Intel® Quartus® Prime Standard Edition User Guide

Timing Analyzer

Updated for Intel® Quartus® Prime Design Suite: **18.1**

This document is part of a collection - Intel® Quartus® Prime Standard Edition User Guides - Combined PDF link

Online Version
Send Feedback
UG-20183
683068
2024.02.21

Contents

1. Timing Analysis Introduction

Comprehensive timing analysis of your design allows you to validate circuit performance, identify timing violations, and drive the Fitter's placement of logic to meet your timing goals. The Intel® Quartus® Prime Timing Analyzer uses industry-standard constraint and analysis methodology to report on all data required times, data arrival times, and clock arrival times for all register-to-register, I/O, and asynchronous reset paths in your design.

The Timing Analyzer verifies that required timing relationships are met for your design to correctly function, and confirms actual signal arrival times against the constraints that you specify. This use guide provides an introduction to basic timing analysis concepts, along with step-by-step instructions for using the Intel Quartus Prime Timing Analyzer.

1.1. Timing Analysis Basic Concepts

This user guide introduces the following concepts to describe timing analysis:

Table 1. Timing Analyzer Terminology

Term	Definition
Arrival time	The Timing Analyzer calculates the data and clock arrival time versus the required time at register pins.
Cell	Device resource that contains look-up tables (LUT), registers, digital signal processing (DSP) blocks, memory blocks, or input/output elements. In Intel Stratix® series devices, the LUTs and registers are contained in logic elements (LE) modeled as cells.
Clock	Named signal representing clock domains inside or outside of your design.
Clock-as-data analysis	More accurate timing analysis for complex paths that includes any phase shift associated with a PLL for the clock path, and considers any related phase shift for the data path.
Clock hold time	Minimum time interval that a signal must be stable on the input pin that feeds a data input or clock enable, after an active transition on the clock input.
Clock launch and latch edge	The launch edge is the clock edge that sends data out of a register or other sequential element, and acts as a source for the data transfer. The latch edge is the active clock edge that captures data at the data port of a register or other sequential element, acting as a destination for the data transfer.
Clock pessimism	Clock pessimism refers to use of the maximum (rather than minimum) delay variation associated with common clock paths during static timing analysis.
Clock setup	Minimum time interval between the assertion of a signal at a data input, and the assertion of a low-to-high transition on the clock input.
Net	A collection of two or more interconnected components.
Node	Represents a wire carrying a signal that travels between different logical components in the design. Most basic timing netlist unit. Used to represent ports, pins, and registers.

continued...

Term	Definition
Pin	Inputs or outputs of cells.
Port	Top-level module inputs or outputs; for example, a device pin.
Metastability	Metastability problems can occur when a signal transfers between circuitry in unrelated or asynchronous clock domains. The Timing Analyzer analyzes the potential for metastability in your design and can calculate the MTBF for synchronization register chains.
Multicorner analysis	Timing analysis of slow and fast timing corners to verify your design under a variety of voltage, process, and temperature operating conditions.
Multicycle paths	A data path that requires a non-default setup or hold relationship for proper analysis.
Recovery and removal time	Recovery time is the minimum length of time for the deassertion of an asynchronous control signal relative to the next clock edge. Removal time is the minimum length of time the deassertion of an asynchronous control signal must be stable after the active clock edge.
Timing netlist	A Compiler-generated list of your design's synthesized nodes and connections. The Timing Analyzer requires this netlist to perform timing analysis.
Timing path	The wire connection (net) between any two design nodes, such as the output of a register to the input of another register.

1.1.1. Timing Path and Clock Analysis

The Timing Analyzer measures the timing performance for all timing paths identified in your design. The Timing Analyzer requires a timing netlist that describes your design's nodes and connections for analysis. The Timing Analyzer also determines clock relationships for all register-to-register transfers in your design by analyzing the clock setup and hold relationship between the launch edge and latch edge of the clock.

1.1.1.1. The Timing Netlist

The Timing Analyzer uses the timing netlist data to determine the data and clock arrival time versus required time for all timing paths in the design. You can generate the timing netlist in the Timing Analyzer any time after running the Fitter or full compilation.

The following figures illustrate how the timing netlist divides the design elements into cells, pins, nets, and ports for measurement of delay.

Figure 1. Simple Design Schematic

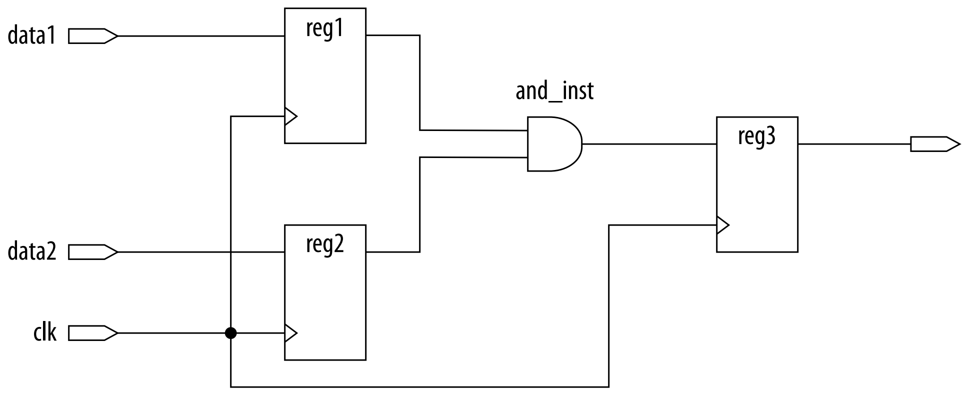
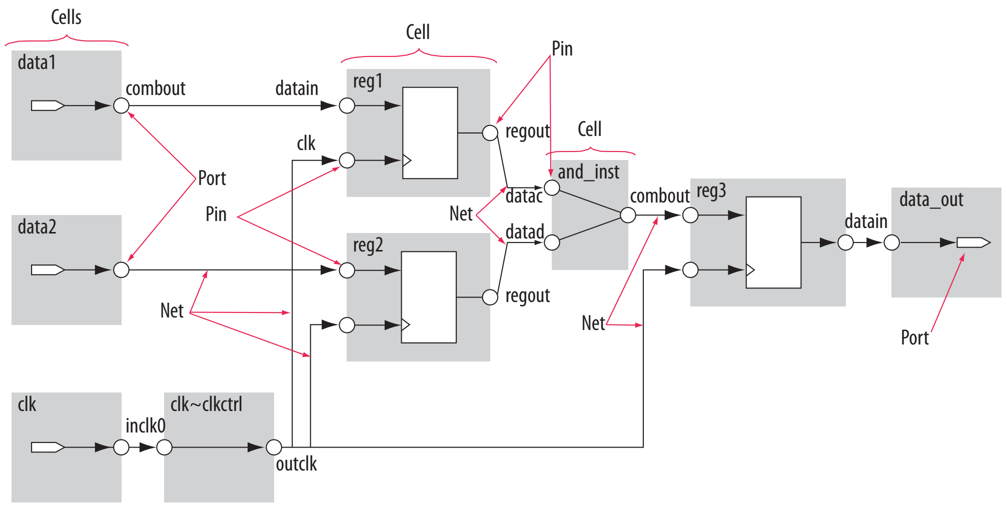


Figure 2. Division of Simple Design Schematic Elements in Timing Netlist



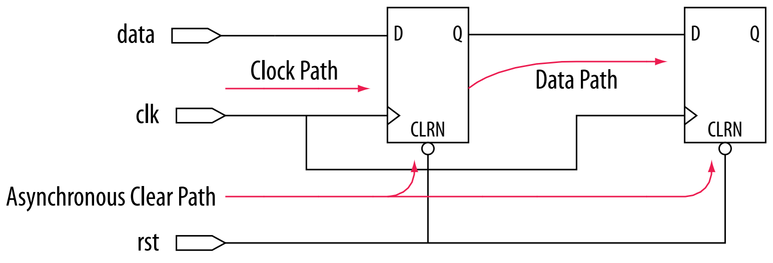
1.1.1.2. Timing Paths

Timing paths connect two design nodes, such as the output of a register to the input of another register.

Understanding the types of timing paths is important to timing closure and optimization. The Timing Analyzer recognizes and analyzes the following timing paths:

- **Edge paths**—connections from ports-to-pins, from pins-to-pins, and from pins-to-ports.
- **Clock paths**—connections from device ports or internally generated clock pins to the clock pin of a register.
- **Data paths**—connections from a port or the data output pin of a sequential element to a port or the data input pin of another sequential element.
- **Asynchronous paths**—connections from a port or asynchronous pins of another sequential element such as an asynchronous reset or asynchronous clear.

Figure 3. Path Types Commonly Analyzed by the Timing Analyzer



In addition to identifying various paths in a design, the Timing Analyzer analyzes clock characteristics to compute the worst-case requirement between any two registers in a single register-to-register path. You must constrain all clocks in your design before analyzing clock characteristics.

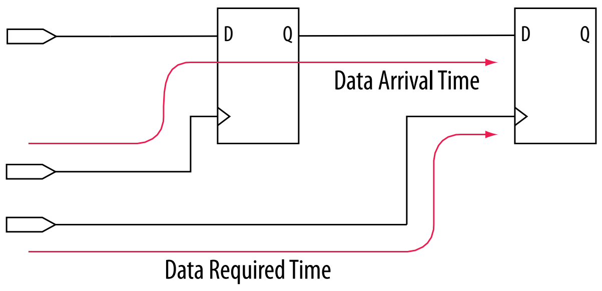
1.1.1.3. Data and Clock Arrival Times

After the Timing Analyzer identifies the path type, the Timing Analyzer can report data and clock arrival times at register pins.

The Timing Analyzer calculates data arrival time by adding the launch edge time to the delay from the clock source to the clock pin of the source register, the micro clock-to-output delay (μt_{CO}) of the source register, and the delay from the source register's data output (Q) to the destination register's data input (D).

The Timing Analyzer calculates data required time by adding the latch edge time to the sum of all delays between the clock port and the clock pin of the destination register, including any clock port buffer delays, and subtracts the micro setup time (μt_{SU}) of the destination register, where the μt_{SU} is the intrinsic setup time of an internal register in the FPGA.

Figure 4. Data Arrival and Data Required Times



The basic calculations for data arrival and data required times including the launch and latch edges.

Figure 5. Data Arrival and Data Required Time Equations

$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Source Clock Delay} + \mu t_{CO} + \text{Register-to-Register Delay}$$

$$\text{Data Required Time} = \text{Latch Edge} + \text{Destination Clock Delay} - \mu t_{SU}$$

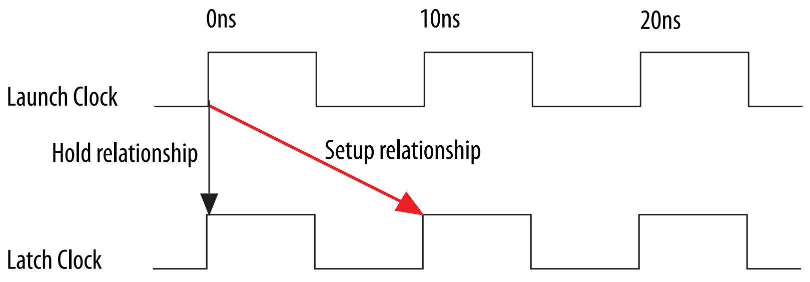
1.1.1.4. Launch and Latch Edges

All timing analysis requires the presence of one or more clock signals. The Timing Analyzer determines clock relationships for all register-to-register transfers in your design by analyzing the clock setup and hold relationship between the launch edge and latch edge of the clock.

The launch edge of the clock signal is the clock edge that sends data out of a register or other sequential element, and acts as a source for the data transfer. The latch edge is the active clock edge that captures data at the data port of a register or other sequential element, acting as a destination for the data transfer.

Figure 6. Setup and Hold Relationship for Launch and Latch Edges 10ns Apart

In this example, the launch edge sends the data from register `reg1` at 0 ns, and the register `reg2` captures the data when triggered by the latch edge at 10 ns. The data arrives at the destination register before the next latch edge.



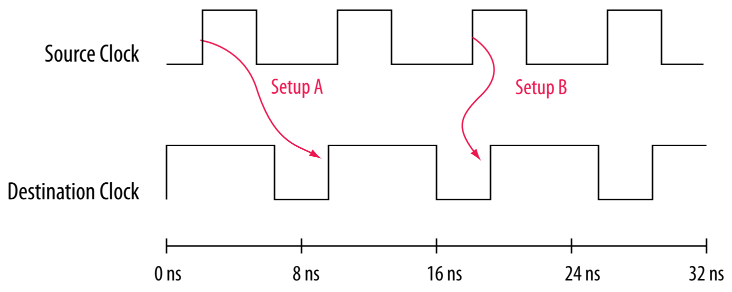
You create define constraints for all clocks and assign the constraints to nodes in your design. These clock constraints provide the structure required for repeatable data relationships. If you do not constrain the clocks in your design, the Intel Quartus Prime software analyzes in terms of a 1 GHz clock to maximize timing based Fitter effort. To ensure realistic slack values, you must constrain all clocks in your design with real values.

1.1.2. Clock Setup Analysis

To perform a clock setup check, the Timing Analyzer determines a setup relationship by analyzing each launch and latch edge for each register-to-register path.

For each latch edge at the destination register, the Timing Analyzer uses the closest previous clock edge at the source register as the launch edge. The following figure shows two setup relationships, setup A and setup B. For the latch edge at 10 ns, the closest clock that acts as a launch edge is at 3 ns and has the setup A label. For the latch edge at 20 ns, the closest clock that acts as a launch edge is 19 ns and has the setup B label. The Timing Analyzer analyzes the most restrictive setup relationship, in this case setup B; if that relationship meets the design requirement, then setup A meets the requirement by default.

Figure 7. Setup Check



The Timing Analyzer reports the result of clock setup checks as slack values. Slack is the margin by which a timing requirement is met or not met. Positive slack indicates the margin by which a requirement is met; negative slack indicates the margin by which a requirement is not met.

Figure 8. Clock Setup Slack for Internal Register-to-Register Paths

$$\begin{aligned}
\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time} \\
\text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay} \\
\text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU} - \text{Setup Uncertainty}
\end{aligned}$$

The Timing Analyzer performs setup checks using the maximum delay when calculating data arrival time, and minimum delay when calculating data required time.

Figure 9. Clock Setup Slack from Input Port to Internal Register

$$\begin{aligned}
\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time} \\
\text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Maximum Delay} + \text{Port-to-Register Delay} \\
\text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU} - \text{Setup Uncertainty}
\end{aligned}$$

Figure 10. Clock Setup Slack from Internal Register to Output Port

$$\begin{aligned}
\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time} \\
\text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Output Port} - \text{Output Maximum Delay} \\
\text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Port Delay}
\end{aligned}$$

1.1.3. Clock Hold Analysis

To perform a clock hold check, the Timing Analyzer determines a hold relationship for each possible setup relationship that exists for all source and destination register pairs. The Timing Analyzer checks all adjacent clock edges from all setup relationships to determine the hold relationships.

The Timing Analyzer performs two hold checks for each setup relationship. The first hold check determines that the data launched by the current launch edge is not captured by the previous latch edge. The second hold check determines that the data launched by the next launch edge is not captured by the current latch edge. From the possible hold relationships, the Timing Analyzer selects the hold relationship that is the most restrictive. The most restrictive hold relationship is the hold relationship with the smallest difference between the latch and launch edges and determines the minimum allowable delay for the register-to-register path. In the following example, the Timing Analyzer selects hold check A2 as the most restrictive hold relationship of two setup relationships, setup A and setup B, and their respective hold checks.

Figure 11. Setup and Hold Check Relationships

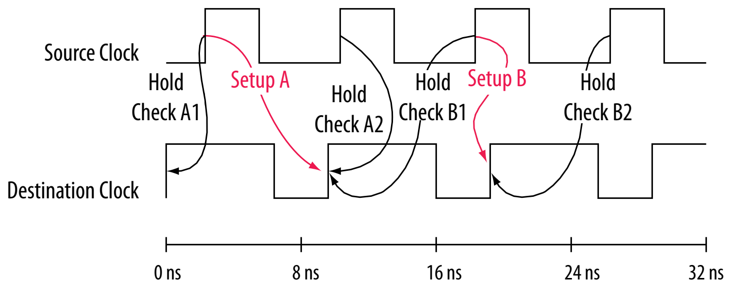


Figure 12. Clock Hold Slack for Internal Register-to-Register Paths

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H} + \text{Hold Uncertainty}$$

The Timing Analyzer performs hold checks using the minimum delay when calculating data arrival time, and maximum delay when calculating data required time.

Figure 13. Clock Hold Slack Calculation from Input Port to Internal Register

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Minimum Delay} + \text{Pin-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H}$$

Figure 14. Clock Hold Slack Calculation from Internal Register to Output Port

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Pin Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay} - \text{Output Minimum Delay}$$

1.1.4. Recovery and Removal Analysis

Recovery time is the minimum length of time for the deassertion of an asynchronous control signal relative to the next clock edge.

For example, signals such as `clear` and `preset` must be stable before the next active clock edge. The recovery slack calculation is similar to the clock setup slack calculation, but the calculation applies to asynchronous control signals.

Figure 15. Recovery Slack Calculation if the Asynchronous Control Signal is Registered

$$\text{Recovery Slack Time} = \text{Data Required Time} - \text{Data Arrival Time}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}$$

Figure 16. Recovery Slack Calculation if the Asynchronous Control Signal is not Registered

$$\text{Recovery Slack Time} = \text{Data Required Time} - \text{Data Arrival Time}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Maximum Delay} + \text{Port-to-Register Delay}$$

Note: If the asynchronous reset signal is from a device I/O port, you must create an input delay constraint for the asynchronous reset port for the Timing Analyzer to perform recovery analysis on the path.

Removal time is the minimum length of time the deassertion of an asynchronous control signal must be stable after the active clock edge. The Timing Analyzer removal slack calculation is similar to the clock hold slack calculation, but the calculation applies asynchronous control signals.

Figure 17. Removal Slack Calculation if the Asynchronous Control Signal is Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} of Source Register + Register-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_H

Figure 18. Removal Slack Calculation if the Asynchronous Control Signal is not Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay + Input Minimum Delay of Pin + Minimum Pin-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_H

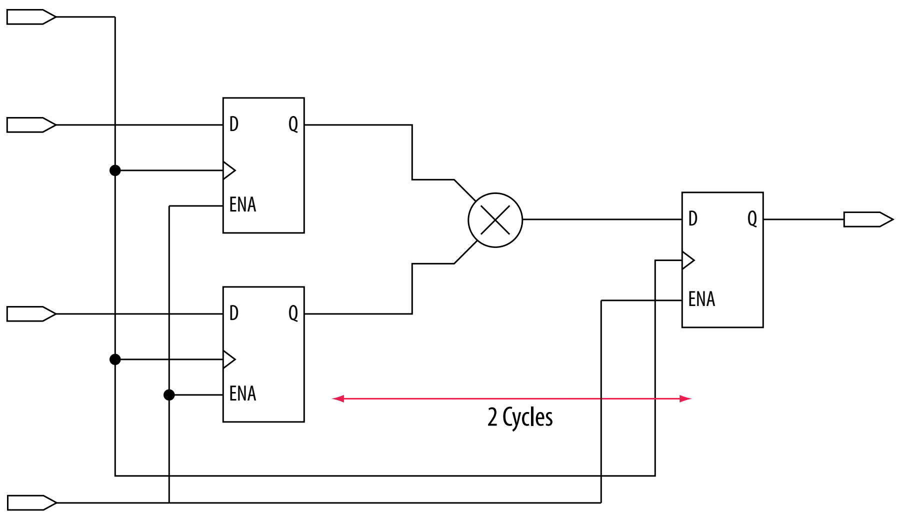
If the asynchronous reset signal is from a device pin, you must assign the **Input Minimum Delay** timing assignment to the asynchronous reset pin for the Timing Analyzer to perform removal analysis on the path.

1.1.5. Multicycle Path Analysis

Multicycle paths are data paths that require either a non-default setup or hold relationship, for proper analysis.

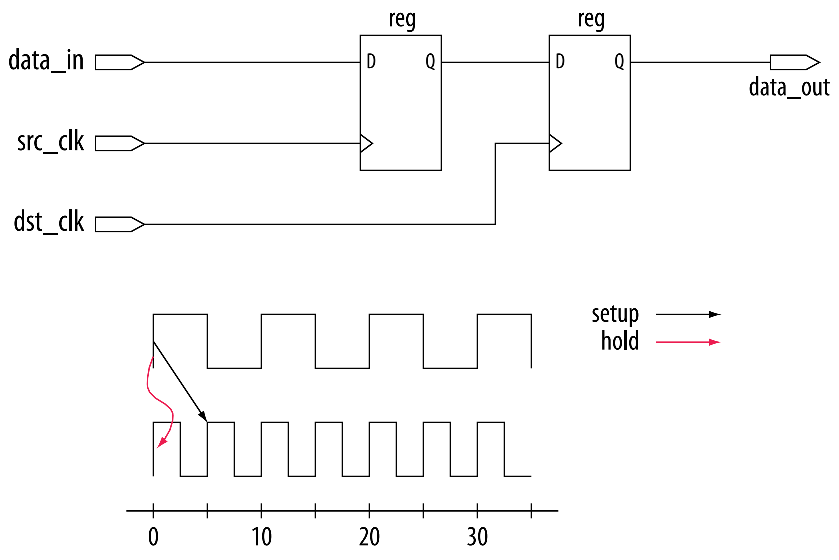
For example, a register may be required to capture data on every second or third rising clock edge. An example of a multicycle path between the input registers of a multiplier and an output register where the destination latches data on every other clock edge.

Figure 19. Multicycle Path



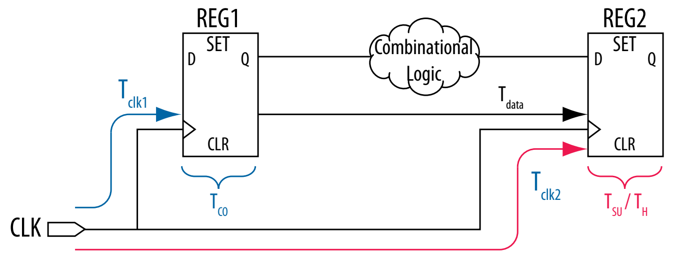
A register-to-register path used for the default setup and hold relationship, the respective timing diagrams for the source and destination clocks, and the default setup and hold relationships, when the source clock, `src_clk`, has a period of 10 ns and the destination clock, `dst_clk`, has a period of 5 ns. The default setup relationship is 5 ns; the default hold relationship is 0 ns.

Figure 20. Register-to-Register Path and Default Setup and Hold Timing Diagram



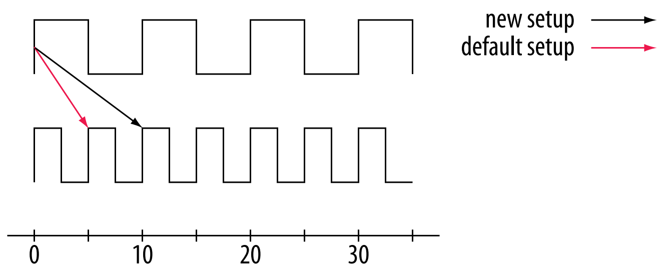
To accommodate the system requirements you can modify the default setup and hold relationships by specifying a multicycle timing constraint to a register-to-register path.

Figure 21. Register-to-Register Path



The exception has a multicycle setup assignment of two to use the second occurring latch edge; in this example, to 10 ns from the default value of 5 ns.

Figure 22. Modified Setup Diagram



1.1.5.1. Multicycle Clock Hold

The number of clock periods between the clock launch edge and the latch edge defines the setup relationship.

By default, the Timing Analyzer performs a single-cycle path analysis, which results in the hold relationship being equal to one clock period (launch edge – latch edge).When analyzing a path, the Timing Analyzer performs two hold checks. The first hold check determines that the data that launches from the current launch edge is not captured by the previous latch edge. The second hold check determines that the data that launches from the next launch edge is not captured by the current latch edge. The Timing Analyzer reports only the most restrictive hold check. The Timing Analyzer calculates the hold check by comparing launch and latch edges.

The calculation the Timing Analyzer performs to determine the hold check.

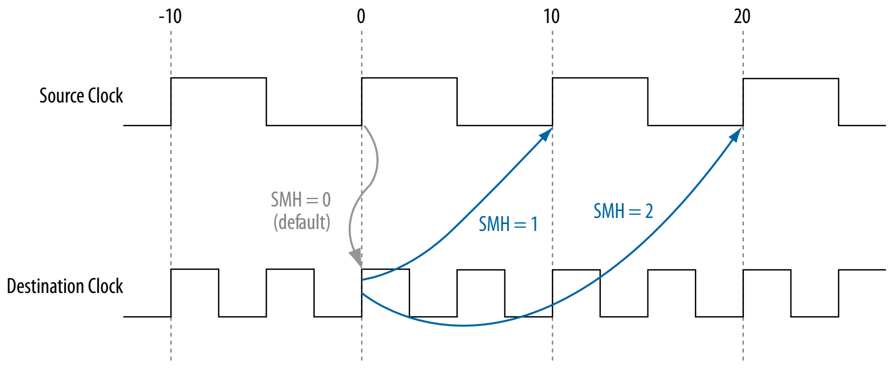
Figure 23. Hold Check

$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$
$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$

Tip: If a hold check overlaps a setup check, the hold check is ignored.

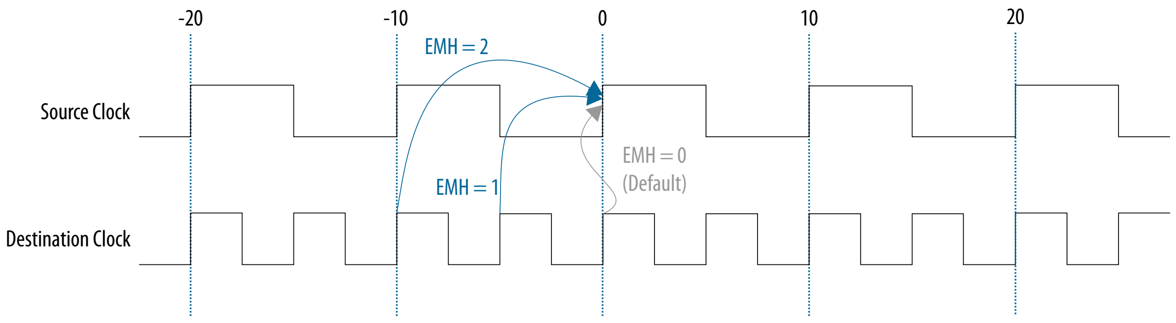
A start multicycle hold assignment modifies the launch edge of the destination clock by moving the latch edge the number of clock periods you specify to the right of the default launch edge. The following figure shows various values of the start multicycle hold (SMH) assignment and the resulting launch edge.

Figure 24. Start Multicycle Hold Values



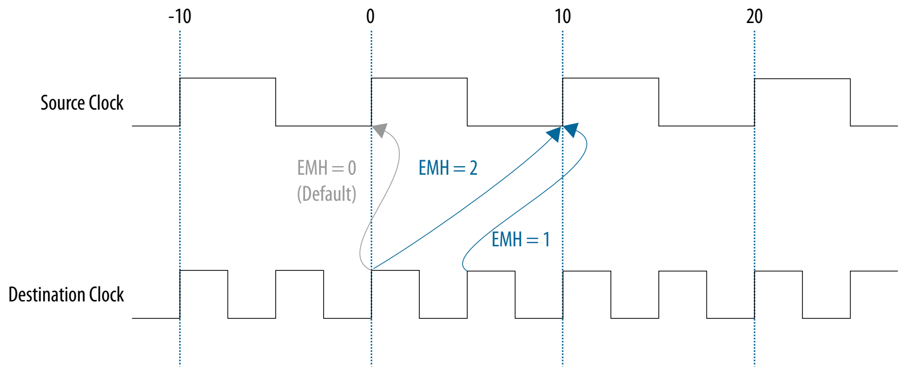
An end multicycle hold assignment modifies the latch edge of the destination clock by moving the latch edge the specific number of clock periods to the left of the default latch edge. The following figure shows various values of the end multicycle hold (EMH) assignment and the resulting latch edge.

Figure 25. End Multicycle Hold Values



The following shows the hold relationship the Timing Analyzer reports for the negative hold relationship:

Figure 26. End Multicycle Hold Values the Timing Analyzer Reports

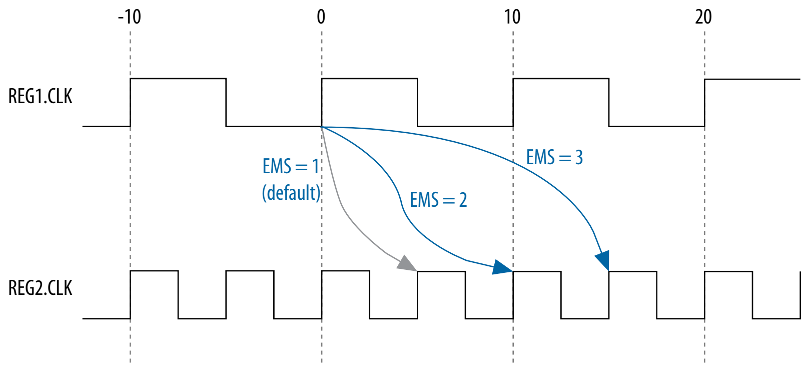


1.1.5.2. Multicycle Clock Setup

The setup relationship is defined as the number of clock periods between the latch edge and the launch edge. By default, the Timing Analyzer performs a single-cycle path analysis, which results in the setup relationship being equal to one clock period (latch edge – launch edge). Applying a multicycle setup assignment, adjusts the setup relationship by the multicycle setup value. The adjustment value may be negative.

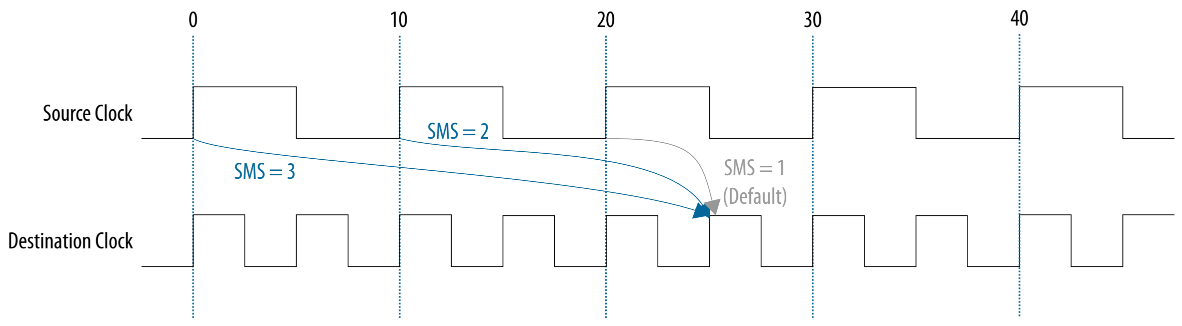
An end multicycle setup assignment modifies the latch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default latch edge. The following figure shows various values of the end multicycle setup (EMS) assignment and the resulting latch edge.

Figure 27. End Multicycle Setup Values



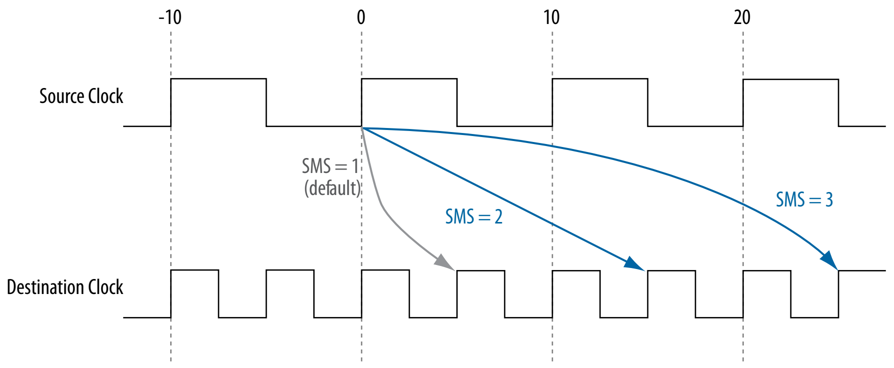
A start multicycle setup assignment modifies the launch edge of the source clock by moving the launch edge the specified number of clock periods to the left of the determined default launch edge. A start multicycle setup (SMS) assignment with various values can result in a specific launch edge.

Figure 28. Start Multicycle Setup Values



The setup relationship reported by the Timing Analyzer for the negative setup relationship.

Figure 29. Start Multicycle Setup Values Reported by the Timing Analyzer



1.1.6. Metastability Analysis

Metastability problems can occur when a signal transfers between circuitry in unrelated or asynchronous clock domains because the signal does not meet setup and hold time requirements.

To minimize the failures due to metastability, circuit designers typically use a sequence of registers, also known as a synchronization register chain, or synchronizer, in the destination clock domain to resynchronize the data signals to the new clock domain.

The mean time between failures (MTBF) is an estimate of the average time between instances of failure due to metastability.

The Timing Analyzer analyzes the potential for metastability in your design and can calculate the MTBF for synchronization register chains. The Timing Analyzer then estimates the MTBF of the entire design from the synchronization chains the design contains.

In addition to reporting synchronization register chains found in the design, the Intel Quartus Prime software also protects these registers from optimizations that might negatively impact MTBF, such as register duplication and logic retiming. The Intel Quartus Prime software can also optimize the MTBF of your design if the MTBF is too low.

Send Feedback

Related Information

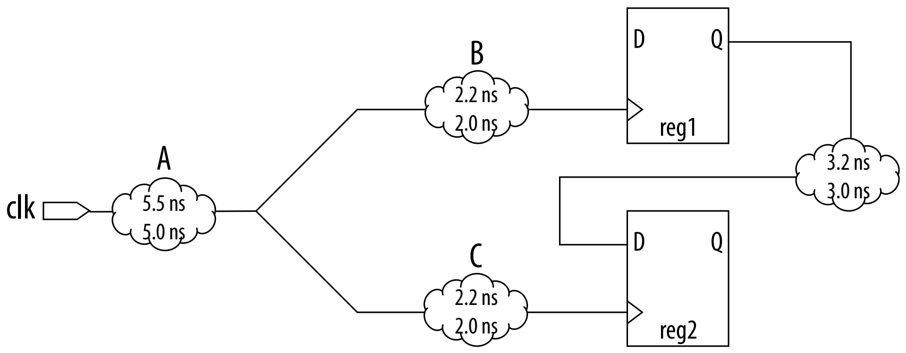
Understanding Metastability in FPGAs

1.1.7. Timing Pessimism

Common clock path pessimism removal accounts for the minimum and maximum delay variation associated with common clock paths during static timing analysis by adding the difference between the maximum and minimum delay value of the common clock path to the appropriate slack equation.

Minimum and maximum delay variation can occur when two different delay values are used for the same clock path. For example, in a simple setup analysis, the maximum clock path delay to the source register is used to determine the data arrival time. The minimum clock path delay to the destination register is used to determine the data required time. However, if the clock path to the source register and to the destination register share a common clock path, both the maximum delay and the minimum delay are used to model the common clock path during timing analysis. The use of both the minimum delay and maximum delay results in an overly pessimistic analysis since two different delay values, the maximum and minimum delays, cannot be used to model the same clock path.

Figure 30. Typical Register to Register Path

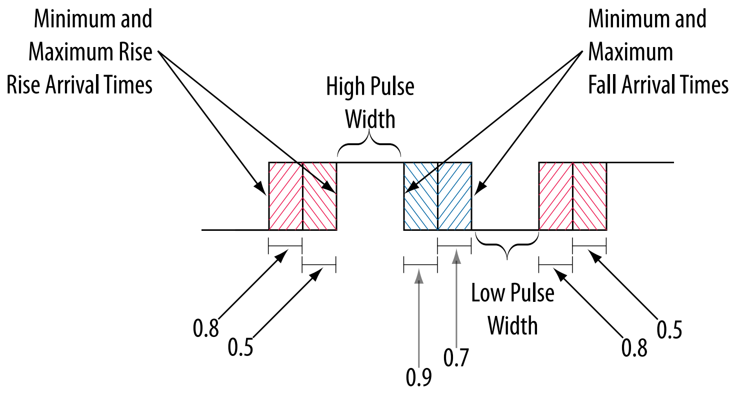


Segment A is the common clock path between `reg1` and `reg2`. The minimum delay is 5.0 ns; the maximum delay is 5.5 ns. The difference between the maximum and minimum delay value equals the common clock path pessimism removal value; in this case, the common clock path pessimism is 0.5 ns. The Timing Analyzer adds the common clock path pessimism removal value to the appropriate slack equation to determine overall slack. Therefore, if the setup slack for the register-to-register path in the example equals 0.7 ns without common clock path pessimism removal, the slack is 1.2 ns with common clock path pessimism removal.

You can also use common clock path pessimism removal to determine the minimum pulse width of a register. A clock signal must meet a register’s minimum pulse width requirement to be recognized by the register. A minimum high time defines the minimum pulse width for a positive-edge triggered register. A minimum low time defines the minimum pulse width for a negative-edge triggered register.

Clock pulses that violate the minimum pulse width of a register prevent data from being latched at the data pin of the register. To calculate the slack of the minimum pulse width, the Timing Analyzer subtracts the required minimum pulse width time from the actual minimum pulse width time. The Timing Analyzer determines the actual minimum pulse width time by the clock requirement you specified for the clock that feeds the clock port of the register. The Timing Analyzer determines the required minimum pulse width time by the maximum rise, minimum rise, maximum fall, and minimum fall times.

Figure 31. Required Minimum Pulse Width time for the High and Low Pulse



With common clock path pessimism, the minimum pulse width slack can be increased by the smallest value of either the maximum rise time minus the minimum rise time, or the maximum fall time minus the minimum fall time. In the example, the slack value can be increased by 0.2 ns, which is the smallest value between 0.3 ns (0.8 ns – 0.5 ns) and 0.2 ns (0.9 ns – 0.7 ns).

1.1.8. Clock-As-Data Analysis

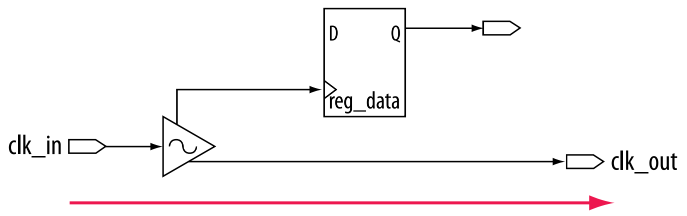
The majority of FPGA designs contain simple connections between any two nodes known as either a data path or a clock path.

A data path is a connection between the output of a synchronous element to the input of another synchronous element.

A clock is a connection to the clock pin of a synchronous element. However, for more complex FPGA designs, such as designs that use source-synchronous interfaces, this simplified view is no longer sufficient. Clock-as-data analysis is performed in circuits with elements such as clock dividers and DDR source-synchronous outputs.

The connection between the input clock port and output clock port can be treated either as a clock path or a data path. A design where the path from port `clk_in` to port `clk_out` is both a clock and a data path. The clock path is from the port `clk_in` to the register `reg_data` clock pin. The data path is from port `clk_in` to the port `clk_out`.

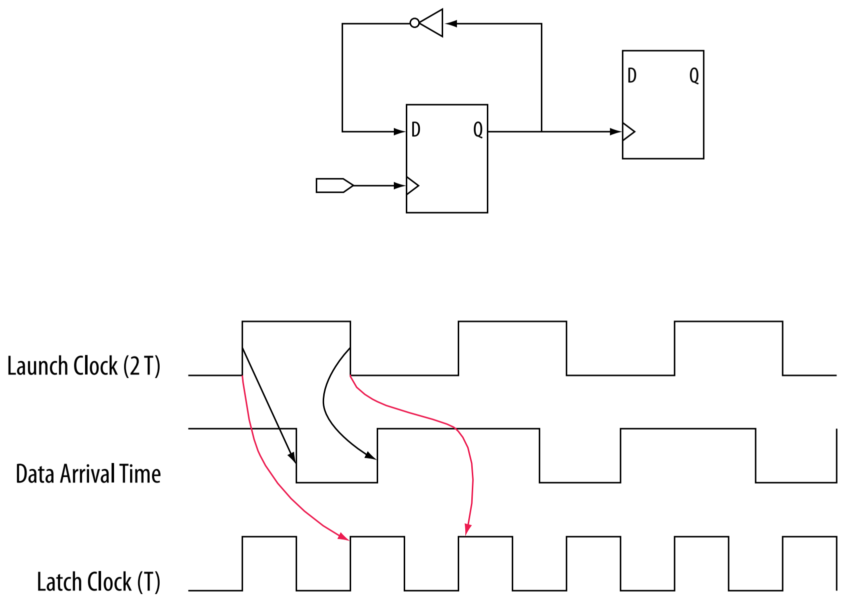
Figure 32. Simplified Source Synchronous Output



With clock-as-data analysis, the Timing Analyzer provides a more accurate analysis of the path based on user constraints. For the clock path analysis, any phase shift associated with the phase-locked loop (PLL) is taken into consideration. For the data path analysis, any phase shift associated with the PLL is taken into consideration rather than ignored.

The clock-as-data analysis also applies to internally generated clock dividers. An internally generated clock divider. In this figure, waveforms are for the inverter feedback path, analyzed during timing analysis. The output of the divider register is used to determine the launch time and the clock port of the register is used to determine the latch time.

Figure 33. Clock Divider



1.1.9. Multicorner Analysis

The Timing Analyzer performs multicorner timing analysis to verify your design under a variety of operating conditions—such as voltage, process, and temperature—while performing static timing analysis.

To change the operating conditions or speed grade of the current device for timing analysis, use the `set_operating_conditions` command.

If you specify an operating condition Tcl object, the `-model`, `-speed`, `-temperature`, and `-voltage` options are available. If you do not specify an operating condition Tcl object, Tcl requires the `-model` option. `-speed`, `-temperature`, and `-voltage` are optional.

Tip: To obtain a list of available operating conditions for the target device, use the `get_available_operating_conditions -all` command.

To ensure that no violations occur under various conditions during the device operation, perform static timing analysis under all available operating conditions.

Table 2. Operating Conditions for Slow and Fast Models

Model	Speed Grade	Voltage	Temperature
Slow	Slowest speed grade in device density	V_{CC} minimum supply (1)	Maximum T_J (1)
Fast	Fastest speed grade in device density	V_{CC} maximum supply (1)	Minimum T_J (1)

Note :
1. Refer to the DC & Switching Characteristics chapter of the applicable device Handbook for V_{CC} and T_J. values

In your design, you can set the operating conditions for to the slow timing model, with a voltage of 1100 mV, and temperature of 85° C with the following code:

```
set_operating_conditions -model slow -temperature 85 -voltage 1100
```

You can set the same operating conditions with a Tcl object:

```
set_operating_conditions 3_slow_1100mv_85c
```

The following block of code shows how to use the `set_operating_conditions` command to generate different reports for various operating conditions.

Example 1. Script Excerpt for Analysis of Various Operating Conditions

```
#Specify initial operating conditions
set_operating_conditions -model slow -speed 3 -grade c -temperature 85 -voltage
1100
#Update the timing netlist with the initial conditions
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C
set_operating_conditions -model slow -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C and a model of fast
set_operating_conditions -model fast -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
```

1.2. Document Revision History

Table 3. Document Revision History

Date	Version	Changes
2018.09.24	18.1.0	• Minor text enhancements for clarity and style.
2016.05.02	16.0.0	Corrected typo in Fig 6-14: Clock Hold Slack Calculation from Internal Register to Output Port
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1.0	Moved Multicycle Clock Setup Check and Hold Check Analysis section from the Timing Analyzer chapter.
June 2014	14.0.0	Updated format
June 2012	12.0.0	Added social networking icons, minor text updates
November 2011	11.1.0	Initial release.

2. Using the Intel Quartus Prime Timing Analyzer

The Intel Quartus Prime Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology. Use the Timing Analyzer GUI or command-line interface to constrain, analyze, and report results for all timing paths in your design.

Related Information

Intel FPGA Technical Training

2.1. Enhanced Timing Analysis for Intel Arria® 10 Devices

The Timing Analyzer supports new timing algorithms for the Intel Arria® 10 device family which significantly improve the speed of the analysis.

These algorithms are enabled by default for Intel Arria 10 devices, and can be enabled for earlier families with an assignment. The new analysis engine analyzes the timing graph a fixed number of times. Previous Timing Analyzer analysis analyzed the timing graph as many times as there were constraints in your Synopsys Design Constraint (SDC) file.

The new algorithms also support incremental timing analysis, which allows you to modify a single block and re-analyze while maintaining a fully analyzed design.

You can turn on the new timing algorithms for use with Arria V, Cyclone® V, and Stratix V devices with the following QSF assignment:

```
set_global_assignment -name TIMEQUEST2 ON
```

2.2. Basic Timing Analysis Flow

The Intel Quartus Prime Timing Analyzer performs constraint validation and reports timing performance as part of the full compilation flow. After creating your design and setting up a project, you define the required timing parameters (that is, constraints) for your design in a Synopsys* Design Constraints (`.sdc`) file. The Fitter attempts to place logic to meet or exceed the constraints you specify. The Timing Analyzer reports conditions that do not meet your constraints, allowing you to locate and correct critical timing issues. The following steps describe the basic timing analysis flow in the Intel Quartus Prime software.

2.2.1. Step 1: Open a Project and Run the Fitter

Before running timing analysis, you must open an Intel Quartus Prime project and run the Fitter to elaborate the design hierarchy, synthesize logic, and perform place and route.

1. Click **File > New Project Wizard** to create a new project, or click **File ➤ Open Project** to open an existing project.
2. To run the Fitter (and any prerequisite Compiler modules), click **Processing ➤ Start ➤ Start Fitter**.

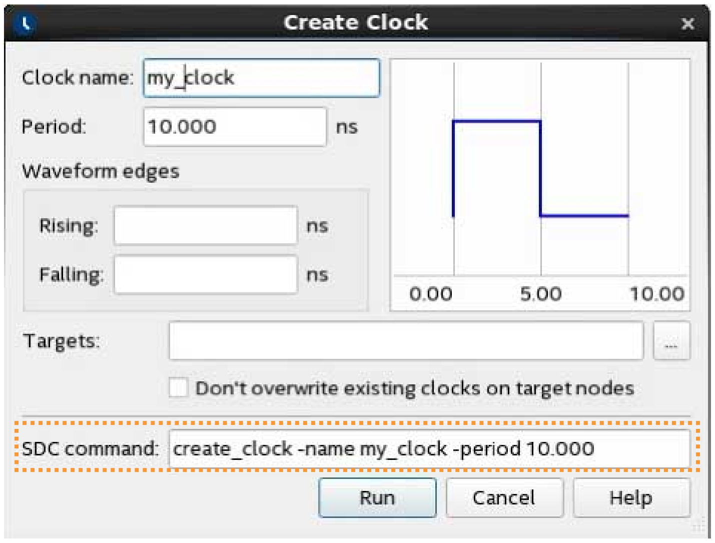
2.2.2. Step 2: Specify Timing Constraints

You must specify timing constraints that describe the clock frequency requirements, timing exceptions, and I/O timing requirements of your design for comparison against actual conditions during timing analysis. You define timing constraints in one or more Synopsys Design Constraints (`.sdc`) files that you add to the project.

If you are unfamiliar with `.sdc` files, you can create an initial `.sdc` file in the Timing Analyzer GUI, or with provided `.sdc` file templates. If you are familiar with timing analysis, you can create an `.sdc` file in any text editor, and then add the file to the project.

1. Use any combination of the following to enter the timing constraints for your design in an `.sdc` file:
    - Enter constraints in the Timing Analyzer GUI—click **Tools ➤ Timing Analyzer**, click **Update Timing Netlist**, and then enter constraints from the Constraints menu. The GUI displays the corresponding SDC command that applies.
    - Create an `.sdc` file on your own. You can start by adding the Recommended Initial SDC Constraints on page 32, and then iteratively modify .sdc constraints and reanalyze the timing results. You must first create clock constraints before entering any constraints dependent on the clock.

Figure 34. Create Clock Dialog Defines Clock Constraints



Clock name: my_clock
Period: 10.000 ns
Targets:
Don't overwrite existing clocks on target nodes
SDC command: create_clock -name my_clock -period 10.000

2. Save the `.sdc` file. When entering constraints in the Timing Analyzer GUI, click **Constraints ➤ Write SDC File** to save the constraints you enter in the GUI to an `.sdc` file.
3. Add the `.sdc` file to your project, as Step 3: Specify General Timing Analyzer Settings on page 22 describes.

2.2.3. Step 3: Specify General Timing Analyzer Settings

Before running timing analysis, you can consider and optionally specify the following Timing Analyzer and Compiler settings that have an impact on the analysis results:

Table 4. Timing Analyzer and Compiler Settings

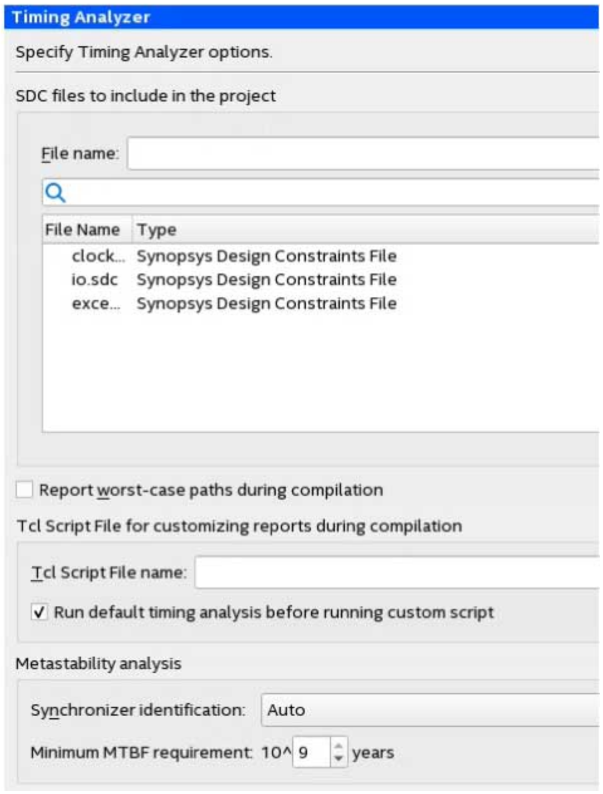
Setting	Description	Location
SDC files to include in the project	Specifies the name and order of Synopsis Design Constraint (`.sdc`) files in the project.	**Assignments ➤ Settings ➤ Timing Analyzer**
Report worst-case paths during compilation	Displays summary of the worst-case timing paths in the design.	**Assignments ➤ Settings ➤ Timing Analyzer**
Tcl Script File name	Specifies the file name for a custom analysis script. You can specify whether to **Run default timing analysis before running the custom script**.	**Assignments ➤ Settings ➤ Timing Analyzer**
Metastability analysis	Specifies how the Timing Analyzer identifies registers as being part of a synchronization register chain for metastability analysis.	**Assignments ➤ Settings ➤ Timing Analyzer**
Enable multicorner support for Timing Analyzer and EDA Netlist Writer	Directs the Timing Analyzer to perform multicorner timing analysis by default, which analyzes the design against best-case and worst-case operating conditions.	**Assignments ➤ Settings ➤ Compilation Process Settings**
Optimization Mode	Specifies the focus of Compiler optimization efforts during synthesis and fitting. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Power**, **Routability**, or **Compile Time**.	**Assignments ➤ Settings ➤ Compiler Settings**

continued...

Send Feedback

Setting	Description	Location
SDC Constraint Protection	Verifies `.sdc` constraints in register merging. This option helps to maintain the validity of `.sdc` constraints through compilation.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
Synchronization Register Chain Length	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. Synchronization chains are sequences of registers with the same clock and no fan-out in between, such that the first register is fed by a pin, or by logic in another clock domain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers. The default length is set to two.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
Optimize Design for Metastability	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). When you enable this setting, the Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the Timing Analyzer for timing-driven compilation. Use the Timing Analyzer `report_metastability` command to review the synchronizers detected in your design and to produce MTBF estimates.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**

Figure 35. Timing Analyzer Settings


Timing Analyzer

Specify Timing Analyzer options.

SDC files to include in the project

File name:

File Name	Type
clock...	Synopsys Design Constraints File
io.sdc	Synopsys Design Constraints File
exce...	Synopsys Design Constraints File

☐ Report worst-case paths during compilation

Tcl Script File for customizing reports during compilation

Tcl Script File name:

✓ Run default timing analysis before running custom script

Metastability analysis

Synchronizer identification: Auto

Minimum MTBF requirement: 10^ 9 years

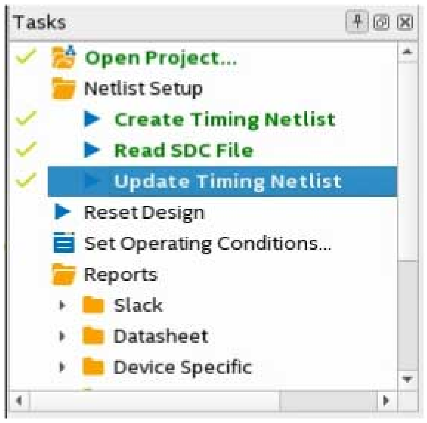
2.2.4. Step 4: Run Timing Analysis

After specifying initial timing constraints, you can run the Fitter or full compilation to generate the timing netlist and run the Timing Analyzer. During compilation, the Fitter attempts to place logic of your design to comply with the timing constraints that you specify. The Timing Analyzer reports the margin (slack) by which your design meets or fails each constraint.

1. To generate the timing netlist, perform either of the following:
   - To run full compilation that includes timing analysis, click **Processing ➤ Start Compilation**. The Timing Analyzer automatically performs multi-corner signoff timing analysis after the Fitter completes.

     Or
   - To run the Fitter, click **Processing ➤ Start ➤ Start Fitter**.
2. To launch the Timing Analyzer, click **Tools ➤ Timing Analyzer**.
3. In the **Tasks** pane, double-click **Update Timing Netlist**. The Timing Analyzer loads the timing netlist, reads all of the project's `.sdc` files, and generates a default set of timing reports, including the Timing Analyzer Summary and Advanced I/O Timing reports.

Figure 36. Timing Analyzer Tasks



4. In the In the **Tasks** pane, under **Reports**, double-click any individual task to generate the report and analyze the results, as Step 5: Analyze Timing Analysis Results on page 25 describes.

Related Information

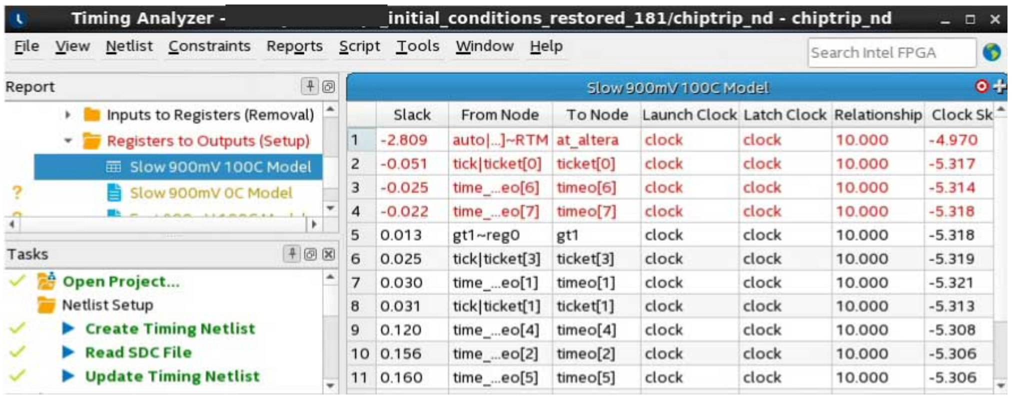
- Timing Analysis of Imported Compilation Results on page 87
- Timing Analyzer Tcl Commands on page 82
- Basic Timing Analysis Flow on page 20
- The quartus_sta Executable on page 82

2.2.5. Step 5: Analyze Timing Analysis Results

During analysis, the Timing Analyzer examines the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as positive slack or negative slack. Negative slack indicates a timing violation. Positive slack indicates the design meets the constraint.

The Timing Analyzer provides very fine-grained reporting and analysis capabilities to identify and correct violations along timing paths. Generate timing reports to view how to best optimize the critical paths in your design. If you modify, remove, or add constraints, re-run timing analysis. This iterative process helps resolve timing violations in your design.

Figure 37. Timing Analyzer Shows Failing Paths in Red



Timing Analyzer - [redacted] initial_conditions_restored_181/chiptrip_nd - chiptrip_nd

Slow 900mV 100C Model

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Sk
1	-2.809	auto|...]~RTM	at_altera	clock	clock	10.000	-4.970
2	-0.051	tick|ticket[0]	ticket[0]	clock	clock	10.000	-5.317
3	-0.025	time_...eo[6]	timeo[6]	clock	clock	10.000	-5.314
4	-0.022	time_...eo[7]	timeo[7]	clock	clock	10.000	-5.318
5	0.013	gt1~reg0	gt1	clock	clock	10.000	-5.318
6	0.025	tick|ticket[3]	ticket[3]	clock	clock	10.000	-5.319
7	0.030	time_...eo[1]	timeo[1]	clock	clock	10.000	-5.321
8	0.031	tick|ticket[1]	ticket[1]	clock	clock	10.000	-5.313
9	0.120	time_...eo[4]	timeo[4]	clock	clock	10.000	-5.308
10	0.156	time_...eo[2]	timeo[2]	clock	clock	10.000	-5.306
11	0.160	time_...eo[5]	timeo[5]	clock	clock	10.000	-5.306

Reports that indicate failing timing performance appear in red text, reports that pass appear in black text. A gold question mark icon indicates reports that are outdated due to SDC changes since generation. Regenerate these reports to show the latest data.

The following sections describe how to generate various timing reports for analysis.

2.2.5.1. Timing Report Commands

The Timing Analyzer generates only a subset of all available reports by default, including the Timing Analyzer Summary report. However, you can generate many other detailed reports in the Timing Analyzer GUI, or with command-line commands. You can customize the display of information in the reports.

Table 5. Timing Analyzer Report Generation Command Summary

Timing Analyzer Tasks Pane GUI	Command-Line	Generates
Custom Reports ➤ Report Timing	`report_timing`	Timing report
Custom Reports ➤ Report Exceptions	`report_exceptions`	Exceptions report
Diagnostic ➤ Report Clock Transfers	`report_clock_transfers`	Clock Transfers report
Slack ➤ Report Minimum Pulse Width Summary	`report_min_pulse_width`	Minimum Pulse Width Summary report
Diagnostic ➤ Report Unconstrained Paths	`report_ucp`	Unconstrained Paths report

2.2.5.2. Set Operating Conditions

You can specify various operating conditions to analyze timing under different power and temperature ranges. There are four operating conditions available that represent the four "timing corners" in multi-corner timing analysis. The Timing Analyzer generates separate reports for each set of operating conditions.

- **Slow 900mV 100C Model**—specifies low voltage, high temperature operating conditions for timing analysis.
- **Slow 900mV 0C Model**—specifies low voltage, low temperature operating conditions for timing analysis.
- **Fast 900mV 100C Model**—specifies high voltage, high temperature operating conditions for timing analysis.
- **Fast 900mV 100C Model**—specifies high voltage, low temperature operating conditions for timing analysis.

Select a voltage/temperature combination and double-click **Report Timing** under **Custom Reports** in the **Tasks** pane to generate timing analysis reports for that model. After generating the report for that model, you can double-click the listings for the other models to generate analysis for those reports without re-generating the timing netlist.

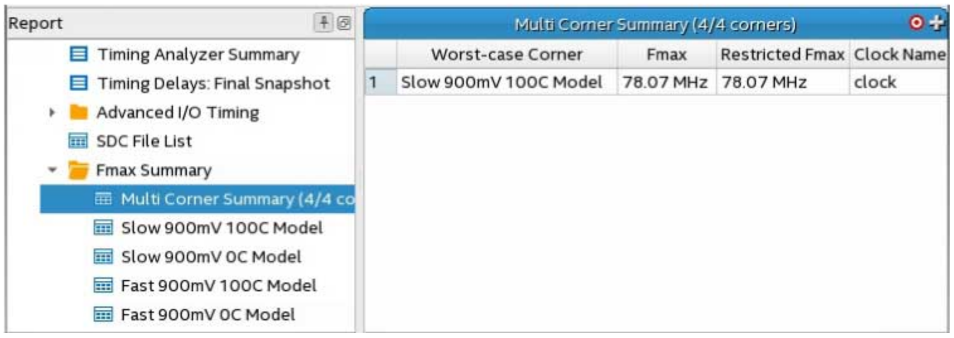
You can use the following context menu options to generate or regenerate reports in the **Report** window:

- **Regenerate**—Regenerates the report you select.
- **Generate in All Corners**—Generate a timing report using all four corners.
- **Regenerate All Out of Date**—Regenerate all reports.
- **Delete All Out of Date**—Removes all previous report data.

2.2.5.3. Fmax Summary Report

The Fmax Summary Report panel lists the maximum frequency of each clock in your design.

Figure 38. Fmax Summary Report



	Worst-case Corner	Fmax	Restricted Fmax	Clock Name
1	Slow 900mV 100C Model	78.07 MHz	78.07 MHz	clock

In some cases the Fmax Summary may indicate a "Limit due to hold check." Typically, hold checks do not limit the maximum frequency (f_{MAX}) because these checks are for same-edge relationships, and therefore independent of clock frequency. For example, when launch equals zero and latch equals zero.

However, if you have an inverted clock transfer, or a multicycle transfer (such as setup=2, hold=0), then the hold relationship is no longer a same-edge transfer and changes as the clock frequency changes.

The value in the **Restricted Fmax** column incorporates limits due to hold time checks, as well as minimum period and pulse width checks. If hold checks limit the f_{MAX} more than setup checks, that is indicated in the **Note** column as "Limit due to hold check."

2.2.5.4. Report Timing Command

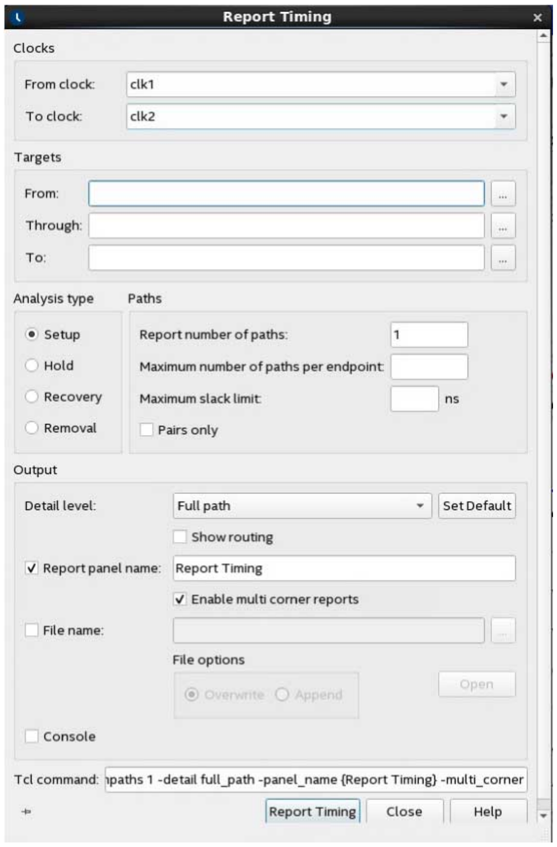
The **Report Timing** command allows you to specify options for reporting the timing on any path or clock domain in the design.

To access **Report Timing** in the Timing Analyzer:

- In the **Tasks** pane, click **Reports ➤ Custom Reports ➤ Report Timing**.
- Right-click on nodes or assignments, and then click **Report Timing**.

You can specify the **Clocks**, **Targets**, **Analysis Type**, and **Output** options that you want to include in the report. For example, you can increase the number of paths to report, add a **Target** filter, add a **From Clock**, or write the report to a text file.

Figure 39. Report Timing Dialog Box



From clock: clk1
To clock: clk2
Report number of paths: 1
Maximum number of paths per endpoint:
Maximum slack limit: ns
Pairs only
Detail level: Full path
Report panel name: Report Timing
Enable multi corner reports
File name:
Tcl command: ıpaths 1 -detail full_path -panel_name {Report Timing} -multi_corner

Table 6. Report Timing Options

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report paths with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. For example, to report only paths within a specific hierarchy: `report_timing -from *|egress:egress_inst|* \ -to *|egress:egress_inst|* -(other options)` When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report for paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**.
Output	The **Detail** level, allows you to specify the path types the analysis includes in output. **Summary** level includes basic summary reports. **Path only** displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. Review the **Clock Skew** column in the **Summary** report. If the skew is less than +/-150ps, the clock tree is well balanced between source and destination. When higher clock skew is present, enable the **Full path** option. This option breaks the clock tree into greater detail, showing every cell, including the input buffer, PLL, global buffer (called CLKCTRL_), and any logic. Review this data to determine the cause of clock skew in your design. Use the **Full path** option for I/O analysis because only the source clock or destination clock is inside the FPGA, and therefore the delay is a critical factor to meet timing.
Enable multi corner reports	Enables or disables multi-corner timing analysis. This option is on by default.
Report panel name	Displays the name of the report panel. You can enable **File name** to write the information to a file. If you append `.htm` as a suffix, the Timing Analyzer produces the report as HTML.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

2.2.5.5. Correlating Constraints to the Timing Report

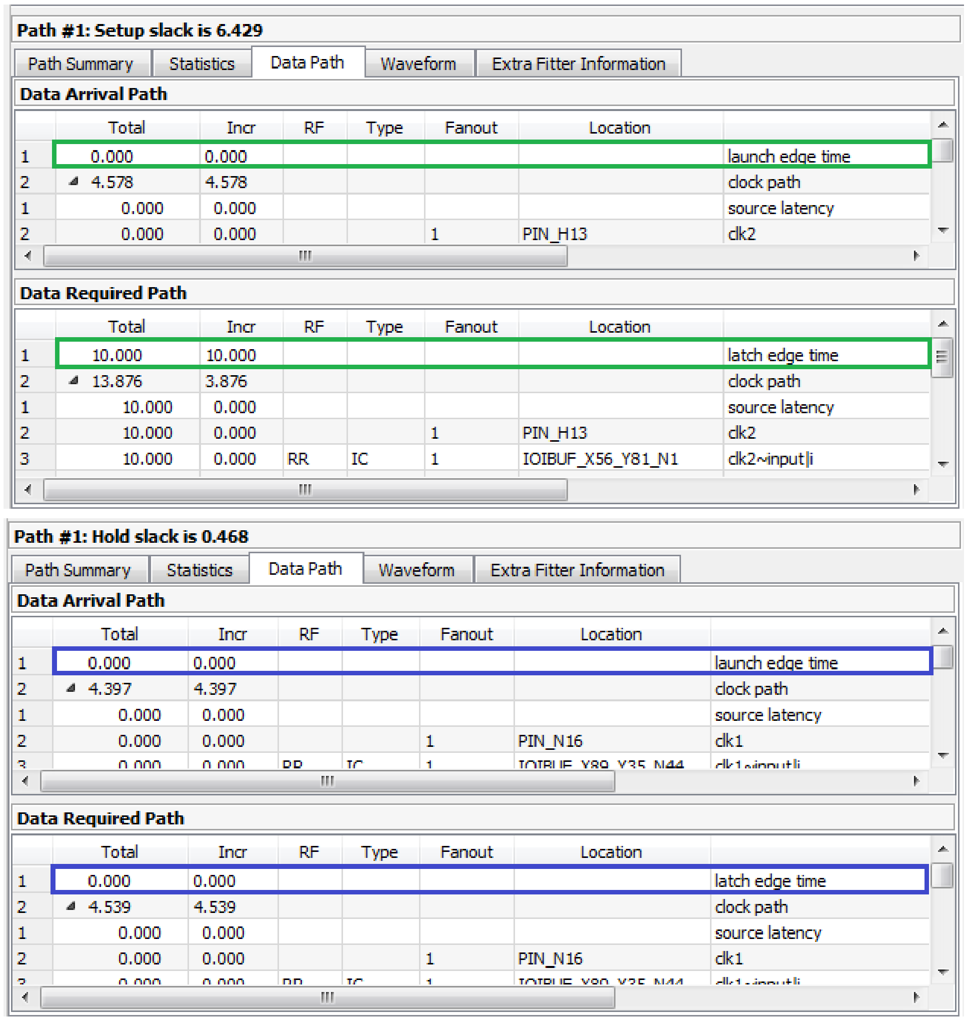
Understanding how timing constraints and violations appear in the timing analysis reports is critical to understanding the results. The following examples show how specific constraints impact the timing analysis reports. Most timing constraints only affect the clock launch and latch edges. Specifically, `create_clock` and `create_generated_clock` create clocks with default relationships. However, the `set_multicycle_path` exception modifies the default setup and hold relationship. The `set_max_delay` and `set_min_delay` constraints are low-level overrides that explicitly indicate the maximum and minimum delays for the launch and latch edges.

The following figures show the results of running **Report Timing** on a particular path.

In the following example, the design includes a clock driving the source and destination registers with a period of 10 ns. This results in a setup relationship of 10 ns (launch edge = 0 ns, latch edge = 10ns) and hold relationship of 0 ns (launch edge = 0 ns, latch edge = 0 ns) from the command:

```
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
```

Figure 40. Setup Relationship 10ns, Hold Relationship 0ns



Path #1: Setup slack is 6.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	10.000	10.000					latch edge time
2	◢ 13.876	3.876					clock path
1	10.000	0.000					source latency
2	10.000	0.000			1	PIN_H13	clk2
3	10.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

Path #1: Hold slack is 0.468

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.397	4.397					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_N16	clk1
3	0.000	0.000	RR	IC	1	IOIBUF_X89_Y35_N44	clk1~input|i

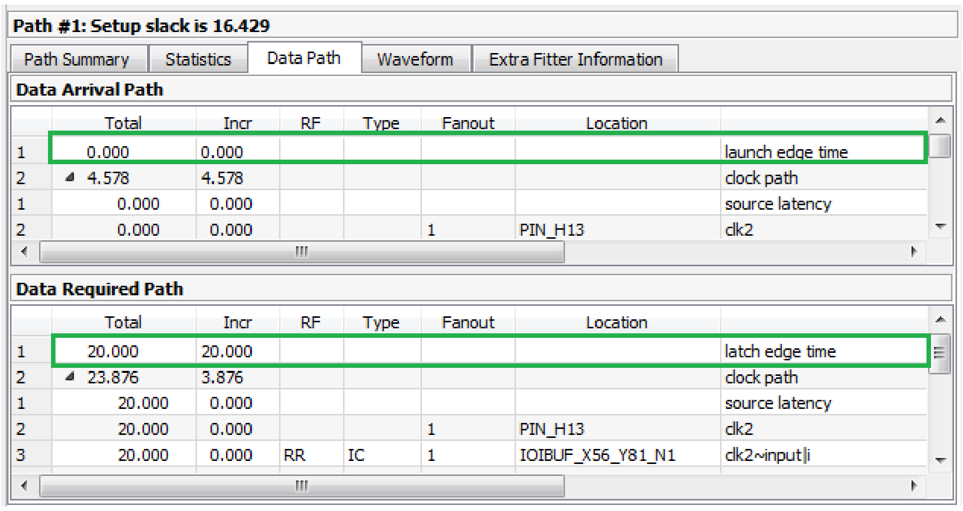
Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					latch edge time
2	◢ 4.539	4.539					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_N16	clk1
3	0.000	0.000	RR	IC	1	IOIBUF_X89_Y35_N44	clk1~input|i

The `set_multicycle_path` constraint adds multicycles to relax the setup relationship, or open the window, making the setup relationship 20 ns while the hold relationship is still 0 ns:

```
set_multicycle_path -from clocktwo -to clocktwo -setup -end 2
set_multicycle_path -from clocktwo -to clocktwo -hold -end 1
```

Figure 41. Setup Relationship 20ns



Path #1: Setup slack is 16.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

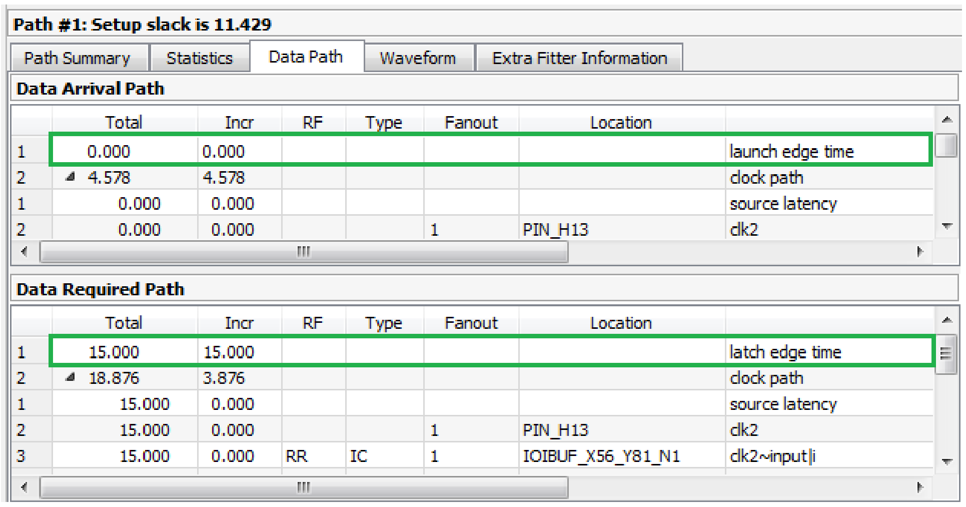
	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	20.000	20.000					latch edge time
2	◢ 23.876	3.876					clock path
1	20.000	0.000					source latency
2	20.000	0.000			1	PIN_H13	clk2
3	20.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

The `set_max_delay` and `set_min_delay` constraints explicitly override the setup relationship. Note that the only thing changing for these different constraints are the launch edge time and latch edge times for setup and hold analysis. Every other line item comes from delays inside the FPGA and are static for a given fit. View these reports to analyze how your constraints affect the timing reports.

Figure 42. Using set_max_delay



Path #1: Setup slack is 11.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

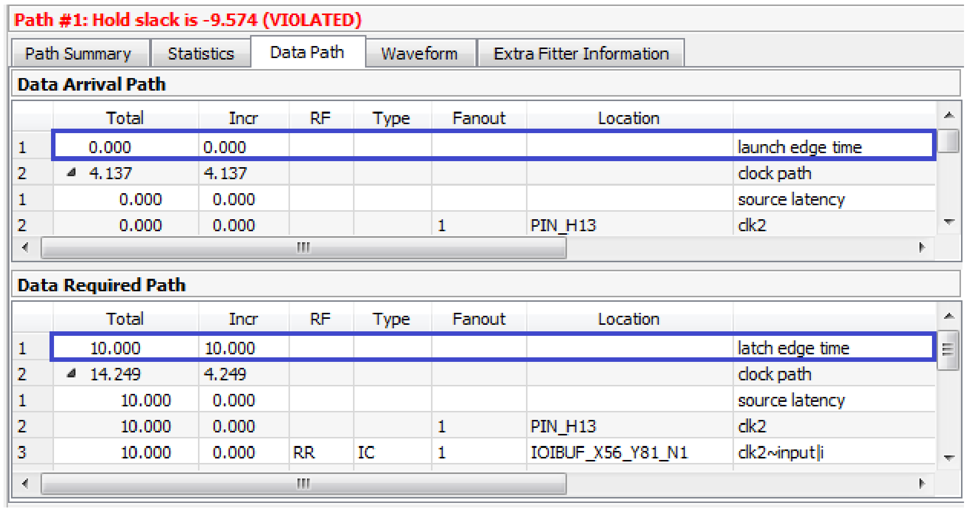
Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	15.000	15.000					latch edge time
2	◢ 18.876	3.876					clock path
1	15.000	0.000					source latency
2	15.000	0.000			1	PIN_H13	clk2
3	15.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

Figure 43. Using set_min_delay



Path #1: Hold slack is -9.574 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.137	4.137					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	10.000	10.000					latch edge time
2	◢ 14.249	4.249					clock path
1	10.000	0.000					source latency
2	10.000	0.000			1	PIN_H13	clk2
3	10.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

For I/O, you must add the `-max` and `-min` values. They are display as **iExt** or **oExt** in the **Type** column. An example is an output port with a `set_output_delay -max 1.0` and `set_output_delay -min -0.5`:

The clock relationships determine the launch and latch edge times, multicycles, and possibly `set_max_delay` or `set_min_delay` constraints. The Timing Analyzer also adds the value of `set_output_delay` as an **oExt** value. For outputs this value is part of the **Data Required Path**, since this is the external part of the analysis. The setup report on the left subtracts the `-max` value, making the setup relationship harder to meet, since the **Data Arrival Path** must be shorter than the **Data Required Path**. The Timing Analyzer also subtracts the `-min` value. This subtraction is why a negative number causes more restrictive hold timing. The **Data Arrival Path** must be longer than the **Data Required Path**.

Related Information

Relaxing Setup with Multicycle (set_multicycle_path) on page 59

2.2.5.6. Locating Timing Paths in Other Tools

You can locate from paths and elements in the Timing Analyzer to other tools in the Intel Quartus Prime software.

You can right-click most paths or node names in the Timing Analyzer GUI and click the **Locate** or **Locate Path** commands. Use these commands in the Timing Analyzer GUI or the `locate` command in the Tcl console to locate to that node in other Intel Quartus Prime tools.

The following examples show how to locate the ten paths with the worst timing slack from Timing Analyzer to the Technology Map Viewer, and locate all ports matching data* in the Chip Planner.

Example 2. Locating from the Timing Analyzer

```
# Locate in the Technology Map Viewer the ten paths with the worst slack
locate [get_timing_paths -npaths 10] -tmv
# locate all ports that begin with data in the Chip Planner
locate [get_ports data*] -chip
```

2.3. Using Timing Constraints

The following section describes correct application of SDC timing constraints that guide Fitter placement and allow accurate timing analysis. You can create an `.sdc` file with a set of initial recommended constraints, and then iteratively modify those constraints as the design progresses.

2.3.1. Recommended Initial SDC Constraints

Include the following basic SDC constraints in your initial `.sdc` file. The following example shows application of the recommended initial SDC constraints for a simple dual-clock design:

```
create_clock -period 20.00 -name adc_clk [get_ports adc_clk]
create_clock -period 8.00 -name sys_clk [get_ports sys_clk]

derive_pll_clocks

derive_clock_uncertainty
```

Create Clock (create_clock) on page 32

Derive PLL Clocks (derive_pll_clocks) on page 33

Derive Clock Uncertainty (derive_clock_uncertainty) on page 34

Set Clock Groups (set_clock_groups) on page 34

2.3.1.1. Create Clock (create_clock)

The **Create Clock** (`create_clock`) constraint allows you to define the properties and requirements for a clock in the design. You must define clock constraints to determine the performance of your design and constrain the external clocks coming into the FPGA. You can enter the constraints in the Timing Analyzer GUI, or in the `.sdc` file directly.

You specify the **Clock name** (`-name`), clock **Period** (`-period`), rising and falling **Waveform edge** values (`-waveform`), and the target signal(s) to which the constraints apply.

The following command creates the `sys_clk` clock with an 8ns period, and applies the clock to the `fpga_clk` port.:

```
create_clock -name sys_clk -period 8.0 \
     [get_ports fpga_clk]
```

Note: Tcl and `.sdc` files are case-sensitive. Ensure that references to pins, ports, or nodes match the case of names in your design.

By default, the clock has a rising edge at time 0 ns, a 50% duty cycle, and a falling edge at time 4 ns. If you require a different duty cycle, or to represent an offset, specify the `-waveform` option.

Typically you name a clock with the same name as the port you assign. In the example above, the following constraint accomplishes this:

`create_clock -name fpga_clk -period 8.0 [get_ports fpga_clk]`

There are now two unique objects called `fpga_clk`, a port in your design and a clock applied to that port.

Note: In Tcl syntax, square brackets execute the command inside them. `[get_ports fpga_clk]` executes a command that finds and returns a collection of all ports in the design that match `fpga_clk`. You can enter the command without using the `get_ports` collection command, as shown in the following example:

```
create_clock -name sys_clk -period 8.0 fpga_clk
```

Warning: Constraints that you define in the Timing Analyzer apply directly to the timing database, but do not automatically transfer to the `.sdc` file. Click **Write SDC File** on the Timing Analyzer **Tasks** pane to preserve constraints changes from the GUI in an `.sdc` file.

Related Information

Creating Base Clocks on page 37

2.3.1.2. Derive PLL Clocks (derive_pll_clocks)

The **Derive PLL Clocks** (`derive_pll_clocks`) constraint automatically creates clocks for each output of any PLL in your design.

The constraint can generate multiple clocks for each output clock pin if the PLL is using clock switchover: one clock for the `inclk[0]` input clock pin, and one clock for the `inclk[1]` input clock pin. Specify the **Create base clocks** (`-create_base_clocks`) option to create base clocks on the inputs of the PLLs by default. By default the clock name is the same as the output clock pin name, or specify the **Use net name as clock name** (`-use_net_name`) option to use the net name.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_pll_clocks
```

When you create PLLs, you must define the configuration of each PLL output. This definition allows the Timing Analyzer to automatically constrain the PLLs with the `derive_pll_clocks` command. This command also constrains transceiver clocks and adds multicycles between LVDS SERDES and user logic.

The `derive_pll_clocks` command prints an Info message to show each generated clock the command creates.

As an alternative to `derive_pll_clocks` you can copy-and-paste each `create_generated_clock` assignment into the `.sdc` file. However, if you subsequently modify the PLL setting, you must also change the generated clock constraint in the `.sdc` file. Examples of this type of change include modifying an existing output clock, adding a new PLL output, or making a change to the PLL's hierarchy. Use of `derive_pll_clocks` eliminates this requirement.

Related Information

- Creating Base Clocks on page 37
- Deriving PLL Clocks on page 43

2.3.1.3. Derive Clock Uncertainty (derive_clock_uncertainty)

The **Derive Clock Uncertainty** (`derive_clock_uncertainty`) constraint applies setup and hold clock uncertainty for clock-to-clock transfers in the design. This uncertainty represents characteristics like PLL jitter, clock tree jitter, and other factors of uncertainty.

You can enable the **Add clock uncertainty assignment** (`-add`) to add clock uncertainty values from any **Set Clock Uncertainty** (`set_clock_uncertainty`) constraint. You can **Overwrite existing clock uncertainty assignments** (`-overwrite`) any `set_clock_uncertainty` constraints.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_clock_uncertainty -add - overwrite
```

The Timing Analyzer generates a warning if you omit `derive_clock_uncertainty` from the `.sdc` file.

Related Information

Accounting for Clock Effect Characteristics on page 48

2.3.1.4. Set Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you specify which clocks in the design are unrelated. By default, the Timing Analyzer assumes that all clocks with a common base or parent clock are related, and that all transfers between those clock domains are valid for timing analysis. You can exclude transfers between specific clock domains from timing analysis by cutting clock groups.

Conversely, clocks without a common parent or base clock are always unrelated, but timing analysis includes the transfers between such clocks, unless those clocks are in different clock groups (or if all of their paths are cut with false path constraints).

You define groups of clock signals, and then define the relationship between the each group. You define the clock signals to include in each Group (-group), and then specify whether the groups are **Logically exclusive** (`-logically_exclusive`), **Physically exclusive** (`-physically_exclusive`, or **Asynchronous** (`-asynchronous`) from one another.

```
set_clock_groups -asynchronous -group {<clock1>...<clockn>} ... \
    -group {<clocka>...<clockn>}
```

- `-logically_exclusive`—defines clocks that are logically exclusive and not active at the same time, such as multiplexed clocks
- `-physically_exclusive`—defines clocks that are physically exclusive not active at the same time.
- The `-asynchronous`—defines completely unrelated clocks that have different ideal clock sources. flag means the clocks are both switching, but not in a way that can synchronously pass data.

For example, if there are paths between an 8ns clock and 10ns clock, even if the clocks are completely asynchronous, the Timing Analyzer attempts to meet a 2ns setup relationship between these clocks, unless you specify that they are not related.

Although the **Set Clock Groups** dialog box only permits two clock groups, you can specify an unlimited number of `-group {<`*`group of clocks`*`>}` options in the `.sdc` file. If you omit an unrelated clock from the assignment, the Timing Analyzer acts conservatively and analyzes that clock in context with all other domains to which the clock connects.

The Timing Analyzer does not currently analyze crosstalk explicitly. Instead, the timing models use extra guard bands to account for any potential crosstalk-induced delays. The Timing Analyzer treats the `-asynchronous` and `-exclusive` options the same.

A clock cannot be within multiple groups (`-group`) in a single assignment; however, you can have multiple `set_clock_groups` assignments.

Another way to cut timing between clocks is to use `set_false_path`. To cut timing between `sys_clk` and `dsp_clk`, you can use:

```
set_false_path -from [get_clocks sys_clk] -to [get_clocks dsp_clk]
```

```
set_false_path -from [get_clocks dsp_clk] -to [sys_clk]
```

This technique is effective if there are only a few clocks, but can become unmanageable with a large number of constraints. In a simple design with three PLLs that have multiple outputs, the `set_clock_groups` command can show which clocks are related in less than ten lines, while using `set_false_path` commands can use more than 50 lines.

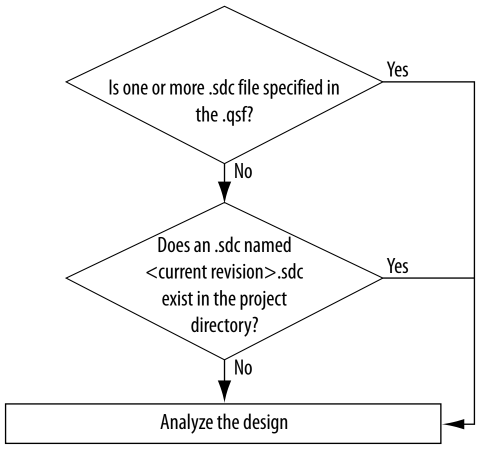
Related Information

- Creating Generated Clocks (create_generated_clock) on page 40
- Relaxing Setup with Multicycle (set_multicycle_path) on page 59
- Accounting for a Phase Shift (-phase) on page 60

2.3.2. SDC File Precedence

You must add any `.sdc` file that you create to the project to be read during fitting and timing analysis. The Fitter and the Timing Analyzer process `.sdc` files in the order they appear in the `.qsf`. If no `.sdc` appears in the `.qsf`, the Intel Quartus Prime software searches for an `.sdc` with the name `<`*`current revision`*`>.sdc` in the project directory.

Figure 44. .sdc File Order of Precedence


Is one or more .sdc file specified in the .qsf?
Yes
No
Does an .sdc named <current revision>.sdc exist in the project directory?
Yes
No
Analyze the design

Click **Settings ➤ Timing Analyzer** to add, remove, or change the processing order of `.sdc` files in the project, as Step 3: Specify General Timing Analyzer Settings on page 22 describes.

If you use the Intel Quartus Prime Text Editor to create an `.sdc` file, the option to **Add file to the project** enables by default when you save the file. If you use any other editor to create an `.sdc` file, you must add the file to the project.

The `.sdc` file must contain only timing constraint commands. Tcl commands to manipulate the timing netlist or control the compilation must be in a separate Tcl script.

When you use IP from Intel, and some third-parties, the `.sdc` files become part of the project through an intermediate Intel Quartus Prime IP File (`.qip`). The `.qip` file references all source and constraints files for the IP. If `.sdc` files for IP blocks in your design are included through with a `.qip`, do not re-add them manually. An `.sdc` file can also be added from a Intel Quartus Prime IP File (`.qip`) included in the `.qsf`.

Note: If you type the `read_sdc` command at the command line without any arguments, the Timing Analyzer reads constraints embedded in HDL files, then follows the `.sdc` file precedence order.

2.3.3. Iterative Constraint Modification

You can iteratively modify `.sdc` constraints and reanalyze the timing results to ensure that you have the optimum constraints for your design.

Use the following steps to iteratively modify constraints:

1. Click **Tools ➤ Timing Analyzer**.
2. Generate the reports you want to analyze. Double-click **Report All Summaries** under **Macros** to generate setup, hold, recovery, and removal summaries, as well as minimum pulse width checks, and a list of all the clock you define. These

summaries cover all paths you constrain in your design. Whenever modifying or correcting constraints, generate the **Diagnostic** reports to identify unconstrained parts of your design, or ignored constraints.

3. Analyze the results in the reports. When you are modifying constraints, rerun the reports to find any unexpected results. For example, a cross-domain path might indicate that you forgot to cut a transfer by including a clock in a clock group.
4. Create or edit the appropriate constraints in your `.sdc` file and save the file.
5. Double-click **Reset Design** in the **Tasks** pane. This removes all constraints from your design. Removing all constraints from your design allows rereading the `.sdc` files, including your changes.
6. Regenerate the reports you want to analyze.
7. Reanalyze the results.
8. Repeat steps 4-7 as necessary.

This method performs timing analysis using new constraints, without any change to logic placement. While the Fitter uses the original constraints for place and route, the Timing Analyzer applies the new constraints. If there is any failing timing against the new constraints, this indicates a need to run place-and-route again.

Related Information

Relaxing Setup with Multicycle (set_multicyle_path) on page 59

2.3.4. Creating Clocks and Clock Constraints

You must define all clocks and any associated clock characteristics, such as uncertainty, latency or skew. The Timing Analyzer supports `.sdc` commands that accommodate various clocking schemes, such as:

- Base clocks
- Virtual clocks
- Multifrequency clocks
- Generated clocks

2.3.4.1. Creating Base Clocks

Base clocks are the primary input clocks to the device. The **Create Clock** (`create_clock`) constraint allows you to define the properties and requirements for clocks in the design. Unlike clocks that are generated in the device (such as an on-chip PLL), base clocks are generated by off-chip oscillators or forwarded from an external device. Define base clocks at the top of your `.sdc` file, because generated clocks and other constraints often reference base clocks. The Timing Analyzer ignores any constraints that reference an undefined clock.

The following examples show common use of the `create_clock` constraint:

create_clock Command

The following specifies a 100 MHz requirement on a `clk_sys` input clock port:

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

100 MHz Shifted by 90 Degrees Clock Creation

The following creates a 10 ns clock, with a 50% duty cycle, that is phase shifted by 90 degrees, and applies to port `clk_sys`. This type of clock definition commonly refers to source synchronous, double-rate data that is center-aligned with respect to the clock.

```
create_clock -period 10 -waveform { 2.5 7.5 } [get_ports clk_sys]
```

Two Oscillators Driving the Same Clock Port

You can apply multiple clocks to the same target with the `-add` option. For example, to specify that you can drive the same clock input at two different frequencies, enter the following commands in your `.sdc` file:

```
create_clock -period 10 -name clk_100 [get_ports clk_sys]
create_clock -period 5 -name clk_200 [get_ports clk_sys] -add
```

Although uncommon to define more than two base clocks for a port, you can define as many as are appropriate for your design, making sure you specify `-add` for all clocks after the first.

Creating Multifrequency Clocks

You must create a multifrequency clock if your design has more than one clock source feeding a single clock node. The additional clock may act as a low-power clock, with a lower frequency than the primary clock. If your design uses multifrequency clocks, use the `set_clock_groups` command to define clocks that are exclusive.

Use the `create_clock` command with the `-add` option to create multiple clocks on a clock node. You can create a 10 ns clock applied to clock port `clk`, and then add an additional 15 ns clock to the same clock port. The Timing Analyzer analyzes both clocks.

```
create_clock –period 10 –name clock_primary –waveform { 0 5 } \
    [get_ports clk]
create_clock –period 15 –name clock_secondary –waveform { 0 7.5 } \
    [get_ports clk] -add
```

Related Information

Accounting for Clock Effect Characteristics on page 48

2.3.4.1.1. Automatic Clock Detection and Constraint Creation

Use the `derive_clocks` command to automatically create base clocks in your design The `derive_clocks` command is equivalent to using the `create_clock` command for each register or port feeding the clock pin of a register. The `derive_clocks` command creates clock constraints on ports or registers to ensure every register in your design has a clock constraints, and it applies one period to all base clocks in your design.

The following command specifies a base clock with a 100 MHz requirement for unconstrained base clock nodes.

```
derive_clocks -period 10
```

Warning: Do not use the `derive_clocks` command for final timing sign-off; instead, you create clocks for all clock sources with the `create_clock` and `create_generated_clock` commands. If your design has more than a single clock, the `derive_clocks` command constrains all the clocks to the same specified frequency. To achieve a thorough and realistic analysis of your design's timing requirements, make individual clock constraints for all clocks in your design.

If you want to create some base clocks automatically, use the `-create_base_clocks` option to `derive_pll_clocks`. With this option, the `derive_pll_clocks` command automatically creates base clocks for each PLL, based on the input frequency information that you specify when you generate the PLL. This feature works for simple port-to-PLL connections. Base clocks do not automatically generate for complex PLL connectivity, such as cascaded PLLs. You can also use the command `derive_pll_clocks -create_base_clocks` to create the input clocks for all PLL inputs automatically.

2.3.4.2. Creating Virtual Clocks

A virtual clock is a clock without a real source in the design, or a clock that does not interact directly with the design. You can use Virtual clocks in I/O constraints to represent the clock at the external device connected to the FPGA.

To create virtual clocks, use the `create_clock` constraint with no value for the *<targets>* option.

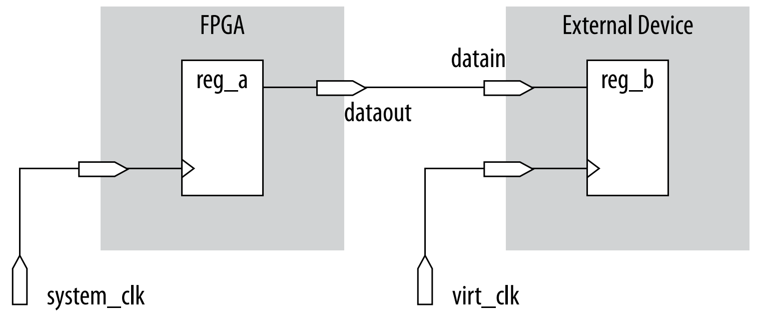
This following example defines a 100Mhz virtual clock because the command includes no *<targets>*.

```
create_clock -period 10 -name my_virt_clk
```

I/O Constraints with Virtual Clocks

For the output circuit shown in the following figure, you can use a base clock to constrain the circuit in the FPGA, and a virtual clock to represent the clock driving the external device. The following figure shows the base clock (`system_clk`), virtual clock (`virt_clk`), and output delay for the Virtual Clock Constraints example below.

Figure 45. Virtual Clock Board Topology



The following creates the 10 ns `virt_clk` virtual clock, with a 50% duty cycle, with the first rising edge occurring at 0 ns. The virtual clock can then become the clock source for an output delay constraint.

Example 3. Virtual Clock Constraints

```
#create base clock for the design
create_clock -period 5 [get_ports system_clk]
#create the virtual clock for the external register
create_clock -period 10 -name virt_clk
#set the output delay referencing the virtual clock
set_output_delay -clock virt_clk -max 1.5 [get_ports dataout]
set_output_delay -clock virt_clk -min 0.0 [get_ports dataout]
```

2.3.4.2.1. Specifying I/O Interface Uncertainty

Virtual clocks are recommended for I/O constraints because they most accurately represent the clocking topology of the design. An additional benefit is that you can specify different uncertainty values on clocks that interface with external I/O ports and clocks that feed register-to-register paths inside the FPGA.

2.3.4.2.2. I/O Interface Clock Uncertainty Example

To specify I/O interface uncertainty, you must create a virtual clock and constrain the input and output ports with the `set_input_delay` and `set_output_delay` commands that reference the virtual clock.

When the `set_input_delay` or `set_output_delay` commands reference a clock port or PLL output, the virtual clock allows the `derive_clock_uncertainty` command to apply separate clock uncertainties for internal clock transfers and I/O interface clock transfers

Create the virtual clock with the same properties as the original clock that is driving the I/O port, as the following example shows:

Example 4. SDC Commands to Constrain the I/O Interface

```
# Create the base clock for the clock port
create_clock -period 10 -name clk_in [get_ports clk_in]
# Create a virtual clock with the same properties of the base clock
# driving the source register
create_clock -period 10 -name virt_clk_in
# Create the input delay referencing the virtual clock and not the base
# clock
# DO NOT use set_input_delay -clock clk_in <delay value>
# [get_ports data_in]
set_input_delay -clock virt_clk_in <delay value> [get_ports data_in]
```

2.3.4.3. Creating Generated Clocks (create_generated_clock)

The **Create Generate Clock** (`create_generated_clock`) constraint allows you to define the properties and constraints of an internally generated clock in the design. You specify the **Clock name** (`-name`), the **Source** node (`-source`) from which clock derives, and the **Relationship to the source** properties. Define generated clocks for any node that modifies the properties of a clock signal, including modifying the phase, frequency, offset, or duty cycle.

You apply generated clocks most commonly on the outputs of PLLs, on register clock dividers, clock muxes, and clocks forwarded to other devices from an FPGA output port, such as source synchronous and memory interfaces. In the `.sdc` file, enter generated clocks after the base clocks definitions. Generated clocks automatically account for all clock delays and clock latency to the generated clock target.

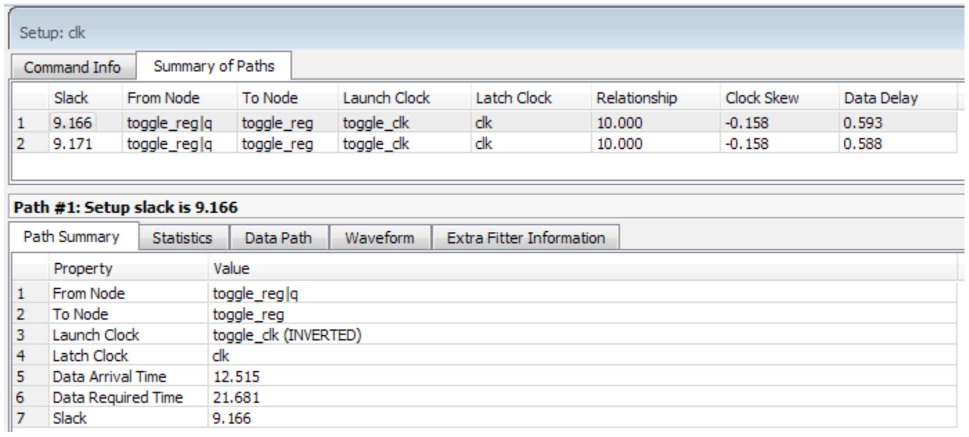
The `-source` option specifies the name of a node in the clock path that you use as reference for your generated clock. The source of the generated clock must be a node in your design netlist, and not the name of a clock you previously define. You can use any node name on the clock path between the input clock pin of the target of the generated clock and the target node of its reference clock as the source node.

Specify the input clock pin of the target node as the source of your new generated clock. The source of the generated clock decouples from the naming and hierarchy of the clock source. If you change the clock source, you do not have to edit the generated clock constraint.

If you have multiple base clocks feeding a node that is the source for a generated clock, you must define multiple generated clocks. You associate each generated clock with one base clock using the `-master_clock` option in each generated clock statement. In some cases, generated clocks generate with combinational logic.

Depending on how your clock-modifying logic synthesizes, the signal name can change from one compilation to the next. If the name changes after you write the generated clock constraint, the Compiler ignores the generated clock because that target name no longer exists in the design. To avoid this problem, use a synthesis attribute or synthesis assignment to retain the final combinational node name of the clock-modifying logic. Then use the kept name in your generated clock constraint.

Figure 46. Example of clock-as-data



	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	9.166	toggle_reg|q	toggle_reg	toggle_clk	clk	10.000	-0.158	0.593
2	9.171	toggle_reg|q	toggle_reg	toggle_clk	clk	10.000	-0.158	0.588

Path #1: Setup slack is 9.166

	Property	Value
1	From Node	toggle_reg|q
2	To Node	toggle_reg
3	Launch Clock	toggle_clk (INVERTED)
4	Latch Clock	clk
5	Data Arrival Time	12.515
6	Data Required Time	21.681
7	Slack	9.166

When you create a generated clock on a node that ultimately feeds the data input of a register, this creates a special case of "clock-as-data." The Timing Analyzer treats clock-as-data differently. For example, if you use clock-as-data with DDR, you must consider both the rise and the fall of this clock, and the Timing Analyzer reports both rise and fall. With clock-as-data, the Compiler treats the **From Node** as the target of the generated clock, and the **Launch Clock** as the generated clock.

In Example of Clock as Data, the first path is from `toggle_clk (INVERTED)` to `clk`, and the second path is from `toggle_clk` to `clk`. The slack in both cases is slightly different due to the difference in rise and fall times along the path. The **Data Delay** column reports the ~5 ps difference. Only the path with the lowest slack value requires consideration. The Timing Analyzer only reports the worst-case path between the two (rise and fall). In this example, if you do not define the generated clock on the register output, then timing analysis reports only one path with the lowest slack value.

You can use the `derive_pll_clocks` command to automatically generate clocks for all PLL clock outputs. The properties of the generated clocks on the PLL outputs match the properties you define for the PLL.

Related Information

Deriving PLL Clocks on page 43

2.3.4.3.1. Clock Divider Example (-divide_by)

A common form of generated clock is the divide-by-two register clock divider. The following example constraint creates a half-rate clock on the divide-by-two register.

```
create_clock -period 10 -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_ports clk_sys] [get_pins reg|q]
```

To specify the clock pin of the register as the clock source:

```
create_clock -period 10 -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_pins reg|clk] [get_pins reg|q]
```

Figure 47. Clock Divider

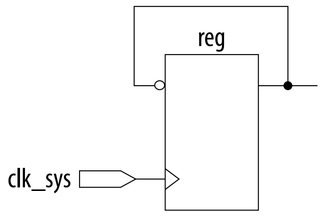
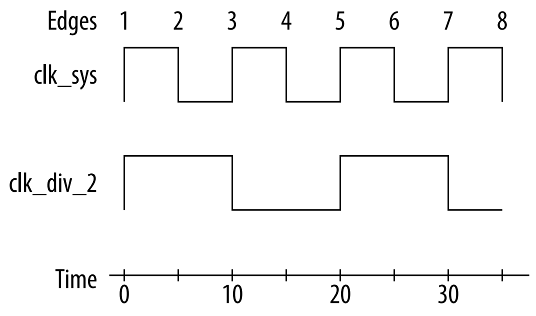


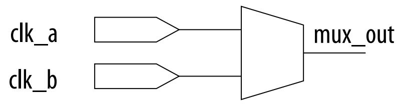
Figure 48. Clock Divider Waveform



2.3.4.3.2. Clock Multiplexor Example

The output of a clock multiplexor (mux) is a form of generated clock. Each input clock requires one generated clock on the output. The following `.sdc` example also includes the `set_clock_groups` command to indicate that the two generated clocks can never be active simultaneously in the design. Therefore, the Timing Analyzer does not analyze cross-domain paths between the generated clocks on the output of the clock mux.

Figure 49. Clock Mux



```
create_clock -name clock_a -period 10 [get_ports clk_a]
create_clock -name clock_b -period 10 [get_ports clk_b]
create_generated_clock -name clock_a_mux -source [get_ports clk_a] \
    [get_pins clk_mux|mux_out]
create_generated_clock -name clock_b_mux -source [get_ports clk_b] \
    [get_pins clk_mux|mux_out] -add
set_clock_groups -exclusive -group clock_a_mux -group clock_b_mux
```

2.3.4.4. Deriving PLL Clocks

The **Derive PLL Clocks** (`derive_pll_clocks`) constraint automatically creates clocks for each output of any PLL in your design. `derive_pll_clocks` detects your current PLL settings and automatically creates generated clocks on the outputs of every PLL by calling the `create_generated_clock` command.

Create Base Clock for PLL input Clock Ports

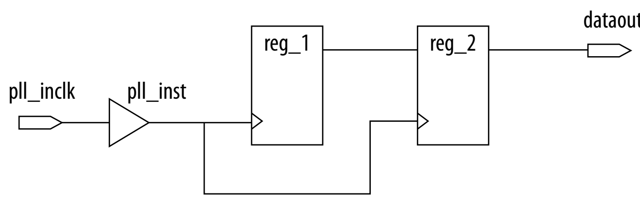
If your design contains transceivers, LVDS transmitters, or LVDS receivers, use the `derive_pll_clocks` to constrain this logic in your design and create timing exceptions for those blocks.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_pll_clocks
```

Include the `derive_pll_clocks` command in your `.sdc` file after any `create_clock` command. Each time the Timing Analyzer reads the `.sdc` file, the appropriate generated clock is created for each PLL output clock pin. If a clock exists on a PLL output before running `derive_pll_clocks`, the pre-existing clock has precedence, and an auto-generated clock is not created for that PLL output.

The following shows a simple PLL design with a register-to-register path:

Figure 50. Simple PLL Design



The Timing Analyzer generates messages like the following example when you use the `derive_pll_clocks` command to constrain the PLL.

Example 5. derive_pll_clocks Command Messages

```
Info:
Info: Deriving PLL Clocks:
Info: create_generated_clock -source pll_inst|altpll_component|pll|inclk[0] -
divide_by 2 -name
pll_inst|altpll_component|pll|clk[0] pll_inst|altpll_component|pll|clk[0]
Info:
```

The input clock pin of the PLL is the node `pll_inst|altpll_component|pll|inclk[0]` which is the `-source` option. The name of the output clock of the PLL is the PLL output clock node, `pll_inst|altpll_component|pll|clk[0]`.

If the PLL is in clock switchover mode, multiple clocks generate for the output clock of the PLL; one for the primary input clock (for example, `inclk[0]`), and one for the secondary input clock (for example, `inclk[1]`). Create exclusive clock groups for the primary and secondary output clocks since they are not active simultaneously.

Related Information

Creating Clock Groups (set_clock_groups) on page 44

2.3.4.5. Creating Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you specify which clocks in the design are unrelated. By default, the Timing Analyzer assumes that all clocks with a common base or parent clock are related, and that all transfers between those clock domains are valid for timing analysis. You can exclude transfers between specific clock domains from timing analysis by cutting clock groups.

The `set_clock_groups` command allows you to cut timing between unrelated clocks in different groups. The Timing Analyzer performs the same analysis regardless of whether you specify `-exclusive` or `-asynchronous` groups. You define a group with the `-group` option. The Timing Analyzer excludes the timing paths between clocks for each of the separate groups.

The following tables show the impact of `set_clock_groups`.

Table 7. set_clock_groups -group A

Dest\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut
B	Cut	Analyzed	Analyzed	Analyzed
C	Cut	Analyzed	Analyzed	Analyzed
D	Cut	Analyzed	Analyzed	Analyzed

Table 8. set_clock_groups -group {A B}

Dest\Source	A	B	C	D
A	Analyzed	Analyzed	Cut	Cut
B	Analyzed	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 9. set_clock_groups -group A -group B

Dest\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut

continued...

B	Cut	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 10. set_clock_groups -group {A C} -group {B D}

Dest\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Cut
B	Cut	Analyzed	Cut	Analyzed
C	Analyzed	Cut	Analyzed	Cut
D	Cut	Analyzed	Cut	Analyzed

Table 11. set_clock_groups -group {A C D}

Dest\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Analyzed
B	Cut	Analyzed	Cut	Cut
C	Analyzed	Cut	Analyzed	Analyzed
D	Analyzed	Cut	Analyzed	Analyzed

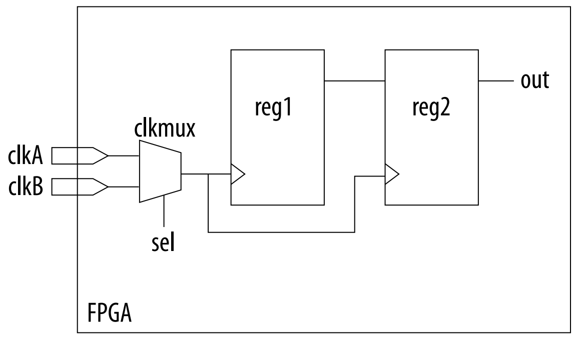
2.3.4.5.1. Exclusive Clock Groups (-exclusive)

You can use the `-exclusive` option to declare that two clocks are mutually exclusive.

If you define multiple clocks for the same node, you can use clock group assignments with the `-exclusive` option to declare clocks as mutually exclusive. This technique can be useful for multiplexed clocks.

For example, consider an input port that is clocked by either a 100-MHz or 125-MHz clock. You can use the `-exclusive` option to declare that the clocks are mutually exclusive and eliminate clock transfers between the 100-MHz and 125-MHz clocks, as the following diagrams and example SDC constraints illustrate:

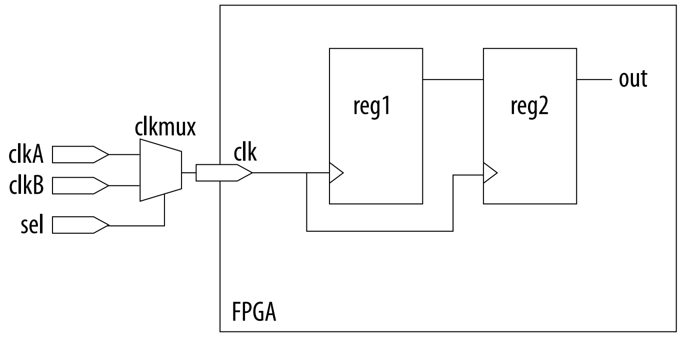
Figure 51. Synchronous Path with Clock Mux Internal to FPGA



Example SDC Constraints for Internal Clock Mux

```
# Create a clock on each port
create_clock -name clk_100 -period 10 [get_ports clkA]
create_clock -name clk_125 -period 8 [get_ports clkB]
# Set the two clocks as exclusive clocks
set_clock_groups -exclusive -group {clk_100} -group {clk_125}
```

Figure 52. Synchronous Path with Clock Mux External to FPGA



Example SDC Constraints for External Clock Mux

```
# Create two clocks on the port clk
create_clock -name clkA -period 10 [get_ports clk]
create_clock -name clkB -period 8 [get_ports clk] -add
# Set the two clocks as exclusive clocks
set_clock_groups -exclusive -group {clkA} -group {clkB}
```

2.3.4.5.2. Asynchronous Clock Groups (-asynchronous)

Use the `-asynchronous` option to create asynchronous clock groups. You can use asynchronous clock groups to break the timing relationship when data transfers through a FIFO between clocks running at different rates.

2.3.4.5.3. set_clock_groups Constraint Tips

When you use `derive_pll_clocks` to create clocks, it can be time consuming to determine all the clock names to include in `set_clock_groups` constraints. However, you can use the following technique to somewhat automate clock constraint creation, even if you do not know all of the clock names.

1. Create a basic `.sdc` file that contains the Recommended Initial SDC Constraints on page 32, except omit the `set_clock_groups` constraint for now.
2. To add the `.sdc` to the project, click **Assignments ➤ Settings ➤ Timing Analyzer**. Specify the `.sdc` file under **SDC files to include in the project**.
3. To open the Timing Analyzer, click **Tools ➤ Timing Analyzer**.
4. In the **Task** pane, double-click **Report Clocks**. The Timing Analyzer reads your `.sdc`, applies the constraints (including `derive_pll_clocks`), and reports all the clocks.
5. From the Clocks Summary report, copy all the clock names that appear in the first column. The report lists the clock names in the correct format for recognition in the Timing Analyzer.

6. Open .sdc file and the paste the clock names into the file, one clock name per line.
7. Format the list of clock names list into the `set_clock_groups` command by cutting and pasting clock names into appropriate groups. Next, paste the following template into the `.sdc` file:

```
set_clock_groups -asynchronous -group { \
} \
 -group { \
} \
-group  { \
} \
-group { \
}
```

8. Cut and paste the clock names into groups to define their relationship, adding or removing groups as necessary. Format the groups to make the code readable.

   Note: This command can be difficult to read on a single line. You can use the Tcl line continuation character "\" to make this more readable. Place a space after the last character, and then place the "\" character at the end of the line. This characters escapes, Be careful not to include any spaces after the escape character, otherwise the space becomes the escape character, rather than the end-of-line character).

```
set_clock_groups -asynchronous \
    -group {adc_clk \
       the_adc_pll|altpll_component_autogenerated|pll|clk[0] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[1] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[2] \
    } \
    -group {sys_clk \
       the_system_pll|altpll_component_autogenerated|pll|clk[0] \
       the_system_pll|altpll_component_autogenerated|pll|clk[1] \
    } \
    -group {the_system_pll|altpll_component_autogenerated|pll|clk[2] \
    }
```

Note: The last group has a PLL output `system_pll|..|clk[2]` while the input clock and other PLL outputs are in different groups. If you use PLLs, and the input clock frequency does not relate to the frequency of the PLL's outputs, you must treat the PLLs asynchronously. Usually most outputs of a PLL relate and are in the same group, but this is not a requirement.

For designs with complex clocking, creating clock groups can be an iterative process. For example, a design with two DDR3 cores and high-speed transceivers can have thirty or more clocks. In such cases, you start by adding the clocks that you manually create. Since the Timing Analyzer assumes that the clocks not appearing in the command relate to every clock, this conservatively groups the known clocks. If there are still failing paths in the design between unrelated clock domains, you can start add the new clock domains as necessary. In this case, a large number of the clocks are not in the `set_clock_groups` command, since they are either cut in the `.sdc` file for the IP core (such as the `.sdc` files that the DDR3 cores generate), or they connect only to related clock domains.

For many designs, that is all that's necessary to constrain the core. Some common core constraints that this section does not describe in detail are:

- Adding multicycles between registers for analysis at a slower rate than the default analysis, increasing the time when data can be read. For example, a 10 ns clock period has a 10 ns setup relationship. If the data changes at a slower rate, or perhaps the registers switch at a slower rate due to a clock enable, then you can apply a multicycle that relaxes the setup relationship (opens the window so that valid data can pass). This is a multiple of the clock period, making the setup relationship 20 ns, 40 ns, and so on, while keeping the hold relationship at 0 ns. You generally apply these types of multicycles to paths.
- You can also use multicycle when you want to advance the cycle in which data is read, shifting the timing window. This generally occurs when your design performs a small phase-shift on a clock. For example, if your design has two 10 ns clocks exiting a PLL, but the second clock has a 0.5 ns phase-shift, the default setup relationship from the main clock to the phase-shift clock is 0.5 ns and the hold relationship is -9.5 ns. Meeting a 0.5 ns setup relationship is nearly impossible, and most likely you intend the data to transfer in the next window. By adding a multicycle from the main clock to the phase-shift clock, the setup relationship becomes 10.5 ns and the hold relationship becomes 0.5 ns. You generally apply this multicycle between clocks.
- Add a `create_generated_clock` to ripple clocks. When a register's output drives the `clk` port of another register, that is a ripple clock. Clocks do not propagate through registers, so you must apply the `create_generated_clock` constraint to all ripple clocks for correct analysis. Unconstrained ripple clocks appear in the **Report Unconstrained Paths** report, so you can easily recognize them. In general, avoid ripple clocks. Use a clock enable instead.
- Add a `create_generated_clock` to clock mux outputs. Without this clock, all clocks propagate through the mux and are related. The Timing Analyzer analyzes paths downstream from the mux where one clock input feeds the source register and the other clock input feeds the destination, and vice-versa. Although this behavior can be valid, this is typically not the behavior you want. By applying `create_generated_clock` constraints on the mux output, which relates them to the clocks coming into the mux, you can correctly group these clocks with other clocks.

2.3.4.6. Accounting for Clock Effect Characteristics

The clocks you create with the Timing Analyzer are ideal clocks that do not account for any board effects. You can account for clock effect characteristics with clock latency and clock uncertainty constraints.

2.3.4.6.1. Set Clock Latency (set_clock_latency)

The **Set Clock Latency** (`set_clock_latency`) constraint allows you to specify additional delay (that is, latency) in a clock network. This delay value represents the external delay from a virtual (or ideal) clock through the longest **Late** (`-late`) or shortest **Early** (`-early`) path, with reference to the **Rise** (`-rise`) or **Fall** (`-fall`) of the clock transition.

The Timing Analyzer uses the late clock latency for the data arrival path, and the early clock latency for the clock arrival path, when calculating setup analysis. The Timing Analyzer uses the early clock latency for the data arrival time, and the late clock latency for the clock arrival time, for hold analysis.

There are two forms of clock latency: clock source latency, and clock network latency. Source latency is the propagation delay from the origin of the clock to the clock definition point (for example, a clock port). Network latency is the propagation delay from a clock definition point to a register's clock pin. The total latency at a register's clock pin is the sum of the source and network latencies in the clock path.

To specify source latency to any clock ports in your design, use the `set_clock_latency` command.

Note: The Timing Analyzer automatically computes network latencies; therefore, you only can characterize source latency with the `set_clock_latency` command. You must use the `-source` option.

2.3.4.6.2. Clock Uncertainty

By default, the Timing Analyzer creates clocks that are ideal and have perfect edges. To mimic clock-level effects like jitter, you can add uncertainty to those clock edges. The Timing Analyzer automatically calculates appropriate setup and hold uncertainties and applies those uncertainties to all clock transfers in your design, even if you do not include the `derive_clock_uncertainty` command in your `.sdc` file. Setup and hold uncertainties are a critical part of constraining your design correctly.

The Timing Analyzer subtracts setup uncertainty from the data required time for each applicable path and adds the hold uncertainty to the data required time for each applicable path. This slightly reduces the setup and hold slack on each path.

The Timing Analyzer accounts for uncertainty clock effects for three types of clock-to-clock transfers: intraclock transfers, interclock transfers, and I/O interface clock transfers.

- Intraclock transfers occur when the register-to-register transfer takes place in the device and the source and destination clocks come from the same PLL output pin or clock port.
- Interclock transfers occur when a register-to-register transfer takes place in the core of the device and the source and destination clocks come from a different PLL output pin or clock port.
- I/O interface clock transfers occur when data transfers from an I/O port to the core of the device or from the core of the device to the I/O port.

To manually specify clock uncertainty, use the `set_clock_uncertainty` command. You can specify the uncertainty separately for setup and hold. You can also specify separate values for rising and falling clock transitions. You can override the value that the `derive_clock_uncertainty` command automatically applies.

The `derive_clock_uncertainty` command accounts for PLL clock jitter, if the clock jitter on the input to a PLL is within the input jitter specification for PLL's in the target device. If the input clock jitter for the PLL exceeds the specification, add additional uncertainty to your PLL output clocks to account for excess jitter with the `set_clock_uncertainty -add` command. Refer to the device handbook for your device for jitter specifications.

You can also use `set_clock_uncertainty -add` to account for peak-to-peak jitter from a board when the jitter exceeds the jitter specification for that device. In this case you add uncertainty to both setup and hold equal to 1/2 the jitter value:

```
set_clock_uncertainty –setup –to <clock name>  \
    -setup –add <p2p jitter/2>
```

```
set_clock_uncertainty –hold –enable_same_physical_edge –to <clock name> \
    –add <p2p jitter/2>
```

There is a complex set of precedence rules for how the Timing Analyzer applies values from `derive_clock_uncertainty` and `set_clock_uncertainty`, which depend on the order of commands and options in your `.sdc` files. The Help topics below contain complete descriptions of these rules. These precedence rules are easier to implement if you follow these recommendations:

- To assign your own clock uncertainty values to any clock transfers, put your `set_clock_uncertainty` exceptions after the `derive_clock_uncertainty` command in the `.sdc` file.
- When you use the `-add` option for `set_clock_uncertainty`, the value you specify is additive to the `derive_clock_uncertainty` value. If you do not specify `-add`, the value you specify replaces the value from `derive_clock_uncertainty`.

2.3.5. Creating I/O Constraints

The Timing Analyzer reviews setup and hold relationships for designs in which an external source interacts with a register internal to the design. The Timing Analyzer supports input and output external delay modeling with the `set_input_delay` and `set_output_delay` commands. You can specify the clock and minimum and maximum arrival times relative to the clock.

Specify internal and external timing requirements before you fully analyze a design. With external timing requirements specified, the Timing Analyzer verifies the I/O interface, or periphery of the device, against any system specification.

2.3.5.1. Input Constraints (set_input_delay)

Input constraints allow specify all the external delays feeding the device. Specify input requirements for all input ports in your design.

```
set_input_delay -clock { clock } -clock_fall -fall -max 20 foo
```

Use the **Set Input Delay** (`set_input_delay`) constraint to specify external input delay requirements. Specify the **Clock name** (`-clock`) to reference the virtual or actual clock. You can specify a clock to allow the Timing Analyzer to correctly derive clock uncertainties for interclock and intraclock transfers. The clock defines the launching clock for the input port. The Timing Analyzer automatically determines the latching clock inside the device that captures the input data, because all clocks in the device are defined.

Figure 53. Input Delay Diagram

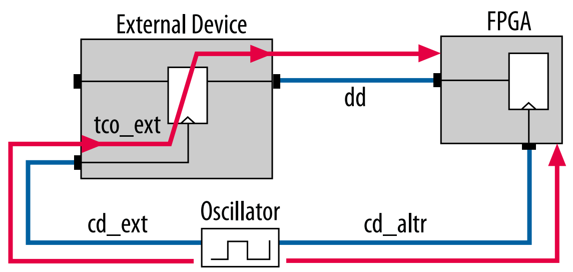


Figure 54. Input Delay Calculation

$$\text{input delay}_{MAX} = (\text{cd\_ext}_{MAX} - \text{cd\_altr}_{MIN}) + \text{tco\_ext}_{MAX} + \text{dd}_{MAX}$$
$$\text{input delay}_{MIN} = (\text{cd\_ext}_{MIN} - \text{cd\_altr}_{MAX}) + \text{tco\_ext}_{MIN} + \text{dd}_{MIN}$$

2.3.5.2. Output Constraints (set_output_delay)

Output constraints specify all external delays from the device for all output ports in your design.

```
set_output_delay -clock { clock } -clock_fall -rise -max 2 foo
```

Use the **Set Output Delay** (`set_output_delay`) constraint to specify external output delay requirements. Specify the **Clock name** (`-clock`) to reference the virtual or actual clock. When specifying a clock, the clock defines the latching clock for the output port. The Timing Analyzer automatically determines the launching clock inside the device that launches the output data, because all clocks in the device are defined. The following figure is an example of an output delay referencing a virtual clock.

Figure 55. Output Delay Diagram

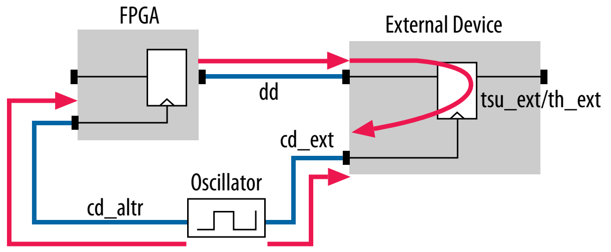


Figure 56. Output Delay Calculation

$$\text{output delay}_{MAX} = \text{dd}_{MAX} + \text{tsu\_ext} + (\text{cd\_altr}_{MAX} - \text{cd\_ext}_{MIN})$$
$$\text{output delay}_{MIN} = (\text{dd}_{MIN} - \text{th\_ext} + (\text{cd\_altr}_{MIN} - \text{cd\_ext}_{MAX}))$$

2.3.6. Creating Delay and Skew Constraints

You can specify skew and delays to model external device timing and board timing parameters.

2.3.6.1. Advanced I/O Timing and Board Trace Model Delay

The Timing Analyzer can use advanced I/O timing and board trace model constraints to model I/O buffer delays in your design.

If you change any advanced I/O timing settings or board trace model assignments, recompile your design before you analyze timing, or use the `-force_dat` option to force delay annotation when you create a timing netlist.

Example 6. Forcing Delay Annotation

```
create_timing_netlist -force_dat
```

2.3.6.2. Maximum Skew (set_max_skew)

The **Set Max Skew** (`set_max_skew`) constraint specifies the maximum allowable skew between the sets of registers or ports or ports you specify. In order to constrain skew across multiple paths, you must constrain all such paths within a single `set_max_skew` constraint.

```
set_max_skew -from_clock { clock } -to_clock { * } -from foo -to blat 2
```

The `set_max_delay`, `set_min_delay`, and `set_multicycle_path` do not affect `set_max_skew` timing constraints for this 18.1 version of the Timing Analyzer. However, `set_false_path` and `set_clock_groups` do impact `set_max_skew`. If your design targets an Intel Arria 10 device or Intel Cyclone 10 device, `set_clock_groups` does not affect `set_max_skew` constraints.

Table 12. set_max_skew Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-exclude <`*`Tcl list`*`>`	A Tcl list of parameters to exclude during skew analysis. This list includes one or more of the following: `utsu`, `uth`, `utco`, `from_clock`, `to_clock`, `clock_uncertainty`, `ccpp`, `input_delay`, `output_delay`, `odv`. *Note:* Intel Arria 10 devices do no support this argument.
`-fall_from_clock <`*`names`*`>`	Valid source clocks (Tcl matches string patterns).
`names>`	Valid destination clocks (Tcl matches string patterns).
`-from <-fall_to_clock <`*`names`*`>`[1]	Valid sources (Tcl matches string patterns).
`-fall_to_clock -from_clock <`*`names`*`>`	Valid source clocks (Tcl matches string patterns).

continued...

(1) Legal values for the -from and -to options are collections of clocks, registers, ports, pins, cells or partitions in a design.

Arguments	Description
`-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>`	Option to interpret skew constraint as a multiple of the clock period.
`-include <Tcl list>`	Tcl list of parameters to include during skew analysis. This list can include one or more of the following: `utsu`, `uth`, `utco`, `from_clock`, `to_clock`, `clock_uncertainty`, `ccpp`, `input_delay`, `output_delay`, `odv`. *Note:* Intel Arria 10 devices do not support this argument .
`-rise_from_clock <names>`	Valid source clocks (Tcl matches string patterns).
`-rise_to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
`-skew_value_multiplier <multiplier>`	Value by which the clock period multiplies to compute skew requirement.
`-to <names>`(1)	Valid destinations (Tcl matches string patterns)
`-to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
`<skew>`	Skew you require.

Applying maximum skew constraints between clocks applies the constraint from all register or ports driven by the clock you specify (with the `-from` option) to all registers or ports driven by the clock you specify (with the `-to` option).

Use the `-include` and `-exclude` options to include or exclude one or more of the following: register micro parameters (`utsu`, `uth`, `utco`), clock arrival times (`from_clock`, `to_clock`), clock uncertainty (`clock_uncertainty`), common clock path pessimism removal (`ccpp`), input and output delays (`input_delay`, `output_delay`) and on-die variation (`odv`).

Max skew analysis can include data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation, and ccpp removal. Among these, only ccpp removal disables during the Fitter by default. When you use `-include` , the default analysis includes those in the inclusion list. Similarly, if you use `-exclude`, the default analysis excludes those in the exclusion list. When both the `-include` and `-exclude` options specify the same parameter, that parameter is excluded.

Note: If your design targets an Intel Arria 10 device, `-exclude` and `-include` are not supported.

Use `-get_skew_value_from_clock_period` to set the skew as a multiple of the launching or latching clock period, or whichever of the two has a smaller period. If you use this option, set `-skew_value_multiplier`, and you may not set the positional skew option. If more than one clock clocks the set of skew paths, Timing Analyzer uses the clock with smallest period to compute the skew constraint.

Click **Report Max Skew** (`report_max_skew`) to view the max skew analysis. Since skew occurs between two or more paths, no results display if the `-from`/`-from_clock` and `-to`/`-to_clock` filters satisfy less than two paths.

2.3.6.3. Net Delay (set_net_delay)

Use the `set_net_delay` command to set the net delays and perform minimum or maximum timing analysis across nets.

The `-from` and `-to` options can be string patterns or pin, port, register, or net collections. When you use pin or net collection, include output pins or nets in the collection.

```
set_net_delay -from reg_a -to reg_c -max 20
```

Table 13. set_net_delay Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-from <names>`	Valid source pins, ports, registers or nets (Tcl matches string patterns).
`-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>`	Option to interpret net delay constraint as a multiple of the clock period.
`-max`	Specifies maximum delay.
`-min`	Specifies minimum delay.
`-to <names>`[2]	Valid destination pins, ports, registers or nets (Tcl matches string patterns).
`-value_multiplier <multiplier>`	Value by which the clock period multiplies to compute net delay requirement.
`<delay>`	Delay value.

If you use the `-min` option, the Timing Analyzer calculates slack by determining the minimum delay on the edge. If you use `-max` option, the Timing Analyzer calculates slack by determining the maximum edge delay.

Use `-get_skew_value_from_clock_period` to set the net delay requirement as a multiple of the launching or latching clock period, or whichever of the two has a smaller or larger period. If you use this option, you must also set `-value_multiplier`, and you must not set the positional delay option. If more than one clock clocks the set of nets, the Timing Analyzer uses the net with smallest period to compute the constraint for a `-max` constraint, and the largest period for a `-min` constraint. If there are no clocks clocking the endpoints of the net (that is, if the endpoints of the nets are not registers or constraint ports), then the Timing Analyzer ignores the net delay constraint.

2.3.6.4. Create Timing Netlist

You can configure or load the timing netlist that the Timing Analyzer uses to calculate path delay data.

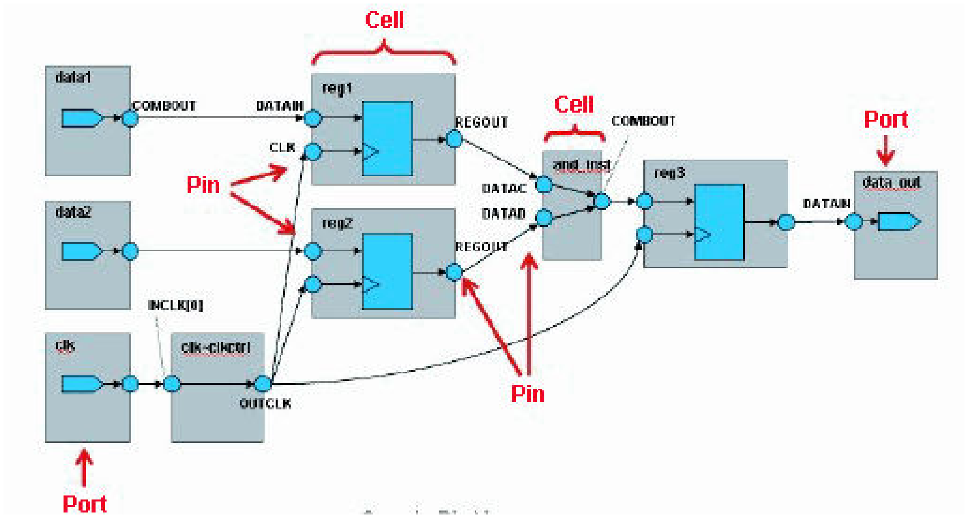
You must generate the timing netlist before running timing analysis. You can use the **Create Timing Netlist** dialog box or the **Create Timing Netlist** command in the **Tasks** pane. **Create Timing Netlist** also generates Advanced I/O Timing reports if you turn on **Enable Advanced I/O Timing** in the **Timing Analyzer** page of the **Settings** dialog box.

[2] If no -to option, or if `-to` is a wildcard ("*") character, all the output pins and registers on timing netlist become valid destination points.

Note: The Compiler creates the timing netlist during compilation. The timing netlist does not reflect any configuration changes that occur after the device enters user mode, such as dynamic transceiver reconfiguration. This applies to all device families except transceivers on Intel Arria 10 devices with the Multiple Reconfiguration Profiles feature.

The following diagram shows how the Timing Analyzer interprets and classifies timing netlist data for a sample design.

Figure 57. How Timing Analyzer Interprets the Timing Netlist



2.3.7. Creating Timing Exceptions

Timing exceptions modify (or provide exception to) the default timing analysis behavior to account for your specific design conditions. Specify timing exceptions after specifying clocks and input and output delay constraints, because timing exceptions modify the default analysis.

2.3.7.1. Timing Constraint Precedence

If the same clock or node names occur in multiple timing exceptions, the Timing Analyzer observes the following order of timing constraint precedence:

1. **Set False Path** (`set_false_path`) is the first priority
2. **Set Minimum Delay** (`set_min_delay`) and **Set Maximum Delay** (`set_max_delay`) are the second priority.
3. **Set Multicycle Path** (`set_multicycle_path`) is the third priority.

The false path timing exception has the highest precedence. Within each category, assignments to individual nodes have precedence over assignments to clocks. For exceptions of the same type:

1. `-from <node>` is the first priority.
2. `-to <node>` is the second priority.
3. `-thru <node>` is the third priority.
4. `-from <clock>` is the fourth priority.
5. `-to <clock>` is the fifth priority.

An asterisk wildcard (*) for any of these options applies the same precedence as not specifying the option at all. For example, `-from a -to *` is treated identically to `-from a` with regards precedence.

Precedence example:

1. `set_max_delay 1 -from x -to y`
2. `set_max_delay 2 -from x`
3. `set_max_delay 3 -to y`

The first exception has higher priority than either of the other two, since the first exception specifies a `-from` (while #3 doesn't) and specifies a `-to` (while #2 doesn't). In the absence of the first exception, the second exception has higher priority than the third, since the second exception specifies a `-from`, which the third does not. Finally, the remaining order of precedence for additional exceptions is order-dependent, such that the assignments most recently created overwrite, or partially overwrite, earlier assignments.

`set_net_delay` or `set_max_skew` exceptions analyze independently of minimum or maximum delays, or multicycle path constraints.

- The `set_net_delay` exception applies regardless the existence of a `set_false_path` exception, or `set_clock_group` exception, on the same nodes.
- When targeting the Intel Arria 10 device or Intel Cyclone 10 device and using the 18.1 version of the Timing Analyzer, the `set_max_skew` exception applies regardless of any `set_clock_group` exception on the same nodes, but a `set_false_path` exception overrides a `set_max_skew` exception.

2.3.7.2. False Paths (set_false_path)

The **Set False Path** (`set_false_path`) constraint allows you to exclude a path from timing analysis, such as test logic or any other path not relevant to the circuit's operation. You can specify the source (`-from`), common through elements (`- thru`), and destination (`-to`) elements of that path.

The following SDC command makes false path exceptions from all registers beginning with A, to all registers beginning with B:

```
set_false_path -from [get_pins A*] -to [get_pins B*]
```

You can specify either a point-to-point or clock-to-clock path as a false path. For example, you can specify a false path for a static configuration register that is written once during power-up initialization, but does not change state again.

Although signals from static configuration registers often cross clock domains, you may not want to make false path exceptions to a clock-to-clock path, because some data may transfer across clock domains. However, you can selectively make false path exceptions from the static configuration register to all endpoints.

The Timing Analyzer assumes all clocks are related unless you specify otherwise. Use clock groups to more efficiently make false path exceptions between clocks, rather than writing multiple `set_false_path` exceptions between each clock transfer you want to eliminate.

Related Information

Creating Clock Groups (set_clock_groups) on page 44

2.3.7.3. Minimum and Maximum Delays

To specify an absolute minimum or maximum delay for a path, use the **Set Minimum Delay** (`set_min_delay`) or the **Set Maximum Delay** (`set_max_delay`) constraints, respectively. Specifying minimum and maximum delay directly overwrites existing setup and hold relationships with the minimum and maximum values.

Use the `set_max_delay` and `set_min_delay` constraints for asynchronous signals that do not have a specific clock relationship in your design, but require a minimum and maximum path delay. You can create minimum and maximum delay exceptions for port-to-port paths through the device without a register stage in the path. If you use minimum and maximum delay exceptions to constrain the path delay, specify both the minimum and maximum delay of the path; do not constrain only the minimum or maximum value.

If the source or destination node is clocked, the Timing Analyzer takes into account the clock paths, allowing more or less delay on the data path. If the source or destination node has an input or output delay, the minimum or maximum delay check also includes that delay.

If you specify a minimum or maximum delay between timing nodes, the delay applies only to the path between the two nodes. If you specify a minimum or maximum delay for a clock, the delay applies to all paths where the clock clocks the source node or destination node.

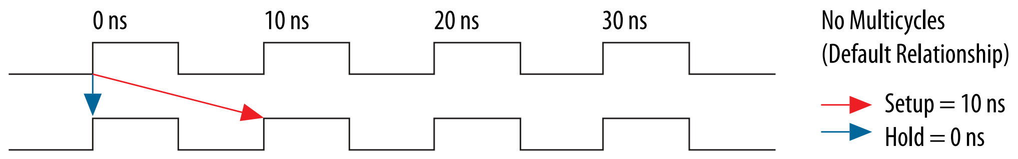
You can create a minimum or maximum delay exception for an output port that does not have an output delay constraint. You cannot report timing for the paths that relate to the output port; however, the Timing Analyzer reports any slack for the path in the setup summary and hold summary reports. Because there is no clock that relates to the output port, the Timing Analyzer reports no clock for timing paths of the output port.

Note: To report timing with clock filters for output paths with minimum and maximum delay constraints, you can set the output delay for the output port with a value of zero. You can use an existing clock from the design or a virtual clock as the clock reference.

2.3.7.4. Multicycle Paths

By default, the Timing Analyzer performs a single-cycle analysis, which is the most restrictive type of analysis. When analyzing a path without a multicycle constraint, the Timing Analyzer determines the setup launch and latch edge times by identifying the closest two active edges in the respective waveforms.

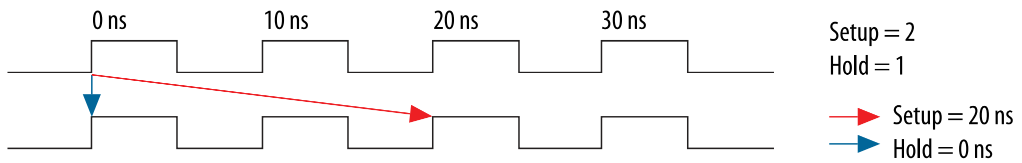
Figure 58. Default Setup and Hold Relationship (No Multicycle)



For hold time analysis, the timing analyzer analyzes the path for two timing conditions for every possible setup relationship, not just the worst-case setup relationship. Therefore, the hold launch and latch times can be unrelated to the setup launch and latch edges. The Timing Analyzer does not report negative setup or hold relationships. When the Timing Analyzer detects either a negative setup or a negative hold relationship, the Timing Analyzer moves both the launch and latch edges until the setup and hold relationship becomes positive.

A multicycle constraint adjusts this default setup or hold relationship by the number of clock cycles you specify, based on the source (`-start`) or destination (`-end`) clock. A setup multicycle constraint of 2 extends the worst-case setup latch edge by one destination clock period. If you do not specify `-start` and `-end` values, the default constraint is `-end`.

Figure 59. Setup and Hold Relationship with Multicycle = 2



Hold multicycle constraints derive from the default hold position (the default value is 0). An end hold multicycle constraint of 1 effectively subtracts one destination clock period from the default hold latch edge.

When the objects are timing nodes, the multicycle constraint only applies to the path between the two nodes. When an object is a clock, the multicycle constraint applies to all paths where the source node (`-from`) or destination node (`-to`) is clocked by the clock. When you adjust a setup relationship with a multicycle constraint, the hold relationship adjusts automatically.

You can use timing constraints to modify either the launch or latch edge times that the Timing Analyzer uses to determine a setup relationship or hold relationship.

Table 14. Multicycle Constraints

Command	Modification
`set_multicycle_path -setup -end <value>`	Latch edge time of the setup relationship.
`set_multicycle_path -setup -start<value>`	Launch edge time of the setup relationship.
`set_multicycle_path -hold -end <value>`	Latch edge time of the hold relationship.
`set_multicycle_path -hold -start <value>`	Launch edge time of the hold relationship.

2.3.7.4.1. Common Multicycle Applications

Multicycle exceptions adjust the timing requirements for a register-to-register path, allowing the Fitter to optimally place and route a design. Two common multicycle applications are relaxing setup to allow a slower data transfer rate, and altering the setup to account for a phase shift.

2.3.7.4.2. Relaxing Setup with Multicycle (set_multicycle_path)

You can use a multicycle exception when the data transfer rate is slower than the clock cycle. Relaxing the setup relationship increases the window when timing analysis accepts data as valid.

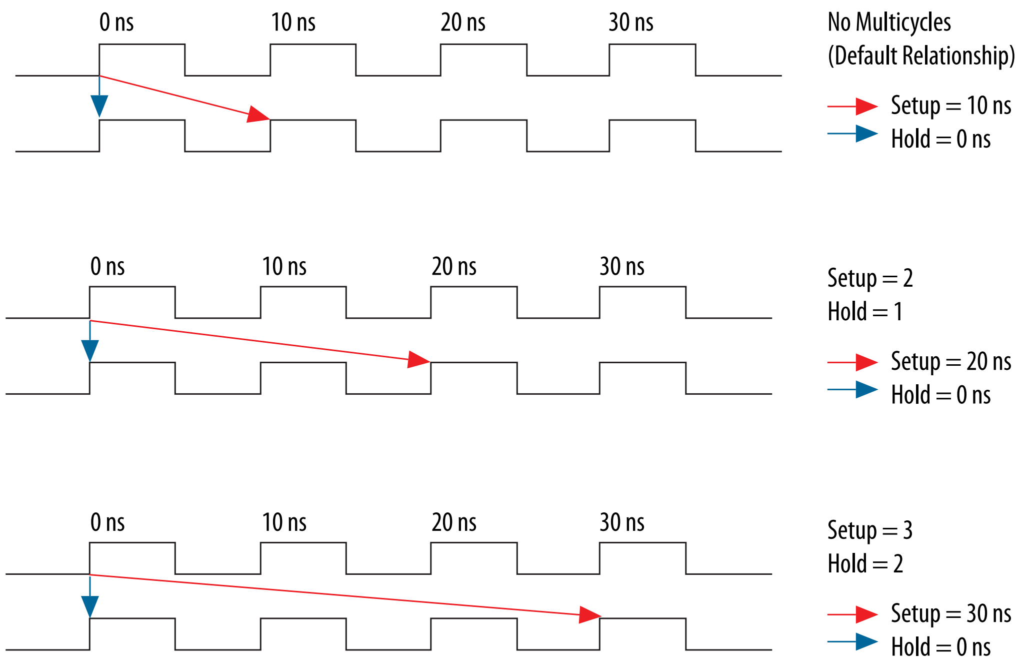
In the following example, the source clock has a period of 10 ns, but the clock enables a group of registers, so the registers only enable every other cycle. Since the registers are fed by a 10 ns clock, the Timing Analyzer reports a setup of 10 ns and a hold of 0 ns. However, since the data is transferring every other cycle, the Timing Analyzer must analyze the relationships as if the clock is operating at 20 ns. That results in a setup of 20 ns, while the hold remains 0 ns, thus extending the window for data recognition.

The following pair of multicycle assignments relax the setup relationship by specifying the `-setup` value of N and the `-hold` value as N-1. You must specify the hold relationship with a `-hold` assignment to prevent a positive hold requirement.

Constraint to Relax Setup and Maintain Hold

```
set_multicycle_path -setup -from src_reg* -to dst_reg* 2
set_multicycle_path -hold -from src_reg* -to dst_reg* 1
```

Figure 60. Multicycle Setup Relationships



You can extend this pattern to create larger setup relationships to ease timing closure requirements. A common use for this exception is when writing to asynchronous RAM across an I/O interface. The delay between address, data, and a write enable may be several cycles. A multicycle exception to I/O ports allows extra time for the address and data to resolve before the enable occurs.

The following constraint relaxes the setup by three cycles:

Three Cycle I/O Interface Constraint

```
set_multicycle_path -setup -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 3
set_multicycle_path -hold -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 2
```

2.3.7.4.3. Accounting for a Phase Shift (-phase)

In the following example, the design contains a PLL that performs a phase-shift on a clock whose domain exchanges data with domains that do not experience the phase shift. This occurs when the destination clock phase-shifts forward, and the source clock does not shift. The default setup relationship becomes that phase-shift, thus shifting the window when data is valid.

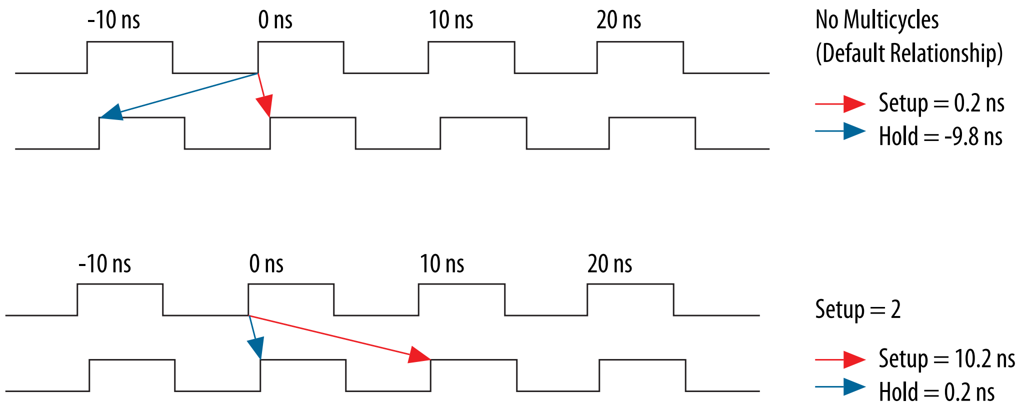
For example, the following code phase-shifts one output of a PLL forward by a small amount, in this case 0.2 ns.

Cross Domain Phase-Shift

```
create_generated_clock -source pll|inclk[0] -name pll|clk[0] pll|clk[0]
create_generated_clock -source pll|inclk[0] -name pll|clk[1] -phase 30 pll|clk[1]
```

The default setup relationship for this phase-shift is 0.2 ns, shown in Figure A, creating a scenario where the hold relationship is negative, which makes achieving timing closure nearly impossible.

Figure 61. Phase-Shifted Setup and Hold



The following constraint allows the data to transfer to the following edge:

```
set_multicycle_path -setup -from [get_clocks clk_a] -to [get_clocks clk_b] 2
```

The hold relationship derives from the setup relationship, making a multicycle hold constraint unnecessary.

Related Information

Same Frequency Clocks with Destination Clock Offset on page 69

2.3.7.5. Multicycle Exception Examples

The examples in this section illustrate how the multicycle exceptions affect the default setup and hold analysis in the Timing Analyzer. The multicycle exceptions apply to a simple register-to-register circuit. Both the source and destination clocks are set to 10 ns.

2.3.7.5.1. Default Multicycle Analysis

By default, the Timing Analyzer performs a single-cycle analysis to determine the setup and hold checks. Also, by default, the Timing Analyzer sets the end multicycle setup assignment value to one and the end multicycle hold assignment value to zero.

The source and the destination timing waveform for the source register and destination register, respectively where HC1 and HC2 are hold checks 1 and 2 and SC is the setup check.

Figure 62. Default Timing Diagram

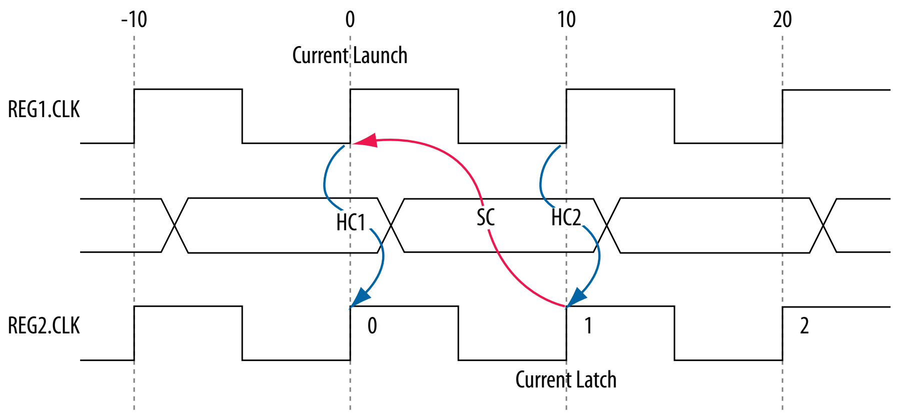


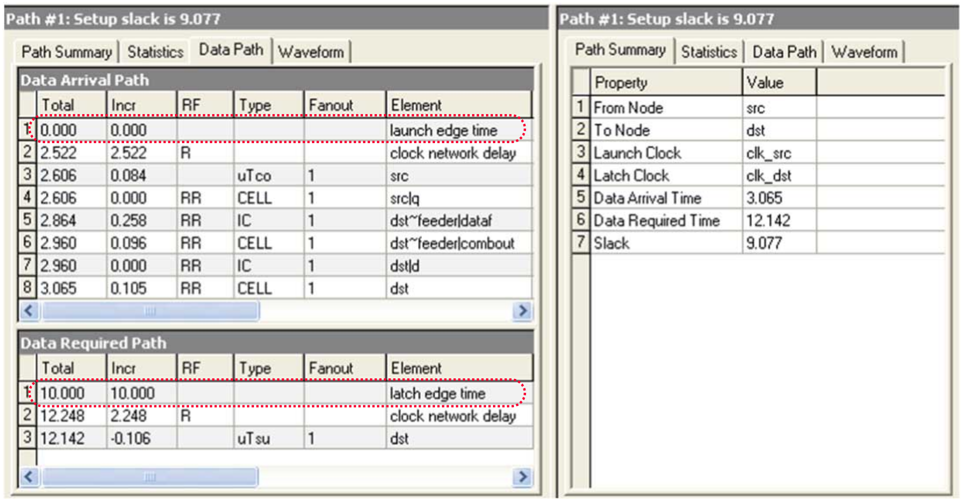
Figure 63. Setup Check Calculation

$$
\begin{aligned}
\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\
&= 10\text{ ns} - 0\text{ ns} \\
&= 10\text{ ns}
\end{aligned}
$$

The most restrictive setup relationship with the default single-cycle analysis, that is, a setup relationship with an end multicycle setup assignment of one, is 10 ns.

The setup report for the default setup in the Timing Analyzer with the launch and latch edges highlighted.

Figure 64. Setup Report



Path #1: Setup slack is 9.077

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	RR	CELL	1	src|q
5	2.864	0.258	RR	IC	1	dst~feeder|dataf
6	2.960	0.096	RR	CELL	1	dst~feeder|combout
7	2.960	0.000	RR	IC	1	dst|d
8	3.065	0.105	RR	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	10.000	10.000				latch edge time
2	12.248	2.248	R			clock network delay
3	12.142	-0.106		uTsu	1	dst

Path #1: Setup slack is 9.077

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Data Arrival Time	3.065
6	Data Required Time	12.142
7	Slack	9.077

Figure 65. Hold Check Calculation

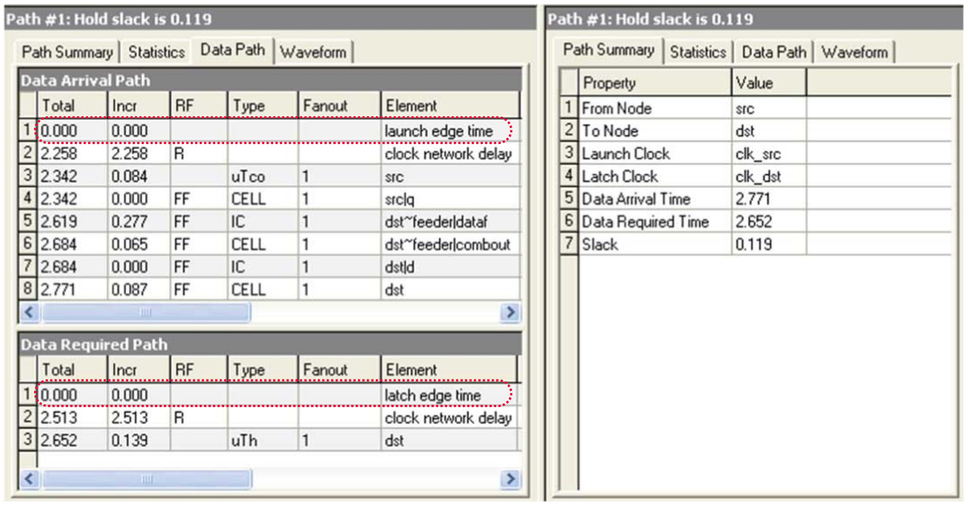
$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 0\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 10\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

The most restrictive hold relationship with the default single-cycle analysis, that a hold relationship with an end multicycle hold assignment of zero, is 0 ns.

The hold report for the default setup in the Timing Analyzer with the launch and latch edges highlighted.

Figure 66. Hold Report



Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.258	2.258	R			clock network delay
3	2.342	0.084		uTco	1	src
4	2.342	0.000	FF	CELL	1	src|q
5	2.619	0.277	FF	IC	1	dst~feeder|dataf
6	2.684	0.065	FF	CELL	1	dst~feeder|combout
7	2.684	0.000	FF	IC	1	dst|d
8	2.771	0.087	FF	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				latch edge time
2	2.513	2.513	R			clock network delay
3	2.652	0.139		uTh	1	dst

Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Data Arrival Time	2.771
6	Data Required Time	2.652
7	Slack	0.119

2.3.7.5.2. End Multicycle Setup = 2 and End Multicycle Hold = 0

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is zero.

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
```

Note: The Timing Analyzer does not require an end multicycle hold value because the default end multicycle hold value is zero.

In this example, the setup relationship relaxes by a full clock period by moving the latch edge to the next latch edge. The hold analysis is does not change from the default settings.

The following shows the setup timing diagram for the analysis that the Timing Analyzer performs. The latch edge is a clock cycle later than in the default single-cycle analysis.

Figure 67. Setup Timing Diagram

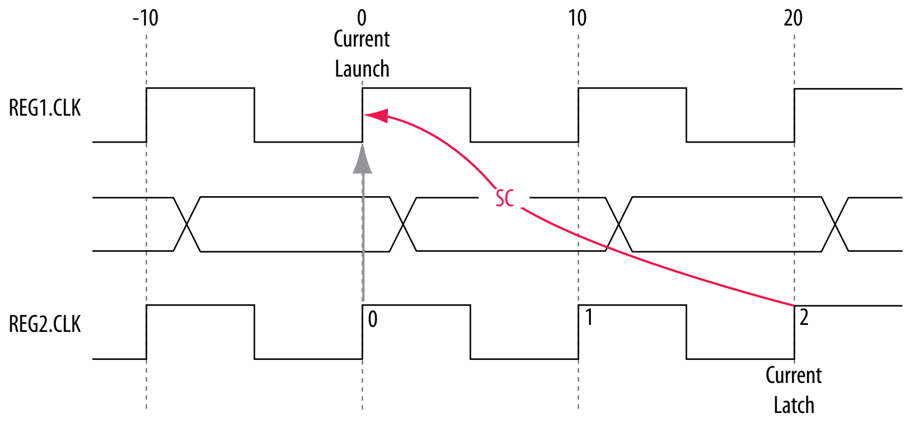


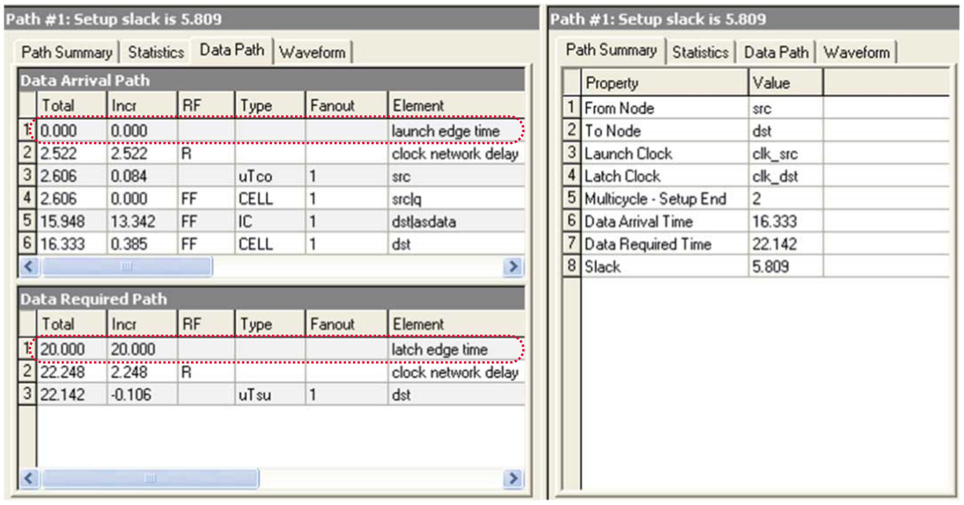
Figure 68. Setup Check Calculation

$$\begin{aligned}\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge}\\ &= 20\text{ ns} - 0\text{ ns}\\ &= 20\text{ ns}\end{aligned}$$

The most restrictive setup relationship with an end multicycle setup assignment of two is 20 ns.

The following shows the setup report in the Timing Analyzer and highlights the launch and latch edges.

Figure 69. Setup Report



Path #1: Setup slack is 5.809

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	FF	CELL	1	src|q
5	15.948	13.342	FF	IC	1	dst|asdata
6	16.333	0.385	FF	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	20.000	20.000				latch edge time
2	22.248	2.248	R			clock network delay
3	22.142	-0.106		uTsu	1	dst

Path #1: Setup slack is 5.809

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Data Arrival Time	16.333
7	Data Required Time	22.142
8	Slack	5.809

Because the multicycle hold latch and launch edges are the same as the results of hold analysis with the default settings, the multicycle hold analysis in this example is equivalent to the single-cycle hold analysis. The hold checks are relative to the setup check. Normally, the Timing Analyzer performs hold checks on every possible setup check, not only on the most restrictive setup check edges.

Figure 70. Hold Timing Diagram

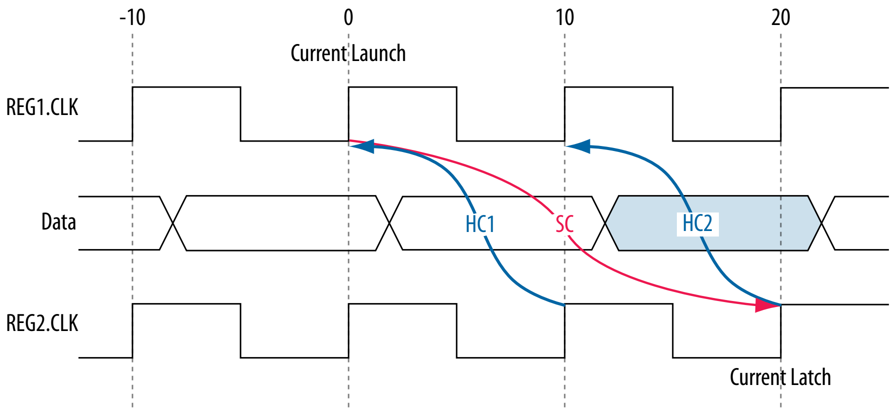


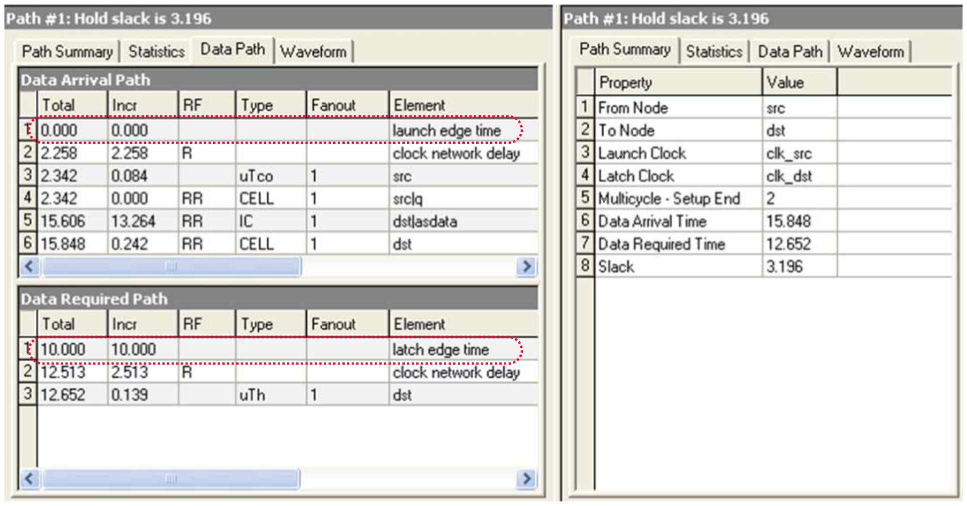
Figure 71. Hold Check Calculation

$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 10\text{ ns}\\ &= -10\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 20\text{ ns}\\ &= -10\text{ ns}\end{aligned}$$

This is the most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of zero is 10 ns.

Figure 72. Hold Report



Path #1: Hold slack is 3.196

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.258	2.258	R			clock network delay
3	2.342	0.084		uTco	1	src
4	2.342	0.000	RR	CELL	1	src|q
5	15.606	13.264	RR	IC	1	dst|asdata
6	15.848	0.242	RR	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	10.000	10.000				latch edge time
2	12.513	2.513	R			clock network delay
3	12.652	0.139		uTh	1	dst

Path #1: Hold slack is 3.196

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Data Arrival Time	15.848
7	Data Required Time	12.652
8	Slack	3.196

2.3.7.5.3. End Multicycle Setup = 2 and End Multicycle Hold = 1

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is one.

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] -hold -
end 1
```

In this example, the setup relationship relaxes by two clock periods by moving the latch edge to the left two clock periods. The hold relationship relaxes by a full period by moving the latch edge to the previous latch edge.

The following shows the setup timing diagram for the analysis that the Timing Analyzer performs:

Figure 73. Setup Timing Diagram

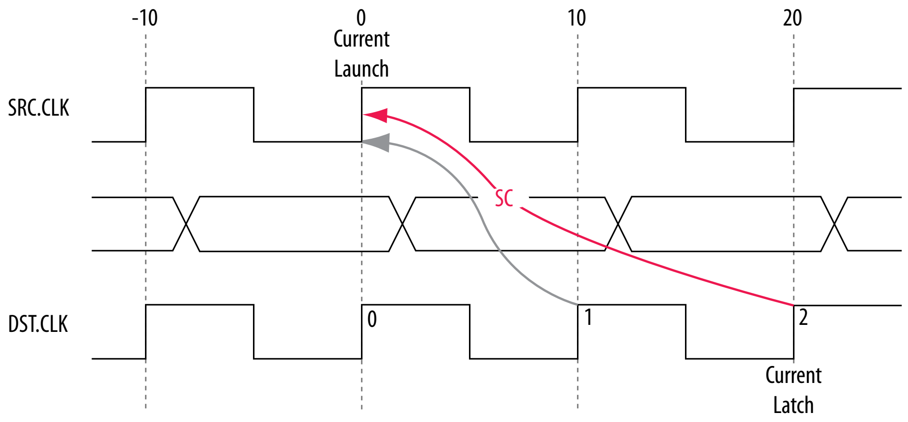


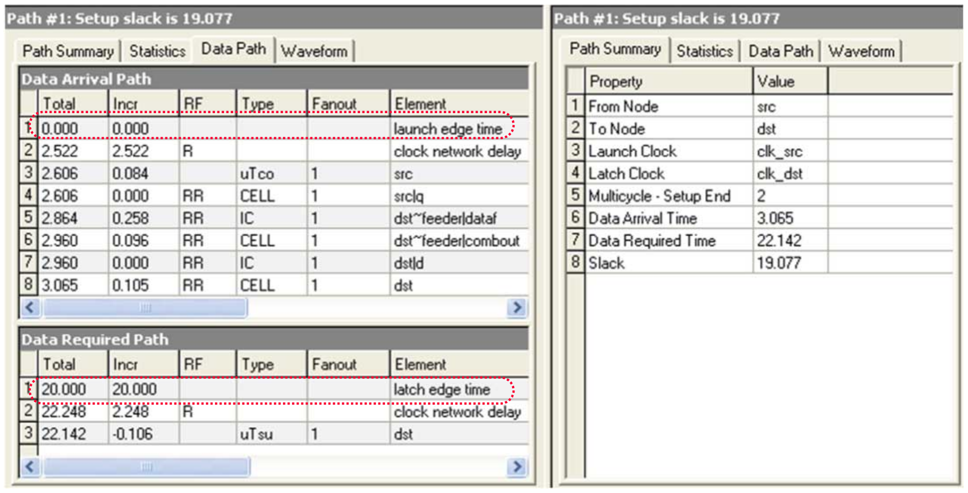
Figure 74. Setup Check Calculation

setup check = current latch edge – closest previous launch edge
= 20 ns – 0 ns
= 20 ns

The most restrictive hold relationship with an end multicycle setup assignment value of two is 20 ns.

The following shows the setup report for this example in the Timing Analyzer and highlights the launch and latch edges.

Figure 75. Setup Report



Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	RR	CELL	1	src|q
5	2.864	0.258	RR	IC	1	dst~feeder|dataf
6	2.960	0.096	RR	CELL	1	dst~feeder|combout
7	2.960	0.000	RR	IC	1	dst|d
8	3.065	0.105	RR	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	20.000	20.000				latch edge time
2	22.248	2.248	R			clock network delay
3	22.142	-0.106		uTsu	1	dst

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Data Arrival Time	3.065
7	Data Required Time	22.142
8	Slack	19.077

The following shows the timing diagram for the hold checks for this example. The hold checks are relative to the setup check.

Figure 76. Hold Timing Diagram

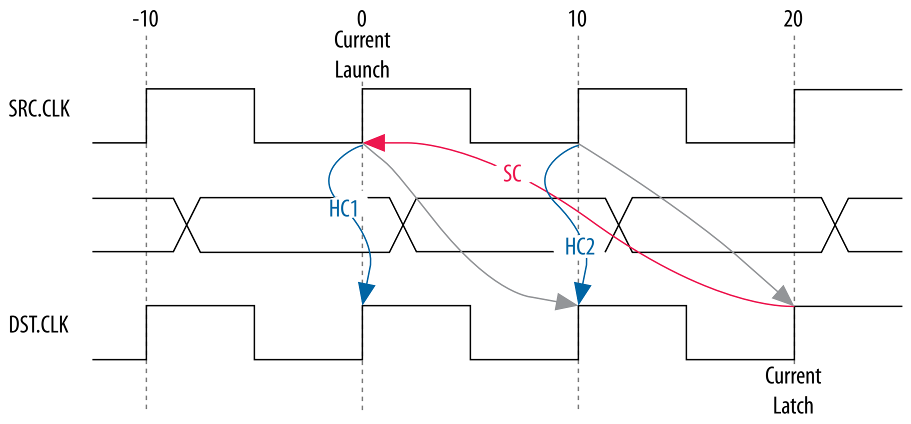


Figure 77. Hold Check Calculation

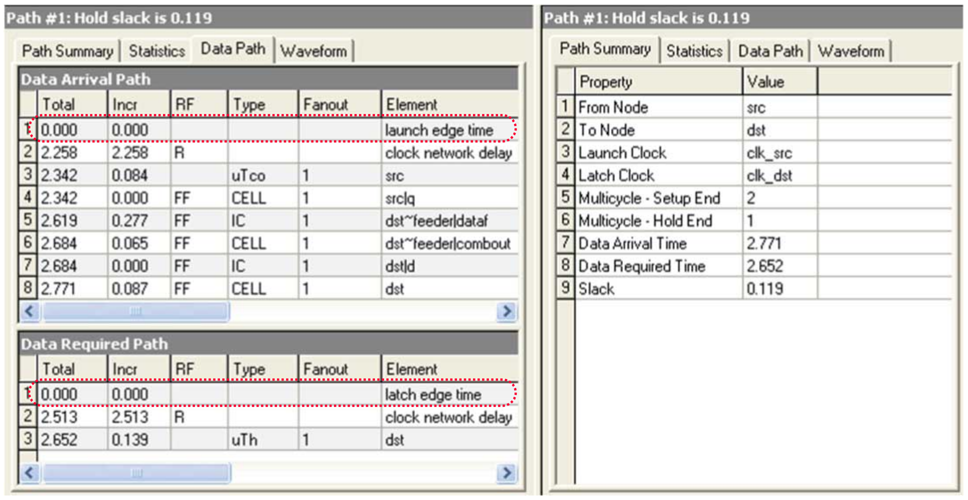
$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 0\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 10\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of one is 0 ns.

The following shows the hold report for this example in the Timing Analyzer and highlights the launch and latch edges.

Figure 78. Hold Report



Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.258	2.258	R			clock network delay
3	2.342	0.084		uTco	1	src
4	2.342	0.000	FF	CELL	1	src|q
5	2.619	0.277	FF	IC	1	dst~feeder|dataf
6	2.684	0.065	FF	CELL	1	dst~feeder|combout
7	2.684	0.000	FF	IC	1	dst|d
8	2.771	0.087	FF	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				latch edge time
2	2.513	2.513	R			clock network delay
3	2.652	0.139		uTh	1	dst

Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

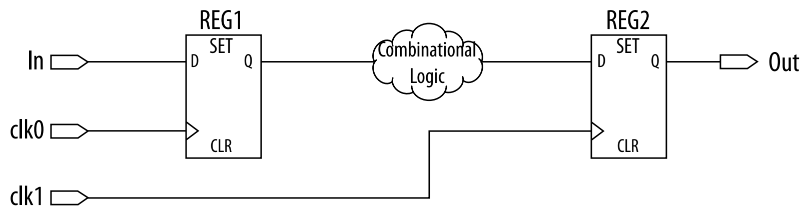
	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Multicycle - Hold End	1
7	Data Arrival Time	2.771
8	Data Required Time	2.652
9	Slack	0.119

2.3.7.5.4. Same Frequency Clocks with Destination Clock Offset

In this example, the source and destination clocks have the same frequency, but the destination clock is offset with a positive phase shift. Both the source and destination clocks have a period of 10 ns. The destination clock has a positive phase shift of 2 ns with respect to the source clock.

The following example shows a design with the same frequency clocks and a destination clock offset.

Figure 79. Same Frequency Clocks with Destination Clock Offset Diagram



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs.

Figure 80. Setup Timing Diagram

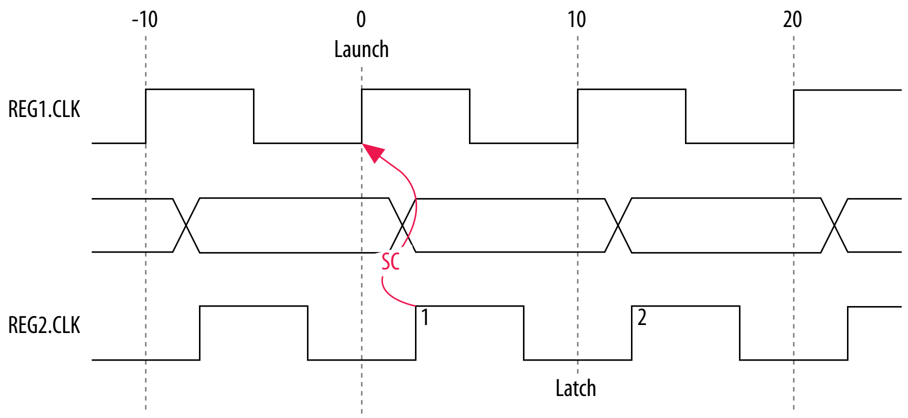


Figure 81. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\text{ ns} - 0\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

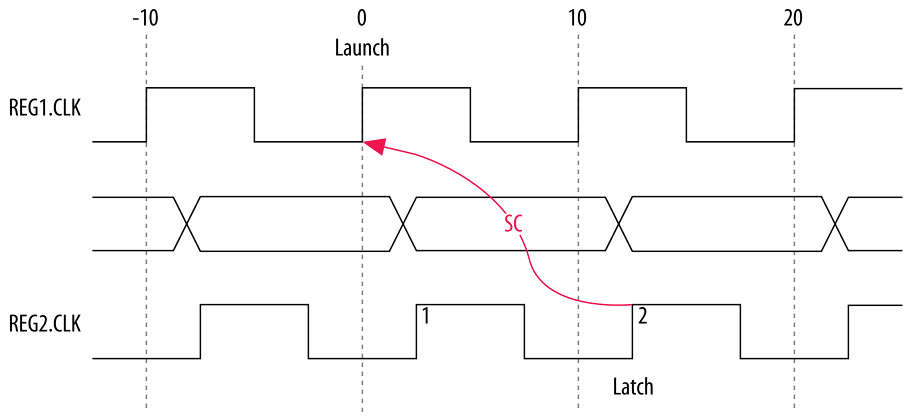
The setup relationship shown is too pessimistic and is not the setup relationship required for typical designs. To adjust the default analysis, you assign an end multicycle setup exception of two. The following shows a multicycle exception that adjusts the default analysis:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 82. Preferred Setup Relationship



The following timing diagram shows the default hold check analysis that the Timing Analyzer performs with an end multicycle setup value of two.

Figure 83. Default Hold Check

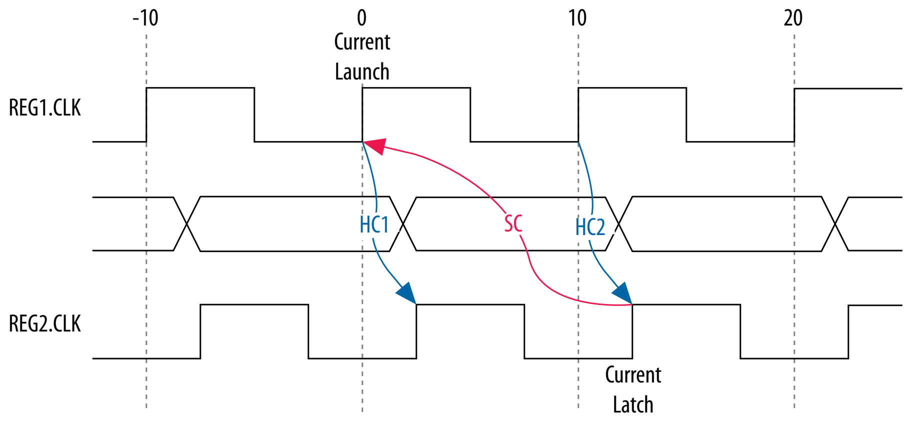


Figure 84. Hold Check Calculation

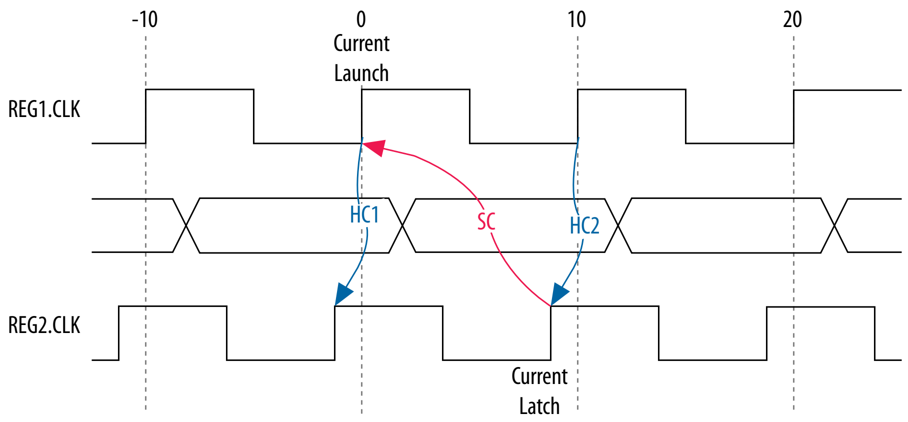
$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 2\text{ ns}\\ &= -2\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 12\text{ ns}\\ &= -2\text{ ns}\end{aligned}$$

In this example, the default hold analysis returns the preferred hold requirements and no multicycle hold exceptions are required.

The associated setup and hold analysis if the phase shift is –2 ns. In this example, the default hold analysis is correct for the negative phase shift of 2 ns, and no multicycle exceptions are required.

Figure 85. Negative Phase Shift

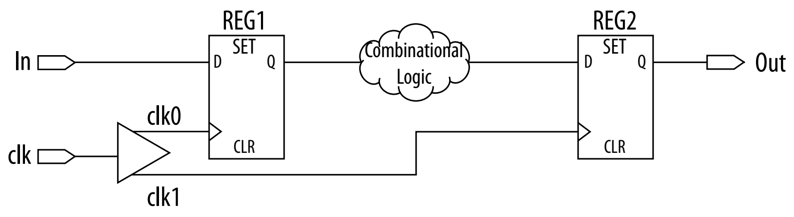


2.3.7.5.5. Destination Clock Frequency is a Multiple of the Source Clock Frequency

In this example, the destination clock frequency value of 5 ns is an integer multiple of the source clock frequency of 10 ns. The destination clock frequency can be an integer multiple of the source clock frequency when a PLL generates both clocks with a phase shift on the destination clock.

The following example shows a design in which the destination clock frequency is a multiple of the source clock frequency.

Figure 86. Destination Clock is Multiple of Source Clock



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs:

Figure 87. Setup Timing Diagram

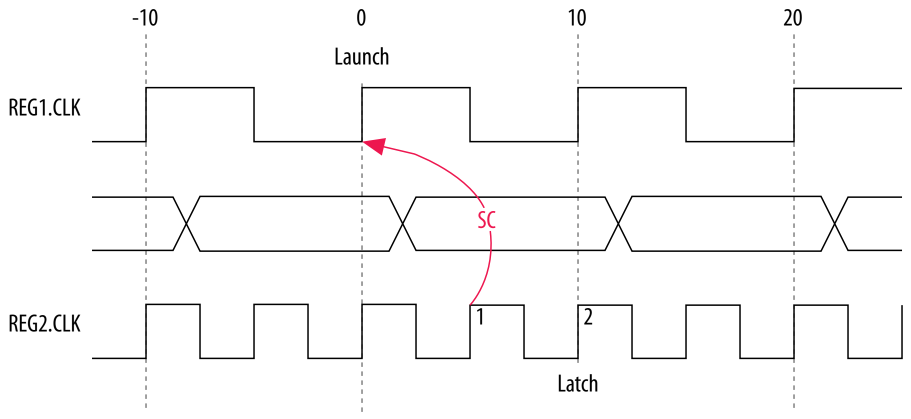


Figure 88. Setup Check Calculation

$$\begin{aligned}\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge}\\ &= 5\text{ ns} - 0\text{ ns}\\ &= 5\text{ ns}\end{aligned}$$

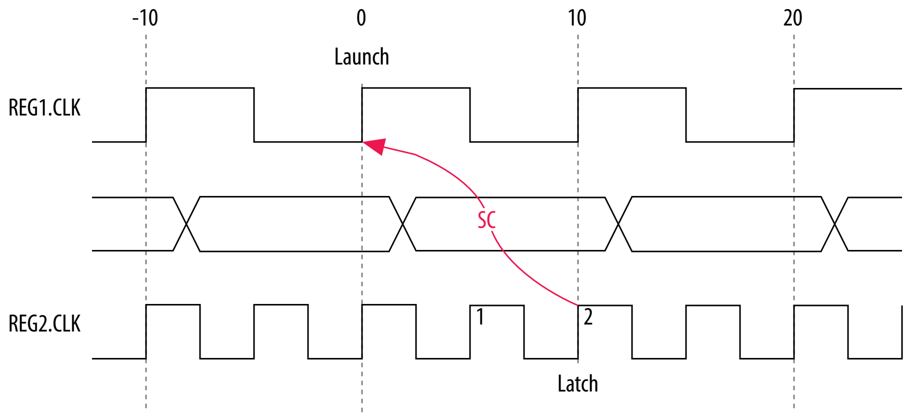
The setup relationship demonstrates that the data requires capture at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you shift the latch edge by one clock period with an end multicycle setup exception of two. The following multicycle exception assignment adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 89. Preferred Setup Analysis



The following timing diagram shows the default hold check analysis the Timing Analyzer performs with an end multicycle setup value of two.

Figure 90. Default Hold Check

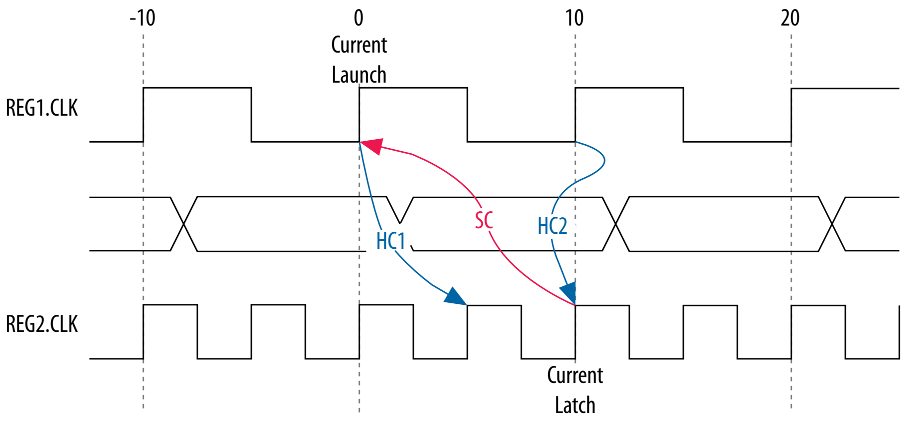


Figure 91. Hold Check Calculation

$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 5\text{ ns}\\ &= -5\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 10\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

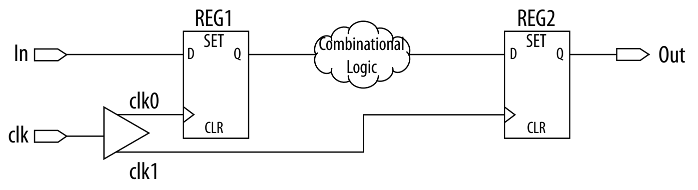
In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and must check against the data captured by the previous latch edge at 0 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

2.3.7.5.6. Destination Clock Frequency is a Multiple of the Source Clock Frequency with an Offset

This example is a combination of the previous two examples. The destination clock frequency is an integer multiple of the source clock frequency, and the destination clock has a positive phase shift. The destination clock frequency is 5 ns, and the source clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The destination clock frequency can be an integer multiple of the source clock frequency. The destination clock frequency can be with an offset when a PLL generates both clocks with a phase shift on the destination clock.

The following example shows a design in which the destination clock frequency is a multiple of the source clock frequency with an offset.

Figure 92. Destination Clock is Multiple of Source Clock with Offset



The timing diagram for the default setup check analysis the Timing Analyzer performs.

Figure 93. Setup Timing Diagram

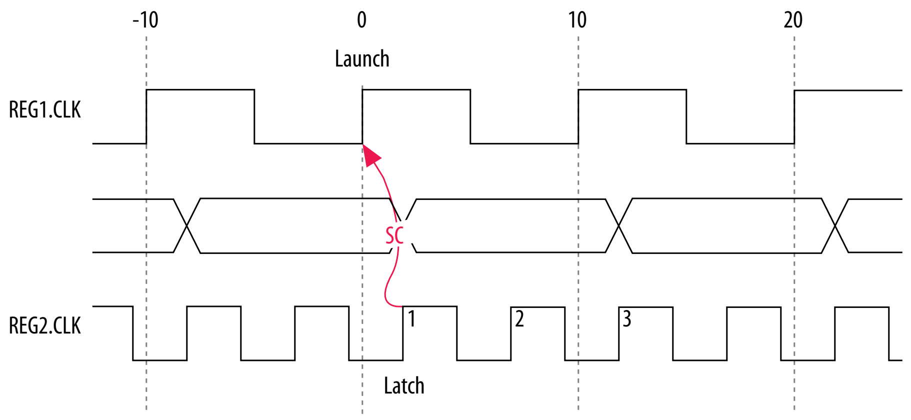


Figure 94. Hold Check Calculation

$$\begin{aligned}\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge}\\ &= 2\text{ ns} - 0\text{ ns}\\ &= 2\text{ ns}\end{aligned}$$

The setup relationship in this example demonstrates that the data does not require capture at edge one, but rather requires capture at edge two; therefore, you can relax the setup requirement. To adjust the default analysis, you shift the latch edge by one clock period, and specify an end multicycle setup exception of three.

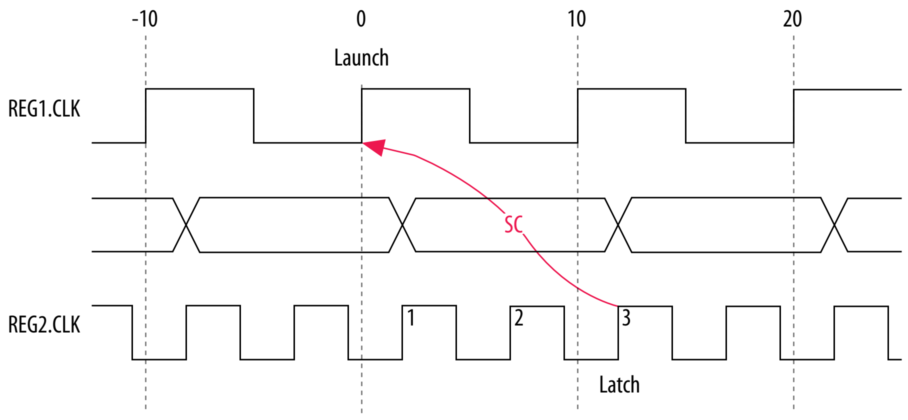
The multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 3
```

The timing diagram for the preferred setup relationship for this example.

Figure 95. Preferred Setup Analysis



The following timing diagram shows the default hold check analysis that the Timing Analyzer performs with an end multicycle setup value of three:

Figure 96. Default Hold Check

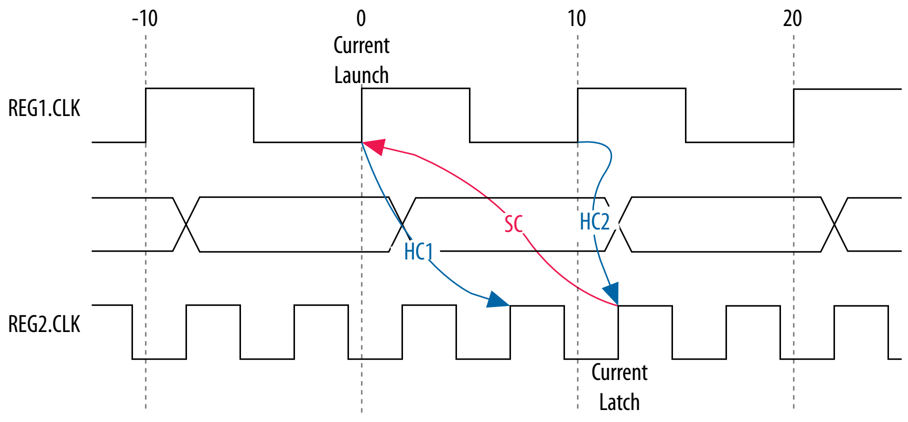


Figure 97. Hold Check Calculation

$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 5\text{ ns}\\ &= -5\text{ ns}\end{aligned}$$

$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 10\text{ ns} - 10\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

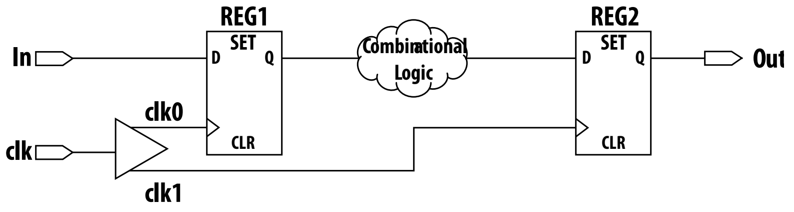
In this example, the hold check one is too restrictive. The data is launched by the edge at 0 ns, and must check against the data that the previous latch edge at 2 ns captures. This event does not occur in hold check one. To adjust the default analysis, you assign end multicycle hold exception of one.

2.3.7.5.7. Source Clock Frequency is a Multiple of the Destination Clock Frequency

In this example, the source clock frequency value of 5 ns is an integer multiple of the destination clock frequency of 10 ns. The source clock frequency can be an integer multiple of the destination clock frequency when a PLL generates both clocks and use different multiplication and division factors.

In the following example the source clock frequency is a multiple of the destination clock frequency:

Figure 98. Source Clock Frequency is Multiple of Destination Clock Frequency:



The following timing diagram shows the default setup check analysis the Timing Analyzer performs:

Figure 99. Default Setup Check Analysis

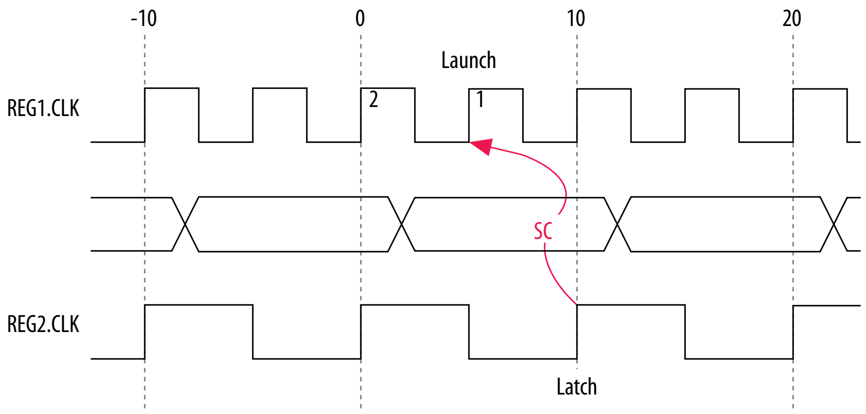


Figure 100. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\text{ ns} - 5\text{ ns} \\ &= 5\text{ ns} \end{aligned}$$

The setup relationship demonstrates that the data launched at edge one does not require capture, and the data launched at edge two requires capture; therefore, you can relax the setup requirement. To correct the default analysis, you shift the launch edge by one clock period with a start multicycle setup exception of two.

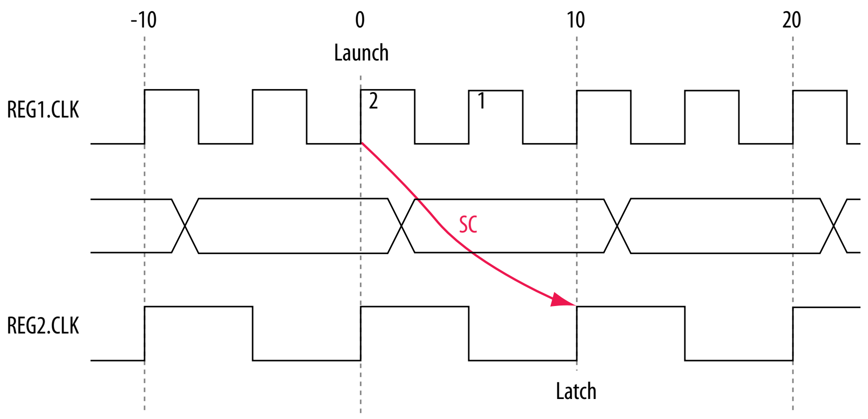
The following multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 101. Preferred Setup Check Analysis



The following timing diagram shows the default hold check analysis the Timing Analyzer performs for a start multicycle setup value of two:

Figure 102. Default Hold Check

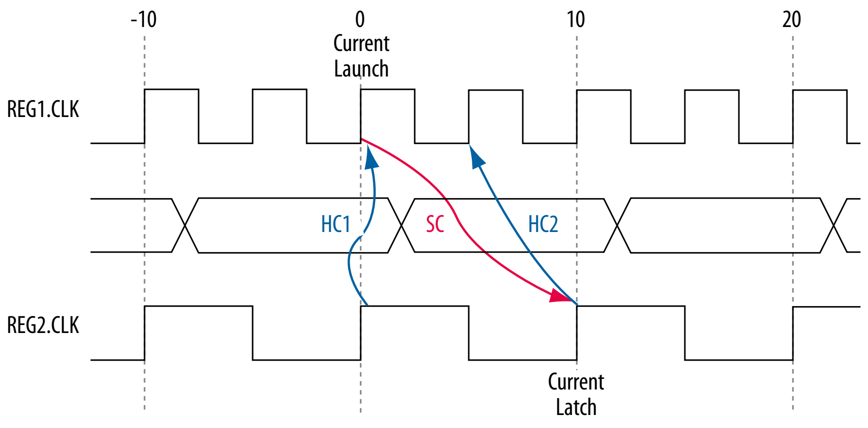


Figure 103. Hold Check Calculation

$$\begin{aligned}\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge}\\ &= 0\text{ ns} - 0\text{ ns}\\ &= 0\text{ ns}\end{aligned}$$

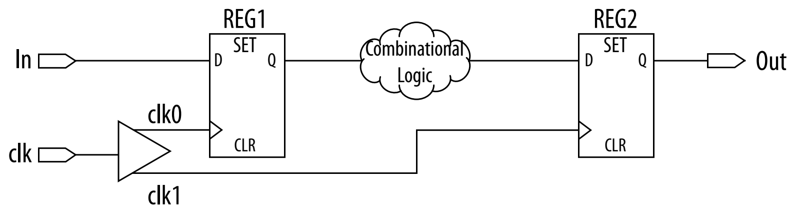
$$\begin{aligned}\text{hold check 2} &= \text{next launch edge} - \text{current latch edge}\\ &= 5\text{ ns} - 10\text{ ns}\\ &= -5\text{ ns}\end{aligned}$$

In this example, the hold check two is too restrictive. The data is launched next by the edge at 10 ns and must check against the data captured by the current latch edge at 10 ns, which does not occur in hold check two. To correct the default analysis, you use a start multicycle hold exception of one.

2.3.7.5.8. Source Clock Frequency is a Multiple of the Destination Clock Frequency with an Offset

In this example, the source clock frequency is an integer multiple of the destination clock frequency and the destination clock has a positive phase offset. The source clock frequency is 5 ns and destination clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The source clock frequency can be an integer multiple of the destination clock frequency with an offset when a PLL generates both clocks with different multiplication.

Figure 104. Source Clock Frequency is Multiple of Destination Clock Frequency with Offset



The following timing diagram shows the default setup check analysis the Timing Analyzer performs:

Figure 105. Setup Timing Diagram

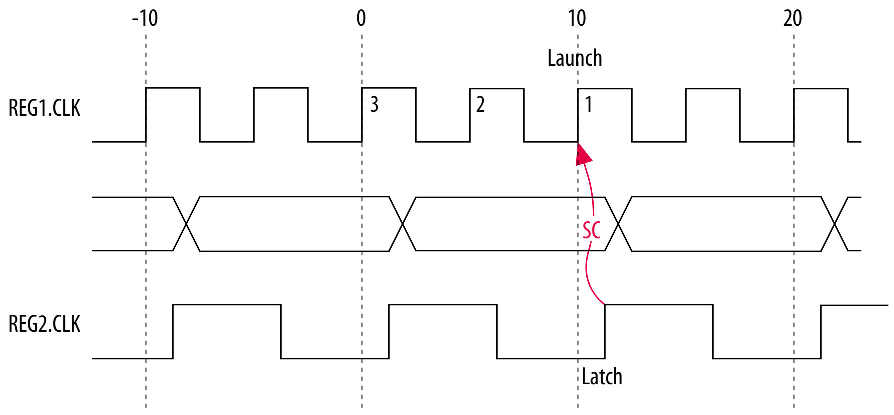


Figure 106. Setup Check Calculation

$$\begin{aligned}\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge}\\ &= 12\text{ ns} - 10\text{ ns}\\ &= 2\text{ ns}\end{aligned}$$

The setup relationship in this example demonstrates that the data is not launched at edge one, and the data that is launched at edge three must be captured; therefore, you can relax the setup requirement. To correct the default analysis, you shift the launch edge by two clock periods with a start multicycle setup exception of three.

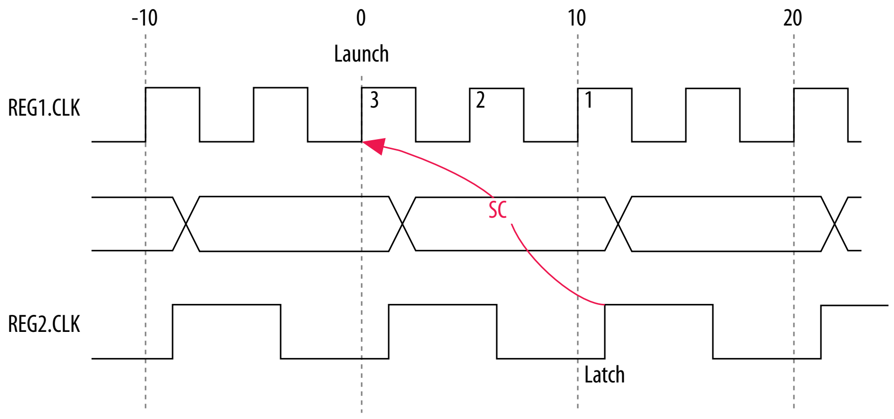
The following multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 3
```

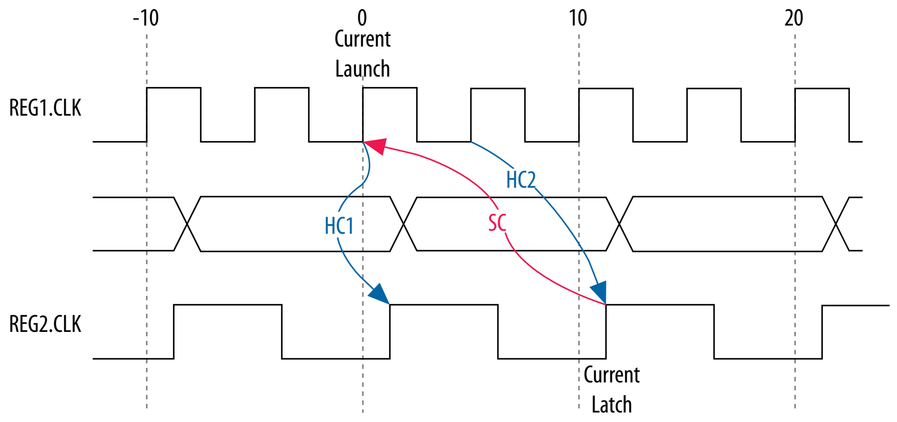
The following timing diagram shows the preferred setup relationship for this example:

Figure 107. Preferred Setup Check Analysis



The following timing diagram shows the default hold check analysis the Timing Analyzer performs for a start multicycle setup value of three:

Figure 108. Default Hold Check Analysis



The Timing Analyzer performs the following calculation to determine the hold check:

Figure 109. Hold Check Calculation

$$\begin{aligned}
\text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\
&= 0\text{ ns} - 2\text{ ns} \\
&= -2\text{ ns}
\end{aligned}$$

$$\begin{aligned}
\text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\
&= 5\text{ ns} - 12\text{ ns} \\
&= -7\text{ ns}
\end{aligned}$$

In this example, the hold check two is too restrictive. The data is launched next by the edge at 10 ns and must check against the data captured by the current latch edge at 12 ns, which does not occur in hold check two. To correct the default analysis, you must specify a multicycle hold exception of one.

2.3.7.6. Delay Annotation

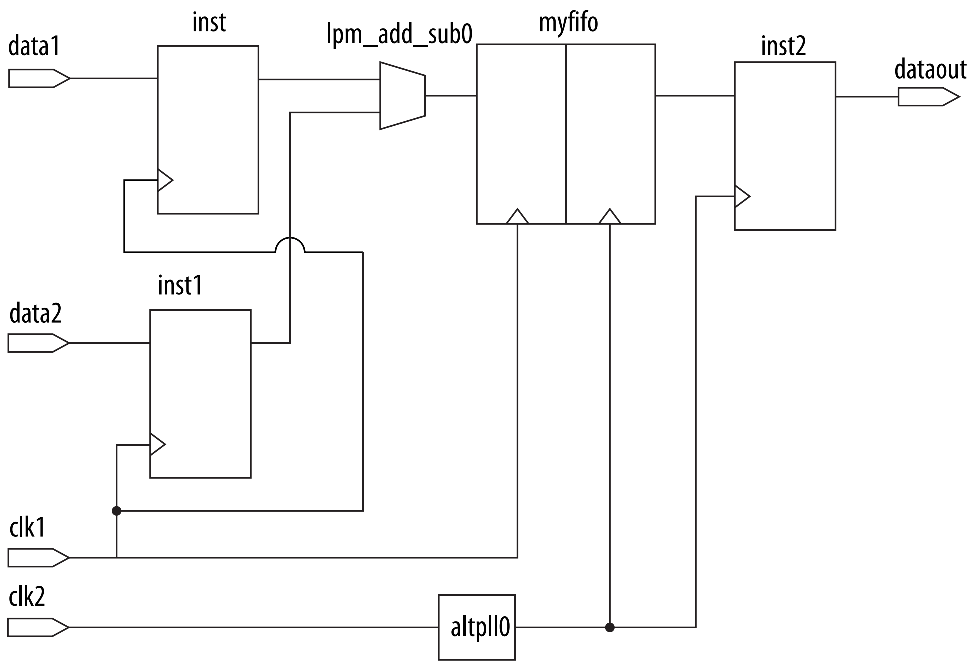
To modify the default delay values used during timing analysis, use the `set_annotated_delay` and `set_timing_derate` commands. You must update the timing netlist to apply these commands.

To specify different operating conditions in a single `.sdc` file, rather than having multiple `.sdc` files that specify different operating conditions, use the `set_annotated_delay -operating_conditions` command.

2.3.8. Example Circuit and SDC File

The following circuit and corresponding `.sdc` file demonstrates constraining a design that includes two clocks, a phase-locked loop (PLL), and other common synchronous design elements.

Figure 110. Dual-Clock Design Constraint Example



The .sdc file contains basic constraints for the example circuit.

Example 7. Basic .sdc Constraints Example

```
# Create clock constraints
create_clock -name clockone -period 10.000 [get_ports {clk1}]
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
# Create virtual clocks for input and output delay constraints
create clock -name clockone_ext -period 10.000
create clock -name clocktwo_ext -period 10.000
derive_pll_clocks
# derive clock uncertainty
derive_clock_uncertainty
# Specify that clockone and clocktwo are unrelated by assigning
# them to separate asynchronous groups
set_clock_groups \
  -asynchronous \
  -group {clockone} \
  -group {clocktwo altpll0|altpll_component|auto_generated|pll1|clk[0]}
# set input and output delays
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data1}]\
     set_input_delay -clock { clockone_ext } -min -1 [get_ports {data1}]
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data2}]\
     set_input_delay -clock { clockone_ext } -min -1 [get_ports {data2}]
set_output_delay -clock { clocktwo_ext } -max 6 [get_ports {dataout}]
set_output_delay -clock { clocktwo_ext } -min -3 [get_ports {dataout}]
```

The `.sdc` file contains the following basic constraints that you typically include for most designs:

- Definitions of `clockone` and `clocktwo` as base clocks, and assignment of those constraints to nodes in the design.
- Definitions of `clockone_ext` and `clocktwo_ext` as virtual clocks, which represent clocks driving external devices interfacing with the FPGA.
- Automated derivation of generated clocks on PLL outputs.
- Derivation of clock uncertainty.
- Specification of two clock groups, the first containing `clockone` and its related clocks, the second containing `clocktwo` and its related clocks, and the third group containing the output of the PLL. This specification overrides the default analysis of all clocks in the design as related to each other.
- Specification of input and output delays for the design.

Related Information

Asynchronous Clock Groups (-asynchronous) on page 46

2.4. Timing Analyzer Tcl Commands

You can optionally use Tcl commands from the Intel Quartus Prime software Tcl Application Programming Interface (API) to constrain, analyze, and collect timing information for your design. This section describes running the Timing Analyzer and setting constraints using Tcl commands. You can alternatively perform these same functions in the Timing Analyzer GUI. Tcl `.sdc` extensions provide additional methods for controlling timing analysis and reporting. The following Tcl packages support the Tcl timing analysis commands this chapter describes:

- `::quartus::sta`
- `::quartus::sdc`
- `::quartus::sdc_ext`

2.4.1. The quartus_sta Executable

The `quartus_sta` executable allows you to run timing analysis without running the full Intel Quartus Prime software GUI. The following methods are available:

- To run the Timing Analyzer as a stand-alone GUI application, type the following at the command prompt:

```
quartus_staw
```

- To run the Timing Analyzer in interactive command-shell mode, type the following at the command prompt:

```
quartus_sta -s <options><project_name>
```

- To run multi-corner timing analysis from a system command prompt, type the following command:

```
quartus_sta <options><project_name>
```

Table 15. quartus_sta Command-Line Options

Command-Line Option	Description
`-h | --help`	Provides help information on `quartus_sta`.
`-t <`*`script file`*`> | --script=<`*`script file`*`>`	Sources the <*script file*>.
`-s | --shell`	Enters shell mode.
`--tcl_eval <`*`tcl command`*`>`	Evaluates the Tcl command <*tcl command*>.

`--do_report_timing`	For all clocks in the design, run the following commands:
	`report_timing -npaths 1 -to_clock $clock`
	`report_timing -setup -npaths 1 -to_clock $clock`
	`report_timing -hold -npaths 1 -to_clock $clock`
	`report_timing -recovery -npaths 1 -to_clock $clock`
	`report_timing -removal -npaths 1 -to_clock $clock`

`--force_dat`	Forces an update of the project database with new delay information.
`--lower_priority`	Lowers the computing priority of the `quartus_sta` process.
`--post_map`	Uses the post-map database results.
`--sdc=<`*`SDC file`*`>`	Specifies the `.sdc` file to use.
`--report_script=<`*`custom script`*`>`	Specifies a custom report script to call.
`--speed=<`*`value`*`>`	Specifies the device speed grade used for timing analysis.
`--tq2pt`	Generates temporary files to convert the Timing Analyzer`.sdc` file to a PrimeTime `.sdc` file.
`-f <`*`argument file`*`>`	Specifies a file containing additional command-line arguments.
`-c <`*`revision name`*`> | --rev=<`*`revision_name`*`>`	Specifies which revision and its associated Intel Quartus Prime Settings File (`.qsf`) to use.
`--multicorner`	Specifies that the Timing Analyzer generates all slack summary reports for both slow- and fast-corners.
`--multicorner[=on|off]`	Turns off multicorner timing analysis.
`--voltage=<`*`value_in_mV`*`>`	Specifies the device voltage, in mV used for timing analysis.
`--temperature=<`*`value_in_C`*`>`	Specifies the device temperature in degrees Celsius, used for timing analysis.
`--parallel [=<`*`num_processors`*`>]`	Specifies the number of computer processors to use on a multiprocessor system.
`--64bit`	Enables 64-bit version of the executable.

2.4.2. Collection Commands

The Timing Analyzer supports collection commands that provide easy access to ports, pins, cells, or nodes in the design. Use collection commands with any constraints or Tcl commands specified in the Timing Analyzer.

Table 16. Collection Commands

Command	Collection Returned
`all_clocks`	All clocks in the design
`all_inputs`	All input ports in the design.
`all_outputs`	All output ports in the design.
`all_registers`	All registers in the design.
`get_cells`	Cells in the design. All cell names in the collection match the specified pattern. Wildcards can be used to select multiple cells at the same time.
`get_clocks`	Lists clocks in the design. When used as an argument to another command, such as the `-from` or `-to` of `set_multicycle_path`, each node in the clock represents all nodes clocked by the clocks in the collection. The default uses the specific node (even if the node is a clock) as the target of a command.
`get_nets`	Nets in the design. All net names in the collection match the specified pattern. You can use wildcards to select multiple nets at the same time.
`get_pins`	Pins in the design. All pin names in the collection match the specified pattern. You can use wildcards to select multiple pins at the same time.
`get_ports`	All ports (design inputs and outputs) in the design.

You can also examine collections and experiment with collections using wildcards in the Timing Analyzer by clicking **Name Finder** from the **View** menu.

2.4.2.1. Wildcard Characters

To apply constraints to many nodes in a design, use the "*" and "?" wildcard characters. The "*" wildcard character matches any string; the "?" wildcard character matches any single character.

If you apply a constraint to node `reg*`, the Timing Analyzer searches for and applies the constraint to all design nodes that match the prefix `reg` with any number of following characters, such as `reg`, `reg1`, `reg[2]`, `regbank`, and `reg12bank`.

If you apply a constraint to a node specified as `reg?`, the Timing Analyzer searches and applies the constraint to all design nodes that match the prefix `reg` and any single character following; for example, `reg1`, `rega`, and `reg4`.

2.4.2.2. Adding and Removing Collection Items

Wildcards that you use with collection commands define collection items that the command identifies. For example, if a design contains registers with the name `src0`, `src1`, `src2`, and `dst0`, the collection command `[get_registers src*]` identifies registers `src0`, `src1`, and `src2`, but not register `dst0`. To identify register `dst0`, you must use an additional command, `[get_registers dst*]`. To include `dst0`, you can also specify a collection command `[get_registers {src* dst*}]`.

To modify collections, use the `add_to_collection` and `remove_from_collection` commands. The `add_to_collection` command allows you to add additional items to an existing collection.

add_to_collection Command

add_to_collection <*first collection*> <*second collection*>

Note: The add_to_collection command creates a new collection that is the union of the two collections you specify.

The remove_from_collection command allows you to remove items from an existing collection.

remove_from_collection Command

remove_from_collection <*first collection*> <*second collection*>

The following example shows use of add_to_collection to add items to a collection.

Adding Items to a Collection

```
#Setting up initial collection of registers
set regs1 [get_registers a*]
#Setting up initial collection of keepers
set kprs1 [get_keepers b*]
#Creating a new set of registers of $regs1 and $kprs1
set regs_union [add_to_collection $kprs1 $regs1]
#OR
#Creating a new set of registers of $regs1 and b*
#Note that the new collection appends only registers with name b*
# not all keepers
set regs_union [add_to_collection $regs1 b*]
```

In the Intel Quartus Prime software, keepers are I/O ports or registers. An .sdc file that includes get_keepers is incompatible with third-party timing analysis flows.

2.4.2.3. Query of Collections

You can display the contents of a collection with the query_collection command. Use the -report_format option to return the contents in a format of one element per line. The -list_format option returns the contents in a Tcl list.

```
query_collection -report_format -all $regs_union
```

Use the get_collection_size command to return the number of items the collection contains. If your collection is in a variable with the name col, use set num_items [get_collection_size $col] rather than set num_items [llength [query_collection -list_format $col]] for more efficiency.

2.4.2.4. Using the get_pins Command

The get_pins command supports options that control the matching behavior of the wildcard character (*). Depending on the combination of options you use, you can make the wildcard character (*) respect or ignore individual levels of hierarchy. The pipe character (|) indicates levels of hierarchy. By default, the wildcard character (*) matches only a single level of hierarchy.

These examples filter the following node and pin names to illustrate function:

- lvl (a hierarchy level with the name lvl)
- lvl|dataa (an input pin in the instance lvl)
- lvl|datab (an input pin in the instance lvl)

- `lvl|cnod` (a combinational node with the name `cnod` in the `lvl` instance)
- `lvl|cnod|datac` (an input pin to the combinational node with the name `cnod`)
- `lvl|cnod|datad` (an input pin to the combinational node `cnod`)

Table 17. Sample Search Strings and Search Results

Search String	Search Result
`get_pins *|dataa`	`lvl|dataa`
`get_pins *|datac`	*<empty>*(3)
`get_pins *|*|datac`	`lvl|cnod|datac`
`get_pins lvl*|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|*|datac`	*<empty>*(3)
`get_pins -hierarchical lvl|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|datac`	`lvl|cnod|datac`
`get_pins -hierarchical lvl|*|datac`	*<empty>*(3)
`get_pins -compatibility_mode *|datac`	`lvl|cnod|datac` (4)
`get_pins -compatibility_mode *|*|datac`	`lvl|cnod|datac`

The default method separates hierarchy levels of instances from nodes and pins with the pipe character (|). A match occurs when the levels of hierarchy match, and the string values including wildcards match the instance or pin names. For example, the command `get_pins <instance_name>|*|datac` returns all the `datac` pins for registers in a given instance. However, the command `get_pins *|datac` returns and empty collection because the levels of hierarchy do not match.

Use the `-hierarchical` matching scheme to return a collection of cells or pins in all hierarchies of your design.

For example, the command `get_pins -hierarchical *|datac` returns all the `datac` pins for all registers in your design. However, the command `get_pins -hierarchical *|*|datac` returns an empty collection because more than one pipe character (|) is not supported.

The `-compatibility_mode` option returns collections matching wildcard strings through any number of hierarchy levels. For example, an asterisk can match a pipe character when using `-compatibility_mode`.

(3) The search result is *<empty>* because the wildcard character (*) does not match more than one hierarchy level, that a pipe character (|) indicates, by default. This command matches any pin with the name `datac` in instances at the top level of the design.

(4) When you use `-compatibility_mode`, the Timing Analyzer does not treat pipe characters (|) as special characters when you use the characters with wildcards.

2.4.3. Identifying the Intel Quartus Prime Software Executable from the SDC File

To identify which Intel Quartus Prime software executable is currently running you can use the `$::TimingAnalyzerInfo(nameofexecutable)` variable from within an SDC file. This technique is most commonly used when you want to use an overconstraint to cause the Fitter to work harder on a particular path or set of paths in the design.

Identifying the Intel Quartus Prime Executable

```
#Identify which executable is running:
set current_exe $::TimingAnalyzerInfo(nameofexecutable)
if { [string equal $current_exe "quartus_fit"] } {
    #Apply .sdc assignments for Fitter executable here
} else {
    #Apply .sdc assignments for non-Fitter executables here
}
if { ! [string equal "quartus_sta" $::TimingAnalyzerInfo(nameofexecutable)] } {
    #Apply .sdc assignments for non-Timing Analyzer executables here
} else {
    #Apply .sdc assignments for Timing Analyzer executable here
}
```

Examples of different executable names are `quartus_map` for Analysis & Synthesis, `quartus_fit` for Fitter, and `quartus_sta` for the Timing Analyzer.

2.5. Timing Analysis of Imported Compilation Results

You can preserve the compilation results for your design as a version-compatible Quartus database file (`.qdb`) that you can open in a later version of the Intel Quartus Prime software without compatibility issues.

When you import and open the `.qdb` in a later version of software, you can run timing analysis on the imported compilation results without re-running the Compiler.

2.6. Using the Intel Quartus Prime Timing Analyzer Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.21	18.1.0	• Clarified constraint precedence in *Maximum Skew* and *Timing Constraint Precedence* topics.
2018.09.24	18.1.0	• Revised "Basic Timing Analysis Flow" section to add sequential step organization, update steps, and add supporting screenshots. • Retitled "SDC Constraint Creation Summary" to " Dual Clock SDC Example." • Retitled "Default Settings" to "Default Multicycle Analysis." • Retitled "SDC (Clock and Exception) Assignments on Blackbox Ports" to "Constraining Design Partition Ports."
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated information on using Intel Arria 10 devices with enhanced timing algorithms.

continued...

Document Version	Intel Quartus Prime Version	Changes
2015.05.04	15.0.0	Added and updated contents in support of new timing algorithms for Arria 10: • Enhanced Timing Analysis for Arria 10 • Maximum Skew (`set_max_skew` command) • Net Delay (`set_net_delay` command) • Create Generated Clocks (clock-as-data example)
2014.12.15	14.1.0	Major reorganization. Revised and added content to the following topic areas: • Timing Constraints • Create Clocks and Clock Constraints • Creating Generated Clocks • Creating Clock Groups • Clock Uncertainty • Running the Timing Analyzer • Generating Timing Reports • Understanding Results • Constraining and Analyzing with Tcl Commands
August 2014	14.0a10.0	Added command line compilation requirements for Arria 10 devices.
June 2014	14.0.0	• Minor updates. • Updated format.
November 2013	13.1.0	• Removed HardCopy device information.
June 2012	12.0.0	• Reorganized chapter. • Added "Creating a Constraint File from Intel Quartus Prime Templates with the Intel Quartus Prime Text Editor" section on creating an SDC constraints file with the **Insert Template** dialog box. • Added "Identifying the Intel Quartus Prime Software Executable from the SDC File" section. • Revised multicycle exceptions section.
November 2011	11.1.0	• Consolidated content from the Best Practices for the Intel Quartus Prime Timing Analyzer chapter. • Changed to new document template.
May 2011	11.0.0	• Updated to improve flow. Minor editorial updates.
December 2010	10.1.0	• Changed to new document template. • Revised and reorganized entire chapter. • Linked to Intel Quartus Prime Help.

continued...

Document Version	Intel Quartus Prime Version	Changes
July 2010	10.0.0	Updated to link to content on SDC commands and the Timing Analyzer GUI in Intel Quartus Prime Help.
November 2009	9.1.0	Updated for the Intel Quartus Prime software version 9.1, including: • Added information about commands for adding and removing items from collections • Added information about the set_timing_derate and report_skew commands • Added information about worst-case timing reporting • Minor editorial updates
November 2008	8.1.0	Updated for the Intel Quartus Prime software version 8.1, including: • Added the following sections: "set_net_delay" on page 7–42 "Annotated Delay" on page 7–49 "report_net_delay" on page 7–66 • Updated the descriptions of the `-append` and `-file` <*name*> options in tables throughout the chapter • Updated entire chapter using 8½" × 11" chapter template • Minor editorial updates

A. Intel Quartus Prime Standard Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Standard Edition FPGA design flow.

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Standard Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.

- Intel Quartus Prime Standard Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics*, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Standard Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Mentor Graphics*, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, Transceiver Toolkit, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Standard Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Mentor Graphics* and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.