




The Quartus® II TimeQuest 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 TimeQuest analyzer GUI or command-line interface to constrain, analyze, and report results for all timing paths in your design.

This chapter contains the following sections:

- “Getting Started with the TimeQuest Analyzer”
- “Constraining and Analyzing with Tcl Commands” on page 7–7
- “Creating Clocks and Clock Constraints” on page 7–14
- “Creating I/O Requirements” on page 7–24
- “Creating Delay and Skew Constraints” on page 7–26
- “Creating Timing Exceptions” on page 7–27
- “Examples of Basic Multicycle Exceptions” on page 7–35
- “Application of Multicycle Exceptions” on page 7–44
- “Timing Reports” on page 7–57

-  For more information about basic timing analysis concepts and how they pertain to the TimeQuest analyzer, refer to the *Timing Analysis Overview* chapter in volume 3 of the *Quartus II Handbook*.
-  For more information about Altera resources available for the TimeQuest analyzer, refer to the *TimeQuest Timing Analyzer Resource Center* of the Altera website.
-  For more information about the TimeQuest analyzer, refer to the *Altera Training* page of the Altera website.

## Getting Started with the TimeQuest Analyzer

This section provides an overview of the design steps for setting up your project for timing and analysis and to constrain your design with the TimeQuest analyzer.

### Running the TimeQuest Analyzer

To run the TimeQuest analyzer directly from the Quartus II software GUI, click **TimeQuest Timing Analyzer** on the Tools menu.

- ❓ For more information about the TimeQuest analyzer GUI, refer to *About TimeQuest Timing Analysis* in Quartus II Help.

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

```
quartus_staw ←
```

To run the TimeQuest analyzer in command-line mode for easy integration with scripted design flows, type the following command at a system command prompt:

```
quartus_sta -s←
```

Table 7-1 describes the available command-line options.

**Table 7-1. Summary of Command-Line Options (Part 1 of 2)**

Command-Line Option	Description
-h   --help	Provides help information on <code>quartus_sta</code> .
-t <script file>   --script=<script file>	Sources the <script file>.
-s   --shell	Enters shell mode.
--tcl_eval <tcl command>	Evaluates the Tcl command <tcl command>.
--do_report_timing	For all clocks in the design, run the following commands: <pre>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</pre>
--force_dat	Forces an update of the project database with new delay information.
--lower_priority	Lowers the computing priority of the <code>quartus_sta</code> process.
--post_map	Uses the post-map database results.
--sdc=<SDC file>	Specifies the <b>.sdc</b> to use.
--report_script=<script>	Specifies a custom report script to call.
--speed=<value>	Specifies the device speed grade used for timing analysis.
--tq2pt	Generates temporary files to convert the TimeQuest analyzer <b>.sdc</b> file(s) to a PrimeTime <b>.sdc</b> .
-f <argument file>	Specifies a file containing additional command-line arguments.
-c <revision name>   --rev=<revision_name>	Specifies which revision and its associated <b>.qsf</b> to use.

**Table 7-1. Summary of Command-Line Options (Part 2 of 2)**

Command-Line Option	Description
--multicorner	Specifies that all slack summary reports be generated for both slow- and fast-corners.
--multicorner[=on off]	Turns off multicorner timing analysis.
--voltage=<value_in_mV>	Specifies the device voltage, in mV used for timing analysis.
--temperature= <value_in_C>	Specifies the device temperature in degrees Celsius, used for timing analysis.
--parallel [=<num_processors>]	Specifies the number of computer processors to use on a multiprocessor system.
--64bit	Enables 64-bit version of the executable.

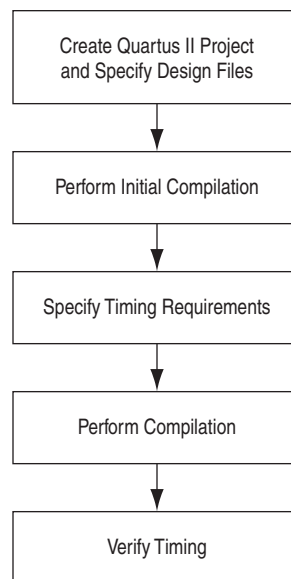
For more information about steps to perform before opening the TimeQuest analyzer, refer to “Recommended Flow” on page 7-3.

For more information about using Tcl commands to constrain and analyze your design, refer to “Constraining and Analyzing with Tcl Commands” on page 7-7.

## Recommended Flow

The Quartus II TimeQuest analyzer performs constraint validation to timing verification as part of the compilation flow. Figure 7-1 shows the recommended design flow to maximize the benefits of the TimeQuest Analyzer.

**Figure 7-1. Design Flow with the TimeQuest Timing Analyzer**





## Creating and Setting Up your Design

You must first create your project in the Quartus II software. Include all the necessary design files, including any existing Synopsys Design Constraints (.sdc) files that contain timing constraints for your design.

❓ For more information, refer to *Managing Files in a Project* in Quartus II Help.



## Performing an Initial Compilation

If you have never compiled your design, or you don't have an `.sdc` file, and you want to use the TimeQuest analyzer to create one interactively, you must compile your design to create an initial design database before you specify timing constraints. You can either perform Analysis and Synthesis to create a post-map database, or perform a full compilation to create a post-fit database. Creating a post-map database is faster than a post-fit database, and is sufficient for creating initial timing constraints. The type of database you create determines the type of timing netlist generated by the TimeQuest analyzer; a post-map netlist if you perform Analysis and Synthesis or a post-fit netlist if you perform a full compilation.


-  If you are using incremental compilation, you must merge your design partitions after performing Analysis and Synthesis to create a post-map database.
-  For more information, refer to *Setting up and Running Analysis and Synthesis* and *Setting up and Running a Compilation* in Quartus II Help.

## Specifying Timing Requirements

Before running timing analysis with the TimeQuest analyzer, you must specify initial timing constraints that describe the clock characteristics, timing exceptions, and signal transition arrival and required times. You can use the TimeQuest Timing Analyzer Wizard to enter initial constraints for your design, and then refine timing constraints with the TimeQuest analyzer GUI or with a Tcl script.

-  The Quartus II software assigns a default frequency of 1 GHz for clocks that have not been constrained, either in the TimeQuest GUI or an `.sdc` file, unless any constraint exists in the design. In that case, all unconstrained clocks remain unconstrained.
-  For more information, refer to *Specifying Timing Constraints and Exceptions* in Quartus II Help.

The `.sdc` must contain only SDC commands. Tcl commands to manipulate the timing netlist or control the compilation flow should be run as part of a separate Tcl script. After you create timing constraints, update the timing netlist to apply the new constraints. The TimeQuest analyzer applies all constraints to the netlist for verification and removes any invalid or false paths in the design from verification.

-  The constraints in the `.sdc` are read in sequence. You must first make a constraint before making any references to that constraint. For example, if a generated clock references a base clock, the base clock constraint must be made before the generated clock constraint.

The Quartus II Text Editor provides templates for SDC constraints. For more information, refer to “Using the Quartus II Templates” on page 7-6.

## Performing a Full Compilation

After creating initial timing constraints, you must fully compile your design. When compilation is complete, you can open the TimeQuest analyzer to verify timing results and to generate summary, clock setup and clock hold, recovery, and removal reports for all defined clocks in the design.

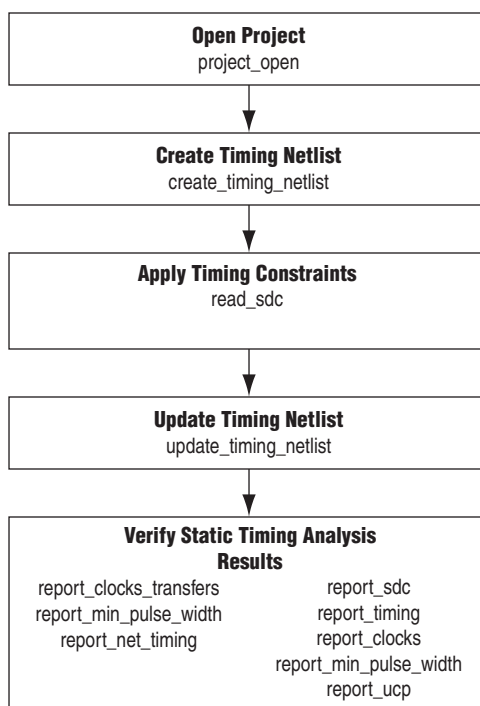
## Verifying Timing

The TimeQuest 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. If you encounter violations along timing paths, use the timing reports to analyze your design and determine how best to optimize your design. If you modify, remove, or add constraints, you should perform a full compilation again. This iterative process helps resolve timing violations in your design.

- For more information, refer to *Viewing Timing Analysis Results* in Quartus II Help.

Figure 7-2 shows the recommended flow for constraining and analyzing your design within the TimeQuest analyzer. Included are the corresponding Tcl commands for each step.

**Figure 7-2. The TimeQuest Timing Analyzer Flow**



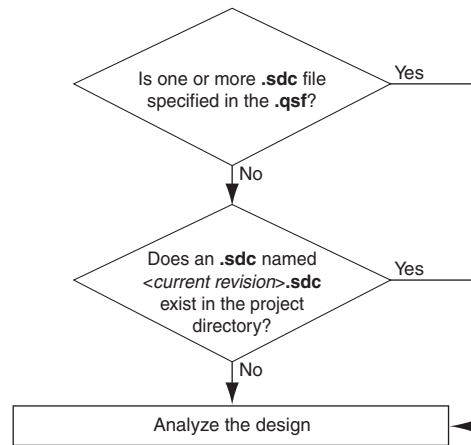
## SDC File Precedence

The Fitter and the TimeQuest analyzer process **.sdc** files in the order you specify in the Quartus II Settings File (**.qsf**). You can specify the files to process and the order they are processed from the **Assignments** menu. Click **Settings**, then **TimeQuest Timing Analyzer**, and specify a processing order in the **SDC files to include in the project** box.

If no **.sdc** files are listed in the **.qsf**, the Quartus II software looks for an **.sdc** named *<current revision>.sdc* in the project directory. An **.sdc** can also be added from a Quartus II IP File (**.qip**) included in the **.qsf**.

Figure 7-3 shows the order in which the Quartus II software searches for an `.sdc`.

**Figure 7-3. .sdc File Order of Precedence**



If you type the `read_sdc` command at the command line without any arguments, the TimeQuest analyzer reads constraints embedded in HDL files, then follows the `.sdc` file precedence order shown in Figure 7-3.

## Using the Quartus II Templates

You can create an `.sdc` from constraint templates in the Quartus II software with the Quartus II Text Editor, or with your preferred text editor.

### Creating a Constraint File with the Quartus II Text Editor

To insert constraints with the Quartus II Text Editor, follow these steps:

1. On the **File** menu, click **New**.
2. In the **New** dialog box, select the **Synopsys Design Constraints File** type from the **Other Files** group. Click **OK**.
3. Click the **Insert Template** button on the text editor menu, or, right-click in the blank `.sdc` file in the Quartus II Text Editor, then click **Insert Template**.
4. In the **Insert Template** dialog box, expand the **TimeQuest** section, then expand the **SDC Commands** section.
5. Expand a command category, for example, **Clocks**.
6. Select a command. The SDC constraint appears in the **Preview** pane.
7. Click **Insert** to paste the SDC constraint into the blank `.sdc` you created in step 2.
8. Repeat as needed with other constraints, or click **Close** to close the **Insert Template** dialog box.

You can now use any of the standard features of the Quartus II Text Editor to modify the `.sdc` or save the `.sdc` to edit in a text editor.

- ❓ For more information on inserting a template with the Quartus II Text Editor, refer to *About the Quartus II Text Editor* in Quartus II Help.

## Constraining and Analyzing with Tcl Commands

You can use Tcl commands from the Quartus II software Tcl Application Programming Interface (API) to constrain, analyze, and collect information for your design. This section focuses on executing timing analysis tasks with Tcl commands; however, you can perform many of the same functions in the TimeQuest analyzer GUI. SDC commands are Tcl commands for constraining a design. SDC extension commands provide additional constraint methods and are specific to the TimeQuest analyzer. Additional TimeQuest analyzer commands are available for controlling timing analysis and reporting. These commands are contained in the following Tcl packages available in the Quartus II software:

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

- ❓ For more information about TimeQuest analyzer Tcl commands and a complete list of commands, refer to `::quartus::sta` in Quartus II Help. For more information about standard SDC commands and a complete list of commands, refer to `::quartus::sdc` in Quartus II Help. For more information about Altera extensions of SDC commands and a complete list of commands, refer to `::quartus::sdc_ext` in Quartus II Help.

## Collection Commands

The TimeQuest analyzer Tcl commands often return port, pin, cell, or node names in a data set called a collection. In your Tcl scripts you can iterate over the values in collections to analyze or modify constraints in your design.

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

Table 7-2 describes the collection commands supported by the TimeQuest analyzer.

**Table 7-2. SDC Collection Commands (Part 1 of 2)**

Command	Description of the collection returned
<code>all_clocks</code>	All clocks in the design.
<code>all_inputs</code>	All input ports in the design.
<code>all_outputs</code>	All output ports in the design.
<code>all_registers</code>	All registers in the design.
<code>get_cells</code>	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.
<code>get_clocks</code>	Clocks in the design. When used as an argument to another command, such as the <code>-from</code> or <code>-to</code> of <code>set_multicycle_path</code> , each node in the clock represents all nodes clocked by the clocks in the collection. The default uses the specific node (even if it is a clock) as the target of a command.
<code>get_nets</code>	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.

**Table 7-2. SDC Collection Commands (Part 2 of 2)**

Command	Description of the collection returned
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	Ports (design inputs and outputs) in the design.

### 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 make an assignment to node `reg*`, the TimeQuest analyzer searches for and applies the assignment 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 make an assignment to a node specified as `reg?`, the TimeQuest analyzer searches and applies the assignment to all design nodes that match the prefix `reg` and any single character following; for example, `reg1`, `rega`, and `reg4`.

### Adding and Removing Collection Items

Wildcards used with collection commands define collection items identified by the command. For example, if a design contains registers named `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 could 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. [Example 7-1](#) shows the `add_to_collection` command and arguments.

#### Example 7-1. add\_to\_collection Command

---

```
add_to_collection <first collection> <second collection>
```

---



The `add_to_collection` command creates a new collection that is the union of the two specified collections.

The `remove_from_collection` command allows you to remove items from an existing collection. [Example 7-2](#) shows the `remove_from_collection` command and arguments.

#### Example 7-2. remove\_from\_collection Command

---

```
remove_from_collection <first collection> <second collection>
```

---



Example 7-3 shows examples of how to add elements to collections.

### Example 7-3. 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 Quartus II software, keepers are I/O ports or registers. An `.sdc` that includes `get_keepers` can only be processed as part of the TimeQuest analyzer flow and is not compatible with third-party timing analysis flows.



For more information about the `add_to_collection` and `remove_from_collection` commands—including full syntax information, options, and example usage—refer to *add\_to\_collection* and *remove\_from\_collection* in Quartus II Help.

### Using the `query_collection` Command

You can display the contents of a collection with the `query_collection` command.

Example 7-4 shows how to report the contents of a collection:

### Example 7-4. `query_collection` Command

---

```
query_collection -report -all $regs_union
```

---

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

### Using the `get_pins` Command

The collection commands `get_pins` allow you to refine searches that include wildcard characters.

Table 7-3 shows examples of search strings that use options to refine the search and wildcards. The examples in Table 7-3 filter the following node and pin names to illustrate function:

- foo
- foo|dataa
- foo|datab
- foo|bar
- foo|bar|datac

■ foo|bar|datac

**Table 7-3. Sample Search Strings and Search Results**

Search String	Search Result
get_pins * dataa	foo dataa
get_pins * datac	<empty> <sup>(1)</sup>
get_pins * * datac	foo bar datac
get_pins foo* *	foo dataa, foo datab
get_pins -hierarchical * * datac	<empty> <sup>(1)</sup>
get_pins -hierarchical foo *	foo dataa, foo datab
get_pins -hierarchical * datac	foo bar datac
get_pins -hierarchical foo* datac	<empty> <sup>(1)</sup>
get_pins -compatibility_mode * datac	foo bar datac
get_pins -compatibility_mode * * datac	foo bar datac

**Note to Table 7-3:**

(1) The search result is <empty> because more than one pipe character (|) is not supported.

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 and/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.

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`.

## Identifying the Quartus II Software Executable from the SDC File

To identify which Quartus II software executable is currently running you can use the `$.:TimeQuestInfo(nameofexecutable)` variable from within an `.sdc`. [Example 7-5](#) shows how to specify different SDC constraints for a specific Quartus II software executable.

### Example 7-5. Identifying the Quartus II Executable

---

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

---

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

## Locating Timing Paths in Other Tools

You can locate paths and elements from the TimeQuest analyzer to other tools in the Quartus II software. Use the **Locate Path** command in the TimeQuest analyzer GUI or the `locate` command.

- ② For more information about locating paths from the TimeQuest analyzer, refer to [Viewing Timing Analysis Results](#) and [locate](#) in Quartus II Help.

[Example 7-6](#) shows how to locate ten paths from TimeQuest analyzer to the Chip Planner and locate all data ports in the Technology Map Viewer.

### Example 7-6. Locating from the TimeQuest Analyzer

---

```
# Locate in the Chip Planner all of the nodes in ten paths with the
longest delay

locate [get_path -npaths 10] -chip

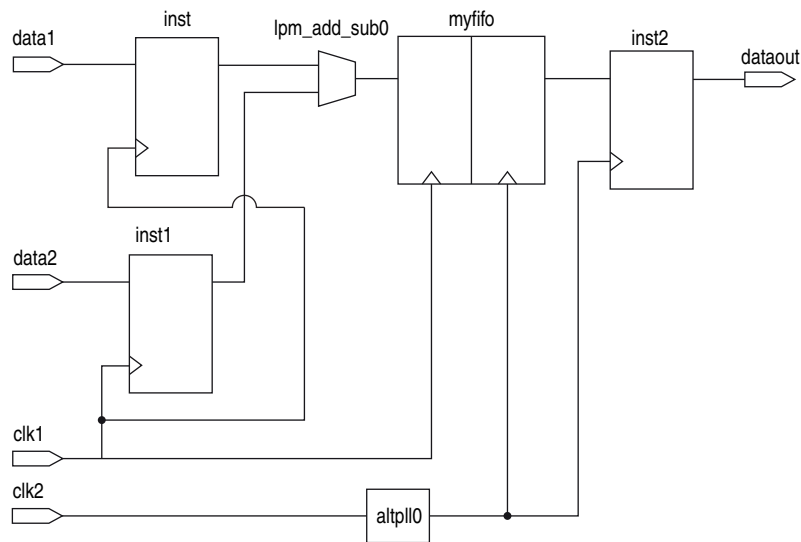
# locate all ports that begin with data to the Technology Map Viewer
locate [get_ports data*] -tmv
```

---

## Design Constraints: An Example

Figure 7-4 shows an example circuit including two clocks, a phase-locked loop (PLL), and other common synchronous design elements.

**Figure 7-4. TimeQuest Constraint Example**



Example 7-7 shows an .sdc file containing basic constraints for the circuit in Figure 7-4.

### Example 7-7. Example Basic SDC Constraints

```
# Create clock constraints
create_clock -name clockone -period 10.000 [get_ports {clk1}]
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]

# Create virtual clocks for input and output delay constraints
create_clock -name clockone_ext -period 10.000
create_clock -name clocktwo_ext -period 10.000

derive_pll_clocks

# derive clock uncertainty
derive_clock_uncertainty

# Specify that clockone and clocktwo are unrelated by assigning
# them to separate asynchronous groups
set_clock_groups \
  -asynchronous \
  -group {clockone} \
  -group {clocktwo} \
  altp110|altp11_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 in Example 7-7 contains the following basic constraints you should include for most designs:

- Definitions of clockone and clocktwo as base clocks, and assignment of those settings to nodes in the design.
- Definitions of clockone\_ext and clocktwo\_ext as virtual clocks, which represent clocks driving external devices interfacing with the FPGA.
- Automated derivation of generated clocks on PLL outputs.
- Derivation of clock uncertainty.
- Specification of two clock groups, the first containing clockone and its related clocks, the second containing clocktwo and its related clocks, and the third group containing the output of the PLL. This specification overrides the default analysis of all clocks in the design as related to each other. For more information about asynchronous clock groups, refer to “Asynchronous Clock Groups” on page 7-22.
- Specification of input and output delays for the design.

The following sections describe each of these constraint types in detail.

## Creating Clocks and Clock Constraints

To ensure accurate static timing analysis results, you must specify all clocks and any associated clock characteristics in your design. The TimeQuest analyzer supports SDC commands that accommodate various clocking schemes and clock characteristics.

The TimeQuest analyzer supports the following types of clocks:

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

Clocks are used to specify requirements for synchronous transfers and guide the Fitter optimization algorithms to achieve the best possible placement for your design.

Specify clock constraints first in the `.sdc` because other constraints may reference previously defined clocks. The TimeQuest analyzer reads SDC constraints and exceptions from top to bottom in the file.

### Creating Base Clocks

Base clocks are the primary input clocks to the device. Unlike clocks from PLLs that are generated in the device, base clocks are generated by off-chip oscillators or forwarded from an external device. Define base clocks first because generated clocks and other constraints often reference base clocks.

To create clock settings for the signal from any register, port, or pin, use the `create_clock` command. You can create each clock with unique characteristics.

[Example 7-8](#) shows how to create a 10 ns clock with a 50% duty cycle that is phase shifted by 90 degrees applied to port `clk_sys`.

#### Example 7-8. 100 MHz Shifted by 90 Degrees Clock Creation

---

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

---

Use the `create_clock` command to constrain all primary input clocks. The target for the `create_clock` command is usually a pin. To specify the pin as the target, use the `get_ports` command. [Example 7-9](#) shows how to specify a 100 MHz requirement on a `clk_sys` input clock port.

#### Example 7-9. create\_clock Command

---

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

---

You can apply multiple clocks on the same clock node with the `-add` option.

[Example 7-10](#) shows how to specify that two oscillators drive the same clock port on the device.

#### Example 7-10. Two Oscillators Driving the Same Clock Port

---

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

---

- For more information about the `create_clock` and `get_ports` commands—including full syntax information, options, and example usage—refer to *create\_clock* and *get\_ports* in Quartus II Help.

## Creating Virtual Clocks

A virtual clock is a clock that does not have a real source in the design or that does not interact directly with the design. Virtual clocks are used in most I/O constraints; they represent the clock at the external device connected to the FPGA.

To create virtual clocks, use the `create_clock` command with no value specified for the `<targets>` option. Use virtual clocks for the reference clocks of `set_input_delay` and `set_output_delay` constraints.

Example 7-11 shows how to create a 10 ns virtual clock. The example is a virtual clock because no target is specified.

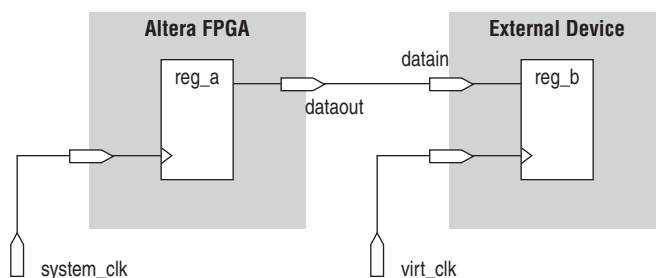
### Example 7-11. Create Virtual Clock

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

- For more information about the `set_input_delay`, and `set_output_delay` commands—including full syntax information, options, and example usage—refer to *set\_input\_delay*, and *set\_output\_delay* in Quartus II Help.

Figure 7-5 shows a design where a virtual clock is required for the TimeQuest analyzer to properly analyze the relationship between the external register and the registers in the design. Because the oscillator, `virt_clk`, does not interact with the Altera device, but acts as the clock source for the external register, you must declare the clock as a virtual clock. After you create the virtual clock, you can perform a register-to-register analysis between the register in the Altera device and the register in the external device.

Figure 7-5. Virtual Clock Board Topology



Example 7-12 shows how to create a 10 ns virtual clock named `virt_clk` with a 50% duty cycle where the first rising edge occurs at 0 ns. The virtual clock is then used as the clock source for an output delay constraint.

#### Example 7-12. Virtual Clock Example

```
#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]
```

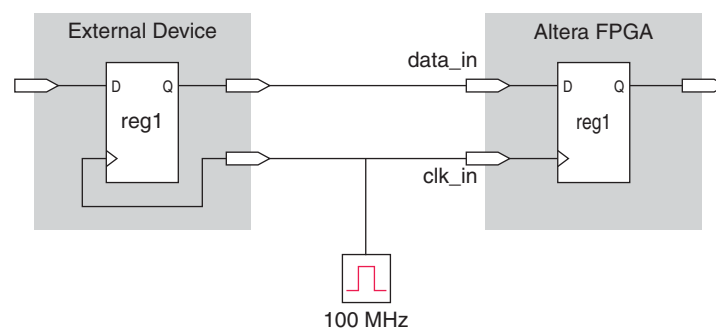
### I/O Interface Uncertainty

Virtual clocks are recommended for I/O constraints because the `derive_clock_uncertainty` command can add different uncertainty values on clocks that interface with an external I/O port than uncertainty values between register paths fed by a clock inside the FPGA.

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. Figure 7-6 shows a typical input I/O interface with clock specifications.

Figure 7-6. I/O Interface Clock Specifications





Example 7-13 shows the SDC commands to constrain the I/O interface shown in Figure 7-6.

#### Example 7-13. 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]
```



For more information about clock uncertainty and clock transfers, refer to “Clock Uncertainty” on page 7-23

## Creating Multifrequency Clocks

You must create a multifrequency clock if your design has more than one clock source feeding a single clock node in your design. The additional clock may act as a low-power clock, with a lower frequency than the primary clock. If your design uses multifrequency clocks, use the `set_clock_groups` command to define clocks that are exclusive. For more information about creating exclusive clock groups, refer to “Creating Clock Groups” on page 7-21.

To create multifrequency clocks, use the `create_clock` command with the `-add` option to create multiple clocks on a clock node. Example 7-14 shows how to create a 10 ns clock applied to clock port `clk`, and then add an additional 15 ns clock to the same clock port. The TimeQuest analyzer uses both clocks when it performs timing analysis.

#### Example 7-14. Multifrequency Clock Example

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

## Creating Generated Clocks

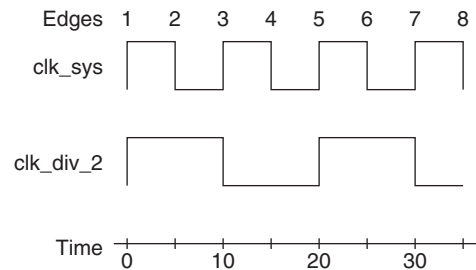
Generated clocks are applied in the design when you modify the properties of a source synchronous clock signal, including phase, frequency, offset, and duty cycle. In the `.sdc`, generated clocks, which can be the outputs of PLLs or register clock dividers, are constrained after all base clocks. Generated clocks capture all clock delays and clock latency where the generated clock target is defined, ensuring that all base clock properties are accounted for in the generated clock.

Use the `create_generated_clock` command to constrain generated clocks in your design. The source of the `create_generated_clock` command should be a node in your design and not a previously constrained clock.

A common form of generated clock is a clock divider. [Example 7-15](#) creates a base clock, `clk_sys`, then defines a generated clock `clk_div_2`, which is the clock frequency of `clk_sys` divided by two.

#### Example 7-15. Clock Divider

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



When you use the `create_generated_clock` command, the `-source` option specifies a node with a clock used as a reference for your generated clock. Best practice is to specify the input clock pin of the target node for your new generated clock. You can also specify the target node of the reference clock. In [Example 7-15](#), the `-source` option specifies the clock port `clk` feeding the clock pin of register `reg`.

If you have multiple base clocks feeding a node that is the source for a generated clock, you must define multiple generated clocks. Each generated clock is associated to one base clock using the `-master_clock` option in each generated clock statement.

The TimeQuest analyzer provides 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 defined for the PLL. For more information about deriving PLL clock outputs, refer to “[Deriving PLL Clocks](#)” on page 7-19.

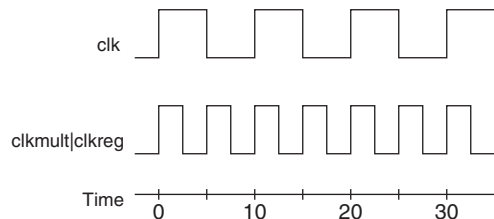
- ① For more information about the `create_generated_clock` and `derive_pll_clocks` commands—including for full syntax information, options, and example usage—refer to [create\\_generate\\_clock](#) and [derive\\_pll\\_clocks](#) in Quartus II Help.

The inverse of a clock divider is a clock multiplier. Figure 7-7 shows the effect of applying a multiplication factor to the generated clock.

**Figure 7-7. Multiplying a Generated Clock**

```
create_clock -period 10 -waveform { 0 5 } [get_ports clk]

# Creates a multiply-by-two clock
create_generated_clock -source [get_ports clk] -multiply_by 2 [get_registers \
clkmult|clkreg]
```



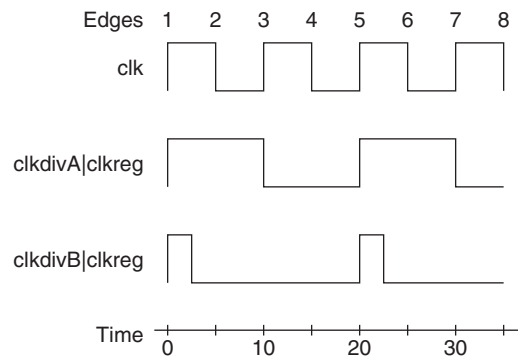
An uncommon but useful type of generated clock is one with shifted edges. Figure 7-8 shows how to modify the generated clock by defining and shifting the edges.

**Figure 7-8. Edge Shifting a Generated Clock**

```
create_clock -period 10 -waveform { 0 5 } [get_ports clk]

# Creates a divide-by-two clock
create_generated_clock -source [get_ports clk] -edges { 1 3 5 } [get_registers \
clkdivA|clkreg]

# Creates a divide-by-two clock independent of the master clock's duty cycle (now 50%)
create_generated_clock -source [get_ports clk] -edges { 1 1 5 } \
-edge_shift { 0 2.5 0 } [get_registers clkdivB|clkreg]
```



❓ For information about creating generated clocks, refer to *create\_generated\_clocks* and *Specifying Timing Constraints and Exceptions* in Quartus II Help.

## Deriving PLL Clocks

Use the `derive_pll_clocks` command to direct the TimeQuest analyzer to automatically search the timing netlist for all unconstrained PLL output clocks. The `derive_pll_clocks` command automatically creates generated clocks on the outputs of every PLL by calling the `create_generated_clock` command. The source for the `create_generated_clock` command is the input clock pin of the PLL.

Example 7-16 shows the command to create a base clock for the PLL input clock port and call `derive_pll_clocks` to create PLL output clocks.

#### Example 7-16. Create Base Clock for PLL Input Clock Ports

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



If your design contains LVDS transmitters, LVDS receivers, or transceivers, Altera recommends using the `derive_pll_clocks` command. The command automatically constrains this logic in your design by adding the appropriate multicycle constraints to account for any deserialization factors.

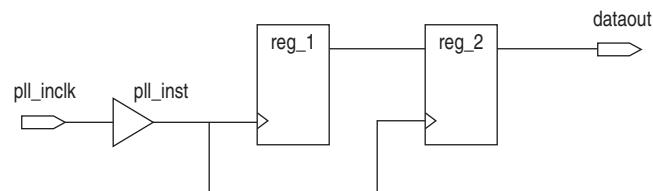


For more information about the `derive_pll_clocks` command—including full syntax information, options, and example usage—refer to *derive\_pll\_clocks* and *Derive PLL Clocks* in Quartus II Help.

You can include the `derive_pll_clocks` command in your `.sdc`, which automatically detects any changes to the PLL settings. Each time the TimeQuest analyzer reads your `.sdc`, the appropriate `create_generated_clocks` command is generated for the PLL output clock pin.

Figure 7-9 shows a simple PLL design with a register-to-register path.

**Figure 7-9. Simple PLL Design**



Example 7-17 shows the messages generated by the TimeQuest analyzer when you use the `derive_pll_clocks` command to automatically constrain the PLL for the design shown in Figure 7-9.

#### Example 7-17. 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]` is used for 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 are created for the output clock of the PLL; one for the primary input clock (for example, `inc1k[0]`), and one for the secondary input clock (for example, `inc1k[1]`). You should create exclusive clock groups for the primary and secondary output clocks.

For more information about creating exclusive clock groups, refer to “Creating Clock Groups” on page 7-21.

## Automatically Detecting Clocks and Creating Default Clock Constraints


To automatically create clocks for all clock nodes in your design, use the `derive_clocks` command. 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.

If there are no clock constraints in your design, the TimeQuest analyzer automatically creates default clock constraints for all detected unconstrained clock nodes to provide a complete clock analysis. The TimeQuest analyzer automatically creates clocks only when all synchronous elements have no associated clocks. For example, the TimeQuest analyzer does not create a default clock constraint if your design contains two clocks and you assigned constraints to one of the clocks. However, if you do not assign constraints to either clock, then the TimeQuest analyzer creates a default clock constraint.


Example 7-18 shows how the TimeQuest analyzer creates a base clock with a 1 GHz requirement for unconstrained clock nodes.

### Example 7-18. Create Base Clock for Unconstrained Clock Nodes

```
derive_clocks -period 1
```

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

You can also use the command `derive_pll_clocks -create_base_clocks` to create the input clocks for all PLL inputs automatically.

-  For more information about the `derive_clocks` command—including full syntax information, options, and example usage—refer to *derive\_clocks* in Quartus II Help.

## Creating Clock Groups

The TimeQuest analyzer assumes all clocks are related unless constrained otherwise. To specify clocks in your design that are exclusive or asynchronous, use the `set_clock_groups` command. The `set_clock_groups` command cuts timing between clocks in different groups, and performs the same analysis regardless of whether you specify `-exclusive` or `-asynchronous`.

- ② For more information about the `set_clock_groups` command—including full syntax information, options, and example usage—refer to *set\_clock\_groups* in Quartus II Help.

### Exclusive Clock Groups

Use the `-exclusive` option to declare that two clocks are mutually exclusive. You may want to declare clocks as mutually exclusive when multiple clocks are created on the same node or for multiplexed clocks. For example, a port can be clocked by either a 25-MHz or a 50-MHz clock. To constrain this port, you should create two clocks on the port, and then create clock groups to declare that they cannot coexist in the design at the same time. Declaring the clocks as mutually exclusive eliminates any clock transfers that may be derived between the 25-MHz clock and the 50-MHz clock.

Example 7-19 shows how to create mutually exclusive clock groups.

#### Example 7-19. Create Mutually Exclusive Clock Groups

---

```
create_clock -period 40 -name clk_A [get_ports {port_A}]
create_clock -add -period 20 -name clk_B [get_ports {port_A}]
set_clock_groups -exclusive -group {clk_A} -group {clk_B}
```

---

A group is defined with the `-group` option. The TimeQuest analyzer excludes the timing paths between clocks for each of the separate groups.

If you apply multiple clocks to the same port, use the `set_clock_groups` command with the `-exclusive` option to place the clocks into separate groups and declare that the clocks are mutually exclusive. The clocks cannot physically exist in your design at the same time.

### Asynchronous Clock Groups

Use the `-asynchronous` option to create asynchronous clock groups. Clocks contained within an asynchronous clock group are considered asynchronous to clocks in other clock groups; however, any clocks within a clock group are considered synchronous to each other.

For example, if your design has three clocks, `clk_A`, `clk_B`, and `clk_C`, and you establish that `clk_A` and `clk_B` are related to each other, but clock `clk_C` operates completely asynchronously, you can set up clock groups to define the clock behavior. If `set_clock_groups` is used with only one group, the clocks in that group are asynchronous with all other clocks in the design. For example, you can create a clock group containing only `clk_C` to ensure that `clk_A` and `clk_B` are synchronous with each other and asynchronous with `clk_C`. Because `clk_C` is the only clock in the constraint, it is asynchronous with every other clock in the design.

Example 7-20 shows how to create a clock group containing clocks `clk_A` and `clk_B` and a second unrelated clock group containing `clk_C`.

#### Example 7-20. Create Asynchronous Clock Groups

---

```
set_clock_groups -asynchronous -group {clk_A clk_B} -group {clk_C}
```

---

- ② For more information about the `set_clock_groups` command—including full syntax information, options, and example usage—refer to *set\_clock\_groups* in Quartus II Help.



## Accounting for Clock Effect Characteristics

The clocks you create with the TimeQuest 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.

### Clock Latency

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.

-  The TimeQuest analyzer automatically computes network latencies; therefore, you only can characterize source latency with the `set_clock_latency` command. You must use the `-source` option.
-  For more information about the `set_clock_latency` command—including full syntax information, options, and example usage—refer to *set\_clock\_latency* in Quartus II Help.f

### Clock Uncertainty

The TimeQuest 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 core of the device and the source and destination clocks come from the same PLL output pin or clock port.
- Interlock 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, or skew, for clock-to-clock transfers, use the `set_clock_uncertainty` command. You can specify the uncertainty separately for setup and hold, and you can specify separate rising and falling clock transitions. The TimeQuest 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.

To automatically apply interclock, intraclock, and I/O interface uncertainties, use the `derive_clock_uncertainty` command. The TimeQuest analyzer automatically applies clock uncertainties to clock-to-clock transfers in the design, and calculates both setup and hold uncertainties for each clock-to-clock transfer.

Any clock uncertainty constraints applied to source and destination clock pairs with the `set_clock_uncertainty` command have a higher precedence than the clock uncertainties derived with the `derive_clock_uncertainty` command for the same source and destination clock pairs. For example, if you use the `set_clock_uncertainty` command to set clock uncertainty between `clk_a` and `clk_b`, the TimeQuest analyzer ignores the values for the clock transfer calculated with the `derive_clock_uncertainty` command. The TimeQuest analyzer reports the values calculated with the `derive_clock_uncertainty` command even if they are not used.

Use `set_clock_uncertainty` or `derive_clock_uncertainty` with the `-overwrite` option to overwrite previously applied clock uncertainty assignments. Use `set_clock_uncertainty` or `derive_clock_uncertainty` with the `-add` option to apply additional clock uncertainty to previously applied clock uncertainty. Use the `remove_clock_uncertainty` command to remove previous clock uncertainty assignments.

- ② For more information about the `set_clock_uncertainty`, `derive_clock_uncertainty`, and `remove_clock_uncertainty` commands—including full syntax information, options, and example usage—refer to *set\_clock\_uncertainty*, *remove\_clock\_uncertainty* and *derive\_clock\_uncertainty*, in Quartus II Help.

## Creating I/O Requirements

The TimeQuest analyzer reviews setup and hold relationships for designs in which an external source interacts with a register internal to the design. The TimeQuest 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.

You must specify timing requirements, including internal and external timing requirements, before you fully analyze a design. With external timing requirements specified, the TimeQuest analyzer verifies the I/O interface, or periphery of the device, against any system specification.

### Input Constraints

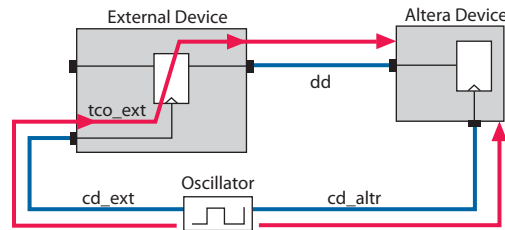
Input constraints allow you to specify all the external delays feeding into the device. Specify input requirements for all input ports in your design.



You can use the `set_input_delay` command to specify external input delay requirements. Use the `-clock` option to reference a virtual clock. Using a virtual clock allows the TimeQuest analyzer to correctly derive clock uncertainties for interclock and intraclock transfers. The virtual clock defines the launching clock for the input port. The TimeQuest analyzer automatically determines the latching clock inside the device that captures the input data, because all clocks in the device are defined.

Figure 7-10 shows an example of an input delay referencing a virtual clock.

**Figure 7-10. Input Delay**



Equation 7-1 shows a typical input delay calculation.

**Equation 7-1. Input Delay Calculation**

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

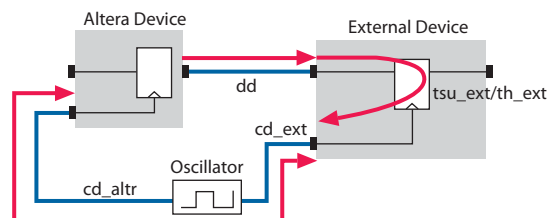
$$\text{input delay}_{\text{MIN}} = (\text{cd\_ext}_{\text{MIN}} - \text{cd\_altr}_{\text{MAX}}) + \text{tco\_ext}_{\text{MIN}} + \text{dd}_{\text{MIN}}$$

## Output Constraints

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

You can use the `set_output_delay` command to specify external output delay requirements. Use the `-clock` option to reference a virtual clock. The virtual clock defines the latching clock for the output port. The TimeQuest analyzer automatically determines the launching clock inside the device that launches the output data, because all clocks in the device are defined. Figure 7-11 shows an example of an output delay referencing a virtual clock.

**Figure 7-11. Output Delay**



Equation 7-2 shows a typical output delay calculation.

---

#### Equation 7-2. output Delay Calculation

---

$$\text{output delay}_{\text{MAX}} = \text{dd}_{\text{MAX}} + \text{tsu}_{\text{ext}} + (\text{cd}_{\text{altr}_{\text{MAX}}} - \text{cd}_{\text{ext}_{\text{MIN}}})$$

$$\text{output delay}_{\text{MIN}} = (\text{dd}_{\text{MIN}} + \text{th}_{\text{ext}} + (\text{cd}_{\text{altr}_{\text{MIN}}} - \text{cd}_{\text{ext}_{\text{MAX}}}))$$


---

- ② For more information about the `set_input_delay` and `set_output_delay` commands—including full syntax information, options, and example usage—refer to *set\_input\_delay* and *set\_output\_delay* in Quartus II Help.

## Creating Delay and Skew Constraints

The TimeQuest analyzer supports the Synopsys Design Constraint format for constraining timing for the ports in your design. These constraints allow the TimeQuest analyzer to perform a system static timing analysis that includes not only the device internal timing, but also any external device timing and board timing parameters.

### Advanced I/O Timing and Board Trace Model Delay

The TimeQuest analyzer can use advanced I/O timing and board trace model assignments 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 7-21 shows how to force delay annotation when creating a timing netlist.


#### Example 7-21. Forcing Delay Annotation

---

```
create_timing_netlist -force_dat ←
```

---

- ② For more information about using advanced I/O timing, refer to *Using Advanced I/O Timing* in Quartus II Help.

 For more information about advanced I/O timing, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

## Maximum Skew

To specify the maximum path-based skew requirements for registers and ports in the design and report the results of maximum skew analysis, use the `set_max_skew` command in conjunction with the `report_max_skew` command.

By default, the `set_max_skew` command excludes any input or output delay constraints.

- ② For more information about the `set_max_skew` and `report_max_skew` commands—including full syntax information, options, and example usage—refer to *set\_max\_skew* *report\_max\_skew* in Quartus II Help.

## Creating Timing Exceptions

Timing exceptions in the TimeQuest analyzer provide a way to modify the default timing analysis behavior to match the analysis required by your design. Specify timing exceptions after clocks and input and output delay constraints because timing exceptions can modify the default analysis.

### Precedence

If a conflict of node names occurs between timing exceptions, the following order of precedence applies:

1. False path
2. Minimum delays and maximum delays
3. Multicycle path

The false path timing exception has the highest precedence. Within each category, assignments to individual nodes have precedence over assignments to clocks. Finally, the remaining precedence for additional conflicts is order-dependent, such that the assignments most recently created overwrite, or partially overwrite, earlier assignments.

### False Paths

Specifying a false path in your design removes the path from timing analysis. Use the `set_false_path` command to specify false paths in your design. You can specify either a point-to-point or clock-to-clock path as a false path. For example, a path you should specify as false path is a static configuration register that is written once during power-up initialization, but does not change state again. Although signals from static configuration registers often cross clock domains, you may not want to make false path exceptions to a clock-to-clock path, because some data may transfer across clock domains. However, you can selectively make false path exceptions from the static configuration register to all endpoints.

[Example 7-22](#) shows how to make false path exceptions from all registers beginning with A to all registers beginning with B.

#### **Example 7-22. False Path**

---

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

---

The TimeQuest analyzer assumes all clocks are related unless you specify otherwise. The [“Creating Clock Groups” on page 7-21](#) describes how you can use clock groups. Clock groups are a more efficient way to make false path exceptions between clocks, compared to writing multiple `set_false_path` exceptions between every clock transfer you want to eliminate.

- ❓ For more information about the `set_false_path` command—including full syntax information, options, and example usage—refer to [set\\_false\\_path](#) in Quartus II Help.

## Minimum and Maximum Delays

To specify an absolute minimum or maximum delay for a path, use the `set_min_delay` command or the `set_max_delay` commands, 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` commands to create 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 TimeQuest 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, that delay is also included in the minimum or maximum delay check.

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 source node or destination node is clocked by the clock.

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 associated with the output port; however, the TimeQuest analyzer reports any slack for the path in the setup summary and hold summary reports. Because there is no clock associated with the output port, no clock is reported for timing paths associated with the output port.



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.



For more information about the `set_min_delay`, and `set_max_delay`, commands—including full syntax information, options, and example usage—refer to *set\_min\_delay*, and *set\_max\_delay*, in Quartus II Help.

## 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 see the results of these commands

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



For more information about the `set_annotated_delay` and `set_timing_derate` commands—including full syntax information, options, and example usage—refer to *set\_annotated\_delay* and *set\_timing\_derate* in Quartus II Help.

## Multicycle Paths

By default, the TimeQuest analyzer performs a single-cycle analysis, which is the most restrictive type of analysis. When analyzing a path, the setup launch and latch edge times are determined by finding the closest two active edges in the respective waveforms. For a hold analysis, the timing analyzer analyzes the path against two timing conditions for every possible setup relationship, not just the worst-case setup relationship. Therefore, the hold launch and latch times may be completely unrelated to the setup launch and latch edges. The TimeQuest analyzer does not report negative setup or hold relationships. When either a negative setup or a negative hold relationship is calculated, the TimeQuest analyzer moves both the launch and latch edges such that the setup and hold relationship becomes positive.

A multicycle constraint adjusts setup or hold relationships by the specified number of clock cycles based on the source (-start) or destination (-end) clock. An end setup multicycle constraint of 2 extends the worst-case setup latch edge by one destination clock period. If -start and -end values are not specified, the default constraint is -end.

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

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

Table 7-4 shows the commands you can use to modify either the launch or latch edge times that the TimeQuest analyzer uses to determine a setup relationship or hold relationship.

**Table 7-4. Commands to Modify Edge Times**

Command	Description of 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

## Common Multicycle Variations

Multicycle exceptions adjust the timing requirements for a register-to-register path, allowing the Fitter to optimally place and route a design in a device. Multicycle exceptions also can reduce compilation time and improve the quality of results, and can be used to change timing requirements. Two common multicycle variations are relaxing setup to allow a slower data transfer rate, and altering the setup to account for a phase shift.

### Relaxing Setup with set\_multicycle\_path

A common type of multicycle exception occurs when the data transfer rate is slower than the clock cycle.

In this example, the source clock has a period of 10 ns, but a group of registers are enabled by a toggling clock, so they only toggle every other cycle. Since they are fed by a 10 ns clock, the TimeQuest analyzer reports a set up of 10 ns and a hold of 0 ns. However, since the data is transferring every other cycle, the relationships should be analyzed as if the clock were operating at 20 ns, which would result in a setup of 20 ns, while the hold remains 0 ns, in essence, extending the window of time when the data can be recognized.

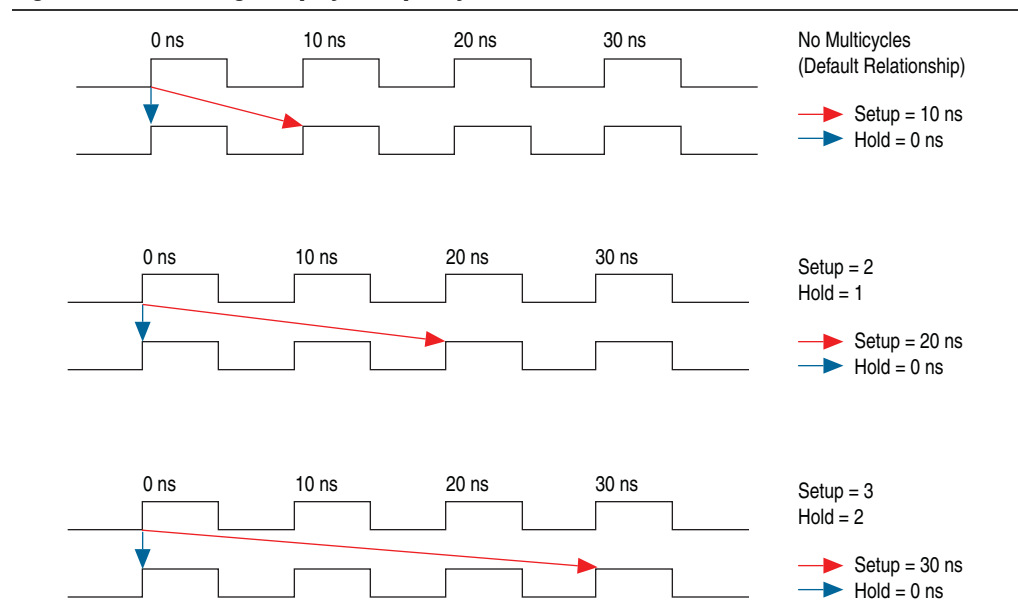
Example 7-23 shows a pair of multicycle assignments that relax the setup relationship by specifying the `-setup` value of `N` and the `-hold` value as `N-1`. You must specify the hold relationship with a `-hold` assignment to prevent a positive hold requirement.

### Example 7-23. Relaxing Setup while Maintaining Hold

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

Figure 7-12 shows how the exception relaxes the setup by two or three cycles.

Figure 7-12. Relaxing Setup by Multiple Cycles



This pattern can be extended to create larger setup relationships in order 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 can allow extra time for the address and data to resolve before the enable occurs.

Example 7-24 shows how a relaxing the setup by three cycles can be achieved.

### Example 7-24. Three Cycle I/O Interface Exception

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

## Accounting for a Phase Shift

In this 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. For example, when the destination clock is phase-shifted forward and the source clock is not, the default setup relationship becomes that phase-shift.

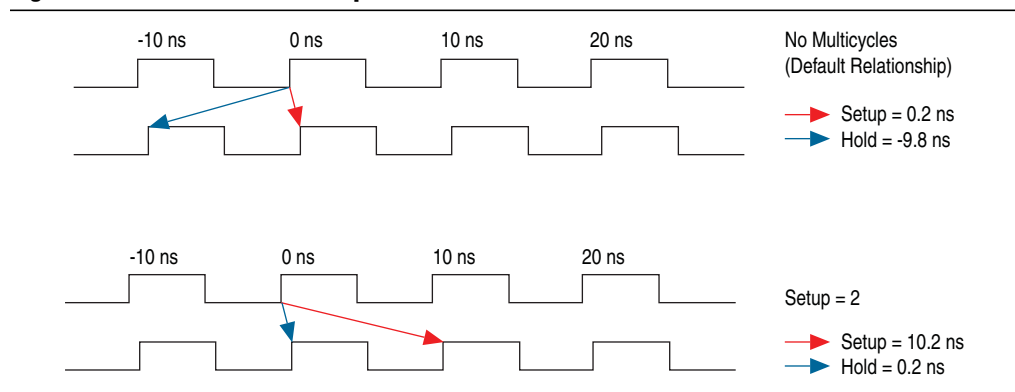
Example 7–25 shows a circumstance where a PLL phase-shifts one output forward by a small amount, for example 0.2 ns.

### Example 7–25. 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 7–13. Phase-Shifted Setup and Hold



Adding the constraint shown in Example Y allows the data to transfer to the following edge.

### Example 7–26. Adjusting the Phase-Shift with a `set_multicycle_path` Constraint

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

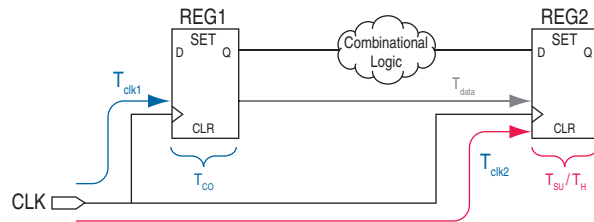
The hold relationship is derived from the setup relationship, making a multicycle hold constraint unnecessary. For a more complete example refer to “Same Frequency Clocks with Destination Clock Offset” on page 7–44.

- ① For more information about the `set_multicycle_path` command—including full syntax information, options, and example usage—refer to `set_multicycle_path` in Quartus II Help.

## Multicycle Clock Setup Check and Hold Check Analysis

You can modify the setup and hold relationship when you apply a multicycle exception to a register-to-register path. Figure 7-14 shows a register-to-register path with various timing parameters labeled.

**Figure 7-14. Register-to-Register Path**

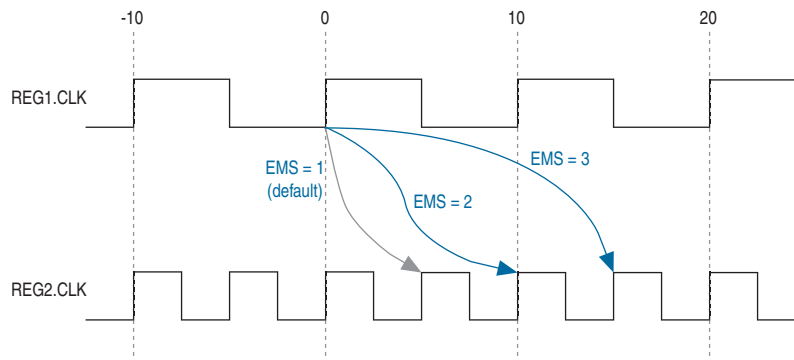


### Multicycle Clock Setup

The setup relationship is defined as the number of clock periods between the latch edge and the launch edge. By default, the TimeQuest 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. Figure 7-15 shows various values of the end multicycle setup assignment and the resulting latch edge.

**Figure 7-15. 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. Figure 7-16 shows various values of the start multicycle setup assignment and the resulting launch edge.

**Figure 7-16. Start Multicycle Setup Values**

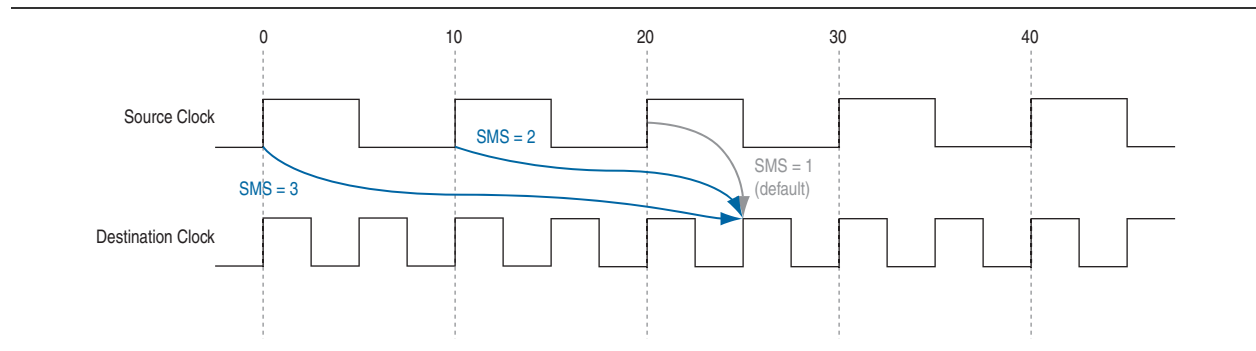
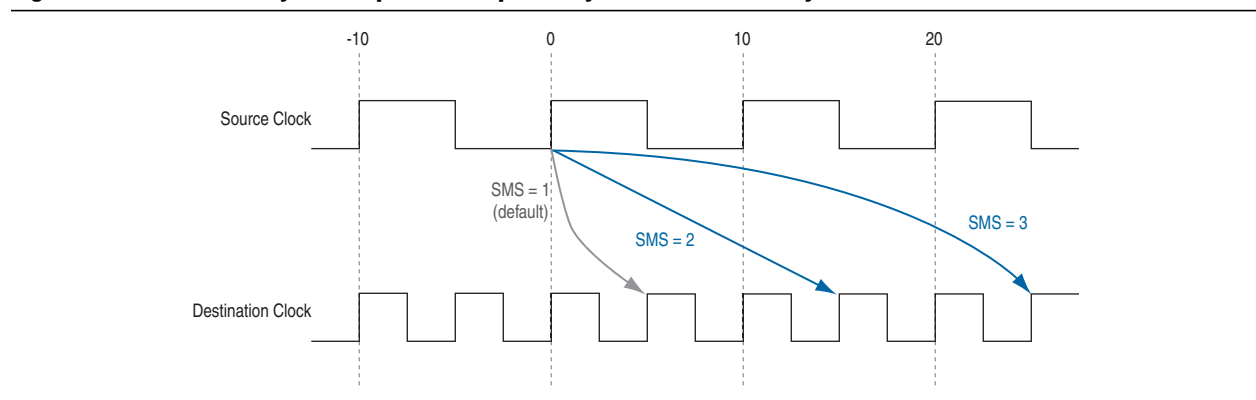


Figure 7-17 shows the setup relationship reported by the TimeQuest analyzer for the negative setup relationship shown in Figure 7-16.

**Figure 7-17. Start Multicycle Setup Values Reported by the TimeQuest Analyzer**




### Multicycle Clock Hold

The setup relationship is defined as the number of clock periods between the launch edge and the latch edge. By default, the TimeQuest analyzer performs a single-cycle path analysis, which results in the hold relationship being equal to one clock period (launch edge – latch edge). When analyzing a path, the TimeQuest analyzer performs two hold checks. 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. The TimeQuest analyzer reports only the most restrictive hold check. Equation 7-3 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

#### Equation 7-3. Hold Check

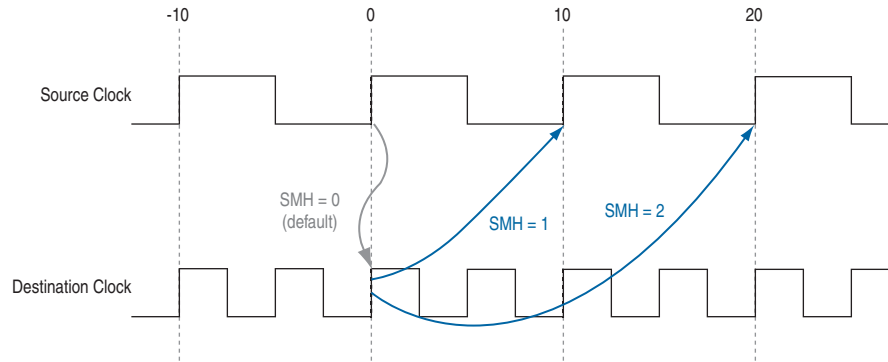
hold check 1 = current launch edge – previous latch edge

hold check 2 = next launch edge – current latch edge

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

A start multicycle hold assignment modifies the launch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default launch edge. Figure 7-18 shows various values of the start multicycle hold assignment and the resulting launch edge.

**Figure 7-18. Start Multicycle Hold Values**



An end multicycle hold assignment modifies the latch edge of the destination clock by moving the latch edge the specific ed number of clock periods to the left of the determined default latch edge. Figure 7-19 shows various values of the end multicycle hold assignment and the resulting latch edge.

**Figure 7-19. End Multicycle Hold Values**

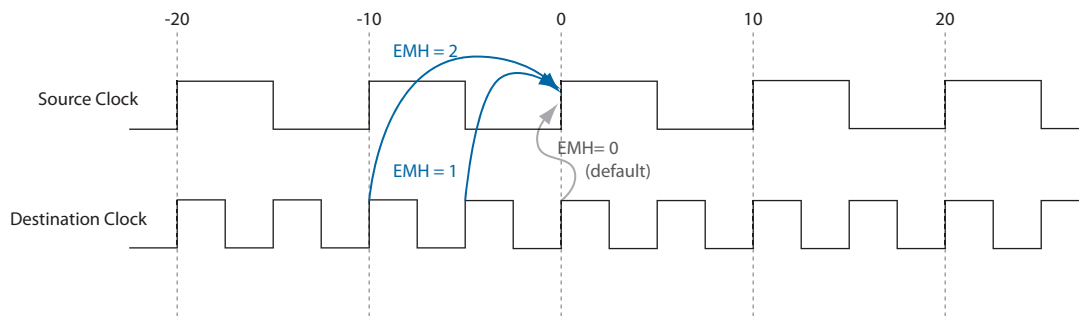
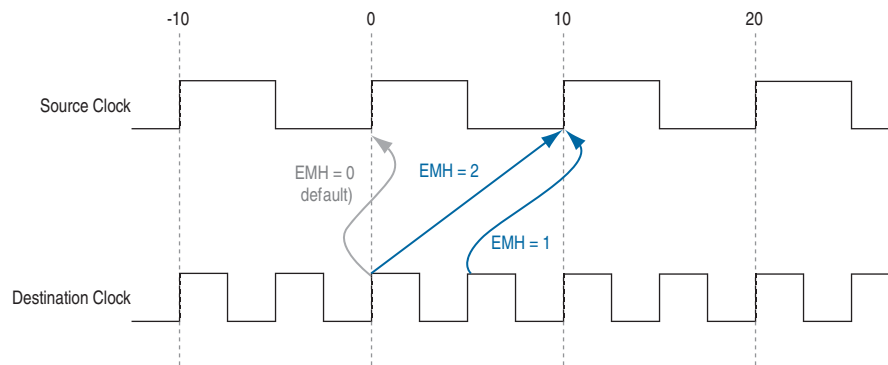


Figure 7-20 shows the hold relationship reported by the TimeQuest analyzer for the negative hold relationship shown in Figure 7-19.

**Figure 7-20. End Multicycle Hold Values Reported by the TimeQuest Analyzer**



## Examples of Basic Multicycle Exceptions

This section describes the following examples of combinations of multicycle exceptions:

- “Default Settings” on page 7-35
- “End Multicycle Setup = 2 and End Multicycle Hold = 0” on page 7-38
- “End Multicycle Setup = 2 and End Multicycle Hold = 1” on page 7-41

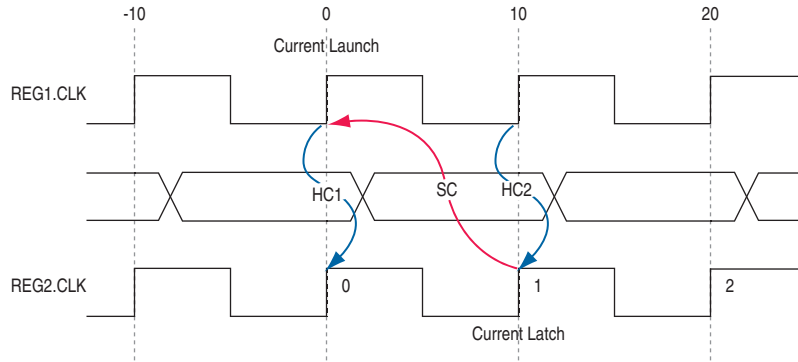
Each example explains how the multicycle exceptions affect the default setup and hold analysis in the TimeQuest analyzer. The multicycle exceptions are applied to a simple register-to-register circuit. Both the source and destination clocks are set to 10 ns.

### Default Settings

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

Figure 7-21 shows the source and the destination timing waveform for the source register and destination register, respectively where HC1 and HC2 are hold checks one and two and SC is the setup check.

**Figure 7-21. Default Timing Diagram**



Equation 7-4 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

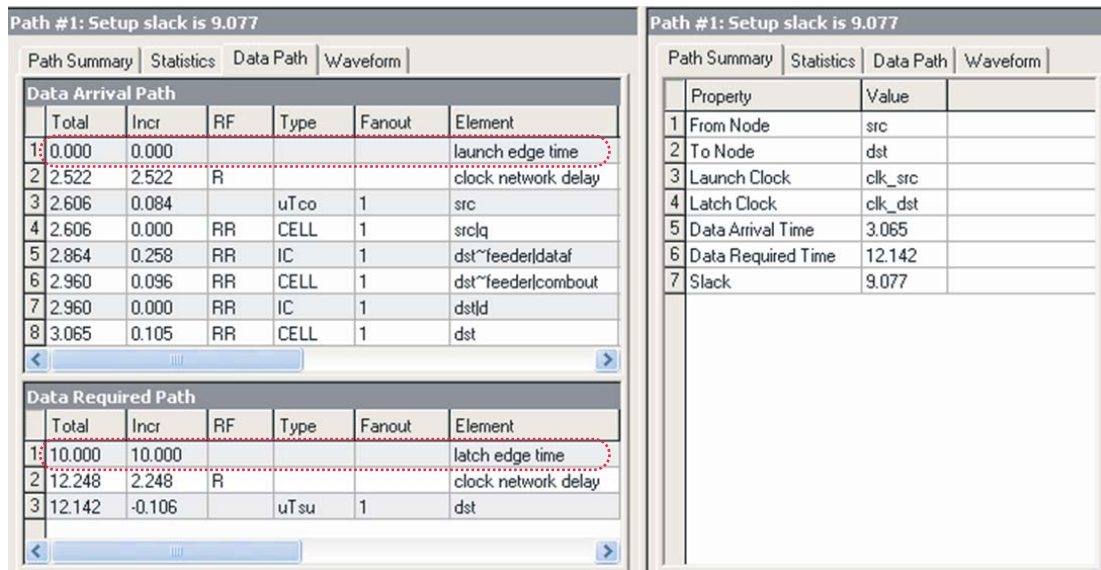
**Equation 7-4. Setup Check**

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

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

Figure 7-22 shows the setup report for the default setup in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-22. Setup Report**



Equation 7-5 shows the calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

**Equation 7-5. Hold Check**

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

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

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

Figure 7-23 shows the hold report for the default setup in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-23. Hold Report**

Path #1: Hold slack is 0.119						
Data Arrival Path						
	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.258	2.258	R			clock network delay
3	2.342	0.084		uTco	1	src
4	2.342	0.000	FF	CELL	1	srcdq
5	2.619	0.277	FF	IC	1	dst~feeder dataf
6	2.684	0.065	FF	CELL	1	dst~feeder combout
7	2.684	0.000	FF	IC	1	dstld
8	2.771	0.087	FF	CELL	1	dst

Path #1: Hold slack is 0.119						
Data Required Path						
	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				latch edge time
2	2.513	2.513	R			clock network delay
3	2.652	0.139		uTh	1	dst

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

### 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. Example 7-27 shows the multicycle exceptions applied to the register-to-register design for this example.

#### Example 7-27. Multicycle Exceptions

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

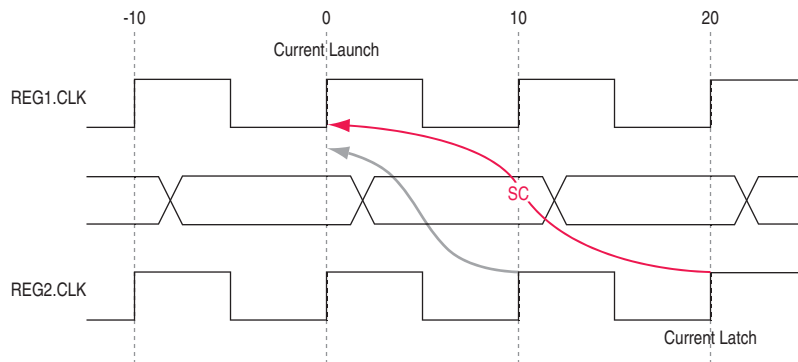


An end multicycle hold value is not required because the default end multicycle hold value is zero.

In this example, the setup relationship is relaxed by a full clock period by moving the latch edge to the next latch edge. The hold analysis is unchanged from the default settings.

Figure 7-24 shows the setup timing diagram. The latch edge is a clock cycle later than in the default single-cycle analysis.

**Figure 7-24. Setup Timing Diagram**



Equation 7-6 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

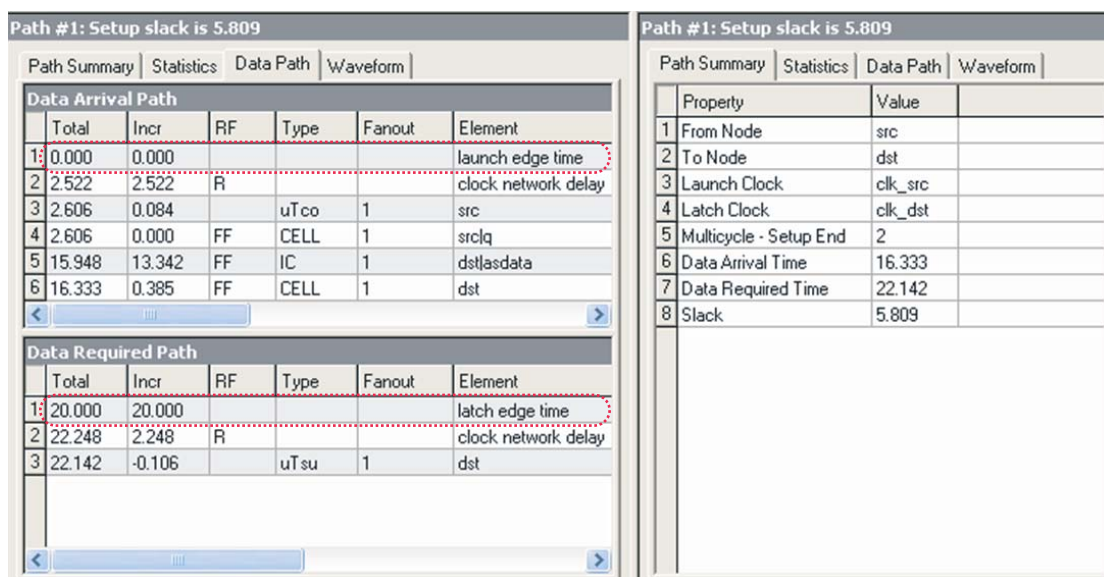
**Equation 7-6. Setup Check**

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

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

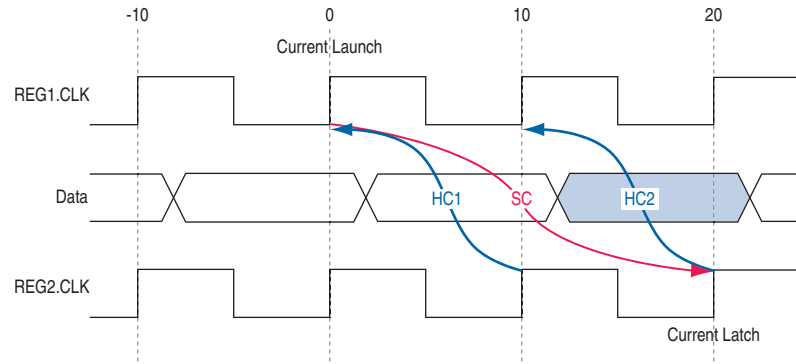
Figure 7-25 shows the setup report in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-25. Setup Report**



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. Figure 7-26 shows the timing diagram for the hold checks for this example. The hold checks are relative to the setup check. Usually, the TimeQuest analyzer performs hold checks on every possible setup check, not only on the most restrictive setup check edges.

**Figure 7-26. Hold Timing Diagram**



Equation 7-7 shows the calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

**Equation 7-7. Hold Check**

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

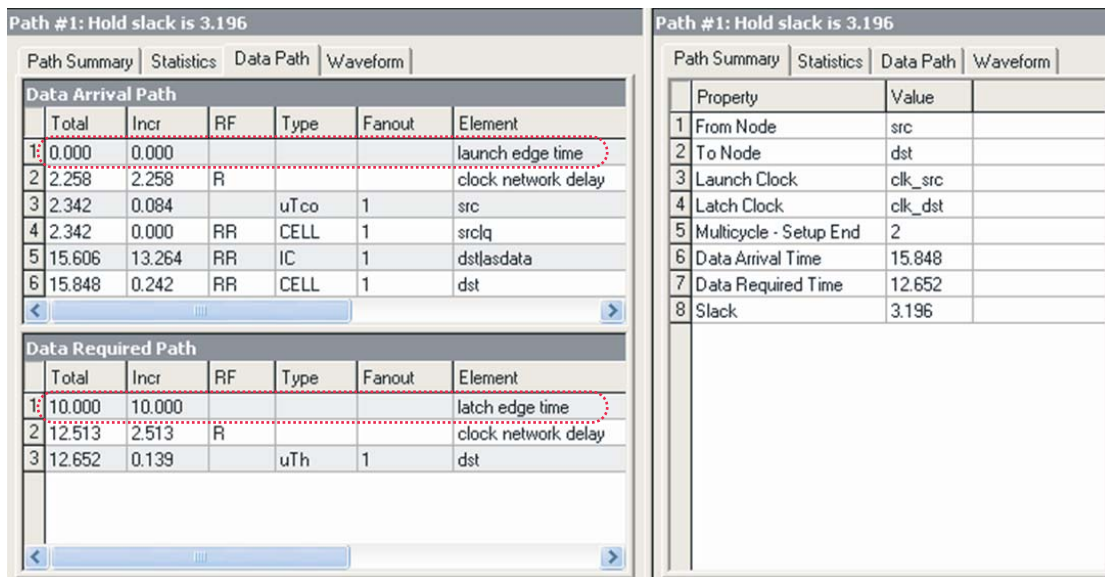
$$\begin{aligned}
 \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\
 &= 10 \text{ ns} - 20 \text{ ns} \\
 &= -10 \text{ ns}
 \end{aligned}$$

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



Figure 7-27 shows the hold report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-27. Hold Report**



### 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. Example 7-28 shows the multicycle exceptions applied to the register-to-register design for this example.

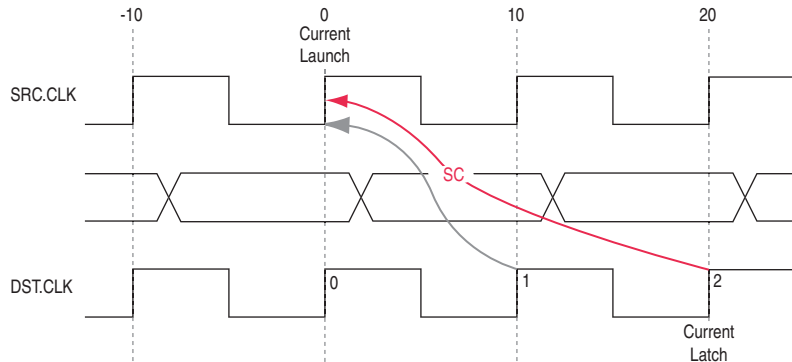
#### Example 7-28. Multicycle Exceptions

```
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 is relaxed by two clock periods by moving the latch edge to the left two clock periods. The hold relationship is relaxed by a full period by moving the latch edge to the previous latch edge.

Figure 7-28 shows the setup timing diagram.

**Figure 7-28. Setup Timing Diagram**



Equation 7-8 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

**Equation 7-8. Setup Check**

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

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

Figure 7-29 shows the setup report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-29. Setup Report**

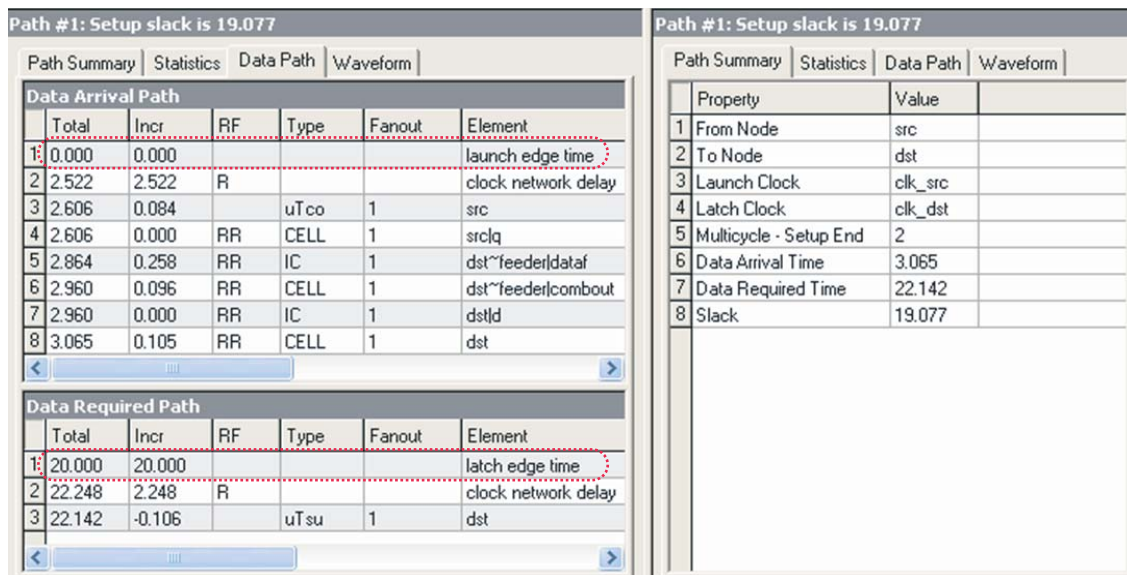
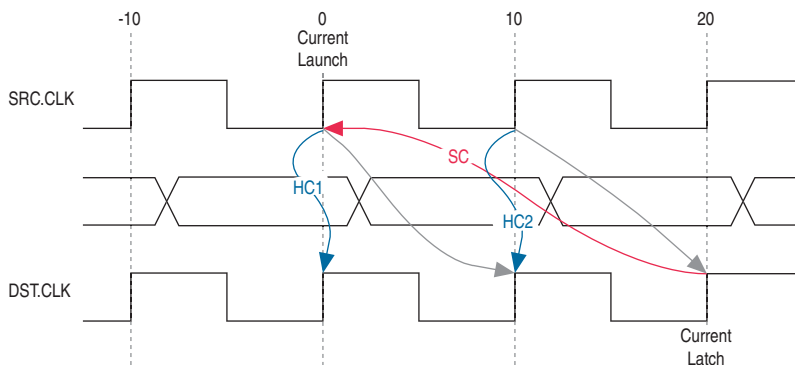


Figure 7-30 shows the timing diagram for the hold checks for this example. The hold checks are relative to the setup check.

**Figure 7-30. Hold Timing Diagram**



Equation 7-9 shows the calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

**Equation 7-9. Hold Check**

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

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

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.

Figure 7-31 shows the hold report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

**Figure 7-31. Hold Report**

Path #1: Hold slack is 0.119

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

Path #1: Hold slack is 0.119

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

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

## Application of Multicycle Exceptions

This section shows the following examples of applications of multicycle exceptions:

- “Same Frequency Clocks with Destination Clock Offset” on page 7-44
- “The Destination Clock Frequency is a Multiple of the Source Clock Frequency” on page 7-47
- “The Destination Clock Frequency is a Multiple of the Source Clock Frequency with an Offset” on page 7-50
- “The Source Clock Frequency is a Multiple of the Destination Clock Frequency” on page 7-52
- “The Source Clock Frequency is a Multiple of the Destination Clock Frequency with an Offset” on page 7-55

Each example explains how the multicycle exceptions affect the default setup and hold analysis in the TimeQuest analyzer. All of the examples are between related clock domains. If your design contains related clocks, such as PLL clocks, and paths between related clock domains, you can apply multicycle constraints.

### 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. Figure 7-32 shows an example of a design with same frequency clocks and a destination clock offset.

**Figure 7-32. Same Frequency Clocks with Destination Clock Offset**

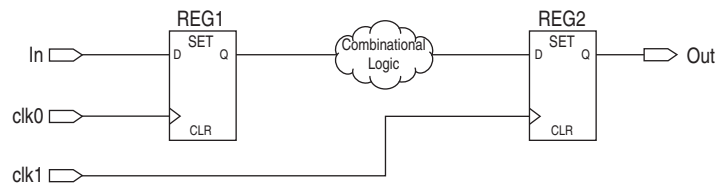
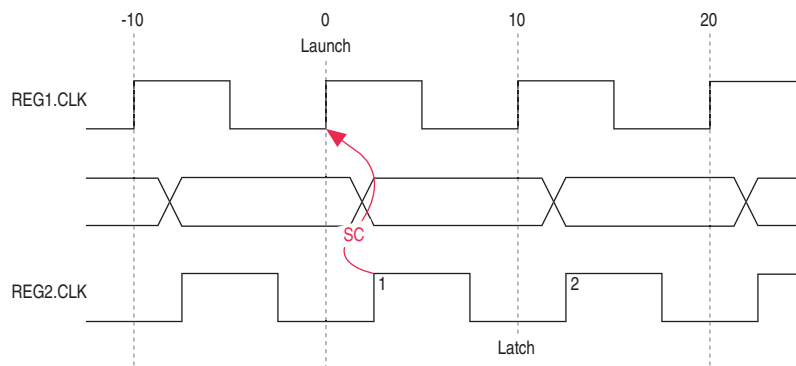


Figure 7-33 shows the timing diagram for default setup check analysis performed by the TimeQuest analyzer.

**Figure 7-33. Setup Timing Diagram**



Equation 7-10 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

**Equation 7-10. Setup Check**

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

The setup relationship shown in Figure 7-33 is too pessimistic and is not the setup relationship required for typical designs. To correct the default analysis, you must use an end multicycle setup exception of two. Example 7-29 shows the multicyle exception used to correct the default analysis in this example.

**Example 7-29. Multicycle Exceptions**

```

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

Figure 7-34 shows the timing diagram for the preferred setup relationship for this example.

**Figure 7-34. Preferred Setup Relationship**

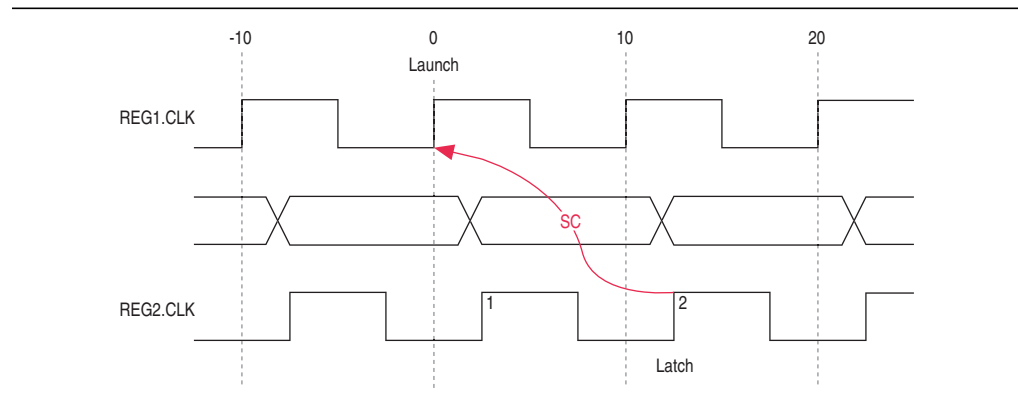
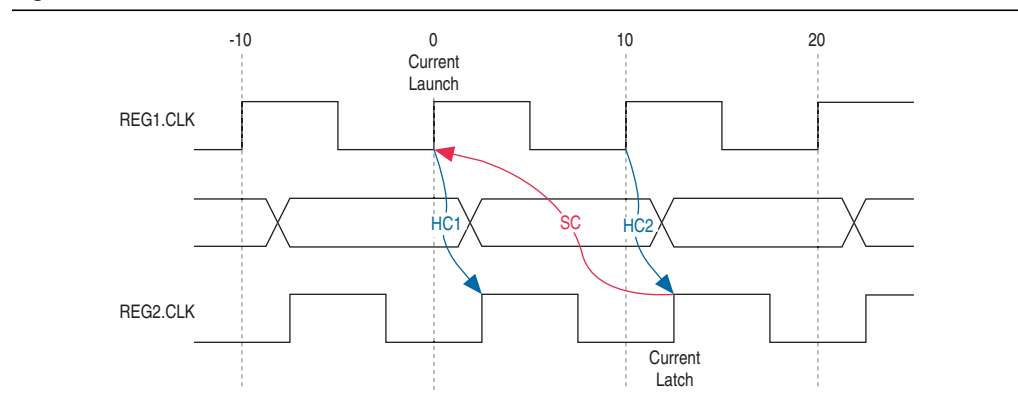


Figure 7-35 shows the timing diagram for default hold check analysis performed by the TimeQuest analyzer with an end multicycle setup value of two.

**Figure 7-35. Default Hold Check**



Equation 7-11 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

**Equation 7-11. Hold Check**

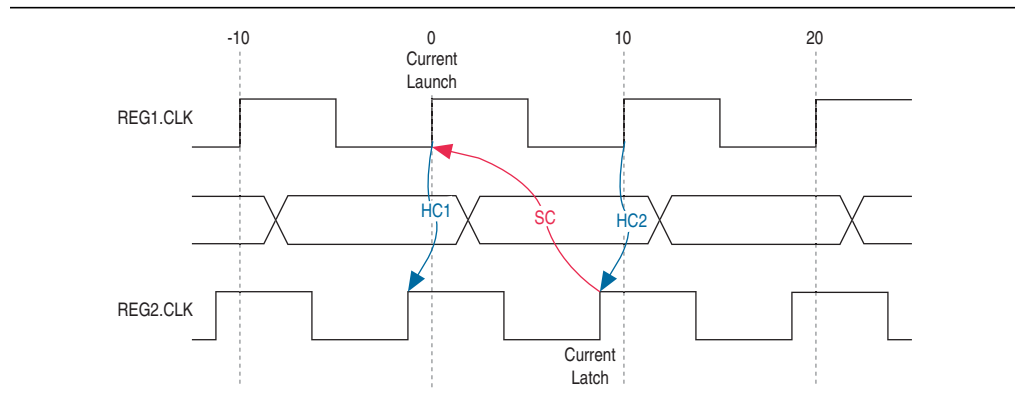
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0 \text{ ns} - 2 \text{ ns} \\ &= -2 \text{ ns} \end{aligned}$$

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

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

Figure 7-36 shows 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 7-36. Negative Phase Shift**



### The 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 is used to generate both clocks with a phase shift applied to the destination clock. Figure 7-37 shows an example of a design where the destination clock frequency is a multiple of the source clock frequency.

**Figure 7-37. Destination Clock is Multiple of Source Clock**

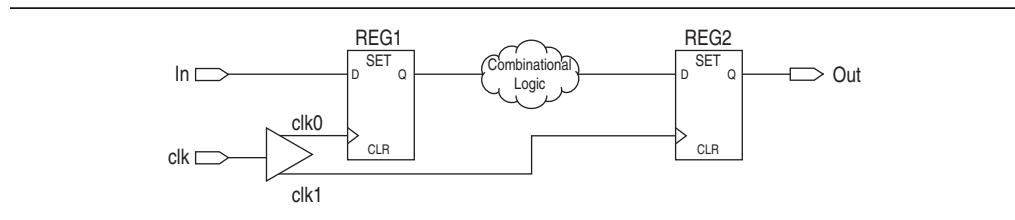
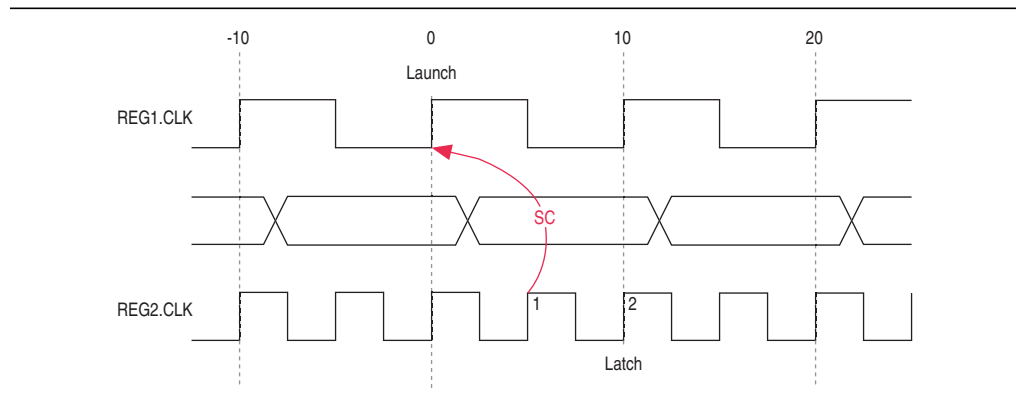


Figure 7-38 shows the timing diagram for default setup check analysis performed by the TimeQuest analyzer.

**Figure 7-38. Setup Timing Diagram**



Equation 7-12 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

**Equation 7-12. Setup Check**

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

The setup relationship shown in Figure 7-38 demonstrates that the data does not need to be captured at edge one, but can be captured at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the latch edge by one clock period with an end multicycle setup exception of two. Example 7-30 shows the multicycle exception used to correct the default analysis in this example.

**Example 7-30. Multicycle Exceptions**

```

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

```



Figure 7-39 shows the timing diagram for the preferred setup relationship for this example.

**Figure 7-39. Preferred Setup Analysis**

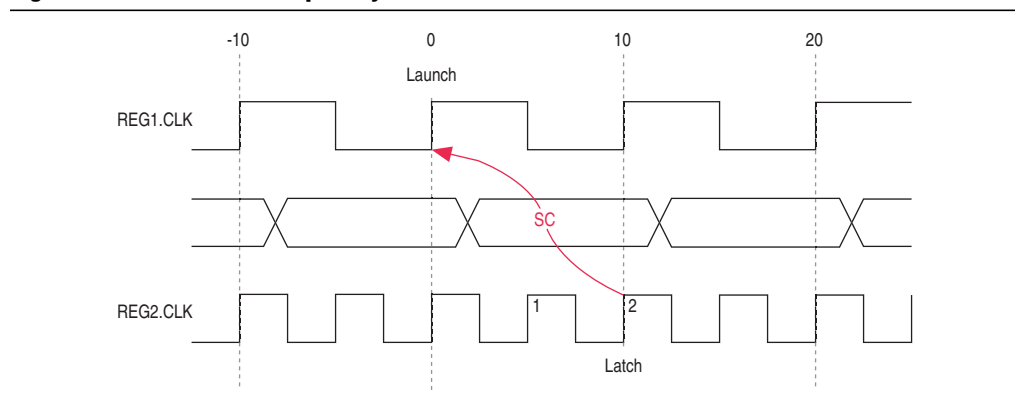
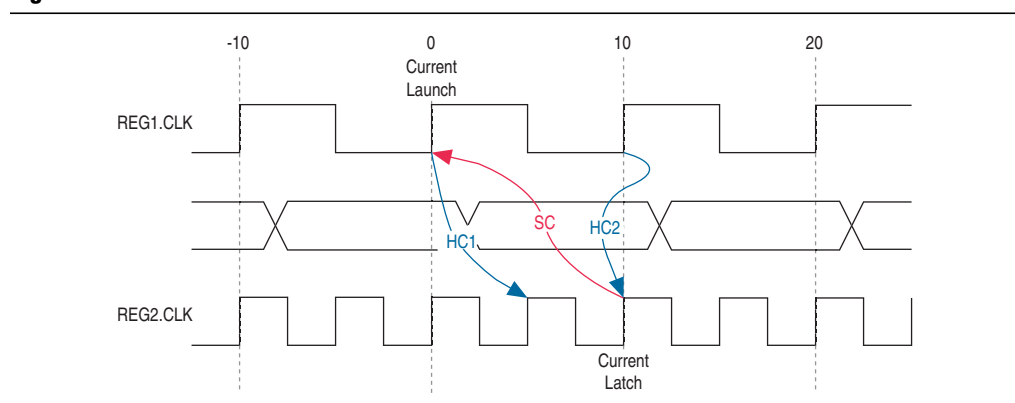


Figure 7-40 shows the timing diagram for default hold check analysis performed by the TimeQuest analyzer with an end multicycle setup value of two.

**Figure 7-40. Default Hold Check**



Equation 7-13 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

**Equation 7-13. Hold Check**

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

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

In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and should 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.

## The 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 with an offset when a PLL is used to generate both clocks with a phase shift applied to the destination clock. Figure 7-41 shows an example of a design in which the destination clock frequency is a multiple of the source clock frequency with an offset.

**Figure 7-41. Destination Clock is Multiple of Source Clock with Offset**

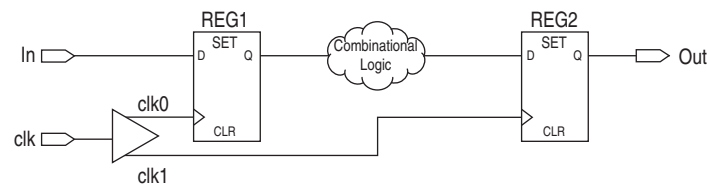
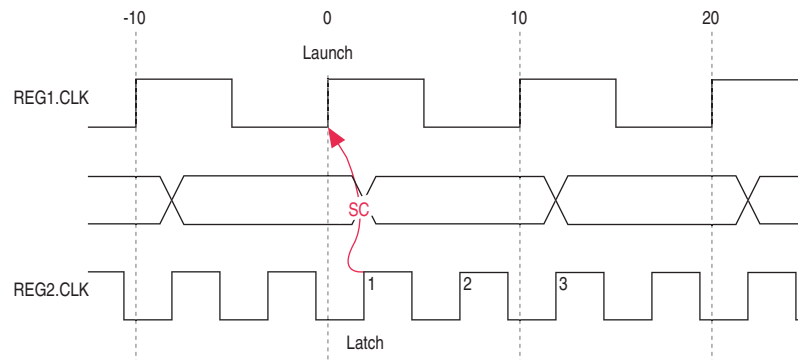


Figure 7-42 shows the timing diagram for default setup check analysis performed by the TimeQuest analyzer.

**Figure 7-42. Setup Timing Diagram**



Equation 7-14 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

### Equation 7-14. Setup Check

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

The setup relationship shown in Figure 7-42 demonstrates that the data does not need to be captured at edge one, but can be captured at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the latch edge by one clock period with an end multicycle setup exception of three.

Example 7-31 shows the multicycle exception used to correct the default analysis in this example.

**Example 7-31. Multicycle Exceptions**

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

Figure 7-43 shows the timing diagram for the preferred setup relationship for this example.

**Figure 7-43. Preferred Setup Analysis**

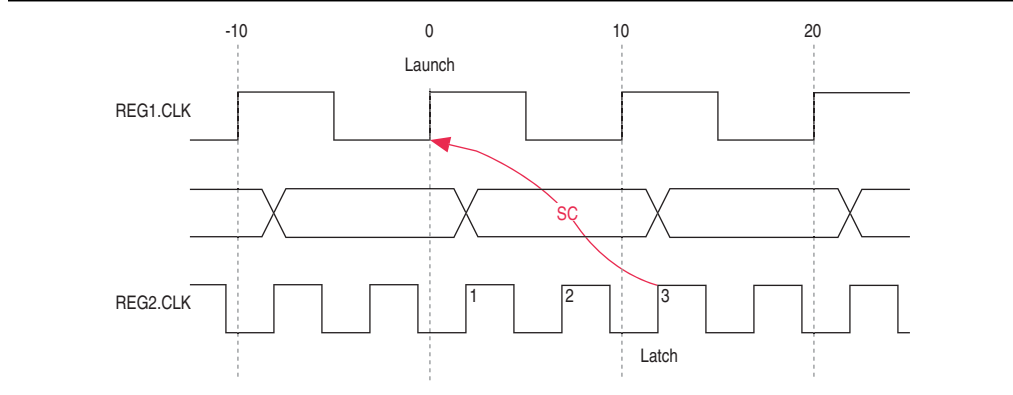
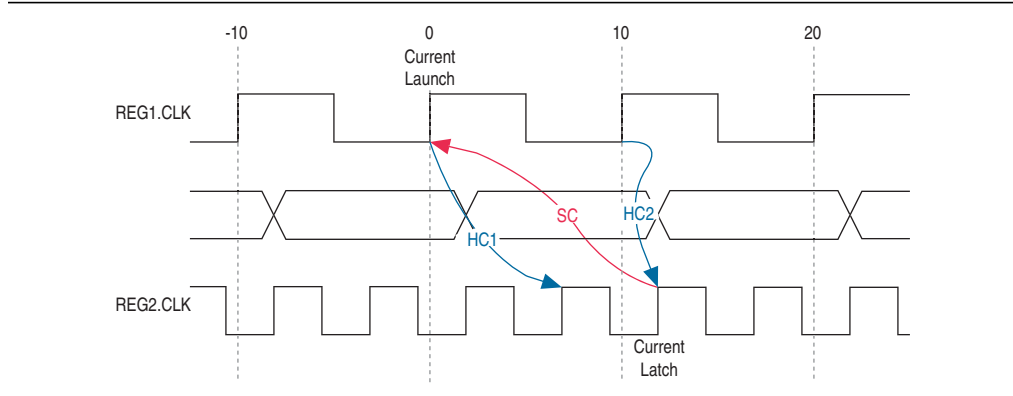


Figure 7-44 shows the timing diagram for default hold check analysis performed by the TimeQuest analyzer with an end multicycle setup value of three.

**Figure 7-44. Default Hold Check**



Equation 7-15 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

---

**Equation 7-15. Hold Check**

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

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


---

In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and should check against the data captured by the previous latch edge at 2 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

### The 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 is used to generate both clocks and different multiplication and division factors are used. Figure 7-45 shows an example of a design where the source clock frequency is a multiple of the destination clock frequency.

**Figure 7-45. Source Clock Frequency is Multiple of Destination Clock Frequency**

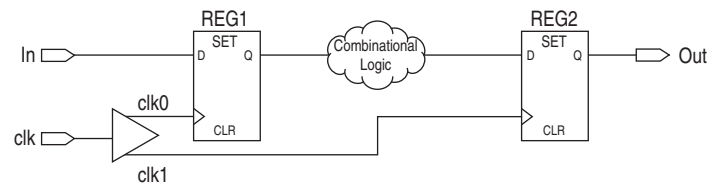
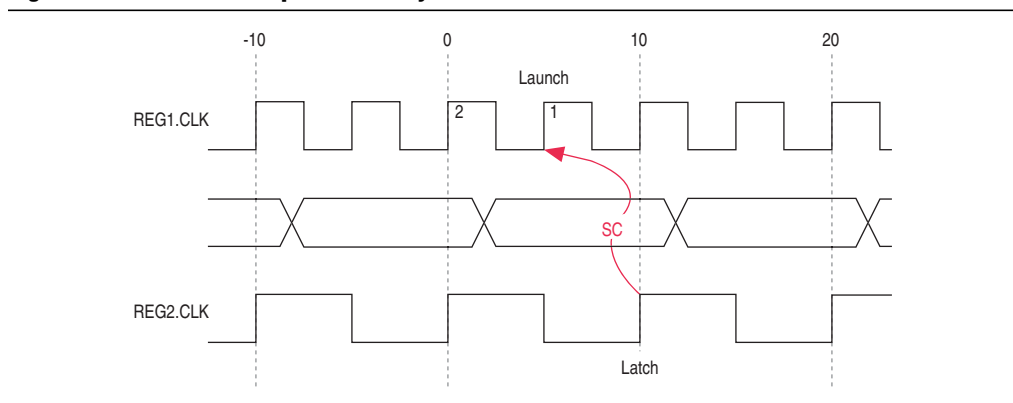


Figure 7-46 shows the timing diagram for default setup check analysis performed by the TimeQuest analyzer.

**Figure 7-46. Default Setup Check Analysis**



Equation 7-16 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

**Equation 7-16. Setup Check**

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

The setup relationship shown in Figure 7-46 demonstrates that the data launched at edge one does not need to be captured, and the data launched at edge two must be captured; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the launch edge by one clock period with a start multicyle setup exception of two.

Example 7-32 shows the multicyle exception used to correct the default analysis in this example.

**Example 7-32. Multicycle Exceptions**

```

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

Figure 7-47 shows the timing diagram for the preferred setup relationship for this example.

**Figure 7-47. Preferred Setup Check Analysis**

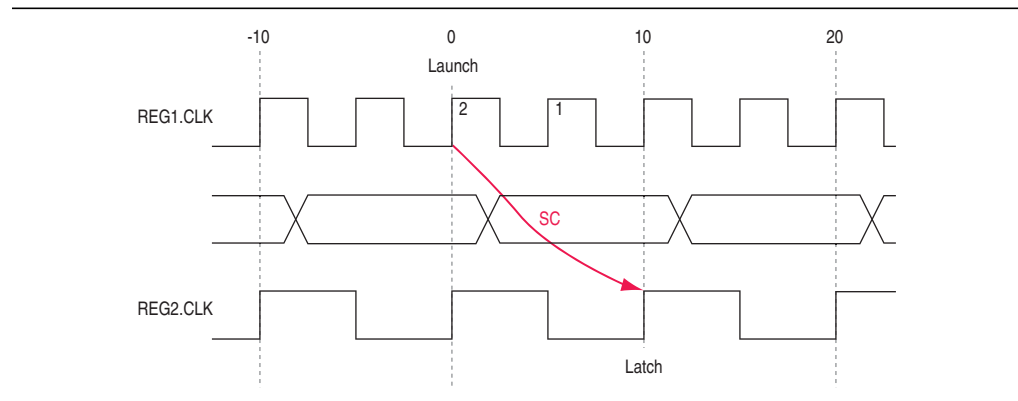
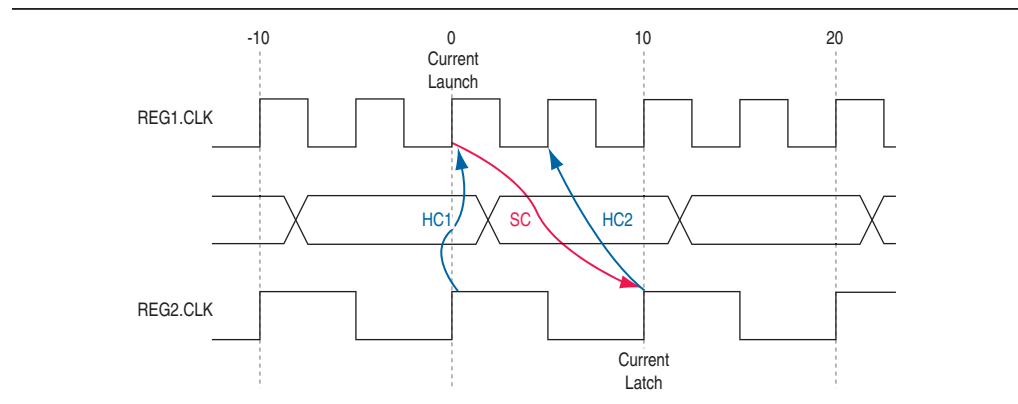


Figure 7-48 shows the timing diagram for default hold check analysis performed by the TimeQuest analyzer with a start multicycle setup value of two.

**Figure 7-48. Default Hold Check**



Equation 7-17 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

**Equation 7-17. Hold Check**

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

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

In this example, hold check two is too restrictive. The data is launched next by the edge at 10 ns and should 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 must use a start multicycle hold exception of one.

## The 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 is used to generate both clocks, different multiplication and division factors are used, and a phase shift applied to the destination clock. Figure 7-49 shows an example of a design where the source clock frequency is a multiple of the destination clock frequency with an offset.

**Figure 7-49. Source Clock Frequency is Multiple of Destination Clock Frequency with Offset**

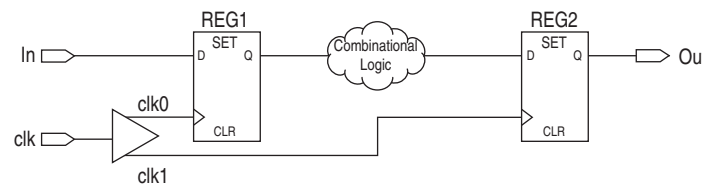
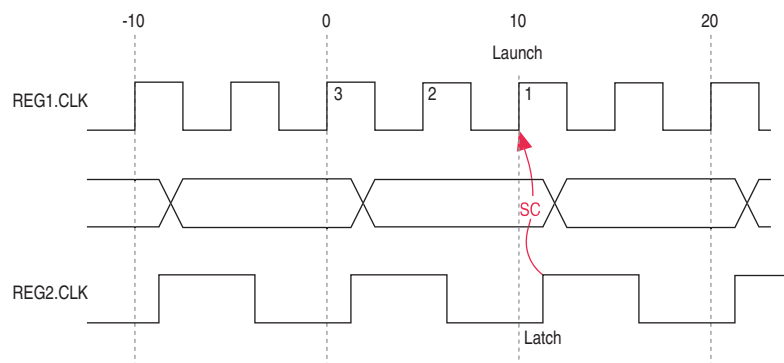


Figure 7-50 shows the timing diagram for default setup check analysis performed by the TimeQuest analyzer.

**Figure 7-50. Setup Timing Diagram**



Equation 7-18 shows the calculation that the TimeQuest analyzer performs to determine the setup check.

### Equation 7-18. setup Check

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

The setup relationship shown in Figure 7-50 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 must shift the launch edge by two clock periods with a start multicycle setup exception of three.

Example 7-33 shows the multicycle exception used to correct the default analysis in this example.

### Example 7-33. Multicycle Exceptions

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

Figure 7-51 shows the timing diagram for the preferred setup relationship for this example.

Figure 7-51. Preferred Setup Check Analysis

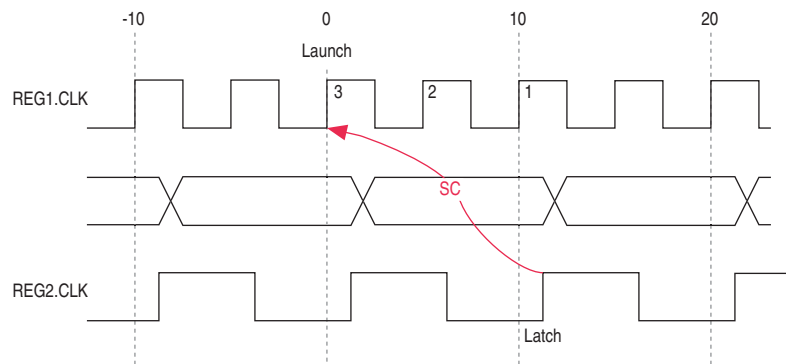
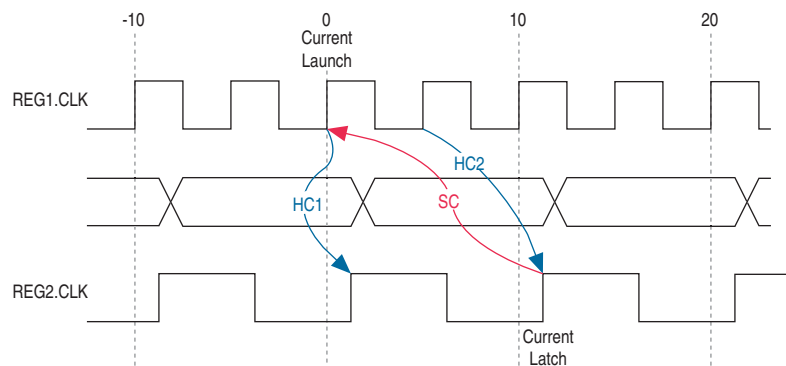


Figure 7-52 shows the timing diagram for default hold check analysis performed by the TimeQuest analyzer with a start multicycle setup value of three.

Figure 7-52. Default Hold Check Analysis



Equation 7-19 shows the calculation that the TimeQuest analyzer performs to determine the hold check.

### Equation 7-19. Hold Check

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

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



**Equation 7-19. Hold Check**

$$= 5 \text{ ns} - 12 \text{ ns}$$

$$= -7 \text{ ns}$$

In this example, hold check two is too restrictive. The data is launched next by the edge at 10 ns and should 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 use a start multicycle hold exception of one.

## Timing Reports

The TimeQuest analyzer provides real-time static timing analysis result reports. The TimeQuest analyzer does not automatically generate reports; you must create each report individually in the TimeQuest analyzer GUI or with command-line commands. You can customize in which report to display specific timing information, excluding fields that are not required.

Table 7-5 shows some of the different command-line commands you can use to generate reports in the TimeQuest analyzer and the equivalent reports shown in the TimeQuest analyzer GUI.

**Table 7-5. TimeQuest Analyzer Reports**

Command-Line Command	Report
report_timing	Timing report
report_exceptions	Exceptions report
report_clock_transfers	Clock Transfers report
report_min_pulse_width	Minimum Pulse Width report
report_ucp	Unconstrained Paths report

- For more information—including a complete list of commands to generate timing reports and full syntax information, options, and example usage—refer to *quartus::sta* in Quartus II Help.

During compilation, the Quartus II software generates timing reports on different timing areas in the design. You can configure various options for the TimeQuest analyzer reports generated during compilation.

- For more information about the options you can set to customize the reports, refer to *TimeQuest Timing Analyzer Page* in Quartus II Help.


You can also use the `TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS` assignment to generate a report of the worst-case timing paths for each clock domain. This report contains worst-case timing data for setup, hold, recovery, removal, and minimum pulse width checks.

Use the `TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS` assignment to specify the number of paths to report for each clock domain.

Example 7-34 shows an example of how to use the `TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS` and `TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS` assignments in the `.qsf` to generate reports.

#### Example 7-34. Generating Worst-Case Timing Reports

```
#Enable Worst-Case Timing Report
set_global_assignment -name TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS ON
#Report 10 paths per clock domain
set_global_assignment -name TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS 10
```

 For more information about timing closure recommendations, refer to the *Area and Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*.

## Document Revision History


Table 7-6 shows the revision history for this chapter.

**Table 7-6. Document Revision History (Part 1 of 2)**

Date	Version	Changes
November 2013	13.1.0	<ul style="list-style-type: none"> <li>Removed HardCopy device information.</li> </ul>
June 2012	12.0.0	<ul style="list-style-type: none"> <li>Reorganized chapter.</li> <li>Added “Using the Quartus II Templates” section on creating an SDC constraints file with the <b>Insert Template</b> dialog box.</li> <li>Added “Identifying the Quartus II Software Executable from the SDC File” section.</li> <li>Revised multicycle exceptions section.</li> </ul>
November 2011	11.1.0	<ul style="list-style-type: none"> <li>Consolidated content from the Best Practices for the Quartus II TimeQuest Timing Analyzer chapter.</li> <li>Changed to new document template.</li> </ul>
May 2011	11.0.0	<ul style="list-style-type: none"> <li>Updated to improve flow. Minor editorial updates.</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>Changed to new document template.</li> <li>Revised and reorganized entire chapter.</li> <li>Linked to Quartus II Help.</li> </ul>
July 2010	10.0.0	Updated to link to content on SDC commands and the TimeQuest analyzer GUI in Quartus II Help.

**Table 7-6. Document Revision History (Part 2 of 2)**

Date	Version	Changes
November 2009	9.1.0	Updated for the Quartus II software version 9.1, including: <ul style="list-style-type: none"> <li>■ Added information about commands for adding and removing items from collections</li> <li>■ Added information about the <code>set_timing_derate</code> and <code>report_skew</code> commands</li> <li>■ Added information about worst-case timing reporting</li> <li>■ Minor editorial updates</li> </ul>
November 2008	8.1.0	Updated for the Quartus II software version 8.1, including: <ul style="list-style-type: none"> <li>■ Added the following sections:                             <ul style="list-style-type: none"> <li>■ “set_net_delay” on page 7-42</li> <li>■ “Annotated Delay” on page 7-49</li> <li>■ “report_net_delay” on page 7-66</li> </ul> </li> <li>■ Updated the descriptions of the <code>-append</code> and <code>-file &lt;name&gt;</code> options in tables throughout the chapter</li> <li>■ Updated entire chapter using 8½” × 11” chapter template</li> <li>■ Minor editorial updates</li> </ul>

 For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).

