Efinity® Timing Closure User Guide

UG-EFN-TIMING-v7.0
November 2024
www.efinixinc.com

Contents

Introduction.......... 5
- About Constraints.......... 5
- Tools for Exploring Timing.......... 6

SDC File Overview.......... 8
- About SDC Files.......... 8
- Create an Empty SDC File.......... 9
- Add an SDC File to Your Project.......... 9
- Using Multiple SDC Files.......... 9
- Efinity Files You Use to Create Constraints.......... 10

Constraining Clocks.......... 11
- Defining Clocks.......... 11
    - Using the create_clock Constraint.......... 12
    - Using the create_generated_clock Constraint.......... 14
    - Virtual Clocks.......... 14
- Clock Latency.......... 16
    - GPIO Clock Latency.......... 16
    - PLL Local Feedback Clock Latency.......... 18
    - PLL Core Feedback Clock Latency.......... 20
    - PLL External Feedback Clock Latency.......... 22
    - PLL Cascading Clock Latency.......... 24
- Clock Relationships.......... 26
    - Setting Constraints for Unrelated Clocks.......... 26
    - Using the set_clock_groups Constraint.......... 26
    - Using the set_false_path Constraint.......... 27
    - Clock Synchronizers.......... 28
    - Metastable Synchronizer Circuit.......... 29
- How to Set Clock Uncertainty.......... 29

Constraining I/O.......... 30
- Constraining Synchronous Inputs and Outputs.......... 30
- Constraining Unsynchronized Inputs and Outputs.......... 33
    - Input Receive Clock Delay.......... 35
    - Output Receive Clock Delay.......... 36
    - Input Forward Clock Delay (GPIO clkout).......... 37
    - Output Forward Clock Delay (GPIO clkout).......... 39
    - Input Forward Clock Delay (GPIO output).......... 40
    - Output Forward Clock Delay (GPIO output).......... 42

Timing Exceptions.......... 44
- Example: Clock-to-Clock Path with Control.......... 44
- Understanding False Paths.......... 45
- Understanding Min and Max Delays.......... 45
- Understanding Multicycle Constraints.......... 47
    - Shifted Capture Window.......... 47
    - Shifted and Widened Window.......... 47
    - Constraints between Fast and Slow Clocks.......... 48

SDC Warnings.......... 49

Common Mistakes.......... 49

SDC Tips and Tricks.......... 51
- SDC Syntax.......... 51
- Wildcard Commands.......... 51
- Regular Expressions.......... 51
- Inverted Clocks.......... 52

Square Brackets in Clock Names....52

SDC Constraints (Alphabetical)....53
create_clock Constraint....53
create_generated_clock Constraint....54
get_fanouts Constraint....55
get_fanins Constraint....56
set_bus_syntax_mode Command....57
set_clock_groups Constraint....58
set_clock_latency Constraint....58
set_clock_uncertainty Constraint....59
set_false_path Constraint....59
set_input_delay and set_output_delay Constraints....60
set_max_delay and set_min_delay Constraints....61
set_multicycle_path Constraint....61
-through Option....62

Object Specifiers....62

SDC Examples....64
Example: Dynamic Multiplexers and create_clock -add....64
Example: FPGA Forwarded Clock....65
Example: Generated Clock with Clock Multiplexer....66
Example: Soft SERDES....67
Example: Disable Impossible Paths....68

Interpreting Timing Results....69
Clock Frequency Summary....69
Clock Relationship Summary....70
Critical Paths....70

Constraining Logic and Routing Manually (Beta)....72
Tiles....72
Working with Primitives....74
Enabling Manual Assignments....75
Assignment Rules....76
Creating a Location Assignment File....76
Constraining Routing Manually (Titanium Only, Beta)....78
Routing Constraint Flow....78
Enabling Routing Constraints....78
Generate .rcf Template....79
Creating a Routing Constraint File....79
Best Practices for Constraining Routing....80
Example Flow....81

Methods for Closing Timing....82
Synthesis Options....82
Handling High Fanouts....85
Place-and-Route Options....87
Beneficial Skew....88
Sweeping Script....88
Optimization Sweeping....89
Seed Sweeping....89
Closing Timing with High DSP Block Utilization....91

Tcl Console....92
General Commands....93
delete_timing_results Command....93
get_available_timing_model Command....93
get_timing_model Command....93
read_sdc Command....93
reset_timing Command....93
set_timing_model Command....93
write_sdc Command....94

Report Commands....94
check_timing Constraint....94
report_clocks Command....94
report_path Command....95
report_sdc Constraint....96
report_timing Command....97
report_timing_summary Command....97
Tcl List Functions (Alphabetical)....98
lappend....98
lassign....98
lindex....98
linsert....99
llength....99
foreach_in_collection....99
foreach....99
lrange....100
lreplace....100
lreverse....100
lsearch....100
lsort....101
Tcl Script Examples....101
Identify Pins with a Regular Expression....101
Create Clocks with Different Periods....102
Generate a Report with get_fanouts....102
Generate Report with a Subset of Clocks....102
Use Variable for Min/Max Delay Calculation....103
Command-Line Tcl Console....103

Appendix....104
About the <*project*>.pt.sdc File....104
About the <*project*>.pt_timing.rpt File....106

Where to Learn More....108

Revision History....109

Introduction

Closing timing is an important part of the design process. The Efinity® software includes tools and reports to help you understand your design's timing requirements and let you adjust settings to close timing. This document explains how to set timing constraints using a Synopsys Design Constraints (**.sdc**) file, and discusses synthesis, placement, and routing options to customize the Efinity® flow. Additionally, it describes how to use the Tcl Console and Tcl commands to explore timing and customize your SDC file.

You can explore timing with just an RTL design and an SDC file. This step helps you understand your design's timing requirements in general terms. If you have not built an interface yet, the placer auto-assigns the interface signals, which you can use to set constraints. After you build your interface, the interface signals are constrained according to the assignments you made in the Interface Designer.

Note: This document includes information that was previously provided in AN 008: Setting Trion Timing Constraints in the Efinity Software.

About Constraints

The Efinity® software supports the Synopsys Design Constraints format for specifying timing constraints. The software validates the timing performance of your design's core logic using industry-standard constraint, analysis, and reporting methodology. During compilation, the software generates a timing analysis report. The pins, nets, and ports used with SDC constraints refer to the post-synthesis netlist.

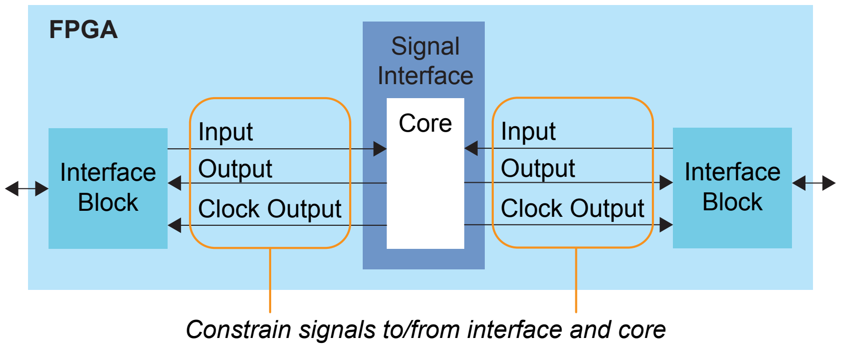
Trion®, Topaz, and Titanium FPGAs feature interface blocks—I/O logic and buffers, I/O banks, PLLs, etc.—that connect the core logic to the package pins. You use the Efinity® Interface Designer to configure these interface blocks for your design. After you configure these blocks, you generate a constraint template file (<*project*>**.pt.sdc**) that you use as the basis for your design's SDC file. You can also refer to report files for the interface blocks, which you can view in the Results tab under the Efinity® Dashboard.

- For synchronous (registered) interfaces, the template defines clocks and sets input and output delays for your design. You simply copy and paste the relevant lines from the SDC template file to your own SDC file and adjust the timing as needed.
- For non-synchronous (unregistered) interfaces, you need to determine the interface timing and board timing and add those to your core settings.

Important: Unlike traditional FPGAs, with Trion®, Topaz, and Titanium FPGAs you make timing constraints at the core level, not the interface or package level.

Efinix recommends that you use registered interfaces as much as possible to simplify the SDC constraints you need.

Figure 1: Set Constraints at the Core Level



Note: Refer to SDC Constraints (Alphabetical) on page 53 for a list of supported SDC constraints and object specifiers.

Tools for Exploring Timing

You use static timing analysis (STA) to measure the timing performance of your design. The software generates a timing report based on the design's place and route results and the project's SDC file. The software provides several tools for viewing and cross-probing timing results:

- The Timing Browser helps you explore your design's critical paths and the cells of those paths.
- The Floorplan tool shows the locations of the paths and cells in the fabric.
- The Tcl Console helps you analyze and explore timing.

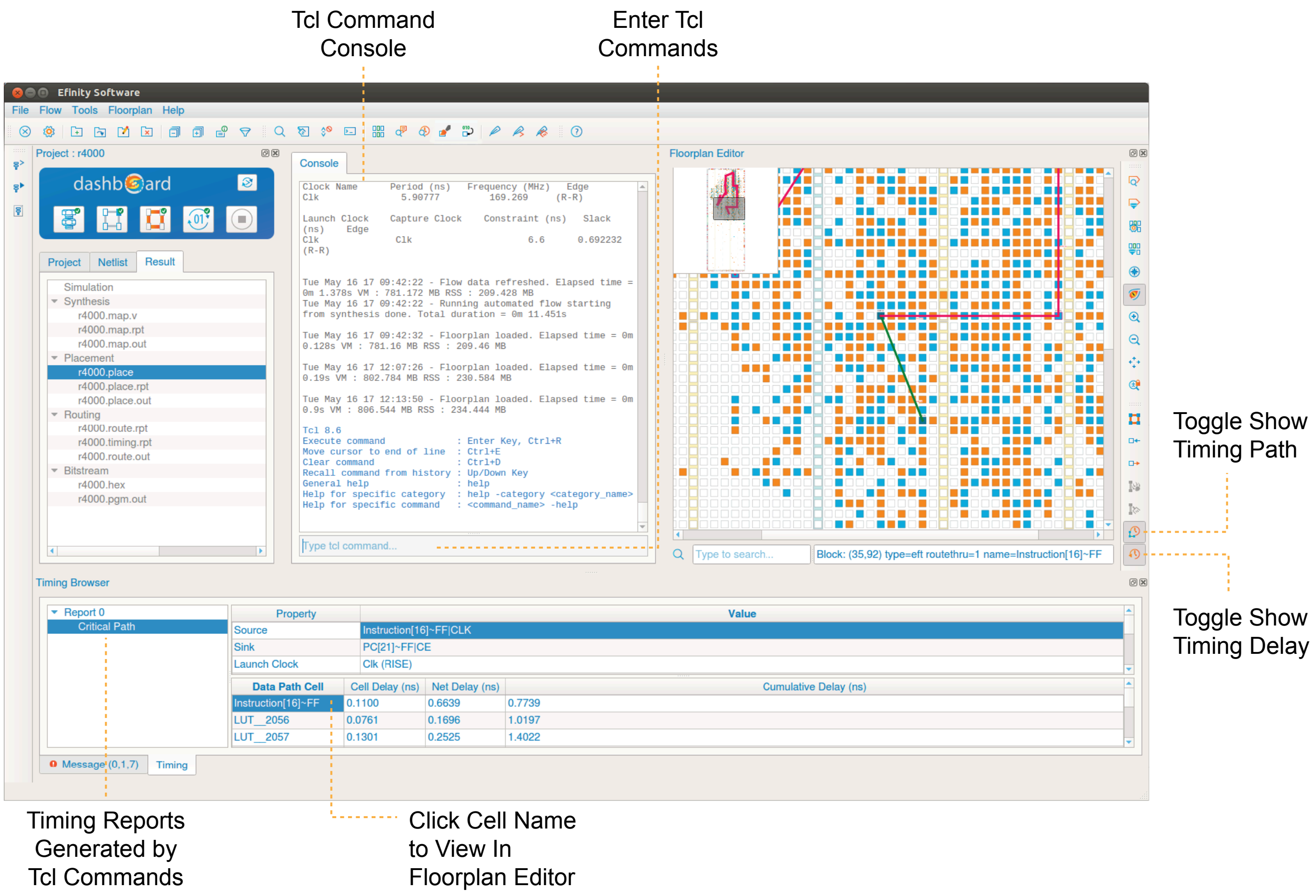
The Efinity software uses a Tcl interpreter to process SDC constraints and to support timing analysis. The Tcl Console is an interactive shell that you use to execute Tcl commands. Refer to Tcl Console for details on using the console in the Efinity GUI or in a terminal.

The software displays Tcl reports in the Timing Browser.

- Click on the report name to view details.
- Click on cell names under **Data Path Cell** to view the location of the cell in the Floorplan Editor.
- Turn on Show Timing Path or Show Timing Delay in the Floorplan Editor to see the path and delay for a particular cell.

Note: The place-and-route data for your project has to be loaded for you to use the Timing Browser and Floorplan tool. Refer to "Auto-Load Place-and-Route Data" in the Efinity Software User Guide for instructions on loading it.

Figure 2: Using the Timing Browser



```
Clock Name      Period (ns)   Frequency (MHz)   Edge
Clk               5.90777         169.269     (R-R)

Launch Clock    Capture Clock    Constraint (ns)   Slack
(ns)    Edge
Clk              Clk                      6.6      0.692232
(R-R)


Tue May 16 17 09:42:22 - Flow data refreshed. Elapsed time =
0m 1.378s VM : 781.172 MB RSS : 209.428 MB
Tue May 16 17 09:42:22 - Running automated flow starting
from synthesis done. Total duration = 0m 11.451s

Tue May 16 17 09:42:32 - Floorplan loaded. Elapsed time = 0m
0.128s VM : 781.16 MB RSS : 209.46 MB

Tue May 16 17 12:07:26 - Floorplan loaded. Elapsed time = 0m
0.19s VM : 802.784 MB RSS : 230.584 MB

Tue May 16 17 12:13:50 - Floorplan loaded. Elapsed time = 0m
0.9s VM : 806.544 MB RSS : 234.444 MB

Tcl 8.6
Execute command              : Enter Key, Ctrl+R
Move cursor to end of line   : Ctrl+E
Clear command                : Ctrl+D
Recall command from history  : Up/Down Key
General help                 : help
Help for specific category   : help -category <category_name>
Help for specific command    : <command_name> -help
```

Block: (35,92) type=eft routethru=1 name=Instruction[16]~FF

Property	Value
Source	Instruction[16]~FF|CLK
Sink	PC[21]~FF|CE
Launch Clock	Clk (RISE)

Data Path Cell	Cell Delay (ns)	Net Delay (ns)	Cumulative Delay (ns)
Instruction[16]~FF	0.1100	0.6639	0.7739
LUT__2056	0.0761	0.1696	1.0197
LUT__2057	0.1301	0.2525	1.4022

Note: Refer to Tcl Console on page 92 for more information on available commands. For help on available Tcl commands, type `help -category` *<sdc or timing>* in the Tcl Command Console.

SDC File Overview

Generally, the steps you follow when creating a new SDC file are:

1. Create an empty SDC file.
2. Add the file to your project. (You can also add multiple files.)
3. Have the <*project*>.pt.sdc and <*project*>.pt_timing.rpt files ready so you can use them when creating constraints.
4. Identify clocks and their relationships.
5. Identify I/O constraints.
6. Identify any timing exceptions.
7. Debug your SDC constraints.

The following sections go over these steps in detail with examples.

About SDC Files

An SDC file is simply a text file with one constraint per line; however, you need to keep some rules in mind when creating it:

- The order of the constraints in the SDC file is important. If there are dependencies between any of the constraints, you must ensure that you have written them in the correct order for them to be valid.
- If a constraint has incorrect syntax, the software ignores it and issues a warning message.
- For some constraints, the argument order is important for the constraint to be valid.
- The minimum content required in an SDC file is a `create_clock` constraint. You should *always* set a clock constraint—even if it is a virtual clock—whenever you create an SDC file.

Important: SDC is case sensitive. If you are using VHDL, which is not case sensitive, be careful when declaring net names. The Efinity software converts all names to lowercase letters during synthesis. Therefore, the SDC should use lowercase letters not mixed case or uppercase.

If you do not define an SDC file, the software defaults to creating clocks with a period of 1 ns for every clock source in your design and does not constrain any I/O pins. It assumes that all of the clocks it finds are related. The Efinity® timing analyzer then identifies the critical path based on this default constraint.

Constraint Order

First, define the clocks and other timing assertions in this order:

1. Primary clocks
2. Virtual clocks
3. Generated clocks
4. Clock groups
5. Input and output delays

Then, define any timing exceptions, in this order:

1. False paths
2. Maximum and minimum delays
3. Multicycle paths

Create an Empty SDC File

You can use the Efinity® Code Editor or any text editor to create an SDC file and save it into your project directory.

If you are working with a new project, start by creating an empty SDC file. Then, copy and paste the Interface Designer-generated SDC constraints into this empty file and modify them to meet your timing requirements.

If you are porting an existing design to Efinix FPGAs, you may already have an SDC file. You may still want to start with an empty SDC file, copy the Interface Designer-generated constraints to the new file and modify them as needed, and then add in any additional constraints from your existing SDC file.

Add an SDC File to Your Project

Add one or more SDC files to your project using these steps:

1. Choose **File > Edit Project**.
2. Click the **Design** tab.
3. Click the Add SDC file button next to the **SDC** box.
4. Browse to the file you created and click **Open**.
5. Click **OK**.

If you add multiple SDC files, the software processes them in the order shown in the table.

Tip: As a shortcut, in the Project pane you can right-click **<project name>** > **Constraint** to pop-up a context-sensitive menu. Choose **Add** if you already have an SDC file or **Create** if you want to create a new empty file.

Using Multiple SDC Files

The Efinity software v2023.1 and higher allows you to use multiple SDC files in your project. You add additional SDC files in the Project Editor. SDC files are processed in the order listed.

During compilation, the software reads the SDC files in order starting from the first listed file and continuing to the next one(s). The same constraint order rules apply to the SDC file list. The software displays messages about the SDC file(s) in the Console. The software reads the files at the beginning of the routing stage. If the software detects any errors in the SDC file(s), it shows the error, file name, and line number.

In the Efinity software v2024.2 and higher you can use the `set sdc_list` Tcl command in your top-level SDC file to reference additional SDC files outside of your project.

```
set sdc_list { <file> <file> <file> ... }
```

For example, use the following Tcl syntax in the top-level SDC file to read the referenced files **file1.sdc**, **file2.sdc**, and **file3.sdc**:

```
set sdc_list { file1.sdc file2.sdc file3.sdc }

foreach f $sdc_list {
    source $f
}
```

The software reads the SDC files in the order you have listed in the `set sdc_list` command. Ensure that you list them in the correct order.

Important: Because they are not added to your project, the Efinity software does not know about the referenced SDC files and cannot track changes to them. For example, if you change an SDC file that is in your project, the Efinity software recognizes there has been a change and prompts you to re-compile. However, if you change a referenced SDC file, the software does not recognize that a change happened and does not prompt a re-compile. This situation could cause unintended effects. Therefore, be careful when using referenced SDC files and always re-compile when you modify them.

Efinity Files You Use to Create Constraints

When you generate constraints in the Interface Designer or compile your project, the Interface Designer generates the <*project*>**.pt.sdc** and <*project*>**.pt_timing.rpt** files in the **outflow** directory. You use these files as a reference when you create your SDC file.

- <*project*>**.pt.sdc** has timing constraints for the design's interface that you build with the Interface Designer.
- <*project*>**.pt_timing.rpt** is a timing report that shows the timing details for the design's interface that you built with the Interface Designer. The report groups the timing information by block type.

This following sections explain how to use these files to create your SDC.

For more information on these files, refer to:
- <*project*>**.pt.sdc**
- <*project*>**.pt_timing.rpt**

For more information on using the Interface Designer, refer to:
- Titanium Interfaces User Guide
- Trion® Interfaces User Guide

Constraining Clocks

The first task when building an SDC file is to define your design's clocks and their relationships. You set constraints to define the clocks and any relationships they have to each other. Then, you can constrain the I/O pins relative to each clock as needed. The following sections explain the theory behind defining clocks and their relationships.

Learn more: The following application notes provide information on clock and reset guidelines:
AN 040: Clocking Source-Synchronous Designs
AN 042: Working with PLLs
AN 044: Aligning LVDS Clock and Data
AN 046: Reset Guidelines for Efinix FPGAs

Defining Clocks

Clock sources can come from interface blocks like PLLs or oscillators, or they can come from your board to the core through GPIO pins. You define and identify clocks using the `create_clock` and `create_generated_clock` constraints.

The `create_clock` constraint defines a real or virtual clock with a specific duty cycle and period (ns). Each target can have multiple clocks associated with it.

Example: Define a Clock

This constraint creates a clock, `clk1`, with a period of 10 ns:

```
create_clock  -period 10  -name clk1 [get_ports clk1]
```

Tip: As you may remember from physics class, the clock period is the inverse of the frequency (T = 1/f). So if you want to specify the period in ns for a 50 MHz clock frequency, you use this calculation:
T = (1/50 MHz) * 1000 Hz/MHz = 20 ns

The `-waveform` option lets you define the clock's rising and falling edges.

Example: Define a Clock with a Waveform

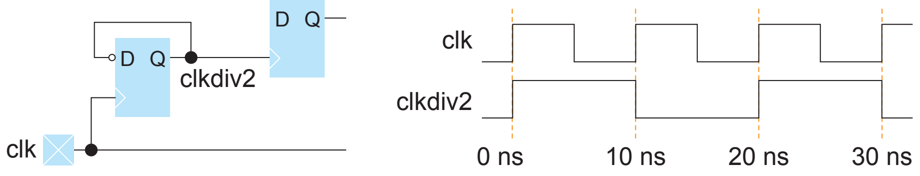
This example defines a clock with a 10 ns period and 50/50 duty cycle, but the first rising clock edge is phase shifted 25% to start at 2.5 ns.

```
create_clock -period 10.00 -waveform {2.50 7.50} -name clk1 [get_ports clk1]
```

The `create_generated_clock` constraint defines a relationship between an internally generated clock and its source clock. This constraint only supports the `divide_by`, `multiply_by`, `duty_cycle`, and `invert` options.

Example: Creating Clocks

The following example shows the constraints for the base clock `clk` and the internally generated clock `clkdiv2`.



```
create_clock -name clk -period 10 [get_ports clk]
create_generated_clock -source clk -divide_by 2 clkdiv2
```

Virtual clocks are clocks that are not assigned to a timing node. They are used to represent off-chip clocks and are used in `set_input_delay` and `set_output_delay` constraints.

For more details, refer to:

- create_clock Constraint on page 53
- create_generated_clock Constraint on page 54

Using the create_clock Constraint

Any interface block that can be a clock source (PLL, GPIOs, MIPI RX Lane, MIPI RX/TX PHY, and JTAG) has an auto-generated `create_clock` constraint in the <*project*>**.pt.sdc**. There are several cases:

- Constraints where the software knows the clock value. <*project*>**.pt.sdc** includes the number.
- Constraints where you have to define the clock value (typically, GPIO resources being used as GCLK, RCLK, and JTAG TCK). These constraints are commented out and have a placeholder for you to add in the number.

A GPIO with a connection type other than GCLK, RCLK, or JTAG TCK does not have a template (because the software thinks they are not clocks). You need to write your own `create_clock` command.

Tip: Common mistakes when using `create_clock` SDC command:

Using the -*name* option without a target (e.g., `get_ports`) and thereby creating a virtual clock by mistake. The Efinity software prints an info message when it finds a virtual clock definition.

Using the instance name instead of the clock pin name. The clock pin name you use in the Interface Designer is the name used in the core timing netlist.

Example: PLL

The PLL Timing Report section shows the details about the clock generated by PLLs in the interface. Details including clock period, phase shift, and whether the clock is inverted are listed in the section. You copy the constraints from <project>.pt.sdc into your SDC file, you do not need to change them.

```
# PLL Constraints
#################
create_clock -period 10.0000 i_hbramClk_fb
create_clock -waveform {1.2500 3.7500} -period 5.0000 i_hbramClk90
...
```

Example: GPIO Clock (GCLK and RCLK)

The following sections have `create_clock` constraint templates that you need to modify:

- GPIO Constraints
- HSIO GPIO Constraints (Titanium only)

To constrain these clocks, replace `<USER_PERIOD>` in the `create_clock` template line with the clock period and uncomment the line. If necessary, you can define the waveform if the clock is not using a standard 50/50 duty cycle.

Example: Template

```
# GPIO Constraints
####################
# create_clock -period <USER_PERIOD> [get_ports {clock}]
...
```

Example: Your SDC File

```
create_clock -period 10 [get_ports {clock}]
```

Example: Regular GPIO Used as a Clock

You need to use a regular GPIO as a clock, for example if you need a bidirectional signal that sometimes acts as a clock. In this case you need to write your own `create_clock` command because the software cannot generate a template for it.

Note: If the Efinity software detects a signal that it thinks is a clock but you have not specified the GPIO as a GCLK or RCLK, the software gives a warning.

Important: Efinix does not recommend using a regular GPIO as a clock for Trion FPGAs because it will have to route to the global clock network (GCLK), which results in additional and variable delay.

For simple inputs and outputs, the instance name and pin name are usually the same (just to make things easier). A GPIO in `inout` mode has three pins with different names. The following example has instance `bclk` with 3 pin names for the input, output, and output enable.

Example: Timing Report for GPIO in inout Mode

```
---------- 1.1 HSIO GPIO Timing Report (begin) ----------

Non-registered HSIO GPIO Configuration:
========================================

+---------------+----------+-----------+----------+----------+
| Instance Name | Pin Name | Parameter | Max (ns) | Min (ns) |
+---------------+----------+-----------+----------+----------+
|      bclk     | bclk_IN  |  GPIO_IN  |  0.828   |  0.552   |
|      bclk     | bclk_OUT | GPIO_OUT  |  2.205   |  1.470   |
|      bclk     | bclk_OE  | GPIO_OUT  |  1.953   |  1.302   |
+---------------+----------+-----------+----------+----------+

---------- HSIO GPIO Timing Report (end) ----------
```

For this case, the `create_clock` constraint is:

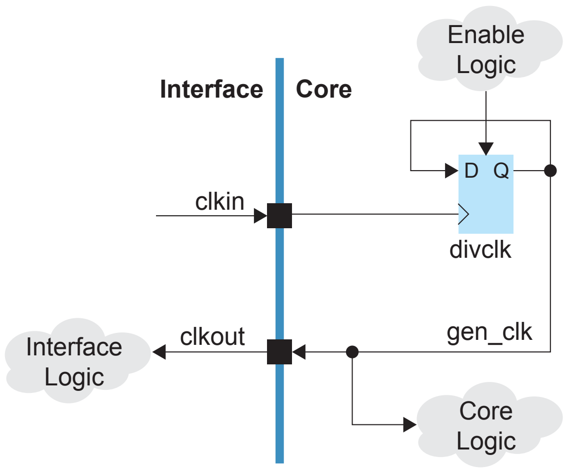
```
create_clock -period 10 -name bclk [get_ports bclk_IN]
```

Note: The constraint does not use `bclk` for `get_ports`, it uses the pin name not the instance.

Using the create_generated_clock Constraint

The Interface Designer does not create SDC constraints for generated clocks. Typically, you implement the generated clock in the core design by dividing down interface clocks. You need to add constraints for these clocks.

Figure 3: Divide Down Clock



SDC Commands:

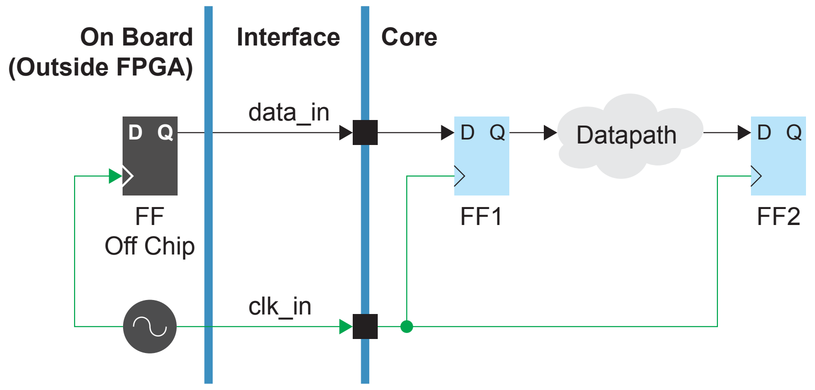
```
create_clock -period 10 -name clk0 [get_ports clkin]
create_generated_clock -source [get_ports clkin] -divide_by 2 [ get_pins divclk|Q ] -name
 gen_clk0
```

Virtual Clocks

A virtual clock represents a system clock that is on the board but is off-chip from the FPGA. In your SDC files, you should use a virtual clock as a reference clock for the input and output delay instead of the board clock. The virtual clock provides a clean interface clock and means you do not have to worry about the shifted waveform on the board. Additionally, the virtual clock prevents timing analysis from treating the I/O path with overly tight and unrealistic requirements.

The following figure shows a virtual clock used with the `set_input_delay` command. The oscillator drives the clock pad, `clk_in`, and the clock pin of an external off-chip flipflop. The path from the oscillator to the `clk_in` pad on the core is through the interface. The Interface Designer can add extra clock latency and clock uncertainty to that path. To remove any extra clock latency and uncertainty for the `data_in` pad, you use a virtual clock.

Figure 4: Virtual Clock with set_input_delay Example



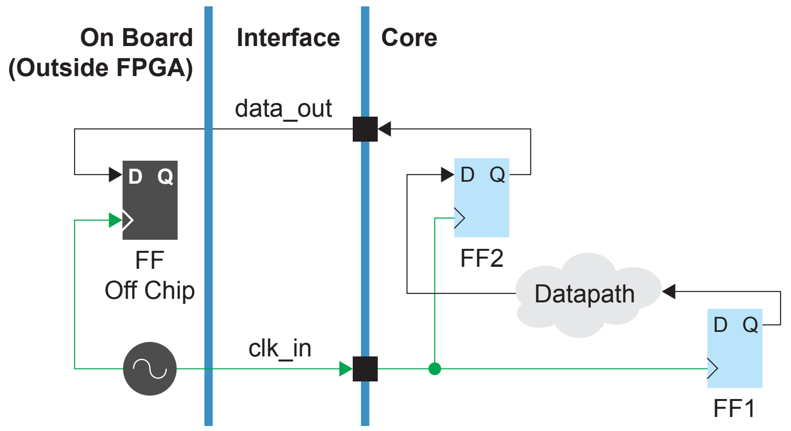
Example: SDC Commands

```
create_clock -period 40 -name clk_in [get_ports clk_in]
create_clock -period 40 -name virtual_clk
set_input_delay -clock virtual_clk -max 0.3 [get_ports data_in]
set_input_delay -clock virtual_clk -min 0.1 [get_ports data_in]
```

Notice that virtual clock has the same period and characteristics as `clkin` but it does not have a clock target referring to a net, port, or pin in the netlist. The Efinity software displays an info message for the virtual clock.

The following figure shows how to use a virtual clock with the `set_output_delay` command.

Figure 5: Virtual Clock with set_output_delay Example



Example: SDC Commands

```
create_clock -period 40 -name clk_in [get_ports clk_in]
create_clock -period 40 -name virtual_clk
set_output_delay -clock virtual_clk -max 0.4 [get_ports data_out]
set_output_delay -clock virtual_clk -min 0.3 [get_ports data_out]
```

Note: In your SDC file, put the virtual clock and core clock in the same clock group so they are related. The software can then analyze the transfers from `virtual_clk` to/from `clk_in`. See **Clock Relationships** on page 26.

Clock Latency

The source clock latency represents the time it takes to get from the clock source on the board to the global clock tree on the FPGA. This delay includes the board delay, buffer delay, and any PLL delay (including PLL compensation delay, which is negative, see AN 042: Working with PLLs).

Most of the time you do not need to use `set_clock_latency`. However, it is required when you want to constrain external signals to core registers to capture the latency effect of the clock signal transferring onto the FPGA.

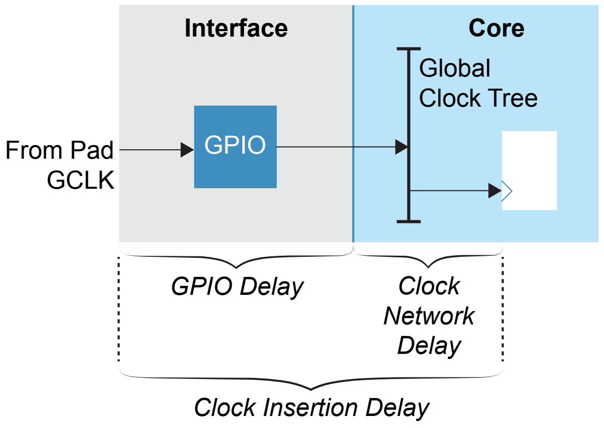
You need to calculate the delay based on the GPIO mode, PLL mode, and any board delays.

The Efinity software v2023.2 and higher creates a template for the `set_clock_latency` constraint in the <*project*>**.pt.sdc** file. The following topics explain how to calculate clock latency for GPIO and PLL clocks and how to use the template to create the SDC constraints.

GPIO Clock Latency

When using a GPIO as a clock source you need to account for the any board delay and the GPIO input buffer delay.

Figure 6: GBUF to Register Delay



The SDC constraint formulas for the receive clock delay are:

```
set_clock_latency -source -setup <max calculation> <clock ports>
set_clock_latency -source -hold <min calculation> <clock ports>
```

The equations are:

$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_CLK\_IN}_{\text{max}}$

$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_CLK\_IN}_{\text{min}}$

The following example shows how to calculate the delays and set the constraints.

Example: Setting GPIO Clock Latency

You need to define the clock latency before the core clock pin `clk`. This example assumes that the clock and data traces on the board are well matched; therefore, there is no external board delay.

The GPIO clock buffer delays are shown in the Excerpt of <*project*>**.pt_timing.rpt**: (non-registered GPIO table):

```
Non-registered HSIO GPIO Configuration:
========================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|      clk      |   clk    | GPIO_CLK_IN |  0.828   |  0.552   |
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

The <*project*>**.pt.sdc** template is:

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max + 0.828> [get_ports {clk}]
# set_clock_latency -source -hold <board_min + 0.552> [get_ports {clk}]
```

There is no board delay in this example, therefore, the equations are:

<*max calculation*> = 0 + 0.828 = 0.828

<*min calculation*> = 0 + 0.552 = 0.552

The resulting constraints are:

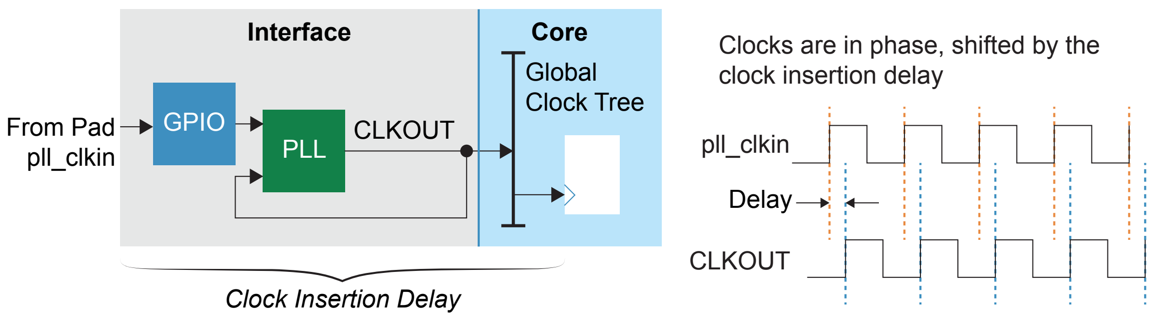
```
set_clock_latency -source -setup 0.828 [get_ports clk]
set_clock_latency -source -hold 0.552 [get_ports clk]
```

PLL Local Feedback Clock Latency

When using a PLL as a clock source you need to account for the any board delay, the GPIO input buffer delay (for the PLL's reference clock pin), and the PLL compensation delay.

When the PLL is in local feedback mode, the compensation delay is zero.

Figure 7: PLL Local Feedback Mode Delay



The SDC constraint formulas for the receive clock delay are:

```
set_clock_latency -source -setup <max calculation> <clock ports>
set_clock_latency -source -hold <min calculation> <clock ports>
```

The equations are:

$<$*max calculation*$>$ = $<$*max board constraint*$>$ + GPIO_IN_{max} - $<$*PLL compensation*$>$

$<$*min calculation*$>$ = $<$*min board constraint*$>$ + GPIO_IN_{min} - $<$*PLL compensation*$>$

The Efinity software v2023.2 and higher calculates the GPIO input buffer and PLL compensation delays and provides them in a template in the <*project*>**.pt.sdc** file. You still need to add any board delays if needed.

The following example shows how to calculate the delays and set the constraints.

Example: Setting PLL Local Feedback Clock Latency

In this example, the PLL clock output is called `clk`. This example assumes that the clock and data traces on the board are well matched; therefore, there is no external board delay.

For the GPIO_IN delays, this example uses the values for the `i` pin.

The PLL compensation delay is 0 in this mode.

Excerpt of <*project*>**.pt_timing.rpt**:

```
Non-registered HSIO GPIO Configuration:
========================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

The <*project*>**.pt.sdc** template is:

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max + 0.828> [get_ports {clk}]
# set_clock_latency -source -hold <board_min + 0.552> [get_ports {clk}]
```

There is no board delay in this example, therefore, the equations are:

<*max calculation*> = 0 + 0.828= 0.828

<*min calculation*> = 0 + 0.552 = 0.552

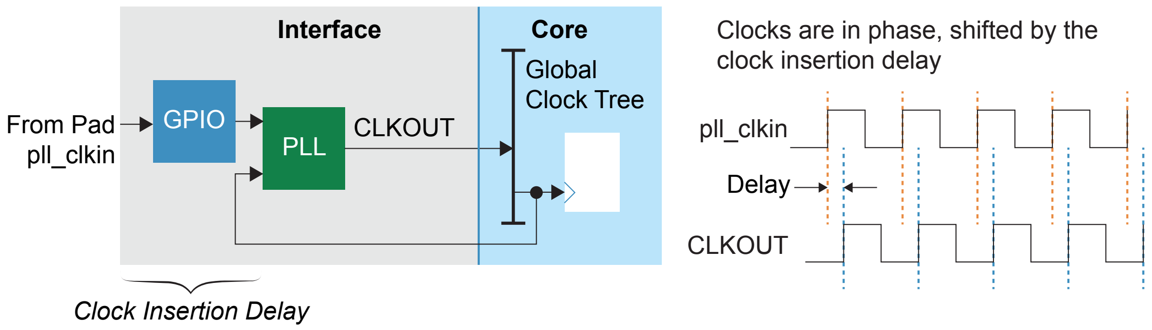
The resulting constraints are:

```
set_clock_latency -source -setup 0.828 [get_ports clk]
set_clock_latency -source -hold 0.552 [get_ports clk]
```

PLL Core Feedback Clock Latency

When using a PLL as a clock source you need to account for the any board delay, the GPIO input buffer delay (for the PLL's reference clock pin), and the PLL compensation delay.

Figure 8: Core Feedback Mode Delay



When the PLL is in core feedback mode, the compensation delay is equal to the clock network delay.

The SDC constraint formulas for the receive clock delay are:

```
set_clock_latency -source -setup <max calculation> <clock ports>
set_clock_latency -source -hold <min calculation> <clock ports>
```

The equations are:

$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_IN}_{\text{max}} - <\textit{PLL compensation}>$

$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_IN}_{\text{min}} - <\textit{PLL compensation}>$

The Efinity software v2023.2 and higher calculates the GPIO input buffer and PLL compensation delays and provides them in a template in the <*project*>**.pt.sdc** file. You still need to add any board delays if needed.

Example: Setting PLL Core Feedback Clock Latency

In this example, the PLL clock output is called `clk`. This example assumes that the clock and data traces on the board are well matched; therefore, there is no external board delay.

Excerpt of <*project*>**.pt_timing.rpt**:

```
---------- 1. PLL Timing Report (begin) ----------

+---------+----------+-----------+-...-+-------------------+-------------------+
| PLL     | Resource | Reference | ... | PLL Compensation  | PLL Compensation  |
|Instance|          |   Clock   | ... | Delay Max (ns)    | Delay Min (ns)    |
+---------+----------+-----------+-...-+-------------------+-------------------+
|   pll   | PLL_TR0  | external  | ... |       4.310       |      2.155        |
+---------+----------+-----------+-...-+-------------------+-------------------+

+-------+-------------+-----------------------+
| Clock | Period (ns) | Phase Shift (degrees) |
+-------+-------------+-----------------------+
|  clk  |   10.0000   |           0           |
+-------+-------------+-----------------------+

Non-registered GPIO Configuration:
===================================

+---------------+----------+-----------+----------+----------+
| Instance Name | Pin Name | Parameter | Max (ns) | Min (ns) |
+---------------+----------+-----------+----------+----------+
|       i       |    i     |  GPIO_IN  |  1.396   |  0.698   |
|     refclk    |  refclk  |  GPIO_IN  |  1.476   |  0.738   |
|       o       |    o     |  GPIO_OUT |  3.829   |  1.915   |
+---------------+----------+-----------+----------+----------+
```

The <*project*>**.pt.sdc** template is:

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max -2.834> [get_ports {clk}]
# set_clock_latency -source -hold <board_min -1.457> [get_ports {clk}]
```

The equations are:

<*max calculation*> = 0 - 2.834 = -2.834

<*min calculation*> = 0 - 1.457 = -1.457

The numbers are negative because the PLL compensation is so much larger than the input delay.

The resulting constraints are:

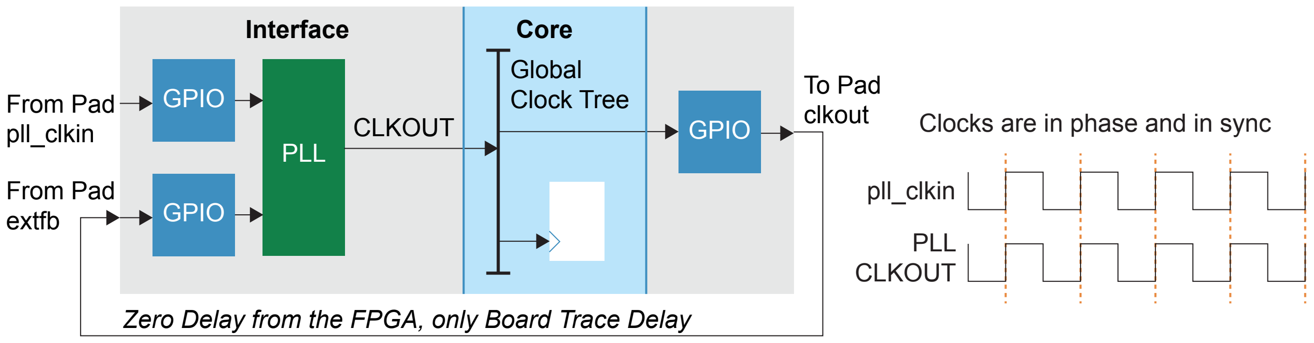
```
set_clock_latency -source -setup -2.834 [get_ports clk]
set_clock_latency -source -hold -1.457 [get_ports clk]
```

PLL External Feedback Clock Latency

When using a PLL as a clock source you need to account for the any board delay, the GPIO input buffer delay (for the PLL's reference clock pin), and the PLL compensation delay.

Note: Trion FPGAs do not have external feeddback mode.

Figure 9: External Feedback Mode Delay



When the PLL is in external feedback mode, the compensation delay is equal to GPIO_IN plus the clock network delay plus GPIO_CLK_OUT.

The SDC constraint formulas for the receive clock delay are:

```
set_clock_latency -source -setup <max calculation> <clock ports>
set_clock_latency -source -hold <min calculation> <clock ports>
```

The equations are:

$$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_IN}_{\text{max}} - <\textit{PLL compensation}>$$

$$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_IN}_{\text{min}} - <\textit{PLL compensation}>$$

The Efinity software v2023.2 and higher calculates the GPIO input buffer and PLL compensation delays and provides them in a template in the <*project*>**.pt.sdc** file. You still need to add any board delays if needed.

The following example shows how to calculate the delays and set the constraints.

Example: Setting PLL External Feedback Clock Latency

In this example, the PLL clock output is called `clk`. This example assumes that the clock and data traces on the board are well matched; therefore, there is no external board delay.

Excerpt of <*project*>**.pt_timing.rpt**:

```
+---------+----------+-----------+-...-+-------------------+-------------------+
| PLL     | Resource | Reference | ... | PLL Compensation  | PLL Compensation  |
|Instance|           |   Clock   | ... | Delay Max (ns)    | Delay Min (ns)    |
+---------+----------+-----------+-...-+-------------------+-------------------+
|  pll    | PLL_BL0  | external  | ... |       5.379       |       3.541       |
+---------+----------+-----------+-...-+-------------------+-------------------+

+-------+-------------+----------------------------+----------------------+----------+
| Clock | Period (ns) | Enable Dynamic Phase Shift | Phase Shift (degrees) | Inverted |
+-------+-------------+----------------------------+----------------------+----------+
|  clk  |   10.0000   |           False            |         0.0          |  false   |
+-------+-------------+----------------------------+----------------------+----------+

---------- PLL Timing Report (end) ----------

---------- 2.1 HSIO GPIO Timing Report (begin) ----------

Clkout GPIO Configuration:
===========================

+---------------+-----------+-------------+----------+----------+--------------------+
| Instance Name | Clock Pin |  Parameter   | Max (ns) | Min (ns) | Reference Pin Name |
+---------------+-----------+-------------+----------+----------+--------------------+
|     clkout    |    clk    | GPIO_CLK_OUT |  2.205   |  1.470   |  clk~CLKOUT~18~1   |
|     refclk    |  refclk   |  GPIO_IN     |  0.828   |  0.552   |                    |
|    pll_fbk    |  pll_fbk  |  GPIO_IN     |  0.828   |  0.552   |                    |
+---------------+-----------+-------------+----------+----------+--------------------+
```

The <*project*>**.pt.sdc** template is:

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max -4.551> [get_ports {clk}]
# set_clock_latency -source -hold <board_min -2.989> [get_ports {clk}]
```

The equations are:

<*max calculation*> = 0 - 4.551 = -4.551

<*min calculation*> = 0 - 2.989 = -2.989

The numbers are negative because the PLL compensation is so much larger than the input delay.

The resulting constraints are:

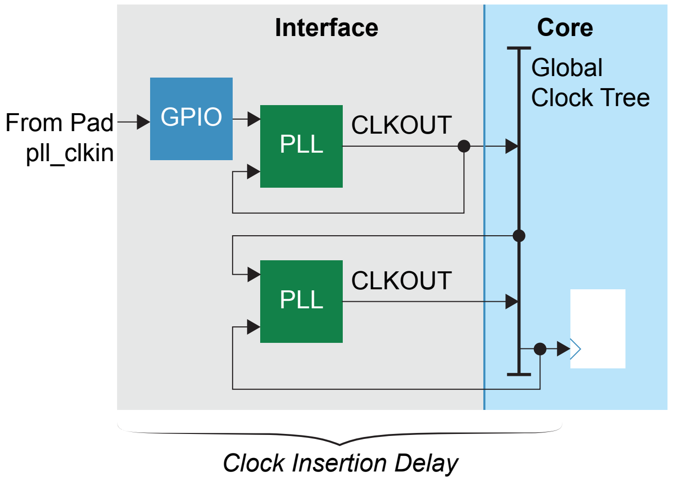
```
set_clock_latency -source -setup -4.551 [get_ports clk]
set_clock_latency -source -hold -2.989 [get_ports clk]
```

PLL Cascading Clock Latency

When using cascaded PLLs as a clock source you need to account for the any board delay, the GPIO input buffer delay (for the PLL's reference clock pin), and the PLL compensation delay.

Note: You should cascade a maximum of 1 PLL, that is, the source PLL and the cascaded one.

Figure 10: PLL Cascade Delay



The SDC constraint formulas for the receive clock delay are:

```
set_clock_latency -source -setup <max calculation> <clock ports>
set_clock_latency -source -hold <min calculation> <clock ports>
```

The equations for the source PLL are:

$<\textit{max calculation}> = <\textit{max board constraint}> - <\textit{source PLL clock latency}>$

$<\textit{min calculation}> = <\textit{min board constraint}> - <\textit{source PLL clock latency}>$

The equations for the cascaded PLL are:

$<\textit{max calculation}> = <\textit{source PLL clock latency}>_{\text{max}} + <\textit{cascaded PLL clock latency}>_{\text{max}}$

$<\textit{min calculation}> = <\textit{source PLL clock latency}>_{\text{min}} + <\textit{cascaded PLL clock latency}>_{\text{min}}$

The Efinity software v2023.2 and higher calculates the GPIO input buffer and PLL compensation delays and provides them in a template in the <*project*>**.pt.sdc** file. You still need to add any board delays (if needed).

For the cascaded PLL, the software includes the clock network delay in the PLL compensation delay value.

The following example shows how to calculate the delays and set the constraints.

Example: PLL Cascading Clock Latency

In this example, the PLL clock output is called `clk`. This example assumes that the clock and data traces on the board are well matched; therefore, there is no external board delay.

Excerpt of <*project*>**.pt_timing.rpt**:

```
---------- 1. PLL Timing Report (begin) ----------

+--------------+----------+-----------------+-...-+---------------------------------
+--------------------------------+
| PLL Instance | Resource | Reference Clock | ... | PLL Compensation Delay Max (ns) | PLL
 Compensation Delay Min (ns) |
+--------------+----------+-----------------+-...-+---------------------------------
+--------------------------------+
|   src_pll    | PLL_BL0  |     external    | ... |              2.346              |
    1.519             |
|   casc_pll   | PLL_BL1  |       core      | ... |              2.341              |
    1.516             |
+--------------+----------+-----------------+-...-+---------------------------------
+--------------------------------+

+--------------+-------------+----------------------------+----------------------+----------+
|    Clock     | Period (ns) | Enable Dynamic Phase Shift | Phase Shift (degrees) | Inverted |
+--------------+-------------+----------------------------+----------------------+----------+
| src_pll_clk  |   10.0000   |           False            |          0.0          |  false   |
| casc_pll_clk |   10.0000   |           False            |          0.0          |  false   |
+--------------+-------------+----------------------------+----------------------+----------+

---------- PLL Timing Report (end) ----------

---------- 2. GPIO Timing Report (begin) ----------

Non-registered GPIO Configuration:
===================================

+---------------+----------+-----------+----------+----------+
| Instance Name | Pin Name | Parameter | Max (ns) | Min (ns) |
+---------------+----------+-----------+----------+----------+
|     refclk    |  refclk  |  GPIO_IN  |  0.828   |  0.552   |
+---------------+----------+-----------+----------+----------+
```

The <*project*>**.pt.sdc** template is:

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max -1.517> [get_ports {src_pll_clk}]
# set_clock_latency -source -hold <board_min -0.967> [get_ports {src_pll_clk}]
# set_clock_latency -source -setup <board_max + 0.004> [get_ports {casc_pll_clk}]
# set_clock_latency -source -hold <board_min + 0.003> [get_ports {casc_pll_clk}]
```

The equations for the source PLL are:

<*max calculation*> = 0 - 1.517 = -1.517

<*min calculation*> = 0 - 0.967 = -0.967

The equations for the cascaded PLL are:

<*max calculation*> = 0 - 1.517 + 0.004 = -1.513

<*min calculation*> = 0 - 0.967 + 0.003 = -0.964

The numbers are negative because the PLL compensation is so much larger than the input delay.

The resulting constraints are:

```
set_clock_latency -source -setup - 1.517 [get_ports {src_pll_clk}]
set_clock_latency -source -hold - 0.967 [get_ports {src_pll_clk}]
set_clock_latency -source -setup - 1.513 [get_ports {casc_pll_clk}]
set_clock_latency -source -hold - 0.964 [get_ports {casc_pll_clk}]
```

Clock Relationships

By default, the Efinity® software assumes that all clocks are related and it analyzes the timing between all clock domains and optimizes all possible paths.

If you set constraints for two clocks, and do not cut the path between them, the software tries to find the tightest clock-to-clock delay requirement between them. If the timer cannot find a common clock period for the two clocks after 1,000 clock cycles, it determines that they are *non-expandable*. The timer gives these clocks a default constraint of 0.01 ns. If you want to override this default, use the set_max_delay or set_min_delay constraint.

Important: Efinix recommends that you explicitly set constraints to indicate unrelated clocks. That way the software does not perform unnecesssary path optimization, which can lead to problems with closing timing.

Setting Constraints for Unrelated Clocks

The first step is to analyze your design to determine which clocks are related and which are not. You then use one of the following constraints:

- `set_clock_groups`—Use when you want to specify bidirectional constraints between clocks. Generally, this is the simplest method, and the fastest for the Efinity® timer to analyze. (See set_clock_groups Constraint on page 58.)
- `set_false_path`—Use when you want to be specific about which clocks connect with which end points. This constraint is one-directional, so you need to specify two constraints, one for each direction. (See set_false_path Constraint on page 59.) Typically, you use these constraints when you want to indicate timing exceptions for a subset of timing end points in one of the clock domains.

Using the set_clock_groups Constraint

Use this constraint to define the relationship between the clocks and generated clocks you defined. Typically, only clocks from the same source are related to each other. For example, clock outputs from the same PLL or clocks from a single clock pin. Any other clocks should be specified as unrelated.

Unrelated clock groups can be *exclusive* or *asynchronous*.

- Exclusive clock groups do not operate at the same time as each other.
- Asynchronous clock groups have no timing relationship between them, for example, clocks driven from two independent PLLs.

You use the `-exclusive` or `-asynchronous` options to define how to treat the clock groups. The Efinity® software treats both options identically, but some third-party EDA tools use these constraints when checking for proper clock domain crossing logic. Therefore, it is a good idea to use the correct option for the relationship.

To illustrate how to set constraints using `set_clock_groups`, consider a design with four clocks, `clk1`, `clk2`, `clk3`, and `clk4`. After design analysis, you determine that `clk1` and `clk2` are related to each other and `clk3` and `clk4` are unrelated to all others. There are two ways to use the `set_clock_groups` constraint, both of which are correct.

Example: Use a Single Constraint

The first method is to define the clocks and groups with a single constraint:

```
set_clock_groups -exclusive -group {clk1 clk2} -group {clk3} -group {clk4}
```

This constraint defines the relationship between clocks `clk1`, `clk2`, `clk3`, and `clk4`. If you later add an additional clock, `clk5`, and do not update the constraints, the software assumes that `clk5` is synchronous to all other clocks.

Example: Use Separate Constraints

The second method is to use separate constraints for each group:

```
set_clock_groups -exclusive -group {clk1 clk2}
set_clock_groups -exclusive -group {clk3}
set_clock_groups -exclusive -group {clk4}
```

In this case, each `set_clock_groups` constraint only specifies one group, which tells the software that the clocks in a given group are asynchronous to all others. With this method, if you later add `clk5`, the software would consider it to be asynchronous to `clk1`, `clk2`, `clk3`, and `clk4`.

It can be tempting to use the second method in case you forget a clock or add one later. However, whichever method you choose, Efinix recommends that you *always include constraints for each clock* in your design and that you update your SDC file when you add clocks.

Using the set_false_path Constraint

The `set_false_path` constraint lets you be more specific when setting clock constraints. This constraint lets you cut the connection between a starting point (*from*) and an ending point (*to*). The from and to can be registers, I/O, or clocks.

The following constraint cuts the connection from `clk1` to `clk2`:

```
set_false_path -from clk1 -to clk2
```

Remember, though, that this only cuts the connection in one direction. To specify that there is no relationship between `clk1` and `clk2`, you also need to use the following constraint:

```
set_false_path -from clk2 -to clk1
```

Example: Using set_false_path Constraints

A complete example of the constraints needed for our hypothetical four-clock design is:

```
set_false_path -from clk1 -to clk3
set_false_path -from clk1 -to clk4
set_false_path -from clk2 -to clk3
set_false_path -from clk2 -to clk4
set_false_path -from clk3 -to clk1
set_false_path -from clk3 -to clk2
set_false_path -from clk3 -to clk4
set_false_path -from clk4 -to clk1
set_false_path -from clk4 -to clk2
set_false_path -from clk4 -to clk3
```

When you want to cut paths between clock domains, as in this simple example, Efinix recommends that you use `set_clock_groups` instead of `set_false_path`. The `set_false_path` constraint becomes more useful when you want to specify exceptions for registers or I/O, or if you want to cut only one direction of a clock domain pair.

Example: Cut Path to a Port or Pin

To cut only the path from `clk1` to a port named `testout`:

```
set_false_path -from clk1 -to [get_ports testout]
```

To cut only the path from `clk1` to a pin named `testout`:

```
set_false_path -from clk1 -to [get_pins instance|testout]
```

Clock Synchronizers

If you have asynchronous clock groups and want to transfer data between them, you need to add synchronizing registers (also known as synchronizers). Synchronizers are register chains in the receiving clock domain that capture data from the sending domain. They prevent meta-stable events from propagating into the receiving clock domain.

To designate a register as a synchronizer, use the `async_reg` synthesis attribute.

When `async_reg` is `true`, synthesis does not perform optimization to reduce, merge, or duplicate these registers. During place and route, the software keeps these registers close together to improve synchronization between asynchronous clock domains.

Verilog HDL:

```
(* async_reg = "true" *) reg [1:0] x;
```

VHDL:

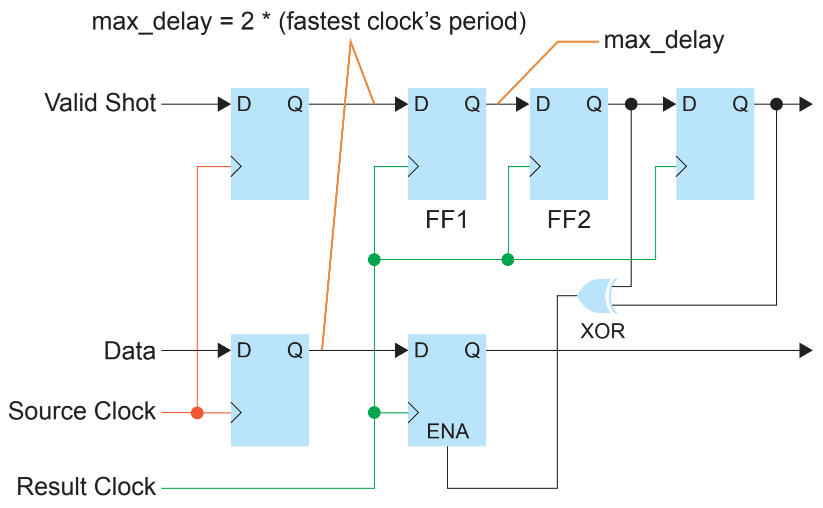
```
attribute async_reg: boolean;
attribute async_reg of x : signal is true;
```

Metastable Synchronizer Circuit

This example shows a synchronizer, which is a circuit that stabilizes an input signal that may produce a metastable output. If possible, the registers in a synchronization chain need to be placed close to each other. Efinix recommends that you use the `async_reg` synthesis attribute for synchronizer registers.

In the following figure, `FF1` and `FF2` should be close together. Use the async_reg synthesis attribute for the `FF1` and `FF2` registers in the RTL netlist., which tells the software to keep those registers close together during place-and-route.

Figure 11: Metastability Synchronizer Example



How to Set Clock Uncertainty

Trion®, Topaz, and Titanium FPGAs have a default clock uncertainty for setup and hold analysis. You can view the clock uncertainty in the Static Timing Analysis Report (<*project name*>**.timing.rpt**). If the you have not set the uncertainty, the report uses the default value. For example, the T8 has 140 ps for setup and 50 ps for hold. You can modify these defaults by including the `set_clock_uncertainty` command in your SDC file.

One reason to add uncertainty is to account for the quality of the clock that feeds into the FPGA, or because you want the design to have more margin. However, keep in mind that clock uncertainty comes from the timing slack reported for your design, so increasing the uncertainty makes it harder to meet timing.

Example: Add 60 ps Clock Uncertainty

You want to add 60 ps to the default uncertainty for `clk` for a T8 design. Add this command to your SDC file:

```
set_clock_uncertainty -to clk -setup 0.06
```

The Efinity® software uses 200 ps of clock uncertainty for setup analysis.

See set_clock_uncertainty Constraint on page 59 for details.

Constraining I/O

As discussed earlier, you need to constrain the connections from the interface to the core. All connections between the core and interface are considered to be *I/O* for timing analysis.

If a given interface block is synchronizing the connection to the core, the Interface Designer SDC template includes the `set_input_delay` and `set_output_delay` SDC constraints that you need to use. When it is not synchronized, you need to add external board delays to the values the Interface Designer shows.

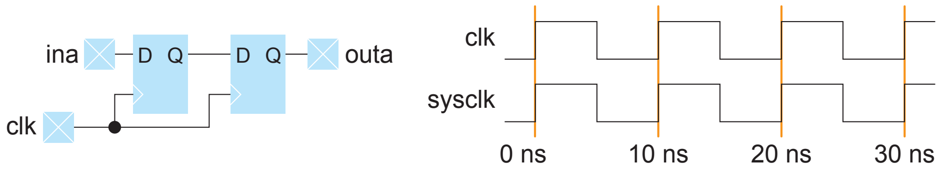
Note: For Trion®, Topaz, and Titanium FPGAs, most interface connections are synchronous. The exceptions are GPIO blocks in bypass mode and LVDS blocks in x1 bypass mode.

Constrain I/O pins to be timing-equivalent to a register that is clocked with the real or virtual clock you defined. Then, use the set_input_delay and set_output_delay constraints.

Example: Constraining I/O Pins

In this example, `sysclk` is a virtual clock.

Figure 12: Clock and I/O Pin Constraint Example



Use these constaints to define the clock and set the delays for the pins:

```
create_clock -name clk -period 10 [get_ports clk]
create_clock -name sysclk -period 10
set_input_delay -clock sysclk -max 2.4 [get_ports ina]
set_output_delay -clock sysclk -max 1.2 [get_ports outa]
```

Constraining Synchronous Inputs and Outputs

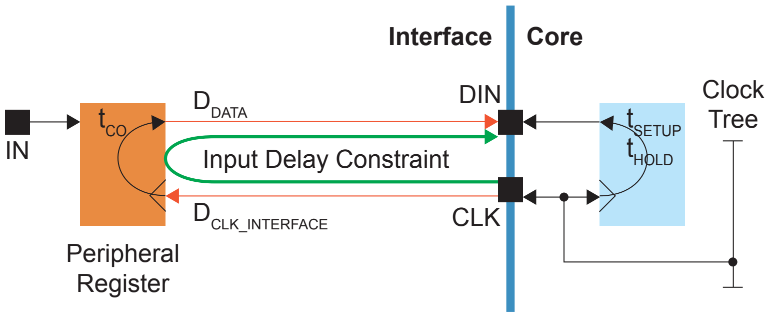
Synchronous inputs and outputs are interface signals that are connected to synchronous elements in the FPGA's periphery. Because the Interface Designer knows how the clock and data signals are connected to the synchronous elements, the software can automatically determine the precise delays for the `set_input_delay` and `set_output_delay` constraints. These delays are provided in the <*project name*>**.pt.sdc** file. When the Efinity software generates the constraints for synchronized output and input pins, it creates a `set_output_delay` or `set_input_delay` that captures the delay values of the synchronous element and the core clock delay of the FPGA.

When the Efinity software models the timing, the minimum and maximum refer to different timing corners (fast corner and slow corner), not the minimum/maximum potential delay in one timing corner.

Understanding Input Delay Values

The following figure shows an example of a peripheral register, clock, clock-to-output delay, and data path.

Figure 13: Input Delay Example



- t_{CO} is the peripheral register's clock-to-output delay.
- D_{DATA} is the delay from the peripheral register to the core.
- $D_{CLK\_INTERFACE}$ is the clock delay to the peripheral register.

So the equations for the output delay are:

Maximum input delay = D_{DATA} (max) + t_{CO} + $D_{CLK\_INTERFACE}$ (max)

Minimum input delay = D_{DATA} (min) + t_{CO} + $D_{CLK\_INTERFACE}$ (min)

For example:

Parameter	Max	Min
D_{DATA}	2	1
t_{CO}	2	1
$D_{CLK\_INTERFACE}$	2	1

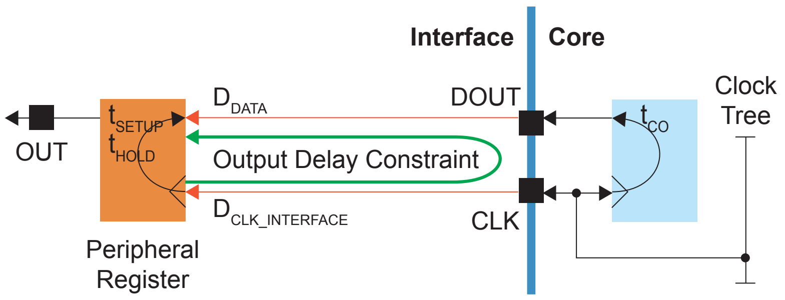
- Maximum input delay = 2 + 2 + 2 = 6
- Mininum input delay = 1 + 1 + 1 = 3

The generated constraint has the `-reference_pin` option, which lets the software automatically calculate the core clock network delay.

Understanding Output Delay Values

The following figure shows an example of a peripheral register, clock, setup/hold, and data path.

Figure 14: Output Delay Example



- t_{SETUP} is the peripheral register's setup requirement.
- t_{HOLD} is the peripheral register's hold requirement.

- D_{DATA} is the delay from the core to the peripheral register.
- $D_{CLK\_INTERFACE}$ is the clock delay to the peripheral register.

So the equations for the output delay are:

Maximum output delay (setup) = D_{DATA} (max) + t_{SETUP} - $D_{CLK\_INTERFACE}$ (max)

Minimum output delay (hold) = D_{DATA} (min) - t_{HOLD} - $D_{CLK\_INTERFACE}$ (min)

For example:

Parameter	Max	Min
D_{DATA}	2	1
t_{SETUP}	2	-
t_{HOLD}	-	1
$D_{CLK\_INTERFACE}$	2	1

- Maximum output delay = 2 + 2 - 2 = 2
- Mininum output delay = 1 - 1 - 1 = -1

The generated constraint has the `-reference_pin` option, which lets the software automatically calculate the core clock network delay.

Set Constraints

To set a constraint for synchronous inputs and outputs in your constraints file:

1. Go to **Result** > **Interface** in the Efinity® dashboard.
2. Double-click <*project name*>**.pt.sdc** to open the report.
3. Copy the `set_input_delay` and `set_output_delay` constraints and paste them into your constraints file.

Example: set_output_delay Constraints

```
set_output_delay -clock Clk -reference_pin [get_ports {Clk~CLKOUT~14~1}] -max 0.287
    [get_ports {MemWrite}]
set_output_delay -clock Clk -reference_pin [get_ports {Clk~CLKOUT~14~1}] -min 0.161
    [get_ports {MemWrite}]
```

Constraining Unsynchronized Inputs and Outputs

Unsynchronized inputs and outputs are simple GPIO blocks in bypass mode or LVDS blocks in x1 bypass mode. For these blocks, you need to factor in any external board delays when calculating the `-min` and `-max` values for the input and output delays.

For blocks in bypass mode, the constraint clock is external to the FPGA:
- A *receive* clock is generated outside of the FPGA and is passed to the FPGA through a GPIO pin.
- A *forward* clock is generated by the FPGA and sent off chip though a GPIO pin in clock out mode.

Both receive and forward clocks synchronize the signal off chip.

For unsynchronized input or output signals, the GPIO block bypasses the register. `GPIO_IN` represents a combinational delay from the pad through the I/O buffer. `GPIO_OUT` represents a combinational delay to the pad through the I/O buffer from either the output or output enable signals.

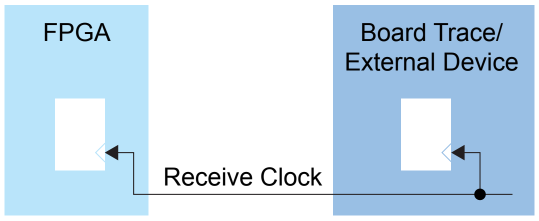
The general procedure for constraining unsynchronized inputs and outputs is:

1. Determine which mode you are constraining (input receive, input forward, output receive, or output forward).
2. Find the mininum (fast) and maximum (slow) timing values in the Interface Designer report file <*design name*>**.pt_timing.rpt**.
3. Use formulas (provided in later sections) to calculate the delay.
4. Add the constraint to your SDC file.

Receive Clock

A receive clock is passed to the FPGA design by configuring a GPIO in input mode and and setting the connection type to GCLK or RCLK. `GPIO_IN_CLK` represents the combinational delay from the pad through the I/O buffer to the global clock tree.

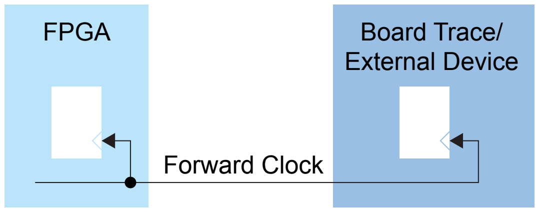
Figure 15: Receive Clocks



Forward Clock Using GPIO in clkout Mode

A forward clock is generated by the FPGA design and sent off chip by configuring a GPIO in **clkout** mode. `GPIO_CLK_OUT` represents the combinational delay through the FPGA clock tree and the I/O buffer to the pad.

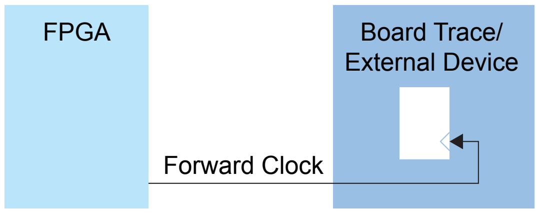
Figure 16: Forward Clocks



Forward Clock Using GPIO in output Mode

Sometimes the clock generated by the FPGA is only used in the external system and is not a clock in the FPGA design. In this case, you use a regular GPIO block in **output** mode to forward the clock off chip.

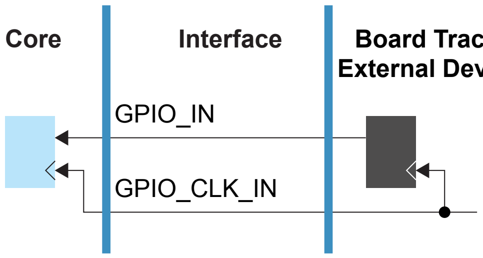
Figure 17: Forward Clocks



Input Receive Clock Delay

This example shows how to set constraints for an input receive clock.

Figure 18: Receive Clock Delay (GPIO Input, Register Bypass)



The SDC constraint formulas for the receive clock delay are:

```
set_input_delay -clock <clock> -max <max calculation> <ports>
set_input_delay -clock <clock> -min <min calculation> <ports>
```

The equations are:

$$<\textit{max calculation}> \; = \; <\textit{max board constraint}> \; + \; \text{GPIO\_IN}_{\text{max}}$$

$$<\textit{min calculation}> \; = \; <\textit{min board constraint}> \; + \; \text{GPIO\_IN}_{\text{min}}$$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Input Receive Clock

You want to constrain the `din` input with respect to clock `clkin` with a max board constraint of 4 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the Interface Designer timing report file is:

```
Non-registered GPIO Configuration:
===================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|     clkin     |  clkin   | GPIO_CLK_IN |  1.954   |  0.526   |
|      din      |   din    |   GPIO_IN   |  1.954   |  0.526   |
|     dout      |   dout   |   GPIO_OUT  |  4.246   |  1.081   |
+---------------+----------+-------------+----------+----------+
```

The equations are:

<max calculation> = 4 + 1.954 = 5.954

<min calculation> = 2 + 0.526 = 2.526

The resulting constraints are:

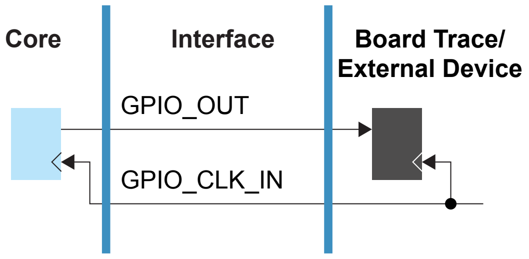
```
set_input_delay -clock clkin -max 5.954 din
set_input_delay -clock clkin -min 2.526 din
```

Note: The `GPIO_CLK_IN` delay is accounted for in the `set_clock_latency` constraint. Therefore, you do not need to include it in the calculation for `set_input_delay`. Refer to **Clock Latency** on page 16.

Output Receive Clock Delay

This example shows how to set constraints for an output receive clock.

Figure 19: Receive Clock Delay (GPIO Output, Register Bypass)



The SDC constraint formulas for the receive clock delay are:

```
set_output_delay -clock <clock> -max <max calculation> <ports>
set_output_delay -clock <clock> -min <min calculation> <ports>
```

The equations are:

$$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_OUT}_{\text{max}}$$

$$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_OUT}_{\text{min}}$$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Output Receive Clock

You want to constrain the `dout` output with respect to clock `clkin` with a max board constraint of 4 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the Interface Design report file is:

```
Non-registered GPIO Configuration:
===================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|     clkin     |  clkin   | GPIO_CLK_IN |  1.954   |  0.526   |
|      din      |   din    |   GPIO_IN   |  1.954   |  0.526   |
|      dout     |   dout   |   GPIO_OUT  |  4.246   |  1.081   |
+---------------+----------+-------------+----------+----------+
```

The equations are:

<max calculation> = 4 + 4.246 = 8.246

<min calculation> = 2 + 1.081 = 3.081

The resulting constraints are:

```
set_output_delay -clock clkin -max 8.246 dout
set_output_delay -clock clkin -min 3.081 dout
```

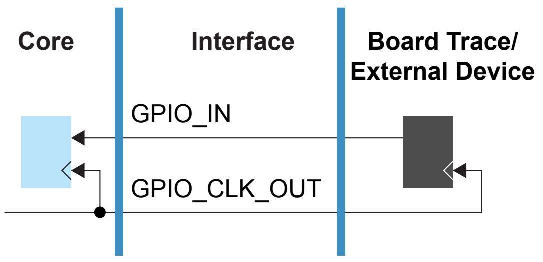
Note: The `GPIO_CLK_IN` delay is accounted for in the `set_clock_latency` constraint. Therefore, you do not need to include it in the calculation for `set_output_delay`. Refer to Clock Latency on page 16.

Input Forward Clock Delay (GPIO clkout)

This example shows how to set constraints for an input forward clock.

Warning: Most designs do not need to use this method. For high-performance designs, you should use the GPIO registers and follow the instructions in Constraining Synchronous Inputs and Outputs on page 30.

Figure 20: Forward Clock Delay (GPIO Input, Register Bypass)



The SDC constraint formulas for the foward clock delay are:

```
set_input_delay -clock <clock> -reference_pin <clkout interface name> \
   -max <max calculation> <ports>
set_input_delay -clock <clock> -reference_pin <clkout interface name> \
    -min <min calculation> <ports>
```

Reference Pin

With forward clocks, you use the `-reference_pin` option to include the clock latency delay in the I/O constraint. The `-reference_pin` pin target is a clkout pad that the software automatically adds to the netlist. The <*project*>**.pt_timing.rpt** file shows the reference pin name.

Calculate the min and max constraints using the following equations:

$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_IN}_{\text{max}} + \text{GPIO\_CLK\_OUT}_{\text{max}}$

$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_IN}_{\text{min}} + \text{GPIO\_CLK\_OUT}_{\text{min}}$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Input Forward Clock

You want to constrain the `i` input with respect to clock `clk_fwd` with a max board constraint of 2 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the <*project*>**.pt_timing.rpt** file is:

```
Clkout GPIO Configuration:
===========================

+---------------+-----------+--------------+----------+----------+--------------------+
| Instance Name | Clock Pin |  Parameter   | Max (ns) | Min (ns) | Reference Pin Name |
+---------------+-----------+--------------+----------+----------+--------------------+
|    clk_fwd    |    clk    | GPIO_CLK_OUT |  2.205   |  1.470   |  clk~CLKOUT~219~1  |
+---------------+-----------+--------------+----------+----------+--------------------+

Non-registered HSIO GPIO Configuration:
========================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|      clk      |   clk    | GPIO_CLK_IN |  0.828   |  0.552   |
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

The equations are:

<max calculation> = 2 + 0.828 + 2.205 = 5.033

<min calculation> = 2 + 0.552 + 1.470 = 4.022

The resulting constraints are:

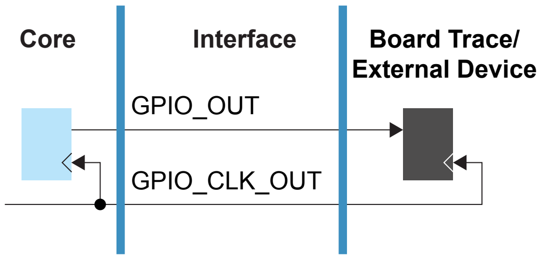
```
set_input_delay -clock clk -reference_pin clk~CLKOUT~219~1 -max 5.033 [get_ports {i}]
set_input_delay -clock clk -reference_pin clk~CLKOUT~219~1 -min 4.022 [get_ports {i}]
```

Output Forward Clock Delay (GPIO clkout)

This example shows how to set constraints for an output forward clock.

Warning: Most designs do not need to use this method. For high-performance designs, use the GPIO registers and follow the instructions in Constraining Synchronous Inputs and Outputs on page 30.

Figure 21: Forward Clock Delay (GPIO Output, Register Bypass)



The SDC constraint formulas for the forward clock delay are:

```
set_output_delay -clock <clock> -reference_pin <clkout interface name> \
   -max <max calculation> <ports>
set_output_delay -clock <clock> -reference_pin <clkout interface name> \
   -min <min calculation> <ports>
```

Calculate the min and max constraints using the following equations:

$<\textit{max calculation}> = <\textit{max board constraint}> + \mathrm{GPIO\_OUT}_{max} - \mathrm{GPIO\_CLK\_OUT}_{max}$

$<\textit{min calculation}> = <\textit{min board constraint}> + \mathrm{GPIO\_OUT}_{min} - \mathrm{GPIO\_CLK\_OUT}_{min}$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Output Forward Clock

You want to constrain the o output with respect to clock clk_fwd with a max board constraint of 2 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the Interface Designer timing report file is:

```
Clkout GPIO Configuration:
===========================

+---------------+-----------+--------------+----------+----------+--------------------+
| Instance Name | Clock Pin |  Parameter   | Max (ns) | Min (ns) | Reference Pin Name |
+---------------+-----------+--------------+----------+----------+--------------------+
|    clk_fwd    |    clk    | GPIO_CLK_OUT |  2.205   |  1.470   |  clk~CLKOUT~219~1  |
+---------------+-----------+--------------+----------+----------+--------------------+

Non-registered HSIO GPIO Configuration:
========================================

+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|      clk      |   clk    | GPIO_CLK_IN |  0.828   |  0.552   |
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

The equations are:

<max calculation> = 2 + 2.205 - 2.205 = 2

<min calculation> = 2 + 1.470 - 1.470 = 2

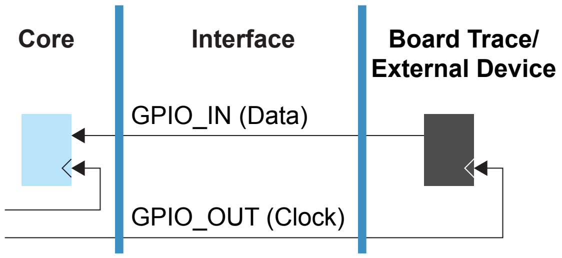
The resulting constraints are:

```
set_output_delay -clock clk -reference_pin clk~CLKOUT~219~1 -max 2 [get_ports {o}]
set_output_delay -clock clk -reference_pin clk~CLKOUT~219~1 -min 2 [get_ports {o}]
```

Input Forward Clock Delay (GPIO output)

This example shows how to set constratints for an input forward clock.

Figure 22: Forward Clock Delay (GPIO Input, Register Bypass)



The SDC constraint formulas for the foward clock delay are:

```
set_input_delay -clock <clock> -reference_pin <clkout interface name> \
   -max <max calculation> <ports>
set_input_delay -clock <clock> -reference_pin <clkout interface name> \
   -min <min calculation> <ports>
```

Reference Pin

With forward clocks, you use the `-reference_pin` option to include the clock latency delay in the I/O constraint. The reference pin target is the pin name of the GPIO output used for the clock.

Constraint Calculation

Calculate the min and max constraints using the following equations:

$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_IN}_{\text{max}}$
$+ <\textit{GPIO\_OUT for clock pad}>_{\text{max}}$

$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_IN}_{\text{min}}$
$+ <\textit{GPIO\_OUT for clock pad}>_{\text{min}}$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Input Forward Clock

You want to constrain the `i` input with respect to clock `clk_fwd` with a max board constraint of 2 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the Interface Designer timing report file is:

```
Non-registered HSIO GPIO Configuration:
========================================
+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|      clk      |   clk    | GPIO_CLK_IN |  0.828   |  0.552   |
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|    clk_fwd    | clk_fwd  |   GPIO_OUT  |  2.205   |  1.470   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

For <*GPIO_OUT for clock pad*>, use the `GPIO_OUT` value for `clk_fwd`.

The equations are:

<*max calculation*> = 2 + 0.828 + 2.205 = 5.033

<*min calculation*> = 2 + 0.552 + 1.470 = 4.022

The reference pin target is the forwarded clock, `clk_fwd`.

In this example, the RTL is using a divided down clock, `divclk`, which is only used to drive the `clk_fwd` signal off chip. Therefore, the `set_input_delay` constraint is relative to that generated clock. See **Example: FPGA Forwarded Clock** on page 65 for a more complete example.

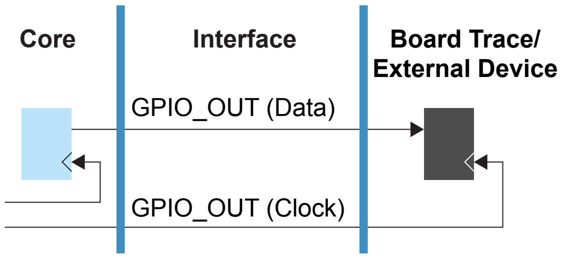
The resulting constraints are:

```
set_input_delay -clock divclk -reference_pin clk_fwd -max 5.033 [get_ports {i}]
set_input_delay -clock divclk -reference_pin clk_fwd -min 4.022 [get_ports {i}]
```

Output Forward Clock Delay (GPIO output)

This example shows how to set constratints for an input forward clock.

Figure 23: Forward Clock Delay (GPIO Output, Register Bypass)



The SDC constraint formulas for the foward clock delay are:

```
set_output_delay -clock <clock> -reference_pin <clkout interface name> \
   -max <max calculation> <ports>
set_output_delay -clock <clock> -reference_pin <clkout interface name> \
   -min <min calculation> <ports>
```

Reference Pin

With forward clocks, you use the `-reference_pin` option to include the clock latency delay in the I/O constraint. The reference pin target is the pin name of the GPIO output used for the clock.

Constraint Calculation

Calculate the min and max constraints using the following equations:

$<\textit{max calculation}> = <\textit{max board constraint}> + \text{GPIO\_OUT}_{\text{max}}$
$- <\textit{GPIO\_OUT for clock pad}>_{\text{max}}$

$<\textit{min calculation}> = <\textit{min board constraint}> + \text{GPIO\_OUT}_{\text{min}}$
$- <\textit{GPIO\_OUT for clock pad}>_{\text{min}}$

The following example shows how to calculate the delays and set the constraints.

Example: Constraining Output Forward Clock

You want to constrain the `o` output with respect to clock `clk_fwd` with a max board constraint of 2 ns and a min board constraint of 2 ns. The non-registered GPIO configuration data from the Interface Designer timing report file is:

```
Non-registered HSIO GPIO Configuration:
========================================
+---------------+----------+-------------+----------+----------+
| Instance Name | Pin Name |  Parameter  | Max (ns) | Min (ns) |
+---------------+----------+-------------+----------+----------+
|      clk      |   clk    | GPIO_CLK_IN |  0.828   |  0.552   |
|       i       |    i     |   GPIO_IN   |  0.828   |  0.552   |
|    clk_fwd    | clk_fwd  |   GPIO_OUT  |  2.205   |  1.470   |
|       o       |    o     |   GPIO_OUT  |  2.205   |  1.470   |
+---------------+----------+-------------+----------+----------+
```

For *<GPIO_OUT for clock pad>*, use the `GPIO_OUT` value for `clk_fwd`.

The equations are:

<max calculation> = 2 + 2.205 - 2.205 = 2

<min calculation> = 2 + 1.470 - 1.470 = 2

In this example, the RTL is using a divided down clock, `divclk`, which is only used to drive the `clk_fwd` signal off chip. Therefore, the `set_input_delay` constraint is relative to that generated clock. See **Example: FPGA Forwarded Clock** on page 65 for a more complete example.

The resulting constraints are:

```
set_output_delay -clock divclk -reference_pin clk_fwd -max 2 [get_ports {o}]
set_output_delay -clock divclk -reference_pin clk_fwd -min 2 [get_ports {o}]
```

Timing Exceptions

Timing exceptions are constraints that override the default behavior between clocks. These constraints are:

- `set_false_path`—Cuts the path between the source and destination.
- `set_max_delay`, `set_min_delay`—Overrides the required time needed from the source to the destination for the specified paths.
- `set_multicycle_path`—Changes the clock edges used for the required timing calculation from the source to the destination.

Tip: Refer to Example: Clock-to-Clock Path with Control on page 44 for an example use case.

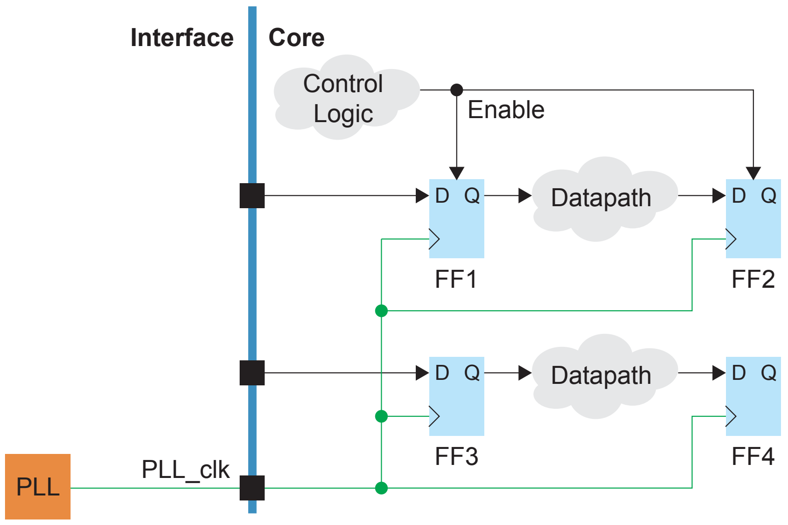
When working with exceptions, if the same path has more than one exception, the constraints are prioritized in the following order:

- `set_clock_groups`
- `set_false_path`
- `set_max_delay and set_min_delay`
- `set_multicycle_path`

Example: Clock-to-Clock Path with Control

The following figure shows a use case in which a specific clock-to-clock path in a design can have special control logic. The path from `FF1` to `FF2` can have a different timing exception compared to other clock-to-clock paths in the design. You define these timing exceptions with `set_false_path`, `set_max_delay`, `set_min_delay`, or `set_multicycle_path` SDC commands.

Figure 24: Timing Exception Example



Understanding False Paths

You use the `set_false_path` constraint to tell the timing analyzer not to analyze (that is, to cut) a path. For example, a clock may only toggle some of the time, and you do not want software to try to optimize timing for it.

You can cut paths between entire clock domains or individual points on the timing graph. If you want to completely cut the path between two clock domains, you should instead use the `set_clock_groups` constraint.

Understanding Min and Max Delays

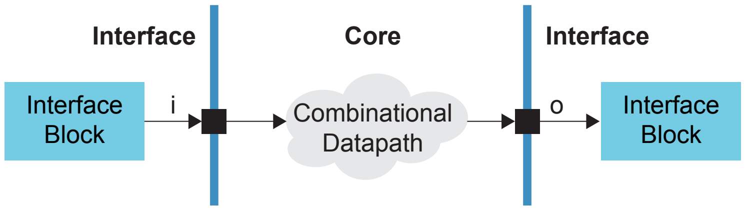
The `set_min_delay` and `set_max_delay` constraints override the timing requirements derived from your clock constraints. These settings tighten or relax the timing requirements for the paths. For example, you could use these constraints to try to minimize skew within a bus of signals.

Important: Using `set_min_delay` and `set_max_delay` is a very risky way to close timing because you can mask real setup and hold time violations unintentionally. If you use `set_max_delay` or `set_min_delay` to override the default clock-to-clock constraint calculated by the software, the software honors your input and does not give any errors. However, the issue would likely appear on your board as a setup or hold violation. This method is especially risky when used with beneficial skew.

Asynchronous Paths

The `set_max_delay` and `set_min_delay` SDC commands support setting a combinational delay on an asynchronous path between ports. This path does not associate with any clock. See Figure 25 on page 45. Clock latency and clock uncertainty are not considered for asynchronous data paths.

Figure 25: Asynchronous Data Path between Ports in Core



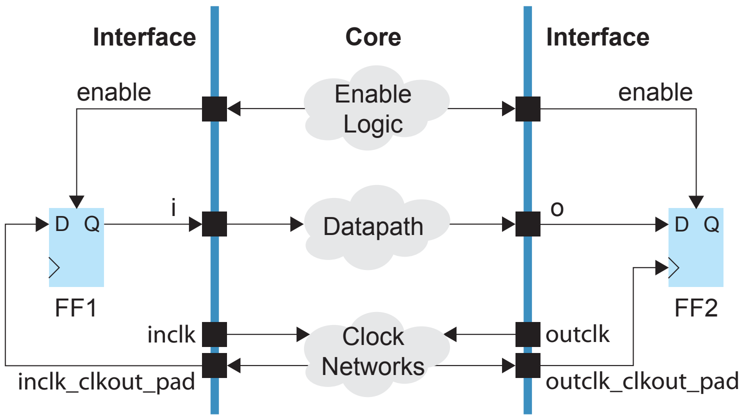
The constraints that represent this example are:

```
set_max_delay -from i to o <max delay>
set_min_delay -from i -to o <min delay>
```

Synchronous Paths

If you specify a maximum delay or a minimum delay for synchronous ports, you must also specify the clock domains for both `-from` and `-to` ports. In the following example, the input and output ports of the core are connected to flipflops in the interface and special enable logic controls the clock relationship.

Figure 26: Synchronous Data Path between Ports in Core



The constraints that represent this example are:

```
create_clock -period <inclk period> -name inclk [get_ports inclk]
create_clock -period <outclk period> -name outclk [get_ports outclk]
set_input_delay -max <input max delay> -clock inclk -reference_pin <inclk_clkout_pad>
set_input_delay -min <input min delay> -clock inclk -reference_pin <inclk_clkout_pad>
set_output_delay -max <output max delay> -clock outclk -reference_pin <outclk_clkout_pad>
set_output_delay -min <output min delay> -clock outclk -reference_pin <outclk_clkout_pad>
set_max_delay -from -i -to o <max delay>
set_min_delay -from i -to o <min delay>
```

Notice that the clock out pads are reference pins for the `set_input_delay` and `set_output_delay` commands. The `set_max_delay` and `set_min_delay` commands override the default clock-to-clock constraints calculated by the system. The clock path latency and clock uncertainty are considered for synchronous ports.

Mixed Asynchronous and Synchronous Paths

The Efinity software issues a warning and ignores the `set_max_delay` and `set_min_delay` SDC commands if one of the `-to`/`-from` ports is synchorous and the other is synchronous. The following example only has a clock associated with the `-from` port:

```
create_clock -name inclk -period 10.00 [get_ports inclk]
set_input_delay -clock inclk 0.1 [get_ports i]
set_max_delay 10 -from [get_ports i] to [get_ports o]
```

The software gives the following warning and ignores the `set_max_delay` command.

```
Ignore the set_max_delay (<sdc_file>:<line#>) constraint due to unconstrained
 port in -to
```

The following example only has a clock associated with the -to post:

```
create_clock -name outclk -period 10.00 [get_ports outclk]
set_output_delay -clock outclk 0.2 [get_ports o]
set_max_delay 10 -from [get_ports i] -to [get_ports o]
```

The software gives the following warning and ignores the `set_max_delay` command.

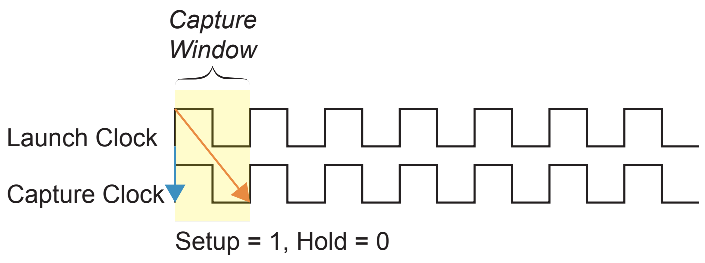
```
Ignore the set_max_delay ((<sdc_file>:<line#>) constraint due to unconstrained
 port in -from
```

Understanding Multicycle Constraints

In a default single-cycle clock relationship, the two clocks are in phase and toggle together. The default setup and hold represent a one clock cycle *capture window* and is the same as setting a constraint of setup = 1 and hold = 0. The hold is checking one clock cycle before the capture clock edge. When you use the `set_multicycle_path` constraint, you are adjusting the capture window by shifting it, widening it, or both.

If you do not use a multicycle constraint, the software assumes you want the default, single-cycle relationship.

Figure 27: Default Single-Cycle Relationship



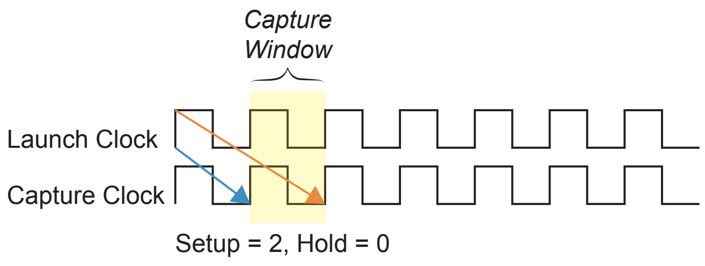
The constraints that represent the default are:

```
set_multicycle_path -setup -from a -to b 1
set_multicycle_path -hold -from a -to b 0
```

Shifted Capture Window

To shift the capture window you use a constraint for the clock setup. The hold is still one clock cycle before the capture clock edge; the software assumes the hold is 0. Therefore, the window is still one clock cycle.

Figure 28: Setup Constraint Shifts the Capture Window



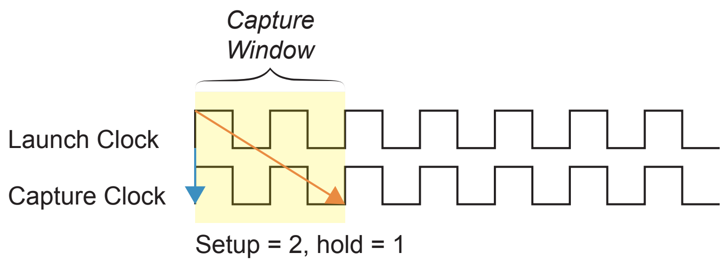
The constraints that represent this example are:

```
set_multicycle_path -setup -from a -to b 2
set_multicycle_path -hold -from a -to b 0
```

Shifted and Widened Window

To shift and widen the capture window you constrain the hold time as well as the setup time. A wider window allows multiple clock cycles to capture data. In the following example, the capture window is two clock cycles.

Figure 29: Setup and Hold Constraints Shift and Widen the Capture Window



The constraints that represent this example are:

```
set_multicycle_path -setup -from a -to b 2
set_multicycle_path -hold -from a -to b 1
```

If *n* is equal to *m*, then the constraint would simply be:

```
set_multicycle_path -setup -from a -to b n
set_multicycle_path -hold -from a -to b n-1
```

To shift the window by *n* clock cycles with a window *m* cycles wide, use the equations:

- setup = *n*
- hold = *m* - 1

For example:

- *n* = 4, *m* = 3
- setup = 4
- hold = 3 - 1 = 2

These values give you a window that is shifted by 4 clock cycles and is 3 clock cycles wide.

```
set_multicycle_path -setup -from a -to b 4
set_multicycle_path -hold -from a -to b 2
```

Constraints between Fast and Slow Clocks

When the launch and capture clocks have the same frequency and phase, it does not matter which clock waveform you use to calculate the setup and hold; the result will be the same. However, when the clock frequencies are different, you need to specify which clock waveform you want to use for the setup and hold calculation using the `-start` and `-end` modifiers. You cannot use both `-start` and `-end` at the same time.

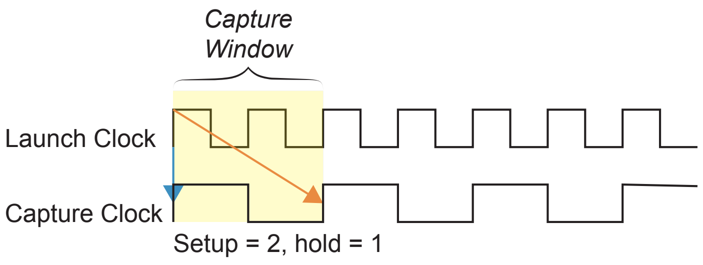
- `-start` uses the launch clock for the calculation.
- `-end` uses the capture clock for the calculation.

For setup, `-start` moves the launch edge backwards and `-end` moves the capture edge forward. The default is `-end`.

For hold, the `-start` moves the launch edge forward and `-end` moves the capture edge backward. The default is `-start`.

When the launch clock is faster than the capture clock, you need to ensure that the `set_multicycle_path` constraint is applied to the launch clock. For the setup constraint, you need to include `-start`. For hold, `-start` is the default so you do not need to include it.

Figure 30: Launch Clock Faster than Capture Clock

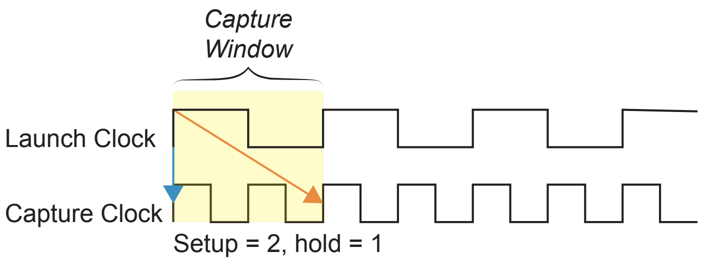


The constraints that represent this example are:

```
set_multicycle_path -setup -start -from a -to b 2
set_multicycle_path -hold -from a -to b 1
```

When the launch clock is slower than the capture clock, you need to ensure that the `set_multicycle_path` constraint is applied to the capture clock. For the setup constraint, you need `-end`, which is the default, so you do not need to include it. For hold, include `-end`.

Figure 31: Launch Clock Slower than Capture Clock



The constraints that represent this example are:

```
set_multicycle_path -setup -from a -to b 2
set_multicycle_path -hold -end -from a -to b 1
```

SDC Warnings

While compiling, the Efinity® software displays messages and warnings in the Console. These messages also are available in the <*project name*>**.place.out** file in the **outflow** directory. You should review all SDC messages and adjust your constraints as needed. Warning messages flag issues that can affect timing closure.

Common Mistakes

This topic describes some common mistakes that affect timing.

Latches and Combinational Loops

If you do not assign an output for all possible conditions in an `if` or `case` statement (that is, incomplete assignment), the software infers a latch. Trion®, Topaz, and Titanium FPGAs do not support latches natively in hardware. The Efinity® synthesis tool infers look-up tables (LUTs) to provide latch behaviour.

You also may create a latch accidentally when you meant to use a flipflop. From a timing perspective, the latch causes a combinational loop and the timing graph cannot have a loop. Therefore, if the software detects a combinational loop it cuts the loop at an arbitrary point.

To resolve this issue, make sure `if` and `case` statements are complete and use flipflops instead of latches.

Unintended Virtual Clock

If you create a clock with the *-name* option without a target (e.g., `get_ports`), you create a virtual clock. Make sure to use a target unless you really want a virtual clock.

```
create_clock -period 40 -name clk_in [get_ports clk_in]    # defined clock
create_clock -period 40 -name virtual_clk                  # virtual clock
```

The Efinity software prints an info message when it finds a virtual clock definition so you can double check your constraints. See **Virtual Clocks** on page 14 for more information.

Undefined Clocks

If you have an SDC file and do not define all clocks, the software cannot perform timing analysis on any logic controlled by those clocks. This situation leads to unoptimized results. Therefore, you should always define all clocks in your design.

Incorrect Constraint Order

The order of constraints in the SDC file is important. If you use the wrong order you get unintended results. For example, always define a clock before using `set_input_delay` or `set_output_delay` constraints for that clock. Refer to **About SDC Files** on page 8 for more information about the expected constraint order.

SDC Tips and Tricks

The following sections provide some tips for working with SDC files.

SDC Syntax

In SDC syntax:
- # starts a comment; remaining text on this line is ignored.
- \ at the end of a line indicates that a command wraps to the next line.

Wildcard Commands

An `*` indicates a wildcard. Use `*` by itself to match all signals, or use it to create a partial wildcard. For example `clk*` would match `clk` and `clk2`.

Example: Constraining with Wildcards

You want to constrain all `Oled` signals with respect to clock `clk`. The resulting constraints are:

```
set_input_delay -max 10.214 -clock clk Oled*
set_input_delay -min 3.607 -clock clk Oled*
```

Regular Expressions

You can use regular expressions (in the Perl regular expression format) with the object specifier. You must encapsulate the object specifiers in square brackets []; arguments must be enclosed in curly braces {}.

To use Perl regular expressions, include the `-regexp` option in your command. Escape Perl regular expression characters if the provided string argument contains those characters.

Example: Using Regular Expressions

Simple wildcard:

```
get_pins y_r[*]~FF|D
```

Using Perl regular expressions:

```
regexp get_pins -regexp { y_r\[.*\]~FF|D }
```

Inverted Clocks

For an inverted external clock (one that uses the negative edge), include the `clock_fall` option in your `set_input_delay` or `set_output_delay` command.

Example: Inverting a Clock

```
set_output_delay -clock <clock> -clock_fall -max -3.1 <ports>
set_output_delay -clock <clock> -clock_fall -min -2.85 <ports>
```

Square Brackets in Clock Names

The Efinity software v2023.2 and higher supports square brackets in clock names. You do not need to use the `-name` option in your SDC constraint.

For versions of software prior to 2023.2, if your clock names have square brackets, you need to use the `-name` option in your SDC constraint (this is a known limitation in the software).

For example, the following constaints for `clk4096rx[1]_2`, `clk4096rx[2]_2`, and `clk4096rx[3]_2` will **NOT** work correctly:

```
create_clock -period 10 clk4096rx[1]_2
create_clock -period 10 clk4096rx[2]_2
create_clock -period 10 clk4096rx[3]_2
```

Instead use the `-name` options:

```
create_clock -period 10 -name clk4096rx1_2 [get_ports {clk4096rx[1]_2}]
create_clock -period 10 -name clk4096rx2_2  [get_ports {clk4096rx[2]_2}]
create_clock -period 10 -name clk4096rx3_2  [get_ports {clk4096rx[3]_2}]
```

SDC Constraints (Alphabetical)

The Efinity software supports the following SDC constraints. Options in square brackets [ ] are optional.

check_timing	set_clock_groups	set_input_delay
create_clock	set_bus_syntax	set_multicycle_path
create_generated_clock	set_clock_latency	set_max_delay
get_fanins	set_clock_uncertainty	set_min_delay
get_fanouts	set_false_path	set_output_delay
report_sdc		

create_clock Constraint

```
create_clock -period <float> [-waveform {rising_edge falling_edge}]  \
    [-name <clock name>] [<targets>] [-add]
```

This command defines a clock with the desired period (in ns) and waveform, and applies it to the target nodes. If you do not specify a target, the software considers the clock to be a virtual clock. You can use the virtual clock to constrain inputs and outputs to a clock external to the design. The tool does not support multiple clock assignments to the same target.

Note: You can refer to netlist clocks using regular expressions.

- `-period` indicates the clock period in ns.
- `-waveform` indicates the rising and falling edges (duty cycle) of the clock as two time values: the first rising edge and the next falling edge. If you omit the waveform option, the command creates a clock with a rising edge at 0 and a falling edge at the half period, which is equivalent to using `-waveform {0 <period/2>}`.
- `-name` indicates the clock name. If omitted, the software gives the clock the name of the first target.
- `-add` defines multiple clocks for the same target. First use `-name` to specify the new clock name. If you already used the same clock name or did not define it, the last SDC command overwrites the existing clock.

If you assign a virtual clock using the `create_clock` command, you must reference it elsewhere in a `set_input_delay` or `set_output_delay` constraint.

For examples, see
- Defining Clocks on page 11
- Example: Dynamic Multiplexers and create_clock -add on page 64

Tip: The timing analysis and place-and-route runtime is affected by the number of clocks you define in your SDC file. Therefore, if possible, you should only define the most critical clocks to reduce the runtime.

create_generated_clock Constraint

```
create_generated_clock -source <source clock object> [-divide_by <factor> | \
    -multiply_by <factor>] [-duty_cycle <percentage>] [-invert] \
    [-name <virtual clock name>] <target> [-master_clock <master clock>] \
    [-phase <degree>][-offset <offset value>] \
    [-edges <n1 n2 n3>] [-edge_shift <n1 n2 n3>] \
    [-add]
```

This constraint is useful for designs with internally generated clock signals because it provides more accurate timing analysis. First, use the `create_clock` constraint on the source clock that generates the internal clock signal. Then, use this constraint.

- `-source` is the generated clock's source port, pin, or net.
- `-divide_by` is the division factor.
- `-multiply_by` is the multiplication factor.
- `-duty_cycle` is the duty cycle as a percentage of the clock period.
- `-invert` inverts the clock.
- `-name` is the name of the generated clock.
- *<target>* is the name of the net that implies that it is an internally generated clock signal.
- `-master_clock` specifies the master clock for the generated clock target.
- `-phase` is the phase shift in degrees based on the master clock (the default is 0).
- `-offset` is the absolute time shift in ns relative to the master clock.
- `-edges` is a list of three values that specify the first rising clock edge, the first falling clock edge, and the second rising clock edge for the generate clock's edges as they relate to the edges of the master clock waveform.
- `-edge_shift` is a list of three values that specify the edge shift in ns relative to the edges defined in `-edges`.
- `-add` defines multiple clocks for the same target. First use `-name` to specify the new clock name. If you already used the same clock name or did not define it, the last SDC command overwrites the existing clock.

You can use `-phase` and `-offset` with the `-divide_by`, `-multiply_by`, and `-invert` options.

You can use `-edges` and `-edge_shift` together; however, you cannot use these options with `-divide_by`, `-multiply_by`, or `-invert`.

See Defining Clocks on page 11 for additional examples.

Tip: The timing analysis and place-and-route runtime is affected by the number of clocks you define in your SDC file. Therefore, if possible, you should only define the most critical clocks to reduce the runtime.

Example: Using -phase and -offset Options

In this example, `clkA` has a 50% duty cycle and a clock period of 10 ns. The `clkA` waveform is `{0 5}`. The `divclk` waveform is derived from `clkA` with a divide by 2, resulting in a clock period os 20 ns. With a phase shift of 45 degree, (20 ns * 45) / 360 = 2.5 ns plus a 4 ns offset, resulting in a 6.5 ns shift. The resulting divclk waveform is `{6.5 16.5}`.

```
create_clock -period 10 clkA
creage_generated_clock -source [get_ports clkA] -divide_by 2 \
    [get_pins divclk|q] - name divclk -phase 45 -offset 4
```

Example: Using -edge and -edge_shift Options

In this example, `-edge {1 2 3}` refers to the first rising_edge, the first falling_edge, and the second rising edge of the generated clock relative to the source clock. In this example, the generated clock's:

- First rising edge is the first edge of the source clock
- First falling edge is the second edge of the source clock
- Second rising edge is the third edge of the source clock

`-edge_shift` indicates the amount of shifted delay (positive or negative) in ns based on the `-edge` values.

```
create_clock -name clkin -period 10 [get_ports clkin]
create_generated_clock -name clkshift -source [get_clocks clkin] \
    -edges {1 2 3} -edge_shift {2.5 0 2.5} [get_ports divclk|Q]

# First rising edge:   0 ns + 2.5 ns = 2.5 ns
# Second rising edge:  5 ns + 0 ns = 5 ns
# Second rising edge:  10 ns + 2.5 ns = 12.5 ns
```

get_fanouts Constraint

```
get_fanouts [-no_logic] -through <names> <start point>
```

This command returns a string of fanout ports and registers. This command supports the following options

- `-no_logic` if used, the software does not follow combinational timing arcs
- `-through` pins, cells or nets (see -through Option on page 62 for supported use cases)
- <*names*> can be a net, cell, or pin
- <*start point*> is a port, pin, or net

By default, the get_fanouts command traces through combinational timing arcs.

The following example sets different multicycle path constraints for registers based on the flipflops' enable signals. The get_fanout SDC command finds which registers are controlled by the specific enable signals.

```
create_clock -period 10.00 -name clkin [get_ports clkin]
set_multicycle_path -setup -from [get_clocks clkin] -to [get_fanouts ce_y] 4
set_multicycle_path -hold -from [get_clocks clkin] -to [get_fanouts ce_y] 3
set_multicycle_path -setup -from [get_clocks clkin] -to [get_fanouts ce_x] 2
set_multicycle_path -hold -from [get_clocks clkin] -to [get_fanouts ce_x] 1
```

Note: This constraint returns a string, not an Efinity Tcl object. Therefore, you cannot use it in a nested format with object specifiers such as `get_nets` or `get_cells`.

get_fanins Constraint

```
get_fanins [-no_logic] -through <names> <start point>
```

This command returns a string of fanout ports and registers. This command supports the following options:

- `-no_logic` if used, the software does not follow combinational timing arcs
- `-through` pins, cells or nets (see -through Option on page 62 for supported use cases)
- <*names*> can be a net, cell, or pin
- <*start point*> is a port, pin, or net

By default, the get_fanins command traces through combinational timing arcs.

```
# Given a set of register clock pins, find the clock source
set reg_pattern "o*\[*\]~FF"
set reg_cells [ get_cells ${reg_pattern} ]
foreach reg $reg_cells {
    # use get_pins to look for clock pin with pattern end with "|CLK"
    set pin_pattern "${reg}|CLK"
    set reg_pins [ get_pins $pin_pattern ]

    # Find the clock source from register clock pins
    foreach clk_pin $reg_pins {
        set fanins [get_fanins $clk_pin]
        puts "CLK_PIN: $clk_pin $fanins"
    }
}
```

Note: This constraint returns a string, not an Efinity Tcl object. Therefore, you cannot use it in a nested format with object specifiers such as `get_nets` or `get_cells`.

set_bus_syntax_mode Command

```
set_bus_syntax_mode <mode>
```

Tcl normally interprets square brackets as containing commands. For bus names, this functionality means that you need to escape any square brackets that do not contain commands, specifically bus names. `set_bus_syntax_mode` changes how the brackets are processed, so you can use square brackets for bus indic(es) without having to use escape characters.

The `set_bus_syntax_mode` command has one option, <*mode*>, which is `natural` (you do not need to use escape characters) or `disabled`. The default is `natural`.

For example:

- In `natural` mode, you can use `bus_name[0]` (single-dimensional bus) or `bus_name[0][1]` (multi-dimensional bus).
- In `disabled` mode, you need to use `bus_name\[0\]`.

You can disable natural bus syntax processing at any point in your SDC script with the command `set_bus_syntax_mode disabled`. The software parses all bus names after the disabled command with normal Tcl syntax. Use `set_bus_syntax_mode natural` to turn it on again. For example:

```
# Using natural mode
set bus_index 10
set bus_name bus[$bus_index]

# disable natural mode
set_bus_syntax_mode disabled
set bus_index 10
set bus_name bus\[$bus_index\]

# Turn natural mode back on
set_bus_syntax_mode natural
```

You can also use the `set_bus_syntax_mode` command in the Tcl console (in the GUI and at the command line). The setting does not persist across sessions. That is, if you close and then re-open the Tcl Command Console (GUI or command-line), you need to use the `set_bus_syntax_mode` again to ensure the correct setting.

The following example shows commands in the Tcl Command Console:

```
set x 10
puts var[$x]
puts var[*1]
set_bus_syntax_mode disabled
puts var\[$x\]
set_bus_syntax_mode natural
puts var[$x]
```

set_clock_groups Constraint

```
set_clock_groups [-exclusive | -asynchronous ] -group {<clock>} [-group {<clock>} -group ...]
```

- `-exclusive` indicates a mutually exclusive clock
- `-group` indicates a clock list
- `-asynchronous` specifies asynchronous clocks

This command instructs the timing analyzer to not analyze paths between one or more specified groups of clock domains in either direction. You can use this command with netlist or virtual clocks in any combination. The `set_clock_groups` constraint is equivalent to a `set_false_path` constraint between the clocks in different groups. For example, the command

```
set_clock_groups -exclusive -group {clk} -group {clk2 clk3}
```

is equivalent to

```
set_false_path -from [get_clocks{clk}] -to [get_clocks{clk2 clk3}]
set_false_path -from [get_clocks{clk2 clk3}] -to [get_clocks{clk}]
```

If you specify only one clock group, it cuts all paths to and from the specified clock domain(s) to all others.

Note: If you do not specify either `-exclusive` or `-asynchronous`, the software defaults to `-exclusive`.

See Setting Constraints for Unrelated Clocks on page 26 for examples.

set_clock_latency Constraint

```
set_clock_latency [-clock <names>] [-rise] [-fall] -source <latency>
    [-setup] [-hold] <latency> <target clock, port, or pin>
```

The source latency represents the time it takes (in ns) for a clock signal to get from the source to the destination such as the delay from an oscillator to the FPGA's input pad. You can only set the clock source latency for clock and clock source pins. You musty specify a source. If do no specify -rise or -fall, the latency is applied to both clock edges.

- `-clock` Specifies a list of clocks associated with the latency assigned to the specified clock source.
- `-rise` Defines the latency for the rising clock edge.
- `-fall` Defines the latency for the falling clock edge
- `-source` Defines the specified <*latency*> as the source latency. This argument is required.
- `-setup` Define the clock edge delay for setup analysis
- `-hold` Define the clock edge delay for hold analysis

```
set_clock_latency -source -rise -0.5 [get_ports clk200]
set_clock_latency -source -fall -0.4 [get_ports clk200]
set_clock_latency -source -fall 0.1 {clk25~FF|Q} -clock {clk200 clk50}
set_clock_latency -source 0.7 [get_pins clk_50~FF|Q]
```

set_clock_uncertainty Constraint

```
set_clock_uncertainty [-setup] [-hold] [-from <clock>] [-rise_from <clock>] \
    [-fall_from <clock>] [-to <clock>] [-rise_to <clock>] [-fall_to <clock>] \
    <uncertainty>
```

This constraint specifies the uncertainty for clocks or clock-to-clocks transfers. The tool added the specified uncertainty value to the derived uncertainty. If you do not specify source or destination clocks, the tool applies the uncertainty to all clocks in the design. If you do not specify a setup or hold, the tool uses the uncertainty value for both setup and hold.

- `-setup` is the clock uncertainty for setup analysis
- `-hold` is the clock uncertainty for hold analysis
- `-from` source clock
- `-rise_from` source clock with rising edge
- `-fall_from` source clock with falling edge
- `-to` destination clock
- `-rise_to` destination clock with rising edge
- `-fall_to` destination clock with falling edge
- <*uncertainty*> is the clock uncertainty value

See How to Set Clock Uncertainty on page 29.

set_false_path Constraint

```
set_false_path [-setup] [-hold] -from <start point> -through <names> -to <end point>
```

This command cuts paths unidirectionally:
- between clock domains
- from start or end points

Points can be a register or an I/O. If you do not specify a setup or hold, the tool cuts both setup and hold.

- `-setup` is the false path for setup analysis
- `-hold` is the false path for hold analysis
- `-to` the clock domain destination, I/O, or register end point
- `-from` the clock domain source, I/O, or register start point
- `-through` pins, cells or nets (see -through Option on page 62 for supported use cases)
- <*start point*> is the the clock domain source, I/O, or register start point
- <*end point*> is the clock domain destination, I/O, or register end point
- <*names*> is a clock domain source/destination, I/O, or register start/end point

Note: Use the set_clock_groups constraint if both directions are false paths.

See Setting Constraints for Unrelated Clocks on page 26 for examples.

set_input_delay and set_output_delay Constraints

```
set_input_delay -clock <clock> [clock_fall] [-max] [-min] <delay> [-reference_pin] \
   <ports> [-add_delay]
set_output_delay -clock <clock> [clock_fall] [-max] [-min] <delay> [-reference_pin] \
   <ports> [-add_delay]
```

- Use `set_input_delay` to analyze timing paths from input I/Os.
- Use `set_output_delay` for timing paths to output I/Os. If you do not specify these commands in your SDC, paths from and to I/Os will not be analyzed.

These commands constrain each I/O pad specified to be timing-equivalent to a register clocked on the clock specified after `-clock`. This register can be either a clock signal in your design or a virtual clock that does not exist in the design but that you use to specify the I/O timing.

The command also adds <*delay*> through each pad, thereby tightening the time constraint along paths traveling through the I/O pad. You can use this additional delay to model board-level delays. `-max` is the setup constraint, `-min` is the hold constraint; if you specify neither, the tool uses <*delay*> for both max and min.

- `-clock` is the clock name
- `-clock_fall` is the input delay relative to the clock's falling edge
- `-max` is the maximum data arrival time
- `-min` is the minimum data arrival time
- `<delay>` is the delay value
- `-reference_pin` is an optional flag to include the clock delay to the pin when calculating the input or output delay (Titanium only). When you generate constraints in the Interface Designer, the software automatically includes this option for synchronous interface signals (such as GPIO or LVDS).
- `<ports>` is the list of input or output ports
- `-add_delay` specifies any additional delay or clock condition for the port. If you do not specify this option, then any latter set_input_delay or set_output_delay command replaces the prior commands.

See Constraining I/O on page 30.

set_max_delay and set_min_delay Constraints

```
set_max_delay -from <start point> -through <names> -to <end point> <delay>
set_min_delay -from <start point> -through <names> -to <end point> <delay>
```

These commands override the default timing constraint (calculated using the information from `create_clock`) with a user-specified delay. This constraint may produce unexpected results.

- `-to` the clock domain destination, I/O, or register end point
- `-from` the clock domain destination, I/O, or register end point
- `-through` pins, cells, or nets (see -through Option on page 62 for supported use cases)
- <*start point*> is the clock domain source, I/O, or register start point
- <*end point*> is the clock domain destination, I/O, or register end point
- `<delay>` is the delay value in ns

Important: Using `set_min_delay` and `set_max_delay` is a very risky way to close timing because you can mask real setup and hold time violations unintentionally. If you use `set_max_delay` or `set_min_delay` to override the default clock-to-clock constraint calculated by the software, the software honors your input and does not give any errors. However, the issue would likely appear on your board as a setup or hold violation. This method is especially risky when used with beneficial skew.

set_multicycle_path Constraint

```
set_multicycle_path [-setup] [-start] [-hold] [-end] -from <start point>] \
    -through <names> -to <end point> <value>
```

This command creates a multicycle at the clock domain level. It adds (<*value*> - 1) times the period of the destination clock to the default setup time constraint. If you do not specify `-setup` or `-hold`, the tool applies the constraint to both.

- `-setup` applies the multicycle value to setup analysis
- `-hold` applies the multicycle value to hold analysis
- `-start` the multicycle value is relative to the source clock
- `-end` the multicycle value is relative to the destination clock (default)
- `-to` the clock domain destination, I/O, or register end point
- `-from` the clock domain source, I/O, or register start point
- `-through` pins, cellls, or nets (see -through Option on page 62 for supported use cases)
- <*value*> is the multicycle value

-through Option

The `set_false_path`, `set_max_delay`, `set_min_delay`, and `set_multicycle_path` commands have a `-through` option; These use cases are supported:

```
set_false_path -through [get_pins <pin name1>]
set_false_path -through [get_pins {add*|CI}]
set_false_path -through [get_pins <pin name1>] -through [get_pin <pin name2>]
set_false_path -through [get_cells {add*}]
set_false_path -through [get_cells <cell name1>] -through [get_cells <cell name2>]
```

However, the `-through` option cannot be used with multiple `get_*` commands (i.e., `get_pins`, `get_cells`, `get_nets`). Cases like the following are not supported:

```
set_false_path -through [[get_pins <pin name1>] [get_pin <pin name2>]]
```

Object Specifiers

SDC constraints and Tcl commands support explicit object specifiers. Implicit naming is implied if you do not use an object specifier with the constraint command. If you do not use an object specifier the software executes the search on the objects in the following order: nets, pins, cells.

The name you provide to the object specifier is based on the post-mapped design name; refer to the generated post-mapped Verilog HDL netlist—autogenerated by the software at the end of synthesis—for these names.

Note: The pipe (|) character is the separator between the instance name and the referenced port name.

In the Efinity software v2024.1 and higher, the object specifiers return Efinity Tcl objects or collections (in earlier versions they returned strings). This functionality allows you to use the results with Tcl list functions to navigate a netlist or generate custom timing reports. (See Tcl List Functions (Alphabetical) on page 98.)

- `all_clocks`—Retrieves a collection Efinity Tcl objects representing all of the design's clocks.
- `all_inputs`—Retrieves a collection of Efinity Tcl objects representing all of the design's input ports.
- `all_registers`—Retrieves a collection of Efinity Tcl objects representing all of the design's register instances.
- `all_outputs`—Retrieves a collection of Efinity Tcl objects representing all of the design's output ports.
- `get_cells [-regexp] [<filter>]`—Retrieves Efinity Tcl objects that match the specified cell name or pattern.[1]
- `get_clocks [-regexp] [<filter>]`—Retrieves Efinity Tcl objects that match the specified clock name or pattern. The tool looks first for the clock name, if it exists. Next, it checks the clock net name (includes virtual clocks).[1]
- `get_nets [-regexp] [<filter>]`—Retrieves a collection of Efinity Tcl objects that matches the specified net name or pattern.[1]

[1] By default, you do not need to escape brackets. However, if you use the -regexp option, you must escape all brackets.

- `get_pins [-regexp] [<filter>]`—Retrieve a collection of Efinity Tcl objects that matches the specified pin name or pattern. The pin name format is *<cell>*|*<port>*. Escape square brackets for cell names; you do not need to escape square brackets for ports if the port has bit indexing.(1)
- `get_ports [-regexp] [<filter>]`—Retrieve a collection of Efinity Tcl objects that matches the specified port name or pattern.(1)

Because these object specifiers return return Efinity Tcl objects, you can use them together in a nested format.

Example 1: Get a group of cells based on a pattern and the find the pins for the selected cells:

```
set pin_list [ get_pins [get_cells <pattern>]]
```

Example 2: Get a set of pins based on a pattern and then find the nets for those pins:

```
set net_list [get_nets [get_pins <pattern>]]
```

Example 3: Get a group of cells based on a pattern and then find the net connection based on the pins:

```
set nets_on_cell_pin [get_nets [get_pins [get_cells <pattern>]]]
```

Example 4: Get the nets for a group of ports:

```
set nets_on_ports [get_nets [get_ports out[*]]]
```

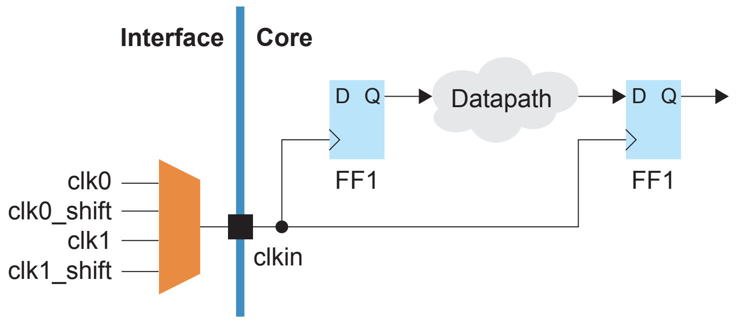
SDC Examples

The following sections provide a variety of examples on how to use the SDC constraints.

Example: Dynamic Multiplexers and create_clock -add

Titanium FPGAs have dynamic multiplexers that you can configure at run-time. You can choose which clock source is active in the Interface Designer. Only one of the four input clock sources is active at a time. You define multiple clocks at the core clock pad using the create_clock `-add` option. The following figure shows the corresponding timing constraint associated with this use case. It is good practice to define only the most critical clock, if possible, because adding more clocks to the system increases the runtime for timing analysis and place-and-route. Notice that the examples uses the `set_clock_groups` command because only one of the four clocks is active at a given time; therefore, from a timing perspective, those clocks are considered exclusive.

Figure 32: Dynamic Clock Multiplexer Example



Example: SDC Commands

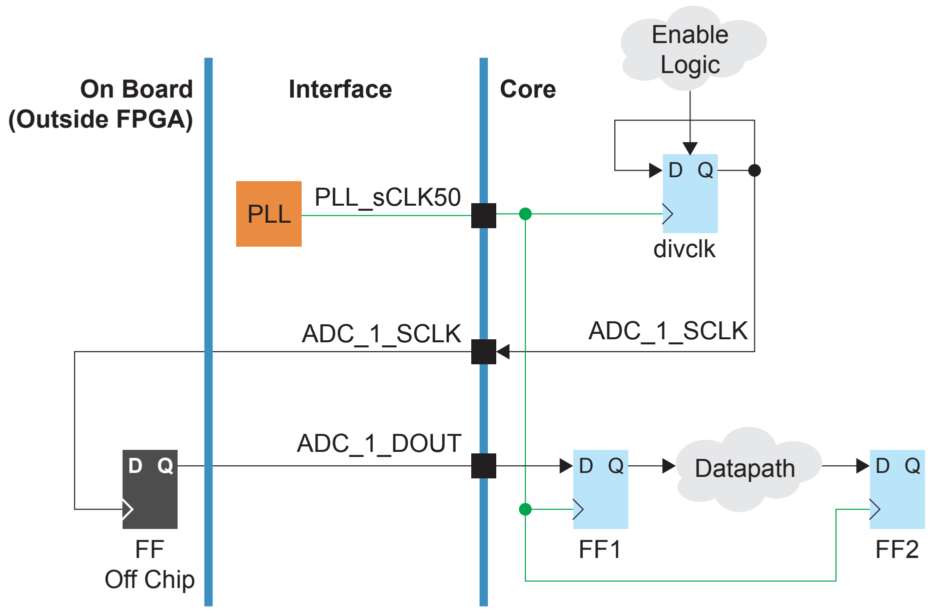
```
create_clock -period 10 [get_ports clkin] -name clk0

# The following constraints use -add to avoid overwriting the previous setting
create_clock -period 10 -waveform {2.5 7.25} [get_ports clkin] -name clk0_shift -add
create_clock -period 20 [get_ports clkin] -name clk1 -add
create_clock -period 20 -waveform {4 16} [get_ports clkin] -name clk1_shift -add

# The four clocks are exclusive because they cannot operate at the same time
set_clock_groups -exclusive -group {clk0} -group {clk0_shift} -group {clk1} -group {clk1_shift}
```

To learn more about the dynamic multiplexers, refer to:
- Titanium data sheets, "Driving the Global Clock Network" section.
- Titanium Interfaces User Guide, "Configuring the Dynamic Clock Multiplexers" section.

Example: FPGA Forwarded Clock

The following figure illustrates how to use a generated clock as a forward clock. The generated clock, `ADC_1_SCLK`, is generated from the PLL output `PLL_sCLK50` in the interface. `ADC_1_SCLK` is a forward clock to clock an off-chip flipflop outside the FPGA. The flipflop generates another signal, `ADC_1_DOUT`, that inputs back to the core.

Figure 33: Generated Clock Forward Clock Example



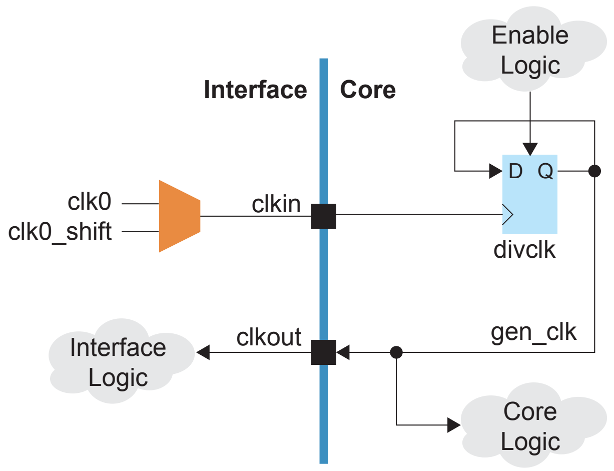
Example: SDC Commands

```
create_clock -name {outclk_0_pll}  -period 20 [get_ports PLL_sCLK50]
create_generated_clock -name ADC_1_SCLK -source [get_ports PLL_sCLK50] -divide_by 4 ADC_1_SCLK
set_input_delay -clock [get_clocks ADC_1_SCLK] -reference_pin [get_ports ADC_1_SCLK] \
    -max 24.000 [get_ports ADC_1_DOUT]
set_input_delay -clock [get_clocks ADC_1_SCLK] -reference_pin [get_ports ADC_1_SCLK] \
    -min 4.000 [get_ports ADC_1_DOUT]
```

Example: Generated Clock with Clock Multiplexer

The following figure illustrates multiple generated clocks defined at the same `divclk Q` pin. Because multiple clocks are defined at the same clkin pin, the generated clock pin also needs to have multiple clocks defined to address all cases. Notice that the examples uses the `set_clock_groups` SDC command because only one of the two clocks are active at a given time.

Figure 34: Dynamic Clock Multiplexer Example



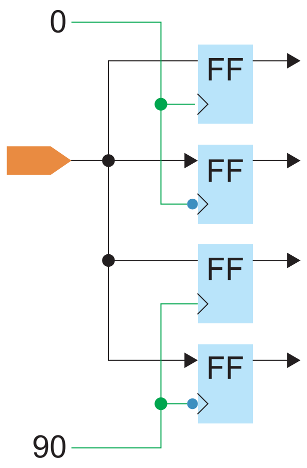
Example: SDC Commands

```
# Create 2 clocks and use the -add option to avoid overwriting the second one
create_clock -period 10 -name clk0 [get_ports clkin]
create_clock -period 10 -waveform {2.5 7.25} -name clk0_shift [get_ports clkin] -add

# There are 2 master clocks, so this example needs 2 generated clocks as well
create_generated_clock -source [get_ports clkin] -master_clock [get_clocks clk0] \
 -divide_by 2 [ get_pins divclk|Q ] -name gen_clk0
create_generated_clock -source [get_ports clkin] -master_clock [get_clocks clk0_shift] \
 -divide_by 2 [ get_pins divclk|Q ] -name gen_clk0_shift -add

# clk0 and it's generated counterpart operate independently from the clk0_shift and
# it's counterpart
set_clock_groups -exclusive -group {clk0 gen_clk0} -group {clk0_shift gen_clk0_shift}
```

Example: Soft SERDES

This example is for a soft SERDES. It has LVDS bypass mode sampling with 4 different clock phases (0, 90, 180 and 270 degrees). In this use case, you use the `-add_delay` option to set constraints for both edges of both clocks. If you did not use the `-add_delay` option then the second constraint would overwrite the first one.

Figure 35: Dynamic Clock Multiplexer Example



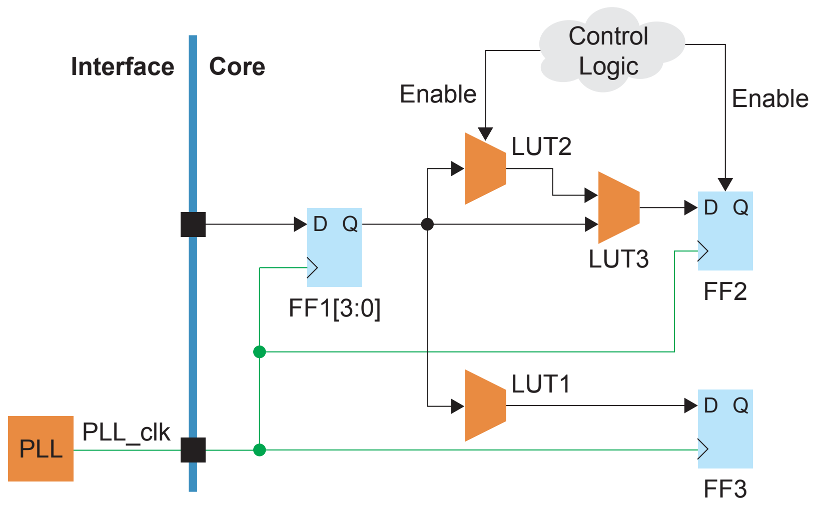
Example: SDC Commands

```
# these constraints are for 0 degrees
set_input_delay -clock i_sclk_000 -max 0.924 [get_ports {i_lvds_rxd}]
set_input_delay -clock i_sclk_000 -min 0.616 [get_ports {i_lvds_rxd}]

# The following constraints use -add_delay to avoid overwriting the previous setting

# these constraints are for 90 degrees,
set_input_delay -clock i_sclk_090 -max 0.924 [get_ports {i_lvds_rxd}] -add_delay
set_input_delay -clock i_sclk_090 -min 0.616 [get_ports {i_lvds_rxd}] -add_delay

# these constraints are for 180 degrees
set_input_delay -clock i_sclk_000 -clock_fall -max 0.924 [get_ports {i_lvds_rxd}] -add_delay
set_input_delay -clock i_sclk_000 -clock_fall -min 0.616 [get_ports {i_lvds_rxd}] -add_delay

# these constraints are for 270 degrees
set_input_delay -clock i_sclk_090 -clock_fall -max 0.924 [get_ports {i_lvds_rxd}] -add_delay
set_input_delay -clock i_sclk_090 -clock_fall -min 0.616 [get_ports {i_lvds_rxd}] -add_delay
```

Example: Disable Impossible Paths

In this example, special control logic exists for LUT2 and FF2. Therfore, the path from FF1 to FF2 though LUT2 can be ignored by timing analysis.

Figure 36: Set a False Path Example

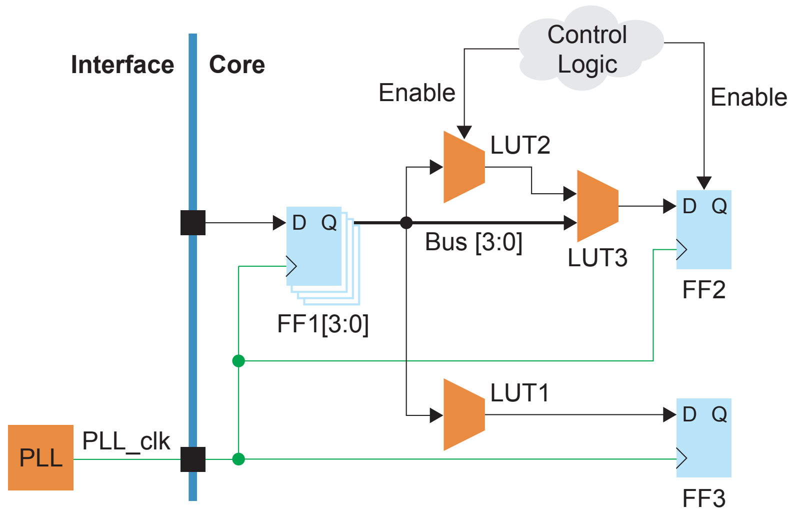


Example: SDC Commands

```
set_false_path -from [get_pins {FF1|CLK}] -through [get_cells LUT2] -to [get_pins {FF2|D]
```

In another example, a group of flipflops, FF1[3:0], use a naming pattern and the same connection as the FF1 flipflop in the previous example. In this case you can use a wildcard with the set_false_path command to constrain the whole group.

Figure 37: Set a False Path for a Bus Example



Example: SDC Commands

```
set_false_path -from [get_pins {FF1[*]|CLK}] -through [get_cells LUT2] -to [get_pins {FF2|D]
```

Interpreting Timing Results

When you compile your design, the Efinity® software generates the static timing analysis report. The first two sections of the report are about the clocks:

- Clock Frequency Summary
- Clock Relationship Summary

Note: The Efinity timing engine uses clock periods in ps; the units for timing in the report files are shown in ns. Therefore, all numbers are shown with 3 decimal places.

Clock Frequency Summary

The Clock Frequency Summary consists of the following data.

- The **User target constrained clocks** table lists the user constrained clocks; that is, the clocks you defined with the `create_clock` and `create_generated_clock` constraints in your SDC file. The clock target is the specific pin or port in the core timing netlist.
- The **Maximum possible analyzed clocks frequency** table shows the maximum clock frequency that each clock can achieve using the critical paths.
- The geomean is the geometric mean of the clock periods.

Note: This **Clock Frequency Summary** only shows the setup result. However, later, the report shows the most critical hold paths (you need to scroll down to after the detailed most critical setup path section).

Example: Clock Frequency Summary

```
---------- 1. Clock Frequency Summary (begin) ----------

User target constrained clocks
 Clock Name  Period (ns)  Frequency (MHz)     Waveform        Targets
    Oclk      99999.992         0.010     {0.000 49999.996}    {Oclk}
    Fclk          6.400       156.250     {0.000 3.200}        {Fclk}
    Sclk         12.800        78.125     {0.000 6.400}        {Sclk}

Maximum possible analyzed clocks frequency
 Clock Name  Period (ns)  Frequency (MHz)     Edge
    Oclk        0.584        1712.329*       (R-R)
    Fclk        0.749        1335.113*       (R-R)
    Sclk        1.038         963.391        (R-R)

* CAUTION: Frequency is limited to 1000.000 MHz by global clock network

Geomean max period: 0.769

---------- Clock Frequency Summary (end) ---------------
```

Clock Relationship Summary

The Clock Relationship Summary section lists the related clocks, their constraints, and the slack. The report shows measurements using the active clock edge. The number in the **Slack (ns)** column shows you how much margin you have for each clock relationship. The **Edge** column shows the active edge for the launch and capture clocks. R is rising edge triggered and F is falling edge triggered.

Important: If any of the clock relationships have negative slack, your design has **not** closed timing.

A timing path with negative slack indicates that there is insufficient time for the signal to arrive, resulting in design instability. When trying to close timing, do not adjust the clock period. Instead, adjust the constraints in your SDC file or modify your design.

Tip: You can quickly see if your design has closed timing in the Result pane in the **Timing** section. If the **Least Slack** number is negative, the design did not close timing.

Example: Did My Design Close Timing?

This summary shows that the design has not closed timing because the slack for the `Fclk` to `Fclk` relationship is negative (-0.875).

```
---------- 2. Clock Relationship Summary (begin) ----------

Setup (Max) Clock Relationship
  Launch Clock    Capture Clock   Constraint (ns)     Slack (ns)          Edge
      Oclk              Oclk         99999.992        99999.408          (R-R)
      Fclk              Fclk             6.400           -0.875          (R-R)
      Sclk              Sclk            12.800           11.762          (R-R)

Hold (Min) Clock Relationship
  Launch Clock    Capture Clock   Constraint (ns)     Slack (ns)          Edge
      Oclk              Oclk           0.000            0.101            (R-R)
      Fclk              Fclk           0.000            0.070            (R-R)
      Sclk              Sclk           0.000            0.109            (R-R)

NOTE: Values are in nanoseconds.

---------- Clock Relationship Summary (end) ---------------
```

Critical Paths

The final two report sections show the path detail reports for the maximum (setup) and minimum (hold) critical paths. The report only shows the most critical path for each relationship. To see additional paths, use the `report_timing` Tcl command (see report_timing Command on page 97). A typical path report consists of the following sections:

- *Header*—Specifies the launch and capture clock domains.
- *Path summary*—Lists the start and end points of a given path. It also shows the launch and capture clock information as well as the associatged clock edges, the slack, and a summary of the arrival and required time calculations.
- *Launch clock path*—The path the clock signals takes.
- *Data path*—The path the data signal takes
- *Capture clock path*—The path the capture clock takes.

The following example shows a snippet of the report for `Oclk`.

Example: Max Critical Path, Detail Report

```
################################################################################
Path Detail Report (Oclk vs Oclk)
################################################################################


++++ Path 1 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Path Begin    : Oled[2]~FF|CLK
Path End      : Oled[3]~FF|D
Launch Clock  : Oclk (RISE)
Capture Clock : Oclk (RISE)
Slack         : 99999.408 (required time - arrival time)
Delay         : 0.358

Logic Level             : 1
Non-global nets on path : 1
Global nets on path     : 0

Launch Clock Path Delay        : 2.342
+ Clock To Q + Data Path Delay : 0.474
----------------------------------------
End-of-path arrival time       : 2.816

Constraint                     :  99999.992
+ Capture Clock Path Delay     :      2.342
- Clock Uncertainty            :      0.110
---------------------------------------------
End-of-path required time      : 100002.224

Launch Clock Path
     name       model name   delay (ns)  cumulative delay (ns) pins on net    location
========================================================================================
Oclk              inpad        0.000             0.000              0        (334,318)
Oclk              inpad        0.110             0.110              6        (334,318)
Oclk              net          2.232             2.342              6        (334,318)
   Routing elements:
      Manhattan distance of X:214, Y:314
Oled[2]~FF|CLK    ff           0.000             2.342              6        (120,4)

Data Path
    name       model name   delay (ns)  cumulative delay (ns) pins on net    location
=======================================================================================
Oled[2]~FF|Q      ff          0.113             0.113              4        (120,4)
Oled[2]           net         0.304             0.417              4        (120,4)
   Routing elements:
      Manhattan distance of X:0, Y:1
LUT__77|in[2]     lut         0.054             0.471              4        (120,5)
LUT__77|out       lut         0.000             0.471              2        (120,5)
Oled[3]~FF|D      ff          0.003             0.474              2        (120,5)

Capture Clock Path
     name       model name   delay (ns)  cumulative delay (ns) pins on net    location
========================================================================================
Oclk              inpad        0.000             0.000              0        (334,318)
Oclk              inpad        0.110             0.110              6        (334,318)
Oclk              net          2.232             2.342              6        (334,318)
   Routing elements:
      Manhattan distance of X:214, Y:313
Oled[3]~FF|CLK    ff           0.000             2.342              6        (120,5)
```

Constraining Logic and Routing Manually (Beta)

The Efinity software v2022.1 and higher lets you assign logic to a specific location in the FPGA's core. With this method, you can place your design's logic manually instead of letting the software place it for you.

In v2022.2 and higher, you can also manually constrain routing to specific paths. When you constrain routing you also need to constraint the logic to which the nets connect.

Placing logic and/or routing manually is an advanced technique, so make sure that you fully understand the rules and restrictions as described in the following sections.

Important: These features are beta.

Tiles

The FPGA is made up of a grid of tiles. Most tiles are for logic/routing and others are for functions like RAM, multipliers, or DSP. The following table shows the types of tiles by family and their use.

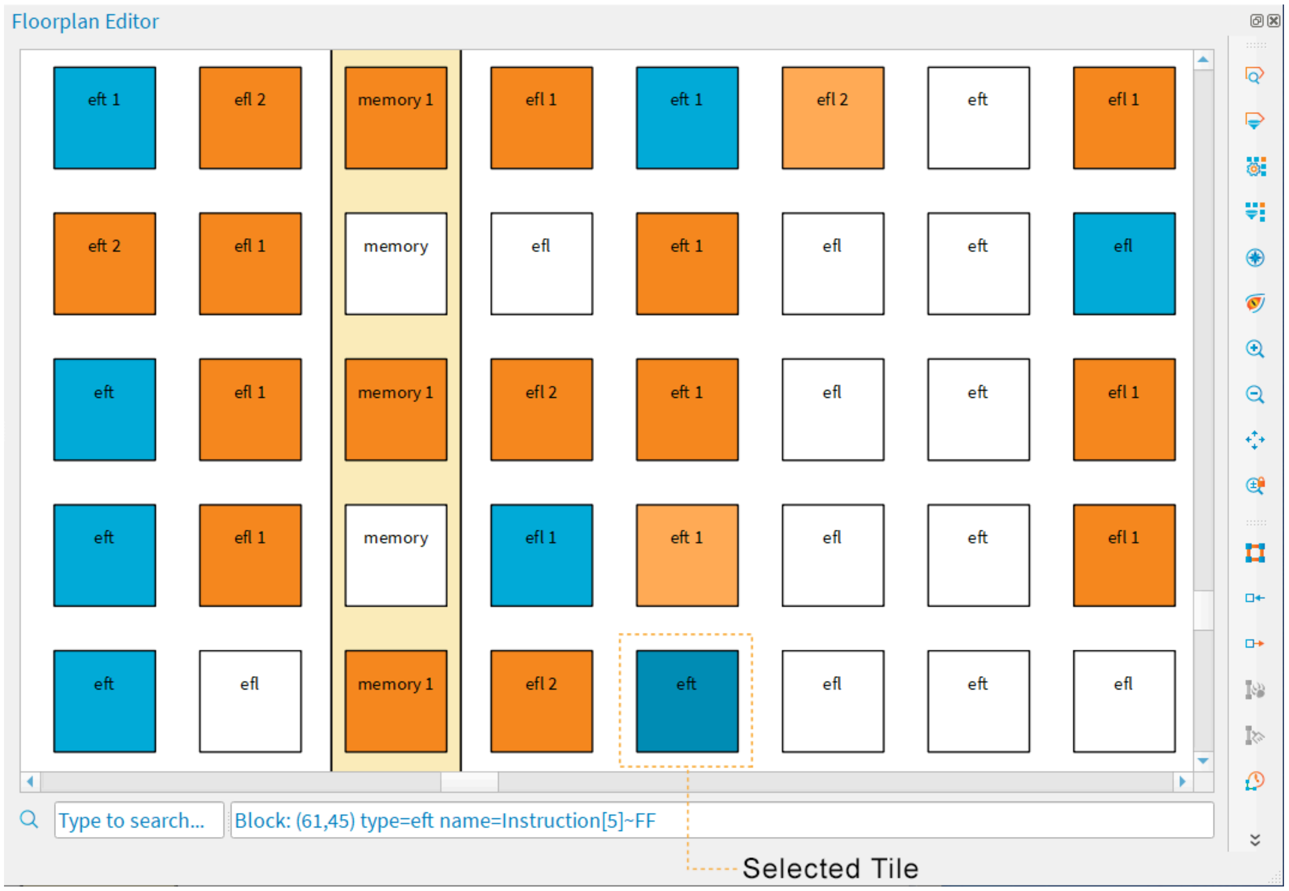
Table 1: FPGA Tile Types

Tile	Trion	Titanium	Used for
EFT	✓	✓	Logic and routing with register
EFL	✓		Logic and routing without register
EFM		✓	Logic, routing, register, and shift register
RAM	✓	✓	RAM blocks
MULT	✓		Multiplier blocks
DSP48		✓	DSP blocks

When you view your design's placement in the Floorplan Editor, you can click on a tile to view its type and other details. In the following figure, the selected blue tile is an EFT and is used for logic.

Tip: The Floorplan Editor provides a graphical way to find logic you want to constrain.

Figure 38: Tiles in the Floorplan Editor



Block: (61,45) type=eft name=Instruction[5]~FF

Notice that some tiles in the floorplan have a number. This number indicates how many routing lines are used in that tile. A tile used for logic (blue) can also be used as routing (indicated by the number). Orange means a tile is only used as routing.

Working with Primitives

During synthesis, the software maps your design's logic—LUTs, RAM, flipflops, etc.—to primitives. These primitives occupy specific locations (tiles or groups of tiles). Each tile has one or more sub-blocks in which to place a primitive. Placing multiple primitives into the same tile is called *packing*.

The following tables show the types of primitives, the tiles where you can place them, and the sub-blocks they can occupy.

Table 2: Mapping Trion Primitives to Tiles and Sub-Blocks

Tile	Sub-Block			
	0	1	2	3
EFT	EFX_LUT4 EFX_ADD	-	EFX_FF	-
EFL	EFX_LUT4 EFX_ADD	-	-	-
RAM	EFX_RAM_5K EFX_DPRAM_5K	Reserved	-	-
MULT	EFX_MULT	-	-	-

Table 3: Mapping Titanium Primitives to Tiles and Sub-Blocks

Tile	Sub-Block			
	0	1	2	3
EFT	EFX_LUT4 EFX_ADD EFX_COMB4	Reserved	EFX_FF	-
EFM	EFX_LUT4 EFX_ADD EFX_COMB4 EFX_SRL8	Reserved	EFX_FF	-
RAM	EFX_RAM10 EFX_DPRAM10	Reserved	-	-
DSP48	EFX_DSP48 EFX_DSP24 EFX_DSP12	EFX_DSP24 EFX_DSP12	EFX_DSP12	EFX_DSP12

The following table shows another view of the same mappings.

Table 4: Mapping Primitives to Tiles

Primitive	Compatible Tiles		Allowed Sub-Block Indices
	Trion	Titanium	
EFX_LUT4	EFT, EFL	EFT, EFM	0
EFX_ADD	EFT, EFL	EFT, EFM	0
EFX_COMB4	EFT, EFL	EFT, EFM	0
EFX_FF	EFT	EFT, EFM	2
EFX_SRL8	-	EFM	0
EFX_RAM_5K	RAM	-	0
EFX_DPRAM_5K	RAM	-	0
EFX_RAM10	-	RAM	0
EFX_DPRAM10	-	RAM	0
EFX_MULT	MULT		0
EFX_DSP48	-	DSP48	0
EFX_DSP24	-	DSP48	0, 1
EFX_DSP12	-	DSP48	0, 1, 2, 3

Finding Primitive Cell Names

When the software maps your design to primitives, it assigns a cell name to each instance. To view the primitive cell names:

- In the Dashboard's **Netlist** tab, click the Load Synthesized Netlist icon and expand **Leaf Cells**.
- Open the <*project*>**.map.v** file (in the Dashboard, go to **Result pane** > **Synthesis**). This file is in the project's **outflow** directory.

Enabling Manual Assignments

Because manual assignments are beta in the Efinity software v2022.1, v2022.2, and 2023.1, you must enable them with an **.ini** file.

1. Create a text file named **efx_pnr_settings.ini** and save it in your project directory.
2. Add the following line to the **.ini** file:

```
loc_assignment = <filename>.placeloc
```

When you synthesize your design, the software uses the assignments in the <*filename*>**.placeloc** file.

Assignment Rules

Follow these rules when creating assignments.

General Rules

- You can only constrain logic in the core (use the Interface Designer for I/O constraints).
- You can only constrain primitive cells. If two primitives cells *can* be packed together, you can assign them to the same location. The sub-block index must be unique for each primitive cell in a location. For example, if you assign four EFX_DSP12 primitives to the same tile, they must each have a different sub-block.
- The software does not pack manually assigned cells with unassigned cells. For example, if you place a EFX_DSP12 into a DSP tile at sub-block `0` and do not assign any other sub-blocks, the software will not pack any other DSP logic into that tile, leaving sub-blocks 1, 2, and 3 empty. Similarly, only assigning flipflops (which use sub-block 2) uses more overall resources because sub-block `0` is left empty.

Important: Because assigned and unassigned cells are not packed together, make sure to "fill up" the tile with logic. Otherwise you can end up using more tiles than needed.

Flipflops

- An EFX_FF can be packed alone or with its driver cell (EFX_LUT4, EFX_SRL8, EFX_ADD, or EFX_COMB4).
- An EFX_FF can only be packed with an EFX_SRL8 if they share CE and CLK inputs and if the EFX_FF does not have an inverted input.
- An EFX_FF cannot be packed if it has an inverted input connected to a multi-fanout net.

RAM, Multiplier, and DSP

- EFX_MULT, EFX_DSP48, and all RAM primitives cannot share a tile with any other cells.
- Two EFX_DSP24 primitives or up to four EFX_DSP12 primitives **not** connected by `CASCIN/CASCOUT` signals can be packed together and share a location.

Chains

EFX_DSP48, EFX_DSP24, EFX_DSP12, EFX_ADD, and EFX_SRL8 can form chains. If one cell in the chain is assigned a location, every other cell in the chain must also be assigned a location, in the correct order.

Creating a Location Assignment File

The location assignment file is a text file with the extension **.placeloc**. Each assignment is on a single line with tabs or spaces between the data:

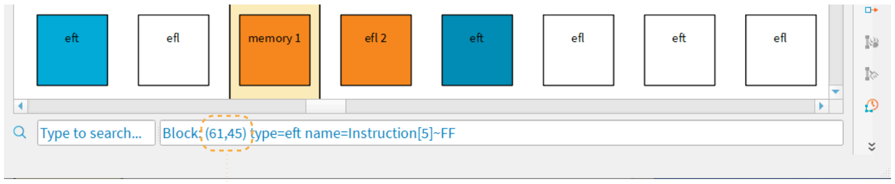
```
<block name>    <x>    <y>    <subblk>
```

- <*block name*> is the primitive cell name.
- <*x*> is the horizontal location.
- <*y*> is the vertical location.
- <*subblk*> is the sub-block location.

You must include all data for each assignment.

Any text following a # character is ignored (treated as a comment).

Tip: Use the Floorplan Editor to help you find the *x, y* coordinates for a tile. When you click a tile the coordinates are shown in ().



Block: (61,45) type=eft name=Instruction[5]~FF

x,y coordinates for the selected tile

To make it easier for you to create assignments, the Efinity software can dump all placement data into a file when placement finishes. You can copy and paste the primitive cells you want to constrain into your **.placeloc** file and then modify the *x, y* coordinates.

To dump the placement data, add the following line to your **efx_pnr_settings.ini** file and re-run the placer.

```
dump_placeloc = on
```

Important: Do NOT simply copy and paste the entire dump file into your **.placeloc** file or the software may not be able to perform placement efficiently. Only copy the primitives you want to constrain.

Example: LUT and Flipflop

The example packs an EFX_FF with its driver, LUT_A, an EFX_LUT4.

```
#block name  x   y   subblk
#----------  --  --  ------
LUT_A        3    3    0
FF_B         3    3    2    # LUT_A drives FF_B
```

Example: SRL8 Chain

This example assigns locations to every cell in an SRL8 chain.

```
#block name   x    y   subblk
#----------   --   --  ------
first_srl8     5   4     0
second_srl8    5   5     0
third_srl8     5   6     0
fourth_srl8    5   7     0
```

Example: Parallel Cascaded DSP Block

This example assigns locations to every EFX_DSP24 across two chains. There can be two EFX_DSP24 cells per DSP tile.

```
#block name      x    y   subblk
#----------      --   --  ------
chain0_dsp24_0   17   2      0
chain1_dsp24_0   17   2      1
chain0_dsp24_1   17   22     0
chain1_dsp24_1   17   22     1
```

Constraining Routing Manually (Titanium Only, Beta)

With the Efinity software v2022.2 and higher the router lets you manually constrain routing traces for Titanium FPGAs. This feature is beta.

Important: The feature is only supported for Titanium FPGAs.

After you compile your design once, you can lock down (or constrain) specific nets to specific paths. For any future compilations, the software routes these constrained nets in the same way. To constrain nets, you also need to constrain the logic to which the nets connect. See Constraining Logic and Routing Manually (Beta) on page 72 for information on making logic constraints.

You can combine constrained logic and constrained routing to preserve the placement and routing of a small part of your design, letting the rest change as you compile. This feature can be useful when logic (such as a sampling delay line) with very specific routing requirements must be locked down early in the design cycle. Additionally, this feature lets you preserve place and route for connections that have difficult timing constraints.

Routing Constraint Flow

To use routing constraints, follow this procedure:

1. Determine which nets and cells should be constrained.
2. Run the Efinity software, adjusting your design for each iteration, until the nets meet timing.
3. When the nets meet timing, use an **.ini** file to tell the software to save the placement and routing data to templates. (See Generate .rcf Template on page 79)
4. Do not make any changes to the design and re-compile.
   The software creates these template files:
   - Placement template <*project*>**.out.placeloc**
   - Routing template <*project*>**.rcf.template**

   The software also creates a routing traces file <*project*>**.troutingtraces**.
5. Move these three files out of the **outflow** directory, for example, move them up one level to the main project directory.
6. Copy and paste the cells and nets you want to constrain from the two template files to your own files. You do not want to copy everything! (See Creating a Routing Constraint File on page 79 and Creating a Location Assignment File on page 76)
7. Add your new constraint files to an **.ini** file. (See Enabling Routing Constraints on page 78)
8. Continue to change your design as needed. When you compile, the software will place and route the constrained logic and nets as defined in the constraint files.

Enabling Routing Constraints

Because routing constraints are beta in the Efinity software v2022.2, you must enable them with an **.ini** file. Because routing constraints are used with logic constraints, you enable them both.

1. Create a text file named **efx_pnr_settings.ini** and save it in your project directory.

2. Add the following lines to the **.ini** file:

```
loc_assignment = <path>/<filename>.placeloc
rcf_file = <path>/<filename>.rcf
```

When you synthesize your design, the software uses the assignments in the specified files.

Generate .rcf Template

You tell the software to generate templates in the **efx_pnr_settings.ini** file. Because routing constraints are used with logic constraints, you enable templates for both.

1. If you do not already have one, create a text file named **efx_pnr_settings.ini** and save it in your project directory.
2. Add the following lines to the **.ini** file:

```
dump_placeloc = on
generate_rcf_template = on
```

When you compile your design, the software generates the <*project*>**.out.placeloc** and <*project*>**.rcf.template** files.

Important: Do not generate these templates until you are ready to lock down the routing.

Creating a Routing Constraint File

The routing constraint file is a text file with the extension **.rcf**. The file format is line-oriented; each command is on a single line with spaces between the data.

To make it easier for you to create assignments, the Efinity software can dump all routing data into a template file when routing finishes. (See Generate .rcf Template on page 79) You copy and paste the nets you want to constrain into your own **.rcf**.

Important: **Do NOT** simply copy and paste the entire template file into your **.rcf** or the software may not be able to perform routing efficiently. Only copy the nets you want to constrain.

The **.rcf** has these components:

- `routeTraceFile` <*path*>/<*filename*>**.troutingtraces** is the file that has the saved net traces you want to use.
- `restoreNetFromTraceFile` <*net*> is the net you want to constrain
- Lines beginning with # are comments

```
# The constrained router flow will use the following trace file to restore constrained nets
routeTraceFile  <path>/<project>/<filename>.troutingtraces

# Here is a list of available nets that can be restored from the trace file
# You can use (#) to comment any net that you would like to exclude
restoreNetFromTraceFile  rst_i
restoreNetFromTraceFile  net_1
restoreNetFromTraceFile  net_2
# restoreNetFromTraceFile  net_3    # this net is ignored
```

Best Practices for Constraining Routing

Follow these guidelines when constraining routing to ensure consistency for register and signal names when you re-compile.

- Use a consistent naming convention, such as `netname_LOCKED`, for all constrained nets. This methodology lets you identify them in the template files more easily.
- Limit routing constraints (if possible) to named single-fanout signals between named registers.
- Use the `syn_keep` synthesis attribute—for all locked registers and the signals between locked registers—to tell synthesis to keep the signals during optimization. If you do not use `syn_keep`, the software might optimize away the net you want to constrain.

  ```
  (* syn_keep = "true" *) wire netname_LOCKED;
  ```

- In your **.rcf**, **do not** point to the **.troutingtraces** file in the project **outflow** directory. This file is overwritten each time you compile. Instead, move the **.troutingtraces** file into another directory and point to it in that location.
- Use routing constraints sparingly; excessive constraints make it hard to close timing.
- Implement constrained routing as late in the design cycle as possible (when you have fewer changes to your design).

Note: Although you can use constrained routing on combinational paths, primitive cell names (for example LUT names) on these paths may change if you modify unrelated sections of the design and re-run synthesis. As a result, you may need to update your <*project*>**.out.placeloc** file to reflect the new primitive cell names.

Example Flow

Assume that your design has the following register path: `rlock0` to `rlock1` to `rlock2`, and that this path meets timing. We want to constrain this path while we modify another part of the design (that is independent from this constrained path).

1. To prevent synthesis from optimizing away the registers and wires, use `syn_keep` in the Verilog HDL design:

```
(* syn_keep = "true" *) reg rlock0;
(* syn_keep = "true" *) reg rlock1;
(* syn_keep = "true" *) reg rlock2;
(* syn_keep = "true" *) wire rlock0_net;
(* syn_keep = "true" *) wire rlock1_net;
(* syn_keep = "true" *) wire rlock2_net;
```

2. Run place and route with the options `dump_placeloc = on` and `generate_rcf_template = on`. You add these options to a **efx_pnr_settings.ini** file, one option per line, and save the file in the project folder.
3. Examine the generated file <*project*>**.out.placeloc** to identify the placed location of the locked registers:

```
rlock0~FF 16 49 2
rlock1~FF 16 50 2
rlock2~FF 16 44 2
```

4. Examine the generated file <*project*>**.rcf.template** to find the nets between the registers in the **.rcf.template** file:

```
restoreNetFromTraceFile rlock0_net
restoreNetFromTraceFile rlock1_net
```

5. Remove all cells except the locked ones from the <*project*>**.out.placeloc** file and save it as your own file called **my.placeloc**. Similarly, remove all nets except the constrained ones from <*project*>**.rcf.template** and save it as your own file called **my.rcf** file.
6. Add the following settings to your **efx_pnr_setting.ini** file:

```
loc_assignment = my.placeloc
rcf_file = my.rcf
```

You can now modify any other part of the design and re-run the synthesis and place and route. The software constrains the paths you specified.

Methods for Closing Timing

You have created your RTL, you have designed your board, now you need to close timing, and you are stuck. Knowing what to do for that last bit of tweaking to achieve your desired f_{MAX} can be difficult. When creating the Efinity® software, Efinix® software engineers have chosen default values for the tool flow to achieve the best trade-off between performance and runtime for a large number of benchmark designs. Your design, however, is unique, and may benefit from non-default settings to get the performance you need.

In general, it is best to start by choosing good synthesis options, then placement options, then routing options. First choose high-level options that work well, then run a seed sweep using those options to take advantage of noise and use the best result. You set synthesis and place-and-route options for your project in the Project Editor.

Tip: When trying to close timing, remember that you can also adjust the constraints in your SDC file or modify your design.

Synthesis Options

Changes in synthesis results may or may not help you achieve your final f_{MAX} target. Usually it is best to use these options with different place and route options as part of your design exploration.

Tip: You set synthesis options in the Project Editor. Choose **File** > **Edit Project** and then click the **Synthesis** tab.

Table 5: Synthesis Options (All Families)

Name	Choices	Description
--allow-const-ram-index	0, 1	Infer RAM if an array is accessed through constant indices. This option can be useful if memory is written such that a constant index refers to each segment (e.g., in a byte-enable read/write). **See example.** 0: Default. Do not infer. 1: Infer.
--blackbox-error	0, 1	Generate an error when synthesis encounters an undefined instance or entity. 0: No error. 1: Default. Generate error.
--blast_const_operand_adders	0, 1	If one of the operands to an arithmetic operation is constant, implement it as logic instead of adders. 0: Disable. 1: Default. Enable.
--bram_output_regs_packing	0, 1	Enables the software to pack registers into the output of BRAM. 0: Disable. 1: Default. Enable

Name	Choices	Description
--create-onehot-fsms	0, 1	Create onehot encoded state machine when appropriate. Synthesis can only create these state machines if the state variables do not have explicit encoding in the HDL. If a state machine is coded using onehot encoding, a new section in the map report (**<project>.map.rpt**) shows the encoding information. **See example**. 0: Default. Disabled. 1: Enabled.
--fanout-limit	0 to *n*	If something is high fanout, the tool duplicates the fanout source. 0: Default. Disable. *n*: Indicate the fanout limit at which to begin duplication.
--hdl-compile-unit	0, 1	When considering multiple source files, resolve `define or parameters independently or across all files. This option only works with SystemVerilog files. 0: Across all files. 1: Default. Independently.
--infer-clk-enable	0, 1, 2, 3, 4	Infer flip-flop clock enables from control logic. **See examples**. 0: disable. 1, 2, 3, 4: Effort levels.
--infer-sync-set-reset	0, 1	Infer synchronous set/reset signals. 0: Disable. 1: Default. Enable.
--max_ram	-1, 0, *n*	-1: Default. There is no limit to the number of RAM blocks to infer. 0: Disable. *n*: Any integer.
--max_mult	-1, 0, *n*	-1: Default. There is no limit to the number of multipliers to infer. 0: Disable. *n*: Any integer.
--max_threads	-1, *n*	Choose how many threads that the synthesis tool can launch. -1: Default. The tool uses the maximum number of available processors. *n*: Any integer.
--min-sr-fanout	0, *n*	Infer the flipflop's synchronous set/reset signal from control logic if the set/reset signal fanout is greater than *n*. This option is useful if the design has a lot of small fanout set/reset signals that may create routing congestion. 0: Default. Disable. *n*: Signal fanout.
--min-ce-fanout	0, *n*	Infer the flipflops clock enable from c ontrol logic if the clock enable signal fanout is grester than *n*. 0: Default. Disable. *n*: Signal fanout.
--mode	speed, area, area2	speed: Default. Optimizes for fastest f_{MAX}. area: Optimizes for smallest area. area2: Uses techniques that help to optimize large multiplexer trees.

Name	Choices	Description
--mult-auto-pipeline	Integer	Performs automatic pipelining for wide-multipliers to increase performance at the cost of extra latency. Inserts pipeline registers at the output of partial multiplies and partial sums. On Titanium FPGAs these pipeline registers can be packed into DSP48 primitives as W registers. The software inserts additional registers at the multiplier input and output to balance latency issues caused by the register insertion. The value of this option determines the number of cycles of latency added as a result of inserting pipeline registers. When the value is set to 1, 1 set of pipeline registers will be inserted into the wide-multiplier DSP chain. The pipeline register will be inserted after the partial adder such that the longest path within the wide-multiply DSP chain will be minimized. Setting this option to a higher value reduces the longest path of the wide-multiplier at the cost of higher latency. Note: the software may not always pack these registers into DSP48 primitives. The value of this option must be smaller than the number of DSP48 primitives in your DSP chain. For example, a 32x32 multiplier is mapped to a chain of 4 DSP48 primitives. In this case, the value of --mult-auto-pipeline must be less than 4. 0: Default. Disabled.
--mult-decomp-retime	0, 1	Perform retiming after decomposition of a wide multiplier to improve performance. 0: Default. Disable. 1: Enable.
--peri-syn-inference	0, 1	Enable unified netlist inference flow. See **syn_peri_port** for the synthesis attribute that you use with this option. 0: Default. Disable. 1: Enable.
--peri-syn-instantiation	0, 1	Enable unified netlist instantiation flow. See **syn_peri_port** for the synthesis attribute that you use with this option. 0: Default. Disable. 1: Enable.
--operator-sharing	0, 1	Extract shared operators 0: Default. Disable 1: Enable
--optimize-adder-tree	0, 1	Optimize skewed adder trees 0: Default. Disable 1: Enable
--optimize-zero-init-rom	0, 1	Opitmize ROMs that are initialized to zero. 0: Disable 1: Default. Enable
--retiming	0, 1, 2	Perform retiming optimization. Software moves registers to balance the combinational delay path. 0: Disable. 1: Default. Enable. 2: Advanced algorithm that can benefit some designs.

Name	Choices	Description
--seq_opt	0, 1	Turn on sequential optimization. This option can reduce LUT usage but may impact f_{MAX}. 0: Disable. 1: Default. Enable.
--seq-opt-sync-only	0, 1	Sequential synthesis only considers synchronous reset flipflops. 0: Default. Consider all flipflops. 1: Consider synchronous flipflops only.
--use-logic-for-small-mem	0 to *n*	Set the size limit of small RAM blocks implemented in LEs. 0: Disable. 64: Default.
--use-logic-for-small-rom	0 to *n*	Set the size limit of small ROM blocks implemented in LEs. The number is the maximum number of LEs used. 0: Disable. 64: Default.

Handling High Fanouts

When a signal has a high fanout, it may have higher path delays and therefore reduced performance. Additionally, when you have optimized the depth of the design so that there are very few levels of logic on the critical path, even modest high-fanout nets may be limiting. In these cases, you can tweak the synthesis to address high fanout either globally or manually.

Limit Fanout Globally

You can set the maximum fanout using the `--fanout-limit` synthesis setting. You set this option in the **Project Editor > Synthesis** tab. This option is a global project setting and affects your whole design.

To identify signals with high fanout:

1. Perform a full compile.
2. Open the timing report (<*project name*>**.timing.rpt**) by double-clicking the filename under **Result > Routing** or opening the file in the **outflow** directory.
3. By default, the timing report shows the most critical path. Fanout is reported in the `pins on net` column.

```
Data Path
pin name              model name    delay (ns)   cumulative delay (ns)    pins on net   location
===============================================================================================
 Fled[1]~FF|Q          ff               0.650                   0.650         300         (55,78)
 Fled[1]~FF|O_seq      eft              5.210                   5.861         300         (55,78)
   Routing elements:
     Manhattan distance of X:55, Y:14
 Fled[1]               io               0.420                   6.281         300         (0,92)
```

If the `pins on net` is a large number (300 in the above example) and there is a large delay, try setting the `--fanout-limit` to a lower number. The software duplicates the logic as needed to limit the fanout. This setting trades off area for delay; it may use more logic but run faster.

Example: Set Fanout Limit to 100

1. Open the Project Editor.
2. Click the **Synthesis** tab.
3. Set the **Value** for **--fanout-limit** to 100.
4. Click **OK**.
5. Recompile.

Limit Fanout Manually

If you do not want to limit the fanout with a global setting, you can limit it manually by duplicating logic and using the `syn_preserve` synthesis attribute to tell synthesis not to minimize the duplicated logic.

This attribute applies to signals. When it is set to `true`, `yes`, or `1`, synthesis keeps the signal through optimization, that is, synthesis does not minimize or remove the signal. This attribute can be helpful when you want to simulate or view a signal in the Debugger. Although the signal is kept, synthesis may still choose to implement downstream functions that depend on this signal independent of this preserved signal.

In the Efinity software v2022.2 and higher, the `syn_preserve` attribute is supported on a user hierarchy instance. The effect is equivalent to tagging all boundary signals of the instance with `syn_preserve`.

Verilog HDL:

```
(* syn_preserve = "true" *) wire x;
```

VHDL:

```
attribute syn_preserve: boolean;
attribute syn_preserve of x : signal is true;
```

Note: A signal with `syn_preserve` usually has it's name preserved through synthesis flow. However, if the signal is connected directly to a top-level port, the name in the **map.v** netlist may be changed to that of the top-level port name.

Place-and-Route Options

The best place to start with adjusting the place-and-route settings is to use one of the optimization levels. Each level is a "recipe" that controls both placement and routing; one is not necessarily better than the other. Efinix developed them to introduce as much useful variation as possible into the place-and-route process. These options will not help all designs, and often the default settings are actually the best choice.

Note: Using these options can cause **significantly higher run-time**. In fact, some options trigger completely different optimization algorithms than the standard flow.

Try running all of these optimization levels and choose the one that works best for your design. The timing values are best for designs that are easy to route while the congestion values are best for designs that are very difficult to route. For congested designs, these options resolve congestion early in the process so that the router can focus on meeting timing. If the number of routing iterations is greater than 20, the design is hard to route. The power options can help reduce a design's power consumption. See Optimization Sweeping on page 89 for details on how to sweep these options using a script.

The seed option introduces random noise in the placer (see Seed Sweeping on page 89 for a detailed discussion). The f_{MAX} difference between the best and worst seed in a 5-seed sweep can be up to 10%. But keep in mind that a *good seed* only applies to one specific design for one Efinity® release on one operating system. So if you want to reproduce the same result for the same design, you need to use the same software release and same operating system.

Sometimes small design changes may appear to reduce f_{MAX} by up to 10%. You should run a 5-seed sweep to verify that the decrease was actually due to the design change and not simply noise from the placer.

Tip: You set these options in the Project Editor. Choose **File** > **Edit Project** and click the **Place and Route** tab.

Table 6: Optimization Options

Optimization	Value	Description
--optimization_level	NULL	Disabled (default).
	TIMING_1	Recipe 1 to meet timing for a non-congested design.
	TIMING_2	Recipe 2 to meet timing for a non-congested design.
	TIMING_3	Recipe 3 to meet timing for a non-congested design.
	CONGESTION_1	Recipe 1 to meet timing and help a congested design route.
	CONGESTION_2	Recipe 2 to meet timing and help a congested design route.
	CONGESTION_3	Recipe 3 to meet timing and help a congested design route.
	POWER_1	Recipe 1 to reduce a design's power consumption.
	POWER_2	Recipe 2 to meet a design's power consumption.
seed	Integer	Enter any integer to insert a random seed into the place-and-route algorithm. Read more about seed sweeping in the next section.
--placer_effort_level	1, 2, 3, 4, 5	Controls how much runtime the placer uses to attempt to improve placement quality. Default: 2

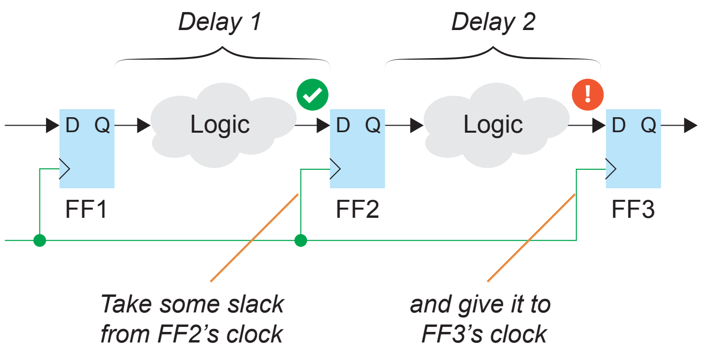
Optimization	Value	Description
--max_threads	Integer	Choose the maximum number of threads that the placer can launch. Typically you want to use the default, -1, which means the placer can launch as many threads as it needs. However, if you are running other processes on your computer, you may want to limit the number of threads. Default: -1
--beneficial-skew	on, off	When turned on, the software "borrows" slack from one clock to meet timing on another. (Titanium FPGAs only) Default: on

Beneficial Skew

Beneficial skew (sometimes called "slack borrowing") is a process in which the software adjusts clocks to better close timing. In the following figure, Delay 1 meets timing and has extra slack. Delay 2 does not meet timing. The software adjust the clocks to take some slack from Delay 1 and give it to Delay 2, allowing both to meet timing.

This option is only available for Titanium FPGAs.

Figure 39: Beneficial Skew Example



Sweeping Script

Efinix provides helper scripts that you can use to compile a design multiple times using various place-and-route settings.

- *Linux*—**scripts/efx_run_pnr_sweep.py**
- *Windows*—**bin/efx_run_pnr_sweep.bat**

The scripts use this syntax:

```
efx_run_pnr_sweep.py <project file> {sweep_opt_levels | sweep_seeds {--num_seeds <integer>}
    {--start_seed <integer>} } [-h]
```

```
efx_run_pnr_sweep.bat <project file> {sweep_opt_levels | sweep_seeds {--num_seeds <integer>}
    {--start_seed <integer>} } [-h]
```

where:

- *<project file>* is your project's XML file
- `sweep_opt_levels` sweeps through all of the possible optimization settings
- `sweep_seeds` sweeps through seeds
- `-h, --help` shows the help

Note: You sweep through either optimizations or seeds. You cannot sweep both at the same time.

The software summarizes the timing results of the runs in the **timing.sum.rpt** file in the project directory. You can find the corresponding result files in the **run_sweep_**<*string*> directory.

Important: You should run these script in your project directory, not the Efinity installation directory. Otherwise, the scripts cannot find all of the required files.

Optimization Sweeping

You can use the scripts to sweep through all of the place-and-route optimization settings to find out which one works best for your design.

Example Usage

To sweep all of the optimization levels for project helloworld:

```
efx_run_pnr_sweep.py helloworld.xml sweep_opt_levels
efx_run_pnr_sweep.bat helloworld.xml sweep_opt_levels
```

Using the Results

When sweeping completes, open the **timing.sum.rpt** file in the project directory. It gives an overview of the clocks analyzed and their frequency for each optimization setting. The following code shows and excerpt of the summary report for sweeping optimization levels for the **pt_demo** project.

```
|---------------------------------------------------------------|
|          Maximum possible analyzed clocks frequency           |
|---------------------------------------------------------------|
|                                                               |
|---------------------------------------------------------------|
| Clock Name: Oclk                                              |
|---------------------------------------------------------------|
|              |   Period (ns)     | Frequency (MHz)   |   Edge   |
|---------------------------------------------------------------|
| CONGESTION_1 |      6.453        |     154.964       |  (R-R)   |
| CONGESTION_2 |      6.867        |     145.620       |  (R-R)   |
| CONGESTION_3 |      8.141        |     122.841       |  (R-R)   |
|   TIMING_1   |      8.570        |     116.682       |  (R-R)   |
|   TIMING_2   |      8.914        |     112.182       |  (R-R)   |
|   TIMING_3   |      8.711        |     114.798       |  (R-R)   |
|---------------------------------------------------------------|
```

You can also review detailed reports in the **run_sweep_**<*string*>**/seed_**<*number*>**/ outflow** directory.

If you are happy with the results for one of the optimization levels, you can set that option in your project:

1. Open the Project Editor.
2. Click the **Place and Route** tab.
3. Choose the **Value** you want to set for the optimization level.
4. Click **OK**.
5. Recompile.

Seed Sweeping

After you choose an appropriate optimization level (or the default), try running a seed sweep and taking the best of these runs to close timing. The annealer uses a random number generator that you can control using a seed parameter. Running a 10-seed sweep typically results in an f_{MAX} variation of 10% to 20% in a well-behaved circuit, but can be higher for random differences that only occur in one seed. Keep these tips in mind:

- Run a seed sweep every time you make a small change to the design.

- No seed value is better than any other.
- Different seed values may be best on different machines.

Example Usage

Sweep seeds using the default settings. In this mode, the script runs 10 different seeds with seed numbers 0 - 9. That is, it runs `seed=0`, `seed=1`, etc.

```
efx_run_pnr_sweep.py helloworld.xml sweep_seeds
efx_run_pnr_sweep.bat helloworld.xml sweep_seeds
```

Compile 6 seeds with seed numbers 0 - 5:

```
efx_run_pnr_sweep.py helloworld.xml sweep_seeds --num_seeds 6
efx_run_pnr_sweep.bat helloworld.xml sweep_seeds --num_seeds 6
```

Compile with seed numbers 3, 4, and 5:

```
efx_run_pnr_sweep.py helloworld.xml sweep_seeds --start_seed 3 --end_seed 5
efx_run_pnr_sweep.bat helloworld.xml sweep_seeds --start_seed 3 --end_seed 5
```

Compile with 6 seeds, starting with seed number 3:

```
efx_run_pnr_sweep.py helloworld.xml sweep_seeds --start_seed 3 --num_seeds 6
efx_run_pnr_sweep.bat helloworld.xml sweep_seeds --start_seed 3 --num_seeds 6
```

Using the Results

When seed sweeping completes, open the **timing.sum.rpt** file in the project directory. It gives an overview of the clocks analyzed and their frequency for each seed. The following code shows and excerpt of the summary report for seed sweeping the **pt_demo** project with the default 10 seeds. The number after `seed_` is the random seed number the compiler used.

```
|-----------------------------------------------------------|
|       Maximum possible analyzed clocks frequency          |
|-----------------------------------------------------------|
|                                                           |
|-----------------------------------------------------------|
| Clock Name: Oclk                                          |
|-----------------------------------------------------------|
|        |   Period (ns)     | Frequency (MHz)   |   Edge   |
|-----------------------------------------------------------|
| seed_0 |      6.211        |     161.006       |  (R-R)   |
| seed_1 |      7.211        |     138.678       |  (R-R)   |
| seed_2 |      7.852        |     127.363       |  (R-R)   |
| seed_3 |      7.883        |     126.858       |  (R-R)   |
| seed_4 |      5.453        |     183.381       |  (R-R)   |
| seed_5 |      5.328        |     187.683       |  (R-R)   |
| seed_6 |      7.563        |     132.231       |  (R-R)   |
| seed_7 |      5.633        |     177.531       |  (R-R)   |
| seed_8 |      5.547        |     180.282       |  (R-R)   |
| seed_9 |      4.656        |     214.765       |  (R-R)   |
|-----------------------------------------------------------|
```

You can also review detailed reports for each run in the **run_sweep_<*string*>/ seed_<*number*>/outflow** directory.

If you are happy with the results for one of the seeds, you can set that option in your project. For example, to apply `seed_9` to your project:

1. Open the Project Editor.
2. Click the **Place and Route** tab.
3. Enter **9** in the **Value** cell.
4. Click **OK**.
5. Recompile.

Closing Timing with High DSP Block Utilization

If your design has a >50% of the DSP Blocks implemented with EFX_DSP24 or EFX_DSP12 primitives, the f_{MAX} can vary significantly depending on the placement seed. Therefore, it is a good idea to try 3 or 4 seeds to see if it helps with timing closure, more so than for a typical design.

Learn more: Refer to the Efinity Timing Closure User Guide for information on how to perform seed sweeping.

Tcl Console

The Efinity software uses a Tcl interpreter to process SDC constraints and to support timing analysis. The Tcl Console is an interactive shell that you use to execute Tcl commands.

To explore timing, you use a combination of the Tcl Console, the Timing Browser, and the Floorplan Editor. In the Tcl Console, you enter commands to query timing reports. The software reports specific timing paths based on their slack or propagation delay. For example:

- Use the `report_path` command to query propagation delays between the specified end points.
- Use the `report_timing` command to query the slack between the specified end points.

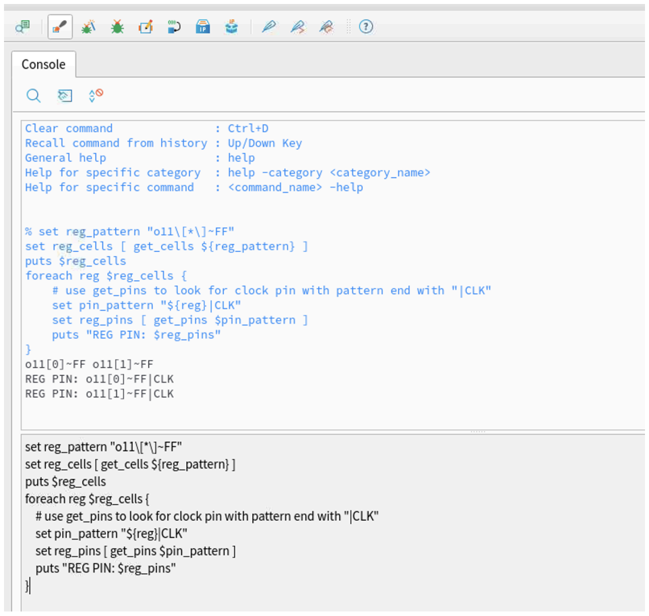
Specify the details of the timing path that you want to analyze. You can specify the starting and ending points explicitly or leave them as implicit. The software analyzes the timing path based on the arguments provided to the constraints.

Open the Tcl Console in the Efinity GUI by:

- Clicking the Show/Hide Tcl Command Console icons in the main toolbar.
- Choosing **File > > Show/Hide Tcl Command Console**.
- Pressing the Ctrl + T keys.

Note: In the Efinity software v2024.1 and higher you can also use the Tcl Console in a terminal (see Command-Line Tcl Console on page 103).

Figure 40: Tcl Console



```
Clear command                : Ctrl+D
Recall command from history  : Up/Down Key
General help                 : help
Help for specific category   : help -category <category_name>
Help for specific command    : <command_name> -help


% set reg_pattern "o11\[*\]~FF"
set reg_cells [ get_cells ${reg_pattern} ]
puts $reg_cells
foreach reg $reg_cells {
    # use get_pins to look for clock pin with pattern end with "|CLK"
    set pin_pattern "${reg}|CLK"
    set reg_pins [ get_pins $pin_pattern ]
    puts "REG PIN: $reg_pins"
}
o11[0]~FF o11[1]~FF
REG PIN: o11[0]~FF|CLK
REG PIN: o11[1]~FF|CLK
```

```
set reg_pattern "o11\[*\]~FF"
set reg_cells [ get_cells ${reg_pattern} ]
puts $reg_cells
foreach reg $reg_cells {
  # use get_pins to look for clock pin with pattern end with "|CLK"
  set pin_pattern "${reg}|CLK"
  set reg_pins [ get_pins $pin_pattern ]
  puts "REG PIN: $reg_pins"
}
```

In the Efinity software v2024.1 and higher you can use multi-line commands in the Tcl Console. Press Shift + Enter to insert a new line (pressing Enter executes the command). You can also copy and paste multi-line commands into the console.

The Tcl Console has an auto-correct feature. In the Efinity software v2024.1 and higher, this feature is turned off by default. You can turn it on in the **Preferences** dialog box (**File > Preferences**).

General Commands

These commands are for reading and writing SDC files, working with timing model information, and clearing or reseting the Tcl Console.

delete_timing_results Command

```
delete_timing_results
```

This command deletes path data—reported using the report_path and report_timing commands—that the software has stored in memory.

get_available_timing_model Command

```
get_available_timing_model
```

Returns a list of available operating conditions for the current FPGA. The FPGA must be loaded for this command to execute.

get_timing_model Command

```
get_timing_model
```

Returns the current operating conditions for the device. The device must be loaded for this command to execute.

read_sdc Command

```
read_sdc <file>
```

This command reads in the SDC file specified with <*file*>. If you do not specify a file, the tool loads the SDC file list that you set in your project. The SDC file overrides previous constraints. If a constraint is overwritten by a later constraint, the software issues a warning similar to:

```
SDC <file name>: <line number>] A clock with name 'Oclk' already exists.
 Overwriting the previous clock with the same name
```

reset_timing Command

```
reset_timing
```

Removes the timing data, all constraints, and all reported paths.

set_timing_model Command

```
set_timing_model
```

Specify the operating conditions for the current design. Use get_available_timing_model to view the list of supported conditions. You can only use this command after you have performed place and route.

write_sdc Command

```
write_sdc <file>
```

Write the timing constraints in memory to the SDC file specified with <*file*>.

Note: The software does not add the SDC file to your project; you need to add it separately in the using the **Project Editor** dialog box.

Report Commands

These commands are for generating various timing reports.

check_timing Constraint

```
check_timing [-file <file>] [-override_defaults <checks>] [-exclude <checks>] [-verbose]
```

This command performs 5 types of checks by default. You can exclude or override specific defaults.

Table 7: Timing Check Types

Check Type	Description
no_clock	Report any synchronous blocks without a clock constraint.
unconstrained_internal_endpoints	Report any flipflop D pins that do not have a clock signature.
no_input_delay	Report any input ports without an input constraint.
no_output_delay	Report any output ports without an output constraint.
multiple_clock	Report clock pins that are associated with multiple clocks. This check helps find unintentional multiple clock assignments. This check also detects clock pins with dynamic clocks.

This command supports the following options:

- `-file` if used, writes the results to the named file in the **outflow** directory.
- `-override_defaults` overrides the default timing checks and run the specified checks only.
- `-exclude` specifies the checks to be excluded from the list of default timing checks.
- `-verbose` lists the pin(s) that violate(s) the specific check(s).

The following example uses the `no_clock` and `multiple_clock` checks only:

```
check_timing -override_defaults {no_clock multiple_clock}
```

The following example excludes the `no_input_delay` and `no_output_delay` checks:

```
check_timing -exclude {no_input_delay no_output_delay}
```

report_clocks Command

```
report_clocks [-file <file>] [-stdout]
```

This command generates a clock report. If you do not specify a file name, the software prints the report to the Console (stdout) by default.

report_path Command

```
report_path [-file <file>] [-npaths <number>] [-nworst <number>] \
    [-show_routing] [-stdout] [-summary] [-through <names>] -to <names> \
    -from <names> -id <number> [-min_path]
```

This constraint reports the longest delay path and the corresponding delay value.

- `-file` writes the results to a file in the **outflow** directory.
- `-npaths` is the maximum number of paths to report. If you do not specify the number of paths, the software only provides the single longest delay path.
- `-nworst` is the maximum number of paths reported for each unique endpoint. Without this option, the number of paths reported for each destination node is restricted by `-npaths` only. If you use this option but not `-npaths`, `-npaths` defaults to the value specified for `-nworst`.
- `-show_routing` displays detailed routing information.
- `-stdout` writes the results to the Console during compilation only. If you re-generate timing, use the `-file` option instead.
- `-summary` generates a table that summarizes the results.
- `-through` restricts analysis to paths that go through specified pins or nets. Paths that are reported can not start before or go beyond a keeper node (register or port); this restriction considers register pins as combinational nodes in the design.
- `-from` and `-to` limit the analysis to specific start and end points. Any node or cell in the design is a valid endpoint.
- `-id` is the position in the Timing Browser where the tool displays the result.
- `-min_path` reports the minimum delay paths.

report_sdc Constraint

```
report_sdc [-from <clocks>] [-to <clocks>] [-details] [-file <file>]
```

This command reports the clock domain crossing (CDC) paths in the design. This command supports the following options:

- `-from` specifies the source clocks.
- `-to` specifies the destination clocks.
- `-details` prints a detailed CDC report.
- `-file` if used, writes the results to the named file in the **outflow** directory.

By default, the report_cdc command would print out a summary report like the following example:

```
Source Clock }| Destination Clock | Exceptions | Endpoints
==========================================================
clk_write     |  clk_read         |  None      |  23
clk_read      |  clk_write        |  None      |  12
```

When you use the `-details` option, the report shows the CDC path description. The current check lists all endpoints, and groups the source/destination as buses by their names. The tool supports the following descriptions:

- 1-bit and multi-bit unknown CDC circuitry
- 1-bit and multi-bit synchronized with ASYNC_REG property
- 1-bit and multi-bit synchronized with missing ASYNC_REG property
- Combinational logic detected before a synchronizer
- 1-bit CDC path on a non-FF primitive
- Multi-clock fan-in to synchronizer

The following code shows an example report:

```
CDC Report
Source Clock: clk_write
Destination Clock: clk_read

Row|Exception| Description                       | Source (From)        |   Destination (To)
=========================================================================================
 1 | None    | 1-bit unknown CDC circuitry      | r_ptr_sync[0]~FF|CLK | data_count_r[0]~FF|D
 2 | None    | Multi-bit unknown CDC circuitry  | w_ptr_q[2:1]~FF|CLK  | data_count_r[2:1]~FF|D
 3 | None    | 1-bit synchronized with missing  | w_prt_q[11]~FF|CLK   | w_grey_sync[11]~FF|D
   |         | ASYNC_REG property               |                      |
```

report_timing Command

```
report_timing [-detail summary|path_only|path_and_clock|full_path] \
    [-file <name>] [-from_clock <names>] -from <names>
    [-fall_from_clock <names>] [-rise_from_clock <names>]
    [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>]
    [-show_routing] [-stdout] [-through <names>] -to <names> -to_clock <names>
    [-rise_to_clock <names>] [-fall_to_clock <names>] [-id <number>]
    [-hold] [-setup]
```

This command reports the worst-case paths and their associated risk. The tool displays paths in order of increasing slack.

- `-detail` specifies how much detail is shown in the path report.
- `-file` writes the results to a file in the **outflow** directory.
- `-from_clock` and `-to_clock` are valid source and destination clocks, respectively.
- `-from` and `-to` limit the analysis to specific start and end points. Any node or cell in the design is a valid endpoint.
- `-fall_from_clock` and `-fall_to_clock` are the starting and ending points of the falling edge of the clock domain, respectively.
- `-rise_from_clock` and `-rise_to_clock` are the starting and ending points of the rising edge of the clock domain, respectively.
- `-less_than_slack` displays only those paths with slack less than the specified limit.
- `-npaths` sets the number of paths to report. If you do not specify the number of paths, the software only provides the single longest delay path.
- `-nworst` limits the number of paths reported for each unique endpoint. Without this option, the number of paths reported for each destination node is restricted by the `-npaths` only. If you use this option but not `-npaths`, `-npaths` defaults to the value specified for `-nworst`.
- `-show_routing` displays detailed routing information.
- `-stdout` writes the results to the Console.
- `-through` restricts analysis to paths that go through specified pins or nets. Paths that are reported can not start before or go beyond a keeper node (register or port); this restriction considers register pins as combinational nodes in the design.
- `-id` is the position in the Timing Browser where the tool displays the result. Use `-id` to overwrite existing constraints in the Timing Browser. `0` is reserved for the critical path. If the specified ID number is larger than the existing constraint ID, the tool ignores this option. By default, it returns clock setup paths if you do not specify `-setup` or `-hold`.
- `-setup` reports the clock setup paths.
- `-hold` reports the clock hold paths.

report_timing_summary Command

```
report_timing_summary [-file <file>] [-hold] [-setup]
```

This command performs timing analysis and generates the critical path timing report. The software saves the report as <*project*>**.timing.rpt** by default. To specify a different file name, use the `-file` option. By default, the tool prints setup and hold paths.

- `-file` writes the results to a file in the **outflow** directory.
- `-setup` reports the clock setup paths.
- `-hold` reports the clock hold paths.

Tcl List Functions (Alphabetical)

The following topics describe the Tcl list functions the Tcl Console supports.

lappend

```
lappend <list name> <new list or element> <new list or element> ...
```

This function appends Efinity Tcl objects to a collection. It modifies the *<list name>* collection and returns the modified object.

```
# Append $ff_list to the end of $lut_list
set lut_list [get_cells LUT*]
set ff_list [get_cells *~FF]
lappend lut_list $ff_list

# Append a single Efinity Tcl object to the end of $lut_list
lappend lut_list [get_cells cell1]
```

lassign

```
lassign <list> <variable 1> <variable 2> ...
```

This function assigns multiple Efinity Tcl objects to multiple variables.

```
# Assign the first three elements on the list to variables
# x, y, and z
lassign [get_cells *~FF] x y z ]
set pins_on_x [get_pins $x]
set pins_on_y [get_pins $y]
set pins_on_z [get_pins $z]
```

lindex

```
lindex <list> <index>
```

This function returns a specific Efinity Tcl objects from the collection based on the specified index value. `end` is the index value of the last element.

```
# Return the index 2 objects from the collection
set element [lindex [ get_cells *~FF ] 2 ]

# Return the last element from the collection
set last [lindex [ get_cells *~FF ] end ]
```

linsert

```
linsert <list> <index> <new list or element>
```

This function inserts a new list or elements before the specified <*index*> in the collection. All elements must be Efinity Tcl objects.

```
# Insert the ff_list in the first position of the lut_list
set lut_list [get_cells LUT*]
set ff_list [get_cells *~FF]
set combined [linsert $lut_list 1 $ff_list]

# Insert the ff_list at the end of the lut_list
set lut_list [get_cells LUT*]
set ff_list [get_cells *~FF]
set combined [linsert $lut_list end $ff_list]

# Add one cell to the end of the lut_list
set combined [linsert $lut_list end [get_cells cell1]]
```

llength

```
llength <list>
```

This function returns the length of an Efinity Tcl object collection.

```
# Return the number of cells with the format *~FF
llength [get_cells *~FF]
```

foreach_in_collection

```
foreach_in_collection <variable> <list>
```

This function loops through a single collection of Efinity Tcl objects.

```
# loop through a group of flops and get pins from each flop
foreach_in_collection c [ get_cells <pattern>] {
   set pin_on_cell [get_pins $c]
   ...
}
```

foreach

```
foreach <variable 1> <list 1> <variable 2> <list 2> ...
```

This function loops through multiple collections of Efinity Tcl objects. All collections must be of the same Efinity Tcl object type.

```
# loop through two flipflop lists and get a timing report for
# each pair; the two lists should have the same length.
foreach launch_ff [get_cells <launch_pattern>] \
              capture_ff [get_cells <capture_pattern>] {
       report_timing -from $launch_ff -to $capture_ff
}
```

lrange

```
lrange <list> <start index> <end index>
```

This function returns a range of Efinity Tcl objects from the collection based on the range of index values. `end` is the index value of the last element. In the following example, the `end-2` index is the third element from the end.

```
# Return the first three elements of a collection
set elements [lrange [get_cells *~FF] 0 2]

# Return the last three elements of a collection
set elements [ lrange [ get_cells *~FF] end-2 end]
```

lreplace

```
lappend <list> <start index> <end index> <new list or element> <new list or
 element> ...
```

This function deletes the elements from *<start index>* to *<end index>* (inclusive) and replaces them with the new specified lists or elements. All elements must be Efinity Tcl objects.

```
set lut_list [get_cells LUT*]
set ff_list [get_cells *~FF]

# Replace the second to fourth elements of $lut_list with elements in $ff_list
set combined [lreplace $lut_list 2 4 $ff_list]

# Append the list $ff_list to the end of $lut_list
set combined [lreplace $lut_list end+1 end+1 $ff_list
```

lreverse

```
lreverse <list>
```

This function reverses the order of elements in an Efinity Tcl object collection.

```
# let cells be  a collection { cell1  cell2  cell3 }
# Return {cell3 cell2 cell1}
lreverse $cells
```

lsearch

```
lsearch [-all | -ascii | -exact | -glob | -not | -regexp | -start <index>]
 <list> <pattern>
```

This function returns the index of the first matching element in the specified *<list>*. It returns `-1` if it does not find a match. Refer to the lsearch function on the Tcl Developer Xchange web site (www.tcl.tk) for a detailed explanaion of the options.

```
# Return the index of o12[1]
lsearch [get_nets [get_ports o12[*]]] {o12\[1\]}
lsearch -exact [get_nets [get_ports o12[*]]] o12[1]

# Return the index of the first element that is not o12[1]
lsearch -not -exact [get_nets [get_ports o12[*]]] o12[1]

# Return the indices of elements with the regex format o12\[.\]
lsearch -all -regexp [get_nets [get_ports o12[*]]] {o12\[.\]}
```

lsort

```
lsort [-increasing] [-decreasing] <list>
```

This function sorts the specified *<list>* according to the order specified (increasing or decreasing). The software sorts the elements using ASCII string comparison.

```
# sort cells in increasing order
set sorted_cells [lsort -increasing [get_cells LUT*]]

# sort cells in decreasing order
set sorted_cells [lsort -decreasing [get_cells LUT*]]
```

Tcl Script Examples

The following example Tcl scripts show how to use Tcl commands in SDC files and for custom reporting.

Identify Pins with a Regular Expression

This script shows how to identify a flipflop's input, output, and clock pins using a regular expression.

```
# Given a register pattern, find the nets connected to the register's pins
set reg_pattern "o11\[*\]~FF"
set reg_cells [ get_cells ${reg_pattern} ]
puts $reg_cells

# get all the pins from cells with the pattern "|*"
foreach reg $reg_cells {
    set pin_pattern "${reg}|*"
    set reg_pins [ get_pins $pin_pattern ]

    foreach p $reg_pins {
        set is_clock_pin 0
        set is_in_pin 0
        set is_out_pin 0

        # For flipflops, the pin type can be identified by specific pin name
        if {[regexp {\|CLK$} $p]} {
            set is_clock_pin 1
        } elseif {[regexp {\|Q$} $p]} {
            set is_out_pin 1
        } else {
            set is_in_pin 1
        }

        set net_on_pin [get_nets $p]
        puts "PIN $p is_clock=${is_clock_pin} is_in_pin=${is_in_pin} /
            is_out_pin=${is_out_pin} NET: $net_on_pin"
}
```

Create Clocks with Different Periods

This script shows how to create clocks that have different clock periods.

```
# Procedure to create a clock with different clock periods
proc define_clock_data {clk_id} {
    set period 10
    set waveform {0 5}
    if {$clk_id >= 0 && $clk_id < 10} {
        set period 20
        set waveform {5 10}
    } elseif {$clk_id >=10 && $clk_id < 20} {
        set period 30
        set waveform {10 15}
    }
    return [ list $period $waveform ]
}

# create the clocks
for {set i 0} {$i < 32} {incr i} {
    set clock_data [ define_clock_data $i ]

    create_clock -period [ lindex $clock_data 0 ] \
        -waveform [ lindex $clock_data 1 ] \
        -name clk_in[$i] [ get_ports clk_in[$i] ]
}

# Set clock groups after all clocks are defined
for {set i 0} {$i < 32} {incr i} {
    set_clock_groups -exclusive -group clk_in[$i]
}
```

Generate a Report with get_fanouts

This script shows how to use the get_fanouts constraint to generate different reports with the report_timing command.

```
set file_path "./tests"
set ena0_ffs [ get_fanouts [get_ports ena[0]] ]

foreach ena0_ff $ena0_ffs {
    report_timing -from $ena0_ff -setup -npaths 10 \
        -file ${file_path}/${ena0_ff}_path.rpt
}
```

Generate Report with a Subset of Clocks

This script shows how to generate a report for a subset of clock indices with the report_timing command.

```
set file_path "./tests"
set clk_index_list { 0 12 24 }

foreach clk_id $clk_index_list {
    report_timing -from_clock [get_clocks clk_in[$clk_id]] -setup -npaths 10 \
        -file ${file_path}/test1_clk_in_${clk_id}_path.rpt
}
```

Use Variable for Min/Max Delay Calculation

This script shows how to use variables in the `set_input_delay` constraint. This code uses the same example as in Input Receive Clock Delay on page 35.

```
set max_board_delay 4
set min_board_delay 2

# copy values from outflow/<project>.pt_timing.rpt
set GPIO_IN_max 1.954
set GPIO_IN_min 0.526

set_input_delay -clock -max [expr $max_board_delay + $GPIO_IN_max ] din
set_input_delay -clock -min [expr $min_board_delay + $GPIO_IN_min ] din
```

Command-Line Tcl Console

In the Efinity software v2024.1 and higher, you can use the Tcl Console at the command line to run Tcl scripts on a fully compiled design. (If you have not compiled yet, the software issues an error message.)

To enter the interactive Tcl Console, use the `sta_tclsh` flow option with the **efx_run.py** Python 3 script. The terminal or command prompts displays an interactive Tcl Console similar to tclsh. You can enter Tcl commands and expressions for evaluation. Use this command to start the Tcl Console:

```
# Windows
efx_run.bat --flow sta_tclsh --prj <project>.xml

# Linux
efx_run.py --flow sta_tclsh --prj <project>.xml
```

You can also run the Tcl Console in batch mode. In this mode you also specify a script name. The Tcl Console runs the script and exists after executing it.

```
# Windows
efx_run.bat --flow sta_tclsh --prj <project>.xml --tcl_script <Tcl script>

# Liinux
efx_run.py --flow sta_tclsh --prj <project>.xml --tcl_script <Tcl script>
```

Note: You can use these shortcuts: `-f` instead of `-flow` and `-t` instead of `-tcl_script`.

Appendix

About the <*project*>.pt.sdc File

When you generate constraints in the Interface Designer, the software creates the <*project*>**.pt.sdc** in the **outflow** directory; this template file has the interface block constraints. You copy and paste these constraints into your project SDC file. Some generated constraints require you to modify them, for example, to add a clock period or name. These constraints are commented out so they do not generate errors if you include them in your SDC file without modifying them.

Important: Do not add the <*project*>.pt.sdc file to your project! It is re-created every time you generate constraints and any changes you make will be overwritten.

The PLL Constraints section has the `create_clock` SDC command for all PLL outout clocks. Use these commands as is without modification.

```
# PLL Constraints
#################
create_clock -period 10.0000 i_hbramClk_fb
create_clock -waveform {1.2500 3.7500} -period 5.0000 i_hbramClk90
...
```

The GPIO Constraints, HSIO GPIO Constraints, and MIPI RX/TX Lane Constraints sections have constraints for these blocks, some of which are templates that you need to modify.

Use SDC constraints for registered inputs and outputs as is without modification.

For GPIO and LVDS blocks used as clock sources, the Interface Designer includes a `create_clock` template line. To constrain these clocks, replace `<USER_PERIOD>` with the clock period and uncomment the line.

Non-registered inputs and outputs also have template lines. Modify them as follows:

- Replace `<CLOCK>` with the clock name.
- (Optional) Replace `<clkout_pad>` with the reference clock pin name and remove the brackets []. If you do not want to use a reference clock pin, delete `[-reference_pin <clkout_pad>]`
- Replace `<MAX CALCULATION>` and `<MIN CALCULATION>` with the values you calculate as described in Constraining Unsynchronized Inputs and Outputs on page 33.
- Uncomment the line.

```
# GPIO Constraints
####################
# create_clock -period <USER_PERIOD> [get_ports {clock}]
# set_input_delay -clock <CLOCK> [-reference_pin <clkout_pad>]
#     -max <MAX CALCULATION> [get_ports {i_arstn}]
# set_input_delay -clock <CLOCK> [-reference_pin <clkout_pad>]
#     -min <MIN CALCULATION> [get_ports {i_arstn}]
set_output_delay -clock_fall -clock i_hbramClk90 -reference_pin [get_ports
   {i_hbramClk90~CLKOUT~75~322}] -max 0.263 [get_ports {hbc_ck_n_LO hbc_ck_n_HI}]
set_output_delay -clock_fall -clock i_hbramClk90 -reference_pin [get_ports
   {i_hbramClk90~CLKOUT~75~322}] -min -0.140 [get_ports {hbc_ck_n_LO hbc_ck_n_HI}]
set_input_delay -clock i_hbramClk_cal -reference_pin [get_ports
   {i_hbramClk_cal~CLKOUT~32~322}] -max 0.414 [get_ports {hbc_dq_IN_LO[0] hbc_dq_IN_HI[0]}]
set_input_delay -clock i_hbramClk_cal -reference_pin [get_ports
   {i_hbramClk_cal~CLKOUT~32~322}] -min 0.276 [get_ports {hbc_dq_IN_LO[0] hbc_dq_IN_HI[0]}]
...
```

The Clock Latency Constraints section has templates for the set_clock_latency constaint. Clock Latency on page 16 and Constraining I/O on page 30 describe how to use these templates.

```
# Clock Latency Constraints
############################
# set_clock_latency -source -setup <board_max -2.834> [get_ports {clk}]
# set_clock_latency -source -hold <board_min -1.417> [get_ports {clk}]
# set_clock_latency -source -setup <board_max + 1.476> [get_ports {refclk}]
# set_clock_latency -source -hold <board_min + 0.738> [get_ports {refclk}]
```

The JTAG Timing Report shows the SDC constraints for the JTAG signals. Additionally, you should use `set_clock_groups` to make the JTAG clocks unrelated to other clocks.

```
# JTAG Constraints
####################
# create_clock -period <USER_PERIOD> [get_ports {jtag_inst1_TCK}]
# create_clock -period <USER_PERIOD> [get_ports {jtag_inst1_DRCK}]
set_output_delay -clock jtag_inst1_TCK -max 0.117 [get_ports {jtag_inst1_TDO}]
set_output_delay -clock jtag_inst1_TCK -min -0.075 [get_ports {jtag_inst1_TDO}]
...
```

About the <*project*>.pt_timing.rpt File

This report shows the timing for your design's interface blocks. When you generate constraints in the Interface Designer, the software creates this file in the **outflow** directory.

Clocks can come from several sources: PLL, GPIOs, MIPI RX Lane, MIPI RX PHY, JTAG.

The PLL Timing Report shows details about the clocks generated by PLLs in the interface, including the clock period, any phase shift, and whether the clock is inverted. This data matches the `create_clock` SDC template. The report also shows the PLL compensation delay. Clock Latency on page 16 and Constraining I/O on page 30 describe how to use the data.

```
---------- 1. PLL Timing Report (begin) ----------
+---------+---------+---------+----------+----+-----------------+--------------+--------------+
|   PLL   |Resource|Reference|Core Clock| FB |       Core      |      PLL     |      PLL     |
|Instance|         |  Clock  |Reference |Mode|  Feedback Pin   | Compensation | Compensation |
|         |         |         |    Pin   |    |                 |Delay Max (ns)|Delay Min (ns)|
+---------+---------+---------+----------+----+-----------------+--------------+--------------+
|  pll    |PLL_TR0 | external|          |core|clk~CLKOUT~40~482|    4.310     |     2.155    |
+---------+---------+---------+----------+----+-----------------+--------------+--------------+

+-------+-------------+-----------------------+
| Clock | Period (ns) | Phase Shift (degrees) |
+-------+-------------+-----------------------+
|  clk  |   10.0000   |           0           |
+-------+-------------+-----------------------+
```

The GPIO Timing Report and HSIO GPIO Timing reports give timing information about the GPIO or HSIO blocks used in your design. The data is grouped by non-registered and registered blocks.

For non-registered blocks, the `Max` value is the worst case (slowest corner) and the `Min` is the best case (fastest corner). When you are constraining unsynchronized inputs and outputs, you use these maximum and minimum numbers for the calculations.

```
Non-registered GPIO Configuration:
===================================

+----------------------+----------------------+-----------+----------+----------+
|    Instance Name     |       Pin Name       | Parameter | Max (ns) | Min (ns) |
+----------------------+----------------------+-----------+----------+----------+
|       i_arstn        |       i_arstn        |  GPIO_IN  |  1.177   |  0.785   |
|         sw1          |         sw1          |  GPIO_IN  |  1.177   |  0.785   |
...
```

For registered blocks, the values correspond to the system timing as observed at the FPGA's pins. The table shows the max (worst) and min (best) values for setup, hold, and clock to output. You use these values to confirm that your system timing requirements are met.

```
Registered HSIO GPIO Configuration:
====================================

+---------------+----------------+-------+-------+------+------+----------+----------+
| Instance Name |   Clock Pin    |  Max  |  Min  | Max  | Min  |   Max    |   Min    |
                                 | Setup | Setup | Hold | Hold | Clock to | Clock to |
                                 |  (ns) |  (ns) | (ns) | (ns) | Out (ns) | Out (ns) |
+---------------+----------------+-------+-------+------+------+----------+----------+
|   hbc_dq[0]   | i_hbramClk_cal | 0.618 |0.412  |-0.408|-0.272|          |          |
|    hbc_ck_n   | ~i_hbramClk90  |       |       |      |      |  2.226   |  1.484   |
|    hbc_ck_p   | ~i_hbramClk90  |       |       |      |      |  2.226   |  1.484   |
...
```

Important: The reported numbers assume that the clock pin came from a GPIO used as a global clock source (GCLK).

The JTAG Timing Report gives the max and min values for the JTAG pins used in your design.

```
---------- 3. JTAG Timing Report (begin) ----------

+---------------+----------------+-----------+----------+----------+
| Instance Name |    Pin Name    | Parameter | Max (ns) | Min (ns) |
+---------------+----------------+-----------+----------+----------+
|   jtag_inst1  | jtag_inst1_TDI |  JTAG_IN  |  3.164   |  2.109   |
|   jtag_inst1  | jtag_inst1_TMS |  JTAG_IN  |  2.471   |  1.647   |
...
```

Where to Learn More

The Efinity® software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software. You can also access the latest versions of PDF documentation in the Support Center:

- Efinity Software User Guide
- Efinity Synthesis User Guide
- Efinity Timing Closure User Guide
- Efinity Software Installation User Guide
- Efinity Trion Tutorial
- Efinity Debugger Tutorial
- Titanium Interfaces User Guide
- Trion Interfaces User Guide
- Efinity Interface Designer Python API
- Quantum® Trion Primitives User Guide
- Quantum® Titanium Primitives User Guide
- Quantum® Topaz Primitives User Guide

In addition to documentation, Efinix field application engineers have created a series of videos to help you learn about aspects of the software. You can view these videos in the Support Center.

Revision History

Table 8: Revision History

Date	Version	Description
November 2024	7.0	Added check_timing and report_cdc commands. (DOC-2168) Described how to read SDC files into a top-level SDC file. (DOC-2168) An EFX_FF primitive cannot be placed in a Trion ELF tile. (DOC-2197)
June 2024	6.0	Added Tcl Console section describing the enhanced Tcl support in v2024.1. (DOC-1824, DOC-1881) Added get_fanins and set_bus_syntax_mode. (DOC-1722) Added new options for create_generated_clock constraint. (DOC-1768) Updated Best Practices for Constraining Routing and added example flow. (DOC-1891) Added note emphasizing that you can only constrain routing for Titanium FPGAs. (DOC-1801)
December 2023	5.0	Added examples for setting min and max delays on synchronous and asynchronous paths. Added get_fanouts constraint. Updated Clock Latency section; the Efinity software v2023.2 now provides a SDC template for set_clock_latency. Added clock latency topic for PLL cascading. Updated section on constraining I/O. Added section of common mistakes. Updated topic about the <*project*>**.pt.sdc** file. Updated topic about the <*project*>**.pt_timing.rpt** file. In SDC files, square brackets are supported in clock names without using the -name option.
June 2023	4.0	Updated the create_clock, create_generated_clock, set_input_delay, and set_ouput_delay constraint descriptions (new flags). Added information on multiple SDC file support, including how the software handles multiple constraints for the same clock. Added new examples for Clock Latency. Added explanation for virtual clocks. Added more detail and examples for constrainting unsynchronized inputs and outputs. Added SDC examples. Updated Best Practices for Constraining Routing to reflect syn_keep synthesis option. Interpreting Timing Results topic updated for new report format. Explained how to use clock names with square brackets in the name.
December 2022	3.0	The create_generated_clock -source option is a port, pin, or net. (DOC-1027) set_false_path, set_min_delay, and set_max_delay support clock domain, I/O and registers as the start and end point; they also support the -through option. (DOC-995) Added section on constraining routing manually.

Date	Version	Description
August 2022	2.4	Added more details on synchronous input and output delays. Added section on constraining logic. Updated description for set_false_path constraint. Removed limitation that one end point much be a clock. (DOC-875) Added set_clock_latency SDC constraint. Updated the synthesis options. (DOC-870) Updated the place-and-route options. (DOC-889)
April 2022	2.3	Added a note to remind users not to include the <*design name*>**.pt.sdc** in their project. (DOC-670)
December 2021	2.2	Added -reference_pin flag to set_input_delay and set_output_delay. (DOC-488) Added new synthesis options.
June 2021	2.1	Updated for Efinity software v2021.1. Added recommendation for closing timing for DSP Blocks.
March 2021	2.0	Incorporated content from AN 008: Setting Trion Timing Constraints in the Efinity Software (DOC-369). Updated section on place-and-route options. Restructured document and added more examples.
December 2020	1.1	Added the -asynchronous option to set_clock_groups (DOC-317). Added the **efx_run_pnr_sweep.bat** helper script.
June 2020	1.0	Initial release.