7

Defining Design Constraints

In addition to specifying the design environment, you must set design constraints before compiling the design. There are two categories of design constraints:

- Design rule constraints
- Design optimization constraints

Design rule constraints are supplied in the technology library you specify. They are referred to as the *implicit* design rules. These rules are established by the library vendor, and, for the proper functioning of the fabricated circuit, they must not be violated. You can, however, specify stricter design rules if appropriate. The rules you specify are referred to as the *explicit* design rules.

Design optimization constraints define timing and area optimization goals for Design Compiler. These constraints are user-specified. Design Compiler optimizes the synthesis of the design, in accordance with these constraints, but not at the expense of the design rule constraints. That is, Design Compiler attempts never to violate the higher-priority design rules.

Note:

In this chapter, setting explicit design rules and optimization constraints is discussed without reference to the particular compile strategy you choose. But the compile strategy you choose does influence your constraint settings.

This chapter contains the following sections:

- Setting Design Rule Constraints
- Setting Optimization Constraints
- Verifying the Precompiled Design

The task of setting timing constraints can be complicated (especially setting the timing exceptions) and includes the following tasks:

- Defining a Clock
- Specifying I/O Timing Requirements
- Specifying Combinational Path Delay Requirements
- Specifying Timing Exceptions

Setting Design Rule Constraints

This section discusses the most commonly specified design rule constraints:

- Transition time
- Fanout load
- Capacitance

Design Compiler also supports cell degradation and connection class constraints. For information about these constraints, see the *Design Compiler Reference Manual: Constraints and Timing*.

Design Compiler uses attributes assigned to the design's objects to represent design rule constraints. Table 7-1 provides the attribute name that corresponds to each design rule constraint.

Table 7-1 Design Rule Attributes

Design rule constraint	**Attribute name**
Transition time	max_transition
Fanout load	max_fanout
Capacitance	max_capacitance min_capacitance
Cell degradation	cell_degradation
Connection class	connection_class

Design rule constraints are attributes specified in the technology library and, optionally, specified by you explicitly.

If a technology library defines these attributes, Design Compiler implicitly applies them to any design using that library when it compiles the design or creates a constraint report. You cannot remove the design rule attributes defined in the technology library, because they are requirements for the technology, but you can make them more restrictive to suit your design.

If both implicit and explicit design rule constraints apply to a design or a net, the more restrictive value takes precedence.

Setting Transition Time Constraints

The transition time of a net is the time required for its driving pin to change logic values. This transition time is based on the technology library data. For the nonlinear delay model (NLDM), output transition time is a function of input transition and output load.

Design Compiler and Library Compiler model transition time restrictions by associating a `max_transition` attribute with each output pin on a cell. During optimization, Design Compiler attempts to make the transition time of each net less than the value of the `max_transition` attribute.

To change the maximum transition time restriction specified in a technology library, use the `set_max_transition` command. This command sets a maximum transition time for the nets attached to the identified ports or to all the nets in a design by setting the `max_transition` attribute on the named objects.

For example, to set a maximum transition time of 3.2 on all nets in the design adder, enter one of the following commands (depending on your shell mode):

```
dc_shell> set_max_transition 3.2 find(design,adder)
```

```
dc_shell-t> set_max_transition 3.2 [get_designs adder]
```

To undo a `set_max_transition` command, use the `remove_attribute` command. For example, enter one of the following commands (depending on your shell mode):

```
dc_shell>remove_attribute find(design,adder) \
         max_transition
```

```
dc_shell-t>remove_attribute [get_designs adder] \
         max_transition
```

Setting Fanout Load Constraints

The maximum fanout load for a net is the maximum number of loads the net can drive.

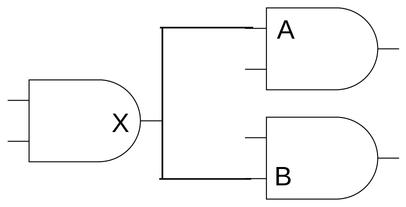
Design Compiler and Library Compiler model fanout restrictions by associating a `fanout_load` attribute with each input pin and a `max_fanout` attribute with each output (driving) pin on a cell.

The fanout load value does not represent capacitance; it represents the weighted numerical contribution to the total fanout load. The fanout load imposed by an input pin is not necessarily 1.0. Library developers can assign higher fanout load values to model internal cell fanout effects.

Design Compiler calculates the fanout of a driving pin by adding the `fanout_load` values of all inputs driven by that pin. To determine whether the pin meets the maximum fanout load restriction, Design Compiler compares the calculated fanout load value with the pin's `max_fanout` value.

Figure 7-1 shows a small circuit in which pin X drives two loads, pin A and pin B. If pin A has a `fanout_load` value of 1.0 and pin B has a `fanout_load` value of 2.0, the total fanout load of pin X is 3.0. If pin X has a maximum fanout greater than 3.0, say 16.0, the pin meets the fanout constraints.

Figure 7-1 Fanout Constraint Example



During optimization, Design Compiler attempts to meet the fanout load restrictions for each driving pin. If a pin violates its fanout load restriction, Design Compiler tries to correct the problem (for example, by changing the drive strength of the component).

The technology library might specify default fanout constraints on the entire library or fanout constraints for specific pins in the library description of an individual cell.

To determine whether your technology library is modeled for fanout calculations, you can search for the `fanout_load` attribute on the cell input pins by entering one of the following commands (depending on your shell mode):

```
dc_shell> get_attribute find(pin, my_lib/*/*) fanout_load
```

```
dc_shell-t> get_attribute [get_pins my_lib/*/*] fanout_load
```

To set a more conservative fanout restriction than that specified in the technology library, use the `set_max_fanout` command on the design or on an input port. (Use the `set_fanout_load` command to set the expected fanout load value for output ports.)

The `set_max_fanout` command sets the maximum fanout load for the specified input ports or for all the nets in a design by setting the `max_fanout` attribute on the specified objects. For example, to set a `max_fanout` requirement of 16 on all nets in the design adder, enter one of the following commands (depending on your shell mode):

```
dc_shell> set_max_fanout 16 find(design, adder)
```

```
dc_shell-t> set_max_fanout 16 [get_designs adder]
```

If you use the `set_max_fanout` command and a library `max_fanout` attribute exists, Design Compiler tries to meet the smaller (more restrictive) fanout limit.

To undo a `set_max_fanout` command, use the `remove_attribute` command. For example, enter one of the following commands (depending on your shell mode):

```
dc_shell> remove_attribute find(design,adder) max_fanout
```

```
dc_shell-t> remove_attribute [get_designs adder] max_fanout
```

Setting Capacitance Constraints

The transition time constraints do not provide a direct way to control the actual capacitance of nets. If you need to control capacitance directly, use the `set_max_capacitance` command to set the maximum capacitance constraint. This constraint is completely independent, so you can use it in addition to the transition time constraints.

Design Compiler and Library Compiler model capacitance restrictions by associating the `max_capacitance` attribute with the output ports or pins of a cell. Design Compiler calculates the capacitance on the output net by adding the wire capacitance of the net to the capacitance of the pins attached to the net. To determine whether the net meets the capacitance constraint, Design Compiler compares the calculated capacitance value with the output pin's `max_capacitance` value.

For example, to set a maximum capacitance of 3 for all nets in the design adder, enter one of the following commands (depending on your shell mode):

```
dc_shell> set_max_capacitance 3 find(design,adder)
```

```
dc_shell-t> set_max_capacitance 3 [get_designs adder]
```

To undo a `set_max_capacitance` command, use the `remove_attribute` command. For example, enter one of the following commands (depending on your shell mode):

```
dc_shell> remove_attribute find(design,adder) \
          max_capacitance

dc_shell-t> remove_attribute [get_designs adder] \
          max_capacitance
```

You can also use the `set_min_capacitance` command to define the minimum capacitance for input ports or pins. Design Compiler attempts to ensure that the load seen at the input port does not fall below the specified capacitance value, but it does not specifically optimize for this constraint.

Setting Optimization Constraints

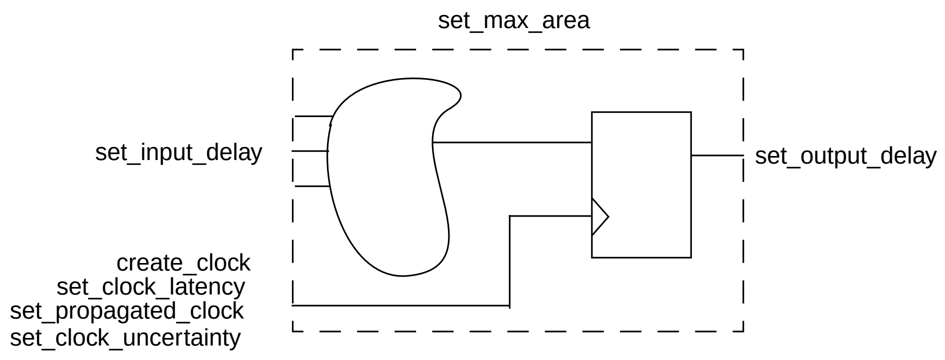
This section discusses the most commonly specified optimization constraints:

- Timing constraints
- Area constraints

Design Compiler also supports power constraints. For information about power constraints, see the *Power Compiler Reference Manual*.

Figure 7-2 illustrates some of the common commands used to define the optimization constraints.

Figure 7-2 Commands Used to Define the Optimization Constraints for Sequential Blocks



Setting Timing Constraints

Timing constraints specify the required performance of the design. To set the timing constraints,

1. Define the clocks.
2. Specify the I/O timing requirements relative to the clocks.
3. Specify the combinational path delay requirements.
4. Specify the timing exceptions.

Table 7-2 lists the most commonly used commands for these steps.

Table 7-2 Commands to Set Timing Constraints

Command	Description
`create_clock`	Defines the period and waveform for the clock.

`set_clock_latency`
`set_propagated_clock`
`set_clock_uncertainty`
	Defines the clock delay.

Table 7-2 Commands to Set Timing Constraints (Continued)

Command	Description
`set_input_delay`	Defines the timing requirements for input ports relative to the clock period.
`set_output_delay`	Defines the timing requirements for output ports relative to the clock period.
`set_max_delay`	Defines maximum delay for combinational paths. (This is a timing exception command.)
`set_min_delay`	Defines minimum delay for combinational paths. (This is a timing exception command.)
`set_false_path`	Specifies false paths. (This is a timing exception command.)
`set_multicycle_path`	Specifies multicycle paths. (This is a timing exception command.)

The following sections describe these steps in more detail.

Defining a Clock

For synchronous designs, the clock period is the most important constraint because it constrains all register-to-register paths in the design.

Defining the Period and Waveform for the Clock. Use the `create_clock` command to define the period (`-period` option) and waveform (`-waveform` option) for the clock. If you do not specify the clock waveform, Design Compiler uses a 50 percent duty cycle.

Use the `create_clock` command on a pin or a port. For example, to specify a 25-megahertz clock on port clk with a 50 percent duty cycle, enter

```
dc_shell> create_clock clk -period 40
```

When your design contains multiple clocks, pay close attention to the common base period of the clocks. The common base period is the least common multiple of all the clock periods. For example, if you have clock periods of 10, 15, and 20, the common base period is 60.

Define your clocks so that the common base period is a small integer multiple of each of the clock periods. The common base period requirement is qualitative; no hard limit exists. If the base period is more than 10 times larger than the smallest period, however, long runtimes and greater memory requirements can result.

As an extreme case, if you have a register-to-register path where one register has a period of 10 and the other has a period of 10.1, the common base period is 1010.0. The timing analyzer calculates the setup requirement for this path by expanding both clocks to the common base period and determining the tightest single-cycle relationship for setup. Internally, for extreme cases such as this, the timing analyzer only approximates the setup requirement because the paths are not really synchronous.

You can work around this problem by specifying a clock period without a decimal point and adjusting the clock period by inserting clock uncertainty.

```
dc_shell> create_clock -period 10 clk1
dc_shell> create_clock -period 10 clk2
dc_shell> set_clock_uncertainty -setup 0.1 clk2
```

Use the `report_clock` command to show information about all clock sources in your design.

Use the `remove_clock` command to remove a clock definition.

Creating a Virtual Clock. In some cases, a system clock might not exist in a block. You can use the `create_clock -name` command to create a virtual clock for modeling clock signals present in the system but not in the block. By creating a virtual clock, you can represent delays that are relative to clocks outside the block.

```
dc_shell> create_clock -period 30 -waveform {10 25} \
          -name sys_clk
```

Specifying Clock Network Delay. By default, Design Compiler assumes that clock networks have no delay (ideal clocks). Use the `set_clock_latency` and `set_clock_uncertainty` commands to specify timing information about the clock network delay. You can use these commands to specify either estimated or actual delay information.

Use the `set_propagated_clock` command to specify that you want the clock latency to propagate through the clock network. For example,

```
dc_shell> set_propagated_clock clk
```

Use the `-setup` or `-hold` options of the `set_clock_uncertainty` command to add some margin of error into the system to account for variances in the clock network resulting from layout. For example, on the 20-megahertz clock mentioned previously, to add a 0.2 margin on each side of the clock edge, enter

```
dc_shell> set_clock_uncertainty -setup 0.2 clk
dc_shell> set_clock_uncertainty -hold 0.2 clk
```

Use the `-skew` option of the `report_clock` command to show clock network skew information. Design Compiler uses the clock information when determining whether a path meets setup and hold requirements.

Specifying I/O Timing Requirements

If you do not assign timing requirements to an input port, Design Compiler responds as if the signal arrives at the input port at time 0. In most cases, input signals arrive at staggered times. Use the `set_input_delay` command to define the arrival times for input ports. You define the input delay constraint relative to the system clock and to the other inputs.

If you do not assign timing requirements to an output port, Design Compiler does not constrain any paths which end at an output port. Use the `set_output_delay` command to define the required output arrival time. You define the output delay constraint relative to the system clock.

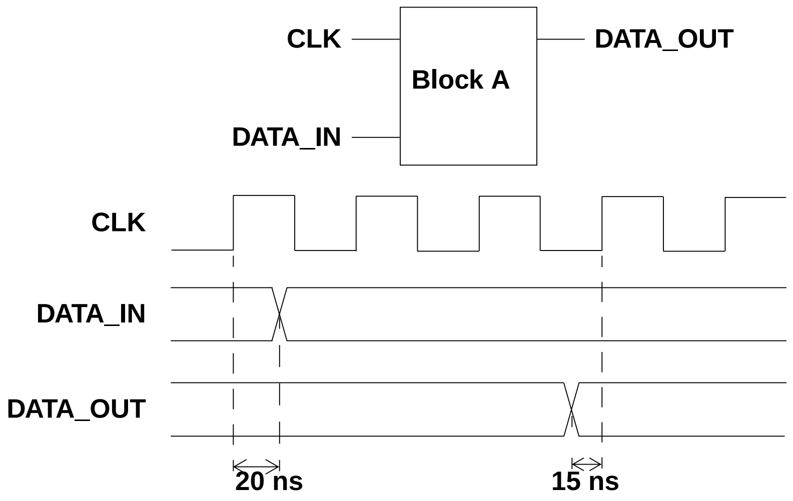
If an input or output port has multiple timing requirements (because of multiple paths), use the `-add_delay` option to specify the additional timing requirements.

Use the `report_port` command to list input or output delays associated with ports.

Use the `remove_input_delay` command to remove input delay constraints. Use the `remove_output_delay` command to remove output delay constraints.

Figure 7-3 shows the timing relationship between the delay and the active clock edge (the rising edge in this example).

Figure 7-3 Relationship Between Delay and Active Clock Edge



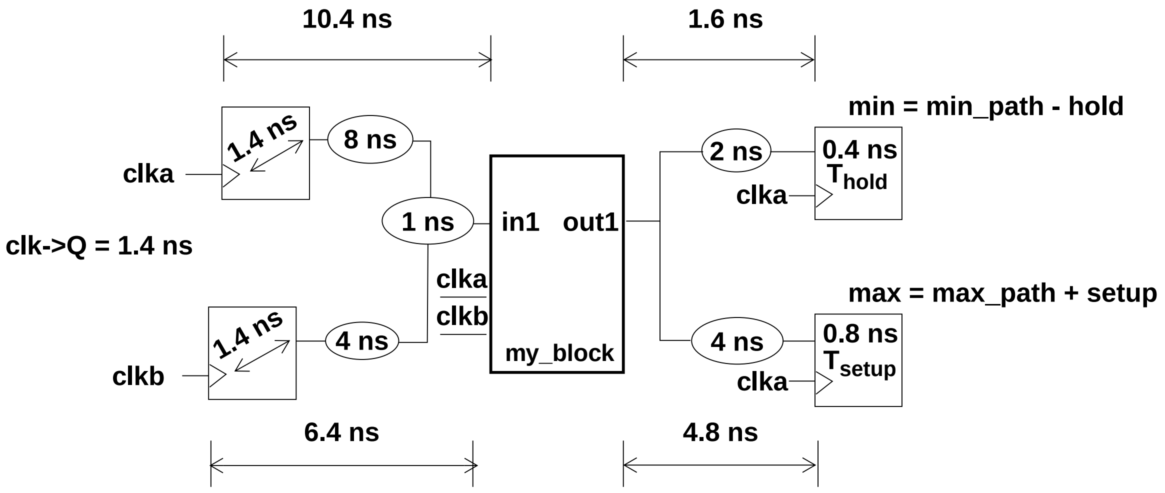
In the figure, block A has an input DATA_IN and an output DATA_OUT. From the waveform diagram, DATA_IN is stable 20 ns after the clock edge, and DATA_OUT needs to be available 15 ns before the clock edge.

After you set the clock constraint by using the `create_clock` command, use the `set_input_delay` and `set_output_delay` commands to specify these additional requirements. For example, enter

```
dc_shell> set_input_delay 20 -clock CLK DATA_IN
dc_shell> set_output_delay 15 -clock CLK DATA_OUT
```

Figure 7-4 illustrates the timing requirements for the constrained design block my_block. Example 7-1 shows the script used to specify these timing requirements.

Figure 7-4 Timing Requirements for my_block



Example 7-1 Timing Constraints for my_block

```
create_clock -period 20 -waveform {5 15} clka
create_clock -period 30 -waveform {10 25} clkb
set_input_delay 10.4 -clock clka in1
set_input_delay 6.4 -clock clkb -add_delay in1
set_output_delay 1.6 -clock clka -min out1
set_output_delay 4.8 -clock clka -max out1
```

Specifying Combinational Path Delay Requirements

For purely combinational delays that are not bounded by a clock period, use the `set_max_delay` and `set_min_delay` commands to define the maximum and minimum delays for the specified paths.

A common way to produce this type of asynchronous logic in HDL code is to use asynchronous sets or resets on latches and flip-flops. Because the reset signal crosses several blocks, constrain this signal at the top level.

For example, to specify a maximum delay of 5 on the RESET signal, enter

```
dc_shell> set_max_delay 5 -from RESET
```

To specify a minimum delay of 10 on the path from IN1 to OUT1, enter

```
dc_shell> set_min_delay 10 -from IN1 -to OUT1
```

Use the `report_timing_requirements` command to list the minimum delay and maximum delay requirements for your design.

Specifying Timing Exceptions

Timing exceptions define timing relationships that override the default single-cycle timing relationship for one or more timing paths. Use timing exceptions to constrain or disable asynchronous paths or paths that do not follow the default single-cycle behavior.

Note:

Specifying numerous timing exceptions can increase the compile runtime. Nevertheless, some designs can require many timing exceptions.

Design Compiler recognizes only timing exceptions that have valid reference points.

- The valid startpoints in a design are the primary input ports and the clock pins of sequential cells.
- The valid endpoints are the primary output ports of a design and the data pins of sequential cells.

Design Compiler does not generate a warning message if you specify invalid reference points. You must use the `-ignored` option of the `report_timing_requirements` command to find timing exceptions ignored by Design Compiler.

You can specify the following conditions by using timing exception commands:

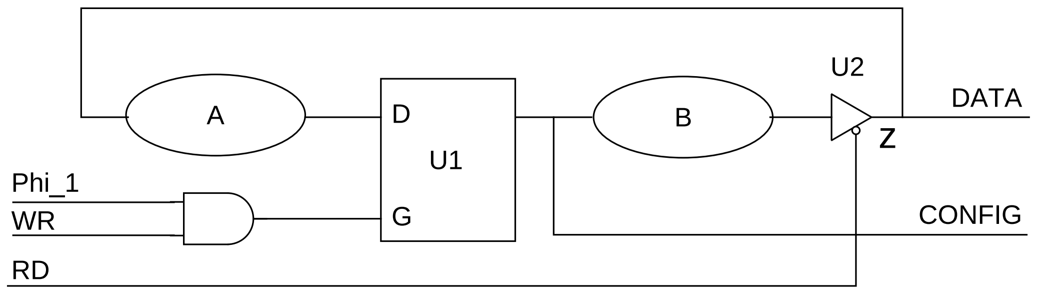
- False paths (`set_false_path`)
- Minimum delay requirements (`set_min_delay`)
- Maximum delay requirements (`set_max_delay`)
- Multicycle paths (`set_multicycle_path`)

Use the `report_timing_requirements` command to list the timing exceptions in your design.

Specifying False Paths. Design Compiler does not report false paths in the timing report or consider them during timing optimization. Use the `set_false_path` command to specify a false path. Use this command to ignore paths that are not timing critical, that can mask other paths that must be considered during optimization, or that never occur in normal operation.

For example, Figure 7-5 shows a configuration register that can be written and read from a bidirectional bus (DATA) in a chip.

Figure 7-5 Configuration Register



The circuit has these timing paths:

1. DATA to U1/D
2. RD to DATA
3. U1/G to CONFIG (with possible time borrowing at U1/D)
4. U1/G to DATA (with possible time borrowing at U1/D)
5. U1/G to U1/D (through DATA, with possible time borrowing)

The first four paths are valid paths. The fifth path (U1/G to U1/D) is a functional false path because normal operation never requires simultaneous writing and reading of the configuration register. In this design, you can disable the false path by using this command:

```
dc_shell> set_false_path -from U1/G -to U1/D
```

To undo a `set_false_path` command, use the `reset_path` command with similar options. For example, enter

```
dc_shell> set_false_path -setup -from IN2 -to FF12/D
dc_shell> reset_path -setup -from IN2 -to FF12/D
```

Creating a false path differs from disabling a timing arc. Disabling a timing arc represents a break in the path. The disabled timing arc permanently disables timing through all affected paths. Specifying a path as false does not break the path; it just prevents the path from being considered for timing or optimization.

Specifying Minimum and Maximum Delay Requirements. You can use the `set_min_delay` and `set_max_delay` commands, described earlier in this chapter, to specify path delay requirements that are more conservative than those derived by Design Compiler based on the clock timing.

To undo a `set_min_delay` or `set_max_delay` command, use the `reset_path` command with similar options.

Register-to-Register Paths. Design Compiler uses the following equations to derive constraints for minimum and maximum path delays on register-to-register paths:

$$\text{min\_delay} = (T_{capture} - T_{launch}) + \text{hold}$$
$$\text{max\_delay} = (T_{capture} - T_{launch}) - \text{setup}$$

You can override the derived path delay ($T_{capture} – T_{launch}$) by using the `set_min_delay` and `set_max_delay` commands.

For example, assume that you have a path launched from a register at time 20 that arrives at a register where the next active edge of the clock occurs at time 35.

```
dc_shell> create_clock -period 40 waveform {0 20} clk1
dc_shell> create_clock -period 40 -waveform {15 35} clk2
```

Design Compiler automatically derives a maximum path delay constraint of (35 – 20) – (library setup time of register at endpoint). To specify a maximum path delay of 10, enter

```
dc_shell> set_max_delay 10 -from reg1 -to reg2
```

Design Compiler calculates the maximum path delay constraint as 10 – (library setup time of register at endpoint), which overrides the original derived maximum path delay constraint.

Register-to-Port Paths. Design Compiler uses the following equations to derive constraints for minimum and maximum path delays on register-to-port paths:

```
min_delay = period - output_delay
max_delay = period - output_delay
```

If you use the `set_min_delay` or `set_max_delay` commands, the value specified in these commands replaces the period value in the constraint calculation. For example, assume you have a design with a clock period of 20. Output OUTPORTA has an output delay of 5.

```
dc_shell> create_clock -period 20 CLK
dc_shell> set_output_delay 5 -clock CLK OUTPORTA
```

Design Compiler automatically derives a maximum path delay constraint of 15 (20 – 5). To specify that you want a maximum path delay of 10, enter

```
dc_shell> set_max_delay 10 -to OUTPORTA
```

Design Compiler calculates the maximum path delay constraint as 5 (10 – 5), which overrides the original derived maximum path delay constraint.

Asynchronous Paths. You can also use the `set_max_delay` and `set_min_delay` commands to constrain asynchronous paths across different frequency domains. Table 7-3 shows dcsh and dctcl examples for constraining asynchronous paths.

Table 7-3 Examples for Constraining Asynchronous Paths

dcsh Example	dctcl Example
`dc_shell>` **set_max_delay 17.1 \\ -from find(clock, clk1) \\ -to find(clock, clk2)**	`dc_shell-t>` **set_max_delay 17.1 \\ -from [get_clocks clk1] \\ -to [get_clocks clk2]**
`dc_shell>` **set_max_delay 23.5 \\ -from find(clock, clk2) \\ -to find(clock, clk3)**	`dc_shell-t>` **set_max_delay 23.5 \\ -from [get_clocks clk2] \\ -to [get_clocks clk3]**
`dc_shell>` **set_max_delay 31.6 \\ -from find(clock, clk3) \\ -to find(clock, clk1)**	`dc_shell-t>` **set_max_delay 31.6 \\ -from [get_clocks clk3] \\ -to [get_clocks clk1]**

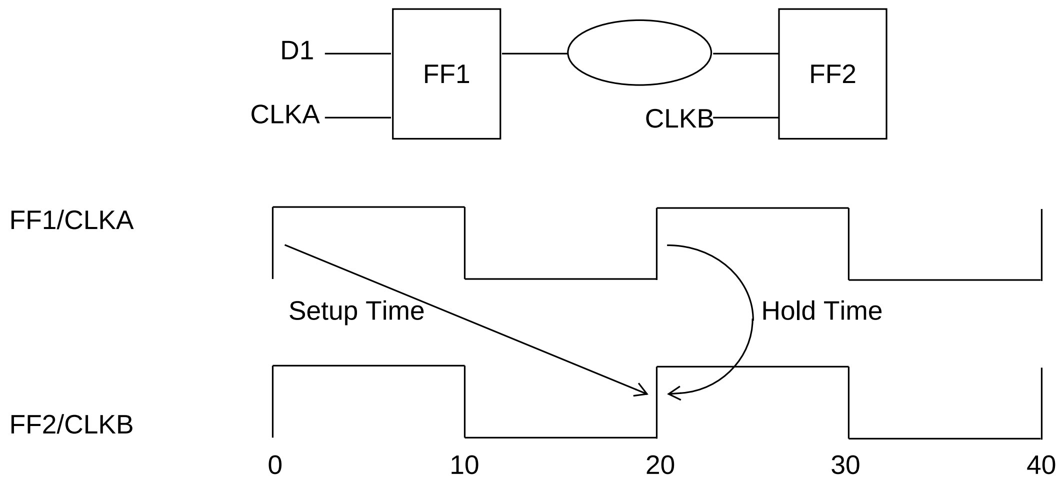
Setting Multicycle Paths. The multicycle path condition is appropriate when the path in question is longer than a single cycle or when data is not expected within a single cycle. Use the `set_multicycle_path` command to specify the number of clock cycles Design Compiler should use to determine when data is required at a particular endpoint.

You can specify this cycle multiplier for setup or hold checks. If you do not specify the `-setup` or `-hold` option with the `set_multicycle_path` command, Design Compiler applies the multiplier value only to setup checks.

By default, setup is checked at the next active edge of the clock at the endpoint after the data is launched from the startpoint (default multiplier of 1). Hold data is launched one clock cycle after the setup data but checked at the edge used for setup (default multiplier of zero).

Figure 7-6 shows the timing relationship of setup and hold times.

Figure 7-6 Setup and Hold Timing



The timing path starts at the clock pin of FF1 (rising edge of CLKA) and ends at the data pin of FF2. Assuming that the flip-flops are rising-edge-triggered, the setup data is launched at time 0 and checked 20 time units later at the next active edge of CLKB at FF2. Hold data is launched one (CLKA) clock cycle (time 20) and checked at the same edge used for setup checking (time 20).

The `-setup` option of the `set_multicycle_path` command moves the edge used for setup checking to before or after the default edge. For the example shown in Figure 7-6,

- A setup multiplier of zero means that Design Compiler uses the edge at time zero for checking

- A setup multiplier of 2 means that Design Compiler uses the edge at time 40 for checking

The `-hold` option of the `set_multicycle_path` command launches the hold data at the edge before or after the default edge, but Design Compiler still checks the hold data at the edge used for checking setup. As shown in Figure 7-6 (assuming a default setup multiplier),

- A hold multiplier of 1 means that the hold data is launched from CLKA at time 40 and checked at CLKB at time 20
- A hold multiplier of -1 means that the hold data is launched from CLKA at time 0 and checked at CLKB at time 20

To undo a `set_multicycle_path` command, use the `reset_path` command with similar options.

Using Multiple Timing Exception Commands. A specific timing exception command refers to a single timing path. A general timing exception command refers to more than one timing path. If you execute more than one instance of a given timing exception command, the more specific commands override the more general ones.

The following rules define the order of precedence for a given timing exception command:

- The highest precedence occurs when you define a timing exception from one pin to another pin.
- A command using only the `-from` option has a higher priority than a command using only the `-to` option.

- For clocks used in timing exception commands, if both `-from` and `-to` are defined, they override commands that share the same path defined by either the `-from` or the `-to` option.

This list details the order of precedence (highest at the top) defined by these precedence rules:

1. *command* -from *pin* -to *pin*
2. *command* -from *clock* -to *pin*
3. *command* -from *pin* -to *clock*
4. *command* -from *pin*
5. *command* -to *pin*
6. *command* -from *clock* -to *clock*
7. *command* -from *clock*
8. *command* -to *clock*

For example, in the following command sequence, paths from A to B are treated as two-cycle paths because specific commands override general commands:

```
dc_shell> set_multicycle_path 2 -from A -to B
dc_shell> set_multicycle_path 3 -from A
```

The following rules summarize the interaction of the timing exception commands:

- General `set_false_path` commands override specific `set_multicycle_path` commands.
- General `set_max_delay` commands override specific `set_multicycle_path` commands.

- Specific `set_false_path` commands override specific `set_max_delay` or `set_min_delay` commands.
- Specific `set_max_delay` commands override specific `set_multicycle_path` commands.

Setting Area Constraints

The `set_max_area` command specifies the maximum area for the current design by placing a `max_area` attribute on the current design. Specify the area in the same units used for area in the technology library.

For example, to set the maximum area to 100, enter

```
dc_shell> set_max_area 100
```

Design area consists of the areas of each component and net. The following components are ignored when Design Compiler calculates design area:

- Unknown components
- Components with unknown areas
- Technology-independent generic cells

Cell (component) area is technology dependent; Design Compiler obtains this information from the technology library.

When you specify both timing and area constraints, Design Compiler attempts to meet timing goals before area goals. To prioritize area constraints over total negative slack (but not over worst negative slack), use the `-ignore_tns` option when you specify the area constraint.

```
dc_shell> set_max_area -ignore_tns 100
```

To optimize a small area, regardless of timing, remove all constraints except for maximum area. You can use the `remove_constraint` command to remove constraints from your design. Be aware that this command removes *all* optimization constraints from your design.

Verifying the Precompiled Design

Before compiling your design, verify that

- The design is consistent

  Use the `check_design` command to verify design consistency. For information about the `check_design` command, see "Checking for Design Consistency" on page 9-2.

- The attributes and constraints are correct

  Design Compiler provides many commands for reporting the attributes and constraints. For information about these commands, see "Analyzing Design Problems" on page 9-8 and "Analyzing Timing Problems" on page 9-9.