechanisms to handle timing constraints, both in pre and post-synthesis stages. The following figure summarizes the entire constraint propagation flow, including the input and output of each stage.

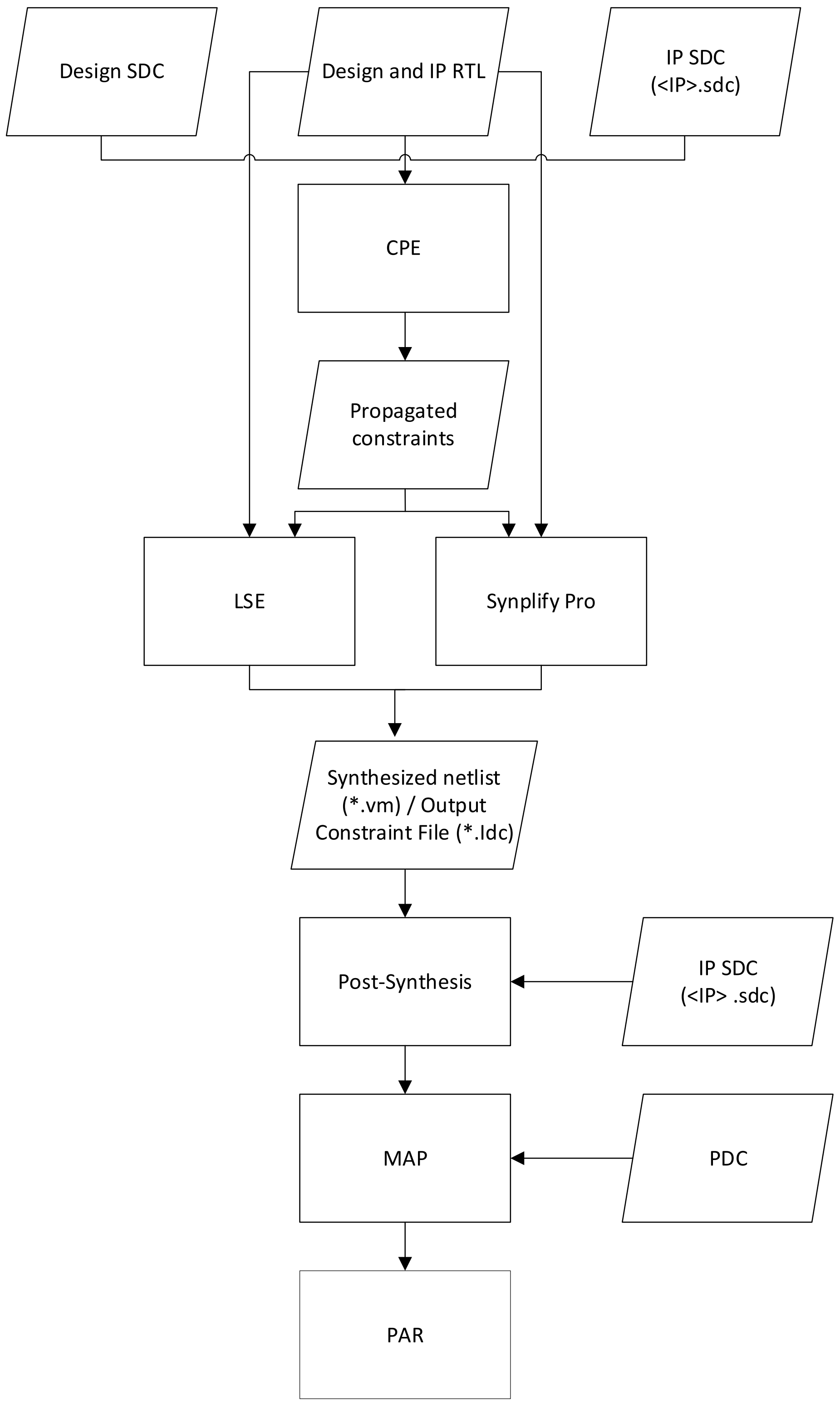


Figure 4.1. Constraint Propagation Flow in the Lattice Radiant Software

Reference

- Lattice Radiant Software web page
- Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059)
- Lattice Insights web page 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 1.0, February 2025

Section	**Change Summary**
All	Initial release.

www.latticesemi.com