



Quartus II TimeQuest Timing Analyzer Cookbook



101 Innovation Drive
San Jose, CA 95134
www.altera.com

MNL-01035-1.3

Software Version: 10.1
Document Version: 1.3
Document Date: January 2011

Copyright © 2011 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Chapter 1. Quartus II TimeQuest Timing Analyzer Cookbook

Clocks and Generated Clocks	1-1
Basic Non-50/50 Duty Cycle Clock	1-1
Offset Clocks	1-2
Basic Clock Divider Using -divide_by	1-2
Toggle Register Generated Clock	1-4
PLL Clocks	1-5
Method 1 – Create Base Clocks and PLL Output Clocks Automatically	1-5
Method 2 – Create Base Clocks Manually and PLL Output Clocks Automatically	1-6
Method 3 – Create Base Clocks and PLL Output Clocks Manually	1-6
Multi-Frequency Analysis	1-7
Clock Multiplexing	1-7
Externally Switched Clock	1-7
PLL Clock Switchover	1-8
I/O Constraints	1-9
Input and Output Delays with Virtual Clocks	1-9
Tri-State Outputs	1-13
System Synchronous Input	1-14
System Synchronous Output	1-15
I/O Timing Requirements t_{SU} , t_{H} , and t_{CO}	1-17
Exceptions	1-18
Multicycle Exceptions	1-18
False Paths	1-20
Miscellaneous	1-21
JTAG Signals	1-21
Input and Output Delays with Multiple Clocks	1-22
Clock Enable Multicycle	1-26

About this User Guide	About-i
Revision History	About-i
How to Contact Altera	About-i
Typographic Conventions	About-i

This manual contains a collection of design scenarios, constraint guidelines, and recommendations. You should be familiar with the TimeQuest Timing Analyzer and the basics of Synopsys Design Constraints (SDC) to properly apply these guidelines.



For information about the TimeQuest analyzer and SDC, refer to www.altera.com/timequest.

Clocks and Generated Clocks

This section shows various clock structures and how to constrain the following structures:

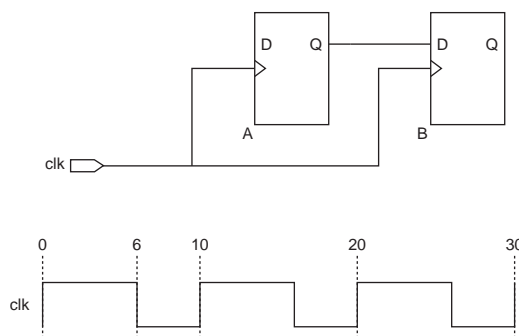
- “Basic Non-50/50 Duty Cycle Clock”
- “Offset Clocks” on page 1-2
- “Basic Clock Divider Using -divide_by” on page 1-2
- “Toggle Register Generated Clock” on page 1-4
- “Multi-Frequency Analysis” on page 1-7

Basic Non-50/50 Duty Cycle Clock

The constraint described in this section is `create_clock`.

The duty cycle of a clock can vary from design to design. By default, the duty cycle for clocks created in the TimeQuest analyzer is set to 50/50. However, you can change the duty cycle of a clock with the `-waveform` option. **Figure 1-1** shows a simple register-to-register path that is clocked by a 60/40 duty cycle clock.

Figure 1-1. Simple Register-to-Register Path



Example 1-1 shows the constraint for a 60/40 duty cycle clock.

Example 1-1. Non-50/50 Duty Cycle Clock Constraints

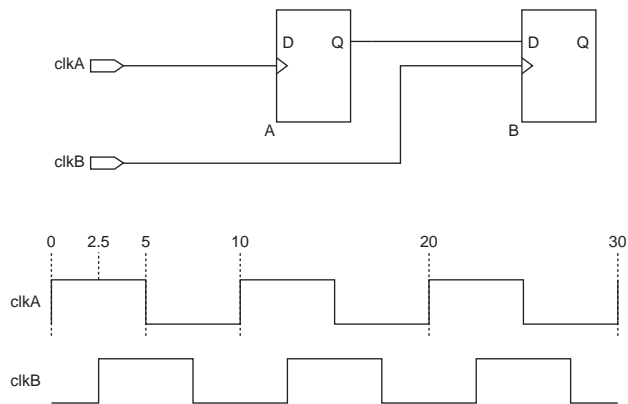
```
#60/40 duty cycle clock
create_clock \
  -period 10.000 \
  -waveform {0.000 6.000} \
  -name clk6040 [get_ports {clk}]
```

Offset Clocks

The constraint described in this section is `create_clock`.

When you constrain clocks in the TimeQuest analyzer, the first rising or falling edge of a clock occurs at an absolute 0. You can create an offset for the clock with the `-waveform` option. Figure 1-2 shows a simple register-to-register path clocked by `clkB`, which is shifted 2.5 ns.

Figure 1-2. Simple Register-to-Register Path Clocked by `clkB`



Example 1-2 shows the constraint for an offset clock.

Example 1-2. Offset Clock Constraints

```
# -waveform defaults to 50/50 duty cycle
create_clock -period 10.000 \
  -name clkA \
  [get_ports {clkA}]

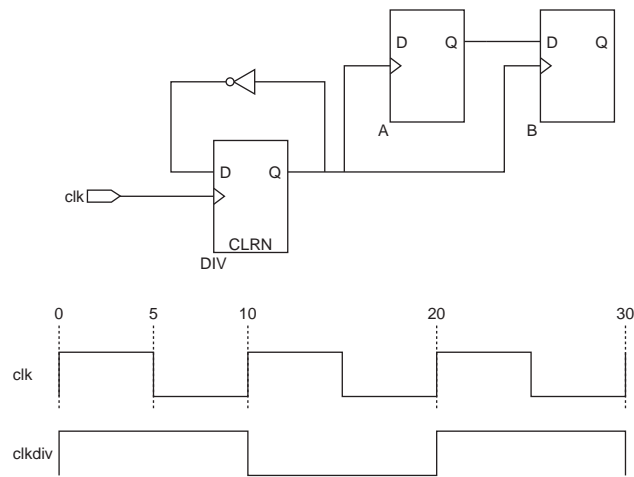
#create a clock with a 2.5 ns offset
create_clock -period 10.000 \
  -waveform {2.500 7.500} \
  -name clkB [get_ports {clkB}]
```

Basic Clock Divider Using `-divide_by`

The constraints described in this section are `create_clock` and `create_generated_clock`.

You can derive clocks in a design from a clock source where the derived clock is slower than the source clock. When constraining a slower clock derived from a clock source, use the `-divide_by` option. Figure 1-3 shows a divide-by-two derived clock.

Figure 1-3. Divide-by-Two Derived Clock



Example 1-3 shows the constraints for a divide-by with `-waveform` clock.

Example 1-3. Divide-by with `-waveform` Clock Constraints

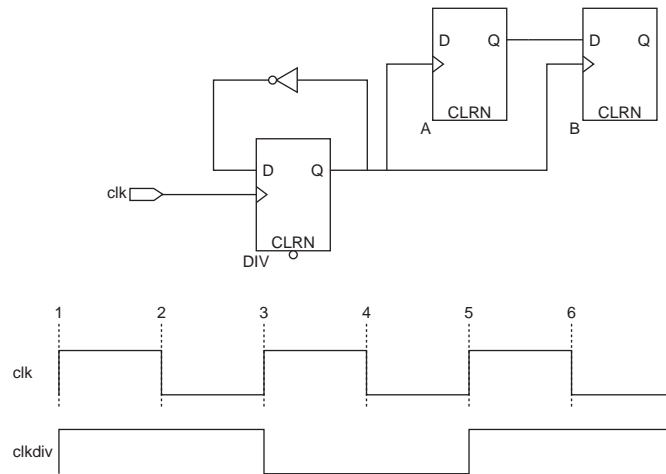
```
create_clock -period 10.000 -name clk [get_ports {clk}]

# Using -divide_by option
create_generated_clock \
  -divide_by 2 \
  -source [get_ports {clk}] \
  -name clkdiv \
  [get_pins {DIV|q}]

# Alternatively use pins to constrain the divider without
# knowing about the master clock
create_generated_clock \
  -divide_by 2 \
  -source [get_pins {DIV|clk}] \
  -name clkdiv \
  [get_pins {DIV|q}]

# the second option works since the
# clock pin of the register DIV is
# connected to the same net fed by the
# clock port clk.
```

You can also create a divide-by clock with the `-edges` option. Figure 1-4 shows a divide-by-two clock with the `-edges` option.

Figure 1-4. Divide-by- Two Clock with the -Edge Option

Example 1-4 shows constraints for the divide-by with `-waveform` clock.

Example 1-4. Divide-by with `-waveform` Clock Constraints

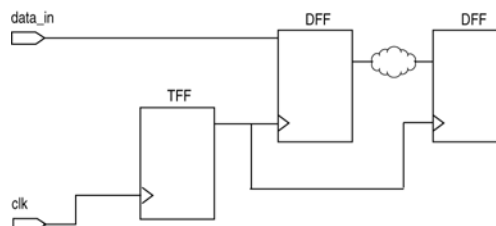
```
# Edge numbers are based on the master clock
create_generated_clock \
  -edges {1 3 5} \
  -source [get_pins {DIV|clk}] \
  -name clkdiv \
  [get_pins {DIV|q}]
```

Toggle Register Generated Clock

Use a toggle register to create a divide-by-two clock. If the data feeding the toggle register is held at a logical “1” value and fed by a 10 ns period clock, the output of the register is a clock with a period of 20 ns.

The constraints for the toggle register clock is very similar to the previous example.

Figure 1-5 shows a toggle register generating a divide-by-two clock.

Figure 1-5. Toggle Register Generating a Divide-by-Two Clock

Example 1-5 shows the constraints for the toggle register.

Example 1-5. Toggle Register Constraints

```
# Create a base clock
create_clock \
  -period 10.000 \
  -name clk \
  [get_ports {clk}]

# Create the generated clock on the output
# of the toggle register.
create_generated_clock \
  -name tff_clk \
  -source [get_ports {clk}] \
  -divide_by 2 \
  [get_pins {tff|q}]
```

PLL Clocks

The constraints described in this section are `derive_pll_clocks`, `create_clock`, and `create_generated_clock`.

Phase-locked loops (PLLs) are used to perform clock synthesis in Altera FPGAs. All output clocks must be constrained for the proper analysis. There are three methods to constrain a PLL:

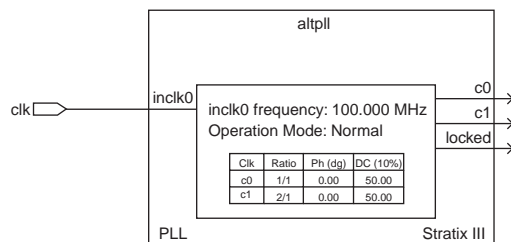
- Create base clocks and PLL output clocks automatically
- Create base clocks manually and PLL output clocks automatically
- Create base clocks manually and PLL output clocks manually

This section shows the advantages for each method.

PLL circuits in Altera FPGAs are incorporated into a design using the `ALTPLL` megafunction.

Figure 1-6 shows an example of the `ALTPLL` megafunction.

Figure 1-6. ALTPLL Megafunction



Method 1 – Create Base Clocks and PLL Output Clocks Automatically

This method allows you to automatically constrain both the input and output clocks of the PLL. All PLL parameters specified in the `ALTPLL` megafunction are used to constrain the input and output clocks of the PLL. A modification to the `ALTPLL` megafunction is automatically updated. You do not have to track changes to the PLL parameters or specify the correct value when creating the PLL’s input and output clocks.

To automatically constrain all inputs and outputs, use the `derive_pll_clocks` command with the `-create_base_clocks` option. The TimeQuest analyzer determines the correct settings based on the MegaWizard™ Plug-In Manager instantiation of the PLL. [Example 1-6](#) shows this command.

Example 1-6. Constraining PLL Base Clocks Automatically

```
derive_pll_clocks -create_base_clocks
```

Method 2 – Create Base Clocks Manually and PLL Output Clocks Automatically

With this method, you can manually constrain the input clock of the PLL and allow the TimeQuest analyzer to automatically constrain the output clocks of the PLL. In addition, you can specify a different input clock frequency instead of the input clock frequency specified in the ALTPLL megafunction. The PLL output clocks are automatically created with the parameters specified in the ALTPLL megafunction. You can try different input clock frequencies, while keeping the same PLL output clock parameters.



Ensure that any input clock frequency specified is compatible with the currently configured PLL.

You can use this method with the `derive_pll_clocks` command and manually create the input clock for the PLL. [Example 1-7](#) shows this command.

Example 1-7. Constraining PLL Base Clocks Manually

```
create_clock -period 10.000 -name clk [get_ports {clk}]
derive_pll_clocks
```

Method 3 – Create Base Clocks and PLL Output Clocks Manually

With this method, you can manually constrain both the input clock and output clocks of the PLL. All PLL parameters are specified and parameter values can differ from those specified in the ALTPLL megafunction. In addition, you can experiment with various PLL input and output frequencies and parameters.

You can use this method with a combination of the `create_clock` and `create_generate_clock` commands. [Example 1-8](#) shows these commands.

Example 1-8. Constraining PLL Output and Base Clocks Manually

```
create_clock -period 10.000 -name clk [get_ports {clk}]

create_generated_clock \
  -name PLL_C0 \
  -source [get_pins {PLL|altpll_component|pll|inclclk[0]}] \
  [get_pins {PLL|altpll_component|pll|clk[0]}]

create_generated_clock \
  -name PLL_C1 \
  -multiply_by 2 \
  -source [get_pins {PLL|altpll_component|pll|inclclk[0]}] \
  [get_pins {PLL|altpll_component|pll|clk[1]}]
```

Multi-Frequency Analysis

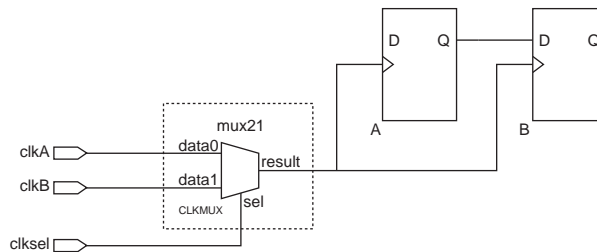
Some designs require multiple clocks driving into the FPGA, where one clock might be faster or slower than the other.

Clock Multiplexing

The constraints described in this section are `create_clock` and `set_clock_groups`.

With clock multiplexing, you can select from two or more clocks. Figure 1-7 shows constraints for a typical 2:1 clock Multiplexer.

Figure 1-7. Constraints for a Typical 2:1 Clock Multiplexer



Example 1-9 shows the constraints for a clock multiplexer.

Example 1-9. Clock Multiplexer Constraints

```
#Create the first input clock clkA to the mux
create_clock -period 10.000 -name clkA [get_ports {clkA}]

#Create the second input clock clkB to the mux
create_clock -period 20.000 -name clkB [get_ports {clkB}]

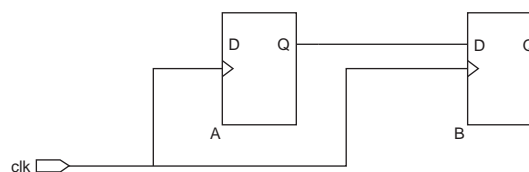
#Cut paths between clkA and clkB
set_clock_groups -exclusive -group {clkA} -group {clkB}
```

Externally Switched Clock

The constraints described in this section are `create_clock` and `set_clock_groups`.

Through an external multiplexer or jumper setting, digital systems are capable of providing different clock frequencies to the same clock port. The TimeQuest analyzer can model this behavior with the `create_clock` constraint and the `-add` option. Figure 1-8 shows a simple register-to-register path where you can drive the clock port clock with a 100-MHz clock or a 50-MHz clock.

Figure 1-8. Simple Register-to-Register Design



Example 1-10 shows the constraints for an externally switched clock.

Example 1-10. Externally Switched Clock Constraints

```
# The clk port can be driven at 100MHz (10ns) or
# 50MHz (20ns)

# clkA is 10ns
create_clock \
  -period 10.000 \
  -name clkA \
  [get_ports {clk}]

# clkB is 20ns assigned to the same port
# Requires -add option
create_clock \
  -period 20.000 \
  -name clkB \
  [get_ports {clk}] \
  -add

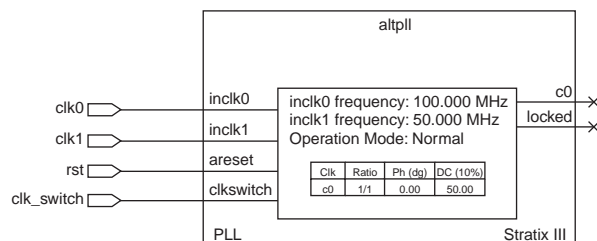
set_clock_groups \
  -exclusive \
  -group {clkA} \
  -group {clkB}
```

PLL Clock Switchover

The constraint described in this section is `derive_pll_clocks`.

The PLL can select between two possible input clocks with the PLL clock-switchover feature in Altera FPGAs (Figure 1-9).

Figure 1-9. PLL Clock-Switchover



[Example 1-11](#) shows the constraints for PLL clock switchover.

Example 1-11. PLL Clock Switchover Constraints

```
#create a 10ns clock for clock port clk0
create_clock \
  -period 10.000 \
  -name clk0 \
  [get_ports {clk0}]

#create a 20ns clock for clock port clk1
create_clock \
  -period 20.000 \
  -name clk1 \
  [get_ports {clk1}]

#automatically create clocks for the PLL output clocks
#derive_pll_clocks automatically makes the proper
#clock assignments for clock-switchover
derive_pll_clocks
```

I/O Constraints

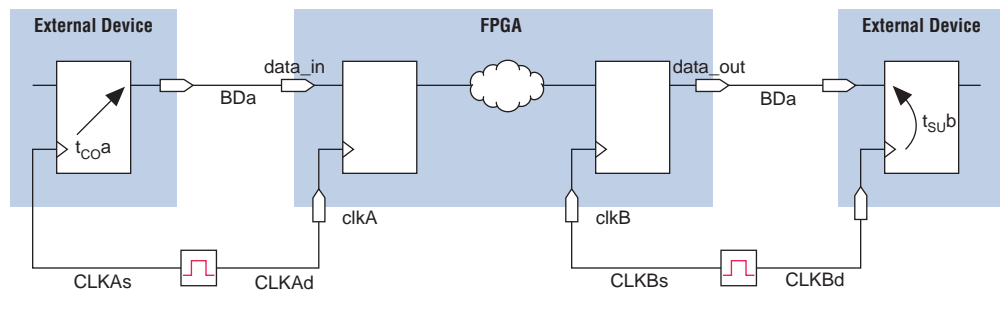
This section contains the following topics:

- [“Input and Output Delays with Virtual Clocks”](#) on page 1-9
- [“Tri-State Outputs”](#) on page 1-13
- [“System Synchronous Input”](#) on page 1-14

Input and Output Delays with Virtual Clocks

All input and output delays should reference a virtual clock so that the TimeQuest analyzer can derive and apply the correct clock uncertainty values when the `derive_clock_uncertainty` command is used. If the input and output delays reference base clocks or PLL clocks rather than virtual clocks, the intra- and inter-clock transfer clock uncertainties, determined by `derive_clock_uncertainty`, are incorrectly applied to the I/O ports. Also, with virtual clocks, additional external clock uncertainties can be applied independent of the clock uncertainties determined by `derive_clock_uncertainty`.

The properties of the virtual clock should be identical to the original clock used to clock either the input (input delay) or output (output delay) ports. [Figure 1-10](#) shows a simple chip-to-chip design where virtual clocks are used for the input and output ports.

Figure 1-10. Chip-to-Chip Design

To constrain the input and output delays that reference virtual clocks shown in [Figure 1-10](#), use the constraints shown in [Example 1-12](#).

Example 1-12. Input and Output Delays Referencing a Virtual Clock (Part 1 of 2)

```
#specify the maximum external clock delay from the external
#device
set CLKAs_max 0.200
#specify the minimum external clock delay from the external
#device
set CLKAs_min 0.100
#specify the maximum external clock delay to the FPGA
set CLKAd_max 0.200
#specify the minimum external clock delay to the FPGA
set CLKAd_min 0.100
#specify the maximum clock-to-out of the external device
set tCOa_max 0.525
#specify the minimum clock-to-out of the external device
set tCOa_min 0.415
#specify the maximum board delay
set BDa_max 0.180
#specify the minimum board delay
set BDa_min 0.120

#create the input maximum delay for the data input to the
#FPGA that
accounts for all delays specified
set_input_delay -clock clk \
-max [expr $CLKAs_max + $tCOa_max + $BDa_max - $CLKAd_min] \
[get_ports {data_in[*]}]

#create the input minimum delay for the data input to the #FPGA that
accounts for all delays specified
set_input_delay -clock clk \
-min [expr $CLKAs_min + $tCOa_min + $BDa_min - $CLKAd_max] \
[get_ports {data_in[*]}]

#create the input clock
create_clock -name clkA -period 10 [get_ports clkA]
#create the associated virtual input clock
create_clock -name clkA_virt -period 10
#specify any uncertainty from the external clock to the virtual clock
set_clock_uncertainty -from { clkA_virt } -setup 0.25

#create the output clock
create_clock -name clkB -period 5 [get_ports clkB]
#create the associated virtual input clock
create_clock -name clkB_virt -period 5
#specify any uncertainty from the external clock to the virtual clock
set_clock_uncertainty -from { clkB_virt } -setup 0.25

#determine internal clock uncertainties
derive_clock_uncertainty
#create the input delay referencing the virtual clock
#specify the maximum external clock delay from the external
#device
set CLKAs_max 0.200
#specify the minimum external clock delay from the external
#device
set CLKAs_min 0.100
#specify the maximum external clock delay to the FPGA
set CLKAd_max 0.200
#specify the minimum external clock delay to the FPGA
set CLKAd_min 0.100
```

Example 1-12. Input and Output Delays Referencing a Virtual Clock (Part 2 of 2)

```

#specify the maximum clock-to-out of the external device
set tCOa_max 0.525
#specify the minimum clock-to-out of the external device
set tCOa_min 0.415
#specify the maximum board delay
set BDa_max 0.180
#specify the minimum board delay
set BDa_min 0.120

#create the input maximum delay for the data input to the
#FPGA that accounts for all delays specified
set_input_delay -clock clkA_virt \
-max [expr $CLKAs_max + $tCOa_max + $BDa_max - $CLKAd_min] \
[get_ports {data_in[*]}]

#create the input minimum delay for the data input to the
#FPGA that accounts for all delays specified
set_input_delay -clock clkA_virt \
-min [expr $CLKAs_min + $tCOa_min + $BDa_min - $CLKAd_max] \
[get_ports {data_in[*]}]

#creating the output delay referencing the virtual clock
#specify the maximum external clock delay from the external
#device
set CLKBs_max 0.100
#specify the minimum external clock delay from the external
#device
set CLKBs_min 0.050
#specify the maximum external clock delay to the FPGA
set CLKBd_max 0.100
#specify the minimum external clock delay to the FPGA
set CLKBd_min 0.050
#specify the maximum clock-to-out of the external device
set tSUB_max 0.500
#specify the hold time of the external device
set tHb 0.400
#specify the maximum board delay
set BDb_max 0.100
#specify the minimum board delay
set BDb_min 0.080

#create the output maximum delay for the data output from the
#FPGA that accounts for all delays specified
set_output_delay -clock clkB_virt \
-max [expr $CLKBs_max + $tSUB_max + $BDb_max - $CLKBd_min] \
[get_ports {data_out}]

#create the output minimum delay for the data output from the
#FPGA that accounts for all delays specified
set_output_delay -clock clkB_virt \
-min [expr $CLKBs_min - $tHb + $BDb_min - $CLKBd_max] \
[get_ports {data_out}]

```

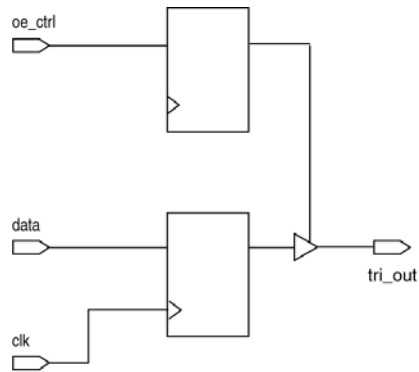
Tri-State Outputs

Tri-state outputs allow either a valid data signal or a high impedance signal to be driven out of an input port. The timing of either signal is important in the overall system timing of the design.

The timing constraints for tri-state outputs are identical to regular output ports.

Figure 1-11 shows a typical output fed by a tri-state buffer.

Figure 1-11. Typical Output Fed by a Tri-State Buffer



Example 1-13 shows the constraints for the tri-state output port.

Example 1-13. Tri-State Output Port Constraints

```
# Base clock
create_clock [get_ports {clk}] \
  -name {clk} \
  -period 10.0 \
  -waveform {0.0 5.0}

# Virtual clock for the output port
create_clock \
  -name {clk_virt} \
  -period 10.0 \
  -waveform {0.0 5.0}

# Output constraints
set_output_delay 2.0 \
  -max \
  -clock [get_clocks {clk_virt}] \
  [get_ports {tri_out}]

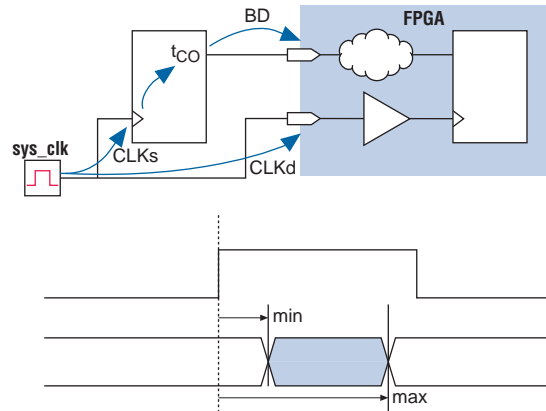
set_output_delay 1.0 \
  -min \
  -clock [get_clocks {clk_virt}] \
  [get_ports {tri_out}]
```

System Synchronous Input

The constraints described in this section are `create_clock` and `set_input_delay`.

Figure 1-12 shows a typical chip-to-chip input interface and the various parameters necessary to specify an input delay for the interface.

Figure 1-12. Simple Chip-to-Chip Input Interface



Example 1-14 shows the constraints for a system synchronous input.

Example 1-14. System Synchronous Input Constraints

```
#specify the maximum external clock delay from the external device
set CLKs_max 0.200

#specify the minimum external clock delay from the external device
set CLKs_min 0.100

#specify the maximum external clock delay to the FPGA
set CLKd_max 0.200

#specify the minimum external clock delay to the FPGA
set CLKd_min 0.100

#specify the maximum clock-to-out of the external device
set tCO_max 0.525

#specify the minimum clock-to-out of the external device
set tCO_min 0.415

#specify the maximum board delay
set BD_max 0.180

#specify the minimum board delay
set BD_min 0.120

#create a clock 10ns
create_clock -period 10 -name sys_clk [get_ports sys_clk]

#create the associated virtual input clock
create_clock -period 10 -name virt_sys_clk

#create the input maximum delay for the data input to the FPGA that
#accounts for all delays specified
set_input_delay -clock virt_sys_clk \
  -max [expr $CLKs_max + $tCO_max + $BD_max - $CLKd_min] \
  [get_ports {data_in[*]}]

#create the input minimum delay for the data input to the FPGA that
#accounts for all delays specified
set_input_delay -clock virt_sys_clk \
  -min [expr $CLKs_min + $tCO_min + $BD_min - $CLKd_max] \
  [get_ports {data_in[*]}]
```

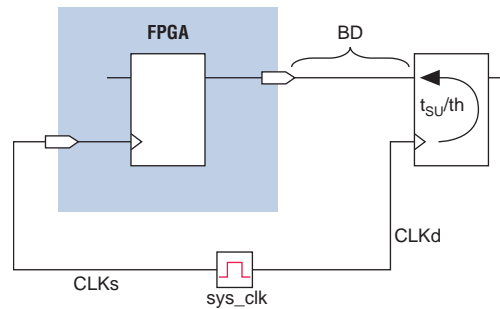


For more information about constraining source synchronous input and output interfaces, refer to *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

System Synchronous Output

The constraints described in this section are `create_clock` and `set_output_delay`.

Figure 1-13 shows a typical chip-to-chip output interface and the various parameters necessary to specify an output delay for the interface.

Figure 1-13. Simple Chip-to-Chip Interface

Example 1-15 shows the constraints for system synchronous output.

Example 1-15. System Synchronous Output Constraints

```
#specify the maximum external clock delay to the FPGA
set CLKs_max 0.200

#specify the minimum external clock delay to the FPGA
set CLKs_min 0.100

#specify the maximum external clock delay to the external device
set CLKd_max 0.200

#specify the minimum external clock delay to the external device
set CLKd_min 0.100

#specify the maximum setup time of the external device
set tSU 0.125

#specify the minimum setup time of the external device
set tH 0.100

#specify the maximum board delay
set BD_max 0.180

#specify the minimum board delay
set BD_min 0.120

#create a clock 10ns
create_clock -period 10 -name sys_clk [get_ports sys_clk]

#create the associated virtual input clock
create_clock -period 10 -name virt_sys_clk

#create the output maximum delay for the data output from the FPGA that
#accounts for all delays specified
set_output_delay -clock virt_sys_clk \
  -max [expr $CLKs_max + $BD_max + $tSU - $CLKd_min] \
  [get_ports {data_in[*]}]

#create the output minimum delay for the data output from the FPGA that
#accounts for all delays specified
set_output_delay -clock virt_sys_clk \
  -min [expr $CLKs_min + $BD_min - $tH - $CLKd_max] \
  [get_ports {data_in[*]}]
```

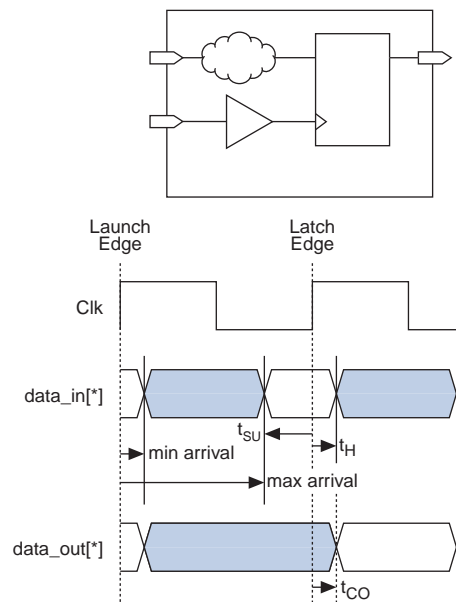
For more information about constraining source synchronous input and output interfaces, refer to *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

I/O Timing Requirements t_{SU} , t_H , and t_{CO}

The constraints described in this section are `set_input_delay` and `set_output_delay`.

Example 1-16 shows how to specify t_{SU} and t_H using `set_input_delay`, and how to specify t_{CO} using `set_output_delay`. **Figure 1-14** shows an FPGA and a timing diagram with the required timing specifications.

Figure 1-14. I/O Timing Specifications



Example 1-16 shows the constraints for t_{SU} , t_H , and t_{CO} .

Example 1-16. t_{SU} , t_H , and t_{CO} Constraints

```
#Specify the clock period
set period 10.000

#Specify the required tSU
set tSU 1.250

#Specify the required tH
set tH 0.750

#Specify the required tCO
set tCO 0.4

#create a clock 10ns
create_clock -period $period -name clk [get_ports sys_clk]

#create the associated virtual input clock
create_clock -period $period -name virt_clk

set_input_delay -clock virt_clk \
  -max [expr $period - $tSU] \
  [get_ports {data_in[*]}]

set_input_delay -clock virt_clk \
  -min $tH \
  [get_ports {data_in[*]}]

set_output_delay -clock virt_clk \
  -max [expr $period - $tCO] \
  [get_ports {data_out[*]}]
```

Exceptions

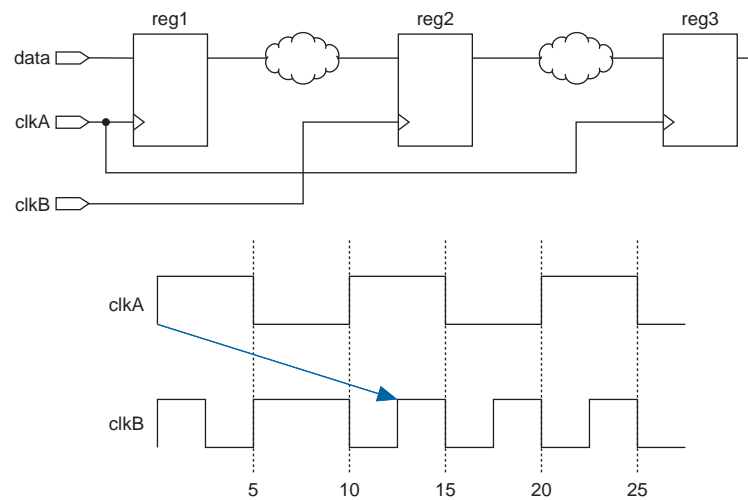
This section contains the following topics:

- “Multicycle Exceptions”
- “False Paths” on page 1-20

Multicycle Exceptions

The constraints described in this section are `create_clock` and `set_multicycle_path`.

By default, the TimeQuest analyzer uses a single-cycle analysis to determine both the setup and hold relationship of any register-to-register path. This results in the most restrictive setup and hold requirements. However, multicycle exceptions can be used to relax the setup or hold relationship of any register-to-register path. Figure 1-15 shows a simple register-to-register path.

Figure 1-15. Register-to-Register Path

Multicycles can be applied to clock-to-clock transfers or to individual registers. Applying multicycles to clock-to-clock transfers affects all the specified setup or hold relationships of the target clocks of register-to-register paths fed by the source and destination clocks. [Example 1-17](#) shows a multicycle constraint.

Example 1-17. Multicycle Clock-to-Clock

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]

set_multicycle_path -from [get_clocks {clkA}] -to [get_clocks {clkB}] -setup -end 2
```

In [Example 1-17](#), the setup relationship is relaxed by an additional destination clock period for any register-to-register path where the source clock is clkA and the destination clock is clkB. This results in registers reg1 and reg2 having a setup relationship of 12.5 ns instead of the default 5 ns. The setup relationship between registers reg2 and reg3 is not affected by the multicycle.

Applying multicycles to individual registers affects only the specified registers setup or hold relationship. [Example 1-18](#) shows the constraints for applying multicycles to individual registers.

Example 1-18. Multicycle Register-to-Register

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]

set_multicycle_path -from [get_pins {reg1|q}] -to [get_pins {reg2|d}] -setup -end 2
```

In [Example 1-18](#), the setup relationship is relaxed by an additional destination clock period for the register-to-register path from register reg1 to register reg2. This results in registers reg1 and reg2 having a setup relationship of 12.5 ns instead of the default 5 ns. The setup relationship between registers reg2 and reg3 is not affected by the multicycle.

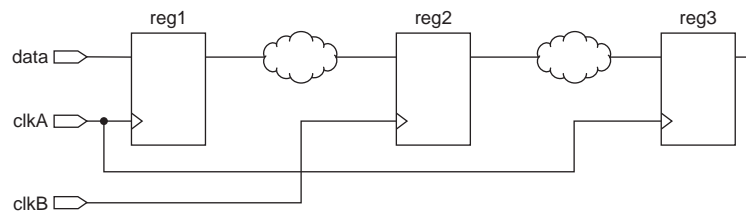
- For more information about the types of multicycle exceptions available in the TimeQuest analyzer, refer to the *Best Practices for the TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

False Paths

The constraints described in this section are `create_clock` and `set_false_path`.

You do not need to analyze timing on all paths. Synchronization of non-critical paths can be removed or cut from timing analysis. When you declare non-critical paths, the Quartus II Fitter can focus on the optimization of critical paths and can reduce overall compilation time. [Figure 1-16](#) shows a simple register-to-register design where you can cut the path from register `reg1` to register `reg2`.

Figure 1-16. Register-to-Register Path



False paths can be applied either to clock-to-clock transfers or to individual registers. Applying false paths to clock-to-clock transfers cuts all paths between the target clocks. [Example 1-19](#) shows the constraints for applying false paths.

Example 1-19. False Path Clock-to-Clock

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]

set_false_path -from [get_clocks {clkA}] -to [get_clocks {clkB}]
```

In [Example 1-19](#), the path is cut and not analyzed by the TimeQuest analyzer for any register-to-register path where the source clock is `clkA` and the destination clock is `clkB`. This does not affect register-to-register paths where the source register is clocked by `clkB` and the destination register is clocked by `clkA`.

- For [Example 1-19](#), the `set_false_path` command cuts paths from clock `clkA` to `clkB`. The command does not cut paths from `clkB` to `clkA`. To cut paths from `clkB` to `clkA`, an additional `set_false_path` command must be applied (for example, `set_false_path -from clkB -to clkA`). Alternatively, you can use `set_clock_groups` to cut paths from `clkA` to `clkB` and from `clkB` to `clkA` with one command.
- For more information about the `set_clock_groups` command, refer to *Set Clock Groups Dialog Box (set_clock_groups)* in Quartus II Help.

Applying false paths to individual registers cuts only the path specified. [Example 1-20](#) shows the constraints for this.

Example 1–20. False Path Register-to-Register

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]

set_false_path -from [get_pins {reg1|q}] -to [get_pins {reg2|d}]
```

In [Example 1–20](#), the register-to-register path from register `reg1` to register `reg2` is cut. All other paths remain unaffected.

Miscellaneous

This section contains the following topics:

- [“JTAG Signals”](#)
- [“Input and Output Delays with Multiple Clocks”](#) on page 1–22
- [“Clock Enable Multicycle”](#) on page 1–26

JTAG Signals

The constraints described in this section are `create_clock`, `set_input_delay`, and `set_output_delay`.

Many in-system debugging tools use the JTAG interface in Altera FPGAs. When you debug your design with the JTAG interface, the JTAG signals TCK, TMS, TDI, and TDO are implemented as part of the design. Because of this, the TimeQuest analyzer flags these signals as unconstrained when an unconstrained path report is generated. [Table 1–1](#) shows the JTAG signals that might appear as unconstrained.

Table 1–1. JTAG Signals

Signal Name	Description
<code>altera_reserved_tck</code>	JTAG test clock input port
<code>altera_reserved_tms</code>	JTAG test mode select input port
<code>altera_reserved_tdi</code>	JTAG test data input line input port
<code>altera_reserved_tdo</code>	JTAG test data output line output port

You can constrain the JTAG signals by applying the SDC commands in [Example 1-21](#).

Example 1-21. JTAG Signal Constraints

```
#JTAG Signal Constraints
#constrain the TCK port
create_clock \
-name tck \
-period "10MHz" \
[get_ports altera_reserved_tck]

#cut all paths to and from tck
set_clock_groups -exclusive -group [get_clocks tck]

#constrain the TDI port
set_input_delay \
-clock tck \
20 \
[get_ports altera_reserved_tdi]

#constrain the TMS port
set_input_delay \
-clock tck \
20 \
[get_ports altera_reserved_tms]

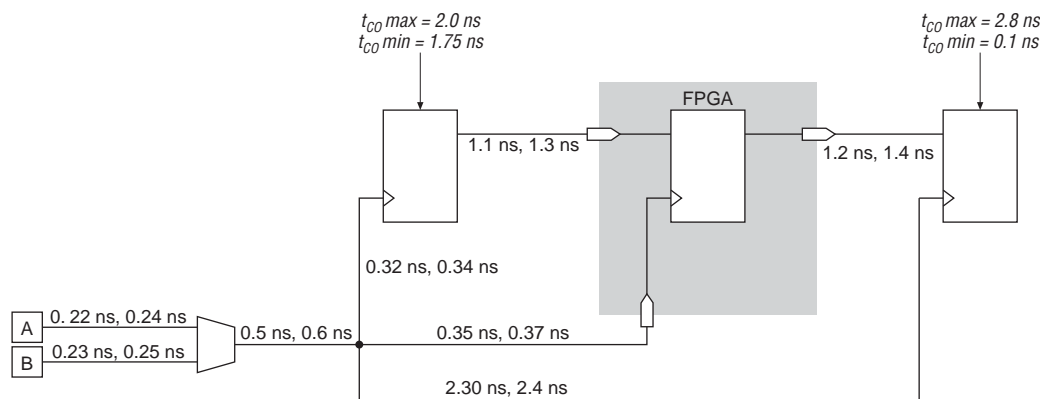
#constrain the TDO port
set_output_delay \
-clock tck \
20 \
[get_ports altera_reserved_tdo]
```

Input and Output Delays with Multiple Clocks

The constraints described in this section are `create_clock`, `create_generated_clock`, `set_clock_groups`, `set_clock_latency`, `set_input_delay`, and `set_output_delay`.

These constraints provide both a primary and secondary clock. The primary clock acts as the main clock and the secondary clock may act as a redundant clock at a slower speed. [Figure 1-17](#) shows an example of this setup.

Figure 1-17. Simple Register-to-Register Design



Example 1-22 shows the command for input delay with multiple clocks.

Example 1-22. Input Delay with Multiple Clocks (Part 1 of 4)

```
#####
# Create all the clocks #
#####
# Create variables for the clock periods.
set PERIOD_CLK_A 10.000
set PERIOD_CLK_B 7.000

# Create the clk_a clock which will represent the clock
# that routes to the FPGA.
create_clock \
  -name {clk_a} \
  -period \
  $PERIOD_CLK_A \
  [get_ports {clk}]

# Create the clk_b clock which will represent the clock
# that routes to the FPGA.
# Note the -add is needed because this is the second clock
# that has the same 'clk' port as a target.
create_clock \
  -name {clk_b} \
  -period $PERIOD_CLK_B \
  [get_ports {clk}] \
  -add

# Create a virtual clock which will represent the clock
# that routes to the external source device when clk_a is
# selected a the external mux.
create_clock \
  -name virtual_source_clk_a \
  -period $PERIOD_CLK_A

# Create a virtual clock which will represent the clock
# that routes to the external source device when clk_b is
# selected a the external mux.
create_clock \
  -name virtual_source_clk_b \
  -period $PERIOD_CLK_B

# Create a virtual clock which will represent the clock
# that routes to the external destination device when clk_a
# is selected a the external mux.
create_clock \
  -name virtual_dest_clk_a \
  -period $PERIOD_CLK_A

# Create a virtual clock which will represent the clock
# that routes to the external destination device when clk_b
# is selected a the external mux.
create_clock \
  -name virtual_dest_clk_b \
  -period $PERIOD_CLK_B
```

Example 1-22. Input Delay with Multiple Clocks (Part 2 of 4)

```
#####
# Cut clock transfers that are not valid #
#####
# Cut this because virtual_source_clk_b can not be clocking
# the external source device at the same time that clk_a is
# clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_a} \
    -group {virtual_source_clk_b}
# Cut this because virtual_source_clk_a can not be clocking
# the external source device at the same time that clk_b is
# clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_b} \
    -group {virtual_source_clk_a}
# Cut this because virtual_dest_clk_b can not be clocking
# the external destination device at the same time that
# clk_a is clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_a} \
    -group {virtual_dest_clk_b}
# Cut this because virtual_dest_clk_a can not be clocking
# the external destination device at the same time that
# clk_b is clocking the FPGA
set_clock_groups -exclusive \
    -group {clk_b} \
    -group {virtual_dest_clk_a}

#####
# Define the latency of all the clocks #
#####
# Since TimeQuest does not know what part of the clock
# latency is common we must simply remove the common part
# from the latency calculation. For example when
# calculating the latency for virtual_source_clk_a we must
# ignore the 220ps,240ps route and the 500ps/600ps mux
# delay if we want to remove the common clock path
# pessimism.
#
# Define fastest and slowest virtual_source_clk_a path to
# the external source device.
set_clock_latency -source \
    -early .320 \
[get_clocks virtual_source_clk_a]
set_clock_latency -source \
    -late .340 \
    [get_clocks virtual_source_clk_a]
# Define fastest and slowest virtual_source_clk_b path to
# the external source device.
set_clock_latency -source \
    -early .320 \
    [get_clocks virtual_source_clk_b]
set_clock_latency -source \
    -late .340 \
    [get_clocks virtual_source_clk_b]
# Define fastest and slowest clk_a path to the FPGA.
set_clock_latency -source \
    -early .350 \
    [get_clocks clk_a]
set_clock_latency -source \
    -late .370 \
    [get_clocks clk_a]
```

Example 1-22. Input Delay with Multiple Clocks (Part 3 of 4)

```
# Define fastest and slowest clk_b path to the FPGA.
set_clock_latency -source \
  -early .350 \
  [get_clocks clk_b]
set_clock_latency -source \
  -late .370 \
  [get_clocks clk_b]
# Define fastest and slowest virtual_dest_clk_a path to
# the external destination device.
set_clock_latency -source \
  -early 2.3 \
  [get_clocks virtual_dest_clk_a]
set_clock_latency -source \
  -late 2.4 \
  [get_clocks virtual_dest_clk_a]
# Define fastest and slowest virtual_dest_clk_b path to
# the external destination device.
set_clock_latency -source \
  -early 2.3 \
  [get_clocks virtual_dest_clk_b]
set_clock_latency -source \
  -late 2.4 \
  [get_clocks virtual_dest_clk_b]

#####
# Constrain the input port 'datain' #
#####
# This Tco is the min/max value of the Tco for the
# external module.
set Tco_max 2.0
set Tco_min 1.75
# Td is the min/max trace delay of datain from the
# external device
set Td_min 1.1
set Td_max 1.3

# Calculate the input delay numbers
set input_max [expr $Td_max + $Tco_max]
set input_min [expr $Td_min + $Tco_min]
# Create the input delay constraints when clk_a is selected
set_input_delay \
  -clock virtual_source_clk_a \
  -max $input_max \
  [get_ports datain]
set_input_delay \
  -clock virtual_source_clk_a \
  -min $input_min \
  [get_ports datain]
# Create the input delay constraints when clk_b is selected
set_input_delay \
  -clock virtual_source_clk_b \
  -max $input_max \
  [get_ports datain] \
  -add_delay
set_input_delay \
  -clock virtual_source_clk_b \
  -min $input_min \
  [get_ports datain] \
  -add_delay
```

Example 1-22. Input Delay with Multiple Clocks (Part 4 of 4)

```
#####
# Constrain the output port 'dataout' #
#####
# This Tsu/Th is the value of the Tsu/Th for the external
# device.
set Tsu 2.8
set Th 0.1
# This is the min/max trace delay of dataout to the
# external device.
set Td_min 1.2
set Td_max 1.4

# Calculate the output delay numbers
set output_max [expr $Td_max + $Tsu]
set output_min [expr $Td_min - $Th]

# Create the output delay constraints when clk_a is
# selected.
set_output_delay \
  -clock virtual_dest_clk_a \
  -max $output_max \
  [get_ports dataout]
set_output_delay \
  -clock virtual_dest_clk_a \
  -min $output_min \
  [get_ports dataout]
# Create the output delay constraints when clk_b is
# selected.
set_output_delay \
  -clock virtual_dest_clk_b \
  -max $output_max \
  [get_ports dataout] \
  -add_delay
set_output_delay \
  -clock virtual_dest_clk_b \
  -min $output_min \
  [get_ports dataout] \
  -add_delay
```

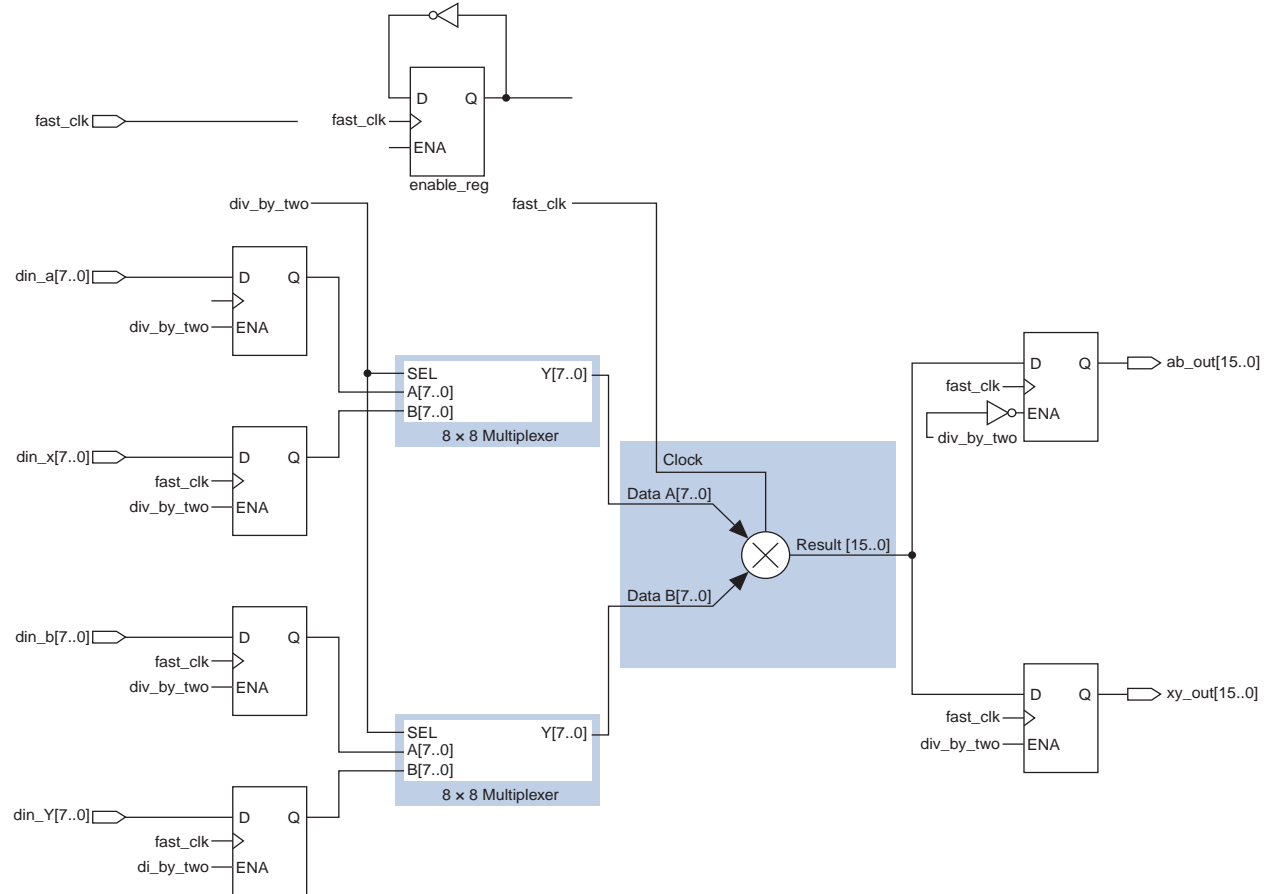
Clock Enable Multicycle

The constraints described in this section are `create_clock`, `set_multicycle_path`, and `get_fanouts`.

You can specify multicycles based on the enabled ports of registers with clock enabled multicycles. For example, [Figure 1-18](#) shows a simple circuit where register `enable_reg` is used to create a registered enabled signal for registers `din_a_reg[7..0]`, `din_b_reg[7..0]`, `din_x_reg[7..0]`, `din_y_reg[7..0]`, `a_times_b`, and `x_times_y`.

The enable register `enable_reg` generates an enable pulse that is two times the clock period of the register; therefore, a multicycle exception must be applied for the correct analysis. You must apply a multicycle setup of 2 and a multicycle hold of 1 to the enable-driven register fed by the register `enable_reg`. The multicycle exception is applied only to register-to-register paths where the destination register is controlled by `enable_reg`. To accomplish this, you can apply the `set_multicycle_path` exception to all enable-driven registers. This can be tedious, because all enable-driven registers must be specified. You can also use the combination of `set_multicycle_path` and `get_fanouts`, as shown in [Example 1-23](#).

Figure 1-18. Clock Enable Multicycle Design



Example 1-23. Clock Enable Multicycle Constraints

```
#Setup multicycle of 2 to enabled driven destination registers
set_multicycle_path 2 -to [get_fanouts [get_pins enable_reg|q*] \
-through [get_pins -hierarchical *|*ena*]] -end -setup
```

```
#Hold multicycle of 1 to enabled driven destination registers
set_multicycle_path 1 -to [get_fanouts [get_pins enable_reg|q*] \
-through [get_pins -hierarchical *|*ena*]] -end -hold
```

The target of the set_multicycle_path exception is limited to all fan-outs of the enable_reg register that feed the enable port of a register. Use the following option:

```
[get_fanouts [get_pins enable_reg|q*] -through [get_pins -hierarchical *|*ena*]]
```

Table 1-2 shows the new setup and hold relationships of all enable-driven register-to-register paths in the design after the multicycle exceptions have been applied.

Table 1-2. Setup and Hold Relationships for Enable-Driven Registers

Source Register	Destination Register	Setup Relationship	Hold Relationship
din_a[*]	din_a_reg[*]	2× (latch edge time)	1× (latch edge time)
din_x[*]	din_x_reg[*]	2× (latch edge time)	1× (latch edge time)
din_b[*]	din_b_reg[*]	2× (latch edge time)	1× (latch edge time)
din_y[*]	din_y_reg[*]	2× (latch edge time)	1× (latch edge time)
fast_mult:mult *	a_times_b[*]	2× (latch edge time)	1× (latch edge time)
fast_mult:mult *	x_times_y[*]	2× (latch edge time)	1× (latch edge time)
enable_reg	din_a_reg[*],din_b_reg[*], a_times_b[*],x_times_y[*]	2× (latch edge time)	1× (latch edge time)

Table 1-2 shows the setup and hold relationships that start at the `enable_reg` register and end at any enable-driven register at 2 and 1, respectively. If these paths do not require the setup and hold relationship to be modified, you can use the following multicyle exceptions to apply the original relationships:

```
set_multicycle_path 1 -from [get_pins enable_reg|q*] -end -setup
set_multicycle_path 0 -from [get_pins enable_reg|q*] -end -hold
```



For more information about multicyle exceptions, refer to the *Best Practices for the TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Revision History

Table 1 displays the revision history for the chapters in this user guide.

Table 1. Document Revision History

Date	Version	Changes
January 2011	1.3	<ul style="list-style-type: none"> ■ Added new sections “Toggle Register Generated Clock” and “Tri-State Outputs”. ■ Minor text edits.
March 2010	1.2	Corrected errors in example script.
March 2010	1.1	Corrected errors in example script.
August 2008	1.0	Initial release.

How to Contact Altera

For the most up-to-date information about Altera products, refer to Table 2.

Table 2. Altera Contact Information

Contact (1)	Contact Method	Address
Technical Support	Website	www.altera.com/mysupport/
Technical training	Website	www.altera.com/training
Product literature	Website	www.altera.com/literature
(Software Licensing)	Email	authorization@altera.com

Note to Table 2:

(1) You can also contact your local Altera sales office or sales representative.








Typographic Conventions

This document uses the typographic conventions shown in Table 3.

Table 3. Typographic Conventions (Part 1 of 2)

Visual Cues	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , iqdesigns directory, d: drive, chiptrip.gdf file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .

Table 3. Typographic Conventions (Part 2 of 2)

Visual Cues	Meaning
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA} , $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pdf file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix \bar{n} , for example, reset \bar{n} . Anything that must be typed exactly as it appears is shown in Courier type. For example: <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also, sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), as well as logic function names (for example, TRI) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets are used in a list of items when the sequence of the items is not important.
	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
	A warning calls attention to a condition or possible situation that can cause injury to the user.
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information about a particular topic.