Timing Closure

Introduction

Every design has to run at a certain speed based on the design requirement. There are generally three types of speed requirement in an FPGA design:

- Timing requirement – how fast or slow a design should run. This is defined through the target clock period (or clock frequency) and a few other constraints.
- Throughput – the average rate of the valid output delivered per clock cycle
- Latency – the amount of the time required when the valid output is available after the input arrives, usually measured in the number of clock cycles

Throughput and latency are usually related to the design architecture and application, and they need to be traded off between each other based on the system requirement. For example, high throughput usually means more pipelining, which increases the latency; low latency usually requires longer combinatorial paths, which removes pipelines, and this can reduce the throughput and clock speed.

More often, FPGA designers deal with the timing requirement to make sure that the design runs at the required clock speed. This can require hard work for high-speed design in order to close timing using various techniques (including the trade-off between throughput and latency, appropriate timing constraint adjustments, etc.) and running through multiple processing iterations including Synthesis, MAP and PAR.

This document focuses on the timing requirement; it explains the timing-driven FPGA implementation processes and shows how to tackle timing issues when timing closure becomes problematic.

Timing Requirements and Constraints

Several types of timing requirement are commonly used in FPGA designs and can be specified in Diamond through constraints and preferences. These are applied to the FPGA implementation processes, including Synthesis, MAP and PAR, as explained in the following sections.

Clock Period/Frequency

Usually the maximum delay (or the most critical path) between any two sequential elements (e.g. registers) in a clock domain determines that clock's maximum frequency. In order to ensure that a design can run at the required speed, the clock period or frequency should be defined as a constraint for the timing-driven process so that the implementation process considers the requirement and ensures that the maximum delay is no larger than the clock period defined.

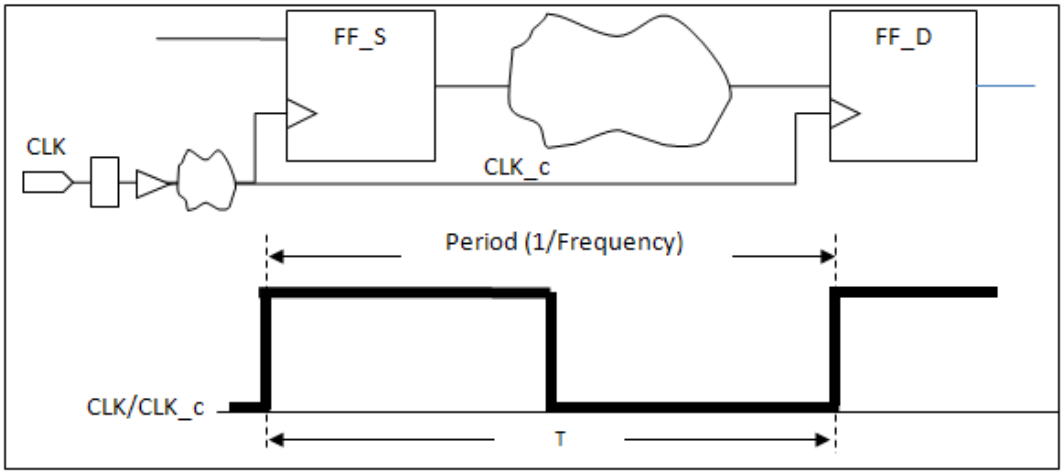
If a design includes multiple clock domains, each clock should be appropriately constrained.

Figure 1 illustrates the following timing preference:

FREQUENCY PORT "CLK" <Frequency> MHZ

It illustrates an ideal clock, its period and frequency definitions. In this diagram, the circuit will operate correctly if the data leaving FF_S (which is created by the first clock edge) arrives at FF_D prior to the second clock edge. The Period (or Frequency) defines how far apart these two clock edges are in time.

Figure 1: Period and Frequency of an Ideal Clock



Relating Two Clocks for Period/Frequency

If a data path crosses between two clock domains, the edge relationship between those two clocks must be known to analyze the timing. For example, if both clocks run with the same period P but there is skew between them, the data path timing must meet a different constraint than T.

If the relationship is not known, then the data path will not be constrained.

Synthesis tools such as Synplify Pro must be told of these relationships. If Synplify Pro constraint is used, this is done by defining two clocks with the same clock group name using define_clock constraints, at the same time, by defining clock skew using define_clock_delay constraint.

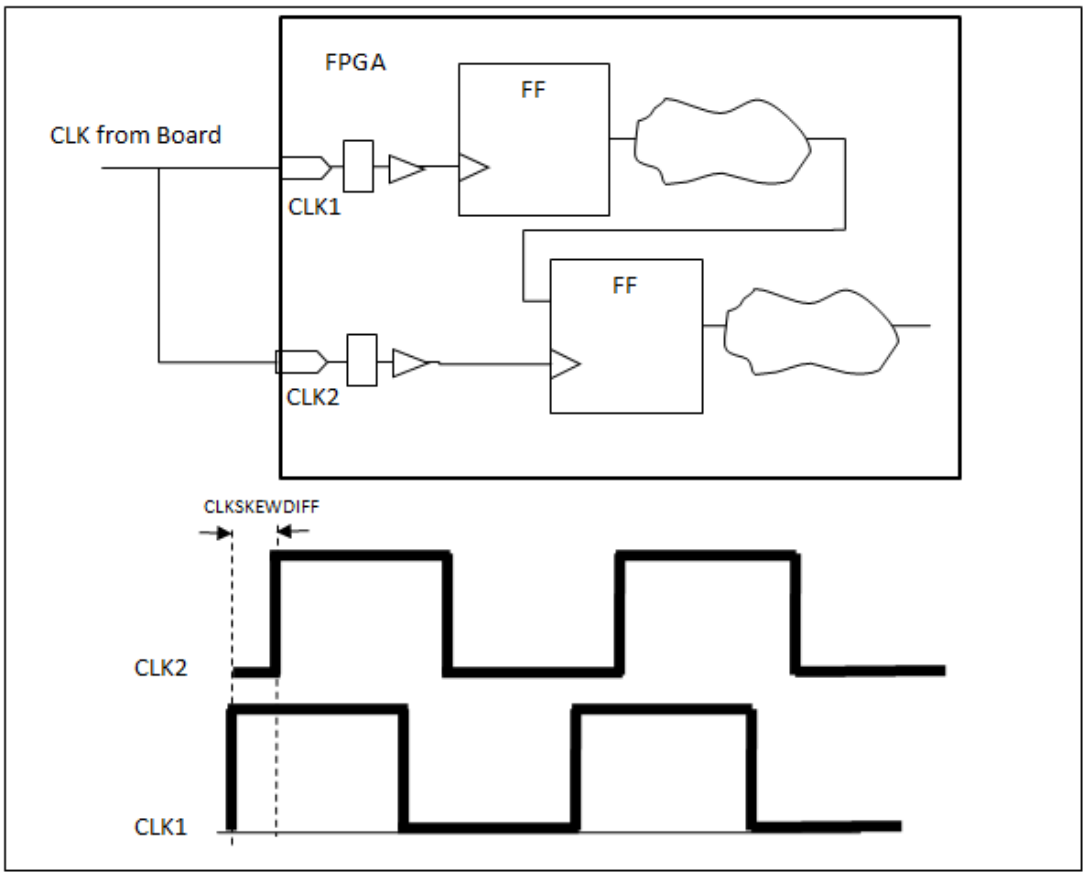
The FPGA implementation tools such as MAP, PAR and TRACE usually can determine the timing relationship between two clocks (e.g. they both come from the same PLL). However, if both clocks come from external pins, the user must specify their relationship. This is done using the CLKSKEWDIFF preference.

Figure 2 illustrates the following timing parameter:

CLKSKEWDIFF CLKPORT "CLK2" CLKPORT "CLK1" <clkskewdiff_value> NS;

CLKSKEWDIFF is used to relate two otherwise unrelated clocks, for example, two top-level clocks. TRACE will not analyze cross-domain paths between unrelated clocks. You can establish a relationship between two unrelated clocks by specifying the amount of clock skew between these clocks using the CLKSKEWDIFF preference, as illustrated in Figure 2.

Figure 2: Using CLKSKEWDIFF



Input/Output Timing

FPGA IO timing basically looks at one part of the register-register timing analysis of the simple Period/Frequency case in Figure 2. The goal is to be able to analyze register-to-register paths that cross between two devices, but focus the analysis on the FPGA device and model the other device within the FPGA board timing environment.

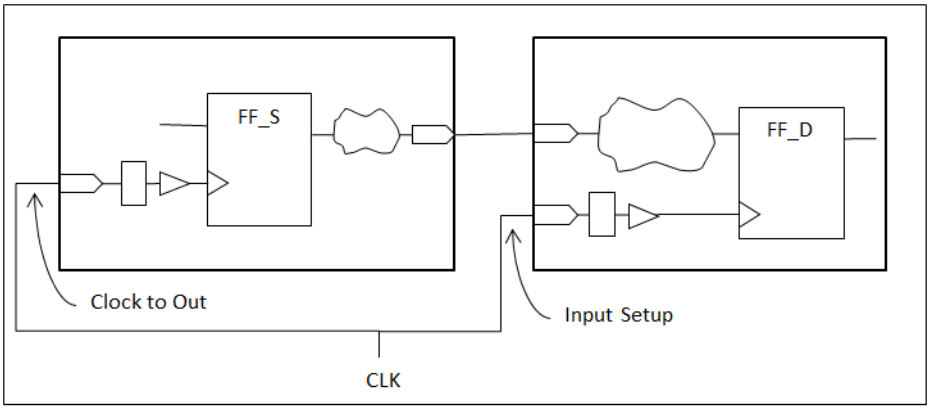
- "Input" case – when the FPGA is receiving data from a "source" device (as input)

  In this case, the timing data of when the other device (and board) guarantees to provide data to the FPGA pins is provided to the analysis as a constraint.

- "Output" case – when the FPGA is sending data to a "destination device" (as output)

  In this case, the timing data of when the other device (and board) needs the data emerging from the FPGA pins is provided to the analysis as a constraint.

Figure 3: Input/Output Timing



Input Setup/Input Delay

Input setup is the time difference between when the data arrives at its FPGA input pin, and when the next clock edge arrives as its FPGA pin. Input setup is a positive value if the data arrives before this clock edge. The input setup value is a function of the clock speed, source device timing (clock to out value) and board timing. This is the time available (i.e. constraint) to the timing of the data and clock paths within the FPGA to meet the internal device timing requirements.

The detailed timing example below shows the components of the two options to define input IO timing constraint: Input setup and Input delay.

The external environment is given: a clock period of 20 ns, board clock skew of 1ns, board trace of 8 ns, and the source device's clock to out spec of 7 ns. This causes the data to arrive at its FPGA data pin at least 4 ns before its

capture clock edge arrives at its FPGA pin – this is the input setup constraint the FPGA internal timing must work within to meet the internal register setup time of 0.50 ns.

Note

The input setup constraint value depends on the clock period value. Thus, if the clock period constraint changes, the input setup constraint should also change.

Input delay is the time between when the previous capture clock edge arrived as its FPGA pin and when the data arrives at its FPGA input pin. Input delay is a positive value if the data arrives after this clock edge.

Note

The input delay value does not depend on the clock period value. Thus, if the clock period constraint changes, the input delay constraint does not change.

Input setup and input delay are two different ways of looking at the same thing (if you know one, you know the other).

Input setup + Input delay = clock period.

Both input setup and input delay forms are specified using the INPUT_SETUP preference.

The HOLD time (in the INPUT_SETUP preference) represents how long the data is valid at the FPGA input pin after the clock edge used for input setup arrives at its FPGA pin. It is used to test for board level hold time violations.

The Hold time is how long the data will remain constant at the FPGA input pin after the clock edge arrives at its FPGA pin.

Figure 4: Input Case

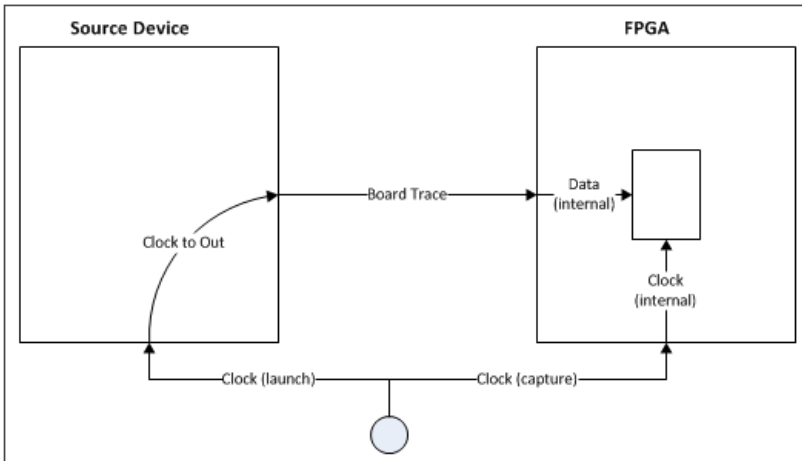
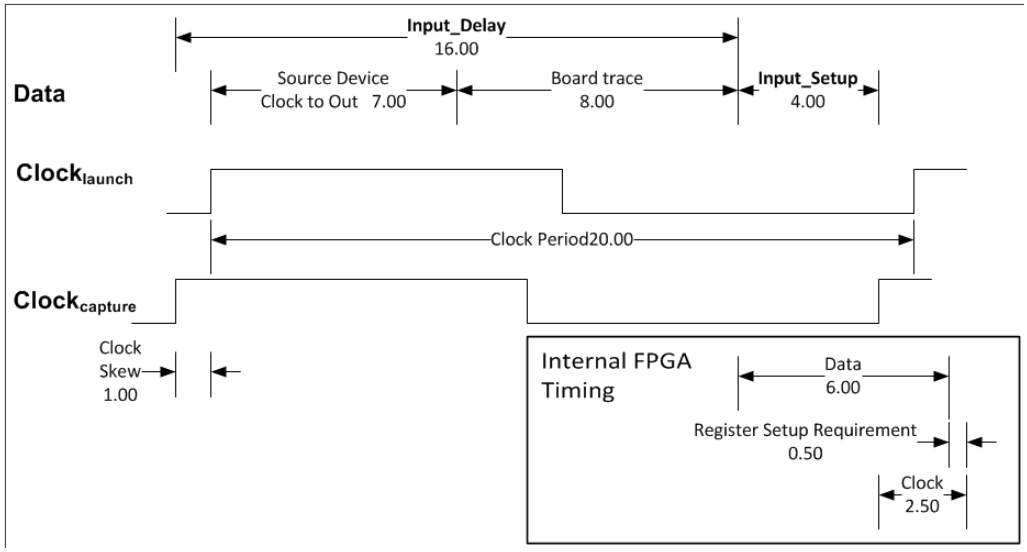


Figure 5: Input Timing



The following shows how Input setup and Input delay forms are specified in the preference language. See the diagram in Figure 6 for reference:

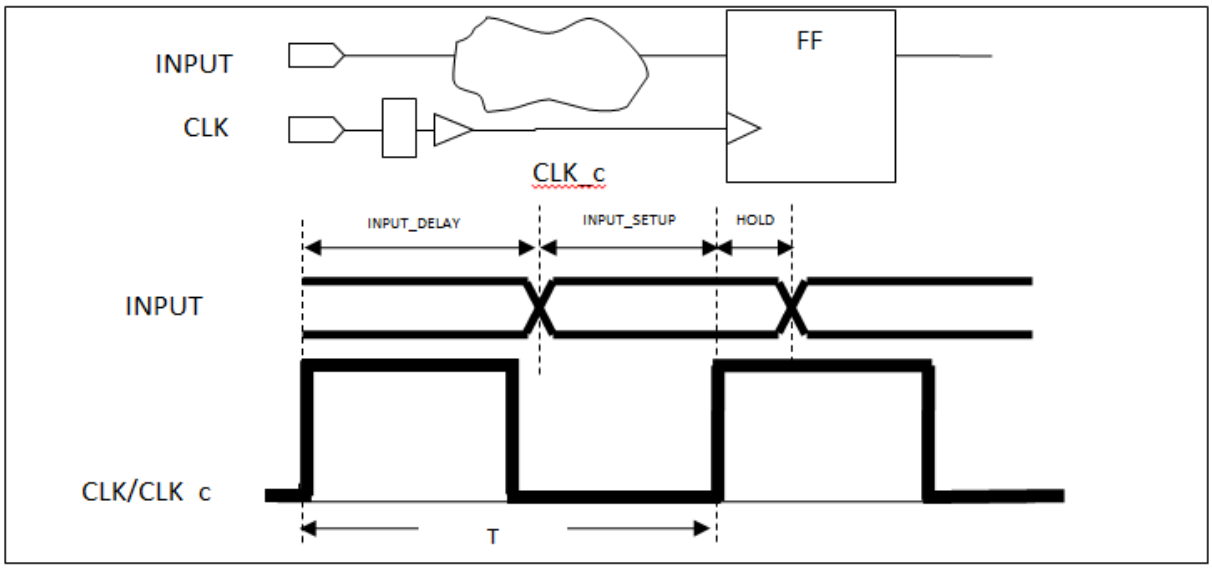
Input_setup (form)
INPUT_SETUP PORT “INPUT” <INPUT_SETUP_value>
HOLD <HOLD_value> CLKPORT “CLK”

Input_delay (form)
INPUT_SETUP PORT “INPUT” INPUT_DELAY
<INPUT_DELAY_value> HOLD <HOLD_value> CLKPORT “CLK”

It shows a no-skew clock with its period/frequency, and an input with its input setup, input delay and hold time. This also shows how the sum of input setup time and input delay is the clock period, so that when one value is known, the

other one can be easily calculated by subtracting the known value from the clock period.

Figure 6: Input Setup/Input Delay



Clock to Output

Clock_to_Out is the time difference between when the launch clock edge arrives at the FPGA input pin, and when the resulting data signal departs the FPGA (pin). The clock to out timing constraint value is when the FPGA must provide the data to meet the board timing and downstream device requirements. It is a function of the clock speed, destination device timing (its input setup requirement) and board timing. The FPGA meets this timing through the choice of the internal clock and data paths used.

The detailed timing example below shows the components of the two options to define output IO timing constraint: clock to out and output delay.

The external environment is given: a clock period of 20 ns, board clock skew of 1ns, board trace of 6 ns, and the destination device's input setup requirement of 5.5ns. This leaves at most 7.5 ns for the FPGA's clock to out timing.

Note

The clock to out value depends on the clock period value; if the clock period constraint changes, the clock to out constraint should also change.

Output delay is the portion of the clock period used by the environment outside of the FPGA. It includes the time for the signal to travel from the FPGA to the destination device (board trace), the input setup time required by

the destination device, and any time lost to board clock skew between the launch and capture clocks.

Note

The output delay value does not depend on the clock period value; if the clock period constraint changes, the output delay constraint does not change.

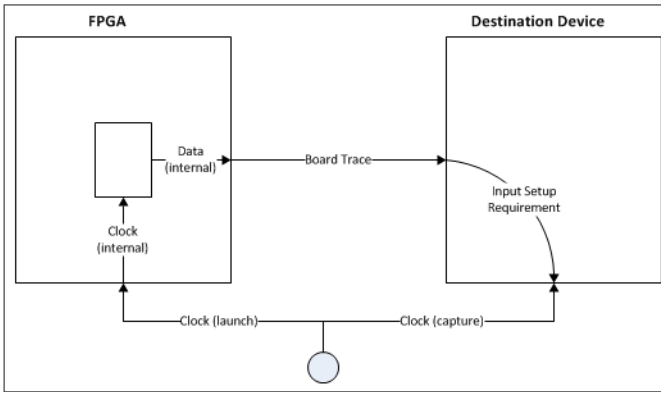
Clock to output and output delay are two different ways of looking at the same thing (if you know one, you know the other).

Clock to output + Output delay = clock period.

Both clock to out and output delay forms are specified using the CLOCK_TO_OUT preference.

The MIN time (in the CLOCK_TO_OUT preference) represents the smallest time for clock to out that will not result in a board level hold time violation.

Figure 7: Output Case



The following shows how clock to out and output delay forms are specified in the preference language. See the diagram in Figure 9 for reference:

Clock to out (form)
CLOCK_TO_OUT PORT “OUTPUT” <clock_to_out value>
MIN <HOLD_value> CLKPORT “CLK”

Output delay (form)
CLOCK_TO_OUT PORT “OUTPUT” OUTPUT_DELAY <output_delay value>
MIN <HOLD_value> CLKPORT “CLK”

Figure 8: Output Timing

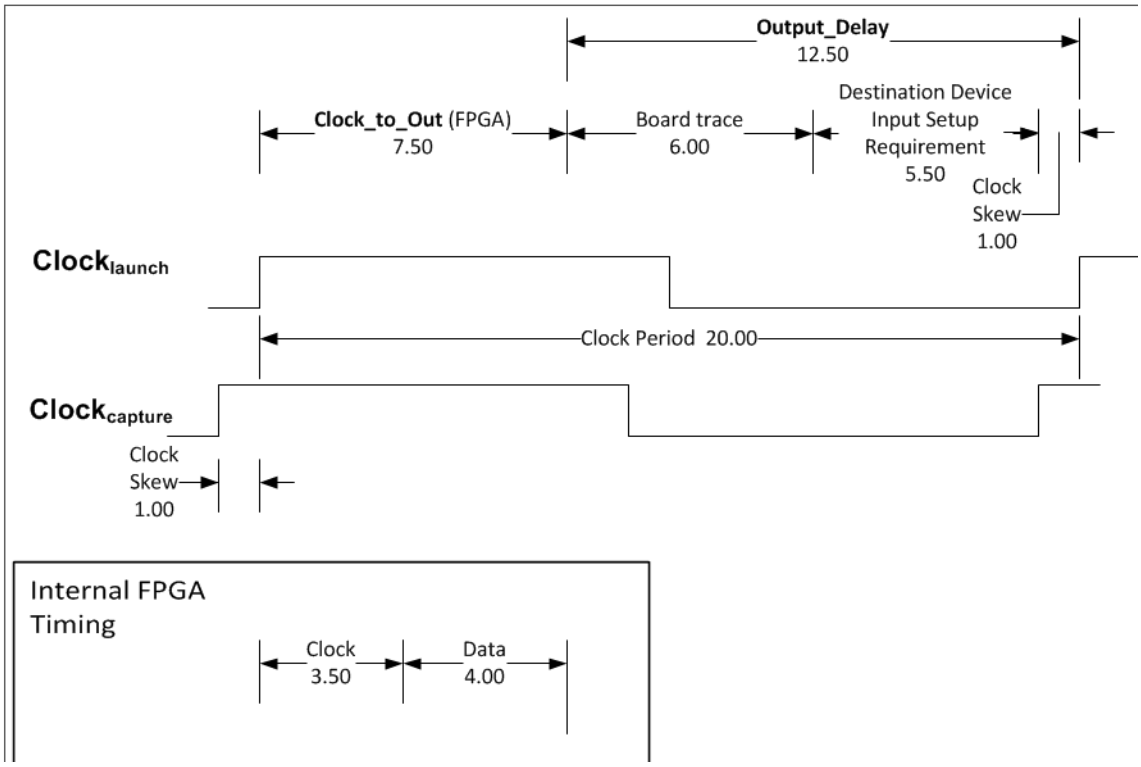
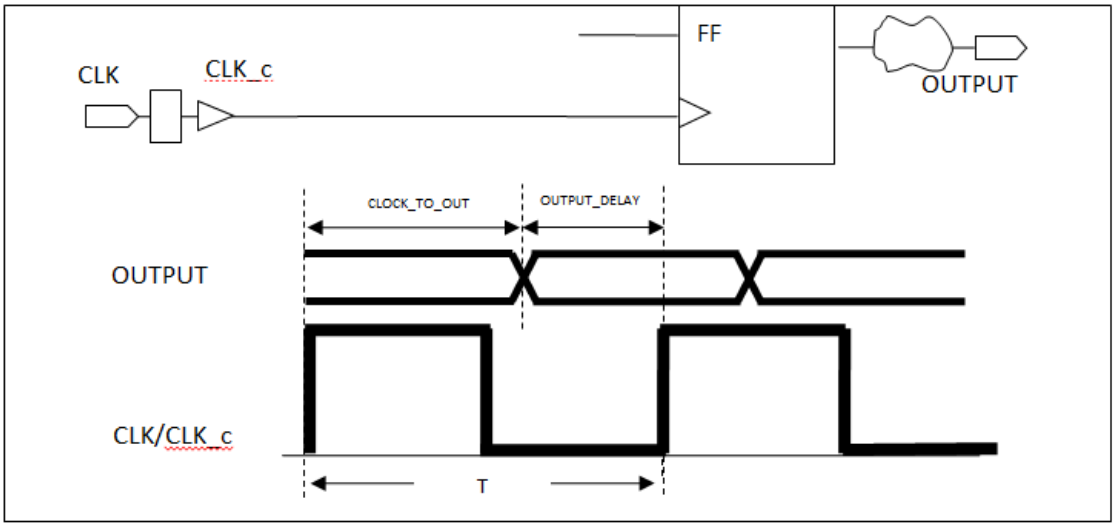


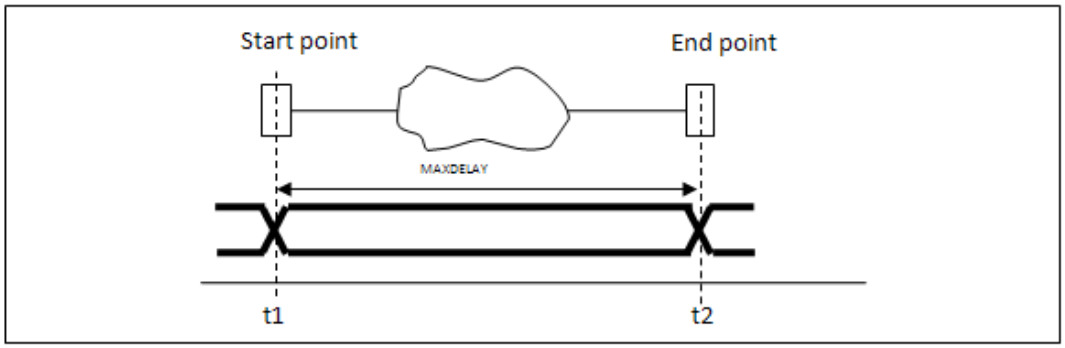
Figure 9: Clock to Output



Maximum Delay

Every net has a delay. The Maximum Delay constraint defines the maximum total time required for a net, bus or path, from a start point to an end point.

Figure 10: Maximum Delay



It illustrates the following timing preference:

MAXDELAY FROM <Start point> TO <End point> <t2-t1> ns

Exceptions

It can be necessary to specify exceptions to the standard timing analysis. These timing exceptions / modifications are considered timing requirements and captured in the timing constraints.

MULTICYCLE

Generally, in a synchronous design, a receiving register captures data using the next active clock edge after the edge that launched the data from the launching register. If both registers are clocked using the same clock signal, then this is one clock cycle. There are cases where the designer intends the time from a launching register to a receiving register to be different than this general case. The MULTICYLCE preference allows the designer to specify a timing requirement that is different than what the general/default case would use.

A MULTICYCLE constraint is a relaxation of the clock period / frequency analysis, and therefore only applies to paths covered by a clock period or frequency constraint. The launching register and the receiving register can be clocked by the same clock or different clocks. If driven by different clocks, these clocks must be related (if they are unrelated, the period / frequency will not be analyzed for the paths the cross between the clock domains).

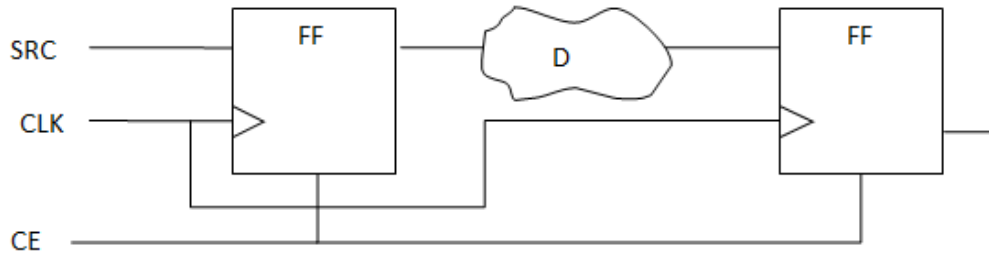
The diagram in Figure 11 illustrates the following timing preference:

MULTICYCLE FROM <Source Register> CLKNET “CLK” CLKEN_NET “CE” TO <Destination Register> CLKNET “CLK” 2 X

It illustrates a portion of a clock domain where a clock enable (CE) is running at half the speed of the clock and therefore slows the actual clock domain’s

effective frequency and allowing the data path more time to reach the capture register. If the clock was constrained to have a period of P, the MULTICYCLE constraint could then be used to constrain the data path D at 2 x P.

Figure 11: Clock Domain

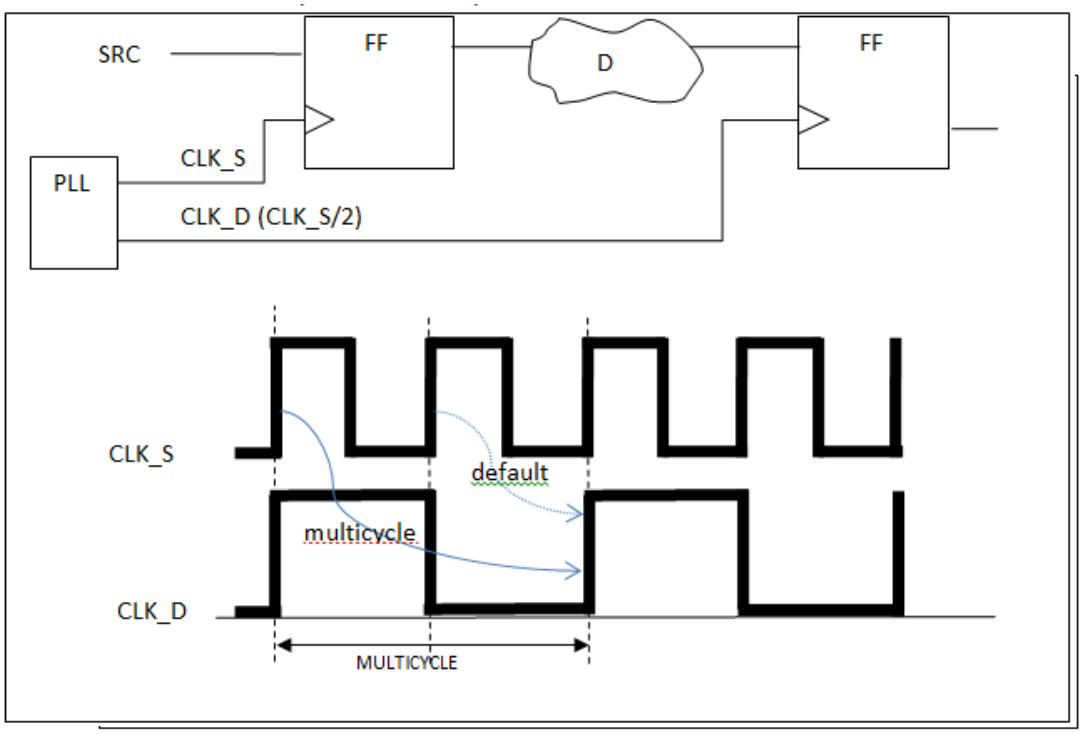


The diagram in Figure 12 illustrates the following timing preference:

MULTICYCLE FROM <Source Register> CLKNET “CLK_S” TO <Destination Register> CLKNET “CLK_D” 1X

It shows a data transfer from a clock domain to a clock domain running at half the speed (there is no skew between the clock domains). The default analysis will determine if the data path D’s time meets the period of the faster clock. MULTICYCLE can be used to constrain the data path to use the period of the slower clock.

Figure 12: Data Transfer



False Paths/Block

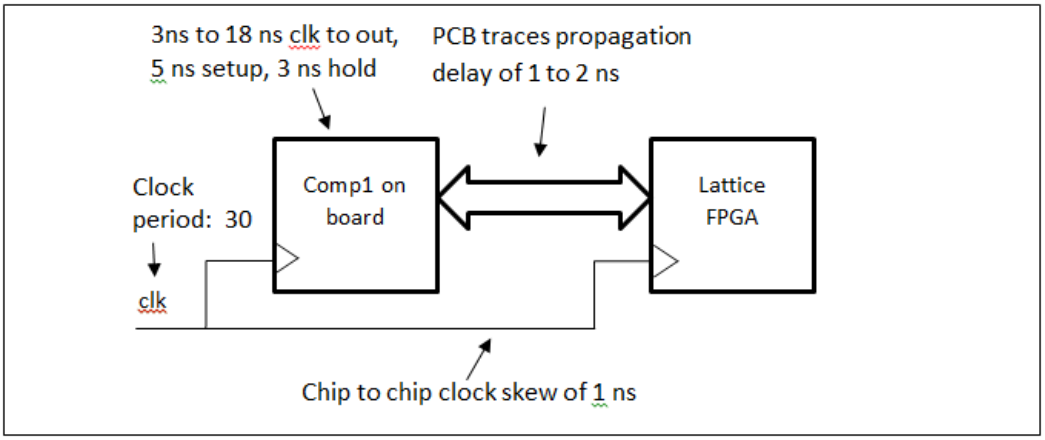
There can be paths in the design that are by default analyzed for timing, but their timing has no impact on the operation of the circuit. A simple example is an input that is tied to a constant VCC or GND on the board. During operation, there is no value transferred from the start of the path to the end, so its timing is not relevant.

Users can specify these paths to the tools using false path (synthesis constraint) and BLOCK (FPGA preference) to keep the flow from working on areas that have no impact.

Example: Calculate Timing Requirement

Understanding the system/board level timing and constraints is the primary requirement for producing a complete FPGA design timing requirement. The example shown in Figure 13 shows how to extract timing requirements from system conditions.

Figure 13: Calculate Timing Requirement



In this example, several parameters have already been provided:

- System clock period: P = 30 ns
- Comp1
  - maximum output propagation delay (clk_to_out): PDMAXp = 18 ns;
  - minimum output propagation delay (clk_to_out): PDMINp = 3 ns
- Comp1 input setup: TSp = 5 ns
- Comp1 input hold specification: THp = 3 ns
- Maximum board propagation delay: PDMAXb = 2 ns
- Minimum board propagation delay: PDMINb = 1 ns
- Clock skew: clock arrives 1ns earlier at FPGA than it arrives at Comp1.

- Clock skew of Comp1 to the FPGA device Tskew = 1 ns

From the information provided, we can capture the following timing requirements for the FPGA (each is written in the Lattice Preference language used for the backend tools):

- Clock period P = 30ns, or the frequency is 33.33MHz.

```
FREQUENCY PORT "clk" 33.33 MHz;
```

- Input setup = P - PDMAXp - PDMAXb - Tskew = 30 - 18 - 2 - 1 = 9 ns

```
INPUT_SETUP ALLPORTS 9 ns CLKPORT "clk" ;
```

- Input hold = PDMINp + PDMINb + Tskew = 3 + 1 + 1 = 5ns

```
INPUT ALLPORTS SETUP 9 ns HOLD 3 ns CLKPORT "clk" ;
```

- Output maximum propagation delay = P - TSp - PDMAXb + Tskew = 30 - 5 - 2 -+ 1 = 24 ns

```
CLOCK_TO_OUT ALLPORTS 24 ns CLKPORT “clk” ;
```

- Output minimum propagation delay = THp - PDMINb + Tskew = 3 - 1 + 1 = 3 ns

```
CLOCK_TO_OUT ALLPORTS MAX 24 ns MIN 3 ns CLKPORT “clk”;
```

Timing-Driven Flow Using Lattice Diamond Design Software

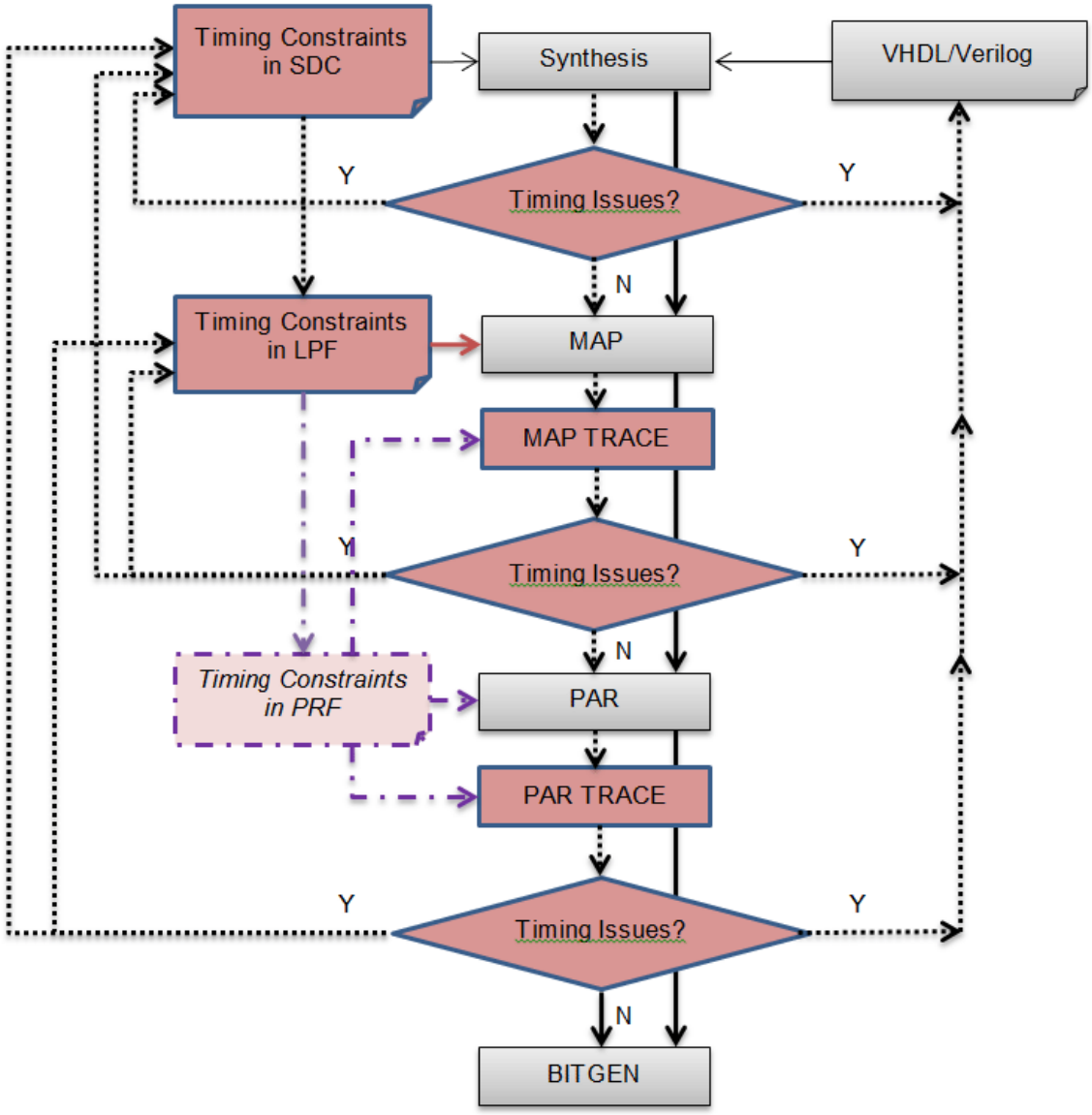
Every design has a timing requirement, no matter how fast or slow it runs. You should always constrain your design with a timing requirement and examine the static timing analysis results to ensure that your design functions correctly across the production silicon. In addition, you should understand how a timing-driven process works and how to interpret the timing-related process reports so that you can identify and fix potential timing issues.

The diagram in Figure 14 shows various places in the Lattice Diamond design flow where a user might need to pay attention and take action in order to achieve timing closure.

As shown in the diagram, each step in the process uses the timing requirements. It is possible to supply different timing requirements to Synthesis and the MAP/PAR processes. MAP/PAR always uses the same timing requirements. The general steps for each process include:

1. Gather timing requirements (as explained in “Example: Calculate Timing Requirement” on page 12) and define timing constraints.
   a. These must be captured in LPF format for the MAP/PAR processes to use.
   b. Optionally, these can also capture preferences in SDC format for the synthesis process.
2. Run the synthesis process. It is possible to run synthesis in a mode that ignores the SDC timing constraints. If you have defined SDC timing

Figure 14: Diamond Timing-Driven Flow



constraints and want them used, see "Timing-Driven Synthesis and Constraints" on page 15.

3. Run MAP and PAR, run TRACE (static timing analysis).
4. Review the process report. Also review the TRACE report, if applicable, and identify timing issues.
5. Based on the observations made in step 4, make adjustments to the RTL, strategy, or timing constraints.
6. Repeat the process until all timing issues are resolved.
7. Move to the next step.

Identifying and addressing timing issues at an early stage such as MAP rather than a later stage such as PAR will save a lot of time. Later processes in the flow usually take longer to run, so doing analysis and debug earlier in the flow provides a faster loop to make changes and see results. At MAP, you can easily see an issue with having much too many levels of logic on a path for the target FMax required and can avoid running PAR to see it.

Timing-Driven Synthesis and Constraints

Synthesis usually runs in a mode that it will use any supplied timing constraints. It is possible to set it up so that timing constraints, even when supplied by the user, get ignored. This section explains how to constrain and run logic synthesis tools, including Synplify Pro and Lattice Synthesis Engine (LSE), so that they do not ignore the supplied constraints.

Note

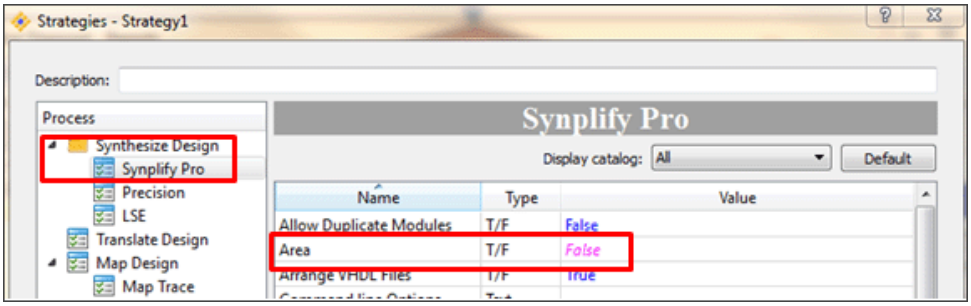
By default, the Map process ignores preference errors when it encounters them. To change this so that Map terminates and issues an error message whenever it encounters preference errors, set the “Ignore Preference Errors” option to “False” in the Map Design section of the active strategy or use the -pe option from the command line.

Synplify Pro

To run Synplify Pro so that it uses timing constraints (timing driven mode), you need to properly set up the active Strategy and define timing constraints.
Strategy Settings for Timing-Driven Mode Synthesis

Synplify Pro will use timing constraints if active Strategy has the “Area” setting = “False”, as illustrated in Figure 15. To accomplish this, you can use the predefined strategy called Timing, or you can make this setting in your own custom Strategy settings.

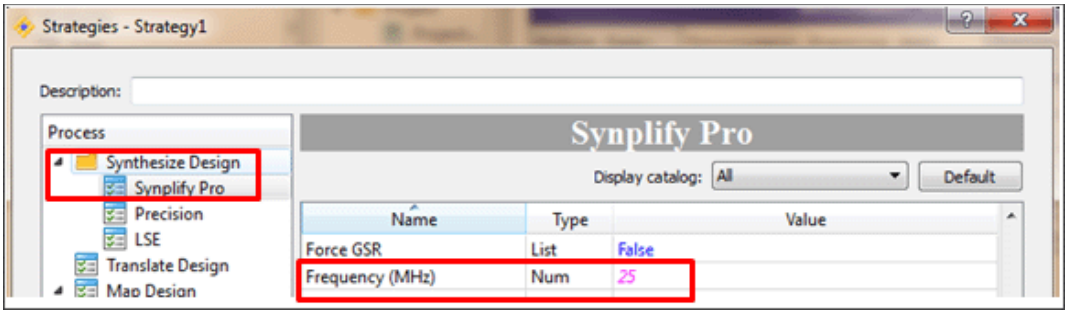
Figure 15: Strategy – Changing Area Strategy for Synplify Pro



Specify Clock Frequency Timing Constraints Setting

You may also need to change the target frequency to the required value for your design, as shown in Figure 16. The default value for this is 200MHz. If you specify all the frequency/period requirements for your clocks in your SDC file, this value will be ignored.

Figure 16: Strategy – Setting Frequency for Synplify Pro



Other Timing Related Strategies

You can use other strategy settings to improve your design's performance. Depending on the actual design and preliminary synthesis result, you might want to use the following strategy settings:

- Pipelining and retiming – allows Synplify Pro to move registers into combinatorial logic or create pipelines for multipliers in order to improve performance. By default, this option is turned off.

  Synthesis retiming usually works better around DSP and EBR blocks; because synthesis timing model is not aligned with PAR, it might not work well in all cases

- Resource sharing – allows Synplify Pro to reduce area by sharing certain resources. Turning this off can improve the performance but at the expense of increased area

Timing Constraints

You can use two types of constraint to define your timing requirements:

- Synplify Design Constraints (Synplify SDC)
- Synopsys Design Constraints (Synopsys SDC)

The two types of constraint cannot be mixed and used together. They cannot be used in the same constraint file (.sdc) or in separate multiple constraint files. You need select one type to drive the Synplify Pro process in timing-driven mode.

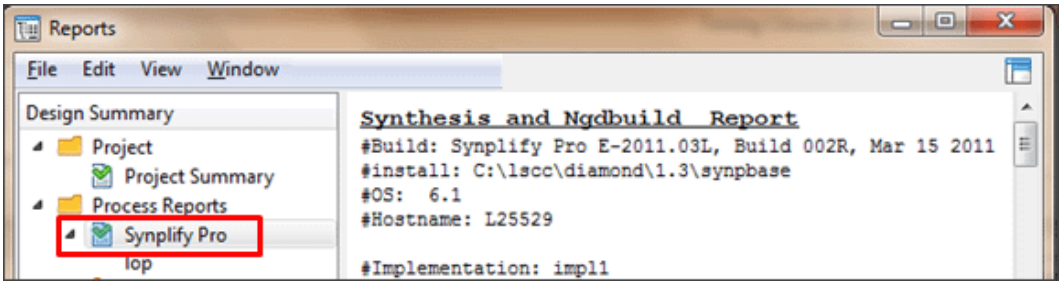
Remember that Synplify Pro must be in timing-driven mode in order to have your timing constraints applied to the synthesis process; otherwise, your timing constraints might be ignored.

For detailed information about using timing constraints through Synplify SDC and Synopsys SDC, refer to the *Synopsys FPGA Synthesis User Guide* and *Synopsys FPGA Reference Manual*.

Understanding the Synplify Pro Timing Report

With the appropriate synthesis strategies and timing constraints, you can start the Synplify Pro synthesis process. The synthesis report includes timing-related information and can be viewed in the Diamond Report View. You can also access the report through the Synplify Pro user interface. Refer to *Synopsys FPGA Synthesis User Guide* and *Synopsys FPGA Reference Manual*)

Figure 17: Synthesis Report in Lattice Diamond



In the report, look at the section enclosed between these 2 lines:

```
##### START OF TIMING REPORT #####[
……
##### END OF TIMING REPORT #####]
```

A few places where you need to pay attention:

- The summary at the top, as shown in the following example. Ensure that the appropriate SDC file was used and that the required frequency and other timing constraints defined in the SDC file were included.

```
Top view:               demo
Requested Frequency:    25.0 MHz
Wire load mode:         top
Paths requested:        3
Constraint File(s):     C:\projects\demo\demo.sdc
```

- Performance summary, which gives the worst slack in the design:

```
Performance Summary
*******************
Worst slack in design: 14.892

Starting Requested Estimated Requested Estimated        Clock    Clock
Clock    Frequency Frequency Period    Period    Slack  Type     Group
------------------------------------------------------------------------
clk      25.0 MHz  97.9 MHz  40.000    10.217    14.892 inferred clkgroup
```

- Clock relationship, which shows register-to-register slacks. If your design includes multiple constrained clocks, they will be all included here:
- Interface information, which shows input setup and clock to output timing information and slacks.

```
*******************
Clocks           | rise to rise     | fall to fall     | rise to fall     | fall to rise
-------------------------------------------------------------------------------------
Starting Ending |constraint slack |constraint slack |constraint slack |constraint
slack
-------------------------------------------------------------------------------------
-clk       clk     | 40.000    30.187 | 40.000    37.483 | 20.000    14.892 | 20.000
16.099
```

For complete information about the Synplify Pro report, refer to the *Synopsys FPGA Synthesis User Guide* and *Synopsys FPGA Reference Manual*.

Remember that the Synplify Pro timing report is generated from the synthesis result, which does not have any placement and routing information. To get the highly accurate timing analysis result, run PAR TRACE, as explained in the section “PAR TRACE” on page 32.

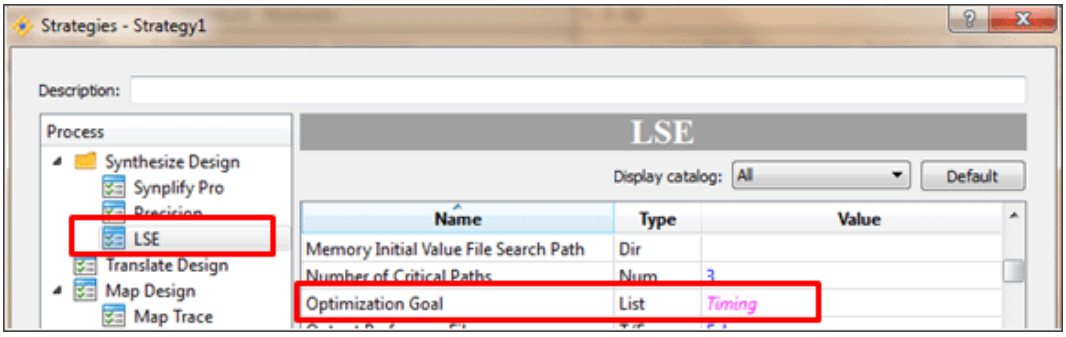
LSE

To run LSE so that it uses timing constraints (timing-driven mode), you need to properly set up the active strategy and define timing constraints.

Strategy Settings for Timing-Driven Mode Synthesis

LSE will use timing constraints if the active strategy has the “Optimization Goal” setting = “Timing,” as illustrated in Figure 18.

Figure 18: Setting LSE Optimization Goal



To accomplish this, you can use the predefined strategy called Timing, or you can make this setting in your own custom strategy settings.

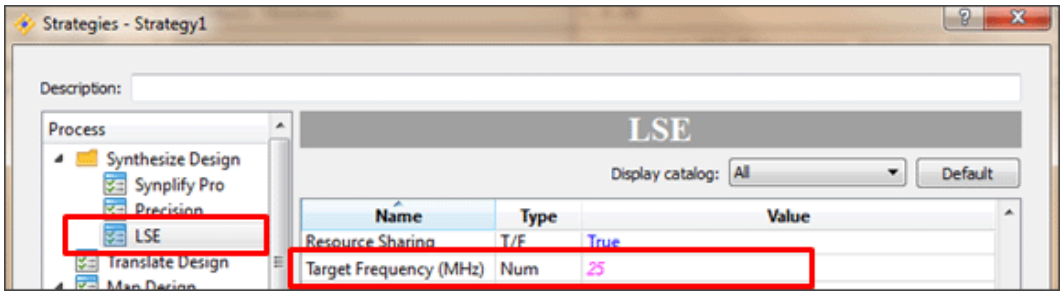
Specify Clock Frequency Timing Constraint Setting

You might also need to change the target frequency to the required value for your design, as shown in Figure 19. The default value for this is 200MHz.

Other Timing Related Strategies

Other strategy settings for LSE may improve your design’s performance. Depending on the actual design and preliminary synthesis result, you can change these strategy settings, but remember that all the following suggested settings are at the expense of increased area:

Figure 19: Setting LSE Clock Frequency



- Use “one hot” state machine encoding style if your design includes state machines.
- Reduce “max fanout limit” to a reasonable number or minimum.
- Disable “Remove Duplicate Registers.”
- Turn off Resource sharing. Resource sharing, when enabled, allows LSE to reduce area by sharing certain resources. Turning this off might improve the performance.

Timing Constraints

LSE supports Synopsys Design Constraints for timing-driven logic synthesis. LSE supports the following Synopsys Design Constraints:

- `create_clock`
- `set_input_delay`
- `set_output_delay`
- `set_max_delay`
- `set_multicycle_path`
- `set_false_path`

Your constraints must be written in an LSE Design Constraint file (.ldc) that is included and set as the active synthesis constraint file in your implementation.

Remember that you must enable timing-driven mode for LSE in order to have your timing constraints (.ldc) applied to the synthesis process; otherwise, your timing constraints will be ignored.

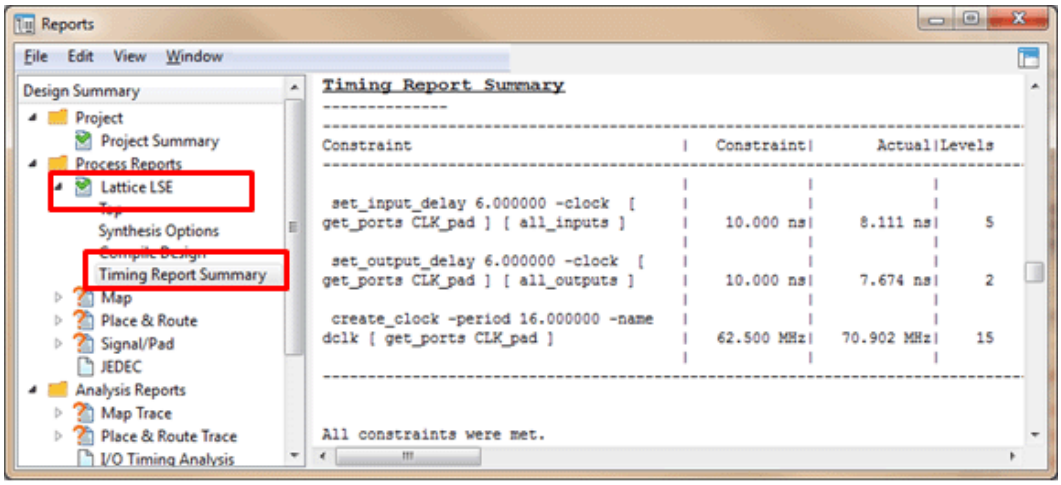
Understanding the LSE Timing Report

With the appropriate synthesis strategies and timing constraints, you can start the LSE synthesis process. The synthesis report includes timing-related information, and it can be viewed in the Diamond Report View, as shown in Figure 20.

Make sure that the report includes all of your defined timing constraints. In this example, we can see that all the requirements (clock period, input setup and output delay) are met.

You can also view the detailed LSE synthesis timing report. The detailed timing report is written in a file in the implementation directory, with the file

Figure 20: LSE Timing Report



name convention <*prj_name*>_lse.twr, where <*prj_name*> is the project name. You can view this file in any text editor.

Remember that the LSE timing report is generated from the synthesis result, which does not have any placement and routing information. To get the accurate timing analysis result, run PAR TRACE, as explained in "PAR TRACE" on page 32.

Understanding TRACE

TRACE is the static timing analysis tool in Diamond.

Static timing analysis (STA) is a fast and powerful verification technique for validating design performance. It is one of the most important steps in the design flow, and it should be considered as important as the functional verification performed with a logic simulator. TRACE verifies circuit timing by totaling the propagation delays along paths between clocked or combinational elements in a circuit. TRACE determines and reports timing data, such as the critical path, setup time and hold time, and the maximum frequency.

You can run TRACE on mapped designs or on completely placed and routed designs.

TRACE enables you to do the following:

- Confirm that the timing constraints supplied to timing-driven MAP and PAR have been met.
- Examine the timing of any part of the design
- Perform what-if scenarios with different device speed grades or timing objectives

TRACE provides the primary-input-to-primary-output timing analysis, i.e., maximum delay, as well as the following types of setup time analysis, together with the hold time analysis:

- From FPGA input to register, i.e., input setup time (INPUT_SETUP)

- From register to register, i.e., maximum clock frequency or minimum period (FREQUENCY or PERIOD)
- From register to output, i.e., clock to output (CLOCK_TO_OUT)

TRACE performs two types of timing analysis: Setup and Hold. Setup time analysis ensures that the data arrives at the receiving registers before the next capturing clock edge. Hold time analysis ensures that the data does not arrive at the receiving registers too early, thus is captured by the clock edge prior to the intended capture edge. The examples from Table 1 explain this in detail.

Table 1: Setup and Hold Timing Analysis

constraints	setup timing analysis	hold timing analysis
FREQUENCY/PERIOD	Does the data from the source register arrive at the destination register early enough relative to the capture clock edge to meet the setup time of the destination register?	Does the data from the source register not arrive at the destination register so early relative to the capture clock edge that it will be clocked by the clock edge prior to the capture clock edge?
INPUT_SETUP	Knowing that the data arrives at the device pin no later than 10ns before its capture clock edge, does the data then arrive at the internal register early enough relative to the capture clock edge to meet the setup time of the internal register? `INPUT_SETUP ALLPORTS 10ns HOLD 2ns CLKPORT "clk";`	Knowing the time that the data will hold its value (and not transition to new data) at the device pin at least 2ns after its capture clock edge, does the data then arrive so early at the internal register relative to the capture clock edge that it will be clocked by the clock edge prior to the capture clock edge? `INPUT_SETUP ALLPORTS 10ns HOLD 2ns CLKPORT "clk";`
CLOCK_TO_OUT	Does the data meet the board level setup time by leaving the output pin 10ns (or earlier/less) after the reference clock edge arrives at the pin? `CLOCK_TO_OUT ALLPORTS MAX 10ns MIN 2ns CLKPORT "clk";`	Does the data meet the board level hold time by leaving the output pin 2ns (or later/more) after the reference clock edge arrives at the pin? `CLOCK_TO_OUT ALLPORTS MAX 10ns MIN 2ns CLKPORT "clk";`

TRACE uses different performance grades and conditions when doing Setup time vs. Hold time analysis. Table 2 has more details. Although Place and Route (PAR) runs as a single process, there are two distinct steps: (1) Meet setup, (2) Meet hold (done via the Hold Time Correction sub-step). The table also summarizes the behaviors of these two steps.

Table 2: Performance Grades and PAR Behavior

	setup timing analysis	hold timing analysis
Default performance grade used (can be changed by the end user)	Performance grade of the target device; for example, 6	-m
Worst case conditions used (from data in speed grade file)	- Slow/max data - Fast/min clock	- Fast/min data - Slow/max clock

Table 2: Performance Grades and PAR Behavior (Continued)

	setup timing analysis	hold timing analysis (Continued)
PAR step – meet setup	Timing score is based on setup violations. PAR Works to make timing score (accumulated setup time negative slacks) zero	Ignored
PAR step – meet hold (via hold time correction (HTC))	Ignored	Delay is added to datapath to fix any hold time violations. Does not add more delay than the minimum required, to avoid creating a setup violation, but a setup violation might be created.

The performance grade -m represents the fastest possible PVT corner. The voltage used for this option is 5% above the nominal value, and the temperature used is -40C.

For register-to-register timing analysis, the default grades used represent the “worst” case for the setup analysis and the hold time analysis.

For FPGA I/O timing analysis, meaning INPUT_SETUP and CLOCK_TO_OUT, it is possible that the default grades used will not represent the worst case. The worst case depends on your design and the final placed and routed design. If PAR TRACE reports no timing errors, you should still run I/O timing analysis to sweep across speed grades faster than your target speed grade to ensure that I/O timing is satisfied. Refer to the section “I/O Timing Analysis” on page 33 for the details.

TRACE can be run on a post MAP netlist prior to place and routed where routing delay is an estimate, or after place and route (see “Understanding the PAR and PAR TRACE Reports” on page 32).

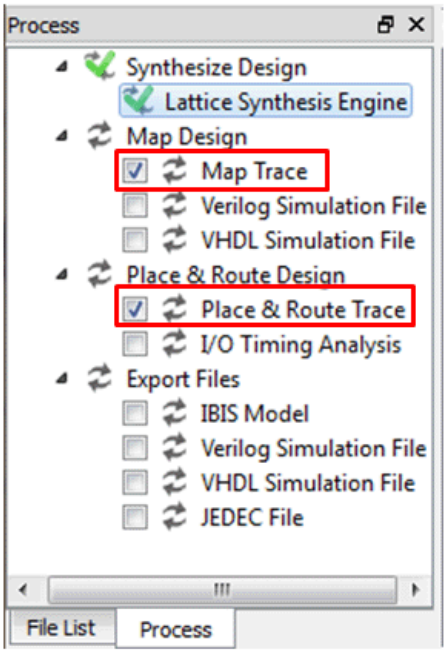
The MAP TRACE and PAR TRACE Reports

The MAP TRACE and PAR TRACE reports do the same analysis, but they are generated independently at different stages of the design flow. The MAP TRACE report can be created earlier and faster, but is less accurate. It is ideal for identifying basic problems with the design constraints or the design itself. The PAR TRACE report is used for detailed timing analysis and signoff.

The two reports are generated in a very similar manner.

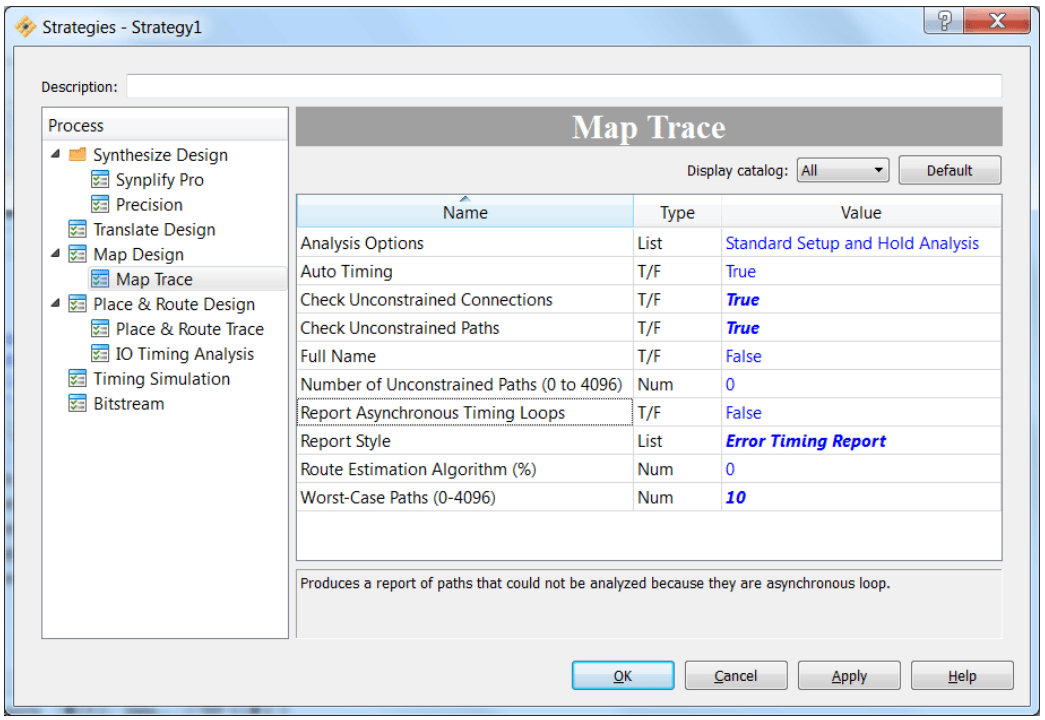
You can enable MAP TRACE and PAR TRACE (the GUI labels it as “Place & Route Trace”) in Diamond Process window, as shown in Figure 21, so that as soon as the MAP (or PAR) process finishes successfully, MAP (or PAR) TRACE starts automatically. You can also run MAP (or PAR) TRACE by double-clicking it in the same window.

Figure 21: Map Trace and PAR Trace Processes



You can specify a few options in the MAP TRACE Strategy settings to control the MAP TRACE process.

Figure 22: MAP TRACE Strategy Settings



These settings can help you quickly identify any timing issues that might exist in your design:

- Check Unconstrained Connections – Setting this to True will list the paths that are not covered by any timing preference.

  Note

  The Check Unconstrained Connections option will be discontinued after the next two Diamond releases.

- Check Unconstrained Paths – Setting this to True will report the paths that are not constrained and shows the start point and end point of each path. TRACE will suggest some timing preferences to constrain the given paths. The unconstrained paths are shown only in the “setup” timing check report to avoid duplication of these same paths in the “hold” timing check report.

  Based on the design and the required performance, only necessary paths should be constrained so that PAR focuses only on the optimization of the important paths. However, the Unconstrained Paths section of the TRACE report is very useful for identifying whether any missing timing constraints are really important to the design. This option does not require you to add more preferences in an attempt to constrain all paths. Instead, it serves as a reminder that there might be a necessary preference that is missing, which could impact the desired performance of the design.

  Note

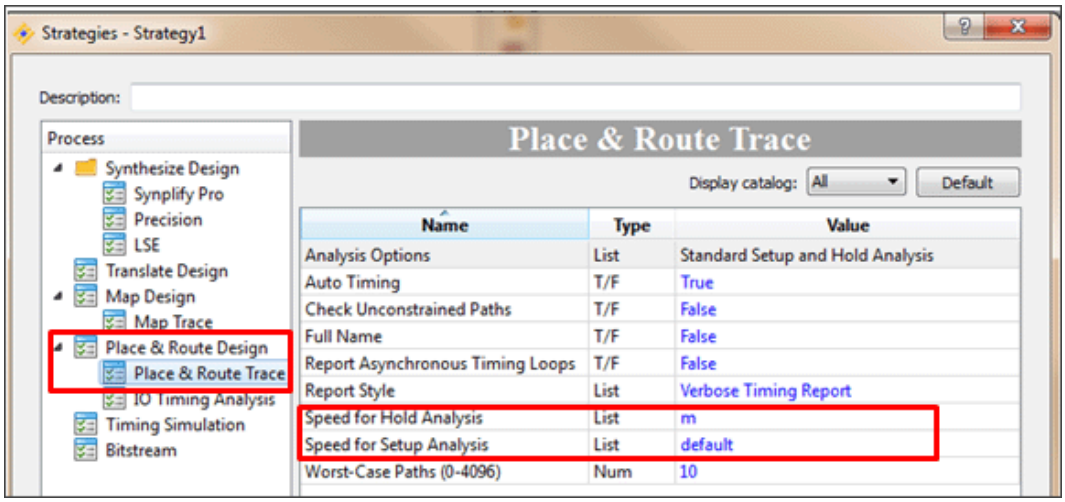
  The Check Unconstrained Paths option cannot be used with the -allprefpath command-line option.

- Full Name – Setting this to True causes TRACE to report full-length component names instead of the truncated names. This enables you to find a specific path easily. Turning this on might, however, cause format alignment issues in the report when names are long.
- Report Style – Set this option to “Error Timing Report” so that TRACE only reports paths and nets that have timing errors. This allows you to identify any timing issue quickly.

You can also specify a few PAR TRACE options through the Strategy settings to control the PAR TRACE process, with the following differences:

- Speed for Hold Analysis – You can select the speed grade for the hold analysis By default, this value is set to “m”, or “minimum”, which represents the virtual silicon that is faster than the fastest speed grade of the device available. If the analysis result reveals no hold time violation using the value “m”, then it guarantees there will be no hold time violation for all speed grades, including the one you selected for your project. There are some cases of hold time violations with the use of “m”, but there might be no violation for the speed grade you selected for your project. If being able to migrate to a faster speed grade is not your concern, you can set this value to the actual number selected for your project.
- Speed for Setup Analysis – You can select the speed grade for the setup analysis. By default, this value is set to “default”, which is the speed grade you selected for your project.

Figure 23: PAR TRACE Strategy Settings



The MAP TRACE and PAR TRACE reports have the same format. Table 3 shows a summary of their differences.

Table 3: TRACE Report Differences

	MAP TRACE	PAR TRACE
Report filename extension	.tw1	.twr
Routing timing	Estimated	Based on actual placed and routed path
Netlist used	NCD that has not been placed or routed	NCD that has been placed and routed
Best used for	▶ Quickly identifying issues with constraints (.e.g, syntax errors) ▶ Quickly finding timing issues with too many logic levels	▶ Detailed timing analysis and signoff

The TRACE report can be viewed in the Diamond Report View, as shown in Figure 24. You can also view the TRACE report files in the implementation directory using a text editor. Both reports use the same naming convention for the prefix and use a different file extension (see table above). The naming convention for the prefix is <*prj_name*>_<*impl_name*>, where <*prj_name*> is your project name and <*impl_name*> is the implementation name.

Based on the type of analysis (setup, hold, etc.) set through the Analysis Options in the MAP TRACE (or PAR TRACE) strategy settings, you might see the report information differently. In the example shown above, the result of setup time and hold time analysis can be examined. You can quickly jump to a few areas to see if the result meets your timing requirements (preferences), or to find more information about your design, as follows:

Figure 24: TRACE Report

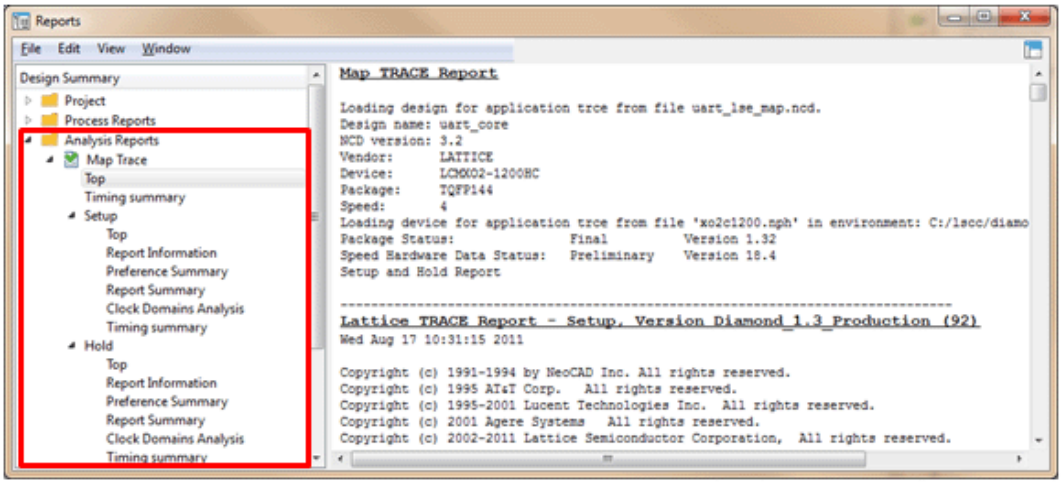
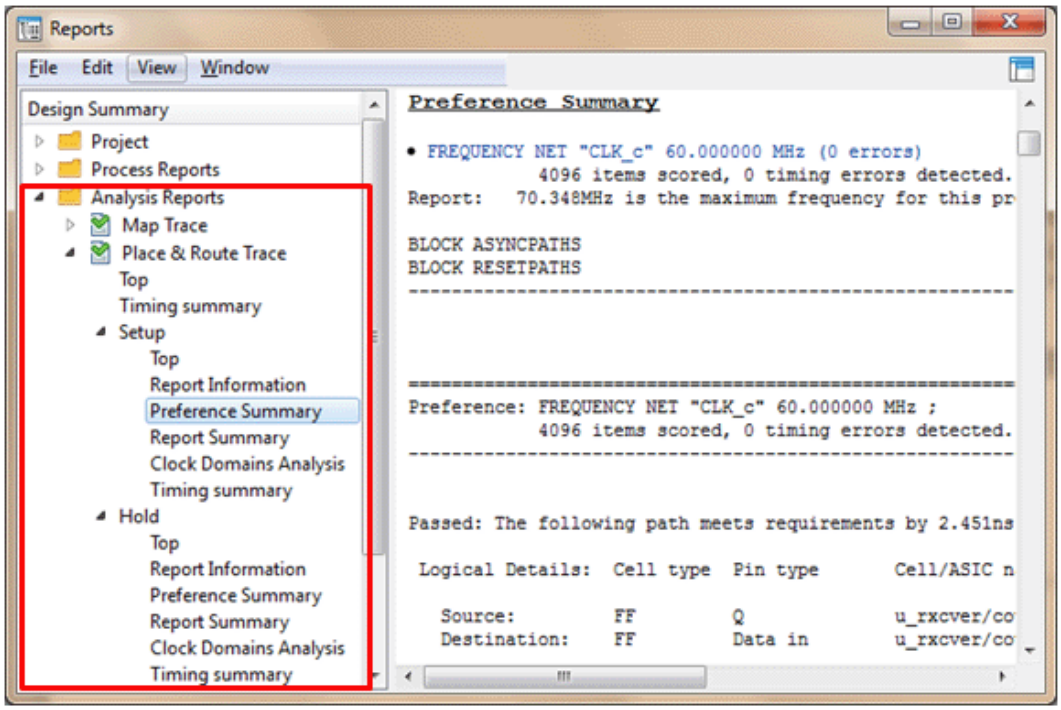


Figure 25: TRACE Report Preference Summary



- Timing summary at the top – This section summarizes the total number of timing errors and timing scores for both setup time analysis and hold time analysis.
- Following is separate Setup and Hold analysis sections. They have the same format.
  - Preference Summary – This section lists your timing requirements and the actual results. It lists the timing preferences and the corresponding

setup or hold analysis results (whichever is appropriate for the report) for defined clock FREQUENCY/PERIOD, MULTICYCLE, MAXDELAY, INPUT_SETUP and CLOCK_TO_OUT preferences. Timing errors are highlighted in red.

- Clock Domains Analysis – this section summarizes the clock domains in your design (e.g. number of loads for each), and the number of transfers between any two clock domains. It also summarizes if or how they are constrained (e.g. FREQUENCY, MULTICYCLE). Missing constraints are reported. The following is an example report:

```
Clock Domains Analysis
------------------------
Found 2 clocks:
Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk1" 300.000000 MHz PAR_ADJ 3.000000 ;
   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Covered under: MULTICYCLE FROM CLKNET "clk2_c" TO CLKNET "clk1_c" 2.000000 ns ;
Transfers: 1

Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk2" 150.000000 MHz ;
   Data transfers from:
   Clock Domain: clk1_c   Source: clk1.PAD
      Covered under: MULTICYCLE FROM CLKNET "clk1_c" TO CLKNET "clk2_c" 3.000000 ns ;
Transfers: 1
```

In this example, there are two clock domains: “clk1” and “clk2”. Both of these clock domains are covered by their own FREQUENCY preferences. In addition, there are cross-domain paths between these two clocks, and they are covered by their own MULTICYCLE preferences.

Remember that MAP TRACE runs on the mapped result, which does not have any placement and routing information; instead, MAP TRACE uses the “Route Estimation Algorithm” defined through the MAP TRACE strategy settings to estimate routing delays. To get the accurate timing analysis result, run PAR TRACE, as explained in “PAR TRACE” on page 32.

MAP Process and Constraints

MAP takes constraints as input, and passes many through (sometimes in a modified form) to the next step, PAR.

There is some information in the constraints that MAP does use to alter the netlist it creates. For example, buffer type and other configuration settings for IO can be specified in the constraints, and MAP will set these up in the netlist that it creates for PAR.

Constraints

There are two types of constraints. Timing constraints are timing goals that the design is to meet. Placement constraints directly affect the physical layout of the netlist when it is put into the device. An example of a placement constraint is assigning a design's top level port to a specific device pin. Constraints passed to MAP can come from two different sources:

- Entered in the HDL and passed to MAP inside the NGD file
- Entered in the Lattice Preference File (LPF)

There are a few ways to create constraints in an LPF file:

- Use the Spreadsheet View (timing preferences are held in the Timing Preferences sheet),
- Text edit the LPF file.
- Instruct the synthesis process to write synthesis timing constraints into an LPF file. The contents of that file can then be copy/pasted into the active LPF using a text editor. You need to enable the option "Output Preference File" in the synthesis strategy settings to instruct the synthesis process to generate an LPF file. By default, this option is turned off.

You can use the following preference commands to define your timing constraints in LPF:

- FREQUENCY/PERIOD
- INPUT_SETUP
- CLOCK_TO_OUT
- MULTICYCLE
- MAXDELAY
- CLKSKEWDIFF
- BLOCK
- OFFSET

See "Timing Requirements and Constraints" on page 2 for commonly used timing constraints.

Timing-Driven MAP Strategy

The relatively high granularity of the FPGA architecture limits the impact that the MAP process can have on timing results. The final timing achieved by a given HDL design is mainly influenced by Synthesis (which can optimize the netlist over a large scope) and PAR (which can do detailed placement and routing).

However, there are a few timing related options in the MAP strategy settings that you can turn on to drive the MAP process. By default, they are turned off and this should be sufficient for most designs. Turning on the following options, in some cases, will be helpful if your design has difficulty meeting the timing requirement. If enabled, these consider the timing constraints input to MAP.

- Register Retiming –Turning on this option instructs the MAP process to move registers across combinatorial logic to balance timing according to the INPUT_SETUP, CLOCK_TO_OUT, and FREQUENCY constraints. You must have these constraints defined to ensure this option works correctly.

  MAP retiming usually works better for congested designs. However, because the MAP timing model is not the same as PAR, it may not work well all the time.

- Timing Driven Mapping – Turning on this option instructs the MAP process to calculate the slack time for all constrained paths and optimize the critical paths based on the slack distributions.
- Timing Driven Node Replication – Turning on this option instructs the MAP process to replicate a LUT4 that has multiple-fanout flip-flops. It adds a LUT for each flip-flop when the LUT belongs to the timing path, thus packing LUT/FF in the same slice for all flip-flops.
- Timing Driven Packing – Turning on this option instructs the MAP process to do timing -driven packing of LUT/FF, FF/LUT, and LUT/LUT in the same slice

MAP TRACE

The content of the MAP TRACE report, and how it is generated, is described in the section “The MAP TRACE and PAR TRACE Reports” on page 22. It can be generated right after the MAP step, and by avoiding the PAR step, to more quickly see if there are gross issues with the defined timing constraints or the design itself.

PAR Process and Constraints

This section explains how to constrain and run PAR in timing-driven mode.

Understanding PAR

PAR performs the following tasks:

- It takes a mapped physical design (.ncd file) and a preference file (.prf) as input files. The .ncd file and .prf file are the outputs of the MAP process. See Preferences and Processes.
- It uses its timing driven engine to place and route the design with the goal of meeting the placement constraints and the timing preferences defined in the input .prf file. As explained in the second table in the section Understanding TRACE, PAR first works to make the setup timing score zero. If auto hold time correction is enabled in PAR, PAR then works to correct hold time violations. Auto Hold Timing Correction is enabled through the PAR strategy settings, as explained in PAR (Place & Route Design) Settings in Strategy for Timing Closure. For releases prior to Diamond 2.0, this must be enabled by the user (i.e. default setting was disabled)

- When PAR finishes successfully, it creates a placed and routed physical design file (.ncd file) that can be processed by the Diamond BITGEN tool.

Placement

The PAR process places the mapped physical design (.ncd file) in two stages: constructive placement and optimizing placement. PAR writes the physical design after each of these stages is complete.

During constructive placement, PAR places components into sites based on factors such as the following:

- Constraints specified in the input file. For example, certain components must be in certain locations.
- The length of connections
- The available routing resources
- Cost tables that assign random weighted values to each of the relevant factors. There are 100 possible cost tables, and they can be set through PAR strategy settings.

Constructive placement continues until all components are placed.

Optimizing placement is a fine-tuning of the results of the constructive placement.

Routing

Routing is also done in two stages: iterative routing and delay reduction routing (also called cleanup). PAR writes the physical design (.ncd file) only after iterations where the routing score (accumulated setup timing slacks) has improved.

During iterative routing, the router attempts to converge on a solution that routes the design to completion or minimizes the number of unrouted nets.

During delay-reduction routing, the router takes the results of iterative routing and reroutes some connections to minimize the signal delays within the device. Two types of delay-reduction routing are performed:

- A faster cost-based cleanup routing, which makes routing decisions by assigning weighted values to the factors (such as the type of routing resources used) that affect delay times between sources and loads
- A more CPU-intensive, delay-based cleanup routing, which makes routing decisions on the basis of computed delay times between sources and loads on the routed nets

Timing-Driven PAR Process

If PAR finds timing preferences in the preference file, timing-driven placement and routing is automatically invoked. It is extremely important to include timing preferences.

The timing-driven PAR process uses the TRACE static timing analysis engine. PAR works to meet the specified timing preferences.

PAR can run in two basic modes. The mode is set via the Auto Hold-Time Correction setting in the PAR (Place & Route Design) section of the active Strategy being used.

- Meet setup and hold. Auto Hold-Time Correction = On. This is the recommended mode. PAR will work to meet both setup and hold time so that there are no violations. This is the default mode for Diamond release 2.0 and later. Therefore, users must turn this mode on in prior releases.
- Meet setup (and report on hold). Auto Hold-Time Correction = Off. This is not the recommended mode. PAR will work to meet setup only. If there are hold time violations, PAR will not attempt to correct them. This is the default mode for Diamond releases prior to 2.0. This mode may be useful early in the design closure process when the focus is on meeting setup time, and user wants to save runtime.

In either mode, Trace report will include setup and/or hold analysis – whatever the user chooses (default is for both) in the Analysis Options of the Trace report Strategy settings. Any violations will be reported. You should examine the PAR TRACE report for setup and hold-time analysis results.

PAR (Place & Route Design) Settings in Strategy for Timing Closure

These settings are set in the active Strategy, under the Place & Route Design section. Most of them are turned off by default because they are not needed for most designs. There are some cases, though, where turning on the following options will be helpful if your design has difficulty meeting the timing requirements.

- Auto Hold Time Correction – this is described in the preceding section.
- Clock Skew Minimization -- If there is any clock signal that is not assigned to the global clock tree, enabling this option will allow PAR to balance routing to reduce clock skews.
- Disable Timing Driven – By default, this option is off, which means that PAR runs timing-driven placement and routing based on your timing constraints. You might want to disable timing-driven PAR on those occasions where you want to have a quick PAR run and get a rough idea of the difficulty of placing and routing your design.
- Path Based Placement – Turning on this option allows PAR to do path-based placement, which usually yields better performance.
- Routing Method – Setting this option to NBR instructs PAR to use an iterative routing algorithm that could produce better results in performance and some other areas. This is on by default.

Timing Constraints

PAR (as with MAP) takes constraints as input. These constraints are passed to PAR from MAP in a file referred to as the Physical Preference File (PRF, has file extension .prf). The PRF is not a user created file. User puts constraints into the LPF file, MAP then generates the PRF from the LPF, and

then PAR runs against the PRF. User edits made directly to the PRF will be lost if/when MAP is run. Therefore, the PRF should not be edited. PRF is not accessible from the Diamond GUI.

PAR TRACE

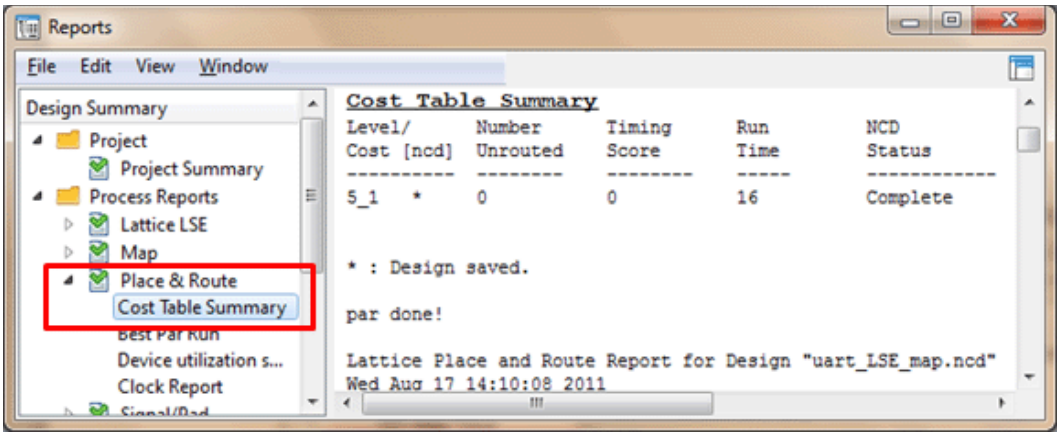
The content of the PAR TRACE report, and how it is generated, is described in the section The MAP TRACE and PAR TRACE Reports. It can be generated right after the PAR step; it holds the final timing for the design.

Understanding the PAR and PAR TRACE Reports

With the appropriate PAR TRACE strategies and timing constraints, you can start the PAR and PAR TRACE processes.

The quickest way to find out if the final PAR result meets the timing requirement is to look at the Timing Score reported in the PAR report, as shown in Figure 26.

Figure 26: PAR Cost Table Summary



In the Cost Table Summary, the Timing Score reported is the sum of all the negative slacks related to setup timing requirements. Therefore, if the number reported is 0, it means that the timing-driven PAR process finished successfully without finding any setup timing issues. The hold timing score is reported by PAR only if Auto Hold-Time Correction is ON in the active Strategy. It can be found later in the PAR report, for example:

```
Hold time optimization iteration 0:
There are 6 hold time violations, the optimization is running
...
End of iteration 0
17 successful; 0 unrouted;  real time: 24 secs

Hold time optimization iteration 1:
There are 4 hold time violations, the optimization is running
...
End of iteration 1
17 successful; 0 unrouted;  real time: 24 secs
```

```
Hold time optimization completed
All hold time violations have been successfully corrected in
speed grade M
```

The PAR TRACE report includes timing scores for both setup and hold if both setup and hold are chosen in the Analysis Options (i.e. Standard Setup and Hold Analysis). This report includes the detailed timing analysis of the design against the constraints. It reports what constraints have been considered, whether they have been met, and the failing paths wherever a constraint has not been met. See the section "The MAP TRACE and PAR TRACE Reports" on page 22 for more info on the format of the TRACE report.

I/O Timing Analysis

If the PAR process finishes successfully, you can also run I/O Timing Analysis from the Process window. This enables you to examine the worst case I/O timing results across performance grades of the selected device for setup time and hold time and verify that your board timing complies. You can access the I/O Timing Analysis report in the Report View.

As explained in the section "Understanding TRACE" on page 20, by default, PAR TRACE uses the chosen performance grade of the target device for setup timing analysis, and it uses the "-m" performance grade, which is the virtual performance grade that represents the fastest (i.e., with minimum delay), for the hold timing analysis.

These default performance grades used are the "worst" case performance grades for register-to-register timing analysis. However, for FPGA I/Os, it is possible for "worst" case condition to be different. This depends on your design and the final placed and routed design. You should always run I/O Timing Analysis to sweep across performance grades and ensure that I/O timing is satisfied, even if the default PAR TRACE reports no timing error. (Note that you can also use Timing Analysis View to run PAR TRACE on all these different performance grades. Timing Analysis View allows you to do this without re-running PAR.)

For example, suppose that the chosen performance grade of the target device is "6", and you use the default performance grades to run PAR TRACE, i.e., grade "6" to do setup timing analysis, and grade "-m" to do hold timing analysis.

Assume PAR TRACE reports no error, so to ensure that I/O timing is satisfied, you run I/O Timing Analysis, and the I/O Timing Analysis produces the following report.

Part 1: The I/O Timing Analysis Report – Summary Section

Part 2: The I/O Timing Analysis Report – detail section for Performance grade M

```
......
I/O Timing Report (All units are in ns)

Worst Case Results across All Performance Grades (M, 9, 8, 7, 6, 6L, 7L, 8L):

// Input Setup and Hold Times

Port  Clock Edge  Setup Performance_Grade  Hold Performance_Grade
------------------------------------------------------------------------
data1 clk1  R     0.469      6       1.180     6
data2 clk2  R     0.596      6       1.087     M
rst   clk1  R     0.458      M       0.437     6
rst   clk2  R     0.514      6       0.245     6
......
```

```
// Input Setup/Hold Times (Performance Grade: M)

Port  Clock Edge    Setup     Hold
------------------------------------------------------------
data1 clk1  R       0.129     1.145
data2 clk2  R       0.183     1.087
rst   clk1  R       0.458     0.195
rst   clk2  R       0.512     0.137
......
```

In this example, from the summary section, the worst-case hold time minimum requirement of the port “data1” is 1.180 ns using performance grade “6” instead of the performance grade “-m”.

As explained previously, PAR TRACE actually uses the speed grade “-m” for the hold timing analysis. In this case, the hold time minimum requirement is 1.145 ns, as shown in the detail section for performance grade M, which is less than 1.180 ns.

When you actually ran PAR TRACE, if the hold time requirement of “data1” that was written in the LPF is less than 1.180 ns but greater than 1.145 ns, it actually reveals reveal an I/O timing problem in your design. If the hold time requirement written in the LPF is greater than 1.180 ns, then your design is fine.

The same situation applies to the setup timing analysis as well.

Timing Analysis View

Timing Analysis View is a “what-if” tool in Diamond that allows you to apply different hypothetical timing constraints to see how they would affect your design. These timing constraints are written to a flow-independent timing preference file (TPF file, with the extension .tpf). Timing Analysis View applies the TPF preferences to the PAR result, runs the static timing analysis, and reports the result for you to analyze.

Timing Analysis View has a simplified Spreadsheet View for creating and editing timing preferences in the TPF file. The view includes Path Tables, which enable you to select a timing preference and see the delay, slack, and other aspects of timing on particular paths. Timing Analysis View allows you to cross probe to Floorplan View or Physical View to see where these paths exist on the chip and what they look like.

To learn more about Timing Analysis View, refer to the Diamond online Help.

General Considerations and Practices for Timing Closure

Steps to Close Timing

There are a few general considerations and trade-offs you need to understand when working with the timing-driven flow using Diamond.

Let us start with some general recommendations:

1. Begin with the creation of meaningful and efficient HDL code. For information about coding techniques for FPGA designs, see the section “HDL Coding Guidelines.”

2. Along with the FPGA-friendly code, use the appropriate and sufficient timing constraints (preferences) to drive synthesis, MAP and PAR. A good set of FPGA timing requirements are crucial for meeting timing goals.
3. Run an initial design process including synthesis, MAP, MAP TRACE, PAR and PAR TRACE. If you have a high performance requirement, select timing-driven placement and specify a low placement effort level for this first PAR process through PAR strategy settings.

   Rule of Thumb: When a timing issue is reported by MAP TRACE, usually it is an RTL issue, and you should correlate the issue in your HDL code. You can save time by using the MAP TRACE report to fix these issues instead of trying to resolve them by needlessly running PAR and PAR TRACE.
4. Examine the MAP report, MAP TRACE report, PAR report, PAR TRACE and PAD report, and analyze the timing information.
5. If necessary, modify timing constraints and preferences. If applicable, assign primary and secondary clocks, tune I/O timing with PLLs, and group components along critical paths.
6. Run a second processing iteration. For PAR, change its strategy settings to use timing-driven placement, and then experiment with increased placement effort and multiple routing passes.
7. Analyze timing again, identifying high-fanout nets, critical path nets, and long delay paths, etc.
8. If necessary, do some floorplanning to direct the physical layout of the circuit. For designs that do not meet performance goals, use groups and regions to place components closer together and shorten routing distances. Use reiterative floor planning, repeating steps 6 through 8 until performance goals are achieved

Synthesis Timing Closure Techniques

General Considerations

1. Use timing-driven mode or not.

   The first decision you might need to make is whether to use timing-driven synthesis.

   Using timing-driven synthesis mode usually yields better performance, but it increases the resource usage at the same time. If your design has a high resource usage ratio in term of the available resource from the chosen device, the increased resource usage might have negative impact on the actual performance and can cause other problems such as congestion and long-path routing, which actually introduce timing problems.

   If your design runs at a low speed, or the timing requirements can be achieved easily, running synthesis in timing-driven mode might not be necessary. In this case, you simply run the synthesis in non-timing-driven

mode, examine the timing report, and proceed to the next process if the estimated performance meets your requirement.

2. Provide sufficient and appropriate timing constraints.

   To ensure that timing-driven synthesis works correctly, you must provide appropriate and sufficient timing constraints in the synthesis constraints file. The essential timing constraint is the clock period or frequency. If you do not provide clock requirements, by default 200MHz will be used for timing-driven synthesis. This can be seen and modified through the synthesis strategy, as explained in “Timing-Driven Synthesis and Constraints” on page 15. If you have multiple clocks, make sure that all of them are constrained with the appropriate values. Other timing constraints, such as setup time, clock to output, etc., should be provided if available.

3. Interpret the synthesis timing report.

   Since Synplify Pro does not have place-and-route information, its timing report is usually aggressive and inaccurate. You should use its timing report as a reference. Usually you can reduce the reported speed by one third to a half.

   On the other hand, LSE is more conservative, and the reported maximum frequency (Fmax) value is usually within 10% of the actual value from the placed and routed result.

4. Over-constrain or not

   There are some common practices suggesting that you should over-constrain the synthesis process in order to get a result with a better performance. This is not always the case, since over-constraining a design can unnecessarily increase the resource usage, and this might not be what you expect. A decision must be made to balance the performance and size of your design.

Using Dedicated GSR Resource for Fmax Improvement

If your design contains high fanout nets of set/reset, it is recommended that you use the dedicated hardwired GSR resource. This will result in less routing congestion and could improve route ability and performance. Otherwise, the design will use the resources of the local set/reset that could be used for other purposes.

MAP sees the whole design and is capable of seeing a large fanout reset net and implementing it on the GSR resource rather than the general routing resource. MAP will do this if the MAP strategy has the “infer GSR” option set to true (which is the default) and synthesis has not already inferred GSR.

Synthesis can also automatically infer GSR:

- When you use Synplify, the default setting is off, which is recommended. You should let MAP infer GSR for the best result.
- When you use the Lattice Synthesis Engine (LSE), the default setting is auto, which allows LSE to decide. This is also the recommended setting.

Using I/O Register to Improve I/O Timing

You can improve the input setup (tSU) and clock to output (tCO) timing by turning on the use of the I/O register. Turning on the input register can improve the input setup time. Turning on the output register can improve the clock to output time.

To use I/O register, there are several options:

- Use synthesis attributes and directives in the RTL code to control each individual port, or apply globally to all top level I/Os. This works for both Synplify Pro and LSE.

  For example, in Verilog:

  ```
  output [15:0] q; // synthesis syn_useioff = 1
  ```

  or

  ```
  module test (a, b, clk, rst, d) /* synthesis syn_useioff
  = 1 */;
  ```

  in VHDL:

  ```
  attribute syn_useioff : boolean;
  attribute syn_useioff of data_in : signal is true; --
  data_in is an I/O port
  ```

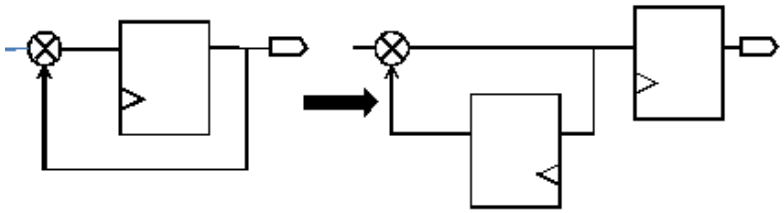
- If you use Synplify Pro, you can use synthesis constraints in the active Synplify Design Constraints file. You can control each individual port, or apply globally to all I/Os:

  ```
  define_attribute {z[3:0]} syn_useioff {1}
  define_global_attribute syn_useioff {1}
  ```

- If you use LSE, you can set the synthesis strategy option “Use IO Registers” to true. This will globally applies the option to all I/Os.

After turning on the use of the I/O register, ensure that the timing can still meet setup time and the Fmax requirements. Using the I/O register helps I/O timing, but it could potentially affect internal Fmax and cause an I/O hold time issue. There are some good cases where register duplication is used to help both I/O and Fmax; for example, the case of a counter with output going off chip, as illustrated in Figure 27.

Figure 27: Counter with Output Going Off Chip



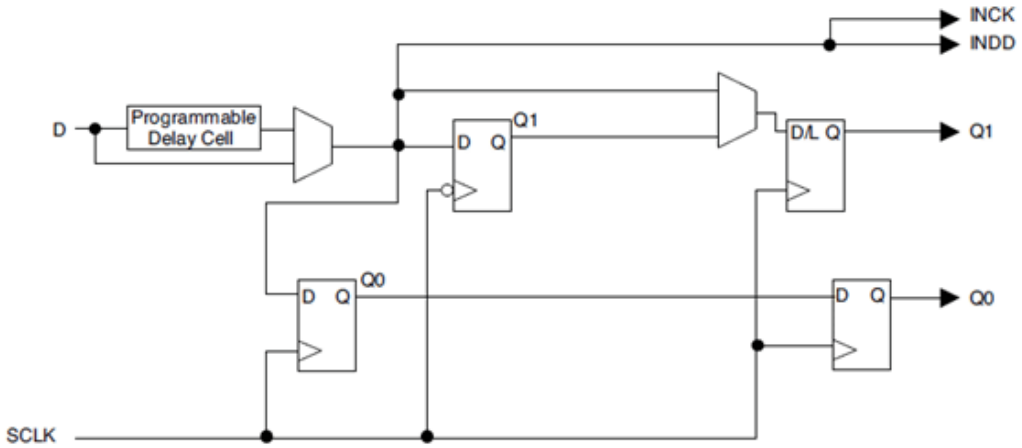
Note that not all FPGA devices facilitate I/O registers. Refer to the hardware datasheet of your target device.

Adding Delays to Input Registers

Designs that have registered inputs can incur hold-time violations if the data path is too fast. For devices that have an edge clock resource, such as ECP3 or XO2 devices, the use of the edge clock usually balances the data and clock to avoid hold time issues. The auto hold time correction after PAR might also help solve any hold time violations.

On the other hand, certain devices have a feature of programmable delay elements built in the silicon to give fixed or dynamic delays on the input registers. This reduces input register hold time requirements when a global clock (i.e., non-edge) is used. The diagram in Figure 28 shows an XO2 input register block diagram for PIOs on the left, top and bottom edges, which includes a programmable delay cell. Note that not all Lattice devices have this feature. Refer to the datasheet of your target device for details.

Figure 28: MachXO2 Input Register



To use this feature, you have the following options. Note that using these elements will compensate input hold time requirements by adding a specific amount of delay in the data input path. At the same time, it will affect the setup time with the same amount of delay value.

- Set the synthesis attribute “FIXEDDELAY” to “true” to add a fixed delay. This is supported by ECP/EC, ECP3, XO and XP2 devices. For example, in Verilog:

  ```
  input [15:0] data_in; // synthesis syn_useioff = 1
  FIXEDDELAY=true
  ```

  in VHDL:

  ```
  attribute syn_useioff : boolean;
  attribute syn_useioff of data_in : signal is true; --data_in
  is an I/O port
  attribute FIXEDDELAY of data_in: signal is “TRUE”;
  ```

  The delay value added depends on the device used. For example, for ECP3 speed grade -9 or -8, this value is 1.3ns. For other devices, refer to their datasheets.

- If you use ECP3 devices, you can instantiate a “DELAYC” element in the HDL to add a fixed delay. The amount of delay added is the same as using “FIXEDDELAY”. For example:

```
input b0;
wire bx, b_temp;
DELAYC myDelay1(.Z(b_temp), .A(b0));
IFS1P3IX b0_reg(.Q(bx), .SP(1'b1), .CD(rst), .SCLK(clk),
.D(b_temp));
```

- If you use ECP3, XP2 or ECP2/M devices, you can instantiate a ‘DELAYB” element in the HDL to add a user-specified amount of delay; for example:

```
input b0;
wire bx, b_temp;
DELAYB
myDelay(.Z(b_temp),.DEL3(1'b0),.DEL2(1'b0),.DEL1(1'b0),.DEL0
(1'b1),.A(b0));
IFS1P3IX b0_reg(.Q(bx), .SP(1'b1), .CD(rst), .SCLK(clk),
.D(b_temp[0]));
```

  The amount of delay value added is defined by the value of DEL[3:0]; This allows you to choose a delay from one of the 16 values. For ECP3 devices, the value increment is 35ps.

- If you use XO2 devices, you can instantiate either a “DELAYE” element (all sides) in the HDL to add a user-specified amount of delay, or a “DELAYD” element (bottom side) in the HDL to add a dynamic delay. For example:

```
component DELAYE
  generic(DEL_MODE: in String;
    DEL_VALUE: in String);
  port (A: in std_logic;
    Z : out std_logic);
end component;
......
inst1: DELAYE
  generic map (  DEL_MODE=> "SCLK_ZEROHOLD",
      DEL_VALUE=> "DELAY31")
  port map (A => IN1,
      Z => insig);
```

  The amount of delay added in this example is defined by “DEL_VALUE”. For details, refer to the XO2 datasheet.

Maximum Fanout Control for Fmax Improvement

Maximizing fanout is a technique of duplicating a driver. This allows less skew on a global signal, because it can be routed within a smaller area. This technique makes it easier to close timing and is usually good for non-clock signals such as clock enables.

You can use the maximum fanout attribute in your HDL code and selectively apply it to the critical path in order to reduce fanout. In most cases, registers are duplicated to reduce the maximum fanout, and it will increase the register count in the design.

Note that this attribute in the HDL code will override the global maximum fanout control. To use the attribute in your code, in Verilog:

```
input [31: 0] data_ in /* synthesis syn_ maxfan= 1000 */;
```

in VHDL:

```
attribute syn_maxfan : integer;
attribute syn_maxfan of data_in : signal is 1000;
```

Clock-Enable Control for Fmax Improvement

The clock enable net is typically a high fanout net driving several D flip-flops.

The placement and routing process uses the fanout to decide whether to implement the clock enable by using a secondary clock resource, which sometimes incurs a larger delay (approximately 3 ns). You can specify a constraint to avoid using the secondary clock.

If some clock enables are in the critical path, you can identify them in the HDL code and set the clock enable to off to avoid a delay. You can do this by setting the attribute “syn_useenables” to 0, as shown below, in Verilog:

```
reg [3: 0] q /* synthesis syn_useenables = 0 */;
always @( posedge clk)
if (enable)
q <=d;
```

in VHDL:

```
signal q_int : std_logic_vector( 3 downto 0);
Attribute syn_useenables : boolean;
attribute syn_useenables of q_int : signal is false;
process( clk)
begin
        if (clk'event and clk = '1') then
                if (enable = '1') then
                      q_int <= d;
                end if;
        end if;
end process;
```

Assigning Black Box Timing

If you instantiate a large embedded block like DSP or EBR, synthesis will treat the large block as a black box. The timing information is usually ignored, and sometimes a warning message will be displayed during synthesis.

If the large block is part of the critical path, you should use synthesis directives to assign timing delay properties to them so that the synthesis tool can apply the correct timing for the synthesis. A few Synplify Pro synthesis directives are:

- syn_isclock – specifies a clock port on a black-box
- syn_tpd<n> – timing propagation for combinational delay through the black box

- syn_tsu<n> – timing setup delay required for input pins relative to the clock
- syn_tco<n> – timing clock to output delay through the black-box

For example, if you use VHDL:

```
COMPONENT spr16x4a
PORT(
di0 : IN std_logic;
di1 : IN std_logic;
di2 : IN std_logic;
di3 : IN std_logic;
ck : IN std_logic;
wre : IN std_logic;
ad0 : IN std_logic;
ad1 : IN std_logic;
ad2 : IN std_logic;
ad3 : IN std_logic;
do0 : OUT std_logic;
do1 : OUT std_logic;
do2 : OUT std_logic;
do3 : OUT std_logic);
END COMPONENT;
attribute syn_tpd1 of rcf16x4z : component is "ado,ad1,ad2,ad3
-> do0,do1,do2,do3 = 1.1";
attribute syn_tsu1 of rcf16x4z : component is "ado,ad1,ad2,ad3
-> ck = 0.5";
attribute syn_tsu2 of rcf16x4z : component is "wre -> ck =
0.5";
```

If you use Verilog:

```
module SPR16X4A (DI0, DI1, DI2, DI3, AD0, AD1, AD2, AD3, WRE,
CK,DO0, DO1, DO2, DO3)
/* synthesis black_box syn_tpd1="AD0,AD1,AD2,AD3-
>DO0,DO1,DO1,DO3 =1.4" syn_tsu1="AD0,AD1,AD2,AD3->CK = 0.5"
syn_tsu2="WRE->CK = 0.5" */;
input AD0,AD1,AD2,AD3,DI0, DI1, DI2, DI3, CK, WRE;
output DO0, DO1, DO2, DO3;
```

Reviewing Synthesis Strategies

Synthesis strategies, which are synthesis optimization options, sometimes have great impact on the final timing result. When applying different synthesis strategies, you should examine the timing report to make sure that there is no negative timing impact on your design.

State Machine Encoding

One-hot state machine encoding is recommended for high-speed designs. However, using one-hot encoding increases resource usage and power consumption.

Resource Sharing

Resource sharing usually increases the number of logic levels, thus introducing additional delays to a path. Synthesis tools usually do a good job of resource sharing if the path is not critical, but this is not always the case. You should examine the critical paths to make sure that resource sharing does not cause any timing issues.

Pipeline and Retiming

Turning on this option allows synthesis tools to rebalance the timing by moving registers forward or backward through a path. Since synthesis tools do not have placement and routing information, the rebalancing is done based on the logic delays. Therefore, you should carefully examine the synthesis report to ensure that it has been done appropriately; otherwise, it ,might introduce timing problems after PAR.

MAP and PAR Timing Closure Techniques

General Strategy Guidelines

1. For timing closure purposes, you should first examine the results of the MAP TRACE report before continuing on to placement and routing. Considerations include the following:

   a. Warnings and errors related to invalid preferences

      Make sure that you correct them to avoid future confusion. (See Ensure a Clean LPF and Avoid Any Error in the *Design Planning* document.)

   b. Warnings, errors and potential design issues

      For example, a high number of logic levels might severely restrict design performance, and performance might benefit from a different partitioning or pipelining.

   c. Clock domain analysis

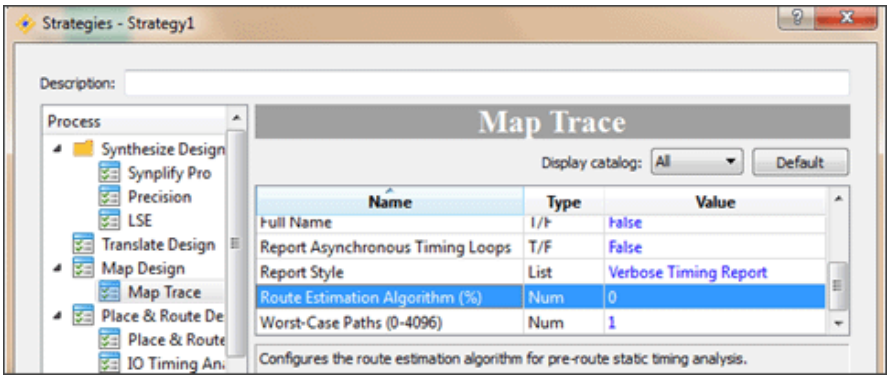
      Check the report and ensure that all clocks are constrained See “Understanding the PAR and PAR TRACE Reports” on page 32.

   d. Clock frequency

      Since no routing exists yet between logical connections, by default the MAP TRACE uses route delay estimation based on a suite of Lattice benchmark designs. You can overwrite the default behavior by specifying logic delay as a percentage of the overall path delay, where the total delay is the sum of logic and route delays. You can do this through the MAP TRACE strategy settings, as illustrated in Figure 29

   e. Logic depth

Figure 29: MAP TRACE Strategy Settings



Check the logic depth in the report and determine if HDL design changes are required. A typical design change example is pipelining, or registering, the data path. This technique might be the only way to achieve high internal frequencies if the design's logic levels are too deep.

2. Perform placement and routing early in the design phase, using a preliminary preference file, to gather information about the design.
3. Tune up your preference file to include all I/O and internal timing paths, as appropriate. Establish the pinout in the preference file. Check the preference coverage through the TRACE report and ensure that your design is fully covered by the timing requirement.
4. Push PAR, when necessary, by running multiple routing iterations and multiple placement iterations.
5. Revise the preference file as appropriate; use MULTICYCLE opportunities when possible.
6. Floorplan your design if necessary.

Use Preferences to Improve Timing

General Consideration

Providing appropriate and sufficient preferences is the key to a successful design. The following recommendations help you avoid over-constraining or under-constraining your design.

Under-Constraining

If a design is under-constrained compared to real system requirements, real timing issues not previously seen during dynamic timing simulations and static timing analysis could appear. These potential problems can be observed on a test board or during production.

Common causes of under-constrained timing preferences:

- No clock preference
- Unexpected data path between unrelated clock domains

- Undefined I/O specifications
- Asynchronous logic without MAXDELAY preferences
- Internally generated or unintentional clocks not specified in the preference file
- Critical paths blocked

These problems can usually be identified in the Clock Domain Analysis section in the TRACE report. See “Understanding the PAR and PAR TRACE Reports” on page 32.

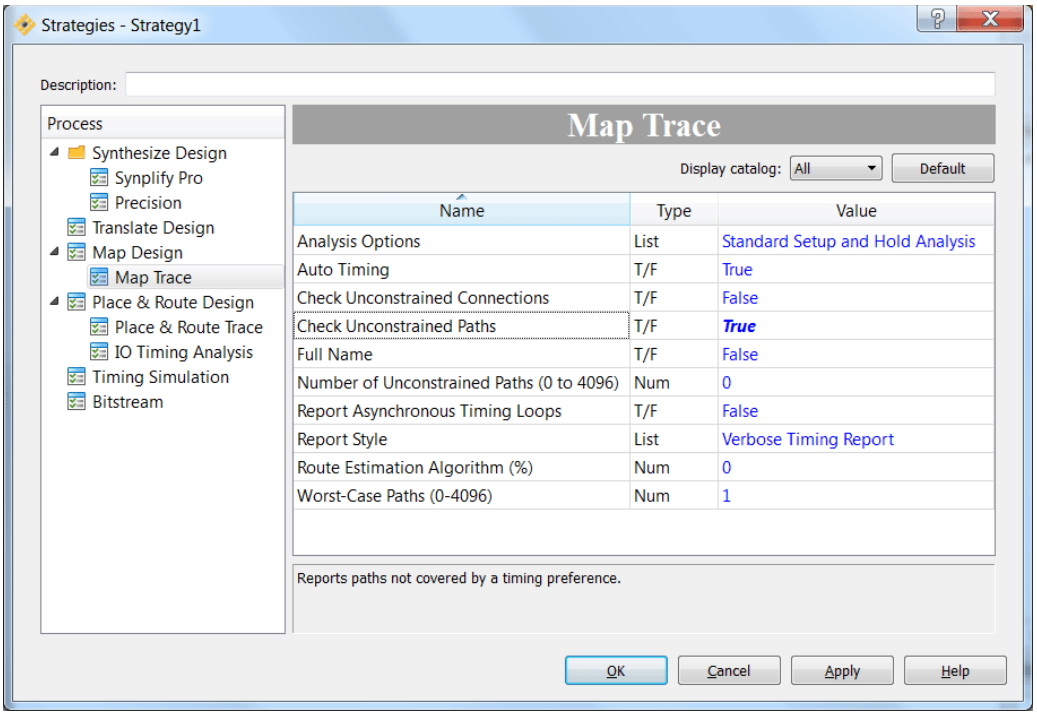
To make sure that no critical paths were left out because of under-constraining, you should check for preference coverage at the end of a TRACE report (.twr) file. An example of such an output is shown below:

```
Timing summary:
---------------
Timing errors: 4906 Score: 25326584
Constraints cover 36575 paths, 6 nets, and 8635 connections
(99.0% coverage)
```

This example shows 99.0% coverage.

To find unconstrained paths, enable the “Check Unconstrained Paths” option in MAP TRACE and PAR TRACE strategy settings, as shown in Figure 30.

Figure 30: Unconstrained Paths Strategy Option



This option gives a list of all of the signals that are not covered under timing analysis. In some designs, many of these signals are a common ground net that indeed does not need to be constrained. You should understand this and

use TRACE to check unconstrained paths and ensure that no timing-critical design paths are being missed.

Also, note the timing score shown in the example report. The timing score shows the total amount of negative slacks, in picoseconds, for all timing preferences constraining the design. Remember that PAR always attempts to minimize the timing score. PAR does not directly attempt to maximize frequency, but it indirectly tries to improve Fmax by reducing timing score. A higher timing score does not necessarily mean a larger gap to your system performance goals; for example, certain timing issues might be resolved by using simple fixes such as HDL modification, preference refinement or process strategy change, while a result with lower timing score might require more efforts to correct the timing problems.

Over-Constraining

If the constraints are tighter than the system requirements, the design will become over-constrained. This can actually lead to worse results as scarce resources are diverted away from their best use. In addition, this will increase core-processing runtime unnecessarily.

Common causes of over-constrained timing preferences include the following:

- Unspecified multi-cycle paths
- Multi-cycle paths to or from I/Os with different specifications
- FREQUENCY/PERIOD requirements that have been purposely set tighter than the actual device speed

It is always a good practice to constrain your design with the actual timing requirements. But sometimes you might want to experiment with over-constraining your design to determine your best constraint settings for achieving the desired results. In this case, instead of purposely over-constraining your design, you should use the PAR_ADJ option when you define your clock period or frequency. The PAR_ADJ keyword allows you to tighten requirements for PAR, but at the same time, preserve the requirements reported by TRACE.

Preferences and Processes

MAP and PAR processes require effective constraints in order to optimize the usage of resources. As explained in the section “Timing-Driven Flow Using Lattice Diamond Design Software” on page 13, for MAP and PAR, the design constraints, or preferences, are provided in an LPF file. You can set and edit design preferences at multiple points in the FPGA design flow.

For detailed information on creating preferences, and how they work in the design process, refer to “Preference Flow in Diamond” in the Design Planning chapter.

Writing Effective Timing Constraints

Understanding which preferences can be used to drive the timing-driven MAP and PAR is easy. Writing these preferences based on your design’s timing requirement also is not difficult, as long as you understand your design and its requirement.

However, creating appropriate timing preferences that can efficiently drive MAP and PAR requires that you fully understand how the timing-driven engine utilizes your constraints and applies them to the MAP and PAR processes and your design. Inappropriate timing preferences usually cause the timing-driven engine to be over-constrained, under-constrained, or both.

This section walks you through a simple example using a systematic approach, starting with no user-defined timing preferences at all and ending with all the timing requirements provided. It includes a few case studies to help you fully understand the timing-driven engine. You should go through and carefully examine all the cases in sequence to learn how to appropriately constrain your design and efficiently drive the timing-driven MAP and PAR processes.

Case Study 1 – No user-defined timing constraint

By default, when you start a new Diamond project and implementation, Diamond will automatically create an LPF file using the implementation's name. This file can be found in the File List view in the LPF Constraint Files section, and it is set as the active LPF file, which means that it will be used to drive the MAP and PAR processes. The file includes the following BLOCK preferences by default:

```
BLOCK RESETPATHS;
BLOCK ASYNCPATHS;
```

BLOCK RESETPATHS is a global preference that blocks all asynchronous set and reset paths that are through an asynchronous set and reset pin of your design.

BLOCK ASYNCPATHS is a global preference. If this preference is not in the LPF file, TRACE will analyze all input-to-register paths (that are not covered by an INPUT_SETUP preference) to see if they are longer than the period of the associated clock. The clock must have a PERIOD or FREQUENCY preference defined to get the period value. This is not very useful analysis, because few inputs will have the entire clock period from the device pin. We recommend that users define INPUT_SETUP preferences for all inputs that accurately reflect the actual board level timing. (See "Case study 4 - INPUT_SETUP" on page 56). The BLOCK ASYNCHPATHS preference is included in the LPF by default so that the less useful analysis is not included in the TRACE report.

If there are absolutely no timing constraints defined in your LPF or HDL (whether or not the two BLOCK preferences in the default LPF are present), then MAP TRACE will automatically generate a FREQUENCY preference for each of the identified clocks in your design. See "Case study 10 – Use PLL FREQUENCY Settings" on page 72 for an example of FREQUENCY defined in the HDL.

The calculation of the auto-generated FREQUENCY preference is based on the logic levels and the hardware recommended routing delay estimation algorithm for the target device. For different devices, the recommended routing delay estimation algorithms might be different. The estimated FREQUENCY number calculated is the fastest one that can be achieved based on your design's longest path.

The automatically generated FREQUENCY preferences are used to drive the timing-driven engine of the MAP and PAR process, and the reports from MAP TRACE, PAR and PAR TRACE summarize the timing requirements and the actual timing results.

To further understand this, examine the following example that uses the HDL code:

```
module example(clk1, clk2, data1, data2, rst, cout);
input clk1, clk2, data1, data2, rst;
output cout;
reg reg11, reg12, reg13;
reg reg21, reg22, reg23;

always @ (posedge clk1)
begin
  if (rst)
begin
  reg11<=1'b0;
  reg12<=1'b0;
  reg13<=1'b0;
end
  else
    begin
      reg11<=data1;
      reg12<=reg11;
      reg13<=reg22;
    end
end

always @ (posedge clk2)
begin
  if (rst)
begin
  reg21<=1'b0;
  reg22<=1'b0;
  reg23<=1'b0;
end
  else
    begin
      reg21<=data2;
      reg22<=reg21;
      reg23<=reg12;
    end
end

assign cout = reg13 & reg23;

endmodule
```

Note

The HDL code and timing constraint examples shown throughout all case studies are solely for explaining the timing constraint concept and are not to be considered as recommended HDL coding practice.

In this example, there are two external, unrelated clocks: “clk1” and “clk2.” This can be examined in the Clock Domains Analysis section of the MAP TRACE or PAR TRACE report:

```
Clock Domains Analysis
------------------------
Found 2 clocks:

Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY NET "clk1_c" 1349.528000 MHz ;

   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Not reported because source and destination domains are unrelated.
      To report these transfers please refer to preference CLKSKEWDIFF to define
      external clock skew between clock ports.

Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   Covered under: FREQUENCY NET "clk2_c" 1349.528000 MHz ;

   Data transfers from:
   Clock Domain: clk1_c   Source: clk1.PAD
      Not reported because source and destination domains are unrelated.
      To report these transfers please refer to preference CLKSKEWDIFF to define
      external clock skew between clock ports.
```

If there is no FREQUENCY or PERIOD preference defined in your LPF or HDL, estimated FREQUENCY preferences will be automatically generated for both “clk1” and “clk2”, and these will be used to drive MAP and PAR. This can be examined in the Clock Domains Analysis section of the TRACE report as shown above, as well as the Preference Summary section in the TRACE reports:

- MAP TRACE report:

```
Preference Summary
• FREQUENCY NET "clk1_c" 1349.528000 MHz (1 errors)
            1 item scored, 1 timing error detected.
Warning: 895.255MHz is the maximum frequency for this
preference.

• FREQUENCY NET "clk2_c" 1349.528000 MHz (1 errors)
            1 item scored, 1 timing error detected.
Warning: 895.255MHz is the maximum frequency for this
preference.

Report Type:     based on TRACE automatically generated
preferences
BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

- PAR TRACE report:

```
Preference Summary
• FREQUENCY NET "clk1_c" 1349.528000 MHz (1 errors)
```

```
            1 item scored, 1 timing error detected.
Warning: 771.010MHz is the maximum frequency for this
preference.
• FREQUENCY NET "clk2_c" 1349.528000 MHz (1 errors)
            1 item scored, 1 timing error detected.
Warning: 955.110MHz is the maximum frequency for this
preference.
Report Type:     based on TRACE automatically generated
preferences
BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

The text in red illustrates the calculated FREQUENCY preferences for both clocks. These preferences are used to drive the MAP and PAR process, and they are also used for the MAP TRACE and PAR TRACE static timing analysis. The “Report Type” highlighted in blue clearly states that the preferences are generated automatically by TRACE.

The warning messages in blue are the actual maximum speed of your design for each clock domain, based on the calculated FREQUENCY preferences.

Also noticeable in this particular example is that, from the HDL code, we know that “clk1” and “clk2” are unrelated. This fact is further proved in the Clock Domain Analysis section. TRACE will not analyze cross-domain paths driven by unrelated clocks, because it cannot determine the relationship between them. This might make your design under-constrained. To relate two clocks, use CLKSKEWDIFF See “Case Study 6 – CLKSKEWDIFF” on page 60.

From the TRACE report, it can also be seen that since there are no user-defined timing constraints in the LPF or the HDL, the two default BLOCK preferences are used, whether or not the two BLOCK preferences are present in the default LPF file.

What is Learned from Case Study 1

From this case study, the following points are learned:

- To avoid over-constraining the engine with the auto-generated FREQUENCY preferences, you should specify a FREQUENCY preference for each clock in you design, based on your design’s requirement.
- On the other hand, you can use this case as an experimental process to estimate how fast or slow you design can run:
  - If the actual maximum frequency reported by PAR TRACE is higher than what your design requires, you can use your actual FREQUENCY requirement to relax the engine to easily achieve your goal. At the same time, this might result in less resource usage
  - If the calculated or the actual FREQENCY is lower than what you design requires, you might need to examine your code for coding or architect improvement.
- If there are no user-defined timing preferences in the LPF or the HDL, the two default BLOCK preferences will be used, whether or not they are present in the active LPF file.

Case Study 2 – Insufficient FREQUENCY preference

Using the same HDL code used in "Case Study 1 – No user-defined timing constraint" on page 47, we now include a FREQUENCY preference, but only for one of the two clocks; for example, "clk1":

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
```

Here clock "clk1" is constrained, but clock "clk2" is not.

Since only one of the clocks is constrained, the Clock Domains Analysis section of the TRACE report now shows different information compared to that of "Case Study 1 – No user-defined timing constraint" on page 47:

```
Clock Domains Analysis
------------------------
Found 2 clocks:

Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk1" 300.000000 MHz ;

   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Not reported because source and destination domains
are unrelated.
      To report these transfers please refer to preference
CLKSKEWDIFF to define
      external clock skew between clock ports.

Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   No transfer within this clock domain is found
```

From this report, we can see that clock "clk1" is now constrained but clock "clk2" is not. This reveals the fact—an important engine behavior—that if your design has more than one clock, and if only some but not all of them are constrained, the engine will not automatically calculate and generate FREQUENCY preferences for those clocks that you did not constrain.

This fact can also be observed in the Preference Summary section of the MAP TRACE and PAR TRACE reports:

- MAP TRACE report:

```
Preference Summary
•  FREQUENCY PORT "clk1" 300.000000 MHz (0 errors)
1 item scored, 0 timing errors detected.
Report:  895.255MHz is the maximum frequency for this
preference.

BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

- PAR TRACE report:

```
Preference Summary
•  FREQUENCY PORT "clk1" 300.000000 MHz (0 errors)
1 item scored, 0 timing errors detected.
```

```
Report:  765.697MHz is the maximum frequency for this
preference.

BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

Both the MAP TRACE report and the PAR TRACE report clearly show that only one clock FREQUENCY preference is defined. Note that the MAP TRACE report does not have “Report Type” that was shown in “Case Study 1 – No user-defined timing constraint” on page 47. This user-defined preference is the only FREQUENCY requirement driving the engine, and there is no automatically generated FREQUENCY preference for clock “clk2”; thus “clk2” is not constrained now.

Upon further examination of either the MAP TRACE report or the PAR TRACE report, the low percentage of the preference coverage (17.6%) should imply the problem as well:

```
Timing summary (Setup):
---------------
Timing errors: 0  Score: 0
Cumulative negative slack: 0
Constraints cover 1 paths, 1 nets, and 3 connections (17.6%
coverage)
------------------------------------------------------------
```

There is one more fact you should be aware of: if there is a FREQUENCY preference defined in your LPF, then including or excluding the two default BLOCK preferences will be different. Suppose that we now have the following LPF preferences:

```
#BLOCK RESETPATHS ;
#BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
```

Note that the two BLOCK preferences are commented out and will not be in effect. Now look at the Preference Summary section in the PAR TRACE report:

```
Preference Summary
•  FREQUENCY PORT "clk1" 300.000000 MHz (0 errors)
4 items scored, 0 timing errors detected.
Report:  457.666MHz is the maximum frequency for this
preference.
------------------------------------------------------------
```

It is clear that the two BLOCK preferences are not shown in the summary. Interestingly, you might also notice that the maximum frequency (457.666MHz) is different now from the previous one (765.697MHz) where the two BLOCK preferences were used. Because “BLOCK ASYNCPATHS” is not present in the latter case, TRACE will analyze input-to-register paths that are covered by a FREQUENCY or a PERIOD preference but not covered by an INPUT_SETUP preference, and the input-to-register paths’ timing requirements will be calculated automatically and used to drive the engine. The calculated value usually equals a clock cycle defined by the FREQUENCY preference. Most of the time, apparently, this value will under-

constrain the engine. We will look into this in “Case study 4 - INPUT_SETUP” on page 56.

What is Learned from Case Study 2

From this case study, the following points are learned:

- You should have all clocks in your design appropriately constrained, either through your HDL or through FREQUENCY preferences defined in the LPF. Otherwise, your design is under-constrained, and you might miss many timing problems in your design.

  The TRACE reports are helpful for finding any unconstrained clocks:

  - The Clock Domains Analysis section should list the total number of clocks identified in your design.
  - The Preference Summary lists all clocks that had been constrained.
  - In addition, the preference coverage reported in the Timing Summary section should help explain whether your design is under-constrained.
- Since PAR TRACE is generated after PAR, which usually takes more runtime, you should carefully examine the MAP TRACE report and correct as many issues as possible before running PAR.
- When there is a valid FREQUENCY or PERIOD preference defined in the LPF, TRACE will behave differently if you include or exclude two default BLOCK preferences. This behavior is different from that shown in “Case Study 1 – No user-defined timing constraint” on page 47, and excluding the two BLOCK preferences might over-constrain your design.

Case study 3 – Sufficient FREQUENCY preference

Using the same HDL code in “Case Study 1 – No user-defined timing constraint” on page 47, we now include FREQUENCY preferences for both clocks:

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
FREQUENCY PORT "clk2" 350.000000 MHz ;
```

Now let us examine the reports from MAP TRACE, PAR, and PAR TRACE.

- Clock domains analysis:

```
Clock Domains Analysis
------------------------
Found 2 clocks:

Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk1" 300.000000 MHz ;

   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Not reported because source and destination domains
are unrelated.
      To report these transfers please refer to preference
CLKSKEWDIFF to define
      external clock skew between clock ports.
```

```
Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk2" 350.000000 MHz ;

   Data transfers from:
   Clock Domain: clk1_c   Source: clk1.PAD
      Not reported because source and destination domains
are unrelated.
      To report these transfers please refer to preference
CLKSKEWDIFF to define
      external clock skew between clock ports.
```

- MAP TRACE report:

```
Preference Summary
• FREQUENCY PORT "clk1" 300.000000 MHz (0 errors)
            1 item scored, 0 timing errors detected.
Report:  895.255MHz is the maximum frequency for this
preference.

• FREQUENCY PORT "clk2" 350.000000 MHz (0 errors)
            1 item scored, 0 timing errors detected.
Report:  895.255MHz is the maximum frequency for this
preference.

BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

- PAR TRACE report:

```
Preference Summary
• FREQUENCY PORT "clk1" 300.000000 MHz (0 errors)
            1 item scored, 0 timing errors detected.
Report:  667.557MHz is the maximum frequency for this
preference.

• FREQUENCY PORT "clk2" 350.000000 MHz (0 errors)
            1 item scored, 0 timing errors detected.
Report:  919.118MHz is the maximum frequency for this
preference.

BLOCK ASYNCPATHS
BLOCK RESETPATHS
------------------------------------------------------------
```

Clock Domains Analysis now reports that there are two clocks and that both of them were constrained. This is also confirmed in the Preference Summary of the MAP TRACE report and the PAR TRACE report. Furthermore, the percentage of the preference coverage is doubled compared with that of “Case Study 2 – Insufficient FREQUENCY preference” on page 51, as shown below:

```
Timing summary (Setup):
---------------
Timing errors: 0  Score: 0
Cumulative negative slack: 0

Constraints cover 2 paths, 2 nets, and 6 connections (35.3%
coverage)
------------------------------------------------------------
```

However, the 35.3% coverage is still poor, and apparently the design is still under-constrained and the constraints need to be improved. We will cover that in later case studies.

Similar to “Case Study 2 – Insufficient FREQUENCY preference” on page 51, if we remove the default BLOCK preferences in the active LFP, we have the following:

```
#BLOCK RESETPATHS ;
#BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
FREQUENCY PORT "clk2" 350.000000 MHz ;
```

The input-to-register paths covered by both “clk1” and “clk2” will be analyzed by TRACE using the automatically calculated timing requirement, meaning one clock cycle. In this example, the input setup timing requirement for all input-to-register paths in clock domain “clk1” is 3.333ns, based on the “clk1” 300MHz FREQUENCY preference. The input setup timing requirement for all input-to-register paths in clock domain “clk2” is 2.857ns, based on the “clk2” 350MHz FREQUENCY preference. This apparently increases the preference coverage from 35.3% to 70.6%, as reported in the Timing Summary section:

```
Timing summary (Setup):
---------------
Timing errors: 0  Score: 0
Cumulative negative slack: 0

Constraints cover 8 paths, 2 nets, and 12 connections (70.6%
coverage)
--------------------------------------------------------------
```

However, a one-clock-cycle input setup timing requirement is usually not realistic, according to your PCB board-timing requirement, and it might have your design under-constrained. You should use an appropriate INPUT_SETUP preference to constrain all inputs of your design. This will be covered in “Case study 4 - INPUT_SETUP” on page 56.

To further define proper timing preferences, you should understand another engine behavior: if you only have your design’s clocks constrained, either in the HDL or through a FREQUENCY or PERIOD preference in the LPF, the following paths will not be covered and your design might be under-constrained:

- Register to output paths
- Input to output paths
- Multi-cycle paths
- False paths
- Cross-domain paths that are between unrelated clocks

  This can be observed in the Clock Domains Analysis section of the TRACE reports.

Cross-domain paths that are between related clocks will be covered, though. Related clocks are those clocks whose relationships the engine is able to

determine. Examples are those clocks internally generated, such as the following:

- A derived clock from a clock divider
- Clocks generated from PLLs

Not all internally generated clocks can be explicitly related. One example of this is a gated clock.

If your design includes cross-domain paths that are between unrelated clocks, you should establish the relationship between the clocks; otherwise, your design will be under-constrained. This will be covered in “Case Study 6 – CLKSKEWDIFF” on page 60.

What is Learned from Case Study 3

From this case study, the following points are learned:

- Each clock in your design should be appropriately constrained in either the HDL or the LPF file, no matter whether the clock is an external one or one that is internally generated.
- The engine is capable of determining certain types of related clocks, such as a derived clock from a clock divider or a PLL. For some other internally generated clocks, you need be very careful to ensure that they are constrained and that the relationships are established. The Clock Domains Analysis section in the TRACE report should help identify all clocks in your design.
- If timing preferences only cover clocks, meaning if only FREQUENCY or PERIOD are defined, the engine will not be able to cover many types of paths and the design will be very under-constrained.
- The two default BLOCK preferences should always be included in your active LPF file to avoid less useful analysis by TRACE

Case study 4 - INPUT_SETUP

In previous case studies, we explained an important engine behavior: that input-setup time will automatically be calculated from a PERIOD or FREQUENCY preference if no INPUT_SETUP preference is defined and if BLOCK ASYNCPATHS is not present in the LPF. This will have your design under-constrained. To specify the accurate requirements, you should use the INPUT_SETUP preference with the appropriate values. The value of INPUT_SETUP should be calculated based on your PCB timing requirement. See “Example: Calculate Timing Requirement” on page 12.

Using the same design in the previous case studies, the following preferences are defined:

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
FREQUENCY PORT "clk2" 350.000000 MHz ;
INPUT_SETUP PORT "data1" 2.000000 ns CLKPORT "clk1" ;
INPUT_SETUP PORT "data2" 1.500000 ns CLKPORT "clk2"
CLK_OFFSET 1.500000 X ;
```

Now the input “data1” has a 2ns INPUT_SETUP requirement instead of 3.333ns, which would otherwise be calculated from the 300MHz FREQUENCY requirement of the reference clock net “clk1.” The reference clock net is defined through CLKPORT in the INPUT_SETUP preference, as shown in the example.

Similarly, the input “data2” has a 1.5ns INPUT_SETUP requirement instead of 2.857ns, which is one clock cycle defined by 350MHz FREQUENCY of the reference clock “clk2”. In addition, CLK_OFFSET is defined along with the INPUT_SETUP. CLK_OFFSET adjusts the timing analysis by a multiple factor of the clock period. In this example, the factor is 1.5, so the input-setup-time requirement for “data2” is as follows:

```
1.5ns + PERIOD * factor = 1.5 + 2.857 * 1.5 = 5.785ns
```

This can be observed in the TRACE report:

```
================================================================================
Preference: INPUT_SETUP PORT "data2" 1.500000 ns CLKPORT "clk2" CLK_OFFSET 1.500000 X
;
            1 item scored, 0 timing errors detected.
--------------------------------------------------------------------------------
Passed:  The following path meets requirements by 5.197ns
 Logical Details:  Cell type  Pin type       Cell/ASIC name  (clock net +/-)
   Source:         Port       Pad            data2
   Destination:    FF         Data in        reg21_0io  (to clk2_c +)
   Max Data Path Delay:     0.508ns  (100.0% logic, 0.0% route), 1 logic levels.
   Min Clock Path Delay:    1.213ns  (37.7% logic, 62.3% route), 1 logic levels.
IOL_L27A attributes: FINE=FDEL0
 Constraint Details:
      0.508ns delay data2 to data2_MGIOL less
      5.785ns offset data2 to clk2 (totaling -5.277ns) meets
      1.213ns delay clk2 to data2_MGIOL less
      1.293ns DI_SET requirement (totaling -0.080ns) by 5.197ns
Physical Path Details:
......
```

The CLK_OFFSET factor can be a floating-point number, and it might be useful to pick the opposite clock edge for the analysis by defining the multiplier factor as 0.5.

The preference coverage is also increased and can be observed from the Timing Summary:

```
Timing summary (Setup):
---------------
Timing errors: 0  Score: 0
Cumulative negative slack: 0
Constraints cover 4 paths, 2 nets, and 8 connections (47.1% coverage)
--------------------------------------------------------------
```

Instead of 35.3% percentage of the coverage shown in the previous case study, now it is 47.1%. However, we still need to improve it.

What is Learned from Case Study 4

From this case study, the following points are learned:

- Accurate INPUT_SETUP needs to be defined according to the design's timing requirement. Otherwise, the input setup time will be calculated automatically from FREQUENCY (if BLOCK ASYNCPATHS is not defined), or INPUT_SETUP will not be analyzed (if BLOCK ASYNCPATHS is defined). In both cases, your design is under-constrained
- You can use additional options to define INPUT_SETUP, such as CLK_OFFSET. For detailed information, refer to the *Constraints Reference Guide* in the Diamond online Help

Case Study 5 - CLOCK_TO_OUT

In "Case study 3 – Sufficient FREQUENCY preference" on page 53, we explained an important engine behavior: that register-to-output paths and some other types of paths will not be covered if only PERIOD or FREQUENCY preferences are defined. This can cause your design to be under-constrained. To specify the accurate requirements, you should use the CLOCK_TO_OUT preference. The value of CLOCK_TO_OUT should come from your PCB timing requirement. See "Example: Calculate Timing Requirement" on page 12.

Using the same example in the previous case studies, we add the CLOCK_TO_OUT preference:

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 300.000000 MHz ;
FREQUENCY PORT "clk2" 350.000000 MHz ;
INPUT_SETUP PORT "data1" 2.000000 ns CLKPORT "clk1" ;
INPUT_SETUP PORT "data2" 1.500000 ns CLKPORT "clk2"
CLK_OFFSET 1.500000 X ;
CLOCK_TO_OUT PORT "cout" 1.000000 ns CLKPORT "clk1" ;
```

The CLOCK_TO_OUT preference constrains the clock-to-output timing requirement of the output "cout" to be 1ns, referencing the clock "clk1." This is illustrated in the diagram in "Clock to Output" on page 7.

When all register-to-output paths are constrained, the preference coverage is increased. This can be observed in the TRACE report:

```
Timing summary (Setup):
---------------
Timing errors: 1  Score: 4298
Cumulative negative slack: 4298
Constraints cover 5 paths, 2 nets, and 10 connections (58.8%
coverage)
--------------------------------------------------------------
```

Now we have 58.8% coverage, compared with 47.1% in the previous case study.

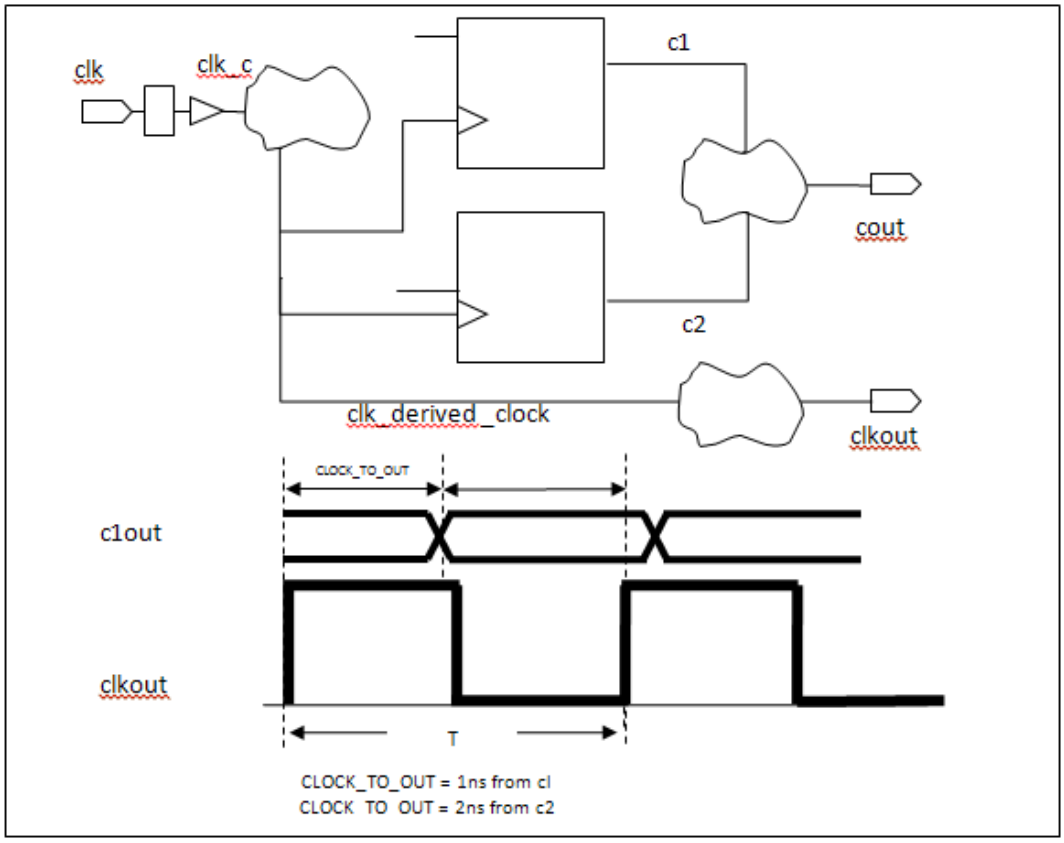
When defining the CLOCK_TO_OUT preference, you can use a clock output, if your design has one, as the reference clock. For example:

```
CLOCK_TO_OUT "cout" 1.0 ns CLKPORT "clk" CLKOUT PORT
"clkout";
CLOCK_TO_OUT "cout" 2.0 ns CLKPORT "clk" FROM "c2" CLKOUT
PORT "clkout";
```

This first CLOCK_TO_OUT preference constrains the clock-to-output timing requirement of all the output paths to “cout” to be 1ns, referencing the clock output “clkout,” where “clkout” is an output driven by the derived clock net “clk_derived_clock.” However, the second preference is more specific than the first one and requires that the clock-to-output timing requirement of the path “c2” to “cout” to be 2ns, referencing the clock output “clkout”.

These preferences and the relationship of the output and the reference clock are illustrated in Figure 31.

Figure 31: Output and Reference Clock



It is important to notice that the second CLOCK_TO_OUT preference is more specific than the first CLOCK_TO_OUT preference. The path driven by “c1” has a 1ns requirement, and the path driven by “c2” has a 2ns requirement; otherwise, both the paths would have 1ns requirement.

What is Learned from Case Study 5

From this case study, the following is learned:

- Accurate and sufficient CLOCK_TO_OUT preferences need to be defined according to the PCB timing requirement. Otherwise, the register-to-

output paths will not be covered, and this can cause your design to be under-constrained.

For detailed information on the CLOCK_TO_OUT preference and available options, refer to the *Constraints Reference Guide* in the Diamond online Help.

Case Study 6 – CLKSKEWDIFF

Sufficient FREQUENCY, INPUT_SETUP and CLOCK_TO_OUT preferences should cover most of the paths in a simple design, especially those designs that only have one clock domain or that have multiple clock domains with no cross-domain paths.

More often, your design will have many paths crossing multiple clock domains, where multiple clock domains have one or both of the following two types:

- Clock domains that are related – For example, when you use a clock divider, PLLs, and certain types of derived clocks, the engine is able to determine the relationship between clock domains.
- Clock domains that are unrelated – for example, when your designs have multiple top-level clock inputs

The example used in the previous case studies shows a typical design that includes unrelated clock domains with paths crossing them. Since the engine is unable to determine the relationship between clock domains, the paths across these domains will not be analyzed; therefore, your design is under-constrained. This fact can be observed in the Clock Domains Analysis section in the TRACE reports:

```
Clock Domains Analysis
------------------------
Found 2 clocks:

Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk1" 300.000000 MHz ;

   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Not reported because source and destination domains are unrelated.
      To report these transfers please refer to preference CLKSKEWDIFF to define
      external clock skew between clock ports.

Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk2" 350.000000 MHz ;

   Data transfers from:
   Clock Domain: clk1_c   Source: clk1.PAD
      Not reported because source and destination domains are unrelated.
      To report these transfers please refer to preference CLKSKEWDIFF to define
      external clock skew between clock ports.
```

The low percentage of the preference coverage reported by TRACE also reveals that there are many paths not covered.

As suggested in the Clock Domains Analysis, you should establish the relationship between the two clock domains using the CLKSKEWDIFF preference. For example:

```
CLKSKEWDIFF CLKPORT "clk1” CLKPORT "clk2" 0.5 ns;
```

This preference informs the engine that “clk1” arrives at the clock input later than “clk2” by 0.5ns.

Now that the relationship between “clk1” and “clk2” is established, the engine will cover the paths crossing these two clock domains, as shown in the Clock Domain Analysis of the TRACE reports:

```
Clock Domains Analysis
------------------------
Found 2 clocks:

Clock Domain: clk1_c   Source: clk1.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk1" 300.000000 MHz ;

   Data transfers from:
   Clock Domain: clk2_c   Source: clk2.PAD
      Covered under: FREQUENCY PORT "clk1" 300.000000 MHz ;   Transfers: 1

Clock Domain: clk2_c   Source: clk2.PAD   Loads: 2
   Covered under: FREQUENCY PORT "clk2" 350.000000 MHz ;

   Data transfers from:
   Clock Domain: clk1_c   Source: clk1.PAD
      Covered under: FREQUENCY PORT "clk2" 350.000000 MHz ;   Transfers: 1
--------------------------------------------------------------
```

Subsequently, the preference coverage percentage is increased as well:

```
Timing summary (Setup):
---------------
Timing errors: 2  Score: 9839
Cumulative negative slack: 9839

Constraints cover 7 paths, 2 nets, and 12 connections (70.6%
coverage)
--------------------------------------------------------------
```

Remember that CLKSKEWDIFF only applies to the top-level clocks, so you need “CLKPORT” to identify a top-level clock and use the clock port name instead of the clock net name. If the clock defined does not exist in your design, the preference will be ignored. You should examine the report to see if this is the case.

For detailed information about the CLKSKEWDIFF preference, refer to the section *Constraints Reference Guide* in the Diamond online Help.

CLKSKEWDISABLE

When calculating the slacks of paths, including those paths between the same clock domain or those paths between cross-domains of the related clocks, clock skews are also taken into account, as shown in the following TRACE report:

```
================================================================================
Preference: FREQUENCY PORT "clk1" 500.000000 MHz ;
            1 item scored, 0 timing errors detected.
--------------------------------------------------------------------------------
Passed: The following path meets requirements by 0.701ns
 Logical Details:  Cell type  Pin type       Cell/ASIC name  (clock net +/-)
   Source:         FF         Q              reg11_0io  (from clk1_c +)
   Destination:    FF         Data in        reg12  (to clk1_c +)
   Delay:               1.047ns  (19.2% logic, 80.8% route), 1 logic levels.
 Constraint Details:
      1.047ns physical path delay data1_MGIOL to SLICE_0 meets
      2.000ns delay constraint less
      0.099ns skew and
      0.153ns M_SET requirement (totaling 1.748ns) by 0.701ns
 Physical Path Details:
      Data path data1_MGIOL to SLICE_0:
   Name    Fanout   Delay (ns)          Site               Resource
C2OUT_DEL   ---     0.201   IOL_L26A.CLK to   IOL_L26A.INB data1_MGIOL (from clk1_c)
ROUTE         1     0.846   IOL_L26A.INB to      R27C2C.M0 reg11 (to clk1_c)
                  --------
                    1.047   (19.2% logic, 80.8% route), 1 logic levels.
 Clock Skew Details:
      Source Clock Path clk1 to data1_MGIOL:
   Name    Fanout   Delay (ns)          Site               Resource
ROUTE         2     1.183       K3.PADDI to   IOL_L26A.CLK clk1_c
                  --------
                    1.183   (0.0% logic, 100.0% route), 0 logic levels.
      Destination Clock Path clk1 to SLICE_0:
   Name    Fanout   Delay (ns)          Site               Resource
ROUTE         2     1.084       K3.PADDI to     R27C2C.CLK clk1_c
                  --------
                    1.084   (0.0% logic, 100.0% route), 0 logic levels.

Report:  769.823MHz is the maximum frequency for this preference.
```

"0.099" clock skew is calculated by:

```
<source clock delay> - <destination clock delay> = 1.183 -
1.084 = 0.099ns
```

For cross-domain paths, you can use the CLKSKEWDISABLE preference to explicitly exclude clock skews from the slack calculation, as shown in the following example:

```
CLKSKEWDISABLE CLKNET "clk1_c" CLKNET "clk2_c";
```

If two clocks are related, this preference excludes clock skews from the slack calculation when scoring cross-domain paths from the "clk1_c" domain to the "clk2_c" domain.

Where two clocks are unrelated, this preference also establishes the relationship from the source domain “clk1” to the destination domain “clk2.” This is similar to the CLKSKEWDIFF preference, which also establishes the relationship between two clock domains. The difference is that CLKSKEWDIFF establishes the relationship in both domain-to-domain directions, while CLKSKEWDISABLE only establish the relationship in one direction. So to build the cross-domain relationship from the “clk2” domain to “clk1” domain, another CLKSKEWDISABLE preference can be used:

```
CLKSKEWDISABLE CLKNET "clk2_c" CLKNET "clk1_c";
```

Note

Clock skews for data paths that have the same source and destination clock nets cannot be disabled by using this preference.

What is Learned from Case Study 6

From this case study, the following points are learned:

- If your design has multiple clock domains, you should carefully examine your design and the TRACE reports to see if there are cross-domain paths and to ensure that cross-domain paths are covered by the engine.
- When your design has multiple top-level clocks, they are usually unrelated. The relationship between unrelated clocks must be established by using the CLKSKEWDIFF preference. Otherwise, cross-domain paths will not be analyzed and your design might be under-constrained.
- Clock skews are usually included and considered by the engine if—and only if—clocks are related

Case Study 7 – Timing Exception 1 – MULTICYCLE

Timing Exceptions

Timing exceptions are preferences that describe the special behavior of certain design paths. Most designs contain paths that require these additional preferences to relax the default timing constraint used by the engine. Without timing exceptions, the static timing analysis performed by TRACE will likely assume worst-case timing scenarios and report lower design performance, and PAR will spend an undue amount of effort optimizing the path. With timing exception preferences that can represent the actual design behavior, the engine will be relaxed.

Two common path types require timing exceptions: multi-cycle paths and false paths. We will discuss multi-cycle in this case study.

Multi-Cycle Paths

In most synchronous circuits, the receiving register captures data launched by a launching register that uses the next active clock edge of the receiving register. This behavior, which is single-cycle behavior, is assumed as the default behavior by TRACE and PAR, and the timing constraint is calculated and used as the default by TRACE and PAR.

- If the launching register and the receiving register of a path use the same clock, then the default timing constraint is one clock cycle. See Multi-Cycle Within the Same Clock Domain.
- If the launching register and the receiving register of a path use two different clocks and these two clocks are related, then the default timing constraint is the worst-case edge pair, which depends on the period of the two clocks. See Multi-Cycle Across Clock Domains.

A multi-cycle path refers to cases where this relationship is different. Since single-cycle behavior is assumed by PAR and TRACE, a multi-cycle type of preference, MULTICYCLE, is used to express the relationship and relax the engine.

MULTICYLE preferences only apply to paths that are within a clock domain or across clock domains that are related. A clock domain is established by defining a FREQUENCY or PERIOD preference. MULTICYLE preferences are used to relax timing requirements on those paths.

Multi-Cycle Within the Same Clock Domain

If the launching and the receiving registers of a path use the same clock, this path is said to be in the same clock domain or to be transferred within the same clock domain.

Since the launching and the receiving registers use the same clock, the default timing constraint is one clock cycle of the clock, which means that the path from the launching register to the receiving register requires one clock cycle. This will be used as the default by TRACE and PAR.

If the default is not appropriate for the path, then you must use the MULTICYCLE preference to change the default timing constraint and relax the engine.

The example in Figure 32 illustrates a single-cycle versus a multi-cycle relationship within the same clock domain. In this example, both the Launching Register and the Receiving Register use the same clock “clk.” By default, the timing requirement for the path between two registers is one clock cycle, and the MULTICYCLE preference will change this behavior.

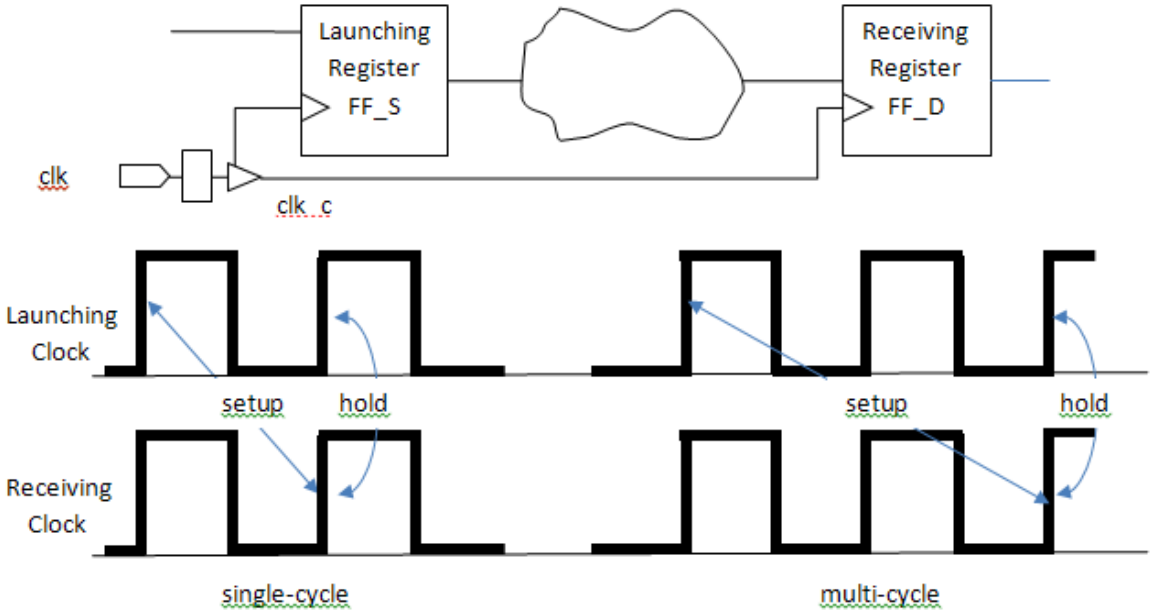
For the multi-cycle timing relationship, the amount of time taken by the data to reach the receiving register can be indicated by a multiplier factor. For example:

```
FREQUENCY PORT "clk" 200 MHZ;
MULTICYCLE FROM CELL "FF_S" TO CELL "FF_D" 2 X;
```

In the example, a multiplier factor of “2 X” is used to inform TRACE and PAR that the data transferred from FF_S to FF_D requires an additional clock cycle.

If a multiplier factor is used in the MULTICYCLE preference, by default this factor will apply to the receiving clock period (destination). You can specify

Figure 32: Single-cycle vs. Multi-cycle Relationship Within the Same Clock Domain



whether the launching register's clock period (source) or the receiving register's clock period (destination) should be applied. For example:

```
MULTICYCLE FROM CELL "FF_S" TO CELL "FF_D" 2 X_DEST;
MULTICYCLE FROM CELL "FF_S" TO CELL "FF_D" 2 X_SOURCE;
```

The calculation formula of the timing requirement for the receiving register is as follows:

```
<default delay calculated> + (n - 1) * <multiplier factor
applied clock period>
```

Here "n" is the multiplier factor. The default delay, or the default timing requirement, is the default register-to-register timing requirement. Since both of the registers are clocked by the same clock, the default delay calculated will be one clock cycle.

In this example, since both "FF_S" and "FF_D" are clocked by CLK, different options will make no difference. The timing requirement from "FF_S" to "FF_D" is calculated by the following:

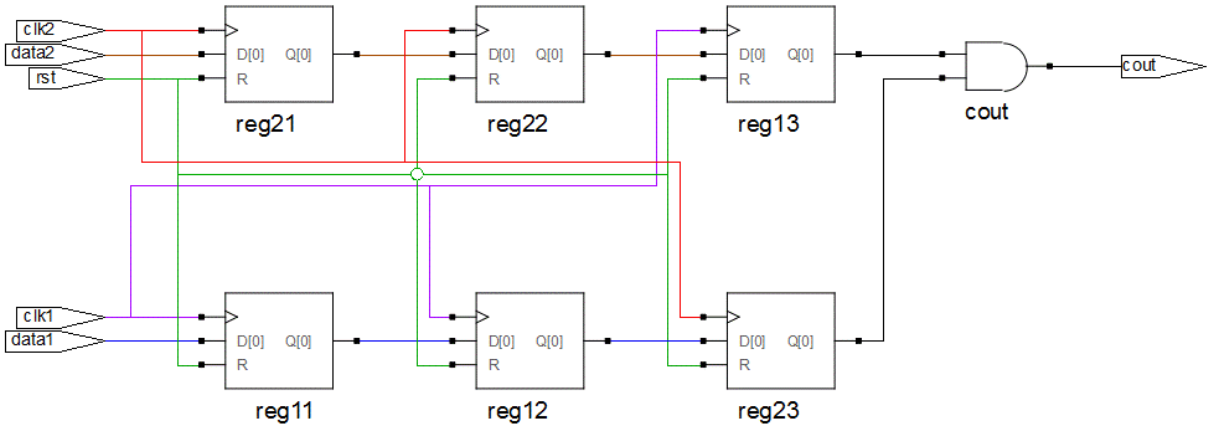
```
5ns + (2 - 1) * 5ns = 10ns
```

This is two clock cycles. So the timing requirement for the path from FF_S to FF_D is two clock cycles (which is 10ns based on the 200MHz FREQUENCY preference) instead of one clock cycle (which is 5ns) that would otherwise be used as the default timing constraint by TRACE and PAR.

Multi-Cycle Across Clock Domains

The example used in the previous case studies has two clock domains and two cross-domain paths, as seen in Figure 33.

Figure 33: Multi-Cycle Across Clock Domains



Cross-domain paths include those from the “clk1” domain to the “clk2” domain, i.e., “reg12” to “reg23,” and those from the “clk2” domain to the “clk1” domain, i.e., “reg22” to “reg13.”

In this example, the clocks “clk1” and “clk2” are unrelated. By default, the engine does not cover paths that are transferred between unrelated clock domains. In “Case Study 6 – CLKSKEWDIFF” on page 60, we use the CLKSKEWDIFF preference and establish the relationship between two clock domains; therefore, the paths between them will be analyzed.

Assume that the clock period of the launching register FF_S is PL, and that the clock period of the receiving register FF_D is PR. When analyzing paths crossing these two clock domains, the engine uses the following approach to calculate and apply the default timing requirement:

1. Align both clocks’ first active edge at time tp0 = 0, which means that at time tp0, the time different between two active clock edges td0 = 0ns.

   By doing this, we will know that at time tpN = N * LCM(PL, PR), the 2 clocks’ active edge will be aligned again. Here N is any integer, and LCM is “least common multiple”. For example, if PL is equal to 2 and PR is equal to 3, then LCM(PL, PR) is 6.

2. Between the time tp0 = 0 and tp1 = 1 * LCM(PL, PR) = LCM(PL , PR), find two positive integers m and n, where m and n meet the following criteria:

   a. tp0 <= m * PL < n * PR <= tp1, that is, 0 <= m * PL < n * PR <= LCM (PL, PR)

   b. the value of tmin is the smallest possible number of t, where

      t = (n * PR) - (m * PL)

3. The value of tmin is the default timing requirement from the launching register FF_S to the receiving register FF_D.

For example, if we have the following preferences defined in the LPF:

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 500.000000 MHz ;
FREQUENCY PORT "clk2" 333.000000 MHz ;
INPUT_SETUP PORT "data1" 2.000000 ns CLKPORT "clk1" ;
INPUT_SETUP PORT "data2" 1.500000 ns CLKPORT "clk2"
CLK_OFFSET 1.500000 X ;
CLOCK_TO_OUT PORT "cout" 1.000000 ns CLKPORT "clk1" ;
CLKSKEWDIFF CLKPORT "clk1" CLKPORT "clk2" 0.500000 ns ;
```

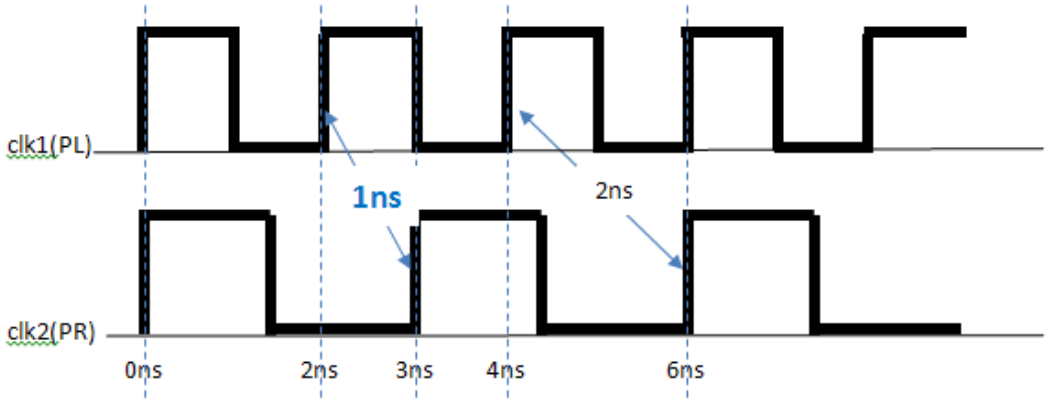
For the cross-domain paths that are from “clk1” domain to “clk2” domain, i.e., “reg12” to “reg23”:

- PL = 2ns, PR = 3ns
- The smallest t is:

  tmin = min (t) = min ((n * PR) - (M*PL)) = 1 * 3 - 1 * 2 = 1ns

By default, 1ns is the timing requirement for these cross-domain paths, as illustrated in Figure 34.

Figure 34: Timing Requirement for Cross-Domain Paths



This can also be found in the TRACE report, as 1ns was reported as the delay constraint.

```
Error: The following path exceeds requirements by 0.609ns (weighted slack = -
1.827ns)
 Logical Details:  Cell type  Pin type       Cell/ASIC name  (clock net +/-)
   Source:         FF         Q              reg12  (from clk1_c +)
   Destination:    FF         Data in        reg23  (to clk2_c +)
   Delay:               0.956ns  (31.7% logic, 68.3% route), 1 logic levels.
 Constraint Details:
      0.956ns physical path delay SLICE_0 to SLICE_1 exceeds
      (delay constraint based on source clock period of 2.000ns and destination
clock period of 3.003ns)
      1.000ns delay constraint less
      0.500ns skew and
      0.153ns M_SET requirement (totaling 0.347ns) by 0.609ns
 Physical Path Details:
......
```

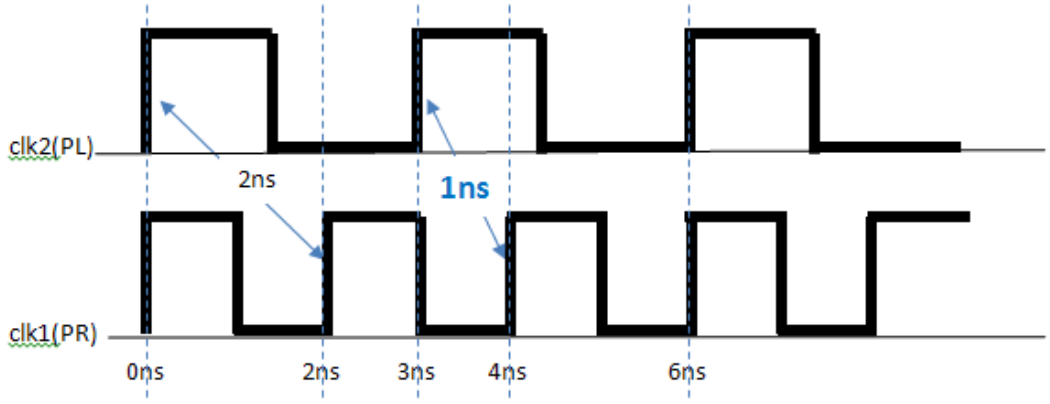
Similarly, for the cross-domain paths that are from the “clk2” domain to the “clk1” domain, i.e., “reg22” to “reg13”:

- PL = 3ns, PR = 2ns
- The smallest t is:

  tmin = min (t) = min ((n * PR) - (M*PL)) = 2 * 2 - 1 * 3 = 1ns

By default, 1ns is also the timing requirement for these cross-domain paths, as illustrated in Figure 35.

Figure 35: Timing Requirement for Cross-Domain Paths



For cross-domain paths, using the default calculated minimum delay between two active clock edges as the timing requirement might not reflect the actual design behavior; and, in most cases, it will have your design and the engine over-constrained. This usually has two side effects:

- TRACE will probably report many timing errors.

- The timing-driven PAR engine will spend a lot of runtime trying to meet the unrealistic requirements, while the true critical paths might be under-covered.

To overcome this issue, you should use the MULTICYCLE preference to describe the relationship of the two clock domains.

As described in Multi-Cycle Within the Same Clock Domain, you can use the multiplier factor when defining MULTICYCLE preferences. For example:

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
FREQUENCY PORT "clk1" 500.000000 MHz ;
FREQUENCY PORT "clk2" 333.000000 MHz ;
INPUT_SETUP PORT "data1" 2.000000 ns CLKPORT "clk1" ;
INPUT_SETUP PORT "data2" 1.500000 ns CLKPORT "clk2"
CLK_OFFSET 1.500000 X ;
CLOCK_TO_OUT PORT "cout" 1.000000 ns CLKPORT "clk1" ;
CLKSKEWDIFF CLKPORT "clk1" CLKPORT "clk2" 0.500000 ns ;
MULTICYCLE FROM CLKNET "clk1_c" TO CLKNET "clk2_c" 2 X;
MULTICYCLE FROM CLKNET "clk2_c" TO CLKNET "clk1_c" 2 X;
```

Then the engine will use the following formula to calculate the timing requirement:

```
<default delay calculated> + (n - 1) * <multiplier factor
applied clock period>
```

In this example, for the path from “clk1” domain to “clk2” domain, the timing requirement is as follows:

```
1ns + (2 - 1) * 3ns = 4ns
```

Similarly, for the path from “clk2” domain to “clk1” domain, the timing requirement is as follows:

```
1ns + (2 - 1) * 2ns = 3ns
```

In addition to using the multiplier factor, you can use an absolute delay value, in nanoseconds, when defining MULTICYCLE preferences. For example:

```
MULTICYCLE FROM CLKNET "clk1_c" TO CLKNET "clk2_c" 3.000000
ns ;
MULTICYCLE FROM CLKNET "clk2_c" TO CLKNET "clk1_c" 2.000000
ns ;
```

Defining MULTICYCLE using clock names will have the constraints apply to all paths covered by the clocks. To specify MULTICYCLE for a specific path, you can use the format of the following example:

```
MULTICYCLE FROM CELL "reg12" TO CELL "reg23" 3.000000 ns ;
MULTICYCLE FROM CELL "reg22" TO CELL "reg13" 2.000000 ns ;
```

For detailed information about MULTICYCLE, its syntax and usage, refer to the *Constraints Reference Guide* in the Diamond online Help.

What is Learned from Case Study 7

From this case study, the following points are learned:

- The timing requirements for cross-domain paths between related clocks can be relaxed by using MULTICYCLE preferences. Otherwise, the engine will use the default calculated timing delay requirement, which is the worst case edge-to-edge delay of the two clocks, and this will cause the engine to be over-constrained in most instances.
- When a multiplier factor is used in a MULTICYCLE preference, the timing requirement calculation formula is as follows:

  ```
  <default delay calculated> + (n - 1) * <multiplier factor
  applied clock period>
  ```

  In order to relax the engine, the multiplier factor “n” must be greater than 1. If it is equal to 1, which is the default, the engine will behave as if the MULTICYCLE preference has not been defined. If it is less than 1, the engine will be even more over-constrained, which is not what you expect. Diamond does not issue a warning when the multiplier factor is less than or equal to 1.

Case study 8 – Clock Over-Constrained

Over-constraining clocks in your design might work for some designs, but you should not use it as a “cure-all” practice, because it can introduce side effects. For those designs that only have a single clock domain, or that have multiple unrelated clock domains, the side effects introduced are not very obvious. But for designs that have multiple clock domains and where cross-domain paths do exist, you should be especially careful. In this case, if cross-domain paths are not appropriately constrained, you can actually drive the engine incorrectly. The engine will then spend a huge amount of time trying to meet the unrealistic timing requirements and eventually fail with a large amount of timing errors.

For example, as explained in “Case Study 7 – Timing Exception 1 – MULTICYCLE” on page 63, for the following FREQUENCY preferences:

```
FREQUENCY PORT "clk1" 300.000000 MHz ;
FREQUENCY PORT "clk2" 150.000000 MHz ;
```

The default delay requirement will be 3.333ns for all cross-domain paths from the “clk1” domain to the “clk2” domain.

Now instead of defining 300MHz for the “clk1,” we over-constrain it by 3MHz through the following preference (similarly, “clk1” can be over-constrained by a few):

```
FREQUENCY NET "clk1" 303.000000 MHz;
FREQUENCY NET "clk2" 150.000000 MHz;
```

The default delay requirement from the “clk1” domain to the “clk2” domain will become 0.066ns, which is the worst-case edge-to-edge delay. This result can be observed in the TRACE report:

```
================================================================================
Preference: FREQUENCY PORT "clk2 " 150.000000 MHz ;
            2 items scored, 1 timing errors detected.
--------------------------------------------------------------------------------
Error: The following path exceeds requirements by 1.700ns (weighted slack = -
171.700ns)
 Logical Details:  Cell type  Pin type       Cell/ASIC name  (clock net +/-)
   Source:         FF         Q              reg12  (from clk1_c +)
   Destination:    FF         Data in        reg23  (to clk2_c +)
   Delay:               1.116ns  (27.2% logic, 72.8% route), 1 logic levels.
 Constraint Details:
      1.116ns physical path delay SLICE_0 to SLICE_1 exceeds
      (delay constraint based on source clock period of 3.300ns and destination clock
period of 6.666ns)
      0.066ns delay constraint less
      0.497ns skew and
      0.153ns M_SET requirement (totaling -0.584ns) by 1.700ns
......
```

The delay constraint calculated decreased dramatically from 3.333ns in the previous cases to 0.066ns, which is apparently unrealistic. The engine is over-constrained well beyond the 3MHz specified in the LPF, which you would probably never expect.

The reason for a “0.066ns” delay constraint is that the engine uses the closest edge gap between two clocks, which is the worst case, as the constraint for the paths crossing multiple domains. As explained in “Case Study 7 – Timing Exception 1 – MULTICYCLE” on page 63, this behavior should be guided with a MULTICYCLE preference.

Using PAR_ADJ

A similar situation could happen if you use “PAR_ADJ” when defining a FREQUENCY or PERIOD preference where the related cross-domain paths are not well constrained. The PAR_ADJ keyword allows you to tighten requirements for PAR while preserving the requirements reported by TRACE. This allows you to over-constrain PAR. For example:

```
FREQUENCY NET "clk1" 300.100000 MHz PAR_ADJ 3;
```

This preference instructs the PAR engine to use 303MHz as the “clk1” FREQUENCY requirement. At the same time, TRACE still uses 300MHz for static timing analysis.

Since the TRACE reports still use the defined FREQUENCY or PERIOD for static timing analysis, you might not notice anything going on incorrectly. But the timing-driven PAR might have completely different numbers to drive itself and spend a huge amount of time trying to meet the timing.

What Is Learned from Case Study 8

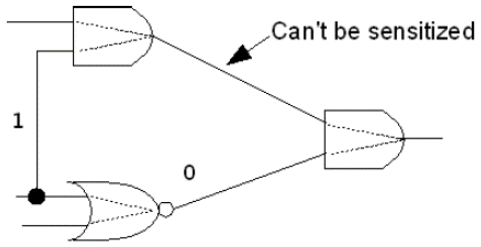
From this case study, the following points are learned:

- Over-constraining your design is not always a good practice and can result in an even more over-constrained engine, especially when your design has cross-domain paths and these paths are not well constrained.
- PAR_ADJ could result in exactly the same problem as over-constraining if cross-domain paths are not handled well in your preferences. It might not be identified as easily as explicit over-constraining, because the PAR engine uses a completely different FREQUENCY or PERIOD number than that used by TRACE.
- Cross-domain paths should be carefully handled using MULTICYLE preferences.

Case study 9 – Timing Exception 2 – False Paths

Many designs include paths that are asynchronous relative to the clocks of the design or connections that never propagate a signal state because of logic encoding. A false path illustration is shown in Figure 36.

Figure 36: False Path



If not well constrained, this condition can “mask” the violations of real timing paths and make the performance results overly pessimistic.

False paths are treated as unconstrained by TRACE and timing-driven PAR. If you can accurately describe false paths, design performance will usually improve, because a false path is treated by PAR as unconstrained. With “relaxed” timing objectives, PAR optimizes the true critical paths instead. In a similar manner, unconstrained paths are ignored by TRACE and true critical paths are reported instead.

You should use the BLOCK preference on those identified false paths. Refer to the BLOCK PATH section for details.

Case study 10 – Use PLL FREQUENCY Settings

Most designs use PLLs to generate clocks driving the FPGA circuit. The following example shows a design that uses a PLL. This design is similar to the one used in all of the previous case studies, but instead of using two unrelated top-level clocks, it uses a PLL.

```
module example(clk1, data1, data2, rst, cout, pll_lock);
input clk1, data1, data2, rst;
output cout, pll_lock;
reg reg11, reg12, reg13;
```

```
reg reg21, reg22, reg23;
wire clkop, clkok;

my_pll i_my_pll (.CLK(clk1), .RESET(rst), .CLKOP(clkop),
.CLKOS(), .CLKOK(clkok), .LOCK(pll_lock));

always @ (posedge clkop)
begin
  if (rst)
begin
  reg11<=1'b0;
  reg12<=1'b0;
  reg13<=1'b0;
end
  else
    begin
      reg11<=data1;
      reg12<=reg11;
      reg13<=reg22;
    end
end

always @ (posedge clkok)
begin
  if (rst)
begin
  reg21<=1'b0;
  reg22<=1'b0;
  reg23<=1'b0;
end
  else
    begin
      reg21<=data2;
      reg22<=reg21;
      reg23<=reg12;
    end
end

assign cout = reg13 & reg23;

endmodule
```

The PLL has the following frequency settings: 100MHz “clk1” input, 300MHz “clkop” output, and 150MHz “clkok” output.

If there is no FREQUENCY preference defined in your LPF file, the FREQUENCY values from the PLL will be used to drive the engine. This can be observed in the TRACE report:

```
Preference Summary
•  FREQUENCY NET "clk1_c" 100.000000 MHz (0 errors)
            0 items scored, 0 timing errors detected.
•  FREQUENCY NET "clkop" 300.000000 MHz (0 errors)
            5 items scored, 0 timing errors detected.
Report:  375.094MHz is the maximum frequency for this
preference.
•  FREQUENCY NET "i_my_pll/CLKOS" 300.000000 MHz (0 errors)
            0 items scored, 0 timing errors detected.
```

```
• FREQUENCY NET "clkok" 150.000000 MHz (0 errors)
            5 items scored, 0 timing errors detected.
Report:  375.094MHz is the maximum frequency for this
preference.
```

Here the “clkok” domain runs 150MHz and the “clkop” domain runs 300MHz, which are from the PLL frequency settings.

This example still shows cross-domain behavior. In addition, since both clocks are outputs from a PLL, by definition, these two clocks are related. Because the engine will analyze cross-domain paths, MULTICYCLE preferences are needed to relax the engine. See “Case Study 7 – Timing Exception 1 – MULTICYCLE” on page 63.

One important fact to remember when using PLL: after the PLL’s clocks are routed, there will be skews between different clock outputs. The skew will be calculated automatically by the engine, as shown in the PAR TRACE report:

```
Preference: FREQUENCY NET "clkok" 150.000000 MHz ;
            5 items scored, 0 timing errors detected.
--------------------------------------------------------------------------------
Passed:  The following path meets requirements by 4.000ns
         The internal maximum frequency of the following component is 375.094 MHz
 Logical Details:  Cell type  Pin name       Component name
   Destination:    FSLICE     CLK            SLICE_1
   Delay:               2.666ns -- based on Minimum Pulse Width
Passed: The following path meets requirements by 2.218ns (weighted slack = 4.436ns)
 Logical Details:  Cell type  Pin type       Cell/ASIC name  (clock net +/-)
   Source:         FF         Q              reg12  (from clkop +)
   Destination:    FF         Data in        reg23  (to clkok +)
   Delay:               1.165ns  (26.0% logic, 74.0% route), 1 logic levels.
 Constraint Details:
      1.165ns physical path delay SLICE_0 to SLICE_1 meets
      3.333ns delay constraint less
     -0.203ns skew and
      0.000ns feedback compensation and
      0.153ns M_SET requirement (totaling 3.383ns) by 2.218ns
Physical Path Details:
......
Clock Skew Details:
      Source Clock Path clk1 to SLICE_0:
   Name    Fanout   Delay (ns)          Site               Resource
PADI_DEL    ---     0.508         K3.PAD to       K3.PADDI clk1
ROUTE         1     0.000       K3.PADDI to PLL_R26C5.CLKI clk1_c
CLKI2OP_DE  ---     0.000 PLL_R26C5.CLKI to *L_R26C5.CLKOP i_my_pll/PLLInst_0
ROUTE         2     1.445 *L_R26C5.CLKOP to     R27C2B.CLK clkop
                  --------
                    1.953   (26.0% logic, 74.0% route), 2 logic levels.
PLL_R26C5.CLKOP attributes:
      Source Clock f/b:
   Name    Fanout   Delay (ns)          Site               Resource
CLKFB2OS_D  ---     0.203 *L_R26C5.CLKFB to *L_R26C5.CLKOS i_my_pll/PLLInst_0
ROUTE         1     1.632 *L_R26C5.CLKOS to *L_R26C5.CLKFB i_my_pll/CLKOS
                  --------
                    1.835   (11.1% logic, 88.9% route), 1 logic levels.
PLL_R26C5.CLKOS attributes: PHASEADJ=22.5
      Destination Clock Path clk1 to SLICE_1:
   Name    Fanout   Delay (ns)          Site               Resource
PADI_DEL    ---     0.508         K3.PAD to       K3.PADDI clk1
ROUTE         1     0.000       K3.PADDI to PLL_R26C5.CLKI clk1_c
CLKI2OK_DE  ---     0.203 PLL_R26C5.CLKI to *L_R26C5.CLKOK i_my_pll/PLLInst_0
ROUTE         2     1.445 *L_R26C5.CLKOK to     R27C2A.CLK clkok
                  --------
                    2.156   (33.0% logic, 67.0% route), 2 logic levels.
      Destination Clock f/b:
   Name    Fanout   Delay (ns)          Site               Resource
CLKFB2OS_D  ---     0.203 *L_R26C5.CLKFB to *L_R26C5.CLKOS i_my_pll/PLLInst_0
ROUTE         1     1.632 *L_R26C5.CLKOS to *L_R26C5.CLKFB i_my_pll/CLKOS
                  --------
                    1.835   (11.1% logic, 88.9% route), 1 logic levels.
PLL_R26C5.CLKOS attributes: PHASEADJ=22.5
```

The “clock skew details” section shows how the clock skew was calculated. From here, you can also see the phase adjustment (shown in blue), that was set when you generated the PLL from IPExpress.

Overwrite PLL FREQUENCY Settings

There are cases where you might want to overwrite PLL FREQUENCY settings. For example:

- when you use a different FREQUENCY to drive the engine and static timing analysis
- when you apply other options such as PAR adjustment using PAR_ADJ, specify hold margin using HOLD_MARGIN, or specify peak-to-peak jitter value for the incoming clock using CLOCK_JITTER

To overwrite PLL FREQUENCY settings, simply add FREQUENCY preferences to your LPF file. For example:

```
FREQUENCY NET "clkok" 165.000000 MHz;
FREQUENCY NET "clkop" 330.000000 MHz;
```

The TRACE reports now show that the new FREQUENCY preferences defined in the LPF are used:

```
Preference Summary
•  FREQUENCY NET "clk1_c" 100.000000 MHz (0 errors)
            0 items scored, 0 timing errors detected.
•  FREQUENCY NET "clkop" 330.000000 MHz (0 errors)
            4 items scored, 0 timing errors detected.
Report:  375.094MHz is the maximum frequency for this
preference.
•  FREQUENCY NET "i_my_pll/CLKOS" 300.000000 MHz (0 errors)
            0 items scored, 0 timing errors detected.
•  FREQUENCY NET "clkok" 165.000000 MHz (0 errors)
            4 items scored, 0 timing errors detected.
Report:  375.094MHz is the maximum frequency for this
preference.

WARNING - trce: The Preference FREQUENCY NET at signal clkop
(CLKOP) or signal i_my_pll/CLKOS (CLKOS) do not match their
divider settings for i_my_pll/PLLInst_0
```

From the report, you can easily see that there is a warning about the overwriting of PLL settings. You should make sure that the overwriting is indeed intended.

One important fact when overwriting the PLL settings: the PLL has been configured and generated from IPExpress and its function has been fixed. The overwriting values defined in the LPF file only affect the engine, to drive other parts of your design, and the TRACE report. TRACE will calculate slacks and other timing values, such as clock skews, based on the preferences defined. To actually change the PLL definition, you must use IPExpress to reconfigure it and regenerate it.

What Is Learned from Case Study 10

From this case study, the following points are learned:

- Designs with PLLs usually have FREQUENCY defined through the PLLs. If FREQUENCY preferences are not defined in your LPF file, the PLL settings will take effect.
- When a design including PLLs has multiple clock domains, MULTICYCLE preferences are still needed. Otherwise, the design is under-constrained.
- FREQUENCY preferences in the LPF file can be used to overwrite PLL settings and drive the engine and static timing analysis.
- Using FREQUENCY preferences in the LPF file allows you to specify other options such as PAR_ADJ, which cannot be done when relying only on PLL settings.
- Overwriting PLL settings will not affect the PLL itself. To reconfigure the PLL using a different FREQUENCY, use IPExpress.

Case study 11 – BLOCK Preferences

The BLOCK preference in Diamond allows you to block certain nets, buses, paths, and component pins that are irrelevant to the timing requirement of your design. This prevents them from being considered by TRACE and the timing-driven engine, and it improves runtime. If well used, the BLOCK preference also allows the engine to focus on true critical paths.

For detailed information about BLOCK preference, refer to the *Constraints Reference Guide* in the Diamond online Help.

Default BLOCK Preferences

When you start a new Diamond project and implementation, two BLOCK preferences are added automatically:

```
BLOCK RESETPATHS;
BLOCK ASYNCPATHS;
```

Refer to “Case Study 1 – No user-defined timing constraint” on page 47 for details.

BLOCK PATH

For identified asynchronous paths, you can use a BLOCK preference to prevent the engine and TRACE from analyzing any defined nets, paths, buses, or component pins that are irrelevant to the timing of the design; for example, asynchronous paths from input pads to registers, asynchronous paths from input pads to output pads, or all asynchronous reset nets in the design. This will release the engine from spending time on those asynchronous paths in order to meet other timing requirements, and it will avoid generating incorrect timing reports. The following is an example of a BLOCK preference:

```
BLOCK PATH FROM CELL "I_pci_slave_reg/*" TO CELL "I_0/*";
```

BLOCK RAM Reads during Write

If you are using PFU-based RAM, the BLOCK RD_DURING_WR_PATHS preference will prevent timing analysis on a RAM read during a write on the same address in a single clock period. By default, this is off.

BLOCK INTERCLOCKDOMAIN PATHS

As explained in “Case Study 7 – Timing Exception 1 – MULTICYCLE” on page 63, when there are multiple clock domains in your design, you should check to see whether there are cross-domain paths between related clocks. If there are, you should use MULTICYCLE preferences to define the timing relationship between these paths. If you do not, you should expect many timing errors in the TRACE reports.

If you need to prevent the engine from considering and analyzing related cross-domain paths temporarily, you can use the following preference:

```
BLOCK INTERCLOCKDOMAIN PATHS
```

Since paths crossing multiple clock domains are not analyzed, now the engine becomes under-constrained, any many cross-domain timing issues are now under-covered. This under-constraining can be examined by looking at the percentage of constraint coverage, where the coverage drops dramatically.

Instead of blocking all the related cross-domain paths, you can also selectively block certain paths using the BLOCK PATH preference. For example, the following preference will block the paths from the “clk1” domain to the “clk2” domain:

```
BLOCK PATH from CLKNET "clk1_c" to CLKNET "clk2_c";
```

You can also block a specific path using the BLOCK PATH preference, as explained in the section BLOCK PATH.

If you block only part of cross-domain paths, you might have a higher constraint coverage percentage than if you use the BLOCK INTERCLOCKDOMAIN PATHS preference, but potential timing issues can still exist in your design.

What Is Learned from Case Study 11

From this case study, the following points are learned:

- If your design has multiple clock domains, you need to pay additional attention to the possibility that cross-domain paths exist between related clocks.
- If your design does have cross-domain paths between related clocks, it must be well-constrained using the MULTICYCLE preference. Otherwise, the TRACE report can contain many unnecessary timing errors, and the engine will be over-constrained.
- You can use the BLOCK INTERCLOCKDOMAIN PATHS preference to temporarily prevent the engine from looking at all the cross-domain paths between related clocks. Or you can use BLOCK PATH preferences to block part of them and relax the engine, but you will miss many potential timing issues existing in your design. Unless ignoring cross-domain paths is desirable, you should appropriately constrain these paths using MULTICYCLE preferences.

Recommendations for Using Timing Preferences

In the section “Understand Precedence Rules for Preferences” in the Design Planning chapter, we discussed the preference precedence rules. To recap:

- Rule number 1: Preferences defined in an LPF file take precedence over attributes and directives defined in the HDL code.

  For example, if you have a FREQUENCY attribute defined in your HDL code, it will be overwritten by a FREQUENCY preference in your LPF file if they constrain the same clock. One example of a FREQUENCY attribute defined in the HDL code is PLLs generated from IP Express

- Rule number 2: Preferences that are more specific take precedence over less specific ones. This means that individual net or path preferences supersede group (bus) preferences, and group preferences supersede global preferences.

  For example, if you have the following two INPUT_SETUP preferences in your LPF,

  ```
  INPUT_SETUP ALLPORTS INPUT_DELAY 3.000000 ns CLKNET
  "clk1_c";
  INPUT_SETUP PORT "data1" 4.000000 ns CLKNET "clk1_c";
  ```

  The input “data1” will have 4ns setup time requirement. In the following example,

  ```
  CLOCK_TO_OUT "cout" 1.0 ns CLKPORT "clk" CLKOUT PORT
  "clkout";
  CLOCK_TO_OUT "cout" 2.0 ns CLKPORT "clk" FROM "c2" CLKOUT
  PORT "clkout";
  ```

  the CLOCK_TO_OUT requirement for “c2” is 2ns instead of 1ns, which is more general.

- Rule number 3: Preferences defined later in an LPF file take precedence over preferences if these preferences are at the same level and are in conflict.

  For example, if you have the following 2 FREQUENCY preferences in your LPF:

  ```
  FREQUENCY NET "clk_c" 300.000000 MHz;
  FREQUENCY NET "clk_c" 200.000000 MHz;
  ```

  “clk_c” FREQUENCY will be 200MHz.

  It is best to avoid this kind of conflict in your LPF file. See “Recommendations for Creating and Editing LPF” in the Design Planning document.

When defining timing constraints, in addition to these general preference precedence rules, you should understand and follow these additional recommendations:

- Rule number 4: Timing Preference Dependency – FREQUENCY/PERIOD

  FREQUENCY and PERIOD preferences are the primary timing requirements, and all other timing preferences depend on them. If none of these preferences is defined, MAP will generate default FREQUENCY preferences, as explained in “Case Study 1 – No user-defined timing constraint” on page 47, and this will usually over-constrain your design.

Appropriate FREQUENCY or PERIOD preferences should be defined for each and all clocks in your design, either in your HDL (such as using PLLs) or in your LPF.

- Rule number 5: Appropriate Timing Preferences

  From the case studies previously discussed, you should understand that in order to properly drive the timing-driven engine and TRACE, every design should include appropriate timing constraints:

  - The two default BLOCK preferences should always be included and stay at the top of your active LPF file.
  - All unrelated clocks must be constrained using either the FREQUENCY or the PERIOD preference. The Clock Domains Analysis section in the TRACE report should help identify the number of clocks in your design and whether they are properly constrained.
  - INPUT_SETUP preferences for all inputs should be defined according to your PCB timing requirements.
  - CLOCK_TO_OUT preferences for all outputs should be defined according to your PCB timing requirements.
  - If your design has multiple clocks, you should examine your design and determine whether any cross-domain paths exist in your design.
  - If there are cross-domain paths between related clocks in your design, you should use MULTICYCLE preferences to define the timing requirements for all those cross-domain paths.
  - If there are cross-domain paths between unrelated clocks in your design, you should first establish the relationship between the clocks using the CLKSKEWDIFF preference, and then you should define the timing requirements using MULTICYCLE preferences.
  - You should not over-constrain your design, but you should appropriately constrain your design with your actual timing requirement.
  - If you need to use BLOCK preferences, you should fully understand what each of them does and what effect it could have on your design. As a good practice, you should put all the BLOCK preferences at the top of your LPF file for easy debugging.
  - It is strongly suggested that you not use the BLOCK INTERCLOCKDOMAIN PATHS preference. If you use it, all cross-domain paths will be blocked and will not be considered by the engine and TRACE, even if you have MULTICYCLE preferences defined and even if the BLOCK preference comes after the MULTICYCLE preferences.

Last Check: Complete Timing Preference Coverage

To ensure that your design and the engine are neither over-constrained nor under-constrained, you should carefully examine your LPF file. In addition to all the recommendations you have read so far, you should make sure that your LPF file includes all of the following preferences, if applicable, and that they are complete and correct:

- The basics:

    - FREQUENCY or PERIOD for each and all clocks. An inappropriate value could over-constrain or under-constrain your design.
- Preferences for tightening the engine with the appropriate values:
    - INPUT_SETUP
    - CLOCK_TO_OUT
    - MAXDELAY
    - CLKSKEWDIFF
- Preferences for relaxing the engine with the appropriate values:
    - MULTICYCLE
    - CLKSKEWDISABLE
    - MAXSKEW
    - BLOCK

Finally and importantly, you should check the preference coverage section in your TRACE report to ensure a high percentage of coverage. To find out those paths that are neither covered nor analyzed by TRACE and PAR, turn on the “Check Unconstrained Paths” option in the TRACE strategy settings and examine the result in the TRACE reports.

Other Considerations

Hold-Time Analysis

If you enable the Hold Analysis through the TRACE strategy settings, which is the default setting, TRACE will produce a hold-time check based on your timing preferences.

By default, TRACE analyzes designs for setup time violations using the worst case operating conditions for the target performance grade. In contrast to setup time analysis, hold time analysis uses “best case” operating conditions. This approach of analyzing at both corners of the operating conditions establishes a well-defined range in which the device will operate successfully.

As explained in “Timing-Driven PAR Process” on page 30, PAR only tries to correct setup time violations when auto-hold time correction is not enabled through the PAR strategy settings. You should always examine the hold time analysis result in the TRACE report to ensure that there are no hold time violations in your final placed and routed design.

Use Primary or Secondary Clocks

For clock planning to help with timing closure, refer to the section “Clock Assignment” in the Design Planning chapter.

Primary clock resources on a device are limited. Therefore, if there is no user preference, the clock nets with the most loads will automatically be assigned the primary clock resources by PAR. If your design has multiple clocks, you

can explicitly assign or prohibit them by using the PRIMARY and SECONDARY preferences.

```
USE PRIMARY NET "CK38A_c" ;
USE SECONDARY NET "CK38B_c" ;
PROHIBIT PRIMARY NET “CLK_c”;
```

Other considerations when using the dedicated clock resources:

- To get an accurate 90-degree phase shift, use two primary clock nets: one for the feedback path and one for the shifted clock. This limits uncertainty to the insertion delay of sysCLOCK PLL (pad to input). The uncertainty can then be reconciled with FDEL settings in 250-picosecond increments.
- Place the source of internally generated clocks (divider) as close to the center of the device as possible to reduce injection time. This is especially important for secondary clocks, since they do not have feed lines.

Tune I/O Timing with PLLs

Tuning the I/O timing with PLLs reconciles internal timing to an external specification.

Group Components along Critical Paths

For the identified critical paths based on the TRACE report, you can try to use UGROUP to group components along the critical paths so that PAR places components close together. This should shorten routing distances along the paths.

MAP Register Retiming

MAP register retiming is an optimization technique that moves registers across combinatorial logic to balance the timing..

There is no guarantee that map register retiming will achieve a better Fmax, since the Fmax constraint activates retiming around all registers. The INPUT_SETUP and CLOCK_TO_OUTPUT constraints might deactivate retiming on I/O registers, depending on the balancing of INPUT_SETUP vs. FREQUENCY and CLOCK_TO_OUTPUT vs. FREQUENCY. However, register retiming can be very useful for optimization because it allows for more delay shifting.

MAP Register Retiming vs. Clock Boosting

See “Clock Boosting” on page 100.

MAP register retiming has the same goal as clock boosting, which adjusts the timing by introducing predefined clock delays. The following considerations should be taken into account when using either of these features for optimizing timing:

- Optimizing with MAP Register Retiming

  MAP register retiming can be either forward or backward. Forward retiming moves a set of registers that are the inputs of logic to a single register at its output. Backward retiming moves a register that is at the output of a logic to a set of registers at its input. Retiming works on a data path and has variable delay shift and variable area cost from design to

design. A drawback to register retiming is that it changes your netlist, making debugging more difficult. It also has a minimum delay shift of one logic level; for example, one LUT.

- Optimizing with Clock Boosting

  Clock boosting works on clock paths and has a fixed delay, such as 0 ns, 1 ns, 2 ns, or 3 ns, and it has a fixed area cost on silicon. The delay shift is accurate after placement and routing and can be as fine as less than or equal to 1 ns. However, clock boosting requires the use of extra silicon area, even if it is not used; and delay shift is limited to a few choices up to about 3 ns or more.

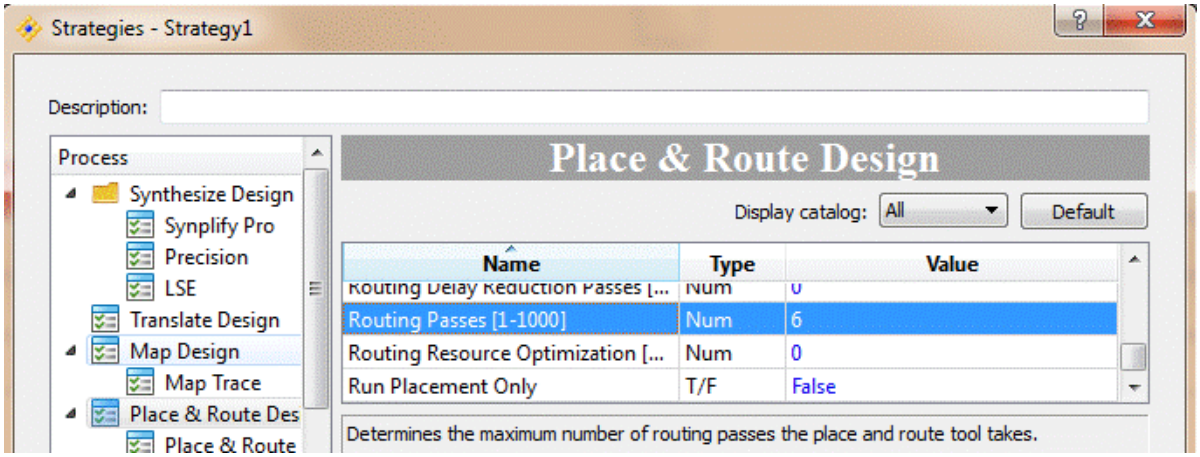
Controlling PAR

Extensive benchmark experiments have been performed to determine the optimum per-device default settings for all PAR options. At times, you can obtain improved timing results on a design-by-design basis by trying different variations of the PAR options. This section describes the techniques that you can use to improve timing results from TRACE on placed and routed designs.

Running Multiple Routing Passes

You can obtain improved timing results by increasing the number of routing passes during the routing phase of PAR. By default, the number of routing passes is 6, but you can change this number through PAR strategy settings, as shown in Figure 37.

Figure 37: Setting the Number of Routing Passes



The router routes the design for the defined number of routing iterations or until all the timing preferences are met, whichever comes first. For example, PAR stops after the second routing iteration if it hits a timing score of zero on the second routing iteration.

You can view the PAR report in the Diamond Report window. The report contains execution information about the PAR run. For example:

```
0 connections routed; 26590 unrouted.
Starting router resource preassignment
Completed router resource preassignment. Real time: 11 mins
31 secs
```

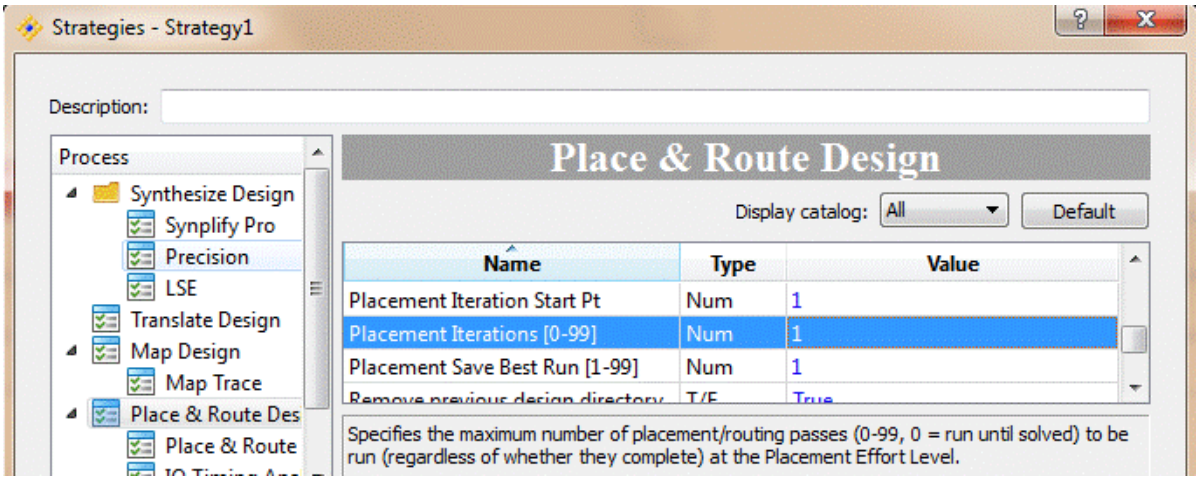
```
Starting iterative routing.
End of iteration 1
26590 successful; 0 unrouted; (151840) real time: 14 mins 29
secs
Dumping design to file
d:\ip\design.ncd.
End of iteration 2
26590 successful; 0 unrouted; (577) real time: 16 mins 23
secs
Dumping design to file
d:\ip\design.ncd.
End of iteration 3
26590 successful; 0 unrouted; (0) real time: 17 mins 39 secs
Dumping design to file
```

The PAR report also shows the steps taken as the program converges on a placement and routing solution. In this routing convergence example, the number in parenthesis is the timing score. In this example, timing was met after three routing iterations, as you can see from the (0) timing score.

Using Multiple Placement Iterations (Cost Tables)

You can specify multiple placement iterations through the PAR strategy settings, as shown in Figure 38.

Figure 38: Setting the Number of Placement Iterations



By default, the number of iterations is set to 1, and the placement start point is set to iteration 1 (cost table 1). You can increase the number of placement iterations and set a different start point. After one PAR iteration is completed, PAR loops back through the PAR flow until the number of iterations has reached the number defined. PAR keeps track of the timing and routing performance for every iteration, and the best result will be used as the final result. If “Placement Iterations” is set to 0, PAR will run indefinitely through multiple iterations until a 0 timing score is reached. In a design that is known to have large timing violations, a 0 timing score is never reached. As a consequence, you must intervene and stop the flow at a given point in time.

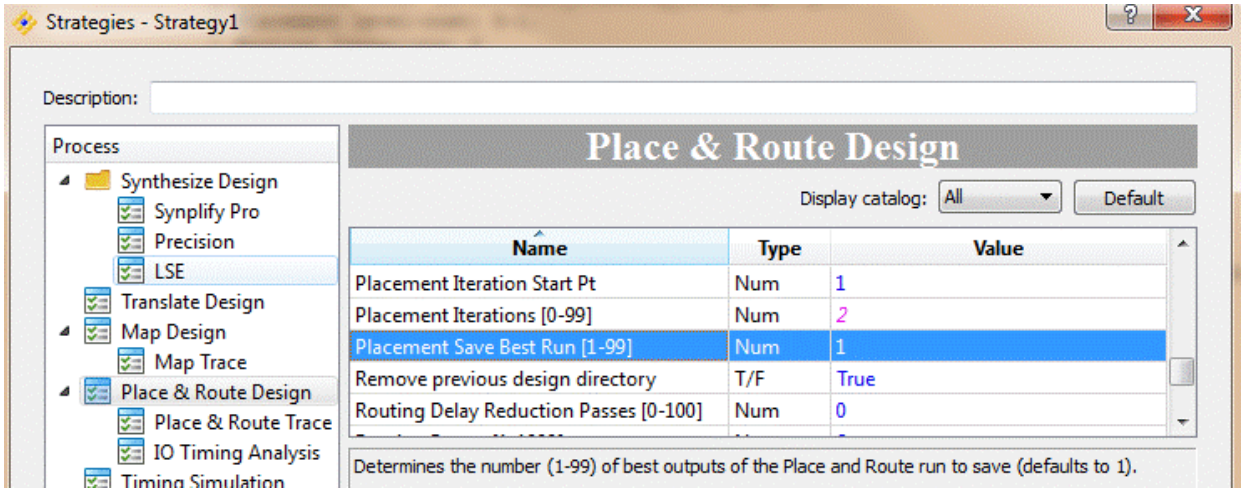
The following is a PAR report example:

```
Cost Table Summary
Level/       Number       Timing       Run          NCD
Cost [ncd]   Unrouted     Score        Time         Status
----------   --------     --------     -----        ------------
5_1   *      0            0            26           Complete
5_2          0            2846         42           Complete
* : Design saved.
```

In this example:

- The “5_” under the Level/Cost column means that the placement effort level was set to 5. The placement effort level can range from 1 (lowest) to 5 (highest).
- Two different iterations ran (2 cost tables).
- Each iteration routed completely.
- Timing scores are expressed in the total number of picoseconds (ps) by which the design is missing constraints on all preferences. This number is additive for all paths in the design.
- Iteration number 1 (cost table 1) achieved a 0 timing score, so it is the design that was saved and is set as the final result. More than one result can be saved. You can control this by setting the value of “Placement Save Best Runs” through the PAR strategy settings, as shown in Figure 39. The default value is 1.

Figure 39: Setting the Number of Best Placement Runs



Sometimes it is a good practice to save more than one result from a multi-PAR run and use PAR TRACE on each result. Since the timing score is a composite of all timing constraints, a low score might not be ideal for your application, unless it is 0.

In general, multiple placement iterations can help placement, but they can also use many CPU cycles. Multiple placement iterations should be used carefully because of system limitations and the uncertainty of results. It is better to fix the root cause of timing problems in the design stage.

Using the NBR Routing Method

The PAR router has two main algorithms: the NBR algorithm (default) and the CDR algorithm. In the default router, the NBR algorithm assumes that all of the critical nets can be routed, and the tool will then work backwards to clean up the legality of the nets. On the other hand, CDR routing is done based on the legality of the nets.

For designs with timing issues, you want to keep the default routing method (NBR) in the PAR strategy settings.

Floorplanning the Design

If performance goals cannot be met with FPGA timing preferences and additional effort levels of the PAR process, you can improve performance by directing the physical layout of the circuit in the FPGA. This step, often referred to as floorplanning, is done by specifying FPGA location preferences.

For detailed information about floor planning, refer to the “Floor Planning” section in the Design Planning document.

Attacking Timing Issues

Introduction

The completion of timing closure, in short, is when an FPGA designer achieves the intended system performance or constraints. Typically, this is measured by the maximum clock frequencies of the system clocks as well as meeting all input setup, clock to output, and hold time requirements. It also involves resolution of any cross-clock domain issues.

Though the practices and considerations discussed in “General Considerations and Practices for Timing Closure” on page 35 should help you achieve your performance goals with most typical designs, there are still cases where timing requirements are hard to achieve. The timing issues in these designs could be caused by various reasons, and different designs might have different timing issue combinations. Identifying the causes and applying the dedicated cures are the keys to successfully addressing the timing problems.

This section discusses the probable causes of timing closure difficulties and how to analyze, debug and identify timing issues. It explains how to apply advance methods such as pipelining, logic retiming, logic grouping, and fan control to improve performance, and it advises you about when and where to apply these advance methods. This section also discusses some software behaviors and work-arounds that can be used to achieve timing closure.

Understand Potential Causes of Timing Closure Issues

Many things can cause timing closure issues. The most common areas of timing closure problems are explained in the following sections.

RTL Coding

This is the most crucial and most effective area for achieving timing closure. Instead of blindly coding your designs and using the “push-button” flow, the best approach is to code your designs specifically for Lattice’s product architecture. This can involve using/instantiating the embedded blocks, pipelining, retiming, etc. To do this effectively, you should understand both hardware and software.

Using Software

When using software tools, you should understand how to best utilize its features and functions. Refer to previous sections and other related documents for details. Misunderstanding or misusing the options and switches of software tools can also lead to timing problems. For example, is timing-driven synthesis really good for a design? (See synthesis “General Considerations” on page 36.) What is the right trade-off between area and speed synthesis mode? Are timing constraints for MAP and PAR accurate and complete? Is the design under-constrained or over-constrained? Are those MAP and PAR options used correctly and appropriately?

Understanding the Hardware

You should fully understand the chosen device. At a minimum, you should fully utilize its capabilities, but not overestimate it. If the architecture and other characteristics of your chosen device are not fully understood, you might not fully utilize it, or you could actually misuse it.

For example, a chosen device has built-in ROM, while a certain part of your design can be implemented as either RAM or ROM, and synthesis tools could infer a RAM based on your coding style, thus leading to a waste of the “free” resource and a potential performance issue.

You should also not overestimate the performance or overlook specific features of your chosen device.

Area Balance

Although parallelism usually means faster, this is not always true. The resource of a chosen FPGA for your design is limited. While usually it will be faster if you increase the degree of parallel processing for the same design, it will also increase the resource usage and lead to a “large” implementation,

with increased numbers of signals and connections. This can introduce long routing or high routing resource usage and push certain non-critical paths to becoming critical. A balance needs to be found.

Resource Utilization

Before you actually analyze the timing problem, you should make sure that your design does not “over utilize” your chosen device.

Over-utilized designs usually cause timing closure issues, because it is nearly impossible for Diamond to honor all of the timing constraints when the required resources exceed the amount available. Experiences shows that slice utilization of more than 85% should be considered as an over-utilized design. In addition, when the number of clock domains exceeds the number of primary clocks available, or block RAM/DSP utilization is 80% or more, the design should be considered an over-utilized design.

To determine the device utilization, look at the details in the Synthesis Report file (if you use Synplify, the file extension is .srr) and the MAP Report file (with the file extension .mrp). These reports can be viewed through either the GUI or a text editor.

The first thing is to look at whether the appropriate resources are allocated or inferred in the synthesis report file. If anything is missing, you should further examine why.

The next step is to check the resource utilization in the MAP Report File. Make sure that the resource usage does not exceed the recommended values.

If the resource utilization exceeds the recommended values, you should recode your HDL or migrate to a larger device. See the document “Congested Design.”

Steps to Resolve Timing Issues

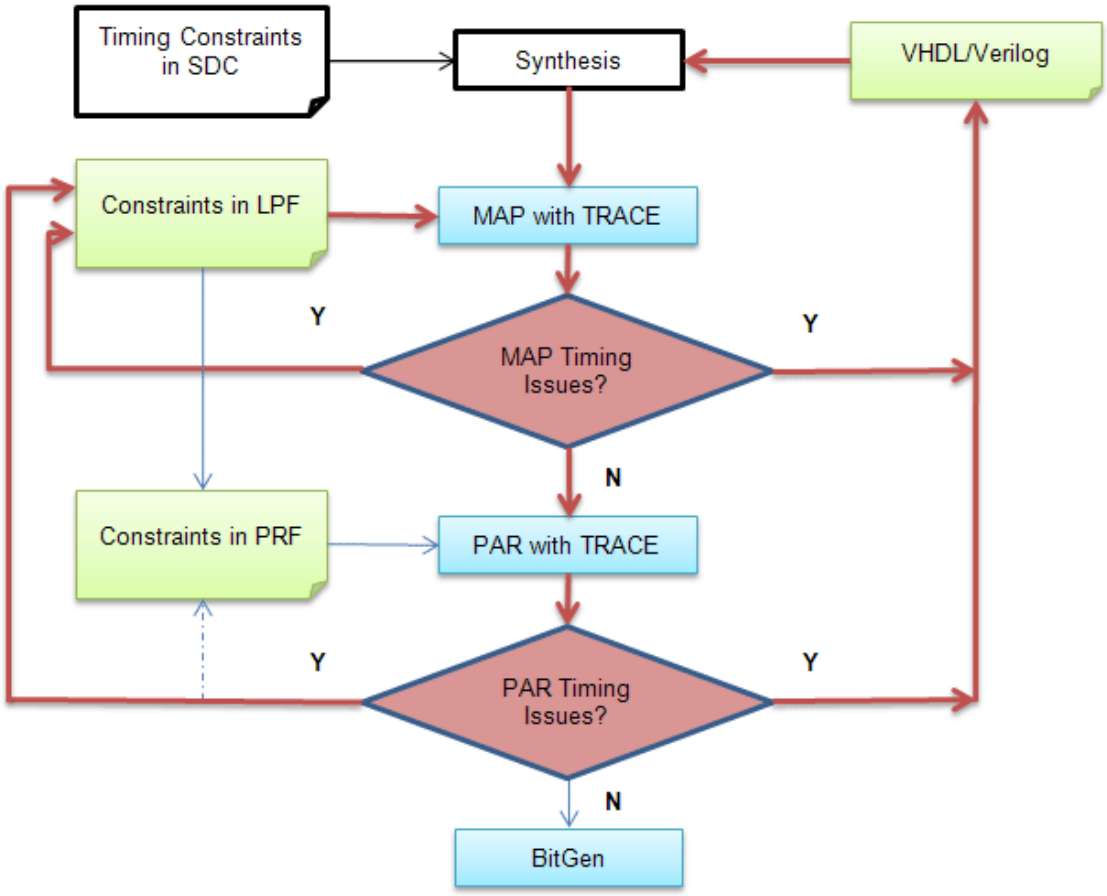
Let us take a systematic approach to address timing issues. In this section, we will bypass the synthesis process and focus on Lattice core processes such as MAP, PAR and TRACE.

As illustrated in Figure 40, we will analyze timing reports from MAP TRACE and PAR TRACE, debug and identify timing issues, fine tune the HDL or preference, and apply various techniques to improve the timing based on issues identified.

RTL Check and Modification

Before diving into the details of analyzing the timing report and debugging timing issues, which require intensive effort, you should first review your HDL code. The following items are a few things you should check and correct as

Figure 40: Resolving Timing



needed. These items tend to cause timing issues. By checking these items, you can fix timing issues before you spend time analyzing the timing reports, and you can get closer to achieving your timing.

- Shift register using distributed RAM
- Block RAM-related design not using the output register
- State machine encoding
- DSP-related functions that do not use all three registers (ECP3)
- Registering and Pipelining opportunities
- DSP block being used to implement no arithmetic functions (ECP3)
- I/O timing: to use or not to use the I/O registers (see "Using I/O Register to Improve I/O Timing" on page 38 and "Adding Delays to Input Registers" on page 39)

Analyzing the MAP TRACE Report

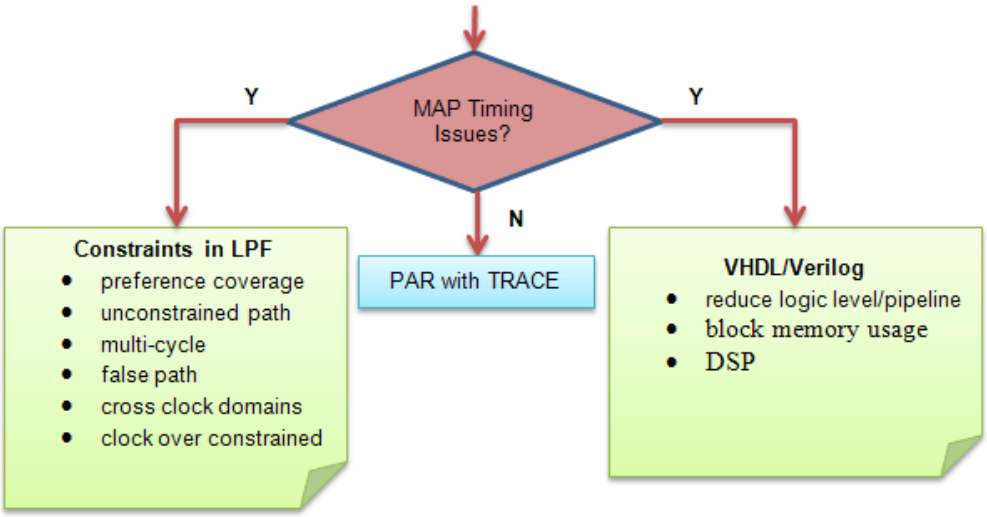
Examining the MAP TRACE report is the first step you should take. Identify any issues of timing or preference correctness, and act from there.

The MAP TRACE report can be viewed in the Diamond Report View, or you can open the report file in any text editor. The MAP TRACE report file is in your implementation directory and has the file extension .tw1.

Rule of thumb: If there is any timing issue reported by MAP TRACE, usually it is an RTL issue and you should correlate the issue and your HDL code.

You should also check your preference coverage and correctness, as illustrated in Figure 41:

Figure 41: Checking Preferences



Review Timing Constraints and Reports

Reviewing the timing constraints is performed by examining the timing constraints used and the timing results in the “Preference Summary” section in the MAP TRACE report (.tw1 file). For example:

```
Preference Summary
• FREQUENCY NET "CK38A_c" 40.000000 MHz (0 errors)
            3831 items scored, 0 timing errors detected.
Report:  148.787MHz is the maximum frequency for this
preference.
• FREQUENCY NET "CK66_keep" 72.000000 MHz (4096 errors)
            4096 items scored, 4096 timing errors detected.
Warning:  65.283MHz is the maximum frequency for this
preference.
```

You need to pay attention to every preference listed in the report, especially those that have the most timing failures. Examining the timing failures gives clues as to where coding improvements can be made in the RTL or what constraints need to be adjusted. The types of coding improvements are registering/pipelining, retiming, or simply recoding to prevent long paths. The types of constraint adjustment are constraint relaxation (if over-constrained) or constraint coverage improvement (if under-constrained) such as adding false paths, multi-cycle and MULTICYCLE for paths that cross related clock domains.

Timing Preference Coverage

Timing preference coverage can be found in the MAP TRACE report in the “Timing summary” section. For example:

```
Timing summary (Setup):
---------------
Timing errors: 18756  Score: 9623209
Cumulative negative slack: 9623209
Constraints cover 5206344 paths, 185 nets, and 176054
connections (99.6% coverage)

Timing summary (Hold):
---------------
Timing errors: 1237  Score: 823517
Cumulative negative slack: 823517
Constraints cover 5206344 paths, 177 nets, and 176050
connections (99.6% coverage)
```

Less than 95% coverage usually is insufficient and need improvement.

Unconstrained Paths

Unconstrained paths can be found in the MAP TRACE report for both setup time and hold time analysis. For example:

```
------------------------------------------------------------
                Connections not covered by the preferences
------------------------------------------------------------
     Delay                          Element                 Net
  e 0.000ns ECC_LINK_OUT_0_MGIOL.IOLDO to ECC_LINK_OUT_0.IOLDO ECC_LINK_OUT_0_c
  e 0.000ns rstn.PADDI to I_SEMF_INIT_GLOB_REG_1/I_22_0/SLICE_20648.B0 rstn_c
  e 4.765ns rstn.PADDI to reqn.PADDT rstn_c
```

You can review the list in order to improve your preference coverage or to identify any paths that should be constrained.

Note that you need to turn on “Check Unconstrained Paths” through the TRACE strategy settings (see “MAP TRACE” on page 29 and “PAR TRACE” on page 32). By default, this is turned off.

Logic Levels

Logic levels can be examined in the synthesis report, MAP TRACE report and PAR TRACE report. You should check to see whether paths exceed the timing constraints and whether logic levels are too high. For example:

This example shows that the path has 20 logic levels and exceeds the requirement by 1.430ns. The “Physical Path Details” section that follows gives more information on the failed path.

When this situation happens, you have the following options:

- Explore the synthesis/map strategy settings, without changing the HDL, to see if better results can be achieved. Possible strategy settings include pipelining and retiming.

```
Error: The following path exceeds requirements by 1.430ns
Logical Details: Cell type  Pin type       Cell/ASIC name  (clock net +/-)

Source:   FF   Q    I_TOP_PCIDMA/U_3/sgdmac_inst/engine/dst_bus_r8_0  (from CK66_keep
+)
Destination:    FF   Data in        I_0/I_RX_HDLC_CT_1/CT2/M_DWD_CNT_9  (to CK66_keep
+)

   Delay:              15.245ns  (26.0% logic, 74.0% route), 20 logic levels.
```

- Recode the HDL to see if a smaller number of logic levels can be achieved.

Other Things to Consider Before PAR

At this point, if you still do not have a good timing picture despite taking all the preceding steps, consider doing the following:

- Continue RTL modifications.
  - Continue finding registering and pipelining opportunities.
  - Change arithmetic calculation from serial to parallel.
- Optimize the state machine.
- Trade off between distributed memory and Block Memory.
- Trade off between DSP block and carry-chain logic.
- Increase you speed grade.
- Change to a larger device.

Analyzing the PAR Report and PAR TRACE Report

When there are no more issues reported by MAP TRACE, you can confidently move to the next step. As illustrated in Figure 42, this will involve more actions and decisions

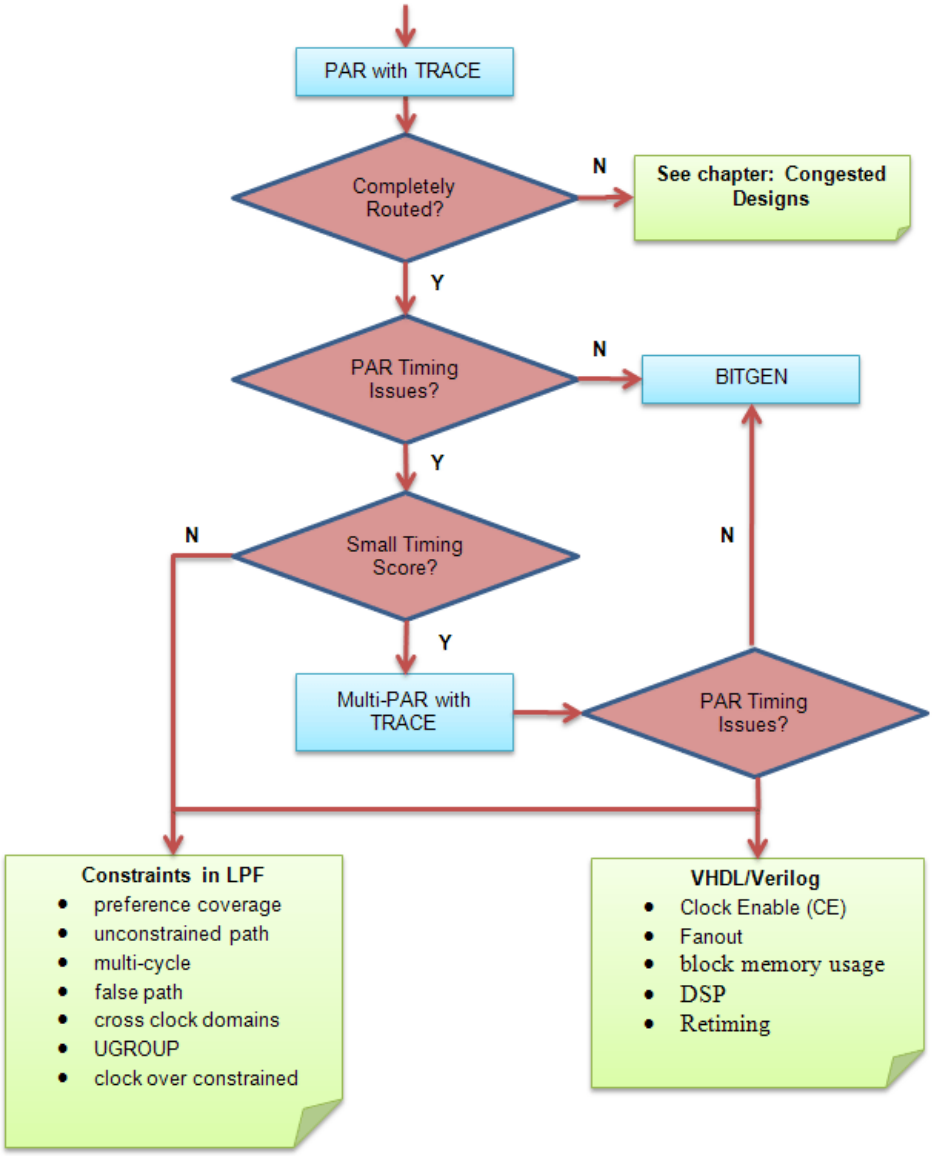
Initial PAR Assessment

If this is your first time running PAR, you should run PAR using the default PAR strategy settings and find out if there is any issue that prevents PAR from finishing successfully. One of the issues might be that you design is not completely routed. If this is the case, see the document “Congested Design.”

If you have finished PAR successfully but have timing issues, you can quickly check the setup time timing score from the “Cost Table Summary” of the PAR report:

```
Cost Table Summary
Level/       Number       Timing       Run          NCD
Cost [ncd]   Unrouted     Score        Time         Status
----------   --------     --------     -----        ------------
5_1    *     0            458          6:01:32      Complete
```

Figure 42: Analyzing PAR and TRACE



If you have a small timing score number (less than a few hundred), you should try a multi-PAR process with some PAR strategy changes (see PAR "PAR (Place & Route Design) Settings in Strategy for Timing Closure" on page 31 and "Controlling PAR" on page 83). A different strategy and multi-PAR cost table might yield better results than the initial single seeded PAR run. If a timing score of zero is achieved after a multi-PAR run, you can move to the next step of the process.

If you still have timing issues, you should analyze the PAR TRACE report, debug the timing problems, and use appropriate approaches to fix the issues.

Multi-PAR

When using multi-PAR, you should make sure that multiple PAR results are saved for later timing analysis. You can modify this through the PAR strategy settings (see section “Controlling PAR” on page 83). Saving multiple PAR results has the following benefits:

- You can run PAR TRACE on multiple PAR results; for example, the top 5 results that have the lowest timing scores. This helps you identify timing issues from multiple views with different angles. Note that failing paths usually show up in more than one result, and this helps you identify problematic areas in your design.
- You can select the best of the PAR results or the one whose timing issues can be quickly resolved.

PAR TRACE Report Analysis

By default, PAR TRACE reports ten critical paths for each timing constraint, and this should be sufficient for most designs. If your design has timing issues, it is recommended that you change the number to at least 300 in order to see the entire timing picture. To do this, change the value of “Worst-Case Paths” through the PAR TRACE strategy settings.

Note that since PAR TRACE works on the placed and routed data, it is more accurate than the MAP TRACE. The critical paths that are reported might be very different from those in the MAP TRACE report.

When analyzing the PAR TRACE report, look for the paths with timing issues. In the Diamond Report View, you can examine the “Preference Summary” section where all failing preferences are highlighted in red. You can click one of them to go to the detailed report. At this point, you might see some groups of similar failing timing paths. These groups of paths are those that you need to focus on until they no longer show up in the subsequent runs of PAR and PAR TRACE.

To help you further analyze the timing issues, you can use the Timing Analyzer View (TAV), which allows you cross-probe to the Physical View or the Floorplan View. This method also provides clues as to what the software has done in terms of far-apart placements and large fanouts, etc.

To focus on these paths, you should correlate them back to the RTL and, if applicable, make the recommended modifications that are explained in the following sections.

Clock Resources

Ideally, the number of clocks in your design should not exceed the number of clock resources available in the target device. Otherwise, general routing resources will be used for some clocks, and this will generate setup time and/or hold time violations. See “Use Primary or Secondary Clocks” on page 81.

The information about clock resource usage can be found in the PAR report. For example:

You should make sure that the general routing resources are not used for your clock. If the number of clocks exceeds the number of clock resources

```
The following 4 signals are selected to use the primary clock routing resources:
 clk_pll_c (driver: PLL_soft_wb_inst/PLL_inst0/PLLInst_0, clk load #: 400)
......

The following 6 signals are selected to use the secondary clock routing resources:
 clk_c (driver: OSCH_inst, clk load #: 393)
   ......

WARNING - par: Signal "clk_c" is selected to use Secondary clock resources; however
its driver comp "clk" is located at "N3", which is not a dedicated pin for connecting
to Secondary clock resources.  General routing has to be used to route this signal,
and it may suffer from excessive delay or skew.
```

available, reconsider your clock resource usage strategy (see “Clock Assignment” in the “Design Planning” chapter), or consider moving to a device with more clock resources. Otherwise, correct the constraint and make sure that general routing resources are not used.

Reduce Register Loads

If a failing path starts with a source register that drives more than one load and has large delays, you should duplicate the source register to reduce its load. The following is a typical example:

```
 Name     Fanout    Delay (ns)          Site               Resource
REG_DEL      ---      0.285    R89C27C.CLK to    R89C27C.Q0 U_core/SLICE_29896 (from
tel_clk_155)
ROUTE          3      4.914     R89C27C.Q0 to    R91C45D.D1 U_core/rx_tu_mode_2
CTOF_DEL     ---      0.180     R91C45D.D1 to    R91C45D.F1 U_core/ddwr_stm8/SLICE_36645
ROUTE          1      0.312     R91C45D.F1 to    R91C45D.D0 U_core/payload_we17_tz_tz
CTOF_DEL     ---      0.180     R91C45D.D0 to    R91C45D.F0 U_core/SLICE_36645
ROUTE          5      0.741     R91C45D.F0 to    R89C48D.B0 U_core/payload_we17
CTOF_DEL     ---      0.180     R89C48D.B0 to    R89C48D.F0 U_core/SLICE_38198
ROUTE          4      0.861     R89C48D.F0 to    R82C50A.C0 U_core/nxt_bcnt_0_sqmuxa_1
CTOF_DEL     ---      0.180     R82C50A.C0 to    R82C50A.F0 U_core/SLICE_38332
ROUTE         50      1.986     R82C50A.F0 to   R75C72B.CE U_core//payload_we_1_sqmuxa_1
(to tel_clk_155)
                    --------
                      9.819   (17.3% logic, 92.7% route), 5 logic levels.
```

This path starts with the register “rx_tu_mode” with fanout number 3 and has 4.914ns routing delay, which contributes more than 50% of the total delay. To address this issue, you should add the following synthesis directive to the HDL to limit the number of loads:

```
reg rx_tu_mode /* synthesis syn_maxfan = 1 */
```

For your actual designs, depending on the value of the desired number of loads, the appropriate number of registers will be generated.

Experience shows that if the value of the desired number of loads is too small, it could cause an unintended effect: that the load of the input source of this register increases. You should check to see whether such an effect occurs when this modification is used.

Using Registers Instead of Distributed Memory

If a failing path has a larger delay due to a shift register implemented using distributed memory, you should change this so that registers are used instead of distributed memory. For example:

```
Name     Fanout   Delay (ns)          Site               Resource
REG_DEL     ---     0.243   R51C141C.CLK to    R51C141C.Q0 SLICE_15 (from CLK_c)
ROUTE        12     0.760    R51C141C.Q0 to    R51C143A.A0 tmp1_0
CTOF_DEL    ---     0.147    R51C143A.A0 to    R51C143A.F0 B_1_CR7_ram_0/SLICE_9
ROUTE         1     2.725    R75C143A.F0 to    R75C142C.B1 B_1_4
C1TOFCO_DE  ---     0.277    R75C142C.B1 to   R75C142C.FCO SLICE_3
ROUTE         1     0.000   R75C142C.FCO to   R75C143A.FCI Y_1_cry_4
FCITOF1_DE  ---     0.177   R75C143A.FCI to    R75C143A.F1 SLICE_4
ROUTE         1     0.811    R75C143A.F1 to IOL_R52A.OPOSA Y_1_6 (to CLK_c)
                  --------
5.140(17.9% logic, 83.1% route), 4 logic levels.
```

You should add “syn_srlstyle” synthesis directive in your HDL:

```
reg [7:0] A_d1, A_d2, A_d3, A_d4 /* synthesis syn_srlstyle = “registers” */;
reg [7:0] B_d1, B_d2, B_d3, B_d4 /* synthesis syn_srlstyle = “registers” */;
always @(posedge CLK)
begin
    A_d1 <= A;
    B_d1 <= B;
    A_d2 <= A_d1;
    B_d2 <= B_d1;
    A_d3 <= A_d2;
    B_d3 <= B_d2;
    A_d4 <= A_d3;
    B_d4 <= B_d3;
end
```

Using Block RAM Output Register

If you design has block RAM, check the PAR TRACE report to make sure that the output registers are used. Otherwise, timing issues could arise. In the follow PAR TRACE report example, 2.484ns delay is quite large without the use of output registers:

You should add the following parameters to the HDL:

- For instantiated PMI

  ```
  parameter pmi_regmode = "reg”
  ```

- For Instantiated EBR memory

  ```
  parameter  REGMODE_A = "REG";
  parameter  REGMODE_B = "REG";
  ```

Disable Using DSP Blocks

When a non-DSP function is implemented in a DSP block, it can cause larger delays, as shown in the following PAR TRACE report example:

```
Name     Fanout   Delay (ns)          Site               Resource
C2Q_DEL      ---     2.484 EBR_R49C2.CLKA to EBR_R49C2.DOA3 U_core/mem_mem_0_8 (from
sys_clk_125)
ROUTE          1     1.730 EBR_R49C2.DOA3 to     R35C15C.C1 U_core/mp_fifo_dout_1_35
CTOF_DEL     ---     0.180     R35C15C.C1 to     R35C15C.F1 U_core/SLICE_13772
ROUTE          3     0.715     R35C15C.F1 to     R33C27B.D1 U_core/mp_fifo_dout_2_35
CTOF_DEL     ---     0.180     R33C27B.D1 to     R33C27B.F1 U_core/SLICE_38961
ROUTE          6     0.661     R33C27B.F1 to     R31C33D.D1 U_core/mp_eop_out_1
CTOF_DEL     ---     0.180     R31C33D.D1 to     R31C33D.F1 U_core/SLICE_35423
ROUTE          1     0.370     R31C33D.F1 to     R31C33D.B0 U_core/
un1_abnormal_empty20_1_m2
CTOF_DEL     ---     0.180     R31C33D.B0 to     R31C33D.F0 U_core/SLICE_35423
ROUTE         32     1.310     R31C33D.F0 to     R26C43B.CE U_core/
un1_abnormal_empty20_1_m4 (to sys_clk_125)
                   --------
7.990(40.1% logic, 59.9% route), 5 logic levels.
```

```
 Name     Fanout   Delay (ns)          Site               Resource
REG_DEL      ---     0.243    R47C42C.CLK to     R47C42C.Q0 SLICE_100 (from CLK_c)
ROUTE          1     1.550     R47C42C.Q0 to *18_R34C51.B14 B_1_32
BYPASS_DEL   ---     0.220 *18_R34C51.B14 to *_R34C51.ROB14 Y_wire_1_pt
ROUTE          1     0.000 *_R34C51.ROB14 to *54_R34C54.A32 Y_wire_1_pt_ROB14
PD_DEL       ---     4.870 *54_R34C54.A32 to *54_R34C54.R40 Y_wire_1_40_0
ROUTE          1     1.283 *54_R34C54.R40 to     R23C59C.M0 Y_wire_40 (to CLK_c)
                   --------
                     8.166(24.1% logic, 75.9% route), 3 logic levels.
```

There is a large delay (4.870ns) through the DSP block. Since the actual function in the HDL is a simple multiplication and using a DSP block is unnecessary, you should use the following directive to prevent a specific logic from using the DSP block:

```
wire [40:0] Y_wire  /* synthesis syn_multstyle = "logic" */;
assign Y_wire = A_d4 * B_d4;
```

Packing Related Logic

Packing unrelated logic together is usually practical for a non-high speed design, but this might not be a good choice for a large design running at high speed. Doing this can cause long paths between two or more sub-blocks, and long paths can introduce timing problems. This issue is seen in the following example:

This example section of the PAR TRACE report shows that the routing part contributes more than 87% of the total delay. Further study shows that the sum of the routing delay to and from “SLICE_23827” is 4.551ns out of 6.715ns total delay. Also notice that unlike all other sub-paths,

```
 Name   Fanout  Delay (ns)      Site               Resource
REG_DEL    ---     0.243   R81C10C.CLK to R81C10C.Q1 U_top/module_A/submodule_B/SLICE_2
(from sys_clk)
ROUTE       3      0.516   R81C10C.Q1  to R81C12D.D0 U_top/module_A/submodule_B/
col_count_4
CTOF_DEL   ---     0.147   R81C12D.D0  to R81C12D.F0 U_top/module_A/submodule_B/
SLICE_25670
ROUTE       1      0.255   R81C12D.F0  to R81C12D.C1 U_top/module_A/submodule_B/
m27_e_s_10_1
CTOF_DEL    ---      0.147     R81C12D.C1 to     R81C12D.F1 U_top/module_A/SLICE_25670
ROUTE         1      0.562     R81C12D.F1 to     R81C14A.A1 U_top/module_A/m27_s_10_1
CTOF_DEL    ---      0.147     R81C14A.A1 to     R81C14A.F1 U_top/module_A/SLICE_9269
ROUTE         9      2.602     R81C14A.F1 to     R57C49C.D1 N_150822
CTOF_DEL    ---      0.147     R57C49C.D1 to     R57C49C.F1 SLICE_23827
ROUTE         1      1.949     R57C49C.F1 to     R75C25C.M1 U_top_module_A_dout_9_0_i_4
(to sys_clk)
                   --------
                     6.715   (12.4% logic, 87.6% route), 5 logic levels.
```

"SLICE_23827" stands alone and is not within a module or a sub-module. This is a typical symptom where unrelated logic is packed together.

To fix this issue, you should add an HGROUP or UGROUP to "U_top/module_A" in your HDL, or add a UGROUP preference to it. For example:

```
module module_A(CLK, A, B, Y)/* synthesis UGROUP="MODULE_A" */;
```

After the modification, you need to run MAP again. This will ensure that unrelated logic is not packed together by MAP.

When MAP finishes successfully, it propagates UGROUP constraints to the generated PRF file that will be used to drive PAR. If grouping is no longer desired, and you want to allow PAR to freely place the elements in the group instead of trying putting them all in one SLICE or closed slices, you can manually edit the PRF file to remove the group.

In this example, the UGROUP added is "MODULE_A." In the generated PRF file, you should see a few lines similar to the following example:

```
PGROUP "MODULE_A"
COMP "U_core/module_A/SLICE_0"
COMP "U_core/module_A/SLICE_1"
......
COMP "U_core/module_A/SLICE_1000"
PGROUP "PGROUP_X"
```

You should remove the line "PGROUP "MODULE_A"" toward the last line of the group. In this example, the last line is the one containing "SLICE1000."

Fixing Clock Enable (CE)

The enable pin on a PFU register usually has larger delays than the data pins. Look at the following example from part of the PAR TRACE report:

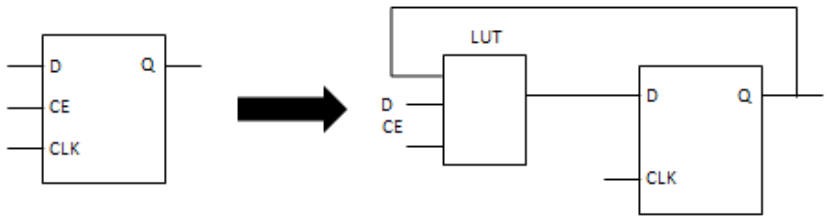
```
6.473ns physical path delay oam_ptp_func_instance/SLICE_19092 to
oam_ptp_func_instance/ptp_func_instance/gmii_rx_1588_0/SLICE_34361 exceeds
      6.410ns delay constraint less
      0.000ns skew and
      0.253ns CE_SET requirement (totaling 6.157ns) by 0.316ns

 Physical Path Details:
   Name  Fanout  Delay (ns)        Site              Resource
REG_DEL   ---    0.243   R32C67A.CLK to R32C67A.Q1 oam_ptp_func_instance/SLICE_19092
(from sys_clk125m_c)
ROUTE      28    3.155   R32C67A.Q1 to R20C126B.M0 oam_ptp_func_instance/
gmii_rx_vlantag_ind_dly1_4
MTOOFX_DEL ---    0.186   R20C126B.M0 to R20C126B.OFX0 oam_ptp_func_instance/
ptp_func_instance/gmii_rx_1588_0/un1_ptp_pack_pulse_0_sqmuxa_3_0/SLICE_40297
ROUTE       1    0.341  R20C126B.OFX0 to R21C126A.D1 oam_ptp_func_instance/
ptp_func_instance/gmii_rx_1588_0/un1_ptp_pack_pulse_0_sqmuxa_3_0
CTOF_DEL  ---    0.147  R21C126A.D1 to  R21C126A.F1 oam_ptp_func_instance/
ptp_func_instance/gmii_rx_1588_0/SLICE_28586
ROUTE       6    2.401  R21C126A.F1 to  R23C72B.CE oam_ptp_func_instance/
ptp_func_instance/gmii_rx_1588_0/message_type_0_sqmuxa_i (to sys_clk125m_c)
                  --------
                    6.473   (8.9% logic, 91.1% route), 3 logic levels.
```

The routing delay in this example contributes 91.1% of the total. The “CE_SET requirement” statement shown in the report gives a clue that this is a Clock Enable delay issue. To fix it, set “syn_useenables” synthesis directive to 0. For example:

```
reg Myreg /* synthesis syn_useenables=0 */
```

The effect of this directive is to convert CE to A/B/C/D pin in a SLICE, as illustrated in Figure 43.

Figure 43: Converting CE to A/B/D pin in a SLICE



Final PAR

With all or most of the critical timing issues identified and addressed, you should have a final PAR run with increased PAR effort using the following options. These options can be modified through the PAR strategy settings. See “Controlling PAR” on page 83.

- Routing method: NBR
- Congestion driven placement: Yes

- Congestion driven router: Yes
- Placement iteration: 10 to 30
- Placement effort: 5
- Routing passes: 10
- Path Based Placement: Yes

You should carefully examine the results to see if every iteration provides significant improvement. If this is not the case, you might have reached a point where a serious design review needs to be performed.

Architecture Specific Topics

Hardware Details

Understanding hardware details should help you fully utilize the hardware capability or avoid unnecessary timing problems caused by improper use. The following sections explain some of these important details.

Embedded Block RAM (EBR) Routing Differences

Routing from EBR to top slices could be different than routing to bottom slices.

Different LUT Pins' Delays

Typically, C and D inputs to a LUT are faster than the A and B inputs to the same LUT.

The Enable Pin on PFU Registers

In general, the enable pin has a larger delay than the data pins.

Clock Boosting

Clock boosting is the deliberate introduction of clock skew on a target flip-flop to increase the setup margin. The automated clock-boosting tool attempts to meet setup constraints by introducing delays to as many target registers as needed to meet timing. In effect, it borrows register hold margins to meet register setup timing. Clock boosting is accomplished through the following features:

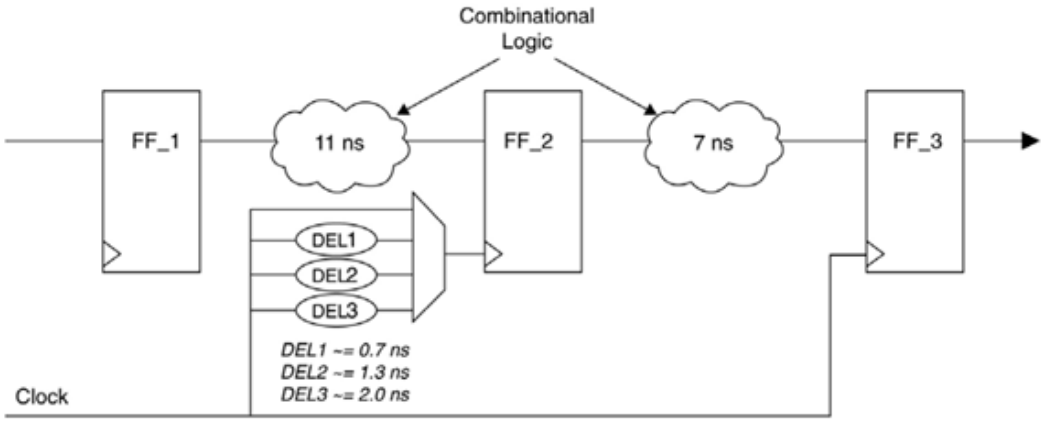
- For the ECP3 device family, this is achieved by rerouting the clock through the switch matrix to gain some delay on the destination clock. It introduces skew only to the destination registers, not on the clock network.

- For certain device families, every programmable flip-flop in the device has programmable delay elements before clock inputs for this purpose.
  - A 4-tap delay cell structure in front of the clock port of every flip-flop in the device (includes I/O flip-flops)
  - The ability to borrow clock cycle time from one easily met path and give this time to a difficult-to-meet path

Clock boosting is typically most useful in designs that are only missing timing on a few paths for one or two preferences. If the design is missing timing by over a few nanoseconds on any given path, clock boosting cannot schedule skew in a way that eliminates enough timing to make the critical preference. Clock boosting run times can be shortened by using a preference file that contains only the failing preferences.

The example illustrated in Figure 44 shows two register-to-register transfers that both need to meet the 10-ns period constraint. By using the DEL2 delay cell to delay the clock input on flip-flop FF_2, the first register transfer makes its period constraint with a new minimum period of approximately 9.7 ns, and the second register transfer makes its period constraint by approximately 8.3 ns. The D1, D2, and D3 delays shown vary, depending on the performance grade and FPGA device family.

Figure 44: Clock Boosting



Other important considerations on the practicality of using clock boosting:

- Some circuits show much improvement, but others show no gain. Clock boosting results are very design-dependent.
- Clock boosting uses minimum delay values that have not yet been validated at the system level.
- Automatic clock boosting identifies skew and hold-time issues. However, after clock boosting is performed, it is recommended that you run PAR TRACE hold analysis to make sure that there is no hold violation.