Intel® Quartus® Prime Pro Edition User Guide

Timing Analyzer

Updated for Intel® Quartus® Prime Design Suite: **22.3**

Answers to Top FAQs:

Online Version
Send Feedback

UG-20140

ID: **683243**
Version: **2022.09.26**

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 user 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. What's New In This Version

- The *Report Register Statistics* command now includes some minor new features, as Report Register Statistics on page 60 describes.
- For change details, refer to the chapter revision histories in this document.

1.2. 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 I/O 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.

continued...

Term	Definition
Clock setup time	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.
Maximum or minimum delay constraint	A constraint that specifies timing path analysis with a non-default setup or hold relationship.
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.
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 number of clock cycles 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 sequential design nodes, such as the output of a register to the input of another register.

1.2.1. 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.

1.2.1.1. 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.

The following figures illustrate division of a simple design schematic into timing netlist delays.

Figure 1. Simple Design Schematic

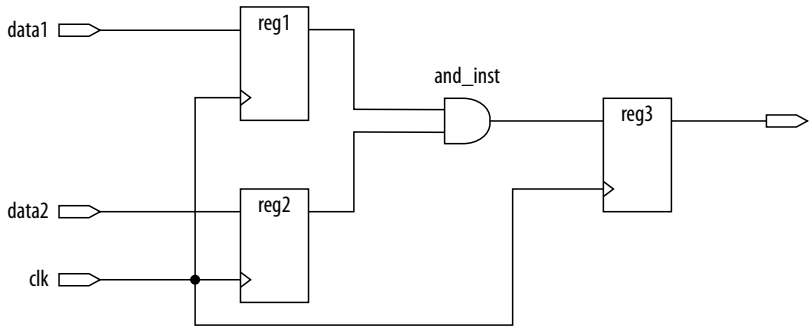
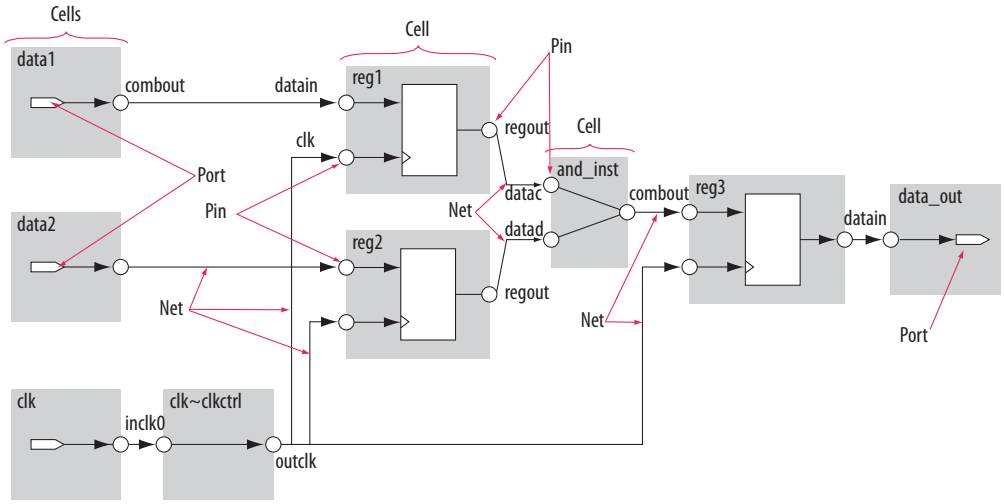


Figure 2. Division of Elements into Timing Netlist Delays



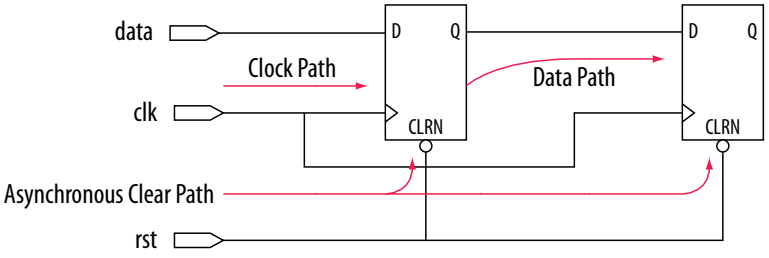
1.2.1.2. Timing Paths

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.

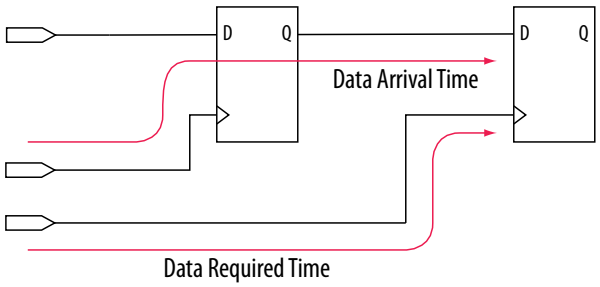
1.2.1.3. Data and Clock Arrival Times

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. It includes any clock port buffer delays and subtracts the micro setup time (μt_{SU}) (or adds the micro hold time) 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

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

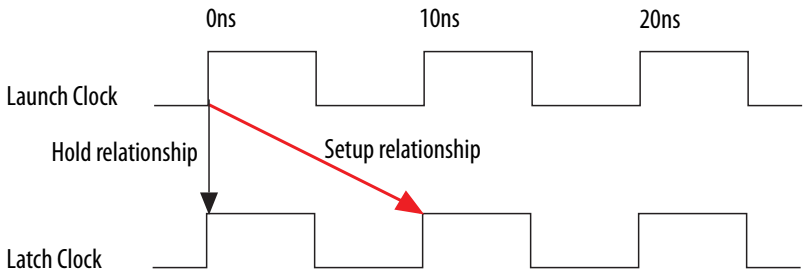
1.2.1.4. Launch and Latch Edges

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.



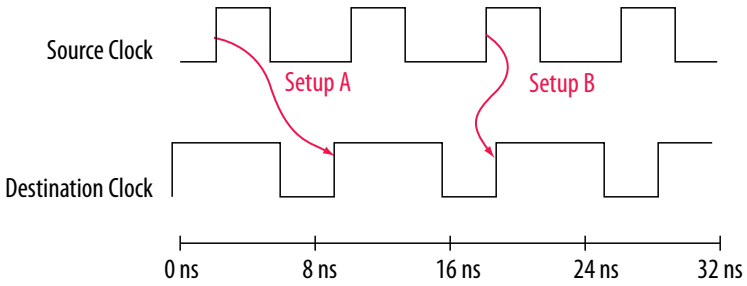
You must define all clocks in your design by assigning a clock constraint to each clock source node. These clock constraints provide the structure required for repeatable data relationships. If you do not constrain the clocks in your design, the Intel Quartus Prime software analyzes all clocks as 1 GHz clocks to maximize timing based Fitter effort. To ensure realistic slack values, you must constrain all clocks in your design with real values.

1.2.2. 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

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

The Timing Analyzer performs setup checks using the maximum delay when calculating data arrival time, and minimum delay when calculating data required time. Some of the spread between maximum arrival path delays and minimum required path delays may be recoverable with path pessimism removal, as Timing Pessimism on page 15 describes.

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

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay + Input Maximum Delay + Port-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

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

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Required Time	=	Latch Edge + Clock Network Delay to Output Port – Output Maximum Delay
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register +μt_{CO} + Register-to-Port Delay

1.2.3. 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

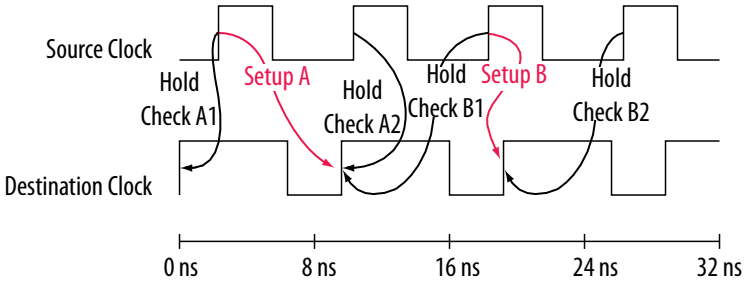


Figure 12. Clock Hold Slack for Internal Register-to-Register Paths

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

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

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

1.2.4. 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

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

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

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

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 to asynchronous control signals.

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

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

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

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

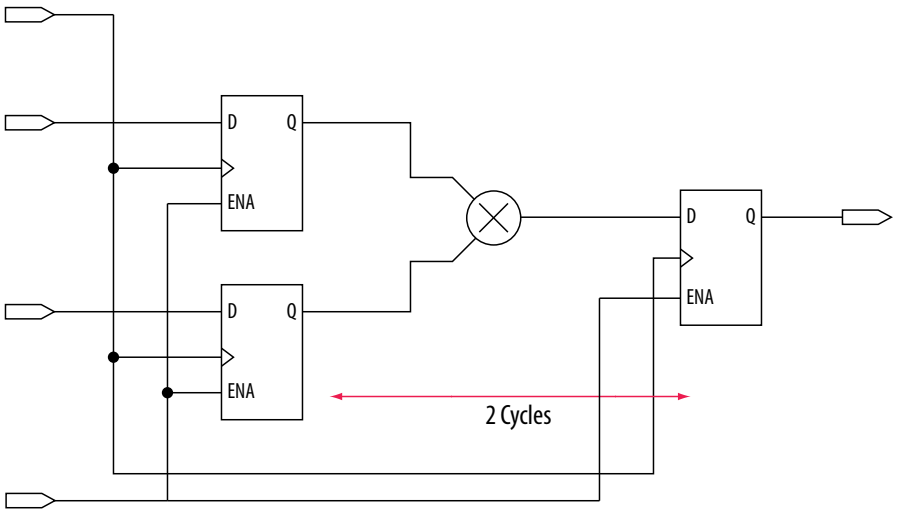
If the asynchronous reset signal is from a device pin, you must create an input delay constraint to the asynchronous reset pin for the Timing Analyzer to perform removal analysis on the path.

1.2.5. Multicycle Path Analysis

Multicycle paths are data paths that require an exception to the default setup or hold relationship, for proper analysis. For example, a register that requires data capture on every second or third rising clock edge (multicycle exception), rather than requiring capture on every clock edge (default analysis).

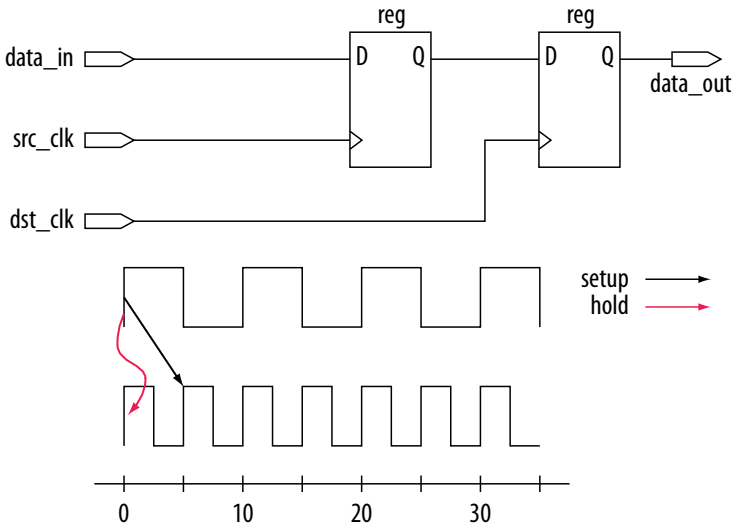
A multicycle path occurs between the input registers of a multiplier, and an output register with a destination that latches data on every other clock edge.

Figure 19. Multicycle Path



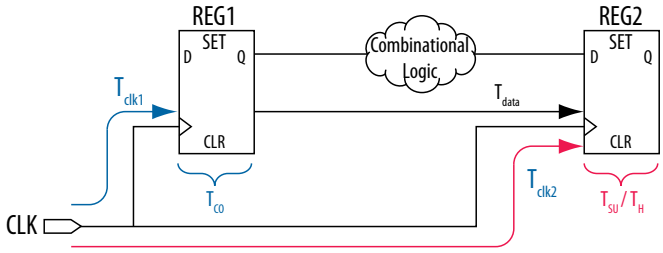
A register-to-register path is for the default setup and hold relationship. Also, for the respective timing diagrams for the source and destination clocks and the default setup and hold relationships, when the source clock, `src_clk`, has a period of 10 ns and the destination clock, `dst_clk`, has a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



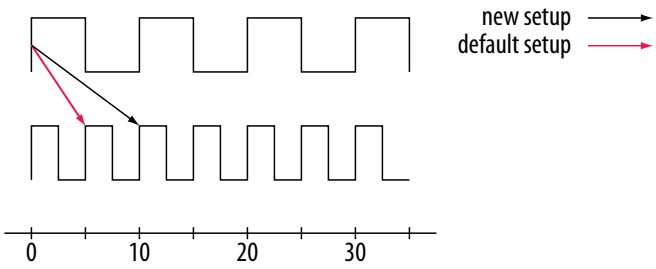
To accommodate the system requirements, you can modify the default setup and hold relationships by specifying a multicycle timing constraint to a register-to-register path.

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



1.2.5.1. 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. 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.

Figure 23. Hold Check

The Timing Analyzer uses the following calculation to determine the hold check.

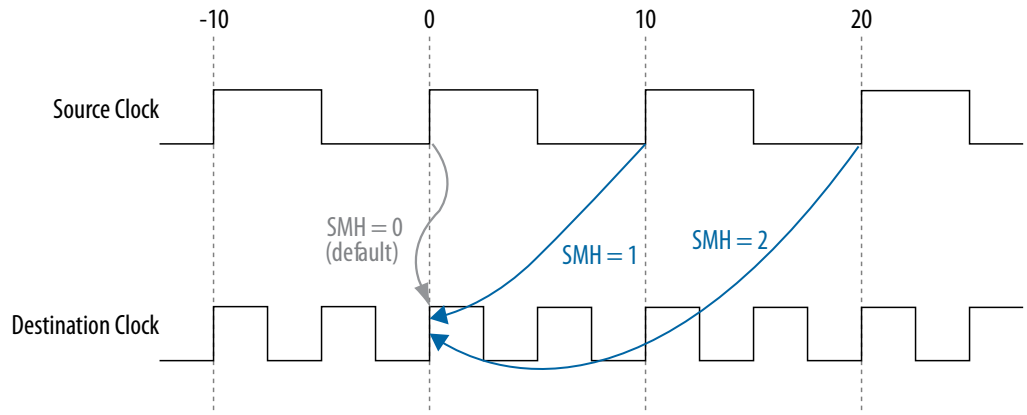
$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$

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

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 source clock by moving the launch 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

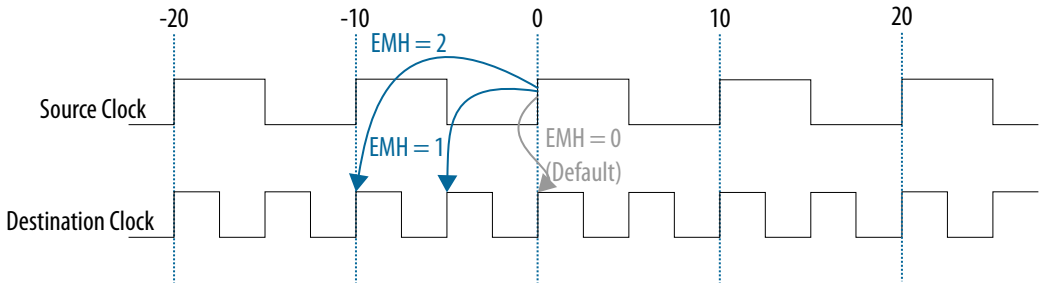
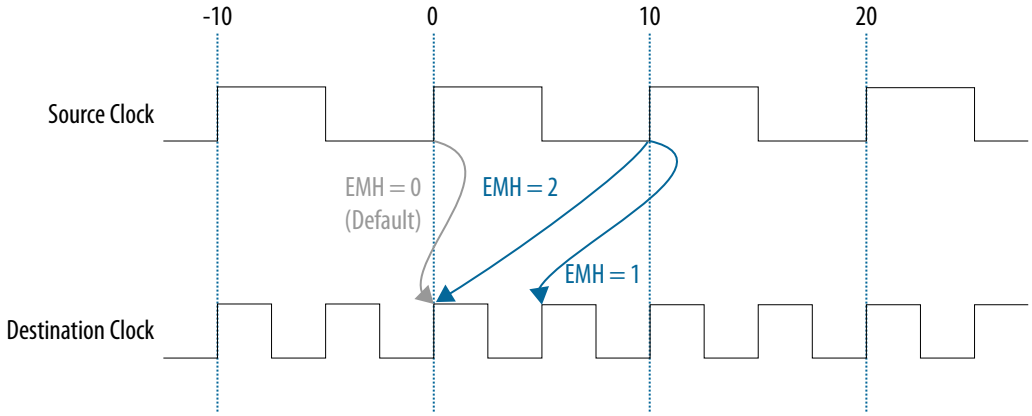


Figure 26. End Multicycle Hold Values the Timing Analyzer Reports

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

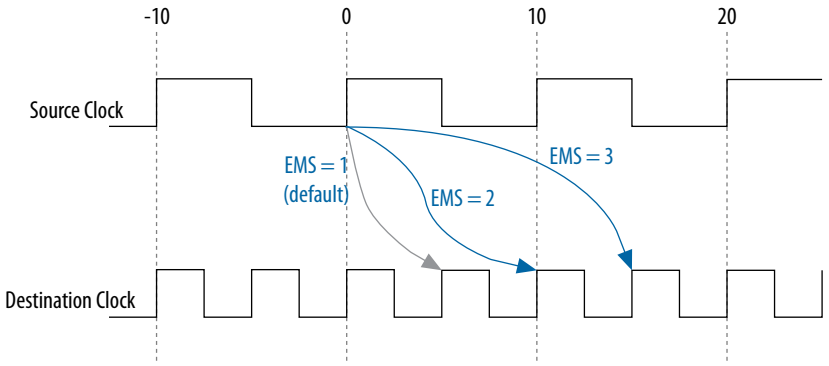


1.2.5.2. Multicycle Clock Setup

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

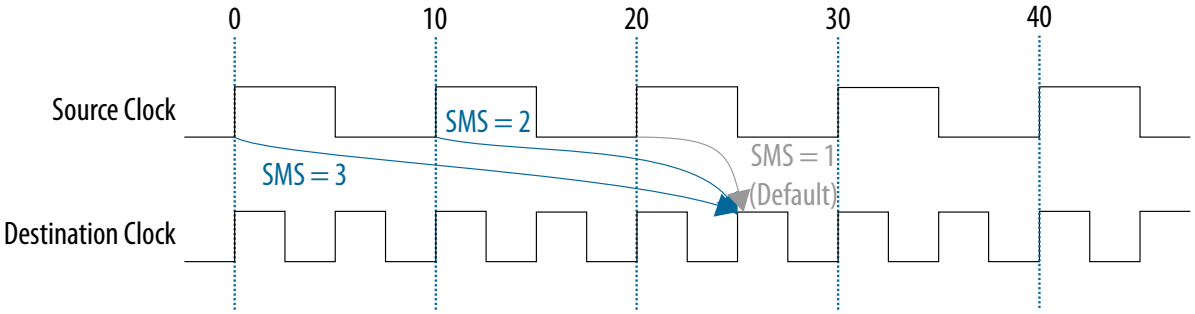
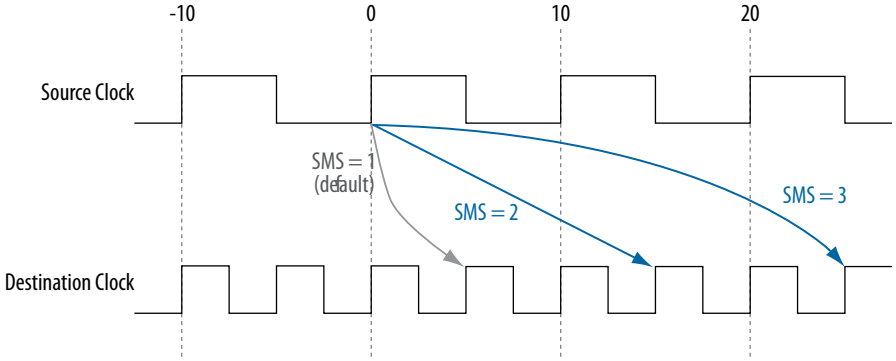


Figure 29. Start Multicycle Setup Values Reported by the Timing Analyzer

The following shows the negative setup relationship reported by the Timing Analyzer.



1.2.6. 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.

Related Information

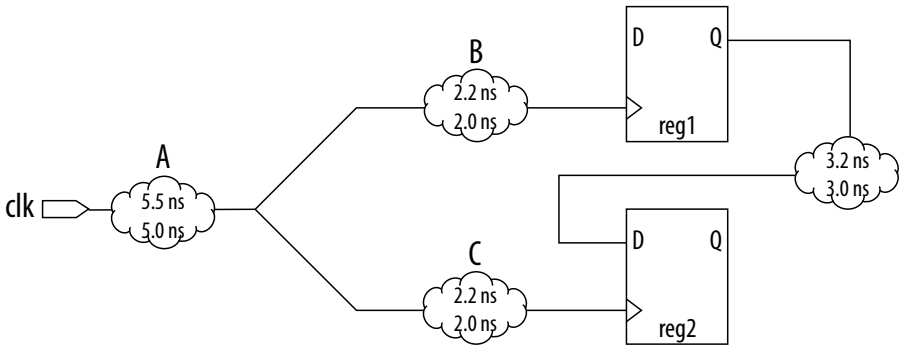
- Report Metastability on page 44
- Step 1: Specify Timing Analyzer Settings on page 24
- Understanding Metastability in FPGAs
- report_metastability

1.2.7. 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

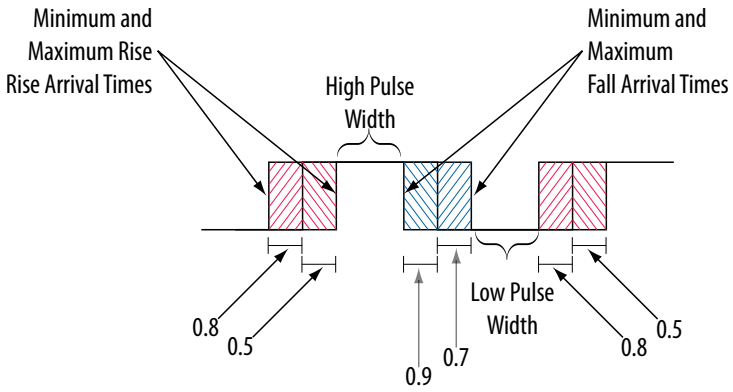


Segment A is the common clock path between `reg1` and `reg2`. The minimum delay is 5.0 ns; the maximum delay is 5.5 ns. The difference between the maximum and minimum delay value equals the common clock path pessimism removal value; in this case, the common clock path pessimism is 0.5 ns. The Timing 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).

1.2.8. Clock-As-Data Analysis

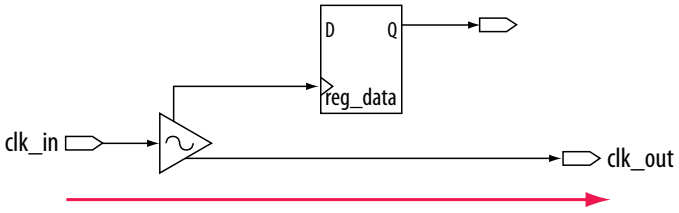
The majority of FPGA designs contain simple connections between any two nodes known as either a data path or a clock path.

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. Simplified Source Synchronous Output shows 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`.

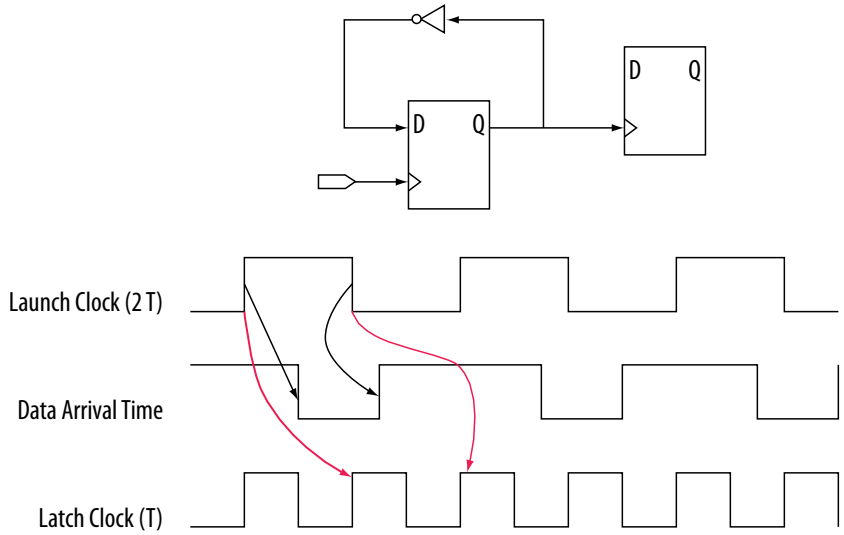
Figure 32. Simplified Source Synchronous Output



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. In the following figure, the waveforms are for the inverter feedback path, analyzed during timing analysis. The output of the divider register determines the launch time, and the clock port of the register determines the latch time.

Figure 33. Clock Divider



1.2.9. Multicorner Timing Analysis

You can direct the Timing Analyzer to perform multicorner timing analysis to verify your design under different voltage, process, and temperature operating conditions.

To ensure that no violations occur under different timing conditions (models) during device operation, you must perform static timing analysis under all available operating conditions.

You specify the operating conditions in the Timing Analyzer prior to running analysis.

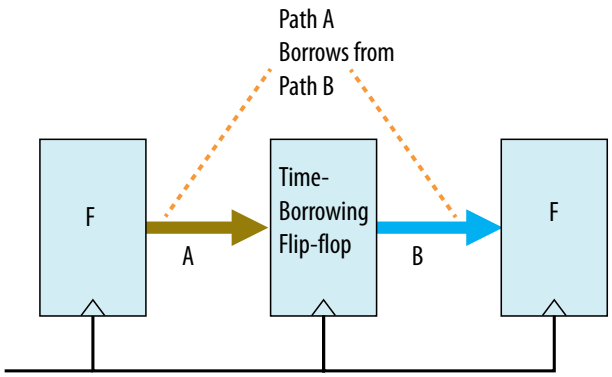
Related Information

Setting the Operating Conditions on page 28

1.2.10. Time Borrowing

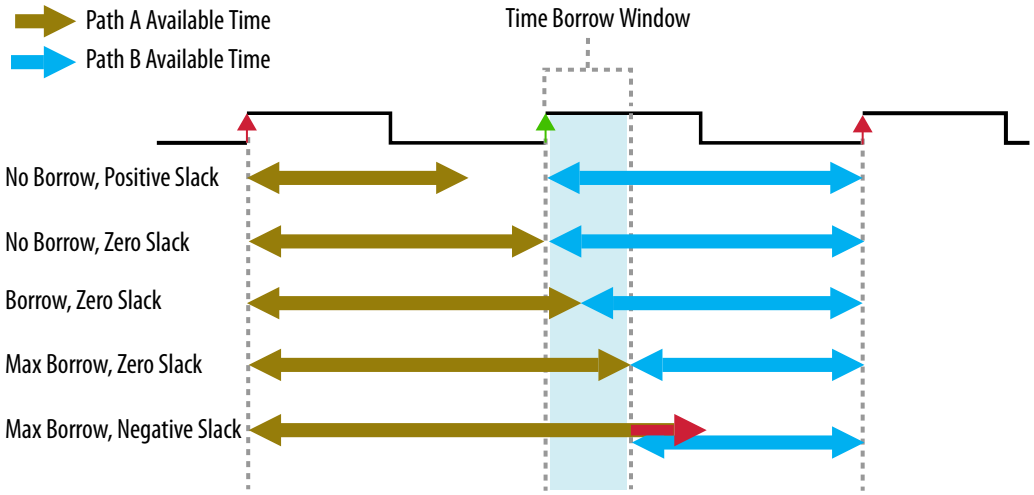
Time borrowing can improve performance by enabling the path ending at a time-borrowing flip-flop or latch to "borrow" time from the next path in the register pipeline. The borrowed time subtracts from the next path, resulting in the same cumulative timing. In this way, time borrowing can shift slack to more critical parts of the design. Without time borrowing, the Timing Analyzer analyzes each path independently and normally assumes exactly one clock cycle for each transfer.

Figure 34. Time Borrowing Example



Some of the flip-flops in Intel Stratix 10 and Intel Arria® 10 devices allow time borrowing. The exact size of the available time borrow window depends on hardware settings. The Fitter (Finalize) stage automatically configures the appropriate borrow window for each time-borrowing flip-flop, based on hardware restrictions and the available hold slack.(1)

Figure 35. Time Borrowing with Various Slack Conditions and Borrow Values



Intel FPGA devices generally support only a few borrow window sizes. For example, Intel Stratix 10 devices support narrow, medium, and wide. Typically, groups of several flip-flops must share the same setting. The actual borrowed amount is completely flexible within a given borrow window. The Timing Analyzer calculates the borrowed amount separately for each operating condition, clock, and signal rise and fall edge. Selecting a wider borrow window reduces hold slack. The Compiler only selects wider settings if hold slack allows. Furthermore, if the Compiler determines that a narrower window is sufficient for a given group of registers (based on the optimal time borrowing solution), the Compiler uses the narrower window, even if there is sufficient hold slack for a wider window.

(1) In Partial Reconfiguration designs, additional restrictions may apply to time-borrowing window size.

For a given borrow window size, the exact size of the borrow window may depend on the register input (for example, `d` or `sclr`), the edge of the incoming signal (rising or falling), the device speed grade, and operating conditions.

You can enable automatic implementation of time borrowing without making any RTL changes. Once enabled, the Fitter automatically configures the window size. The Fitter also determines the optimal time borrow amount within the available borrow window for any design registers that the Fitter places in time-borrowing flip-flops.

Proper timing analysis of designs that contain level-sensitive latches typically requires time borrowing. However, the automatic Fitter time borrowing optimizations do not apply for level-sensitive latches, as Time Borrowing with Latches on page 21 describes in detail.

Related Information

- Enabling Time Borrowing Optimization on page 29
- Report Time Borrowing Data on page 64

1.2.10.1. Time Borrowing Limitations

Time borrowing optimization, which occurs in the Fitter (Finalize) stage, cannot occur for the following registers. For these registers, the time borrowing optimization is zero, and the maximum operating frequency in the **Fmax Summary** reports include zero time borrowing:

- Any register that is the source of a cross-clock transfer
- Any register that is the source of a `set_max_skew` or `set_max_delay` assignment
- Any register in a clock domain with one or more level-sensitive latch

Furthermore, registers that are destinations of cross-clock transfers, `set_max_skew` or `set_max_delay` constraints do not have borrowing values that are optimized for such transfers (but may still have non-zero borrowing from other transfers).

If any such registers are on the critical timing path, you can possibly report better performance by enabling Dynamic time borrowing mode, which reports time borrowing for all borrow-capable registers. Dynamic time borrowing mode can then provide a more accurate (less pessimistic) analysis, but only at a specific set of clock frequencies that you specify in the `.sdc`.

To view time borrowing results for such registers:

1. Set the clock frequency in the `.sdc` file to a value higher than the clock frequency that **Fmax Summary** reports.
2. Reset your design and re-read the SDC file in the Timing Analyzer.
3. Run the `update_timing_netlist -dynamic_borrow` command.
4. View the results in **Slack Summary** reports (**Reports ➤ Slack** to determine if timing passes. The **Fmax Summary** report does not reflect any gains from dynamic borrowing.

Note: Time borrowing is similar to the beneficial clock skew technique, whereby you delay the clock to a given register, giving more time to the incoming paths at the expense of the outgoing paths. However, time borrowing and beneficial clock skew have the following important differences:

- Time borrowing offers more flexibility than clock skew in distributing setup slack. Skewing is typically limited to fixed increments. You can use borrowing to shift any amount of slack (however small) from one side of the register to the other, as long as the amounts fit within the available borrow window. Furthermore, borrow amounts calculate separately for each operating condition, making it possible to shift the optimum amount of slack for each operating condition. This type of shifting is not possible with skewing.
- You can use beneficial clock skew to increase Hold slack on outgoing paths from the register where the clock is skewed. Time borrowing does not offer this benefit.

1.2.10.2. Time Borrowing with Latches

The Intel Quartus Prime Timing Analyzer treats level-sensitive latches similar to registers. The Timing Analyzer treats the latch enable pin as a clock pin, while modifying the clock relationship appropriately.

You can run **Reports ➤ Constraint Diagnostics ➤ Check Timing** in the Timing Analyzer **Tasks** pane to view a list of the level-sensitive latches in your design.

Implementation of latch time borrowing requires that you enable Dynamic borrowing mode (`update_timing_netlist -dynamic_borrow`). Otherwise, the Timing Analyzer calculates zero time borrowing for latches. In Dynamic mode, the Timing Analyzer simply reports the amount of time borrowing that would physically happen in the circuit, given the clock frequencies you specify in SDC constraints, and does not actually optimize borrowing in any way.

For latches, the setup relationship is to the opening edge of the latch, which allows time borrowing. The hold relationship is to the closing edge of the latch. For example, a path from a positive register to another positive register has a default setup clock relationship of one clock period. A path from a positive register to a positive (open-high), level-sensitive latch has a default setup clock relationship of zero clock periods, plus any time borrow value.

The Timing Analyzer treats paths to and from a latch as two separate paths. For example, in a positive register--> positive latch--> negative register transfer, the Timing Analyzer does not analyze the overall register-->register transfer, even though you expect the latch to be transparent for the entire duration of the transfer. The Timing Analyzer analyzes and reports the paths to and from the latch separately.

The Timing Analyzer automatically computes the maximum amount of time borrowing available for each latch. Typically, the maximum amount of time borrowing available is roughly equivalent to half the clock period. The exact amount of time borrowing available is based on:

- The timing of opening and closing latch edges
- Physical latch implementation (closing-edge μt_{SU} of the latch)
- Clock uncertainty and other effects

The time borrowing never exceeds the maximum borrow value. However, you can specify a smaller maximum borrow time with the `set_max_time_borrow` SDC constraint. For example:

```
#Borrow at most 3ns at all "lat*" latches:
set_max_time_borrow 3 [get_registers lat*]
```

Specifying a clock with a negative borrow window can result in negative maximum borrowable time, which is equivalent to a minimum pulse width violation. For example, this condition can occur if half the clock period is smaller than the closing-edge μt_{SU} of the latch. If such a violation occurs, a warning indicates that the design cannot pass timing.

Note: Whether you use time borrowing or not, do not rely on the timing analysis **Fmax Summary** report for any clock domains with latches. The **Fmax Summary** values for such clock domains include no borrowing, and are therefore significantly pessimistic.

1.3. Timing Analysis Overview Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2022.03.28	22.1	• Added Top FAQ navigation to cover page. • Added new *What's New In This Version* topic.
2020.09.28	20.3	• Revised setup arrow direction in multiple timing diagrams for consistency.
2020.04.13	20.1	• Added "Time Borrowing" section.
2019.09.30	19.3	• Updated "MultiCorner Timing Analysis" code example and stated limitation for operating conditions.
2019.07.15	19.2	• Updated "MultiCorner Analysis" for SmartVID timing models.
2018.09.24	18.1	Minor text enhancements for clarity and style.

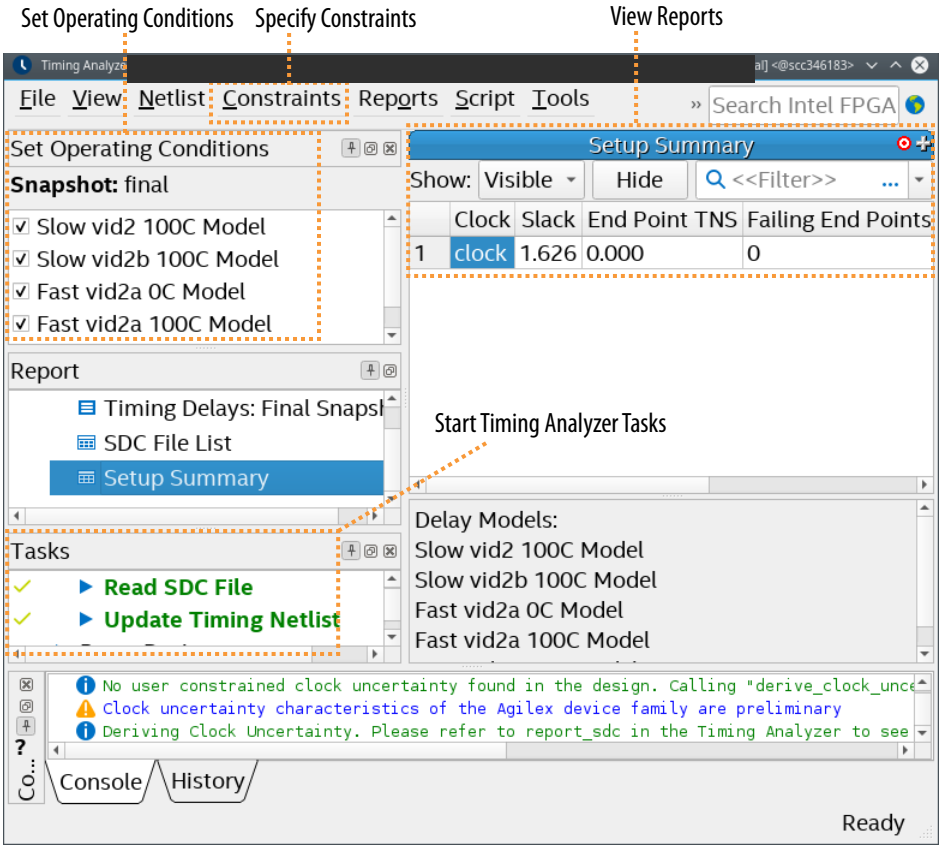
Table 2. Document Revision History

Date	Version	Changes
2016.10.31	16.1	• Implemented Intel rebranding.
2016.05.02	16.0	Corrected typo in Fig 6-14: Clock Hold Slack Calculation from Internal Register to Output Port
2015.11.02	15.1	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1	Moved Multicycle Clock Setup Check and Hold Check Analysis section from the Timing Analyzer chapter.
June 2014	14.0	Updated format
June 2012	12.0	Added social networking icons, minor text updates
November 2011	11.1	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.

Figure 36. Intel Quartus Prime Timing Analyzer



Related Information

- Timing Analyzer Quick-Start Tutorial: Intel Quartus Prime Pro Edition
- Intel FPGA Technical Training

2.1. Timing Analysis Flow

After creating your design and setting up a project, you define the required timing constraints for your design in a Synopsys* Design Constraints (.sdc) file. The Fitter then attempts to place logic to meet or exceed your constraints. The Timing Analyzer reports conditions that do not meet constraints, allowing you to correct critical timing issues. The following steps describe the basic timing analysis flow:

- Step 1: Specify Timing Analyzer Settings on page 24
- Step 2: Specify Timing Constraints on page 26
- Step 3: Run the Timing Analyzer on page 27
- Step 4: Analyze Timing Reports on page 31

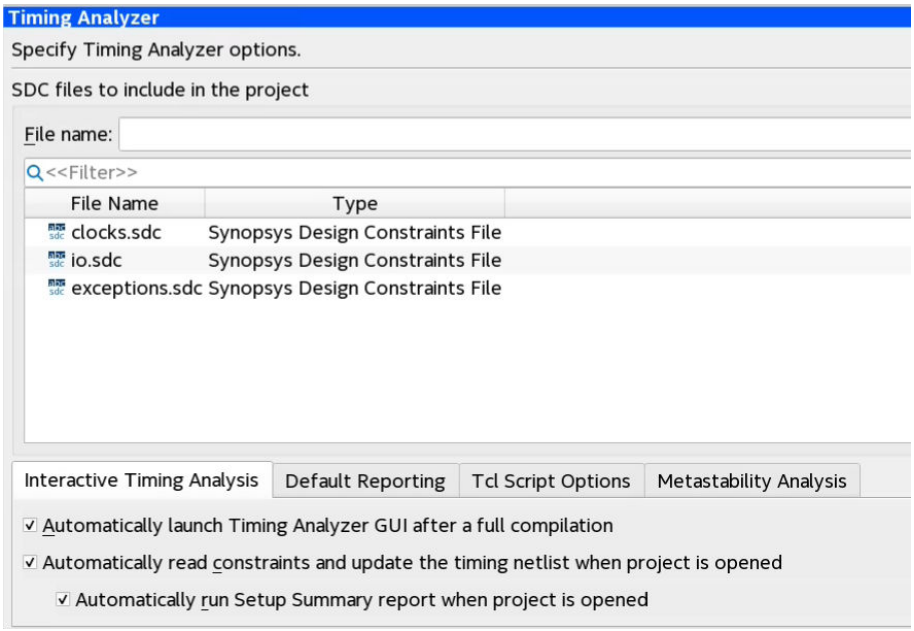
2.2. Step 1: Specify Timing Analyzer Settings

Before running timing analysis, you must open an Intel Quartus Prime project and run the Fitter to generate the timing netlist. You can consider and specify general settings for timing analysis, as well as other project-wide Compiler settings that impact the timing analysis results:

To specify general Timing Analyzer settings, follow these steps:

1. Click **File > New Project Wizard** to create a new project, or click **File ➤ Open Project** to open an existing project.
2. Click **Assignments ➤ Settings ➤ Timing Analyzer** to open the **Timing Analyzer** settings.

Figure 37. Timing Analyzer Page (Settings Dialog Box)



3. In the **Timing Analyzer** page, specify one or more .sdc file for timing analysis, and any of the following options:

Table 3. Timing Analyzer General Settings

Setting	Description
SDC files to include in the project	Specifies the name and processing order of Synopsis Design Constraint (`.sdc`) files in the project.
Interactive Timing Analysis	Specify options for automatically running timing analysis, reading constraints, and generating reports automatically. Turn on or off: • **Automatically launch Timing Analyzer GUI after a full compilation** (default, on) • **Automatically read constraints and update the timing netlist when project is opened** in Timing Analyzer (default, on) • **Automatically run setup summary report when project is opened** in Timing Analyzer (default, on)
Default Reporting	Specify options to automatically **Report worst-case paths during compilation** (default, on). Specify the **Paths reported per clock domain** (default, 10), and whether to **Show routing** (default, off) in reports.
Tcl Script Options	**Tcl Script File name** specifies the file name for a custom timing analysis script. You can specify whether to **Run default timing analysis before running custom script**.
Metastability Analysis	Specifies how the Timing Analyzer identifies registers as being part of a synchronization register chain for metastability analysis.

4. Consider and specify project-wide Compiler settings that can have a significant impact on Timing Analysis:

Table 4. Compiler Settings Impacting Timing Analysis

Setting	Description	Location
Enable multicorner support for Timing Analyzer and EDA Netlist Writer(default, on)	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 (default, Balanced)	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**
SDC Constraint Protection (default, off)	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 (default, 3)	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. 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.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
Optimize Design for Metastability (default, on)	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). 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)**

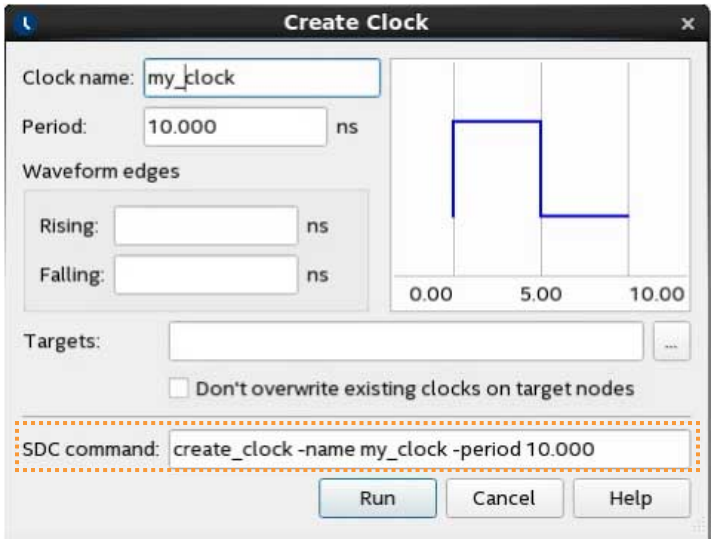
2.3. Step 2: Specify Timing Constraints

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** in the **Tasks** pane, 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 77, 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 38. Create Clock Dialog Defines Clock Constraints



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 1: Specify Timing Analyzer Settings on page 24 describes.

Related Information

- Using Entity-bound SDC Files on page 82
- Applying Timing Constraints on page 76

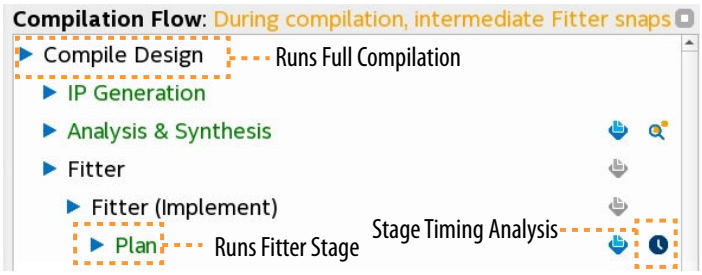
2.4. Step 3: Run the Timing Analyzer

You must run the Fitter to generate a timing netlist before running the Timing Analyzer. The Fitter attempts to place logic of your design to comply with the timing constraints that you specify. The Timing Analyzer then reports the margin (slack) by which your design meets or fails each constraint.

1. To generate the timing netlist and run the Timing Analyzer:
   - Click the Fitter's **Plan**, **Place**, **Route**, **Retime**, or **Fitter (Finalize)** stage from the Compilation Dashboard, and then click the stage's Timing Analyzer icon on the Compilation Dashboard. By default, the Timing Analyzer runs analysis on the latest available Fitter snapshot, and then opens the Setup Summary report automatically.

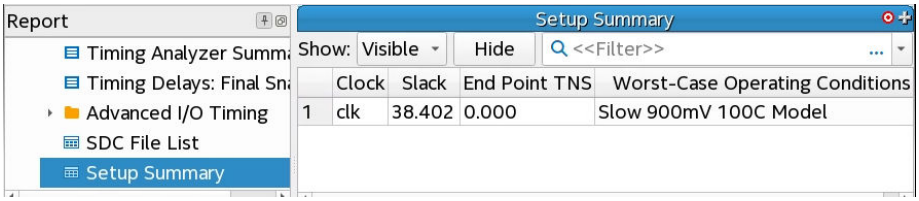
     Or
   - Run a full compilation by clicking **Compile Design** on the Compilation Dashboard. By default, the Timing Analyzer runs analysis with the final timing netlist, and then opens the Setup Summary report automatically.

Figure 39. Fitter Plan Stage Timing Analyzer Icon



2. Review the timing reports. To generate additional timing reports for analysis, click the **Reports** menu, and then click a **Slack**, **Datasheet**, **Diagnostic**, **Custom**, or **Design Metrics** timing report, as Step 4: Analyze Timing Reports on page 31 describes.

Figure 40. Setup Summary Report



3. To run timing analysis under different operation conditions, click **Set Operating Conditions** on the **Tasks** pane and specify options, as Setting the Operating Conditions on page 28 describes. By default, the Timing Analyzer generates reports for all supported operating conditions.
4. If you specify any settings or constraints that impact timing analysis, click **Update Timing Netlist** on the **Tasks** pane to apply the new constraints to the timing netlist.

Related Information

Timing Analyzer Tcl Commands on page 134

2.4.1. Setting the Operating Conditions

Click **View ➤ Timing Corners** in the Timing Analyzer to specify operating conditions for analysis under different power and temperature ranges. The available operating conditions vary by device and speed grade. The operating conditions represent the "timing corners" in a multi-corner timing analysis.

Select one or more voltage/temperature combinations and double-click **Report Timing...** under **Timing Slack** in the **Tasks** pane to configure the generation of timing analysis reports for that model. The generated report shows the worst-case timing path slack across all operating conditions that you specify. After generating the report for that model, the report shows the worst-case timing path slacks across all operating conditions that you select.

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

- **Regenerate**—regenerates the report you select.
- **Regenerate All Out of Date**—regenerates all reports that are outdated because of SDC changes since the last generation.
- **Delete All Out of Date**—removes all outdated report data.

The following operating conditions are available, depending on your target device and speed grade:

Table 5. Operating Conditions for Timing Analysis

Model	Description	Speed Grade	Voltage	Temperature
Slow 900mV 100C Model	High voltage, high temperature	Slowest speed grade in device density	V_{CC} minimum supply (1)	Maximum T_J (1)
Slow 900mV 0C Model	High voltage, low temperature	Slowest speed grade in device density	V_{CC} minimum supply (1)	Minimum T_J (1)
Fast 900mV 100C Model	High voltage, high temperature	Fastest speed grade in device density	V_{CC} maximum supply (1)	Maximum T_J (1)
Fast 900mV 0C Model	High voltage, low temperature	Fastest speed grade in device density	V_{CC} maximum supply (1)	Minimum T_J (1)
<*n*> Slow vid<*n*> 100C Model	High voltage, high temperature operating conditions for timing analysis with SmartVID.(2)	Slowest speed grade in device density	V_{CC} maximum supply (1)	Maximum T_J (1)
<*n*> Slow vid<*n*> 0C Model	High voltage, low temperature operating conditions for timing analysis with SmartVID.(2)	Fastest speed grade in device density	V_{CC} maximum supply (1)	Minimum T_J (1)
Slow vid<*n*> 0C Model(3)	High voltage, low temperature operating conditions for timing analysis with SmartVID.(2)	Slowest speed grade in device density	V_{CC} maximum supply (1)	Minimum T_J (1)
Slow vid<*n*> 100C Model(3)	High voltage, high temperature operating conditions for timing analysis with SmartVID.(2)	Slowest speed grade in device density	V_{CC} maximum supply (1)	Maximum T_J (1)

continued...

Model	Description	Speed Grade	Voltage	Temperature
Fast vid<*n*> 0C Model(3)	High voltage, low temperature for timing analysis with SmartVID. (2)	Fastest speed grade in device density.	V_{cc} maximum supply (1)	Minimum T_J (1)
Fast vid<*n*><*a*> 100C Model(4)	High voltage, high temperature for timing analysis with SmartVID. (2)	Fastest speed grade in device density.	V_{cc} maximum supply (1)	Maximum T_J (1)
	Note : 1. Refer to the applicable device Handbook for V_{cc} and T_J values . 2. Intel Stratix 10 and Intel Agilex™ SmartVID designs require this additional model to ensure complete coverage. 3. Intel Agilex devices only. 4. *n* refers to the device speed grade. Letters (a, b, and so on) following *n* refer to different timing corners that capture possible operating conditions. In general, you can consider these letters as somewhat arbitrary labels.			

As an alternative to the GUI, you can run the `set_operating_conditions` command with the `-model`, `-speed`, `-temperature`, and `-voltage` options to specify the operating conditions. When running `set_operating_conditions`, you must only specify valid operating conditions for the current device. Run the `get_available_operating_conditions` command to return a list of all valid operating conditions for the current device.

The following example sets the operating condition to the slow timing model, with a voltage of 1100 mV, and temperature of 85° C:

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

Related Information

Multicorner Timing Analysis on page 18

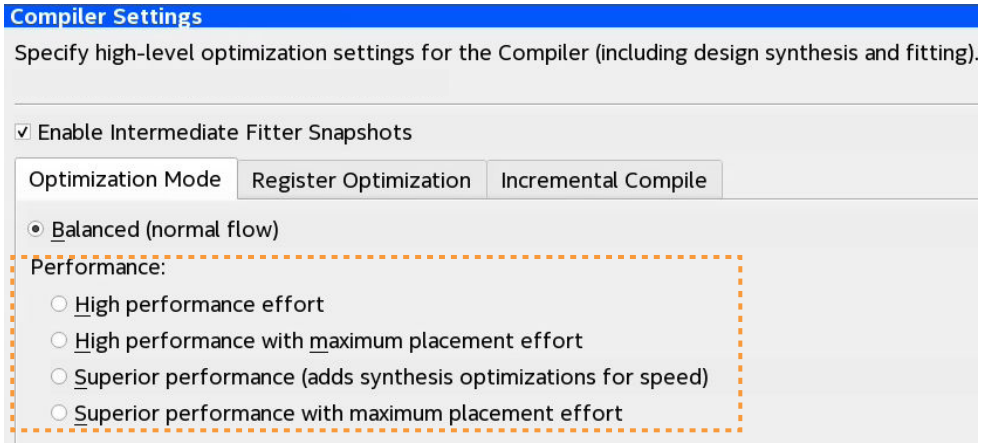
2.4.2. Enabling Time Borrowing Optimization

During any High or Superior **Performance** compilation, the Compiler automatically computes and stores **Optimal** time borrow values for Intel Stratix 10 and Intel Arria 10 designs during the Finalize stage. By default, the subsequent timing analysis results reflect the **Optimal** borrow values from the Finalize stage.

Follow these steps to enable time borrowing for supported devices:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Optimization Mode**. Select any high or superior **Performance** setting.
2. Run the Fitter and Timing Analyzer, as Step 3: Run the Timing Analyzer on page 27 describes.
3. To generate reports showing time borrowing data, click **Reports ➤ Timing Slack ➤ Report Timing**. Time borrowing data appears on the critical path for a given clock domain, as Report Time Borrowing Data on page 64 describes.

Figure 41. Performance Compiler Optimization Mode Settings



- To specify time borrowing optimization without changing the Compiler **Optimization Mode**, specify the following assignment in the project `.qsf`:

```
set_global_assignment -name ENABLE_TIME_BORROWING_OPTIMIZATION <ON|OFF>
```

- To manually specify the time borrow mode during timing analysis, run one of the following `update_timing_netlist` command options:

Table 6. Time Borrowing Modes

Time Borrowing Mode	Command Option	Default Mode For
Optimal—timing analysis includes optimal time borrow values from the Finalize stage. You can optionally add the `recompute_borrow` option to `update_timing_netlist` to recompute the borrow amounts, but not the borrow window sizes.	`update_timing_netlist`	High and Superior performance compilations for Intel Stratix 10 and Intel Arria 10 designs.
Dynamic—timing analysis reports the time borrowing that would physically occur on the device, with respect to your SDC constraints, without any optimization. That is, timing analysis applies as much borrowing as necessary to fix all negative slack. Timing analysis assumes maximum possible borrowing for any timing path where the maximum amount of time borrowing is insufficient to eliminate all negative slack. Only mode that allows borrowing for level-sensitive latches.	`update_timing_netlist -dynamic_borrow`	None
Zero—timing analysis uses zero time borrowing.	`update_timing_netlist –no_borrow`	Unsupported devices, or any Compiler Optimization mode other than a **Performance** mode.

Note: Dynamic mode cannot yield the optimal results with overconstrained clocks, as overconstrained clocks result in excessive negative slack on almost every path. This condition causes use of maximum time borrowing everywhere, which is unlikely to be optimal. When using Partial Reconfiguration, if you compile the base design with time borrowing enabled, compile the implementation design(s) with time borrowing enabled. Otherwise, time borrowing amounts in the base design are reset to zero, and the design may not pass timing. If this condition occurs, you can use the `update_timing_netlist –recompute_borrow` command to restore time borrowing amounts throughout the design.

Related Information

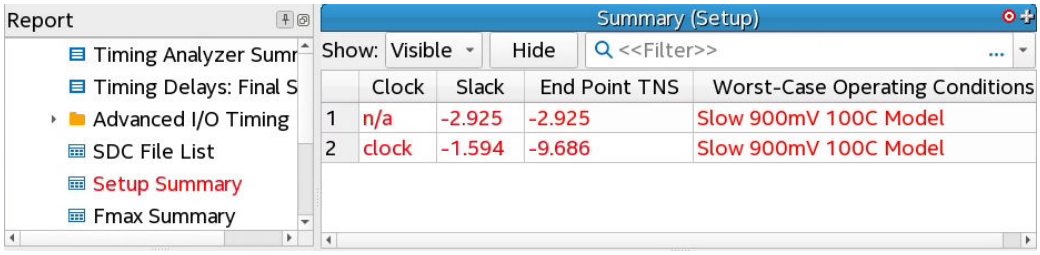
Time Borrowing on page 18

2.5. Step 4: Analyze Timing Reports

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 that timing requirements are met.

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 42. Timing Analyzer Shows Failing Paths in Red



Reports that indicate failing timing performance appear in red text, and 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.

2.5.1. Generating Timing Reports

The Timing Analyzer generates only a subset of all available reports by default, including the Setup Summary and Timing Analyzer Summary reports. However, you can generate dozens of other detailed reports in the Timing Analyzer GUI, or with command-line commands to help pin-point timing issues. You can customize the display of information in the reports.

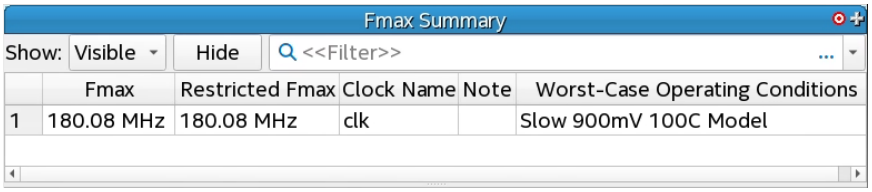
The following describe generation of various timing analysis reports:

Report Fmax Summary on page 33
Report Timing on page 34
Report Timing By Source Files on page 40
Report Data Delay on page 40
Report Net Delay on page 40
Report Clocks and Clock Network on page 41
Report Clock Transfers on page 43

2.5.1.1. Report Fmax Summary

The Timing Analyzer **Reports ➤ Datasheet ➤ Report Fmax Summary** command generates a report panel showing the potential maximum frequency for every clock in your design. The **Fmax** column reports the fastest frequency that your clock can run, and still pass `report_timing -setup -intra_clock` with a slack of `0`. The equivalent console command is `report_clock_fmax_summary`.

Figure 43. Fmax Summary Report



Note: The related `get_clock_fmax_info` command returns a Tcl list, which is useful for scripting and parsing. Refer to *get_clock_fmax_info (::quartus::sta)* in *Intel Quartus Prime Pro Edition User Guide Scripting*.

f_{MAX} is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, f_{MAX} is computed as if the rising and falling edges of the clock are scaled along with f_{MAX}, such that duty cycle (by percentage) is maintained.

However, the **Fmax** report does not indicate whether your design meets timing for recovery, removal, nor setup or hold without the `intra_clock` option. For these reasons, always make sure to view the Setup, Hold, Recovery, Removal, and Min Pulse Width slack summaries to determine whether your design meets timing.

The **Restricted Fmax** column reports the lesser of the following values:

- The fastest frequency that your clock can run, and still pass `report_timing -hold -intra_clock` with a slack of `0` or `report_min_pulse_width` with a slack of `0`.
- The **Fmax** column value.[(2)]

Restricted Fmax considers hold timing in addition to setup timing, as well as minimum pulse and minimum period restrictions. Similar to unrestricted f_{MAX}, the analysis computes the restricted f_{MAX} as if the rising and falling edges of the clock scale along with f_{MAX}, such that the duty cycle (in terms of a percentage) is maintained.

The **Restricted Fmax** may display text indicating any of the following limiting factors:

- Limit due to hold check
- Limit due to minimum pulse width restriction
- Limit due to high minimum pulse width restriction
- Limit due to low minimum pulse width restriction

(2) The **Restricted Fmax** column never reports a value higher than the **Fmax** column.

Typically, hold checks do not limit the maximum frequency (f_{MAX}) because the checks are for same-edge relationships, and therefore independent of clock frequency. For example, when launch equals zero and latch equals zero. However, with an inverted clock transfer, or a multicycle transfer, the hold relationship is not a same-edge transfer and changes with the clock frequency.

Refer to the timing reports, such as those that you can generate using `report_timing`, or using minimum pulse width reports via the `report_min_pulse_width` command for details of specific paths, registers, or ports.

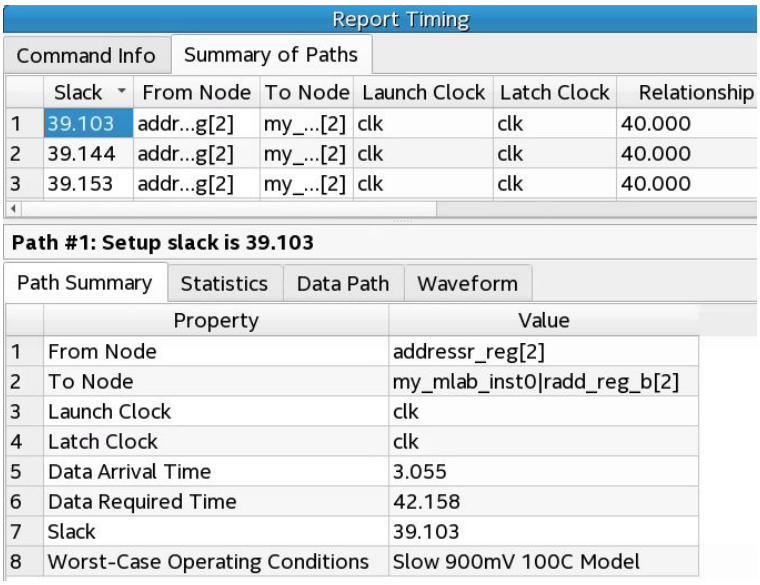
Related Information

Intel Quartus Prime Pro Edition User Guide: Scripting

2.5.1.2. Report Timing

The Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing...** command allows you to specify settings to report the timing of any path or clock domain in the design. The equivalent scripting command is `report_timing`.

Figure 44. Report Timing Report



You can specify various options to customize the reporting. You can specify the **Clocks** and **Targets** that the report displays, the **Analysis Type** to run, whether to display **Extra Info** in the report, and the **Output** options for the report. For example, you can increase the number of paths to report, add a **Target** filter, and add a **From Clock**.

Figure 45. Report Timing Dialog Box (Top Section)

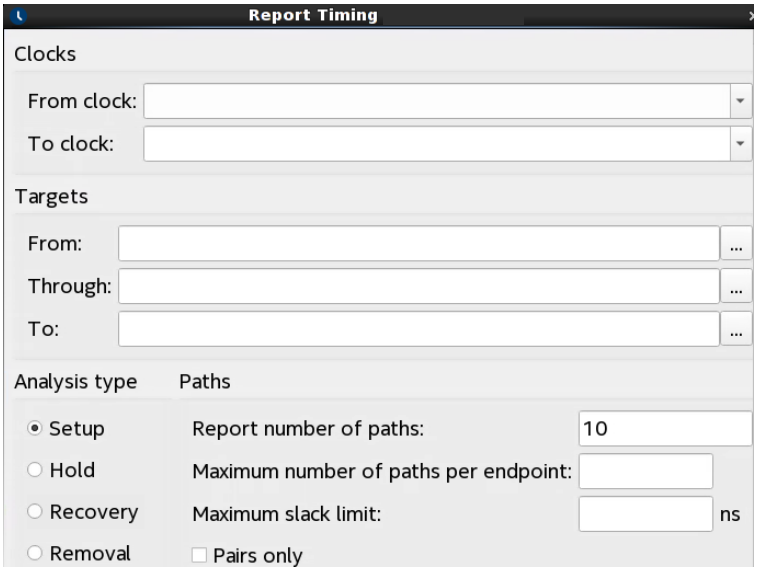


Figure 46. Report Timing Dialog Box (Bottom Section)

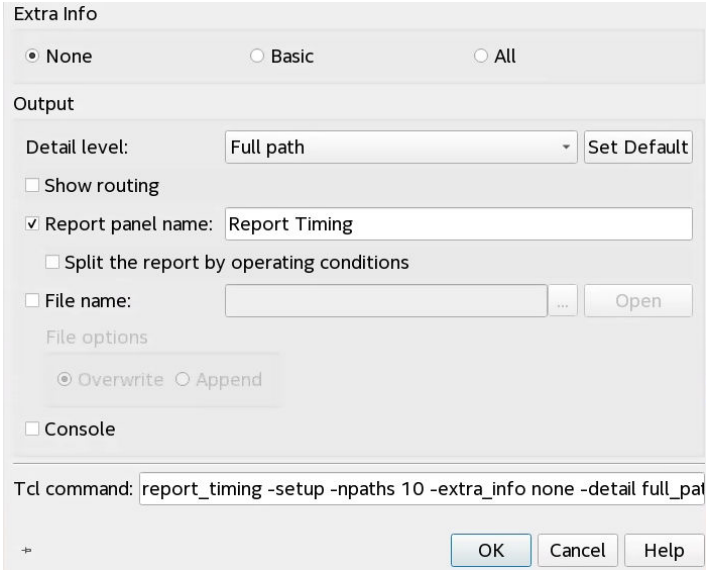


Table 7. Report Timing Settings

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)`

continued...

Option	Description
	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 to 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**. The Timing Analyzer reports the results for the type of analysis you select.
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.
Extra Info	Provides additional data that is relevant for diagnosing timing failure root cause, such as setup slack breakdown, and unexpected routing detours caused by congestion and hold time fix-up. Specify whether to include **None**, **Basic**, or **All** extra information in the report. The **Extra Info** tab data can help you identify potential, unnecessary routing detours, as well as placement or circuit issues that restrict the path f_{MAX} performance. Refer to Setup Slack Breakdown On the Extra Info Tab on page 37. • **All**—report includes **Extra Info** tab that reports extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block. The **Data Path** tab includes Estimated Delay Added for Hold and Route Stage Congestion Impact data. • **Basic**—report includes the **Extra Info** tab but no extra information on the **Data Path** tab. • **None**—report includes no **Extra Info** tab or other extra information on the **Data Path** tab.
Output	Specify the path types the analysis includes in output for **Detail level**: • **Summary**—level includes basic summary reports. 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. • **Path only**—displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. • **Path and Clock**—displays the same as **Path only** with respect to the clock. • **Full path**—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.
Show routing	Shows routing data in the report.
Split the report by operating conditions	For the operating condition timing corners, subdivides the data by each operating condition.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.

Figure 47. Extra Info Tab

Path Summary | Statistics | Data Path | Extra Info | W

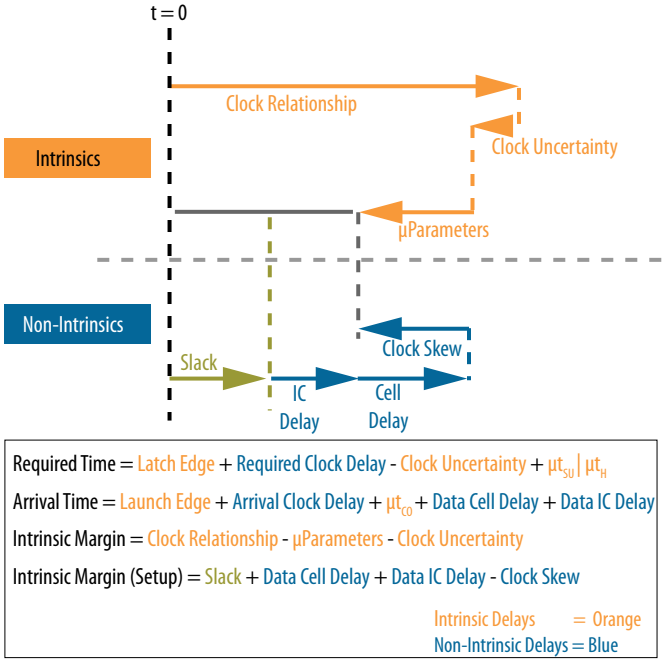
	Property	Value
1	Setup Slack Breakdown [=A+B-C-D]	9.259
1	[A] Intrinsic Margin [=a+b-c-d]	9.852
1	[a] Clock Edge...nship [=aa-bb]	10.000
2	[b] Clock Uncertainty	-0.030
3	[c] uTco	0.212
4	[d] uTsu	-0.094
2	[B] Clock Skew	-0.001
3	[C] Cell Delay	0.589
4	[D] Interconnect Delay	0.003
2	From Node Info	
1	Type	ALM Register
2	Retiming Restriction	--
3	Power-Up "Don't Care"	Yes
3	To Node Info	
4	Interconnect Info	
1	Max Fanout	4
2	Route Stage Congestion Impact	--
3	Estimated Delay Added for Hold	0.000
4	Sufficient Setup Margin for Hold	Yes
5	Bounding Box Info	
1	Source/Destination Bounding Box	--
2	Source/Destination Area Covered	0
3	Source/Destination Relative Area	0.000
4	Cell Bounding Box	--
5	Cell Area Covered	0
6	Cell Relative Area	0.000

Setup Slack Breakdown On the Extra Info Tab

The **Extra Info** tab contains other timing metrics to help you diagnose timing closure issues, including **Setup Slack Breakdown** for the path.

The slack of a path specifies the margin by which the path meets its timing requirement. The setup slack breakdown is a numeric value that the Timing Analyzer calculates from the following timing requirements and path element delays:

Figure 48. Setup Slack Breakdown Calculations



A path can fail timing requirements for many different reasons. For example, the clock relationship can be impossibly tight, or there can be excessive routing delays that alone cause failure for the timing path. Calculating the intrinsic margin of a timing path, and then comparing that margin to other delays of the path, can help identify the specific reasons why a path fails its timing requirement.

The **Extra Info** tab can help you identify potential significant or unexpected routing detours caused by congestion and hold time fix-up. The **Extra Info** tab can also report extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block.

You can review the **Extra Info** data and **Locate Path** or **Locate Chip Area** in Chip Planner, Technology Map Viewer, or Resource Property Viewer to determine whether to make changes to improve placement and routing.

Some delay elements are more sensitive to a path's placement and routing than others. Intrinsic delays that are part of **Setup Slack Breakdown** are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.

Table 8. Extra Info Tab Data

Extra Info Data	Description
Intrinsic Margin	Reports the intrinsic and non-intrinsic timing elements that comprise the timing path slack value. Intrinsic margin is a numeric value that the Timing Analyzer calculates from the timing requirements and path element delays. The Timing Analyzer also derives the slack of the path from the same requirements and delays, but with a different calculation. Intrinsic delays are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.
From Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the From Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
To Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the To Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
Max Fanout	Reports the maximum fan-out of register and combinational nodes in the path.
Route Stage Congestion Impact	Reports whether routing has a **Low**, **Medium**, or **High** impact on congestion. A **Low** value suggests timing issues are not congestion related. A **High** value suggests competition for scarce routing resources plays a role in poor timing.
Estimated Delay Added for Hold	Reports the estimated amount of delay added on to the fastest delay route to satisfy hold. This value can help you determine whether delays are routing congestion or Hold related.
Sufficient Setup Margin for Hold	Reports whether the setup margin is suitable for the hold timing. **Yes**, indicates that the setup margin is sufficient. **No** indicates that the setup margin is insufficient for hold timing.
Source/Destination Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers. In an ideal case, the **Source/Destination Bounding Box**, **Cell Bounding Box**, and **Interconnect Bounding Box** values are roughly the same, and the relative areas are approximately 1.0. If the cell bounding box size grows relative to the **Source/Destination Bounding Box**, that can indicate a potential unnecessary routing detour on the path.
Source/Destination Area Covered	Reports the total area covered in terms of LABs.
Source/Destination Relative Area	Reports the area for the source and destination, relative to the **Source/Destination Bounding Box**. The value is always 1.0, which equals the same size.
Cell Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers, and any cells in the path.
Cell Area Covered	Reports the area for the cell, relative to the **Source/Destination Bounding Box**. A value of 1.0 equals the same size. A value greater than 1.0 can indicate a path has a cell outside of the space between the registers in the path.

The following describe the interpretation of timing conditions indicated by the **Setup Slack Breakdown**:

- **When the Setup Slack Breakdown is less than 0**—the path has such a tight timing relationship, a large difference in microparameters, or such significant clock source uncertainty, that the path fails before the addition of any delay. Review the SDC constraints to verify that the timing relationship is correct. An incorrect relationship can exist between unrelated clocks that lack the proper timing cut. Ensure that parameterizable hard blocks (such as 20K RAM and DSP blocks) are fully registered. Investigate clock sources to verify that the clocks use global signals for routing.
- **When the clock skew exceeds the Setup Slack Breakdown**—address the clock transfer to meet timing on the path. You may need to create clock region assignments. You might also need to redesign cross-clock transfers to switch from synchronous to asynchronous implementation, such as with a FIFO or other handshake.

- **When the cell delay is greater than its intrinsic margin**—reduce the cell delay, as the path would fail timing even if the clocks are perfect and use no routing wires. Rewrite RTL to reduce the logic depth, restructure logic to allow the Compiler to use faster LUT inputs, or unblock retiming optimizations. The Compiler can automatically retime registers to reduce logic depth, but only in ways that maintain functionality and that the device architecture supports. To unblock the Hyper-Retimer, remove asynchronous resets and initial conditions.
- **When the interconnect delay is greater than its intrinsic margin**—the path would fail timing even if the clocks are perfect, and there is no logic. This occurs if registers are too far apart, or a timing path detours around a congested chip area. Review the fan-in and fan-out of registers that are far apart. Apply Logic Lock regions so the Fitter places the registers closer together. Use Logic Lock regions only after determining why placement is initially poor.

Related Information

Retiming Restrictions and Workarounds, Intel Hyperflex™ Architecture High-Performance Design Handbook

2.5.1.3. Report Timing By Source Files

The Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing By Source Files** command allows you to specify settings to report the timing of any path or clock domain in the design, similar to Report Timing on page 34. However, **Report Timing By Source Files** groups the timing paths by the containing entity, and groups the entities by the source file that defines the entity.

This report allows you to attribute timing paths to exactly one instance in the design. The Path TNS column shows the sum of all negative slacks within a file or entity. The equivalent scripting command is `report_timing_by_source_files`.

2.5.1.4. Report Data Delay

You can run the Timing Analyzer's **Reports ➤ Other Slack Analyses ➤ Report Data Delay...** command to generate a custom report showing the worst-case slack for the datapath delay exception for a given path. The report shows only paths covered by data delay (`set_data_delay`) constraints, including paths whose non-datapath analysis is cut by a false path.

Note: Exclusive clock groups (set with `set_clock_group -exclusive`) override `set_data_delays` constraints.

Related Information

Timing Exception Precedence on page 106

2.5.1.5. Report Net Delay

The Timing Analyzer's **Reports ➤ Other Slack Analyses ➤ Report Net Delay...** command to configure and display a customized report that details the results of net delay analysis, as defined by timing constraints and exceptions.

Each `set_net_delay` command is treated as a separate analysis and `report_net_delay` reports the results of all `set_net_delay` commands in a single report. The report contains each `set_net_delay` command with the worst case slack result followed by the results of each edge matching the criteria set by that `set_net_delay` command. These results are ordered based on the slack value.

2.5.1.6. Report Clocks and Clock Network

The Timing Analyzer's **Reports ➤ Clocks ➤ Report Clocks** command reports all clock signals in the design. The equivalent scripting command is `report_clocks`.

Report Clocks generates the Clock Summary report that lists details about all of the signals with clock setting constraints in the design.

Figure 49. Clock Summary Report Shows Properties of Clock Signals in Design

Clocks Summary

Show: Visible | Hide | <<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase	Offset
1	clk	Base	40.000	25.0 MHz	0.000	20.000					

Similarly, you can click the **Reports ➤ Clocks ➤ Report Clock Network...** command to generate a custom report that helps you identify and evaluate advanced clock structures, such as clock muxes, clock gates, and clock dividers.

Figure 50. Report Clock Network Dialog Box

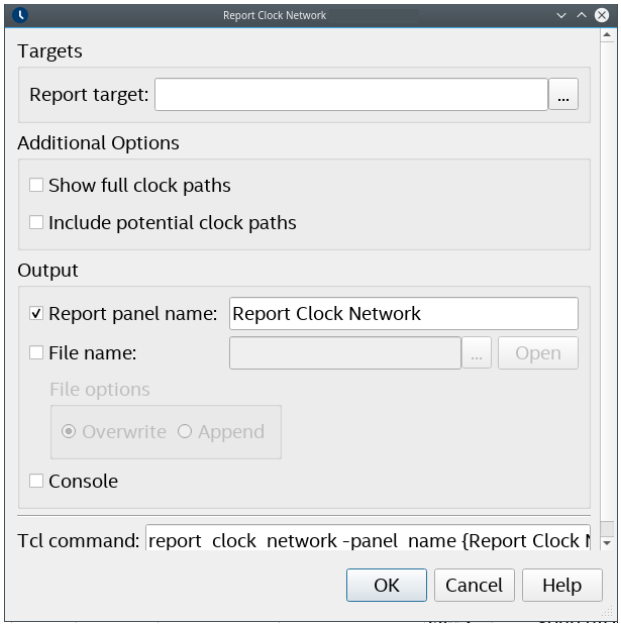
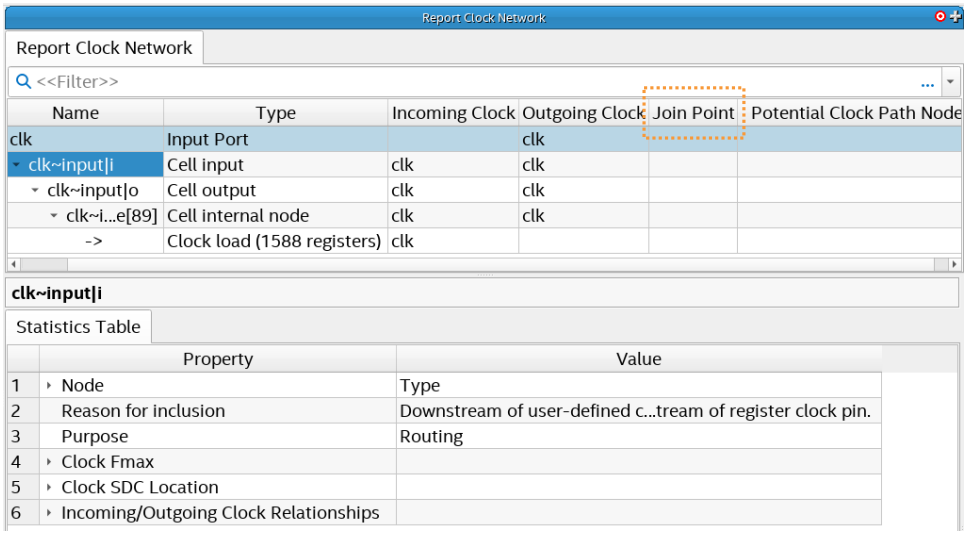


Table 9. Report Clock Network Dialog Box Settings

Option	Description
Report target	Specifies the collection of clocks and nodes that you want to analyze and report.
Expand clock path	Displays all subordinate nodes in expanded view. The default display collapses trivial nodes in the report.
Include potential clock paths	Includes nodes in the report that are not on a clock path, but are upstream of a register clock port.
Report panel name	Specifies the name that appears in the report panel title bar.
File name	Specifies the name of an optional output file to contain report data.
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.

This report shows the nodes on the clock network hierarchically, starting from the input clock ports, followed by any other nodes that transform or route the clocks to the clock loads. The **Join Points** indicate whether the clock network has convergence, such as with clock muxes. The **Statistics Table** provides more details about the signals that you select in the report, such as the relationships between the incoming and outgoing clocks of this node.

Figure 51. Report Clock Network Report

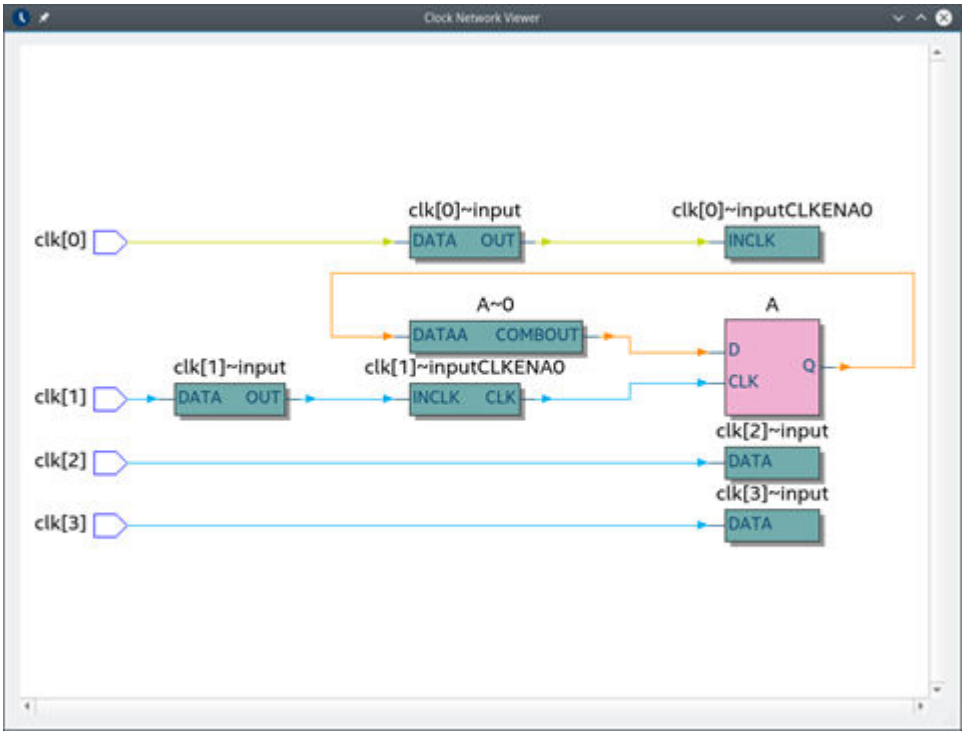


Right-click any of the nodes in the Clock Network report to click **Open Clock Network Viewer**. The Clock Network Viewer displays a graphical representation of the clock domains and constraints on the clock network to help you to see clock tree problems, such as signals entering and exiting globals. Use this graphical view to determine which clocks drive portions of the design. The Clock Network Viewer color codes the clock connection edge to indicate the clock types.

- Blue—the base clock
- Orange—a derived clock
- Green—a multicycle clock

The display shows a truncated signal name by default. Hover the mouse over the clock signals to display the full signal name. Right-click any signal to display the **Color Legend**. Click the **Zoom** controls to view more detail. You can export the schematic view as a PDF from the right-click menu.

Figure 52. Clock Network Viewer



You can also right-click and choose **Report Paths from Node**, **Report Paths Thru Node**, **Report Paths To Node**, or **Focus On Node** to rerun the report on the selected node.

2.5.1.7. Report Clock Transfers

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Clock Transfers** command reports all clock-to-clock transfers in the design. The equivalent scripting command is `report_clock_transfers`.

Report Clock Transfers generates the Setup Transfers report and the Hold Transfers report that display data about the clock-to-clock transfers.

Figure 53. Setup Transfers Report Shows Clock-to-Clock Transfers

Setup Transfers

Show: Visible | Hide | <<Filter>>

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths	Clock Pair Classification	Worst-Case Slack	Worst-Case Operating Conditions
1	clock	clock	34	0	0	0	Intra-Clock (Timed Safe)	3.319	Slow vid2 100C Model

The Setup Transfers report and Hold Transfers report display all possible transfers, including rising clock edge to rising clock edge (RR), falling clock edge to rising clock edge (FR), rising clock edge to falling clock edge (RF), and falling clock edge to falling clock edge (FF) paths.

- If a path exists in the design, the report column cell is white and lists the number of paths.
- If a path is a false path, the report column cell is light gray and contains the text "false path."
- If a path does not exist in the design, then the report column cell is dark gray.

The Setup Transfers report and Hold Transfers report also lists the Worst-Case Slack for setup, the Worst-Case Operating Conditions, and the Clock Pair Classification for each clock path. The Clock Pair Classification includes the following:

Table 10. Clock Pair Classifications

Clock Pair Classification	Definition
Intra-Clock (Timed Safe)	• From Clock and To Clock are the same. • No timing constraint required.
Inter-Clock Synchronous (Timed Safe)	• From Clock and To Clock relate synchronously, and have a known phase and frequency relationship. • Multicycle path constraint may or may not exist.
Asynchronous (Timed Unsafe)	• From Clock and To Clock are asynchronous. • Timing constraints (false path, clock groups, set_max_skew) do not exist.
Ignored (Not Timed)	• From Clock and To Clock are asynchronous. • Timing constraints (false path, clock groups, set_max_skew) exist and setup and hold slack are not applicable.

2.5.1.8. Report Metastability

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Metastability...** command generates a list of synchronization register chains found in the design, and can provide estimates of the Mean Time Between Failures (MTBF) of each chain. The equivalent scripting command is `report_metastability`.

Metastable registers have outputs hovering at a voltage between high and low for a length of time beyond the normal t_{CO} for the register, which may cause subsequent registers that use this metastable signal to latch different values. Synchronize register chains when transferring data between unrelated clock domains to reduce the probability of the captured data signal becoming metastable.

A synchronization register chain is a sequence of registers with the same clock, that is driven by a pin, or logic from an unrelated clock domain. All but the last register in the chain must connect only to the next register, but may do so through logic.

The Metastability Report displays the following for each synchronization chain the analysis discovers:

- Typical Mean Time Between Failures (MTBF) for the chain
- Number of synchronization registers in the chain
- Names of synchronization registers in the chain

- Data toggle rate used in the MTBF estimation
- Source clock domain names
- Synchronization clock domain names

Related Information

- Metastability Analysis on page 15
- Step 1: Specify Timing Analyzer Settings on page 24

2.5.1.9. Report CDC Viewer

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report CDC Viewer...** command allows you to configure and display a custom clock domain crossing report and the Clock Domain Crossing (CDC) Viewer. The CDC Viewer graphically displays the setup, hold, recovery, or removal analysis of all clock transfers in your design. The equivalent scripting command is `report_cdc_viewer`.

Table 11. Report Clock Domain Crossing Viewer Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Analysis type	Options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Transfers	Specifies the type of clock transfers to include or exclude from the report, including **Timed transfers**, **Fully cut transfers**, **Clock groups, Inactive clocks**, and **Non-crossing transfers**. You can specify the **Maximum slack limit** and **Grid options** for the report.
Detail level	**Full** shows all details of the report and **Summary** filters the details and shows summary data.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data, and specify **Grid** or **List** format. *Note:* In grid format reports, clocks with non-crossing transfers always appear if they have transfers between other clocks.
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.

You can specify the following options to customize CDC Viewer reporting:

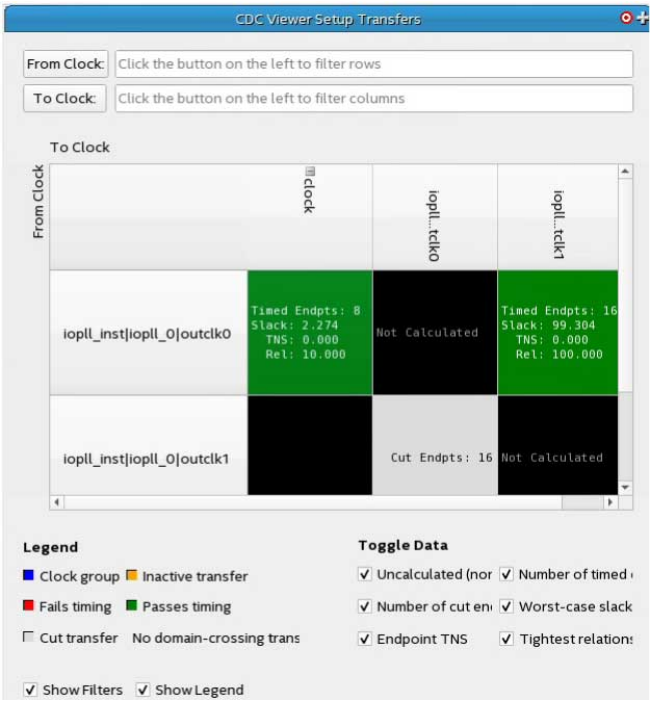
Table 12. CDC Viewer Report Controls

Control	Description
From Clock: and **To Clock:**	Filters the display according to the clock names you specify. Click **From Clock:** or **To Clock:** to search for specific clock names.
Legend	Defines the status colors. A color coded grid displays the clock transfer status. The clock headers list each clock with transfers in the design. The GUI truncates long clock names, but you can view the full name in a tool tip or by resizing the clock header cell. The GUI represents the generated clocks as children of the parent clock. A '+' icon next to a clock name indicates the presence of generated clocks. Clicking on the clock header displays the generated clocks associated with that clock.
Toggle Data	The text in each transfer cell contains data specific to each transfer. Turn on or off display of the following types of data:

continued...

Control	Description
	• **Number of timed endpoints** between clocks— the number of timed, endpoint-unique paths in the transfer. A path being "timed" means that analysis occurs on that path. Only paths with unique endpoints count towards this total. • **Number of cut endpoints** between clocks— the number of cut endpoint-unique paths, instead of timed paths. These paths are cut by either a false path or clock group assignment. Timing analysis skips such paths. • **Worst-case slack** between clocks— the worst-case slack among all endpoint-unique paths in the transfer. • **Total negative slack** between clocks— the sum of all negative slacks among all endpoint-unique paths in this transfer. • **Tightest relationship** between clocks— the lowest-value setup, hold, recovery, or removal relationship between the two clocks in this transfer.
Show Filters and **Show Legend**	Turns on or off Filters and **Legend**.

Figure 54. CDC Viewer Setup Transfers Report



Each block in the grid is a transfer cell. Each transfer cell uses color and text to display important details of the paths in the transfer. The color coding represents the following states:

Table 13. Transfer Cell Content

Cell Color	Color Legend
Black	Indicates no transfers. There are no paths crossing between the source and destination clock of this cell.
Green	Indicates passing timing. All timing paths in this transfer, that have not been cut, meet their timing requirements.

continued...

Cell Color	Color Legend
Red	Indicates failing timing. One or more of the timing paths in the transfer do not meet their timing requirements. If the transfer is between unrelated clocks, the paths likely require a synchronizer chain.
Blue	Indicates clock groups. The source and destination clocks of these transfers are cut by means of asynchronous clock groups.
Gray	Indicates a cut transfer. All paths in this transfer are cut by false paths. Therefore, timing analysis does not consider these paths.
Orange	Indicates inactive clocks. One of the clocks in the transfer is an inactive clock (with the `set_active_clocks` command). The Timing Analyzer ignores such transfers.

Right-click menus allow you to perform operations on transfer cells and clock headers. When the operation is a Timing Analyzer report or SDC command, a dialog box opens containing the contents of the transfer cell.

Table 14. Transfer Cell Right-Click Menus

Command	Description
Copy	Copies the contents of the transfer cell or clock header to the clipboard.
Report Timing	Reports timing. Not available for transfer cells with no valid paths (gray or black cells).
Report Endpoints	Reports endpoints. Not available for transfer cells with no cut paths (gray or black cells).
Report False Path	Reports false paths. Not available for transfer cells with no valid paths (black cells).
Report Exceptions	Reports exceptions. Only available for clock group transfers (blue cells).
Report Exceptions (with clock groups)	Reports exceptions with clock groups. Only available for clock group transfers (blue cells).
Set False Path	Sets a false path constraint.
Set Multicycle Path	Sets a multicycle path exception.
Set Min Delay	Sets a min delay constraint.
Set Max Delay	Sets a max delay constraint.
Set Clock Uncertainty	Sets a clock uncertainty constraint.

Table 15. Clock Header Right-Click Menus

Command	Description
Copy (include children)	Copies the name of the clock header, and the names of each of its derived clocks. This option only appears for clock headers with generated clocks.
Expand/Collapse All Rows/Columns	Shows or hides all derived clocks in the grid.
Create Slack Histogram	Generates a slack histogram report for the clock you select.
Report Timing From/To Clock	Generates a timing report for the clock you select. If you do not expand the clock to display derived clocks, the timing report includes all clocks that derive from the clock. To prevent this, expand the clock before right-clicking it.
Remove Clock(s)	Removes the clock you select from the design. If you do not expand the clock, timing analysis removes all clocks that derive from the clock.

You can view CDC Viewer output in any of the following formats:

- A report panel in the Timing Analyzer
- Output in the Timing Analyzer Tcl console
- A plain-text file
- An HTML file you can view in a web browser.

Related Information

- Report Asynchronous CDC on page 48
- Constraining CDC Paths on page 100

2.5.1.10. Report Asynchronous CDC

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Asynchronous CDC...** command allows you to classify and report all asynchronous clock-domain-crossing (CDC) transfers in your design. Asynchronous CDCs include single-bit transfers, multibit transfers, and asynchronous reset CDCs. Designs often contain unintended CDCs or transfers. Use this report to ensure that the Timing Analyzer correctly detects all CDCs.

Table 16. Report Asynchronous CDC Information Settings

Option	Available Settings
Clocks	Filters the report to only show CDCs that originate from **From clock** and terminate at **To clock**.
Targets	Filters the report to only show CDCs that originate from **From** register and terminate at **To** register. Both the **From** and **To** must be registers. This option is optional with the **Clocks** option.
Entries	Limits the number of entries that are reported per CDC category
Detail	Chooses whether to show a summary of all CDC's or give detail on each individual CDC
CDC Categories	Specifies the CDC categories to be reported.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can overwrite or append the file with latest data.
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.

Figure 55. Report Asynchronous CDC Summary View

Asynchronous CDC Summary

<<Filter>>

	CDC Type	CDC Count	Total Transfer Width
1	Single-bit CDC		
1	Unsynchronized Transfer	5	5
2	Unconstrained Synchronizer	1	1
3	Synchronizer Dri...le Clock Domains	2	2
4	Intra-Clock False Path Synchronizer	2	2
5	Synchronizer Not...otected for MTBF	1	1
6	Compliant Synchronizer	11	11
7	Transfer Precede...binational Logic	0	0
2	Multi-bit CDC		
1	Unconstrained CE-type CDC Bus	2	5
2	Compliant CE-type CDC Bus	2	5
3	Unconstrained MUX-type CDC Bus	2	8
4	Compliant MUX-type CDC Bus	2	8
5	Unsynchronized Synchronizer Bus	2	8
6	Unconstrained Synchronizer Bus	1	2
7	Compliant Synchronizer Bus	1	4
8	Synchronizer Bu... Uneven Lengths	0	0
9	Synchronizer afte...th Control Signal	0	0
3	Asynchronous Reset CDC		
1	Unsynchronized Reset Synchronizer	1	1
2	Unconstrained Reset Synchronizer	1	1
3	Reset Synchroniz...Multiple Sources	1	1
4	Inactive Reset Synchronizer	1	1

The summary view of the Asynchronous CDC Report provides an overview of the number of CDC's that fall into each category in your design. CDC Count gives the number of topologies detected for a category. **Total Transfer Width** shows the total number of CDC crossings for a category. This value is different than **CDC Count** for multibit CDC's because each topology consists of multiple crossings.

The full view of the Asynchronous CDC report shows the source and destination registers and clocks for each detected CDC topology in your design. You may click on any row of this report. Clicking on a row that contains the name of a CDC category brings up the description for that category and associated Design Assistant rules that check for such topologies. Clicking on a row that contains a CDC displays detailed information on that CDC in the **CDC Statistics** table. The information available varies depending on the topology of the CDC.

Figure 56. Report Asynchronous CDC Full View

Asynchronous CDC Full Report

	CDC Type	Source Nodes	Destination Nodes
1	Single-bit CDC		
1	Unsynchronized Transfer (5)		
1	--	unsynch_chain|src_reg	unsynch_c...ynch_reg1
2	--	chain_with_...out|src_reg	chain_with...synch_reg1
3	--	logic_within...hain|src_reg	logic_withi...|synch_reg1
4	--	logic_within...hain|src_reg	logic_withi...|synch_reg1
5	--	synch_reg_o...ain|src_reg	synch_reg_...synch_reg1
2	Unconstrained Synchronizer (1)		
1	--	unconstrain...ain|src_reg	unconstrai...synch_reg1
3	Synchronizer Drive... Clock Domains (2)		
1	--	multi_clock_...ain|src_reg1	multi_cloc...synch_reg1
2	--	comb_logic_...ain|src_reg	comb_logic...synch_reg1
4	Intra-Clock False Path Synchronizer (2)		
1	--	intra_clock_...t_chain|rst1	intra_clock..._chain|rst2
2	--	intra_clock_...hain|src_reg	intra_clock...|synch_reg1

Figure 57. Statistics Table Showing Detailed Information on Each CDC

Unsynchronized Transfer #4

CDC Statistics

	Property	Value
1	Number of Source Registers	1
2	Source Registers	
1	logic_within_forced_chain|src_reg	
3	Number of Discovered Synchronization Registers	2
4	Discovered Synchronization Register	Synchr...Number
1	logic_within_forced_chain|synch_reg1	1
2	logic_within_forced_chain|synch_reg2	2
5	Method of Synchronizer Identification	User Specified
6	Chain Length Requested by User	3
1	User-requested chain le...rs in the synchronizer.	
7	The destination register ...ata synchronizer because:	
1	It has SYNCHRONIZER_I...nous reset connected.	
8	The synchronizer is termi...he last register because:	
1	The following nodes ca...ynchronizer registers:	
1	dataOut[91]	
9	Chain Length Protected	1
10	Protecting less synchron...than discovered because:	
1	The synchronizer regis...mode is set to FORCED:	
1	logic_within_forced_chain|synch_reg1	

Related Information

- Report CDC Viewer on page 45
- Constraining CDC Paths on page 100

2.5.1.11. Report Logic Depth

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Logic Depth...** command allows you to report the number of logic levels within a clock domain. This value typically corresponds to the number of look-up tables (LUTs) that a path passes through.

The equivalent scripting command is `report_design_metrics -logic_depth`. **Report Logic Depth** shows the distribution of logic depth among the critical paths, allowing you to identify areas where you can reduce logic levels in your RTL.

Figure 58. Report Logic Depth (Histogram)

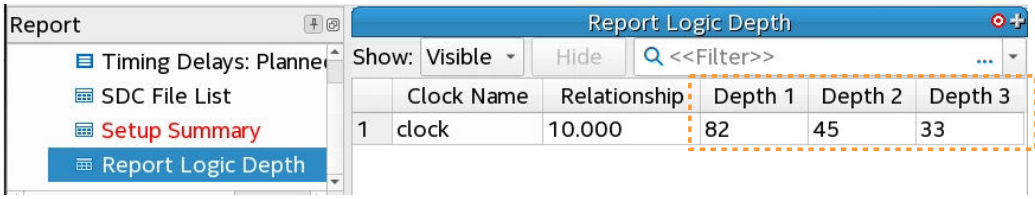


Figure 59. Report Paths of Depth 3

Call report logic depth by topology for each clock, intraclock only.

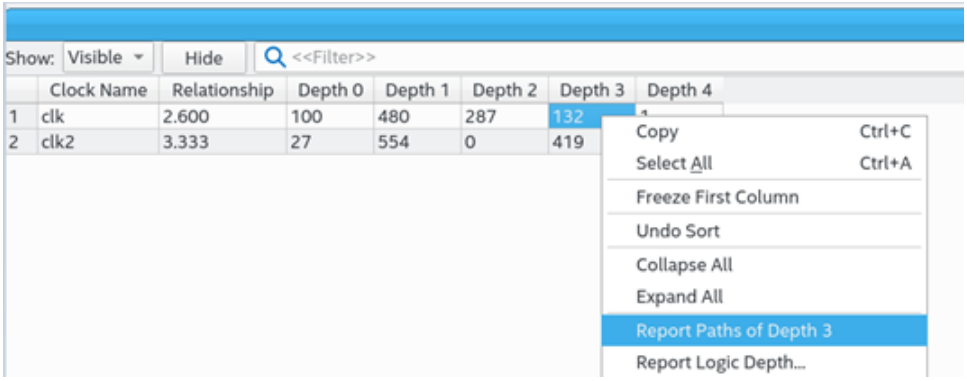
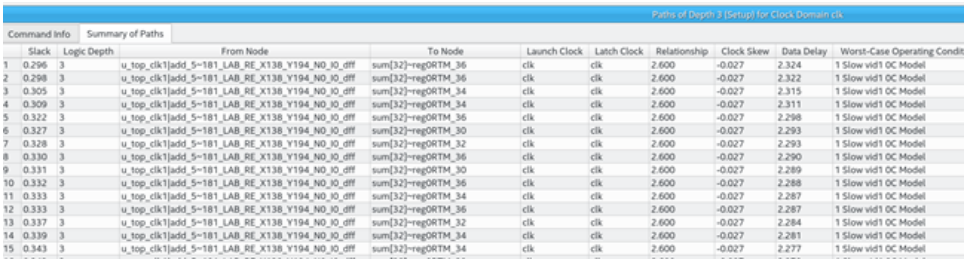


Figure 60. Summary of Paths

Close timing with accurate histogram cross probing.



You can specify various options to customize the reporting.

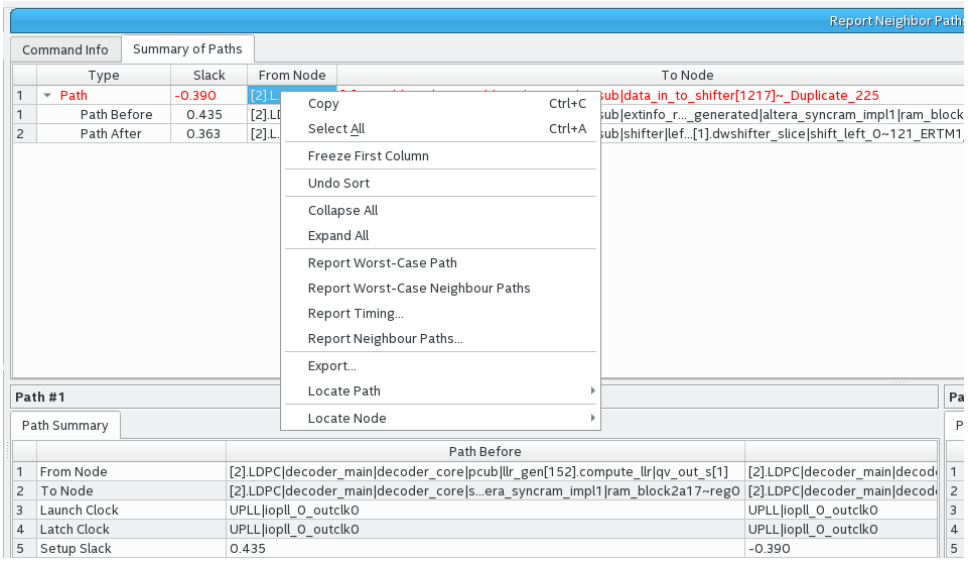
Table 17. Report Logic Depth Settings

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 logic depth with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. 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 **Setup**, **Hold**, **Recovery**, and **Removal** analyses report the logic depths of the top *X* paths by slack. **Topology** analysis reports the logic depths of the top *X* paths by logic depth.
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.
Detail	Specify whether to display on **Histogram** or full **Path** level of detail.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.

2.5.1.12. Report Neighbor Paths

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Neighbor Paths...** command helps you to determine the root cause of critical paths (for example, high logic level, retiming limitation, sub-optimal placement, I/O column crossing, hold fix-up, time borrowing, or others). The equivalent scripting command is `report_design_metrics -neighbor_paths`.

Figure 61. Report Neighbor Paths Report



Report Neighbor Paths reports the most timing-critical paths in the design, including associated slack, additional path summary information, and path bounding boxes. **Report Neighbor Paths** shows the most timing-critical **Path Before** and **Path After** each critical **Path**. You can optionally view multiple before and after paths. Retiming or logic balancing of the **Path** can simplify timing closure if there is negative slack on the **Path**, but positive slack on the **Path Before** or **Path After**.

Table 18. Report Neighbor Path Dialog Box Settings

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 neighbor 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. 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**. The Timing Analyzer reports the results for the type of analysis you select.
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.
Report Number of Neighbor Paths	Specifies the number of neighbor paths to report, allowing you to view a number of the top adjacent paths entering the critical path, and the top paths exiting the critical path.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.
Extra Info	Specifies extra info.

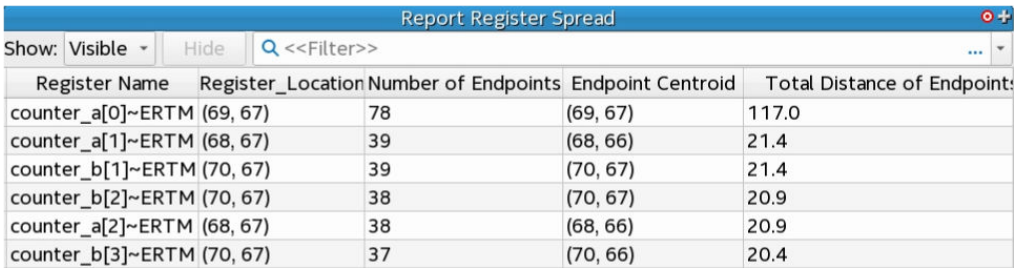
2.5.1.13. Report Register Spread

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Register Spread...** command analyzes the final placement to identify registers with sinks pulling them in various directions. These registers are potential candidates for duplication. The equivalent scripting command is `report_register_spread`.

Registers that drive in opposite directions and connect to high fan-out can have placement-warping effects on the floorplan that impact f_{MAX}. The placement-warping may not cause timing failures. Therefore, you can view this report to identify such registers. Taking steps to address the registers listed in the report can make placement of the design easier and improve f_{MAX} performance.

You can automate duplication of registers with the `DUPLICATE_REGISTER` and `DUPLICATE_HIERARCHY_DEPTH` `.qsf` assignments, or you can manually modify RTL to duplicate registers or refactor logic. Refer to "Automatic Register Duplication: Hierarchical Proximity" in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*.

Figure 62. Report Register Spread Report



Register Name	Register_Location	Number of Endpoints	Endpoint Centroid	Total Distance of Endpoint
counter_a[0]~ERTM	(69, 67)	78	(69, 67)	117.0
counter_a[1]~ERTM	(68, 67)	39	(68, 66)	21.4
counter_b[1]~ERTM	(70, 67)	39	(70, 67)	21.4
counter_b[2]~ERTM	(70, 67)	38	(70, 67)	20.9
counter_a[2]~ERTM	(68, 67)	38	(68, 66)	20.9
counter_b[3]~ERTM	(70, 67)	37	(70, 66)	20.4

You can specify various options to customize the report.

Table 19. Report Register Spread Settings

Option	Available Settings
Spread Type	Specifies the type of spread data in the report: • **Tension**—reports the sum over each sink of the distance from it to the centroid of all the sinks. • **Angle**—reports how far around the source register the fan-outs wrap, expressed from 0 to 360 degrees. This value corresponds to 360 minus the maximum angle between any two angularly adjacent sinks. This metric complements **Tension** by identifying registers which are surrounded by their sinks in all directions, and not those registers only being pulled in a few directions. • **Span**—reports the maximum 1-dimensional delta between the left bottom-most sink and the right top-most sink. • **Area**—reports the coverage of the sinks by number of LABs on the FPGA device. This option multiplies the span of the sinks in both X- and Y- dimensions. This metric complements **Span** by incorporating both dimensional spans of the sinks, and not only the maximum sink. • **Count**—reports registers with the largest sink counts.
Sink Type	Specifies the type of sink in the report: • **Endpoint**—the nodes (usually registers) that terminate timing paths from a register. • **Immediate Fanout**—the immediately connected nodes of the register. For example, lookup tables, other registers, RAM, or DSP blocks.
From Clock	Filters paths in the report to show only the launching clocks you specify.
To Clock	Filters paths in the report to show only the latching clocks you specify, allowing you to debug one clock at a time.
Report number of registers	Specifies the number of registers to display in the report. The default value for **Report number of registers** is 10.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.

Figure 63. Report Register Spread Types

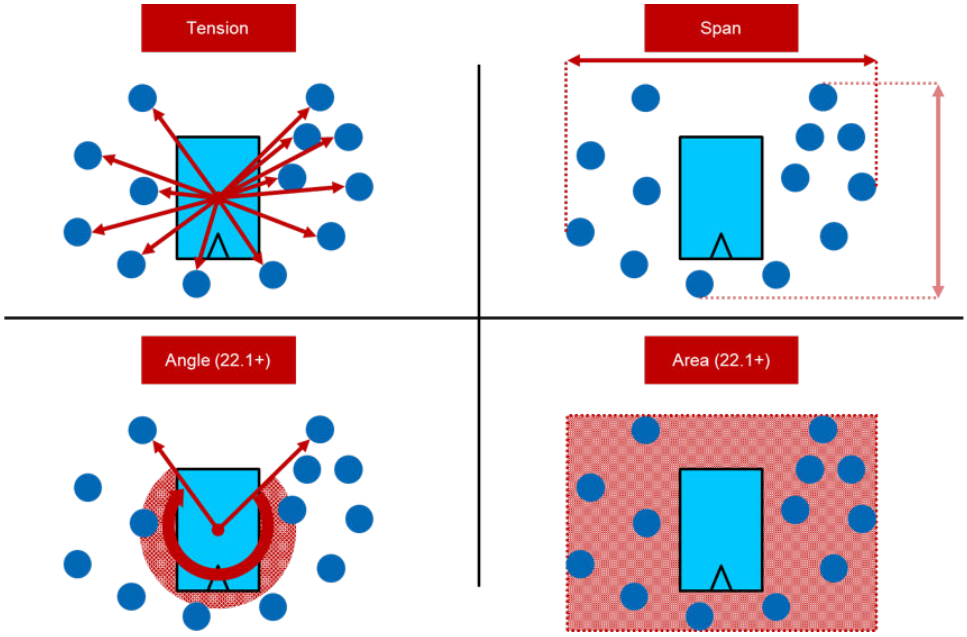
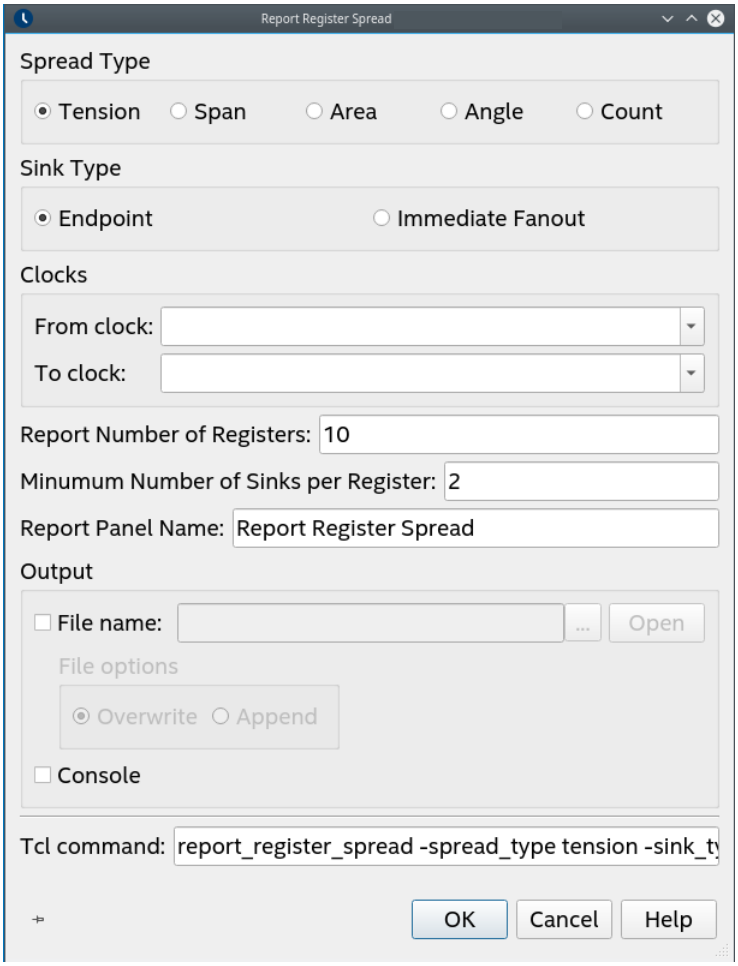


Figure 64. Report Register Spread Dialog Box



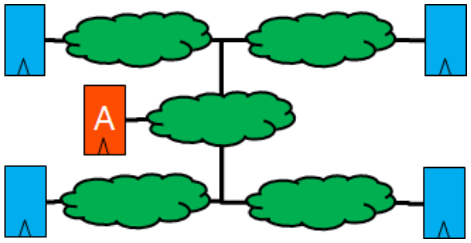
Related Information

Automatic Register Duplication: Hierarchical Proximity, Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.5.1.13.1. Registers with High Timing Path Endpoint Tension

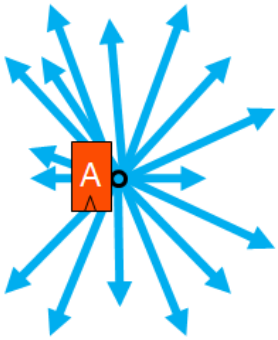
Timing path endpoints are the nodes (usually registers) that terminate timing paths from a register. The Timing path endpoint is equivalent to the nodes that the `get_fanouts` command returns, or the overall set of nodes that appear as a "From Node" after running the `report_timing` command. Register duplication is necessary, but not always sufficient, in helping to distribute the signal more efficiently. In addition, you may need to duplicate or restructure any intermediate logic before duplicating the register.

Figure 65. Register A has 4 Timing Path Endpoints



Tension is the sum over each sink of the distance from the sink to the centroid of all the sinks. The value of tension is therefore dependent on the number of sinks. Register duplication can help to break up these clouds, since they may be the result of the placement solution getting "warped" by the presence of the register.

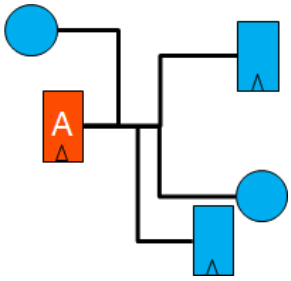
Figure 66. Register A has High Tension



2.5.1.13.2. Registers with High Immediate Fan-Out Tension

There are two **Sink Type** options: **Endpoint** and **Immediate Fanout**. The immediate fan-outs are the immediately connected nodes (lookup tables, other registers, RAM or DSP blocks, and others) of the register. This fan-out is equivalent to fan-outs that the Chip Planner displays, and in various high fan-out reports. Register duplication directly distributes the immediate fan-outs of a register among the duplicates.

Figure 67. Register A has High Immediate Fan-Out



2.5.1.14. Report Route Net of Interest

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Route Net of Interest...** command allows you to report the nets that require the most effort from the router. The report shows the percentage of total router effort for the nets reported. The equivalent scripting command is `report_route_net_of_interest`.

This report allows you to identify nets that should not require significant router effort. For example, you might expect that low speed management interface nets are not timing critical, and therefore not require much router effort. However, if **Report Route Net of Interest** reports that some nets in the low speed management interface require significant effort from the router, you can investigate that further. The investigation can determine whether the timing constraints are correct, whether the fan-out is significant and can reduce through driver duplication, or whether the net passes through congested areas.

Figure 68. Report Route Net of Interest Report

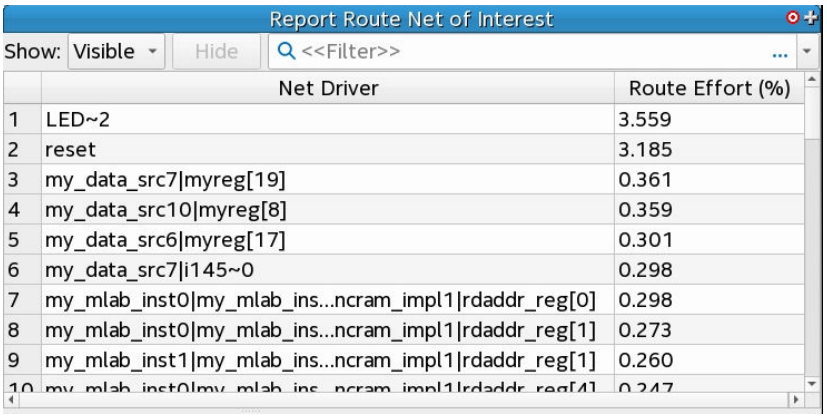


Figure 69. Report Route Net of Interest Dialog Box

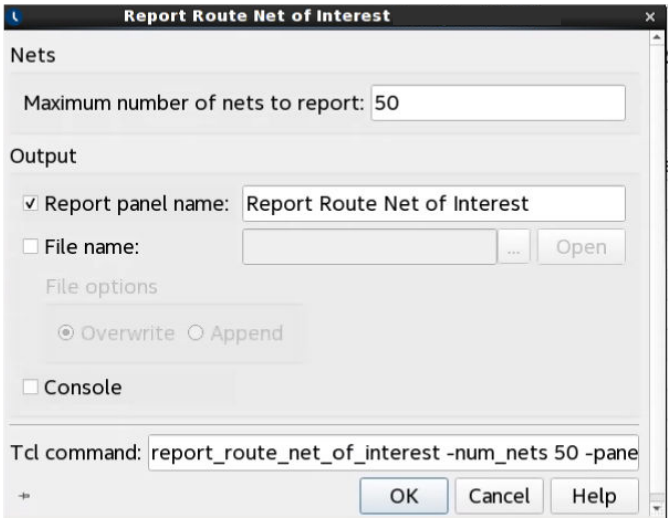


Table 20. Report Route Net of Interest Settings

Option	Available Settings
Nets	Specifies the **Maximum number of nets to report**. The default value is 50.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.

2.5.1.15. Report Retiming Restrictions

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Retiming Restrictions...** command allows you to report the occurrences of design conditions that restrict Hyper-Retiming, such as Power-up "Care" restrictions, and don't touch or preserve attributes for each port. You can refer to this report to improve the circuit and remove retiming restrictions that limit circuit performance. `report_retiming_restrictions` is the equivalent scripting command.

Figure 70. Report Retiming Restrictions Report

Report Retiming Restrictions

<<Filter>>

	Compilation Hierarchy Node	Register is part of a synchronization chain	Preserve assignm
1	|	114 (0)	292 (50)
1	|my_data_src7|	21 (21)	0 (0)
1	|myreg[*]	20 (20)	0 (0)
2	|enable_reg	1 (1)	0 (0)
2	|my_data_src8|	21 (21)	0 (0)
1	|myreg[*]	20 (20)	0 (0)
2	|enable_reg	1 (1)	0 (0)
3	|my_data_src5|	19 (19)	0 (0)
1	|myreg[*]	18 (18)	0 (0)
2	|enable_reg	1 (1)	0 (0)
4	|my_data_src6|	19 (19)	0 (0)

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of retiming restrictions in the specific entity alone. The numbers listed outside of parentheses indicate the number of retiming restrictions in the specific entity and all of its sub-entities in the hierarchy.

For table entries with two number values, the number in parentheses indicates the number of retiming restrictions in the specific entity alone. The number listed outside of parentheses indicates the number of retiming restrictions in the specific entity and all of its sub-entities in the `hierarchy`.

Related Information

Retiming Restrictions and Workarounds, Intel Hyperflex Architecture High-Performance Design Handbook

2.5.1.16. Report Register Statistics

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Register Statistics** command allows you to report the number of synchronous and asynchronous resets, hyper registers, and registers with clock enables in the design. You can use this information, combined with timing slack, congestion, and other analysis reports, to identify timing-critical parts of your design that can have resets removed or control schemes changed to meet timing requirements more efficiently.

Figure 71. Report Register Statistics

Report Register Statistics

<<Filter>> (use !<string> to invert filter)

	Compilation Hierarchy Node	Register Count	Without a Clock	Unique Clocks	Hyper-Register Count	Synchronous Reset	Asynchronous Reset	Clock Enable	Full Hierarchy Name
1	|	623 (0)	0 (0)	4 (0)	53 (0)	0 (0)	0 (0)	0 (0)	|
1	|chain1|	145 (145)	0 (0)	1 (1)	10 (10)	0 (0)	0 (0)	0 (0)	chain1
2	|chain2|	212 (197)	0 (0)	1 (1)	30 (23)	0 (0)	0 (0)	0 (0)	chain2
1	|bent3|	3 (3)	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	0 (0)	chain2|bent3
2	|bent5|	1 (1)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	chain2|bent5
3	|bent6|	3 (3)	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	0 (0)	chain2|bent6
4	|bent9|	8 (8)	0 (0)	1 (1)	5 (5)	0 (0)	0 (0)	0 (0)	chain2|bent9
3	|chain3|	106 (32)	0 (0)	1 (1)	9 (0)	0 (0)	0 (0)	0 (0)	chain3
1	|bent3|	6 (6)	0 (0)	1 (1)	2 (2)	0 (0)	0 (0)	0 (0)	chain3|bent3
2	|bent5|	9 (9)	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	0 (0)	chain3|bent5
3	|bent6|	16 (16)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	chain3|bent6
4	|bent8|	17 (17)	0 (0)	1 (1)	2 (2)	0 (0)	0 (0)	0 (0)	chain3|bent8
5	|bent9|	26 (26)	0 (0)	1 (1)	4 (4)	0 (0)	0 (0)	0 (0)	chain3|bent9
4	|chain4|	160 (125)	0 (0)	1 (1)	4 (2)	0 (0)	0 (0)	0 (0)	chain4
1	|bent3|	3 (3)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	chain4|bent3
2	|bent5|	1 (1)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	chain4|bent5
3	|bent6|	6 (6)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	chain4|bent6
4	|bent8|	9 (9)	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	0 (0)	chain4|bent8
5	|bent9|	16 (16)	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	0 (0)	chain4|bent9

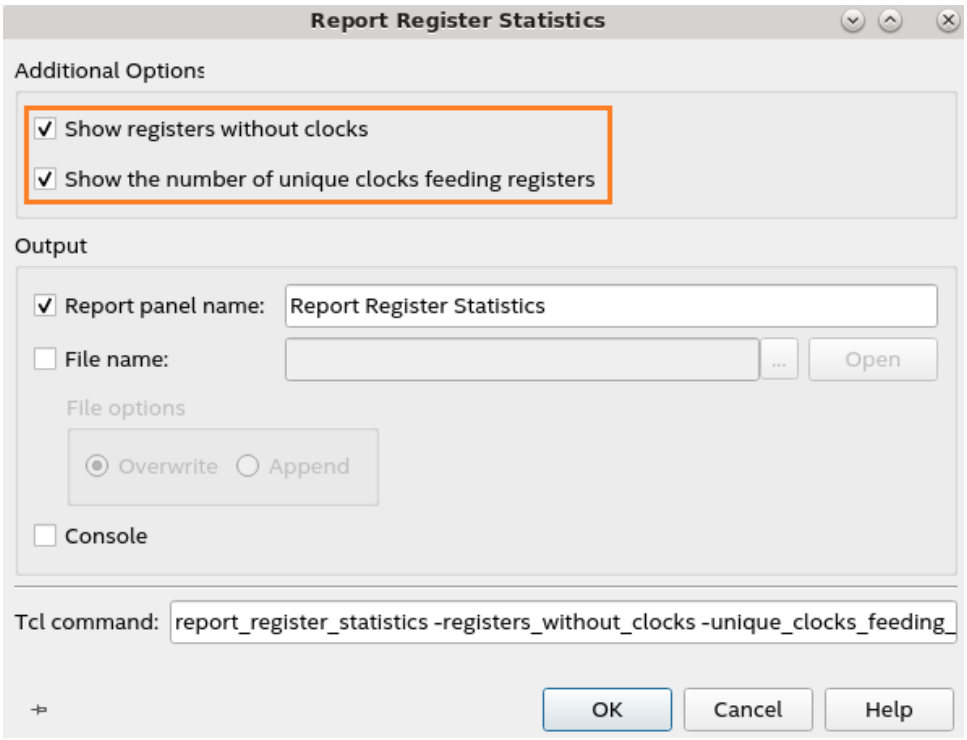
Note:

- This report works similarly in both post-synthesis (DNI flow) and post-plan timing analysis. However, the report's **Without a Clock** column is more helpful for the post-synthesis timing analysis because conventional (non-SDC-on-RTL) SDCs are not typically loaded in the post-synthesis mode, so through this report, you can analyze how timing gets affected in the absence of the SDCs.
- Clocks generated from `derive_clocks` commands do not count as user clocks.

The **Without a Clock** column informs you of the number of registers where no defined clock feeds the registers in the hierarchy shown in the **Register Count** column. A value of 0 in this column suggests that your design has SDC-defined clocks feeding registers in the design. The **Unique Clocks** column indicates the number of unique SDC-defined clocks feeding registers in the hierarchy identified by the

Register Count. To view these columns, enable **Show registers without clocks** and **Show the number of unique clocks feeding registers** additional options in the dialog that displays when you run the report, as shown in the following image:

Figure 72. Report Register Statistics Additional Options Dialog

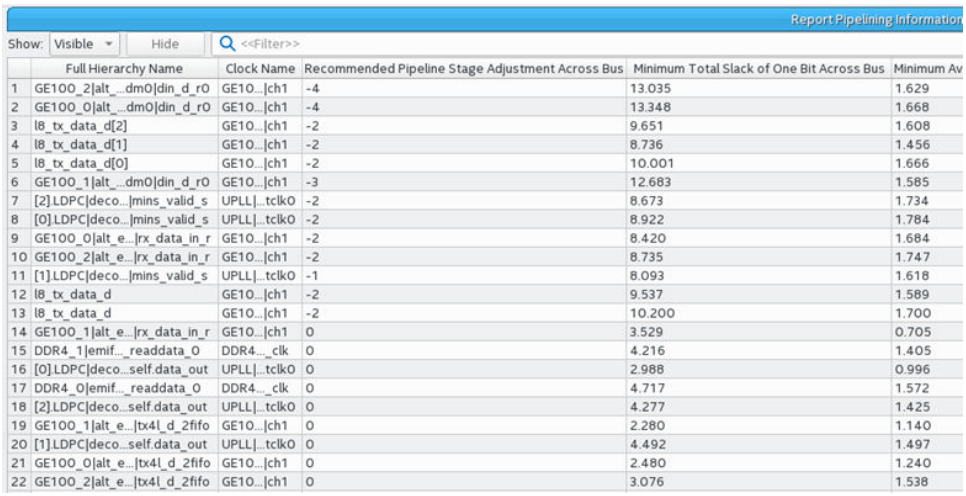


2.5.1.17. Report Pipelining Information

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Pipelining Information...** command allows you to generate a report that can help you to identify potential areas of over-pipelining in your design. Excessive pipelining unnecessarily consumes area. The equivalent scripting command is `report_pipelining_info`.

Report Pipelining Information... does not perform any functional analysis in making the recommended pipeline stage adjustment. You must be aware of any potential functional changes from removing pipeline stages. There may be circumstances when all the stages in a register pipeline are necessary for functional reasons. The report helps to identify location with more registers than necessary for covering distance.

Figure 73. Report Pipelining Information Report

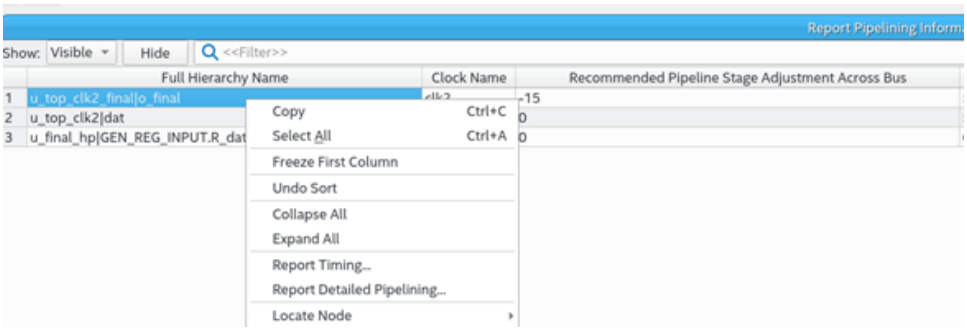


Report Pipelining Information

Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus	Minimum Total Slack of One Bit Across Bus	Minimum Av
1	GE100_2|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.035	1.629
2	GE100_0|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.348	1.668
3	l8_tx_data_d[2]	GE10...|ch1	-2	9.651	1.608
4	l8_tx_data_d[1]	GE10...|ch1	-2	8.736	1.456
5	l8_tx_data_d[0]	GE10...|ch1	-2	10.001	1.666
6	GE100_1|alt_...dm0|din_d_r0	GE10...|ch1	-3	12.683	1.585
7	[2].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.673	1.734
8	[0].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.922	1.784
9	GE100_0|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.420	1.684
10	GE100_2|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.735	1.747
11	[1].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-1	8.093	1.618
12	l8_tx_data_d	GE10...|ch1	-2	9.537	1.589
13	l8_tx_data_d	GE10...|ch1	-2	10.200	1.700
14	GE100_1|alt_e...|rx_data_in_r	GE10...|ch1	0	3.529	0.705
15	DDR4_1|emif..._readdata_0	DDR4..._clk	0	4.216	1.405
16	[0].LDPC|deco...self.data_out	UPLL|...tclk0	0	2.988	0.996
17	DDR4_0|emif..._readdata_0	DDR4..._clk	0	4.717	1.572
18	[2].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.277	1.425
19	GE100_1|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.280	1.140
20	[1].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.492	1.497
21	GE100_0|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.480	1.240
22	GE100_2|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	3.076	1.538

Figure 74. Report Detailed Pipelining



Report Pipelining Inform

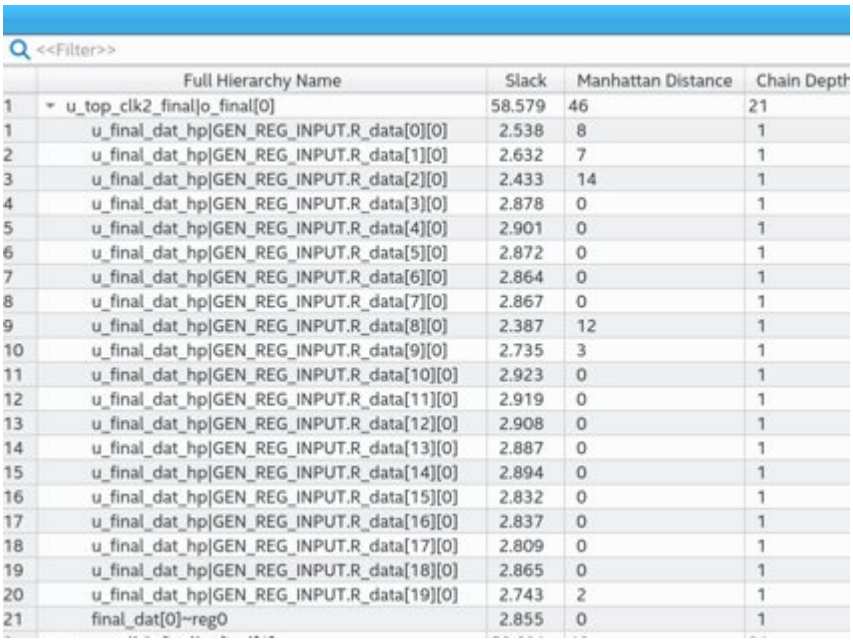
Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus
1	u_top_clk2_final|o_final		-15
2	u_top_clk2|dat		0
3	u_final_hp|GEN_REG_INPUT.R_dat		0

Copy Ctrl+C
Select All Ctrl+A
Freeze First Column
Undo Sort
Collapse All
Expand All
Report Timing...
Report Detailed Pipelining...
Locate Node

The detailed report shows every register in a tree structure. Over- or under-pipelining recommendations are in the main report. The following shows every single register inside the bus chain in a tree structure:

Figure 75. Detailed Pipelining Result



<<Filter>>

	Full Hierarchy Name	Slack	Manhattan Distance	Chain Depth
1	u_top_clk2_final|o_final[0]	58.579	46	21
1	u_final_dat_hp|GEN_REG_INPUT.R_data[0][0]	2.538	8	1
2	u_final_dat_hp|GEN_REG_INPUT.R_data[1][0]	2.632	7	1
3	u_final_dat_hp|GEN_REG_INPUT.R_data[2][0]	2.433	14	1
4	u_final_dat_hp|GEN_REG_INPUT.R_data[3][0]	2.878	0	1
5	u_final_dat_hp|GEN_REG_INPUT.R_data[4][0]	2.901	0	1
6	u_final_dat_hp|GEN_REG_INPUT.R_data[5][0]	2.872	0	1
7	u_final_dat_hp|GEN_REG_INPUT.R_data[6][0]	2.864	0	1
8	u_final_dat_hp|GEN_REG_INPUT.R_data[7][0]	2.867	0	1
9	u_final_dat_hp|GEN_REG_INPUT.R_data[8][0]	2.387	12	1
10	u_final_dat_hp|GEN_REG_INPUT.R_data[9][0]	2.735	3	1
11	u_final_dat_hp|GEN_REG_INPUT.R_data[10][0]	2.923	0	1
12	u_final_dat_hp|GEN_REG_INPUT.R_data[11][0]	2.919	0	1
13	u_final_dat_hp|GEN_REG_INPUT.R_data[12][0]	2.908	0	1
14	u_final_dat_hp|GEN_REG_INPUT.R_data[13][0]	2.887	0	1
15	u_final_dat_hp|GEN_REG_INPUT.R_data[14][0]	2.894	0	1
16	u_final_dat_hp|GEN_REG_INPUT.R_data[15][0]	2.832	0	1
17	u_final_dat_hp|GEN_REG_INPUT.R_data[16][0]	2.837	0	1
18	u_final_dat_hp|GEN_REG_INPUT.R_data[17][0]	2.809	0	1
19	u_final_dat_hp|GEN_REG_INPUT.R_data[18][0]	2.865	0	1
20	u_final_dat_hp|GEN_REG_INPUT.R_data[19][0]	2.743	2	1
21	final_dat[0]~reg0	2.855	0	1

To help identify potential over-pipelining, **Report Pipelining Information** reports:

- The recommended pipeline stage adjustment across bus
- The minimum total slack of one bit across bus
- The minimum average slack of one bit across bus
- The distance between the registers
- The width of buses in your design
- The number of sequential registers
- The number of registers on the bus

The Recommended Pipeline Stage Adjustment Across Bus reports the number of registers that you can remove from the bus for each bit. The Average Distance Per Stage, Max Distance Per Stage, and Min Distance Per Stage columns report the Manhattan distance measured in logic array blocks (LABs). The Bus Average Depth, Bus Max Depth, and Bus Min Depth columns report the number of sequential, single fan-out registers. For registers that have more than one clock source, the report lists the fastest one.

The **1+** sign under Recommended Pipeline Stage Adjustment Across Bus column means that the bus might need to add more registers to meet timing requirement. Refer to the Fast Forward Timing Closure Recommendations report.

If the report identifies a large register chain with multiple sequential registers, and the distance between registers is low, that condition can suggest over-pipelining. You may be able to remove some registers to recover some of the device area and reduce congestion.

The following options are available for this report:

Figure 76. Report Pipelining Information Dialog Box

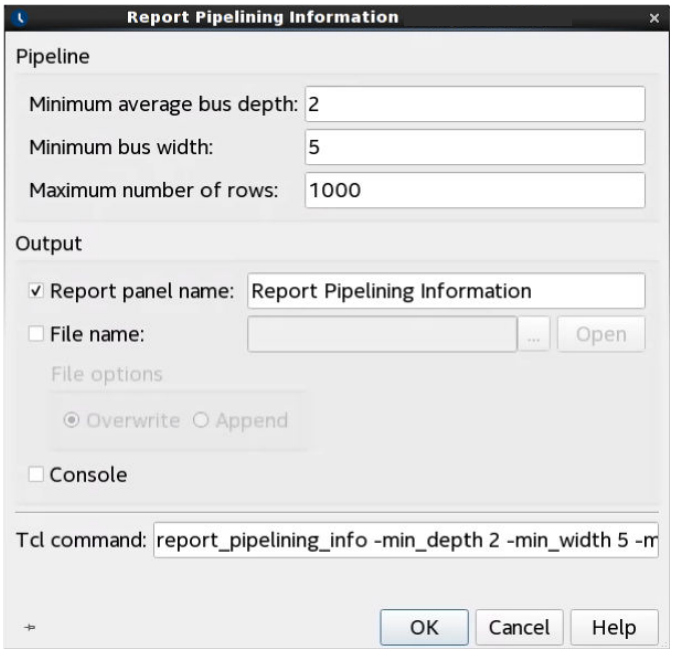


Table 21. Report Pipelining Information Settings

Option	Available Settings
Pipeline	Specifies the thresholds for reporting a register pipeline. You can define the **Minimum average bus depth**, the **Minimum bus width**, and the **Maximum number of rows** that the report includes.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
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.

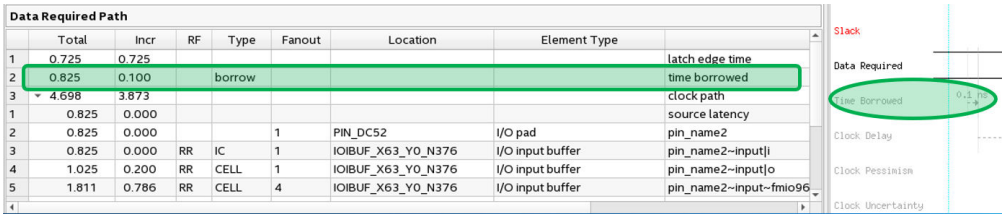
2.5.1.18. Report Time Borrowing Data

You can run the Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing...** command to generate a report showing time borrowing data. The equivalent scripting command is `report_timing` (with specific arguments).

The Timing Analyzer reports time borrowing data for the **Data Arrival Path** or **Data Required Path**, according to whether borrowing occurs at the destination or the source.

Figure 77. Report for Time Borrowing At Destination

The following report shows 100ps borrowed on the Data Required path. The setup slack improves on incoming paths, at the expense of worse setup slack for outgoing paths.

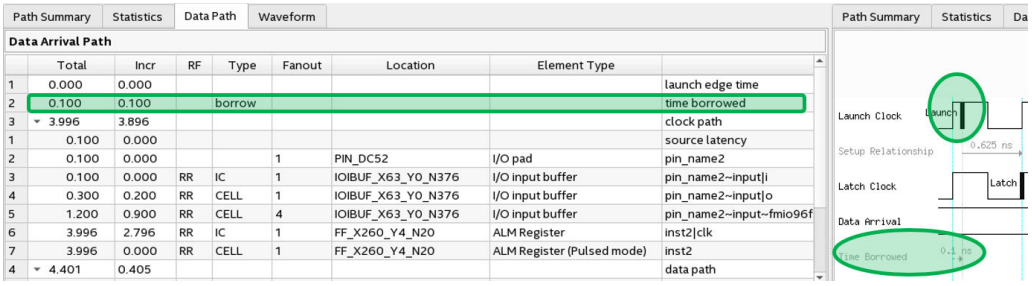


Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element Type	
1	0.725	0.725						latch edge time
2	0.825	0.100		borrow				time borrowed
3	4.698	3.873						clock path
1	0.825	0.000						source latency
2	0.825	0.000			1	PIN_DC52	I/O pad	pin_name2
3	0.825	0.000	RR	IC	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|i
4	1.025	0.200	RR	CELL	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|o
5	1.811	0.786	RR	CELL	4	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input~fmio96

Figure 78. Report for Time Borrowing At Source

The following report shows 100ps borrowed on the Data Arrival Path, worsening the setup relationship for the outgoing path.



Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element Type	
1	0.000	0.000						launch edge time
2	0.100	0.100		borrow				time borrowed
3	3.996	3.896						clock path
1	0.100	0.000						source latency
2	0.100	0.000			1	PIN_DC52	I/O pad	pin_name2
3	0.100	0.000	RR	IC	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|i
4	0.300	0.200	RR	CELL	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|o
5	1.200	0.900	RR	CELL	4	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input~fmio96f
6	3.996	2.796	RR	IC	1	FF_X260_Y4_N20	ALM Register	inst2|clk
7	3.996	0.000	RR	CELL	1	FF_X260_Y4_N20	ALM Register (Pulsed mode)	inst2
4	4.401	0.405						data path

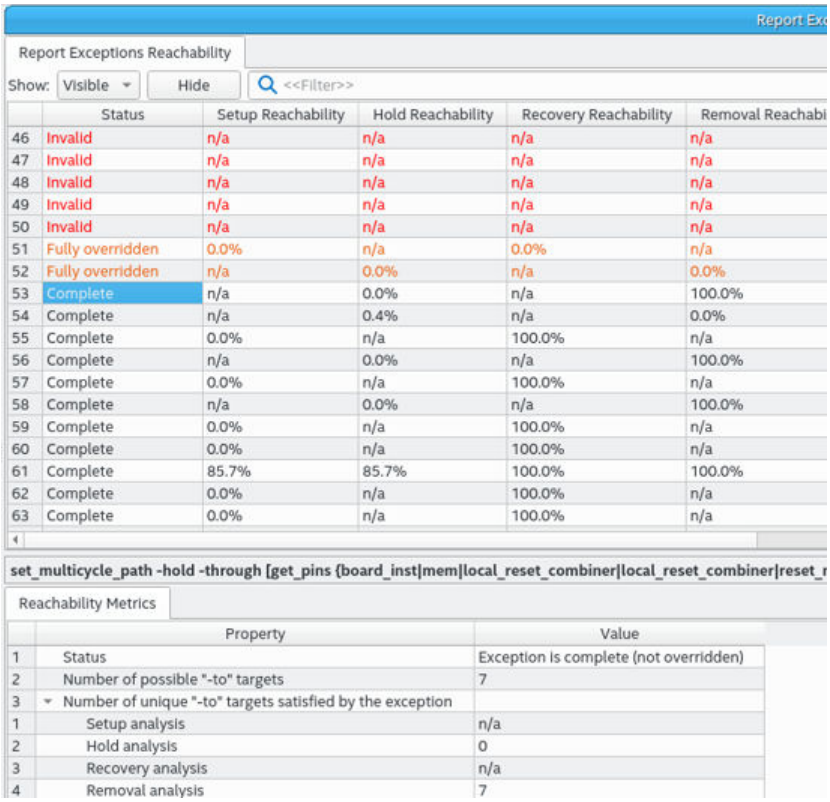
Related Information

Time Borrowing on page 18

2.5.1.19. Report Exceptions and Exceptions Reachability

The Timing Analyzer's **Reports ➤ Constraint Diagnostics ➤ Report Exceptions...** command allows you to report all exceptions to default timing analysis conditions, as specified by the **Set False Path**, **Set Multicycle Path**, **Set Minimum Delay**, or **Set Maximum Delay** commands (and the corresponding Tcl commands: `set_false_path`, `set_multicycle_path`, `set_min_delay`, and `set_max_delay`.)

Figure 79. Report Exceptions Reachability Report



	Status	Setup Reachability	Hold Reachability	Recovery Reachability	Removal Reachabi
46	Invalid	n/a	n/a	n/a	n/a
47	Invalid	n/a	n/a	n/a	n/a
48	Invalid	n/a	n/a	n/a	n/a
49	Invalid	n/a	n/a	n/a	n/a
50	Invalid	n/a	n/a	n/a	n/a
51	Fully overridden	0.0%	n/a	0.0%	n/a
52	Fully overridden	n/a	0.0%	n/a	0.0%
53	Complete	n/a	0.0%	n/a	100.0%
54	Complete	n/a	0.4%	n/a	0.0%
55	Complete	0.0%	n/a	100.0%	n/a
56	Complete	n/a	0.0%	n/a	100.0%
57	Complete	0.0%	n/a	100.0%	n/a
58	Complete	n/a	0.0%	n/a	100.0%
59	Complete	0.0%	n/a	100.0%	n/a
60	Complete	0.0%	n/a	100.0%	n/a
61	Complete	85.7%	85.7%	100.0%	100.0%
62	Complete	0.0%	n/a	100.0%	n/a
63	Complete	0.0%	n/a	100.0%	n/a

	Property	Value
1	Status	Exception is complete (not overridden)
2	Number of possible "-to" targets	7
3	Number of unique "-to" targets satisfied by the exception	
1	Setup analysis	n/a
2	Hold analysis	0
3	Recovery analysis	n/a
4	Removal analysis	7

Similarly, you can click on **Reports ➤ Constraint Diagnostics ➤ Report Exceptions Reachability...** to report the scope of exception constraints in your project. This report allows you to determine whether constraints apply fully or partially, are overridden, and whether the source and destination node are reachable for the constraint by providing a "reachability" percentage. Reachability is the percentage of paths to which the constraint applies. Low reachability indicates a constraint that may be too broad, potentially covering many unrelated items. High reachability indicates that the exception is very targeted.

2.5.1.20. Report Bottlenecks

You can run the Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Bottlenecks...** command to list all nodes in a design ranked by specified criteria. The equivalent scripting command is `report_bottleneck`.

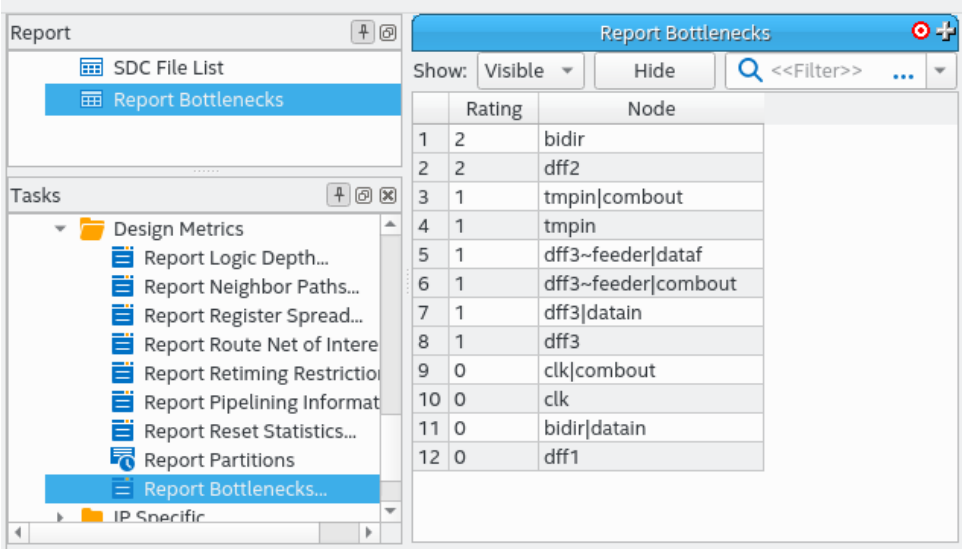
The following ranking criteria are pre-defined:

- `num_fpaths`—the number of paths that fail timing through a node.
- `num_fanins`—the number of fan-in edges from a node.
- `num_fanouts`—the number of fan-out edges from a node.
- `num_paths`—the number of paths through a node.
- `tns`—the total negative slack of all the paths through a node.

When using scripting, you can specify the paths for analysis by passing the result of any `get_timing_paths` call as the last argument to `report_bottleneck`. When using the GUI, the **Report Bottlenecks** dialog box handles this argument automatically. If you specify no paths, `report_bottleneck` analyzes the worst 1000 setup paths in the design by default.

You can direct the report output to the Tcl console (`-stdout`), the Timing Analyzer GUI (`-panel`), or to a combination of console and GUI.

Figure 80. Report Bottlenecks Rated on Number of Failing Paths Through a Node



2.5.1.20.1. Specifying Custom Bottleneck Criteria

You can optionally specify your own custom criteria for evaluating nodes based on the combination of the number of fan-outs, fan-ins, failing paths, and total paths.

To specify custom bottleneck criteria, follow these steps:

1. Create a Tcl procedure that takes one argument. For example, `arg`.
2. Use `upvar $arg metric` in the procedure.
3. Calculate the rating based on `$metric(tns)`, `$metric(num_fanouts)`, `$metric(num_fanins)`, and `$metric(num_fpaths)`.
4. Return the rating with `return $rating`.
5. Pass the name of your custom criteria procedure to `report_bottleneck` using the `-cmetric` option

2.5.2. Cross-Probing with Design Assistant

The Intel Quartus Prime Design Assistant can automatically report any violations against a standard set of Intel FPGA-recommended design guidelines during stages of compilation. You can specify which rules you want the Design Assistant to check in your design, and customize the severity levels, thus eliminating or waiving rule checks that are not important for your design.

When you run Design Assistant during compilation, Design Assistant utilizes the in-flow (transient) data that generates during compilation to check for rule violations.

When you run Design Assistant in analysis mode from the Timing Analyzer, Design Assistant performs design rule checks using the static compilation snapshot data that you load.

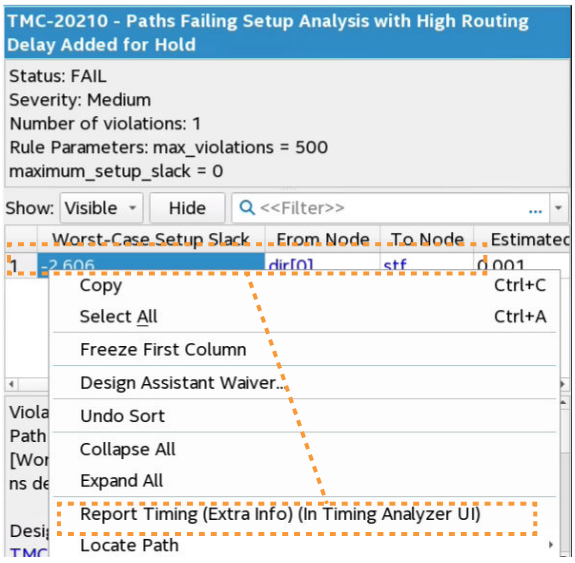
Some Design Assistant rule violations allow cross-probing into the related timing analysis data. Cross-probing can help you to more quickly identify the root cause and resolve any Design Assistant rule violations. For example, for a path with a setup analysis violation, you can cross-probe into the Timing Analyzer to identify the edge that has delay added for hold time.

Note: You must run the Compiler through at least the Plan stage before you can cross-probe to Timing Analyzer.

2.5.2.1. Cross-Probing from Design Assistant to Timing Analyzer

Some Design Assistant rule violations allow cross-probing into Timing Analyzer. For example, for a path that Design Assistant flags with a setup analysis violation due to delay added for hold, you can cross-probe into the Timing Analyzer to view more information on the affected path and edge.

Figure 81. Cross Probing from Design Assistant Rule TMC-20210 Violations to Timing Analyzer



Follow these steps to cross-probe from such Design Assistant rule violations to the Timing Analyzer:

1. Compile the design through at least the Compiler's Plan stage.
2. Locate a rule violation in the Design Assistant folder of the Compilation Report.
3. Right-click the rule violation to display any **Report Timing** commands available for the violation.
4. Click the **Report Timing** command. The Timing Analyzer opens and reports the timing data for the violation path. **Report Timing (Extra Info)** includes Estimated Delay Added for Hold and Route Stage Congestion Impact extra data.

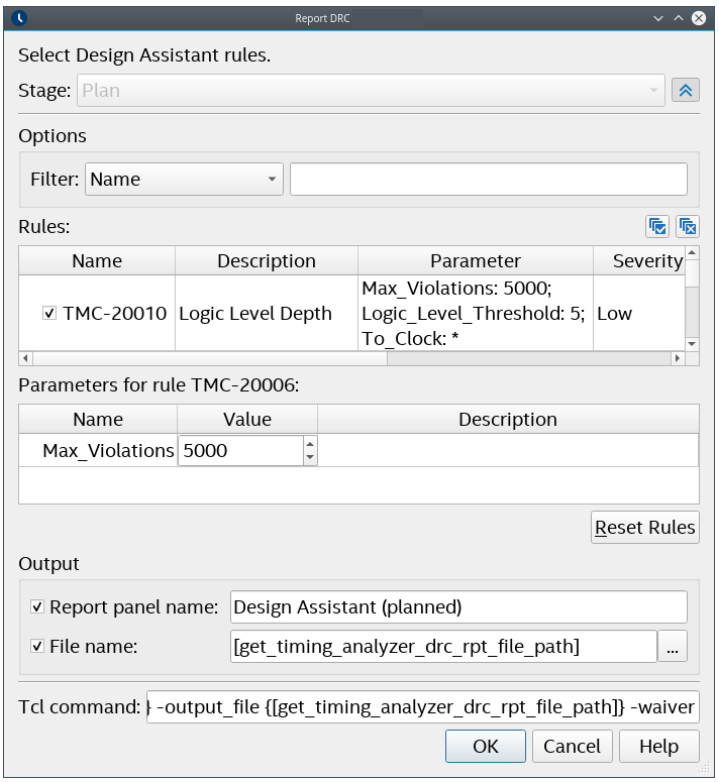
2.5.3. Launching Design Assistant from Timing Analyzer

You can run Design Assistant directly from the Timing Analyzer to assist when optimizing timing paths and other timing conditions. When you launch Design Assistant from the Timing Analyzer, Design Assistant is preset to check only rules that relate to timing analysis.

Follow these steps to run the Design Assistant from the Timing Analyzer:

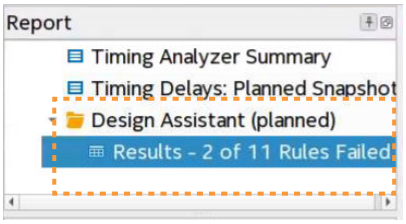
1. Compile the design through at least the Compiler's Plan stage.
2. Open the Timing Analyzer for the Compiler stage from the Compilation Dashboard.
3. In the Timing Analyzer, click **Reports ➤ Design Assistant ➤ Report DRC...**. The **Report DRC** (design rule check) dialog box opens.
4. Under **Rules**, disable any rules that are not important to your analysis by removing the check mark.
5. Consider whether to adjust rule parameter values in the **Parameters** field.

Figure 82. Report DRC (Design Rule Check) Dialog Box



6. Confirm the **Report panel name** and optionally specify an output **File name**.
7. Click **Run**. The Results reports generate and appear in the **Report** pane, as well as the main Compilation Report.

Figure 83. Design Assistant Reports in Timing Analyzer Report Pane

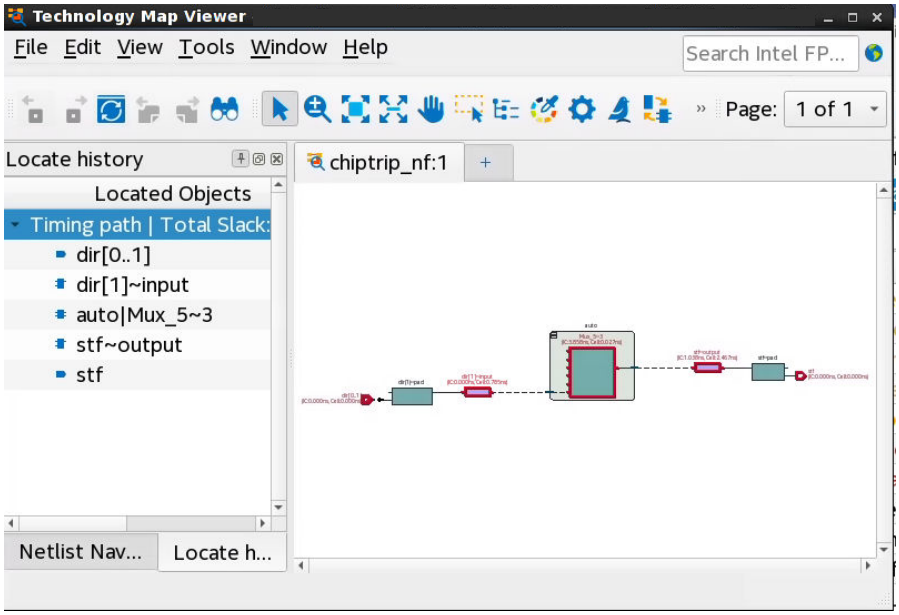


2.5.4. Locating Timing Paths in Other Tools

You can locate from paths and elements in the Timing Analyzer to other visualization tools in the Intel Quartus Prime software, such as the Chip Planner, Technology Map Viewer, or Resource Property Viewer.

You can right-click most paths or node names in the Timing Analyzer reports and click the **Locate Node** 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.

Figure 84. Locate Path from Timing Analyzer to Technology Map Viewer



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 1. Locating from the Timing Analyzer

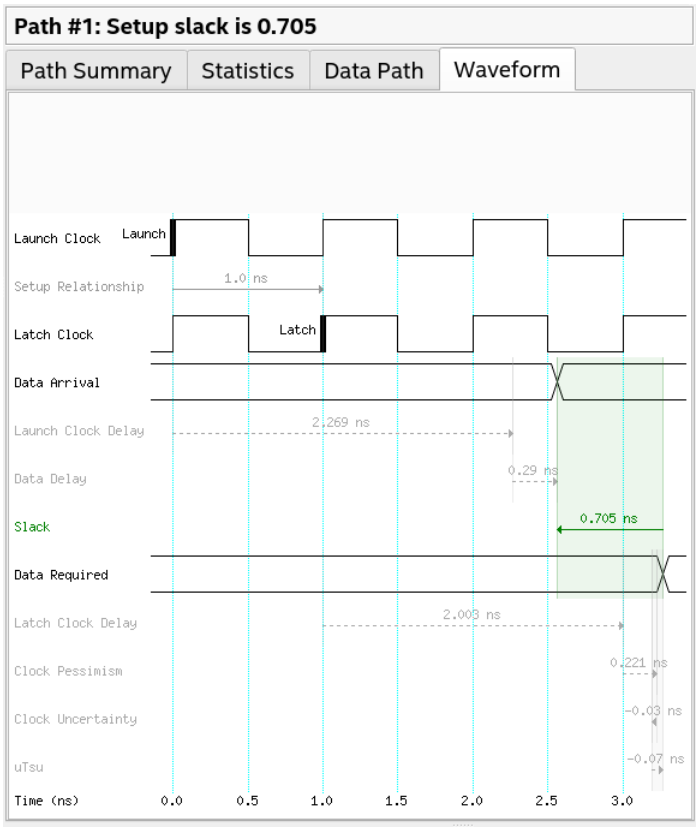
```
# Locate in the Technology Map Viewer the ten paths with the worst slack
locate [get_timing_paths -npaths 10] -tmv
# locate all ports that begin with data in the Chip Planner
locate [get_ports data*] -chip
```

2.5.5. 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 relationships. 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. You can view the incremental delays on the **Data Path** and **Waveform** tabs after running **Report Timing**. The **Waveform** tab allows you to visually reference the **Data Path** data, as well as the original `.sdc` constraints. You can use the **Waveform** tab to easily see how and where the constraints apply.

Figure 85. Report Timing (Waveform Tab)



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 86. Setup Relationship 10ns

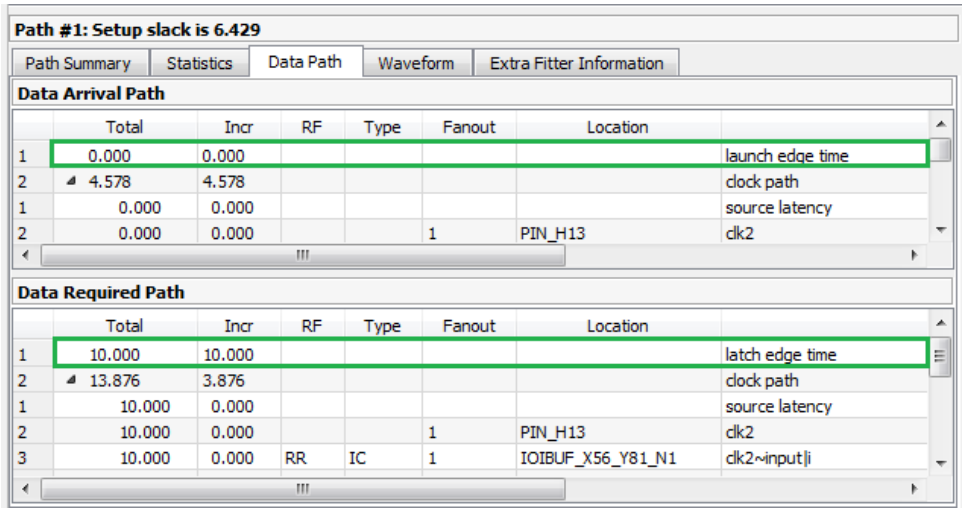
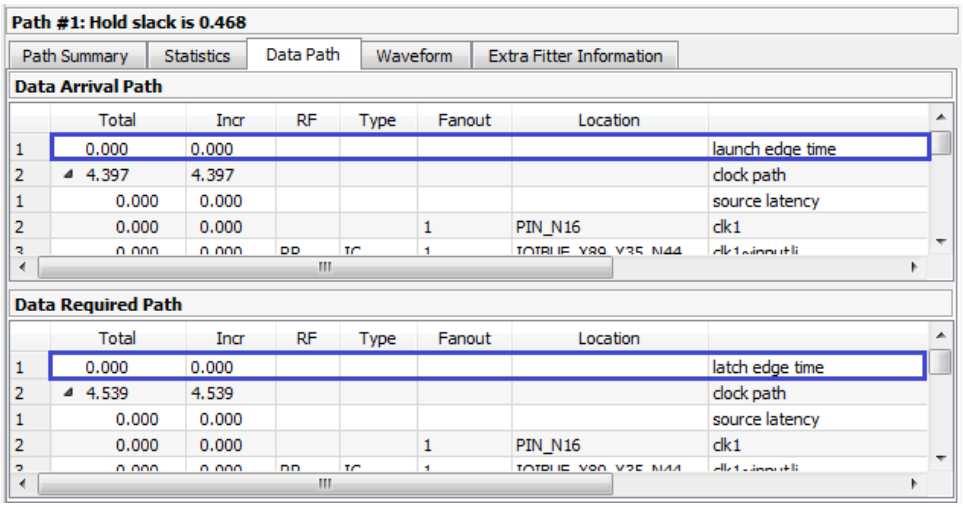


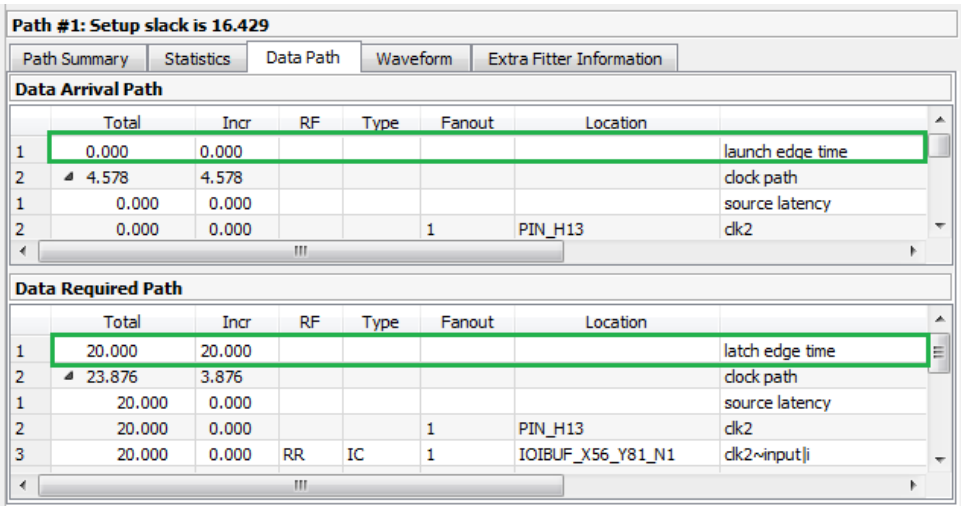
Figure 87. Hold Relationship 0ns



Adding `set_multicycle_path` constraints adds multicycles to relax the setup relationship, or open the window, making the setup relationship 20 ns while maintaining the hold relationship at 0 ns:

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

Figure 88. Setup Relationship 20ns

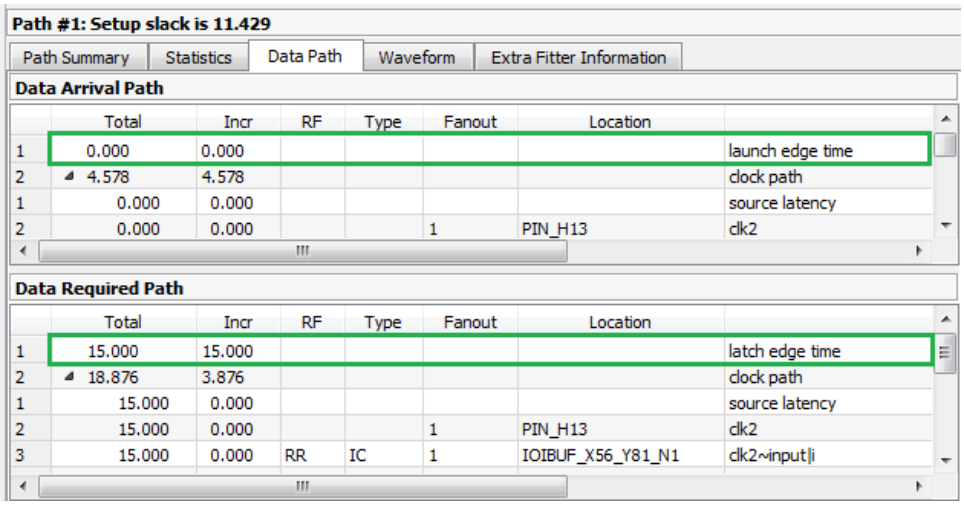


Adding the following `set_max_delay` constraints explicitly overrides the setup relationship:

```
set_max_delay -from [get_registers {regA}] -to \
     [get_registers {regB}] 15
```

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 89. Using set_max_delay



For I/O, you must add `set_input_delay` and `set_output_delay` constraints, as the following example shows. These constraints describe delays on signals from outside of the FPGA design that connect to the design's I/O ports.

```
create_clock -period 10 [get_ports clk]
# Clock used by the transfer, clock relationship is 10ns

# Setup constraints
set_output_delay -clock clk -max 1.2 [get_ports out]
# Subtracted from Data Required Path as oExt
set_max_delay -from [get_registers B] 12
# Sets latch edge time

# Hold constraints
set_output_delay -clock clk -min 2.3 [get_ports out]
# Subtracted from Data Required Path as oExt
set_min_delay -from [get_registers B] 8
# Sets latch edge time
```

The values of these constraints are the delays of the external signals between an external register and a port on the design. The `-clock` argument to the `set_input_delay` and `set_output_delay` specifies the clock domain that the external signal belongs to, or rather, the clock domain of the external register connected to the I/O port. The `-min` and `-max` options specify the worst-case or best-case delay; not specifying either option causes the worst- and best-case delays to be equal. I/O delays display as **iExt** or **oExt** in the **Type** column, as the following example reports shows.

Figure 90. Setup Slack Path Report and Waveforms for a Reg-To-Output Same-Clock Transfer

Path #1: Setup slack is 3.027

Path Summary | Statistics | Data Path | Waveform

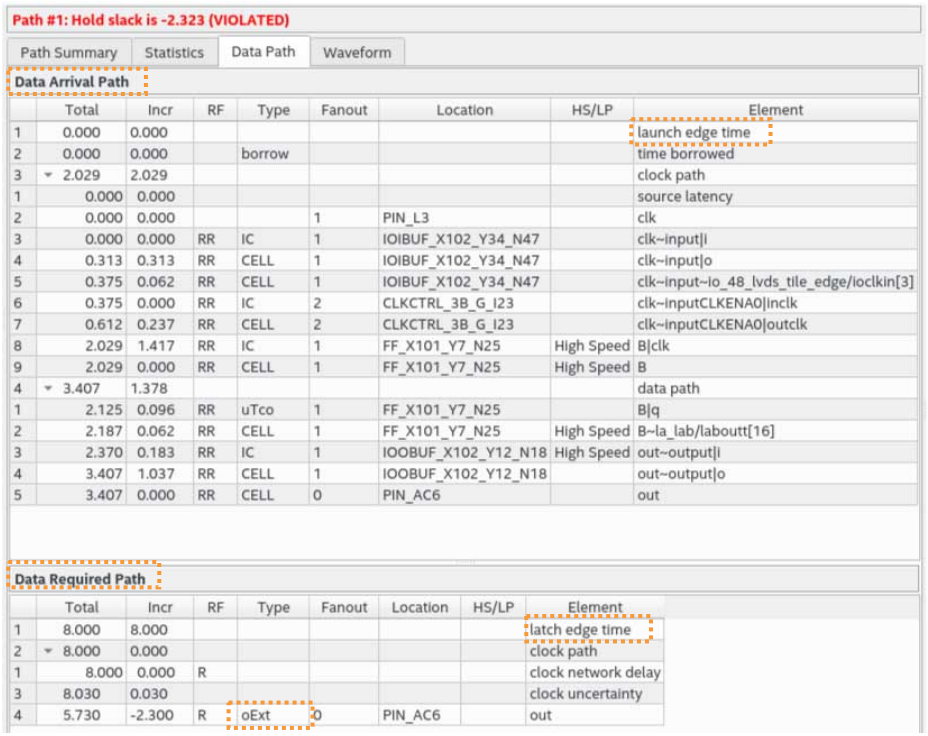
Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	0.000	0.000						launch edge time
2	0.000	0.000		borrow				time borrowed
3	4.088	4.088						clock path
1	0.000	0.000						source latency
2	0.000	0.000			1	PIN_L3		clk
3	0.000	0.000	RR	IC	1	IOIBUF_X102_Y34_N47		clk~input|i
4	0.675	0.675	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input|o
5	0.856	0.181	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input~io_48_lvds_tile_edge/ioclkin[3]
6	0.856	0.000	RR	IC	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|inclk
7	1.476	0.620	RR	CELL	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|outclk
8	4.088	2.612	RR	IC	1	FF_X101_Y7_N25	High Speed	B|clk
9	4.088	0.000	RR	CELL	1	FF_X101_Y7_N25	High Speed	B
4	7.743	3.655						data path
1	4.322	0.234	FF	uTco	1	FF_X101_Y7_N25		B|q
2	4.450	0.128	FF	CELL	1	FF_X101_Y7_N25	High Speed	B~la_lab/laboutt[16]
3	5.012	0.562	FF	IC	1	IOOBUF_X102_Y12_N18	High Speed	out~output|i
4	7.743	2.731	FF	CELL	1	IOOBUF_X102_Y12_N18		out~output|o
5	7.743	0.000	FF	CELL	0	PIN_AC6		out

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	12.000	12.000						latch edge time
2	12.000	0.000						clock path
1	12.000	0.000	R					clock network delay
3	11.970	-0.030						clock uncertainty
4	10.770	-1.200	F	oExt	0	PIN_AC6		out

Figure 91. Hold Slack Path Report and Waveforms for a Reg-To-Output Same-Clock Transfer



Path #1: Hold slack is -2.323 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	0.000	0.000						launch edge time
2	0.000	0.000		borrow				time borrowed
3	2.029	2.029						clock path
1	0.000	0.000						source latency
2	0.000	0.000			1	PIN_L3		clk
3	0.000	0.000	RR	IC	1	IOIBUF_X102_Y34_N47		clk~input|i
4	0.313	0.313	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input|o
5	0.375	0.062	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input~io_48_lvds_tile_edge/ioclkin[3]
6	0.375	0.000	RR	IC	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|inclk
7	0.612	0.237	RR	CELL	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|outclk
8	2.029	1.417	RR	IC	1	FF_X101_Y7_N25	High Speed	B|clk
9	2.029	0.000	RR	CELL	1	FF_X101_Y7_N25	High Speed	B
4	3.407	1.378						data path
1	2.125	0.096	RR	uTco	1	FF_X101_Y7_N25		B|q
2	2.187	0.062	RR	CELL	1	FF_X101_Y7_N25	High Speed	B~la_lab/laboutt[16]
3	2.370	0.183	RR	IC	1	IOOBUF_X102_Y12_N18	High Speed	out~output|i
4	3.407	1.037	RR	CELL	1	IOOBUF_X102_Y12_N18		out~output|o
5	3.407	0.000	RR	CELL	0	PIN_AC6		out

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	8.000	8.000						latch edge time
2	8.000	0.000						clock path
1	8.000	0.000	R					clock network delay
3	8.030	0.030						clock uncertainty
4	5.730	-2.300	R	oExt	0	PIN_AC6		out

A clock relationship, which is the difference between the launching and latching clock edge of a transfer, is determined by the clock waveform, multicycle constraints, and minimum and maximum 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 subtracts the `-max` value, making the setup relationship harder to meet, since the **Data Arrival Path** delay must be shorter than the **Data Required Path** delay. The Timing Analyzer also subtracts the `-min` value. This subtraction is why a negative number causes more restrictive hold timing. The **Data Arrival Path** delay must be longer than the **Data Required Path** delay.

Related Information

- Relaxing Setup with Multicycle (set_multicyle_path) on page 110
- Creating Virtual Clocks on page 88
- Creating I/O Constraints on page 101

2.6. Applying Timing Constraints

The following section describes correct application of SDC timing constraints that guide design synthesis, Fitter placement, and produce 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.

2.6.1. 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
```

Note: Only Intel Arria 10 and Intel Cyclone® 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

Create Clock (create_clock) on page 77

Derive PLL Clocks (derive_pll_clocks) on page 78

Derive Clock Uncertainty (derive_clock_uncertainty) on page 79

Set Clock Groups (set_clock_groups) on page 79

2.6.1.1. 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 `sys_clk` example 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`.

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 86

2.6.1.2. 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.

Note: Only Intel Arria 10 and Intel Cyclone 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

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.

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 86
- Deriving PLL Clocks on page 92

2.6.1.3. 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 an information message if you omit `derive_clock_uncertainty` from the `.sdc` file.

Related Information

Accounting for Clock Effect Characteristics on page 98

2.6.1.4. Set Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you to 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 the clock signals to include in each Group (-group), then specify the relationship between different groups, 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>}
```

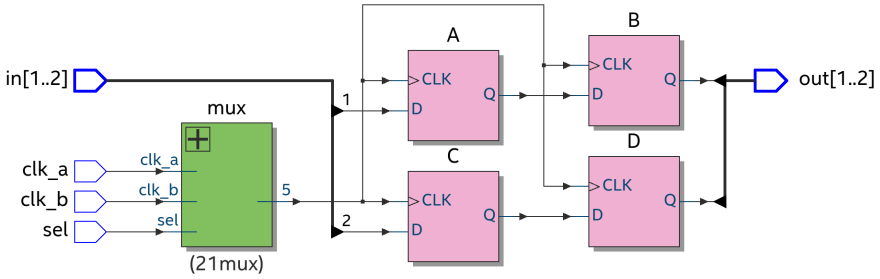
- `-logically_exclusive`—defines clocks that are logically exclusive and not active at the same time, such as multiplexed clocks
- `-physically_exclusive`—defines clocks that that cannot be physically on the device at the same time.
- `-asynchronous`—defines completely unrelated clocks that have different ideal clock sources. This 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.

The following shows example constraints for a clock mux with two inputs, with multi-rate clocks, and the appropriate combinations of logical and physical exclusivity:

```
# First profile
create_clock -name clk_a1 -period 10 [get_ports clk_a]
create_clock -name clk_b1 -period 20 [get_ports clk_b]

# Second profile
create_clock -name clk_a2 -period 100 [get_ports clk_a] -add
create_clock -name clk_b2 -period 200 [get_ports clk_b] -add

# Mark base clocks as asynchronous to each other
set_clock_groups -asynchronous -group {clk_a?} -group {clk_b?}

# Define muxed clocks for each profile
set muxout [get_pins -compatibility_mode {mux*|combout}]
foreach profile {1 2} {
     set mux_clk_a "mux_clk_a${profile}"
     set mux_clk_b "mux_clk_b${profile}"

     create_generated_clock -name $mux_clk_a -source [get_ports clk_a] \
          -master_clock "clk_a${profile}" $muxout -add
     create_generated_clock -name $mux_clk_b -source [get_ports clk_b] \
          -master_clock "clk_b${profile}" $muxout -add

     # Mark each muxed clock as logically exclusive to each other
     set_clock_groups -logically_exclusive -group $mux_clk_a \
          -group $mux_clk_b
}

# Mark profile clocks as physically exclusive to each other
# (Do this after defining the derived clocks so they get cut too)
set_clock_groups -physically_exclusive -group {*clk_?1} \
     -group {*clk_?2}
```

Figure 92. Example Constraints Design Topology



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 for crosstalk-related analysis, but treats asynchronous and exclusive clock groups differently for things like max skew reporting and synchronizer detection.

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 cut timing between clocks with less than ten lines, but the `set_false_path` command may require more than 50 lines.

Related Information

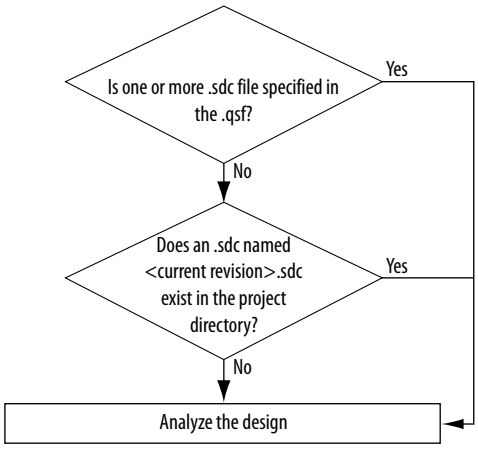
- Creating Generated Clocks (create_generated_clock) on page 90
- Relaxing Setup with Multicycle (set_multicyle_path) on page 110
- Accounting for a Phase Shift (-phase) on page 111

2.6.2. 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.

Note: Intel FPGA IP packages RTL and the SDC constraints for the IP within a single `.ip` file. Therefore, within a project's .qsf there may be references to SDC constraints packaged with the containing `.ip` file.

Figure 93. .sdc File Order of Precedence



Click **Settings ➤ Timing Analyzer** to add, remove, or change the processing order of `.sdc` files in the project, as Step 1: Specify Timing Analyzer Settings on page 24 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.

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 reads the `.sdc` files following the `.sdc` file precedence order.

2.6.3. Modifying Iterative Constraints

Iteratively modify `.sdc` constraints and reanalyze the timing results to ensure that you have complete constraints for your design.

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, summaries for supported reports, and a list of all the defined clocks in the design. These summaries cover all paths you constrain in your design. Whenever modifying or correcting constraints, generate the **Constraint 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 you see any failing timing against the new constraints, run place-and-route again.

Related Information

Relaxing Setup with Multicycle (set_multicyle_path) on page 110

2.6.4. Using Entity-bound SDC Files

The Intel Quartus Prime Pro Edition Timing Analyzer supports the assignment of a Synopsys Design Constraints (`.sdc`) file to a specific design entity (module) in your project.

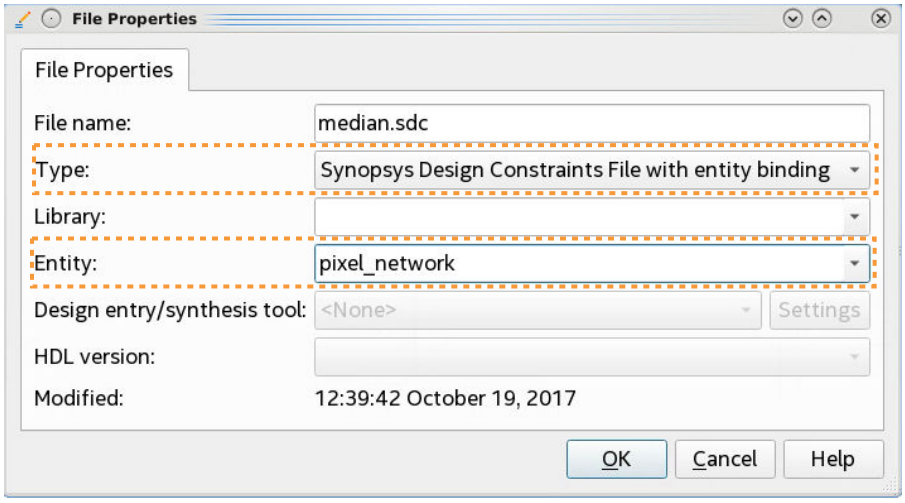
Normally, timing constraints that you specify in an `.sdc` file apply globally in your project, rather than to any particular design entity. However, you can use the **Properties** dialog box or `SDC_ENTITY_FILE` assignment to bind an `.sdc` file to a design entity. Entity-bound constraints make timing constraints more portable, and allow you to specify more targeted constraints.

- Timing constraint portability—all design partitions that contain the assigned entity automatically include entity-bound `.sdc` constraints. You can optionally specify export of these constraints with a partition you export with the **Include entity-bound SDC files** (`--include_sdc_entity_in_partition`) option. This allows you to hand-off synthesized or final design blocks or IP packaged with timing constraints.
- Timing constraint precision—Apply timing constraints only to specific entities rather than globally, simplifying constraint entry. This method avoids unintended side effects of global constraints that apply to more than you intend, especially when using wildcard (*) timing constraints. By default, entity-bound `.sdc` constraints apply automatically to all instances of the assigned entity in the project. Alternatively, you can apply all of the constraints globally by default, and choose which of its constraints target only the current entity, by using the `get_current_instance` command, as Entity-bound Constraint Scope on page 84 describes.

Follow these steps to create or modify an entity-bound `.sdc` file:

1. Create an `.sdc` file, click **Project ➤ Add/Remove files in project**, and add the `.sdc` file. The `.sdc` file appears in the **Files** list.
2. In the **Files** list, select the `.sdc` file and click the **Properties** button.
3. For **Type**, select **Synopsys Design Constraints File with entity binding**.

Figure 94. Entity Rebinding



4. For **Entity**, select the entity you want to bind to the `.sdc`.
5. Click **OK**.

Alternatively, add the following assignment to the `.qsf` to bind the entity you specify to the `.sdc` file you specify:

QSF Assignment Syntax:

```
set_instance_assignment -entity <entity_name> -name \
    SDC_ENTITY_FILE <sdc_file_name> [-no_sdc_promotion] \
    [–no_auto_inst_discovery]
```

For an SDC_ENTITY_FILE assignment, you must specify a `-library` parameter that matches the `-library` argument of the referenced entity. If you specify no library, the software uses the default `altera_work` library.

Both the entity and the library must match the target RTL module for the SDC_ENTITY_FILE to be applied.

The following options apply to the SDC_ENTITY_FILE assignment:

- No options—enable automatic constraint scoping. The Timing Analyzer reads in entity `.sdc` files once for each bound instance, and `get_current_instance` returns a value. The collection filters (except for clock and top-level port filters) get prepended with the hierarchy path of the current instance (that is, the return value of `get_current_instance`).
- `-no_sdc_promotion`—turns on manual promotion. The Timing Analyzer reads in entity `.sdc` files once for each bound instance, and `get_current_instance` returns a value. The Timing Analyzer does not modify the collection filters.
- `-no_sdc_promotion –no_auto_inst_discovery`—turns off constraint scoping. The Timing Analyzer reads in each entity `.sdc` file once, and does not modify the collection filters. `get_current_instance` returns an empty string.

2.6.4.1. Entity-bound Constraint Scope

Entity-bound `.sdc` files can have an automatic or manual scope in your project. The scope determines how widely the constraints apply. Automatic scoping applies by default.

Table 22. Entity-bound Constraint Scope

Constraint Scope Type	Constraints Apply	To Enable Instance-bound Scoping
Automatic	To all instances of the assigned entity throughout the project, except for top-level ports (`get_ports`) and clock names (`get_clocks`).	Default mode for SDC_ENTITY_FILE. No additional steps required.
Manual	To the current instance of the assigned entity, except for top-level ports and clock names, which have a global scope. Collection filters also have global scope, unless you prepend them with `get_current_instance`, which sets the instance scope.	Prepend the collection filter with `get_current_instance`.

The following example constraint shows use of `get_current_instance` to return the hierarchical path to the current entity for manual constraint scoping:

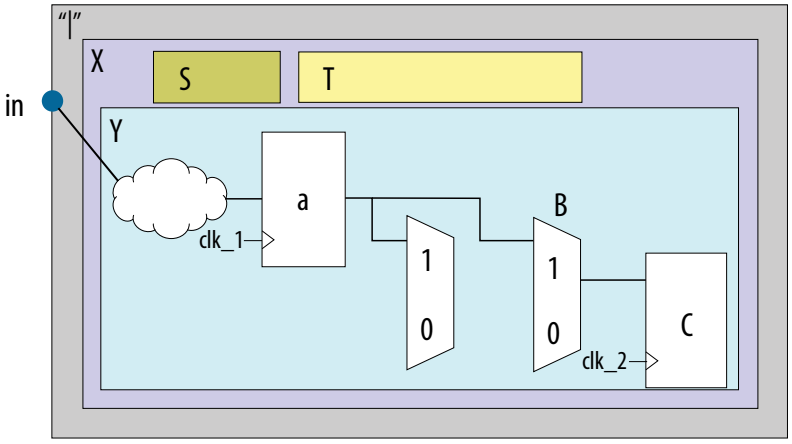
```
set_false_path –from [get_registers "reg_a"] –to \
     [get_pins “[get_current_instance]|*reset”]
```

Note: If you use the `-from *` or `-to *` options without using one of the `get_` commands (such as `get_keepers`), no constraint scoping occurs on those filters (that is to say, scoping is not done on from/to collection filters of *, but scoping can still occur on other collection filters in the same SDC command).

2.6.4.2. Entity-bound Constraint Examples

The following examples show the automatic and manual scope of entity-bound constraints.

Figure 95. Automatic Scope Example



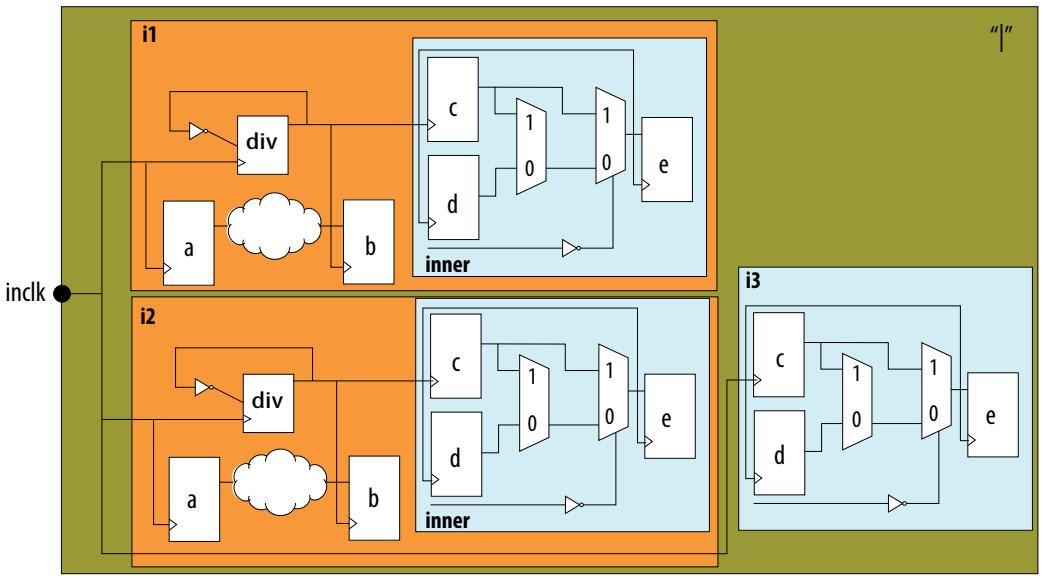
The following table illustrates the automatic scope of constraints as they apply to Automatic Scope Example.

Table 23. Automatic Constraint Scoping Examples

Constraint Example	Auto-Scope Constraint Interpretation for Instance X|Y
`set_false_path -from [get_keepers a]`	`set_false_path -from [get_keepers` **`X|Y|`**`a]`
`set_false_path -from [get_registers a] -to "*"`	`set_false_path -from [get_registers` **`X|Y|`**`a]`
`set_false_path –from [get_clocks clk_1] –to [get_clocks clk_2]`	`set_false_path –from [get_clocks` **`clk_1`**`] –to [get_clocks` **`clk_2`**`]`
`set_max_delay –from [get_ports in] -to [get_registers A] 2.0`	`set_max_delay –from [get_ports` **`in`**`] -to [get_registers` **`X|Y|`**`A] 2.0`
get_ports *	get_ports *
get_clocks *	get_clocks *
get_ports a	get_ports a
get_clocks a	get_clocks a

Note: In table Automatic Scope Example, `get_ports a` and `get_clocks a` are simply examples that use an arbitrary name for the collection filter. These examples show that collection filters for `get_ports` and `get_clocks` are not subject to automatic constraint scoping because the ports and clocks are global, top-level objects that are never in the scope of an instance.

Figure 96. Manual Scope Example



The following table illustrates the manual scope of constraints as they apply to Manual Scope Example.

Table 24. Manual Scope Constraint Examples

Constraint Example	Manual Scope Constraint Interpretation

`set_false_path -from [get_current_instance]|d\`
`  -to [get_current_instance]|e`
	`set_false_path -from i1|inner|d -to i1|inner|e`
	`set_false_path -from i2|inner|d -to i2|inner|e`
	`set_false_path -from i3|d -to i3|e`

`create_generated_clock -divide_by 2 -source \`
`  [get_ports inclk] -name \`
`  [get_current_instance]_divclk \`
`  [get_current_instance]|div`
`set_multicycle_path -from [get_current_instance]|a\`
`  -to [get_current_instance]|b 2`
	`create_generated_clock -divide_by 2 -source \`
	`  [get_ports inclk] -name "i1_divclk" i1|div`
	`set_multicycle_path -from i1|a -to i1|b 2 \`
	`create_generated_clock -divide_by 2 -source \`
	`  [get_ports inclk] -name "i2_divclk" i2|div`
	`set_multicycle_path -from i2|a -to i2|b 2`

2.6.5. 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

2.6.5.1. 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 base 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 100Mhz -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 10ns -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 10ns -name clk_100 [get_ports clk_sys]
create_clock -period 5ns -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 physically exclusive (that is, clocks that are not physically present at the same time).

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

Related Information

Accounting for Clock Effect Characteristics on page 98

2.6.5.1.1. 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
```

Caution: 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 realistic analysis of your design's timing requirements, do not use `derive_clocks` command for final timing sign-off. Instead, use `create_clock` and `create_generated_clock` commands to 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.

2.6.5.2. 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 clocks that drive external devices 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 100 MHz virtual clock because the command includes no *<targets>*.

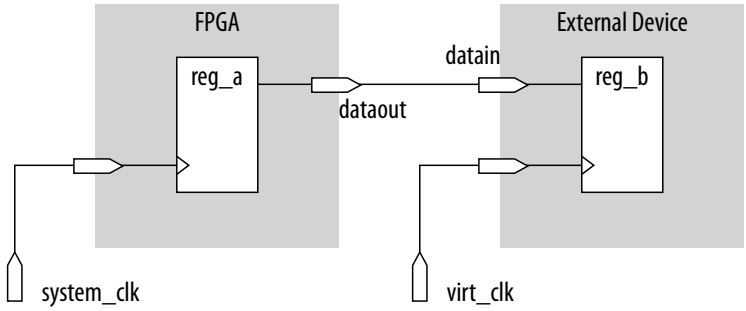
```
create_clock -period 10 -name my_virt_clk
```

I/O Constraints with Virtual Clocks

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. .

Figure 97. Virtual Clock Board Topology

The figure shows the base clock (`system_clk`), virtual clock (`virt_clk`), and output delay for the virtual clock constraints example



The following example 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 2. 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]
```

2.6.5.2.1. 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.

2.6.5.2.2. 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 3. 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]
```

2.6.5.3. 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.

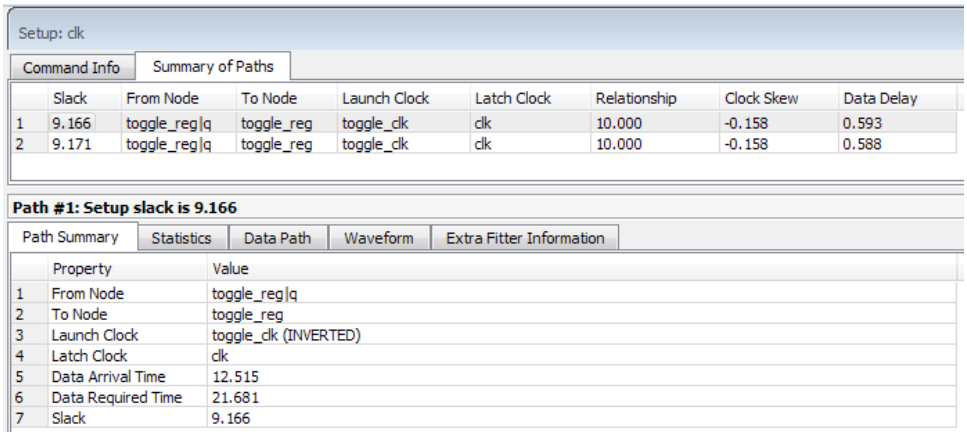
The `-source` option specifies the name of a node in the clock path that you use as reference for your generated clock. The source of the generated clock must be a node in your design netlist, and not the name of a clock you previously define. You can use any node name on the clock path between the input 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. By accepting a node as the generated clock's source clock, the generated clock constraint decouples from the source clock. If you change the source clock for the generated clock and the source node is the same, 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 source or target node 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 98. Example of clock-as-data


Setup: clk

Command Info | Summary of Paths

	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

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

	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 92
- create_generated_clock
- derive_pll_clocks

2.6.5.3.1. 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 10ns -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 10ns -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 99. Clock Divider

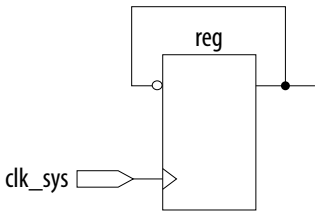
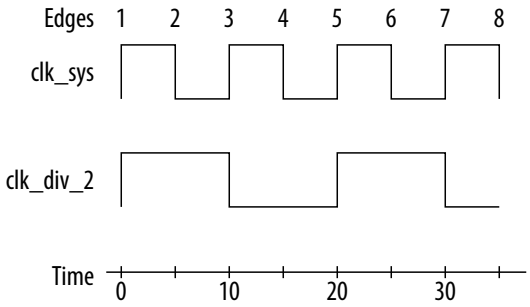


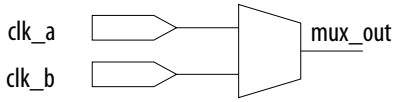
Figure 100. Clock Divider Waveform



2.6.5.3.2. Clock Multiplexer Example

The output of a clock multiplexer (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 101. 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 -logically_exclusive -group clock_a_mux -group clock_b_mux
```

2.6.5.4. 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.

Note: Only Intel Arria 10 and Intel Cyclone 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

Create Base Clock for PLL input Clock Ports

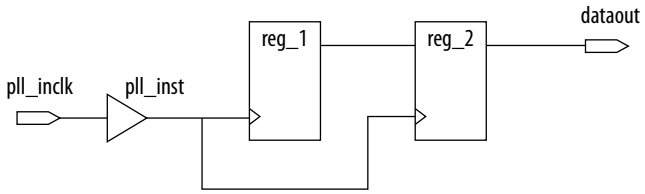
If your design contains transceivers, LVDS transmitters, or LVDS receivers, use the `derive_pll_clocks` to constrain this logic in your design and create timing exceptions for those blocks.

```
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 102. 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 4. 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 93

2.6.5.5. Creating Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you to specify which clocks in the design are unrelated.

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 clock 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 25. set_clock_groups -group A

Destination\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 26. set_clock_groups -group {A B}

Destination\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 27. set_clock_groups -group A -group B

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

Table 28. set_clock_groups -group {A C} -group {B D}

Destination\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 29. set_clock_groups -group {A C D}

Destination\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

Related Information

- Timing Exception Precedence on page 106
- set_clock_groups Command, Intel Quartus Prime Help

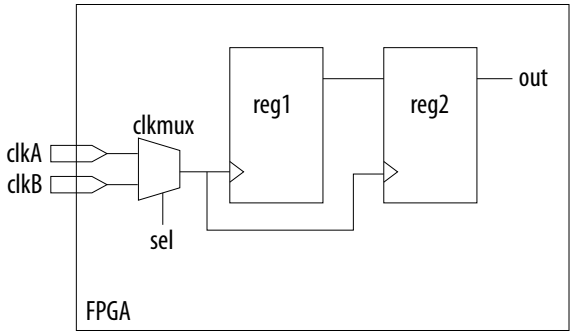
2.6.5.5.1. Exclusive Clock Groups (-logically_exclusive or -physically_exclusive)

You can use the `logically_exclusive` option to declare that two clocks are physically active simultaneously, but the two clocks are not actively used at the same time (that is, the clocks are logically mutually exclusive). The `physically_exclusive` option declares clocks that cannot be physically on the device at the same time.

If you define multiple clocks for the same node, you can use clock group assignments with the `logically_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 `logically_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 show:

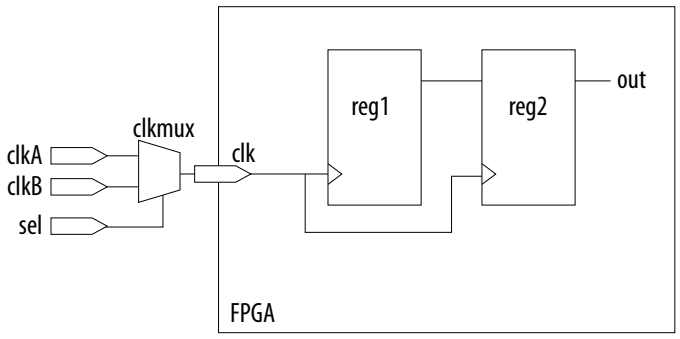
Figure 103. Synchronous Path with Clock Mux Internal to FPGA



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]
# Create derived clocks on the output of the mux
create_generated_clock -name mux_100 -source [get_ports clkA] \
     [get_pins clkmux|combout]
create_generated_clock -name mux_125 -source [get_ports clkB] \
     [get_pins clkmux|combout] -add
# Set the two clocks as exclusive clocks
set_clock_groups -logically_exclusive -group {mux_100} -group {mux_125}
```

Figure 104. Synchronous Path with Clock Mux External to FPGA



Example SDC Constraints for External Clock Mux

```
# Create virtual clocks for the external primary clocks
create_clock -period 10 -name clkA
create_clock -period 20 -name clkB
# Create derived clocks on the port clk
create_generated_clock -name mux_100 -master_clock clkA [get_ports clk]
create_generated_clock -name mux_125 -master_clock clkB [get_ports clk] -add
# Assume no clock network latency between the external clock sources & the \
     clock mux output
set_clock_latency -source 0 [get_clocks {mux_100 mux_125}]
# Set the two clocks as exclusive clocks
set_clock_groups -physically_exclusive -group mux_100 -group mux_125
```

2.6.5.5.2. 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.

2.6.5.5.3. 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 77, 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. 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. Typically the outputs of a PLL are related 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. Timing Analyzer assumes that the clocks not appearing in the clock groups command relate to every clock and conservatively groups the known clocks. If the design still has failing paths between unrelated clock domains, you can add the new clock domains as necessary. In this case, a large number of the clocks are not in the `set_clock_groups` command, because they are either cut in the `.sdc` file for the IP (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 IP.

Related Information

Multicycle Paths on page 109

2.6.5.6. 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.

2.6.5.6.1. 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.

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

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 can only characterize source latency with the `set_clock_latency` command using the `-source` option.

Related Information

set_clock_latency Command, Intel Quartus Prime Help

2.6.5.6.2. 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`.

Related Information

- set_clock_uncertainty, Intel Quartus Prime Help
- derive_clock_uncertainty, Intel Quartus Prime Help
- remove_clock_uncertainty, Intel Quartus Prime Help

2.6.5.7. Constraining CDC Paths

It is essential to apply timing constraints to the multibit clock domain crossing (CDC) paths in your design. You can use the following constraints to constrain CDC paths:

Attention: As of Intel Quartus Prime Pro Edition software version 21.3, the `set_false_path` constraint does not override the `set_max_skew` constraint. You can now apply the `set_false_path` and `set_max_skew` constraints on the same path without override.

Table 30. CDC Path Constraints

Constraints	Description
`set_false_path` `set_clock_group -asynchronous`	Both constraints prevent the Compiler from optimizing slack between asynchronous domain crossings. `set_clock_group` is the most aggressive constraint. • Clock-based false paths are less aggressive because these constraints only cut timing on the `from_clock` to `to_clock` order specified. • Clock-based false paths are unlike clock groups that cut the path in both directions. • Path-based false paths are the most specific constraint because they cut only on the specified `from` and `to` nodes.
`set_max_skew`	Sets a bound on the allowable skew between different bus bits. • Check the timing of your clock domain crossing by running the **Report Max Skew Summary** command. • The actual skew requirements depends on your design characteristics, and how you handle the clock domain crossing.
`set_net_delay -max` `set_data_delay`	Sets a bound on the allowable datapath delay on any bit of a bus transfer. • `set_net_delay` is for constraining individual clock edges and nets. Run the **Report Net Delay Summary** command to report data for this constraint. • `set_data_delay` is for constraining entire paths. Run the **Report Data Delay** command to report data for this constraint.

The following shows example constraints for a clock domain crossing between `data_a` in clock domain `clk_a`, and `data_b` in clock domain `clk_b`:

```
create_clock -name clk_a -period 4.000 [get_ports {clk_a}]
create_clock -name clk_b -period 4.500 [get_ports {clk_b}]
set_clock_groups -asynchronous -group [get_clocks {clk_a}] -group \
     [get_clocks {clk_b}]
set_net_delay -from [get_registers {data_a[*]}] -to [get_registers \
     {data_b[*]}] -max -get_value_from_clock_period \
     dst_clock_period -value_multiplier 0.8
set_max_skew -from [get_keepers {data_a[*]}] -to [get_keepers \
     {data_b[*]}] -get_skew_value_from_clock_period min_clock_period \
     -skew_value_multiplier 0.8
```

The following examples show applying `set_false_path` for a design that contains a DCFIFO block to avoid timing failures in the synchronization registers. These examples are for constraining single-bit synchronizer CDC paths:

- For paths crossing from the write into the read domain, apply a false path assignment between registers `delayed_wrptr_g` and `rs_dgwp`:

```
set_false_path -from [get_registers {*dcfifo*delayed_wrptr_g[*]}] \
-to [get_registers {*dcfifo*rs_dgwp*}]
```

- For paths crossing from the read into the write domain, apply a false path assignment between registers `rdptr_g` and `ws_dgrp`:

```
set_false_path -from [get_registers {*dcfifo*rdptr_g[*]}] \
-to [get_registers {*dcfifo*ws_dgrp*}]
```

2.6.6. 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.

2.6.6.1. Input Constraints (set_input_delay)

Input constraints specify delays for all external signals feeding the FPGA. 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 105. Input Delay Diagram

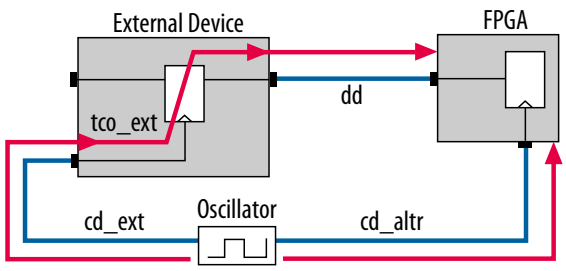


Figure 106. Input Delay Calculation

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

If your design includes partition boundary ports, you can use the `-blackbox` option with `set_input_delay` to assign input delays. The `-blackbox` option creates a new keeper timing node with the same name as the boundary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_input_delay` constraint. The new keeper timing nodes display when you use the `get_keepers` command. You can remove these black box constraints with `remove_input_delay -blackbox`.

2.6.6.2. 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 107. Output Delay Diagram

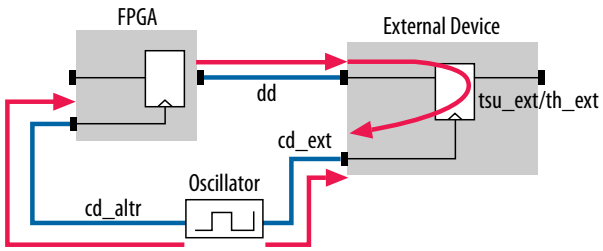


Figure 108. Output Delay Calculation

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

If your design includes partition boundary ports, you can use the `-blackbox` option with `set_ouput_delay` to assign output delays. The `-blackbox` option creates a new keeper timing node with the same name as the boundary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_output_delay` constraint. The new keeper timing nodes display when you use the `get_keepers` command.

You can remove blackbox constraints with `remove_output_delay -blackbox`.

Related Information

- set_input_delay Command, Intel Quartus Prime Help
- set_output_delay Command, Intel Quartus Prime Help

2.6.7. Creating Delay and Skew Constraints

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

2.6.7.1. 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 5. Forcing Delay Annotation

```
create_timing_netlist -force_dat
```

2.6.7.2. 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 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` constraints do not affect the `set_max_skew` timing constraint. However, the `set_clock_groups` constraint does impact the `set_max_skew` constraint.

Note: Exclusive clock groups (set with `set_clock_group -exclusive`) override `set_max_skew` constraints.

The Timing Analyzer does not compare two paths for skew if their clocks are exclusive to each other. However, the Timing Analyzer does analyze for skew paths whose clocks are asynchronous.

Table 31. set_max_skew Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-fall_from_clock <names>`	Valid source clocks (Tcl matches string patterns). Analysis only considers paths from falling clock edges.
`-fall_to_clock <names>`	Valid destination clocks (Tcl matches string patterns). Analysis only considers paths from falling clock edges.

continued...

Arguments	Description
`-from <names>`[3]	Valid sources (Tcl matches string patterns).
`-from_clock <names>`	Valid source clocks (Tcl matches string patterns).
`-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.
`-rise_from_clock <names>`	Valid source clocks (Tcl matches string patterns). Analysis only considers paths from rising clock edges.
`-rise_to_clock <names>`	Valid destination clocks (Tcl matches string patterns). Analysis only considers paths to rising clock edges.
`-skew_value_multiplier <multiplier>`	Value by which the clock period multiplies to compute skew requirement.
`-to <names>`[3]	Valid destinations (Tcl matches string patterns)
`-to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
<skew>	The value of the 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).

Maximum skew analysis can include data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation, and clock pessimism removal. Among these, the Fitter only disables clock pessimism removal by default.

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.

Related Information

- Timing Exception Precedence on page 106
- report_max_skew Command, Intel Quartus Prime Help

2.6.7.3. 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. A net delay constraint is invalid if there is a combinational cell between the From and To nodes.

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

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 32. set_net_delay Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-from <names>`(4)	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>`(5)	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_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 not set the positional delay option. If more than one clock clocks the set of nets, the Timing Analyzer uses the net with the smallest period to compute the constraint for a `-max` constraint, and the largest period for a `-min` constraint. If no clocks are clocking the endpoints of the net (that is, if the endpoints of the nets are not registers or constraint ports), the Timing Analyzer ignores the net delay constraint.

Related Information

- Timing Exception Precedence on page 106
- report_net_delay Command, Intel Quartus Prime Help

(4) If option is a wildcard ("*") character, all the output pins and registers on timing netlist become valid source points.

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

2.6.8. 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.

2.6.8.1. Timing Exception Precedence

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

1. **Set False Path** (`set_false_path`) is the first priority. False paths and clock groups have identical priority, except when you use the `-latency_insensitive` or `-no_synchronizer` options with a false path exception. With either option, the false path has priority over a clock group.
2. **Set Clock Groups** (`set_clock_groups`) is the second priority.
3. **Set Minimum Delay** (`set_min_delay`) and **Set Maximum Delay** (`set_max_delay`) are the third priority.
4. **Set Multicycle Path** (`set_multicycle_path`) is the fourth 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.

The `set_net_delay`, `set_max_skew`, and `set_data_delay` constraints analyze independently of minimum or maximum delays, or multicycle path constraints.

- The `set_net_delay` exception applies regardless of the existence of a `set_false_path` exception, or `set_clock_groups` exception, or other path-based constraint or exception. It is a net-based exception, and net-based and path-based exceptions are applied independently of each other.
- The `set_max_skew` exception applies on paths cut by an asynchronous clock group, and regardless of any `set_false_path` exception. Exclusive clock groups override max skew exceptions, because paths between exclusive clocks are entirely inactive and should not be analyzed for timing or skew requirements. This precedence allows you to define more targeted constraints on asynchronous CDC bus transfers.
- The `set_data_delay` exception specifies a maximum datapath delay exception for a given path. Exclusive clock groups override data delay exceptions, because paths between exclusive clocks are entirely inactive and should not be analyzed for timing or data delay requirements. Asynchronous clock groups do not override data delay exceptions. False path exceptions override data delay exceptions in the Intel Quartus Prime Pro software version 21.2 and earlier. Beginning in version 21.3, false path exceptions do not override data delay exceptions. This change in precedence allows you to write more targeted constraints on asynchronous CDC bus transfers.

Related Information

- False Paths (set_false_path) on page 107
- Creating Clock Groups (set_clock_groups) on page 93
- Constraining CDC Paths on page 100
- Creating Clock Groups (set_clock_groups) on page 93
- Maximum Skew (set_max_skew) on page 103
- Minimum and Maximum Delays on page 108
- Net Delay (set_net_delay) on page 104
- Report Data Delay on page 40

2.6.8.2. 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 starting with A, to all registers starting 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. A false path's `-from` and `-to` targets can be either nodes or clocks. However, the `-thru` targets can only be combinational nodes. For example, you can specify a false path for a static configuration register that writes 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

- Timing Exception Precedence on page 106
- Creating Clock Groups (set_clock_groups) on page 93
- Constraining CDC Paths on page 100
- set_false_path Command, Intel Quartus Prime Help

2.6.8.3. 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.

Related Information

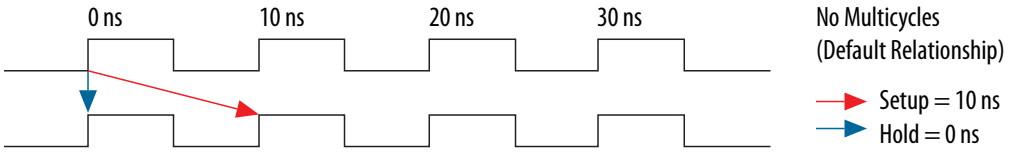
- Timing Exception Precedence on page 106

- set_max_delay Command, Intel Quartus Prime Help
- set_min_delay Command, Intel Quartus Prime Help

2.6.8.4. 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.

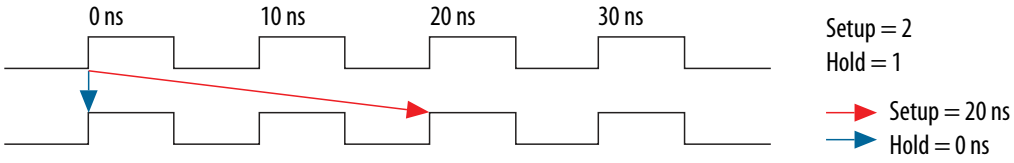
Figure 109. Default Setup and Hold Relationship (No Multicycle)



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.

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 110. 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 default 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 33. 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.

2.6.8.4.1. 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.

2.6.8.4.2. Relaxing Setup with Multicycle (set_multicyle_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.

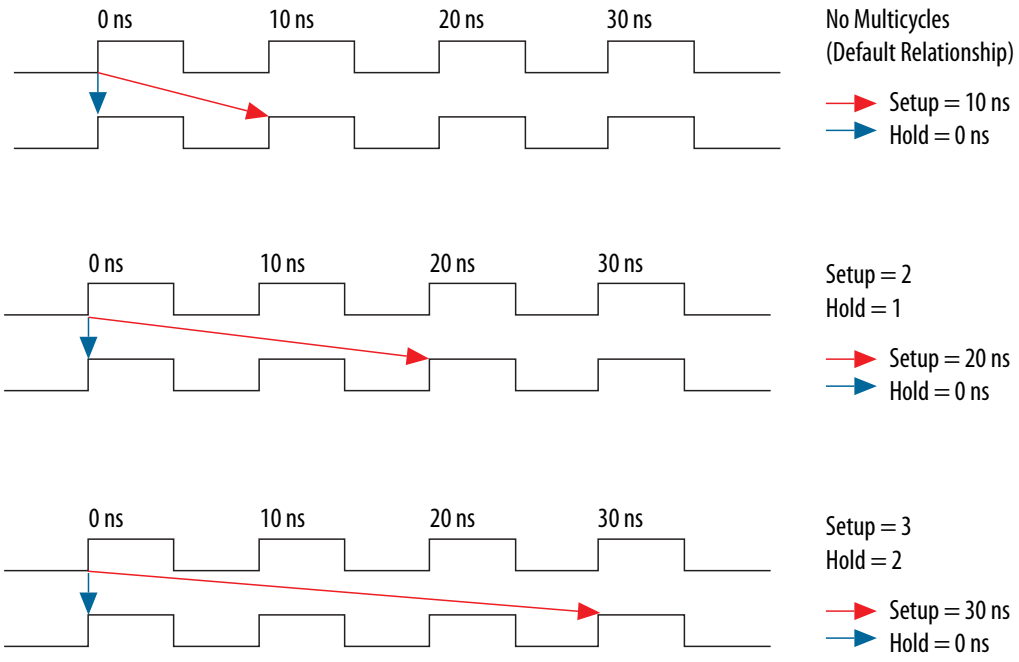
In the following example, the source clock has a period of 10 ns, but the clock enable signal controls a group of latching registers, so the registers only enable every other cycle. The 10 ns clock feeds registers , so the Timing Analyzer reports a setup of 10 ns and a hold of 0 ns. However, the data is transferring every other cycle, so the Timing Analyzer must analyze the relationships as if the clock is operating at 20 ns. The result is 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 111. 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
```

2.6.8.4.3. 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.

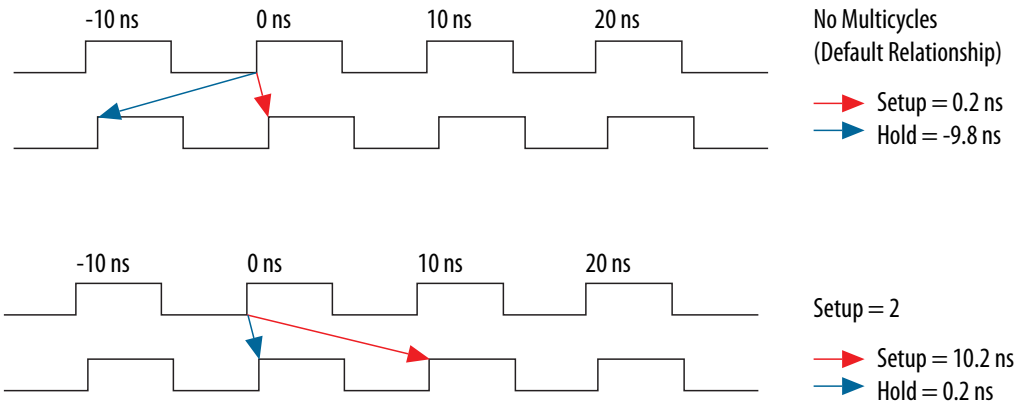
For example, the following code phase-shifts one output of a PLL forward by a small amount, in this case 0.2 ns.

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 112. 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 120
- set_multicycle_path Command, Intel Quartus Prime Help

2.6.8.5. 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.

2.6.8.5.1. 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 113. Default Timing Diagram

The timing waveforms show the source and destination registers of a data transfer. HC1 and HC2 are the hold checks that Timing Analyzer performs. SC is the setup check that Timing Analyzer performs.

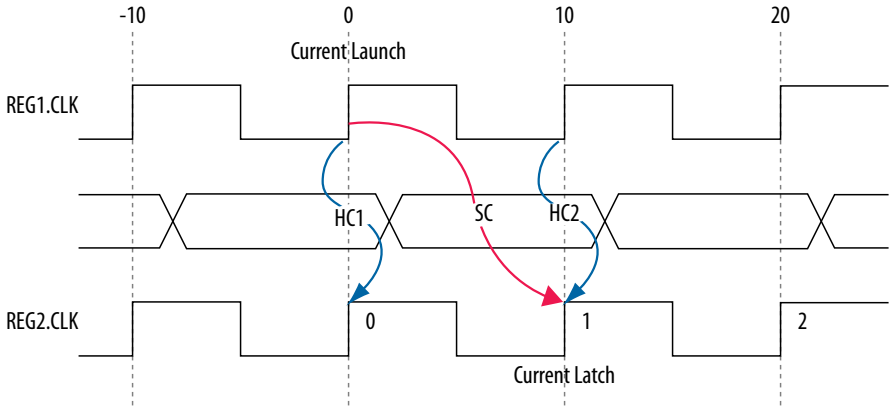


Figure 114. Setup Check Calculation

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

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

Figure 115. Default Setup Report

Path #1: Setup slack is 8.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	▾ 2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	▾ 3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	10.000	10.000			latch edge time
2	10.000	0.000		borrow	time borrowed
3	▾ 11.983	1.983			clock path
1	11.983	1.983	R		clock network delay
4	11.953	-0.030			clock uncertainty
5	11.996	0.043		uTsu	dst

Figure 116. Hold Check Calculation

The figure shows the setup timing report with the launch and latch edge times highlighted.

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

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

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

Figure 117. Default Hold Report

The figure shows the hold timing report with the launch and latch edge times highlighted.

Path #1: Hold slack is 0.230

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			latch edge time
2	0.000	0.000		borrow	time borrowed
3	1.134	1.134			clock path
1	1.134	1.134	R		clock network delay
4	1.164	0.030			clock uncertainty
5	1.299	0.135		uTh	dst

2.6.8.5.2. 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 118. Setup Timing Diagram

The figure shows the setup timing diagram for the analysis that the Timing Analyzer performs. Without the multicycle constraint the latching edge is edge 1. However, with the multicycle constraint the latching edge is edge 2.

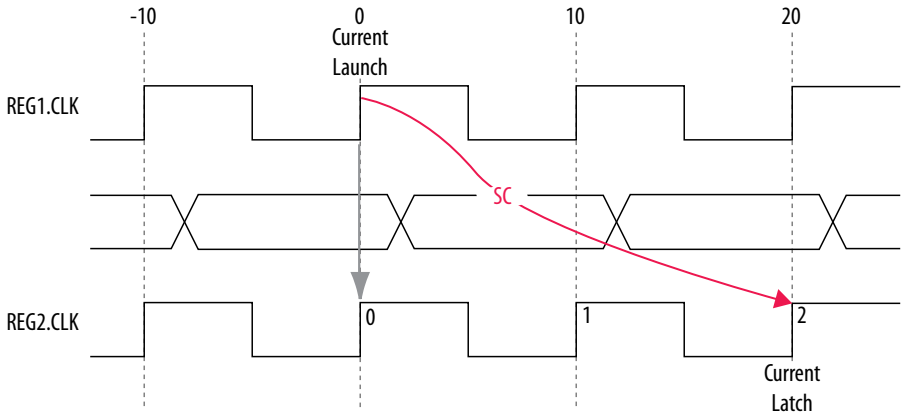


Figure 119. Setup Check Calculation

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

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 120. Setup Report with Setup Multicycle Exception

Path #1: Setup slack is 18.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	20.000	20.000			latch edge time
2	20.000	0.000		borrow	time borrowed
3	21.983	1.983			clock path
1	21.983	1.983	R		clock network delay
4	21.953	-0.030			clock uncertainty
5	21.996	0.043		uTsu	dst

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 121. Hold Timing Diagram

The figure shows the hold latching edges are now at 10 and 20 ns, instead of 0 and 10 ns.

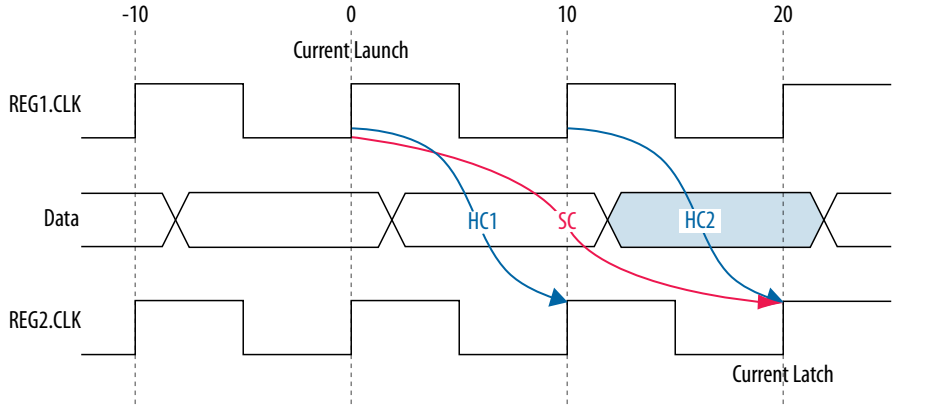
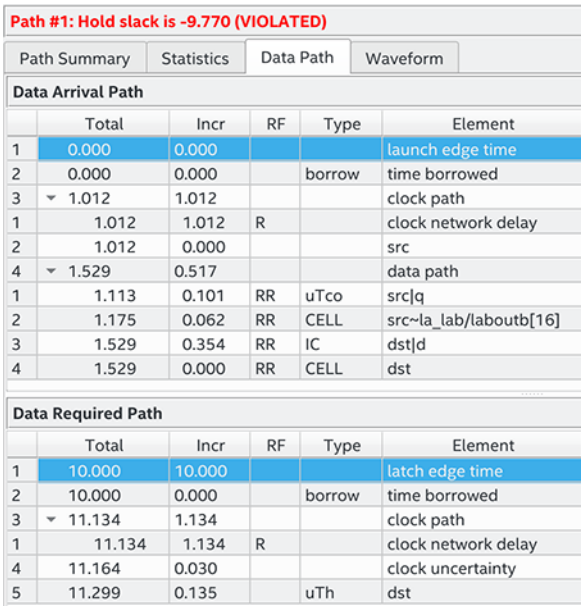


Figure 122. Hold Report with Setup Multicycle Exception



Path #1: Hold slack is -9.770 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	10.000	10.000			latch edge time
2	10.000	0.000		borrow	time borrowed
3	11.134	1.134			clock path
1	11.134	1.134	R		clock network delay
4	11.164	0.030			clock uncertainty
5	11.299	0.135		uTh	dst

Figure 123. Hold Check Calculation

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

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

2.6.8.5.3. 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 one clock period by moving the latching edge to the right of the default latching edge by 1 clock period. The hold relationship relaxes by one clock period by moving the latch edges to the left of the default latching edges by one.

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

Figure 124. Setup Timing Diagram

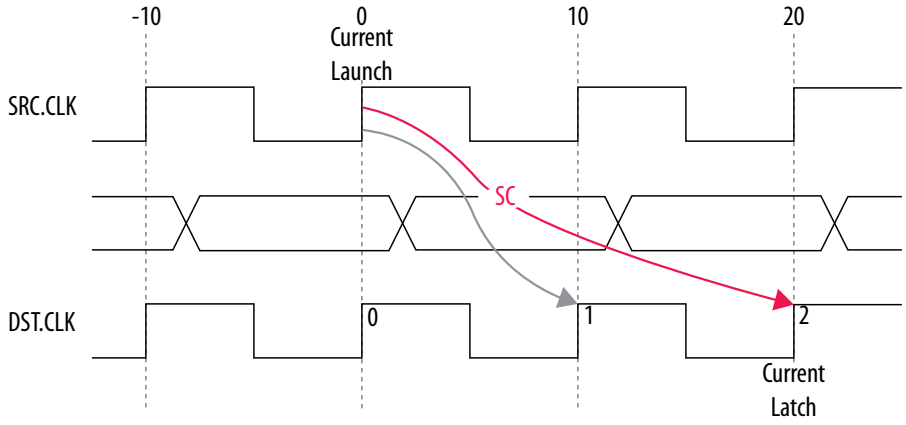


Figure 125. Setup Check Calculation

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

The most restrictive 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 126. Setup Report with Setup and Hold Multicycle Exception

Path #1: Setup slack is 18.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	20.000	20.000			latch edge time
2	20.000	0.000		borrow	time borrowed
3	21.983	1.983			clock path
1	21.983	1.983	R		clock network delay
4	21.953	-0.030			clock uncertainty
5	21.996	0.043		uTsu	dst

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

Figure 127. Hold Timing Diagram

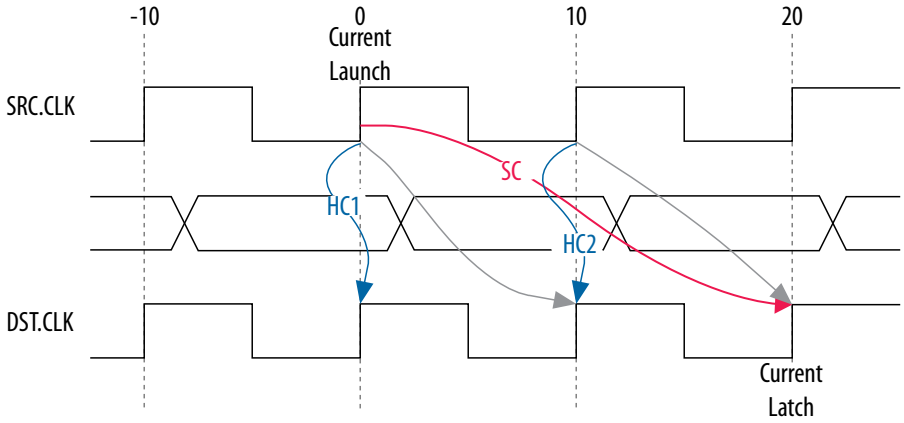


Figure 128. Hold Check Calculation

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

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

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 129. Hold Report with Setup and Hold Multicycle Exception

Path #1: Hold slack is 0.230

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

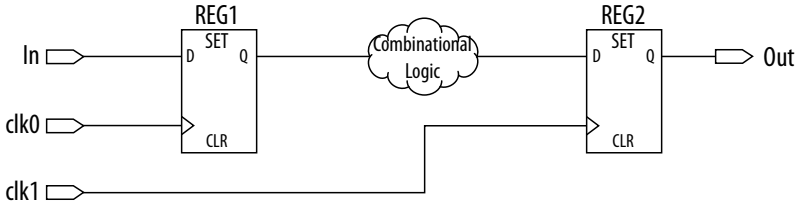
	Total	Incr	RF	Type	Element
1	0.000	0.000			latch edge time
2	0.000	0.000		borrow	time borrowed
3	1.134	1.134			clock path
1	1.134	1.134	R		clock network delay
4	1.164	0.030			clock uncertainty
5	1.299	0.135		uTh	dst

2.6.8.5.4. 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 130. Same Frequency Clocks with Destination Clock Offset Diagram



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

Figure 131. Setup Timing Diagram

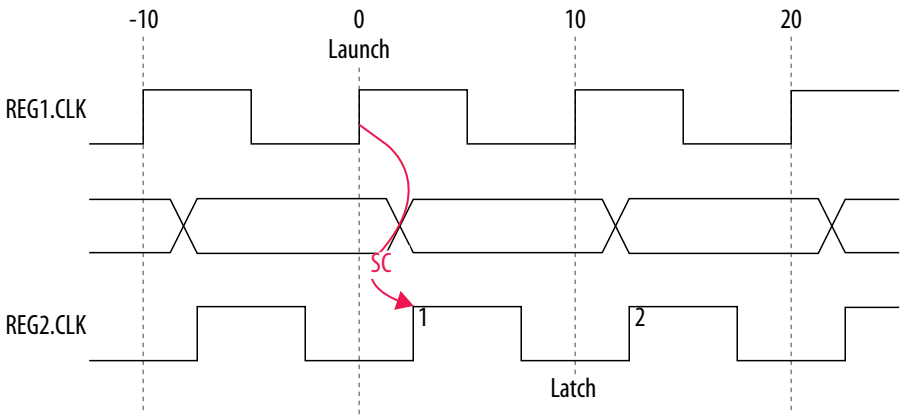


Figure 132. Setup Check Calculation

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

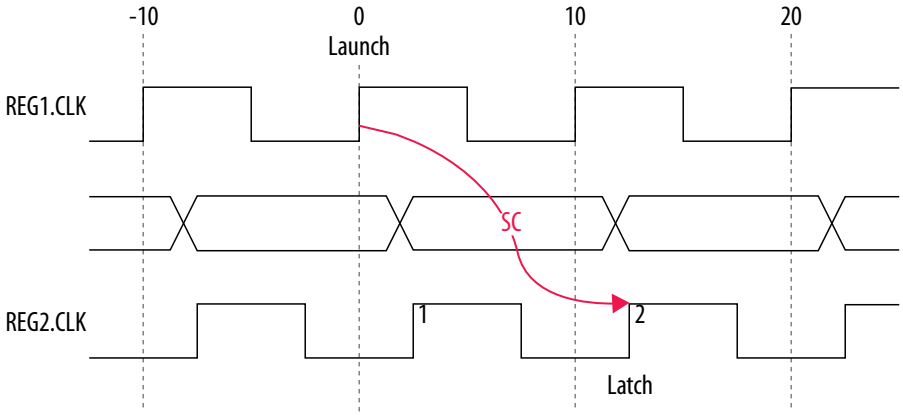
The setup relationship shown is too pessimistic and is not the setup relationship required for typical designs. To adjust the default analysis, you assign an end multicycle setup exception of two. The following shows a multicycle exception that adjusts the default analysis:

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 133. 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 134. Default Hold Check

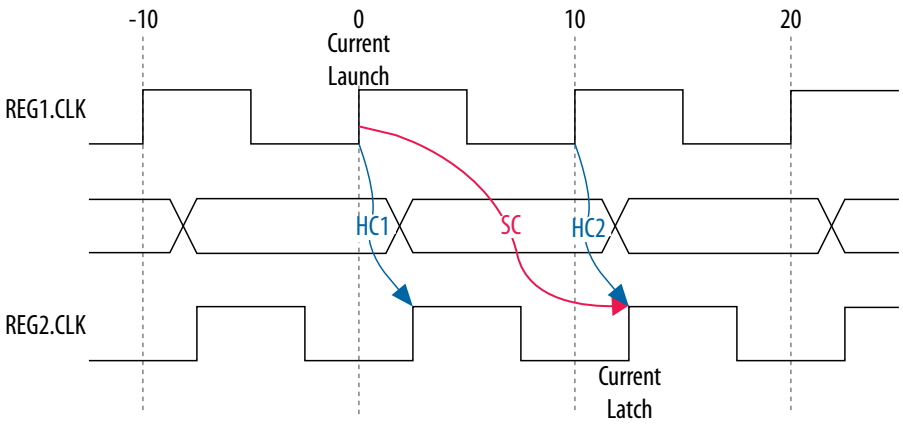


Figure 135. Hold Check Calculation

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

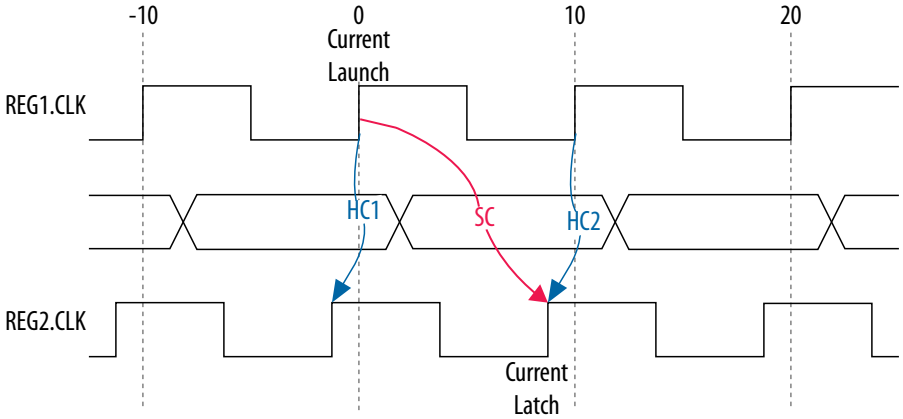
$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$
$$= 10\text{ ns} - 12\text{ ns}$$
$$= -2\text{ ns}$$

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 136. Negative Phase Shift

The figure shows an example of the setup and hold analysis for a negative phase shift of -2 ns. In this example, the default setup and hold analysis are correct, and no multicycle exceptions are required.

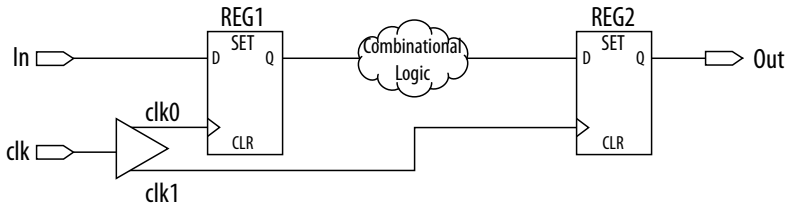


2.6.8.5.5. Destination Clock Frequency is a Multiple of the Source Clock Frequency

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 137. Destination Clock is Multiple of Source Clock



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

Figure 138. Setup Timing Diagram

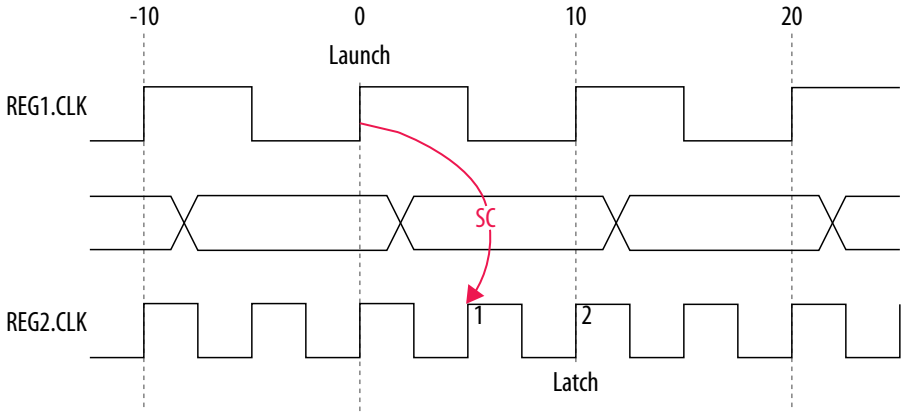


Figure 139. Setup Check Calculation

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

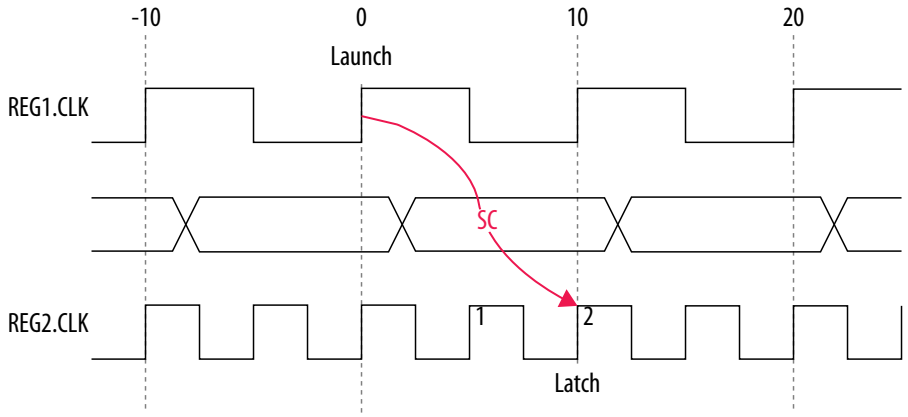
The setup relationship demonstrates that the data requires capture at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you shift the latch edge by one clock period with an end multicycle setup exception of two. The following multicycle exception assignment adjusts 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 140. 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 141. Default Hold Check

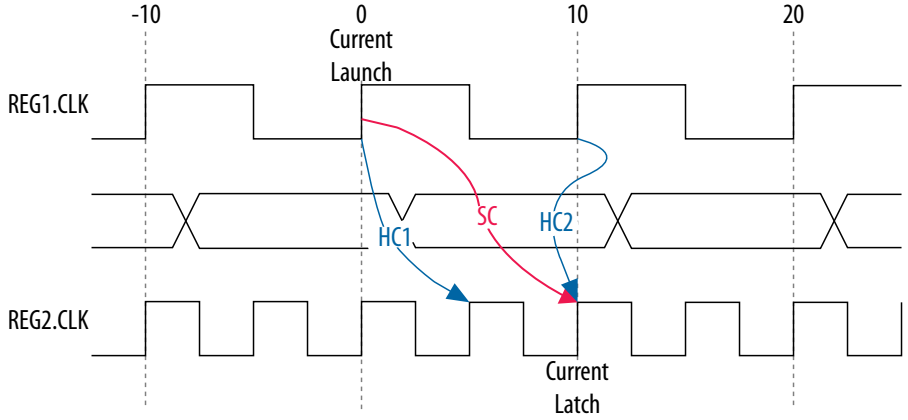


Figure 142. Hold Check Calculation

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

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

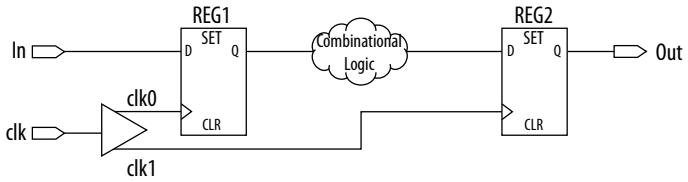
In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and must check against the data captured by the previous latch edge at 0 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

2.6.8.5.6. 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 143. Destination Clock is Multiple of Source Clock with Offset



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

Figure 144. Setup Timing Diagram

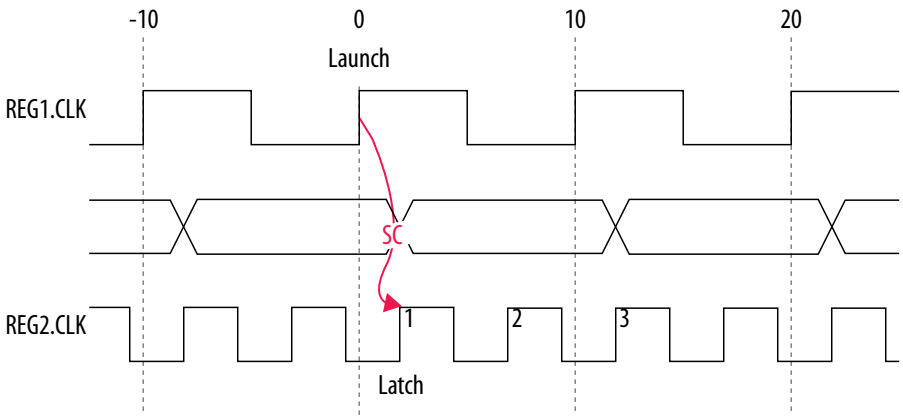


Figure 145. Setup Check Calculation

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

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

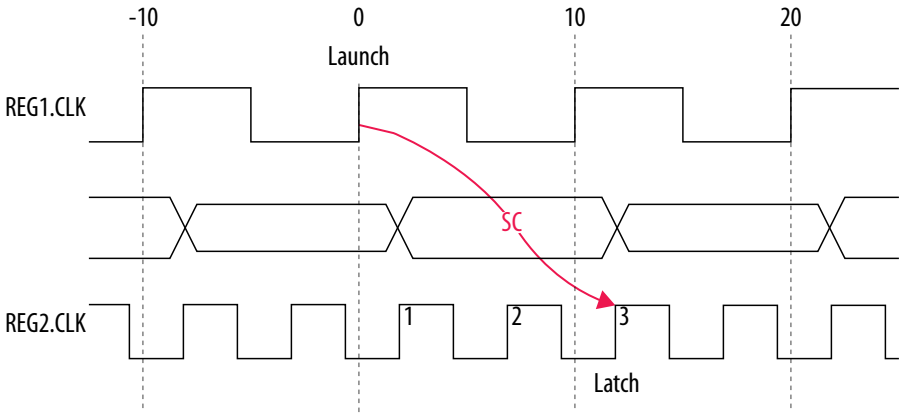
The multicycle exception adjusts the default analysis in this example:

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 146. 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 147. Default Hold Check

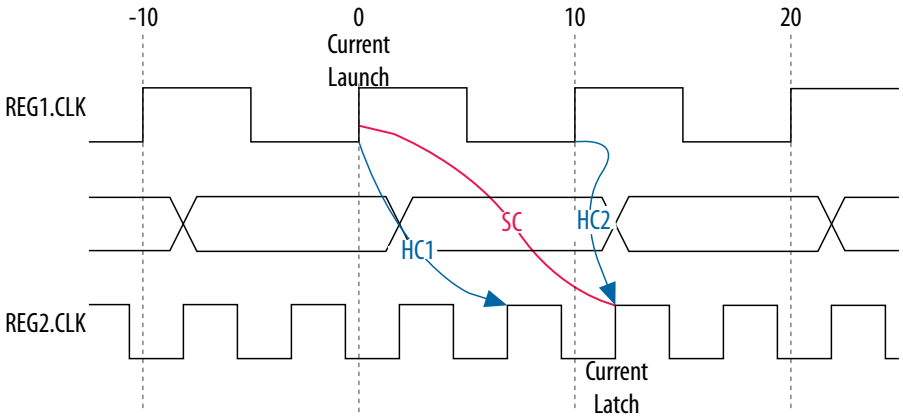


Figure 148. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 7\ \text{ns} \\ &= -7\ \text{ns} \end{aligned}$$

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

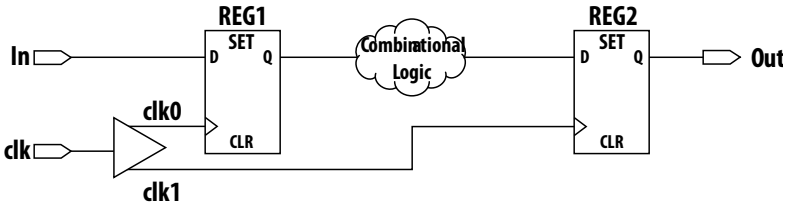
In this example, the hold check one is too restrictive. The data is launched by the edge at 0 ns, and must check against the data that the previous latch edge at 2ns captures. You can use the multicycle hold assignment of 1 to correct this.

2.6.8.5.7. 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 149. Source Clock Frequency is Multiple of Destination Clock Frequency:



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

Figure 150. Default Setup Check Analysis

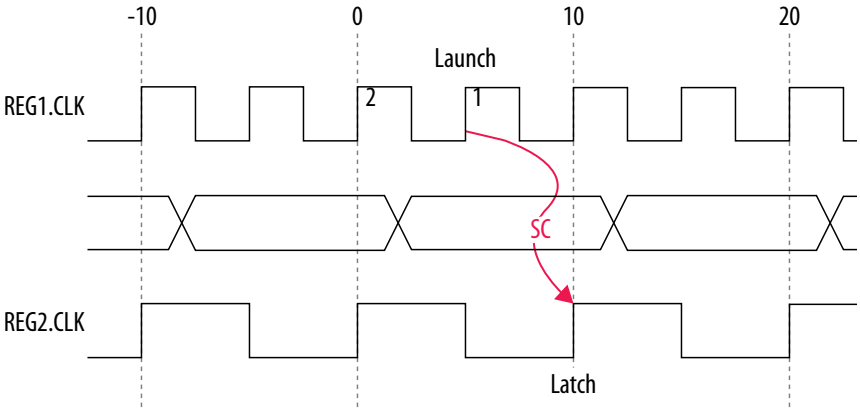


Figure 151. Setup Check Calculation

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

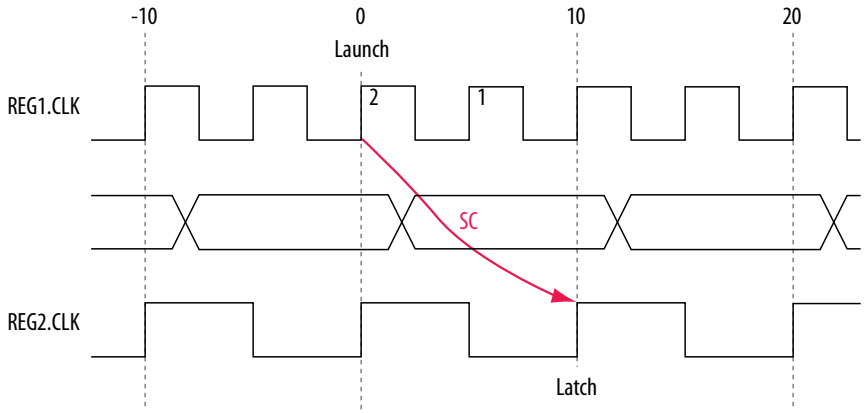
The setup relationship demonstrates that the data launched at edge one does not require capture, and the data launched at edge two requires capture; therefore, you can relax the setup requirement. To correct the default analysis, you shift the launch edge by one clock period with a start multicycle setup 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 152. 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 153. Default Hold Check

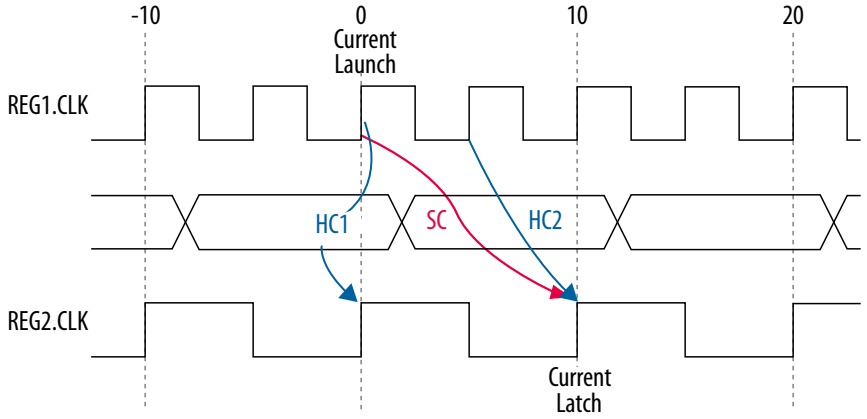


Figure 154. Hold Check Calculation

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

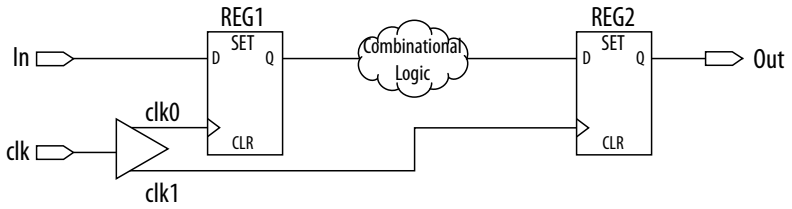
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\text{ ns} - 10\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

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.

2.6.8.5.8. 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 155. 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 156. Setup Timing Diagram

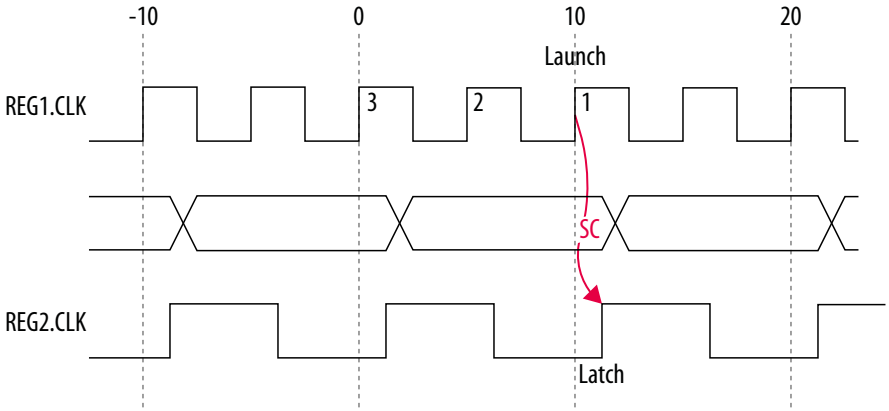


Figure 157. Setup Check Calculation

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

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.

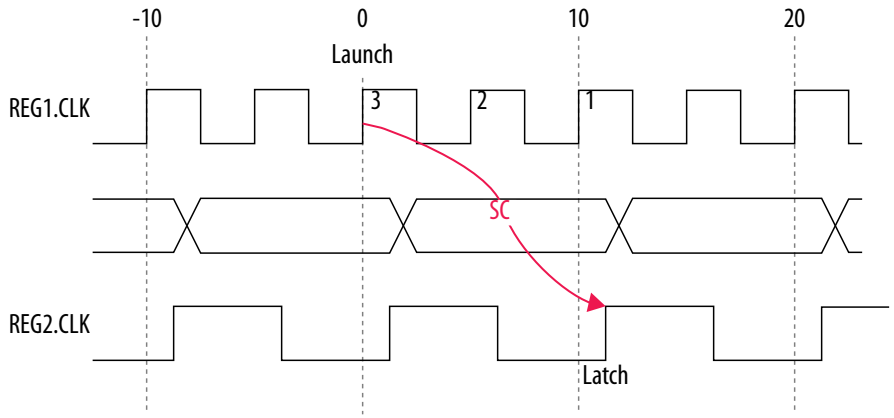
The following multicycle exception adjusts the default analysis in this example:

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 158. Preferred Setup Check Analysis



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

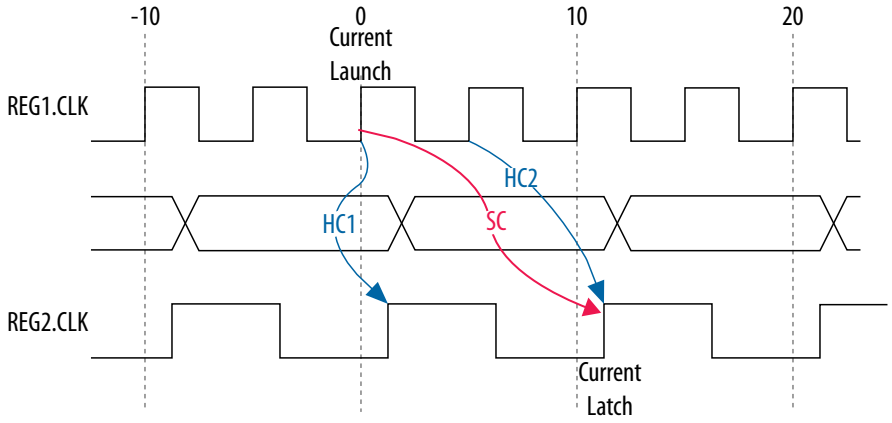
Figure 159. Hold Check Calculation

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

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

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

Figure 160. Default Hold Check Analysis



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.

2.6.8.6. 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.

Related Information

- set_timing_derate Command, Intel Quartus Prime Help
- set_annotated_delay Command, Intel Quartus Prime Help

2.6.8.7. Constraining Design Partition Ports

You can assign clock definitions and SDC exceptions to design partition ports. The block-based design and partial reconfiguration design flows require the use of design partitions.

The Compiler represents design partition ports in your timing netlist as combinational nodes with persistent names that the Compiler cannot optimize away. You can safely refer to these ports as clock sources or `-through` points in SDC constraints. You can also use design partition port names as `-to` and `-from` points in the `report_path` command.

If a port on `partition_a` has the name `clk_divide`, then the SDC constraint is:

```
create_generated_clock –source clock -divide_by 2 \
     top|partition_a|clk_divide
```

If a set of ports on `partition_b` has the name `data_input[0..7]`, then the SDC constraint is:

```
set_multicycle_path –from top|partition_a|data_reg* \
     -through top|partition_b|data_input* 2
```

You can use multiple `-through` clauses. This technique allows you to specify paths that go through output ports of one design partition, and then through the input ports of another, downstream design partition.

To add constraints to partition ports:

1. Run Analysis & Synthesis or run full compilation on a design containing design partitions.
2. To open the RTL Viewer and locate the partition ports of interest, click **Tools ➤ Netlist Viewers ➤ RTL Viewer**.
3. Using the same names as the **RTL Viewer**, add clock and other SDC constraints to the `.sdc` file for your project. You can use wildcards to refer to more than one port.
4. Recompile the design to apply the new definitions and constraints.

   Aside from block-based and PR flows, this technique also aids in emulation of ASICs using FPGAs. In this type of design, clock networks often span multiple hierarchies of partitions. Typically, designers remove the clock-dividing circuitry from the netlist, since they cannot easily emulate this circuitry on Intel FPGAs. For such clock networks, this technique allows you to define different versions of the clock signal in places where the circuitry is removed.

   You must design and place your partitions strategically, and then define the appropriate ports on these partitions. Ensure that your ports and partitions coincide with the part of the clock network which contains the special circuitry. You can manually edit the emulated ASIC netlist to annotate appropriate clock definitions and clock relationships. You can also use this technique in any projects where arbitrary locations on paths require constrained timing or defined clock sources.

Related Information

- Output Constraints (set_output_delay) on page 102
- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
- Input Constraints (set_input_delay) on page 101
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design

2.6.9. Using Fitter Overconstraints

Fitter overconstraints are timing constraints that you adjust to overcome modeling inaccuracies, mis-correlation, or other deficiencies in logic optimization. You can overconstrain setup and hold paths in the Fitter to force more aggressive timing optimization of specific paths.

Overconstraints for Intel Stratix 10 Designs

When designing for Intel Stratix 10 devices, you can target specific nodes with Fitter overconstraints to prevent the Compiler from retiming and optimizing these paths (nodes may have multiple timing requirements and the Compiler treats as "don't

touch"). If the constraint targets specific nodes, use the `is_post_route` Tcl function. This function allows you to apply overconstraints and adjust slack for modules of the Fitter (Plan, Place, Route), while allowing post-route retiming and not affecting sign-off timing analysis.

```
# Example Fitter overconstraint targeting specific nodes (allows for post-route
retiming)
if { ! [is_post_route]} {
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

Note: The `is_post_route` function is inclusive. To exclude the function, use the negation syntax (`!`).

Overconstraints for Designs that Target All Other Device Families

You can assign Fitter overconstraints that check the name of the current executable, (either `quartus_fit` or `quartus_sta`) to apply different constraints for Fitter optimization and sign-off timing analysis.

```
set fit_flow 0
if { $::TimingAnalyzerInfo(nameofexecutable) == "quartus_fit" } {
   set fit_flow 1
}
if {$fit_flow} {
  # Example Fitter overconstraint targeting specific nodes (restricts retiming)
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

2.6.10. Example Circuit and SDC File

The following `.sdc` file demonstrates constraining a dual-clock, phase-locked loop (PLL) example that illustrates. and other common synchronous design elements.

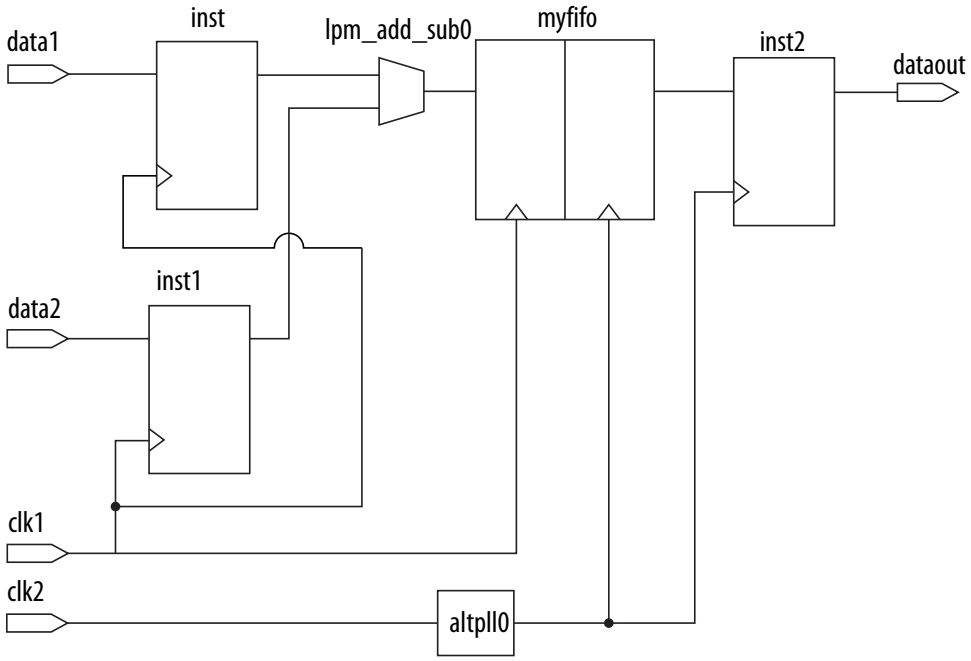
Example 6. Basic .sdc Constraints Example

```
# Create clock constraints
create_clock -name clockone -period 10.000Ns [get_ports {clk1}]
create_clock -name clocktwo -period 10.000Ns [get_ports {clk2}]
# Create virtual clocks for input and output delay constraints
create clock -name clockone_ext -period 10.000Ns
create clock -name clocktwo_ext -period 10.000Ns
# derive PLL clocks to create the altpll0| clock referenced later
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 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.

Figure 161. Dual-Clock Design Constraint Example



Related Information

Asynchronous Clock Groups (-asynchronous) on page 96

2.7. 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`

Related Information

- ::quartus::sta
  For more information about Timing Analyzer Tcl commands and a complete list of commands, refer to Intel Quartus Prime Help.
- ::quartus::sdc
  For more information about standard SDC commands and a complete list of commands, refer to Intel Quartus Prime Help.
- ::quartus::sdc_ext
  For more information about Intel FPGA extensions of SDC commands and a complete list of commands, refer to Intel Quartus Prime Help.

2.7.1. 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
```

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

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

You can optionally use command line options available to perform iterative timing analysis on large designs. You can perform a less intensive analysis with `quartus_sta --mode=implement`. In this mode, the Intel Quartus Prime software performs a reduced-corner timing analysis. When you achieve the desired result, you can use `quartus_sta --mode=finalize` to perform final Fitter optimizations and a full multi-corner timing analysis under all operating conditions.

Table 34. 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>*.

continued...

Command-Line Option	Description

`--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.
`-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.
`--mode=implement|finalize`	Regulates whether Timing Analyzer performs a reduced-corner analysis for intermediate operations (`implement`),or a four-corner analysis for final Fitter optimization and placement (`finalize`).

2.7.2. 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 35. 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.

continued...

Command	Collection Returned
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. The -of_objects option takes a node like a register and returns the clocks that drive it.
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.
get_registers	Gets the specified registers in the design.
get_keepers	Gets the specified keepers in the design. Keepers are I/O ports or registers.

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

2.7.2.1. 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`.

2.7.2.2. 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.

Related Information

- add_to_collection Command, Intel Quartus Prime Help
- remove_from_collection Command, Intel Quartus Prime Help

2.7.2.3. 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.

2.7.2.4. 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 36. Sample Search Strings and Search Results

Search String	Search Result
`get_pins *|dataa`	`lvl|dataa`
`get_pins *|datac`	*<empty>*[6]
`get_pins *|*|datac`	`lvl|cnod|datac`
`get_pins lvl*|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|*|datac`	*<empty>*[6]
`get_pins -hierarchical lvl|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|datac`	`lvl|cnod|datac`
`get_pins -hierarchical lvl|*|datac`	*<empty>*[6]
`get_pins -compatibility_mode *|datac`	`lvl|cnod|datac` [7]
`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 an 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.

(6) 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.

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

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`.

2.8. 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.

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

Document Version	Intel Quartus Prime Version	Changes
2022.09.26	22.3	• Renamed the report title "Report Reset Statistics" as "Report Register Statistics." • Revised the *Report Register Statistics* topic to describe some new features of the report. • Revised *Report Fmax Summary* topic to refer to get_clock_fmax_info command and provide more detail. • Revised *Example SDC Constraints for External Clock Mux* to replace logically_exclusive with physically_exclusive.
2022.03.28	22.1	• Described new Clock Network Viewer in *Report Clocks and Clock Networks* topic. • Updated *Report Register Spread* topic for new angle and area spread types and -to_clock filtering. • Added new *Report Timing By Source Files* topic. • Added new *Report Metastability* topic. • Added new *Report Bottlenecks* topic. • Added new *Specifying Custom Bottleneck Criteria* topic. • Revised Basic .sdc Constraints Example in *Example Circuit and SDC File* topic. • Added more detailed constraint example and diagram to *Set Clock Groups* topic. • Revised scripting examples and screenshots in *Correlating Constraints to the Timing Report* topic. • Revised scripting example in *Creating Base Clocks* topic. • Revised scripting example in *Clock Divider Example* topic. • Revised *Constraining CDC Paths* topic. • Added note about referenced SDCs within IP to *SDC File Precedence* topic.
2021.09.27	21.3	• Updated name of Report Hierarchical Retiming Restrictions command and report to Report Retiming Restrictions. • Added *Constraining CDC Paths* topic and linked to related topics. • Updated *Setting the Operating Conditions* with details about operating condition nomenclature. • Replaced Custom Reports, Device Specific, Diagnostic, and Slack report folder names throughout. • Added *Report Exceptions and Exceptions Reachability* topic describing new report. • Added *Report Clocks and Clock Networks* topic describing new report.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Added *Report Data Delay* topic describing report. • Mentioned option to view multiple before and after paths in *Report Neighbor Paths* topic. • Added `set_clock_groups` to *Timing Exception Precedence* • Updated content of Extra Info tab in *Report Timing* topic. • Corrected the *set_clock_groups -group A -group B* table in the *Creating Clock Groups* topic. • Removed *Report Custom CDC Viewer Command* topic. • Revised assignment examples in *Exclusive Clock Groups* topic.
2021.04.05	21.1	• Added: — "Report Reset Statistics" — "Report Asynchronous CDC" — Two new fields to "Report Pipelining Information". — New screenshots to "Report Logic Depth" and "Report Neighbor Paths" — `get_registers` and `get_keepers` to "Collection Commands". • Removed `-include` and `-exclude` options from "Maximum Skew"
2021.02.22	20.3	Added extra `SDC_ENTITY_FILE` info to "Using Entity-bound SDC Files"
2020.09.28	20.3	• Added "Cross Probing with Design Assistant" section. • Updated "Step 3: Run the Timing Analyzer" for multiple methods. • Updated "Step 1: Specify Timing Analyzer Settings for new tabbed dialog box and options. • Added new "Report Register Spread," "Report Route Net of Interest," "Report Hierarchical Retiming Restrictions," and "Report Pipelining Information" topics. • Updated "Report Clock Transfers" topic for new data columns. • Updated "Report Timing" topic for Extra Info tab data. • Updated "Report Fmax Summary," "Report Logic Depth," "Report Neighbor Paths," "Report CDC Viewer," and "Report Custom CDC Viewer" topics for latest GUI and consistency.
2020.04.13	20.1	• Added details and Intel Agilex device examples to "Setting the Operating Conditions" topic. • Added "Report Logic Level Depth" topic. • Added "Report Neighbor Paths" topic. • Added "Enabling Time Borrowing Optimization" topic. • Added "Report Time Borrowing Data" topic.
2019.07.15	19.2	• Updated "Setting Operating Conditions" for SmartVID timing models. • Added step for setting operating conditions to "Step 4: Run Timing Analysis." • Added details about exclusive paths to "Maximum Skew" topic. • Added GUI steps for creating entity-bound SDC files to "Using Entity-bound SDC Files" topic.
2019.04.15	19.1	• Corrected typo in "Timing Constraint Precedence" topic. • Corrected typo in "Maximum Skew" topic. • Updated "Viewing Design Assistant Recommendations" for latest GUI changes.
2018.11.07	18.1	• Improved description and diagram for "Exclusive Clock Groups" topic.
2018.09.24	18.1	• Added "Using Entity-bound SDC Files" topic. • Added "Scoping Entity-bound Constraints" topic. • Added "Entity-bound Constraint Examples" topic. • Revised "Basic Timing Analysis Flow" section to add sequential step organization, update steps, and add supporting screenshots.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Added Timing Analyzer screenshot to "Using the Timing Analyzer" topic. • Removed "Creating a Constraint File from Templates with the Text Editor" topic due to limitations of this feature in this version of the software. • 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." • Added "Viewing Design Assistant Recommendations" topic.
2018.05.07	18.0	• First release as part of the stand-alone *Timing Analyzer User Guide*
2017.11.27	17.1.0	• Removed outdated figure: Design Flow with the Timing Analyzer. • Updated Performing an Initial Analysis and Synthesis topic with Intel Quartus Prime Pro Edition commands.
2017.11.06	17.1	• Updated *Using Fitter Overconstraints* topic for Intel Stratix 10 support.
2017.05.08	17.0	• Added *Using Fitter Overconstraints* topic. • Added *Clock Domain Crossing report* topics
2016.10.31	16.1	• Implemented Intel rebranding. • Added support for `-blackbox` option with `set_input_delay`, `set_output_delay`, `remove_input_delay`, `remove_output_delay`.
2016.05.03	16.0	Added new topic: SCDS (Clock and Exception) Assignments on Blackbox Ports
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added a description of running three- and four-corner analysis with `--mode=implement|finalize`. • Added description for new `set_operating_conditions` UI.
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	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	• Minor updates. • Updated format.
November 2013	13.1	• Removed HardCopy device information.

continued...

Document Version	Intel Quartus Prime Version	Changes
June 2012	12.0	• 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	• Consolidated content from the Best Practices for the Intel Quartus Prime Timing Analyzer chapter. • Changed to new document template.
May 2011	11.0	• Updated to improve flow. Minor editorial updates.
December 2010	10.1	• Changed to new document template. • Revised and reorganized entire chapter. • Linked to Intel Quartus Prime Help.
July 2010	10.0	Updated to link to content on SDC commands and the Timing Analyzer GUI in Intel Quartus Prime Help.
November 2009	9.1	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	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

2.10. Intel Quartus Prime Pro Edition User Guide: Timing Analyzer Archive

For the latest and previous versions of this user guide, refer to Intel Quartus Prime Pro Edition User Guide: Timing Analyzer. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Intel Quartus Prime Pro Edition User Guides

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

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro 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, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro 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 Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro 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 Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, 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 Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro 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, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro 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 Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro 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 Pro 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 Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.