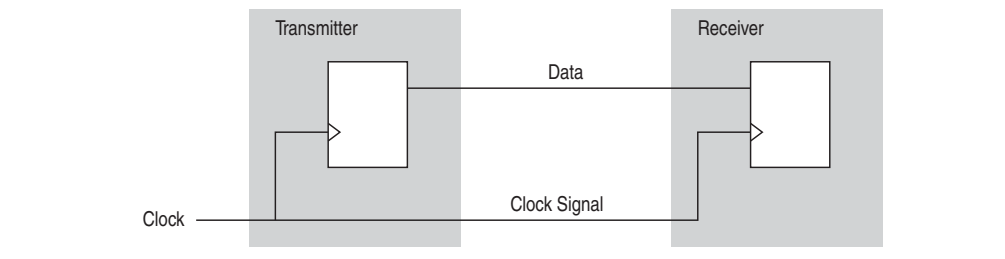


This application note describes techniques for constraining and analyzing source-synchronous interfaces. In source-synchronous interfaces, the source of the clock is the same device as the source of the data, rather than another source, such as a common clock network.

Figure 1 shows a block diagram of a basic source-synchronous interface.

Figure 1. Basic Source-Synchronous Interface



Introduction

Source-synchronous interfaces are used for high-speed data transfer. DDR memory, HyperTransport buses, and the SPI-4.2 standard all use source-synchronous interfaces.

Constraining source-synchronous interfaces can be complex. The Synopsys Design Constraints (SDC) format provides the necessary detail and precision for a proper analysis. Familiarize yourself with the SDC format and the TimeQuest timing analyzer before you read this application note.



For more information, refer to the *SDC and TimeQuest API Reference Manual* and the *Quartus II TimeQuest Timing Analyzer* chapter of the *Quartus II Handbook*.

This application note is divided into two main sections:

- “Source-Synchronous Outputs” on page 6
- “Source-Synchronous Inputs” on page 39

These sections include descriptions of the output and input interfaces and details about the three types of SDC constraints or exceptions that apply to each direction.

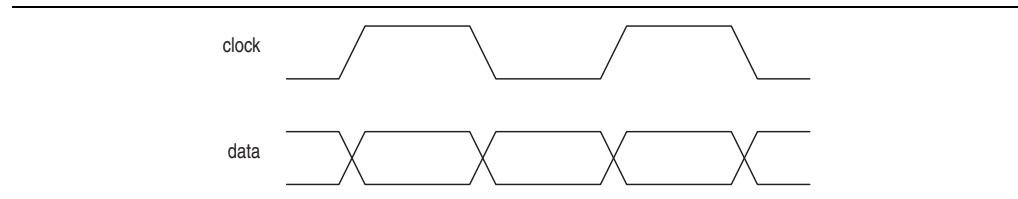
You can use a script included with the Quartus® II software installation to guide you through the process of creating constraints for source-synchronous interfaces. To use the script, type the following command in your project directory:

```
quartus_sta --ssc <project name>
```

Clock and Data Relationship

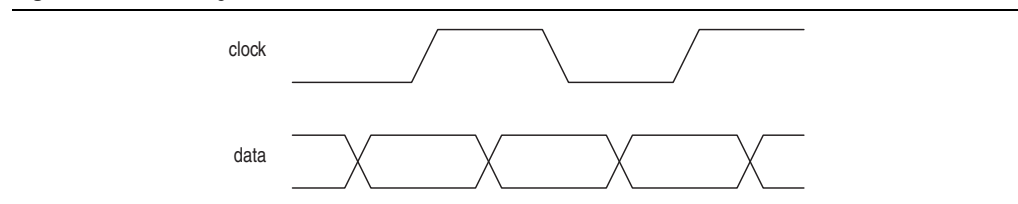
Some source-synchronous interfaces use a clock that is edge-aligned with the data, as shown in [Figure 2](#). When a clock is edge-aligned with the data, the receiving device shifts the clock as necessary to capture the data. Some interfaces capture data after the first rising or falling clock edge. Therefore, further logic is required in addition to a clock shift after the first rising or falling latch edge.

Figure 2. Edge-Aligned Clock and Data



Other source-synchronous interfaces use a clock that is shifted with respect to the data (typically center-aligned with the data), as shown in [Figure 3](#). The receiving device directly uses the shifted clock to capture the data, especially in low-speed interfaces.

Figure 3. Center-Aligned Clock and Data

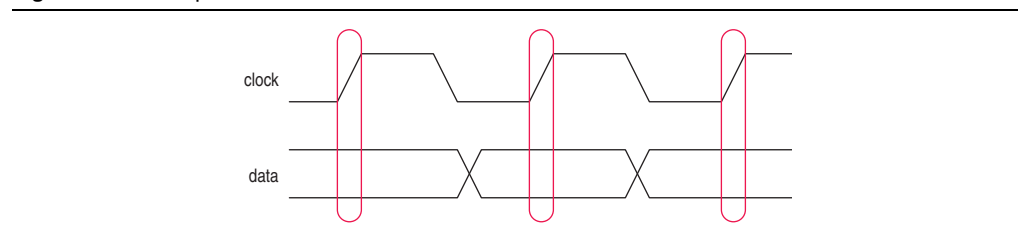


When a source-synchronous input clock directly latches the data, the receiving device does not perform any extra clock alignment. However, in some interfaces, a phase-locked loop (PLL) shifts the input clock, which is then used to latch the data. If a PLL shifts the input clock, you can adjust the clock and data timing relationship by adjusting the PLL phase offset.

SDR (Single Data Rate) and DDR (Double Data Rate)

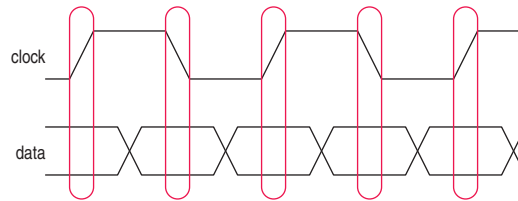
In source-synchronous SDR interfaces, one edge of the clock, typically the rising edge, transfers the data, as shown in [Figure 4](#). The time required to transmit one bit, known as the unit interval (UI), is equal to the period of the clock.

Figure 4. SDR Capture



In source-synchronous DDR interfaces, data is transferred on both edges of the clock, as shown in [Figure 5](#). The UI is equal to half the period of the clock, assuming a 50/50 duty cycle.

Figure 5. DDR Capture



Data constraints are necessary for each active clock edge. SDR interfaces require constraints for only one active clock edge, typically the rising edge. DDR interfaces require constraints that are relative to the rising and falling clock edges.

[Example 1](#) shows constraints that are relative to the rising clock edge. For an SDR interface, no other data constraints are necessary.

Example 1. Sample Output Constraints for an SDR Interface

```
set_output_delay -clock [get_clocks output_clk] -max 2 [get_ports data_out]
set_output_delay -clock [get_clocks output_clk] -min -1 \
  [get_ports data_out] -add_delay
```

A DDR interface requires data constraints that are relative to the falling and rising edges of the clock.

When you make data constraints for DDR interfaces, duplicate the constraints that are relative to the rising clock edge, and add the `-clock_fall` and `-add_delay` options so the constraints are relative to the falling clock edge.

[Example 2](#) shows data constraints that are relative to the rising and falling clock edges. The `-clock_fall` option makes the constraint relative to the falling clock edge, and the `-add_delay` option allows multiple maximum or minimum delay constraints to apply to the same port.

Example 2. Sample Output Constraints for a DDR Interface

```
set_output_delay -clock [get_clocks output_clk] -max 2 [get_ports data_out]
set_output_delay -clock [get_clocks output_clk] -min -1 \
  [get_ports data_out] -add_delay
set_output_delay -clock [get_clocks output_clk] -max 2 -clock_fall \
  [get_ports data_out] -add_delay
set_output_delay -clock [get_clocks output_clk] -min -1 -clock_fall \
  [get_ports data_out] -add_delay
```

Interface Constraints

Source-synchronous interfaces require the following three types of SDC constraints or exceptions:

- **Clock constraints**—Define the clocks used in the interface. Clock constraints define the period and include any other clock characteristics such as offset and uncertainty.
- **Input or output delay constraints**—Describe the required times for data to be valid at the interface. Input and output delay constraints are derived from timing parameters, such as skew, t_{SU} , or t_{CO} that specify the interface operation.

There are two methods for deriving input and output delays for source-synchronous interfaces based on the available or specified I/O timing:

- **System-centric method**—Takes into account the timing information for the FPGA as part of a larger system. Such timing information includes board trace delays and I/O timing requirements of the external device to which the FPGA interfaces. You can create constraints to describe these delays that are part of the system outside the FPGA. Use the system-centric method when you have timing information about the system with which the FPGA interfaces, or if you want to verify the system timing on that interface.
- **FPGA-centric (or data sheet) method**—Focuses on the clock and data relationship at the boundary of the FPGA. This method does not require any information about timing parameters outside the FPGA, such as board trace delays and I/O timing requirements of external devices. You can create constraints to specify the maximum acceptable skew across the data bus, and the timing relationship between the data and clock signals (center or edge alignment, for example). Use the FPGA-centric method when you constrain the source-synchronous interface for a specific skew and clock and data relationship. You can also use the FPGA-centric approach when you do not know the external device timing parameters.
- **Timing exceptions**—Control launch and latch edges used in timing analysis. Timing exceptions ensure that valid timing paths in the interface are analyzed, and invalid paths are not analyzed. For more information about why timing exceptions are necessary, refer to [“Default Timing Analysis Behavior”](#).



If you do not plan to migrate your design to HardCopy® devices, you can use the `set_max_skew` constraint to constrain source-synchronous interfaces. Altera recommends you use the methods described in this application note to constrain and analyze source-synchronous interfaces.

Default Timing Analysis Behavior

By default, timing analysis operates on the assumption that data launched by the rising clock edge is latched by the next rising clock edge. Source-synchronous interfaces, however, often exhibit different behavior. Data may be latched by the same edge that launches it, and source-synchronous DDR interfaces launch and latch data on rising and falling clock edges.

Figure 6 shows the setup relationships analyzed by default in an edge-aligned DDR interface. Solid red arrows indicate same-edge transfers (rise-to-rise and fall-to-fall), and dashed red arrows indicate opposite-edge transfers (rise-to-fall and fall-to-rise).

Figure 6. Setup Relationships in an Edge-Aligned DDR Interface

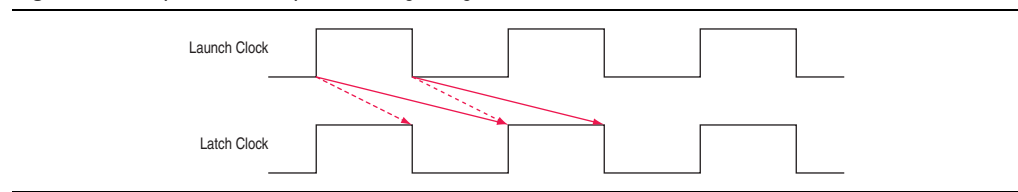


Figure 7 shows the hold relationships analyzed by default in an edge-aligned DDR interface. Solid blue arrows indicate hold relationships for same-edge transfers (rise-to-rise and fall-to-fall), and dashed blue arrows indicate hold relationships for opposite-edge transfers (rise-to-fall and fall-to-rise).

Figure 7. Hold Relationships in an Edge-Aligned DDR Interface

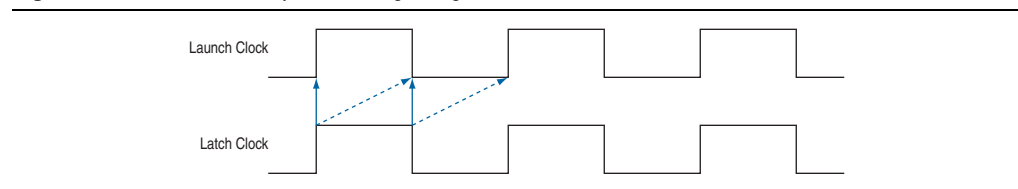


Figure 8 shows the setup relationships analyzed by default in a center-aligned DDR interface. Solid red arrows indicate same-edge transfers (rise-to-rise and fall-to-fall), and dashed red arrows indicate opposite-edge transfers (rise-to-fall and fall-to-rise).

Figure 8. Setup Relationships in a Center-Aligned DDR Interface

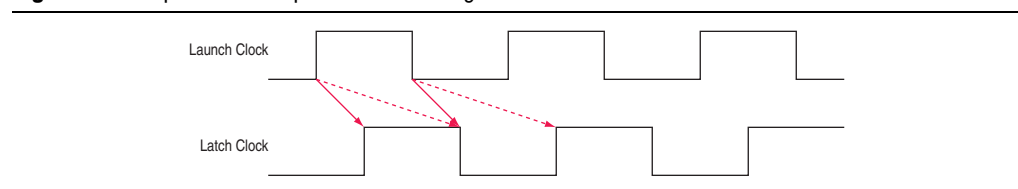
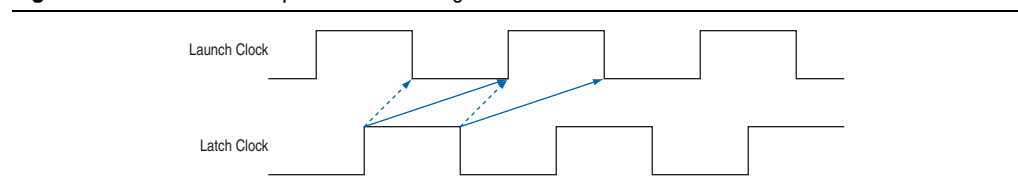


Figure 9 shows the hold relationships analyzed by default in a center-aligned DDR interface. Solid blue arrows indicate hold relationships for same-edge transfers (rise-to-rise and fall-to-fall), and dashed blue arrows indicate hold relationships for opposite-edge transfers (rise-to-fall and fall-to-rise).

Figure 9. Hold Relationships in a Center-Aligned DDR Interface



Depending on the alignment of your clock and data and the edges used to launch and latch data, you may have to add timing exceptions or adjust timing constraints to ensure correct timing analysis.

For example, if your interface implements same-edge data transfer, you must add false path exceptions to the opposite-edge transfers identified in [Figure 6](#) through [Figure 9](#), because opposite-edge transfers are invalid for your implementation. Likewise, if your interface implements opposite-edge data transfer, you must add false path exceptions to the same-edge transfers identified in [Figure 6](#) through [Figure 9](#), because same-edge transfers are invalid for your implementation. The false path exceptions exclude the invalid paths from timing analysis.

Source-Synchronous Outputs

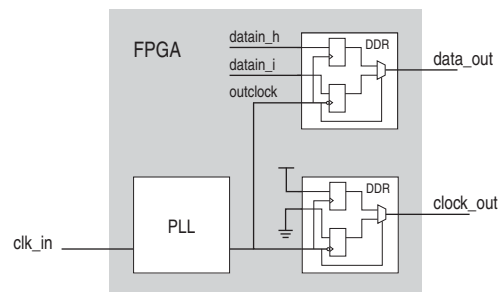
Source-synchronous outputs require the following types of clock constraints, output delay constraints, and timing exceptions:

- [“Output Clock Constraints” on page 7](#)
- [“System-Centric Output Delay Constraints” on page 11](#)
- [“FPGA-Centric Output Delay Constraints” on page 18](#)
- [“System-Centric Output Timing Exceptions” on page 12](#)
- [“FPGA-Centric Output Timing Exceptions” on page 25](#)

Output Clocks

The output clock is sourced by the FPGA. [Figure 10](#) shows an example of how a destination clock is sourced by the FPGA.

Figure 10. Destination Clock Sourced by the FPGA



You can use a variety of circuits to generate the output clock. The following are some types of output clocks:

- Clocks from a system PLL
- Clocks generated by a toggling clock output register, such as in the ALTDDIO megafunction
- Clocks controlled by a state machine that clocks data on one clock edge and the output clock on another clock edge
- Clocks driven by the same clock that clocks the data output

For sample clock circuits and SDC constraints, refer to [“Output Clock Constraints” on page 7](#).

Two common options are to use a clock that drives directly off chip, or to use a clock that drives off chip through DDR output registers with the ALTDDIO_OUT megafunction.

When output clocks are generated independently from the data output register clocks (with two PLL taps, for example), you can change the clock and data timing relationship by adjusting the relationship between clocks, such as by adjusting the PLL phase.

In an FPGA, there is low skew between clock and data outputs when the output clock drives through DDR registers because the routing from each DDR register to the corresponding output pin is nearly identical. When the clock drives directly off chip, the difference in routing types between the DDR data registers and the global clock network causes larger skew between clock and data outputs. Also, the allowable clock frequency is lower. Using the clock directly is acceptable in interfaces that run at lower clock speeds, and that tolerate more output skew between clock and data.

If you implement your design in a HardCopy II device, to achieve the best performance, use dedicated PLL outputs for the clock outputs instead of DDR output registers. You can use DDR output registers for the clock outputs if the interface is a low-speed interface, or if there are more clock outputs than available dedicated PLL outputs.

Clocks for the data and clock output DDR registers are generated with a PLL. In some cases, the same PLL output drives the data and clock output DDR registers. In other cases, two PLL outputs are used. You can use the same PLL output if your design contains:

- Center- or edge-aligned SDR outputs
- Edge-aligned DDR outputs

You should use separate PLL outputs if your design contains:

- Center-aligned DDR outputs (clock and data are 90° out of phase)
- A clock driving directly off chip (compensate for delay differences between clock and data)
- Clock and data that is not center- or edge-aligned (different than 90° or 180° out of phase)
- Precise adjustment of clock and data relationship (fine-tuning phase adjustment)

Output Clock Constraints

As with any circuit that includes a PLL, you must create generated clocks on the PLL outputs. Use the `derive_pll_clocks` command to automatically create all the generated clocks and keep generated clock characteristics (such as period, phase shift, and multiplication and division factors) synchronized with your PLL settings. The `derive_pll_clocks` also names the generated clocks according to the PLL output and hierarchy name.

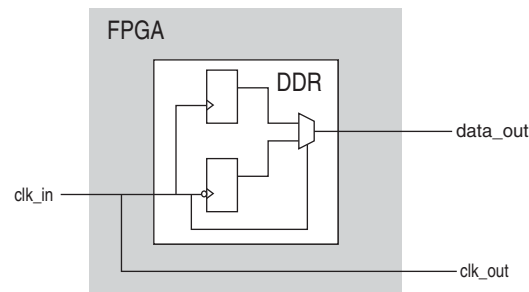
Instead of using the `derive_pll_clocks` command, you can use the `create_generated_clock` command to separately specify each PLL output clock. Creating generated clocks individually gives you flexibility in naming the clocks; however, you must remember to manually update the clock definitions in your SDC file when you change the PLL settings.

In addition to the generated clocks on PLL outputs, you must create a generated clock that is applied to the FPGA output clock port. This generated clock represents the clock that drives the destination of the source-synchronous interface, and it is the clock reference for output delay constraints. As with any generated clock, you must use appropriate phase and inversion options based on the behavior of the circuit that drives the clock.

Sample Clock Circuits and Constraints

Figure 11 and Example 3 show the circuit and constraints for an output with a common data clock and output clock.

Figure 11. Direct Clocking

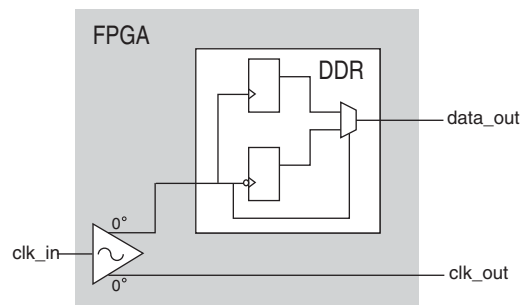


Example 3. Clock Constraints for an Output with Common Data and Output Clocks

```
create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name output_clock -source [get_ports clk_in] \
  [get_ports clk_out]
```

Figure 12 and Example 4 show the circuit and constraints for an edge-aligned output with independent data clocks and output clocks.

Figure 12. PLL Clocking for Edge-Aligned Output Clocks



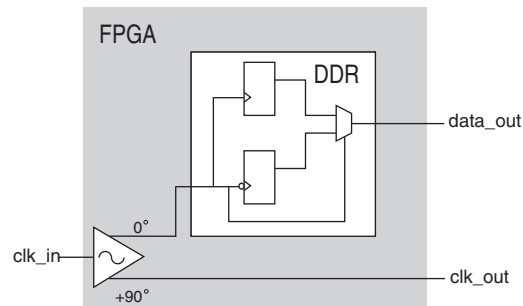
Example 4. Clock Constraints for an Output with Independent Data and Output Clocks

```

create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name data_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[0]]
create_generated_clock -name unshifted_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[1]]
create_generated_clock -name output_clock -source [get_pins PLL|clk[1]] \
  [get_ports clk_out]

```

Figure 13 and Example 5 show the circuit and constraints for a DDR center-aligned output with independent data clocks and output clocks.

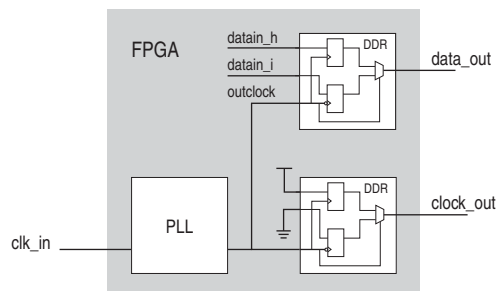
Figure 13. PLL Shifted Clocking for Center-Aligned Output Clocks**Example 5.** Clock Constraints for a Center-Aligned Output with Independent Data and Output Clocks

```

create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name data_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[0]]
create_generated_clock -name shifted_clock -phase 90 -source \
  [get_pins PLL|inclk[0]] [get_pins PLL|clk[1]]
create_generated_clock -name output_clock -source \
  [get_pins PLL|clk[1]] [get_ports clk_out]

```

Figure 14 and Example 6 show the circuit and constraints for a DDR output with a common data clock and output clock. The output clock is connected via an instance of the ALTDDIO_OUT megafunction.

Figure 14. Circuit with Common Data Clock and Output Clocks

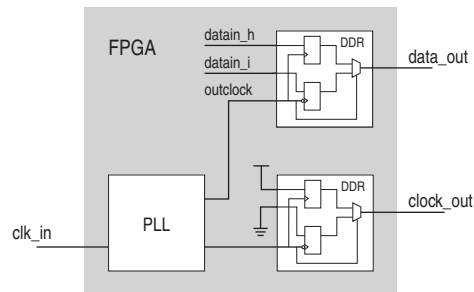
Example 6. Clock Constraints for an Edge-Aligned Output with Common Data and Output Clocks

```

create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name common_clock -source \
  [get_pins PLL|inclck[0]] [get_pins PLL|clk[0]]
create_generated_clock -name output_clock -source \
  [get_pins DDR|ddio_outa[0]|muxsel] [get_ports clk_out]

```

Figure 15 and Example 7 show the circuit and constraints for one PLL output driving the data DDR registers, and a separate PLL output driving the clock DDR registers.

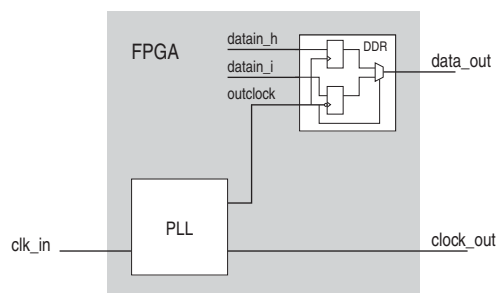
Figure 15. Output Circuit with Separate Output Clock and Input Clocks**Example 7.** Clock Constraints for an Output with Independent Data and Output Clocks

```

create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name data_clock -source [get_pins PLL|inclck[0]] \
  [get_pins PLL|clk[0]]
create_generated_clock -name pll_clock -source [get_pins PLL|inclck[0]] \
  [get_pins PLL|clk[1]]
create_generated_clock -name output_clock -source \
  [get_pins DDR|ddio_outa[0]|muxsel] [get_ports clk_out]

```

Figure 16 and Example 8 show the circuit and constraints for a PLL that drives the clock directly off chip.

Figure 16. Output Circuit, Driving Clock Directly Off Chip

Example 8. Clock Constraints for an Output with Direct Output Clocking

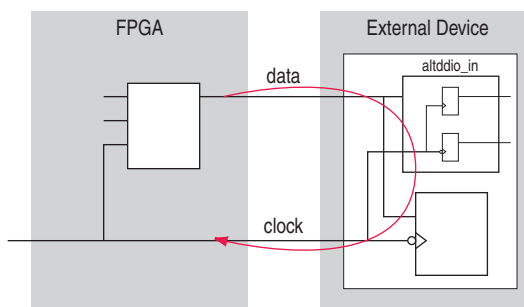
```

create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name data_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[0]]
create_generated_clock -name pll_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[1]]
create_generated_clock -name output_clock -source \
  [get_pins PLL|clk[1]] [get_ports clk_out]

```

System-Centric Output Delay Constraints

With the system-centric constraint method, the output delay value can include the setup or hold time requirement of the receiving device, as well as the board delays on data and clock traces, as shown in [Figure 17](#).

Figure 17. System-Centric Output**Output Maximum Delay**

[Equation 1](#) shows the equation to calculate the output maximum delay value. The output maximum delay value specifies an upper bound for the output delay because it uses the longest data path and the shortest clock path. It defines the setup relationship with the destination register.

Equation 1. Output Maximum Delay Calculation

$$\text{output maximum delay value} = \text{maximum trace delay for data} + t_{\text{SU}} \text{ of external register} - \text{minimum trace delay for clock}$$

[Example 9](#) shows the output maximum delay SDC constraint with the output maximum delay value from [Equation 1](#). This example is useful if your source-synchronous interface contains an SDR output.

Example 9. Output Maximum Delay Constraint

```

set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out]

```

[Example 10](#) shows a DDR output that has a duplicate constraint that applies to the falling clock edge.

Example 10. Output Maximum Delay Constraints for a DDR Output

```
set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out]
set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out]
-add_delay
```

Output Minimum Delay

Equation 2 shows how to calculate the output minimum delay value. The output minimum delay specifies a lower band for the output delay because it uses the minimum data delay and the maximum clock delay. It defines the hold relationship with the destination register.

Equation 2. Output Minimum Delay Calculation

$$\text{output minimum delay} = \text{minimum trace delay for data} - t_H \text{ of external register} - \text{maximum trace delay for clock}$$

Example 11 shows the output minimum delay SDC constraint, with the output minimum delay value from Equation 2. This example is useful if your source-synchronous interface contains an SDR output.

Example 11. Output Minimum Delay Constraint

```
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out*]
```

Example 12 shows a DDR output that has a duplicate constraint that applies to the falling clock edge.

Using the `-clock_fall` option causes the output delay constraint to apply to the falling clock edge. Using the `-add_delay` option allows multiple output delays to apply to the `data_out` ports.

Example 12. Output Minimum Delay Constraints for a DDR Output

```
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out*]
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out*] \
-add_delay
```

System-Centric Output Timing Exceptions

The following sections describe different alignment and capture edge combinations, and show any additional timing exceptions or adjusted constraints that are appropriate for each combination. For additional timing exceptions or constraint modifications that are necessary for the correct operation, refer to one of the following use cases:

- “Same-Edge Capture Edge-Aligned Output” on page 13
- “Opposite-Edge Capture Edge-Aligned Output” on page 16

- [“Same-Edge Capture Center-Aligned Output” on page 16](#)
- [“Opposite-Edge Capture Center-Aligned Output” on page 17](#)

Same-Edge Capture Edge-Aligned Output

Figure 18 shows the setup and hold relationships that must be analyzed for SDR same-edge capture, edge-aligned output. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 18. Desired Setup and Hold for SDR Same-Edge Capture

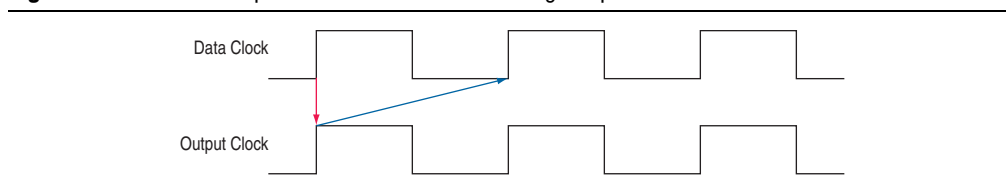
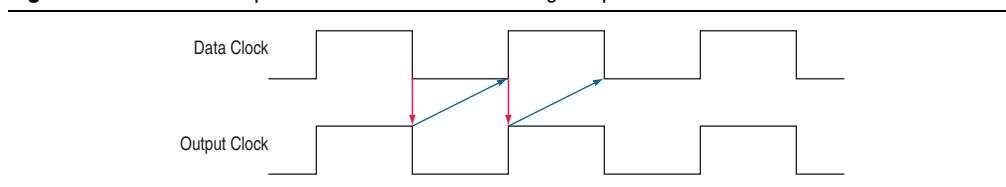


Figure 19 shows the setup and hold relationships that must be analyzed for DDR same-edge capture. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 19. Desired Setup and Hold for DDR Same-Edge Capture



To change the default timing analysis method to ensure that the correct setup and hold relationships for same-edge capture are analyzed, use one of the following exceptions:

- [“Destination Setup Multicycle Exception” on page 14](#)
- [“Add One Clock Period to Output Maximum Delay” on page 14](#)

Both exception types result in the same timing analysis results, but the multicycle exception is a better representation of design intent.

When using these timing exceptions, you must also add false path exceptions, as shown in [Example 13](#), to ensure that opposite-edge transfers are not analyzed.

Example 13. False Path Exceptions

```
set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
```

Destination Setup Multicycle Exception

Example 14 shows the SDC commands to modify the default timing analysis method by adding two destination setup multicycle exceptions with a value of zero.

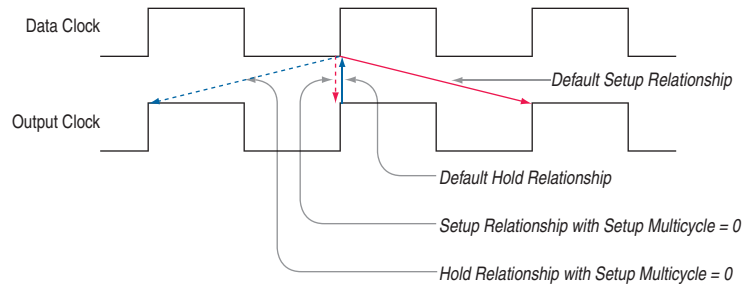
Example 14. Destination Setup Multicycle Exceptions

```
set_multicycle_path -setup -end 0 -rise_from [get_clocks data_clock] \
  -rise_to [get_clocks output_clock]
set_multicycle_path -setup -end 0 -fall_from [get_clocks data_clock] \
  -fall_to [get_clocks output_clock]
```

You must specify the rise-to-rise and fall-to-fall edges to ensure that the hold relationships are correct.

When you use destination setup multicycle exceptions with a value of zero, the destination clock edges used for setup and hold analysis shift back by one clock edge. These timing exceptions adjust the latch edge used for setup analysis to one clock edge earlier, that is, zero cycles after the launch edge; and adjust the latch edge used for hold analysis to one clock edge earlier, that is, one cycle before the launch edge. These latch edge adjustments for setup and hold analysis ensure that the data transfer between the clocks happens in zero cycles, as shown in **Figure 20**. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 20. Adding a Destination Multicycle Setup of Zero



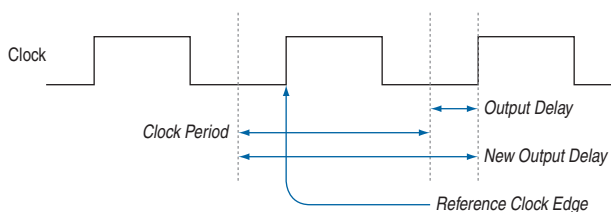
Add One Clock Period to Output Maximum Delay

Example 15 shows the SDC commands to modify the output maximum delay value to add one clock period.

Example 15. Adding One Clock Period

```
set_output_delay -max <output maximum delay value + clock period> -clock \
  [get_clocks output_clock] [get_ports data_out]
set_output_delay -max <output maximum delay value + clock period> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out]
```

Adding one clock period to the output maximum delay value does not adjust the latch edge, but it shifts the data required time one cycle earlier in time, as shown in **Figure 21**.

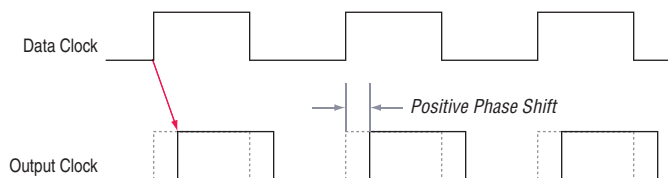
Figure 21. Adding Clock Period to Output Maximum Delay**Example 16.** False Path Exceptions

```

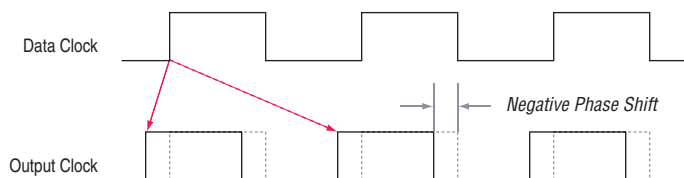
set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]

```

If you use a small positive phase shift to better align the clock and data outputs, do not use the multicycle exception of zero, or the extra clock period described previously. A small positive phase shift results in a small setup relationship, and the latch edge, analyzed by default, is the same clock edge as the launch edge (with a small shift), as shown in [Figure 22](#). Red arrows indicate setup relationships.

Figure 22. Small Positive Phase Shift, Same-Edge Capture

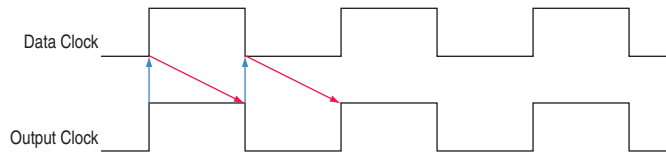
If you use a small negative phase shift to better align the clock and data outputs, you must use the multicycle exception of zero, or the extra clock period described previously. With a small negative phase shift, the correct setup relationship is to the edge shifted just before the launch edge, as shown by the solid arrow in [Figure 23](#). The dashed arrow indicates the default setup relationship, which is to the next clock edge after the launch edge. Red arrows indicate setup relationships.

Figure 23. Small Negative Phase Shift, Same-Edge Capture

Opposite-Edge Capture Edge-Aligned Output

Figure 24 shows the setup and hold relationships that must be analyzed for opposite-edge capture, edge-aligned output. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 24. Desired Setup and Hold for Opposite-Edge Capture, Edge-Aligned Output



Do not use multicycle exceptions for opposite-edge transfers, and do not add a clock cycle to the output maximum delay value. The only exceptions necessary for correct timing analysis are false path exceptions, as shown in Example 17.

Example 17. False Path Exceptions

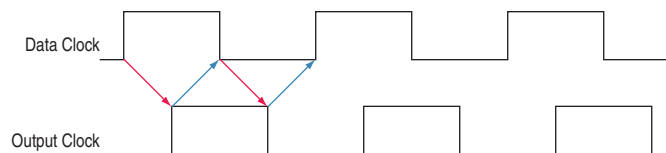
```
set_false_path -setup -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
```

If you use a small positive or negative phase shift to better align the clock and data outputs, the false path exceptions shown in Example 17 are still sufficient.

Same-Edge Capture Center-Aligned Output

Figure 25 shows the setup and hold relationships that must be analyzed in a same-edge capture, center-aligned output. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 25. Desired Setup and Hold for Same-Edge Capture, Center-Aligned Output



Do not use any multicycle exceptions, and do not add a clock cycle to the output maximum delay value. The only exceptions necessary for correct timing analysis are false path exceptions for opposite-edge transfers, as shown in Example 18.

Example 18.

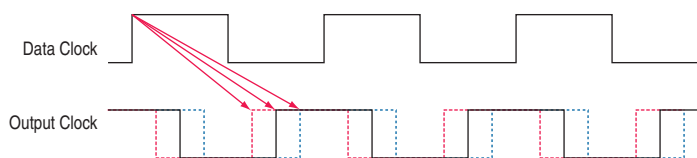
```

set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]

```

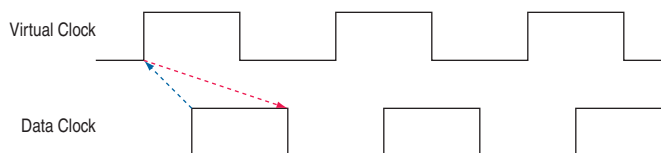
If you use a small positive or negative phase shift to better align the clock and data outputs, the false path exceptions in [Example 18](#) are still sufficient. [Figure 26](#) shows small positive and negative phase shifts that must be analyzed for a center-aligned source-synchronous output. The latch edge for the same-edge transfer never changes. Red arrows indicate setup relationships.

Figure 26. Small Positive or Negative Phase Shifts, Same-Edge Capture

**Opposite-Edge Capture Center-Aligned Output**

[Figure 27](#) shows the setup and hold relationships that must be analyzed for opposite-edge capture, center-aligned output. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 27. Desired Setup and Hold for Opposite-Edge Capture, Center-Aligned Output



The only exceptions required for correct timing analysis are false path exceptions, as shown in [Example 19](#), to prevent timing analysis on same-edge transfers.

Example 19.

```

set_false_path -setup -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]

```

FPGA-Centric Output Delay Constraints

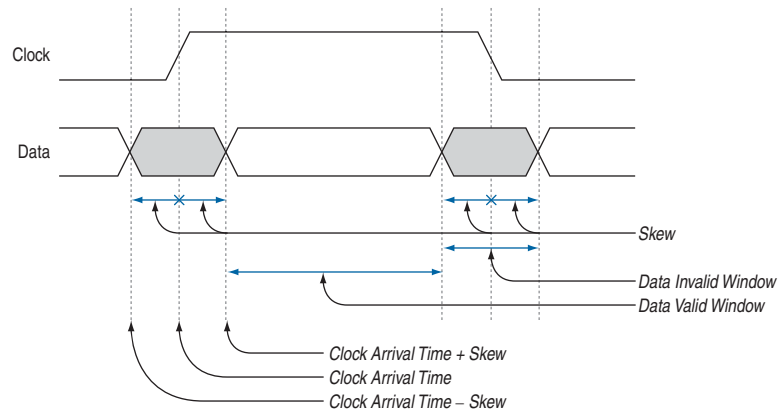
The FPGA-centric constraint method uses constraints derived from the clock and data relationship, as shown in [Equation 3](#). The data must arrive within a skew time window on either side of the clock arrival time.

Equation 3. Clock and Data Relationship Timing

$$\text{clock arrival time} - \text{skew} < \text{data arrival time} < \text{clock arrival time} + \text{skew}$$

[Figure 28](#) shows the clock and data arrival time relationship.

Figure 28. Clock and Data Arrival Time Relation



FPGA-centric constraints are based on a clock offset between the data and output clocks, and a skew requirement for the data. Clock offset is the time difference between the data clock edge and output clock edge. In an edge-aligned interface, the clock offset is zero. In a center-aligned interface, the clock offset is half the UI. In an SDR interface, the UI is equal to the clock period. In a DDR interface, the UI is equal to half the clock period. Therefore, in an SDR interface, the clock offset is one half of a clock period, and in a DDR interface, the clock offset is one quarter of a clock period.

You can use the following two approaches to compute FPGA-centric output constraints:

- [“Minimum Data Valid Constraints”](#)
- [“Maximum Data Invalid Constraints”](#) on page 24

Minimum Data Valid Constraints

You can use the following approaches to derive minimum data valid constraints based on the clock offset and skew values:

- Equations based on setup and hold relationships between the data and output clocks, described in [“Constraints Derived from Setup and Hold Relationships”](#) on page 19.
- Equations derived from early and late margins based on the clock offset and skew values, described in [“Constraints Derived from Early and Late Margins”](#) on page 21.

- If you use t_{CO} and minimum t_{CO} values to derive output delay constraints for your interface, use the approach described in “Constraints Derived from Early and Late Margins” on page 21.

All three approaches are correct and result in equivalent constraints.

Constraints Derived from Setup and Hold Relationships

To calculate constraint values with launch and latch edges, use one set of equations that specify a setup relationship, and one set of equations that specify a hold relationship.

Setup Relationship—Use a setup check to specify the left-hand side of the clock and data relationship shown in Equation 3 on page 18. A setup check verifies that the latest data arrival time (data arrival + skew) is earlier than the data required time (clock arrival). Equation 4 shows the components of arrival time and Equation 5 shows the components of required time.

Equation 4. Arrival Time

$$\text{arrival time} = \text{data arrival time} + \text{skew}$$

Equation 5. Required Time

$$\text{required time} = (\text{latch} - \text{launch}) + \text{clock arrival time} - \text{maximum delay of data}$$

Equation 6 shows the inequality that must be satisfied for positive slack, then shows the substitution of arrival and required times from Equation 4 and Equation 5, respectively.

Equation 6. Inequality for Positive Slack

$$\begin{aligned} &\text{arrival time} < \text{required time} \\ \text{data arrival time} + \text{skew} &< (\text{latch} - \text{launch}) + \text{clock arrival time} - \text{maximum delay of data} \end{aligned}$$

In a source-synchronous circuit, the data arrival time must match the clock arrival time. Equation 7 shows how you can create an equation that cancels out the data arrival and clock arrival time, and shows the value for output maximum delay of data.

Equation 7. Output Maximum Delay Calculation

$$\begin{aligned} \text{skew} &< (\text{latch} - \text{launch}) - \text{maximum delay of data} \\ \text{output maximum delay of data} &= (\text{latch} - \text{launch}) - \text{skew} \end{aligned}$$

Example 20 shows the resultant SDC command for an SDR output with the output maximum delay value derived from Equation 7.

Example 20. Output Maximum Delay Constraint for SDR Interfaces

```
set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out]
```

A DDR output has a duplicate constraint that applies to the falling clock edge, as shown in [Example 21](#).

Example 21. Output Maximum Delay Constraints for DDR Interfaces

```
set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out]
set_output_delay -max <output maximum delay value> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out]
-add_delay
```

Hold Relationship—Use a hold check to specify the right-hand side of the clock and data relationship shown in [Equation 3 on page 18](#). A hold check verifies that the earliest data arrival time (data arrival – skew) is later than the data required time (clock arrival). [Equation 8](#) shows the components of arrival time and [Equation 9](#) shows the components of required time.

Equation 8. Arrival Time

$$\text{arrival time} = \text{data arrival time} - \text{skew}$$

Equation 9. Required Time

$$\text{required time} = (\text{latch} - \text{launch}) + \text{clock arrival time} - \text{minimum delay of data}$$

[Equation 10](#) shows the inequality that must be satisfied for positive slack, then shows the substitution of arrival and required times from [Equation 8](#) and [Equation 9](#), respectively.

Equation 10. Inequality for Positive Slack

$$\begin{aligned} \text{arrival time} &> \text{required time} \\ \text{data arrival time} - \text{skew} &> (\text{latch} - \text{launch}) + \text{clock arrival time} - \text{minimum delay of data} \end{aligned}$$

In a source-synchronous circuit, the data arrival time must match the clock arrival time. [Equation 11](#) shows how you can create an equation that cancels out the data arrival and clock arrival time, and shows the value for output maximum delay of data.

Equation 11. Output Minimum Delay Calculation

$$\begin{aligned} -\text{skew} &> (\text{latch} - \text{launch}) - \text{minimum delay of data} \\ \text{output minimum delay of data} &= (\text{latch} - \text{launch}) + \text{skew} \end{aligned}$$

[Example 22](#) shows the resultant SDC command for an SDR output with the output maximum delay value derived from [Equation 11](#).

Example 22. Output Minimum Delay Constraint for SDR Interfaces

```
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out*]
```

A DDR output has a duplicate constraint that applies to the falling clock edge, as shown in [Example 23](#).

Example 23. Output Minimum Delay Constraints for DDR Interfaces

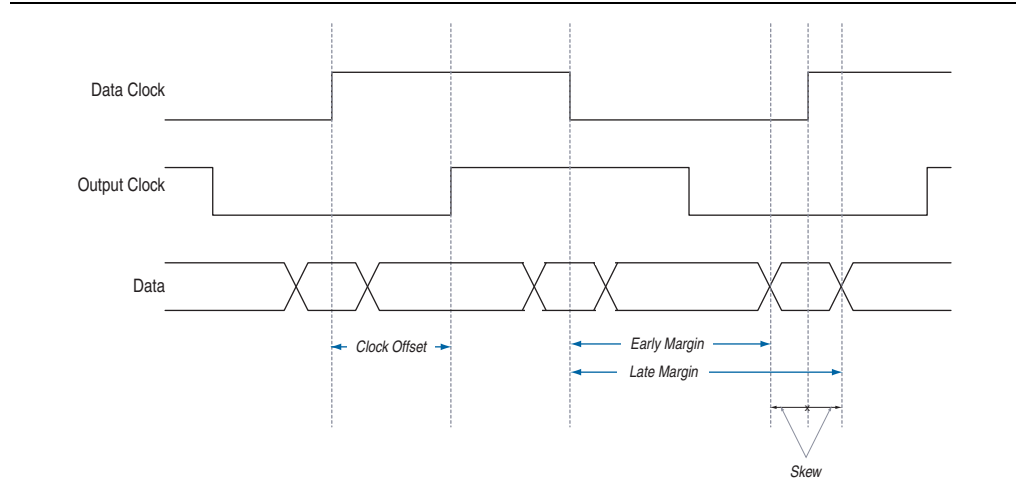
```
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] [get_ports data_out*]
set_output_delay -min <output minimum delay value> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out*]
-add_delay
```

Constraints Derived from Early and Late Margins

You can use the clock offset and skew values to determine early and late margins for the data to change, thus defining the minimum data valid window. The clock offset is the difference between the launch and latch edges. In a center-aligned interface, the launch and latch edges are half a UI apart. In an edge-aligned output, the launch and latches are at the same time, so the clock offset is zero.

The early margin corresponds to the minimum t_{CO} value and the late margin corresponds to the maximum t_{CO} value. [Figure 29](#) shows early and late margins indicated on a center-aligned DDR interface timing diagram.

Figure 29. Early and Late Margins



[Equation 12](#) shows how to calculate the output maximum delay value. The late margin is equivalent to a t_{CO} value.

Equation 12. Output Maximum Delay Value Derivation

$$\begin{aligned} \text{output maximum delay value} &= \text{unit interval} - \text{late margin} \\ \text{late margin} &= \text{unit interval} - \text{clock offset} + \text{output skew} \\ \text{output maximum delay value} &= \text{unit interval} - (\text{unit interval} - \text{clock offset} + \text{output skew}) \\ \text{output maximum delay value} &= \text{clock offset} - \text{output skew} \end{aligned}$$

Equation 13 shows how to calculate the output minimum delay value. The early margin is equivalent to a minimum t_{CO} value.

Equation 13. Output Minimum Delay Value Derivation

$$\begin{aligned} \text{output minimum delay value} &= 0 - \text{early margin} \\ \text{early margin} &= \text{unit interval} - \text{clock offset} - \text{output skew} \\ \text{output minimum delay value} &= 0 - (\text{unit interval} - \text{clock offset} - \text{output skew}) \\ \text{output minimum delay value} &= -\text{unit interval} + \text{clock offset} + \text{output skew} \end{aligned}$$

Figure 30 shows the clock values from Equation 12 on page 21 and Equation 13 with an edge-aligned SDR output timing diagram. In an edge-aligned interface, the clock offset value is zero.

Figure 30. Edge-Aligned SDR Output

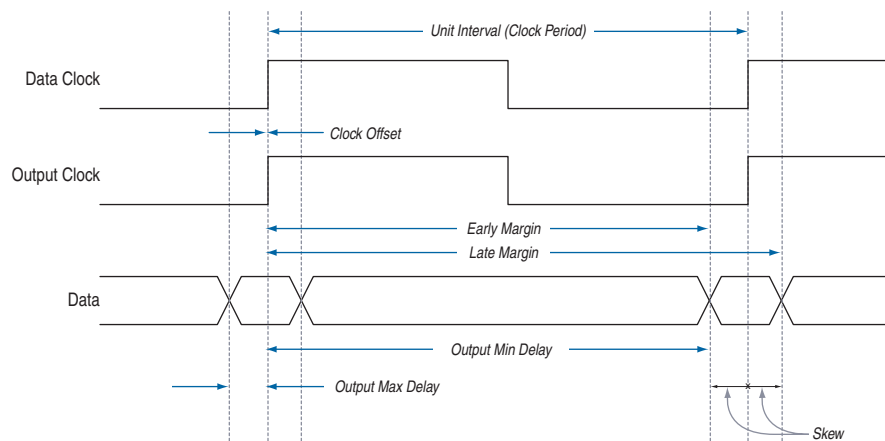


Figure 31 shows the clock values from Equation 12 on page 21 and Equation 13 with a center-aligned SDR output timing diagram. In a center-aligned interface, the clock offset is half the UI.

Figure 31. Center-Aligned SDR Output

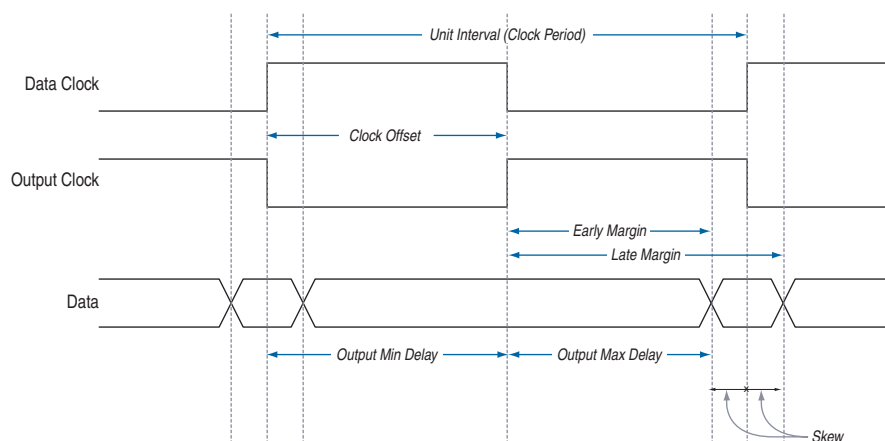


Figure 32 shows the clock values from Equation 12 on page 21 and Equation 13 on page 22 with an edge-aligned DDR output timing diagram. In an edge-aligned interface, the clock offset value is zero.

Figure 32. Edge-Aligned DDR Output

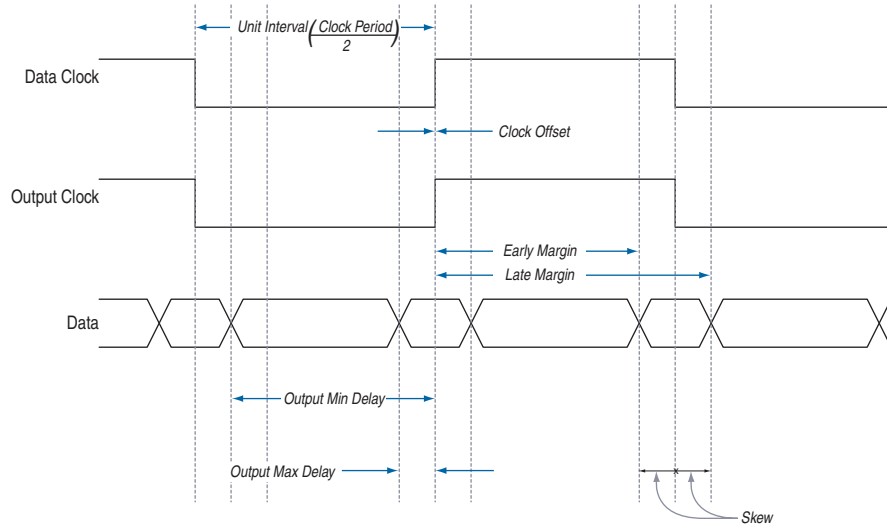
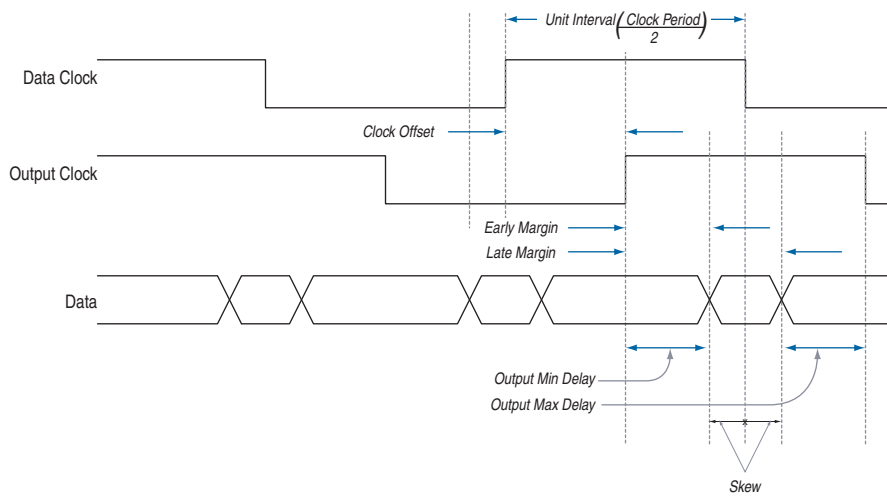


Figure 33 shows the clock values from Equation 12 on page 21 and Equation 13 on page 22 with a center-aligned DDR output timing diagram. In a center-aligned DDR interface, the clock offset is half the UI, or one quarter of the clock period.

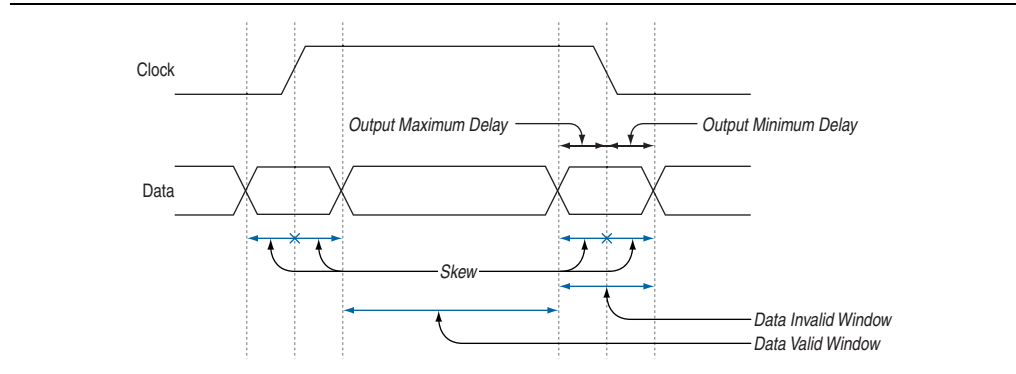
Figure 33. Center-Aligned DDR Output



Maximum Data Invalid Constraints

Figure 34 shows the data invalid window derived from the positive and negative skew values.

Figure 34. Data Invalid Window



Altera recommends constraining the maximum data invalid time instead of the minimum data valid time. To constrain the maximum data invalid time, you must set up the output minimum delay and output maximum delay constraints. The output minimum delay constraint value is the positive skew requirement, and the output maximum delay constraint is the negative skew requirement. Depending on the operation of your interface, you may have to adjust the constraint values or add exceptions to ensure correct timing analysis, but there are no calculations required to determine the initial output delay values.

The output minimum delay shown in Figure 34 corresponds to the minimum t_{CO} requirement, which is the earliest time that data can change after the clock edge. When you constrain the maximum data invalid time, the earliest time that data can change after the clock edge is the positive skew time. The output maximum delay shown in Figure 34 corresponds to the t_{CO} requirement, which is the latest time that data changes after the clock edge. When you constrain the maximum data invalid time, the latest time that data changes occur is skew requirement, which is before the clock edge. To express that time as occurring after the clock edge, you must use the negative slack value.



If you use this approach to constrain the maximum data invalid time and you use the PrimeTime software, you must modify your output maximum and minimum delay values to work around a limitation of the PrimeTime software. Using an output maximum delay value that is less than the corresponding output minimum delay value results in incorrect timing analysis. The following sections describe constraint modifications necessary for each combination of edge capture and alignment. For an appropriate workaround, refer to the section that corresponds to the operation of your interface.

FPGA-Centric Output Timing Exceptions

Depending on the operation of your interface, you may have to add exceptions as described in the following sections to ensure proper timing analysis.

The values of launch and latch are the times that the respective clock edges occur for setup and hold checks. However, there are several ways in which you can combine the output delay constraints with exceptions to constrain different configurations of source-synchronous output interfaces.

The following sections describe different alignment and capture edge combinations, and show any additional timing exceptions or adjusted constraints that are appropriate for each combination. For additional timing exceptions or constraint modifications that are necessary for correct operation, refer to the section in the following list that corresponds to the operation of your interface:

- [“Same-Edge Capture Edge-Aligned Output”](#)
- [“Opposite-Edge Capture, Edge-Aligned Output” on page 28](#)
- [“Same-Edge Capture Center-Aligned Output” on page 30](#)
- [“Opposite-Edge Capture Center-Aligned Output” on page 34](#)

Same-Edge Capture Edge-Aligned Output

You can use two types of exceptions to change the default analysis so the correct setup and hold relationships for same-edge capture are analyzed. You can use the following methods to constrain the edge-aligned interface for same-edge capture:

- [“Destination Multicycle Exceptions Method” on page 26](#)
- [“Add One Clock Period” on page 27](#)

Both methods result in the same timing analysis results. Using multicycle exceptions is the best representation of the design intent.

When you constrain DDR outputs, you must also use the false path exceptions in [Example 24](#), so that the opposite-edge transfers are not analyzed.

Example 24.

```
set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \  
  [get_clocks output_clock]  
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \  
  [get_clocks output_clock]  
set_false_path -hold -rise_from [get_clocks data_clock] -fall_to \  
  [get_clocks output_clock]  
set_false_path -hold -fall_from [get_clocks data_clock] -rise_to \  
  [get_clocks output_clock]
```

Destination Multicycle Exceptions Method

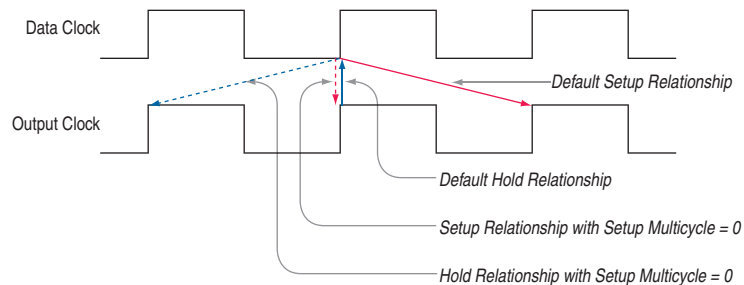
Use two destination setup multicycle exceptions with a value of zero, and two destination hold multicycle exceptions with a value of -1 , as shown in [Example 25](#).

Example 25.

```
set_multicycle_path -setup -end 0 -rise_from [get_clocks \
  data_clock] -rise_to [get_clocks output_clock]
set_multicycle_path -setup -end 0 -fall_from [get_clocks \
  data_clock] -fall_to [get_clocks output_clock]
set_multicycle_path -hold -end -1 -rise_from [get_clocks \
  data_clock] -rise_to [get_clocks output_clock]
set_multicycle_path -hold -end -1 -fall_from [get_clocks \
  data_clock] -fall_to [get_clocks output_clock]
```

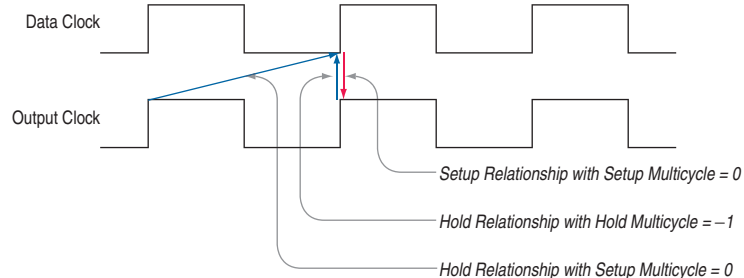
Using destination setup multicycle exceptions adjusts the setup latch edge to one edge earlier (zero cycles after the launch edge) so data transfer between the clocks happens in zero cycles, as shown in [Figure 35](#). Red arrows indicate setup relationships, and blue arrows indicate hold relationships. Using a destination setup multicycle exception of 0 also causes the default hold analysis edge to move one edge earlier (one cycle before the launch edge).


Figure 35. Adding a Destination Multicycle Setup of Zero



However, hold analysis must also occur for the same launch/latch edges as the setup analysis occurred. Adjusting the edge used for hold analysis requires a destination hold multicycle exception with a value of -1 . The value of -1 moves the latch edge used for hold analysis one cycle later, as shown in [Figure 36](#). Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 36. Destination Multicycle Setup of -1

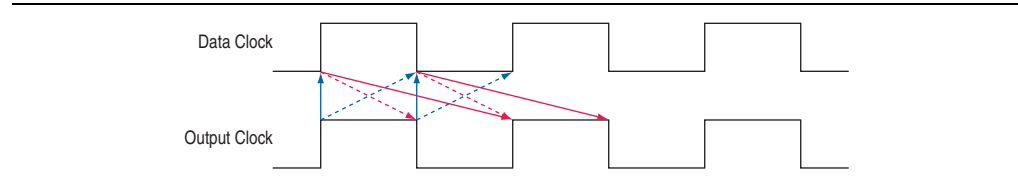


 Using the destination multicycle exception method is not compatible with the PrimeTime software, because the output maximum delay value is less than the output minimum delay value.

Add One Clock Period

Figure 37 shows the default setup and hold relationships that are analyzed for same-edge capture edge-aligned outputs. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 37. Default Setup and Hold Relationships



Using the FPGA-centric approach to constrain the interface requires you to make adjustments to cause the latch edge to be the same as the launch edge for setup and hold analysis.

Figure 38 shows the setup and hold relationships that must be analyzed for SDR same-edge capture. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 38. Desired Setup and Hold for SDR Same-Edge Capture

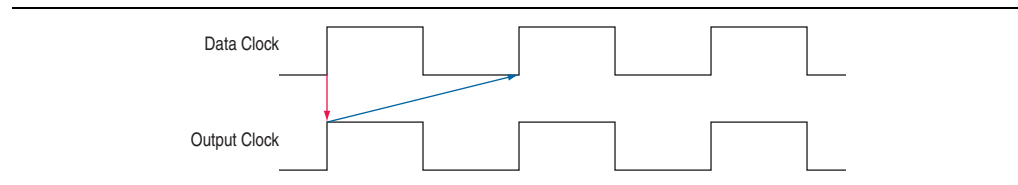
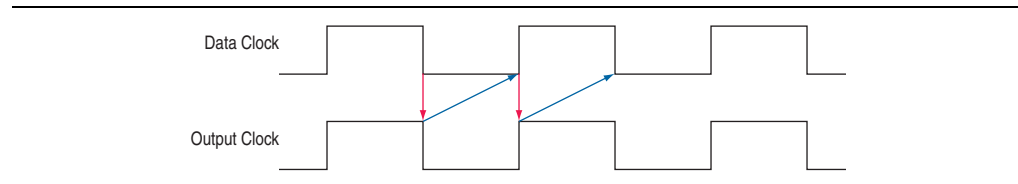


Figure 39 shows the setup and hold relationships that must be analyzed for DDR same-edge capture. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 39. Desired Setup and Hold for DDR Same-Edge Capture



In a same-edge capture configuration, the default setup relationship is between a rising edge and the rising edge that follows it, or a falling edge and the falling edge that follows it. The setup relationship must be between edges that occur at the same time. To cancel out the effect of the one-period-long setup time, you must add one period to the output maximum delay, as shown in Equation 14.

Equation 14.

$$\text{output maximum delay} = (\text{latch} - \text{launch}) - \text{period skew}$$

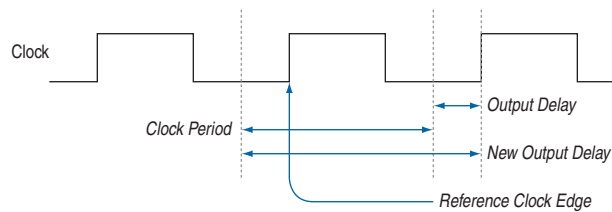
Example 26 shows the modified constraints.

Example 26.

```
set_output_delay -max <output maximum delay value + (latch - launch)> \
  -clock [get_clocks output_clock] [get_ports data_out]
set_output_delay -max <output maximum delay value + (latch - launch)> \
  -clock [get_clocks output_clock] -clock_fall [get_ports data_out]
```

Adding one clock period to the output maximum delay value does not actually adjust the latch edge, but it shifts the data required time one cycle earlier in time, as shown in Figure 40.

Figure 40. Add Clock Period to Output Maximum Delay



Adding one clock period to the output maximum delay values is compatible with the PrimeTime software.

The default hold relationship between the same edges is correct, with the launch edge occurring at the same time as the latch edge. Therefore, the latch and launch terms cancel out of Equation 15, resulting in an output minimum delay value, which is unchanged from its original value.

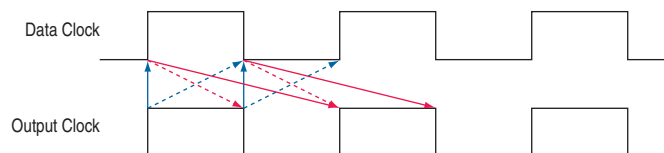
Equation 15.

$$\begin{aligned} \text{output minimum delay} &= (\text{latch} - \text{launch}) - \text{skew} \\ \text{output minimum delay} &= \text{skew} \end{aligned}$$

Opposite-Edge Capture, Edge-Aligned Output

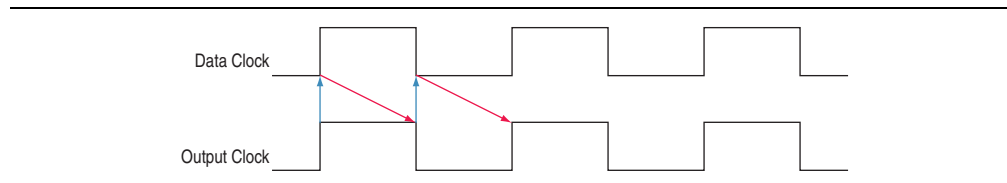
Figure 41 shows the default setup and hold relationships that are analyzed for opposite-edge capture edge-aligned outputs. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 41. Opposite-Edge Capture, Edge-Aligned Output



Using the FPGA-centric approach to constrain the interface requires you to make adjustments to cause the latch edge time to be the same as the launch edge time for setup and hold analysis. Figure 42 shows the setup and hold relationships that must be analyzed for opposite-edge capture. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 42. Desired Setup and Hold for Opposite-Edge Capture



In an opposite-edge capture configuration, the default setup relationship is between a launch edge and the next opposite edge that follows it. The launch and latch edges are half a period apart. The setup relationship must be between edges that occur at the same time. To cancel out the effect of the half-period-long setup time, you must add half a period to the output maximum delay, as shown in Equation 16.

Equation 16.

$$\text{output maximum delay} = (\text{latch} - \text{launch}) - \text{skew}$$

$$\text{output maximum delay} = \left(\frac{\text{period}}{2}\right) - \text{skew}$$

The hold relationship is between a launch edge and the latch edge that precedes it, so the launch and latch edges are half a period apart. The hold relationship must be between edges that occur at the same time. To cancel out the effect of the half-period-long hold time, you must subtract half a period from the output minimum delay, as shown in Equation 17. Subtract the half period because the launch time is later than the latch time, so (latch – launch) is negative.

Equation 17.

$$\text{output minimum delay} = (\text{latch} - \text{launch}) + \text{skew}$$

$$\text{output minimum delay} = \left(\frac{\text{period}}{2}\right) + \text{skew}$$

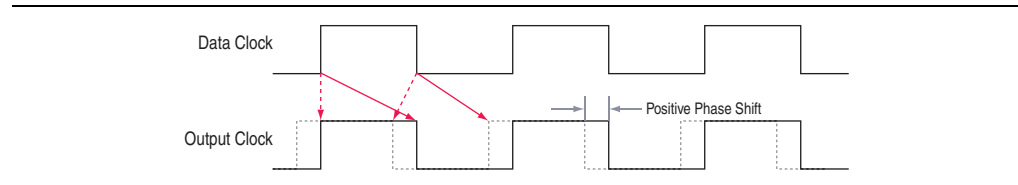
Do not use any multicycle or delay exceptions for opposite-edge transfers. The only exceptions necessary for correct timing analysis are false path exceptions, shown in Example 27.

Example 27.

```
set_false_path -setup -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
```

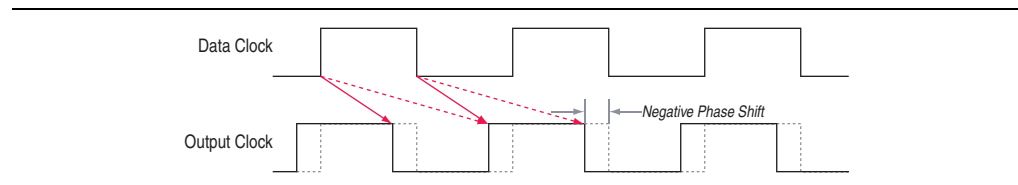
If you use a small positive phase shift to align the clock and data outputs better, the false path exceptions in [Example 27](#) are still sufficient. A small positive phase shift results in a small setup relationship, shown in [Figure 43](#). Red arrows indicate setup relationships. False path exceptions are shown with dashed arrows, and the false path exceptions cover rise-rise and fall-fall setup and hold paths.

Figure 43. Small Positive Phase Shift with False Path Exceptions



If you use a small negative phase shift to better align the clock and data outputs, the false path exceptions in [Example 27](#) are still sufficient. The default latch edge is the next clock edge after the launch edge, and it is still the opposite edge, as shown in [Figure 44](#). Red arrows indicate setup relationships.

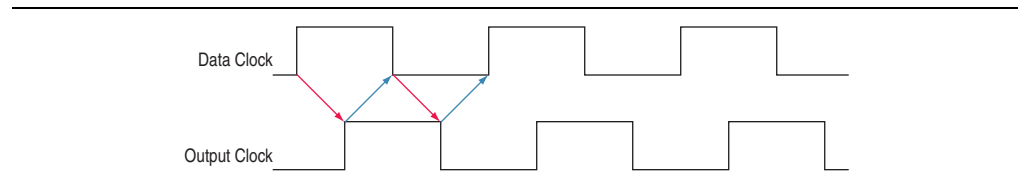
Figure 44. Small Negative Phase Shift, Opposite-Edge Capture



Same-Edge Capture Center-Aligned Output

[Figure 45](#) shows the default setup and hold relationships that are analyzed in a same-edge capture, center-aligned output. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 45. Default Setup and Hold for Center-Aligned Output



You can use two types of exceptions to change the default analysis so the correct setup and hold relationships for same-edge capture are analyzed. Both methods have the effect of causing the latch edge to align with the latch edge for setup and hold analysis. You can use the following methods to constrain the center-aligned interface for same-edge capture:

- “Maximum and Minimum Delay Exceptions”
- “Add Partial Clock Period” on page 33

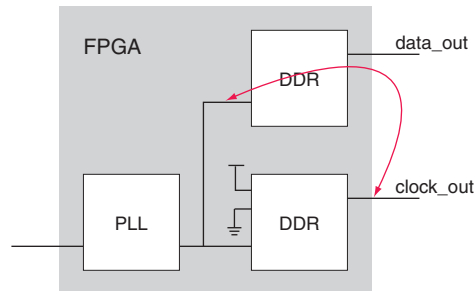
Both methods result in the same timing analysis results, but the maximum and minimum delay exceptions method is not compatible with the PrimeTime software.

Maximum and Minimum Delay Exceptions

Using maximum and minimum delay exceptions between the data clock and the output clock allows you to directly control the timing relationship between the two.

Figure 46 shows the points for the timing relationship specification with a red curved arrow.

Figure 46. Maximum and Minimum Delay Exceptions




The nominal value for the maximum and minimum delay exceptions between the data clock and output clock is zero. Example 28 shows sample maximum and minimum delay exceptions that correspond to Figure 46. There are situations when it is appropriate to use non-zero values, described below.

Example 28.

```
set_max_delay -from [get_clocks data_clock] -to [get_clocks output_clock] 0
set_min_delay -from [get_clocks data_clock] -to [get_clocks output_clock] 0
```

Using the maximum and minimum delay exceptions of zero affects only timing analysis, not the actual circuit operation. The output clock continues to be shifted by the amount implemented in the PLL that drives it. Setting maximum and minimum delay values of 0 causes the timing analyzer to override any output clock phase shift when it performs timing analysis. Effectively, the maximum and minimum delay values of zero cause the interface to be analyzed as if it were an edge-aligned interface. When the interface is edge-aligned, you can use the positive and negative skew values for the output minimum delay and output maximum delay values, as described in “Maximum Data Invalid Constraints” on page 24.

 Do not use this method if you also use the PrimeTime software to perform timing analysis for your design. If the value for an output maximum delay is more negative than the value for an output minimum delay, the PrimeTime software uses the more negative value for the output maximum and output minimum delays. The TimeQuest timing analyzer uses both output maximum and output minimum delay values, regardless of their positive or negative relationship.

[Example 29](#) shows the output delay constraints that use the positive and negative skew values.

Example 29.

```
set_output_delay -max <negative skew> -clock \
  [get_clocks output_clock] [get_ports data_out]
set_output_delay -max <negative skew> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out] -add_delay
set_output_delay -min <positive skew> -clock \
  [get_clocks output_clock] [get_ports data_out*]
set_output_delay -min <positive skew> -clock \
  [get_clocks output_clock] -clock_fall [get_ports data_out*] -add_delay
```

Use the maximum and minimum delay exceptions to specify any deviation from the ideal clock shift you intend. If you shift the PLL slightly to center the data valid window, update your maximum and minimum delay exceptions to reflect the difference between your ideal phase shift and your actual phase shift.

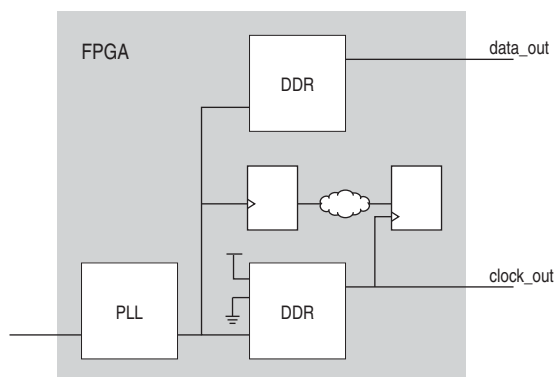
For example, a center-aligned DDR output with a 5 ns UI has an ideal PLL phase shift of 90° (2.5 ns). If you shift the clock output by 10° to center the data valid window, change the maximum and minimum delay exceptions to 138 ps to reflect the 10° shift. The 10° phase shift corresponds to 138 ps because the shift amount is $5 \text{ ns} \times 10^\circ / 360^\circ$.

The value you use for the maximum and minimum delay exceptions must include the skew between the clocks. In those cases in which skew is negligible, such as when you use ALTDDIO_OUT megafunctions for both the data and clock outputs, you can use zero for the skew. You must not use zero for the skew when a PLL output drives directly off chip, for example.

If there are data paths between the source and destination clocks in addition to the source-synchronous output registers, as shown in [Figure 47](#), change the value of the `-from` option in the delay exceptions shown in [Example 28 on page 31](#). Use a collection of registers with a wildcard that restricts the collection to the source-synchronous output registers driven by the source clock. For example, if your source-synchronous output registers are instantiated in a module named `ss_data_if`, and you use the ALTDDIO_OUT megafunction, use the following collection:

```
[get_registers *ss_data_if*altddio_out_component*]
```

Figure 47. Extra Data Paths

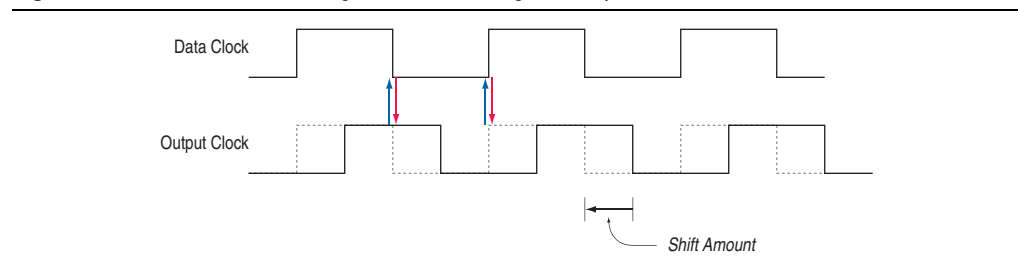


Add Partial Clock Period

In a same-edge capture configuration, the setup and hold relationships must be between the same launch and latch edges, and in a center-aligned configuration, the latch clock is shifted by a half-UI. In an SDR interface, center alignment requires a clock shift of half a period, and in a DDR interface, center alignment requires a clock shift of a quarter period.

Figure 48 shows the launch and latch waveforms for a center-aligned output, and identifies the launch and latch edges used for setup and hold analysis. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 48. Launch and Latch Edges for Center-Aligned Output



Equation 18 shows how to calculate the value for the output maximum delay constraint.

Equation 18.

$$\begin{aligned} \text{output maximum delay} &= (\text{latch} - \text{launch}) - \text{skew} \\ \text{output maximum delay} &= \text{shift} - \text{skew} \end{aligned}$$

This equation is derived from Equation 4 on page 19, Equation 5 on page 19, Equation 6 on page 19, and Equation 7 on page 19. Figure 48 shows that the latch edge for setup analysis occurs one quarter period after the launch edge. Therefore, the value for the output maximum delay is $(1/4)\text{period} - \text{skew}$.

Equation 19 shows how to calculate the value for the output minimum delay constraint.

Equation 19.

$$\begin{aligned} \text{output minimum delay} &= (\text{latch} - \text{launch}) + \text{skew} \\ \text{output minimum delay} &= -\text{shift} + \text{skew} \end{aligned}$$

Equation 19 is derived from Equation 8 on page 20, Equation 9 on page 20, Equation 10 on page 20, and Equation 11 on page 20. Figure 48 shows that the latch edge for hold analysis occurs three quarters of a period before the launch edge. Therefore, the value for the output minimum delay is $(-3/4)\text{period} + \text{skew}$.

Example 30 shows the false path exceptions that are necessary for correct timing analysis.

Example 30.

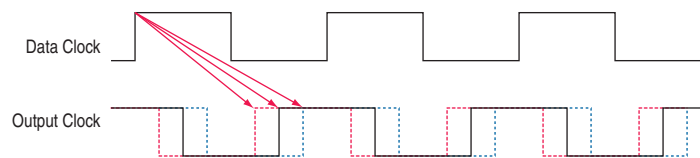
```

set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]

```

Even if you use a small positive or negative phase shift to align the clock and data outputs better, do not use any other exceptions or constraints. [Figure 49](#) shows small positive and negative phase shifts for a same-edge capture, center-aligned source-synchronous output. The latch edge is always the same edge as the launch edge, but with a shift. Red arrows indicate setup relationships.

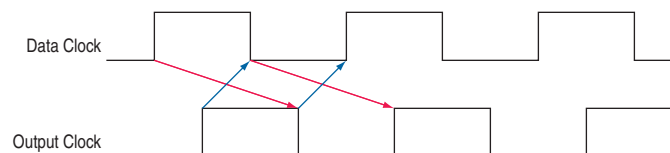
Figure 49. Small Positive or Negative Phase Shifts, Same-Edge Capture



Opposite-Edge Capture Center-Aligned Output

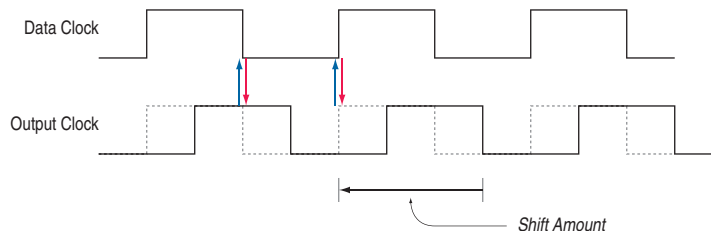
[Figure 50](#) shows the setup and hold relationships that must be analyzed for opposite-edge, center-aligned capture. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 50. Desired Setup and Hold for Center-Aligned Output



In an opposite-edge capture configuration, the setup and hold relationships are between opposite launch and latch edges, and in a center-aligned interface, the latch clock is shifted.

[Figure 51](#) shows the launch and latch waveforms for an opposite-edge capture center-aligned output, and identifies the launch and latch edges used for setup and hold analysis. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 51. Launch and Latch Edges for Center-Aligned Output

Equation 20 shows how to calculate the value for the output maximum delay constraint. The equation is derived from Equation 4 on page 19, Equation 5 on page 19, Equation 6 on page 19, and Equation 7 on page 19 in “Constraints Derived from Setup and Hold Relationships” on page 19.

Equation 20.

$$\begin{aligned} \text{output maximum delay} &= (\text{latch} - \text{launch}) - \text{skew} \\ \text{output maximum delay} &= \text{shift} - \text{skew} \end{aligned}$$

Figure 51 shows that the latch edge for setup analysis occurs three quarters of a period after the launch edge. Therefore, the value for the output maximum delay is $(3/4)\text{period} - \text{skew}$.

Equation 21 shows how to calculate the value for the output minimum delay constraint. The equation is derived from Equation 8 on page 20, Equation 9 on page 20, Equation 10 on page 20, and Equation 11 on page 20 in “Constraints Derived from Setup and Hold Relationships” on page 19.

Equation 21.

$$\begin{aligned} \text{output minimum delay} &= (\text{latch} - \text{launch}) + \text{skew} \\ \text{output minimum delay} &= -\text{shift} + \text{skew} \end{aligned}$$

Figure 51 shows that the latch edge for hold analysis occurs one quarter of a period before the launch edge. Therefore, the value for the output minimum delay is $(-1/4)\text{period} + \text{skew}$.

You must also add false path exceptions on same-edge transfers shown in Example 31 on page 35. The paths that are cut are for same-edge capture. Setting these false paths ensures that timing analysis is performed with respect to opposite-edge transfers.

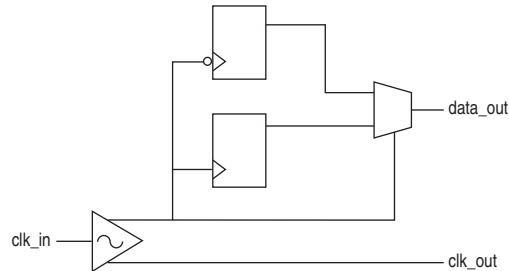
Example 31.

```
set_false_path -setup -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
```

Timing Analysis and Timing Closure

Figure 52 shows a simple source-synchronous DDR output design example used to illustrate timing analysis concepts.

Figure 52. Simple Source-Synchronous Output



To report timing for the output, use the `report_timing` command with `-from_clock` and `-to_clock` options. Use the name of the clock that drives the data output registers for the `-from_clock` option, and the name of the generated clock on the output clock port for the `-to_clock` option. For example, use the following two commands to report setup and hold timing for the circuit described previously:

- `report_timing -from_clock data_clock -to_clock output_clock \`
`-setup`
- `report_timing -from_clock data_clock -to_clock output_clock \`
`-hold`

You must perform timing analysis at all timing corners to ensure the interface meets its timing requirements.

Timing Closure

The slack values reported by each `report_timing` command indicate by how much the data meets its timing requirement. A negative value indicates that the constraint is not satisfied.

To achieve timing closure, the setup and hold slack values must be positive, and the setup and hold slack values must be balanced, or equal. The timing margin of the interface is equal to the lesser of the slack values. The best margin occurs when the setup slack equals the hold slack. It is not possible to balance the setup and hold slack values perfectly over the entire operating range of the FPGA. You must adjust the interface timing until the slack values are similar. Acceptable values depend on the operation of your interface. High-speed interfaces have small data valid windows; therefore, it is important for slack values to be very close. Low-speed interfaces can tolerate larger slack differences because of the larger data valid windows.

If the setup and hold slack values are not balanced, you must adjust parts of your interface. If you use separate clocks for the data output and clock output, it is straight-forward to shift one of the clocks (typically the output clock) to balance setup and hold slack values.

Use the steps shown in Equation 22 to compute the required phase shift.

Equation 22.

$$\text{time shift} = \frac{(\text{hold slack} - \text{setup slack})}{2}$$
$$\text{phase shift} = 360^\circ \times \frac{\text{time shift}}{\text{unit interval}}$$

When you shift a clock, you may also have to modify output delay constraints or timing exceptions if the latch clock edge moves past the launch clock edge. For example, if you add a positive phase shift to the output clock in a same-edge capture, edge-aligned circuit, you may have to modify the exceptions to use exceptions for center-aligned circuits.

If you use a common clock for the data output and clock output, you must take other steps. If you want to adjust the slack values by small amounts, you can adjust delay chain settings. Delay chains provide small and variable delays in the output path from a register to a pin.



For information about delay chains, refer to the appropriate handbook for the device family you are using. For information about setting delay chain values, refer to the Quartus II Help.

If you want to adjust the slack values more than you can with delay chains, you must change the location of elements of your interface, such as the data output registers.

Example

The following example is based on the output interface in [Figure 52 on page 36](#). The clock sourced by the device has a 10 ns period and is edge-aligned with the data. There is a +/-100 ps skew requirement for the data. The output is constrained as shown in [Example 32](#).

Example 32. SDC Constraints for Same-Edge Capture Edge-Aligned Output Example

```

# Clock constraints
create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name data_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[0]]
create_generated_clock -name pll_clock -source [get_pins PLL|inclk[0]] \
  [get_pins PLL|clk[1]]
create_generated_clock -name output_clock -source [get_pins PLL|clk[1]] \
  [get_ports clk_out]

# Output delay constraints of +/- skew
set_output_delay -clock output_clock [get_ports data_out] -max -0.1
set_output_delay -clock output_clock [get_ports data_out] -min 0.1 \
  -add_delay
set_output_delay -clock output_clock -clock_fall [get_ports data_out] \
  -max -0.1-add_delay
set_output_delay -clock output_clock -clock_fall [get_ports data_out] \
  -min 0.1 -add_delay

# Destination multicycle exceptions
set_multicycle_path -setup -end 0 -rise_from [get_clocks data_clock] \
  -rise_to [get_clocks output_clock]
set_multicycle_path -setup -end 0 -fall_from [get_clocks data_clock] \
  -fall_to [get_clocks output_clock]
set_multicycle_path -hold -end -1 -rise_from [get_clocks data_clock] \
  -rise_to [get_clocks output_clock]
set_multicycle_path -hold -end -1 -fall_from [get_clocks data_clock] \
  -fall_to [get_clocks output_clock]

# False path exceptions for opposite-edge transfers
set_false_path -setup -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -setup -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]
set_false_path -hold -rise_from [get_clocks data_clock] -fall_to \
  [get_clocks output_clock]
set_false_path -hold -fall_from [get_clocks data_clock] -rise_to \
  [get_clocks output_clock]

```

Use the commands in [Example 33](#) to report timing. Report timing uses both slow and fast corner delay models.

Example 33.

```

report_timing -from_clock data_clock -to_clock output_clock -setup
report_timing -from_clock data_clock -to_clock output_clock -hold

```

Table 1 shows the timing analysis results.

Table 1. Slack Values

Timing Corner	Setup Slack	Hold Slack
Slow	-2.107 ns	2.307 ns
Fast	-0.841 ns	1.041 ns

The worst case setup slack is -2.107 ns, and the worst case hold slack is 1.041 ns. The time shift is $(1.041 \text{ ns} - 2.107 \text{ ns}) \div 2$, which equals 1.574 ns. A time shift of 1.574 ns equals a phase shift of $360^\circ \times 1.574 \text{ ns} \div 10 \text{ ns}$, or 56° .

Change the phase of the PLL output that generates the output clock to 56° and update the SDC generated clock as shown in Example 34.

Example 34.

```
create_generated_clock -name pll_clock -phase 56 \
  -source [get_pins PLL|inclk[0]] [get_pins PLL|clk[1]]
```

The setup and hold slack values are now balanced, and the interface operates with the largest amount of margin.

Source-Synchronous Inputs

Source-synchronous inputs require the following types of constraints and exceptions:

- **Clock constraints**—Create clock constraints as described in “Input Clock Constraints” on page 40
- **Input delay constraints**—Create input delay constraints with one of the following methods:
 - “System-Centric Input Delay Constraints” on page 44
 - “FPGA-Centric Input Delay Constraints” on page 47
- **Timing exceptions**—Create timing exceptions as described in “Input Timing Exceptions” on page 50

Input Clocks

The input clock for the source-synchronous interface can clock input capture registers directly, or it can drive a PLL that clocks the input capture registers. There are usually larger timing margins for interfaces that use a PLL to clock the input capture registers than there are for interfaces that use the input clock directly to clock the capture registers. This is especially true for high-speed DDR interfaces. In SDR interfaces, the timing margins are often large enough that you can directly connect the input clock to the input capture registers. Direct clocking has the advantage of eliminating the PLL as a source of clock uncertainty. However, the PLL has the advantage of providing clock compensation over power, voltage, and temperature (PVT). At high interface speeds, the benefit of using a PLL outweighs the associated clock uncertainty.

If you use a PLL for your input clock, you should configure it in source-synchronous compensation mode. In source-synchronous compensation mode, the clock and data relationship at the input capture registers (in an I/O element, or IOE) is identical to the relationship at the FPGA device inputs. The PLL maintains the same phase relationship. This mode simplifies the constraint and adjustment process for timing closure, because you do not have to calculate any PLL phase shift to meet input timing requirements.

You can configure the PLL for other modes of operation, such as normal mode, but you might have to adjust the phase shift to meet timing. Different PLL modes adjust the clock to compensate for different delays in the FPGA. A phase shift is required if the clock and data relationship at the input capture registers do not meet timing because the PLL is compensating for a different delay.

Whether you use a direct clock connection or clock through a PLL depends on the type of interface and timing requirements.

Use a PLL on the input clock for the following inputs:

- Edge-aligned SDR or DDR inputs (create phase shift to latch data in the middle of the data valid window)
- High-speed inputs
- Precise adjustment of clock and data relationship (fine tuning phase adjustment)

Altera does not recommend using a PLL on the input clock for the following inputs:

- Center-aligned SDR or DDR inputs
- Low-speed inputs

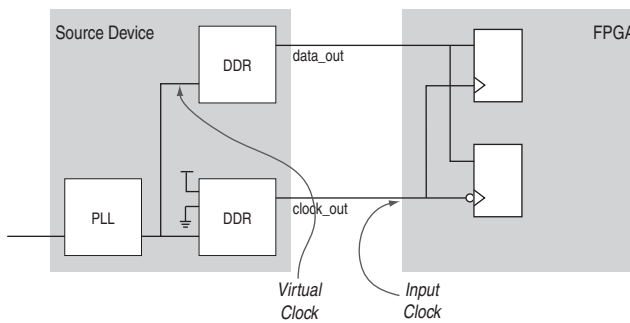
Input Clock Constraints

The source-synchronous interface is a register-to-register transfer between registers in separate devices, and the input clock is generated by the sourcing device. You must create a clock on the input clock port of your interface that describes the characteristics of the source clock, such as its period and phase shift. For example, if your sourcing device sends DDR data that is center-aligned (a clock phase of 90°), create a clock on the input clock port of your interface with a 90° phase shift with the `-waveform` option.

Virtual Clocks

You should create a virtual clock to represent the clock that clocks the data output registers in the source device. Then, use the virtual clock as the reference clock for input delay constraints. Create the virtual clock with the same period and phase shift as the real clock in the source device. If that clock has any phase shift, specify the rising and falling edges that correspond to the phase shift with the `-waveform` option.

Figure 53 shows the input and virtual clocks for a source-synchronous interface.

Figure 53. Input Clocking with a Virtual Clock

It is not necessary to use a virtual clock to constrain the input delays. You can create input delay constraints relative to the input clock instead of the virtual clock, but using a virtual clock makes the constraining of the interface easier and more accurate. A virtual clock makes it easy to constrain inputs with the skew-based FPGA-centric approach. You can use the positive and negative skew requirement values for the input maximum and minimum delay constraints with no other calculations. Even if the source or destination clock is shifted, the input maximum and minimum delay values do not change. Refer to [“Maximum Data Skew” on page 49](#) for more details.

You can apply clock uncertainty to the virtual clock that is independent of the uncertainty you apply to the clock feeding the FPGA. For more details, refer to [“Clock Uncertainty”](#).

Generated Clocks

For any circuit that includes a PLL, you must create generated clocks on the PLL outputs. Using the `derive_pll_clocks` command in your `.sdc` creates all the generated clocks automatically and keeps generated clock characteristics (such as period, phase shift, and multiplication and division factors) synchronized with your PLL settings. It also names the generated clocks according to the PLL output and hierarchy name.

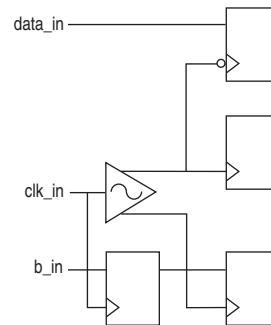
Instead of using the `derive_pll_clocks` command, you can use individual `create_generated_clock` commands to specify each PLL output clock separately. Creating generated clocks individually gives you flexibility in naming the clocks, but you must remember to manually update the clock definitions in your `.sdc` when you change PLL settings.

Clock Uncertainty

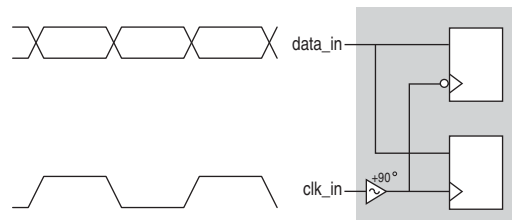
You should use the `set_clock_uncertainty` constraint to specify clock uncertainty for each clock in your input circuit. Use the clock uncertainty constraint to account for jitter, PLL phase shift error, and duty cycle distortion.

When you use a virtual clock as the reference clock for the input delay constraints, you can specify clock uncertainty for the I/O independently of the core clock. If your input clock is the reference clock for the input delay constraints, I/O clock uncertainty can affect the core clock uncertainty if the I/O clock is used for I/O and core transfers.

[Figure 54](#) shows a case where `clk_in` feeds I/O and core registers. Applying clock uncertainty to `clk_in` affects I/O and core timing.

Figure 54. Clock Jitter Example**Sample Input Clock Circuits and Constraints**

When clock and data arrive at the FPGA edge-aligned, it is common to use a PLL to shift the clock used to latch data, as shown in [Figure 55](#).

Figure 55. Edge-Aligned Input Clock

In this case, do not specify a phase adjustment on the input clock, because clock and data are edge-aligned. Specify the PLL phase adjustment on the generated clock applied to the PLL output. [Example 35](#) shows SDC constraints for [Figure 55](#).

Example 35.

```
create_clock -name virtual_clock -period 10.000
create_clock -name input_clock -period 10.000 [get_ports clock_in]
create_generated_clock -name plus_90_degrees -source [get_pins \
  PLL|inclk[0]] -phase 90
```

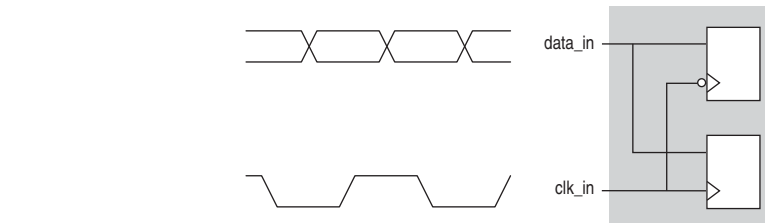
When the clock and data signals arrive at the FPGA center-aligned, specify that clock phase adjustment on the input clock with the `-waveform` option, as shown in [Example 36](#). If the clock drives input registers through a PLL, specify any phase shift applied by the PLL with the `-phase` option for the generated PLL clock.

Example 36.

```
create_clock -name virtual_clock -period 10.000
create_clock -name input_clock -period 10.000 [get_ports clock_in] -waveform \
  { 2.5 7.5 }
create_generated_clock -name plus_0_degrees -source \
  [get_pins PLL|inclk[0]]
```

Figure 56, Figure 57, and Figure 58 show various input clock configurations followed by the corresponding SDC constraints.

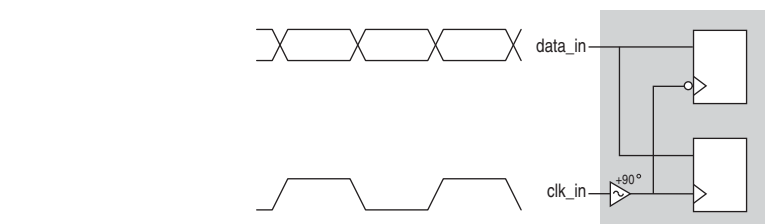
Figure 56. Direct Clocking with Center-Aligned Data



Example 37.

```
create_clock -name virtual_source -period 10.000
create_clock -name input_clock -period 10.000 -waveform { 2.5 7.5 } \
  [get_ports clk_in]
```

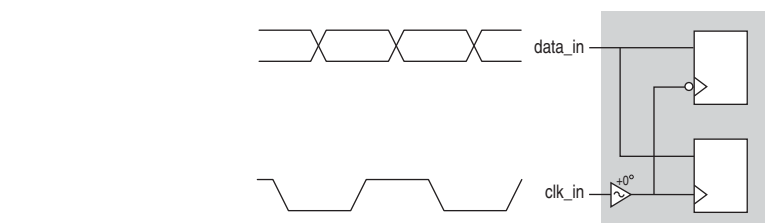
Figure 57. PLL Shifted Clocking with Edge-Aligned Data



Example 38.

```
create_clock -name virtual_source -period 10.000
create_clock -name input_clock -period 10.000 [get_ports clk_in]
create_generated_clock -name shifted_clock -source \
  [get_pins PLL|inclk[0]] -phase 90 [get_pins PLL|clk[0]]
```

Figure 58. PLL Clocking with Center-Aligned Data



Example 39.

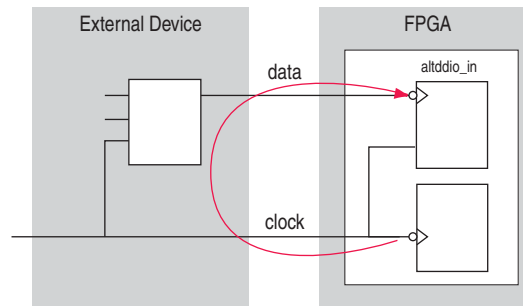
```

create_clock -name virtual_source -period 10.000
create_clock -name input_clock -period 10.000 -waveform { 2.5 7.5 } \
  [get_ports clk_in]
create_generated_clock -name internal_clock -source \
  get_pins PLL|inclk[0]] [get_pins PLL|clk[0]]

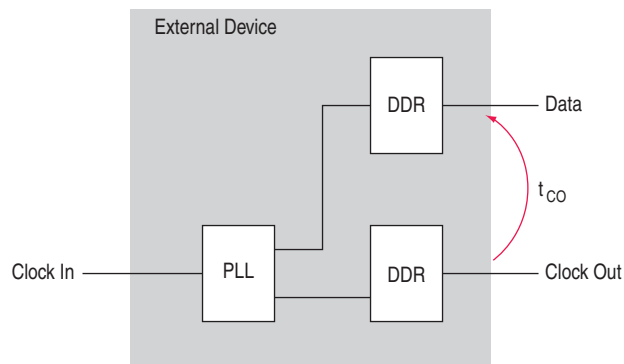
```

System-Centric Input Delay Constraints

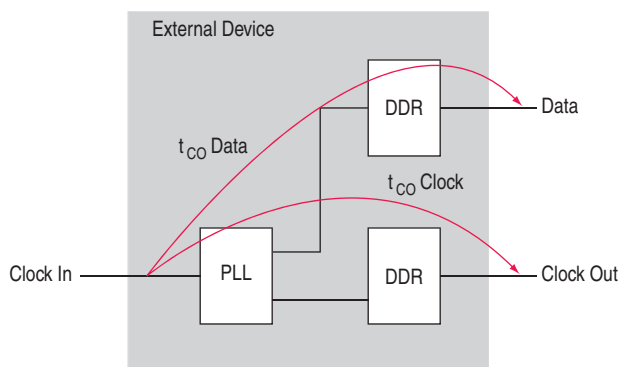
With the system-centric constraint approach, the input delay value includes board delays on the clock and data traces, and clock-to-out times, as shown in [Figure 59](#).

Figure 59. System-Centric Input Delay

If the external device t_{CO} specification is relative to the output clock, as shown in [Figure 60](#), use the specified t_{CO} and $t_{CO\ min}$ values directly to calculate input maximum and minimum delay values.

Figure 60. t_{CO} Relative to Output Clock

If the external device t_{CO} specification is relative to the input clock, as shown in [Figure 61](#), use $(t_{CO\ DATA} - t_{CO\ min\ CLOCK})$ for the t_{CO} value, and $(t_{CO\ min\ DATA} - t_{CO\ CLOCK})$ for the $t_{CO\ min}$ value, to calculate the input maximum and minimum delay values.

Figure 61. t_{CO} Relative to Input Clock **t_{CO} and t_{CO} min Method**

When the external device specification includes t_{CO} and t_{CO} min values, use [Equation 23](#) to calculate the input maximum delay value. The input maximum delay value specifies an upper bound for the input delay because it uses the longest data path and the shortest clock path.

Equation 23.

$$\text{input maximum delay value} = \text{maximum trace delay for data} + t_{CO} \text{ of external device} \\ - \text{minimum trace delay for clock}$$

Write the input maximum delay SDC constraint as shown in [Example 40](#) with the input maximum delay value from [Equation 23](#). This example is suitable for an SDR input.

Example 40.

```
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
```

A DDR input has a duplicate constraint that applies to the falling clock edge, shown in [Example 41](#).

Example 41.

```
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] -clock_fall [get_ports data_in] -add_delay
```

[Equation 24](#) shows the equation to calculate the input minimum delay value. The input minimum delay value specifies a lower bound for the input delay because it uses the shortest data path and the longest clock path.

Equation 24.

$$\text{input minimum delay value} = \text{min trace delay for data} + t_{CO} \text{ min of external device} \\ - \text{max trace delay for clock}$$

Write the input minimum delay SDC constraint as shown in [Example 42](#) with the input minimum delay value from [Equation 24](#) on [page 45](#). This example is suitable for an SDR input.

Example 42.

```
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
```

A DDR input has a duplicate constraint that applies to the falling clock edge, shown in [Example 43](#).

Example 43.

```
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] -clock_fall [get_ports data_in] -add_delay
```

Setup and Hold Method

Some source-synchronous input interfaces may connect to devices that specify setup and hold time parameters for the data output. The setup and hold parameters specify the setup and hold times for the data output with respect to the clock output. In this case, use the following equations to calculate the maximum input delay value. The input maximum delay value specifies an upper bound for the input delay because it uses the longest data path and the shortest clock path.

Equation 25.

$$\text{input maximum delay value} = \text{maximum trace delay for data} + \text{unit interval} \\ - t_{\text{SU}} \text{ of external device} - \text{minimum trace delay for clock}$$

Write the SDC constraint as shown in [Example 44](#) with the input maximum delay value from [Equation 25](#).

Example 44.

```
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
```

A DDR input has a duplicate constraint that applies to the falling clock edge, shown in [Example 45](#).

Example 45.

```
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
set_input_delay -max <input maximum delay value> -clock \
  [get_clocks virtual_clock] -clock_fall [get_ports data_in]
```

[Equation 26](#) shows the equation to calculate the input minimum delay value. The input minimum delay value specifies a lower bound for the input delay because it uses the shortest data path and the longest clock path.

Equation 26.

$$\text{input minimum delay value} = \text{minimum trace delay for data} + t_{\text{H}} \text{ of external device} \\ - \text{maximum trace delay for clock}$$

Write the SDC constraint as shown in [Example 46](#) with the input minimum delay value from [Equation 26](#).

Example 46.

```
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
```

A DDR input has a duplicate constraint that applies to the falling clock edge, shown in [Example 47](#).

Example 47.

```
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] [get_ports data_in]
set_input_delay -min <input minimum delay value> -clock \
  [get_clocks virtual_clock] -clock_fall [get_ports data_in]
```

FPGA-Centric Input Delay Constraints

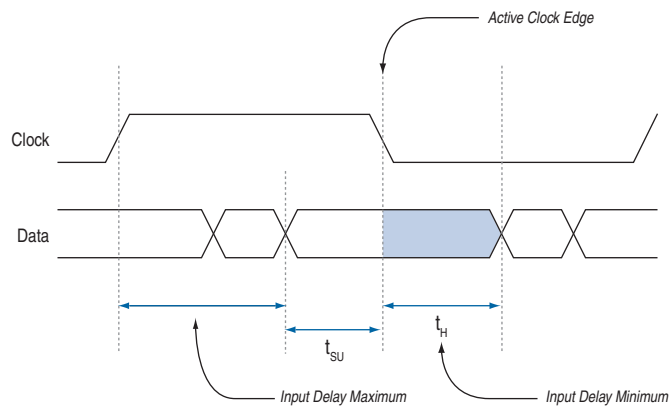
Source-synchronous input requirements are specified at the FPGA boundary in the following two ways:

- “Setup and Hold”
- “Maximum Data Skew” on page 49

Setup and Hold

When your FPGA source-synchronous input interface has setup and hold time requirements, calculate the equivalent maximum and minimum input delays based on the setup and hold requirements.

[Figure 62](#) shows a timing waveform with setup and hold requirements, and the corresponding input maximum and minimum delays. To convert a setup time requirement to an input maximum delay constraint, subtract the setup time from the UI. In a DDR interface, the UI value is half the clock period, because data is transferred on both clock edges. A hold time requirement value is equivalent to an input minimum delay value.

Figure 62. Setup and Hold Requirements with Input Delays

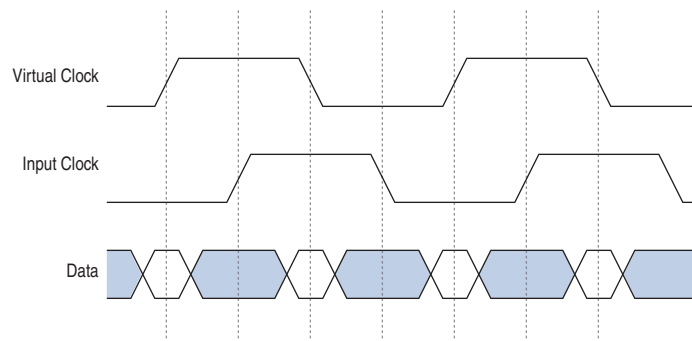
Example 48 shows the input delay constraints.

Example 48.

```
set_input_delay -max [expr <unit interval> - <setup time>] -clock \
  [get_clocks input_clock] -add_delay [get_ports data_in]
set_input_delay -min <hold time> -clock [get_clocks input_clock] \
  -add_delay [get_ports data_in]
```

If you derive input delay constraints from setup and hold requirements using a virtual clock as the input delay clock reference and the input data is not edge-aligned, you must modify the constraint values to compensate for the clock shift.

Figure 63 shows the need for compensation.

Figure 63. Compensating for Center Alignment

The virtual clock has no phase shift, the input clock has a 90° phase shift, and the input data is center-aligned with respect to the input clock (with equal t_{SU} and t_H requirements).

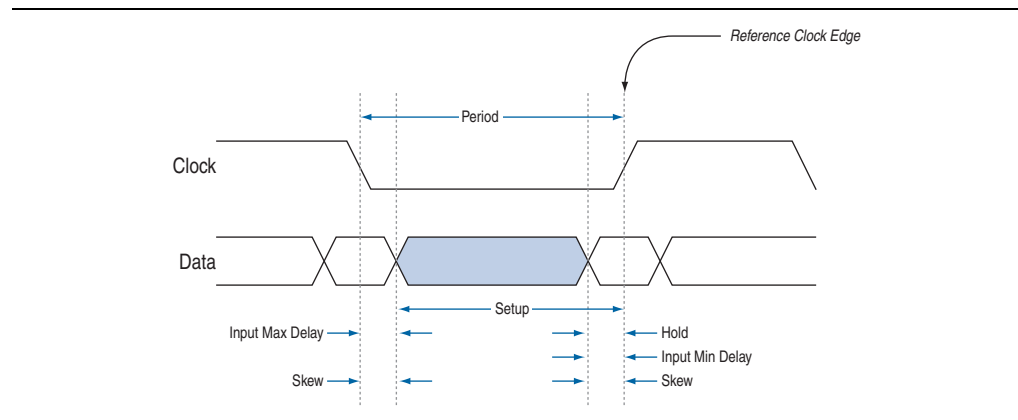
However, when you make input delay constraints with respect to the virtual clock, the setup and hold relationships are not the same as they are with respect to the input clock. The input data is edge-aligned with respect to the virtual clock. The hold requirement with respect to the virtual clock must decrease by the amount of the shift ($\text{period} \div 4$ for a center-aligned DDR interface). The setup requirement with respect to the virtual clock must increase by the amount of the shift ($\text{period} \div 4$ for a center-aligned DDR interface). The input maximum and minimum delay values both increase by the amount of the shift ($\text{period} \div 4$ for a center-aligned DDR interface).

Maximum Data Skew

When you specify the maximum data skew an input can tolerate, you must constrain the minimum data valid window.

Figure 64 has clock and data waveforms that show how the input maximum and minimum delay values are derived from skew requirements.

Figure 64. Clock and Data Waveforms, Input Max and Min Delay Values



The input maximum delay is equal to the skew requirement, because input maximum delay is equivalent to the UI minus the setup value. The input minimum delay is equal to the negative skew requirement value because input minimum delay is equivalent to hold. The hold value is negative because data can change (become invalid) before the reference clock edge.

Using a virtual clock as the clock reference for input delay constraints makes FPGA-centric constraints easy to apply. As Figure 64 shows, the input maximum delay value is the positive value of the skew requirement, and the input minimum delay value is the negative value of the skew requirement.

Example 49 shows the input delay constraints for a DDR interface, using the positive and negative skew values.

Example 49.

```

set_input_delay -max <skew> -clock [get_clocks virtual_clock] \
  [get_ports data_in]
set_input_delay -max <skew> -clock [get_clocks virtual_clock] \
  -clock_fall [get_ports data_in] -add_delay
set_input_delay -min <negative skew> -clock [get_clocks virtual_clock] \
  [get_ports data_in] -add_delay
set_input_delay -min <negative skew> -clock [get_clocks virtual_clock] \
  -clock_fall [get_ports data_in] -add_delay

```

Even if the virtual clock (the source clock driving the output registers in the source device) or the input clock (the clock driving the destination register in the FPGA) have any phase shifts, the input delay values are still the positive and negative skew values.

Input Timing Exceptions

The following sections describe different alignment and capture edge combinations, and show any additional timing exceptions or adjusted constraints that are appropriate for each combination. For additional timing exceptions or constraint modifications that are necessary for correct operation, refer to the section that corresponds to the operation of your interface:

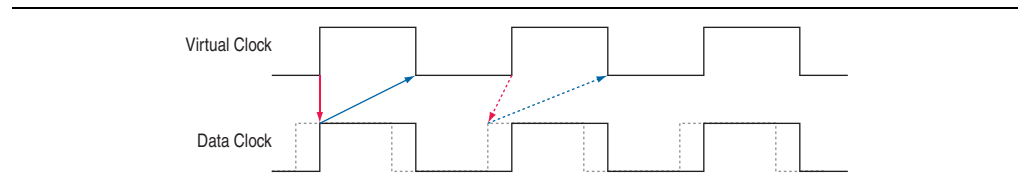
- “Same-Edge Capture Edge-Aligned Input”
- “Same-Edge Capture Center-Aligned Input” on page 51
- “Opposite-Edge Capture Edge-Aligned Input” on page 52
- “Opposite-Edge Capture Center-Aligned Input” on page 52

Same-Edge Capture Edge-Aligned Input

In an edge-aligned input configuration, the constraints specify the clock and data relationship at the FPGA inputs. The clock signal is often delayed in the FPGA (typically with a PLL) so that it can meet the micro-timing parameters (ut_{SU} and ut_H) of the FPGA registers used to capture the data.

Figure 65 shows the setup and hold relationships that must be analyzed if your input circuit does not include a PLL, or includes a PLL that clocks the data capture registers with a phase shift less than or equal to 0. The dashed line shows the desired setup and hold relationships when there is a phase shift less than 0. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 65. Desired Setup and Hold for Edge-Aligned Input



In a same-edge capture configuration, the setup relationship is between the same launch and latch edges. The hold relationship is between opposite edges, with the launch edge occurring one period after the latch edge. Use the destination setup multicycle exceptions in [Example 50](#) to ensure that the proper setup and hold relationships are analyzed.

Example 50.

```
set_multicycle_path -setup -end -rise_from [get_clocks virtual_clk] \
  -rise_to [get_clocks data_clk] 0
set_multicycle_path -setup -end -fall_from [get_clocks virtual_clk] \
  -fall_to [get_clocks data_clk] 0
```

Use the false path exceptions in [Example 51](#) to prevent timing analysis on opposite-edge transfers.

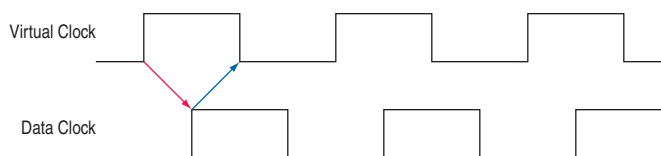
Example 51.

```
set_false_path -setup -fall_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -setup -rise_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
set_false_path -hold -rise_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -hold -fall_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
```

Same-Edge Capture Center-Aligned Input

[Figure 66](#) shows the setup and hold relationships that must be analyzed if data and clock are center-aligned, or your input circuit includes a PLL that clocks the data capture registers with a phase shift greater than 0. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 66. Default Setup and Hold for Center-Aligned Input



In this case the default setup and hold relationships are the ones that must be analyzed for DDR inputs. The only exceptions necessary are the false path exceptions to prevent timing analysis on opposite-edge transfers, shown in [Example 52](#).

Example 52.

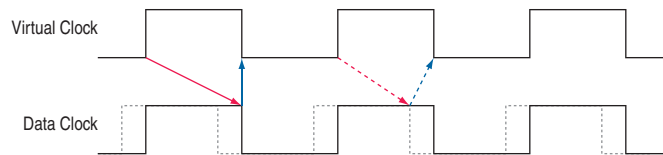
```

set_false_path -setup -fall_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -setup -rise_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
set_false_path -hold -rise_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -hold -fall_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]

```

Opposite-Edge Capture Edge-Aligned Input

Figure 67 shows the setup and hold relationships that must be analyzed for opposite-edge capture if your circuit does not include a PLL, or includes a PLL that clocks the data capture registers with a phase shift less than or equal to 0. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 67. Desired Setup and Hold for Edge-Aligned Input

In this case, false path exceptions are necessary, because the setup and hold relationships that must be analyzed are not the ones that are analyzed by default for DDR inputs. Add the false path exceptions in Example 53 to prevent timing analysis on same-edge transfers.

Example 53.

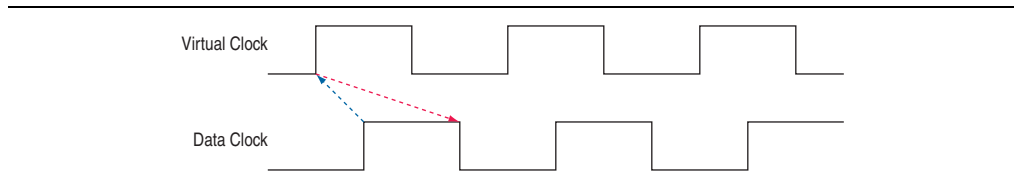
```

set_false_path -setup -rise_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -setup -fall_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
set_false_path -hold -fall_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -hold -rise_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]

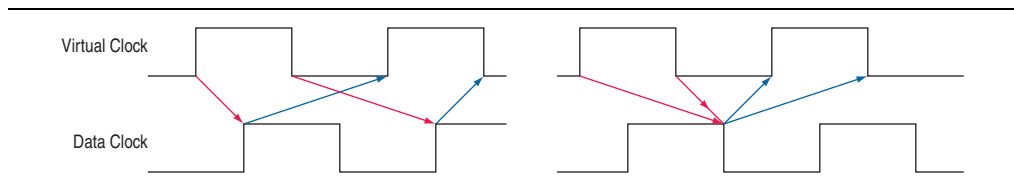
```

Opposite-Edge Capture Center-Aligned Input

Figure 68 shows the setup and hold relationships that must be analyzed for opposite-edge capture, if data and clock are center-aligned, or your input circuit includes a PLL that clocks the data capture registers with a phase shift greater than zero. Red arrows indicate setup relationships, and blue arrows indicate hold relationships.

Figure 68. Desired Setup and Hold for Opposite-Edge Capture, Center-Aligned Input

Opposite-edge capture center-aligned inputs require a complex combination of destination multicycle exceptions and false path exceptions to change the default timing analysis behavior to the desired behavior. Without any multicycle or false path exceptions, [Figure 69](#) shows the timing relationships that are analyzed by default. Red arrows indicate setup relationships, and blue arrows indicate hold relationships. The first waveform shows the setup relationships for rise-rise and rise-fall transfers in red, and the hold relationships in blue. The second waveform shows the setup relationships for fall-rise and fall-fall transfers in red, and the hold relationships in blue.

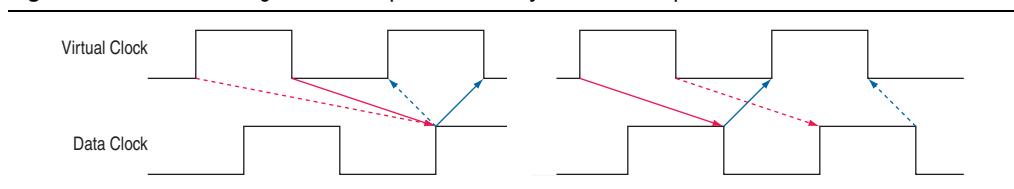
Figure 69. Default Timing Relationships Without Multicycle or False Path Exceptions

Use the multicycle exceptions shown in [Example 54](#) to change the setup and hold relationships for the same-edge data transfer.

Example 54.

```
set_multicycle_path -setup -end -rise_from [get_clocks virtual_clk] \
  -rise_to [get_clocks data_clk] 2
set_multicycle_path -setup -end -fall_from [get_clocks virtual_clk] \
  -fall_to [get_clocks data_clk] 2
```

[Figure 70](#) shows the setup and hold relationships after the multicycle exceptions are applied. The dashed lines show the relationships after the destination setup multicycle exceptions are applied. Red arrows indicate setup relationships, and blue arrows indicate hold relationships. The first waveform shows the setup relationships for rise-rise and rise-fall transfers in red, and the hold relationships in blue. The second waveform shows the setup relationships for fall-rise and fall-fall transfers in red, and the hold relationships in blue.

Figure 70. Default Timing Relationships with Multicycle Path Exceptions

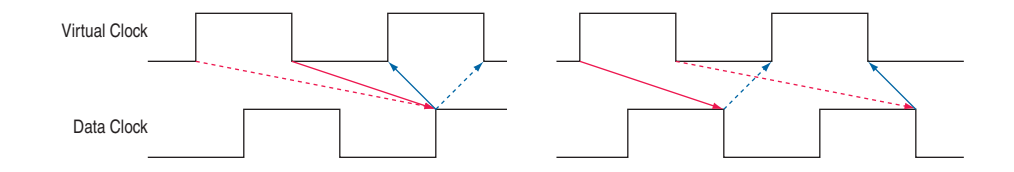
Finally, add the false path exceptions shown in [Example 55](#) to cut same-edge transfers.

Example 55.

```
set_false_path -setup -rise_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -setup -fall_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
set_false_path -hold -fall_from [get_clocks virtual_clk] -rise_to \
  [get_clocks data_clk]
set_false_path -hold -rise_from [get_clocks virtual_clk] -fall_to \
  [get_clocks data_clk]
```

[Figure 71](#) indicates the paths that are cut by the false path exceptions in [Example 55](#) with dotted lines. It indicates the paths that are analyzed with solid lines. Red arrows indicate setup relationships, and blue arrows indicate hold relationships. The first waveform shows the setup relationships for rise-rise and rise-fall transfers in red, and the hold relationships in blue. The second waveform shows the setup relationships for fall-rise and fall-fall transfers in red, and the hold relationships in blue.

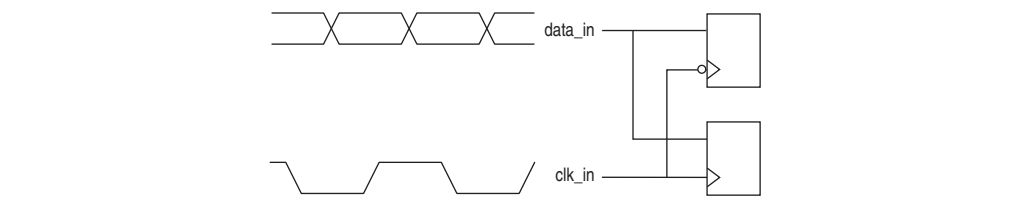
Figure 71. Default Timing Relationships with False Path Exceptions



Timing Analysis

[Figure 72](#) shows a simple source-synchronous DDR input design example used to illustrate timing analysis concepts.

Figure 72. Simple Source-Synchronous Input



The clock sent to the device has a 90° phase shift to center align it with the data, so no shift is necessary in the input circuit. There is a +/-100 ps skew requirement for the data. The input is constrained with the constraints shown in [Example 56](#).

Example 56.

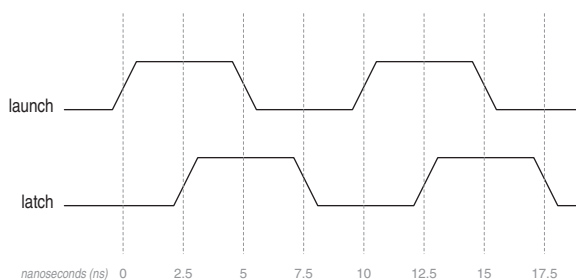
```

create_clock -name virtual_source -period 10.000
create_clock -name input_clock -period 10.000 -waveform { 2.5 7.5 } \
  [get_ports clk_in]
set_input_delay -clock virtual_source -max 0.100 [get_ports data_in]
set_input_delay -clock virtual_source -min -0.100 [get_ports data_in] \
  -add_delay
set_input_delay -clock virtual_source -clock_fall -max 0.100 \
  [get_ports data_in] -add_delay
set_input_delay -clock virtual_source -clock_fall -min -0.100 \
  [get_ports data_in] -add_delay

```

Figure 73 shows the clock waveforms for the launch and latch clocks.

Figure 73. Clock Waveforms

**Report Timing**

To report timing for the input, use the `report_timing` command with the `-from_clock` and `-to_clock` options. Use the name of the virtual clock for the `-from_clock` option, and the name of the clock driving the input registers for the `-to_clock` option. For example, use the two commands in [Example 57](#) to report setup and hold timing for the circuit described previously.

Example 57.

```

report_timing -from_clock virtual_source -to_clock input_clock -setup
report_timing -from_clock virtual_source -to_clock input_clock -hold

```

Perform setup and hold analysis at all available timing corners to ensure the design meets timing. The slack numbers reported by each `report_timing` command indicate by how much the data meets its timing requirement. A negative value indicates that the constraint is not satisfied. Add the worst-case setup and hold slacks to determine the interface margin.

Document Revision History

Table 2 shows the revision history for this application note.

Table 2. Document Revision History

Date and Document Version	Changes Made
May 2016 v2.8	Minor change to Figure 61
January 2015 v2.7	Moved start of t_{SU} interval in Figure 62
March 2014 v2.6	Changed “outclk” to “muxsel” in Example 6 and Example 7
February 2014 v2.5	Minor editorial change in “Interface Constraints” section
December 2013 v2.4	Fixed error in Example 2
June 2010 v2.3	Minor technical changes to Figure 45 and Figure 50
March 2010 v2.2	Minor technical and editorial changes
November 2009 v2.1	Updated “Introduction”
December 2007 v2.0	Major revision; the entire application note was changed
May 2007 v1.0	Initial release



101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support

Copyright © 2015 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.



I.S. EN ISO 9001