Timing Closure User Guide

UG612 (v 14.3) October 16, 2012

This document applies to the following software versions: ISE Design Suite 14.3 through 14.7

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	
10/16/20112	14.3	• Removed sentence regarding two unrelated clocks. • Removed figure "Two Unrelated Clocks Entering the FPGA Device Through Separate External Pins" and accompanying text. • Removed figure "Input Clock Goes to a DCM Example" and accompanying text. • Removed section "Two Related Clocks Entering the FPGA Device Through Separate External Pins" and accompanying figure. • Removed section "Period Constraint Syntax." • Removed section "Asynchronous Design Technique Example." • Removed section "Constraining Maximum Data Path Delay." • Removed section "Two Unrelated Clocks Entering Through Separate External Pins." • Replaced various references to DCM with DCM/PLL/MMCM. • Added sentence "The CLOCK_DEDICATED_ROUTE constraint applies to the INSTANCE PIN or NET." • Added sentence "Setup and hold analysis is done during timing analysis for Virtex-5 and newer devices." • Added new paragraph beginning "AREA_GROUP is attached to logical blocks in the design ..." • Added new paragraph beginning " The component switching limit is a device or silicon limit ..."
01/18/2012	13.4	• Moved timing constraints information from the *Constraints Guide (UG625)* to this guide.

Table of Contents

Chapter 1

Introduction

The *Timing Closure User Guide (UG612)* addresses timing closure in high-performance applications. The Guide is designed for all FPGA designers, from beginners to advanced.

The high performance of today's Xilinx® devices can overcome the speed limitations of other technologies and older devices. Designs that formerly only fit or ran at high clock frequencies in an ASIC device are finding their way into Xilinx FPGA devices. Designers must have a proven methodology for obtaining their performance objectives.

This Guide discusses:

- The fundamentals of timing constraints.
- The ability to group elements and provide a better understanding of the constraint system tool.
- The analysis of the basic constraints, with clock skew and clock uncertainty.
- Specifying timing constraints in the Xilinx Synthesis Technology (XST).
- Specifying timing constraints in Synplify.

Chapter 2

Timing Constraint Methodology

You must have a proven methodology in order to meet your design objectives. This chapter discusses how to:

- Understand your design requirements
- Constrain your design to meet these requirements

Before starting a design, you must understand:

- The performance requirements of the system
- The features of the target device

This knowledge allows you to use proper coding techniques using the features of the device to give the best performance.

The FPGA device requirements depend on the system and the upstream and downstream devices. Once the interfaces to the FPGA device are known, the internal requirements can be outlined. How to meet these requirements depends on the device and its features.

You should understand:

- The device clocking structure
- RAM and DSP blocks
- Any hard IP contained within the device

For more information, see the device user guide cited in Appendix A, Additional Resources.

Timing constraints communicate all design requirements to the implementation tools. This also implies that all paths are covered by the appropriate constraint. This chapter provides general guidelines that explain the strategy for identifying and constraining the most common timing paths in FPGA devices as efficiently as possible.

Basic Constraints Methodology

In order to ensure a valid design, the timing requirements for all paths must be communicated to the implementation tools. The timing requirements can be broken down into several global categories based on the type of path that is to be covered. The most common types of path categories include:

- Input paths
- Register-to-register paths
- Output paths
- Path specific exceptions

A Xilinx® timing constraint is associated with each of these global category types. The most efficient way to specify these constraints is to begin with global constraints, then add path specific exceptions as needed. In many cases, only the global constraints are required.

The FPGA implementation tools are driven by the specified timing requirements. The tools assign device resources, and expends the appropriate amount of effort necessary to ensure that the timing requirements are met. However, when a requirement is over-constrained (or specified as a value greater than the design requirement), the effort to meet this constraint increases significantly, and results in increased memory use and tool runtime. In addition, over-constraining can degrade performance for not only that particular constraint, but for other constraints as well. For this reason, Xilinx recommends that you specify the constraint values using the actual design requirements.

The method of applying constraints given in this guide uses User Constraints File (UCF) constraint syntax examples. This format highlights the constraints syntax that conveys the design requirements. However, the easiest way to enter design constraints is to use Constraints Editor, which:

- Provides a unified location in which to manage all timing constraints associated with a design.
- Provides assistance in creating timing constraints from the design requirements.

Timing requirements fall into several global categories depending on the type of path to be covered.

The most common types of path categories include:

- Input paths
- Synchronous element to synchronous element paths
- Path specific exceptions
- Output paths

A Xilinx® timing constraint is associated with each global constraint type. To efficiently specify these constraints:

- Begin with global constraints.
- Add path specific exceptions as needed.

Only global constraints are required in many cases.

Over-Constraining

The FPGA device implementation tools are driven by the specified timing requirements. They assign device resources, and expend the appropriate amount of effort to meet timing requirements.

The effort spent by the tools to meet this constraint increases significantly when a requirement is:

- Over-constrained, or
- Specified as a value with a greater frequency than the design requirement

This extra effort results in:

- Increased memory use
- Increased tool runtime

Over-constraining can result in loss of performance for both:

- The constraint in question
- Other constraints

For this reason, Xilinx recommends that you specify the constraint values using actual design requirements.

If a design is correctly constrained, including INPUT_JITTER and SYSTEM_JITTER requirements, over-constraining is not necessary.

Commenting the Design File

Always comment the constraints file. This allows other designers to understand why each constraint is used.

Include in your comments:

- The constraint source
- Whether the Period constraint is based on an external clock

Using Constraints Editor

This Guide uses XST Constraint File (XCF) syntax examples. This format passes the design requirements to the implementation tools.

However, the easiest way to enter design constraints is to use Constraints Editor.

- Constraints Editor provides a unified location in which to manage all timing constraints.
- Constraints Editor helps you create timing constraints from the design requirements using XCF syntax.

Input Timing Constraints

This section discuss the methodology for specifying input timing constraints. The input timing constraints cover the data path from the external pin or pad of the package of the FPGA device to the internal synchronous element or register that captures that data. The OFFSET IN constraint is used to specify the input timing requirements.

The input timing requirements are based upon the type (source or system synchronous) and the data rate (SDR or DDR) of the interface. These categories include:

- System Synchronous Input
- Timing Diagram for Ideal System Synchronous SDR Interface Example
- Source Synchronous Inputs
- Timing Diagram for Ideal Source Synchronous DDR Interface Example

The OFFSET IN constraint defines the relationship between the data and the clock edge used to capture that data at the pin or pads of the FPGA device. The analysis of the OFFSET IN includes all internal factors affecting the delay of the clock signal and the data signal. These factors include:

- Frequency and phase transformation of the clock
- Clock Uncertainties
- Data Delay Adjustments

The input clock uncertainty and clock arrival times are derived from the PERIOD constraint associated with the interface clock referenced in the OFFSET IN constraint. For more information on the Period constraint and adding Input Jitter, see Period Constraints in Chapter 3, Timing Constraint Principles.

System Synchronous Inputs

In a system synchronous interface, a common system clock both transfers and captures the data. This interface uses a common system clock. The board trace delays and clock skew limit the operating frequency of the interface.

The lower frequency also results in the system synchronous input interface typically being an SDR application.

Simplified System Synchronous Interface with Associated SDR Timing

In the system synchronous SDR application example shown in the following figure, the data is:

1. Transmitted from the source device on one rising clock edge, and
2. Captured in the FPGA device on the next rising clock edge.

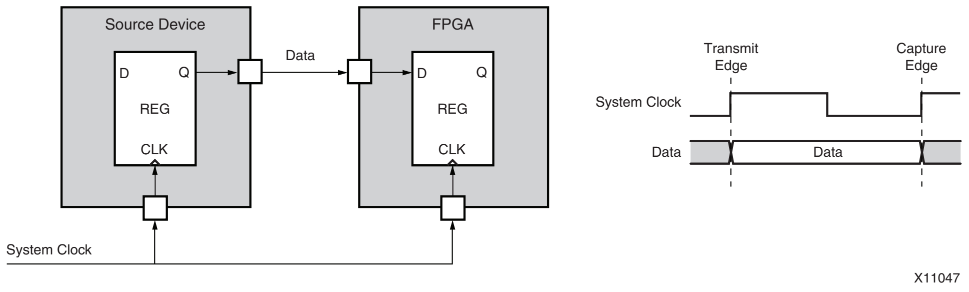


Figure 2-1: **Simplified System Synchronous Interface with Associated SDR Timing**

The global Offset In constraint is the most efficient way to specify the input timing for a system synchronous interface. In this method, one Offset In constraint is defined for each system synchronous input interface clock. This single constraint covers the paths of all input data bits that are captured in synchronous elements triggered by the specified input clock.

Specifying Input Timing

To specify the input timing:

- Define the clock Period constraint for the input clock associated with the interface
- Define the global Offset In constraint for the interface

Ideal System Synchronous SDR Interface

The following figure shows a timing diagram for an ideal System Synchronous SDR interface.

- The interface has a clock period of 5 ns.
- The data for both bits of the bus remains valid for the entire period.

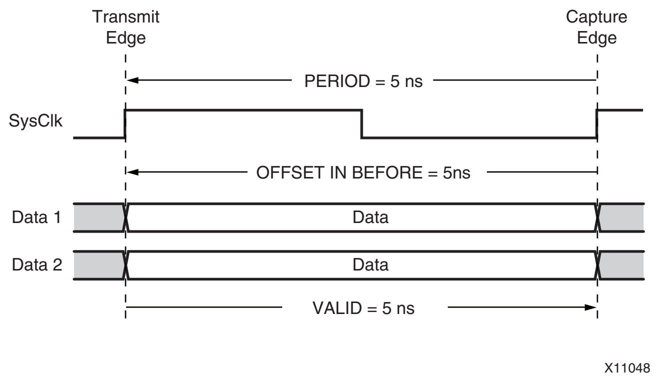


Figure 2-2: **Timing Diagram for an Ideal System Synchronous SDR Interface**

Global Offset In Constraint

The global Offset In constraint is:

```
OFFSET = IN value VALID value BEFORE clock;
```

In the Offset In constraint, **`OFFSET=IN`** ***`<value>`*** determines the time from the capturing clock edge to the time in which data first becomes valid. In this system synchronous example, the data becomes valid 5 ns before the capturing clock edge. In the Offset In constraint, the **VALID *<value>*** determines the duration in which data remains valid. In this example, the data remains valid for 5 ns.

For this example, the complete Offset In specification with associated Period constraint is:

```
NET "SysClk" TNM_NET = "SysClk";
TIMESPEC "TS_SysClk" = PERIOD "SysClk" 5 ns HIGH 50%;
OFFSET = IN 5 ns VALID 5 ns BEFORE "SysClk";
```

This global constraint covers both the data bits of the bus:

- data1
- data2

Source Synchronous Inputs

In a source synchronous input interface, a clock is regenerated and transmitted along with the data from the source device along similar board traces. This clock captures the data in the FPGA device.

The board trace delays and board skew no longer limit the operating frequency of the interface. The higher frequency also results in the source synchronous input interface typically being a dual data rate (DDR) application.

Simplified Source Synchronous Input Interface with Associated DDR Timing

In the source synchronous DDR application example shown in the following figure, unique data is:

1. Transmitted from the source device on both the rising and falling clock edges, and
2. Captured in the FPGA device using the regenerated clock.

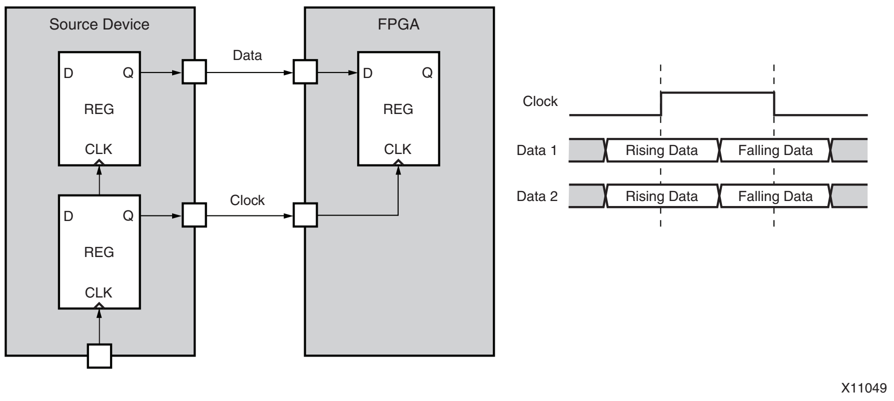


Figure 2-3: **Simplified Source Synchronous Input Interface with Associated DDR Timing**

The global Offset In constraint is the most efficient way to specify the input timing for a source synchronous interface. In the DDR interface, one Offset In constraint is defined for each edge of the input interface clock. These constraints cover the paths of all input data bits that are captured in registers triggered by the specified input clock edge.

Specifying Input Timing

To specify the input timing, define the constraints as shown in the following table.

Table 2-1: **Specifying Input Timing**

Constraint	Define For
Clock Period	Input clock
Global Offset In	Rising edge
Global Offset In	Falling edge

Ideal Source Synchronous DDR Interface

The following figure shows a timing diagram for an ideal Source Synchronous DDR interface.

- The interface has a clock period of 5 ns with a 50/50 duty cycle.
- The data for both bits of the bus remains valid for the entire ½ period.

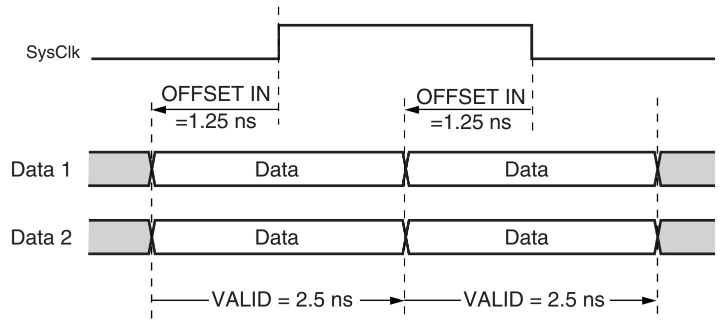


Figure 2-4: **Timing Diagram for Ideal Source Synchronous DDR**

Global Offset In Constraint

The global Offset In constraint for the DDR case is:

```
OFFSET = IN value VALID value BEFORE clock RISING;
OFFSET = IN value VALID value BEFORE clock FALLING;
```

In the Offset In constraint, **OFFSET=IN** <*value*> determines the time from the capturing clock edge in which data first becomes valid.

In this source synchronous input example:

- The rising data becomes valid 1.25 ns before the rising clock edge.
- The falling data also becomes valid 1.25 ns before the falling clock edge.

In the Offset In constraint, the **VALID** <*value*> determines the duration in which data remains valid. In this example, both the rising and falling data remains valid for 2.5 ns.

For this example, the complete Offset In specification with an associated Period constraint is:

```
NET "SysClk" TNM_NET = "SysClk";
TIMESPEC "TS_SysClk" = PERIOD "SysClk" 5 ns HIGH 50%;

OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SysClk" RISING;
OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SysClk" FALLING;
```

This global constraint covers both the data bits of the bus:

- data1
- data2

UCF Source Synchronous DDR Edge Aligned Example

The Source Synchronous Dual Data Rate (DDR) Edge aligned case consists of an interface where the clock is sent from the transmitting device edge aligned with the data to the FPGA. In a dual data rate interface, data is captured with both the rising and falling clock edges. In the DDR case, separate Offset In constraints must be defined for the rising and falling clock edge registers capturing the data. The use of the RISING and FALLING keywords with the Offset In constraint simplifies this task.

Example Waveform

In this example a dual data rate interface is shown with a clock period of 5 ns and 50/50 duty cycle. The rising and falling data is valid for 2 ns and is centered in the high and low portion of the clock waveform. This results in a 250 ps margin before and after data valid window.

Rising Edge Constraints

The rising edge Offset In constraint defines the time that the data becomes valid prior to rising clock edge used to capture the data. In this example, the data becomes valid 250 ps after the rising clock edge. This results in an OFFSET IN BEFORE value of -250 ps with the value negative because it begins after the clock edge. Once the data begins, it remains valid for 2 ns. This results in a VALID value of 2 ns. The RISING keyword is used with this constraint to indicate that the constraint applies to only the rising edge synchronous elements, and that the OFFSET IN BEFORE value is specified to the rising clock edge.

Falling Edge Constraints

The falling edge Offset In constraint defines the time that the data becomes valid prior to falling clock edge used to capture the data. In this example, the data becomes valid 250 ps after the falling clock edge. This results in an OFFSET IN BEFORE value of -250 ps with the value negative because it begins after the clock edge. Once the data begins, it remains valid for 2 ns. This results in a VALID value of 2 ns. The FALLING keyword is used with this constraint to indicate that the constraint applies to only the falling edge synchronous elements, and that the OFFSET IN BEFORE value is specified to the falling clock edge.

UCF Syntax

The complete UCF syntax of the clock PERIOD and Offset In constraint for the example is shown below.

```
NET “clock” TNM_NET = CLK; TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%; OFFSET = IN -250
ps VALID 2 ns BEFORE clock RISING; OFFSET = IN -250 ps VALID 2 ns BEFORE clock FALLING
```

UCF Source Synchronous DDR Center Aligned Example

The Source Synchronous Dual Data Rate (DDR) Center aligned case consists of an interface where the clock is sent from the transmitting device aligned with the center of the data. In a dual data rate interface, data is captured with both the rising and falling clock edges. In the DDR case, separate Offset In constraints must be defined for the rising and falling clock edge registers capturing the data. Using the RISING and FALLING keywords with the Offset In constraint simplifies this task.

Example Waveform

In this example a dual data rate interface is shown with a clock period of 5 ns and 50/50 duty cycle. The rising and falling data is valid for 2 ns and is centered over the high and low clock edges. This results in a 250 ps margin before and after data valid window.

Rising Edge Constraints

The rising edge Offset In constraint defines the time that the data becomes valid prior to rising clock edge used to capture the data. In this example, the data becomes valid 1 ns before the rising clock edge. This results in an OFFSET IN BEFORE value of 1 ns with the value positive because it begins before the clock edge.

Once the data begins, it remains valid for 2 ns. This results in a VALID value of 2 ns. The RISING keyword is used with this constraint to indicate that the constraint applies to only the rising edge synchronous elements, and that the OFFSET IN BEFORE value is specified to the rising clock edge.

Falling Edge Constraints

The falling edge Offset In constraint defines the time that the data becomes valid prior to falling clock edge used to capture the data. In this example, the data becomes valid 1 ns before the falling clock edge. This results in an OFFSET IN BEFORE value of 1 ns with the value positive because it begins before the clock edge.

Once the data begins, it remains valid for 2 ns. This results in a VALID value of 2 ns. The FALLING keyword is used with this constraint to indicate that the constraint applies to only the falling edge synchronous elements, and that the OFFSET IN BEFORE value is specified to the falling clock edge.

UCF Syntax

The complete UCF syntax of the clock PERIOD and Offset In constraint for the example is shown below.

```
NET "clock" TNM_NET = CLK; TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%; OFFSET = IN 1 ns
VALID 2 ns BEFORE clock RISING; OFFSET = IN 1 ns VALID 2 ns BEFORE clock FALLING;
```

UCF System Synchronous SDR Examples

The System Synchronous Single Data Rate (SDR) case consists of an interface where the clock is sent from the transmitting device with one clock edge and captured by the FPGA with the next clock edge. In the single data rate interface data is sent once per clock cycle and requires only one Offset In constraint.

Example Waveform

In this example a single data rate interface is shown with a clock period of 5 ns and 50/50 duty cycle. The data is valid for 4 ns and begins 500 ps after the transmitting clock edge.

Input Constraints

The Offset In constraint defines the time that the data becomes valid prior to rising clock edge used to capture the data. In this example, the data becomes valid 500 ps after the transmitting clock edge, or 4.5 ns before the clock edge used to capture the data. This results in an OFFSET IN BEFORE value of 4.5 ns with the value positive because it begins before the clock edge. Once the data begins, it remains valid for 4 ns. This results in a VALID value of 4 ns.

UCF Syntax

The complete UCF syntax of the clock PERIOD and Offset In constraint for the example is shown below.

```
NET “clock” TNM_NET = CLK; TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%; OFFSET = IN 4.5 ns
VALID 4 ns BEFORE clock;
```

Register-To-Register Timing Constraints

This section discuss the methodology for the period specification of register-to-register synchronous path timing requirements (Period constraint).

The Period constraint:

- Defines the timing of the clock domains.
- Covers the synchronous data path between internal registers.
- Analyzes the paths within a single clock domain.
- Analyzes all paths between related clock domains as well.
- Takes into account all frequency, phase, and uncertainty differences between the clock domains during analysis.

The application and methodology for constraint synchronous clock domains falls under several common categories. These categories include

- Automatically Related Synchronous DLL, DCM, PLL, and MMCM Clock Domainss
- Manually Related Synchronous Clock Domains
- Asynchronous Clock Domains

By allowing the tools to automatically create clock relationships for DCM, PLL, and MMCM output clocks, and manually defining relationships for externally related clocks, all synchronous cross-clock-domain paths are covered by the appropriate constraints, and properly analyzed. The proper application of Period constraints that follow this methodology eliminates the need for additional cross-clock-domain constraints.

For more information, see Period Constraints in Chapter 3, Timing Constraint Principles.

Automatically Related Synchronous DLL, DCM, PLL, and MMCM Clock Domains

The most common type of clock circuit is one in which:

- The input clock is fed into a DCM, PLL, or MMCM.
- The outputs are used to clock the synchronous paths in the device.

The recommended methodology is to define a Period constraint on the input clock to the DCM, PLL, or MMCM.

By placing PERIOD on the input clock, the tools automatically:

- Derive a new PERIOD for each of the DCM, PLL, or MMCM output clocks.
- Determine the clock relationships between the output clock domains.
- Perform an analysis for any paths between these synchronous domains.

Example

In this example, the input clock goes to a DCM. The following figure shows the circuit for this example.

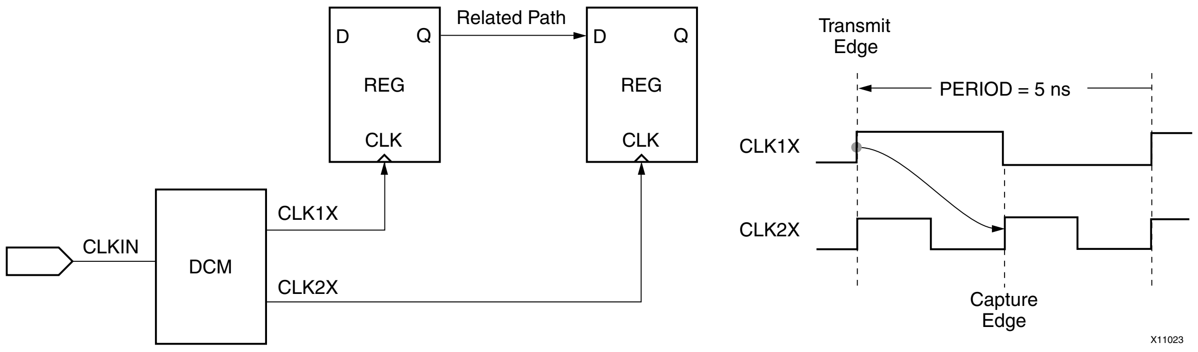


Figure 2-5: **Input Clock Goes to a DCM**

The Period constraint syntax for this example is:

```
NET “ClockName” TNM_NET = “TNM_NET_Name”;
TIMESPEC “TS_name” = PERIOD “TNM_NET_Name” PeriodValue HIGH HighValue%;
```

For PERIOD, the PeriodValue defines the duration of the clock period. In this case, the input clock to the DCM has a period of 5 ns. The HighValue of the PERIOD constraint defines the percent of the clock waveform that is HIGH.

In this example, the waveform has a 50/50 duty cycle resulting in a HighValue of 50%. The syntax for this example is:

```
NET “ClkIn” TNM_NET = “ClkIn”;
TIMESPEC “TS_ClkIn” = PERIOD “ClkIn” 5 ns HIGH 50%;
```

Based on the input clock PERIOD constraint given above, the DCM automatically:

- Creates two output clock constraints for the DCM outputs.
- Performs analysis between the two domains.

Period Constraint Syntax

The Period constraint syntax for this example is:

```
NET "ClockName" TNM_NET = "TNM_NET_Name";
TIMESPEC "TS_name" = PERIOD "TNM_NET_Name" PeriodValue HIGH HighValue%;
```

Table 2-2: **Period Constraint Values**

Value	Defines	This Example
PeriodValue	Duration of the clock period	Input clock to the DCM has a period of 5 ns
HighValue	Percent of the clock waveform that is High	Waveform has a 50/50 duty cycle resulting in a HighValue of 50%

The syntax for this example is:

```
NET "ClkIn" TNM_NET = "ClkIn";
TIMESPEC "TS_ClkIn" = PERIOD "ClkIn" 5 ns HIGH 50%;
```

Based on the input clock Period constraint given above, the DCM:

- Creates two output clock constraints for the DCM outputs.
- Performs analysis between the two domains.

Manually Related Synchronous Clock Domains

In some cases, the tools cannot automatically determine the relationship between synchronous clock domains (for example, when related clocks enter the FPGA device on separate pins). In this case, the recommended constraint methodology is to:

- Create separate Period constraints for both input clocks.
- Define a manual relationship between the clocks.

Once the manual relationship is defined:

- All paths between the two synchronous domains are automatically analyzed.
- All frequency, phase, and uncertainty information is automatically taken into account.

The Xilinx constraints system allows for complex manual relationships to be defined between clock domains using PERIOD. This manual relationship can include clock frequency and phase transformations. The methodology for this process is:

1. Define PERIOD for the primary clock.
2. Define the PERIOD constraint for the related clock using the first PERIOD constraint as a reference.

Example

In this example, two related clocks enter the FPGA device through separate external pins.

- The first clock (CLK1X) is the primary clock.
- The second clock (CLK2X180) is the related clock.

The circuit for this example is shown in the following figure.

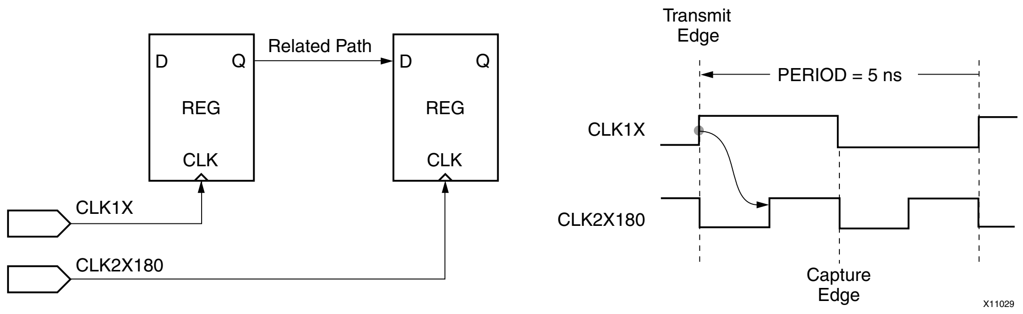


Figure 2-6: **Two Related Clocks Entering the FPGA Device Through Separate External Pins**

The PERIOD syntax for this example is:

```
NET “PrimaryClock” TNM_NET = “TNM_Primary”;
NET “RelatedClock” TNM_NET = “TNM_Related”;
TIMESPEC “TS_primary” = PERIOD “TNM_Primary” PeriodValue HIGH
HighValue%;
TIMESPEC “TS_related” = PERIOD “TNM_Related” TS_Primary_relation PHASE
value;
```

In the related PERIOD definition, the PERIOD value is defined as a time unit (period) relationship to the primary clock. The relationship is expressed in terms of the primary clock TIMESPEC.

In this example CLK2X180 operates at twice the frequency of CLK1X which results in a PERIOD relationship of ½. In the related PERIOD definition, the PHASE value defines the difference in time between the rising clock edge of the source clock and the related clock. In this example, the CLK2X180 clock is 180 degrees shifted, so the rising edge begins 1.25 ns after the rising edge of the primary clock.

The syntax for this example is:

```
NET“Clk1X"TNM_NET=“Clk1X";
NET“Clk2X180"TNM_NET=“Clk2X180";
TIMESPEC"TS_Clk1X"=PERIOD"Clk1X7 5ns;
TIMESPEC"TS_Clk2X180"=PERIOD"Clk2X180“TS_Clk1X/2PHAS2 +1.25ns;
```

Asynchronous Clock Domains

Asynchronous clock domains are those in which the transmit and capture clocks bear no frequency or phase relationship. Because the clocks are not related, it is not possible to determine the final relationship for setup and hold time analysis. For this reason, Xilinx recommends using proper asynchronous design techniques to ensure that the data is successfully captured. However, while not required, in some cases designers wish to constrain the maximum data path delay in isolation without regard to clock path frequency or phase relationship.

The Xilinx constraints system allows for the constraining of the maximum data path delay without regard to source and destination clock frequency and phase relationship.

This requirement is specified using From-To with the DATAPATHONLY keyword.

The methodology for this process is:

1. Define a time group for the source registers
2. Define a time group for the destination registers
3. Define the maximum delay of the net using From-To between the two time groups with the DATAPATHONLY keyword.

For more information on using From-To with the DATAPATHONLY keyword, see From-To.

In some cases, the relationship between synchronous clock domains can not be automatically determined - for example, when related clocks enter the FPGA device on separate pins. In this case, Xilinx recommends that you:

- Define a separate Period constraint for each input clock.
- Define a manual relationship between the clocks.

Path Analysis

Once you define the manual relationship, the tools analyze all paths between the two synchronous domains. The analysis takes into account all frequency, phase, and uncertainty information.

Defining Complex Manual Relationships

The Xilinx constraint system allows you to define complex manual relationships among clock domains using the Period constraint including:

- Clock frequency
- Phase transformations

To define complex manual relationships among clock domains using the Period constraint, define the Period constraint for:

- The primary clock
- The related clock using the first Period constraint as a reference

For more information on using the Period constraint to define clock relationships, see Period Constraints in Chapter 3, Timing Constraint Principles.

Output Timing Constraints

This section discuss the methodology for specify output timing constraints. The output timing constraints cover the data from the internal synchronous element or register to the external pin or pad of the package of the FPGA device.

The OFFSET OUT constraint is used to specify the output timing requirements. The output timing requirements are based upon the type (source or system synchronous) and the data rate (SDR or DDR) of the interface.

The OFFSET OUT constraint defines the relationship between the data and the clock edge used to launch that data at the pin or pads of the FPGA device. The analysis of the OFFSET OUT includes all internal factors affecting the delay of the clock signal and the data signal. These factors include:

- Frequency and phase transformation of the clock
- Clock Uncertainties
- Data Delay Adjustments

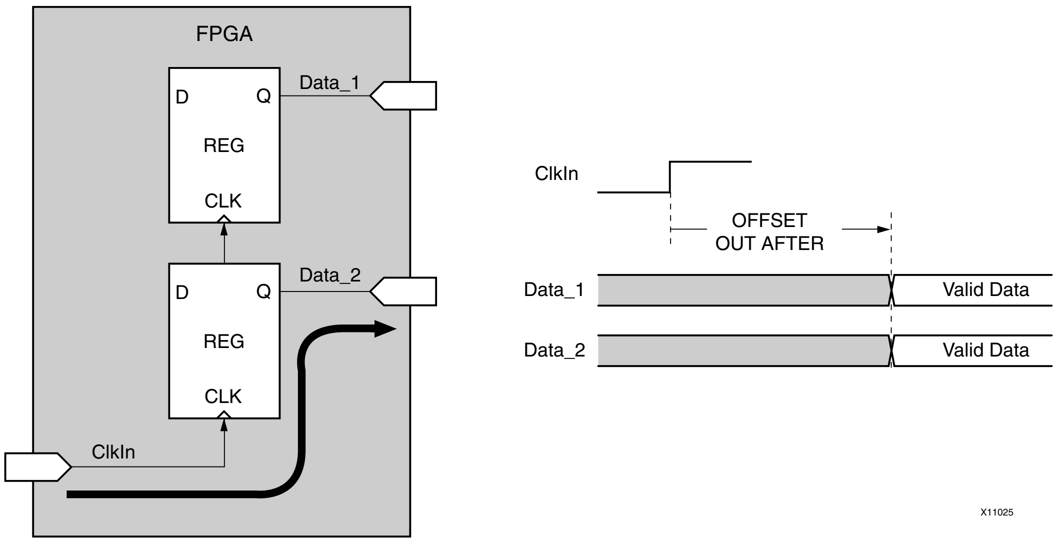


Figure 2-7: **Output Delay Path Example**

The input clock uncertainty and clock arrival times are derived from the PERIOD constraint associated with the interface clock referenced in the OFFSET OUT constraint.

For more information on the Period constraint and adding Input Jitter, see Period Constraints in Chapter 3, Timing Constraint Principles.

System Synchronous Output

In the system synchronous output interface, a common system clock both transfers and captures the data. Because this interface uses a common system clock, only the data is transmitted from the FPGA device to the receiving device.

See the following figure.

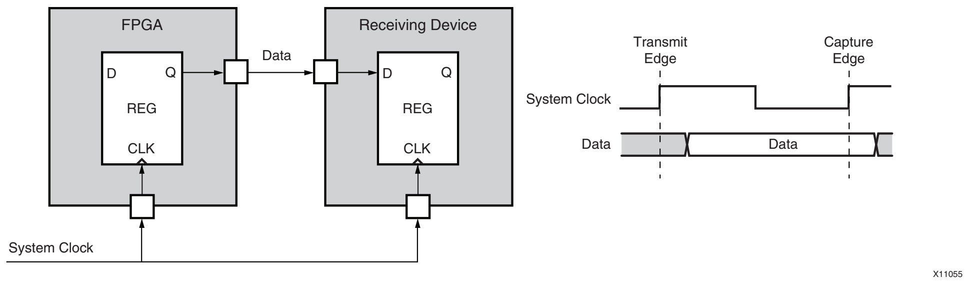


Figure 2-8: **Simplified System Synchronous Output Interface with Associated SDR Timing**

Specifying Output Timing

If these paths must be constrained, the global Offset Out constraint is the most efficient way to specify the output timing for the system synchronous interface. In the global method, one Offset Out constraint is defined for each system synchronous output interface clock. This single constraint covers the paths of all output data bits sent from registers triggered by the specified input clock.

To specify the output timing, define:

- A Timing Name for the output clock to create a time group, which contains all output registers triggered, by the input clock.
- The global Offset Out constraint for the interface.

System Synchronous SDR Output Interface

The following figure shows a timing diagram for a System Synchronous SDR output interface. The data in this example must become valid at the output pins a maximum of 5 ns after the input clock edge at the pin of the FPGA device.

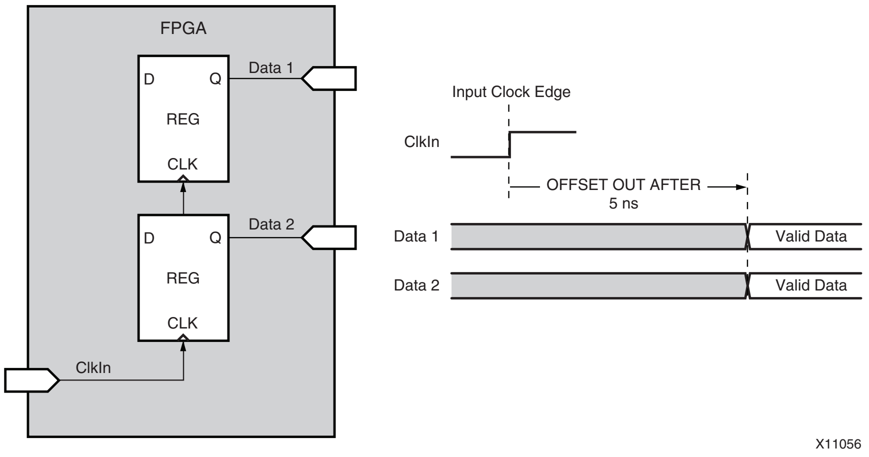


Figure 2-9: **Timing Diagram for System Synchronous SDR Output Interface**

Global Offset Out Constraint

The global Offset Out constraint for the system synchronous interface is:

```
OFFSET = OUT value AFTER clock;
```

In the Offset Out constraint, **`OFFSET=OUT`** **<***value***>** determines:

1. The maximum time from the rising clock edge at the input clock port, until -->
2. The data first becomes valid at the data output port of the FPGA device.

In this system synchronous example, the output data must become valid at least 5 ns after the input clock edge.

For this example, the complete Offset Out specification is:

```
NET "ClkIn" TNM_NET = "ClkIn";
OFFSET = OUT 5 ns AFTER "ClkIn";
```

This global constraint covers both data bits of the bus:

- data1
- data2

Source Synchronous Output

In the source synchronous output interface, a clock is regenerated and transmitted with the data from the FPGA device. The regenerated clock is transmitted with the data.

The interface is limited in performance primarily by:

- System noise, and
- The skew between the regenerated clock and the data bits

See the following figure.

Simplified Source Synchronous Output Interface with Associated DDR Timing

In this interface, the time from the input clock edge to the output data becoming valid is not as important as the skew between the output data bits. In most cases, it can be left unconstrained.

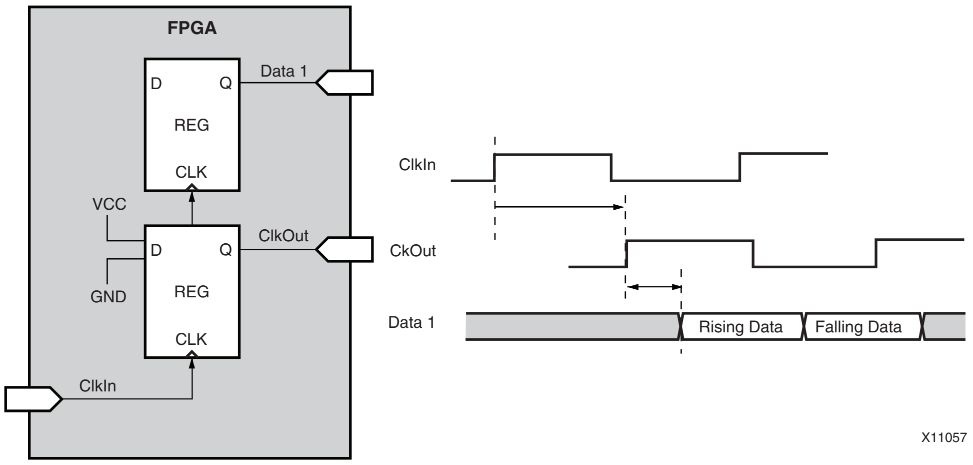


Figure 2-10: **Simplified Source Synchronous Output Interface with Associated DDR Timing**

Offset Out Constraint

The global Offset Out constraint is the most efficient way to specify the output timing for a source synchronous interface.

In the DDR interface, one Offset Out constraint is defined for each edge of the output interface clock. These constraints cover the paths of all output data bits that are transmitted by registers triggered with the specified output clock edge.

Specifying Input Timing

To specify the input timing, define:

- A Timing Name constraint for the output clock to create a time group containing all output registers triggered by the output clock.
- The global Offset Out constraint for the rising edge (Rising) of the interface.
- The global Offset Out constraint for the falling edge (Falling) of the interface.

Ideal Source Synchronous DDR Interface

The following figure shows a timing diagram for an ideal Source Synchronous DDR interface.

- The interface has a clock period of 5 ns with a 50/50 duty cycle.
- The data for both bits of the bus remains valid for the entire ½ period.

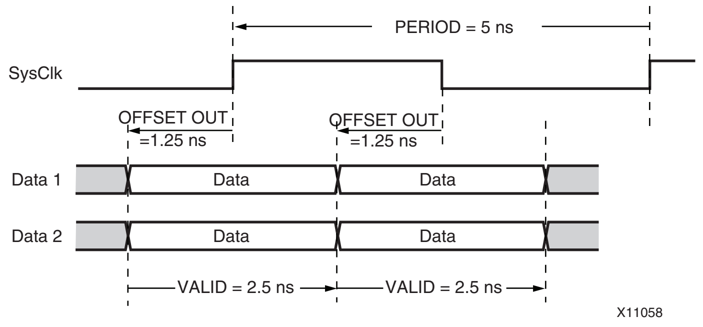


Figure 2-11: **Timing Diagram for an Ideal Source Synchronous DDR**

Offset Out Constraint

In the Offset Out constraint, **OFFSET=OUT** **<***value***>** determines:

1. The maximum time from the rising clock edge at the input clock port, until -->
2. The data first becomes valid at the data output port of the FPGA device.

When **<***value***>** is omitted from the Offset Out constraint, the constraint becomes a report-only specification that reports the skew of the output bus.

The Reference Pin keyword defines the regenerated output clock as the reference point against which the skew of the output data pins is reported.

For this example, the complete Offset Out specification for both the rising and falling clock edges is:

```
NET “ClkIn” TNM_NET = “ClkIn”;
OFFSET = OUT AFTER “ClkIn” REFERENCE_PIN “ClkOut” RISING;
OFFSET = OUT AFTER “ClkIn” REFERENCE_PIN “ClkOut” FALLING;
```

By using the global definitions of the input, register-to-register, and output timing constraints, the majority of the paths are properly constrained. However, in certain cases a small number of paths contain exceptions to the global constraint rules.

The most common types of exceptions are:

- False Paths (Paths Between Registers That Do Not Affect Timing)
- Multi-Cycle Paths

The global definitions of the following constraints properly constrain the majority of the paths:

- Input
- Register-to-register
- Output timing

In some cases, a small number of paths contain exceptions to the global constraint rules.

False Paths (Paths Between Registers That Do Not Affect Timing)

You can remove a set of paths from timing analysis if you are certain that these paths do not affect timing performance.

Use the From-To constraint with the Timing Ignore keyword to specify the set of paths to be removed from timing analysis. This allows you to:

- Specify a set of registers in a source time group.
- Specify a set of registers in a destination time group.
- Remove all paths between those time groups from analysis.

To specify the Timing Ignore constraint for this method, define:

- A set of registers for the source time group.
- A set of registers for the destination time group.
- A From-To constraint with a Timing Ignore keyword to remove the paths between the groups.

The following figure shows a path between two registers that does not affect timing. You want to remove this path from analysis.

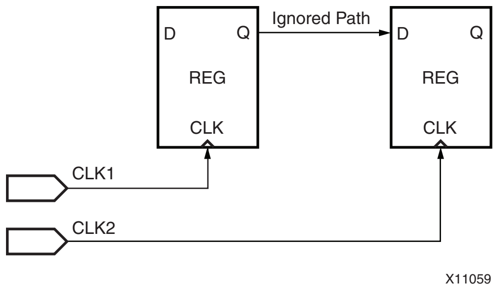


Figure 2-12: **Path Between Two Registers That Does Not Affect Timing**

The generic syntax for defining a Timing Ignore constraint between time groups is:

```
TIMESPEC "TSid" = FROM "SRC_GRP" TO "DST_GRP" TIG;
```

In the From-To Timing Ignore example:

- The SRC_GRP defines the set of source registers at which path tracing begins.
- The DST_GRP defines the set of destination registers at which the path tracing ends.
- All paths that begin in the SRC_GRP and end in the DST_GRP are ignored.

Syntax Example

```
NET "CLK1" TNM_NET = FFS "GRP_1";
NET "CLK2" TNM_NET = FFS "GRP_2";
TIMESPEC TS_Example = FROM "GRP_1" TO "GRP_2" TIG;
```

Multi-Cycle Paths

In a Multi-Cycle path, data is transferred from source to destination synchronous elements at a rate less than the clock frequency defined in the Period specification.

This occurs most often when the synchronous elements are gated with a common clock enable signal. By defining a Multi-Cycle path, the timing constraints for these synchronous elements are relaxed over the default Period constraint.

The Multi-Cycle path constraint can be defined with respect to the Period constraint identifier (**`TS_clk125`**) and state the multiplication or the number of period cycles (**`TS_clk125 * 3)`**. The implementation tools can then prioritize the implementation of these paths.

Specifying the Set of Multi-Cycle Paths

One common way to specify the set of Multi-Cycle paths is to define a time group using the clock enable signal. This allows you to:

- Define one time group containing both the source and destination synchronous elements using a common clock enable signal.
- Apply the Multi-Cycle constraint to all paths between these synchronous elements.

To specify the From:To (Multi-Cycle) constraint for this method, define:

- A Period constraint for the common clock domain.
- A set of registers based on a common clock enable signal.
- A From:To (Multi-Cycle) constraint describing the new timing requirement.

Path Between Two Registers Clocked by a Common Clock Enable Signal

The following figure shows a hypothetical case in which a path between two registers is clocked by a common clock enable signal.

In this example, the clock enable is toggled at a rate that is one-half of the reference clock.

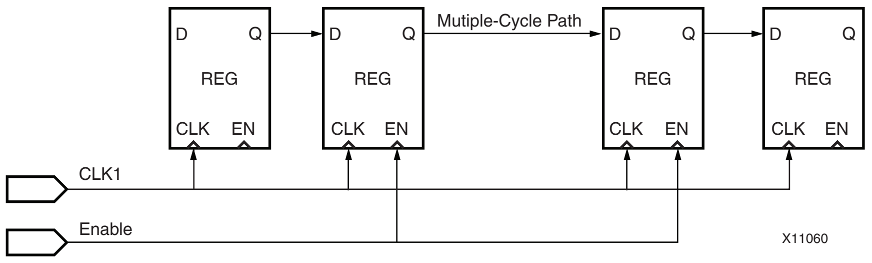


Figure 2-13: **Path Between Two Registers Clocked by a Common Clock Enable Signal**

Generic Syntax

The generic syntax for defining a Multi-Cycle path between time groups is:

```
TIMESPEC "TSid" = FROM "MC_GRP" TO "MC_GRP" <value>;
```

MC_GRP

In the From:To (Multi-Cycle) example:

- The MC_GRP defines the set of registers that are driven by a common clock enable signal.
- Paths that begin in the MC_GRP and end in the MC_GRP have the Multi-Cycle timing requirement applied to them.
- Paths into and out of the MC_GRP are analyzed with the appropriate Period specification.

Syntax Example

```
NET "CLK1" TNM_NET = "CLK1";
TIMESPEC "TS_CLK1" = PERIOD "CLK1" 5 ns HIGH 50%;
NET "Enable" TNM_NET = FFS "MC_GRP";
TIMESPEC TS_Example = FROM "MC_GRP" TO "MC_GRP" TS_CLK1*2;
```

Chapter 3

Timing Constraint Principles

This chapter discusses the fundamentals of timing constraints, including:

- Period Constraints
- Offset Constraints
- From:To (Multi-Cycle) Constraints

This chapter also discusses the ability to group elements in order to provide a better understanding of the constraint system subsystem.

Constraint System

The constraint system is that portion of the implementation tools (NGDBuild) that parses and understands the design physical and timing constraints.

The constraint system:

- Parses the constraints from the following files and delivers this information to the other implementation tools:
  - NCF
  - XCF
  - EDN, EDF, and EDIF
  - NGC
  - NGO
- Confirms that the constraints are correctly specified.
- Applies the necessary attributes to the corresponding elements.
- Issues error and warning messages for constraints that do not correlate correctly with the design.

DLL, DCM, PLL, BUFR, PMCD, and MMCM Components

When a Timespec Period specification on the input pad clock net is traced or translated through a DCM, DLL, PLL, BUFR, PMCD. or MMCM component (also known as a clock-modifying block), the derived or output clocks are constrained with new Period constraints.

In order to generate the destination-element-timing group, during transformation each clock output pin of the clock-modifying block is given:

- A new Timespec Period constraint
- A corresponding Timing Name Net constraint

The new Timespec Period constraint is based upon the manipulation of the clock modifying block component. The transformation:

- Takes into account the phase relationship factor of the clock outputs
- Performs the appropriate multiplication or division of the Period requirement value

Transformation Conditions

The transformation occurs when:

- The Timespec Period constraint is traced into the clkin pin of the clock modifying block component, and
- The group associated with the Period constraint:
  - Is used in exactly *one* Period constraint.
  - Is *not* used in any other timing constraints, including From:To (Multi-Cycle) constraints or Offset constraints.
  - Is *not* referenced or related to any other user group definition.

Example of New Period Constraints on DCM Outputs

If the Transformation Conditions are met, constraint (1) below is translated into constraints (2) and (3) below.

```
(1) TIMESPEC "TS_clk20" = PERIOD "clk20_grp" 20 ns HIGH 50%;
(2) CLK0: TS_clk20_0=PERIOD clk20_0 TS_clk20*1.000000 HIGH 50.000000%
(3) CLK90: TS_clk20_90=PERIOD clk20_90 TS_clk20*1.000000 PHASE + 5.000000 nS HIGH 50.000000%
```

These constraints are based upon the clock structure shown in the following figure.

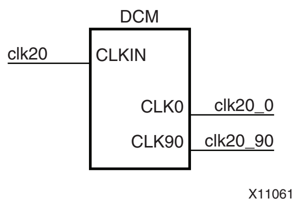


Figure 3-1: **New Period Constraints on DCM Outputs**

Report Message

The following message appears in the NGDBuild Report (`design.bld`) or the MAP Report (`design.mrp`):

```
INFO:XdmHelpers:851 - TNM " clk20_grp ", used in period specification "TS_clk20", was traced
into DCM instance "my_dcm". The following new TNM groups and period specifications were
generated at the DCM output(s):

clk0: TS_clk20_0=PERIOD clk20_0 TS_clk20*1.000000 HIGH 50.000000%
clk90: TS_clk20_90=PERIOD clk20_90 TS_clk20*1.000000 PHASE + 5.000000 nS HIGH 50.000000%
```

Adjustment of Translated Period Constraints

If the CLKIN_DIVIDE_BY_2 attribute is set to True for the DCM in Figure 3-1, New Period Constraints on DCM Outputs, the translated Period constraints are adjusted accordingly. The following constraints are the result of this attribute:

```
CLK0:   TS_clk20_0=PERIOD clk20_0 TS_clk20*2.000000 HIGH 50.000000%
CLK90: TS_clk20_90=PERIOD clk20_90 TS_clk20*2.000000 PHASE + 5.000000 nS HIGH 50.000000%
```

If Transformation Conditions Are Not Met

If the Transformation Conditions are *not* met:

- The Period constraint is *not* placed on the output or derived clocks of the clock modifying block component, and
- An error or warning message appears in the NGDBuild Report.

Error Message Example

```
"ERROR:NgdHelpers:702 - The TNM "PAD_CLK" drives the CLKIN pin of CLKDLL "$I1". This TNM
cannot be traced through the CLKDLL because it is not used in exactly one PERIOD
specification. This TNM is used in the following user groups and/or specifications:

TS_PAD_CLK=PERIOD PAD_CLK 20000.000000 pS HIGH 50.000000%
TS_01=FROM PAD_CLK TO PADS 20000.000000 pS"
```

The original Timespec Period constraint:

- Is reported in the Timing Report.
- Shows **`0 items analyzed`**.

The newly created Timespec Period constraints contain all paths associated with the clock modifying block component.

If the Period constraint is not translated, and traces only to the clock modifying block component:

- The Timing Report shows **`0 items analyzed`**.
- No other Period constraints are reported.

If the Period constraint traces to other synchronous elements, the analysis includes only those synchronous elements.

Synchronous Elements

Synchronous elements include:

- Flip Flops
- Latches
- Distributed RAM
- Block RAM
- Distributed ROM
- ISERDES
- OSERDES
- PPC405
- PPC440

- MULT18X18
- DSP48
- MGTs (GT, GT10, GT11, GTP, GTX, GTH)
- MCB
- SRL16
- EMAC
- FIFO (16, 18, and 36)
- PCIE
- TEMAC

Analysis With Net Period Constraint

When a Net Period constraint is applied to the input clock pad or net, this constraint is not translated through the clock modifying block component. This can result in zero items or paths analyzed for these constraints.

The Net Period constraint is analyzed only during MAP, PAR, and Timing analysis. When **`MAP -timing`** and PAR call the timing tools, the timing tools manipulate the clock modifying block for placement and routing, but not for the timing analysis Timing Reports.

When a Timespec Period constraint is traced into an input pin on a clock modifying block, NGDBuild or the translate Period transforms the original Timespec Period constraint into new Timespec Period constraints based upon the derived output clocks. The NGDBuild Report (`design.bld`) indicates this transformation.

MAP, PAR, and Timing Analyzer use the new derived clock Timespec Period constraints that are propagated to the Physical Constraints File (PCF).

The original Timespec Period constraint:

- Is unchanged during this transformation.
- Is used as a reference for the new Timespec Period constraints.

Constraints Editor sees only the original Period constraint. Constraints Editor does *not* see the newly transformed Period constraints.

Phase Keyword

The Phase keyword is used in the relationship between related clocks. The timing analysis tools use this relationship for the Offset constraints and cross-clock domain path analysis.

The Phase keyword can be entered:

- In the UCF or NCF, or
- Through the translation of the DCM, DLL, and PLL components during NGDBuild.

If the phase shifted value of a DCM, PLL, or DLL component is changed in FPGA Editor, the change is not reflected in the PCF file.

The timing analysis tools use the Phase keyword value in the PCF to emulate the DLL, DCM, or PLL component phase shift value. In order to see the change that was made in FPGA Editor, the PCF must also be modified manually with the corresponding change.

DLL, DCM, and PLL Component Manipulation with Phase

The following table displays the new DCM, DLL, or PLL component output clock net derived Timespec Period constraints. These new constraints are based upon the original Period (TS_CLKIN) constraints. TS_CLKIN is expressed as a time value.

If TS_CLKIN is expressed as a frequency value, the multiply and divide operations are reversed. If the DCM attributes FIXED_PHASE_SHIFT or VARIABLE_PHASE_SHIFT are used, the amount of the phase-shifted value is included in the Phase keyword value.

The DCM attributes FIXED_PHASE_SHIFT or VARIABLE_PHASE_SHIFT phase shifting amount on the DCM is not reflected in the following table.

Table 3-1: **Transformation of Period Constraint Through DCM**

Output Pin	Period Value	Phase Shift value
CLK0	TS_CLKIN * 1	None
CLK90	TS_CLKIN * 1	PHASE + (clk0_period * ¼)
CLK180	TS_CLKIN * 1	PHASE + (clk0_period * ½)
CLK270	TS_CLKIN * 1	PHASE + (clk0_period * ¾)
CLK2x	TS_CLKIN / 2	None
CLK2x180	TS_CLKIN / 2	PHASE + (clk2x_period * ½)
CLKDV	TS_CLKIN * clkdv_divide (clkdv_divide = value of CLKDV_DIVIDE property (default = 2.0))	None
CLKFX	TS_CLKIN / clkfx_factor (clkfx_factor = value of CLKFX_MULTIPLY property (default = 4.0) divided by value of CLKFX_DIVIDE property (default = 1.0))	None
CLKFX180	TS_CLKIN / clkfx_factor (clkfx_factor = value of CLKFX_MULTIPLY property (default = 4.0) divided by value of CLKFX_DIVIDE property (default = 1.0))	PHASE + (clkfx_period * ½)

Timing Group Creation with Timing Name or Timing Name Net Attributes

All design elements with the same Timing Name or Timing Name Net attribute are considered a timing group. A design element may be in multiple timing groups (Timing Name or Timing Name Net).

The Timing Name or Timing Name Net attributes can be applied to:

- Net Connectivity (NET)
- Instance or Module (INST)
- Instance Pin (PIN)

To ensure correct timing analysis, place only one Timing Name or Timing Name Net on each element, driver pin, or macro driver pin.

Net Connectivity

Identifying groups by net connectivity allows the grouping of elements by specifying a net or signal that eventually drives synchronous elements and pads.

This method identifies Multi-Cycle path elements that:

- Are controlled by a clock enable, and
- Can be constrained as a From:To (Multi-Cycle) constraint.

This method uses Timing Name Net or Timing Name on a net. The Timing Name attribute is commonly used on HDL port declarations, which are directly connected to pads.

Timing Name Attribute Placed on Net or Signal

If a Timing Name attribute is placed on a net or signal, the constraints parser traces the signal or net downstream to the synchronous elements.

Use a Timing Name attribute to identify the elements that make up a timing group. This timing group can then be used in a timing constraint.

Those synchronous elements are tagged with the same Timing Name attribute. The Timing Name attribute name is used in a Timing Specifications constraint or in a Timing constraint.

Timing Name on the CLOCK Pad or Net

The clock net in the following figure is traced forward to the two flip-flops.

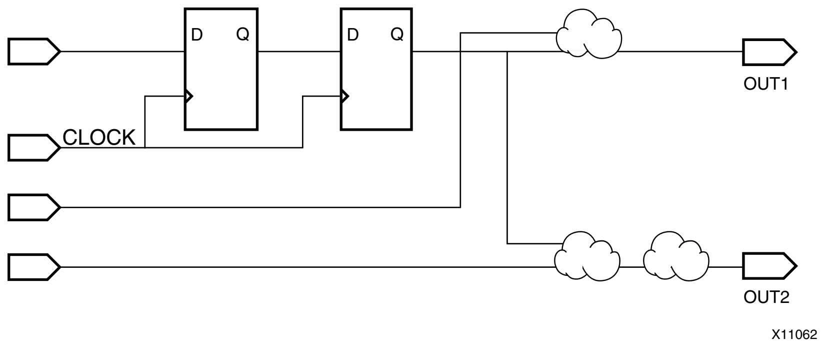


Figure 3-2: **Timing Name on the CLOCK Pad or Net Traces Downstream to the Flip-Flops**

Flagging a Common Input

A common input is typically a**:**

- Clock signal, or
- Clock enable signal

Flag a common input to group:

- Flip-flops
- Latches
- Other synchronous elements

The Timing Name is traced forward along the path (through any number of gates, buffers, or combinatorial logic) until it reaches a flip-flop, input latch, or synchronous element. Those elements are added to the specified Timing Name or time group.

Timing Name on the A0 Net Traced Through Combinatorial Logic to Synchronous Elements (Flip-Flops)

The following figure shows the user of Timing Name on a net that traces forward to create a group of flip-flops.

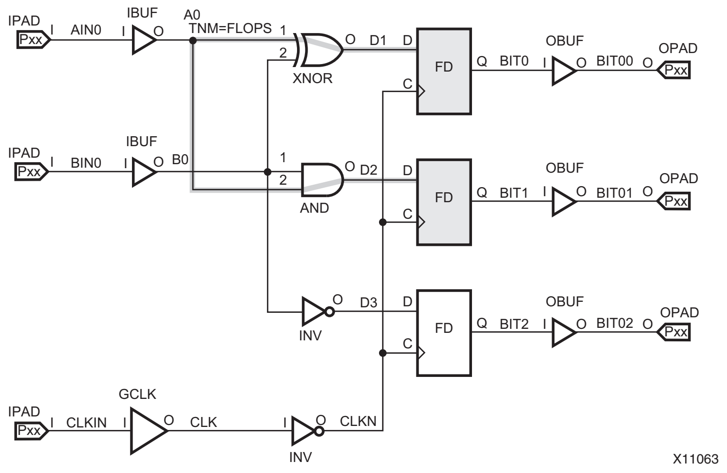


Figure 3-3: **Timing Name on the A0 Net Traced Through Combinatorial Logic to Synchronous Elements (Flip-Flops)**

Using a Qualifier

When you place a Timing Name constraint on a net, use a qualifier to narrow the list of elements in the time group. A qualified Timing Name is traced forward until it reaches the first synchronous element that matches the qualifier type. The qualifier types are the predefined time groups.

If that type of synchronous element matches the qualifier, the synchronous element is given that Timing Name attribute. Whether or not there is a match, the Timing Name is not traced through the synchronous element.

Predefined Time Groups

The following keywords are predefined time groups.

FFS

All SLICE and IOB edge-triggered flip-flops and shift registers

PADS

All I/O pads

DSPS

- All DSP48 in Virtex®-4 devices
- All DSP48E in Virtex-5 devices

- All DSP48E1 in Virtex-6 and Xilinx® 7 series FPGA devices
- All DSP48A1 in Spartan®-6 devices

RAMS

All single-port and dual-port SLICE LUT RAMs and block Rams

MULTS

All synchronous and asynchronous multipliers in Virtex-4 and Virtex-5 devices:

HSIOS

- All GT and GT10 in Virtex-4 devices
- All GTP in Virtex-5 devices
- All GTHE1 and GTXE1 in Virtex-6 devices
- ALL GTPA1 in Spartan-6 devices
- All GTHE2 and GTXE2 in Xilinx 7 series FPGA devices

CPUS

- All PPC405 in Virtex-4 devices
- All PPC450 in Virtex-5 devices

LATCHES

All SLICE level-sensitive latches

BRAMS_PORTA

Port A of all dual-port block RAMs

BRAMS_PORTB

Port B of all dual-port block RAMs

Differences Between Timing Name Net (TNM_NET) and Timing Name (TNM) on a Net

The Timing Name Net constraint:

- Is equivalent to Timing Name on a net.
- Produces different results on pad nets.

The Translate Process or NGDBuild command never transfers a Timing Name Net constraint from the attached net to an input pad, as it does with the Timing Name constraint.

Use Timing Name Net only with nets. If you use Timing Name Net with any other object (such as a pin or instance), the tool:

- Issues a warning.
- Ignores the Timing Name Net definition.

In the case of a Timing Name attribute on a pad net, or the net between the IPAD and the IBUF, the constraints parser traces the signal or net upstream to the pad element. See Figure 3-4, Differences Between Timing Name and TNM_NET. The Timing Name Net attribute is traced through the buffer to the synchronous elements.

In HDL designs, the IBUF output signal is the same as the IPAD or port name. There are no differences between the Timing Name Net and Timing Name attributes. Both Timing Name attributes trace downstream to the synchronous elements.

Propagation Rules for the Timing Name Net Constraint

Following are the propagation rules for the Timing Name Net (TNM_NET) constraint.

TNM_NET Applied to a Pad Net

If applied to a pad net, the Timing Name Net constraint propagates forward through the IBUF elements and any other combinatorial logic to synchronous elements or pads.

TNM_NET Applied to a Clock-Pad Net

If applied to a clock-pad net, the Timing Name Net constraint propagates forward through the clock buffer to synchronous elements or pads.

TNM_NET Applied to Certain Input Clock Nets

If applied to an input clock net of a DCM, DLL, PLL, PMCD, or BUFR component, and associated with a Period constraint, the Timing Name Net constraint propagates forward through the clock-modifying block to synchronous elements or pads.

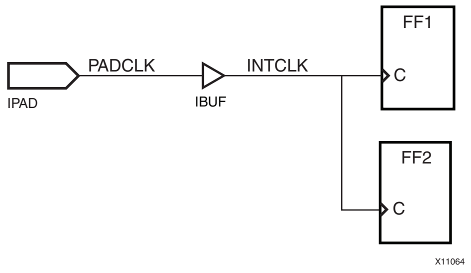


Figure 3-4: **Differences Between Timing Name and TNM_NET**

In the figure above, a Timing Name associated with the IPAD signal includes only the PAD symbol as the member of a time group. A Timing Name Net constraint associated with the IPAD signal includes all synchronous elements after the IBUF as members of a time group.

Creating Time Groups Using IPAD Signal

The following examples show different ways of creating time groups using the IPAD signal.

NET PADCLK TNM = PAD_grp;

- Use the **`padclk`** net to define the time group **`PAD_grp`**.
- Contains the IPAD element.

NET PADCLK TNM = FFS "FF_grp";

- Use the **`padclk`** net to define the time group **`FF_grp`**.
- Contains no flip-flop elements.

NET PADCLK TNM_NET = FFS FF2_grp;

- Use the **`padclk`** net to define the timing group **`FF2_grp`**.
- Contains all flip-flop elements associated with this net.time

In the figure above, a TNM associated with the IBUF output signal can only include the synchronous elements after the IBUF as members of a time group.

Time Groups Including Only the IBUF Output Signal

The following time groups include only the IBUF output signal.

NET INTCLK TNM = FFS FF1_grp;

- Use the **`intclk`** net to define the time group **`FF1_grp`**.
- Contains all flip-flop elements associated with this net.

NET INTCLK TNM_NET = RAMS Ram1_grp;

- Use the **`intclk`** net to define the time group **`Ram1_grp`**.
- Contains all distributed and block RAM elements associated with this net.

Instance or Hierarchy

When a Timing Name attribute is placed on a module or macro, the constraints parser traces the macro or module down the hierarchy to the synchronous elements and pads.

The attribute traverses through all levels of the hierarchy rather than forward along a net or signal. This feature is illustrated in:

- Figure 3-2, Timing Name on the CLOCK Pad or Net Traces Downstream to the Flip-Flops
- Figure 3-3, Timing Name on the A0 Net Traced Through Combinatorial Logic to Synchronous Elements (Flip-Flops)

Timing Name Attribute

Those synchronous elements are tagged with the same Timing Name attribute. The Timing Name attribute name is used in a Timing Specifications or timing constraint. This method uses a Timing Name on a block. Multiple instances of the same Timing Name attribute are used to identify the time group.

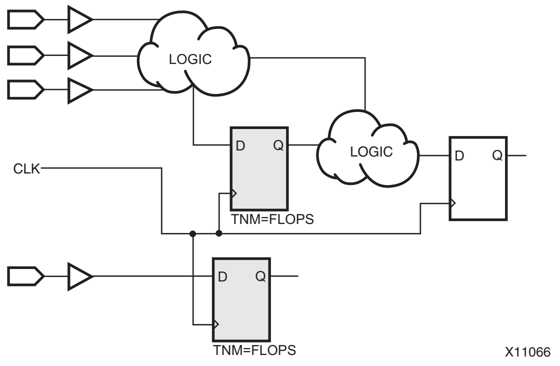


Figure 3-5: **Grouping By Instances**

Macros and Modules

A macro or module:

- Performs some general purpose higher level function.
- Typically has a lower level design that consists of:
  - Primitives or elements, and/or
  - Other macros or modules.

  These components are connected together to implement the higher level function.

A Timing Name constraint attached to a module or macro indicates that all elements inside the macro or module (at all levels of hierarchy below the tagged module or macro) are part of the named time group.

Use the Keep Hierarchy attribute to ensure that the design hierarchy is maintained. This feature is illustrated in the following figure.

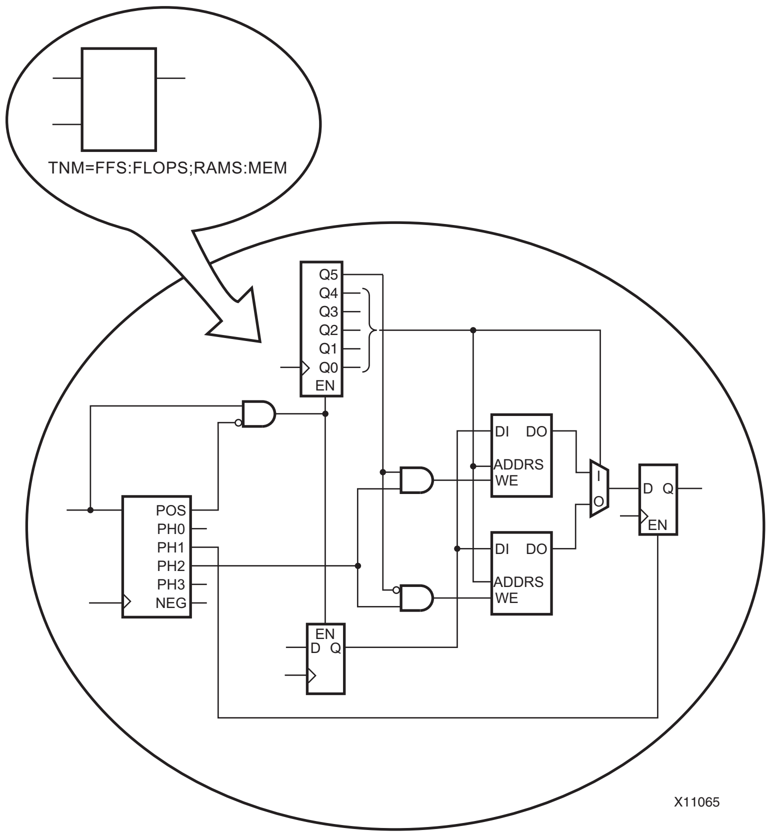


Figure 3-6: **Timing Name on Upper Left Hierarchy is Traced Down to Lower Level Element**

Using Wildcard Characters

Use wildcard characters to traverse the hierarchy of a design.

- A question mark (?) represents one character.
- An asterisk (*) represents multiple characters.

The following example uses a wildcard character to traverse the hierarchy where **`Level1`** is a top level module:

- **`Level1/*`**

  Traverses all blocks in Level1 and below

- **`Level1/*/`**

  Traverses all blocks in Level1 but no further

The instances described below are either:

- Symbols on a schematics, or
- A symbol name as it appears in the EDIF netlist

Wildcard Traversing the Design Hierarchy

An example of the wildcard traversing the design hierarchy is shown in Figure 3-7, Traversing Hierarchy with Wildcards, for the following instances:

`INST *`

All synchronous elements are in this timing group.

`INST /*`

All synchronous elements are in this timing group.

`INST /*/`

Top level elements or modules are in this timing group:

- **A1**
- **B1**
- **C1**

`INST A1/*`

All elements one or more levels of hierarchy below the A1 hierarchy are in this timing group:

- **A21**
- **A22**
- **A3**
- **A4**

`INST A1/*/`

All elements one level of hierarchy below the A1 hierarchy are in this timing group:

- **A21**
- **A22**

`INST A1/*/*`

All elements two or more levels of hierarchy below the A1 hierarchy are in this timing group:

- **A3**
- **A4**

INST A1/*/*/

All elements two levels of hierarchy below the A1 hierarchy are in this timing group:

- **A3**

INST A1/*/*/*

All elements three or more levels of hierarchy below the A1 hierarchy are in this timing group:

- **A4**

INST A1/*/*/*/

All elements three levels of hierarchy below the A1 hierarchy are in this timing group:

- **A4**

INST /*/*22/

All elements with instance name of 22 are in this timing group:

- **A22**
- **B22**
- **C22**

INST /*/*22

All elements with instance name of 22 and elements one level of hierarchy below are in this timing group:

- **A22**
- **A3**
- **A4**
- **B22**
- **B3**
- **C22**
- **C3**

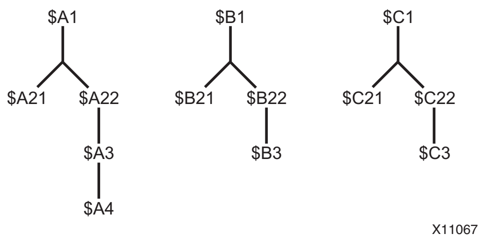


Figure 3-7: **Traversing Hierarchy with Wildcards**

Instance Pin

Identifying groups by pin connectivity allows you to group elements by specifying a pin that eventually drives synchronous elements and pads. This method uses Timing Name on a pin.

If a Timing Name attribute is placed on a pin, the constraints parser traces the pin downstream to the synchronous elements. A Timing Name attribute identifies the elements that make up a timing group. This timing group can be then used in a timing constraint. See the following figure. For more information, see Using a Qualifier.

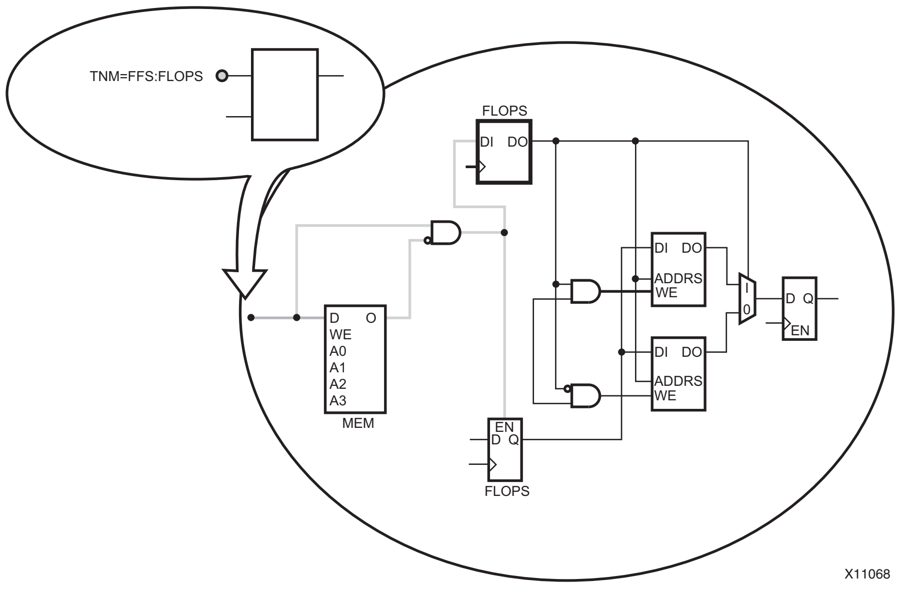


Figure 3-8: **Timing Name Placed on Macro Pin Traces Downstream to Synchronous Elements**

Grouping Constraints

Grouping constraints allow you to group similar elements together for timing analysis. They can be defined in the following files:

- UCF
- NGC
- EDN
- EDIF
- EDF

The timing analysis is on timing constraints, which are applied to logical paths. The logic paths typically start and stop at pads and synchronous elements.

The grouped elements signify the starting and ending points for timing analysis. These starting and ending points can be based upon predefined groups, user-defined groups, or both.

The timing groups are ideal for identifying groups of logic that operate at different speeds, or that have different timing requirements.

Time Groups

Time groups are used in timing analysis. The user-defined groups and the predefined time groups report to the timing analysis tools the start point and the end point for each path being analyzed.

Time groups are used in the following constraints:

- Period
- Offset In
- Offset Out
- From:To (Multi-Cycle)
- Timing Ignore

When using a specific net or instance name, use its full hierarchical path name. This allows the implementation tools to find the net or instance.

Use pattern matching wildcards to specify the element name when creating timing groups with predefined timing group qualifiers. Place the pattern in parentheses after the timing group qualifier.

Predefined Time Groups

The predefined time groups can reference the following (among others):

- Flip-flops
- Latches
- Pads
- RAMs
- CPUs
- Multipliers
- High-speed-inputs or outputs

You can use the predefined time group keywords globally, and to create user-defined sub-groups. The predefined time groups:

- Are reserved keywords.
- Define the types of synchronous elements and pads in the FPGA device.

User-Defined Time Groups

The user-defined time group name:

- Is case sensitive
- Can overlap with:
  - Other user-defined time groups
  - Predefined time groups

This causes design elements to be in multiple time groups. In those cases, a register is in:

- The FFS predefined time group
- The **`clk`** time group associated with the Period constraint

Defining User-Defined Time Groups

Use the following keywords to define user-defined time groups:

- TNM
- TNM_NET
- TIMEGRP

If the instance or net associated with the user-defined time group matches internal reserved words, the timing group or constraint is rejected. The same is true for the user-defined time group name.

Double Quotes

The NCF, UCF, and PCF constraints files may reject some instances or variable names unless the names are enclosed in double quotes.

Enclose an instance or net name in double quotes if the name:

- Matches an internal reserved word, or
- Contains special characters such as the tilde (~) or dollar sign ($).

Xilinx recommends using double quotes on all net and instances.

Timing Name and Timing Name Net Attributes

All elements with the same Timing Name or Timing Name Net attributes are considered a timing group.

For more information about Timing Name and Timing Name Net attributes, see Constraint System in Chapter 3, Timing Constraint Principles.

Timing Group Attribute

Use the Timing Group attribute to:

- Combine existing pre-defined or user-defined time groups.
- Remove common elements from existing time groups, and create a new user-defined time group.
- Create a new time group by pattern matching.

  Pattern matching is grouping a set of objects that all have output nets that begin with a given string.

Creating Subsets of an Existing Time Group

Use the following keywords to create subsets of an existing time group:

- Except

  Remove common elements
- Rising

  Rising edge synchronous elements
- Falling

  Falling edge synchronous elements

Except Keyword

Use the Except keyword with a Timing Group attribute to remove elements from an already-created time group.

The overlapping items to be removed from the original time group must be in the excluded or Except time group.

If the excluded time group does not overlap with the original time group, none of the design elements are removed. In that case, the new time group contains the same elements as the original time group.

Rising and Falling Keywords

Use Timing Group to:

- Include multiple time groups.
- Exclude multiple time groups.
- Create sub-groups with the Rising and Falling keywords.

Use Rising and Falling to create groups based upon the synchronous element triggered clocking edge (rising or falling edges).

Pattern Matching

Pattern matching on either net or instance names can define the user-defined time group.

Use wildcards to:

- Define a user-defined time group of symbols whose associated net name or instance name matches a specific pattern.
- Generalize the group selection of synchronous elements.
- Shorten and simplify the full hierarchical path to the synchronous elements.

Table 3-2: **Pattern Matching Symbols**

Name	Symbol	Matches
Asterisk	*	Any string of zero or more characters
Question mark	?	A single character

Table 3-3: **Pattern Matching Examples**

String	Indicates	Examples
DATA*	Any net or instance name that begins with **DATA**	**DATA1**, **DATA22**, and **DATABASE**
NUMBER?	Any net names that begin with **NUMBER** and ends with one single character	**NUMBER1** or **NUMBERS**, but not **NUMBER** or **NUMBER12**

A pattern may contain more than one wildcard character. For example, ***AT?** specifies any net name that:

- Begins with one or more characters followed by **AT,** and
- Ends with any one character.

The following net names are included in ***AT?**:

- BAT2
- CAT4
- THAT9

Time Group Examples

Following are six time group examples.

Predefined Group of RAMs Time Group Example

The following time groups are created with a search string and a predefined group of RAMs in a Multi-Cycle constraint.

- `INST my_core TNM = RAMS my_rams;`

  This time group (**my_rams**) is the RAM components of the hierarchical block **my_core**
- `TIMSPEC TS01 = FROM FFS TO my_rams 14.24ns;`
- `NET clock_enable TNM_NET = RAMS(address*) fast_rams;`

  This time group (fast_rams) is the RAM components driven by net name of **clock_enable** with an output net name of **address***
- `TIMSPEC TS01 = FROM FFS TO fast_rams 12.48ns;` OR
- `TIMESPEC TS01 = FROM FFS TO RAMS(address*) 12.48ns;`

  The destination time group is based upon RAM components with an output net name of **address***

Predefined Group of FFS Time Group Example

The following time group is created with a search string and a predefined group of FFS in a Multi-Cycle constraint.

```
TIMESPEC TS01 = FROM RAMS TO FFS(macro_A/Qdata?) 14.25ns;
```

The destination time group is based upon flip flop components with an output net named **`macro_A/Qdata?`**

Predefined Group on a Hierarchical Instance Timing Group Example

The following time groups are created with the predefined group on a hierarchical instance.

- **`INST macroA TNM = LATCHES latch_grp;`**

  This time group (**`latch_grp`**) consists of the latch components of the hierarchical instance **`macroA`**

- **`INST macroB TNM = RAMS memory_grp;`**

  This time group (**`memory_grp`**) consists of the RAM components of the hierarchical instance **`macroB`**

- **`INST tester TNM = overall_grp;`**

  This time group (**`overall_grp`**) consists of synchronous components (such as RAMS, FFS, LATCHES, and PADS) of the hierarchical instance **`tester`**.

Combining Time Groups Examples

The following examples show how to define a new time group by combining it with other time groups.

- **`TIMEGRP "larger_grp" = "small_grp" "medium_grp";`**

  Combines **`small_grp`** and **`medium_grp`** into a larger group called **`larger_grp`**

- **`TIMEGRP memory_and_latch_grp = latch_grp memory_grp;`**

  Combines the elements of **`latch_grp`** and **`memory_grp`**.

Removing Time Groups Examples

The following examples use the Except keyword with the Timing Group attribute.

- **`TIMEGRP new_time_group = Original_time_group EXCEPT a_few_items_time_grp;`**

  Removes the elements of **`a_few_items_time_grp`** from **`Original_time_group`**

- **`TIMEGRP "medium_grp" = "small_grp" EXCEPT "smaller_grp";`**
  - Creates a time group **`medium_grp`** from the elements of **`small_grp`**
  - Removes the elements of **`smaller_grp`**

- **`TIMEGRP all_except_mem_and_latches_grp = overall_grp EXCEPT memory_and_latch_grp;`**

  Removes the common elements between **`memory_and_latch_grp`** and **`overall_grp`**

Clock Edges Time Group Examples

The following examples define a sub-group based upon the triggering clock edge.

- **`TIMEGRP "rising_clk_grp" = RISING clk_grp;`**
    - Creates a time group **`rising_clk_grp`**
    - Includes all *rising* edged synchronous elements of **`clk_grp`**
- **`TIMEGRP "rising_clk_grp" = FALLING clk_grp;`**
    - Creates a time group **`rising_clk_grp`**
    - Includes all *falling* edged synchronous elements of **`clk_grp`**

Constraint Priorities

During design analysis, the timing analysis tools determine which constraint analyzes which path. Each constraint type has different priority levels.

Priority Order

Following are the constraint priorities, from highest to lowest:

- Timing Ignore
- From:Thru:To
    - Source and destination are user-defined groups
    - Source or destination are user-defined groups
    - Source and destination are pre-defined groups
- From:To
    - Source and destination are user-defined groups
    - Source or destination are user-defined groups
    - Source and destination are pre-defined groups
- Offset
    - Specific Data IOB (Net Offset)
    - Time Group of Data IOB components (Grouped Offset)
    - All Data IOB components (Global Offset)
- Period

  This determination is based upon the constraint prioritization or which constraint appears later in the PCF file, if there are overlapping constraints of the same priority.
- MAXSKEW and MAXDELAY

  Net delay and net skew specifications are analyzed independently of path delay analysis and do not interfere with one another. NET TIG do interact with the NET constraints and take precedence.

Constraint Set Interaction

There are circumstances in which constraint priority may not operate as expected. These cases include supersets, subsets, and intersecting sets of constraints. See the following figure.

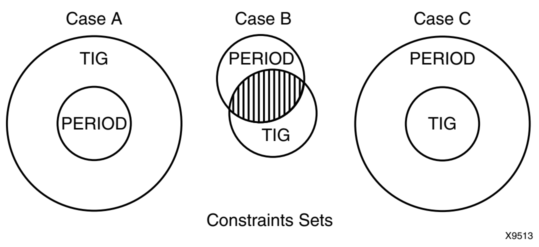


Figure 3-9: **Interaction Between Constraint Sets**

- In Case A, the Timing Ignore superset conflicts with the Period set.
- In Case B, the intersection of the Period and Timing Ignore sets creates an ambiguous circumstance. In this instance, constraints may sometimes be considered as part of Timing Ignore, and at other times part of Period.

Two Period Constraints Covering the Same Paths

If two Period constraints in the PCF file cover the same paths:

- The first Period constraint shows **`0 paths analyzed`** in the Timing Report.
- The second Period constraint analyzes the paths.

To force the timing analysis tools to use the first Period constraint instead of the second Period constraint:

- Use the Priority keyword on the Period constraints, or
- Use a Multi-Cycle or From:To constraint to cover these paths.

Use the Priority keyword with a value in order to (1) prioritize within a constraint type, or (2) avoid a conflict between two timing constraints that cover the same path.

- The value can range from -255 to +255.
- The lower the value, the higher the priority.
- The value does not affect which paths are placed and routed first.
- The value affects only which constraint covers and analyzes the path with two timing constraints of equal priority.

A constraint *with* a Priority keyword always has a higher priority than a constraint *without* a Priority keyword.

Timing Constraint Priority Syntax

Use the following syntax to define the priority of a timing constraint:

- **`TIMESPEC TS_01 = FROM A_grp TO B_grp 10 ns PRIORITY 5;`**

  `TS_01` has a lower priority than **`TS_02`**.
- **`TIMESPEC TS_02 = FROM A_grp TO B_grp 20 ns PRIORITY 1;`**

Using the Priority Keyword

The Priority keyword:

- Can be applied *only* to Timespec constraints with TSidentifiers (for example, TS03).
- Can *not* be applied to Maxdelay, Maxskew, and Offset constraints.

Priority with a BUFGMUX Component

This situation can occur when two clock signals from the DCM drive the same BUFGMUX. See the following figure.

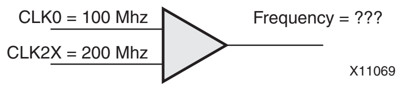


Figure 3-10: **Priority with a BUFGMUX Component**

Period Constraint Using Priority Examples

The following examples show a Period constraint using the Priority keyword:

```
TIMESPEC "TS_Clk0" = PERIOD "clk0_grp" 10 ns HIGH 50% PRIORITY 2;
TIMESPEC "TS_Clk2X" = PERIOD "clk2x_grp" TS_Clk0 / 2 PRIORITY 1;
```

Timing Constraints

Timing constraints provide a basis for the design of timing goals. Use global timing constraints to set timing requirements that cover all constrainable paths.

Global timing constraints are the easiest way to:

- Provide coverage of constrainable connections.
- Guide the implementation tools to meeting timing requirements for all paths.

Global timing constraints constrain the entire design.

Fundamental Timing Constraints

The following fundamental timing constraints are needed for every design:

- Clock definitions with a Period constraint for each clock

  Constrains synchronous element to synchronous element paths
- Input requirements with Global Offset In constraints

  Constrains interfacing inputs to synchronous elements paths
- Output requirements with Global Offset Out constraints

  Constrains interfacing synchronous elements to outputs to paths
- Combinatorial path requirements with Pad-to-Pad constraints

You can use more specific path constraints for Multi-Cycle or static paths.

- A Multi-Cycle path is a path between two registers or synchronous elements with a timing requirement that is a multiple of the clock Period constraint for the registers or synchronous elements.
- A static path does not include clocked elements such as Pad-to-Pad paths.

Timing Constraint Exceptions

Once you have laid the foundation of the timing constraints, specify and constrain the exceptions.

Table 3-4: **Timing Constraint Exceptions**

For This Constraint...	Create Exceptions With ...
Period	• From: To (Multi-Cycle) constraints
Global Offset	• Pad Timing Group based Offset constraints • Net based Offset constraints

Setting Timing Constraint Requirements

Xilinx recommends that you:

- Set the timing constraint requirements to the exact timing requirement value required for a path.
- Do *not* over-tighten the requirement.

Tighter constraint requirements can:

- Lengthen place and route (PAR) or implementation runtimes.
- Increase memory usage.
- Degrade the quality of results (QOR).

Period Constraints

The Period (Clock Period Specification) constraint is a fundamental timing and synthesis constraint.

Period constraints:

- Define clocks.
- Cover all synchronous paths within each clock domain.
- Cross check clock domain paths between related clock domains.
- Define clock duration.
- Can be configured to have different duty cycles.
- Are preferred over From: To constraints because Period constraints:
  - Cover a majority of the paths.
  - Decrease implementation tool runtime.

Clock Period Specification

The Clock Period Specification defines:

- The timing between synchronous elements (FFS, RAMS, LATCHES, HSIOS, CPUs, and DSPS) clocked by a specific clock net that is terminated at a registered clock pin. See the following figure.
- The timing between related clock domains based upon the destination clock domain.

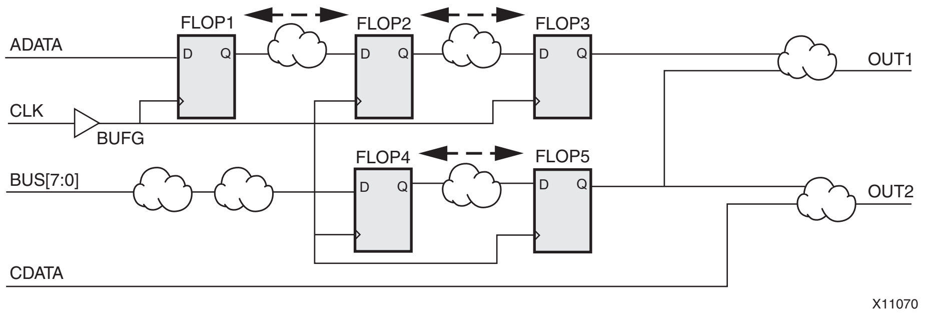


Figure 3-11: **Period Constraints Covering Register to Register Paths**

Period Constraint on Clock Net

The Period constraint on a clock net analyzes all delays on all paths that terminate at a pin with a setup and hold analysis relative to the clock net. A typical analysis includes the data paths of:

- Intrinsic clock-to-out delay of the synchronous elements
- Routing and logic delay
- Intrinsic setup or hold delay of the synchronous elements
- Clock skew between the source and destination synchronous elements
- Clock phase (DCM phase and negative edge clocking)
- Clock duty cycles

Included in Period Constraint

The Period constraint includes:

- Clock path delay in the clock skew analysis for global and local clocks
- Local clock inversion
- Setup and hold time analysis
- Phase relationship between related clocks

  Related and derived clocks can be a function of another clock (* and /)
- DCM Jitter, Duty-Cycle Distortion, and DCM Phase Error for Virtex-4, DCM Jitter, PLL Jitter, Duty-Cycle Distortion, and DCM Phase Error for Virtex-5, and new families as Clock Uncertainty
- User-Defined System and Clock Input Jitter as Clock Uncertainty
- Unequal clock duty cycles (non 50%)
- Clock phase including DCM phase and negative edge clocking

Related Timespec Period Constraints

Xilinx recommends that you associate a Period constraint with every clock. The preferred method is to use the Timespec Period constraint. Timespec allows you to define derived clock relationships with other Timespec Period constraints.

An example of this complex derivative relationship is done through the DLL, DCM, PLL, BUFR, PMCD, and MMCM Components component outputs. The derived relationship is defined with one Timespec Period in terms of another Timespec Period. When a data path goes from one clock domain to another clock domain, and the Period constraints are related, the timing tools perform a cross-clock domain analysis. This is common with the outputs from the DLL, DCM, PLL, BUFR, PMCD, and MMCM Components.

For more information about Timing Name and Timing Name Net attributes, see Constraint System in Chapter 3, Timing Constraint Principles.

During cross-clock domain analysis of related Period constraints, the Period constraint on the destination element covers the data path.

Related Period Constraints

In the following figure, **`TS_PERIOD#1`** is related to **`TS_PERIOD#2`**, The data path is analyzed by **`TS_PERIOD#2`**.

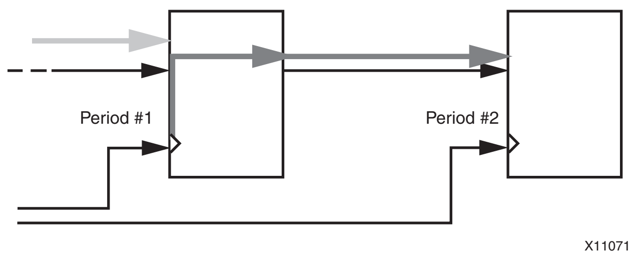


Figure 3-12: **Related Period Constraints**

When Period constraints are related to each other, the design tools can determine the inter-clock domain path requirements. See the following figure.

Period Constraint Syntax

Following is an example of the Period constraint syntax. The **`TS_Period_2`** constraint value is a multiple of the **`TS_Period_1 TIMESPEC`**.

```
TIMESPEC TS_Period_1 = PERIOD "clk1_in_grp" 20 ns HIGH 50%;
TIMESPEC TS_Period_2 = PERIOD "clk2_in_grp" TS_Period_1 * 2;
```

If the two Period constraints are not related in this method, the cross clock domain data paths is not covered or analyzed by any Period constraint.

Unrelated Clock Domains

In the following figure, because CLKA and CLKB are not related or asynchronous to each other, the data paths between register four and register five are not analyzed by either Period constraint.

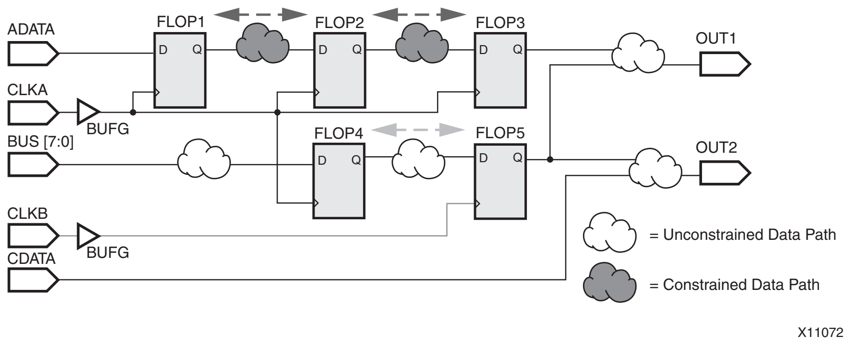


Figure 3-13: **Unrelated Clock Domains**

Paths Covered by Period Constraints

The Period constraint covers paths only between synchronous elements.

The following are *not* included in this analysis.

- Pads

  NGDBuild issues a warning if pad elements are included in the Period time group.
- Analysis between unrelated or asynchronous clock domains

The Period constraint analysis includes the setup and hold analysis on synchronous elements.

Setup Analysis

The setup analysis ensures that the data changes at the destination synchronous element before the clock arrival. The data must become valid at its input pins at least a setup time before the arrival of the active clock edge at its pin.

Setup Analysis Equation

The setup analysis equation is:

```
Setup Time = Data Path Delay + Synchronous Element Setup Time - Clock Path Skew
```

Setup Analysis Timing Report

The Timing Report analysis includes Clock Uncertainty and determines the slack value for the setup analysis. The Data Path includes the Data Path Delay and the Synchronous Element Setup Time.

```
Slack = Requirement - (Data Path - Clock Path Skew + Clock Uncertainty)
```

Setup Analysis Clock Uncertainty

As clock uncertainty increases, the setup margin decreases. See the following figure.

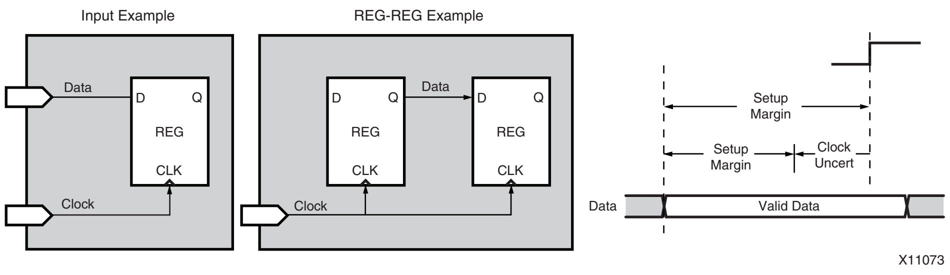


Figure 3-14: **Reduced Setup Margin by Clock Uncertainty/Jitter**

Hold Analysis

The hold analysis ensures that the data changes at the destination synchronous element after the clock arrival. The data must stay valid at its input pins at least a hold time after the arrival of the active clock edge at its pin.

Hold Analysis Equation

The equation for the hold analysis is:

```
Hold Time = Clock Path Skew + Synchronous Element Hold Time - Data Path Delay
```

A hold time violation occurs when the positive clock skew is greater than the data path delay.

Hold Analysis Timing Report

The Timing Report analysis:

- Includes clock uncertainty
- Determines the slack value for the hold analysis.

The data path includes:

- Data path delay
- Synchronous element hold time

```
Slack = Requirement - (Clock Path Skew + Clock Uncertainty - Data Path)
```

Hold Analysis Clock Uncertainty

As clock uncertainty increases, the hold margin decreases. See the following figure.

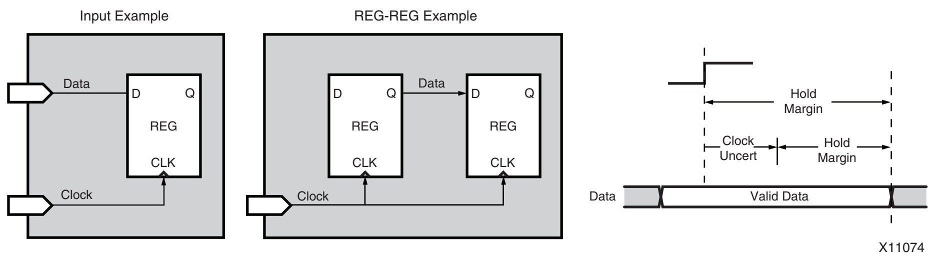


Figure 3-15: **Reduce Hold Margin by Clock Uncertainty/Jitter**

Both equations also include the **`Clock-to-Out`** time of the synchronous source element as a portion of the data path delay.

In the following figure, because the positive clock skew is greater than the data path delay, the timing analysis issues a hold violation.

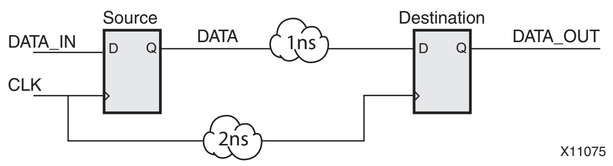


Figure 3-16: **Hold Violation (Clock Skew > Data Path)**

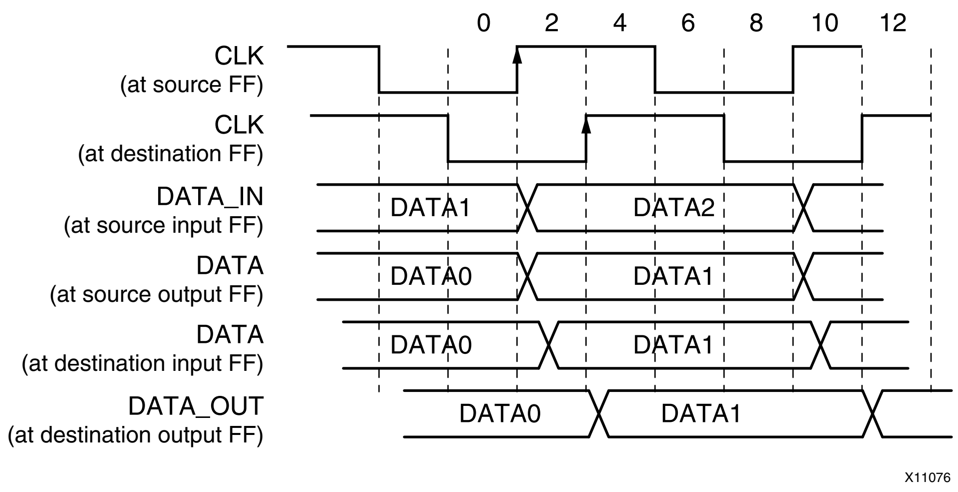


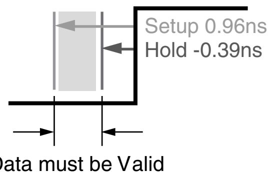
Figure 3-17: **Hold Violation Waveform**

The Timing Report does not list the hold paths unless the path causes a hold violation.

To report the hold paths for each constraint, use the **`-fastpaths`** switch in **`trce`** or **`Report Fast Paths Option`** in Timing Analyzer. The following figure shows an example of setup and hold times from the device data sheet. The setup and hold analysis in the Timing Report is usually smaller than the values in the device data sheet.

While the values in the data sheet cover every pin and synchronous element, the Timing Report is specific to the design for a particular pin or synchronous element.

Offset Constraints



Setup and Hold Times With Respect to Clock at IOB input Register						
Pad, no delay	T_{iopick}/T_{ioickp}	All	0.88/-0.39	0.96/-0.39	1.11/0.45	ns, min

X11077

Figure 3-18: **Setup/Hold Times from Data Sheet**

Offset constraints:

- Are fundamental timing constraints.
- Define the timing relationship between:
    - An external clock pad, and
    - Its associated data-in pad or data-out pad.

This relationship is also known as constraining the *pad to setup* or *clock to out* paths on the device.

Specifying Timing Interfaces With External Components

The following constraints specify timing interfaces with external components.

- Pad to Setup (Offset In Before)

  Allows the external clock and external input data to meet the setup time on the internal flip-flop.
- Clock to Out (Offset Out After)

  Gives you more control over the setup and hold requirement of the downstream devices and with respect to the external output data pad and the external clock pad.
- Offset In Before and Offset Out After

  Allow you to specify the internal data delay from the input pads or to the output pads with respect to the clock.

Specifying External Data and Clock Relationships

Alternatively, the Offset In After and Offset Out Before constraints allow you to specify external data and clock relationship for the timing on the path to the input pads and to the output pads for the Xilinx device.

The timing tools determine the internal requirements without the need of either of the following constraints:

- From PADS To FFS
- From FFS To PADS

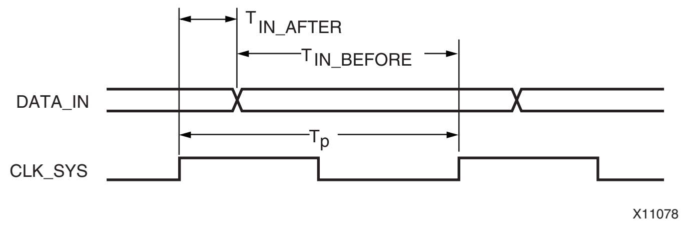


Figure 3-19: **Timing Reference Diagram of Offset In Constraint**

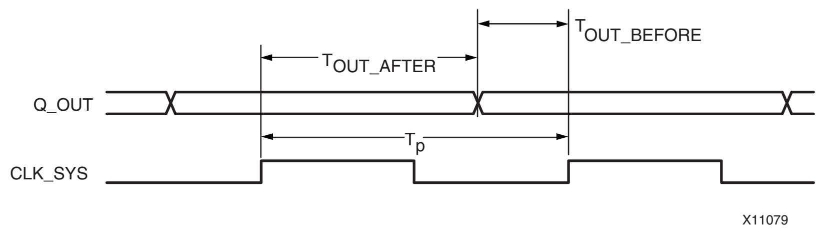


Figure 3-20: **Timing Reference Diagram of Offset Out Constraint**

Included in Offset Constraints

Offset constraints:

- Include clock path delay in the analysis for each individual synchronous element.
- Include paths for all synchronous element types (such as FFS, RAMS, and LATCHES)
- Allow a global syntax that allows all inputs or outputs to be constrained with respect to an external clock.
- Analyze setup and hold time violation on inputs.

Clocking Path Delays

Offset constraints account for the following clocking path delays when defined and analyzed with the Period constraint:

- Provide accurate timing information and use the jitter defined on the associated Period constraint.
- Increase the amount of time for input signals to arrive at synchronous elements (clock and data paths are in parallel).
  - Subtracts the clock path delay from the data path delay for inputs

- Reduce the amount of time for output signals to arrive at output pins (clock and data paths are in series).
  - Adds the clock path delay to the data path delay for outputs
- Include clock phase introduced by a DLL or DCM component for each individual synchronous element defined by the associated Period constraint.
- Include clock phase introduced by a rising or falling clock edge.

Initial Clock Edge

The initial clock edge for analysis of Offset constraints is defined by the High or Low keyword of the Period constraint.

- High keyword => the initial clock edge is *rising*
- Low keyword => the initial clock edge is *falling*

The initial clock edge for analysis of an Offset constraint can override the Period constraint's default clock edge with the following keywords of the Offset constraints:

- Rising keyword => the initial clock edge is *rising*
- Falling keyword => the initial clock edge is *falling*

External Clock Pad and External Data Pads

Offset constraints define the relationship between:

- The external *clock* pad, and
- The external *data* pads

Synchronous Elements

The common component between the external *clock* pad and the external *data* pads are the synchronous elements. If the synchronous element is driven by an internal clock net, a From:To constraint is needed to analyze this data path.

Internal clocks generated by a DCM, PLL, DLL, PMCD, or BUFR component are exceptions to this rule.

From:To Constraint

The From:To constraint provides similar analysis as the Offset constraints in the following situations:

- Calculates whether a setup time is violated at a synchronous element whose data or clock inputs are derived from internal nets
- Specifies the delay of an external output net derived from the Q output of an internal synchronous element that is clocked from an internal net

Paths Covered by Offset Constraints

Offset constraints cover the following paths:

- From input pads to synchronous elements (Offset In)
- From synchronous elements to output pads (Offset Out)

See the following figure.

Offset constraints do not return any paths during timing analysis if the clock net that clocks a synchronous element does not come from an input pad (for example, if it is derived from another clock or from a synchronous element).

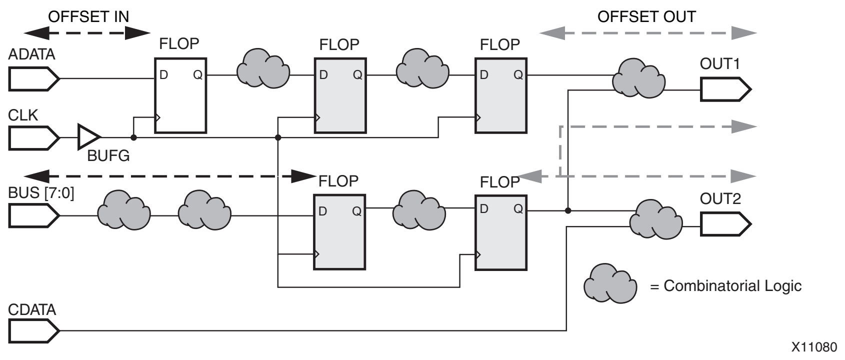


Figure 3-21: **Circuit Diagram of Offset Constraints**

The Offset constraint is analyzed with respect to a single clock edge only. If the Offset constraint must analyze multiple clock phases or clock edges, as in source synchronous designs or dual-data rate applications, the Offset constraint must be manually adjusted by the clock phase.

The Offset constraint does not optimize paths clocked by an internally generated clock. Use From:To or Multi-Cycle constraints for these paths, taking into account the clock delay.

I/O Timing Analysis

Use the following option to obtain I/O timing analysis on internal clocks or derived clocks:

- Create a From:To or Multi-Cycle constraint on these paths, or
- Determine if the internal clock is related to an external clock signal.
- Change the requirement based upon the relationship between the two clocks.

  For example, if:

  - The internal clock is a divide by two version of the external clock, and
  - The original requirement of the Offset Out with the internal clock was 10 ns, then
  - The requirement of the Offset Out with the external clock is 20 ns.

Levels of Coverage

You can specify Offset constraints in three levels of coverage:

- Global Offset

  Applies to all inputs or outputs for a specific clock.
- Group Offset

  Identifies a group of input or outputs clocked by a common clock, that have the same timing requirement.
- Net-Specific Offset

  Specifies the timing by each input or output.

Group and Global Offset Constraints

Offset constraints with a *specific* scope override Offset constraints with a *general* scope.

- A Group Offset constraint overrides a Global Offset constraint for the same I/O.
- A Net-Specific Offset constraint overrides both Global Offset constraints and Group Offset constraints.

This priority allows you to start with Global Offset constraints, then create Group Offset constraints or Net-Specific Offset constraints for I/O with special timing requirements.

Reducing Memory Usage and Runtime

Use Global Offset constraints and Group Offset constraints to reduce memory usage and runtime.

Using wildcards in a Net-Specific Offset constraint creates multiple Net-Specific Offset constraints, not a Group Offset constraint.

Register Groups and Pad Groups

Group Offset constraints can include both a register group and a pad group. Group Offset constraints allow you to group pads or registers, or both, to use the same requirement.

- Register groups

  Identify path sources or destinations that have different requirements *from* or *to* a *single pad* on a clock edge.

- Pad groups

  Identify path sources or destinations that have different requirements *from* or *to* a *group of pads* on the same clock edge.

You can group and constrain the pads and registers all at once, which is useful if a clock is used on the rising and falling edge for inputs and outputs.

Rising and Falling Groups

The rising and falling groups require different Group Offset constraints.

In the following figure, registers A, B, and C are different time groups, even though these registers have the same data and clock source.

The different time groups are:

- TIMEGRP AB = RISING FFS;
- TIMEGRP C = FALLING FFS;

This allows you to perform two different timing analyses for these registers.

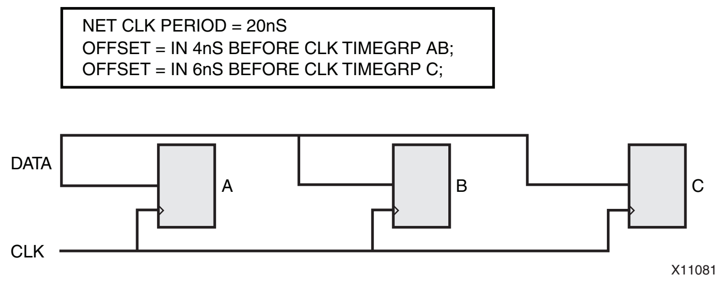

NET CLK PERIOD = 20nS
OFFSET = IN 4nS BEFORE CLK TIMEGRP AB;
OFFSET = IN 6nS BEFORE CLK TIMEGRP C;
DATA
CLK
A
B
C
X11081

Figure 3-22: **Offset with Different Timing Groups**

CPLD Designs

For CPLD designs, clock inputs referenced by Offset constraints must be explicitly assigned to a global clock pin using either a BUFG symbol or applying the BUFG=CLK constraint to an ordinary input. Otherwise, the Offset constraint is not used during timing driven optimization.

From:To (Multi-Cycle) Constraints

A Multi-Cycle path is path that is allowed to take multiple clock cycles.

These types of paths are typically covered by a Period constraint by default. They may cause errors because a Period constraint is a one-cycle constraint.

To eliminate these errors, place a specific Multi-Cycle constraint on the path. This removes the path from the Period constraint.

Multi-Cycle Constraints

Multi-Cycle constraints:

- Are applied by using a From:To constraint.
- Have a higher priority than Period constraints and Offset constraints. It pulls paths out of the lower priority constraints and the paths are analyzed by the Multi-Cycle constraints.
- Can be tighter or looser than lower priority constraints.
- Constrain a specific path.

The specific path can:

- Be within the same clock domain, but
- Have a different requirement than the Period constraint.

Alternatively, the specific path with a data path, which crosses clock domains are constrained with a Multi-Cycle constraint.

From:To Constraints

From:To constraints:

- Have a higher priority than Period constraints.
- Remove the specified paths from the Period constraint to the From:To constraint.
- Begin at a synchronous element and end at a synchronous element.

For example, if a portion of the design must run slower than the Period requirement, use a From:To constraint for the new requirement.

The Multi-Cycle path can also mean that there is more than one cycle between each enabled clock edge.

Declaring Start and End Points

When using a From:To constraint, specify the constrained paths by declaring the start points and the end points.

The start points and the end points must be:

- Pre-specified time groups (such as PADS, FFS, LATCHES, RAMS), or
- User-specified time groups, or
- User-specified synchronous points (see TPSYNC).

From or To Optional

From or To is optional when constraining a specific path.

- From Multi-Cycle constraint

  Covers a From or source timing group to the next synchronous elements or pads elements.
- To Multi-Cycle constraint

  Covers all previous synchronous elements or pad elements to a To or destination time group.

Following are some possible combinations:

- From:To
- From:Thru:To
- Thru:To
- From:Thru
- From
- To
- From:Thru:Thru:Thru:To

A From:To constraint can cover the Multi-Cycle paths that cover the path between clock domains. For example, while one clock covers a portion of the design, and another clock covers the rest, some paths go between these two clock domains. See the following figure.

You must have a clear idea of the design specifics, and take into account the multiple clock domains.

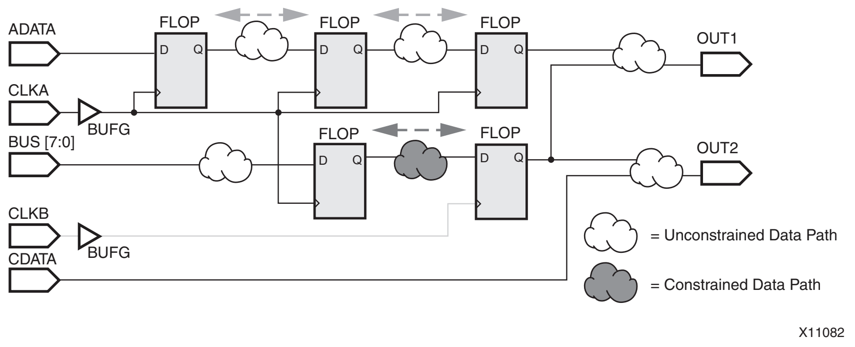


Figure 3-23: **Multi-Cycle Constraint Covers a Cross Clock Domain Path**

Cross Clock Domain Paths

The cross clock domain paths between unrelated Period constraints are analyzed in the Unconstrained Paths Report.

Create a Multi-Cycle or From:To constraint if these paths are related incorrectly, or if they require a different timing requirement.

The From:To constraint can be a specific value, related to another Timing Specifications identifier, or Timing Ignore. A path can be ignored during timing analysis with the label of Timing Ignore.

Create a From:To constraint if the clocks are unrelated by the definition of the constraints, but have valid paths between them.

Constraining Paths Between Two Clock Domains

To constrain the paths between two clock domains:

1. Create time groups based upon each clock domain.
2. Create a From:To for each direction that the paths pass between the two clock domains.

The following example shows a cross clock domain using a From:To constraint.

```
TIMESPEC TS_clk1_to_clk2 = FROM clk1 TO clk2 8 ns;
```

Constrain from time group **`clkA`** to timing group **`clkB`** to be 8 ns. See the following figure.

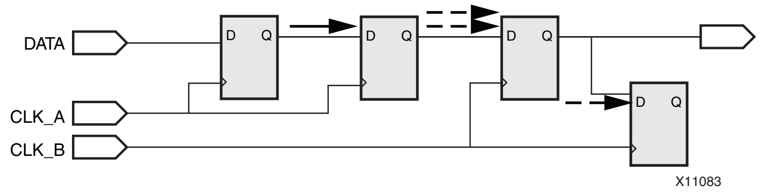


Figure 3-24: **Cross Clock Domain Path Analyzed Between CLK_A Clock Domain and CLK_B Clock Domain**

Pad-to-Pad Path

The Pad-to-Pad path (or asynchronous path) is a fundamental From:To constraint. The From:Pads:To:Pads constraint covers the combinatorial path with the Pad instances of the design as the start points and end points. Because these types of paths are asynchronous, they are usually left unconstrained. See the following figure.

Following is an example of this type of constraint:

```
TIMESPEC TS_Pad2Pad = FROM PADS TO PADS 14.4 ns;
```

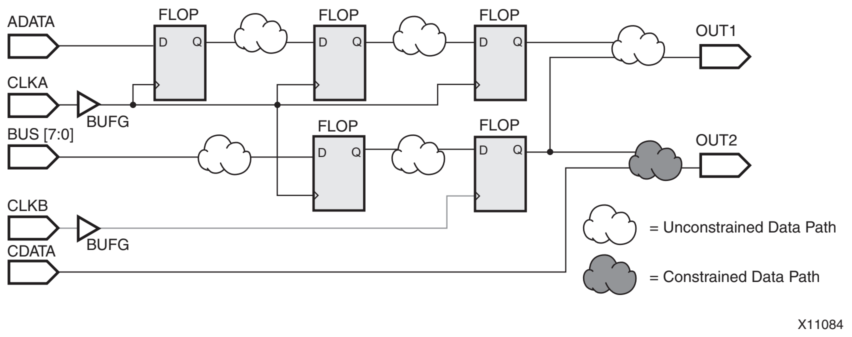


Figure 3-25: **Pad-to-Pad Multi-Cycle Constraint Covers Path**

Slow Exceptions

In addition to using Multi-Cycle constraints in the Pad-to-Pad path, you can use Multi-Cycle constraints to define a **slow** exception. This is an exception from the Period constraint, which constrains the majority of the design.

Using a From:To Slow Exception in Conjunction with Period Constraint

The following figure shows the use of a From:To **slow** exception in conjunction with a Period constraint. Xilinx does not recommend this method for slow exceptions.

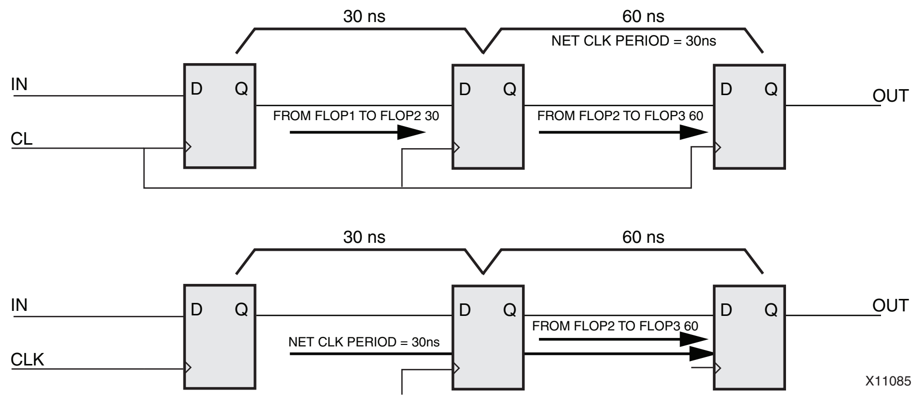


Figure 3-26: **Slow Exception Multi-Cycle Constraint Overlaps a Period Constraint**

Using a Clock Enable Net to Define a Slow Exception

A Clock Enable net can define a **slow** exception. See the following figure. Xilinx recommends this method for **`slow`** exceptions, which is based upon timing groups

```
NET clk_en TNM = slow_exception;
NET clk TNM = normal;
TIMESPEC TS01 = PERIOD normal 8 ns;
TIMESPEC TS02 = FROM slow_exception TO slow_exception TS01*2;
```

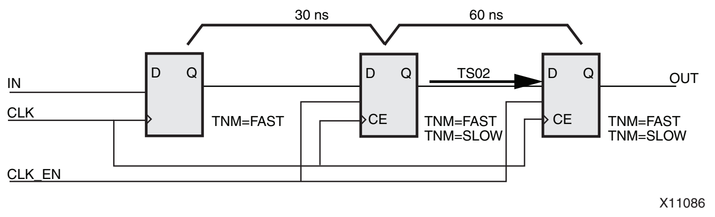


Figure 3-27: **Slow Time Group Overlaps the Fast Time Group for a From:To Exception**

Ignoring a Path

Use a Timing Ignore constraint to ignore a path between **`flopa`** and **`flopb`** passing through net **`netand`**. See Figure 3-28, Ignore a Path Between Registers.

To create this from the From:To:TIG constraint:

1. Tag **`flopa`** for time group **`FFA_grp`**
2. Tag **`flopb`** for time group **`FFB_grp`**
3. Create the following constraint:

```
TIMESPEC TS_FFA_to_FFB = FROM FFA_grp TO FFB_grp TIG;
```

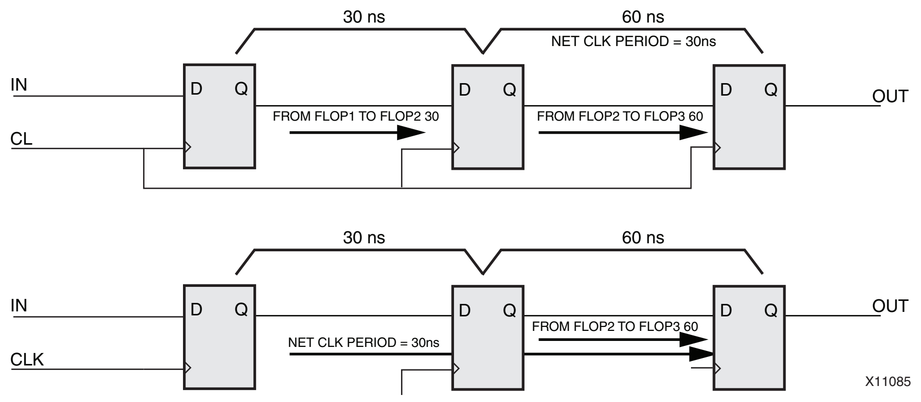


Figure 3-28: **Ignore a Path Between Registers**

If a specific path must be constrained at a faster or slower than the Period constraint, create a From:To for that path. If there are multiple paths between a source and destination synchronous elements, create a From:Thru:To constraint to capture specific paths.

This constraint applies to a specific path that:

1. Begins at a source time group, then
2. Passes through intermediate points, and
3. Ends at a destination time group.

The source and destination time groups can be:

- User-defined time groups, or
- Predefined time groups

The intermediate points of the path are defined using the Tpthru constraint. There is no limitation on the number of intermediate points in a From:To constraint.

From:Thru:To Constraint Example

Following is an example of a From:Thru:To constraint:

```
NET $3M17/On_the_Way TPTHRU = abc;
TIMESPEC TS_mypath = FROM my_src_grp THRU abc TO my_dest_grp 9 ns;
```

Constrain from time group **`my_src_grp`** through thru group **`abc`** to the time group **`my_dest_grp`** to be 9 ns.

- The **`my_src_grp`** constrains the FIFO shown in the following figure.
- The **`my_dest_grp`** constrains the registers shown in the following figure.

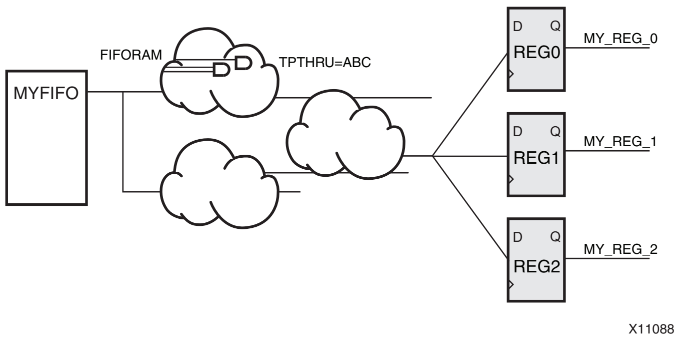


Figure 3-29: **Net Tpthru Example with Previous From:Thru:To Constraint Example**

False Paths or Timing Ignore Constraint

A Net Timing Ignore constraint:

- Covers a specific net.
- Marks nets that are to be ignored for timing analysis purposes.

A From:To TIG constraint:

- Covers several paths between two synchronous groups or pad groups.
- Marks all nets going between the synchronous groups that are to be ignored for timing analysis purposes.

See the following figure.

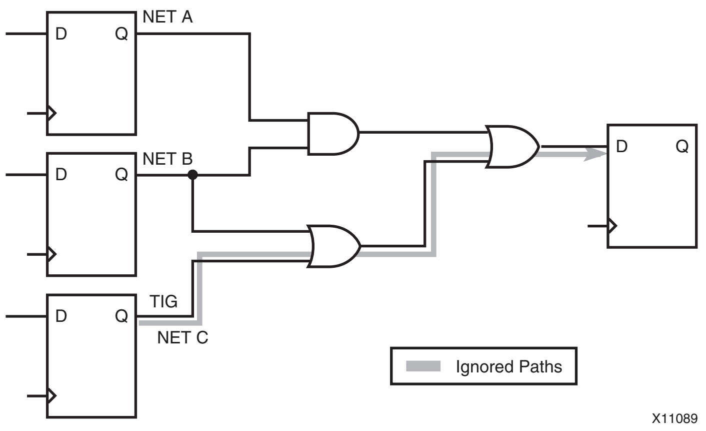


Figure 3-30: **Timing Ignore on a Path Between Two Flip-Flops**

Defining a Non-Synchronous Path

Use the From:Thru:To constraint to define a non-synchronous path, such as using a common bus for several modules.

The timing analysis constrains between these modules, even though the modules do not interact with each other.

Because these modules do not interact with each other, you can use a constraint, or set the From:To constraint to a large requirement. The following figure shows an example.

Common Bus is the Through Point Example

```
NET DATA_BUS* TPTHRU = DataBus;
TIMESPEC TS_TIG = FROM FFS THRU DataBus TO FFS TIG;
OR
TIMESPEC TS_data_bus = FROM FFS THRU DataBus TO FFS 123ns;
```

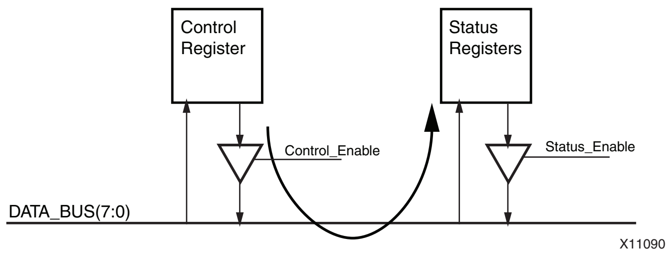


Figure 3-31: **Common Bus is the Through Point**

Applying a Timing Point Synchronization Constraint

In addition to using a Timing Thru Points (Tpthru) constraint, you can apply a Timing Point Synchronization (TPSYNC) constraint to specific pins or combinatorial logic in order to force the timing analysis to stop or start at a non-synchronous point.

Constraint to Three-State Buffer With From:To

The Timing Point Synchronization constraint defines non-synchronous points as synchronous points for Multi-Cycle constraints and analysis. The path to a three-state buffer, for example, can be constrained with the Timing Point Synchronization constraint.

The following figure shows an example of constraining the path to the three-state buffer.

```
NET $3M17/Blue TPSYNC = Blue_S;
TIMESPEC TS_1A = FROM FFS TO Blue_S 15;
```

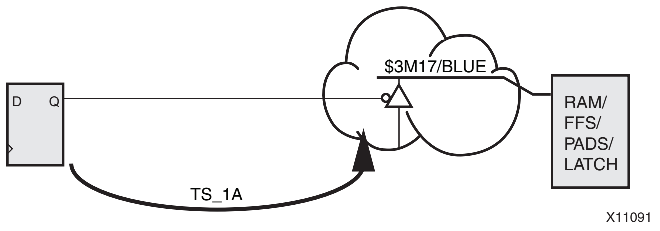


Figure 3-32: **Constraint to Three-State Buffer With From:To**

Paths Covered by From:To Constraints

The From:To constraint:

- Defines a timing requirement between two timing groups.
- Is used in conjunction with the Period and Offset In and Offset Out constraints
- Is used to define the fast and slow exceptions.

The From:To constraint is highly versatile. See the following figure for examples for a simple design in:

```
TIMESPEC TS_C2S = FROM FFS TO FFS 12 ns;
TIMESPEC TS_P2S = FROM PADS TO FFS 10 ns;
TIMESPEC TS_P2P = FROM PADS TO PADS 13 ns;
TIMESPEC TS_C2P = FROM FFS TO PADS 8 ns;
```

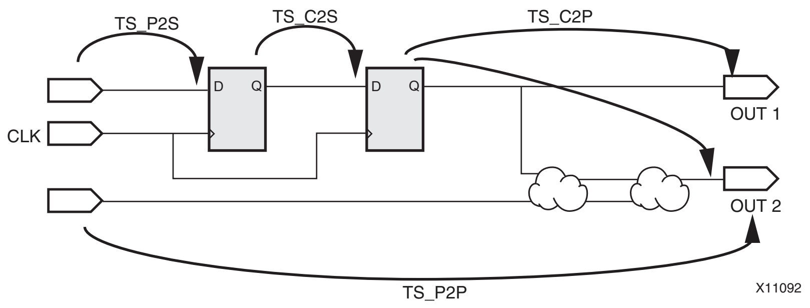


Figure 3-33: **All Paths Constrained on a Sample Design**

Changing Analysis from *Period* to *From:To*

When changing analysis from a Period constraint to From:To constraint, the number of paths analyzed can be larger than the situation in which:

- A path is covered with a Period constraint, but
- The number of unconstrained paths does not increase.

The destination Timing Group for the From:To constraint probably contains distributed Dual-Port Synchronous RAM components. Paths to these RAM components are both *synchronous* and *asynchronous*.

- The path to the data input (D) is *synchronous*.
- The paths to the read address inputs (DPRA) are *asynchronous*.

A Period constraint constrains only *synchronous* paths. A From:To constraint constrains both the *synchronous* and *asynchronous* paths to this RAM component.

- A path from a flip-flop to the D input of this RAM component is a *synchronous* path. This data path is covered by a Period or a From:To constraint.
- A path from a flip-fop to the DPRA input of this RAM component is an *asynchronous* path to the read address input. This data path is covered only by a From:To constraint.

Grouping Constraint Syntax

For information about grouping constraint syntax, see the following constraints in the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

- Timing Name
- Timing Name Net
- Timing Group
- Period
- Offset In
- Offset Out
- From:To (Multi-Cycle)

Creating Timing Constraints

You can create timing constraints in two ways:

- In the HDL design. See:
  - Specifying Timing Constraints in XST
  - Specifying Timing Constraints in Synplify
- With Constraints Editor (UCF)

Creating Timing Constraints With Constraints Editor

Constraints Editor uses information from the NGD file to create constraints in the UCF file. Because Constraints Editor parses the NGD file, the exact UCF syntax for each design element and constraint is used by the implementation tools.

The Constraints Editor allows you to create time groups and timing constraints. Because the clocks and I/O components are supplied, you do not need the exact spelling of the names. You need define only the timing requirements, and not the syntax, of the constraints.

When creating specific time groups, element names are provided, and exceptions to the global constraints can be made using those groups.

Condensing the Size of the Time Groups

The Constraints Editor does not create time groups or constraints with wildcards. You must manually modify the UCF to condense the size of the time groups. The condensing of the size of the time groups in the UCF is done with wildcards on the unique portions of the design element and the common portion remains.

Condensed Time Groups Example

```
INST my_bus* TNM = my_output_bus_grp;
```

The asterisk (*) wildcard causes the constraint system to apply the Timing Name attribute to all instances with the base name **my_bus**.

Chapter 4

Specifying Timing Constraints in XST

This chapter discusses the methods for specifying timing constraints in the Xilinx® Synthesis Tool (XST).

For information on specifying timing constraints in Synplify, see Chapter 5, Specifying Timing Constraints in Synplify.

Syntax Examples for XST Timing Constraints below gives syntax examples for individual Xilinx timing constraints in VHDL, Verilog, and XCF.

For more information, see:

- *Synthesis and Simulation Design Guide (UG626)*, cited in Appendix A, Additional Resources.
- *XST User Guide for Virtex®-6, Spartan®-6, and 7 Series Devices (UG687)*, cited in Appendix A, Additional Resources.

Applying XST Timing Constraints

Apply XST timing constraints with:

- The **-glob_opt** command line switch, or
- The XST Constraint File (XCF)

Since timing constraints specified in the source code do not propagate to the netlist, all timing constraints must be specified in the UCF.

Timing Model

The timing model used by XST for timing analysis takes into account:

- Logic delays

  Logic delays data are identical to the delays reported by TRACE (Timing Analyzer) after Place and Route.
- Net delays

  The Net delay model is estimated based on the fanout load.

These delays are:

- Highly dependent on the speed grade that can be specified to XST.
- Dependent on the selected technology.

XCF Constraint Priority

Constraints are processed in the following order:

1. Specific constraints on signals
2. Specific constraints on top module
3. Global constraints on top module

For example, constraints on two different domains or two different signals have the same priority (that is, PERIOD clk1 can be applied with PERIOD clk2).

Methods for Specifying Timing Constraints in XST

Use any of the following to specify timing constraints in XST:

- Hardware Description Language (HDL) code
- XST Constraint File (XCF)
- The **-global_opt** command line switch

To specify timing constraints before synthesis:

- Specify the timing constraints in your design:
    - HDL
        - VHDL
        - Verilog
    - Schematic

  OR
- Specify the timing constraints in an XCF.

Specifying Timing Constraints in HDL

Timing constraints specified in HDL are written in the style of the attributes.

Specifying Timing Constraints in XCF

XST supports an XST Constraints File (XCF) syntax to specify synthesis and timing constraints. The constraint file method allows you to use the native XCF timing constraint syntax.

XST Supported XCF Constraints

Using the XCF syntax, XST supports constraints such as:

- Timing Name Net
- Timing Group
- Period
- Timing Ignore
- From-To

This includes wildcards and hierarchical names.

Timing constraints are not written to the NGC file by default. Timing constraints are written to the NGC file only when:

- **Write Timing Constraints** is checked in ISE in **Process > Properties**, or
- The **-write_timing_constraints** option is specified in the command line.

XCF Syntax Limitations

XCF syntax has the following limitations:

- Nested model statements are not supported.
- Instance or signal names listed between the BEGIN MODEL statement and the END statement are only those visible inside the entity.
- Hierarchical instance or signal names are not supported.

Specifying Timing Constraints Using the -glob_opt Command Line Switch

Timing constraints supported by XST can also be applied using the **`-glob_opt`** command line switch. Using the **`-glob_opt`** command line switch is the same as selecting:

Process > Properties > Synthesis Options > Global Optimization Goal.

This method allows you to apply global timing constraints to the entire design. You cannot specify a value for these constraints. XST optimizes them for the best performance. These constraints are overridden by constraints specified in the constraints file.

Syntax Examples for XST Timing Constraints

The sections below give syntax examples for individual Xilinx timing constraints in VHDL, Verilog, and an XCF file. Not all constraints give examples of all three methods.

- Asynchronous Register
- Clock Signal
- Maximum Delay
- Maximum Delay
- Maximum Skew
- Offset
- Period
- System Jitter
- NET/PIN/INST Timing Ignore
- Timing Group
- Timing Specifications
- Timing Name
- Timing Name Net

If you specify timing constraints in the XCF file, Xilinx strongly suggests that you to use the forward slash (/) character as a hierarchy separator, instead of the underscore character (_).

Asynchronous Register

The Asynchronous Register (ASYNC_REG) constraint can be attached only on registers or latches with asynchronous input (D input or the CE input).

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Asynchronous Register VHDL Syntax

```
attribute ASYNC_REG : string;
attribute ASYNC_REG of instance_name: signal is "{TRUE|FALSE}";
```

Asynchronous Register VHDL Syntax Example

```
architecture behavioral of top_yann_mem_infrastructure is
begin
signal sys_rst      : std_logic;
attribute ASYNC_REG : string;
attribute ASYNC_REG of sys_rst: signal is "TRUE";
--source code
End behavioral;
```

Asynchronous Register Verilog Syntax

```
(* ASYNC_REG = "{TRUE|FALSE}" *)
```

Asynchronous Register Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
  );
 wire clk_0;
 wire clk_90;
 wire clk_200;
(* ASYNC_REG = "TRUE" *)
 reg sys_rst;
// source code
End module;
```

Clock Signal

Clock Signal applies to all FPGA devices. Clock Signal does not apply to CPLD devices.

If a clock signal goes through combinatorial logic before being connected to the clock input of a flip-flop, XST cannot identify which input pin or internal signal is the real clock signal. The Clock Signal (CLOCK_SIGNAL) constraint allows you to define the clock signal.

Clock Signal VHDL Syntax

```
attribute clock_signal : string;
attribute clock_signal of signal_name : signal is "{yes|no}";
```

Clock Signal VHDL Syntax Example

```
entity top_yann_mem is
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);
SYS_CLK_P : in std_logic;
SYS_CLK_N : in std_logic;
CLK200_P : in std_logic;
CLK200_N : in std_logic
  );
attribute clock_signal : string;
attribute clock_signal of clk200_p : signal is "yes";
end entity;
```

Clock Signal Verilog Syntax

```
(* clock_signal = "{yes|no}" *)
```

Clock Signal Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
  );
(* clock_signal = "yes" *)
wire clk_0;
 wire clk_90;
 wire clk_200;
 reg sys_rst;
// source code
End module;
```

Clock Signal XCF Syntax

```
BEGIN MODEL "entity_name"
NET "primary_clock_signal" clock_signal={yes|no|true|false};
END;
```

Clock Signal XCF Syntax Example

```
BEGIN MODEL "top_yann_mem"
NET "CLK200_P" clock_signal = yes;
END;
```

Maximum Delay

The Maximum Delay (MAXDELAY) constraint:

- Defines the maximum allowable delay on a net.
- Applies to the nets in FPGA devices only.

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Maximum Delay VHDL Syntax

```
attribute maxdelay of signal_name: signal is "value [units]";
```

where

- **`value`** is a positive integer;

  Valid units are ps, ns, us, ms, Hz, kHz, MHz. The default is ns.

Maximum Delay VHDL Syntax Example

```
entity top_yann_mem_data_path_iobs_0 is
port (
   CLK      : in std_logic;
   dqs_delayed  : out std_logic_vector(31 downto 0);
   READ_EN_DELAYED_RISE : out std_logic_vector(31 downto 0);
   READ_EN_DELAYED_FALL : out std_logic_vector(31 downto 0);
   );
attribute maxdelay: string;
attribute maxdelay of READ_EN_DELAYED_RISE: signal is "800 ps";
attribute maxdelay of READ_EN_DELAYED_FALL: signal is "800 ps";
end entity;
```

Maximum Delay Verilog Syntax

```
(*MAXDELAY = "value [units]" *)
```

Maximum Delay Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input       sys_clk_p,
  input       sys_clk_n,
  input       clk200_p,
  input       clk200_n,
  input       sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
  );
wire clk_0;
 wire clk_90;
 wire clk_200;
 (*MAXDELAY= " 800 ps" *)
 wire read_en;
 reg sys_rst;
// source code
End module;
```

Maximum Skew

The Maximum Skew (MAXSKEW) constraint controls the amount of skew on a net. Skew is the difference between the delays of all loads driven by the net.

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Maximum Skew VHDL Syntax

```
attribute maxskew: string;
attribute maxskew of signal_name : signal is "allowable_skew [units]";
```

where

- **`allowable_skew`** is the timing requirement
- valid units are ms, micro, ns, or ps. The default is ns.

Maximum Skew VHDL Syntax Example

```
entity top_yann_mem_infrastructure is
port (
   SYS_CLK_P: in std_logic;
   SYS_CLK_N: in std_logic;
   CLK200_P: in std_logic;
   CLK200_N: in std_logic;
   CLK         : out std_logic;
   REFRESH_CLK     : out std_logic;
   sys_rst       : out std_logic;
   );
attribute maxskew: string;
attribute maxskew of sys_rst : signal is "3 ns";
end entity;
```

Maximum Skew Verilog Syntax

```
(* MAXSKEW = "allowable_skew [units]" *)
```

where

- **`allowable_skew`** is the timing requirement
- valid units are ms, micro, ns, or ps. The default is ns.

Maximum Skew Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
   );
wire clk_0;
 wire clk_90;
 wire clk_200;
 (*MAXSKEW= " 3 ns" *)
 wire read_en;
 reg sys_rst;
// source code
End module;
```

Offset

The Offset (OFFSET) constraint:

- Specifies the timing relationship between:
  - An external clock, and
  - Its associated data-in or data-out pin.
- Is used only for pad related signals.
- Cannot be used to extend the arrival time specification method to the internal signals.

For more information, see Chapter 3, Timing Constraint Principles.

Offset XCF Syntax

```
OFFSET = {IN|OUT} offset_time [units] {BEFORE|AFTER}
clk_name [TIMEGRP group_name];
```

where

- offset_time [units]

  The difference in time between the capturing clock edge and the start of the data to be captured.

  The time can be specified with or without explicitly declaring the units. If no units are specified, the default value is nanoseconds (ns). The valid values are ps, ns, micro, and ms.
- BEFORE | AFTER

  Defines the timing relationship of the start of data to the clock edge.

  The best method of defining the clock and data relationship is to use the BEFORE option. BEFORE describes the time the data begins to be valid relative to the capturing clock edge.
  - *Positive* values of BEFORE indicate that the data begins *before* the capturing clock edge.
  - *Negative* values of BEFORE indicate that the data begins *after* the capturing clock edge.
- clk_name

  Defines the fully hierarchical name of the input clock pad net.
- Valid keyword

  Not applicable to the Offset constraint.

Offset XCF Syntax Example

```
OFFSET = IN 2 ns BEFORE "CLK200_N" ;
OFFSET = IN 3.85 ns BEFORE "SYS_CLK_P" ;
OFFSET = OUT 4 ns AFTER "CLK200_N" ;
OFFSET = OUT 7 ns AFTER "SYS_CLK_P" ;
NET "main_00/top_00/iobs_00/data_path_iobs_00/v4_dq_iob_0/DDR_DQ" TNM=
DDR2_DQ_Grp;
OFFSET = OUT 6.7 ns AFTER "SYS_CLK_P" TIMEGRP DDR2_DQ_Grp;
OFFSET = IN 3.2 ns BEFORE "SYS_CLK_P" TIMEGRP DDR2_DQ_Grp ;
```

Period

The Period (PERIOD) constraint is a basic timing constraint and synthesis constraint.

A clock Period specification checks timing between all synchronous elements within the clock domain as defined in the destination element group. The Period specification is attached to the clock net.

The timing analysis tools:

- Take into account any inversions of the clock net at register clock pins and clock phase.
- Include all synchronous item types in the analysis.
- Check for hold violations.

For more information, see Chapter 3, Timing Constraint Principles.

Period VHDL Syntax

The Period constraint applies only to a specific clock signal.

```
attribute period: string;
attribute period of signal_name : signal is "period [units]";
```

Period VHDL Syntax Example

```
entity top_yann_mem is
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);
SYS_CLK_P : in std_logic;
SYS_CLK_N : in std_logic;
CLK200_P : in std_logic;
CLK200_N : in std_logic
  );
attribute period: string;
attribute period of SYS_CLK_P : signal is "5 ns";
end entity;
```

Period Verilog Syntax

The Period constraint applies only to a specific clock signal.

```
(* PERIOD = "period [units]" *)
```

where

- **`period`** is the required clock period
- **`units`** is an optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ns, or micro to indicate the intended units.

Period Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
  );
(*PERIOD = "5 ns"*)
wire clk_0; // The clk_0 is assigned with the period of 5 ns
 wire clk_90;
 wire clk_200;
 wire read_en;
 reg sys_rst;
// source code
End module;
```

Timing Specifications Period XCF Syntax

This is the primary method for specifying Period XCF syntax. Xilinx recommends this version.

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference period" [units]
[{HIGH |LOW} [high_or_low_time [hi_lo_units]]] INPUT_JITTER value [units];
```

NET Period XCF Syntax

This is the secondary method for specifying Period XCF syntax. Xilinx DOES NOT recommend this version.

```
NET "net_name" PERIOD=period [units]
[{HIGH|LOW}[high_or_low_time[hi_lo_units]]];
```

where

- **`identifier`** is a reference identifier that has a unique name
- **`TNM_reference`** is the identifier name that is attached to a clock net (or a net in the clock path) using the Timing Name or Timing Name Net constraint. When a Timing Name Net constraint is traced into the CLKIN input of a DLL, DCM or PLL component, new Period specifications may be created at the DLL, DCM, or PLL outputs.
- **`period`** is the required clock period.
- **`units`** is an optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ms, micro, or % to indicate the intended units.
- **`HIGH`** or **`LOW`** indicates whether the first pulse is to be High or Low.

  `HIGH` and **`LOW`** values are not taken into account during timing estimation and optimization. They are propagated to the final netlist only if **`WRITE_TIMING_CONSTRAINTS = yes`**.

- **`high_or_low_time`** is the optional **`HIGH`** or **`LOW`** time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no **`high_or_low_time`** is specified, the default duty cycle is 50 percent.
- **`hi_lo_units`** is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the **`high_or_low_time`** number can be followed by ps, micro, ms, or % if the **`HIGH`** or **`LOW`** time is an actual time measurement.

The following statement assigns a clock period of 40 ns to the net named **`CLOCK`**, with the first pulse being **`HIGH`** and having duration of 25 nanoseconds.

```
NET "CLOCK" PERIOD=40 HIGH 25;
```

The following statement assigns a clock period of 5 ns in the style of Timing Specifications.

```
NET "infrastructure0/SYS_CLK_IN" TNM_NET = "SYS_CLK";
TIMESPEC "TS_SYS_CLK" = PERIOD "SYS_CLK" 5 ns HIGH 50 %;
```

System Jitter

The System Jitter (SYSTEM_JITTER) constraint specifies the system jitter of the design. System Jitter depends on various design conditions, such as the number of flip-flops changing at one time and the number of I/Os changing.

System Jitter applies to all clocks within a design. System Jitter can be combined with the Input Jitter keyword on the Period constraint to generate the Clock Uncertainty value shown in the Timing Report.

For more information, see Chapter 3, Timing Constraint Principles.

System Jitter is another way to specify an additional timing margin where there is no real way to characterize the jitter of the system. This constraint is useful to test the limitations of designs with a tight timing margin. System Jitter is used within the clock uncertainty calculation for all constraints that analyze a clock in the design.

Some devices have a default System Jitter included in the speed files. This can be checked by using SpeedPrint.

Another way to perform the same test is to modify the Input Jitter for a specific input clock. This works only for a specific clock domain rather than the full system.

System Jitter VHDL Syntax

```
attribute SYSTEM_JITTER: string;
attribute SYSTEM_JITTER of
{component_name|signal_name|entity_name|label_name}:
{component|signal|entity|label} is "value ps";
```

where

- *value* is a numerical value. The default is ps.

System Jitter VHDL Syntax Example

```
entity top_yann_mem is
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);
SYS_CLK_P : in std_logic;
SYS_CLK_N : in std_logic;
CLK200_P : in std_logic;
CLK200_N : in std_logic
  );
attribute SYSTEM_JITTER : string;
attribute SYSTEM_JITTER of top_yann_mem: entity is "10 ps";
end entity;
```

System Jitter Verilog Syntax

```
(* SYSTEM_JITTER = "value ps" *)
```

where

- *value* is a numerical value. The default is ps.

System Jitter Verilog Syntax Example

```
module mig_22
 ( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0]  cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
  );
(*SYSTEM_JITTER = "10 ps"*)
wire clk_0; // The clk_0 is assigned with system_jitter of 10 ps
 wire clk_90;
 wire clk_200;
 wire read_en;
 reg sys_rst;
// source code
End module;
```

System Jitter XCF Syntax

```
MODEL "entity_name" SYSTEM_JITTER = value ps;
```

System Jitter XCF Syntax Example

```
MODEL "top_yann_mem" SYSTEM_JITTER = 10;
```

NET/PIN/INST Timing Ignore

The Timing Ignore (TIG) constraint:

- Is a basic timing constraint.
- Is a synthesis constraint.
- Applies to FPGA devices only.
- Does not apply to CPLD device.

Timing Ignore causes paths that fan forward from the point of application (of Timing Ignore) to be treated as if they do not exist (for the purposes of the timing model) during implementation.

For more information, see Chapter 3, Timing Constraint Principles.

Timing Ignore XCF Syntax

```
NET "net_name" TIG;
PIN "ff_inst.RST" TIG=TS_1;
INST "instance_name" TIG=TS_2;
TIG=TSidentifier1,..., TSidentifiern
```

where

- **`identifier`** refers to a timing specification that should be ignored

When attached to an instance, Timing Ignore is pushed to the output pins of that instance. When attached to a net, Timing Ignore pushes to the drive pin of the net. When attached to a pin, Timing Ignore applies to the pin.

Timing Ignore XCF Syntax Example

```
NET "main_?0/top_?0/ddr2_controller_?0/load_mode_reg*" TIG;
```

The following statement specifies that the timing specifications TS_fast and TS_even_faster are ignored on all paths fanning forward from the net RESET.

```
NET "RESET" TIG=TS_fast, TS_even_faster;
```

Timing Group

The Timing Group (TIMEGRP) constraint is a basic grouping constraint.

In addition to naming groups using the Timing Name identifier, you can also define groups in terms of other groups. Place Timing Group constraints in a constraints file such as an XST Constraint File (XCF) or a Netlist Constraints File (NCF).

For more information, see Chapter 3, Timing Constraint Principles.

Timing Group XCF Syntax

```
TIMEGRP newgroup = existing_grp1 existing_grp2 [existing_grp3 ...];
```

where

- **`newgroup`** is a newly created group that consists of existing groups created by means of Timing Name constraints, predefined groups or other TIMEGRP attributes

Timing Group XCF Syntax Example

```
TIMEGRP Top_Group = GroupA GroupB GroupC;
```

Multi-Cycle Path

The Multi-Cycle Path constraint specifies a timing constraint between two groups.

For more information, see Chapter 3, Timing Constraint Principles.

Multi-Cycle Path XCF Syntax

```
TIMESPEC TSname =FROM "group1" TO "group2" value;
```

where

- **`TSname`** must always begin with **`TS`**. Any alphanumeric character or underscore may follow.
- **`group1`** is the source timing group
- **`group2`** is the destination timing group
- **`value`** is **`ns`** by default. Other possible values are **`MHz`** or another timing specification such as **`TS_C2S/2`** or **`TS_C2S*2.`**

XST supports the From-To constraint with the following limitations:

- From-Thru-To is not supported
- Linked timing specification is not supported
- Pattern matching for predefined groups is not supported, such as:

  ```
  TIMESPEC TS_1 = FROM FFS(machine/*) TO FFS 2 ns;
  ```

Multi-Cycle Path XCF Syntax Example

```
TIMESPEC TS_MY_PathA = FROM "my_src_grp" TO "my_dst_grp" 23.5 ns;
TIMESPEC TS_ DQS_UNUSED = FROM FFS TO "control_unused_dqs" TIG;
```

Timing Specifications

The Timing Specifications (TIMESPEC) constraint:

- Is a basic timing related constraint.
- Serves as a placeholder for timing specifications (TS attribute definitions).

A TS attribute:

- Defines the allowable delay for paths.
- Begins with the letters **TS.**
- Ends with a unique identifier that can consist of:
    - Letters
    - Numbers
    - The underscore character (_).

The number of Timing Specification constraints can significantly impact the runtime and memory usage of the implementation and analysis tools.

Timing Specifications XCF Syntax

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" value units;
```

where

- **TSidentifier** is a unique name for the **TS** attribute
- *value* is a numerical value. It defines the maximum delay for the attribute. Nanoseconds (ns) are the default units for specifying delay time in **TS** attributes. You can also specify delay with other units, such as picoseconds (ps) or megahertz (MHz).
- *units* can be ms, micro, ps, or ns.

Keywords, such as FROM, TO, and TS, appear in the documentation in upper case. You can specify them in the Timing Specifications primitive in either upper or lower case.

Timing Specifications XCF Syntax Examples

- Defining a Maximum Allowable Delay Timing Specifications XCF Syntax Example
- Defining a Clock Period XCF Syntax Example
- Specifying Derived Clocks XCF Syntax Example
- Timing Ignore Paths XCF Syntax Examples

Defining a Maximum Allowable Delay Timing Specifications XCF Syntax Example

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" allowable_delay [units];
```

Defining a Clock Period XCF Syntax Example

Defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period.

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" value [units] [{HIGH | LOW}
[high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

where

- **`identifier`** is a reference identifier with a unique name
- **`TNM_reference`** is the identifier name attached to a clock net (or a net in the clock path) using a Timing Name constraint
- **`value`** is the required clock period
- **`units`** is an optional field to indicate the units for the allowable delay. The default units are nanoseconds (ns), but the timing number can be followed by micro, ms, ps, ns, GHz, MHz, or kHz to indicate the intended units
- **`HIGH`** or **`LOW`** can be optionally specified to indicate whether the first pulse is to be High or Low
- **`high_or_low_time`** is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.
- **`hi_lo_units`** is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, micro, ms, ns or % if the High or Low time is an actual time measurement.

Specifying Derived Clocks XCF Syntax Example

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" "another_PERIOD_identifier" [/ | *] number
[{HIGH | LOW} [high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

where

- **`TNM_reference`** is the identifier name attached to a clock net (or a net in the clock path) using a Timing Name constraint
- **`another_PERIOD_identifier`** is the name of the identifier used on another period specification
- **`number`** is a floating point number
- **`HIGH`** or **`LOW`** can be optionally specified to indicate whether the first pulse is to be High or Low
- **`high_or_low_time`** is the optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.
- **`hi_lo_units`** is an optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the High or Low time number can be followed by ps, micro, ms, or % if the High or Low time is an actual time measurement.

Timing Ignore Paths XCF Syntax Examples

This form is not supported for CPLD devices.

There are situations in which a path that exercises a certain net should be ignored because all paths through the net, instance, or instance pin are not important from a timing specification point of view.

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" TIG;
```

where

- **`identifier`** is an ASCII string made up of the characters A-Z, a-z 0-9, and _
- **`source_group`** and **`dest_group`** are user-defined or predefined groups

The following statement indicates that the timing specification **TS_35** calls for a maximum allowable delay of 50 ns between the groups **`here`** and **`there`**.

```
TIMESPEC "TS_35"=FROM "here" TO "there" 50;
```

The following statement indicates that the timing specification **TS_70** calls for a 25 ns clock period for **clock_a**, with the first pulse being High for a duration of 15 ns.

```
TIMESPEC "TS_70"=PERIOD "clock_a" 25 high 15;
```

Timing Name

The Timing Name (TNM) constraint:

- Is a basic grouping constraint.
- Identifies the elements that make up a group for use in a timing specification.
- Tags specific predefined groups as members of a group to simplify the application of timing specifications to the group.
- Supports the Rising and Falling keywords.

For more information, see Chapter 3, Timing Constraint Principles.

Timing Name XCF Syntax

```
{NET|INST|PIN} "net_or_pin_or_inst_name" TNM=[predefined_group] identifier;
```

where

- **predefined_group** can be all the members or a subset of a predefined group using the keywords FFS, RAMS, LATCHES, PADS, CPUS, HSIOS, BRAMS_PORTA, BRAMS_PORTB, DSPS, and MULTS
- **`identifier`** can be any combination of letters, numbers, or underscores.

Timing Name XCF Syntax Example

```
NET clk TNM = FFS (my_flop) Grp1;
INST clk TNM = FFS (my_macro) Grp2;
```

Timing Name Net

The Timing Name Net (TNM_NET) constraint identifies the elements that make up a group for use in a timing specification.

Timing Name Net is essentially equivalent to Timing Name on a net except for input pad nets.

For more information, see Chapter 3, Timing Constraint Principles.

Timing Name Net XCF Syntax

```
{NET|INST} "net_name" TNM_NET= [predefined_group] identifier;
```

where

- **`predefined_group`** can be all the members of a predefined group using the keywords FFS, RAMS, PADS, MULTS, HSIOS, CPUS, DSPS, BRAMS_PORTA, BRAMS_PORTB or LATCHES. A subset of elements in a **`predefined_group`** can be defined as follows:
    - **`predefined_group (name_qualifier1... name_qualifiern)`**
    - **`-name_qualifiern`** can be any combination of letters, numbers, or underscores. The **`name_qualifier`** type (net or instance) is based on the element type that Timing Name Net is placed on. If the Timing Name Net is on a NET, the **name_qualifier** is a net name. If the Timing Name Net is an instance (INST), the **name_qualifier** is an instance name.
- **`identifier`** can be any combination of letters, numbers, or underscores

  The identifier cannot be any the following reserved words: FFS, RAMS, LATCHES, PADS, CPUS, HSIOS, MULTS, RISING, FALLING, TRANSHI, TRANSLO, or EXCEPT.

XST supports Timing Name Net with the limitation that only a single pattern is supported for predefined groups.

Table 4-1: **Timing Name Net Support Limitations**

Supported	NET "PADCLK" TNM_NET=FFS "GRP1"; #
Not supported	NET "PADCLK" TNM_NET = FFS(machine/*:xcounter/*) TG1; #

Timing Name Net XCF Syntax Example

```
NET clk TNM_NET = FFS (my_flop) Grp1;
INST clk TNM_NET = FFS (my_macro) Grp2;
```

Chapter 5

Specifying Timing Constraints in Synplify

This chapter discusses the methods for specifying timing constraints in Synplify. For information on the methods for specifying timing constraints in the Xilinx® Synthesis Tool (XST), see Chapter 4, Specifying Timing Constraints in XST.

The sections below give syntax examples for individual Xilinx timing constraints in VHDL and Verilog.

For more information, see:

- *Synthesis and Simulation Design Guide (UG626), cited in Appendix A, Additional Resources*
- *Synopsys FPGA Synthesis Reference Manual*

The methods for specifying timing constraints in Synplify are:

- Specifying Timing Constraints in HDL
- Specifying Timing Constraints in an .sdc File (Tcl)
- Specifying Timing Constraints in a SCOPE Spreadsheet

If there are multiple timing exception constraints on the same object, the synthesis tool uses the guidelines described in “Conflict Resolution for Timing Exceptions” in the *Synopsys FPGA Synthesis Reference Manual*, to determine which constraint takes precedence.

Constraint Types

Table 5-1, Constraint Types for Each Timing Constraint Entry in Synplify, lists the timing constraints and related commands in alphabetical order, according to the methods used to enter them. The timing constraints for HDL are all directives.

Table 5-1: **Constraint Types for Each Timing Constraint Entry in Synplify**

HDL	Tcl (.sdc File)	SCOPE
black_box_tri_pins		
	define_clock	Clocks Panel
	define_clock_delay	Clock to Clock Panel
	define_compile_point	Compile Points Panel
	define_current_design	
	define_false_path	False Paths Panel
	define_input_delay	Inputs/Outputs Panel
	define_io_standard	I/O Standard Panel
	define_multicycle_path	Multi-Cycle Paths Panel
	define_output_delay	Inputs/Outputs Panel
	define_path_delay	Max Delay Paths Panel
	define_reg_input_delay	Registers Panel
	define_reg_output_delay	Registers Panel
syn_force_seq_prim *		
syn_gatedclk_clock_en *		
syn_gatedclk_clock_en_polarity *		
syn_isclock		
syn_tpdn		
syn_tcon		
syn_tsun		

* This constraint is available in Synplify Pro and Synplify Premier only.

Specifying Timing Constraints in HDL

Write source code attributes or directives in Hardware Description Language (HDL) code.

Enter black box timing directives in the source code. Do not include any other timing constraints in the source code. The source code becomes less portable, and you must recompile the design for the constraints to take effect.

You can also enter *attributes* using a SCOPE Spreadsheet. You must use source code for *directives*.

Syntax Examples for HDL Timing Constraints

The following sections give syntax examples for HDL timing constraints

- black_box_pad_pin
- black_box_tri_pins
- syn_force_seq_prim
- syn_gatedclk_clock_en
- syn_gatedclk_clock_en_polarity
- syn_isclock
- syn_tpdn
- syn_tcon
- syn_tsun

black_box_pad_pin

The **`black_box_pad_pin`** directive specifies pins on a user-defined black box component as I/O pads visible to the environment outside the black box.

If more than one port is an I/O pad, list the ports:

- Inside double-quotes separated by commas
- Without enclosed spaces

black_box_pad_pin Verilog Syntax

```
object /* synthesis syn_black_box black_box_pad_pin = "portList" */ ;
```

where

- **`portList`** is a spaceless, comma-separated list of the names of the ports on black boxes that are I/O pads.

black_box_pad_pin Verilog Syntax Example

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_pad_pin="GIN[2:0],Q" */;
```

black_box_pad_pin VHDL Syntax

```
attribute black_box_pad_pin of object : objectType is "portList" ;
```

where

- **`object`** is an architecture or component declaration of a black box. Data type is string.
- **`portList`** is a spaceless, comma-separated list of the black box port names that are I/O pads.

black_box_pad_pin VHDL Syntax Example

```
library ieee;
use ieee.std_logic_1164.all;
package my_components is
component BBDLHS
port (D: in std_logic;
E: in std_logic;
GIN : in std_logic_vector(2 downto 0);
Q : out std_logic );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of BBDLHS : component is "GIN(2:0),Q";
end package my_components;
```

black_box_tri_pins

The **`black_box_tri_pins`** directive specifies that an output port on a component defined as a black box is a tristate. The **`black_box_tri_pins`** directive eliminates multiple driver errors when the output of a black box has more than one driver. A multiple driver error is issued unless you use the **`black_box_tri_pins`** directive to specify that the outputs are tristates.

If there is more than one port that is a tristate, list the ports:

- Inside double-quotes separated by commas
- Without enclosed spaces

black_box_tri_pins Verilog Syntax

```
object /* synthesis syn_black_box black_box_tri_pins = "portList" */ ;
```

where

- **`portList`** is a spaceless, comma-separated list of multiple pins.

black_box_tri_pins Verilog Syntax Example

Following is a **`black_box_tri_pins`** Verilog syntax example with a single port name.

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_tri_pins="PAD" */;
Here is an example with a list of multiple pins:
module bb1(D,E,tri1,tri2,tri3,Q)
/* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
For a bus, specify the port name followed by all the bits on the bus:
module bb1(D,bus1,E,GIN,GOUT,Q)
/* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```

black_box_tri_pins VHDL Syntax

```
attribute black_box_tri_pins of object : objectType is "portList" ;
```

where

- **`object`** is a component declaration or architecture. Data type is string.
- **`portList`** is a spaceless, comma-separated list of the tristate output port names

black_box_tri_pins VHDL Syntax Example

```
library ieee;
use ieee.std_logic_1164.all;
package my_components is
component BBDLHS
port (D: in std_logic;
E: in std_logic;
GIN : in std_logic;
GOUT : in std_logic;
PAD : inout std_logic;
Q: out std_logic );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_tri_pins : string;
attribute black_box_tri_pins of BBDLHS : component is "PAD";
end package my_components;
```

Multiple pins on the same component can be specified as a list:

```
attribute black_box_tri_pins of bb1 : component is "tri,tri2,tri3";
```

To apply this directive to a port that is a bus, specify all bits on the bus:

```
attribute black_box_tri_pins of bb1 : component is "bus1[7:0]";
```

syn_force_seq_prim

The **syn_force_seq_prim** directive indicates that gated clocks should be fixed for this black box, and the fix gated clocks algorithm can be applied to the associated primitive. The **syn_force_seq_prim** directive is available only in Synplify Pro and Synplify Premier.

To use the **syn_force_seq_prim** directive with a black box, you must also identify the clock signal with the **syn_isclock** directive and the enable signal with the **syn_gatedclk_clock_en** directive. The data type is Boolean.

syn_force_seq_prim Verilog Syntax

```
object /* synthesis syn_force_seq_prim = 1 */ ;
```

where

- **object** is the module name of the black box

syn_force_seq_prim Verilog Syntax Example

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */ ;
input clk /* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

syn_force_seq_prim VHDL Syntax

```
attribute syn_force_seq_prim of object: objectType is true ;
```

where

- **object** is the entity name of the black box.

syn_force_seq_prim VHDL Syntax Example

```
library ieee;
use ieee.std_logic_1164.all;
entity bbram is
port (addr: IN std_logic_VECTOR(6 downto 0);
din: IN std_logic_VECTOR(7 downto 0);
dout: OUT std_logic_VECTOR(7 downto 0);
clk: IN std_logic;
en: IN std_logic;
we: IN std_logic);
attribute syn_black_box : boolean ;
attribute syn_black_box of bbram : entity is true ;
attribute syn_isclock : boolean;
attribute syn_isclock of clk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of clk : signal is "en";
end entity bbram;
architecture bb of bbram is
attribute syn_force_seq_prim : boolean;
attribute syn_force_seq_prim of bb : architecture is true;
begin
end architecture bb;
```

syn_gatedclk_clock_en

The **`syn_gatedclk_clock_en`** directive specifies the enable pin to be used in fixing the gated clocks. To use the **`syn_gatedclk_clock_en`** directive with a black box, you must:

- Identify the clock signal with the **`syn_isclock`** directive, and
- Indicate that the fix gated clocks algorithm can be applied with the **`syn_force_seq_prim`** directive.

The data type is **`string`**.

syn_gatedclk_clock_en Verilog Syntax

```
object /* synthesis syn_gatedclk_clock_en = "value" */ ;
```

where

- **`object`** is the module name
- **`value`** is the name of the enable pin

syn_gatedclk_clock_en Verilog Syntax Example

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */;
input clk
/* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

syn_gatedclk_clock_en VHDL Syntax

```
attribute syn_gatedclk_clock_en of object: objectType is value ;
```

where

- **`object`** is the entity name of the black box

syn_gatedclk_clock_en VHDL Syntax Example

```
architecture top of top is component bbram
port (myclk : in bit;
opcode : in bit_vector(2 downto 0);
a, b : in bit_vector(7 downto 0);
rambus : out bit_vector(7 downto 0) );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of bbram: component is true;
attribute syn_force_seq_prim : boolean
attribute syn_force_seq_prim of bbram: component is true;
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of bbram: signal is "ena
//Other code
```

syn_gatedclk_clock_en_polarity

The **`syn_gatedclk_clock_en_polarity`** directive indicates the polarity of the clock enable port on a black box. This allows the synthesis tool to apply the algorithm to fix gated clocks. If you do not set any polarity with this attribute, the synthesis tool assumes a positive polarity by default.

syn_gatedclk_clock_en_polarity Verilog Syntax

```
object /* synthesis syn_gatedclk_clock_en_polarity = 1 | 0 */ ;
```

where

- **`object`** is the module name of the black box.

The value can be 1 or 0. A value of 1 indicates positive polarity of the enable signal (active high) and a value of 0 indicates negative polarity (active low). If the attribute is not defined, the synthesis tool assumes a positive polarity by default.

syn_gatedclk_clock_en_polarity Verilog Syntax Example

```
module bbe1 (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */;
input clk /* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */
/* synthesis syn_gatedclk_clock_en_polarity = 0 */;
input data_in,ena;
output data_out;
endmodule
```

syn_gatedclk_clock_en_polarity VHDL Syntax

```
attribute syn_gatedclk_clock_en_polarity of object: objectType is true
| false;
```

syn_gatedclk_clock_en_polarity VHDL Syntax Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity bbe1 is
port (ena : in std_logic;
clk : in std_logic;
data_in : in std_logic;
data_out : out std_logic );
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_gatedclk_clock_en_polarity of clk: signal is false;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
end bbe1;
```

```
architecture arch_bbe1 of bbe1 is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbe1: architecture is true;
attribute syn_force_seq_prim of arch_bbe1: architecture is true;
begin
end arch_bbe1;
```

syn_isclock

The **`syn_isclock`** directive specifies an input port on a black box as a clock. Use the **`syn_isclock`** directive to specify that an input port on a black box is a clock, even though its name does not correspond to a recognized name. Using the **`syn_isclock`** directive connects it to a clock buffer if appropriate. The data type is Boolean.

syn_isclock Verilog Syntax

```
object /* synthesis syn_isclock = 1 */ ;
```

where

- **`object`** is an input port on a black box

syn_isclock Verilog Syntax Example

```
module ram4 (myclk,out,opcode,a,b) /* synthesis syn_black_box */;
output [7:0] out;
input myclk /* synthesis syn_isclock = 1 */;
input [2:0] opcode;
input [7:0] a, b;
//Other code
```

syn_isclock VHDL Syntax

```
attribute syn_isclock of object: objectType is true ;
```

where

- **`object`** is a black box input port

syn_isclock VHDL Syntax Example

```
library synplify;
entity ram4 is
port (myclk : in bit;
opcode : in bit_vector(2 downto 0);
a, b : in bit_vector(7 downto 0);
rambus : out bit_vector(7 downto 0) );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
// Other code
```

syn_tpd*n*

The **syn_tpd*n*** directive supplies information on timing propagation for combinational delay through a black box. The **syn_tpd*n*** directive can be entered as an attribute using the Attribute panel of the SCOPE editor. The information in the object, attribute, and value fields must be manually entered.

syn_tpd*n* Verilog Syntax

```
object /* syn_tpdn = "bundle -> bundle = value" */ ;
```

where

- **bundle** is a collection of buses and scalar signals

  To assign values to bundles, use the following syntax. The values are in **ns**.

  ```
  "bundle -> bundle = value"
  ```

The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C which lists three signals.

syn_tpd*n* Verilog Syntax Example

The following example defines **syn_tpd*n*** along with other black box timing constraints:

```
module ram32x4(z,d,addr,we,clk); /* synthesis syn_black_box
syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0"
syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

syn_tpd*n* VHDL Syntax

```
attribute syn_tpdn of object : objectType is "bundle -> bundle = value"
;
```

where

- **bundle** is a collection of buses and scalar signals.

To assign values to **bundle**, use the following syntax. The values are in **ns**.

```
"bundle -> bundle = value"
```

The objects of a **bundle** must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

syn_tpd*n* VHDL Syntax Examples

In VHDL, there are ten predefined instances of each directive in the Synplify library. For example:

```
syn_tpd1, syn_tpd2, syn_tpd3, … syn_tpd10
```

If you are entering the timing directives in the source code and you require more than ten different timing delay values for any one of the directives, declare the additional directives with an integer greater than ten.

```
attribute syn_tpd11 : string;
attribute syn_tpd11 of bitreg : component is "di0,di1 -> do0,do1 = 2.0";
attribute syn_tpd12 : string;
attribute syn_tpd12 of bitreg : component is "di2,di3 -> do2,do3 = 1.8";
```

The following example assigns **syn_tpd*n*** together with some of the black box constraints.

```
-- A USE clause for the Synplify Attributes package was included
-- earlier to make the timing constraint definitions visible here.
architecture top of top is
component rcf16x4z
port (ad0, ad1, ad2, ad3 : in std_logic;
di0, di1, di2, di3 : in std_logic;
clk, wren, wpe : in std_logic;
tri : in std_logic;
do0, do1, do2, do3 : out std_logic );
end component;
attribute syn_tpd1 of rcf16x4z : component is
"ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is "ad0,ad1,ad2,ad3 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is "wren,wpe -> clk = 0.0";
// Other code
```

sdc File Syntax

```
define_attribute {v:blackboxModule} syn_tpdn { bundle -> bundle = value}
```

where

- **v:** indicates that the directive is attached to the view
- **blackboxModule** is the symbol name of the black-box
- **n** is a numerical suffix that lets you specify different input to output timing delays for multiple signals/bundles
- **bundle** is a collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A, B, C, which lists three signals.
- **value** is input to output delay value in ns

sdc File Syntax example

```
define_attribute {v:MEM} syn_tpd1 {MEM_RD->DATA_OUT[63:0]=20}
```

syn_tco*n*

The **syn_tco*n*** directive supplies the clock to output timing delay through a black box. The **syn_tco*n*** directive can be entered as an attribute using the Attribute panel of the SCOPE editor. The information in the object, attribute, and value fields must be manually entered.

syn_tco*n* Verilog Syntax

```
object /* syn_tcon = "[!]clock -> bundle = value" */ ;
```

where

- **bundle** is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are in **ns**.

  `"[!]clock -> bundle = value"`
- **!** is an optional exclamation mark indicating a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

syn_tco*n* Verilog Syntax Example

Following is an example defining **syn_tco*n*** with other black box constraints.

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0"
syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

syn_tco*n* VHDL Syntax

```
attribute syn_tcon of object : objectType is "[!]clock -> bundle =
value" ;
```

where

- **bundle** is a collection of buses and scalar signals. To assign values to **bundle**, use the following syntax. The values are in **ns**.

  `"[!]clock -> bundle = value"`
- **!** is an optional exclamation mark indicating a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

In VHDL, there are ten predefined instances of each directives in the Synplify library. For example:

```
syn_tco1, syn_tco2, syn_tco3, … syn_tco10
```

If you are entering the timing directives in the source code and you require more than ten different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10.

syn_tco*n* VHDL Syntax Examples

```
attribute syn_tco11 : string;
attribute syn_tco11 of bitreg : component is "clk -> do0,do1 = 2.0";
attribute syn_tco12 : string;
attribute syn_tco12 of bitreg : component is "clk -> do2,do3 = 1.8";
```

The following example assigns **syn_tco*n*** along with other black box constraints.

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component Dpram10240x8
port (
-- Port A
ClkA, EnA, WeA: in std_logic;
AddrA : in std_logic_vector(13 downto 0);
DinA : in std_logic_vector(7 downto 0);
DoutA : out std_logic_vector(7 downto 0);
-- Port B
ClkB, EnB: in std_logic;
AddrB : in std_logic_vector(13 downto 0);
DoutB : out std_logic_vector(7 downto 0) );
end component;
attribute syn_black_box : boolean;
attribute syn_tsu1 : string;
attribute syn_tsu2 : string;
attribute syn_tco1 : string;
attribute syn_tco2 : string;
attribute syn_isclock : boolean;
attribute syn_black_box of Dpram10240x8 : component is true;
attribute syn_tsu1 of Dpram10240x8 : component is
"EnA,WeA,AddrA,DinA -> ClkA = 3.0";
attribute syn_tco1 of Dpram10240x8 : component is "ClkA -> DoutA[7:0] = 6.0";
attribute syn_tsu2 of Dpram10240x8 : component is "EnB,AddrB -> ClkB = 3.0";
attribute syn_tco2 of Dpram10240x8 : component is "ClkB -> DoutB[7:0] = 13.0";
// Other code
```

syn_tco*n* sdc File Syntax

```
define_attribute {v:blackboxModule} syn_tcon { [!]clock -> bundle =
value}
```

where

- **v:** indicates that the directive is attached to the view
- **blackboxModule** is the symbol name of the black box
- **n** is a numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles
- **!** is an optional exclamation mark indicating that the clock is active on its falling (negative) edge
- **clock** is the name of the clock signal

- **`bundle`** is a collection of buses and scalar signals.

  The objects of a **`bundle`** must be separated by commas with no intervening spaces. A valid bundle is A, B, C, which lists three signals.

- **`value`** is the clock to output delay value in ns

syn_tco*n* sdc File Syntax Example

```
define_attribute {v:RCV_CORE} syn_tco1 {CLK-> R_DATA_OUT[63:0]=20}
define_attribute {v:RCV_CORE) syn_tco2 {CLK-> DATA_VALID=30<n>
```

syn_tsu*n*

The **`syn_tsun`** directive:

- Supplies information on timing setup delay required for input pins (relative to the clock) in a black box.
- Can be entered as an attribute using the Attribute panel of the SCOPE editor.

The information in the object, attribute, and value fields must be manually entered.

syn_tsu*n* Verilog Syntax

```
object /* syn_tsun = "bundle -> [!]clock = value" */ ;
```

where

- **`bundle`** is a collection of buses and scalar signals

  To assign values to bundles, use the following syntax. The values are in **`ns`**.

  ```
  "bundle -> [!]clock = value"
  ```

- **`!`** is an optional exclamation mark indicating a negative edge for a clock.

  The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

syn_tsu*n* Verilog Syntax Example

The following example defines **`syn_tsun`** together with other black box constraints:

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0" syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

syn_tsu*n* VHDL Syntax

```
attribute syn_tsun of object : objectType is "bundle -> [!]clock = value" ;
```

In VHDL, there are ten predefined instances of each directive in the Synplify library. For example:

```
syn_tsu1, syn_tsu2, syn_tsu3, … syn_tsu10
```

If you are entering the timing directives in the source code and you require more than ten different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10.

syn_tsu*n* VHDL Syntax Examples

```
attribute syn_tsu11 : string;
attribute syn_tsu11 of bitreg : component is "di0,di1 -> clk = 2.0";
attribute syn_tsu12 : string;
attribute syn_tsu12 of bitreg : component is "di2,di3 -> clk = 1.8";
```

where

- **`bundle`** is a collection of buses and scalar signals.

  To assign values to bundles, use the following syntax. The values are in **`ns`**.

  ```
  "bundle -> [!]clock = value"
  ```

- **`!`** is an optional exclamation mark indicating a negative edge for a clock.

  The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals

In addition to the syntax used in the code above, you can also use the following Verilog-style syntax to specify this attribute:

```
attribute syn_tsu1 of inputfifo_coregen : component is "rd_clk->dout[48:0]=3.0";
```

The following example assigns **`syn_tsun`** together with other black box constraints:

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component rcf16x4z
port (ad0, ad1, ad2, ad3 : in std_logic;
di0, di1, di2, di3 : in std_logic;
clk, wren, wpe : in std_logic;
tri : in std_logic;
do0, do1, do2, do3 : out std_logic );
end component;
attribute syn_tco1 of rcf16x4z : component is
"ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is "ad0,ad1,ad2,ad3 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is "wren,wpe -> clk = 0.0";
// Other code
```

syn_tsu*n* sdc File Syntax

```
define_attribute {v:blackboxModule} syn_tsun { bundle -> [!]clock =
value}
```

where

- **`v:`** indicates that the directive is attached to the view
- **`blackboxModule`** is the symbol name of the black box
- **`nA`** is a numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles
- **`!`** is an optional exclamation mark indicating that the clock is active on its falling (negative) edge
- **`clock`** is the name of the clock signal

- **`bundle`** is a collection of buses and scalar signals.

  The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals.
- **`valueInput`** is the clock setup delay value in **`ns`**

syn_tsu*n* sdc File Syntax Example

```
define_attribute {v:RTRV_MOD} syn_tsu4 {RTRV_DATA[63:0]->!CLK=20}
```

Specifying Timing Constraints in an .sdc File (Tcl)

Write Tcl commands in an .sdc (Tcl) file.

Constraint files have an `.sdc` file extension. They can include timing constraints, general attributes, and vendor-specific attributes.

Create the `.sdc` file manually in a text editor. Use a SCOPE Spreadsheet to generate the constraint syntax.

The following sections lists each type of Tcl timing constraints in detail.

- define_clock
- define_clock_delay
- define_compile_point
- define_current_design
- define_false_path
- define_input_delay
- define_io_standard
- define_multicycle_path
- define_output_delay
- define_path_delay
- define_reg_input_delay
- define_reg_output_delay

define_clock

The **`define_clock`** constraint defines a clock with a specific duty cycle and frequency or clock period goal. You can have multiple clocks with different clock frequencies.

Use the **`set_option -frequency`** Tcl command in the project file to set the default frequency for all clocks. If you do not specify a global frequency, the timing analyzer uses a default.

Use the **`define_clock timing`** constraint to:

- Override the default, and
- Specify unique clock frequency goals for specific clock signals.

You can also use **`define_clock`** to set the clock frequency for a clock signal output of clock divider logic. The clock name is the output signal name for the register instance.

define_clock Syntax

```
define_clock [ -disable ] [ -virtual] {clockObject} [ -freq MHz | -period ns ] [ -clockgroup
domain ] [ -rise value -fall value] [ -route ns ] [-name clockName] [ -comment textString ]
```

where

- **`disable`** disables a previous clock constraint
- **`virtual`** specifies arrival and required times on top level ports that are enabled by clocks external to the chip (or block) that you are synthesizing.

  When specifying **`-name`** for the virtual clock, the field can contain a unique name not associated with any port or instance in the design.
- **`clockObject`** is a required parameter that specifies the clock object name

  Clocks can be defined on the following:

  - Top-level input ports (p:)
  - Nets (n:)
  - Hierarchical ports (t:)
  - Instances (i:)

  For Xilinx technologies, specify the **`define_clock`** constraint on an instance.

  - Output pins of instantiated cells (t:)
  - Internal pins of instantiated cells (t:)

  Clocks defined on any of the following WILL NOT be honored:

  - Top-level output ports
  - Input pins of instantiated gates
  - Pins of inferred instances
- **`name`** specifies a name for the clock if you want to use a name other than the clock object name. This alias name also appears in the Timing Reports.
- **`freq`** defines the frequency of the clock in MHz. You can specify either **`freq`** or **`period`**, but not both.
- **`period`** specifies the period of the clock in ns. Specify either **`period`** or **`freq`**, but not both.
- **`clockgroup`** allows you to specify clock relationships

You assign related (synchronized) clocks to the same clock group and unrelated clocks in different groups. The synthesis tool calculates the relationship between clocks in the same clock group, and analyzes all paths between them. Paths between clocks in different groups are ignored (false paths).

- **`rise/fall`** specifies a non-default duty cycle

  By default, the synthesis tool assumes that the clock is a 50% duty cycle clock, with the rising edge at 0 and the falling edge at period/2. If you have another duty clock cycle, specify the appropriate Rise At and Fall At values.

- **`route`** is an advanced user option that improves the path delays of all registers controlled by this clock

  The value of **`route`** is the difference between the synthesis Timing Report path delays and the value in the Place and Route Timing Report.

  The **`route`** constraint applies globally to the clock domain, and can over-constrain registers where constraints are not needed.

  Before you use this option, evaluate the path delays on individual registers in the optimization Timing Report and try to improve the delays by applying the constraints **`define_reg_input_delay`** and **`define_reg_output_delay`** only on the registers that need them.

define_clock Syntax Examples

In the following example, a clock is defined on the Q pins of instances **`myInst1`** and **`myInst2`**.

```
define_clock {CLK1} -period 10.0 -clockgroup default_clkgroup
define_clock {CLK3} -period 5.0 -clockgroup default_clkgroup
-uncertainty 0.2 -name INT_REF3
define_clock -virtual {CLK2} -period 20.0 -clockgroup g2
define_clock {CLK4} -period 20.000 -clockgroup g3 -rise 1.000 -fall
11.000 -ref_rise 0.000 -ref_fall 10.000
define_clock Pin-Level Constraint Examples
define_clock {i:myInst1.Q} -period 10.000 -clockgroup default -rise
0.200 -fall 5.200 -name myff1
define_clock {i:myInst2.Q} -period 12.000 -clockgroup default -rise
0.400 -fall 5.400 -name myff2
```

define_clock_delay

The **`define_clock_delay`** command defines the delay between the clocks. By default, the synthesis tool calculates clock delay based on the clock parameters you define with the **`define_clock`** command. However, if you use **`define_clock_delay`**, the specified delay value overrides any calculations made by the synthesis tool. The results shown in the Clock Relationships section of the Timing Report are based on calculations made using this constraint.

define_clock_delay Syntax

```
define_clock_delay [-rise|fall ] {clockName1} [-rise|fall ] {clockName2} delayValue
```

where

- **`rise|fall`** specifies the clock edge
- **`clockName`** specifies the clocks to constrain

  The clock must be already defined with define_clock.
- **`delayValue`** specifies the delay, in nanoseconds, between the two clocks

  You can also specify a value **false** which defines the path as a false path.

define_ clock_delay Syntax Example

```
Define_clock_delay -rise {clk0} -rise {clk2x} 2
```

define_compile_point

The **define_compile_point** command:

- Is available for Synplify Pro and Synplify Premier only.
- Defines a compile point in a top-level constraint file.

Use one **define_compile_point** command for each compile point you define.

define_compile_point Syntax

```
define_compile_point [ -disable ] { regionName | moduleName } -type { locked }
[-cpfile { } ] [ -comment textString ]
```

where

- **disable** disables a previous compile point definition
- **type** specifies the type of compile point. This must be locked.
- **cpfile** is for Synplicity internal use only

define_compile_point Syntax Example

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

define_current_design

The **`define_current_design`** command:

- Is available for Synplify Pro and Synplify Premier only.
- Specifies the compile-point region or module to which the constraints that follow it apply
- Must be the first command in a compile-point constraint file.

define_current_design Syntax

```
define_current_design {regionName | libraryName.moduleName }
```

define_current_design Syntax Example

```
define_current_design {lib1.prgm_cntr}
```

Objects in all constraints that follow this command relate to **`prgm_cntr`**.

define_false_path

The **`define_false_path`** constraint defines paths to ignore (remove) during timing analysis and give lower (or no) priority during optimization. The false paths are also passed on to supported place and route tools.

define_false_path Syntax

```
define_false_path {-from startPoint | -to endPoint | -through throughPoint}
[-comment textString]
```

where

- **`from`** specifies the starting point for the false path

  The **`From`** point defines a timing start point. It can be any of the following:

  - Clocks (c:)
  - Registers (i:)
  - Top-level input or bi-directional ports (p:)
  - Black box outputs (i:)

  For more information, see the *Synopsys FPGA Synthesis Reference Manual.*

- **`to`** specifies the ending point for the false path

  The **`to`** point defines a timing end point. It can be any of the following:

  - Clocks (c:)
  - Registers (i:)
  - Top-level output or bi-directional ports (p:)
  - Black box inputs (i:)

- **`through`** specifies the intermediate points for the timing exception

  Intermediate points can be any of the following:

  - Combinational nets (n:)
  - Hierarchical ports (t:)
  - Pins on instantiated cells (t:)

By default, the **`through`** points are treated as an OR list. The constraint is applied if the path crosses any points in the list.

To keep the signal name intact through synthesis, set the **`syn_keep directive`** (Verilog or VHDL) on the signal.

define_false_path Syntax Example

The following example shows the syntax for setting **`define_false_path`** between registers:

```
define_false_path -from {i:myInst1_reg} -through {n:myInst2_net}
-to {i:myInst3_reg}
```

The constraint is defined from the output pin of **`myInst1_reg`**, through **`net myInst2_net`**, to the input of **`myInst3_reg`**. If an instance is instantiated, a pin-level constraint applies on the pin, as defined. However, if an instance is inferred, the pin-level constraint is transferred to the instance.

For **`through`** points specified on pins, the constraint is transferred to the connecting net. You cannot define a **`through`** point on a pin of an instance that has multiple outputs. When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

define_input_delay

The **define_input_delay** constraint:

- Specifies the external input delays on top-level ports in the design. It is the delay outside the chip before the signal arrives at the input pin.
- Models the interface of the inputs of the FPGA device with the outside environment. The synthesis tool cannot detect the input delay unless you specify it in a timing constraint.

define_input_delay Syntax

```
define_input_delay [ -disable ] { inputportName } | -default ns [ -route
ns ]
[ -ref clockName:edge ] [ -comment textString ]
```

where

- **disable** disables a previous delay specification on the named port
- **inputportName** is the name of the input port
- **default** sets a default input delay for all inputs.

  Use this option to set an input delay for all inputs. You can set **define_input_delay** on individual inputs to override the default constraint.

  This example sets a default input delay of 3.0 **ns**:

  ```
  define_input_delay -default 3.0
  ```

  This example overrides the default and sets the delay on **input_a** to 10.0 **ns**:

  ```
  define_input_delay {input_a} 10.0
  ```
- **ref** (recommended) is the clock name and edge that triggers the event

  The value must include either the rising edge or falling edge.

  - **r**

    rising edge
  - **f**

    falling edge

  For example:

  ```
  define_input_delay {portb[7:0]} 10.00 -ref clock2:f
  ```
- **route** is an advanced option that includes route delay when the synthesis tool tries to meet the clock frequency goal

  Use the **-route** option on an input port when the place and route Timing Report shows that the timing goal is not met because of long paths through the input port.

define_input_delay Syntax Examples

```
define_input_delay {porta[7:0]} 7.8 -ref clk1:r
define_input_delay -default 8.0
define_input_delay -disable {resetn}
```

define_io_standard

The **define_io_standard** constraint specifies a standard I/O pad type to use for specific Actel, Altera, and Xilinx device families.

define_io_standard Syntax

```
define_io_standard [-disable|-enable] {objectName} -delay_type
input_delay|output_delay columnTclName{value}
[columnTclName{value}...]
```

where

- **delay_type** is either **input_delay** or **output_delay**

define_io_standard Syntax Example

```
define_io_standard {DATA1[7:0]} -delay_type input_delay
syn_pad_type{LVCMOS_33} syn_io_slew{high} syn_io_drive{12}
syn_io_termination{pulldown}
```

define_multicycle_path

The **define_multicycle_path** constraint:

- Specifies a path that is a timing exception because it uses multiple clock cycles
- Provides extra clock cycles to the designated paths for timing analysis and optimization

define_multicycle_path Syntax

```
define_multicycle_path [ -start | -end ] { -from startPoint | -to endPoint |
-through throughPoint }clockCycles [ -comment textString ]
```

where

- **start| end** specifies the clock cycles to use for paths with different start and end clocks.

  This option determines the clock period to use as the multiplicand in the calculation for clock distance. If you do not specify a **start** or **end** option, the **end** clock is the default.

- **from** specifies the start point for the Multi-Cycle timing exception

  The **from** point defines a timing start point. It can be any of the following:

  - Clocks (c:)
  - Registers (i:)
  - Top-level input or bi-directional ports (p:)
  - Black box outputs (i:)

- **to** specifies the end point for the Multi-Cycle timing exception

  The **to** point defines a timing start point. It can be any of the following:

  - Clocks (c:)
  - Registers (i:)
  - Top-level input or bi-directional ports (p:)
  - Black box outputs (i:)

- **through** specifies the intermediate points for the timing exception

  Intermediate points can be:

  - Combinational nets (n:)
  - Hierarchical ports (t:)
  - Pins on instantiated cells (t:)

  By default, the intermediate points are treated as an OR list. The exception is applied if the path crosses any points in the list.

  For more information, see Specify From/To/Through Points in Chapter 5, Specify From/To/Through Points.

  Combine this option with **-to** or **-from** to get a specific path. To keep the signal name intact throughout synthesis when you use this option, set the **syn_keep directive** (Verilog or VHDL) on the signal.

- **clockCycles** is the number of clock cycles to use for the path constraint

Timing exception constraints must contain object types in the specification. Timing exceptions, such as Multi-Cycle path and false path constraints, require that you explicitly specify the object type (**`n:`** or **`i:`**) in the instance name parameter. For example:

```
define_multicycle_path -from {i:inst2.lowreg_output[7]} -to {i:inst1.DATA0[7]} 2
```

If you use SCOPE to specify timing exceptions, it attaches object type qualifiers to the object names.

For more information, see the *Synopsys FPGA Synthesis Reference Manual*.

define_multicycle_path Syntax Examples

```
define_multicycle_path -from{i:regs.addr[4:0]} -to{i:special_regs.w[7:0]} 2
define_multicycle_path -to {i:special_regs.inst[11:0]} 2
define_multicycle_path -from {p:porta[7:0]}  -through {n:prgmcntr.pc_sel44[0]} -to
{p:portc[7:0]} 2
define_multicycle_path -from {i:special_regs.trisc[7:0]} -through {t:uc_alu.aluz.Q}
-through {t:special_net.Q} 2
```

The following example shows the syntax for setting a Multi-Cycle path constraint between registers:

```
define_multicycle_path -from {i:myInst1_reg} -through {n:myInst2_net} -to {i:myInst3_reg} 2
```

The constraint is defined from the output of **`myInst1_reg`**, through net **`myInst2_net`**, to the input pin **`myInst3_reg`**. If the instance is instantiated, a pin-level constraint applies on the pin, as defined. However, if the instance is inferred, the pin-level constraint is transferred to the instance.

For **`through`** points specified on pins, the constraint is transferred to the connecting net. You cannot define a **`through`** point on a pin of an instance that has multiple outputs. When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

define_output_delay

The **`define_output_delay`** constraint:

- Specifies the delay of the logic outside the FPGA device driven by the top-level outputs.
- Models the interface of the outputs of the FPGA device with the outside environment.

The default delay outside the FPGA device is 0.0 ns. Output signals typically drive logic outside the FPGA device. The synthesis tool cannot detect the delay for that logic unless you specify it with a timing constraint.

define_output_delay Syntax

```
define_output_delay [ -disable ] { outputportName } |-default ns [ -route ns ]
[ -ref clockName:edge ] [ -comment textString ]
```

where

- **`disable`** disables a previous delay specification on the named port
- **`outputportName`** is the name of the output port
- **`default`** sets a default input delay for all outputs

  Use this option to set a delay for all outputs. Setting **`define_output_delay`** on individual outputs overrides the default constraint. This example sets a default output delay of 8.0 ns. The delay is outside the FPGA device.

define_output_delay Syntax Examples

```
define_output_delay -default 8.0
```

The following example overrides the default and sets the output delay on **`output_a`** to 10.0 **`ns`**. Accordingly, **`output_a`** drives 10 **`ns`** of combinational logic before the relevant clock edge.

```
define_output_delay {output_a} 10.0
```

where

- **`ref`** defines the clock name and edge that controls the event

The value must be one of the following:

  - **`r`**

    rising edge
  - **`f`**

    falling edge

  For example:

  ```
  define_output_delay {portb[7:0]} 10.00 -ref clock2:f.
  ```

- **`route`** is an advanced option that includes route delay when the synthesis tool tries to meet the clock frequency goal

Output Pad Clock Domain Default

By default, **define_output_delay** constraints with no reference clock are constrained against the global frequency, instead of the start clock for the path to the port. The synthesis tool assumes the register and pad are not in the same clock domain. This change affects the Timing Report and timing driven optimizations on any logic between the register and the pad.

You must specify the clock domain for all output pads on which you have set output delay constraints. For the pads for which you do not specify a clock, add the **-ref** option to the **define_output_delay** constraint.

```
define_output_delay {LDCOMP} 0.50 -improve 0.00 -route 0.25 -ref {CLK1:r}
```

define_path_delay

The **define_path_delay** constraint specifies point-to-point delay in nanoseconds (ns) for maximum and minimum delay constraints. To specify the start, end, or through points, use:

- The following options:
  - **-from**
  - **-to**
  - **-through**

  OR
- Any combination of these options

If you specify both **define_path_delay -max** and **define_multicycle_path** for the same path, the synthesis tool uses the more restrictive of the two constraints.

When you specify **define_path_delay** and you also define input or output delays, the synthesis tool adds the input or output delays to the path delay. The timing constraint that is forward-annotated includes the I/O delay with the path delay. This could result in discrepancies with the Xilinx place and route tool, which ignores the I/O delays and reports the path delay only.

define_path_delay Syntax

```
define_path_delay [-disable] {-from {startPoint} | -to {endPoint} | -through {throughPoint}
-max delayValue [-comment textString ]
```

where

- **disable** disables the constraint
- **from** specifies the starting point of the path.

  The **from** point defines a timing start point. It can be any of the following:
  - Clocks (c:)
  - Registers (i:)
  - Top-level input or bi-directional ports (p:)
  - Black box outputs (i:)
- **to** specifies the ending point of the path.

  The **to** point must be a timing end point. It can be any of the following:
  - clocks (c:)
  - registers (i:)
  - top-level output or bi-directional ports (p:)
  - black box inputs (i:)

  Combine this option with **-from** or **-through** to get a specific path.
- **through** specifies the intermediate points for the timing exception.

  Intermediate points can be:
  - combinational nets (n:)
  - hierarchical ports (t:)
  - pins on instantiated cells (t:)

By default, the intermediate points are treated as an OR list. The exception is applied if the path crosses any points in the list. Combine this option with **`-to`** or **`-from`** to get a specific path. To keep the signal name intact throughout synthesis when you use this option, set the **`syn_keep`** directive (Verilog or VHDL) on the signal.

- **`max`** sets the maximum allowable delay for the specified path

  This is an absolute value in nanoseconds (ns) and is shown as **`max analysis`** in the Timing Report.

define_path_delay Syntax Examples

```
define_path_delay -from {i:dmux.alu [5]} -to {i:regs.mem_regfile_15[0]} -max 0.800
```

The following example sets a max delay of 2 **`ns`** on all paths to the falling edge of the flip-flops clocked by **`clk1`**.

```
define_path_delay -to {c:clk1:f} -max 2
```

The following example sets the path delay constraint on the pins between registers:

```
define_path_delay -from {i:myInst1_reg} -through {t:myInst2_net.Y}
-to {i:myInst3_reg} -max 0.123
```

The constraint is defined from the output pin of **`myInst1`**, through pin **`Y`** of net **`myInst2`**, to the input pin of **`myInst3`**. If the instance is instantiated, a pin-level constraint applies on the pin, as defined. If the instance is inferred, the pin level constraint is transferred to the instance.

For through points specified on pins, the constraint is transferred to the connecting net. You cannot define a through point on a pin of an instance that has multiple outputs.

When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

define_reg_input_delay

The **define_reg_input_delay** constraint speeds up paths feeding a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with **define_clock**). Use this constraint to speed up the paths feeding a register.

define_reg_input_delay Syntax

```
define_reg_input_delay { registerName } [ -route ns ] [ -comment textString ]
```

where

- **registerName** is:
  - a single bit
  - an entire bus, or
  - a slice of a bus
- **route** is an advanced user option to tighten constraints during resynthesis

  Use **route** when the place and route Timing Report shows the timing goal is not met because of long paths to the register.

define_reg_output_delay

The **`define_reg_output_delay`** constraint speeds up paths coming from a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with **`define_clock`**). Use this constraint to speed up the paths coming from a register.

define_reg_output_delay Syntax

```
define_reg_output_delay { registerName } [ -route ns ] [ -comment textString ]
```

where

- **`registerName`** is:
  - A single bit
  - An entire bus, or
  - A slice of a bus
- **`route`** is an advanced user option to tighten constraints during resynthesis

  Use **`route`** when the place and route Timing Report shows the timing goal is not met because of long paths to the register.

Specify From/To/Through Points

This section discusses:

- From/To Points
- Through Points
- Clocks as From/To Points

From/To Points

From specifies the starting point for the timing exception. **To** specifies the ending point for the timing exception. See the following table.

Table 5-2: **Objects That Can Serve as Starting and Ending Points**

From Points	To Point
Clocks	Clocks
Registers	Registers
Top-level input or bi-directional ports	Top-level output or bi-directional ports
Instantiated library primitive cells (gate cells)	
Black box outputs	Black box inputs

You can specify multiple from points in a single exception. This is most common when specifying exceptions that apply to all bits of a bus. For example, you can specify constraints **From A[0:15] to B.** In this case, there is an exception, starting at any of the bits of **A** and ending on **B**.

Similarly, you can:

- Specify multiple to points in a single exception, and
- Specify both multiple starting points and multiple ending points such as **From A[0:15] to B[0:15]**.

Through Points

Although **through** points are limited to nets, there are many ways to specify these constraints:

- Single Through Point
- Single List of Through Points
- Multiple Through Points
- Multiple Lists of Through Points

You can also define these constraints in:

- The appropriate SCOPE panels, or
- The Sum of Products interface

When a port and a net have the same name, preface the name of the through point with:

- **n:**

  nets

- **t:**
  hierarchical ports
- **p:**
  top-level ports

For example:

```
n:regs_mem[2] or t:dmux.bdpol
```

The **n:** prefix must be specified to identify nets. Otherwise, the associated timing constraint is not be applied for valid nets.

Single Through Point

You can specify a single through point.

```
define_false_path -through regs_mem[2]
```

In this example, the constraint is applied to any path that passes through:

- regs_mem[2]:

Single List of Through Points

If you specify a single list of through points, the -**through** option:

- Behaves as an **OR** function
- Applies to any path that passes through any of the points in the list.

```
define_path_delay -through {regs_mem[2], prgcntr.pc[7], dmux.alub[0]}
-max 5 -min 1
```

In this example , the constraint is applied to any path through:

- **regs_mem[2]**
  OR
- **prgcntr.pc[7]**
  OR
- **dmux.alub[0]**

Multiple Through Points

To specify multiple points for the same constraint, precede each point with the **-through** option.

```
define_path_delay -through regs_mem[2] -through prgcntr.pc[7] -through dmux.alub[0] -max 5
-min 1
```

In this example, the constraint operates as an **AND** function and applies to paths through:

- **regs_mem[2]**
  AND
- **prgcntr.pc[7]**
  AND
- **dmux.alub[0]**

Multiple Lists of Through Points

If you specify multiple **-through** lists, the constraint:

- Behaves as an **AND/OR** function
- Is applied to the paths through all points in the lists

Multiple Lists of Through Points Example One

```
define_false_path -through {A1 A2...An} -through {B1 B2 B3}
```

In this example the constraint applies to all paths that pass through:

- **{A1 or A2 or...An}**

  AND
- **{B1 or B2 or B3}**

Multiple Lists of Through Points Example Two

```
define_multicycle_path -through {net1, net2} -through {net3, net4} 2
```

In this example, all paths that pass through the following nets are constrained at 2 clock cycles:

```
net1 AND net3
OR net1 AND net4
OR net2 AND net3
OR net2 AND net4
```

Clocks as From/To Points

You can specify clocks as **from-to** points in your timing exception constraints.

Clocks as From/To Points Syntax

```
define_timing_exception -from | -to { c:clock_name [: edge] }
```

where

- **timing_exception** is one of the following constraint types:
  - **multicycle_path**
  - **false_path**
  - **path_delay**
- **c:clock_name:edge** is the name of the clock and clock edge (**r** or **f**)

If you do not specify a clock edge, both edges are used by default.

Multi-Cycle Path Clock Points

When you specify a clock as a **from** or **to** point, the Multi-Cycle path constraint applies to all registers clocked by the specified clock.

The following example allows two clock periods for all paths from the rising edge of the flip-flops clocked by **clk1**:

```
define_multicycle_path -from {c:clk1:r} 2
```

You cannot specify a clock as a **through** point. However, you can set a constraint **from** or **to** a clock and **through** an object:

- net
- pin
- hierarchical port

The following example allows two clock periods for all paths to the falling edge of the flip-flops clocked by **clk1** and **through** bit 9 of the hierarchical net:

```
define_multicycle_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]} 2
```

False Path Clock Points

When you specify a clock as a **from** or **to** point, the false path constraint is set on all registers clocked by the specified clock. The timing analyzer ignores all false paths.

The following example disables all paths from the rising edge of the flip-flops clocked by **clk1**:

```
define_false_path -from {c:clk1:r}
```

You cannot specify a clock as a **through** point. However, you can set a constraint **from** or **to** a clock and **through** an object:

- net
- pin
- hierarchical port

The following example disables all paths to the falling edge of the flip flops clocked by **clk1** and **through** bit 9 of the hierarchical net.

```
define_false_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]}
```

Path Delay Clock Points

When you specify a clock as a **from** or **to** point for the path delay constraint, the constraint is set on all paths of the registers clocked by the specified clock.

The following example sets a max delay of 2 **ns** on all paths to the falling edge of the flip-flops clocked by **clk1**:

```
define_path_delay -to {c:clk1:f} -max 2
```

You cannot specify a clock as a **through** point. However, you can set a constraint **from** or **to** a clock and **through** an object:

- net
- pin
- hierarchical port

The following example sets a max delay of 0.2 **ns** on all paths from the rising edge of the flip-flops clocked by **clk1** and **through** bit 9 of the hierarchical net:

```
define_path_delay -from {c:clk1:r} -through (n:MYINST.mybus2[9]} -max .2
```

Specifying Timing Constraints in a SCOPE Spreadsheet

The Synthesis Constraints Optimization Environment® (SCOPE) is a spreadsheet-like interface for entering and managing timing constraints and synthesis attributes.

The SCOPE spreadsheet can generate constraint files in Tcl format. Use this method for specifying constraints wherever possible. You can use it for most constraints, except for source code directives.

To create and open a new SCOPE dialog box:

- Choose **File > New > Constraint file (SCOPE)** from the Project view,

  OR
- Click the SCOPE icon on the toolbar

For each TCL timing constraint type, there is an equivalent SCOPE spreadsheet interface.

For more information, see the *Synplify User's Guide* (SCOPE and Timing Constraints > Scope Constraints).

Forward Annotation

The synthesis tool generates vendor-specific constraint files that can be forwarded and annotated with the place and route tools. The constraint files are generated by default. To disable this feature, deselect the following option:

Project > Implementation Option > Implementation Results > Write Vendor Constraint File

The constraint file generated for Xilinx place and route tools has an `ncf` file extension (`.ncf`).

The timing constraints described in the TCL and SCOPE sections are forward-annotated to Xilinx in this file. In addition to these constraints, the synthesis tool forward-annotates relationships between different clocks.

For more information, see:

- I/O Timing Constraints
- Clock Groups
- Relaxing Forward-Annotated I/O Constraints
- Digital Clock Manager/Delay Locked Loop

I/O Timing Constraints

By default, the synthesis tool forward-annotates the **`define_input_delay`** and **`define_output_delay`** timing constraints to the Xilinx `.ncf` file. The **`syn_forward_io_constraints`** attribute controls forward annotation.

A value of **`1`** or **`true`** (default) enables forward annotation. A value of **`0`** or **`false`** disables it.

Use this attribute at the top level of a VHDL or Verilog file, or use the Attributes panel of the SCOPE spreadsheet to add the attribute as a global object.

Clock Groups

If two clocks are in the same clock group, the synthesis tool writes out the Xilinx `.ncf` file for forward-annotation so that one clock is a fraction of the other.

In the following example, **`clk1`** is derived as a fraction of **`clk2`**, which signals the place and route tool that the two clocks are part of the same clock group.

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" "TS_clk2" * 2.000000 HIGH 50.00%;
```

In the following example, the clocks are declared independently, so the place and route tool considers the clocks separately for timing calculation:

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" 20.000 ns HIGH 50.00%;
```

Relaxing Forward-Annotated I/O Constraints

If the **`xc_use_timespec_for_io`** attribute is enabled (1), I/O constraints are forward-annotated using the Xilinx **`TIMESPEC FROM ... TO`** command. In this case, there is no relaxation of the constraints.

For more information, see the *Synopsys FPGA Synthesis Reference Manual*.

The synthesis tool constrains input-to-register, register-to-register and register-to-output paths with the FREQUENCY constraint. However, if the Period constraint is too tight for the input-to-register or register-to-output paths, the synthesis tool tries to relax the constraints to these paths.

Digital Clock Manager/Delay Locked Loop

The synthesis tool can take advantage of the Frequency Synthesis and Phase Shifting features of Digital Clock Manager (DCM) and Delay Locked Loop (DLL) for Xilinx devices.

If you are using a DLL or DCM for on-chip clock generation, you need only define the clock at the primary inputs. The synthesis tool propagates clocks through any number of DLLs or DCMs. It generates clocks at the outputs of a DLL or DCM, as needed, taking into account any phase shift or frequency change.

To specify the phase shift and frequency multiplication parameters, use Xilinx standard properties such as:

- **`duty_cycle_correction`**
- **`clkdv_divide`**
- **`clkfx_multiply`**
- **`clkfx_divide`**

The synthesis tool also takes into account the fact that these clocks are related (synchronized) to each other, and puts them in the same clock group. However, only the clock at the input of a DLL/DCM is forward-annotated in the `.ncf` file. The back end tools understand the DLL and DCMs, and do their own clock propagation across them.

Chapter 6

Timing Analysis

Use the **trce** command to analyze timing constraints. Run the **trce** command from:

- Timing Analyzer, or
- The command line.

Multi-Corner, Multi-Node Timing Analysis

The multi-corner, multi-node timing analysis ensures that the timing analysis is guaranteed over Process, Voltage, and Temperature (PVT) variations.

- The design is analyzed at the Fast Process Corner and at the Slow Process Corner.
- The worst case timing analysis is reported in the Timing Report.

Speed File Values

The Fast Process Corner and Slow Process Corner speed file values are based upon the characterization data.

- Each Process Corner has maximum and minimum measured delays.
- The Fast Process Corner and Slow Process Corner are analyzed simultaneously.
- The Process Corner with the greatest minimum or maximum variance is reported as worst case.

This analysis is performed for Virtex®-6, Spartan®-6, and Xilinx 7 series device families only.

Process Corner Information

The Process Corner information reports which Process Corner was used to characterize the delay values.

Slow Process Corner

The Slow Process Corner is defined as:

- High temperature
- Low voltage

This is the traditional worst case or maximum PVT.

Fast Process Corner

The Fast Process Corner is defined as:

- Low temperature
- High voltage

This is the traditional absolute minimum speed grade.

Worst Case Analysis

In the majority of designs:

- The Fast Process Corner is reported for the worst case *hold* analysis.
- The Slow Process Corner is reported for worst case *setup* analysis.

In some clock and data topologies, the Fast Process Corner is reported for the worst case *setup* analysis.

Asynchronous Reset Paths

The analysis of asynchronous reset paths, including the recovery time and reset pin to output time, is not included in the Period constraint analysis by default.

In order to see asynchronous reset and asynchronous set paths, enable a path tracing control (PTC) as follows.

- ENABLE = REG_SR_R;

  For recovery time
- ENABLE = REG_SR_O;

  For output time

These path-tracing controls enable the path from the asynchronous reset pin through the synchronous element and the reset recovery time of the synchronous element.

The asynchronous SR and CLR recovery paths are controlled by the REG_SR_R PTC. The asynchronous SR and CLR propagation and removal paths are controlled by the REG_SR_O PTC. These PTC control individual timing delay names and timing arcs.

Timing Analyzer

The timing constraint is analyzed using Timing Analyzer or the **`trce`** command.

This timing analysis:

- Provides a detailed path analysis of the timing path with regards to the timing constraint requirements.
- Ensures that the specific timing constraints are passed through the implementation tools.

The path specific details and includes the following:

- Confirms that the timing requirements were met for all path per constraint.
- Confirms the setup and hold requirements were met for all path per constraint.
- Confirms that the device component are performing within operational frequency limits.
- Provides a list of unconstrained path that may be a critical path that was not analyzed.

Timing Report

This section discusses a typical Timing Report.

Timing Report Contents

A typical Timing Report contains the following sections:

- Constraint Details
- Data Sheet
- Summary

Constraint Details

The Constraint Details section shows path details per constraint.

Data Sheet

The Data Sheet section shows:

- General Setup
- Hold
- Clock to Out times

Summary

The Summary section shows:

- Timing Errors
- Timing Score
- Constraint Coverage
- Design Statistics

Path Details

The path details are shown under each timing constraint, including the following:

- Constraint header, with:
  - Path analyzed
  - Endpoints analysis
  - Failing endpoints
  - Timing errors detected

- Minimum Period/Offset In
- Setup Paths

  Individual path with setup analysis with a specific slack equation
- Hold Paths

  Individual path with hold analysis with a specific slack equation
- Component Switching Limits for Period constraints with a specific slack equation

Period Analysis

periodanalysis

Synchronous to synchronous elements are analyzed in the Period analysis. The Period constraint defines the timing relationship of the clock domains.

The analysis includes:

- Paths within a single clock domain.
- Paths between related clock domains and related Period constraints.
- Frequency or period; phase; and uncertainty differences between the source and destination synchronous elements.
- Single clock domain paths.
- Cross-clock domain paths.

Header Summary

The analysis for the Period constraint includes a header summary. The header summary summarizes information about the constraint, including:

- Number of paths and number of endpoints analyzed
- Setup, hold, or component switching limit errors

This information allows you to verify that the constraint covered the expected number of endpoints and paths and the overall worst case performance of this constraint.

Component Switching Limit Analysis

The component switching limit analysis:

- Is performed in addition to the traditional setup and hold analysis.
- Ensures that the operating frequency of the device component is not exceeded and is within device specifications.
- Is performed on:
  - Larger device components (such as DSP and BRAM)
  - Smaller device components (such as ILOGIC, OLOGIC, and SLICE)
  - Clocking components (such as DCM and PLL) in a constrained clock domain

Common Component Switching Limits

The most common component switching limits are:

- MINPERIOD
- MINLOWPULSE
- MINHIGHPULSE

Additional Component Switching Limits

Some components have the following additional component switching limits:

- MAXPERIOD
- MAXLOWPULSE
- MAXHIGHPULSE

Path Analysis Details

The details for each path analyzed are shown after the header summary for the Period constraint. Each path is a synchronous element to another synchronous element with either the setup or hold timing of the destination synchronous element.

Period constraints constrain those data paths from synchronous elements to synchronous elements. The most common examples are:

- Single clock domain
- Two-phase clock domain
- Multiple clock domains

A Timing Report example is provided for each common type of path a Period constraint may cover in your design.

First Paragraph Contents

The first paragraph includes:

- Overall slack of the path
- Synchronous path performance
- Source design synchronous element
- Destination design synchronous element
- Source and destination clock signal with the corresponding clock edge,
- Total data path delay
- Clock uncertainty
- Slack equation
- Clock uncertainty equations

Second Paragraph Contents

The second paragraph includes the data path details between:

- The source synchronous element, and
- The destination synchronous element.

This includes the individual elements that make up this data path, which is the device resource utilized and net routing delays of the data path.

Clock Domains

Period constraints constrain those data paths from synchronous elements to synchronous elements. The most common examples are:

- Single clock domains
- Two-phase clock domains
- Multiple clock domains

A timing report example is provided for each common type of path a Period constraint may cover in your design.

Gated Clocks

The Period constraint does not analyze gated or internally derived clocks correctly. If the clock is gated or goes through a LUT, the timing analysis traces back through each input of the LUT to the source (synchronous elements or pads) of the signals and reports the corresponding Clock Skew.

The Clock Skew derived from a LUT is very large, depending on the levels of logic or number of LUTs.

If the clock has been divided by using internal logic and not by a DCM, the Period constraint on the clock pin of the Divide Down Flip Flop does not trace through this flip-flop to the **`clk_div`** signal. See the following figure.

The timing analysis does not include the downstream synchronous elements, which are driven by the new gated-clock signal.

Unless a global buffer is used, the new clock derived from the Divide Down Flip-Flop is on local routing. If a Period constraint is placed on the output of the Divide down Flip-Flop (shown as the **`clk_div`** signal in the following figure), and is related back to the original Period constraint, the timing analysis includes the downstream synchronous elements.

To ensure that the relationship and the cross-clock domain analysis is correct, include the difference between the divided clock and the original clock in the Period constraint with the Phase keyword. The Clock Skew can be large, depending on the relationship between the two clocks.

Because the Phase keyword defines the difference between the two clocks, this becomes the timing constraint requirement for the cross clock domain path analysis. If the Phase keyword value is too small, it is impossible to meet the cross clock domain path analysis.

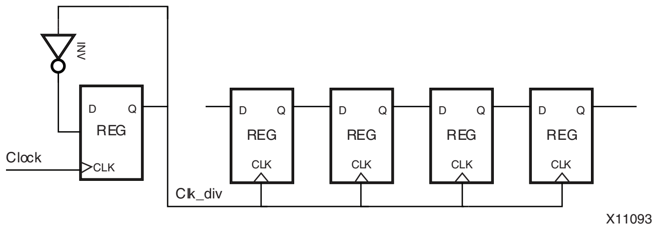


Figure 6-1: **Gated Clock with Divide Down Flip Flop**

Single Clock Domain

A single clock domain is easy to understand and analyze. All synchronous elements are on the same clock domain, and are analyzed on the rising-edge of the clock or all elements are analyzed on the falling-edge of the clock.

The clock source is driven by the same clock source, which can be a PAD, DCM, DLL, PLL, or PMCD component with only one output.

The timing analysis tool reports the active edges of the clock driver and the corresponding time for the data path between the synchronous elements.

The following figure shows a simple design. The Period constraint is analyzed from the User Constraints File (UCF).

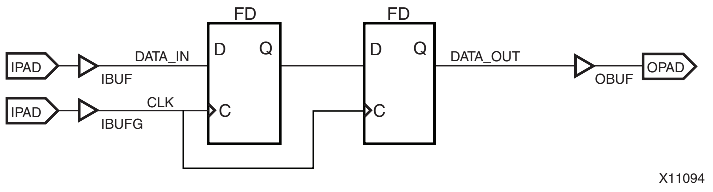


Figure 6-2: **Single Clock Domain Schematic**

Single Clock Domain Timing Report Example

```
Slack (setup path): 3.904ns (requirement - (data path - clock path skew + uncertainty))
  Source:              IntA_1 (FF)
  Destination:         XorA_1 (FF)
  Requirement:         8.000ns
  Data Path Delay:     4.036ns (Levels of Logic = 1)
  Clock Path Skew:     0.000ns
  Source Clock:        clk0 rising at 0.000ns
  Destination Clock:   clk0 rising at 8.000ns
  Clock Uncertainty:   0.060ns
```

Two-Phase Clock Domain

The analysis of a data path that uses both edges of the clock is known as a two-phase clock domain, or a two-phase data path. See the following figure.

This clock can be driven by the same clock source, such as a PAD, DCM, DLL, PLL, or PMCD component with only one output. These synchronous elements can also be driven by two related clocks, such as the CLK0 and CLK180 or CLK90 and CLK270 of a DCM, DLL, PLL, or PMCD component.

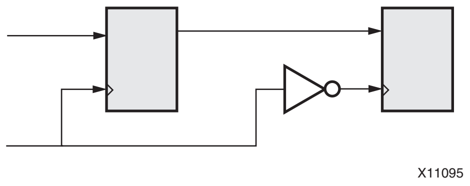


Figure 6-3: **Two-Phase Clock**

The timing analysis tool reports the active clock signal and the corresponding active clock arrival time for the source and destination synchronous element. The difference in clock arrival times for the source and destination synchronous elements determines the data path requirement. In a two-phase data path, the data path requirement is a fraction of the single-phase data path requirement. See the following figure.

The timing analysis tool reports the data path details by the slack value. The slack value states the relationship between the data path requirement and the data path delays. The data paths are ordered based upon the slack value, with the largest negative values (falling) down to the largest positive values (passing).

Minimum Period Value

Occasionally the largest worst or negative slack value data path does not match the Minimum Period value. This failure is usually caused by the slack value of a two-phase data path that is not on the top of the list of data paths.

In the majority of the cases, the data path at the top of the list corresponds to the Minimum Period value. In some cases the two-phase data path corresponds to the Minimum Period value. In the two-phase data path situation, the timing analysis tools determine the fractional relationship between the original single or full phase data path requirement and the two-phase data path requirement.

This fractional value is used to convert the total data path delay of the two-phase data path back to a single or full phase data path delay equivalent. If the fractional relationship is determined to be half, the two-phase data path delay is doubled for the full phase data path delay equivalent. The Minimum Period value is only in full phase data path delay, not fractional data path delays.

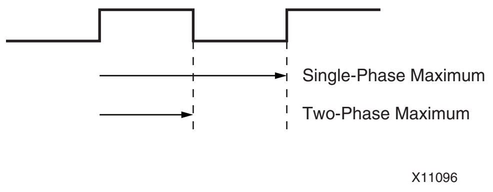


Figure 6-4: **Relationship Between Single-Phase and Two-Phase Clocks**

Two-Phase Example Design with Period Constraint

An example design with a Period constraint or full-phase data path requirement of 6ns has both full-phase and two-phase data paths.

Table 6-1: **Example Design with Period Constraint**

Data Path	Total Data Path Delay (ns)	Slack (ns)
Full-Phase	8	-2
Two-Phase	4.036	-1.096

Although the full-phase data path is at the top of the list, followed by the two-phase data path, the Minimum Period value is 8.192ns. The Minimum Period value corresponds to the two-phase data path, not the full-phase data path.

Timing Report Example One

```
Slack (setup path): -1.096ns (requirement - (data path - clock path skew + uncertainty))
  Source: IntA_1 (FF)
  Destination: XorA_1 (FF)
  Requirement: 3.000ns
  Data Path Delay: 4.036ns (Levels of Logic = 1)
  Clock Path Skew: 0.000ns
  Source Clock: clk0 rising at 0.000ns
  Destination Clock: clk0 falling at 3.000ns
  Clock Uncertainty: 0.060ns
```

Timing Report Example Two

```
Timing constraint: TS_DRAM_CTRL_U_u_infrastructure_clk_pll = PERIOD TIMEGRP
     "DRAM_CTRL_U_u_infrastructure_clk_pll" TS_clk_303 / 0.5 HIGH 50%;

56924 paths analyzed, 17458 endpoints analyzed, 366 failing endpoints
452 timing errors detected. (366 setup errors, 86 hold errors, 0 component switching limit
errors)
Minimum period is 24447.220ns.
--------------------------------------------------------------------------------

Paths for end point DRAM_CTRL_U/bank_conflict (SLICE_X39Y106.C3), 31 paths
--------------------------------------------------------------------------------
Slack (setup path):     -3.666ns (requirement - (data path - clock path skew + uncertainty))
  Source:               DRAM_CTRL_U/out_add[1].rd_addr_fifo/USE_SDPRAM_LUT.sdpram_lut_inst/
depth_le_5.gen_sdpram[0].sdpram32_RAMB (RAM)
  Destination:          DRAM_CTRL_U/bank_conflict (FF)
  Requirement:          0.002ns
  Data Path Delay:      3.146ns (Levels of Logic = 3)(Component delays alone exceeds
constraint)
  Clock Path Skew:      -0.250ns (2.637 - 2.887)
  Source Clock:         clk_250 rising at 13312.000ns
  Destination Clock:    DRAM_CTRL_U/clk rising at 13312.002ns
  Clock Uncertainty:    0.272ns

  Clock Uncertainty:          0.272ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter (TSJ):  0.070ns
    Discrete Jitter (DJ):       0.157ns
    Phase Error (PE):           0.185ns

  Maximum Data Path at Slow Process Corner: DRAM_CTRL_U/out_add[1].rd_addr_fifo/
USE_SDPRAM_LUT.sdpram_lut_inst/depth_le_5.gen_sdpram[0].sdpram32_RAMB to DRAM_CTRL_U/
bank_conflict
    Location             Delay type         Delay(ns)  Physical Resource
                                                       Logical Resource(s)
    -------------------------------------------------  -------------------
    SLICE_X42Y107.BMUX   Tshcko                 1.492   DRAM_CTRL_U/rd_addr_pre<1><1>
                                                      DRAM_CTRL_U/out_add[1].rd_addr_fifo/
USE_SDPRAM_LUT.sdpram_lut_inst/depth_le_5.gen_sdpram[0].sdpram32_RAMB
    SLICE_X41Y107.D5     net (fanout=3)         0.331   DRAM_CTRL_U/rd_addr_pre<1><2>
    SLICE_X41Y107.D      Tilo                   0.068   DRAM_CTRL_U/rd_addr_1<2>
                                                       DRAM_CTRL_U/
Mmux_last_bank[2]_last_bank[2]_MUX_834_o11
    SLICE_X41Y106.C4     net (fanout=1)         0.502   DRAM_CTRL_U/
Mmux_last_bank[2]_last_bank[2]_MUX_834_o1
    SLICE_X41Y106.C      Tilo                   0.068   DRAM_CTRL_U/n0728<0>
```

```
                                                          DRAM_CTRL_U/
Mmux_last_bank[2]_last_bank[2]_MUX_834_o12
    SLICE_X39Y106.C3      net (fanout=1)       0.612    DRAM_CTRL_U/
Mmux_last_bank[2]_last_bank[2]_MUX_834_o11
    SLICE_X39Y106.CLK     Tas                  0.073    DRAM_CTRL_U/bank_conflict
                                                          DRAM_CTRL_U/
Mmux_last_bank[2]_last_bank[2]_MUX_834_o19
                                                          DRAM_CTRL_U/bank_conflict
    ------------------------------------------------  ---------------------------
    Total                                        3.146ns (1.701ns logic, 1.445ns route)
                                                          (54.1% logic, 45.9% route)
```

In Timing Report Example Two, **Minimum period is 24447.220 ns** is based upon the clock arrival relationship between the **Source Clock** and the **Destination Clock**. The timing engine:

1. Analyzes these clock networks, and
2. Determines the two closest clock edges in time.

The two closest clock edges are reported as clock arrival times. The difference is defined as the **Requirement**. This requirement is a fraction of the full-cycle Period constraint requirement. Because the full-cycle Period constraint requirement is 13.33 ns, the relationship between the new requirement and the original full-cycle requirement is 1/6665.

This setup path analysis is 1/6665 portion of the full-cycle. The Minimum period value is a full-cycle value. When the setup total (3.668ns) is multiplied by 6665, the Minimum period is 24,447.220 ns.

Multiple Clock Domains

A cross clock domain path has two different clocks for the source and destination synchronous elements. One clock drives the source. A different clock drives the destination.

If the source clock Period constraint is related to the destination clock Period constraint, the destination clock Period constraint covers the cross clock domain analysis.

Xilinx® recommends relating the clocks by means of Period constraints. By so doing, the analysis properly includes the cross clock domain paths.

If the clocks are not related, the cross clock domain paths are not analyzed. Xilinx recommends using a From:To or Multi-Cycle constraint to either flag it as a false path, or as a Multi-Cycle path.

Clocks from DCM outputs

Because the clock signals produced by a DCM, DLL, PLL, or PMCD component are related to each other, the Period constraints should also be related.

To relate the Period constraints:

- Allow NGDBuild to create new Period constraints based upon the input clock signal Period constraint.

  OR

- Manually create Period constraints based upon the output clock signals of the DCM, DLL, PLL, or PMCD component, and manually relate the Period constraints.

Clk0 Clock Domain

Because the clocks produced by the DCM, DLL, PLL, or PMCD component are related, the timing tools consider this relationship during analysis. The synchronous element clock pin is driven by the same clock net from a DCM, DLL, PLL, or PMCD component output. The timing analysis tool reports the active edges of the clock and the corresponding time for the data path between the synchronous elements.

The example in the following figure shows a **CLK0** clock circuit with a simple design. This clock domain has the same requirement and phase shifting as the original requirement.

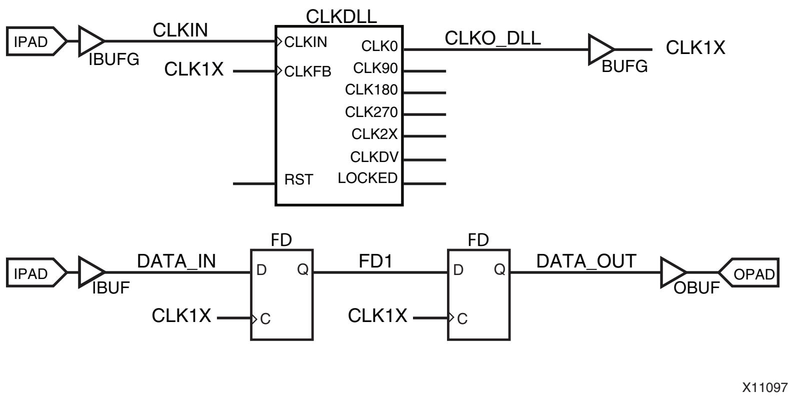


Figure 6-5: **Clk0 DCM Output Schematic**

Clk0 DCM Output Timing Report Example

```
Slack (setup path): 3.904ns (requirement - (data path - clock path skew + uncertainty))
  Source:                IntA_1 (FF)
  Destination:           XorA_1 (FF)
  Requirement:           8.000ns
  Data Path Delay:       4.036ns (Levels of Logic = 1)
  Clock Path Skew:       0.000ns
  Source Clock:          clk0 rising at 0.000ns
  Destination Clock:     clk0 rising at 8.000ns
  Clock Uncertainty:     0.060ns
```

Clk90 Clock Domain

Because the clocks produced by the DCM, DLL, PLL, or PMCD component are related, the timing tools consider this relationship during analysis. The synchronous element clock pins are driven by different clock nets from a DCM, DLL, PLL, or PMCD component output. The timing analysis tool reports the active edges of the clock and the corresponding time for the data path between the synchronous elements.

Clk90 Clock Phase Between DCM Outputs Schematic

The example in the following figure shows **CLK0** and **CLK90** signals where the phase difference is 90 degrees.

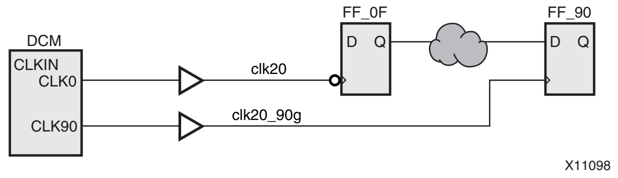


Figure 6-6: **Clk90 Clock Phase Between DCM Outputs Schematic**

Another cause of the Minimum Period value differing from the first path listed in the Timing Report is a cross-clock domain analysis of phase-shifted clocks.

If the phase difference between the two clock domains is 90 degrees, the total data delay is multiplied by four to get to a full period value.

If the data path is 1.5ns for this clock90 constraint, the equivalent full period value is 6 ns.

In addition, for this example, the data path goes from a falling-edge of CLK0 clock signal to the rising-edge of CLK90 clock signal, and the timing analysis includes the two-phase information from CLK0 to do the analysis. See the following figure.

Although the original Period constraint was set to 20 ns, this cross-clock domain analysis has a new requirement of 15 ns. This new requirement compensates for the phase difference between the two clocks. See the preceding figure.

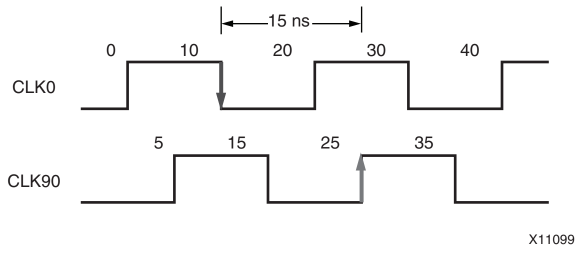


Figure 6-7: **Clock Edge Relationship**

Clk90 Timing Report Example

```
Slack (setup path): 5.398ns (requirement - (data path - clock path skew + uncertainty))
  Source:                IntB_2 (FF)
  Destination:           XorB_2 (FF)
  Requirement:           8.000ns
  Data Path Delay:       2.542ns (Levels of Logic = 1)
  Clock Path Skew:       0.000ns
  Source Clock:          clk0  falling at 2.000ns
  Destination Clock:     clk90 rising at 10.000ns
…
Slack (setup path): 13.292ns (requirement - (data path - clock path skew + uncertainty))
  Source:                IntC_2 (FF)
  Destination:           XorB_2 (FF)
  Requirement:           15.000ns
  Data Path Delay:       2.594ns (Levels of Logic = 1)
  Clock Path Skew:       -0.086ns
  Source Clock:          clk0  falling at 10.000ns
  Destination Clock:     clk90 rising at 25.000ns
  Clock Uncertainty:     0.200ns
```

Clk2x Clock Domain

Because the clocks produced by the DCM, DLL, PLL, or PMCD component are related, the timing tools consider this relationship during analysis. The following figure shows a simple design of a CLK2X clock domain. The clock is driven by the same clock source. This clock source is an output of a DCM, DLL, PLL, or PMCD component.

The timing analysis tool reports the active edges of the clock and the corresponding time for the data path between the synchronous elements. This clock domain has the requirement of the original requirement. The phase shifting is the same as the phase shifting of the original requirement.

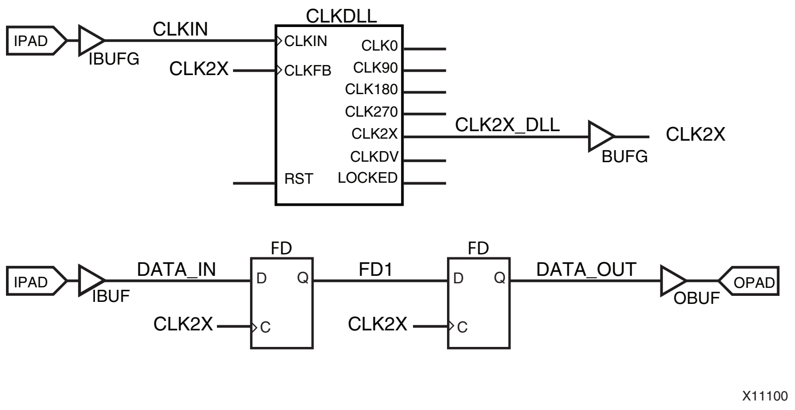


Figure 6-8: **Clk2x DCM Output Schematic**

Clk2x Timing Report Example

```
Slack (setup path): -1.663ns (requirement - (data path - clock path skew + uncertainty))
  Source:                IntA_3 (FF)
  Destination:           OutB_3 (FF)
  Requirement:           2.000ns
  Data Path Delay:       3.443ns (Levels of Logic = 0)
  Clock Path Skew:       -0.020ns
  Source Clock:          clk2x rising at 0.000ns
  Destination Clock:     clk2x falling at 2.000ns
  Clock Uncertainty:     0.200ns
```

CLKDV/CLKFX Clock Domain

Because the clocks produced by the DCM, DLL, PLL, or PMCD component are related, the timing tools consider this relationship during analysis. Use the CLKDV and CLKFX outputs to create clock signals that are derivatives of the original input clock signal. See Table 3-1, Transformation of Period Constraint Through DCM.

The clock is driven by two different outputs of the DCM, DLL, PLL, or PMCD component. The timing analysis tool reports the active edges of the clock and the corresponding time for the data path between the synchronous elements.

The following figure shows a simple design of a CLKDV clock domain, with the DIVIDE_BY factor set to 2.

- This clock domain has twice the requirement as the original requirement.
- The phase shifting is the same as the phase shifting of the original requirement.

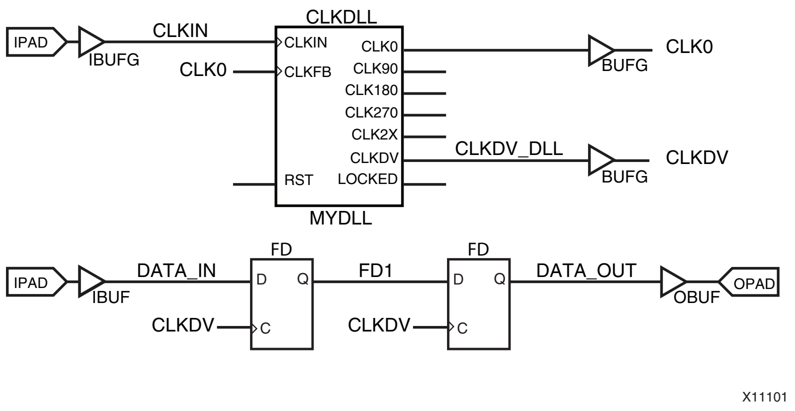


Figure 6-9: **ClkDV DCM Output Schematic**

ClkDV Timing Report Example

```
Slack (setup path): 1.909ns (requirement - (data path - clock path skew + uncertainty))
  Source:              XorC_7 (FF)
  Destination:         OutC_7 (FF)
  Requirement:         4.000ns
  Data Path Delay:     1.810ns (Levels of Logic = 0)
  Clock Path Skew:     0.000ns
  Source Clock:        clk0  rising at 0.000ns
  Destination Clock:   clkdv rising at 4.000ns
  Clock Uncertainty:   0.281ns
```

From:To (Multi-Cycle) Analysis

multicycleanalysis

The analysis of path exceptions is associated with From-To constraints. The path exception constraints override global constraints for a specific set of paths specified in the From:To constraint.

This constraint specifies a unique timing requirement for a specific set of paths with a faster or slower requirement than the global timing constraints. The requirement can be a value, or a Timing Ignore.

Header Summary

The analysis of the exception constraint starts with a header summary. The header summary is a summary of the specific constraint and contains:

- Constraint syntax
- Number of paths and endpoints covered by the constraint
- Setup errors
- Hold errors

The information in the header summary:

- Verifies that the constraint covers the expected number of paths and endpoint.
- Provides an overall worst case performance of the constraint.

Analysis of the Exception Constraint Path

The analysis of the exception constraint path includes the path details for the clock and data paths. This analysis contains all the information for a single path.

First Paragraph Contents

The first paragraph includes:

- Overall slack value
- Source and destination design elements
- Source clock and destination clock signals and clock edges
- Total data path delay
- Clock skew
- Clock uncertainty
- Slack equations
- Clock uncertainty equations.

Second Paragraph Contents

The second paragraph provides the path details for the clock and data paths for the output interface. This includes the description of all device resources utilized and the routing delays for both clocking and data paths.

Analysis of the From:To (Multi-Cycle) Constraint

The analysis of the From:To (Multi-Cycle) constraint includes the clock skew between the source and destination synchronous elements.

Clock skew is calculated based upon the clock path to the destination synchronous element, minus the clock path to the source synchronous element.

Clock skew is analyzed for all constrained clocks. The analysis includes:

- Setup analysis for all device families.
- Setup and hold analysis for Virtex-5 devices and newer.

DATAPATHONLY Keyword

The DATAPATHONLY keyword:

- Instructs the tools to ignore the clock skew in From:To constraints.
- Indicates that the From:To constraint does not take clock skew or phase information into account during analysis.
- Results in only the data path between the groups being considered and analyzed.

Setup and Hold Analysis

Setup paths are sorted by slacks, based upon the following equation:

```
Tsu slack = constraint_requirement - Tclock_skew - Tdata_path - Tsu
```

The setup analysis of a From:To is done by default. The hold analysis is reported for Virtex-5 devices and newer by default.

For older devices, set the environment variable (XIL_TIMING_HOLDCHECKING YES) to enable the hold analysis.

Hold analyses are performed on register-to-register paths by taking the data path (**`Tcko+Troute_total+Tlogic_total`**) and subtracting the clock skew (T**`dest_clk - Tsrc_clk`**) and the register hold delay (**`Th`**).

Evaluating the Hold Check

The TWR Report uses slack to evaluate the hold check.

Table 6-2: **Hold Violations**

Slack	Hold Violation
Negative	Yes
Positive	No

Hold Slack Calculations

Use the following equation for hold slack calculations:

```
Hold Slack = Tdata - Tskew - Th
```

Detailed Path Reporting

The detailed path is reported under the constraint that contains that data path. The path is listed by the slack with respect to the requirement.

There is a (**`-Th`**) identifier of the hold path delay type. This (**`-Th`**) identifier appears after the hold delay type to help identify race conditions and hold violations.

Hold Analyses

Hold analyses are performed on all global and local clock resources. The data path is not adjusted to show possible variances in the PVT across the silicon.

Hold Violations

Hold violations are rare. A very short data path delay and a large clock skew must coincide before this problem occurs.

If a hold violation does occur, PAR can change the routing to fix the violation. PAR and the timing engines:

- Reduce the clock skew.
- Increase the clock delay for a specific data path if necessary.

Hold Slack

The hold slack is not related to the constraint requirement. This may be confusing when reviewing the slack and the minimum delay ns period for the constraint.

The hold slack is related to the relationship between the clock skew and the data path delay. Only the slack from setup paths affects the minimum delay ns period for the constraint.

Accounting for Known External Skew

The From:To constraint requirement will account for any known external skew between the clock sources if:

- The endpoint registers do not share a common clock, or
- The clocks are unrelated to each other.

If the registers share any single common clock source, the skew is calculated only for the unique portions of the clock path to the synchronous elements. If no common clock source points are found, the skew is the difference between the maximum and minimum clock paths.

The path header:

- Reports the clock skew.
- Does not include the delay details to the source clock pin and destination clock pin.

To determine these delays, use **`Analyze Against User Specified Paths ... by defining Endpoints...`** in Timing Analyzer.

1. Specify the clock pad input as the source.
2. Specify the registers or synchronous elements in the hold or setup analysis as the destination.

The clock delay from the pad to each register clock pin is reported. This custom analysis also works for DLL, DCM, and PLL clock paths.

To obtain the clock skew, subtract the *destination clock delay* from the *source clock delay*. The paths are sorted by total path delay and not slack.

Example One

Constrain the DQS path from an IDDR to the DQ CE pins to be approximately one-half cycle. This insures that the DQ clock enables are de-asserted before any possible DQS glitch at the end of the read postamble can arrive at the input to the IDDR. This value is clock-frequency dependent.

```
INST */gen_dqs*.u_iob_dqs/u_iddr_dq_ce TNM = TNM_DQ_CE_IDDR;
INST */gen_dq*.u_iob_dq/gen_stg2_*.u_iddr_dq TNM = TNM_DQS_FLOPS;
TIMESPEC TS_DQ_CE = FROM TNM_DQ_CE_IDDR TO TNM_DQS_FLOPS TS_SYS_CLK * 2;
```

The requirement is based upon the system clock.

Example Two

Constrain the paths from a select pin of a MUX to the next stage of capturing synchronous elements. This value is clock-frequency dependent:

```
NET clk0 TNM = FFS TNM_CLK0;
NET clk90 TNM = FFS TNM_CLK90;
# MUX Select for either rising/falling CLK0 for 2nd stage read capture
INST */u_phy_calib_0/gen_rd_data_sel*.u_ff_rd_data_sel TNM = TNM_RD_DATA_SEL;
TIMESPEC TS_MC_RD_DATA_SEL = FROM TNM_RD_DATA_SEL TO TNM_CLK0 TS_SYS_CLK * 4;
```

This requirement is based upon the system clock.

Example Three

Constrain the path between DQS gate driving IDDR and the clock enable input to each DQ capture IDDR in that DQS group. This requirement is frequency dependent. The user sets the following requirement:

```
INST */gen_dqs[*].u_iob_dqs/u_iddr_dq_ce TNM = TNM_DQ_CE_IDDR;
INST */gen_dq[*].u_iob_dq/gen_stg2_*.u_iddr_dq TNM = TNM_DQS_FLOPS;
TIMESPEC TS_DQ_CE = FROM TNM_DQ_CE_IDDR TO TNM_DQS_FLOPS 1.60 ns;
```

This requirement is based upon a system clock of 333 MHz.

Offset In Analysis

offsetinanalysis

The input timing analysis includes the following constraints:

- Offset In
- From:Pads:To
- Both Offset In and From:Pads:To

The input timing constraint covers the data path from the external pin or pad of the FPGA to the internal synchronous element or register that captures that data.

The traditional constraint for this path is Offset In.

- Offset In specifies the input timing for the design.
- Offset In defines the relationship between the data and clock edge used to capture that data at the pin or pads of the device.

This analysis is used to analyze the setup and the hold paths of the synchronous elements, which capture the data. The internal routing and delays of the clock and data paths are included in the Offset In analysis in the timing analysis tools. The frequency and the phase transformation of the clock, clock uncertainties, IOStandard, and other data delay adjustments.

Worst Case Paths

The Timing Object Table shows the worst case paths for the selected constraint. The table shows each patch per row, including common timing analysis details.

The common timing analysis details include the following elements:

- Slack
- Data Path
- Clock Path
- Source
- Destination

Constraint Summary

The details of each Timing Report constraint show a summary of the constraint, including:

- The number of paths and the endpoints covered by the constraint
- Setup errors
- Hold errors

Use the analysis information to:

- Verify that the constraint covered the expected number of path and endpoints.
- View a high-level view of the performance of the constraint.

Bus Base Analysis

The bus base analysis for input timing paths includes the input timing interfaces consisting of several data signals associated with a single input clock. The interface depends greatly on the entire bus operating correctly. The bus-based timing analysis of the interface and the analysis of each bit of the bus is included.

The bus base analysis provides specific detailed analysis for each bit of the bus in order to:

- Determine the common sources of errors
- Determine how to adjust clock and data delay to optimize bus performance.

Bus Base Analysis Summary

During the bus base analysis, the datasheet section of the Timing Report contains a sub-section with a summary of the bus base analysis.

High Level Timing Detail

The sub-section provides a high level timing detail for each bit of the interface bus. These details are based upon the detailed section of the Timing Report under the Offset In constraint. They include:

- Setup and hold requirements
- Setup and hold slacks for each bit of the capturing register or synchronous element inside the device

Overall Bus Performance

The sub-section also displays more information on the overall bus performance of the bus. This includes the worst case summary row and source offset to center column.

- Source Offset to Center

  Provides the data path delay adjustments required to center the data bits of the interface over the clock edge to provide maximum timing margin for this interface.
- Ideal Clock Offset to Actual Clock

  Provides the clock path delay adjustment required to center the clock edge with respect to the bus. This additional clock path delay is usually done by Phase shifting the clock through a clock modifying block (DCM, PLL, or MMCM).
- Worst Case Data

  Provides the overall worst case setup, plus hold time window for the bus interface.

Detailed Path Analysis

The detailed path analysis section of the Timing Report provides clock and data path details of the input interface. This analysis includes all the necessary delays for the setup and hold analysis of the input interface.

Detailed Path Analysis Section Contents

This section discusses the detailed path analysis section contents.

Summary Header

For each Offset In constraint, a summary header provides information about the:

- Constraint syntax
- Paths analyzed
- Endpoints analyzed

Path Header

For each path analyzed, a path header provides:

- Summary of the input timing path performance in a slack value
- Slack equation of the timing check
- Source pad and destination synchronous element information
- Capturing clock network name and clock edge
- Clock and data path delay totals
- Clock uncertainty

Data and Cock Path Details

The data and clock path details include a detailed description of all component and routing network delays utilized for both the clock and data paths of an input interface.

Offset In Constraint

The Offset In constraint:

- Defines the Pad-To-Setup timing requirement
- Is an external clock-to-data relationship specification

Setup Requirement

The setup requirement is:

(**`data_delay + setup - clock_delay - clock_arrival`**)

When analyzing the setup requirement, the Offset In constraint takes into account the:

- Clock delay
- Clock edge
- DLL or DCM introduced clock phase

Clock Arrival

Clock arrival takes into account any clock phase generated by the DLL or DCM, or clock edge. If the Timing Report does not display a clock arrival time for an Offset constraint, the timing analysis tools did not analyze a Period constraint for that specific synchronous element.

Creating Pad-To-Setup Requirements

When you create pad-to-setup requirements, incorporate any phase or Period adjustment factor into the value specified for an Offset In constraint.

For the following example, see the schematic in Figure 3-3, Timing Name on the A0 Net Traced Through Combinatorial Logic to Synchronous Elements (Flip-Flops).

If the net from the CLK90 pin of the DLL or DCM clocks a register, adjust the Offset value by one quarter of the Period constraint value.

For example, if the Period constraint value is 20 ns, and is from the CLK90 of the DCM, adjust the Offset In value by an additional 5 ns.

- Original Constraint

  ```
  NET "PAD_IN" OFFSET = IN 10 BEFORE "PADCLKIN";
  ```

- Modified Constraint

  ```
  NET "PAD_IN" OFFSET = IN 15 BEFORE "PADCLKIN"
  ```

The clock net name required for Offset constraints is the clock net name attached to the IPAD. In the above example, the clock pad is PADCLKIN, not CLK90.

Offset In Before Constraint

The Offset In Before constraint defines the time available for data to propagate from the pad to setup at the synchronous element. See the following figure. This time can be visualized as the time that the data arrives at the edge of the device before the next clock edge arrives at the device.

This **`OFFSET = IN 2 ns BEFORE clock_pad`** constraint reads that the data is valid at the input data pad, some time period (2 ns) BEFORE the reference clock edge arrives at the clock pad. The tools calculate and control internal data and clock delays to meet the flip-flop setup time.

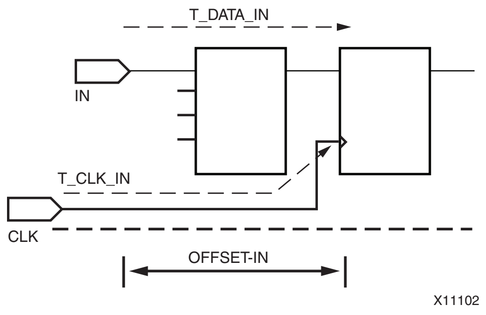


Figure 6-10: **Circuit Diagram with Calculation Variables for Offset In Before Constraints**

Setup Relationship Equation

The following equation defines the setup relationship.

```
TData + TSetup - TClock <= Toffset_IN_BEFORE
```

where

```
TSetup = Intrinsic Flip Flop setup time
TClock = Total Clock path delay to the Flip Flop
TData = Total Data path delay from the Flip Flop
Toffset_IN_BEFORE = Overall Setup Requirement
```

Offset In Requirement Value

The Offset In requirement value is used as a setup time requirement of the FPGA device during the setup time analysis.

Valid Keyword

The Valid keyword:

- Is used in conjunction with the requirement to create a hold time requirement during a hold time analysis.
- Specifies the duration of the incoming data valid window, and the timing analysis tools do a hold time analysis.

By default, the Valid value is equal to the Offset time requirement, which specifies a zero hold time requirement (see the following figure).

If the Valid keyword is not specified, no hold analysis is done by default. In order to receive hold analysis without the Valid keyword, use the **`fastpaths`** option (**`trce -fastpaths`**) during timing analysis.

Hold Relationship Equation

The following equation defines the hold relationship.

```
TClock - TData + Thold <= Toffset_IN_BEFORE_VALID
```

where

```
Thold = Intrinsic Flip Flop hold time
TClock = Total Clock path delay to the Flip Flop
TData = Total Data path delay from the Flip Flop
Toffset_IN_BEFORE_VALID = Overall Hold Requirement
```

Offset In Constraint with Valid Keyword Coding Example

```
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK RISING;
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK FALLING;
```

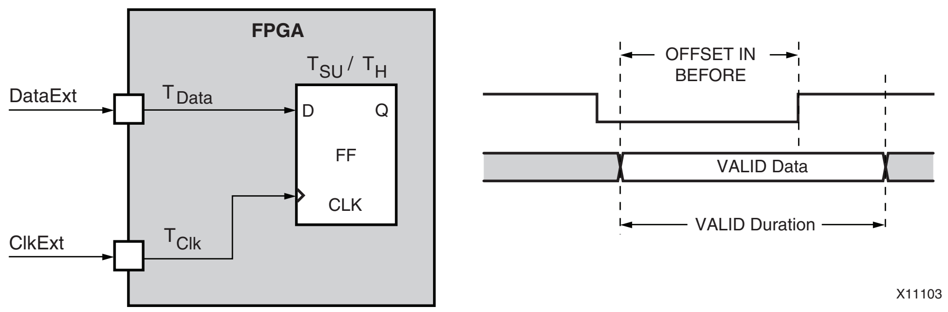


Figure 6-11: **Offset In Constraint with Valid Keyword Schematic Example**

The Offset constraint is analyzed with respect to the rising clock edge, which is specified with the High keyword of the Period constraint. Set the Offset constraint to Rising or Falling to override the High or Low setting defined by the Period constraint.

This is extremely useful for DDR design, with a 50 percent duty cycle, when the signal is capturing data on the rising and falling clock edges, or producing data on rising and falling clock edges.

For example, if the Period constraint is set to High, and the Offset constraint is set to Falling, the falling edged synchronous elements have the clock arrival time set to zero.

Offset In Constraint Set to Rising and Falling Coding Example

Following is an example of the Offset In constraint set to Rising and Falling:

```
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK FALLING;
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK RISING;
```

The equation for external setup included in the Offset In analysis of the FPGA device is:

```
External Setup = Data Delay + Flip Flop Setup time - Prorated version of Clock Path Delay
```

The longer the clock path delay, the smaller the external setup time becomes. The prorated clock path delay is used to obtain an accurate setup time analysis. The general prorating factors are 85% for Global Routing and 80% for Local Routing.

The prorated clock path delays are not used for families older than Virtex-II device families.

The equation for external hold included in the Offset In analysis of the FPGA device is:

```
External Hold = Clock Path Delay + Flip Flop Hold time - Prorated version of Data Delay
```

If the data delay is longer than the clock delay, the result is a smaller hold time. The prorated data delays are similar to the prorated values in the setup analysis.

The prorated data delays are not used for families older than Virtex-II device families.

Offset In Constraint Simple Example

A simple example of the Offset In constraint has an initial clock edge at 0 ns based upon the Period constraint. The Timing Report displays the initial clock edge as the clock arrival time.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

If the Timing Report does not display a clock arrival time, the timing analysis tools did not recognize a Period constraint for that particular synchronous element.

In the following figure, the Offset requirement is 3 ns before the initial clock edge. The equation used in timing analysis is:

```
Slack = (Requirement - (Data Path - Clock Path - Clock Arrival))
```

Constraint Syntax Example

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock;
```

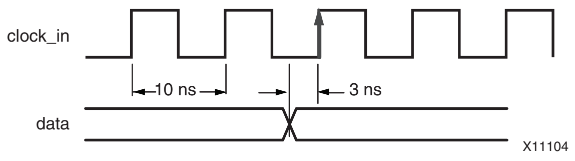


Figure 6-12: **Timing Diagram of Simple Offset In Constraint**

Timing Report Example

```
Slack: -0.191ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:                reset (PAD)
  Destination:           my_oddrA_ODDR_inst/FF0 (FF)
  Destination Clock:     clock0_ddr_bufg rising at 0.000ns
  Requirement:           3.000ns
  Data Path Delay:       2.784ns (Levels of Logic = 1)
  Clock Path Delay:      -0.168ns (Levels of Logic = 3)
  Clock Uncertainty:     0.239ns
```

Two-Phase Example

A two-phase or both clock edge example of the Offset In constraint has an initial clock edge which correlates to the two edges of the clock:

- The first clock edge is 0 ns based upon the Period constraint
- The second clock edge is one-half the Period constraint

The Timing Report displays the clock arrival time for each edge of the clock.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In this example, the Period constraint has the clock arrival on the falling edge, based upon the Falling keyword. Therefore, the clock arrival time for the falling edge synchronous elements is zero. The rising edge synchronous elements is one-half the Period constraint. If both edges are used, as in Dual-Data Rate, two Offset constraints are created: one for each clock edge.

In the following figure, the Offset requirement is 3 ns before the initial clock edge. If the Period constraint is set to High, and the Offset In constraint is set to Falling, the following constraints produce the same example report:

```
TIMESPEC TS_clock = PERIOD clock 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock RISING;
OFFSET = IN 3 ns BEFORE clock FALLING;

TIMESPEC TS_clock=PERIOD clock 10 ns LOW 50%;
OFFSET=IN 3 ns BEFORE clock;
```

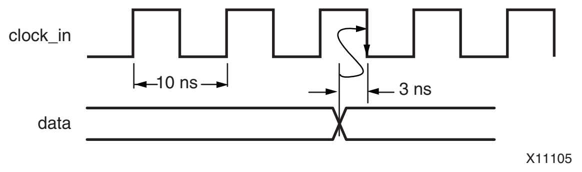


Figure 6-13: **Timing Diagram with Two-Phase Offset In Constraint**

Timing Report Example

```
Slack: 0.231ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:                DataD<9> (PAD)
  Destination:           TmpAa_1 (FF)
  Destination Clock:     clock0_ddr_bufg falling at 0.000ns
  Requirement:           3.000ns
  Data Path Delay:       2.492ns (Levels of Logic = 2)
  Clock Path Delay:      -0.038ns (Levels of Logic = 3)
  Clock Uncertainty:     0.239ns
```

Phase-Shifted Example

A DCM phase-shifted clock (**`CLK90`**) example of the Offset In constraint has an initial clock edge at 0 ns based upon the Period constraint. Because the clock is phase-shifted by the DCM, the Timing Report displays the clock arrival time as the phase-shifted amount. If the **`CLK90`** output is used, the phase-shifted amount is one quarter of the Period.

In this example:

- The Period constraint has the initial clock arrival on the rising edge.
- The clock arrival value is at 2.5 ns.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In the following figure, the Offset requirement is 3 ns before the initial clock edge.

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET=IN 3 ns BEFORE clock;
```

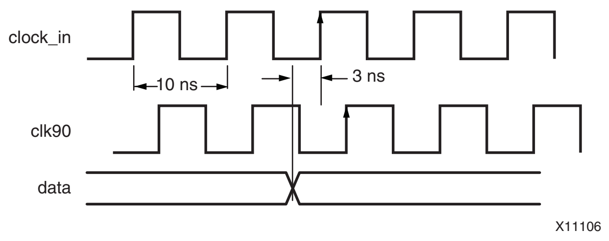


Figure 6-14: **Timing Diagram for Phase Shifted Clock in Offset In Constraint**

Timing Report Example

```
Slack: 2.309ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:                 reset (PAD)
  Destination:            my_oddrA_ODDR_inst/FF0 (FF)
  Destination Clock:      clock90_bufg rising at 2.500ns
  Requirement:            3.000ns
  Data Path Delay:        2.784ns (Levels of Logic = 1)
  Clock Path Delay:       -0.168ns (Levels of Logic = 3)
  Clock Uncertainty:      0.239ns
```

Fixed Phase-Shifted Example

A DCM fixed phase-shifted clock example of the Offset In constraint has an initial clock edge at 0 ns based upon the Period constraint.

- Because the clock is phase-shifted by the DCM, the Timing Report displays the clock arrival time as the phase-shifted amount.
- If the CLK0 output is phase-shifted by a user-specified amount, the phase-shifted amount is a percentage of the Period.

In the following example:

- The Period constraint has the initial clock arrival on the rising edge.
- The clock arrival value is at the fixed phase shifted amount.

  See the example Timing Report.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In the following figure, the Offset requirement is 3 ns before the initial clock edge.

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock;
```

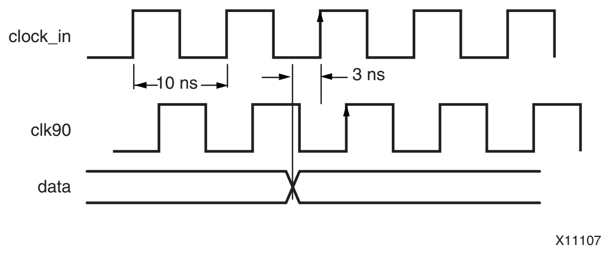


Figure 6-15: **Timing Diagram of Fixed Phase Shifted Clock in Offset In Constraint**

Timing Report Example

```
Slack: 4.731ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:               DataD<9> (PAD)
  Destination:          TmpAa_1 (FF)
  Destination Clock:    clock1_fixed_bufg rising at 4.500ns
  Requirement:          3.000ns
  Data Path Delay:      2.492ns (Levels of Logic = 2)
  Clock Path Delay:     -0.038ns (Levels of Logic = 3)
  Clock Uncertainty:    0.239ns
```

Offset In Constraint Dual-Data Rate Example

A Dual-Data Rate example of the Offset In constraint has an initial clock edge at 0 ns and half the Period constraint, which correlates to the two clock edges. The Timing Report displays the clock arrival time for each edge of the clock.

Because the timing analysis tools do not automatically adjust any of the clock phases during analysis, the constraints must be manually adjusted for each clock edge.

The timing analysis tools offer two options to manage the falling edge clock arrival time.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

Manage Falling Edge Clock Arrival Time Option One

For the first option to manage the falling edge clock arrival time:

1. Create two time groups:
   - One time group for *rising* edge synchronous elements
   - One time group for *falling* edge synchronous elements

2. Create an Offset In constraint for each time group.

   The second Offset In constraint has a different requirement.

The falling edge Offset In constraint requirement equals:

(original requirement) minus (one-half the Period constraint)

Therefore, if the original requirement is (3 ns with a Period of 10 ns), the falling edge Offset In constraint requirement equals (-2 ns).

This compensates for the clock arrival time associated with the falling edge synchronous elements. The negative value is legal in the constraints language.

Manage Falling Edge Clock Arrival Time Option Two

For the second option to manage the falling edge clock arrival time:

1. Create one time group and one corresponding Offset In constraint with the original constraint requirement for each clock edge.
2. Add the Rising or Falling keyword if the Period constraint has the High keyword.

The analysis with the Rising or Falling keywords is based upon the active clock edge for the synchronous element.

- The requirement for the *rising* clock edge elements is set in the Offset In Rising constraint.
- The requirement for the *falling* clock edge elements are set in the Offset In Falling constraint.

In this example, because the Period constraint has the clock arrival on both the Rising edge and Falling edge, the clock arrival value is 0 ns and 5 ns. In the following figure, the Offset requirement is 3 ns before the initial clock edge.

Constraint Syntax Example

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock RISING;
OFFSET = IN 3 ns BEFORE clock FALLING;
```

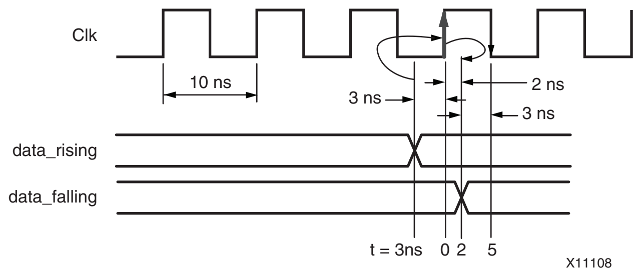


Figure 6-16: **Timing Diagram for Dual Data Rate in Offset In Constraint**

Timing Report Example for OFFSET = IN 3 ns Before Clock Rising

```
Slack: 0.101ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:              DataA<3> (PAD)
  Destination:         TmpAa_3 (FF)
  Destination Clock:   clock0_ddr_bufg rising at 0.000ns
  Requirement:         3.000ns
  Data Path Delay:      2.654ns (Levels of Logic = 2)
  Clock Path Delay:     -0.006ns (Levels of Logic = 3)
  Clock Uncertainty:    0.239ns
```

Timing Report Example for OFFSET = IN 3 ns Before Clock Falling

```
Slack: 0.101ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:               DataA<3> (PAD)
  Destination:          TmpAa_3 (FF)
  Destination Clock:    clock0_ddr_bufg falling at 0.000ns
  Requirement:          3.000ns
  Data Path Delay:      2.654ns (Levels of Logic = 2)
  Clock Path Delay:     -0.006ns (Levels of Logic = 3)
  Clock Uncertainty:    0.239ns
```

Offset In After Constraint

The Offset In After constraint describes the time used by the data external to the FPGA device.

Offset In subtracts this time from the Period declared for the clock to determine the time available for the data to propagate from the pad to the setup at the synchronous element.

This time can be visualized as the difference of data arriving at the edge of the device after the current clock edge arrives at the edge of the device.

This **`OFFSET = IN 2 ns AFTER clock_pad`** constraint reads that the data to be registered in the FPGA device is available on the FPGA input pad, some time period (2ns), *after* the reference clock edge is seen by the upstream device. For the purposes of the Offset constraint syntax, assume no skew on CLK between the chips.

The following equation defines this relationship.

```
TData + TSetup - TClock <= TPeriod - Toffset_IN_AFTER
```

where

```
TSetup = Intrinsic Flip Flop setup time
TClock = Total Clock path delay to the Flip Flop
TData = Total Data path delay from the Flip Flop
TPeriod = Single Cycle PERIOD Requirement
Toffset_IN_AFTER = Overall Setup Requirement
```

A Period or Frequency constraint is required for Offset In constraints with the After keyword.

Offset Out Analysis

offsetoutanalysis

The output interface analysis is done under the Offset Out output timing constraint. The output timing analysis covers the data path from the external clock pad through any logic and from the synchronous element that is tied to the external data pad. The constraint defines the maximum time from the time the clock edge arrives at the external pad until the first data appears at the external data pad.

Detailed Path Analysis

The timing analysis includes internal factors that affect the delays associated with the clock and data paths. These internal factors include:

- The frequency and phase transformation of the clock
- The clock uncertainties
- The data delay adjustment

Bus Base Analysis

The datasheet section of the Timing Report reports the overall bus skew relative to a reference pin or fastest bit for source synchronous interfaces.

Bus-Based Timing Analysis

Output timing interfaces typically consist of several data signals associated with a single input clock. To ensure that the entire bus is operating correctly, the bus-based timing analysis of the interface reports the worst case bus skew across the entire bus in a source synchronous design.

Bit Analysis

The bus-based timing analysis reports the analysis of each bit of the bus, including:

- Source synchronous elements
- Pad element
- Overall delay

  The overall delay includes the delay from the clock input to the output data bit.
- Bus skew

  The bus skew is the skew of each bit relative to the reference pin or the smallest data bit delay.

The detail of the path analysis of the output interface includes the analysis of the clock and data path of the output interface. The analysis includes the information for a single data path for single output data path.

Timing Object Table

The Timing Object Table provides a timing summary for the path analysis in Timing Analyzer, including:

- Output timing of the path
- Contribution of the clock and data components of the path

First Paragraph Contents

The first paragraph contains the path summary for this single path, including:

- Overall performance summary in a slack value with the slack equation.
- Source synchronous element
- Destination pad element
- Transmitting clock network description
- Clock and data path delay details
- Clock uncertainty value
- Clock uncertainty equation

Second Paragraph Contents

The second paragraph contains the path details for the clock and data paths for the output interface, including:

- Description of all device resources utilized
- Routing delays for both clocking and data paths

Header Summary Section

For each analysis of the Offset Out constraint, a header summary section includes information about:

- Constraint syntax
- Number of paths and endpoints analyzed by the constraint
- Timing errors

The header summary section also:

- Verifies that the constraint has covered the expected number of path and endpoints
- Reviews the worst case performance for this constraint.

Offset Out Constraint

The Offset Out constraint:

- Defines the Clock-to-Pad timing requirements.
- Is an external clock-to-data specification.
- Takes into account the following when analyzing the clock to out requirements:
    - Clock delay
    - Clock edge
    - DLL or DCM introduced clock phase

```
Clock to Out = clock_delay + clock_to_out + data_delay + clock_arrival
```

Clock Arrival Time

Clock arrival time takes into account any clock phase generated by the DLL or DCM component, or clock edge.

If the Timing Report does not display a clock arrival time, the timing analysis tools did not analyze a Period constraint for that specific synchronous element.

Clock-to-Pad Requirements

When you create clock-to-pad requirements, incorporate any phase or Period adjustment factor into the value specified for an Offset Out constraint. For the following example, see Figure 6-6, Clk90 Clock Phase Between DCM Outputs Schematic.

If the register is clocked by the net from the CLK90 pin of the DCM, which has a Period of 20 ns, adjust the Offset value by 5 ns less than the original constraint.

- Original Constraint

  ```
  NET "PAD_OUT" OFFSET = OUT 15 AFTER "PADCLKIN";
  ```

- Modified Constraint

  ```
  NET "PAD_OUT" OFFSET = OUT 10 AFTER "PADCLKIN";
  ```

Offset Out After Constraint

The Offset Out After constraint defines the time available for the data to propagate from the synchronous element to the pad. See the following figure.

This time can be visualized as the data leaving the edge of the device after the current clock edge arrives at the edge of the device.

Offset Out After Constraint Example

`OFFSET = OUT 2 ns AFTER clock_pad`

This constraint reads that the data to be registered in the downstream device is available on the FPGA data output pad 2 ns *after* the reference clock pulse is seen by the FPGA at the clock pad.

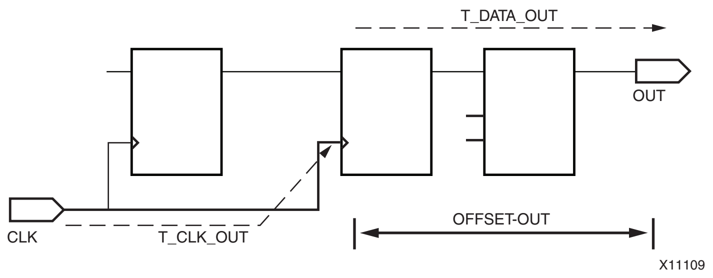


Figure 6-17: **Circuit Diagram with Calculation Variables for Offset Out After Constraints**

The following equation defines this relationship.

```
Q + TData2Out + TClock <= Toffset_OUT_AFTER
```

where

```
TQ = Intrinsic Flip Flop Clock to Out
TClock = Total Clock path delay to the Flip Flop
TData2Out = Total Data path delay from the Flip Flop
Toffset_OUT_AFTER = Overall Clock to Out Requirement
```

The analysis of this constraint involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) do not exceed the specified offset.

Offset Rising and Offset Falling Keywords

The Offset Rising and Offset Falling keywords can override the High or Low keyword defined by the Period constraint. This is useful for DDR design, with a 50% duty cycle, when the signal is capturing data on the rising and falling clock edges or producing data on a rising and falling clock edges.

For example, if the Period constraint is High, and the Offset constraint is Falling, the clock arrival time of the falling edged synchronous elements is set to zero.

Offset Out Set to Rising or Falling Example

```
TIMEGRP DATA_OUT OFFSET = OUT 10 AFTER CLK FALLING;
TIMEGRP DATA_OUT OFFSET = OUT 10 AFTER CLK RISING;
```

Offset Out Constraint Simple Example

A simple example of the Offset Out constraint has the initial clock edge at 0 ns based upon the Period constraint. The Timing Report displays the initial clock edge as the clock arrival time.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

If the Timing Report does not display a clock arrival time, the timing analysis tools did not recognize a Period constraint for that particular synchronous element.

In the following figure, the Offset requirement is 3 ns. The equation used in timing analysis is:

```
Slack =  (Requirement - (Clock Arrival + Clock Path + Data Path))

TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT 3 ns AFTER clock;
```

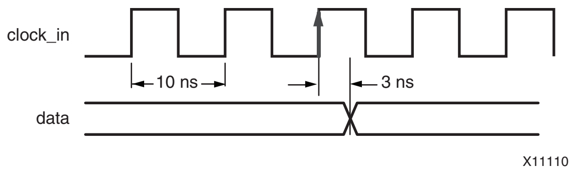


Figure 6-18: **Timing Diagram of Simple Offset Out Constraint**

Timing Report Example

```
Slack: -0.865ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:               OutD_7 (FF)
  Destination:          OutD<7> (PAD)
  Source Clock:         clock3_std_bufg rising at 0.000ns
  Requirement:          3.000ns
  Data Path Delay:      3.405ns (Levels of Logic = 1)
  Clock Path Delay:     0.280ns (Levels of Logic = 3)
  Clock Uncertainty:    0.180ns
```

Two-Phase Example

In a two-phase (use of both edges) example of the Offset Out constraint, the initial clock edge correlates to the two edges of the clock.

- The first clock edge is at 0 ns based upon the Period constraint.
- The second clock edge is one-half the Period constraint.

The Timing Report displays the clock arrival time for each edge of the clock. In this example, the clock arrival for the Period Low constraint is on the falling edge. Therefore the clock arrival time for the falling edge synchronous elements is zero. The rising edge synchronous elements are half the Period constraint.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In the following figure, the Offset requirement is 3 ns.

```
TIMESPEC TS_clock=PERIOD clock 10 ns LOW 50%;
OFFSET = IN 3 ns AFTER clock;
```

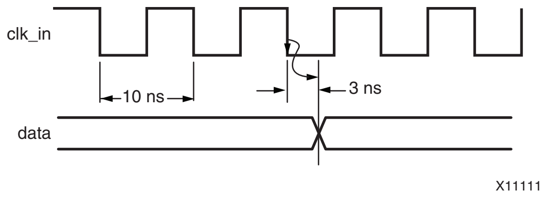


Figure 6-19: **Timing Diagram of Two-Phase in Offset Out Constraint**

Timing Report Example

```
Slack: -0.865ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:               OutD_7 (FF)
  Destination:          OutD<7> (PAD)
  Source Clock:         clock3_std_bufg falling at 0.000ns
  Requirement:          .3.000ns
  Data Path Delay:      3.405ns (Levels of Logic = 1)
  Clock Path Delay:     0.280ns (Levels of Logic = 3)
  Clock Uncertainty:    0.180ns
```

Phase-Shifted Example

A DCM phase-shifted, CLK90, example of the Offset Out constraint has the initial clock edge at 0 ns based upon the Period constraint. Because the clock is phase-shifted by the DCM, the Timing Report displays the clock arrival time as the phase-shifted amount. If the CLK90 output is used, the phase-shifted amount is one quarter of the Period. The clock arrival time corresponds to the phase shifting amount, which is 2.5 ns in this case.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In the following figure, the Offset requirement is 5 ns.

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT R ns AFTER clock;
```

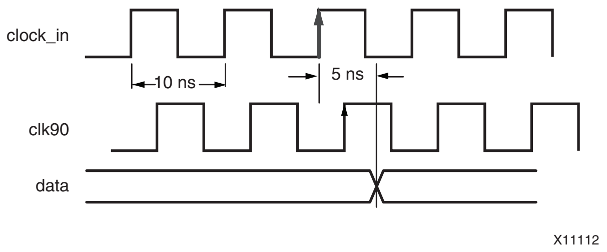


Figure 6-20: **Timing Diagram of Phase Shifted Clock in Offset Out Constraint**

Timing Report Example

```
Slack: -1.365ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:                OutD_7 (FF)
  Destination:           OutD<7> (PAD)
  Source Clock:          clock3_std_bufg rising at 2.500ns
  Requirement:           5.000ns
  Data Path Delay:       3.405ns (Levels of Logic = 1)
  Clock Path Delay:      0.280ns (Levels of Logic = 3)
  Clock Uncertainty:     0.180ns
```

Fixed Phase-Shifted Example

A DCM fixed phase-shifted example of the Offset Out constraint has the initial clock edge at 0 ns, based upon the Period constraint.

- Because the clock is phase-shifted by the DCM, the Timing Report displays the clock arrival time as the phase-shifted amount.
- If the CLK0 output is phase-shifted by a user-specified amount, the phase-shifted amount is a percentage of the Period.

In this example:

- The Period constraint has the initial clock arrival on the rising edge.
- The clock arrival value is at the fixed phase-shifted amount.

The clock arrival time corresponds to the phase-shifting amount.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

In the following figure, the Offset requirement is 5 ns.

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT 5 ns AFTER clock;
```

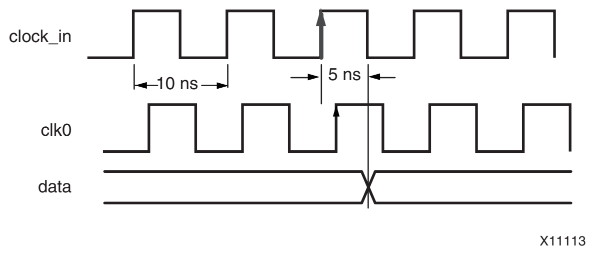


Figure 6-21: **Timing Diagram of Fixed Phase Shifted Clock in Offset Out Constraint**

Timing Report Example

```
Slack: 0.535ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:                OutD_7 (FF)
  Destination:           OutD<7> (PAD)
  Source Clock:          clock3_std_bufg rising at 0.600ns
  Requirement:           5.000ns
  Data Path Delay:       3.405ns (Levels of Logic = 1)
  Clock Path Delay:      0.280ns (Levels of Logic = 3)
Clock Uncertainty:    0.180ns
```

Offset Out Constraint Dual-Data Rate Example

A dual-data rate example of the Offset Out constraint has the initial clock edge at 0 ns and one half the Period constraint, which correlates to the two edges of the clock. The Timing Report displays the clock arrival time for each edge of the clock.

Because the timing analysis tools do not automatically adjust any of the clock phases during analysis, the constraints must be manually adjusted for each clock edge.

The timing analysis tools offer two options to manage the falling edge clock arrival time.

The Timing Report displays the:

- Data path
- Clock path
- Clock arrival time

Manage Falling Edge Clock Arrival Time Option One

For the first option to manage the falling edge clock arrival time:

1. Create two time groups:
   - One time group for *rising* edge synchronous elements
   - One time group for *falling* edge synchronous elements
2. Create an Offset In constraint for each time group.

   The second Offset In constraint has a different requirement.

The falling edge Offset In constraint requirement equals:

(original requirement) minus (one-half the Period constraint)

Therefore, if the original requirement is (3 ns with a Period of 10 ns), the falling edge Offset In constraint requirement equals (-2 ns).

This compensates for the clock arrival time associated with the falling edge synchronous elements. The negative value is legal in the constraints language..

Manage Falling Edge Clock Arrival Time Option Two

For the second option to manage the falling edge clock arrival time:

1. Create one time group and one corresponding Offset In constraint with the original constraint requirement for each clock edge.
2. Add the Falling keyword for the falling edge elements and the Rising keyword for the rising edge elements

In the following figure, the Offset requirement is 3 ns.

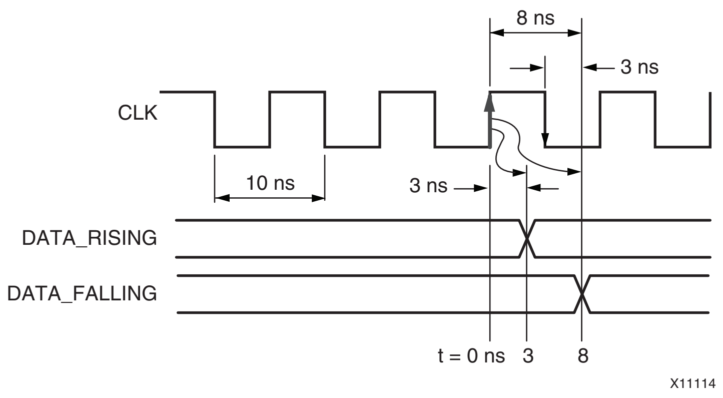


Figure 6-22: **Timing Diagram of Dual Data Rate in Offset Out Constraint**

Timing Report Example of OFFSET = OUT 3 ns After Clock Rising

```
Slack: -0.783ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:                 OutA_4 (FF)
  Destination:            OutA<4> (PAD)
  Source Clock:           clock0_ddr_bufg rising at 0.000ns
  Requirement:            3.000ns
  Data Path Delay:        3.372ns (Levels of Logic = 1)
  Clock Path Delay:       0.172ns (Levels of Logic = 3)
  Clock Uncertainty:      0.239ns
```

Timing Report Example of OFFSET = OUT 8 ns After Clock Falling

```
Slack: -0.783ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:                 OutA_4 (FF)
  Destination:            OutA<4> (PAD)
  Source Clock:           clock0_ddr_bufg falling at 0.000ns
  Requirement:            3.000ns
  Data Path Delay:        3.372ns (Levels of Logic = 1)
  Clock Path Delay:       0.172ns (Levels of Logic = 3)
  Clock Uncertainty:      0.239ns
```

Offset Out Before Constraint

The Offset Out Before constraint defines the time used by the data external to the FPGA.

Offset subtracts this time from the clock Period to determine the time available for the data to propagate from the synchronous element to the pad.

This time can be visualized as the data leaving the edge of the device before the next clock edge arrives at the edge of the device.

This **`OFFSET = OUT 2 ns BEFORE clock_pad`** constraint reads that the Data to be registered in the Downstream Device is available on the FPGA output Pad, some time period, BEFORE the clock pulse is seen by the Downstream Device. For the purposes of the Offset constraint syntax, assume no skew on CLK between the chips.

The following equation defines this relationship.

```
TQ + TData2Out + TClock <= TPeriod - Toffset_OUT_BEFORE
```

where

```
TQ = Intrinsic Flip Flop Clock to Out
TClock = Total Clock path delay to the Flip Flop
TData2Out = Total Data path delay from the Flip Flop
TPeriod = Single Cycle PERIOD Requirement
Toffset_OUT_BEFORE = Overall Clock to Out Requirement
```

The analysis of the Offset Out constraint involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) do not exceed the clock period minus the specified offset.

A Period or FREQUENCY is required for Offset Out constraints with the BEFORE keyword.

Clock Skew

Clock skew analysis is included in both a setup and hold analysis. Clock skew is calculated based upon:

(clock path delay to the *destination* synchronous element) minus (clock path delay to the *source* synchronous element)

Causes of Large Clock Skew

In most designs with a large clock skew, the skew can be attributed to one of the following:

- One or both clocks use local routing.
- One or both clocks are gated.
- DCM drives one clock, but not the other clock.

Difference Between Clock Skew and Phase

Clock skew is not the same as Phase. Phase is the difference in the clock arrival times, indicated by the source clock arrival time and the destination clock arrival time in the Timing Report. Clock arrival times are based upon the Phase keyword in the Period constraint. Clock skew is not included in the clock arrival times.

In the rising-to-rising setup or hold analysis shown in the following figure, the positive clock skew greatly increases the chance of a hold violation and helps the setup calculation.

During setup analysis, positive clock skew is truncated to zero for Virtex-4 devices and older. Virtex-5 devices and newer utilize the positive and negative clock skew in the setup analysis. Positive clock skew is used during the hold analysis for this path.

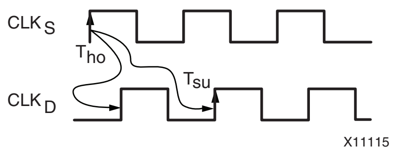


Figure 6-23: **Rising to Rising Setup/Hold Analysis**

In the rising-to-falling setup or hold analysis shown in the following figure, the positive clock skew is less, but the **`Tho`** window is smaller and minimizes the chance for a hold violation.

A two-phase clock:

- Is less likely to have a hold violation.
- Can handle more positive clock skew than a single-phase clock path.

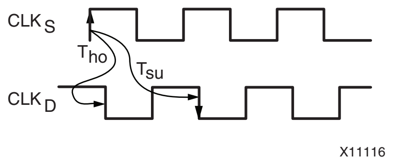


Figure 6-24: **Rising to Falling Setup/Hold Analysis**

During hold analysis, negative clock skew is truncated to zero for Virtex-4 devices and older. Virtex-5 devices and newer utilize the negative and positive clock skew in the hold analysis. Negative clock skew is used during the setup analysis for this path.

During analysis of setup and hold, the negative clock skew and positive clock skew, respectively, decrease the margin on the Period constraint requirement. See the following figure.

To determine how the timing analysis tools calculated the total clock skew for a path, use the **Analyze -> Against User Specified Paths** command in Timing Analyzer. Select the source and destination of the path in question, and analyze from the clock source to the two elements in the path.

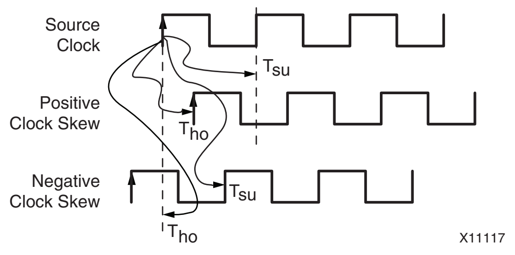


Figure 6-25: **Positive and Negative Clock Skew**

In the above figure:

- `Tsu` and `Tho` represent the active edge the setup/hold violation calculation is done one, respectively.
- The dashed lines show the positive and negative clock skew being truncated to zero for setup and hold checks, respectively.

The report displays the clock path to the source and the clock path to the destination. Review the paths to determine if the design has one of the causes of clock skew that were previously mentioned. The timing analysis tools subtract the clock path delays to produce the clock skew, as reported in the Timing Report.

The DLY file, produced by Reportgen (after PAR), can also be used to determine the values used to calculate the clock skew value that was reported.

When calculating the clock path delay, the timing analysis tool traces the clock path to a common driver. In the following figure, the common driver of the clock path is at the DCM.

If the tools can not find a common driver, the analysis starts at the clock pads. In clock path delay, the timing analysis tool traces the clock path to a common driver.

In Figure 3-16, Hold Violation (Clock Skew > Data Path):

- The clock path delay from the DCM to the destination element is (0.860 + 0.860 + 0.639) = 2.359
- The clock path delay from the DCM to the source element is (0.852 + 0.860 + 0.639) = 2.351.
- The total clock skew is 2.359 - 2.351 = 0.008 ns

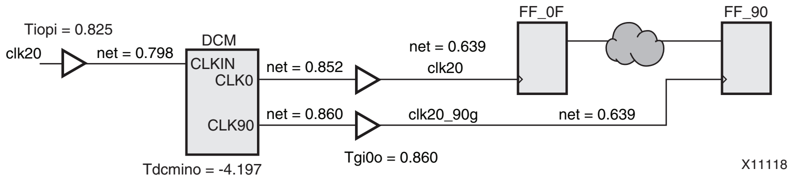


Figure 6-26: **Clock Skew Example**

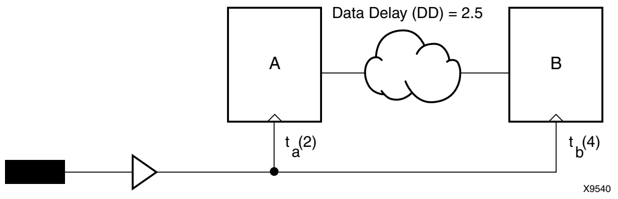


Figure 6-27: **Maximum Skew Example**

In the preceding diagram:

- For ta(2), 2 ns is the maximum delay for the Register A clock.
- For tb(4), 4 ns is the maximum delay for the Register B clock.
- Maximum Skew defines the maximum of tb minus the maximum of ta, that is, 4-2=2.

In some cases, relative minimum delays are used on a net for setup and hold timing analysis. When the Maximum Skew constraint is applied to network resources which use relative minimum delays, the Maximum Skew constraint takes relative minimum delays into account in the calculation of skew.

Overusing the Maximum Skew constraint, or too strict of a requirement (value), can cause long PAR runtimes.

Clock Uncertainty

In addition to the Clock Skew affecting the margin on the Period constraint requirement, clock uncertainty also affects it. Clock uncertainty increases timing accuracy by accounting for system, board level, and DCM clock jitter.

The System Jitter constraint and the Input Jitter keyword on the Period constraint tell the timing analysis tools that the design has external jitter affecting the timing of this design. See the following figure.

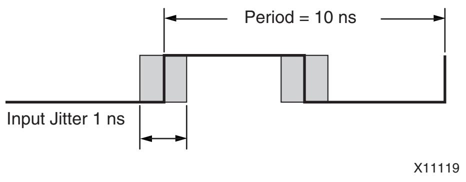


Figure 6-28: **Input Jitter on Clock Signal**

The following are also included in the clock uncertainty during the analysis for Virtex-4 devices and newer:

- DCM/PLL/MMCM Jitter
- DCM/PLL/MMCM Phase Error
- DCM/PLL/MMCM Duty Cycle Distortion (DCD) or Jitter

The individual components that make up clock uncertainty are also reported. The timing analysis tools calculate the clock uncertainty for the source and destination of a data path and combine them together to form the total clock uncertainty.

DCM Clock Uncertainty Equation

Following is the equation for DCM Clock Uncertainty:

```
Clock Uncertainty = [√(INPUT_JITTER²  + SYSTEM_JITTER²) +
DCM_Discrete_Jitter]/2 + DCM_Phase_Error
```

DCM Discrete Jitter and DCM Phase Error are provided in the speed files for Virtex-4 devices and newer. However, DCM Discrete Jitter and DCM Phase Error are not available in **speedprint**.

Clock Uncertainty Examples

- INPUT_JITTER: $200ps^2 = 40000ps$
- SYSTEM_JITTER: $150ps^2 = 22500ps$
- DCM/PLL/MMCM Discrete Jitter: 120ps
- DCM/PLL/MMCM Phase Error: 0ps
- Clock Uncertainty: 185ps

Period Constraint with Input Jitter Example

Following is an example of a Period constraint with the Input Jitter keyword:

```
TIMESPEC "TS_Clk0" = PERIOD "clk0" 4 ns HIGH 60% INPUT_JITTER 200 ps PRIORITY 1;
```

The System Jitter constraint:

- Defines jitter that impacts the system.
- Can represent the jitter from:
    - Power noise
    - Board noise
    - Any extra jitter of the overall system

The System Jitter value can depend upon the design condition, such as:

- The number of synchronous elements changing at the same time.
- The number of inputs and outputs changing at the same time.

The System Jitter value can be based upon the difference between (1) the input clock edge noise (or jitter); and (2) the power noise. This difference can be measured on the board by the differences between (1) the clock edges; and (2) the power plane and ground plane movements.

A user-specified System Jitter constraint overrides the default System Jitter value (if any) for a given device family.

Not all device families have a default System Jitter value. In that case, the user must specify a value.

Xilinx recommends a System Jitter value of 300ps. This value:

- Applies to all clocks in the design.
- Is combined with the Input Jitter value for a given clocking network topology.

System Jitter Constraint in the UCF Example

The following is an example of the System Jitter constraint in the UCF:

```
SYSTEM_JITTER = 300 ps;
```

Clock jitter consists of both random and discrete jitter components. Because the Input Jitter and System Jitter are random jitter sources, and typically follow a Gaussian distribution, the combination of the two is added in a quadratic manner to represent the worst case combination.

Because the DCM/PLL/MMCM Jitter is a discrete jitter value, it is added directly to the clock uncertainty.

In the analysis of clock uncertainty all jitter components, both random and discrete, are specified as peak-peak values. Peak-peak values represent the total +/- range by which the arrival time of a clock signal varies in the presence of jitter.

In a worst case analysis, only the delay variation that causes a decrease in timing slack is used. For this reason, only the peak jitter value, or one-half the peak-to-peak value, is used for each setup and hold timing check.

The phase error component of clock uncertainty is a value representing the phase variation between two clock signals. Because this value is discrete, and represents the actual phase difference between the DCM/PLL/MMCM clocks, it is added directly to the clock uncertainty value.

PLL/MMCM Clock Uncertainty Equation

```
Clock Uncertainty = [√(INPUT_JITTER²  + SYSTEM_JITTER² + PLL/
MMCM_Discrete_Jitter²)]/2 + PLL/MMCM Phase_Error
```

PLL/MMCM Discrete Jitter and PLL/MMCM Phase Error are provided in the speed files for Virtex-5 devices.

In the analysis of clock uncertainty all jitter components, both random and discrete, are specified as peak-peak values. Peak-peak values represent the total +/- range by which the arrival time of a clock signal varies in the presence of jitter. In a worst case analysis, only the delay variation that causes a decrease in timing slack is used.

Only the peak jitter value, or one-half the peak-to-peak value, is used for each setup and hold timing check.

The phase error component of clock uncertainty is a value representing the phase variation between two clock signals. Because this value is discrete, and represents the actual phase difference between the PLL/MMCM clocks, it is added directly to the clock uncertainty value.

Chapter 7

Achieving Timing Closure

Timing closure is a major design challenge.

- The high performance requirements of many designs, and the size of the target devices, often make it difficult to achieve timing closure.
- Designs that formerly fit on ASIC devices, or that ran at high clock frequencies on those devices, are now finding their way onto Xilinx® FPGA devices.

You must have a proven methodology for achieving your performance objectives. This chapter addresses timing closure issues by providing a recommended methodology with examples and use cases.

The guidelines in this chapter are a road map for improving performance and meeting your timing objectives.

When Timing Closure Is Achieved

Timing closure is achieved when all timing constraints for a design are met under all legal operating conditions:

- Process
- Voltage
- Temperature

Timing Score

Timing closure is achieved when the design is fully constrained and the timing score is zero. The timing score:

- Is the total value representing the timing analysis for all constraints, and the amount by which the constraints are failing
- Is the *sum* in picoseconds of all timing constraints that have not been met
- Shows the total amount of error (in picoseconds) for all timing constraints in the design
- Can be viewed in the PAR Report at each phase of the router algorithm.

PAR Report Example One

Phase 1:	373040 unrouted;	REAL time: 2 mins 2 secs
Phase 2:	324361 unrouted;	REAL time: 2 mins 24 secs
Phase 3:	133339 unrouted;	REAL time: 6 mins 1 secs
Phase 4:	134608 unrouted;	(Setup: 23596, Hold: 3309336, Component Switching Limit: 0)
Phase 5:	0 unrouted;	(Setup: 46800, Hold: 319725, Component Switching Limit: 0)
Phase 6:	0 unrouted;	(Setup: 29212, Hold: 319991, Component Switching Limit: 0)
Phase 7:	0 unrouted;	(Setup: 29232, Hold: 319991, Component Switching Limit: 0)
Phase 8:	0 unrouted;	(Setup: 29232, Hold: 319991, Component Switching Limit: 0)
Phase 9:	0 unrouted;	(Setup: 27588, Hold: 320002, Component Switching Limit: 0)

Three timing score values are reported:

- Setup
- Hold
- Component switching limits

Each timing score value is analyzed in more detail later in this chapter.

The final timing score is displayed in the PAR Report and the TRCE Report.

PAR Report Example Two

```
Phase 1: 235879 unrouted;      REAL time: 54 secs
Phase 2: 206616 unrouted;      REAL time: 59 secs
Phase 3: 76322 unrouted;      REAL time: 3 mins 11 secs
Phase 4: 76327 unrouted; (Setup:2126, Hold:23834, Component Switching Limit:0)
REAL time: 3 mins 44 secs   Intermediate status: 4 unrouted;       REAL time: 33
mins 16 secs Updating file: crypto_subsystem_wrap.ncd with current fully routed
design.
Phase 5: 0 unrouted; (Setup:177520, Hold:17912, Component Switching Limit:0)
REAL time: 33 mins 47 secs   Intermediate status: 917 unrouted;       REAL time:
1 hrs 4 mins 11 secs
Phase 6: 0 unrouted; (Setup:9720, Hold:17991, Component Switching Limit:0)
REAL time: 1 hrs 5 mins 17 secs
Phase 7: 0 unrouted; (Setup:9720, Hold:17991, Component Switching Limit:0)
REAL time: 1 hrs 5 mins 17 secs
Phase 8: 0 unrouted; (Setup:9720, Hold:17991, Component Switching Limit:0)
REAL time: 1 hrs 5 mins 17 secs
Phase 9: 0 unrouted; (Setup:4053, Hold:0, Component Switching Limit:0)     REAL
time: 1 hrs 5 mins 37 secs
Timing Score: 2124 (Setup: 2124, Hold: 0 Component Switching Limit: 0)
```

TRCE Report Example

```
Timing Summary:
---------------
Timing errors: 119 Score: 2124 (Setup[/Max: 2124, Hold: 0)
Constraints cover 23109382 paths, 24 nets, and 339654 connections
```

Prerequisites to Achieving Timing Closure

To achieve timing closure, you must understand the following *before* starting your design:

- The performance requirements of the system
- The features of the target device

Knowing these requirements and features enables you to use the correct coding methods to achieve optimal performance.

Device Requirements

The requirements of the target device depend on:

- The system, and
- The upstream and downstream devices.

Once you know the interfaces to the target device, you can outline the internal requirements.

How to meet these requirements depends on the target device and its available features. You must understand:

- Device clocking structure
- RAM and DSP blocks
- Any hard macros

For more information on each family, see the device data sheet cited in Appendix A, Additional Resources.

Timing Closure Flowchart

The following figure outlines the steps to follow in order to achieve timing closure. Each step is addressed individually in the remainder of this chapter.

Step 8: Run TRCE and Analyze Timing Results and Report is the primary step. Many use cases and scenarios are presented with proposed debugging steps and resolutions.

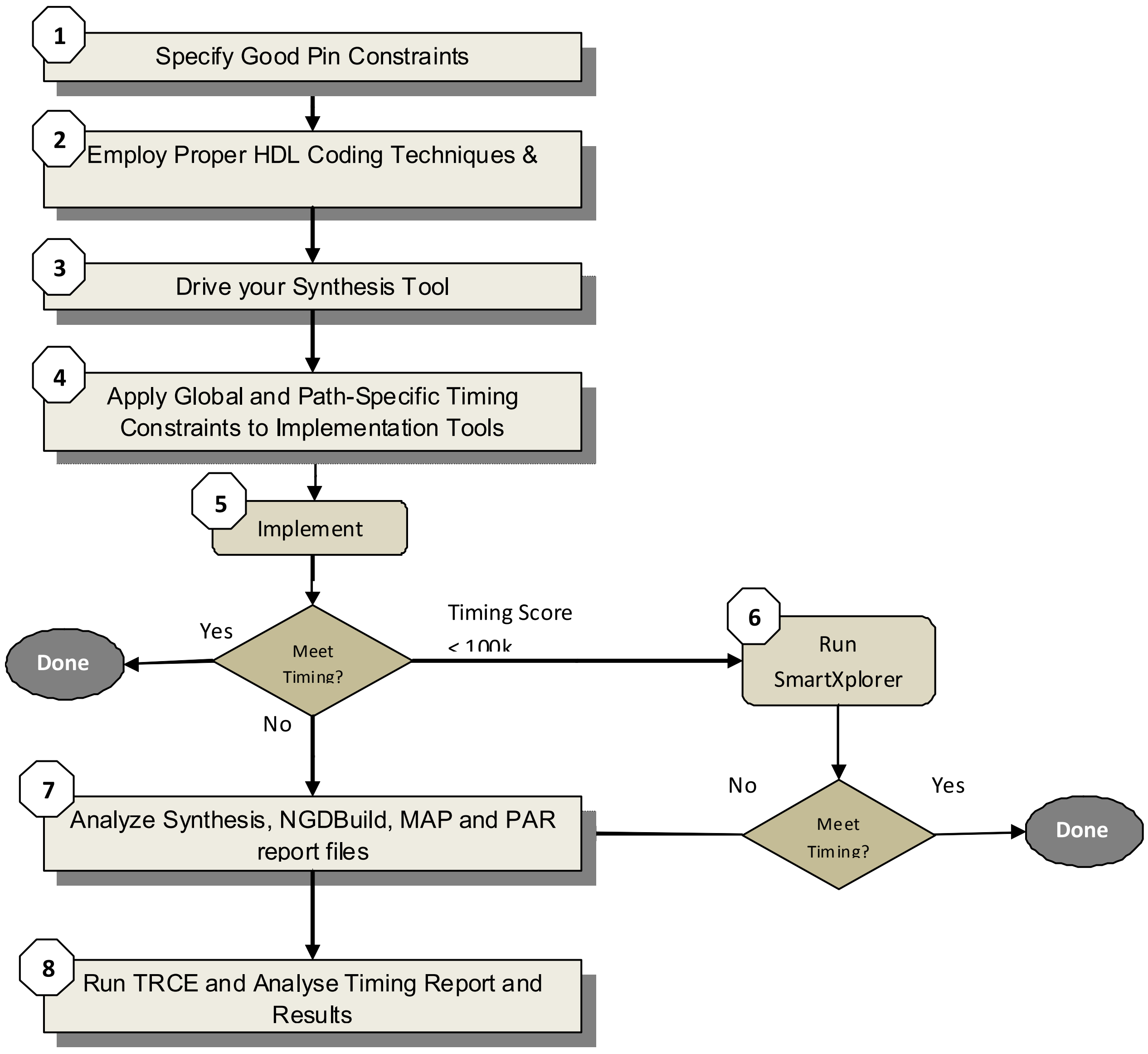

1
Specify Good Pin Constraints
2
Employ Proper HDL Coding Techniques &
3
Drive your Synthesis Tool
4
Apply Global and Path-Specific Timing Constraints to Implementation Tools
5
Implement
Meet Timing?
Yes
Done
Timing Score < 100k
6
Run SmartXplorer
No
7
Analyze Synthesis, NGDBuild, MAP and PAR report files
Meet Timing?
No
Yes
Done
8
Run TRCE and Analyse Timing Report and Results

Figure 7-1: **Timing Closure Flowchart**

Steps to Achieving Timing Closure

- Step 1: Specify Good Pin Constraints
- Step 2: Use Proper Coding Techniques and Architectural Resources
- Step 3: Drive the Synthesis Tool
- Step 4: Apply Global and Path Type Timing Constraints
- Step 5: Run Implementation
- Step 6: Run SmartXplorer
- Step 7: Review Reports
- Step 8: Run TRCE and Analyze Timing Results and Report

Step 1: Specify Good Pin Constraints

Pin constraints are often required early in the design cycle to allow board development to begin.

Create pin constraints that take advantage of:

- FPGA architecture
- Design flow
- Board requirements.

To create these constraints, use your knowledge of:

- The FPGA fabric
- Design input to outputs
- Data flow through your design
- Your design in general

Designs with Large Components

Designs with large components, such as block RAM components, drive the data flow through the device. Having a good knowledge of the number of clocks, and how they relate to each other, also impacts pin placement.

The clock structure can dictate overall design performance. For that reason, it is of utmost importance. The pin placement can also be driven by the interfaces to the upstream and downstream devices, such as memory interface locations on the board.

Pin Location

Pay close attention to pin location. Evaluate I/O location constraints in the PlanAhead™ design analysis tool to ensure that these constraints are not forcing critical logic to span the device. If so, you may need to insert pipeline stages.

Partial Reconfiguration, Partitioning and Floorplanning

As devices and designs increase in size, partial reconfiguration, partitioning and floorplanning have become more important. Good pin location allows the design to be well floorplanned and to use the device structure most efficiently.

Pin Placement Strategy

Use the PlanAhead tool for your pin placement strategy. Before using this tool, see the tutorials on the Xilinx support web site.

You can generate I/O package pin assignments:

- Manually on a pin-by-pin drag and drop basis
- By semi-automatically dragging and dropping groups of ports
- With a fully automatic pin placement algorithm

This process can begin with:

- A synthesized EDIF netlist
- An un-synthesized HDL netlist
- A comma separated value (CSV) file
- A completely blank project in which the design ports are created inside the tool for export

Pin placement can affect the timing of the final design. It is easier to write code that meets timing for pins in a single bank, or for pins in adjacent banks, than it is to write code for pins in banks on opposite sides of the chip.

Table 7-1: **Ease of Writing Code That Meets Timing**

Easier	Harder
• Pins in a single bank • Pins in adjacent banks	• Pins in banks on opposite sides of the chip

Embedded Elements

When pin planning, consider embedded elements such as which RTL will communicate with the following components:

- MGT
- block Ram
- DSP

Use the following information when writing the RTL code:

- Which RTL hierarchy will communicate with these components
- Which hierarchy will be pulled apart by a given pinout

The pin assignment suggestions in the following steps can help increase productivity for optimal I/O placement.

Step 2: Use Proper Coding Techniques and Architectural Resources

Each Xilinx device family has specific features and resources, although many are common across platforms. The design must use these resources optimally and efficiently.

For more information on individual devices, see the device user guide cited in Appendix A, Additional Resources.

Device Architectural Resources

Following are examples of available device architectural resources:

- Shift Register LUT (SRL16/ SRLC16)
- F5, F6, F7, and F8 multiplexers
- Carry logic
- Multipliers (DSP48)
- Global clock buffers (such as BUFG, BUFGCE, BUFGMUX, BUFGDLL, and BUFPLL)
- SelectIO™ standard (single-ended, differential)
- I/O registers (SDR, DDR)
- Memories (BRAM, DRAM)
- DCM, PMCD, PLL, MMCM
- Local clock buffers (BUFIO, BUFR)
- PPCs, MicroBlaze
- MGTs

You must understand the particular device you are targeting and the specific resources available within that device. Using these resources necessarily impacts the performance of the design and tools.

Coding Guidelines

Xilinx recommends that you:

- Implement synchronous design techniques
- Use Xilinx specific coding
- Use cores

The *XST User Guide for Virtex®-6, Spartan®-6, and 7 Series Devices (UG687)* contains many example of how to code efficiently to target available device features. For a link to this guide, see Appendix A, Additional Resources.

Follow these coding guidelines to ensure an optimal netlist:

- Avoid high level loop constructs.
- Use **case** statements for large decoding.
- Avoid nested **if-then-else** statements.
- Do not create internally generated clocks except though DCM or PLL.
- Minimize the number of clocks in the design.
- Make sure that internally created resets are synchronous.

- Use only one edge of the clock.
- Use edge-triggered flip-flops (avoid latches).
- Cross-clock domains via synchronization circuits.
- Register top-level inputs and outputs for fastest performance and increased pin-locking capability.
- Use hierarchy to separate functionality and clock domains.
- Employ pipelining for critical paths.
- Comment your code to highlight Multi-Cycle paths and critical paths.

Clocking Guidelines

The clocking structure varies across the range of devices, which is highlighted in the Spartan®-6 family. To achieve timing closure, use this clocking structure to take full advantage of all features.

Xilinx recommends that you follow these clocking guidelines.

Minimize Clocking Components

Use a minimal number of clocking components.

Evaluate Connectivity of Clocking Components

Evaluate connectivity of all clocking components in the PlanAhead tool to ensure that there are no duplicate structures that may cause unnecessary use of clock components (for example, one BUFG driving another BUFG).

Do Not Use CLOCK_DEDICATED_ROUTE

Do not use the CLOCK_DEDICATED_ROUTE constraint in a production design. Use CLOCK_DEDICATED_ROUTE only as a temporary workaround to a clock failure in MAP in order to produce an NCD file to debug the design in FPGA Editor. The CLOCK_DEDICATED_ROUTE constraint applies to the INSTANCE PIN or NET.

For more information, see Xilinx Answer Record 30355.

Do Not Use Gated Clocks

Do not use gated clocks.

Resets and Clock Enables Guidelines

Xilinx recommends that you follow these guidelines for resets and clock enables.

Avoid Asynchronous Resets

Avoid asynchronous resets. Asynchronous resets:

- Prevent control set reduction in synthesis
- Prevent certain power optimizations from occurring
- Prevent logic optimization into SR path for improved timing
- Are more difficult to time

Minimize Resets and Clock Enables

Minimize the use of resets and clock enables when possible.

Consequences of a Large Number of Resets and Clock Enables

A large number of resets and clock enables results in a large number of control sets. A large number of control sets in a design in which each control set has a small number of loads impacts the packing of registers into a slice. This can lead to fitting and timing issues in all device families.

Xilinx recommends combining or simplifying the resets and clock enable signals. These signals share routing resources, and can prevent the placer from using locations that might help the performance of the design and timing paths.

Use Active High Resets for Spartan-6 Devices

For Spartan®-6 devices, use active-High resets when possible. Because there is no local inversion in the slice for resets in Spartan-6 devices, the inversion must be done in a LUT.

For designs in which hierarchy is maintained in synthesis, or in which partitions are used, this can lead to multiple LUTs. This can have implications for timing due to an extra LUT for the inversion.

Run MAP with **-detail** Switch

Run MAP with the **-detail** switch to get a complete listing of control sets and the loading on each set in the MAP report (`*mrp`). Verify that a large number of control sets are not being caused by fanout optimization of a high fanout reset/ce.

When you generate a detailed list of control sets in your MAP report, look for reset/ce nets with very similar names but with `rep` or `fast` appended to the name. This can indicate that this net was replicated. You can also verify this in the Synthesis Report.

Minimize Resets

Minimize resets. Resets may cause suboptimal mapping of shift registers into SRLs.

Block RAM and DSP Guidelines

Xilinx recommends that you follow these block RAM and DSP guidelines.

Use Dedicated Registers

Verify that all block RAM and DSP48 blocks use dedicated registers when possible to minimize setup and **`clk2out`** time. Use a PlanAhead tool DRC check to identify this situation.

Use the PlanAhead tool schematic view to identify why the registers are not being merged with the block RAM or DSP components.

Infer Block RAM and DSP

Infer block RAM and DSP when possible to provide flexibility and optimal usage.

Follow XST Coding Styles

Follow XST coding styles to ensure proper inferencing.

For more information, see the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*, cited in Appendix A, Additional Resources.

Regenerate or Resynthesize All Block RAM Components

Regenerate or resynthesize all block RAM components, or both, using the latest version of ISE® Design Suite. Block RAM specifications continue to change through releases. The best way to ensure that you have the latest recommended configuration of block RAM components is to use the latest core or synthesis tool result.

Examine MAP File

Examine the MAP (`.mrp`) file for any errors or warnings concerning block RAM components.

Verify Block RAM Behavior

To verify proper block RAM behavior, run extensive functional and timing simulations.

- When using the BRAMB8BWER in SDP mode (256x36), if using it synchronously with the same clock connected to the read/write port, set WRITE_MODE on both ports to READ_FIRST.
- Starting after Release 12.2, when using the RAM in this mode with different clocks on read/write ports, ISE® Design Suite allows the use of WRITE_FIRST mode. This avoids address overlap, and is the preferred setting.
- For Release 12.2 and before, Xilinx recommends using a RAMB16BWER (512x36) mode in WRITE_FIRST configuration to avoid address overlap.
- For Release 11.5 and after, when using the BRAMB8BWER in SDP mode (256x36), where one port is 36-bits and the other is 18-bits or less, that mode is no longer allowed. For this mode, Xilinx recommends:
  - Use a RAMB16BWER (easier but uses more memory space than potentially needed), or
  - Construct the proper logic to allow the block RAM to be configured with 36-bits on both ports (the only supported widths for RAMB8 in SDP mode).
- In general, Xilinx recommends registering the input and outputs of the design and of any given module.
- Determine if Distributed or block RAM memory is ideal.
  - Smaller memories offer higher performance with distributed RAM.
  - Larger memory arrays are better in block RAM.

Step 3: Drive the Synthesis Tool

It is important to drive the synthesis tools and apply period and input and output constraints to drive optimization results from synthesis. Multi-Cycle and false paths can also be applied.

The synthesis tools work on paths using the logic delay as guidance. Without any constraints, the tools treat the longest path (most logic delay) as the most critical.

For instance, in a two clock system, **`clka`** with 10 ns of logic delays and **`clkb`** with 20 ns of logic delays, **`clkb`** is seen as the critical path. Because the tools have no knowledge of clock requirement without constraints guiding the synthesis tools, **`clkb`** may not actually be the most critical path.

Apply a Period constraint to the tool specifying that **`clka`** has a requirement of 5 ns and **`clkb`** has a requirement of 25 ns. The tools now consider **`clka`** as the critical path.

For more information on constraining synthesis, see Chapter 6, Timing Analysis.

Pipelining the Design

Pipelining the design:

- Increases the efficiency of the synthesis tool.
- Is optimal for interface bandwidth
- Is not ideal for latency.

While latency can be important, it is usually the latency in a different order of magnitude than the one caused by pipelining.

Because FPGA devices have many registers, re-timing and the innovative use of arithmetic functions can yield greatly enhanced performance. If you must balance the latency among different paths in the system, use SRLs to compensate efficiently for delay differences. Using SRLs can negatively affect the control sets and packing of other logic around them.

Synthesis Options That Impact Timing

The following synthesis options impact the timing of a design.

For more information on each option, see the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*, cited in Appendix A, Additional Resources.

- Keep Hierarchy
- LUT Combining
- RAM Extraction and ROM Extraction
- Use DSP Block
- Global Max Fanout
- Shift Register Extraction and Shift Register Minimum Size
- Register Balancing
- Netlist Hierarchy
- Read Cores
- Asynchronous to Synchronous
- Resource Sharing
- Equivalent Register Removal

- Pack I/O Registers Into IOB Components
- State Machine Encoding
- Shift Register Inferencing (Spartan-6 and Virtex-6 Devices)

Keep Hierarchy

Maintaining hierarchy:

- Enables easier debugging in static timing analysis
- Improves your opportunities to floorplan and to implement incremental or modular design techniques

However, maintaining hierarchy can have a negative impact on the results. When hierarchy is maintained, the synthesis tool is limited to optimizing within the boundary of the hierarchy. For some designs that do not have a well-defined hierarchy, it is necessary to allow the tools to optimize across the hierarchy.

Check the Synthesis Report to see if the global Keep Hierarchy (KEEP_HIERARCHY) constraint is set to **soft** or **yes,** or if Keep Hierarchy (KEEP_HIERARCHY) or Keep (KEEP) constraints have been set on specific instances. If so, run with these constraints removed. These constraints could be impacting optimizations on critical paths if the constraints are not applied at the proper boundaries.

LUT Combining

The LUT Combining (LC) constraint maps two small LUTs into a single LUT, taking advantage of the dual outputs on the LUT. LUT Combining can cause problems with placement resulting in timing issues. When LUTs are combined, the placer tool is restricted with a single LUT trying to satisfy multiple timing paths. While this can lead to timing issues, this option is useful when trying to reduce the design utilization.

LUT Combining is set to **auto** by default in XST. Review the MAP Report to see if this option is having a large impact by the number of LUTs using both the O5 and O6 outputs.

Consider disabling this option in XST to improve performance.

LUT Combining can provide an area savings. Consider disabling LUT Combining in XST, and enabling it MAP to ensure the most accurate view of timing.

RAM Extraction and ROM Extraction

To optimally infer block RAM or distributed RAM components:

- Follow the coding techniques outlined in the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*, cited in Appendix A, Additional Resources.
- Use the extraction constraints.

Use the pipelining registers available in the block RAM resource in the devices for optimal timing performance. Find a good balance between block RAM and distributed RAM components.

Use DSP Block

As with RAM extraction, use the USE_DSP48 (Use DSP48) constraint to instruct the tools to infer DSP blocks. For optimal timing performance, use the pipelining registers in the device DSP resource.

This constraint has different names in different devices.

Table 7-2: **Use DSP Block Names**

Devices	Name
• Virtex-4	Use DSP48
• Virtex-5 • Spartan®-3A DSP	Use DSP Block

Global Max Fanout

Reducing the fanout of control signals:

- Can greatly improve the ability to meet timing.
- Is not necessary for global logic.
- Can allow the tools to place the design more efficiently.
- Increases the number of logic levels in the design.

Xilinx recommends using only enough registers to improve the placement and performance.

- Do not use as a global setting.
- Attach it to individual paths.

The synthesis tools replicate with no knowledge of the destination locations. Because this also increases the control sets, use it sparingly. The XST report includes a control signal report that can help understand the nets with high fanout.

If the design has area problems, and the value is very low, increase the value to see the impact on area.

If there are still high fanout nets after increasing the value:

- Determine if any of the nets are timing critical.
- Apply a Max Fanout (MAX_FANOUT) attribute specifically on the net.

Try reducing this value:

- If the global Max Fanout was not changed, and
- There are many high fanout nets impacting performance.

Shift Register Extraction and Shift Register Minimum Size

Use caution when inserting pipelining in the design. The tools may infer an SRL, thus removing the pipelining. SRL inference can be controlled with this constraint and set the minimum shift register size before SRL inference takes place.

Register Balancing

The Register Balancing (REGISTER_BALANCING) constraint enables flip-flop retiming. The main goal of register balancing is to move flip-flops and latches across logic to increase clock frequency.

Explore these options to see if they provide a performance advantage. Combining these options can lead to increased register usage and potentially more LUT usage due to SRL inferencing. Therefore, if area limited, this may hurt more than it helps.

Netlist Hierarchy

The Netlist Hierarchy (**-netlist_hierarchy**) constraint:

- Controls the form in which the final NGC netlist is generated.
- Allows you to write the hierarchical netlist even if the optimization was done on a partially or fully flattened design.
- Is set to **as_optimized** by default.

  In many designs, Netlist Hierarchy is set to **rebuilt** to make it easier for floorplanning. Sometimes, however, this can cause worse timing. Explore this option carefully to see if it impacts timing.

Read Cores

Reading in cores during synthesis insures that XST reads in any IP cores generated by the CORE Generator™ tool. By reading in the cores, XST can better optimize the logic connected to these cores.

Asynchronous to Synchronous

If the design has asynchronous resets, use the Asynchronous to Synchronous (ASYNC_TO_SYNC) switch to convert the asynchronous resets to synchronous resets. Doing so can impact performance, area, and power. Because this can impact functionality, verify that the design is functioning correctly after synthesis.

Resource Sharing

Synthesis tools use resource sharing to decrease circuit area, usually resulting in lower performance. Resource Sharing (RESOURCE_SHARING):

- Minimizes the number of arithmetic operators, resulting in reduced device utilization
- Works with adders, subtractors, adders/subtractors, and multipliers
- Is on by default

An HDL Advisor message informs you when resource sharing has taken place.

Consider disabling resource sharing if the design is unable to meet timing. If the design has a limited number of LUTs, consider moving some of these arithmetic operators into DSP48s if available.

Equivalent Register Removal

The Equivalent Register Removal (EQUIVALENT_REGISTER_REMOVAL) constraint:

- Removes equivalent registers if they are described at the Register Transfer Level (RTL)
- Does not remove instantiated flops
- Is on by default

Consider disabling Equivalent Register Removal if:

- The design is trying to describe equivalent registers to minimize fanout, or
- The design is trying to keep certain blocks isolated.

Equivalent Register Removal can be disabled:

- Globally, or
- On specific instances.

It is not always necessary to remove registers, because most designs use more LUTs than registers, making registers abundant. Review the Synthesis Report to see if registers have been removed due to Equivalent Register Removal.

Pack I/O Registers Into IOB Components

The decision to move flip-flops into and out of IOB components can also be made by the MAP process during implementation. A constraint can be applied during synthesis.

Xilinx recommends using IOB flip-flops to improve interface timing. Using the Offset In or Offset Out constraints drives the placement of the flip-flops into the IOB sites.

State Machine Encoding

Use One-Hot State Encoding when implementing Finite State Machine (FSM) components. By using One-Hot State Encoding in Xilinx FPGA devices, the next-state decoding logic can be simplified to logic equations with four inputs or fewer. This can fit into a single LUT, and maximizes the performance of the state machine.

Many synthesis tools choose One-Hot State Encoding for state machines when targeting a Xilinx FPGA device.

Shift Register Inferencing (Spartan-6 and Virtex-6 Devices)

The minimum shift register size for inferring LUTs as shift registers (SRLs) is two for XST. For many designs, this can lead to a large increase in LUTs, which may negatively impact fitting and performance. Use the Shift Register Minimum Size (SHREG_MIN_SIZE) option to globally control the default shift register size that XST uses.

Use the Shift Register Extraction (SHREG_EXTRACT) constraint to completely disable the inference of SRLs. This can be useful when a design is becoming very limited on LUTs and particularly SLICEMs.

The Shift Register Extraction (SHREG_EXTRACT) constraint can be applied globally or to a specific instance.

For more information, see the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687),* cited in Appendix A, Additional Resources.

Step 4: Apply Global and Path Type Timing Constraints

The implementation tools do not attempt to place and route the design to obtain the best speed. Instead, they try to meet the performance expectations communicated by the timing constraints. Timing constraints improve the design performance by helping place logic closer together resulting in shorter routing resources used. However, the tools do not optimize the design or change the netlist in any way. This can only improve placement and routing.

Use timing constraints to define performance objectives. Applying constraints that are tighter than necessary increases compile time. Unrealistic constraints cause the implementation tools to stop with non-optimal results.

> ***Caution!*** Do not use tighter constraints than required. Tighter constraints cause the tool to work harder than necessary to meet timing, and may give less than optimal performance.

Timing Ignore constraints and Multi-Cycle constraints allow the tools to relax on certain paths, and concentrate on meeting timing on the most critical paths. Over-constraining a design is considered later in the chapter.

The FPGA device requirements depend on the downstream and upstream devices which will dictate the I/O requirements for the FPGA. All of the clocks in the design should be constrained.

Basic Timing Model

The following figure shows a basic timing model which highlights the impact of these devices on the FPGA timing.

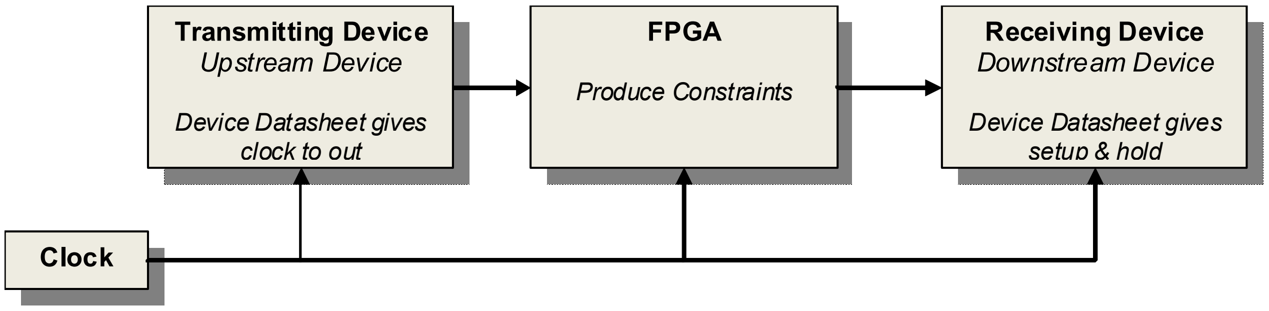


Figure 7-2: **Basic Timing Model**

Isolate Global Constraints

When applying design constraints, first isolate the global constraints. These are the first to be constrained. Run the tools with global constraints only, then apply path specific constraints as necessary. Cover all paths in the design by constraints.

Global Timing Constraints

From the basic timing model isolate the inputs, outputs, and clocked logic within the design. Once you understand these paths, you can proceed with the basic global timing constraints that will apply to your design.

For more information, see Chapter 2, Timing Constraint Methodology.

Example Scenarios

The following example scenarios show how the above parameters correlate while applying constraints.

FPGA Interfaced with a SDRAM (Example Scenario One)

In Example Scenario One, the FPGA device is being interfaced with a SDRAM on the board. The requirements of the SDRAM are:

- Minimum setup time: 2ns
- Maximum Clock to Out: 6ns

You must include the board trace delays. In this case, they are:

- Setup path = 500ps
- Clock to Out path = 300ps

In this case, the SDRAM is the downstream device as well as the upstream device.

- OFFSET OUT is 2.5ns
  - 2ns is the minimum setup time of SDRAM
  - 0.5ns is because of board delay
- OFFSET IN is 6.3ns
  - 6ns is Clock to out for SDRAM
  - 0.3ns is because of board delay

This example does not contain System Clock Frequency.

Three Devices Running at 100MHz (Example Scenario Two)

In this example, there are three devices running at 100MHz. Assuming the three devices to be simple synchronous elements. From one element to another, the delay should be 10ns (synchronous elements to synchronous elements).

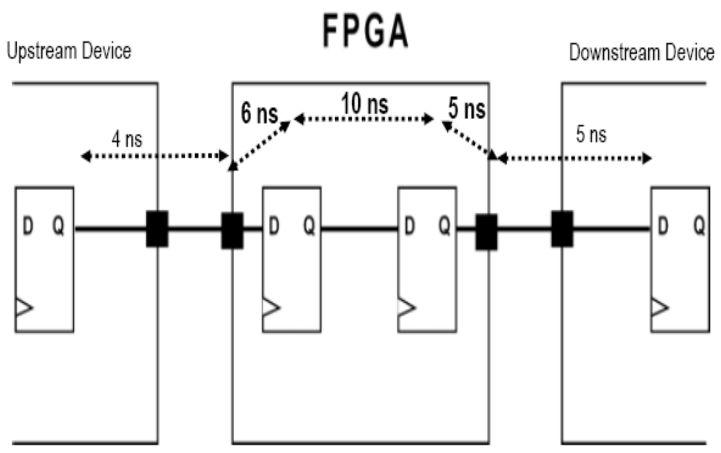


Figure 7-3: **Example Scenario Two**

Within the FPGA device, the time taken for the data path in between synchronous elements is 10ns.

- PAD to Synchronous elements - 6ns. (Requirement: 10ns - 4ns).

  It takes a delay of 4ns from the synchronous element of the upstream device to the Input PAD. The time requirement from Input PAD to synchronous element of FPGA is covered by the Offset In constraint.

- Synchronous elements to PAD - 5ns (Requirement:- 10ns - 5ns).

  It takes a delay of 5ns from out Pad to synchronous element of downstream device. This time requirement from Synchronous element to PAD of FPGA is covered by the Offset Out constraint.

Over-Constraining a Design

Use SYSTEM_JITTER to over-constrain a design. Do not increase clock frequency to over-constrain. Changing the clock frequency changes the relationship between the clock edges. To over-constrain a specific clock, increase the Input Jitter on that specific clock.

For more information on applying the Jitter constraint, see Clock Uncertainty in Chapter 6.

Step 5: Run Implementation

Now that the design uses the available device features, and is correctly constrained, it is necessary to run the design through the tools to determine the timing performance.

Xilinx recommends that you start with the default options to get a first impression of the performance.

The following implementation options have the greatest impact on timing.

- Physical Synthesis Options
- Ignore Keep Hierarchy
- Multiple Cost Tables
- Area Based Options

For more information on specific implementation options, see the *Command Line Tools User Guide (UG628)*, cited in Appendix A, Additional Resources.

Physical Synthesis Options

SmartXplorer explores all physical synthesis options in MAP such as:

- Global optimization

  Note: Does not apply to 7 series devices.
- Register duplication
- Logic optimization
- Retiming

Global optimization set to **`speed`** can often impact timing significantly, especially on Synplify PRO generated netlists.

Ignore Keep Hierarchy

If you must maintain hierarchy in synthesis and through the implementation flow for debug, run MAP with **-ignore_keep_hierarchy** to evaluate its impact on performance and area.

Multiple Cost Tables

When timing is close, cost tables in MAP can often vary the placement enough to obtain timing closure. The first ten cost tables provide the most variability.

This is an effective way to explore optimal block RAM and DSP48 placement. Good placement reduces compile runtime.

Area Based Options

The following area based options may also affect timing:

- LUT Combining
- Global Optimization Area

LUT Combining

The LUT Combining (LC) constraint has two values:

- auto
- area

Both values typically degrade performance, but **`auto`** is less severe.

For some LUT- limited designs, **`auto`** can actually increase performance by:

- Reducing the overall LUT count, and
- Giving the placer tool more flexibility.

Global Optimization Area

If LUT Combining (LC) does not provide enough area savings, and a design is unable to fit in the target device, try **-global_opt area**. This typically has a much larger impact on performance than LUT Combining.

Review the following report files to check for warnings that may highlight issues with the design:

- Synthesis Report
- NGDBuild Report
- MAP Report
- PAR Report

For more information, see Step 7: Review Reports.

Timing Score Options

When you are satisfied with the results, check the timing score in the PAR Report. There may be a timing score of 0 highlighting that all the constraints are met:

```
Timing Score: 0 (Setup: 0, Hold: 0, Component Switching Limit: 0)
```

Checking Timing Results

Even if the timing score is 0, you must still check the specific timing results in Timing Analyzer to ensure that all constraints have been analyzed as expected.

The TSI Report highlights the interactions among all constraints. If there are multiple clocks and propagated constraints, the interaction among them are highlighted.

If you have applied Timing Ignore or Multi-Cycle constraints, the TSI Report displays:

- The number of paths that these constraints cover.
- The specific global constraints that these constraints relax.

For more information, see Step 8: Run TRCE and Analyze Timing Results and Report.

Timing Score Between 0 and 100,000

If there is a timing score of between 0 and 100,000, Xilinx recommends running SmartXplorer to check which tool options have a positive or negative impact on timing.

For more information, see Step 6: Run SmartXplorer.

Timing Score Greater Than 100,000

Although SmartXplorer generally does not resolve timing issues when the timing score is greater than 100,000, it may nonetheless be useful to run SmartXplorer to understand the impact of the various tools options. Generally, however, you should analyze the timing results to understand the reason for the high timing score.

If the timing score is greater than 100,000 see Step 7: Review Reports.

Step 6: Run SmartXplorer

Use SmartXplorer to run your design through the tools with different sets of options simultaneously on different machines.

Before ISE® Design Suite Release 12.1, SmartXplorer applied only to *implementation* options. SmartXplorer was enhanced in Release 12.1 to apply to *synthesis* options as well.

SmartXplorer Documentation

Xilinx recommends that you review the following documents before running SmartXplorer:

- *Timing Closure Exploration Tools with SmartXplorer and PlanAhead Tools* (White Paper 287)
- *SmartXplorer for Command Line Users (UG688)*
- *SmartXplorer for Project Navigator Users (UG689)*
- *Command Line Tools User Guide (UG628)*

These documents are cited in Appendix A, Additional Resources.

SmartXplorer Features

SmartXplorer has three key features:

- SmartXplorer performs design exploration by using a set of built-in or user-created implementation strategies to try to meet timing.

  Note: A design strategy is a set of tool options and the corresponding values intended to achieve a particular design goal such as area, speed, or power.
- SmartXplorer allows you to run these strategies in parallel on multiple machines, completing the job much faster.
- SmartXplorer allows you to efficiently explore:
  - Input and output placement
  - Data flow
  - Block RAM placement
  - DSP placement

When to Run SmartXplorer

Xilinx generally recommends running SmartXplorer only when the timing score is less than 1,000,000.

Running SmartXplorer allows you to check the impact of the various tool options on the design.

Some implementation options may have a positive impact or a negative impact on the design. Run multiple cost tables to check the design over the full range of the algorithm.

How to Run SmartXplorer

To run SmartXplorer from Project Navigator, select **Tools > SmartXplorer > Launch SmartXplorer**. Once the dialog box opens, configure SmartXplorer as required for your project.

To run SmartXplorer from the command line, see the tutorial on www.xilinx.com.

Running SmartXplorer to Resolve Timing Issues

You may experience timing issues when you move between major versions of the implementation tools. Xilinx recommends running SmartXplorer with multiple cost tables to resolve these timing issues.

With a single run, the timing results can range based upon the changes introduced in the new version of the implementation tools.

Multiple cost tables:

- Reduce this range and the random effects of changing cost tables.
- Provide more consistent timing results.

For example, consider a design that:

- Met timing in Release 10.x
- No longer meets timing in Release 13.x

If this occurs, Xilinx recommends that you:

- Run multiple cost tables for each version of the tools.
- Compare both the best results and the average results.

The end result will be nearly equivalent, and demonstrates that:

- There was no tool degradation.
- The single run happened to fall into the low range of possible results.

For more information about synthesis options, see Step 3: Drive the Synthesis Tool. For more information about implementation options, see Step 5: Run Implementation.

Step 7: Review Reports

Once SmartXplorer has completed its multiple runs, or if there is an initial timing score of greater than 100,000, review the reports.

If SmartXplorer has been run, analyze the impact of the various options and cost tables to determine which have a positive effect on the design.

The SmartXplorer results show the timing score from each individual run. If one or more runs results in timing being met, modify the design to use these options as default. See the previous sections of this chapter for more information on specific options.

If timing still fails, analyze the timing results in Timing Analyzer.

Reviewing Reports

Review the following reports:

- Synthesis Report
- NGDBuild Report
- Map Report
- PAR Report
- Timing Report

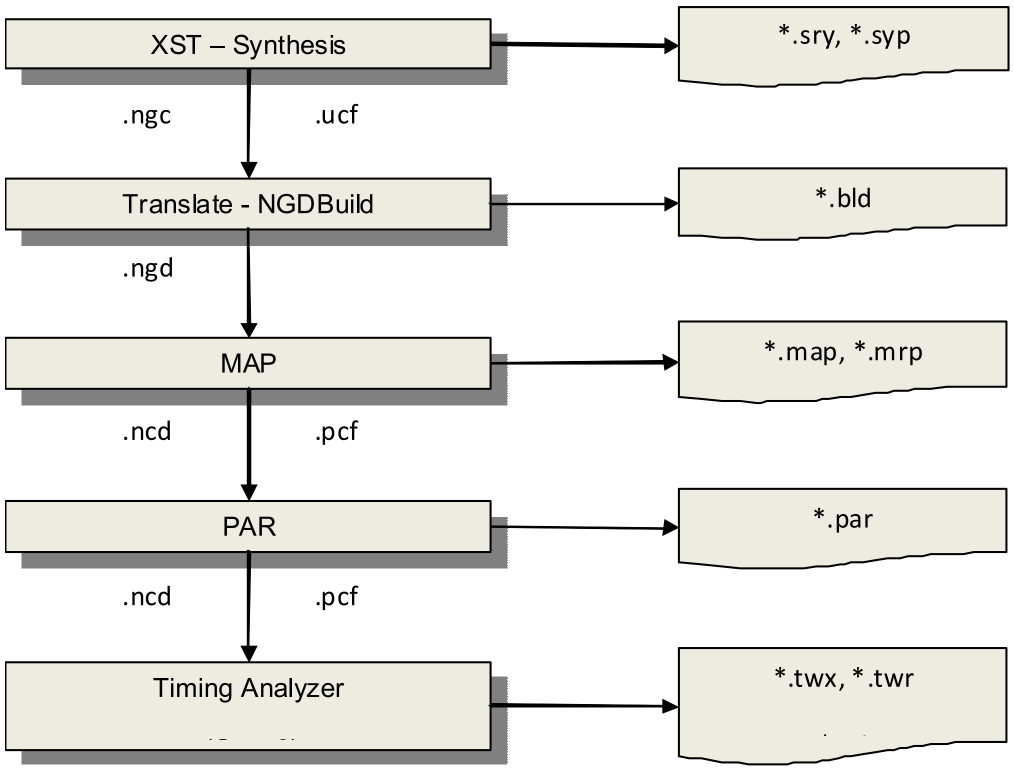


Figure 7-4: **Report Files**

Synthesis Report

Review the Synthesis Report as follows.

Review HDL Advisor Warnings

Review HDL Advisor warnings. These warnings may provide hints for achieving full performance.

```
INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some
arithmetic operations in this design can share the same physical
resources for reduced device utilization. For improved clock frequency
you may try to disable resource sharing.
```

Review the Report for Overuse of Synthesis Constraints

Review the Synthesis Report for overuse of synthesis constraints. Overusing synthesis constraints may lead to:

- Less optimization (KEEP), or
- Excessive replication (MAX_FANOUT).

Review the Report for Excessive Replication

Review the Synthesis Report for excessive replication that could also be caused by:

- The global MAX_FANOUT switch, or
- Register duplication.

Review the Report for Inferred Macros

Review the Advanced HDL Synthesis Report to see which macros are being inferred. This evaluation may indicate the best physical resource to which to map some of the macros (for example, multiplier to DSP48).

If that physical resource is not being used, use attributes such as USE_DSP to force the mapping into certain blocks.

Review Primitive and Black Box Usage for Inferred Primitives

Review Primitive and Black Box Usage to check which primitives are being inferred. This evaluation shows whether block RAM or DSP blocks were not inferred as expected.

Review the report for any asynchronous resets based upon the type of registers that were inferred.

For more information on each primitive, see the *Libraries Guides*, cited in Appendix A, Additional Resources.

NGDBuild Report

Review the NGDBuild Report as follows.

Review Warning and Information Messages

Review warning and information messages relating to the Constraint System.

```
ConstraintSystem:178 - TNM ***, used in period specification 'TS_***,
was traced into MMCM_ADV instance ***. The following new TNM groups and
period specifications were generated at the MMCM_ADV output(s):CLKOUT1:
<TIMESPEC TS_*** = PERIOD "***" TS_*** HIGH 50%
```

Review Messages Relating to Propagating Constraints

Review messages relating to propagating constraints overriding each other.

```
NGDBuild:1345 - The constraint <TIMESPEC TS_*** = PERIOD "****" TS_***/
0.15 HIGH 50% PRIORITY 10;> [top.ucf(4)] is overridden by the constraint
<TIMESPEC TS_*** = PERIOD "****" TS_*** / 0.15 HIGH 50% PRIORITY 1>. The
overriden constraint usually comes from the input netlist or ncf files.
Please set XIL_NGDBUILD_CONSTR_OVERRIDE_ERROR to promote this message
to an error.
```

Confirm Correct Cores and UCF

Confirm in the NGDBuild Report that the correct cores and UCF files have been read into the design.

If there are multiple UCF files, confirm that these files have been used in the NGDBuild Report.

Map Report

Review the Map Report as follows.

Determine if Packing is Suboptimal

Review warnings to determine if packing is suboptimal. Suboptimal packing can create timing closure issues.

```
WARNING:Pack:2549 - The register "reg_1" has the property IOB=TRUE, but
was not packed into the OLOGIC component. The output signal for register
symbol "reg_out" requires general routing to fabric, but the register
can only be routed to ILOGIC, IODELAY, and IOB.
```

Confirm Utilization

Confirm that utilization is as expected, and that erroneous trimming is not occurring.

Look for component types above 65% utilization to determine which component types are becoming limited.

Review Number of LUTs Used as Memory

If limited by LUTs, review the number of LUTs used as memory.

- Can SRLs be dissolved?
- Can DistMem be moved to block RAM?
- Can any arithmetic functions can be moved into DSPs? (See the Synthesis Report).

Review Number of LUTs Used as Shift Registers

If limited by registers, review the number of LUTs used as shift registers.

If the number of LUTs used as shift registers is low, go back to synthesis to see what factors might be preventing the use of SRLs.

Review the Map Report to see if the number of LUTs used as exclusive route-thrus is high. A high exclusive route-thru count can indicate that SRLs are not being properly inferred.

Review the Map Report to see how many unique control sets are reported. If the number of unique control set is over 1,000, rerun MAP with the **-detail** switch to perform a detailed analysis of the control sets.

Physical Synthesis Report

Review the Physical Synthesis Report to understand which optimizations occurred when you use any of the physical synthesis options such as:

- Global optimization
- Logic optimization
- Equivalent register removal
- Retiming
- Register balancing

PAR Report

Review the PAR Report as follows.

Ensure That All Clocks Are Utilizing the Proper Resource

Review the Clock Report to ensure that all clocks are utilizing the proper resource. Large clock skew on a local resource can indicate that it has a connection to an improper component.

Verify That the Component Switching Limit Score Is 0

When the final timing score is reported, verify that the component switching limit score is 0. If not, review the Timing Report to see which component specs are being violated.

Ensure That All Constraints Are Being Properly Analyzed

Review the final Timing Report to ensure that all constraints are being properly analyzed.

If there is a constraint for which no paths are analyzed:

- There may be a problem with the constraint definition, or
- Another constraint might be overriding the first constraint.

If you suspect that another constraint is overriding the first constraint, generate a timespec interaction report in TRCE.

Review the number of global clock buffers to ensure that all clocks are driven by a global clock buffer. MAP adds circuits to drive unused clocking resources with low speed local clocks.

Timing Report

For information on reviewing the Timing Report, see Step 8: Run TRCE and Analyze Timing Results and Report.

Step 8: Run TRCE and Analyze Timing Results and Report

When a design fails timing, review the Timing Report to check the constraint that is failing and the type of failure.

This section analyzes various timing scenarios and related topics to show how to understand the timing results, and how to use the information most effectively.

Ask the following questions if the design is failing timing:

- Are my constraints correct?
  - Should the failing path be covered by a Multi-Cycle of false path constraint?
  - Is the failing path due to over- constraining?
  - Are the synthesis timing constraints consistent with the implementation constraints specified in the UCF file?
- Is the netlist reasonable?
  - Is synthesis behaving as expected?
  - Are there unexpected high fanout nets?
  - Are clock trees leading to large skew?
- Is place and route behaving as expected?
  - Is placement spread out?
  - Is routing satisfactory?

Each question is answered in example scenarios discussed in the next section. These scenarios examine different timing failures and provide recommendations for each failure.

Chapter 8

Overcoming Timing Failures

Use Timing Analyzer or the **`trce`** command to analyze timing constraints.

This timing analysis:

- Provides a detailed path analysis of the timing path with regards to the timing constraint requirements.
- Ensures that the specific timing constraints are passed through the implementation tools.

The path specific details include the following:

- Confirms that the timing requirements were met for all paths per constraint
- Confirms the setup and hold requirements were met for all paths per constraint
- Confirms that the device components are performing within operational frequency limits
- Provides a list of unconstrained paths that may be a critical path that was not analyzed

Reviewing Timing Results

The timing results can be reviewed in Timing Analyzer with the TWX and any text editor with the TWR. In both cases, all worst case or critical paths are reported per constraint.

Reporting Paths by Endpoints

Timing Analyzer and TRCE can also report the paths by endpoints for each constraint. This reporting provides more details on the failing endpoints that are the most critical for each constraint.

The same path details are reported, including:

- Clock to out of the source element
- Some routing and logic
- Setup of the destination element.

The failing paths are shown in red in the Timing Analyzer index panel. Running the analysis by endpoints provides the number of paths to a single endpoint. This discloses the location of the common critical paths for each constraint.

TimeSpec Interaction (TSI) Report

If a path is being analyzed under a different constraint than expected, the TimeSpec Interaction (TSI) Report provides insight into:

- The interaction between constraints, and
- The constraints that can be combined to reduce memory and runtime of the implementation tools.

Time Group Membership

The time group membership may be the root cause of an unexpected interaction. Timing Analyzer can generate a Query Time Group Report showing the elements associated with each time group.

Make the corrections to the time group memberships to remove paths from the interacting timing constraints.

Device Utilization

Review the Design Summary in the MAP and PAR Reports to verify device utilization. There will likely be some variations between the Synthesis Utilization Report and the MAP Design Summary Report.

Review the placement of the clock networks to gain insight into the critical paths. Timing Analyzer's Report on Net Delays reports on the clock network delays and clock loads.

Clock Report

Verify the clock networks and the associated clocking elements in:

- The Clock Report section of the PAR Report, and
- The Timing Analyzer Report on Clock Regions

The Clock Report helps to ensure that the clocks were not routed using incorrect routing resources, such as local routing resources.

The Clock Report lists:

- Clock networks detected by PAR
- Clocking buffer resources that the clock net are routed through
- Clock fanout
- Net skew
- Clock net delay to the clock loads

Timing Summary

The PAR Report Timing Summary provides:

- A snapshot of the performance requirements
- The best-case achievable performance for each clock domain

If the design has failing constraints, the Timing Summary reports the failing constraints with the worst case slack, timing errors, timing score, and best-case achievable per constraint.

Useful Strategies

Use the following strategies to improve:

- Failing constraints
- Runtime
- Memory
- Overall design performance

Use Timing Constraints in Synthesis

Use timing constraints in the synthesis tool for better design implementation.

Use Global Timing Constraints on Clocks

Use global timing constraints instead of individual timing constraints on every clock:

- Offset In constraint on all inputs (Global Offset In)
- Offset Out constraint on all outputs (Global Offset Out)
- Period constraint on the input clock signal

Use the Feedback Constraint

In order to insure that the Offset In and Offset Out constraints are analyzed correctly with an off-chip deskewing clock topology, the Feedback constraint provides the external PCB delay for the overall Offset In and Offset Out analysis.

If the off-chip delay is set with the Feedback constraint, the timing analysis incorporates the PCB delay into the clock path of the Offset In and Offset Out constraint analysis.

Do Not Over-Constrain

Do not over-constrain the design. Set the Period constraint to the actual frequency at which the design will operate.

Over-constraining the design:

- Makes it more difficult for the implementation tools to achieve overall performance.
- Can produce worse results than using the realistic timing performance objectives.
- Is the most common cause of long implementation runtime.

Use Pad Time Group Specific Constraints

Use pad time group specific Offset In and Offset Out constraints:

- For exceptions from the global Offset In and Offset Out constraints, when
- The input or output signals:
  - Are clocked by the same clock signal, but
  - Have different timing requirements.

Use From:To or Multi-Cycle Constraints

Use From:To or Multi-Cycle constraints to define a Multi-Cycle path that does not have the same timing requirement as the Period or single cycle.

The individual timing elements are defined with time groups or can be specified with the pre-defined time groups (such as FFS or RAMS).

Setup and hold analysis is done during timing analysis for Virtex-5 and newer devices.

Review Failing Timing Paths and Critical Paths

Review the failing timing paths and the critical paths in the Timing Analysis Report.

Change Synthesis and Implementation Options

Change the synthesis and implementation options. Use implementation tool options in MAP and PAR, such as SmartXplorer.

For more information, see the *Command Line Tools User Guide (UG628)*, cited in Appendix A, Additional Resources.

Use Floorplanning

Use floorplanning techniques on the critical path to improve placement and packing.

For more information, see the *Floorplanning Methodology Guide (UG633)*, cited in Appendix A, Additional Resources.

Use Clock Region Area Groups

Use clock region area groups with time groups as area groups to confine the synchronous elements of the global clock buffers to specific clock regions to prevent contention in clock regions between global clocks.

AREA_GROUP is attached to logical blocks in the design. The string value of the constraint identifies a named group of logical blocks that are to be packed together by mapper and placed in the ranges if specified by PAR.

If AREA_GROUP is attached to a hierarchical block, all sub-blocks in the block are assigned to the group.

Once defined, an AREA_GROUP can have additional constraints associated with it to control its implementation.

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Use Relationally Place Macros (RPM) Constraints

Use Relationally Place Macros (RPM) constraints to improve packing and placement by defining the relative placement of the individual synchronous elements.

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Use LOC Constraints

Use LOC constraints to manually constrain the placement of the larger components, such as BlockRAM, Multiplier/DSP, and other clock modifying blocks (such as DCM and PLL) to reduce the implementation runtime. This improves placement and packing by placing the individual synchronous elements in a specific location on the device.

Common Causes of Timing Failures

The most common causes of timing failures are as follows.

High Fanout Nets

High fanout nets result in poor synthesis, placement, routing, or any combination of these. Use logic replication or duplication techniques in synthesis or HDL code.

High Delay Nets

High delay nets results in poor placement, routing, or both. Use Area Groups to confine the placement

High Number of Logic Levels

High number of logic levels results in poor placement, routing, or both. Add pipeline registers, use one-hot state machines, and use **`case`** statements instead of **`if/else`** statement.

High Number of Asynchronous Resets

High number of asynchronous resets, which are not analyzed by default. Add ENABLE constraints for asynchronous paths through the synchronous element (REG_SR_O) and/or for asynchronous reset recovery time of the synchronous element (REG_SR_R).

Poor Packing in MAP

Poor packing in MAP results in poor placement, poor routing, or both. Use any of the following strategies:

- Use BLKNM to force elements to be packed together.
- Use XBLKNM to force elements to *not* be packed together
- Use Area Groups to confine the packing and placement.

Poor Placement

To correct poor placement in general, use Area Groups and Relationally Placed Macros (RPMs) to confine the placement.

Poor I/O Timing

Poor I/O timing results in poor placement, poor routing, or both.

Move the IOB Flip Flops or SLICE Flip Flops to meet timing.

Timing Failure Design Scenarios

The following design scenarios show different timing failures:

- Designs With High Number of Levels of Logic
- Designs With High Fanout
- Designs With High Clock Skew
- Designs With Non-Optimal Placement
- Designs Failing Offset In
- Designs Failing Offset Out
- Designs Fail in Hardware Even Though Timing is Met

Designs With High Number of Levels of Logic

A data path is considered to have a high number of logic levels when the logic delay exceeds a given percentage of the total path delay. This implies that there is too much logic between timing end points. Reduce the amount of logic to meet timing requirements.

The given percentage of the total path delay was traditionally around 50% for older architectures, and around 60% for Virtex® families. There are exceptions to this rule for carry chain paths, in which the logic delays are much smaller and allow for a higher number of logic levels or a lower component percentage.

The Timing Report may show a result similar to the following:

```
Requirement: 2.500ns
Data Path Delay: 2.366ns (Levels of Logic = 17)
```

The following appears in the data path calculation:

```
-------------------------------------------------------
Total            2.366 ns (2.079ns logic, 0.287ns route)
                      (87.9% logic, 12.1% route)
```

Evaluate the number of logic levels to see if the number is unrealistic for the timing requirement.

Evaluate paths with too many levels of logic in synthesis.

If synthesis does not see them as timing critical, try over-constraining in synthesis to reduce the logic levels.

Reduce Levels of Logic

To reduce the levels of logic, return to the source and try the following:

- Issue state machine optimization suggestions.
  For more information, see *Xilinx® Answer Record 9411*.
- Use **case** statements instead of nested **if-else** statements.
- Use tristate instead of large MUXes (7 or more inputs).
- Use creative math. For example, shift instead of multiplying by multiples of two.
- Use decoders instead of comparators.
- Balance logic around registers.
- Pyramid logic with parentheses instead of serial implementation.

- Use **if-then-else** statements only to:
  - Pre-decode and register counter values
  - Add a level of pipelining to pre-decode and register input signals
- Use MUXes with more than 7-bit wide buses only to do the following:
  - Instead of logic, use registers that are in a tristate condition.
  - Drive enable signals from registers; tristate are in a tristate condition when enable signals are **1**, and drive signals when the enable is **0**.
  - Use floorplan tristates.
- Add pipeline registers.

How to Debug Designs with High Logic Levels

This section shows how to debug designs with high logic levels.

The design under consideration has the following parameters:

- The design is a 68-bit counter.
- The clock for this counter is being derived using a DCM.
- The input frequency to the DCM is 100 MHz.
- The clock that drives the counter is 400 MHz.

When this design is implemented, a setup violation is reported. The timing summary shows that the datapath delay is very high for the requirement. 17 levels of logic is high for a 2.5 ns requirement.

```
Slack:                 -0.022 ns (requirement - (data path - clock path skew + uncertainty))
   Source:                 TestCounter/Count_0 (FF)
   Destination:            TestCounter/Count_67 (FF)
   Requirement:           2.500ns
   Data Path Delay:       2.366ns (Levels of Logic = 17)
   Clock Path Skew:       -0.061ns (1.007 - 1.068)
   Source Clock:          Clock4X rising at 0.000ns
   Destination Clock:     Clock4X rising at 2.500ns
   Clock Uncertainty:     0.095ns

   Clock Uncertainty:           0.095ns ((TSJ^2 + DJ^2)^1/2) / 2 + PE
     Total System Jitter (TSJ): 0.070ns
     Discrete Jitter (DJ):       0.176ns
     Phase Error (PE):           0.000ns
```

Reviewing the detail of the datapath shows that logic is a large proportion of the datapath delay.

```
Maximum Data Path: TestCounter/Count_0 to TestCounter/Count_67
     Location             Delay type         Delay(ns)  Physical Resource
                                                         Logical Resource(s)
     ------------------------------------------------- -------------------
     SLICE_X48Y48.AQ       Tcko                  0.346   TestCounter/Count<3>
                                                        TestCounter/Count_0
     SLICE_X48Y48.A4      net (fanout=1)        0.278    TestCounter/Count<0>
     SLICE_X48Y48.COUT     Topcya                0.384   TestCounter/Count<3>
                                                   TestCounter/Mcount_Count_lut<0>_INV_0
                                                        TestCounter/Mcount_Count_cy<3>
     SLICE_X48Y49.CIN     net (fanout=1)        0.000    TestCounter/Mcount_Count_cy<3>
     SLICE_X48Y49.COUT     Tbyp                  0.082   TestCounter/Count<7>
                                                        TestCounter/Mcount_Count_cy<7>
     SLICE_X48Y50.CIN     net (fanout=1)        0.009    TestCounter/Mcount_Count_cy<7>
     SLICE_X48Y50.COUT     Tbyp                   0.082   TestCounter/Count<11>
…      …      …
…      …      …
…      …      …
SLICE_X48Y63.CIN net (fanout=1) 0.000 TestCounter/Mcount_Count_cy<59>
     SLICE_X48Y63.COUT     Tbyp                   0.082   TestCounter/Count<63>
                                                         TestCounter/Mcount_Count_cy<63>
     SLICE_X48Y64.CIN     net (fanout=1)        0.000    TestCounter/Mcount_Count_cy<63>
     SLICE_X48Y64.CLK      Tcinck                 0.119   TestCounter/Count<67>
                                                         TestCounter/Mcount_Count_xor<67>
                                                         TestCounter/Count_67
     ----------------------------------------------------------------------------
     Total                                      2.366ns (2.079ns logic, 0.287ns route)
                                                        (87.9% logic, 12.1% route)
```

Because a large portion of the total delay is logic delay, the path must be optimized. The implementation tools can not optimize the path by default, because the component delays exceed the routing delays for the path.

View the design in FPGA Editor or the PlanAhead tool to check the data path and various logic delays. For more information, see:

- Cross Probing Between FPGA Editor and Timing Analyzer in Chapter 9
- Cross Probing From the PlanAhead tool to FPGA Editor in Chapter 9

The counter is 68 bits wide. If the 68-bit counter has been split into two 34-bit counters in the HDL code, the number of levels of logic can be reduced.

```
wire [33:0] TestCount1;
Counter TestCounter1 (.Clock (Clock4X),
 .Reset  ( Reset | ~ClockReady ),
  .Enable ( Channel0 ),
  .Count  ( TestCount1 ) );
              defparam TestCounter2.width = 34;
wire [33:0] TestCount2;
reg [33:0] TestCount1_33;
reg TestCounter2En;
  always@(posedge Clock4X)
begin
  TestCount1_33 <= TestCount1[33:0];
    if (Xilinx TestCount1_33) TestCounter2En <= 1'b1;
      else TestCounter2En <= 1'b0;
    end
counter TestCounter2 ( .Clock  ( Clock4X ),
  .Reset  ( Reset | ~ClockReady ),
  .Enable (TestCounter2En),
  .Count  ( TestCount2 ) );
```

The **`enable`** signal of the first counter is always driven **`high`**. When the output of **`counter1`** becomes **`34'b1`**, the **`enable`** signal is active for **`counter2`**. Splitting the counter of 68 data width into two counters of 34 bits each can reduce the timing violation because of number of logic levels.

Designs With High Fanout

Review the fanout on various nets in the details section of the Timing Report. If a path is failing timing, examine the fanout of the various signals.

Design With High Fanout Example

```
Location Delay type Delay(ns) Physical Resource
                                                   Logical Resource(s)
     -------------------------------------------------------------------
     SLICE_X19Y78.YQ       Tcko                  0.258   d_mid
                                                        d_mid
     SLICE_X22Y81.BY      net (fanout=16)       0.520    d_mid
     SLICE_X22Y81.CLK      Tdick                 0.210   d_Aux<8>
                                                        d_Aux_8
     ---------------------------------------------------------------------------
     Total                                      0.988ns (0.468ns logic, 0.520ns route)
                                                        (47.4% logic, 52.6% route)
```

Resolving a Path with High Fanout

Use one of the following methods to resolve a path with high fanout leading to long net delays:

- Floorplan or apply an AreaGroup constraint to the logic to reduce the net delay.
- Apply a LOC constraint to the origin and add a global buffer to the high fanout signal.

  This method applies only for a very high fanout reset net. While global buffers are generally used only for clocks, they can also be used when high fanout nets are required, assuming available resources.
- Duplicate the driver and instruct the synthesis tool not to remove the duplicate logic.
- Use specific net fanout control on the specific net, if the synthesis tool allows. This method is generally the most effective. For more information, see Max Fanout in the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*, cited in Appendix A, Additional Resources.

Designs With High Clock Skew

The timing tool supports a path delay analysis that accounts for clock skew. The clock skew is added to the calculated data path delay to arrive at a total path delay that is compared to the constraint. Alternatively, it is reported as the delay for the path when the constraint has no value.

Skew is taken into account only when it works against the constraint. Skew is truncated to zero if the reverse is true. This gives worst case timing results.

Design with High Clock Skew Example

What constitutes high clock skew depends on the device, architecture, and specific clock path and structure. Following is an example:

```
Requirement: 14.000ns
   Data Path Delay:      5.401ns (Levels of Logic = 0)
   Clock Path Skew:      -9.178ns (2.994 - 12.172)
```

Clock Path Skew of 9.178 ns is considered very high in all circumstances. Because the example crosses between two asynchronous clock domains, do not consider clock skew for this path. Because the source CLK is driven from a FF, and the destination clock is from a GTP through a BUFG, PLL, BUFG, DCM, or BUFG, it has a long delay.

This design:

- Has a From-To between the clock domains.
- Does not use the DATAPATHONLY keyword to instruct the tools to ignore the clock skew.

This is a common mistake in many cross clock domain constraints.

For more information, see the *Constraints Guide (UG625)*, cited in Appendix A, Additional Resources.

Because the tools assume a relationship between the clocks for analysis even if they are asynchronous, instruct the tools to ignore the clock skew if necessary.

Debugging Timing Reports With High Clock Skew

You must first understand the source clock and destination clock and their relationship.

When the Source and Destination Clock Are the Same

When the source and destination clock are the same, the tools use the common node on the clock path to determine the clock skew. It is difficult to manually confirm the skew in this case, as the common node on the clock path can be difficult to find. Xilinx recommends calculating the skew back to the common driver to determine if the skew in the timing analysis is somewhat similar.

When the Source and Destination Clock Are Not the Same

When the source and destination clocks are not the same, the tools propagate the clock back to the common driver to determine the clock skew. The tools always use the worst case path for skew analysis.

In the case of a multiplexing clock using a BUFGMUX, the tools may use the wrong clock for the specific analysis. To correct this, apply a PIN Timing Ignore constraint on the BUFGMUX pin that does not require analysis.

An example of the constraint is:

```
PIN "BUFGMUX_inst_name.I1_pin_name" TIG
```

Priority Constraint

The tools do *not* use the Priority keyword to determine the clock skew. Because the Period constraint constrains only the data path, it is not used in the clock skew calculation. The recommendation given above is the only way to control clock skew calculation when multiplexing clocks.

Cross Probing

The best way to analyze the clock paths is to use FPGA Editor or the PlanAhead tool and cross probe with Timing Analyzer. For more information, see Chapter 9, Cross Probing.

Designs With Non-Optimal Placement

There are many different scenarios in which non-optimal placement can cause timing issues. In Virtex®-6 and Spartan®-6 devices, placement rather than routing has the biggest impact on timing closure.

Design With Non-Optimal Placement Example

Following is an example of non-optimal placement causing timing failure. This example scenario is a route between a DSP and block RAM in a Spartan-6 device.

To debug this timing issue, you must understand the device architecture and use the cross probing techniques outlined in Chapter 9, Cross Probing.

```
Slack: -0.188ns (requirement - (data path - clock path skew + uncertainty))
   Source:                ingressLoop[0].ingressFifo/buffer_fifo/Mram_fifo_ram (RAM)
   Destination:           arnd1/transformLoop[0].ct/Maddsub_n00271 (DSP)
   Requirement:           5.804ns
   Data Path Delay:       5.920ns (Levels of Logic = 0)
   Clock Path Skew:       -0.037ns (0.440 - 0.477)
   Source Clock:          bftClk_BUFGP rising at 0.000ns
   Destination Clock:     bftClk_BUFGP rising at 5.804ns
   Clock Uncertainty:     0.035ns

   Clock Uncertainty:           0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
     Total System Jitter (TSJ): 0.070ns
     Total Input Jitter (TIJ):    0.000ns
     Discrete Jitter (DJ):        0.000ns
     Phase Error (PE):            0.000ns

   Maximum Data Path at Slow Process Corner: ingressLoop[0].ingressFifo/buffer_fifo/
Mram_fifo_ram to arnd1/transformLoop[0].ct/Maddsub_n00271
     Location             Delay type         Delay(ns)  Physical Resource
                                                         Logical Resource(s)
     -------------------------------------------------------------------
     RAMB16_X1Y20.DOB18   Trcko_DOB             2.900   ingressLoop[0].ingressFifo/
buffer_fifo/Mram_fifo_ram
                                                    ingressLoop[0].ingressFifo/buffer_fifo/
Mram_fifo_ram
     DSP48_X0Y10.B2       net (fanout=2)        2.783   toBft<1><2>
     DSP48_X0Y10.CLK      Tdspdck_B_B0REG       0.237   arnd1/transformLoop[0].ct/
Maddsub_n00271
                                                  arnd1/transformLoop[0].ct/Maddsub_n00271
     ---------------------------------------------------------------------------
     Total                                      5.920ns (3.137ns logic, 2.783ns route)
                                                        (53.0% logic, 47.0% route)
```

The Timing Report shows that the only variable in the path is the route between the RAM and DSP. This confirms that placement is most likely the issue with this failing path.

Viewing the Path

View this path in the PlanAhead tool or FPGA editor to obtain a full understanding of the placement.

The following figure shows the failing path in the PlanAhead tool.

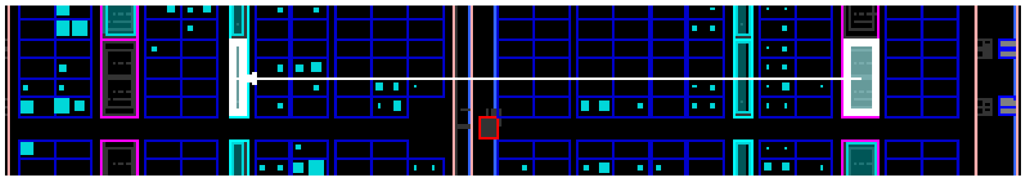


Figure 8-1: **Failing Path**

The path is routing from a RAM on the right hand side of the device to a DSP on the left hand side. Improving the placement will help resolve this timing issue.

The following figure shows the failing path in FPGA Editor. The routing is highlighted in red.

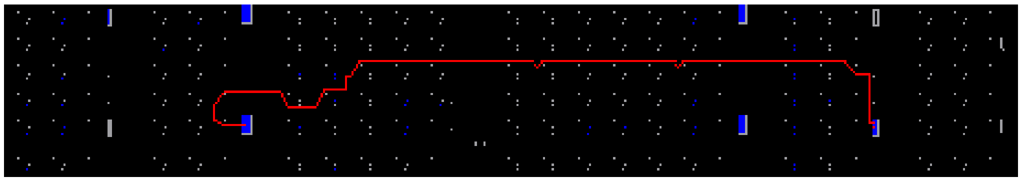


Figure 8-2: **FPGA Editor Highlighted Routing**

Place DSP and RAM on the Same Side of the Device

To achieve timing closure, place the DSP and RAM on the same side of the device.

- Create specific LOC constraints for the DSP and RAM so the instances have a LOC constraint applied in order to achieve timing closure. While this can be difficult to do for a full design, this method is quiet effective if there is only a single timing failure.
- Create AREA GROUP constraints to lock logic to a specific area of a clock region. This requires the placer tool to place logic in a specific area. Use the PlanAhead tool to create a Pblock for the block in question.
- Apply a MAX_DELAY on the path between the RAM and DSP, giving it a higher precedence than the Period constraint. This method is effective for most, but not necessarily all, designs.
- Pipeline the logic between the RAM and DSP blocks to give the placer tool maximum flexibility in achieving timing closure for the full design.

Designs Failing Offset In

An Offset In constraint basically constrains the input path. In case of a violation, review the Timing Report to assess the problem.

Design Failing Offset In Example

```
Timing constraint: OFFSET = IN 1.5 ns VALID 10 ns BEFORE COMP "CLK" "RISING";
 1 path analyzed, 1 endpoint analyzed, 1 failing endpoint
 1 timing error detected. (1 setup error, 0 hold errors)
 Minimum allowable offset is   1.561ns.
 --------------------------------------------------------------------------------

 Paths for end point DATAX (SLICE_X62Y38.AX), 1 path
 --------------------------------------------------------------------------------
 Slack (setup path):     -0.061ns (requirement - (data path - clock path - clock arrival + 
uncertainty))
   Source:               DATAIN (PAD)
   Destination:          DATAX (FF)
   Destination Clock:    CLK1 rising at 0.000ns
   Requirement:          1.500ns
   Data Path Delay:      0.983ns (Levels of Logic = 1)
   Clock Path Delay:     -0.410ns (Levels of Logic = 3)
   Clock Uncertainty:    0.168ns
```

Improving the Offset In Slack

To improve the Offset In slack, you can:

- Reduce the data path.
- Increase the clock path.
- Add positive shift to the clock.

  The clock arrival is now positive if the clock is generated internally using a DCM, MMCM, or PLL

The preferred method to use depends on the design. See the following examples.

- If the data path is from a pad to a SLICE that implements the input FF, choose a LOC for the pad or SLICE to reduce the data path.
- If the input FF is implemented on the ILOGIC in the proximity of the PAD, the data path is already minimal.

Improving the Offset In Slack Example

Following is the result of adding a Phase Shift of 20 degrees to a clock which is clocking the input FF:

```
========================================================================
 Timing constraint: OFFSET = IN 1.5 ns VALID 10 ns BEFORE COMP "CLK" "RISING";
 1 path analyzed, 1 endpoint analyzed, 0 failing endpoints
 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum allowable offset is   1.057ns.
 --------------------------------------------------------------------------------

 Paths for end point DATAX (SLICE_X62Y77.AX), 1 path
 --------------------------------------------------------------------------------
 Slack (setup path):      0.443 ns (requirement - (data path - clock path - clock arrival +
uncertainty))
   Source:                 DATAIN (PAD)
   Destination:            DATAX (FF)
   Destination Clock:     CLK1 rising at 0.547ns
   Requirement:           1.500ns
   Data Path Delay:       1.065ns (Levels of Logic = 1)
   Clock Path Delay:      -0.410ns (Levels of Logic = 3)
   Clock Uncertainty:     0.129ns
```

The clock arrival has been changed from **`rising at 0.000 ns`** to **`rising at 0.547 ns`**. This was enough to bring the slack to pass with slack of +0.433ns.

Designs Failing Offset Out

Violations with the Offset Out constraint are similar to violations with the Offset In constraint in terms of debugging procedure. Only the path covered is different.

Design Failing Offset Out Example

```
========================================================================
 Timing constraint: OFFSET = OUT 2.5 ns AFTER COMP "CLK";
 1 path analyzed, 1 endpoint analyzed, 1 failing endpoint
 1 timing error detected.
 Minimum allowable offset is   2.818ns.
 --------------------------------------------------------------------------------

 Paths for end point DATAOUT (T7.PAD), 1 path
 --------------------------------------------------------------------------------
 Slack (slowest paths): -0.318ns (requirement - (clock arrival + clock path + data path +
uncertainty))
   Source:                DATAOUT (FF)
   Destination:           DATAOUT (PAD)
   Source Clock:          CLK2 rising at 0.000ns
   Requirement:           2.500ns
   Data Path Delay:       3.066ns (Levels of Logic = 1)
   Clock Path Delay:      -0.408ns (Levels of Logic = 3)
   Clock Uncertainty:     0.160ns
```

Improving the Offset Out Slack

To improve the Offset Out slack, you can:

- Make clock arrival less positive or more negative when there is a clock component such as an MMCM or a PLL.
- Reduce the data path
- Reduce the clock path

Improving the Offset Out Slack Example

The above example is a failing Offset Out constraint in which there was already a phase shift of -20 degrees on the MMCM. After changing the phase shift to -30 degrees and rerunning implementation, the slack changed to positive as shown below.

```
========================================================================
 Timing constraint: OFFSET = OUT 2.5 ns BEFORE COMP "CLK";
 1 path analyzed, 1 endpoint analyzed, 0 failing endpoints
 0 timing errors detected.
 Maximum allowable offset is   2.613ns.
 --------------------------------------------------------------------------------

 Paths for end point DATAOUT (A11.PAD), 1 path
 --------------------------------------------------------------------------------
 Slack (slowest paths):   0.113 ns (requirement - (clock arrival + clock path + data path +
uncertainty))
   Source:                 DATAOUT (FF)
   Destination:            DATAOUT (PAD)
   Source Clock:          CLK2 rising at -0.391ns
   Requirement:           2.500ns
   Data Path Delay:       3.061ns (Levels of Logic = 1)
   Clock Path Delay:      -0.408ns (Levels of Logic = 3)
   Clock Uncertainty:     0.125ns
```

Understanding How the Constraints Interact

One of the biggest problems in most designs is that of constraint interaction. You must understand the following:

- How do the constraints interact with each other?
- How is the precedence of the constraints understood?

Incorrect understanding of how the constraints interact may result in paths which been incorrectly constrained.

To understand the interaction of constraints, generate a TSI Report from the command line, or in Timing Analyzer.

Generating a TSI Report from the Command Line

To generate a TSI Report from the command line, use the **-tsi** options in the TRCE command.

For more information, see the *Command Line Tools User Guide (UG628*, cited in Appendix A, Additional Resources.

Generating a TSI Report from Timing Analyzer

To generate a TSI Report in Timing Analyzer:

1. Select **Timing > Run Analysis**.
2. In the Run Timing Analysis dialog box, select **A separate constraints interaction report**.

Constraint Interaction Report Example

The TSI Report has a section showing constraint interaction as follows:

```
Constraint Interaction Report
 =============================

 Constraint interactions for TS_SYS_CLK = PERIOD TIMEGRP "clk_250mhz" 4 ns HIGH 50%;
        1438 paths removed by TS_i_Clocking_i_PLL_250_CLKOUT0_BUF = PERIOD TIMEGRP
"i_Clocking_i_PLL_250_CLKOUT0_BUF" TS_clk_100M / 2.5 HIGH 50%;

 Constraint interactions for TS_clk_27M = PERIOD TIMEGRP "TNM_clk_27M" 37.037 ns HIGH 50%;
          51 paths removed by PATH "TS_resync_regs_path" TIG;
```

The example shows that 1438 paths are removed from TS_SYS_CLK Period constraint by a Period constraint that propagates through a PLL.

Timing Report or PAR Report Example

Information on the propagated constraints can be viewed in the Timing Report or the PAR Report.

```
Derived Constraint Report
Derived Constraints for TS_SYS_CLK
+-------------------------------+-------------+-------------+-------------+-------------+-
------------+-------------+-------------+
|                                         |   Period          |       Actual Period
|      Timing Errors           |       Paths Analyzed        |
|          Constraint                      | Requirement |-------------+-------------
|-------------+------------- |-------------+------------- |
|                                              |                       |   Direct        |
Derivative |   Direct      | Derivative |   Direct     | Derivative |
+-------------------------------+-------------+-------------+-------------+-------------+-
------------+-------------+-------------+
|TS_SYS_CLK                    |     4.000ns |     1.818ns|     1.162ns|            0|
0|            0|          1039|
| TS_MC_RD_DATA_SEL      |     16.000ns|      4.648ns|         N/A |               0|
0|          404|              0|
| TS_MC_RDEN_SEL_MUX |    16.000ns|     2.891ns|        N/A |            0|          0|
160|             0|
```

The Constraints Interaction Report shows that the Timing Ignore constraint removes 51 paths from a Period constraint.

Clock Domain Overlap Report

The Clock Domain Overlap Report highlights how the clock domains overlap. The tool reports all the elements that are common to the specific clock domains.

```
Clock Domain Overlap Report
===========================

TS_i_Clocking_clk_148M5_i = PERIOD TIMEGRP "i_Clocking_clk_148M5_i" TS_clk_74M         / 2
HIGH 50%;
TS_i_Clocking_clk_74M_pll = PERIOD TIMEGRP "i_Clocking_clk_74M_pll" TS_clk_74M        HIGH
50%;
TS_i_Clocking_i_27M_PLL_CLKOUT2_BUF = PERIOD TIMEGRP
"i_Clocking_i_27M_PLL_CLKOUT2_BUF" TS_clk_74M / 0.363636364 HIGH 50%;
TS_i_Clocking_clk_13M5_i = PERIOD TIMEGRP "i_Clocking_clk_13M5_i" TS_clk_74M / 0.181818182
HIGH 50%;
TS_clk_force_pp_148M = PERIOD TIMEGRP "clk_force_pp_148M" 6.734 ns HIGH 50%        PRIORITY
1;
 {
    i_PreProcessor/i_Video/i_TRS_Insert/sample_number_1 (i_PreProcessor/i_Video/
i_TRS_Insert/sample_number<3>.CLK)
    i_PreProcessor/i_Video/i_TRS_Insert/sample_number_2 (i_PreProcessor/i_Video/
i_TRS_Insert/sample_number<3>.CLK)
    i_PreProcessor/i_Video/i_TRS_Insert/sample_number_3 (i_PreProcessor/i_Video/
i_TRS_Insert/sample_number<3>.CLK)
```

Xilinx recommends that you review both the Constraints Interaction Report and the Clock Domain Overlap Report to ensure that all the constraints have been used as required.

Constraints propagate through all clock capable components in the FPGA device, such as a BUFG, DCM, PLL, and MMCM. Reviewing this report is important to see how the constraints propagate and interact or overlap with each other.

Grouping logic without fully understanding the logic contained within the group can lead to problems with interacting constraints. For example, a Timing Ignore constraint may be interacting and overriding more logic than expected, resulting in incorrect implementation runs and timing analysis.

Reviewing the Unconstrained Path Report

When examining timing performance, review the Unconstrained Path Report to ensure that no paths were overlooked when constraining. Generally, there should be no unconstrained paths.

A design may meet timing but still fail in hardware. The design should work in hardware if the design is constrained correctly (that is, all paths have appropriate constraints applied).

Do not add Period constraints without applying Offset In and Offset Out constraints. Without Offset In constraints, the tools have no knowledge of the relationship between clock and data arriving at the FPGA devices. In this case, the setup and hold time at the first synchronous element will not be analyzed.

To turn on the unconstrained path analysis, select **Do unconstrained analysis and report unconstrained paths** in the Run Timing Analysis dialog box. Each constraint type is displayed separately. This makes it easier to see exactly which clocks, input paths, output paths, or individual paths are unconstrained.

Component Switching Limits Check

Use Component Switching Limits to confirm that the switching limits of the hardware (such as a DCM and BUFG) as specified in the device datasheet have been met. These are reported as a separate timing score in the PAR Report.

```
Phase 6: 0 unrouted; (setup:29212, Hold:319991, Component Switching Limit:0)
```

In the Timing Report in Timing Analyzer, component switching limits are analyzed as a separate analysis to the setup and hold. The component switching limit is a device or silicon limit. It is not a design limit. If the clock frequency is changed, the component switching limits may change for the components in the design. Component Switching Analysis is done only on synchronous elements, such as DSP48, Block Ram, BUFG, and MMCM.

The design should have no component switching limit violations. The component switching limit violations are used to highlight that the clock frequency is not within the specified limits of the device. Component switching limit violations can impact the tools performance, resulting in non-optimal placement and routing. These are the first errors that should be resolved when trying to close timing.

Designs Fail in Hardware Even Though Timing is Met

Some designs may fail in hardware even though timing is met. These failures may be related to timing rather than the functionality of the design on the hardware. For example:

- The design fails in high temperatures, but works in low temperatures.
- The design fails on some boards, but works on others.

Debugging Designs That Fail in Hardware Even Though Timing is Met

To debug designs in which timing is met but the design fails in hardware:

1. Check the unconstrained paths.
   a. Make sure the design is fully constrained.
   b. Apply the Timing Ignore (TIG) constraint to those paths that can be ignored.
   c. Be sure that valid multi-cycle paths are applied, and that all valid cross-clock domain paths are covered correctly.
2. Add System Jitter and Input Jitter information in the User Constraint File (UCF), especially when the worst case slack is very small. In many designs, jitter has not been considered, implying that the design is under-constrained.

   For more information on applying Jitter constraints, see Clock Uncertainty in Chapter 6.
3. The design failure can be caused by a board issue. Be sure to follow the SSO, SSN and PCB design guidelines:

   http://www.xilinx.com/products/design_resources/signal_integrity/si_pcbcheck.htm
4. Narrow the problem down to a specific failing path in the FPGA. Use the ChipScope™ tool, or probe the internal signals to unused pads in FPGA Editor. After the failing path is found, slow down the frequency of the clock driving the source and the destination.
   a. If the failure persists, it may be caused by a hold time violation. Try to add a route through LUT to the path in FPGA Editor to see if this resolves the problem.
   b. If the failure does not exist, the problem is caused by a setup time violation.
   c. If this is the case, the delay values in the speed file may not be accurate since the timing violation is not displayed in the Timing Report.
5. Run post-route simulation to determine whether the problem is caused by the timing analysis or by a functional issue. Simulation may narrow the issue to a specific signal or instance.

Chapter 9

Cross Probing

Cross probing is useful for debugging timing violations. Cross probing allows you to:

- View the problem causing the timing violation.
- Easily cross probe to the:
  - Source and destination components
  - Data and clock paths
- View their respective delays.

Cross Probing Between FPGA Editor and Timing Analyzer

Cross probing is possible between FPGA Editor and Timing Analyzer. After implementation in ISE® Design Suite, the NCD file can be:

- Viewed as a standalone file in:
  - FPGA Editor
  - Timing Analyzer
- Launched from ISE Design Suite.

The paths and components are hyperlinked in the Timing Analyzer Report. Click a hyperlink to:

- Cross probe to FPGA Editor.
- Display the selected path or component.
  - Paths are highlighted.
  - Components are shown with a dot.

Alternatively, to view the path or component in FPGA Editor:

1. Open FPGA Editor if it is not already open.
2. Right click the path or component in the Timing Analyzer.
3. Click **`Show in FPGA Editor`** to view the path or component.

Cross Probing From Timing Analyzer to Technology Viewer

To cross-probe from Timing Analyzer to Technology Viewer:

1. Right-click the timing path in the Timing Analyzer Report.
2. Select **`Show in Technology Viewer`**.

   Technology Viewer launches and displays the path.

You cannot view components in the Technology Viewer.

Cross Probing From the PlanAhead tool to FPGA Editor

To cross probe from the PlanAhead™ design tool to FPGA Editor:

1. Select a timing path from Timing Results View or Device View.
2. Select **`Cross Probe to FPGA Editor`** from the popup menu.

   FPGA Editor opens with the selected path or instance highlighted.

You can also select individual logic instances to cross probe to FPGA Editor.

The PlanAhead tool can also help in debugging a timing problem. The tool offers the flexibility of viewing the placement of a failing path in the implemented design, as well as the schematic associated with a path.

Viewing Timing Paths in Device View

To view timing paths in Device View, load the design in the PlanAhead tool.

- If the design was implemented in the PlanAhead tool, the placements are already available.
- If the design was *not* implemented in the PlanAhead tool, import the placement results using **File > Import Placement**.

Review timing paths in Device View when you select a path row or rows in the Timing Results view. The path is highlighted in Device View. You can select multiple paths. All instances found in the path are selected and highlighted.

Viewing Timing Paths in Schematic View

To view timing paths in Schematic View:

1. Right click the timing path.
2. Select **Schematic** in the dropdown menu.

   The Schematic window opens showing the relevant path.

For more information, see *Analyzing Implementation Results* in the *PlanAhead User Guide (UG632)*, cited in Appendix A, Additional Resources.

Using Cross Probing During Debugging

You can cross probe the following to FPGA Editor from the report in XML format.

- Source and destination components
- Clock and data paths
- Individual components in the clock and data paths
- Nets in the clock and data paths

Viewing the Data Path

Click **Data Path** in the Timing Report to highlight the data path in FPGA Editor. FPGA Editor gives a quick view of the routings and logic involved in the data path.

This can be useful when trying to understand the reason for a high data path delay. For example, you can check to see if floorplanning will help the implementation tool meet timing.

Tracing Through the Clock Networks

You can trace through the clock nets from the source or destination component back to the origin of the clocks. This can be useful to:

- Investigate a high clock skew.
- View a gated clock to see if it can be improved in terms of delay or location.
- Check the BUFG location chosen by the tool for a clock.

To trace through the clock networks:

1. Cross probe the clock net from Timing Analyzer, or search for the net from FPGA Editor.
2. Right click the net.
3. Open **`Properties`**.

   The properties dialog box displays:

   - A list of the destination pins
   - One source pin to which the clock net is connected
4. Click **`Go To`** to select and zoom into the output pin.

   The component which generates the clock is given focus in FPGA Editor.

You may need to continue tracing backward until you find the origin of the clock net. For example, if the clock net comes from a BUFG, once the BUFG is brought into focus, you can highlight the input net to the BUFG and work backward with the same steps.

Viewing the Detailed Path

The Timing Report shows:

- Detailed paths for data and clock paths
- The full name of each net and component along each path

Each net or component can be cross probed directly in FPGA Editor. This allows you to view:

- The individual logic along a data or clock path
- How a net on the path is fanned out to other components

Showing the Delays

To obtain design delays, use FPGA Editor with the post PAR netlist opened.

Path Delays

For a *path*, click the destination pin to show the delay for the net.

- Click **`Delay`** to view the delay.

  OR
- Highlight the source pin and the destination pin at the same time.

Component Delays

For a *component*, highlight the input and output pins.

Click **`Delay`** to show a pin to pin component delay.

Understanding the BELs

To view the configuration of a component in FPGA Editor:

1. Select the component.
2. Double click the component.

   The Block window opens, showing the inner details of the component.
3. Click **F=** in the FPGA Editor to show the attributes of the component.

Slice Attributes Example

```
Name: demodata<3>
Config : A6LUT:#LUT:O6=((~A6*A5)+(A6*A4)) AFF:#FF AFFINIT:INIT0 AFFMUX:O6 AFFSR:SRLOW
B6LUT:#LUT:O6=((~A5*A6)+(A5*A4)) BFF:#FF BFFINIT:INIT0 BFFMUX:O6 BFFSR:SRLOW
C6LUT:#LUT:O6=((~A5*A3)+(A5*A6)) CFF:#FF CFFINIT:INIT0 CFFMUX:O6 CFFSR:SRLOW CLKINV:CLK
D6LUT:#LUT:O6=((~A6*A4)+(A6*A5)) DFF:#FF DFFINIT:INIT0 DFFMUX:O6 DFFSR:SRLOW SYNC_ATTR:SYNC
A6LUT: ((~A6*A5)+(A6*A4))
B6LUT: ((~A5*A6)+(A5*A4))
```

LUT Equation

The LUT equations show the sum of the products of the LUT inputs. Take the example of the above equations:

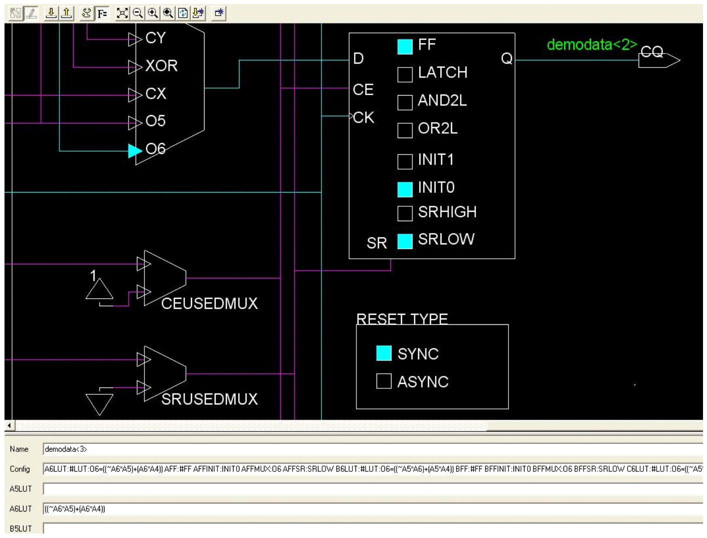
`A6LUT: ((~A6*A5)+(A6*A4))` means `(NOT (A6) and A5) OR (A6 and A4).`

These are also given in the Config string as:

`A6LUT:#LUT:O6=((~A6*A5)+(A6*A4))`

where

`O6` is the output of the LUT



```
Name    demodata<3>
Config  A6LUT:#LUT:O6=((~A6*A5)+(A6*A4)) AFF:#FF AFFINIT:INIT0 AFFMUX:O6 AFFSR:SRLOW B6LUT:#LUT:O6=((~A5*A6)+(A5*A4)) BFF:#FF BFFINIT:INIT0 BFFMUX:O6 BFFSR:SRLOW C6LUT:#LUT:O6=((~A5
A5LUT
A6LUT   ((~A6*A5)+(A6*A4))
B5LUT
```

Figure 9-1: **LUT Equation**

INIT Strings

The init strings are INIT values or the initial values of certain components such as:

- Flip flops
- RAMs
- Shift registers

The INIT values are either INIT0 or INIT1. These are basically the state of the flip flops immediately after configuration of the device.

This is also given as part of the Config string, for example:

`DFF:#FF DFFINIT:INIT0`

where

- The `D` flip flop has an initial state of `0` after Global Set Reset (GSR).

See the above example in which the flip flop with CQ output **`demodata<2>`** has INIT0.

Attribute Boxes

Attribute boxes are displayed in the Block window. They show an attribute of the component. For example, the RESET TYPE attribute box shows the SYNC and ASYNC options.

Another example is the phase shift of the DCM or MMCM. This is commonly checked in case the clock arrival values on Timing Report shows unexpected values.

This is useful for checking the attributes of a BEL such as:

- LUT equations
- DSP48 attributes
- PLL attributes

In situations such as the following, you can check to see how a slice has been configured:

- Whether route thru LUTs have been used
- Whether a MUX was used
- Whether a flip flop had the reset connected

Appendix A

Additional Resources

Xilinx® Resources

- **Device User Guides**:
  http://www.xilinx.com/support/documentation/user_guides.htm
- **Glossary of Terms**: http://www.xilinx.com/company/terms.htm
- ***ISE Design Suite: Installation and Licensing Guide*** (UG798):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/iil.pdf
- ***ISE Design Suite: Release Notes Guide*** (UG631):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/irn.pdf
- **Product Support and Documentation:** http://www.xilinx.com/support

ISE Documentation

- **ISE Design Suite Documentation**:
  http://www.xilinx.com/support/documentation/dt_ise14-3.htm
- ***Command Line Tools User Guide*** (UG628):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/devref.pdf
- ***Synthesis and Simulation Design Guide*** (UG626):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/sim.pdf
- ***XST User Guide for Virtex-6, Spartan®-6, and 7 Series Devices*** (UG687):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/xst_v6s6.pdf
- ***Constraints Guide*** (UG625):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/cgd.pdf
- ***Command Line Tools User Guide*** (UG628):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/devref.pdf
- **Libraries Guides**:
  http://www.xilinx.com/support/documentation/dt_ise14-3_librariesguides.htm

SmartXplorer Documentation

- *Timing Closure Exploration Tools with SmartXplorer and PlanAhead Tools* (White Paper 287), http://www.xilinx.com/support/documentation/white_papers/wp287.pdf
- ***SmartXplorer for Command Line Users*** (UG688):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug688.pdf
- ***SmartXplorer for Project Navigator Users*** (UG689):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug689.pdf

PlanAhead tool Documentation

- ***Floorplanning Methodology Guide*** (UG633):
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/Floorplanning_Methodolgy_Guide.pdf
- ***PlanAhead User Guide*** (UG632):
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/PlanAhead_UserGuide.pdf