Fusion Compiler™ Timing Analysis User Guide

Version T-2022.03-SP4, September 2022

SYNOPSYS®

Contents

About This User Guide

The Synopsys Fusion Compiler tool provides a complete netlist-to-GDSII design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementations throughout the design flow.

This guide describes the Fusion Compiler implementation and integration flow. For more information about the Fusion Compiler tool, see the following companion volumes:

- *Library Manager User Guide*
- *Fusion Compiler Design Planning User Guide*
- *Fusion Compiler Data Model User Guide*
- *Fusion Compiler Implementation User Guide*
- *Fusion Compiler Graphical User Interface User Guide*

This user guide is for design engineers who use the Fusion Compiler tool to implement designs.

To use the Fusion Compiler tool, you need to be skilled in physical design and synthesis and be familiar with the following:

- Physical design principles
- The Linux or UNIX operating system
- The tool command language (Tcl)

This preface includes the following sections:

- New in This Release
- Related Products, Publications, and Trademarks
- Conventions
- Customer Support

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Fusion Compiler Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the Fusion Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- IC Validator
- PrimeTime® Suite

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing =` **`true`** `| false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.

Convention	Description
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

https://solvnetplus.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

Feedback

1

Defining Modes, Corners, and Scenarios

A block might operate under several different conditions, such as different temperatures and voltages, and might operate in several different functional modes. For timing analysis, each set of conditions is represented by a *corner* and each functional mode is represented by a *mode*. A *scenario* is a combination of a corner and mode used to perform timing analysis and optimization.

Before you start working with a block, you must define the modes, corners, and scenarios that are used for the block, and define the constraints associated with these modes, corners, and scenarios. These tasks are described in the following topics:

- Creating and Removing Modes
- Querying Modes
- Creating and Removing Corners
- Querying Corners
- Creating a Scenario
- Setting the Active Analysis Types for a Scenario
- Querying Scenarios
- Removing Duplicate Scenarios, Modes, and Corners
- Fusion Compiler Timing Constraints
- Setting Timing Constraints
- Converting an SDC File for Fusion Compiler Timing Analysis
- Importing Timing Constraints From the Design Compiler or IC Compiler Tool
- Removing SDC Constraints

Creating and Removing Modes

To create a functional mode, use the `create_mode` command. Each mode must have a unique name.

For example, to create a mode named mode1, use the following command:

```
fc_shell> create_mode mode1
```

To change the current mode to a specific mode, use the `current_mode` command, as shown in the following example:

```
fc_shell> current_mode mode1
```

To remove modes for the current block, use the `remove_modes` command. To remove all modes, use the `-all` option. To remove specific modes, specify the modes using a Tcl list or collection. When you remove a mode, all scenarios associated with that mode are also removed.

For example, to remove the modes named mode1 and mode2, use the following command:

```
fc_shell> remove_modes {mode1 mode2}
```

Querying Modes

To obtain

- The current mode, use the `current_mode` command without any arguments.

  For example,

  ```
  fc_shell> current_mode
  {"mode1"}
  ```

- A list of modes defined for a block, use the `get_modes` command.

  By default, the command returns a collection that contains all modes defined for the current block (similar to the `all_modes` command). To get the modes for another block, use the `-design` option.

  You can restrict the set of modes returned by the command by filtering the modes based on attribute values (`-filter` option), associated corners (`-of_objects` option), or name patterns.

- Detailed information about one or more modes of the current block, use the `report_modes` command.

  The report includes the associated scenarios and their active analysis types, for each mode.

  The following example reports the detailed information for all modes for the current block:

  ```
  fc_shell> report_modes
  ```

  The following example reports the detailed information of the modes named mode1 and mode2:

  ```
  fc_shell> report_modes {mode1 mode2}
  ```

See Also

- Setting the Active Analysis Types for a Scenario

Creating and Removing Corners

To create a timing corner, use the `create_corner` command. Each corner must have a unique name.

For example, to create a corner named corner1, use the following command:

```
fc_shell> create_corner corner1
```

To change the current corner to a specific corner, use the `current_corner` command, as shown in the following example:

```
fc_shell> current_corner  corner1
```

To remove corners for the current block, use the `remove_corners` command. To remove all corners, use the `-all` option. To remove specific corners, specify the corners using a Tcl list or collection. When you remove a corner, all scenarios associated with that corner are also removed.

For example, to remove the corners named corner1 and corner2, use the following command:

```
fc_shell> remove_corners {corner1 corner2}
```

Querying Corners

To obtain

- The current corner, use the `current_corner` command without any arguments.

  For example,

  ```
  fc_shell> current_corner
  {"corner1"}
  ```

- The list of corners defined for a block, use the `get_corners` command.

  By default, the command returns a collection that contains all corners defined for the current block (similar to the `all_corners` command). To get the corners for another block, use the `-design` option.

  You can restrict the set of corners returned by the command by filtering the corners based on attribute values (`-filter` option), associated corners (`-of_objects` option), or name patterns.

- Detailed information about one or more corners of the current block, use the `report_corners` command.

  The report includes the associated scenarios and their active analysis types, for each corner.

  The following example reports the detailed information for all corners of the current block:

  ```
  fc_shell> report_corners
  ```

  The following example reports the detailed information for the corners named corner1 and corner2:

  ```
  fc_shell> report_corners {corner1 corner2}
  ```

Creating a Scenario

To create a timing scenario, use the `create_scenario` command. Each scenario must have a unique name. By default, when you create a scenario, it combines the current mode with the current corner. To specify the mode, use the `-mode` option. To specify the corner, use the `-corner` option.

For example, to create a scenario named scenario1 that combines mode1 with corner1, use the following command:

```
fc_shell> create_scenario -mode mode1 -corner corner1 -name \
   scenario1
```

When you create a scenario, by default, it is active and it becomes the current scenario. In addition, its mode becomes the current mode and its corner becomes the current corner.

To change the active status of a scenario, use the `set_scenario_status -active true|false` command. To modify the active analysis types for the scenario, use the `set_scenario_status` command.

To remove scenarios for the current block, use the `remove_scenarios` command. To remove all scenarios, use the `-all` option. To remove specific scenarios, specify the corners using a Tcl list or collection.

For example, to remove the scenarios named scenario1 and scenario,2 use the following command:

```
fc_shell> remove_scenarios {scenario1 scenario2}
```

See Also

- Setting the Active Analysis Types for a Scenario

Setting the Active Analysis Types for a Scenario

By default, a scenario is active. To modify the active analysis types for one or more scenarios, use the `set_scenario_status` command as shown in the following table.

Table 1 Analysis Options of the set_scenario_status Command

Analysis type	Command option
All	`-all`
None	`-none`
Setup time	`-setup true | false`
Hold time	`-hold true | false`
Leakage Power	`-leakage_power true | false`
Dynamic Power	`-dynamic_power true | false`
Maximum transition time	`-max_transition true | false`
Maximum capacitance	`-max_capacitance true | false`
Minimum capacitance	`-min_capacitance true | false`
Cell electromigration	`-cell_em true | false`
Signal electromigration	`-signal_em true | false`

You can temporarily disable the individual analysis settings by using the `-active false` option. When you use this option, all analysis is disabled for the specified scenarios; however, the individual settings are retained in the design library. When you use the `-active true` option, the individual settings take effect.

Querying Scenarios

To obtain

- The current scenario, use the `current_scenario` command without any arguments.

  For example:

  ```
  fc_shell> current_scenario
  {"scenario1"}
  ```

- The scenarios defined for a block, use the `get_scenarios` command.

  By default, the command returns a collection that contains all scenarios defined for the current block (similar to the `all_scenarios` command).

  You can restrict the set of scenarios returned by the command by filtering the scenarios based on attribute values (`-filter` option), associated corners (`-corners` option), associated modes (`-modes` option), or name patterns.

- Detailed information about one or more scenarios of the current block, use the `report_scenarios` command.

  The report includes the scenario name, its associated mode, its associated corner, and its active analysis types, for each scenario.

  By default, the command reports all scenarios for the current block.

  To report

  - Specific scenarios, use the `-scenarios` option.
  - Scenarios associated with specific modes, use the `-modes` option.
  - Scenarios associated with specific corners, use the `-corners` option.

  The following example reports the detailed information for all scenarios of the current block:

  ```
  fc_shell> report_scenarios
  ```

  The following example reports the detailed information for the scenarios named scenario1 and scenario2:

  ```
  fc_shell> report_scenarios -scenarios {scenario1 scenario2}
  ```

Removing Duplicate Scenarios, Modes, and Corners

To improve runtime and capacity, you can remove duplicate scenarios, corners and modes by using the `remove_duplicate_timing_contexts` command. This command

- Removes scenarios that are duplicate and have the same scenario status.
- Reassigns the scenarios of the duplicate modes to a common mode and removes the duplicate modes.

  When reassigning scenarios, the tool creates new corners, if necessary.
- Reassigns the scenarios of the duplicate corners to a common scenario and removes the duplicate corners.

When determining duplicate modes, the tool does not consider the annotated switching activity. Therefore, reapply the switching activity after you run this command.

If you use of the `set_block_to_top_map` command to map modes, corners, and scenarios at the block level to those at the top level, you must manually update this mapping after you run the `remove_duplicate_timing_contexts` command. To determine the relationship between the modes, corners, and scenarios at the top level and those at the block level, use the `report_block_to_top_map` command.

Fusion Compiler Timing Constraints

Fusion Compiler uses the following classifications for timing constraints:

- Mode-specific constraints

  Constraints that define or modify the timing graph, which is a weighted graph that represents the timing of a circuit, are mode-specific constraints. Mode-specific constraints include

  - Clock definitions
  - Maximum and minimum delay constraints

  At a minimum, the mode-specific constraints must contain a clock definition for each clock signal.
- Corner-specific constraints

  Constraints that modify the calculated delay on an object are corner-specific constraints. Corner-specific constraints include

  - Operating conditions
  - Port load capacitances

- Parasitic information and scaling factors
- Timing derating factors

- Scenario-specific constraints

  Constraints that have a delay value and refer to a modal object, such as a clock, are scenario-specific. Scenario-specific constraints include

  - Input and output delays
  - Clock characteristics, such as clock latency, uncertainty, and transition

  At a minimum, the scenario-specific constraints should contain an input delay or output delay for each I/O port.

- Netlist-specific constraints

  Constraints that apply to all corners, modes, and scenarios are netlist-specific constraints. Netlist-specific constraints include ideal network settings.

  They can apply globally to all designs or locally to a specific design. For example, the constraints set by the `set_ideal_network` command are netlist-specific.

Setting Timing Constraints

You can specify the timing constraints by loading Synopsys Design Constraints (SDC) files or by using individual SDC commands. For details about the SDC commands, see the *Using the Synopsys Design Constraints Format Application Note*.

To load an SDC file, use the `read_sdc` command.

```
fc_shell> read_sdc block.sdc
```

Caution:

If the SDC file does not contain unit settings, they are derived from the main logic library. If the SDC file does contain unit settings, they must be consistent with those in the main logic library.

To ensure efficiency and accuracy during block setup, use the following steps:

1. Split the block constraints into separate files as follows:
   - Mode-specific file for each mode
   - Corner-specific file for each corner

   - Scenario-specific file for each scenario
   - Global constraint file for the blocks

2. Define all modes, corners, and scenarios for the block.
3. Set the current mode, corner, or scenario appropriately, before you apply the corresponding constraints.

   When you load an SDC file or run an SDC command, depending on the constraint type, the constraints only apply to the current scenario or its associated mode or corner.

For example, assume a block that has the following:

- Two modes named M1 and M2
- A corner named C
- Two scenarios as follows:
   - A scenario named M1@C, which is the combination of mode M1 and corner C
   - A scenario named M2@C, which is the combination of mode M2 and corner C

The following example script shows how to create modes, corner, and scenarios and apply the corresponding constraint and settings:

```
#Create all modes, corners, and scenarios

create_mode M1
create_mode M2

create_corner C

create_scenario -mode M1 -corner C -name M1@C
create_scenario -mode M2 -corner C -name M2@C

#Apply all modes, corners, and scenarios specific constraints
#and settings

current_mode M1
read_sdc M1_mode.sdc

current_mode M2
read_sdc M2_mode.sdc

current_corner C
read_sdc C_corner.sdc;

current_scenario M1@C
read_sdc M1@C_scenario.sdc
```

```
current_scenario M2@C
read_sdc M2@C_scenario.sdc
```

Converting an SDC File for Fusion Compiler Timing Analysis

The Fusion Compiler tool supports only OCV timing analysis. If your SDC file is written for best-case/worst-case (BCWC) timing analysis, you must convert the SDC file into two OCV SDC files.

To convert your BCWC SDC files into two OCV SDC files,

1. Create a mode for your design using the `create_mode` command, as described in Creating and Removing Modes.

   For example,

   ```
   fc_shell> create_mode my_mode
   ```

2. Create the following for the best-case constraints:

   - A corner by using the `create_corner` command, as described in Creating and Removing Corners.
   - A scenario by using the `create_scenario` command, as described in Creating a Scenario.

   The best-case scenario is not used for setup analysis. Therefore, disable setup analysis for this scenario by using the `set_scenario_status -setup false` command, as described in Setting the Active Analysis Types for a Scenario.

   For example,

   ```
   fc_shell> create_corner bc_corner
   fc_shell> create_scenario -mode my_mode -corner bc_corner \
      -name bc_scenario
   fc_shell> set_scenario_status -setup false bc_scenario
   ```

3. Read the best-case constraints from the BCWC SDC file by using the `read_sdc` command.

   By default, the Fusion Compiler tool reads all constraints in the SDC file. To limit the constraints to only the best-case constraints, set the `time.convert_constraint_from_bc_wc` application option to `bc_only` before reading the SDC file.

For example,

```
fc_shell> set_app_options \
   -name time.convert_constraint_from_bc_wc -value bc_only
fc_shell> read_sdc bc_wc.sdc
```

4. Create a corner and scenario for the worst-case constraints.

   The worst-case scenario is not used for hold analysis. Therefore, disable hold analysis for this scenario by using the `set_scenario_status -hold false` command

   For example,

```
fc_shell> create_corner wc_corner
fc_shell> create_scenario -mode my_mode -corner wc_corner \
   -name wc_scenario
fc_shell> set_scenario_status -hold false wc_scenario
```

5. Read the worst-case constraints from the BCWC SDC file by using the `read_sdc` command.

   To limit the constraints to only the worst-case constraints, set the `time.convert_constraint_from_bc_wc` application option to `wc_only` before reading the SDC file.

   For example,

```
fc_shell> set_app_options \
   -name time.convert_constraint_from_bc_wc -value wc_only
fc_shell> read_sdc bc_wc.sdc
```

6. Reset the `time.convert_constraint_from_bc_wc` application option to `none`.

   For example,

```
fc_shell> set_app_options \
   -name time.convert_constraint_from_bc_wc -value none
```

Importing Timing Constraints From the Design Compiler or IC Compiler Tool

To import timing constraints from the Design Compiler or IC Compiler tool to the Fusion Compiler tool, use the following steps:

1. Generate the constraints from the Design Compiler or IC Compiler tool by using the `write_timing_context -format icc2` command.

   By default, this command generates the timing constraints for all scenarios, both active and inactive, and the scenario-independent timing constraints for the block. To generate the timing constraints for specific scenarios, use the `-scenarios` option. You must specify a name for the output by using the `-output` option. The output is a directory that contains the Tcl scripts shown in the following table:

Table 2 Output of the write_timing_context -format icc2 Command

Tcl script name	Content
top.tcl	Commands for ◦ Creating the modes, corners, and scenarios ◦ Populating the modes, corners, scenarios, and the design with constraints
scenario_<scenario_name>.tcl	Constraints for the specific scenario
design.tcl	Constraints for the design that are scenario independent

2. Create the corresponding modes, corners, and scenarios and apply the constraints in the Fusion Compiler tool by sourcing the top.tcl script.

The following example generates the timing constraints for all active scenarios and the scenario-independent timing constraints from the Design Compiler tool:

```
dc_shell> write_timing_context -scenarios [get_scenarios \
     -active true] -output BLK1_const
```

The following example imports the constraints generated by the Design Compiler tool into the Fusion Compiler tool:

```
fc_shell> open_block BLK1
fc_shell> source BLK1_const/top.tcl
```

Removing SDC Constraints

To remove SDC constraints from a block, use the `remove_sdc` command.

By default, the command removes all SDC constraints in the current block. However, you can control the SDC constraints that are removed as follows:

- To remove the constraints from specific modes, corners, or scenarios, use the `-modes`, `-corners`, or `-scenarios` option.

  For example, to remove the mode-specific constraints from mode M1, corner-specific constraints from the C1 and C2 corners, and the scenario-specific constraints from the M1C1 and M1C2 scenarios, use the following command:

  ```
  fc_shell> remove_sdc -modes M1 -corners {C1 C2} \
     -scenarios {M1C1 M1C2}
  ```

- To remove the design-specific constraints that are not associated to any mode, corner, or scenario, use the `-design` option.
- To include or exclude specific types of constraints from removal, use the `-include` or `-exclude` option.

  For example, to remove only the case analysis settings, use the following command:

  ```
  fc_shell> remove_sdc -include case_analysis
  ```

  To remove all the SDC constraints except for ideal network settings, use the following command:

  ```
  fc_shell> remove_sdc -exclude ideal_network
  ```

  For a complete list of valid values for the `-include` and `-exclude` options, see the man page for the `remove_sdc` command.

2

Defining Clocks

An essential part of timing analysis is to accurately specify clocks and clock effects, such as latency and uncertainty. To specify, report, and analyze clocks see the following topics:

Creating Real Clocks

A real clock is a clock that physically exists and is defined on pin or port objects of a block. To create a real clock, use the `create_clock` command and specify the following information:

- The period of the clock by using `-period` option
- The source objects, pins or ports, on which to create the clock

  If you do not specify the source objects, the tool creates a virtual clock.

In addition, you can specify the following:

- The waveform of the clock by using the `-waveform` option

  If you do not specify the waveform, by default, the tool uses a 50 percent duty cycle with a rising edge at time zero and a falling edge at one-half the period.
- A name for the clock with the `-name` option

  If you do not specify a name explicitly, the clock gets its name from the source object.
- Additional clocks on the same source object by using the `-add` option

  If you do so, you must specify names for the additional clocks by using the `-name` option.

The following commands create a clock on the port named CLK, and another clock named CLK_1 with a rising edge at 1.0 and a falling edge at 2.0 on the same port:

```
fc_shell> create_clock -period 5.0 [get_ports CLK]
fc_shell> create_clock -period 4.0 -waveform {1.0 2.0} \
   -name CLK_1 -add [get_ports CLK]
```

Creating Virtual Clocks

A virtual clock is a clock that physically does not exist, but it can be used to constrain a block. To create a virtual clock, use the `create_clock` command and specify the following information:

- The period of the clock by using `-period` option
- A name for the clock with the `-name` option

In addition, you can specify the following:

- The waveform of the clock by using the `-waveform` option

  If you do not specify the waveform, by default, the tool uses a 50 percent duty cycle with a rising edge at time zero and a falling edge at one-half the period.

The following command create a clock named CLK_VIR that has a default waveform.

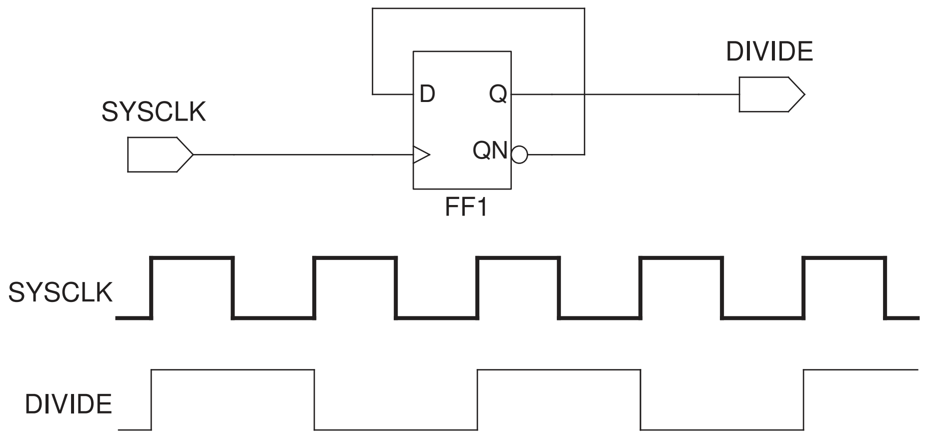
```
fc_shell> create_clock -period 5.0 -name CLK_VIR
```

Creating Generated Clocks

A clock that is generated based on another clock is called a generated clock.

The following figure shows an example of the clock generation logic for a divide-by-two generated clock, and the ideal waveforms of the original (master) clock and the generated clock.

Figure 1 Divide-by-two Clock Generator



The tool does not derive the behavior of the generated clock from the logic. You must specify the behavior of the generated clock, in terms of the master clock, by using the `create_generated_clock` command. When you do so, you must specify the following information:

- The source objects, ports, pins, or nets, on which to create the generated clock
- The source of the master clock by using the `-source` option

- How the frequency or the waveform of the generated clock is derived, by using one of the following three methods:
    - To derive the generated clock frequency by dividing the master clock frequency, use the `-divide_by` option and specify the division factor.
    - To derive the generated clock frequency by multiplying the master clock frequency, use the `-multiply_by` option and specify the multiplication factor.
    - To derive the generated clock waveform based on specific edges of the master clock, use the `-edges` option and specify the list of edges of the master clock to use.

Optionally, you can do the following:

- Specify a name for the generated clock by using the `-name` option.

  If you do not specify a name for the generated clock, the tool uses the name of the first object in the list of source objects you specified.
- Add multiple generated clocks on the same generated clock source object by using the `-add` option.

  When you do so, you must specify the master clocks for the additional generated clocks by using the `-master_clock` option.
- Consider only the combinational paths between the generated clock source and the master clock source, when calculating the source latency of the generated clock, by using the `-combinational` option.

  During timing analysis, by default, the tool considers both the combinational and sequential paths between the generated clock source and the master clock source.
- Invert the waveform by using the `-invert` option when you create a divide-by or multiply-by generated clock with the `-divide_by` or `-multiply_by` option.
- Use the inverted sense of the master clock by using the `-preinvert` option.
- Change the duty cycle by using the `-duty_cycle` option when you create a multiply-by generated clock with the `-multiply_by` option.
- Delay or shift the selected master clock edges by using the `-edge_shift` option when you create a generated clocks derived from specific edges of the master clock with the `-edges` option.

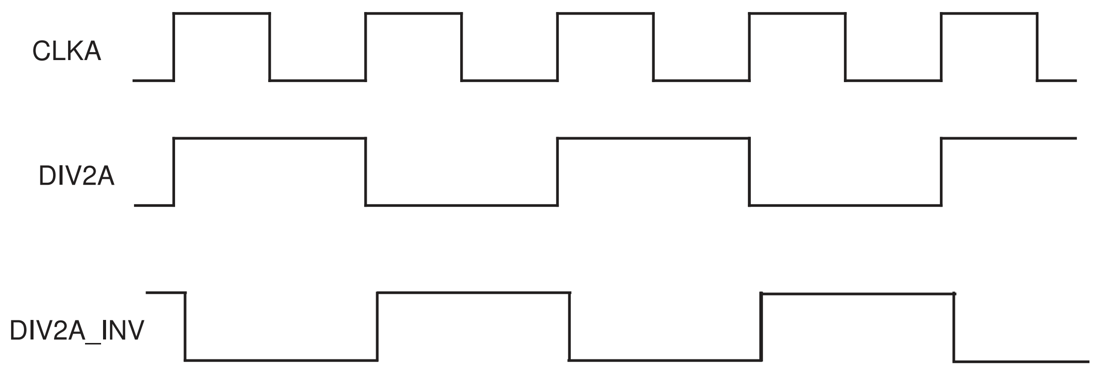
Generated Clock Examples

The following example creates two generated clocks named DIV2A and DIV2A_INV that has half the frequency of the master clock named CLKA. In addition, the waveform of the generated clocks named DIV2_INV is inverted from that of the master clock.

```
fc_shell> create_generated_clock -name DIV2A \
     -source [get_ports CLKA] -divide_by 2 \
     [get_pins U15/Q]
fc_shell> create_generated_clock -name DIV2A_INV \
     -source [get_ports CLKA] -divide_by 2 -invert \
     [get_pins U15/QN]
```

The following figure shows the waveforms for the master clock and generated clocks of this example.

Figure 2 Waveforms of the Master Clock and Divide-by-Two Generated Clocks

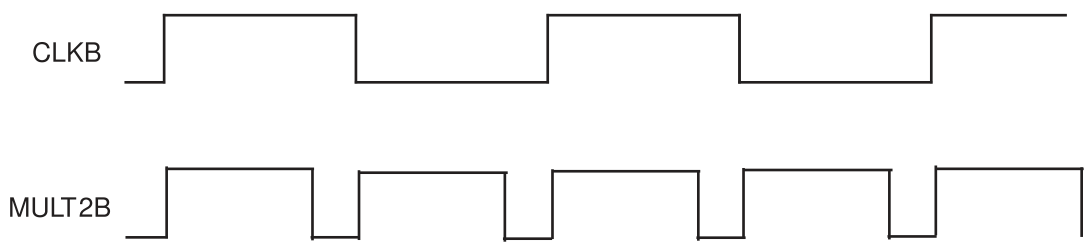


The following example creates a generated clock named MULT2B that has twice the frequency of the master clock named CLKB and a 75% active duty cycle.

```
fc_shell> create_generated_clock -name MULT2B -duty_cycle 75 \
     -source [get_ports CLKB] -multiply_by 2 \
     [get_pins U29/GC]
```

The following figure shows the waveforms for the master clock and generated clock of this example.

Figure 3 Waveform of the Master Clock and Multiply-by-Two Generated Clock



The following example creates a master clock named CLKC and a generated clocks named DIV3C that has a waveform based on the third, fifth, and ninth edge of the master clock and each edge shifted by 2.2 time units.

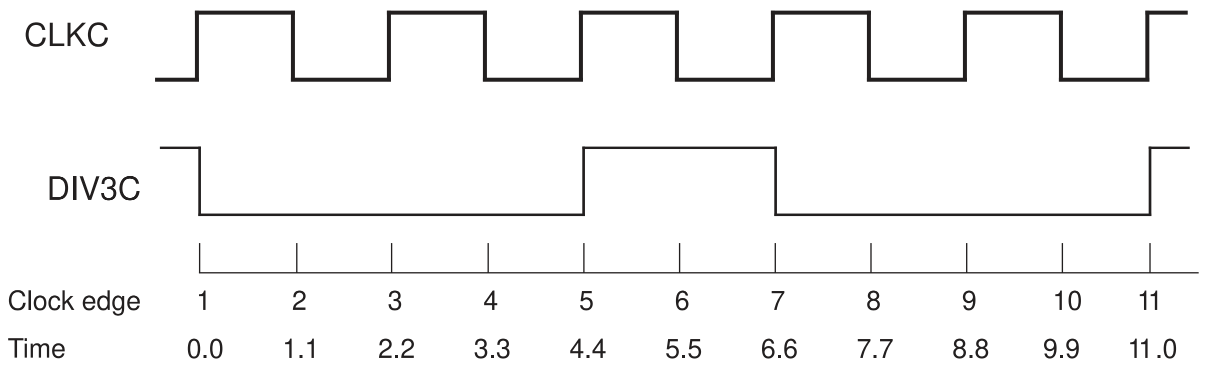
```
fc_shell> create_clock -period 2.2 -name CLKC [get_ports CLKC]

fc_shell> create_generated_clock -name DIV3C \
  -edges {3 5 9} -edge_shift {2.2 2.2 2.2}  \
  -source [get_ports SYSCLK] [get_pins U41/QN]]
```

The `report_clocks` command reports the specified clocks as follows:

```
    p - propagated_clock
    G - Generated clock

Clock          Period   Waveform        Attrs     Sources
-------------------------------------------------------------

CLKC             2.20    {0 1.1}                    {CLKC
DIV3C           6.60    {4.4 6.6}        G        {U41/Q}



Generated     Master       Generated     Waveform
Clock         Source       Source        Modification
-------------------------------------------------------------

DIV3C         MYCLK        U41/Q         edges( 3 5 9 )
                                         shifts( 2.2 2.2 2.2 )
```

The following figure shows the waveforms for the master clock and generated clock of this example.

Figure 4 Waveforms of the Master Clock and Divide-by-Three Generated Clock With Shifted Edges

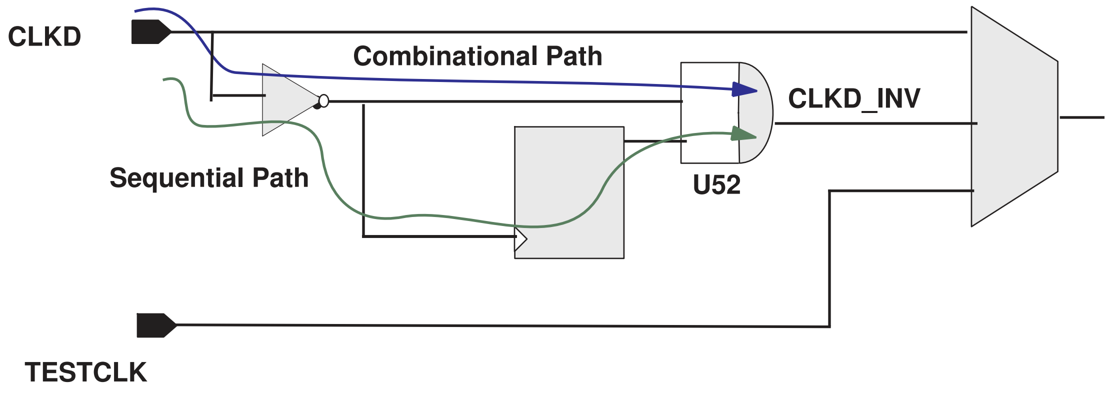


The following example creates a generated clock named CLKD and specifies that the tool uses only combinational paths between the generated clock source and mater clock source, when calculating the source latency for this generated clock:

```
fc_shell> create_generated_clock -name CLKD_INV -divide_by 1 \
       -combinational -source [get_ports CLKD] [get_pins U52/Y]}
```

The following figure shows the all available paths between the generated clock source and mater clock source for this example.

Figure 5 Paths in a Generated Clock Source Network



Clock Network Effects

A clock network has the following characteristics:

- Latency

  This is the amount of time a clock signal takes to propagate from the original clock source to the sequential elements in the design.

  The latency consists of the following two components:

  - Source latency

    This is the delay from the clock source to the clock definition pin in the design.

  - Network latency

    This is the delay from the clock definition point to the register clock pin.

    You can represent the network latency by using one of the following methods:

    - Estimating and explicitly specifying the latency of each clock

      This is referred to as ideal network latency and the corresponding clocks are referred to as ideal clocks. Ideal clocks are usually used before you perform clock tree synthesis.

    - Allowing the tool to compute the latency by propagating the delays along the clock network

      This is referred to as propagated network latency and the corresponding clocks are referred to as propagated clocks. Propagated clocks should be used only after you perform clock tree synthesis.

- Uncertainty

  This is the maximum variation in the clock arrival time at the register clock pins of the same clock or different clocks.

  The clock uncertainty has the following two components:

  - The clock jitter, which is the variation in the generation of successive edges of clock at the source clock, with respect to an ideal clock waveform

  - The clock skew, which is the difference in clock arrival times resulting from different propagation delays from the clock source of the block to the different register clock pins.

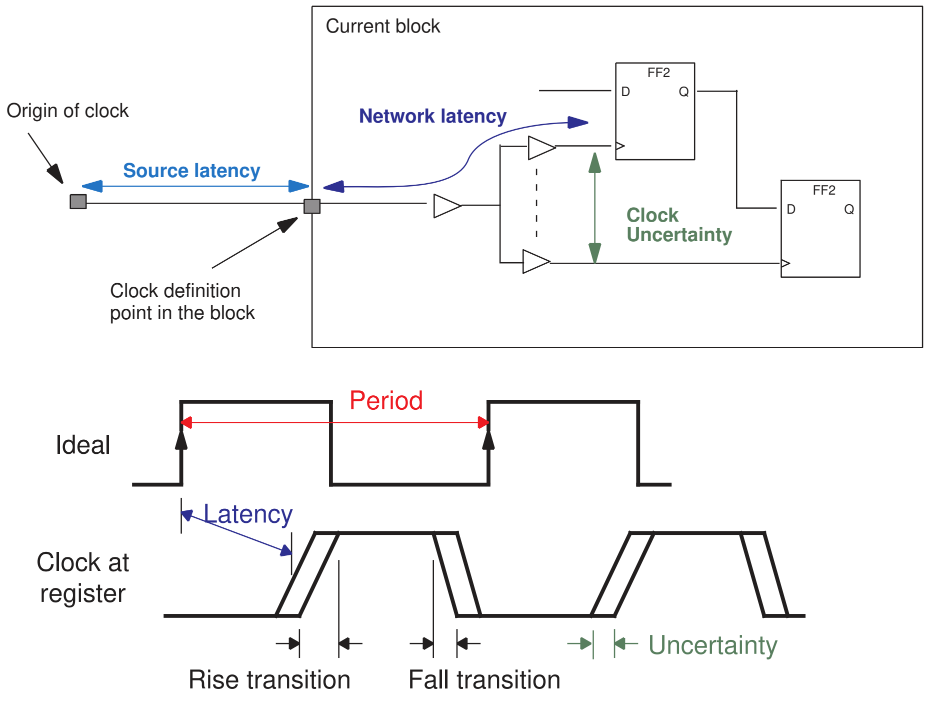
    This is applicable only before clock tree synthesis. After clock tree synthesis, the clock skew is accounted for by the propagated clock network latency.

- Transition time

  This is the amount of time it takes for a signal to change from logic low to logic high (rise time), or from logic high to logic low (fall time).

The following figure shows the timing effects of clock networks.

Figure 6 Clock Network Effects



Specifying Clock Source Latency

To set a source latency for both ideal and propagated clocks, use the `set_clock_latency -source` command specify the following information:

- The delay value
- The list of objects affected, which can contain one or more clocks, ports, or pins.

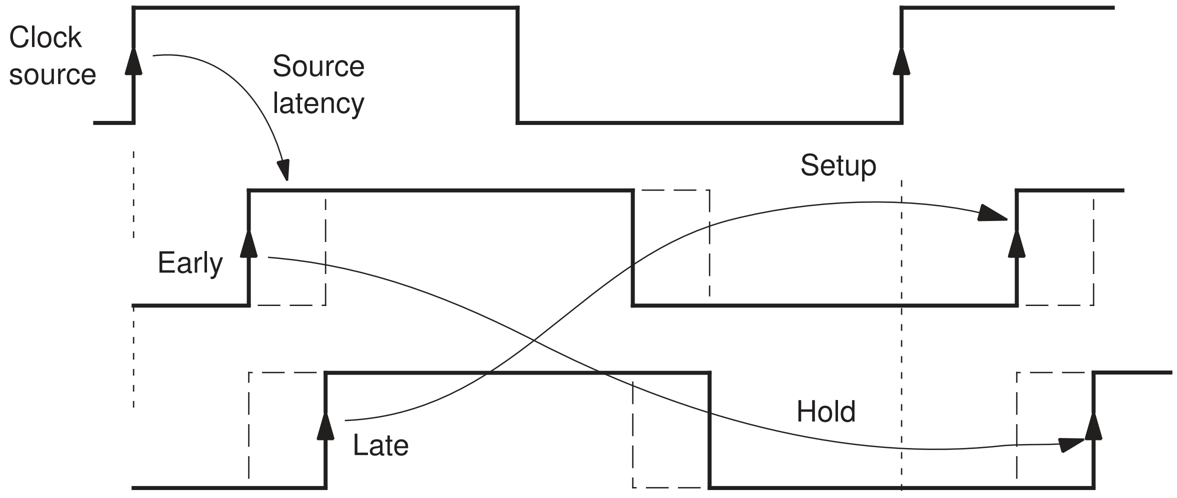
  If the list of objects specified are pins or ports, you must specify the clock associated with the ideal network latency by using the `-clock` option.

In addition, you can specify

- The early and late variation in the source latency by using the `-early` and `-late` options.

  The tool uses the most restrictive source latency value (either early or late) for each startpoint and endpoint clocked by the constraining clock. For a setup check, it uses the late value for each startpoint and the early value for each endpoint. For a hold check, it uses the early value for each startpoint and the late value for each endpoint, as shown in the following figure.

Figure 7 Early and Late Source Latency Waveforms



- The dynamic variation component of the source latency, which is the component that differs between successive clock cycles, by using the `-dynamic` option.

For designs with multiple scenarios, by default, the source latency applies only to the current scenario. To specify the source latency for

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The following commands specify the source latency of the clock CLK1 with an external clock network delay varying from 1.5 to 2.5 and having a dynamic component of 0.5:

```
fc_shell> set_clock_latency 1.5 -source -early -dynamic 0.5 \
   [get_clocks CLK1]
fc_shell> set_clock_latency 2.5 -source -late -dynamic 0.5 \
   [get_clocks CLK1]
```

To remove the source latency you specify with the `set_clock_latency` command, use the `remove_clock_latency` command.

Specifying Ideal Network Latency

To specify a clock as an ideal clock, use the `set_ideal_network` command. Until you complete clock tree synthesis, specify all clocks as ideal.

To specify the ideal network latency, use the `set_clock_latency` command and specify the following information:

- The delay value
- The list of objects affected, which can contain one or more clocks, ports, or pins.

  If the list of objects specified are pins or ports, you must specify the clock associated with the ideal network latency by using the `-clock` option.

  If you specify the ideal latency on a pin or port of a clock network and also the corresponding clock object, the value applied on the pin or port applies to all the register clock pins in the fanout and it overrides latency applied to the clock object.

In addition, you can restrict the ideal network latency setting to only

- The rising or falling edges of the clock by using the `-rise` or `-fall` option. Otherwise, the setting applies to both rising and falling edges.
- The minimum or maximum operating condition by using the `-min` or `-max` option. Otherwise, the setting applies to all operating conditions.

For designs with multiple scenarios, by default, the ideal clock latency applies only to the current scenario. To specify the ideal clock network latency for

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The following example sets the expected rise latency to 1.2 and the fall latency to 0.9 for the clock names CLK1 for the current design:

```
fc_shell> set_clock_latency -rise 1.2 [get_clocks CLK1]
fc_shell> set_clock_latency -fall 0.9 [get_clocks CLK1]
```

To remove the ideal network latency you specify with the `set_clock_latency` command, use the `remove_clock_latency` command.

Specifying Clock Uncertainty

To specify the clock uncertainty, use the `set_clock_uncertainty` command and specify the uncertainty value and one of the following:

- A simple uncertainty or jitter, which is the uncertainty between successive edges of the same clock, by specifying an object list consisting of clocks, ports, or pins

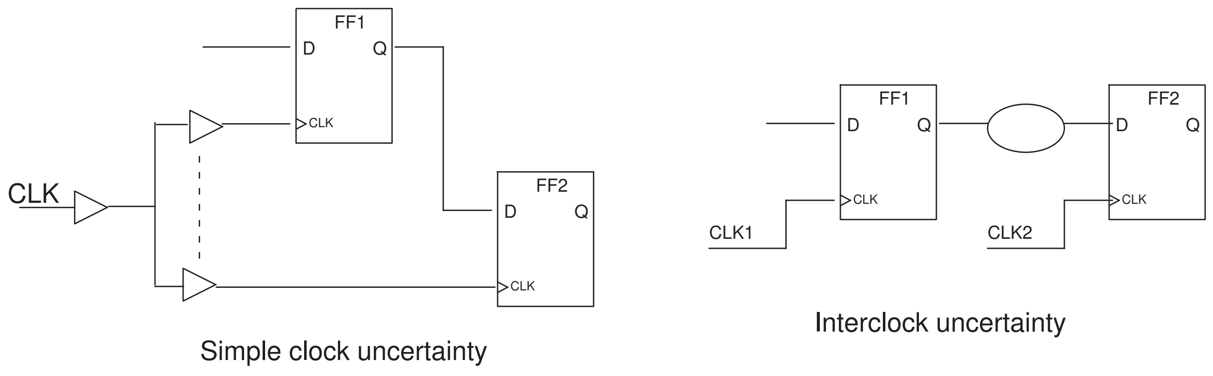
  The simple uncertainty applies to all the register clock pins in the fanout of the clocks, ports, or pins you specify.

- An interclock uncertainty between two clocks by using one of the following methods:

  - The source clock by using the `-from`, `-rise_from`, or `-fall_from` option
  - The destination clock by using the `-to`, `-rise_to`, or `-fall_to` option

    When you specify the interclock uncertainty between two clocks, if you have timing paths from the first clock to the second and from the second clock to the first, you must specify the uncertainty for both directions, even if the value is the same.

Figure 8 Simple and Interclock Uncertainty



To specify the simple clock or interclock uncertainty for:

- Setup checks, use the `-setup` option.
- Hold checks, use the `-hold` option.

For designs with multiple scenarios, by default, the clock uncertainty applies only to the current scenario. To specify the clock uncertainty for:

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

For example, to set a simple setup uncertainty of 0.21 and a hold uncertainty of 0.33 for all paths leading to endpoints clocked by the clock named CLK1, use the following commands:

```
fc_shell> set_clock_uncertainty -setup 0.21 [get_clocks CLK1]
fc_shell> set_clock_uncertainty -hold 0.33 [get_clocks CLK1]
```

To set an interclock uncertainty of 2 between clocks named CLKA and CLKB, for both setup and hold, use the following commands:

```
fc_shell> set_clock_uncertainty 2 -from [get_clocks CLKA] \
          -to [get_clocks CLKB]
fc_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] \
          -to [get_clocks CLKA]
```

To remove clock uncertainty settings, use the `remove_clock_uncertainty` command.

Specifying Clock Jitter

Use the `set_clock_jitter` command to set one of the following types of clock jitter on the master clock:

- *Cycle jitter* models the variation between the edges that are in phase (that is, it models the variation between the edges that are multiple clock cycles apart).
- *Duty-cycle jitter* models the variation between the edges that are out of phase (that is, those that are 0.5,1.5, 2.5, … cycles apart).

Clock jitter is inherited by all launch clocks and capture clocks generated from the master clock.

To determine the right type of clock jitter between a launch clock and capture clock, the command traces the launch edge and capture edge to corresponding edges of the master clock, as follows:

- If the corresponding edges of the master clock are multiple cycles apart, cycle jitter is used.
- If they are not an integral number of cycles apart, duty-cycle jitter is used.
- If they are traced back to the same edge of the master clock, no clock jitter is used.

To get a report on the cycle jitter and duty-cycle jitter for the master clock and all clocks generated from it, use the `report_clock_jitter` command.

To remove clock jitter settings from the master clock, use the `remove_clock_jitter` command. After clock jitter is removed from the master clock, it is removed from the generated clocks automatically.

Specifying Ideal Clock Transition

The default transition time of an ideal clock is zero. To specify a nonzero transition time for an ideal clock, use the `set_clock_transition` command and specify the following:

- The ideal transition value
- The list of clocks it applies to

In addition, you can specify that the transition time applies to only:

- The rising or falling edges of the clock by using the `-rise` or `-fall` option.
- The minimum conditions or maximum conditions by using the `-min` or `-max` option.

For designs with multiple scenarios, by default, the ideal clock transition applies only to the current scenario. To specify the ideal clock transition for:

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options.
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The tool uses the transition times you specify for all register clock pins in the transitive fanout of clocks you specify.

For example, for the current scenario, to specify a transition time of 0.64 at the clock pins of all the registers of the clock named CLK1, use the following command:

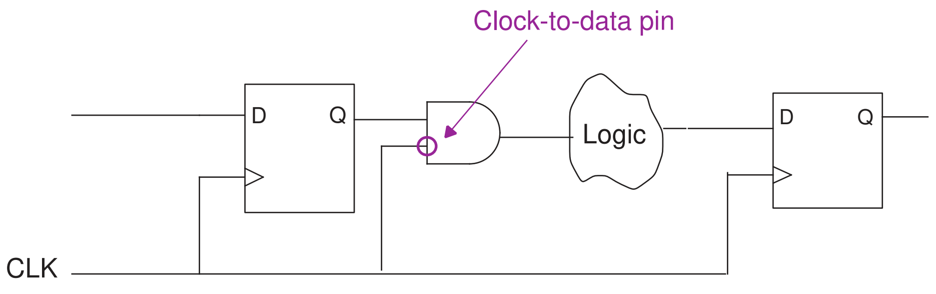
```
fc_shell> set_clock_transition 0.64 [get_clocks CLK1]
```

To remove the ideal transition specified by this command, use the `remove_clock_transition` command.

Controlling the Clock Marking Behavior

Starting from the clock root, the tool traverses each clock network and identifies all the clock-network pins by setting the `is_clock_used_as_clock` attribute to `true` for each pin. However, if a clock network reaches a pin beyond which there are no sequential pins, as shown in the following figure, the tool sets the `is_clock_used_as_data` attribute of that pin to `true`. Such a pin, where the clock network crosses over to the data network, is known as a clock-to-data pin.

Figure 9 Clock-to-Data Pin



You can use the following command to identify the clock-to-data pins in your block:

```
fc_shell> get_pins -filter "is_clock_used_as_data==true"
```

The process of identifying clock-network pins and clock-to-data pins is known as clock marking. The clock marking behavior is slightly different between the Fusion Compiler and PrimeTime tools. You can specify that the Fusion Compiler tool uses the same clock marking behavior as the PrimeTime tool by setting the `time.clock_marking` application option to `primetime`. The default is `native`.

For more information about the differences in the Fusion Compiler and PrimeTime clock marking behavior, see the man page of the `time.clock_marking` application option.

Enabling Clock-to-Data Analysis for Ideal Clocks

If a propagated clock reaches a clock-to-data pin, the tool uses the propagated clock arrival time as the data arrival time and the propagated clock transition time as the data transition time at the pin.

When an ideal clock reaches a clock-to-data pin, by default, the tool does not use the ideal clock latency as the data arrival time and the ideal clock transition time as the data transition time at the pin. To use the ideal clock latency and transition time at the clock-to-data pin, set the `time.enable_clock_to_data_analysis` application option to `true`.

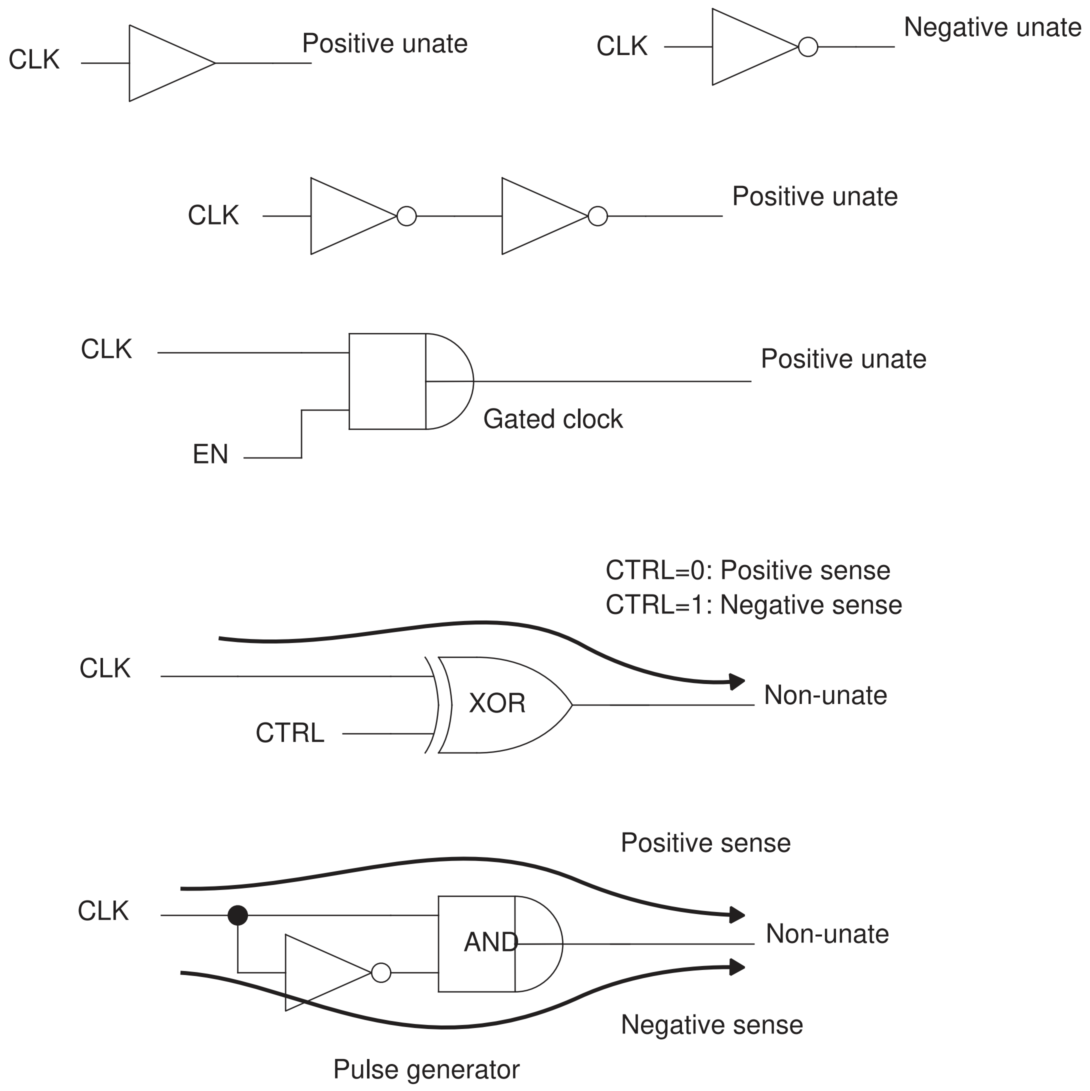
Unateness of Clocks

A clock signal is:

- Positive unate if a rising edge at the clock source can only cause a rising edge at the register clock pin, and a falling edge at the clock source can only cause a falling edge at the register clock pin.
- Negative unate if a rising edge at the clock source can only cause a falling edge at the register clock pin, and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted.
- Non-unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate. The clock sense could be either positive or negative, depending on the state of the other input to the XOR gate.

The following figure shows examples of clock logic that result in unate and non-unate clock signals.

Figure 10 Unate and Non-Unate Clock Signals



Propagating a Specific Clock Sense

If clock network contains only logic that results in positive or negative unate clock signals, the tool can derive the sense of the clock signal arriving at each register clock pin. If the clock network contains logic that results in a non-unate clock signal, you can resolve this ambiguity by using the `set_sense` command to only propagate a specific sense of a clock.

The following example propagates only the positive sense of the clock named CLK1 and the negative sense of the clock named CLK2 beyond the cell pin named U29/z:

```
fc_shell> set_sense -positive \
          -clocks [get_clocks CLK1] [get_pins U29/z]
fc_shell> set_sense -negative \
          -clocks [get_clocks CLK2] [get_pins U29/z]
```

The following example prevents the tool from propagating all clocks, both as clock and data, beyond the pin named U32/z:

```
fc_shell> set_sense -stop_propagation [get_pins U32/z]
```

Creating Pulse Clocks

A pulse clock, which are often used to improve performance and reduce power consumption, consists of a sequence of short pulses whose rising and falling edges are both triggered by the same edge of another clock.

To analyze the timing of a circuit containing pulse clocks, the tool needs information about the timing characteristics of the clock. To do so, you can use a pulse generator cell that has been characterized in its logic library description. In that case, no additional action is necessary to specify the pulse clock characteristics. For information about specifying the pulse generator characteristics of a library cell, see the Library Compiler documentation.

If characterized pulse generator cells are not available in the cell library, you must specify the pulse clock characteristics at each pulse generation point in the design by using one of the following methods:

- Use the `create_generated_clock` command to describe the pulse timing with respect to the source clock.

  This creates a new clock domain at the pulse generation point.

- Use the `set_sense` command to specify the sense of the generated pulses with respect to the source clock.

  This does not create a new clock domain, but merely specifies the sense for an existing clock downstream from the specified point.

Creating a Pulse Clock With the create_generated_clock Command

You can use the `-edges` option with the `create_generated_clock` command to create a pulse clock. When using the `-edges` option,

- The position of the repeated digit determines whether an active-high or active-low pulse is generated.
- The edge number that is repeated determines the type of edge in the master clock used to trigger the pulse.

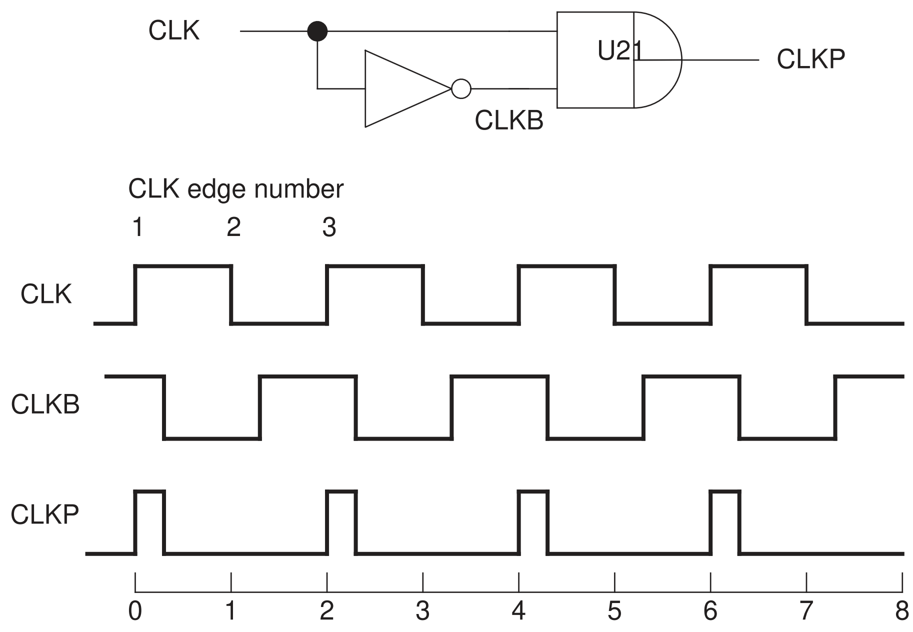
For example, if the

- Rising edge of the source triggers a high pulse, use the `-edges {1 1 3}` option setting.
- Falling edge of the source triggers a high pulse, use the `-edges {2 2 4}` option setting.
- Rising edge of the source triggers a low pulse, use the `-edges {1 3 3}` option setting.
- Falling edge of the source triggers a low pulse, use the `-edges {2 4 4}` option setting.

Specifying the generated clock as a pulse clock using repeated edge digits ensures correct checking of delays between the source clock and the pulse clock.

Consider the pulse clock circuit shown in the following figure:

Figure 11 Pulse Clock Specified as a Generated Clock



Edge number 1 of the source triggers both the rising and falling edges of the pulse clock. The pulse width is determined by the delay of the inverter.

To specify the generated pulse clock CLKP as a generated clock:

```
fc_shell> create_generated_clock -name CLKP -source CLK \
          -edges {1 1 3} [get_pins U21/z]
```

Defining a Pulse Clock With the set_sense Command

You can use the `-pulse` option with the `set_sense` command to specify an existing clock as a pulse clock.

To define a

- High pulse that is rise triggered, use the `-pulse rise_triggered_high_pulse` option setting.
- Low pulse that is rise triggered, use the `-pulse rise_triggered_low_pulse` option setting.

- High pulse that is fall triggered, use the `-pulse fall_triggered_high_pulse` option setting.
- Low pulse that is fall triggered, use the `-pulse fall_triggered_low_pulse` option setting.

The pulse clock is not defined as a separate clock domain. Instead, only the specified sense of the source clock is propagated downstream beyond the specified point in the clock network.

The following command specified that the clock at the output of the cell named U21 is a pulse that rises and falls on the rising edge of the source clock:

```
prompt> set_sense -pulse rise_triggered_high_pulse \
          [get_pins U21/z]
```

Specifying the Nominal Width of a Pulse Clock

The nominal width of the generated pulses is zero whether you use a pulse generator cell defined in the cell library, the `create_generated_clock` command, or the `set_sense` command. To determine the actual pulse width, the tool considers the different rise and fall latency values at the pulse generator output pin:

(high pulse width) = (fall network latency) – (rise network latency)

(low pulse width) = (rise network latency) – (fall network latency)

You can use the `set_clock_latency` command to specify the latency values (and therefore the pulse width) explicitly for an ideal clock, or you can allow the tool to calculate the propagated latency from the circuit for a propagated clock.

For example, to set an ideal pulse width to 0.5 for high pulses, for all registers downstream from the output of the cell named U21, and with an overall latency of 0.6, use the following commands:

```
fc_shell> set_clock_latency -rise 0.6 [get_pins and2/z]
fc_shell> set_clock_latency -fall 1.1 [get_pins and2/z]
```

Setting and Reporting Minimum Pulse Width Constraints

The width of the clock pulse at the register might be reduced due to the differences in the propagation delays of the rising and falling clock edges. If the clock pulse is too narrow at a register clock pin, the register might not capture data properly. Therefore, ensuring the clock meets a minimum pulse width constraint is necessary for proper operation of sequential circuits.

Library cell pins can have a minimum pulse width defined. In addition, you can explicitly set the pulse width constraints for specific clocks or clock pins by using the `set_min_pulse_width` command.

By default, the constraint is applied to both the high and low pulse of the clock. To specify for only the high or low pulse, use the `-high` or `-low` option.

The following example sets a high and low minimum pulse width constraint of 0.6 and 1.0 for the clock signal at the reg1/CK pin:

```
set_min_pulse_width -high 0.6 reg1/CK
set_min_pulse_width -low 1.0 reg1/CK
```

To check for pulse width violations, use the `report_min_pulse_width` or `report_constraints -min_pulse_width` command.

```
fc_shell> report_min_pulse_width
...
                   Required      Actual
  Pin              pulse width   pulse width   Slack
  -----------------------------------------------------
 reg1/CK(low)        1.00          0.87        -0.13
 reg1/CK(high)       0.60          1.13         0.53
```

To remove these constraints, use the `remove_min_pulse_width` command.

Checking the Minimum Period

Minimum period checking looks for violations of the `min_period` constraint for a cell pin, as defined in the cell library. This constraint requires the time between successive arrivals of clock edges of the same type (rising or falling) to be at least a specified value.

To perform minimum period checking, use one of these commands:

- `report_min_period [-verbose] ...`
- `report_constraint -min_period [-verbose] ...`
- `report_analysis_coverage -check_type min_period`

The following example shows how the tool calculates and reports the minimum period check with the `report_min_period` command.

```
pt_shell>fc_shell report_min_period -path_type full_clock uff2/CP
...
  Pin: uff2/CP
  Related clock: clk
  Check: sequential_clock_min_period

  Point                                         Incr      Path
```

```
---------------------------------------------------------
clock clk' (fall edge)                   0.00     0.00
clock source latency                     0.20     0.20
clk (in)                                 0.00     0.20 r
ckbf1/Y (IBUFFX2_HVT)                    1.11 H   1.31 f
ckbf2/Y (INVX1_HVT)                      0.08     1.39 r
ckbf3/Y (INVX1_HVT)                      0.03     1.41 f
uff2/CP (FD1)                           -0.05     1.37 f
open edge clock latency                           1.37

clock clk' (fall edge)                   3.50     3.50
clock source latency                     0.20     3.70
clk (in)                                 0.00     3.70 r
ckbf1/Y (IBUFFX2_HVT)                    0.81 H   4.51 f
ckbf2/Y (INVX1_HVT)                      0.06     4.57 r
ckbf3/Y (INVX1_HVT)                      0.02     4.59 f
uff2/CP (FD1)                           -0.04     4.55 f
clock reconvergence pessimism            0.32     4.87
clock uncertainty                       -0.02     4.85
close edge clock latency                          4.85
---------------------------------------------------------
required min period                               3.50
actual period                                     3.48
---------------------------------------------------------
slack (VIOLATED)                                 -0.02
```

In this example, the `report_min_period` command reports a minimum period violation at the CP pin of cell uff2. It shows two successive falling edges arriving at the pin with a spacing of less than 3.50 time units, the `min_period` constraint for that pin.

The “open edge clock latency” is the latest arrival time of the first edge, taking into account the maximum delays through the cells in the clock network.

The “close edge clock latency” is the earliest arrival time of the second edge, taking into account the minimum delays through the same cells in the clock network. It also adds back clock reconvergence pessimism, which lessens the calculated violation; and subtracts clock uncertainty, which contributes to the calculated violation.

The “actual period” is the “close edge clock latency” minus the “open edge clock latency.” In this example, the actual period is less than the required period, resulting in a reported slack of –0.02.

Defining the Relationship Between Clock Groups

For designs with multiple clocks, by default, the tool checks for timing paths between all clocks during timing analysis. However, if there is no interaction between specific clocks, you can specify that information by using `set_clock_groups` command, which can reduce the runtime for timing analysis.

When you use the `set_clock_groups` command, you must specify the following information:

- The relationship between the clock groups as one of the following:
  - Physically exclusive by using the `-physically_exclusive` option

    The tool assumes that these clock groups do not coexist in the design. Therefore, it does not check for timing paths between these clock groups and does not perform crosstalk analysis between the nets of the clock groups.
  - Logically exclusive by using the `-logically_exclusive` option

    The tool assumes that there is no phase relationship between these clock groups. Therefore, it does not check the timing paths between the clocks. However, it performs crosstalk analysis between the nets of the clock groups.
  - Asynchronous by using the `-asynchronous` option

    The tool assumes that there is no phase relationship between these clock groups. Therefore, it does not check the timing paths between the clocks. However, it performs crosstalk analysis between the nets of the clock groups using infinite arrival time windows.
- One or more clock groups by using the `-group` option one or more times

  If you specify the `-group` option
  - One time, the tool assumes the specified relationship between the clocks in that group and all other clocks in the design.
  - Two or more times, tool assumes the specified relationship between each pair of clock groups.

Optionally, you can

- Specify a name for the clock grouping by using the `-name` option

  If you do not specify a name, the tool derives a unique name for each grouping.
- Enable timing analysis between specific clocks of asynchronous grouping, specified with the `-asynchronous` option, by using `-allow_paths` option.

To remove clock grouping declarations made with the `set_clock_groups` command, use the `remove_clock_groups` command.

For example, to specify that clock CK1 is physically exclusive with respect to all other clocks in the design, use the following command:

```
fc_shell> set_clock_groups -physically_exclusive -group {CK1}
```

To specify that clocks CK2 and CK3 are logically exclusive with clocks CK4 and CK5, use the following command:

```
fc_shell> set_clock_groups -logically_exclusive \
   -group {CK2 CK3} -group {CK4 CK5}
```

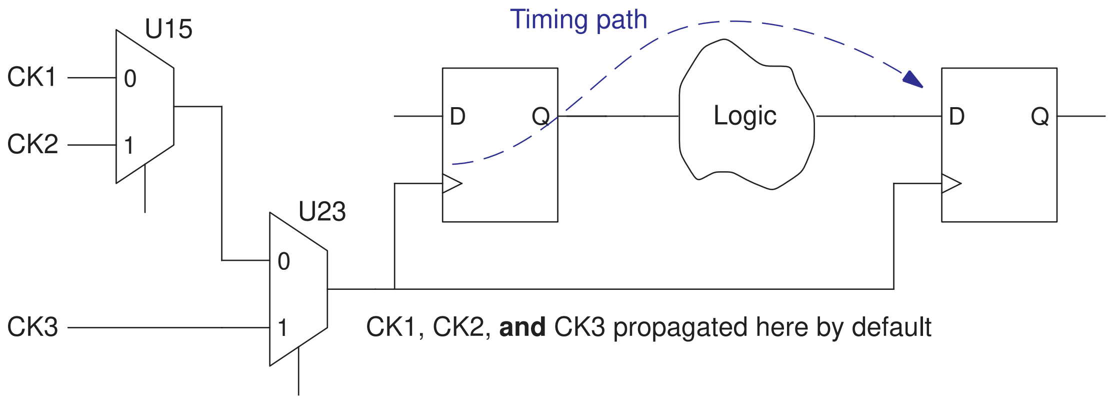
To specify that clocks CK6, CK7, and CK8 are asynchronous with respect to each other, use the following command:

```
fc_shell> set_clock_groups -asynchronous \
   -group {CK6} -group {CK7} -group {CK8}
```

Multiplexed Clock Exclusivity Points

Modern chip designs often use multiple clocks that can be applied to a given circuit at different times, for example, to operate at a higher frequency during periods of high workload and lower frequency at other times to save power. Multiplexer (MUX) cells are often used to select these clocks for operation, as shown in the following figure.

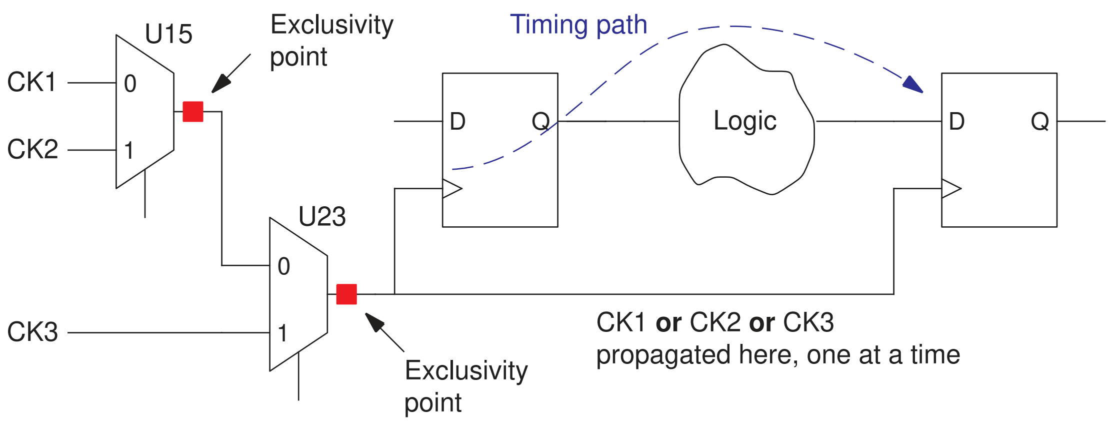
Figure 12 Multiplexed Clocks



By default, the Fusion Compiler tool considers all combinations of launch and capture clocks, such as data launched by CK1 and captured by CK2. However, for clocks multiplexed as shown in the figure, only one of the three clocks can operate on the path at any given time; they are exclusive clocks downstream from the MUX cells as long as the MUX control signals remain static between launch and capture.

One method of specifying exclusive clocks is to set “exclusivity points” to mark the pins in the design where only one clock at a time can be propagated, as shown in the following figure. For complex clock MUX configurations, this method typically offers better simplicity, accuracy, completeness, and turnaround time than other methods.

Figure 13 Multiplexed Clocks With Exclusivity Points



You can set clock exclusivity points at specific pins in the design, or you can have the tool automatically find and place these points at clock MUX output pins. The exclusivity setting applies to checking of delay, minimum pulse width, and minimum period constraints.

Setting Clock Exclusivity Points Explicitly

To set exclusivity points manually, use the `set_clock_exclusivity` command:

```
fc_shell> set_clock_exclusivity -type mux -output "U15/Z"

fc_shell> set_clock_exclusivity -type mux -output "U23/Z"
```

Setting a clock exclusivity point allows only one clock at a time to be propagated through the pin on which it is set. This enforces a physically exclusive relationship between the different clocks in the transitive fanout of the pins. The clock exclusivity effect works whether the `time.enable_auto_mux_clock_exclusivity` application option is set to `true` or `false`.

Use the `remove_clock_exclusivity` command to remove exclusivity points manually.

Inferring Clock Exclusivity Points Automatically

To have the tool automatically find and place exclusivity points at the outputs of clock MUX cells, set the `time.enable_auto_mux_clock_exclusivity` application option to `true`:

```
fc_shell> set_app_options \
   -name time.enable_auto_mux_clock_exclusivity -value true
```

With this application option set to `true`, the tool automatically finds mutiple clock signals entering a MUX cell and places exclusivity points at the output of the MUX. This enforces a physically exclusive relationship between the clocks in the transitive *fanout* of the MUX cell, without affecting possible clock interactions in the transitive *fanin* to the MUX cell.

To disable this automatic inference at selected MUX cells in the design, use the following command:

```
fc_shell> set_disable_auto_mux_clock_exclusivity MUX32/Z
```

Automatic detection of multiplexed clocks works only for MUX cells, as determined by the cell's `is_mux_cell` attribute. A multiplexing function built out of random logic is not automatically detected, but you can still manually define an exclusivity point in such logic by using the `set_clock_exclusivity` command. For example,

```
fc_shell> set_clock_exclusivity -type user_defined \
   -inputs "OR1/A OR1/B" -output OR1/Z
```

For a non-MUX cell, you must specify the clock input pins and exclusivity output pin.

Reporting Exclusivity Points

To report the exclusivity points, use the `-exclusivity` option of the `report_clocks` command:

```
fc_shell> report_clocks -exclusivity
...
clock exclusivity points:

   -type mux
   -output MUX88/Z
```

This reports all exclusivity points, whether inferred globally by using the `time.enable_auto_mux_clock_exclusivity` application option or explicitly by using the `set_clock_exclusivity` command.

To query the exclusivity of a pin:

```
fc_shell> get_attribute -class pin [get_pins U53/Z] \
   is_clock_exclusivity
true
```

Reporting Clock Information

To get a report on clocks that have been defined with the `create_clock` and `create_generated_clock` commands, use the `report_clocks` command. By default, this command reports:

- The period, waveform, attributes, and source for all clocks.
- The master clock source, master clock name, and waveform modification for all generated clocks.

Optionally, you can also report the following:

- Information for selected clocks by specifying the clock names
- Ideal clock network characteristics, including the source and network latency specified with the `set_clock_latency` command, uncertainty specified with the `set_clock_uncertainty` command, and ideal transition specified with the `set_clock_transition` command by using the `-skew` option
- Clock grouping information specified with the `set_clock_groups` command by using the `-group` option
- Clock information for selected modes by using the `-modes` option

To create a collection of clocks, use the `get_clocks` or `all_clocks` command. To distinguish between a clock object and port object that share the same name, use the `get_clocks` and `get_ports` commands.

For example, to generate a report on the clocks that have a period less than or equal to 15.0, use the following command:

```
fc_shell> report_clocks [get_clocks -filter "period <= 15.0" *]
...
Attributes:

    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock

Clock          Period   Waveform            Attrs     Sources
------------------------------------------------------------------------
PCI_CLK         15.00   {0 7.5}                       {pclk}
SDRAM_CLK        7.50   {0 3.75}                      {sdram_clk}
SD_DDR_CLK       7.50   {0 3.75}            G         {sd_CK}
SYS_CLK          8.00   {0 4}                         {sys_clk}
------------------------------------------------------------------------

Generated     Master         Generated      Master         Waveform
Clock         Source         Source         Clock          Modification
------------------------------------------------------------------------
SD_DDR_CLK    sdram_clk      {sd_CK}        SDRAM_CLK      divide_by(1)
------------------------------------------------------------------------
```

Removing Clocks

To remove a

- Real or virtual clock created with the `create_clock` command, use the `remove_clocks` command.
- Generated clock created with the `create_generated_clock` command, use the `remove_generated_clocks` command.

Feedback

3 Constraining Timing Paths

A timing path is a point-to-point sequence through a design that starts at a register clock pin or an input port, passes through combinational logic elements, and ends at a register data input pin or an output port.

The following topics provide information about timing paths and tasks related to constraining timing paths:

Timing Paths and Path Groups

The Fusion Compiler tool analyzes and optimizes timing paths in the design. Each path has a:

- Startpoint, where data is launched.

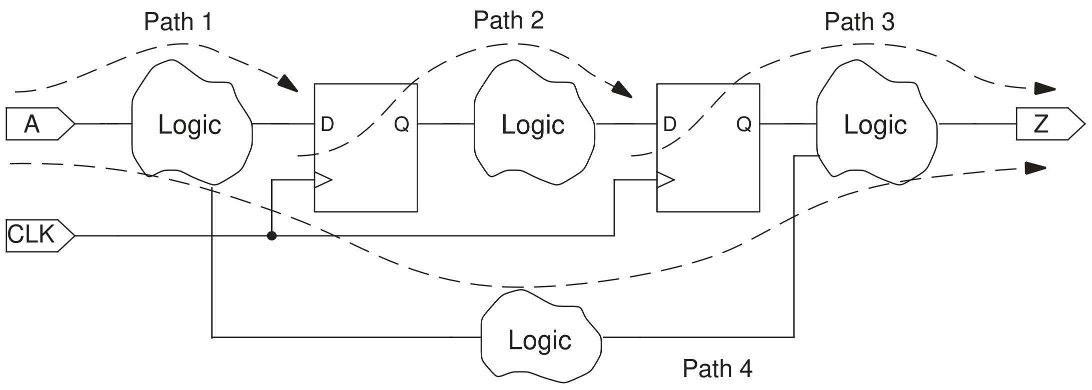
  The startpoint of a path is a clock pin of a sequential element or an input port of the design. The arrival of an active clock edge at a startpoint launches data through the path. An input port can be considered a startpoint due to the arrival of data launched by an external source.

- Endpoint, where the data is captured after it traverses through combinational logic.

  The endpoint of a path is a data input pin of a sequential element or an output port of the design. The arrival of an active clock edge at the clock input of the capture device captures data at the end of the path. An output port can be considered an endpoint due to the external capture of the data leaving the output port.

The following figure shows an example of a design and its timing paths.

Figure 14 Timing Path Types



Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point:

- Path 1 starts at an input port and ends at the data input of a sequential element.
- Path 2 starts at the clock pin of a sequential element and ends at the data input of a sequential element.
- Path 3 starts at the clock pin of a sequential element and ends at an output port.
- Path 4 starts at an input port and ends at an output port.

Each path in the design has an associated timing slack. The slack is a time value that can be positive, zero, or negative. The single path having the worst slack is called the critical path. This is the path that has the most negative slack, or if there are no timing violations, the one with the smallest slack among all paths in the design.

The tool organizes timing paths into groups, and performs timing analysis and optimization separately for each group. By default, there is one path group for each clock in the design. All timing paths clocked by a given clock at the path endpoint belong to the path group for that clock.

The tool also creates a path group called **default** (including the asterisks). The **default** group contains any paths that cannot be categorized by the clock used at the path endpoint, such as feedthrough and asynchronous paths.

All timing paths within a path group are optimized for timing together, starting with the critical path.

All path groups have the same weight by default, and paths within a group are optimized and reported separately from other path groups. Within each path group, the tool optimizes and improves the timing of the critical path until another paths in the group becomes more critical.

If multiple clocks arrive at the clock pin of a sequential cell, the tool constrains the cell with respect to each clock.

Creating Path Groups

To change the paths in a path groups, you can use the `group_path` command as follows:

- Create a new path group and specify its name by using the `-name` option.
- Assign paths to an existing path group and specify the existing path group name by using the `-name` option.
- Move paths from their current path group to the default path group they belong by using the `-default` option.

To identify the timing paths, use one or more of the following methods:

- Specify the startpoint of the path by using the `-from`, `-rise_from`, or `-fall_from` option
- Specify one or more throughpoints of the path by using the `-through`, `-rise_through`, or `-fall_through` option one or more times
- Specify the endpoint of the path by using the `-to`, `-rise_to`, or `-fall_to` option

You can influence how optimization handles the path group as follows:

- Change the weight of the path group by using the `-weight` option.
- Specify a critical range to identify critical paths for optimization by using the `-critical_range` option.

  The tool works on all critical paths within the range specified by the `-critical_range` option, starting from the most critical path.

The following example creates a path group named slct that has weight of 2.5, a critical range of 5, and contains all the timing paths starting from the register named slct_reg. It then adds the timing paths starting from the input port named slct_nxt to this path group. However, it removes the paths that start at the input port named slct_nxt and go through both the pins named U29/A and U54/Y from this group and returns these paths to their default path group.

```
fc_shell> group_path -name slct -weight 2.5 -critical_range 5 \
   -from [get_cells slct_reg]
fc_shell> group_path -name slct -from [get_ports slct_nxt]
fc_shell> group_path -default [get_ports slct_nxt] \
    -through [get_pins U29/A] -through [get_pins U54/Y]
```

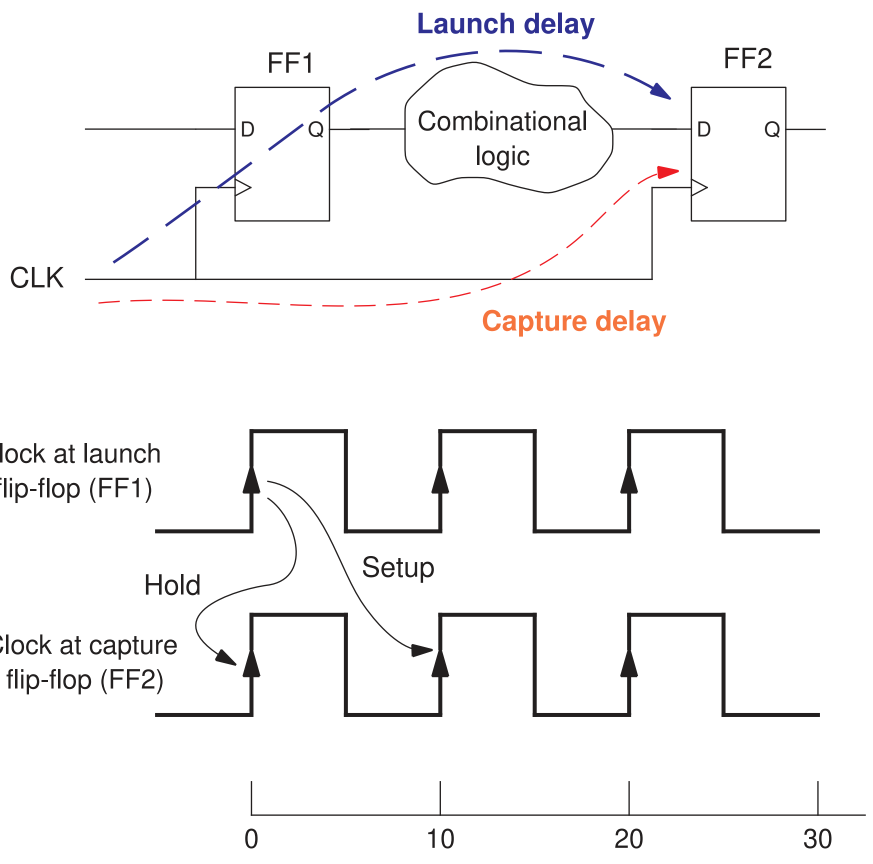
To report path group information, use the `report_path_groups` command and to remove path groups, use the `remove_path_groups` command.

Path Delay Analysis for the Same Launch and Capture Clock

By default, the tool checks for data arrival at the next capture clock edge following the launch event. For a single-clock design, launch occurs on a clock edge and capture occurs at the next clock edge.

The following figure shows how the tool determines the setup and hold constraints for a path that is launched and captured by the same clock, which has a period of 10.

Figure 15 Setup and Hold Analysis for the Same Launch and Capture Clock



During timing analysis the tool performs

- A setup check to verify that the data launched from FF1 at time 0 arrives at the D input of FF2 in time to meet the setup requirements of the capture edge at time 10.

  If the data takes too long to arrive and it does not meet the data setup requirements, the tool reports a setup violation.
- A hold check to verify that the data launched from FF1 at time 0 does not arrive too early at FF2, before the previous data at FF2 has met the hold requirements.

  If the data arrives too early, the tool reports a hold violation.

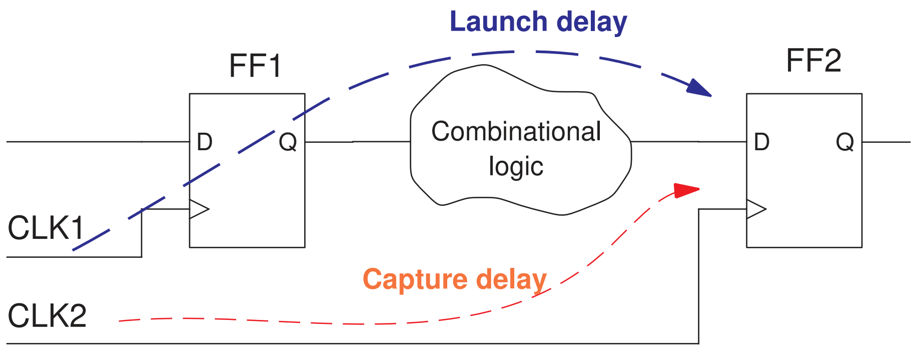
When calculating the delays along the launch and capture paths, the tool includes the delays due to the effects of the clock network, such as clock source latency, network latency, and clock uncertainty.

Path Delay Analysis for Different Launch and Capture Clocks

For a design with multiple clocks, by default, the tool assumes that all clocks are synchronous to each other, with a fixed phase relationship. During timing analysis, for paths where the launch and capture clocks are different, it considers the nearest possible capture clock edge that can occur after the launch clock edge and determines the most restrictive setup and hold requirements for the path.

The following figure shows a path that is launched and captured by two different clocks.

Figure 16 Path That is Launched and Captured by Different Clocks



The clocks in this example, CLK1 and CLK2, are created with the following commands:

```
fc_shell> create_clock -period 10 -waveform {0 5} CLK1
fc_shell> create_clock -period 25 -waveform {5 12.5} CLK2
```

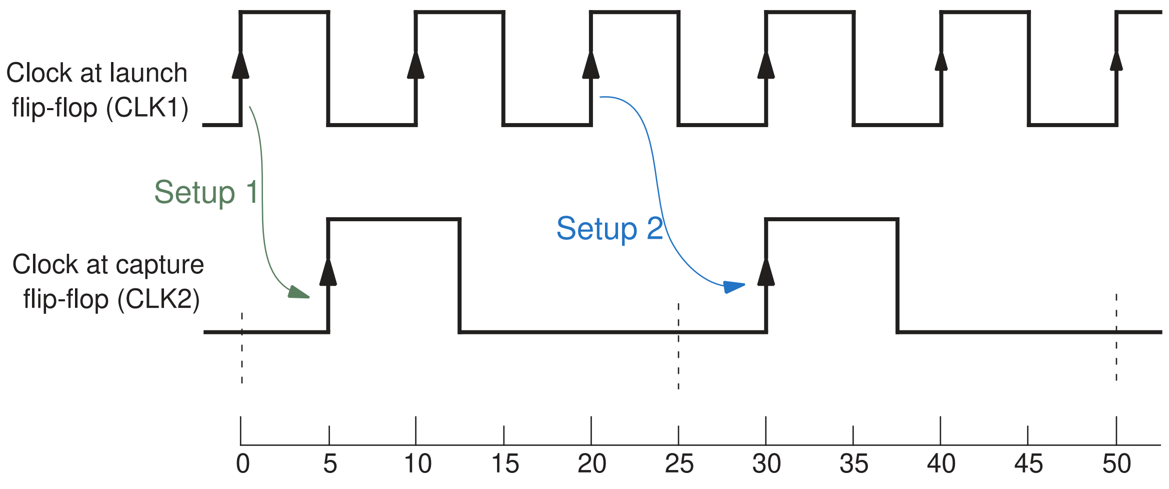
Setup Analysis for Different Launch and Capture Clocks

When the launch and capture clock are different, the tool looks at the relationship between the active edges of the launch and capture clocks over a number of repeating cycles that is the least common multiple of the two clock periods. For each capture edge at the destination flip-flop, the tool assumes that the corresponding launch edge is the nearest source clock edge occurring before the capture edge.

In the following figure, there are two launch and capture edge combinations that the tool considers. They are

- The capture edge at time=5 and the nearest preceding launch edge is at time=0, which is labeled Setup 1.
- The capture edge at time=30 and the nearest preceding launch edge is at time=20, which is labeled Setup 2.

Figure 17 Setup Analysis for Different Launch and Capture Clock


Clock at launch flip-flop (CLK1)
Setup 1
Setup 2
Clock at capture flip-flop (CLK2)
0
5
10
15
20
25
30
35
40
45
50

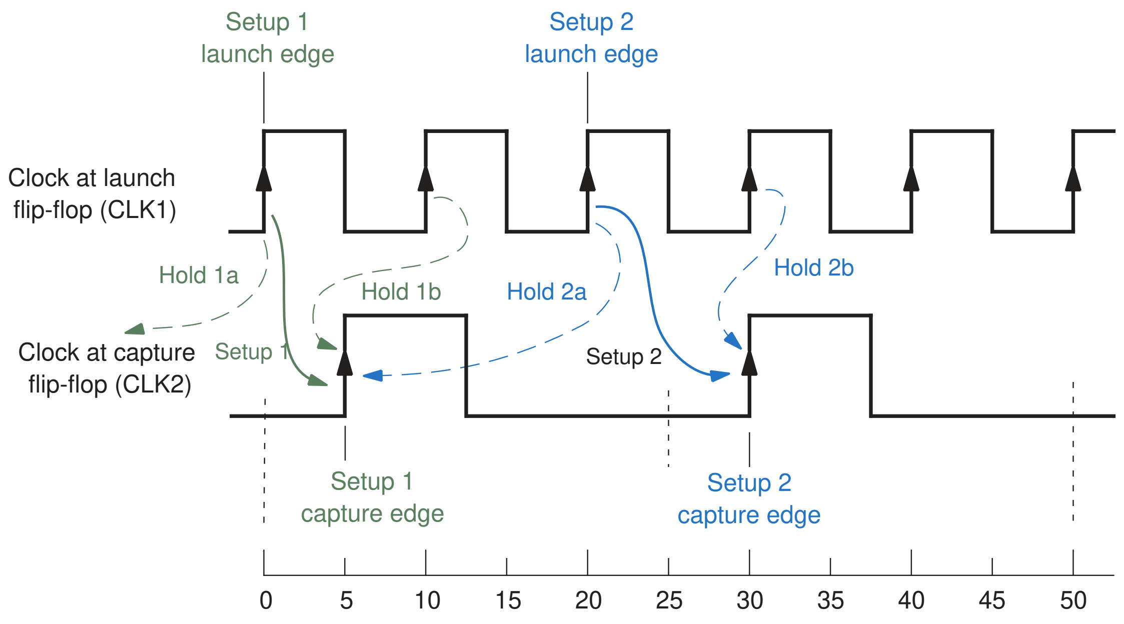
The source clock edge at time=30 occurs at the same time as the capture edge, not earlier, so it is not considered the corresponding launch edge.

Setup 1 allows less time between launch and capture. Therefore, it is the more restrictive constraint and determines the maximum allowed delay for the path, which is 5 for this example.

Hold Analysis for Different Launch and Capture Clocks

The hold relationships checked by the tool are based on the clock edges adjacent to those used to determine the setup relationships. To determine the most restrictive hold relationship, the tool considers all valid setup relationships, including both Setup 1 and Setup 2 in the following figure.

Figure 18 Hold Analysis for Different Launch and Capture Clock



For each setup relationship, the tool performs two different hold checks:

- The data launched by the setup launch edge must not be captured by the *previous* capture edge.
- The data launched by the *next* launch edge must not be captured by the setup capture edge.

For the setup relationship labeled Setup 1, the corresponding two hold checks are labeled Hold1a and Hold1b. Similarly, for the setup relationship labeled Setup 2, the two hold checks are labeled Hold2a and Hold2b.

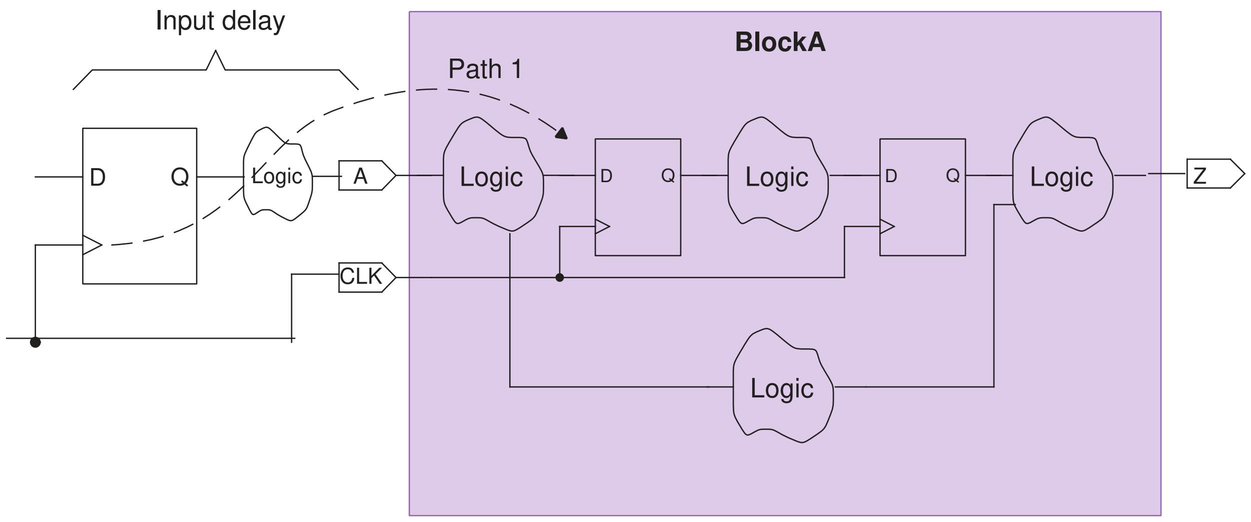
Of these hold checks, the most restrictive is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b. Therefore, Hold 2b determines the minimum allowed delay for this path, which is 0.

Introduction to Input and Output Delays

To analyze a timing path, both the startpoint and endpoint of the path must be constrained. However, by default, the tool does not know the data arrival times at input ports and the data required times at output ports. Therefore, to constraint input and output ports, you must specify this information.

In the following figure, the data arriving at the input port named A of the block named BlockA is launched by an external sequential device.

Figure 19 External Input Delay

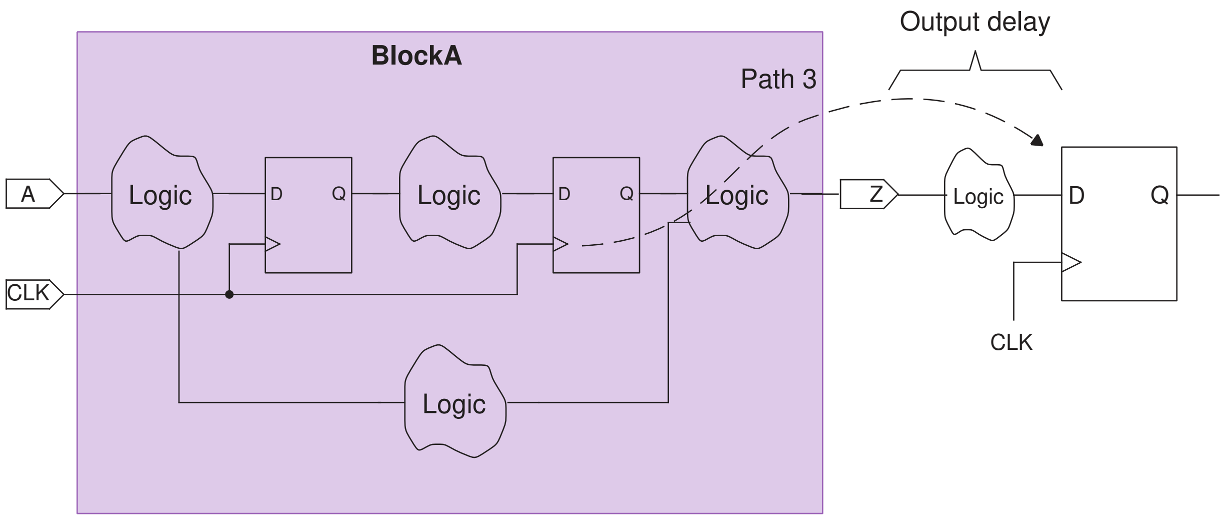


To constrain the input port A, use the `set_input_delay` command and specify

- The clock that launches the data at the external sequential device
- The delay for the data to arrive at the input port after it is launched

In the following figure, the data leaving the output port named Z of the block named BlockA is captured by an external sequential device.

Figure 20 External Output Delay



To constrain the output port Z, use the `set_output_delay` command and specify

- The clock that captures the data at the external sequential device
- The delay for the data to arrive at the external sequential device after it leaves port Z

Setting Input and Output Delays for Ports

To constrain an input or output port by specifying the external delay outside the port, use the `set_input_delay` or `set_output_delay` command. You can also use these commands to apply an input or output delay on a pin within the block, thereby making that pin a startpoint or endpoint of a timing path.

When you use the `set_input_delay` or `set_input_delay` command, you must specify the following information:

- The input or output delay value
- The ports or pins that you are constraining

For a more accurate timing analysis, specify a related clock by using one of the following two methods:

- Specify the `-clock` option with the name of the reference clock

  When you specify the `-clock` option, by default, the tool

  - Assumes the input or output delay is relative to the rising edge of the clock. To specify that the delay is relative to its falling edge, use the `-clock_fall` option with the `-clock` option.
  - Adds the source and network latencies specified for the reference clock to the launch or capture path associated with the input or output delay.

    To specify that the source and network latencies of the reference clock are already included in the input or output delay, use the `-source_latency_included` and `-network_latency_included` options. If the reference clock is a propagated clock, the `-network_latency_included` option is ignored.
- Specify the `-reference_pin` option with a pin or port on the reference clock network

  When you specify the `-reference_pin` option, by default, the tool

  - Uses all the clocks that reach the reference pin to constrain the ports. To select a specific clock, use the `-clock` option with the `-reference_pin` option.
  - Adds the source and network latencies specified for the for any ideal clocks that reach the reference pin, to the corresponding launch or capture path associated with the input or output delay,

    However, for any propagated clocks that reach the reference pin, tool adds the source delay specified for that clock and the propagated clock network delay up to the reference pin to the corresponding launch or capture path associated with the input or output delay.

If you do not specify a relative clock, the tool derives a new clock based on the clocks of the current block.

When you use the `set_input_delay` or `set_output_delay` command, you can specify that the delay:

- Is only for a rise or fall transition by using the -`rise` or `-fall` option.

  By default, the tool uses the delay for both the rise and fall transition at the input or output port.
- Is only for the longest or shortest path by using the -`max` or `-min` option.

  By default, the tool uses the delay is for both the longest and shortest paths.
- Is with respect to a level-sensitive latch by using the `-level_sensitive` option.

By default, the tool assumes the timing paths associated with the input or output delay are launched or captured by an edge-sensitive flip-flop.

- Should not overwrite any existing input or output delay settings by using the `-add_delay` option.

  Use this option to specify different input or output delays with respect to different reference clocks, for the same port.

For designs with multiple scenarios, by default, the input or output delay applies only to the current scenario. To specify the input or output delay for

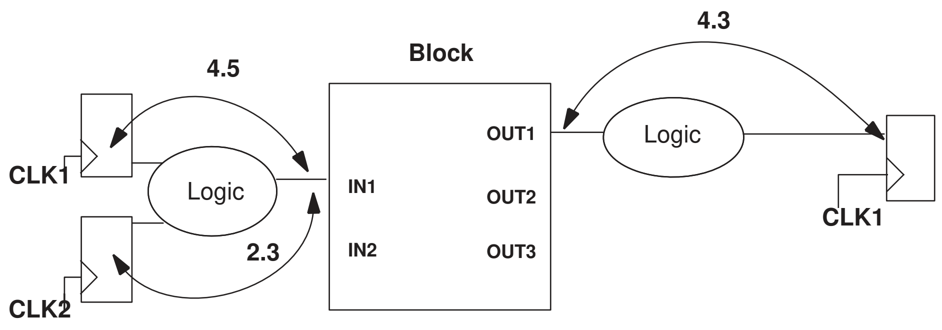
- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

To remove input or output delays from ports, use the `remove_input_delay` or `remove_output_delay` command.

The following figure shows the external paths for one input and output of a block, and the delays associate with those external paths.

Figure 21 External Delays for an Input and Output Port



The following commands constrain the input named IN1 by setting an input delay of 4.5 with respect to the clock named CLK1 and an input delay of 2.3 with respect to the clock named CLK2:

```
fc_shell> set_input_delay 4.5 -clock CLK1 [get_ports IN1]
fc_shell> set_input_delay 2.3 -clock CLK2 [get_ports IN1] \
   -add_delay
```

The following command constrains the output named OUT1 by setting an output delay of 4.3 with respect to the clock named CLK1:

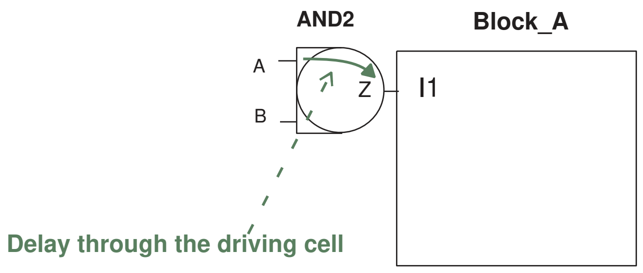
```
fc_shell> set_input_delay 4.3 -clock CLK1 [get_ports OUT1]
```

Considering the Effect of Delays Through Driving Cells

If you constrain an input or inout port using the `set_driving_cell` command, as shown in the following example, the tool includes the load-dependent value of the external driving-cell delay in the timing paths that begin at the port, as shown in the figure.

```
fc_shell> set_driving_cell -lib_cell AND2 \
   -from_pin A [get_ports I1]
```

Figure 22 Delay Through the Driving Cell



To prevent this delay from being counted twice, estimate the load-dependent delay of the driving cell, then subtract that amount from the input delay you specify on the port.

Effect of Input Delay on Clock Ports

The tool considers the input delay on clock source ports or pins as source latency if the clock is propagated. The input delay can be relative to no clock, or to the clock of that source. The source latency value is added to the clock network delay to determine total latency.

Do not set an input delay on all input ports, as shown in the following example:

```
fc_shell> set_input_delay 2 -clock CLK [all_inputs]
```

Instead, set in input delay on all inputs excluding the clock ports, as shown in the following example:

```
fc_shell> set_input_delay 2 -clock CLK \
      [remove_from_collection [all_inputs] [get_ports CLK]]
```

Use the `set_clock_latency` command with the `-source` option to define the actual source latency, if any.

Specifying Timing Exceptions

The Fusion Compiler tool allows you to set the following types of timing exceptions:

- False paths, which identifies paths that do not need to be analyzed for timing.

  To do so, use the `set_false_path` command, as described in Specifying False Path Exceptions.

- Minimum and maximum delay values, which sets minimum and maximum path delay requirements between specified startpoints and endpoints.

  To do so, use the `set_max_delay` and `set_min_delay` commands, as described in Specifying Maximum and Minimum Path Delay Exceptions

- Multicycle paths, which overrides the default single cycle requirement by setting the number of clock cycles required for the specified paths.

  To do so, use the `set_multicycle_path` command, as described in Specifying Multicycle Path Exceptions

- Path timing margins, which tightens or loosens the timing requirement for the specified paths by a specified amount.

  To do so, use the `set_path_margin` command, as described in Specifying Path Timing Margin Exceptions

Each timing exception command can apply to a single path or to a group of related paths, such as all paths from one clock domain to another, or all paths that pass through a specified point in the design.

To report all the timing exceptions that have been applied to a design, use the `report_exceptions` command.

To restore a path to its default before timing exceptions were applied, use the `reset_paths` command or the `-reset_path` option of each timing exception command.

Specifying False Path Exceptions

A false path is a path that does not need to be analyzed for timing. For example, a path can exist between two multiplexed logic blocks that are never selected at the same time, so that path is not valid for timing analysis. Specifying such paths as false paths prevents the tool from spending time analyzing timing paths that would not occur in the real design.

To define timing paths as false paths, use the `set_false_path` command. When you do so, you must specify the paths you want to set as a false paths by using one or more of the following methods:

- Specify the startpoint of the path by using the `-from`, `-rise_from`, or `-fall_from` option

  Valid startpoints are

  - Clocks

    If you specify a clock, timing paths starting from all the sequential cells and primary ports constrained by that clock are set as false paths.
  - Sequential cells
  - Clock pins of a sequential cells
  - Data pins of latches
  - Input or inout ports
  - Pins with an input delay settings
- Specify one or more throughpoints of the path by using the `-through`, `-rise_through`, or `-fall_through` option one or more times

  Valid throughpoints are

  - Pins
  - Ports
  - Cells
  - Nets

  If all paths going through a pin are false paths, using the `set_disable_timing` to disable a timing arc of that pin is more efficient than using the `set_false_path -through` command.

- Specify the endpoint of the path by using the `-to`, `-rise_to`, or `-fall_to` option

  Valid endpoints are

  - Clocks

    If you specify a clock, timing paths ending at all the sequential cells and primary ports constrained by that clock are set as false paths.

  - Sequential cells
  - Data pins of sequential cells
  - Output or inout ports
  - Pins with an output delay settings

Optionally, you can limit the false-path setting to

- Setup or hold analysis only by using the `-setup` or `-hold` options.

  By default, the false-path setting applies to both setup and hold analysis.

- Rise or fall analysis only by using the `-rise` or `-fall` options.

  By default, the false-path setting applies to both rise and fall analysis.

To remove a false path setting, use the `set_false_path -reset_path` or `reset_paths` command.

The following example shows how to set a very specific path, which starting from a register clock pin, going through several pins, and ending at a register data pin, as a false path:

```
fc_shell> set_false_path -from [get_pins reg2/CP] \
  -through [get_pins U31/Z] -through [get_pins U15/Z] \
  -to [get_pins reg7/D]
```

The following example shows how to set all timing paths between two clocks, ck1 and ck2, as false paths:

```
fc_shell> set_false_path -from [get_clocks ck1] \
          -to [get_clocks ck2]
fc_shell> set_false_path -from [get_clocks ck2] \
          -to [get_clocks ck1]
```

For this example, an alternative is to use the `set_clock_groups` command to exclude all paths between clocks ck1 and ck2 from timing analysis.

Specifying Maximum and Minimum Path Delay Exceptions

By default, the tool calculates the maximum and minimum allowable path delays by considering the launch and capture clock edge times. To override the default maximum or minimum allowable path delay with your own specific delay constraint, use the `set_max_delay` or `set_min_delay` command. You can also use these commands to constrain a purely combinational timing path that is not constrained by a clock.

When use the `set_max_delay` or `set_min_delay` command, you must specify

- The maximum or minimum delay value
- The path you want to constrain by using one or more of the following methods:
    - Specify the startpoint of the path by using the `-from`, `-rise_from`, or `-fall_from` option.

      Valid startpoints are

        - Clocks

          If you specify a clock, timing paths starting from all the sequential cells and primary ports constrained by that clock are set as false paths.
        - Sequential cells
        - Clock pins of a sequential cells
        - Data pins of latches
        - Input or inout ports
        - Pins with an input delay settings
    - Specify one or more throughpoints of the path by using the `-through`, `-rise_through`, or `-fall_through` option one or more times.

      Valid throughpoints are

        - Pins
        - Ports
        - Cells
        - Nets
    - Specify the endpoint of the path by using the `-to`, `-rise_to`, or `-fall_to` option.

      Valid endpoints are

- Clocks

  If you specify a clock, timing paths ending at all the sequential cells and primary ports constrained by that clock are set as false paths.

- Sequential cells
- Data pins of sequential cells
- Output or inout ports
- Pins with an output delay settings

Optionally, you can limit the maximum or minimum path delay setting to

- Setup or hold analysis only by using the `-setup` or `-hold` options.

  By default, it applies to both setup and hold analysis.

- Rise or fall analysis only by using the `-rise` or `-fall` options.

  By default, it applies to both rise and fall analysis.

If the startpoint or endpoint of the path for which you applied the `set_max_delay` or `set_min_delay` command is constrained by a clock, the tool uses the clock latencies of the startpoint and endpoint during timing analysis. To ignore the clock latencies of the startpoint and endpoint, use the `-ignore_clock_latency` option.

To remove a maximum or minimum path delay setting, use the `-reset_path` option of the `set_max_delay` or `set_min_delay` command or use the `reset_paths` command.

The following example sets a maximum path delay between registers REGA and REGB to 12 and minimum path delay to 2:

```
fc_shell> set_max_delay 12.0 \
          -from [get_cells REGA] -to [get_cells REGB]
fc_shell> set_min_delay 2.0 \
          -from [get_cells REGA] -to [get_cells REGB]
```

Specifying Multicycle Path Exceptions

If the number of clock cycles required to propagate data from the start to the end of a path is more than the single cycle requirement used by the tool, you can specify additional clock cycles for the path by using the `set_multicycle_path` command. Then, the tool calculates the setup or hold constraint according to the specified number of cycles.

When use the `set_multicycle_path` command, you must specify

- The number of clock cycles you want to allocate for the path.

  If the launch and capture clocks for the path are different clocks, specify if the number of clock cycles are based on the launch or capture clock by using the `-start` or `-end` options. By default, the tool moves the setup check relative to the capture clock, and the hold check relative to the launch clock.

- The path you want to constrain by using one or more of the following methods:
  - Specify the startpoint of the path by using the `-from`, `-rise_from`, or `-fall_from` option.

    Valid startpoints are

    - Clocks

      If you specify a clock, timing paths starting from all the sequential cells and primary ports constrained by that clock are set as false paths.

    - Sequential cells
    - Clock pins of a sequential cells
    - Data pins of a latches
    - Input or inout ports
    - Pins with an input delay settings
  - Specify one or more throughpoints of the path by using the `-through`, `-rise_through`, or `-fall_through` option one or more times.

    Valid throughpoints are

    - Pins
    - Ports
    - Cells
    - Nets
  - Specify the endpoint of the path by using the `-to`, `-rise_to`, or `-fall_to` option.

    Valid endpoints are

    - Clocks

      If you specify a clock, timing paths ending at all the sequential cells and primary ports constrained by that clock are set as false paths.

- Sequential cells
- Data pins of sequential cells
- Output or inout ports
- Pins with an output delay settings

Optionally, you can limit the multicycle path setting to

- Setup or hold analysis only by using the `-setup` or `-hold` options.

  By default, it applies to both setup and hold analysis.

  The tool interprets the `-setup` and `-hold` values in the `set_multicycle_path` command differently. The integer value for the `-setup` argument specifies the number of clock cycles for the multicycle path. In the absence of a timing exception, the default is 1. The integer value for the `-hold` argument specifies the number of clock cycles to move the capture edge backward with respect to the default position, relative to the valid setup relationship. The default is 0.

- Rise or fall analysis only by using the `-rise` or `-fall` options.

  By default, it applies to both rise and fall analysis.

To remove a multicycle path setting, use the `set_multicycle_path -reset_path` or `reset_paths` command.

Specifying the Correct Number of Multicycles for Hold Analysis

The following example shows how to determine the correct number of cycles required for hold analysis, when you know the number of cycles required for setup analysis.

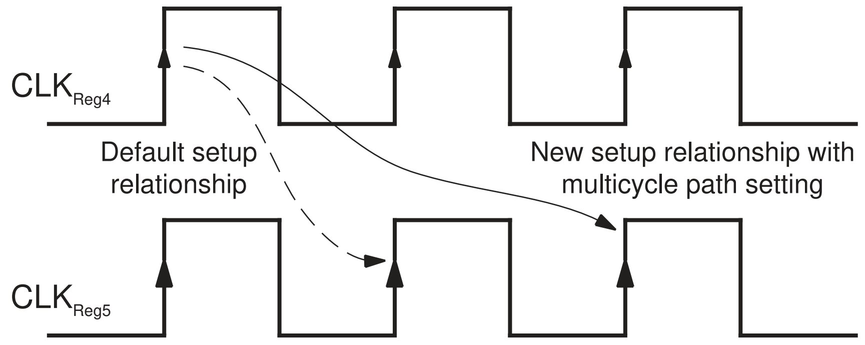
In this example, the timing path between registers named Reg4 to Reg5 is designed to take two clock cycles. By default, the tool assumes single-cycle timing for all paths. Therefore, you need to specify a timing exception for this path by using the following command:

To set the multicycle path for this block, use the following command:

```
fc_shell> set_multicycle_path -setup 2 \
          -from [get_cells Reg4] -to [get_cells Reg5]
```

This command specifies that the path takes two clock cycles, establishing the new setup relationship as shown in the following figure. The second capture edge following the launch edge becomes the applicable edge for the end of the path.

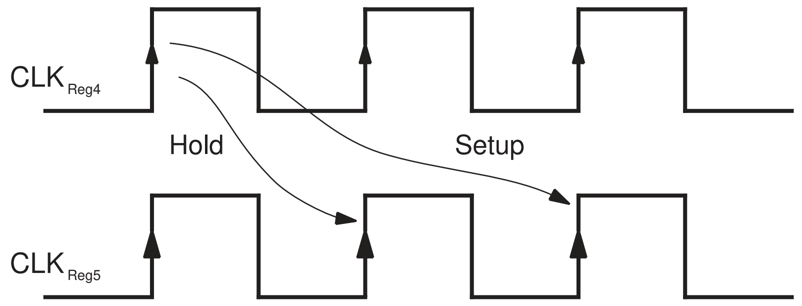
Figure 23 Multicycle Path Setup


CLK_{Reg4}
Default setup relationship
New setup relationship with multicycle path setting
CLK_{Reg5}

```
fc_shell> set_multicycle_path -setup 2 \
          -from [get_cells Reg4] -to [get_cells Reg5]
```

Changing the setup relationship implicitly changes the hold relationship as well because all hold relationships are based on the valid setup relationships. The tool verifies that the data launched by the setup launch edge is not captured by the previous capture edge. The new hold relationship is shown in the following figure.

Figure 24 Multicycle Path Hold Based on New Setup



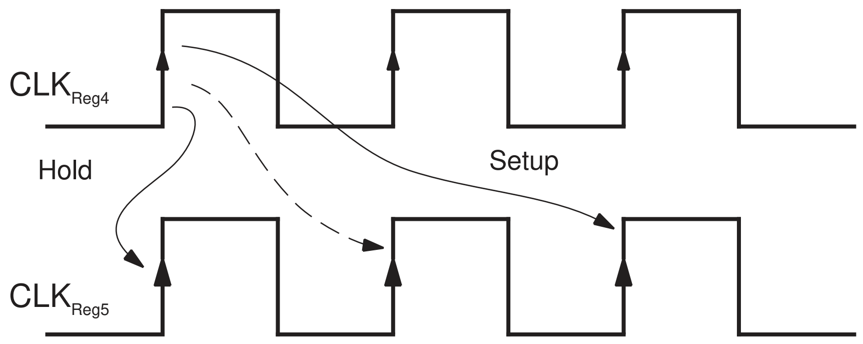
```
fc_shell> set_multicycle_path -setup 2 \
          -from [get_cells Reg4] -to [get_cells Reg5]
```

The hold relationship shown in this figure is probably not the correct relationship for the design. If Reg4 does not need to hold the data beyond the first clock edge, you need to specify a multicycle hold timing exception, as shown by the following commands:

```
fc_shell> set_multicycle_path -setup 2 \
          -from [get_cells Reg4] -to [get_cells Reg5]
fc_shell> set_multicycle_path -hold 1 \
          -from [get_cells Reg4] -to [get_cells Reg5]
```

The following figure shows the setup and hold relationships set correctly with these two `set_multicycle_path` commands. The second `set_multicycle_path` command moves the capture edge of the hold relationship backward by one clock cycle, from the dashed-line arrow to the solid-line arrow.

Figure 25 Multicycle Path Hold Set Correctly



```
fc_shell> set_multicycle_path -setup 2 \
          -from [get_cells Reg4] -to [get_cells Reg5]
fc_shell> set_multicycle_path -hold 1 \
          -from [get_cells Reg4] -to [get_cells Reg5]
```

The tool determines the number of hold cycles as follows:

(hold cycles) = (setup option value) – 1 – (hold option value)

By default, the hold cycles = 1 – 1 – 0 = 0. For this example the hold cycles = 2 – 1 – 1 = 0, which gives the default hold behavior.

Specifying Path Timing Margin Exceptions

You can make the timing check of a specified path less or more restrictive by a given amount of time by using the `set_path_margin` command.

When use the `set_path_margin` command, you must specify

- The margin value

  A positive value results in a more restrictive or tighter check. A negative value results in a less restrictive or looser check.

- The path you want to constrain by using one or more of the following methods:
  - Specify the startpoint of the path by using the `-from`, `-rise_from`, or `-fall_from` option.

    Valid startpoints are

    - Clocks

      If you specify a clock, timing paths starting from all the sequential cells and primary ports constrained by that clock are set as false paths.
    - Sequential cells
    - Clock pins of a sequential cells
    - Data pins of a latches
    - Input or inout ports
    - Pins with an input delay settings
  - Specify one or more throughpoints of the path by using the `-through`, `-rise_through`, or `-fall_through` option one or more times.

    Valid throughpoints are

    - Pins
    - Ports
    - Cells
    - Nets
  - Specify the endpoint of the path by using the `-to`, `-rise_to`, or `-fall_to` option.

    Valid endpoints are

    - Clocks

      If you specify a clock, timing paths ending at all the sequential cells and primary ports constrained by that clock are set as false paths.
    - Sequential cells
    - Data pins of sequential cells
    - Output or inout ports
    - Pins with an output delay settings

Optionally, you can limit the path margin setting to

- Setup or hold analysis only by using the `-setup` or `-hold` options.

  By default, it applies to both setup and hold analysis.
- Rise or fall analysis only by using the `-rise` or `-fall` options.

  By default, it applies to both rise and fall analysis.

To remove a path margin setting, use the `set_path_margin -reset_path` command.

The following example makes all setup checks from cell Reg4 to cell Reg5 more restrictive by 1.2 time units.

```
fc_shell> set_path_margin 1.2 \
   -from [get_cells Reg4] -to [get_cells Reg5]
```

The specified margin applies to a timing check in addition to any minimum-delay, maximum-delay, and multicycle path exceptions set on the same paths. A false path exception applied to the same path has priority over a path timing margin exception, causing the margin setting to be ignored.

Specifying Exceptions Efficiently

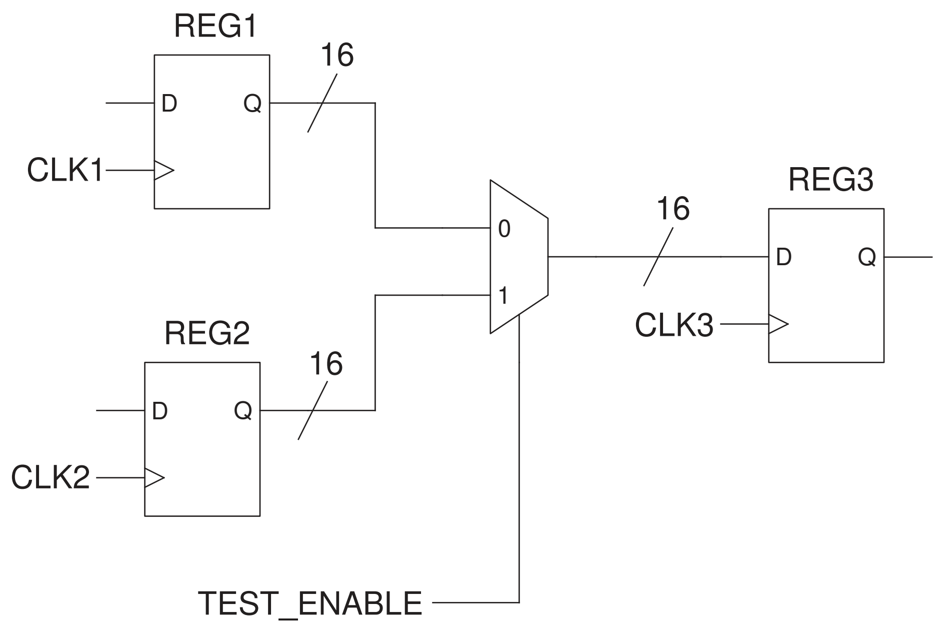
In many cases, you can specify the exception paths many different ways. To reduce the runtime during timing analysis, use the following guidelines when specifying exception:

- Look at the root cause that makes the exceptions necessary and find the simplest way to control the timing analysis for the affected paths.
- Before using false paths, consider
  - Setting a case analysis by using the `set_case_analysis` command.
  - Declaring an exclusive relationship between clocks by using the `set_clock_groups` command.
  - Disabling analysis of part of the design by using the `set_disable_timing` command.

  These alternatives can be more efficient than using the `set_false_path` command.
- If you must set false paths, avoid specifying a large number of paths using the `-through` option or wildcards.

For example, consider the circuit shown in the following figure. The three 16-bit registers are clocked by three different clocks. Each register represents 16 flip-flops. Register REG2 is only used for test purposes, so all paths from REG2 to REG3 are false paths during normal operation of the circuit.

Figure 26 Multiplexed Register Paths



To ignore the timing from REG2 to REG3, you can use any of the following methods:

- Use case analysis to consider the case when the test enable signal is 0.
- Set an exclusive relationship between the CLK1 and CLK2 clock domains.
- Declare the paths between clock domains CLK2 and CLK3 to be false.

  ```
  fc_shell> set_false_path \
            -from [get_clocks CLK2] -to [get_clocks CLK3]
  ```

  This method is an efficient way to specify the false paths because the tool only needs to keep track of the specified clock domains. It does not have to keep track of exceptions on registers, pins, nets, and so on.

- Declare the 16 individual paths from REG2 to REG3 to be false.

  ```
  fc_shell> set_false_path -from [get_pins REG2[0]/CP] \
            -to [get_pins REG3[0]/D]
  fc_shell> set_false_path -from [get_pins REG2[1]/CP] \
            -to [get_pins REG3[1]/D]
  fc_shell> ...
  ```

  This method is less efficient because the tool must keep track of timing exceptions for 16 different paths.

- Declare all paths from REG2 to REG3 to be false.

  ```
  fc_shell> set_false_path -from [get_pins REG2[*]/CP] \
            -to [get_pins REG3[*]/D]
  ```

  This method is even less efficient because the tool must keep track of paths from each clock pin of REG2 to each data pin of REG3, a total of 256 paths.

- Declare all paths from REG2 to be false.

  ```
  fc_shell> set_false_path -from [get_pins REG2[*]/CP]
  ```

  This method is similar to the previous one. The tool must keep track of all paths originating from each clock pin of REG2, a total of 256 paths.

Order of Precedence for Exceptions

For every timing path in a given block, the tool identifies all the timing exceptions that are applicable for that path. If multiple timing exceptions are applicable for a particular path, the exception used for that path depends on

- The type of exception specified, as described in Precedence Based on Exception Type
- The path specification methods used when setting the exception, as described in Precedence Based on Path Specification

Precedence Based on Exception Type

The following pairs of timing exception types are not considered to be in conflict, so both settings can be valid for a path:

- Two `set_false_path` settings
- `set_min_delay` and `set_max_delay` settings
- `set_multicycle_path -setup` and `-hold` settings
- `set_path_margin` and `set_min_delay`, `set_max_delay`, or `set_multicycle_path` settings

In case of conflicts, the timing exception types have the following order of priority, from highest to lowest:

1. `set_false_path` settings
2. `set_max_delay` and `set_min_delay` settings
3. `set_multicycle_path` settings

For example, if you declare a path to be false and also set its maximum delay to some value, the false path declaration has priority. The maximum delay setting is ignored.

Precedence Based on Path Specification

A timing exception on more specific object, such as pin or port, take precedence over the same type of exceptions applied to a more general object, such as a clock.

The various path specification methods have the following order of priority, from highest to lowest:

1. `-from` *`pin`*, `-rise_from` *`pin`*, `-fall_from` *`pin`* settings
2. `-to` *`pin`*, `-rise_to` *`pin`*, `-fall_to` *`pin`* settings
3. `-through`, `-rise_through`, `-fall_through` settings
4. `-from` *`clock`*, `-rise_from` *`clock`*, `-fall_from` *`clock`* settings
5. `-to` *`clock`*, `-rise_to` *`clock`*, `-fall_to` *`clock`* settings

Here are some possible path specification combinations, listed in order of priority from highest to lowest, according to the preceding priority rules:

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

For example, assume you set the following exceptions:

```
fc_shell> set_max_delay 12 -from [get_clocks CLK1]
fc_shell> set_max_delay 15 -from [get_clocks CLK1] \
   -to [get_clocks CLK2]
```

The first command sets the maximum delay of all paths starting from CLK1. However, the second command is more specific, so it overrides the first command for paths starting at CLK1 and ending at CLK2.

Example on the Order of Precedence of Exceptions

Assume the following exception settings on a block that has four clocks named clkA, clkB, clkC, and clkD:

```
fc_shell> set_false_path \
          -from [get_clocks clkB] -to [get_clocks clkC]
fc_shell> set_multicycle_path  2 \
          -from [get_clocks clkA] -to [get_clocks clkC]
fc_shell> set_multicycle_path 3 -to [get_pins Reg53/D]
fc_shell> set_multicycle_path 3 -from [get_pins Reg24/CK]
```

For a timing path that begins at Reg24 cell, which is clocked by clkB clock, and ends at Reg32 cell, which is clocked by clkD clock, only the following exception is applicable:

```
fc_shell> set_multicycle_path 3 -from [get_pins Reg24/CK]
```

However, for a timing path that begins at Reg24 cell, which is clocked by clkB clock, and ends at cell Reg11 cell, which is clocked by clkC clock, the following two exceptions are applicable:

```
fc_shell> set_false_path \
          -from [get_clocks clkB] -to [get_clocks clkC]
fc_shell> set_multicycle_path 3 -from [get_pins Reg24/CK]
```

In this case, the `set_false_path` command has priority over the `set_multicycle_path` command, so the tool uses the false-path exception for this path.

For a timing path that begins at Reg61 cell, which is clocked by clkA clock, and ends at Reg53 cell, which is clocked by clkC clock, the following two exceptions are applicable:

```
fc_shell> set_multicycle_path  2 \
          -from [get_clocks clkA] -to [get_clocks clkC]
fc_shell> set_multicycle_path 3 -to [get_pins Reg53/D]
```

In this case, there are two `set_multicycle_path` commands with different path specifications, so the tool uses the more specific path specification, which is the multicycle path setting to the D pin of the Reg53 cell.

Disabling Timing Paths Within Lower-Level Blocks

For designs with hierarchy, you can disable timing paths that are entirely within lower-level blocks for timing analysis and optimization by using the `set_timing_paths_disabled_blocks` command.

To disable the timing paths entirely within

- Specific subblocks, list them after the command
- All subblocks, use the `-all_sub_blocks` option

To restore the disabled timing paths within subblocks, use the `remove_timing_paths_disabled_blocks` command.

Using Case Analysis

Case analysis lets you perform timing analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design.

If you set a case analysis with

- A logic constant, the tool disables paths where the constant is propagated.

  To do so, use the `set_case_analysis 0` or `set_case_analysis 1` command, as shown in the following example:

  ```
  fc_shell> set_case_analysis 0 [get_ports TEST_EN]
  ```

- A logic transition, either rising or falling, the tool eliminates certain paths by limiting the transitions considered during analysis.

  To do so, use the `set_case_analysis rising` or `set_case_analysis falling` command, as shown in the following example:

  ```
  fc_shell> set_case_analysis rising [get_ports RESET]
  ```

In addition, the tool treats the following as case analysis values also:

- Cell pins that are connected to tie-high or tie-low cells
- Cell pins that are unconnected, which the tool ties to constant values

You can control case analysis by using the following commands or application option settings:

- Propagate case analysis values through clock-gating cells by setting the `time.case_analysis_propagate_through_icg` application option to `true`.

  By default, the tool stops propagating case analysis values at clock-gating cells.

- Propagate case analysis values through sequential cells by setting the `time.case_analysis_sequential_propagation` application option to `always`.

  By default, this application option is set to `never` and the tool stops propagating case analysis values at sequential cells.

- Disable the propagation of logic constants except for those set with the `set_case_analysis` command by setting the `time.disable_case_analysis_ti_hi_lo` application option to `true`.

  By default, the tool propagates all logic constants.

- Disable the propagation of all case analysis values by setting the `time.disable_case_analysis` application option to `true`.

  By default, the tool propagates all case analysis values.

- Remove specific case analysis values by using the `remove_case_analysis` command.

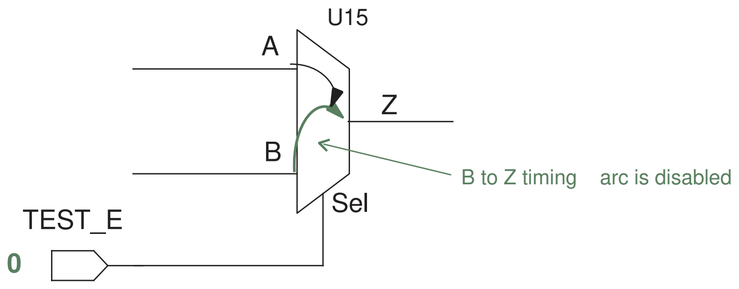
To report

- Case analysis values, use the `report_case_analysis` command.
- Timing arcs that have been disabled by various causes, including case analysis, use the `report_disable_timing` command.

The following example disables the timing path going through pins B and Z of the multiplexer named U15 shown in Figure 27 by specifying a case analysis setting of `0` on the input port named TEST_EN:

```
fc_shell> set_case_analysis 0 [get_ports TEST_EN]
```

Figure 27 Using Constants to Disable Timing Arcs



Disabling Timing Arcs

You can disable a timing path by disabling a net or cell timing arc on that path using the `set_disable_timing` command. You can disable the timing arcs of

- Leaf-level cells or library cells

  When you do so, by default, all timing arcs to the output of the cell are disabled. To disable a specific timing arc of the cell, use the `-to` and `-from` options.

For example, the following commands disable all timing arcs to the output of library cell named AND3 and the timing arc from pin A2 to pin ZN of the cell instance named U17:

```
fc_shell> set_disable_timing my_lib/AND3
fc_shell> set_disable_timing {U17} -from A2 -to ZN
```

- Leaf-level-cell pins, library-cell pins, or ports

  When you do so, all timing arcs to and from the pin or port are disable. However, you cannot disable a specific arcs by using the `-to` and `-from` options.

  For example, the following command disable all timing arcs to and from the leaf-cell pins named U37/A2 and U51/Z:

  ```
  fc_shell> set_disable_timing {U37/A U51/Z}
  ```

When you disable the timing arcs of a cell or cell pin, the tool applies a size-only attribute setting on the corresponding cell, which prevents that cell from being removed during optimization. However, it can be resized during optimization.

To report all timing arcs that are disabled in a block, including those disabled by the `set_disable_timing` command, use the `report_disable_timing` command.

Enabling Preset and Clear Timing

By default, the tool does not analyze timing paths that go through the asynchronous preset or clear input of a flip-flop and ends at the Q output of the flip-flop. To enable timing analysis and optimization on these paths, set the `time.enable_preset_clear_arcs` application option to `true`.

```
fc_shell> set_app_options -name time.enable_preset_clear_arcs \
    -value true
```

Introduction to Data-to-Data Checks

The tool can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins in the design. A timing constraint between two data (nonclock) signals is called a nonsequential constraint. This feature can be useful for checking the following types of timing constraints:

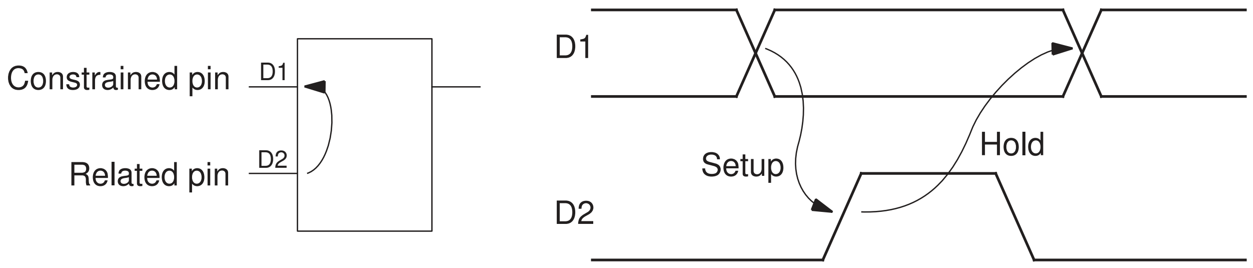
- Constraints on handshaking interface logic
- Constraints on asynchronous or self-timed circuit interfaces
- Constraints on signals with unusual clock waveforms that cannot be easily specified with the `create_clock` command

- Constraints on skew between bus lines
- Recovery and removal constraints between asynchronous preset and clear input pins

You can define such checks by using the `set_data_check` command, or they can be defined in the logic library as nonsequential constraints. You should use data checks only in situations such as those described previously. Data checks should not be considered a replacement for ordinary sequential checking.

The following figure shows a simple example of a cell that has a nonsequential constraint. The cell has two data inputs, D1 and D2. The rising edge of D2 is the active edge that might be used to latch data at D1. Pin D1 is called the constrained pin and pin D2 is called the related pin.

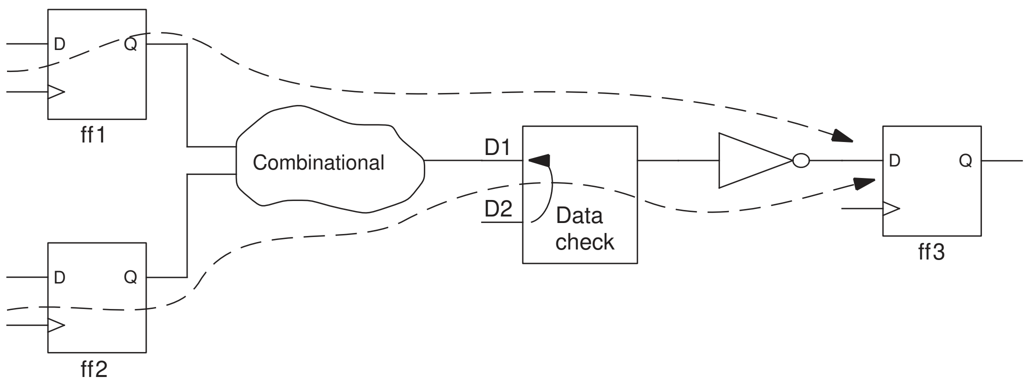
Figure 28 Simple Data Check Example



In this example, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge. If these nonsequential constraints are not already defined for the library cell, you can define them in the tool with the `set_data_check` command.

Data checks are nonsequential, so they do not break timing paths. In the following figure, the data check between pins D1 and D2 does not interrupt the timing paths shown by the dashed-line arrows. If you define the signal at D2 as a clock, the check is sequential and the paths is terminated at D1.

Figure 29 Timing Paths Not Broken by Data Checks



In a data check, signals arriving at a constrained pin or related pin can come from different clock domains. The tool checks the signal paths separately and puts them into different clock groups, just like in ordinary sequential checks.

Specifying Data-to-Data Checks

To specify that data-to-data checks be performed between two data pins, use the `set_data_check` command as follows:

- To specify a pin or port as the related pin, use the `-from`, `-rise_from`, or `-fall_from` option.
- To specify a pin or port as the constrained pin, use the `-to`, `-rise_to`, or `-fall_to` option.
- To specify that the data check value be for setup only or hold only, use the `-setup` option or `-hold` option. Otherwise, the value applies to both setup and hold.
- To specify the name of a single clock that launches the signal for the related pin, use the `-clock` option.

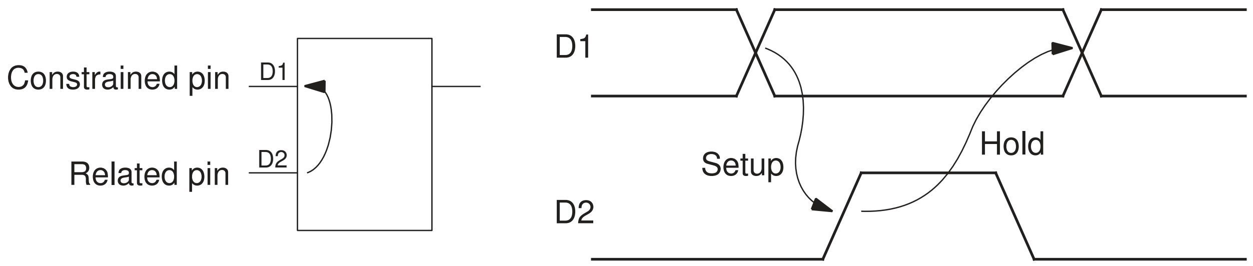
To remove data checks set with the `set_data_check` command, use the `remove_data_check` command.

Examples of Data Checks

In the following figure, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge.

Feedback

Figure 30 Data Check on Rising Edge

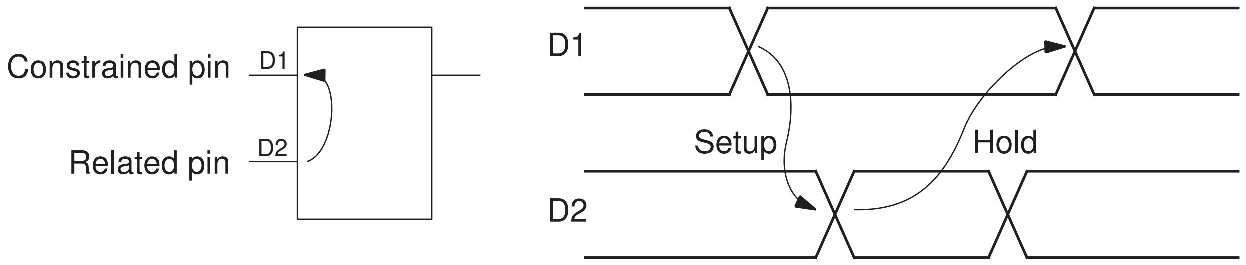


To define these data checks, use the following commands:

```
fc_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
fc_shell> set_data_check -rise_from D2 -to D1 -hold 6.0
```

In the following figure, the data checks apply to both rising and falling edges on the related pin.

Figure 31 Data Checks on Rising and Falling Edges



To define these data checks, use the following commands:

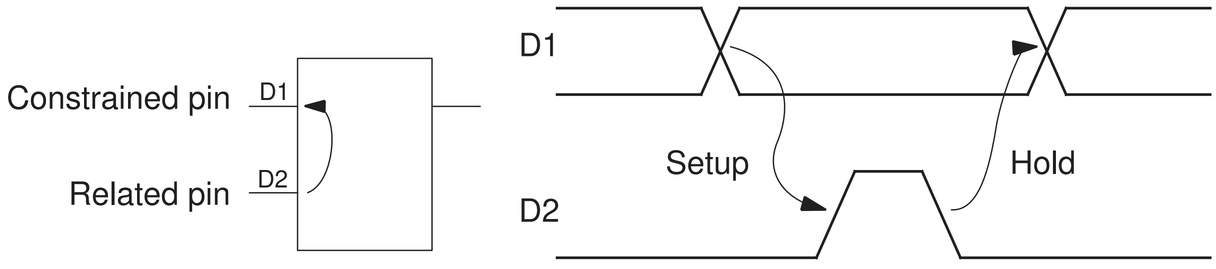
```
fc_shell> set_data_check -from D2 -to D1 -setup 3.5
fc_shell> set_data_check -from D2 -to D1 -hold 6.0
```

You can define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge, by using the following commands:

```
fc_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
fc_shell> set_data_check -fall_from D2 -to D1 -hold 3.0
```

The tool interprets this as a no-change check on a positive-going pulse. The resulting timing check is shown in the following figure.

Figure 32 No-Change Data Check



Library-Based Data Checks

The tool performs data checking for any cell that has nonsequential timing constraints defined in the library cell, if the signal at the related pin is not defined to be a clock. By default, the `time.enable_non_sequential_checks` application option is set to `true`.

If the signal at the related pin is defined as a clock, the tool ignores the library-defined nonsequential timing constraints.

To ignore the library-defined nonsequential timing constraints, set the `time.enable_non_sequential_checks` application option to `false`, as shown in the following example:

```
fc_shell> set_app_options -name time.enable_non_sequential_checks \
    -value false
```

Note:

This behavior is different from that of PrimeTime, which always honors nonsequential arc checks.

In the logic library, the nonsequential constraints are defined for a cell by specifying a related pin and by assigning the following `timing_type` attributes to the constrained pin:

- `non_seq_setup_rising`
- `non_seq_setup_falling`
- `non_seq_hold_rising`
- `non_seq_hold_falling`

For more information about defining nonsequential constraints in the logic library, see the Library Compiler documentation.

Using nonsequential constraints defined in the library cell results in a more accurate analysis than using the `set_data_check` command because the setup and hold times can be made sensitive to the slew of the constrained pin and the related pin. The `set_data_check` command is not sensitive to slew.

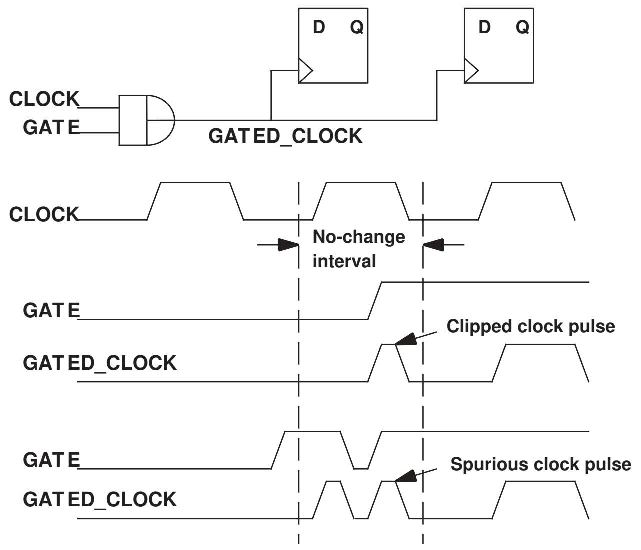
The `remove_data_check` command does not remove data checks defined in library cells.

Introduction to Clock-Gating Checks

A gated clock signal occurs when a clock network contains logic other than inverters or buffers. The tool does not automatically check setup and hold violations on the gating signals of clock-gated cells. Therefore, it is possible for these signals to undergo transitions while clock pulses are passing through the gating cells, which can lead to both clipped and spurious clock pulses.

The following figure shows examples of distorted clock waveforms that can be caused by invalid changes on the clock-gating inputs in the absence of clock-gating checks.

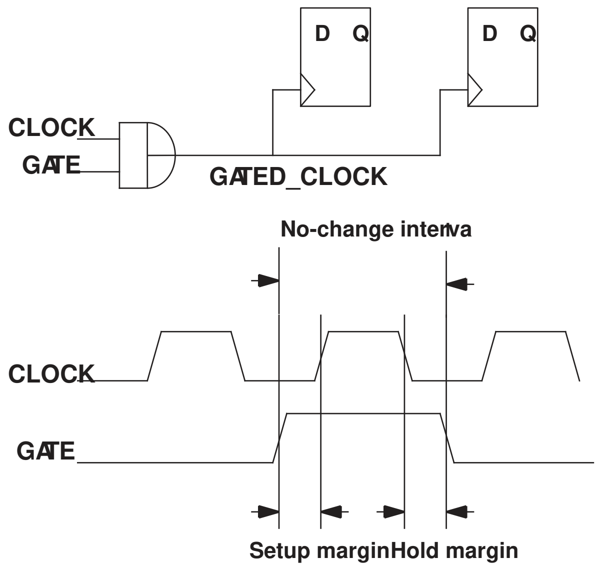
Figure 33 Distorted Clock Waveforms



To control the transition of the gating signal, you can specify a set and hold margin with respect to the clock by using the for the `set_clock_gating_check` command.

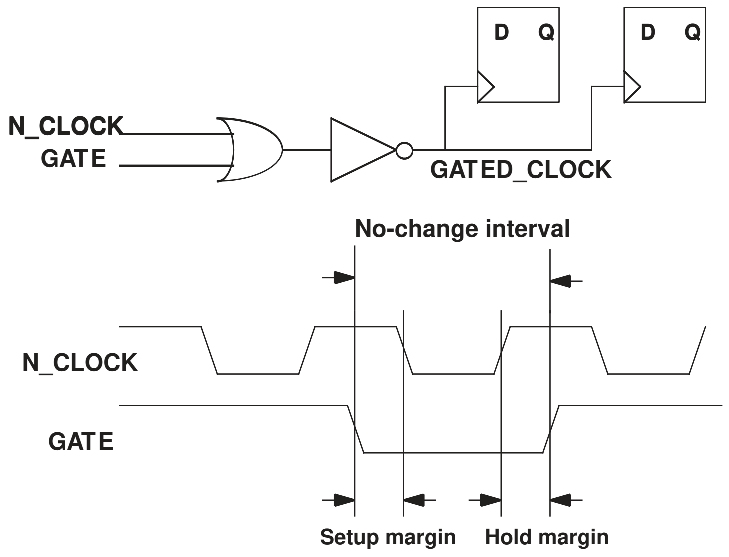
The following figure shows a clock gating check performed on a gating element that is an AND or NAND gate.

Figure 34 Clock-Gating Checks for AND and NAND Gates



The following figure, shows a clock gating check performed on a gating element that is an OR or NOR gate.

Figure 35 Clock Gating Checks for OR and NOR Gates



Specifying Clock-Gating Checks

To specify a clock gating check, use the `set_clock_gating_check` command.

When you use this command, by default, the tool

- Performs the clock gating check on all the clock gating elements of the current block.

  To limit the check to a specific cell, clock, or the transitive fanout of a port or pin, specify the required objects.
- Uses a setup and hold value of zero. To set a specific setup or hold value, use the `-setup` or `-hold` options.
- Constraints both the rising and falling edges of the gating signal.

  To control only the rising or falling edge, use the `-rise` or `-fall` options.
- Determines whether to use the high or low level of the clock to constrain the gating signal, based on the gating logic.

  If the gating cell is:

  - An AND or NAND gate, the tool performs the check on the high level of the clock.
  - An OR or NOR gates, the tool performs the check on the high level of the clock
  - Any other gate, the tool does not perform the check, unless you specify which level of the clock to use.

  To specify which level of the clock to perform the check, use the `-high` or `-low` option.

The tool handles clock-gating checks like other timing constraints and tries to adjust the delays of the logic driving the gating inputs to avoid setup and hold violations. During optimization, the tool can only size cells with clock-gating checks.

Clock-gating checks can be performed only between a clock signal and a nonclock signal, not between two clock signals or between two nonclock signals.

To remove clock-gating checks, use the `remove_clock_gating_check` command.

You can disable clock-gating checks on specific cells and pins by using the `set_disable_clock_gating_check` command to list the cells and pins you want the tool to ignore. To cancel the effect of this command, use the `remove_disable_clock_gating_check` command.

The following example shows how to specify a setup requirement of 0.2 and a hold requirement of 0.4 on all gates in the clock network of CLK1.

```
fc_shell> set_clock_gating_check -setup 0.2 -hold 0.4 CLK1
```

4 Specifying Operating Conditions

The operating conditions include the process, voltage, and temperature parameters under which the chip operates. You can specify the operating conditions for a block and the tool analyzes and optimizes the block under the conditions you specify.

By default, the tool uses on-chip variation (OCV) mode to perform timing analysis, which models the effects of variation in operating conditions across the chip. In addition, the tool supports advanced on-chip variation (AOCV), which is an optional method for improving accuracy during timing analysis.

The following topics provide information on operating conditions, on-chip variation, advanced on-chip variation, and other related concepts and tasks:

- Introduction to Clock Reconvergence Pessimism Removal
- Enabling Clock Reconvergence Pessimism Removal
- Reporting Clock Reconvergence Pessimism Removal Calculations

Specifying the Operating Conditions

The operating conditions of a block include the process, voltage, and temperature parameters under which the chip operates. The Fusion Compiler tool analyzes and optimizes the block under the conditions you specify. You can specify the same or a different set of operating conditions for both early and late analysis.

The Fusion Compiler tool supports the following two methods for specifying the operating conditions:

- Specify each of the following parameters:
    - The process factor by using the `set_process_number` command

      By default, the process factor applies to both early and late analysis. To specify value only for early or late analysis, use the `-early` and `-late` options.
    - The voltage by using the `set_voltage` command

      By default, the voltage applies to both early and late analysis. To specify a different value for early analysis, use the `-min` option.
    - The temperature by using the `set_temperature` command

      By default, the temperature applies to both early and late analysis. To specify a different value for early analysis, use the `-min` option.

  By default, these commands apply to the current corner for all objects in the block. You can specify the affected corners by using the `-corners` option. You can specify the affected objects by using the `-object_list` option.
- Specify the operating condition by using the `set_operating_conditions` command.

  The operating condition you specify
    - Applies to both early and late analysis.

      To specify an operating condition only for early or late analysis, use the `-min` or `-max` option.

  - Applies only to the current corner.

    For multicorner-multimode designs, you must specify an operating condition for each corner in the block.

A cell library can contain multiple sets of operating conditions. To get a list of the operating conditions available in a particular library and to view their characteristics, use the `report_lib` command.

In general, use only one method to specify the operating conditions for a block. If you must use both methods, for example, to set operating voltages for a multivoltage design that override the operating conditions set by the SDC file, use the `set_operating_conditions` command first, and then use the individual commands to override the specific values.

For more information, see the *Library Manager User Guide*.

Creating and Using Scaling Groups

By default, timing analysis and optimization are performed only at the characterization points (process, voltage, and temperature) of the logic libraries loaded in memory. To enable timing analysis and optimization at intermediate voltage and temperature values, you must create a scaling group, which defines the set of libraries with the same process setting that is used for interpolation.

Creating Scaling Groups

To create a scaling group, use the `define_scaling_lib_group` command to specify the logic libraries in the scaling group.

For example, assume that you load the following four logic libraries, which are generated for the same process:

- typ_v1t1.db, which is characterized at 1.0V and 0°C
- typ_v1t2.db, which is characterized at 1.0V and 125°C
- typ_v2t1.db, which is characterized at 0.8V and 0°C
- typ_v2t3.db, which is characterized at 0.8V and 125°C

To create a scaling group using these logic libraries, use the following command:

```
fc_shell> define_scaling_lib_group \
   {typ_v1t1.db typ_v1t2.db typ_v2t1.db typ_v2t2.db}
```

By default, the `define_scaling_lib_group` command

- Searches all reference libraries in memory to locate the specified logic libraries. To restrict the search to specific libraries, use the `-reflib` option.
- Names the scaling group slg*n*, where *n* is a unique integer. To specify a name for the scaling group, use the `-name` option.

The scaling groups are saved into and restored from the design library, not the reference libraries. If you change the reference libraries associated with a design library, the scaling groups are removed from the design library.

When generating the settings of a block with the `write_script` command, you can include or exclude scaling group settings by specifying the `scaling_lib_groups` value with the `-include` or `-exclude` option. For example, to output only the scaling group settings, use the following command:

```
fc_shell> write_script -include scaling_lib_groups \
     -output  blk1_slg
```

To remove scaling groups, use the `remove_scaling_lib_group` command.

Using Scaling Groups

By default, all scaling groups can be used in all corners, for both minimum and maximum delay analysis. However, you can control the usage of scaling groups with the `set_scaling_lib_group` command.

- To apply specific scaling groups to specific corners, use the `-corners` option.

  If you do not specify this option, the current corner is used. The following example applies the SG1 scaling groups to the C1 corner:

  ```
  fc_shell> set_scaling_lib_group -corners C1 SG1
  ```

- To apply all or none of the scaling groups to specific corners, use the `-all` or `-none` option with the `-corners` option.

  The following example prevents the use of all scaling groups in the C2 corner:

  ```
  fc_shell> set_scaling_lib_group -corners C2 -none
  ```

- To apply specific scaling groups for minimum or maximum delay analysis, use the `-min` or `-max` option.

The `report_corners` command reports the scaling groups available in each corner.

Using Process Labels for PVT Matching

Cells in a cell library can be characterized for one or more operating conditions. When performing timing analysis on a gate-level cells in a block, if the operating conditions specified for the block exactly matches those available for the cell in the cell library, the tool uses that data. Otherwise, the tool uses the closest match.

You can use process labels to guide the tool to select the appropriate timing models from a cell library. Process labels are applied to the characterization points in a cell library during library preparation, as shown in the following example:

```
icc2_lm_shell> read_db -process_label fast fast_lib_0p95v125c.db
```

To guide the Fusion Compiler tool to use a specific characterization point for a corner, use the `set_process_label` command to specify the process label for a corner, as shown in the following example:

```
fc_shell> create_corner c_fast
fc_shell> set_process_label -corners c_fast fast
```

Reporting PVT Information

To report PVT information, including mismatches between the operating conditions specified for corners and the timing views of the cells used in the block, use the `report_pvt` command. The report includes the specified PVT settings and the settings that are used when an exact match is not found.

In addition to the `report_pvt` command, you can access information on the PVT values used for the current block by querying the timing attributes shown in the following table.

Table 3 Operating Condition Attributes

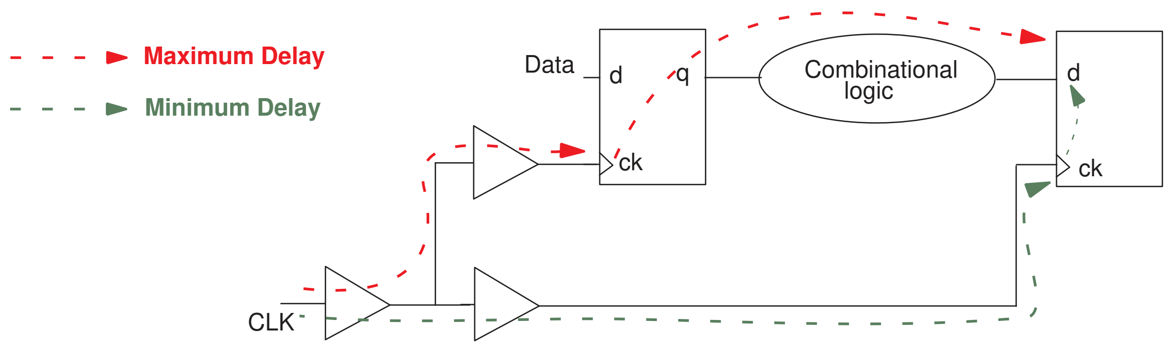
Operating condition setting	Specified values	Resolved values	Mismatch count
Process label	`process_label_early` `process_label_late`	`effective_process_label_early` `effective_process_label_late`	`process_label_mismatches`
Process number	`process_number_early` `process_number_late`	`effective_process_number_early` `effective_process_number_late`	`process_number_mismatches`
Voltage	`voltage_early` `voltage_late`	`effective_voltage_early` `effective_voltage_late`	`voltage_mismatches`
Temperature	`temperature_early` `temperature_late`	`effective_temperature_early` `effective_temperature_late`	`temperature_mismatches`

On-Chip Variation Delay Analysis

During timing analysis, the tool uses the on-chip variation (OCV) mode to perform timing, which models the effects of variation in operating conditions across the chip. This mode performs a conservative timing analysis by simultaneously applying minimum and maximum delays to different paths.

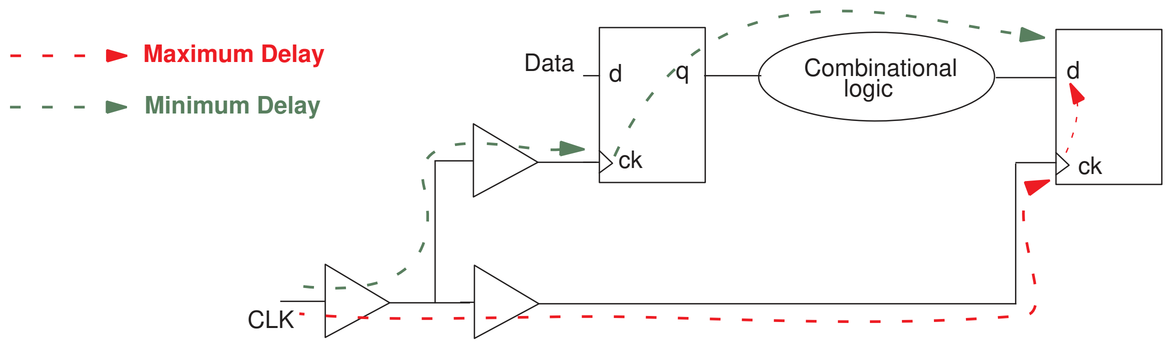
For a setup check, the tool uses maximum delays for the launch clock path and datapath and minimum delays for the capture clock path, as shown in the following figure.

Figure 36 On-Chip Variation for Setup Analysis



For a hold check, the tool uses minimum delays for the launch clock path and datapath and maximum delays for the capture clock path.

Figure 37 On-Chip Variation for Hold Analysis



The following table shows the clock arrival times, delays, operating conditions, and derating used for setup checks and for hold analysis.

Table 4 Timing Parameters Used for Setup and Hold Analysis

Analysis type	**Launch clock path**	**Data path**	**Capture clock path**
Setup	Late clock, maximum delay in clock path, late derating, worst-case operating condition	Maximum delay, late derating, maximum operating condition	Early clock, minimum delay in clock path, early derating, minimum operating condition
Hold	Early clock, minimum delay in clock path, early derating, best-case operating condition	Minimum delay, early derating, minimum operating condition	Late clock, maximum delay in clock path, late derating, maximum operating condition

During timing analysis, the tool simultaneously considers the minimum and maximum values specified for the following design parameters:

- Input and output external delays
- Port wire load models
- Port fanout number
- Net capacitance
- Net resistance
- Net wire load model
- Clock latency
- Clock transition time
- Input port driving cell

Specifying the Timing Derating Factors

Timing derating factors model the effects of varying operating conditions by adjusting the delay values calculated for the individual timing arcs of a block. By default, the timing derating factors are 1.0 and the tool does not adjust the calculated delay values.

To set derating factors, use the `set_timing_derate` command and specify the following information:

- The derating factor
- Whether the derating factor is for early or late delays by using the `-early` or `-late` options.

Optionally, you can apply the derating factor to

- Specific leaf-level instance, hierarchical instance, or library cell by specifying the object.

  By default, it applies to the current block.
- Rise or fall delays only by using the `-rise` or `-fall` options.

  By default, it applies to both rise and fall delays.
- Clock or data paths only by using the `-clock` or `-data` options.

  By default, it applies to both clock and data paths.
- Net delays, cell delays, or cell timing checks by using the `-net_delay`, `-cell_delay`, or `-cell_check` option.

  By default, it applies to all three.

  When you specify timing derating factors for net delays by using the `set_timing_derate -net_delay` command, it applies to both the dynamic and static components of the net delay. You can specify a different derating factor for the dynamic and static components by using the `-dynamic` and `-static` options.
- A specific corner by using the `-corners` option.

  By default, it applies to the current corner.

To apply an increment to an existing timing derate, use the `-increment` option. This increment is added to the existing derating factor.

The following example reduces all minimum delays by 10 percent and increases all maximum delays by 20 percent for the current corner:

```
fc_shell> set_timing_derate -early 0.9 -late 1.2
```

To report the derating factors, use the `report_timing_derate` command. By default, the command reports the derating factors for all corners. To report the derating factors for specific corners, use the `-corners` option.

To reset the derating factors to 1.0, use the `reset_timing_derate` command. By default, the command resets the derating factors for the current corner for the current block and all its cell instances. To reset the derating factors for specific corners, use the `-corners` option. To reset the derating factors for specific objects, specify the objects.

Introduction to AOCV

Advanced on-chip variation (AOCV) is an optional method for improving accuracy by using varying derating factors for different paths based on the path depth or physical distance spans. A path that has more levels of logic or covers a greater physical distance tends to have less total variation because the random variations from gate to gate tend to cancel each other out. Accordingly, AOCV applies higher derating values to short paths and lower derating values to long paths.

This method is less pessimistic than a conventional OCV analysis, which relies on constant derating factors that do not consider path-specific metrics. The improved timing accuracy affects both timing reports and design optimization.

Fast Path-Based AOCV

The Fusion Compiler AOCV flow consists of the following high-level steps:

1. Setup for AOCV analysis, as described in Setting Up for AOCV Analysis.
2. Apply AOCV data, as described in Applying AOCV Data.
3. Verify AOCV data, as described in Verifying AOCV Data.
4. Perform timing analysis and optimization using the AOCV data.

Setting Up for AOCV Analysis

Enable AOCV analysis by setting the `time.aocvm_enable_analysis` application option to `true`.

You can change the default behavior of AOCV analysis by using the following application option settings:

- Change how path depth is calculated during AOCV analysis by using the `time.aocvm_analysis_mode` application option, which has the following three settings:
  - `separate_launch_capture_depth`

    This setting, which is the default, specifies the tool to use both the clock and data network object and calculates a separate depth for capture and launch paths.
  - `combined_launch_capture_depth`

    This setting specifies the tool to use both the clock and data network object and calculates a combined depth for the entire path, considering both the capture and launch paths.
  - `separate_data_and_clock_metrics`

    This setting specifies the tool to calculate a separate depth for the clock and data paths.
- Consider physical distance, in addition to the gate counts, by setting the `time.ocvm_enable_distance_analysis` application option to `true`.

  By default, the tool does not consider the physical distance of the path.
- Use only cell distance and depth metrics for deriving both cell and net AOCV derating by setting the `time.aocvm_enable_single_path_metrics` application option to `true`.
- Ignore OCV derating factors, which are specified with the `set_timing_derate` command, when AOCV derating factors are available by setting the `time.ocvm_precedence_compatibility` application option to `true`.

  By default, the tool considers both OCV and AOCV derate settings based on the following priority, from the highest to the lowest:

  1. OCV derate setting on the leaf cell
  2. AOCV derate setting on the library cell
  3. OCV derate setting on the library cell
  4. AOCV derate setting on the hierarchical cell
  5. OCV derate setting on the hierarchical cell
  6. AOCV derate setting on the design
  7. OCV derate setting on the design

Applying AOCV Data

During AOCV timing analysis, the tool uses the AOCV data to derive derating values based on the number of successive gates in the path (the path depth) and optionally, the physical distance span of the path.

You can provide the AOCV data using the following two methods:

- Using library-based AOCV data in the logic (.db) library
- Using file-based AOCV data that you can read into the tool by using the `read_ocvm` command

  By default, the AOCV data is applied only to the current corner. To apply the data to different corners, specify them using the `-corners` option. The following example applies the AOCV data to all corners of the current block:

  ```
  fc_shell> read_ocvm -corners [all_corners] IST.aocvm
  ```

Both library-based and file-based AOCV data can be generated by the Synopsys SiliconSmart characterization tool. For details, see the *SiliconSmart ACE User Guide*, available on SolvNetPlus.

You can use both library-based and file-based AOCV derating data at the same time, and the following priority is used to resolve any conflicts, starting from the highest to the lowest:

1. File-based AOCV derate setting on the library cell
2. File-based AOCV derate setting on the hierarchical cell
3. File-based AOCV derate setting on the design

You can specify the following additional AOCV information:

- A guard band for AOCV derating factors by using the `set_timing_derate -aocvm_guardband` command.

  The following example specifies a guard band of 5 percent for early and late delays:

  ```
  fc_shell> set_timing_derate -aocvm_guardband -early 0.95
  fc_shell> set_timing_derate -aocvm_guardband -late 1.05
  ```

- A depth adjustment factor at the library cell or cell instance level by using the `set_aocvm_coefficient` command.

By default, the tool considers all cells to have a depth of one, when calculating the total depth (levels of logic) of a timing path. You can increase or decrease the depth for a specific library cell by using this command. The following example increases the depth of a library cell named MGX16 to 1.5:

```
fc_shell> set_aocvm_coefficient 1.5 \
   [get_lib_cells lib_fast/MGX16]
```

To remove some or all AOCV derating values from objects in the design, use the `remove_ocvm` command.

File-Based AOCV Data

The AOCV data file specifies the derating values for cells, library cells, or nets as a function of the path depth, and optionally, the physical distance spanned by the path. The Fusion Compiler tool accepts both binary and compressed data files produced by the `write_binary_aocvm` command in the PrimeTime tool.

The following table shows the syntax definition for the AOCV file format.

Table 5 AOCV File Format Syntax

Field specifier	Field description
`version`	On-chip variation data file version number.
`ocvm_type`	`aocvm` This field is supported only in version 4.0 or later.
`object_type`	`design | cell | lib_cell`
`rf_type`	`rise | fall | rise fall`
`delay_type`	`cell | net | cell net`
`derate_type`	`early | late`
`path_type`	`clock | data | clock data` This field is supported only in version 2.0 or later.
`object_spec`	A string that specifies the object name. The string can be an expression that is evaluated based on the attributes of the object, similar to the `regexp` Tcl command within commands that create collections. If the `object_type` is `design`, leave this field blank.

Table 5 AOCV File Format Syntax (Continued)

Field specifier	Field description
`voltage`	A floating-point value. An optional field that allows AOCV derate values to be defined for a specific voltage. If the operating voltage of the corner falls within two of the voltage specific tables, the tool scales the derate values.
`depth`	A set of M floating-point values, where M can be zero.
`distance`	A set of N floating-point values, where N can be zero. Distance values are used only when the `time.ocvm_enable_distance_analysis` application option is set to `true`.
`table`	A set of N x M floating-point values. There are also the following special cases: • If N==0, the table is of size M. • If M==0, the table is of size N. • If M==0 and N==0, the table is of size 1. Linear interpolation is used to determine points that have not been defined in the table. The tool does not extrapolate beyond the lowest or highest values specified in the table.

To add a comment in any location within the file, use double forward slashes (`//`).

The following example AOCV data file sets derates values for the cells in the clock paths of the design:

```
version          4.0
ocvm_type:        aocvm
object_type:      design
rf_type:          rise fall
delay_type:       cell
derate_type:      late
path_type:        clock
object_spec:
voltage:          1.2
depth:            1  2  3
distance:         100  1000
table:            1.21  1.11  1.09 \
                  1.23  1.16  1.14
```

The AOCV data in the file is specified for 1.2v.

Verifying AOCV Data

To view AOCV derate table data, use the `report_ocvm` command. You can view design objects annotated with early, late, rise, fall, cell, or net derate tables. You can also use this command to determine cells and nets that are annotated or not annotated with AOCV information.

For example, to get the AOCV derate data for the timing arc from pin A to pin Z of cell U57, use the following command:

```
fc_shell> report_ocvm -type aocvm \
  [get_timing_arcs -from U57/A -to U57/Z]
```

To get the AOCV derate data for the library cell liba/inv2x, use the following command:

```
fc_shell> report_ocvm -type aocvm \
  [get_lib_cells liba/inv2x]
```

You can report the derate values used to calculate a specific cell or net delay arc by using the `-derate` option with the `report_delay_calculation` command.

Introduction to Parametric On-Chip Variation Analysis

Parametric on-chip variation (POCV) analysis is an optional timing analysis mode that takes a statistical approach to modeling on-chip variation. In this mode, the tool computes arrival time, required time, and slack as statistical distributions rather than fixed minimum and maximum values. The timing results are less pessimistic, providing a better basis for timing optimization.

In POCV timing analysis, instead of specifying absolute minimum and maximum delays for each timing arc, the tool calculates the delay as a function of the Gaussian or normal distribution *P*:

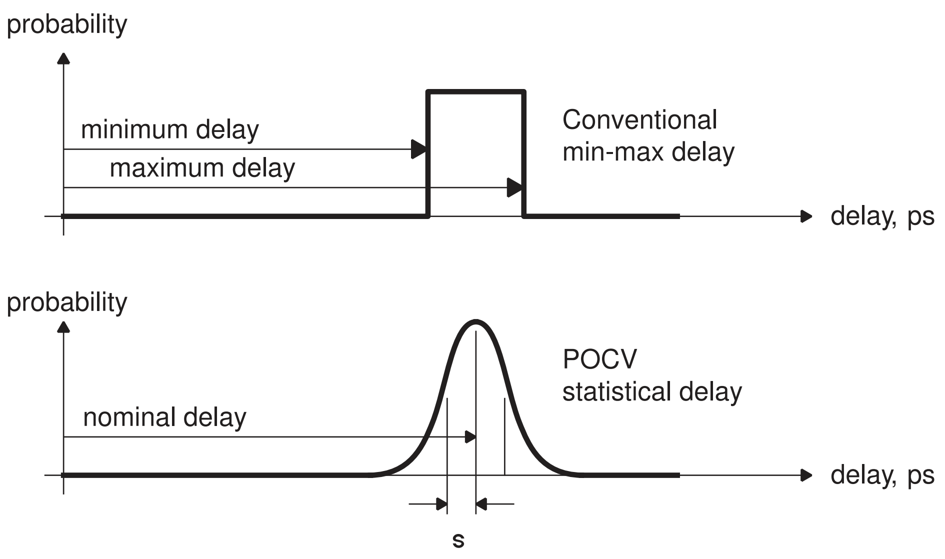
$delay = nominal\_delay + s * P$

$= nominal\_delay + (C * nominal\_delay) * P$

The tool gets the delay variation value σ (sigma) from the logic library, if available, or calculates it from a coefficient *C* supplied in a POCV data file, where σ = *C* * *nominal_delay*.

The following figure graphically compares conventional min-max delay with POCV statistical delay.

Figure 38 Comparison of Min-Max Delay and POCV Statistical Delay



The differences between these two delay variations are:

- The min-max delay specifies the absolute minimum and maximum delay values for the timing arc. The actual delay has an equal probability of occurring anywhere between these two extremes, and no chance of occurring beyond these extremes.
- The POCV statistical model specifies a nominal delay value and a variation σ . The delay has the highest probability of occurring at the nominal value and smaller probabilities of occurring farther away from the nominal value. The actual delay has a 99.7 percent chance of falling within 3σ of the nominal delay.

After the tool determines the delay distribution of each timing arc, it propagates them statistically through timing paths to calculate each arrival time, required time, and slack value as a total nominal value plus a cumulative variation.

To determine the cumulative delay of a path, the tool statistically combines the delay distribution of each stage. This is more accurate than simply adding the worst-case value from each stage. The resulting delay and slack values are more realistic and less pessimistic than values calculated by simple min-max addition.

By default, the final slack reported by the `report_timing` command is the slack at three standard deviations (3σ) less than the nominal slack:

reported_slack = *nominal_slack* – (3σ)

You can change the sigma multiplier from 3.0 to a larger or smaller value to tighten or loosen the timing constraint.

Application OptionsVariables and Commands for Parametric On-Chip Variation

The following tables summarize the application options variablesand commands for POCV.

Table 6 Application OptionsVariables for POCV Analysis

Application OptionVariable	Description
`parasitics_enable_tail_annotation`	Enables reading of tail annotation data from parasitic data files for via variation analysis
`time.timing_enable_constraint_variation`	Enables constraint variation for setup and hold constraints
`time.timing_enable_slew_variation`	Enables slew variation for cell delays
`time.timing_enable_via_variation`	Enables physical via variation analysis
`time.timing_enable_via_variation`	Enables via variation analysis using data read with the `read_ivm` command
`time.ocvm_enable_distance_analysis`	Enables distance-based derating during POCV analysis
`time.timing_pocvm_corner_sigma`	Specifies the corner sigma for POCV delay constraint analysis
`time.timing_pocvm_enable_analysis`	Enables POCV analysis
`time.timing_pocvm_enable_extended_moments`	Enables usage of asymmetric moment-based modeling data
`timing_pocvm_max_transition_sigma`	Specifies the corner sigma for POCV maximum transition time analysis
`time.timing_pocvm_precedence`	Specifies the order of priority between file-based and library-based POCV coefficients
`timing_pocvm_report_sigma`	Specifies the sigma for reporting
`timing_use_slew_variation_in_constraint_arcs`	Enables use of slew variation in constraint arc variation computation (requires that `timing_enable_constraint_variation` be set to `true`)

Table 7 Commands for POCV Analysis

Command	**Description**
`read_ocvm` *`pocvm_file`*	Reads POCV tables
`read_ivm` *`via_file`*	Reads via variation tables
`report_delay_calculation -derate`	Reports details about derating for delay calculation
`report_ocvm -type pocvm`	Displays POCV information, including POCV coefficient and distance-based derating table data
`report_ivm`	Displays via variation table data
`report_timing -derate`	Reports POCV information in timing report
`report_timing_derate -pocvm_coefficient_scale_factor`	Reports POCV scaling
`report_timing_derate -pocvm_guardband`	Reports POCV guard banding
`reset_timing_derate -pocvm_coefficient_scale_factor`	Removes POCV scaling
`reset_timing_derate -pocvm_guardband`	Removes POCV guard banding
`set_timing_derate -cell_check`	Specifies the derate factor for cell timing checks
`set_timing_derate -pocvm_coefficient_scale_factor`	Specifies POCV coefficient scaling
`set_timing_derate -pocvm_guardband`	Specifies POCV guard banding
`set_timing_derate -pocvm_subtract_sigma_factor_from_nominal`	Specifies a POCV sigma factor used to subtract sigma from nominal

POCV Data Formats

To perform POCV timing analysis, the Fusion Compiler tool needs the variation σ for each timing arc. There are two ways to provide this information:

- Library-based POCV data

  The tool supports logic-library-based POCV variation data for delay, transition, and constraint arcs in the Liberty Variation Format (LVF).

- For a delay arc, each LVF table entry specifies the delay variation σ for a combination of input slew and output load.
- For a transition arc, each LVF table entry specifies the transition variation σ for a combination of input slew and output load.
- For a constraint arc, each LVF table entry specifies the constraint variation σ for a combination of the transition of the constrained pin and the related pin.

Note:

If your logic libraries contain LVF tables, ensure that the Fusion Compiler cell libraries are created using the K-2015.06-SP1 or later versions of the Library Manager tool. The LVF information is included in the cell libraries and read into your Fusion Compiler session automatically.

- File-based POCV data

  You read in a coefficient data file for a specific corner in plain text format using the `read_ocvm` command. Coefficient files contain delay variation data, but do not support transition or constraint variation data. The file applies a single coefficient value *C* for each library cell, hierarchical cell, or design specified in the text file. The delay variation σ is calculated as *C* * *nominal_delay*. All the timing arcs of a cell share the same coefficient *C*, irrespective of input slew and output load.

Both library-based and file-based POCV data can be generated by the Synopsys SiliconSmart characterization tool. For details, see the *SiliconSmart ACE User Guide*, available on SolvNetPlus.

File-Based POCV Data

The POCV data file specifies the delay variation coefficients *C* for the design, for hierarchical cells in the design, or library cells used in the design. The tool accepts both binary and compressed data files generated by the `write_binary_aocvm` command in PrimeTime. For transition and constraint variation data, the LVF library-based data must be used. POCV analysis always applies to both data paths and clock paths.

The following table shows the syntax definition for the POCV file format.

Table 8 POCV File Format Syntax

Field specifier	Field description
`version`	On-chip variation data file version number, which must be version 4.0 or later.
`ocvm_type`	`pocvm`
`object_type`	`design | cell | lib_cell`

Table 8 POCV File Format Syntax (Continued)

Field specifier	Field description
`rf_type`	`rise | fall | rise fall`
`delay_type`	`cell | net | cell net`
`derate_type`	`early | late`
`path_type`	`clock | data | clock data`
`object_spec`	A string that specifies the object name. The string can be an expression that is evaluated based on the attributes of the object, similar to the `regexp` Tcl command within commands that create collections. If the `object_type` is `design`, leave this field blank.
`voltage`	A floating-point value. An optional field that allows POCV data to be defined for a specific voltage. If the operating voltage of the corner falls within two of the voltage specific POCV data sets, the tool scales the data.
`coefficient`	A floating-point value.

To add a comment in any location within the file, use double forward slashes (`//`).

The following example POCV data file sets coefficients for the early cell and net delays in the clock paths of the design:

```
version            4.0
ocvm_type:          pocvm
object_type:        design
rf_type:            rise fall
delay_type:         cell net
derate_type:        early
path_type:          clock
object_spec:
voltage:            1.2
coefficient:        0.5
```

The POCV data in the file is specified for 1.2v.

Precedence of File- and Library-POCV Data

You can use both file- and library-based POCV data at the same time. By default, the tool uses the following priority to resolve any conflicts, starting from the highest to the lowest:

1. File-based POCV coefficient setting on the library cell
2. File-based POCV coefficient setting on the hierarchical cell
3. File-based POCV coefficient setting on the design
4. Library-based POCV LVF data on the library cell or on the logic library

To change the default precedence of the file- and library-based POCV data, use the `time.pocvm_precedence` application option. The default is `file`.

If you set this application option to

- `library`, the tool uses the following precedence:
    1. Library-based POCV LVF data on the library cell or on the logic library
    2. File-based POCV coefficient setting on the library cell
    3. File-based POCV coefficient setting on the hierarchical cell
    4. File-based POCV coefficient setting on the design
- `lib_cell_in_file`, the tool uses the following precedence:
    1. File-based POCV coefficient setting on the library cell
    2. Library-based POCV LVF data on the library cell or on the logic library
    3. File-based POCV coefficient setting on the hierarchical cell
    4. File-based POCV coefficient setting on the design

Converting AOCV Data to POCV Data

You can use the `convert_aocv_pocv` command to convert AOCV tables annotated on design objects (previously loaded by the `read_ocvm` command) to POCV tables and apply the converted POCV tables onto the same design object.

The command supports the following options:

- Use the `-depth` option to specify the depth row entry in the AOCV table to do the conversion. By default, the first entry of the depth row in the AOCV table is used.
- Use the `-corners` option to indicate the list of corners for which the AOCV derate factors are to be converted and the converted POCV coefficients are to be applied. If this option is not provided, the corner associated with the current scenario is used.

To read AOCVM data from an existing POCVM database, use the following commands:

```
fc_shell> set_app_options -name time.aocvm_enable_analysis \
   -value false
fc_shell> set_app_options -name time.pocvm_enable_analysis \
   -value true
fc_shell> read_ocvm -corners [all_corners] ./aocv.txt
fc_shell> convert_aocv_pocv -depth 1 -corners [all_corners]
```

Performing Parametric On-Chip Variation Analysis

The tool supports POCV analysis for timing analysis and optimization. It requires file- or library-based POCV data in Liberty Variation Format (LVF). The tool supports delay, transition, and constraint LVF tables. POCV analysis can also be performed with distance-based derate tables.

If your logic libraries contain LVF tables, ensure that the Fusion Compiler cell libraries are created using the K-2015.06-SP1 or later version of the Library Manager tool. The LVF information is included in the cell libraries and read into your Fusion Compiler session automatically.

To perform POCV analysis, use the following steps:

1. Define all modes, corners, and scenarios for the block.
2. Enable POCV analysis by using the `time.pocvm_enable_analysis` application option.

   ```
   fc_shell> set_app_options -name time.pocvm_enable_analysis \
      -value true
   ```

3. (Optional) Change the number of standard deviations (σ) used to calculate the worst-case values from statistical parameters during POCV analysis as follows:
   - To change the value for the entire block, use the `time.pocvm_corner_sigma` application option.
   - To change the value for a specific corner, use the `set_pocvm_corner_sigma -corners` command.

A corner-specific value set with the `set_pocvm_corner_sigma` command has a higher precedence over a block-specific value set with the `time.pocvm_corner_sigma` application option. To report the corner-sigma value used for timing analysis, use the `report_ocvm -type pocvm -corner_sigma` command.

By default, timing analysis uses the values at 3 standard deviations (3σ) from the mean. Increasing the number of standard deviations, tightens the timing requirement.

The following example changes the number of standard deviations to 3.5 for the current block and 4 for the corner named corner1:

```
fc_shell> set_app_options -name time.pocvm_corner_sigma \
                -value 3.5 -block [current_block]
fc_shell> set_pocvm_corner_sigma -corners corner1 4.0
```

4. (Optional) If you use LVF logic libraries with slew variation data, enable the use of slew variation during POCV analysis by using the `time.enable_slew_variation` application option.

```
fc_shell> set_app_options -name \
time.enable_slew_variation -value true
```

5. (Optional) If you use LVF logic libraries with constraint variation data, enable the use of constraint variation during POCV analysis by using the `time.enable_constraint_variation` application option.

```
fc_shell> set_app_options -name \
time.enable_constraint_variation -value true
```

6. (Optional) If you use LVF logic libraries with moment-based variation models, enable the use of these models during POCV analysis by using the `time.pocvm_enable_extended_moments` application option. For more information, see Enabling Analysis With Moment-Based Modeling.

```
fc_shell> set_app_options -name \
   time.pocvm_enable_extended_moments -value true
```

7. (Optional) If you have distance-based derate tables, enable the use of distance-based derating during POCV analysis by using the `time.ocvm_enable_distance_analysis` application option and read in the distance-based derate table for each corner.

```
fc_shell> set_app_options -name \
time.ocvm_enable_distance_analysis -value true
fc_shell>  foreach_in_collection corner [all_corners] {
      current_corner $corner
      read_ocvm $corner.distance_derate.pocv
   }
```

8. (Optional) If you have POCV coefficient data files, read them by using the `read_ocvm` command.

   ```
   fc_shell> read_ocvm coefs.pocv
   ```

   If an object has both a file- and library-based POCV data, the file-based POCV data takes higher precedence by default. You can change this precedence by using the `time.pocvm_precedence` application option. For more information, see Precedence of File- and Library-POCV Data.

9. (Optional) Apply an additional guard band by using the `set_timing_derate -pocvm_guardband` command.

   You can apply the guard band to cell delays, net delays, or cell timing checks by using the `-cell_delay`, `-net_delay`, or `-cell_check` option, as shown in the following example:

   ```
   fc_shell> set_timing_derate -cell_delay -pocvm_guardband -early 0.98
   fc_shell> set_timing_derate -cell_delay -pocvm_guardband -late 1.03
   ```

   Applying a guard band increases the POCV derating effect, which makes timing analysis more restrictive.

10. (Optional) Modify the variation calculated from coefficients or the variation in LVF by using the `set_timing_derate -pocvm_coefficient_scale_factor` command.

    You can apply the scale factor to cell delays, net delays, or cell timing checks by using the `-cell_delay`, `-net_delay`, or `-cell_check` option, as shown in the following example:

    ```
    fc_shell> set_timing_derate -net_delay \
         -pocvm_coefficient_scale_factor -early 0.86
    fc_shell> set_timing_derate -net_delay \
         -pocvm_coefficient_scale_factor -late 1.16
    ```

    Applying coefficient scale factors alters the variation used in POCV analysis for the specified design objects.

11. (Optional) For cell timing checks, adjust the mean (nominal) value by a factor of the sigma (σ) value by using the `set_timing_derate -cell_check -pocvm_subtract_sigma_factor_from_nominal` command.

    When you do so, the tool adjusts the mean value of the constraint, but leaves the sigma value unchanged. The following example subtracts 0.75σ from the mean value for early cell timing checks during POCV analysis:

    ```
    fc_shell> set_timing_derate -cell_check -early \
       -pocvm_subtract_sigma_factor_from_nominal 0.75
    ```

This derate value can only be applied for cell timing checks and you must enable constraint variation during POCV analysis by setting the `time.enable_constraint_variation` application option to `true`.

This feature allows constraint variation to be computed as a linear sum, instead of the default root-sum-squared. To do so,

a. Add a factor of sigma into the mean by specifying a negative value by using the `set_timing_derate -pocvm_subtract_sigma_factor_from_nominal` command.

b. Remove the constraint sigma by setting a coefficient scale factor of zero by using the `set_timing_derate -pocvm_coefficient_scale_factor` command.

For example,

```
fc_shell> set_timing_derate -cell_check -early \
     -pocvm_subtract_sigma_factor_from_nominal -3.0
fc_shell> set_timing_derate -cell_check -early \
     -pocvm_coefficient_scale_factor 0.0
```

12. (Optional) Report the values specified with the `-pocvm_guardband`, `-pocvm_coefficient_scale_factor`, or `-pocvm_subtract_sigma_factor_from_nominal` option of the `set_timing_derate` command by using the `report_timing_derate` command.

13. (Optional) For the objects that do not have POCV data, apply OCV derating factors by using the `set_timing_derate` command.

    If both POCV and OCV data is available for the same object, the tool decides which value to use based on the following precedence, from the highest to the lowest:

    a. OCV data applied on the leaf cell

    b. POCV data applied on the library cell

    c. OCV data applied on the library cell

    d. POCV data applied on the hierarchical cell

    e. OCV data applied on the hierarchical cell

    f. POCV data applied on the design

    g. OCV data applied on the design

    To ignore OCV derating factors when POCV data is available, set the `time.ocvm_precedence_compatibility` application option to `true`.

14. (Optional) Report the design objects that are annotated with POCV coefficients by using the `report_ocvm -type pocvm` command.

The following example reports the POCV coefficients for the cell instance named U21:

```
fc_shell> report_ocvm -type pocvm [get_cells U21]
```

The following example reports the LVF data for the library cell named liba/inv2x:

```
fc_shell> report_ocvm -type pocvm \
    [get_lib_timing_arcs -from liba/inv2x/A -to liba/inv2x/Z]
```

You can report the derate values used to calculate a specific cell or net delay arc by using the `-derate` option with the `report_delay_calculation` command.

15. Perform timing analysis by using the `report_timing` command.

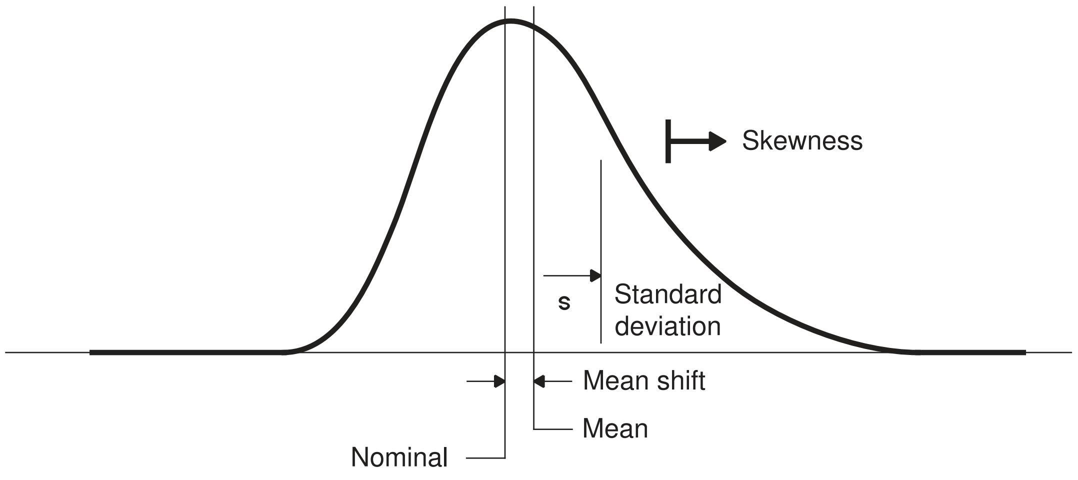
```
fc_shell> report_timing -variation -derate
```

When you use the `-variation` option, the tool prints the statistical information (mean and sigma) in the timing report.

Enabling Analysis With Moment-Based Modeling

The tool supports analysis using cell libraries containing moment-based Liberty Variation Format (LVF) delay and constraint data. This type of data more accurately models the non-Gaussian statistical variation characteristics of advanced process nodes and very low supply voltages. The following diagram shows a typical asymmetric cell delay distribution.

Figure 39 Moment-Based Asymmetric Delay Distribution



The statistical parameters for this distribution are the mean shift, standard deviation, and skewness. In LVF modeling syntax, these parameters are specified by the following attributes:

```
ocv_std_dev_*
ocv_mean_shift_*
ocv_skewness_*
```

For example, an `ocv_std_dev_cell_rise` table in the LVF model specifies the standard deviation of the cell delay distribution as a function of input transition and output load.

To enable timing analysis using moment-based modeling data, set the application option:

```
fc_shell> set_app_options -name time.pocvm_enable_extended_moments \
   -value true
```

In variation reports, the mean value reported for a parameter includes both the nominal and the mean shift adjustment.

Querying POCV Timing Path and Timing Point Attributes

Use the following timing path and timing point attributes to query the mean, variation, and skewness information from moment-based POCV analysis.

Table 9 POCV timing path and timing point attributes

Attribute	Object class	Sub-attributes
`variation_arrival`	`timing_path`, `timing_point`	`mean`, `std_dev`, `skewness`
`variation_common_path_pessimism`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_endpoint_clock_latency`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_endpoint_hold_time_value`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_endpoint_recovery_time_value`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_endpoint_removal_time_value`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_endpoint_setup_time_value`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_increment`	`timing_point`	`mean`, `std_dev`, `skewness`
`variation_required`	`timing_path`	`mean`, `std_dev`, `skewness`
`variation_slack`	`timing_path`, `timing_point`	`mean`, `std_dev`, `skewness`
`variation_startpoint_clock_latency`	`timing_path`	`mean`, `std_dev`, `skewness`

Table 9 POCV timing path and timing point attributes (Continued)

Attribute	Object class	Sub-attributes
variation_transition	timing_point	mean, std_dev, skewness

The mean sub-attribute contains “nominal + mean_shift” if moment-based LVF is enabled and ‘nominal’ when it is not.

Use the get_timing_path command to create a path collection to query these attributes:

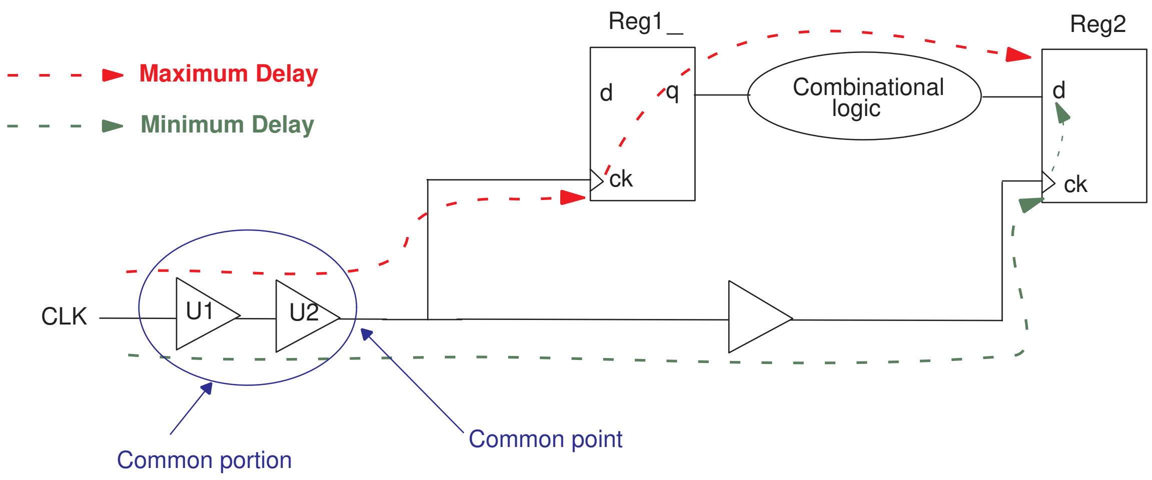
```
# timing path example
fc_shell> set path [get_timing_path -to reg_14/D]
fc_shell> get_attribute $path variation_slack.mean
0.182163

# timing point example
fc_shell> foreach_in_collection p [get_attribute $path points] {
            echo [get_object_name [get_attribute $p object]]
                 [get_attribute $p variation_arrival.mean] }
reg_11/CP 0.000000
reg_11/Q 0.046558
```

Introduction to Clock Reconvergence Pessimism Removal

Clock reconvergence pessimism (CRP) is a difference in delay along the common part of a launching and capturing clock path when you simultaneously use minimum and maximum delays during on-chip variation analysis. It is an accuracy limitation in timing analysis. Automated correction of clock reconvergence pessimism is called clock reconvergence pessimism removal (CRPR).

Consider the following figure where the launch and capture portions of the timing path from cell Reg1 to Reg2 share the clock tree until the output of the cell U2. The shared segment is called the *common portion*, consisting of cells U1 and U2 in this example. The last cell output in the shared clock segment is called the *common point*, which is the output of U1 in this case.

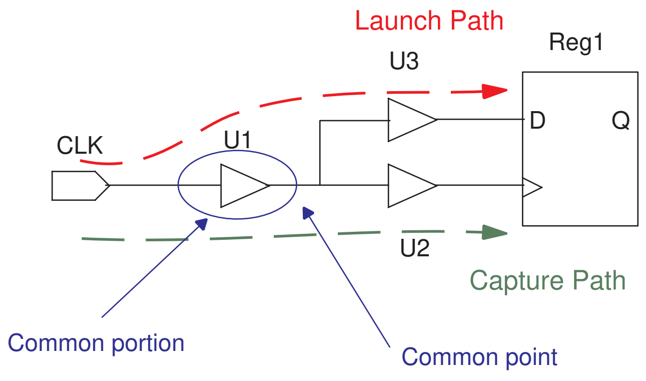
Figure 40 Clock Reconvergence Pessimism Example



During setup analysis, for the common portion, the tool uses the maximum delay for the launch path and the minimum delay capture path, resulting in clock reconvergence pessimism

Pessimism can also be introduced on paths that fan out from a clock source to the data pin of a sequential device and a portion of the clock path is shared by the launch and capture paths, as shown in the following figure.

Figure 41 Timing Path Fanout From Clock Source to Data Pin



Enabling Clock Reconvergence Pessimism Removal

To enable CRPR, set the `time.remove_clock_reconvergence_pessimism` application option to `true`. By default, it is set to `false` and CRPR is not performed.

Enabling CRPR results in a less pessimistic analysis, but increases the runtime and memory usage. Any change in this application option setting causes a complete timing update.

To further control CRPR, use the following application option settings:

- To remove pessimism on the same- or opposite-sense clock transitions of the common portion, set the `time.clock_reconvergence_pessimism` application option as follows:
    - To perform CRPR only if the clock transitions in the common portion have the same sense, set the value to `same_transition`.
    - To perform CRPR for both same- and opposite-sense clock transitions in the common portion, set the value to `normal` . This is the default.
- To remove pessimism between clock-to-data paths and clock paths, set the `time.crpr_remove_clock_to_data_crp` application option to `true`.

  The default is `false`.

Reporting Clock Reconvergence Pessimism Removal Calculations

The `report_crpr` command reports the calculation of clock reconvergence pessimism (CRP) between two register clock pins or ports. You specify the pins of the launch and capture registers, the clock, and type of check (setup or hold). For example,

```
fc_shell> report_crpr -from [get_pins ffa/CP] \
            -to [get_pins ffd/CP] -setup
```

The command generates a report that shows the

- Location of the common point
- Launch and capture clock edge types (rising or falling)
- Four calculated arrival times (early and late, and rise and fall) at the common point
- Calculated CRP values (rise and fall)
- Values used for opening-edge and closing-edge pessimism removal

The amount of CRP reported by the `report_crpr` command can be slightly different from the amount reported by the `report_timing` command. For computational efficiency, the `report_timing` command merges multiple points for CRPR calculations when the CRP differences between adjacent points are too small.

Feedback

5

Constraining Ports and Nets

Timing analysis and timing optimization depend on constraints that describe the characteristics of the ports and nets in your blocks. The following topics describe these constraints and the associated tasks:

Specifying Drive Characteristics at Input and Inout Ports

To constrain an input or inout port, you should specify a drive resistance and an input transition for it by using one of the following methods:

- Specify an external driving cell by using the `set_driving_cell` command, which the tool uses to calculate the drive resistance and input transition. In addition, the port inherits other characteristics, such as design rule constraints, from the driving cell.

  This is the more accurate representation of the drive characteristics.

- Specify a drive resistance and input transition values by using the `set_drive` and `set_input_transition` commands.

  This is the less accurate representation of the drive characteristics.

If you use both methods, the drive resistance and input transition derived based on the driving cell specified with the `set_driving_cell` command replace the values specified with the `set_drive` and `set_input_transition` commands.

To report the driving cell or the drive resistance and input transition on ports, use the `report_ports -drive` command.

Setting a Driving Cell for Ports

To constrain an input or inout port, you can model the external driver characteristics of the port by using the `set_driving_cell` command. The tool uses the timing arcs of the driving cell to calculate the drive characteristics of the port being constrained.

You must specify the following information with the `set_driving_cell` command:

- The ports you are constraining.
- The library cell for the external driver by using the `-lib_cell` option.

  When you use this option, you also specify the name of the cell library to obtain this cell from by using the `-library` option. If you do not specify this option, the tool obtains the cell from the first cell library it finds with a matching cell.

Optionally, you can specify

- A separate driving cell for rise and fall transition by using the `-rise` and `-fall` options.

  By default, the driving cell applies to both rise and fall transitions.

- A driving cell for early and late analysis by using the `-min` and `-max` options.

  By default, the driving cell applies to both early and late analysis.

- An output pin of the library cell to drive the port by using the `-pin` option.

  If the library cell you specify has multiple output pins and you do not use this option, the tool randomly picks an output pin of the cell to drive the port being constrained.

- An input pin of the driving cell for selecting a timing arc to calculate the drive characteristics by using the `-from_pin` option.

  If you do use this option, the tool randomly picks an input pin of the driving cell.

- An input rise and fall transition at the input of the driving cell for calculating the drive characteristics by using the `-input_transition_rise` and `-input_transition_fall` options.

  By default, the tool uses zero transition at the input of the driving cell.

- That the tool should not scale the drive characteristics based on the operating conditions of the block by using the `-dont_scale` option.

  By default, the tool scales the drive characteristics based on the operating conditions of the block.

- That the tool should not apply the design rules of the driving cell on the port being constrained by using the `-no_design_rule` option.

  By default, the tool applies the design rules of the driving cell on the port being constrained.

- A value to multiply the calculated transition for the port being constrained by using the `-multiply_by` option.

For designs with multiple scenarios, by default, the driving cell you specify applies only to the current scenario. To specify a driving cell for

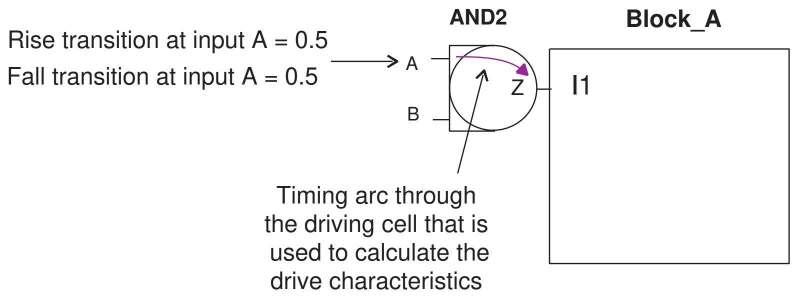
- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The following constrains the port named I2 by specifying a library cell of type AND2 as the driving cell, selecting the timing arc from its input pin named A, and specifying a rise and fall transition of 0.5 for this input pin:

```
fc_shell> set_driving_cell [get_ports I2] \
 -lib_cell AND2 -from_pin A \
 -input_transition_rise 0.5 -input_transition_fall 0.5
```

Figure 42 Specifying Input Pin and Input Transition for the Driving Cell


AND2
Block_A
Rise transition at input A = 0.5
Fall transition at input A = 0.5
A
B
Z
I1
Timing arc through the driving cell that is used to calculate the drive characteristics

Setting Drive Resistance

To constrain an input or inout port, you can model the external driver strength as a resistance value by using the `set_drive` command. The external driver is modeled as the supply voltage connected in series with the specified resistance value.

Specify the following information with the `set_drive` command:

- The ports you are constraining
- The drive resistance.

Optionally, you can specify

- A separate drive resistance for rise and fall transition by using the `-rise` and `-fall` options.

  By default, the drive resistance applies to both rise and fall transitions.

- A separate drive resistance for early and late analysis by using the `-min` and `-max` options.

  By default, the drive resistance applies to both early and late analysis.

For designs with multiple scenarios, by default, the drive resistance applies only to the current scenario. To specify a drive resistance for

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.

- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options.
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The following example sets the rise and fall drives of ports A, B, and C to 2.0

```
fc_shell> set_drive 2.0 {A B C}
```

Using the `set_driving_cell` command is the more accurate method for specifying the drive characteristics of an input or inout port. However, if you cannot specify a driving cell, specify the external drive characteristics by using the `set_drive` and `set_input_transition` commands.

Setting an Input Transition Time

To specify a fixed transition time for input or inout ports, use the `set_input_transition` command. The port has zero cell delay. The tool uses the specified transition time only in calculating the delays of logic driven by the port.

Specify the following information with the `set_input_transition` command:

- The ports you are constraining
- The input transition.

Optionally, you can specify

- A separate input transition for rise and fall transition by using the `-rise` and `-fall` options.

  By default, the input transition applies to both rise and fall transitions.
- A separate input transition for early and late analysis by using the `-min` and `-max` options.

  By default, the input transition applies to both early and late analysis.
- The input transition relative to a clock by using the `-clock` option.

  By default, the tool uses the rising edge of the clock. To use the falling edge of the clock, specify the `-clock_fall` option.

For designs with multiple scenarios, by default, the input transition applies only to the current scenario. To specify an input transition for

- All the scenarios of specific modes, use the `-modes` option.
- All the scenarios of specific corners and the current mode, use the `-corners` option.
- All the scenarios of specific modes and corners, use the `-modes` and `-corners` options
- Specific scenarios, use the `-scenarios` option.

  When you use this option, you cannot use the `-modes` or `-corners` options.

The following example sets the rise and fall input transition of ports A, B, and C to 0.5

```
fc_shell> set_input_transition 0.5 {A B C}
```

Using the `set_driving_cell` command is the more accurate method for specifying the drive characteristics of an input or inout port. However, if you cannot specify a driving cell, specify the external drive characteristics by using the `set_drive` and `set_input_transition` commands.

Removing Drive Characteristics From Ports

The commands shown in the following table remove drive information from ports.

Table 10 Commands to Remove Drive Information

To remove this	Use this
Driving cell information from a list of ports	`remove_driving_cell`
Drive resistance	`set_drive 0.0`
Input transition	`set_input_transition 0.0`
Drive data and all user-specified data, such as clocks, input and output delays	`reset_design`

Specifying Port Load Capacitance

To accurately perform timing analysis, the tool needs information about the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance. You can explicitly specify the load capacitance on a port with the `set_load` command.

When you use the `set_load` command to constrain a port, you must specify the following information:

- The ports you are constraining
- The capacitance value.

Optionally, you can specify

- A separate capacitance for rise and fall transition by using the `-rise` and `-fall` options.

  By default, the capacitance applies to both rise and fall transitions.
- A separate capacitance for early and late analysis by using the `-min` and `-max` options.

  By default, the capacitance applies to both early and late analysis.
- That the capacitance should be treated as a wire load by using the `-wire_load` option or both a wire load and a pin load by using the `-wire_load` and `-pin_load` options.

  By default, the tool treats the capacitance as a pin load.

The following example sets a capacitance of 3.5 on all inputs and 5 on all outputs.

```
fc_shell> set_load  3.5 [all_inputs]
fc_shell> set_load  5 [all_outputs]
```

Introduction to Ideal Networks

An ideal network is a network of cells, nets, and pins that are exempt from timing updates, timing optimization, and DRC fixing. For objects in an ideal network, the maximum capacitance and transition design rules are ignored. As a result, runtime and timing optimization are improved. In addition, the tool does not remove the source port or leaf-level pin of the ideal network during optimization.

For example, if you identify certain high-fanout nets that you intend to synthesize separately, such as scan-enable and reset nets, as ideal nets, you can reduce runtime by avoiding unnecessary retiming and unwanted design changes during optimization.

When you specify the source port or leaf-level pin of an ideal network, the nets, cells, and pins in the transitive fanout of this source are treated as ideal objects. Ideal objects have the following properties:

- They are marked as don't touch.
- They are not affected by timing updates, delay optimization, or DRC fixing.
- They are assigned ideal timing properties: ideal latency, ideal transition time, and ideal capacitance of zero. You can change the latency and transition values by using the `set_ideal_latency` and `set_ideal_transition` commands, respectively.

The `size_only` attribute is set on the cell that contains or drives the source. This guarantees that the ideal network source is not lost during a compile operation.

Propagation of the Ideal Network Property

When you specify the source object of an ideal network, all the nets, cells, and pins in the transitive fanout of the source objects are treated as ideal. Any input port or internal pin of the current design can be a source object, except for a pin at a hierarchical boundary.

The tool automatically spreads the ideal network property to the nets, cells, and pins in the transitive fanout of the source object, according to certain propagation rules.

The tool propagates the ideal network property and stops when it reaches

- A sequential cell.
- A point beyond which there is no forward timing arc, such as an output port or a disabled timing arc.
- An object that is not ideal.

The tool considers the following objects to be ideal

- A pin that it is one of the following:
  - A pin specified in the object list of the `set_ideal_network` command
  - A driver pin and its cell is ideal
  - A load pin attached to an ideal net
- A net that has all its driving pins as ideal.
- A combinational cell that has one of the following characteristics:
  - All its input pins are ideal

- ◦ An input pin is attached to a constant net and all other input pins are ideal.

  An object with the case analysis attribute is not treated as constant.

If an ideal network overlaps a clock network, the clock timing information, including clock latency and transition values, overrides the ideal timing for the clock portion of the overlapped networks.

For the circuit in the following figure, assume you specified pin U1/Z as the source of the ideal network. The tool propagates the ideal network property along the nets, cells, and pins in the transitive fanout of pin U1/Z and stops at the sequential cell named REG1. In addition, the tool propagates a `dont_touch` attribute is propagated to these nets, cells, and pins. and sets a `size_only` attribute on the driver of the ideal network source, cell IV1.

Figure 43 Ideal Networks

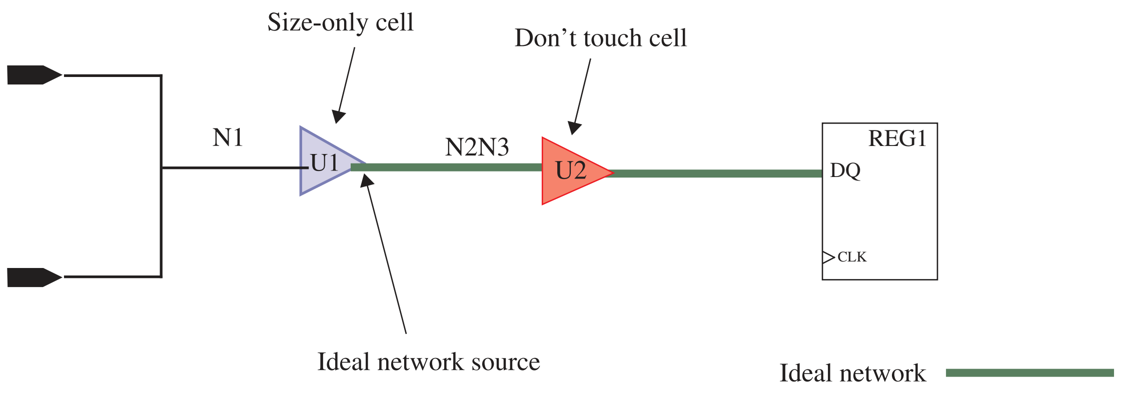
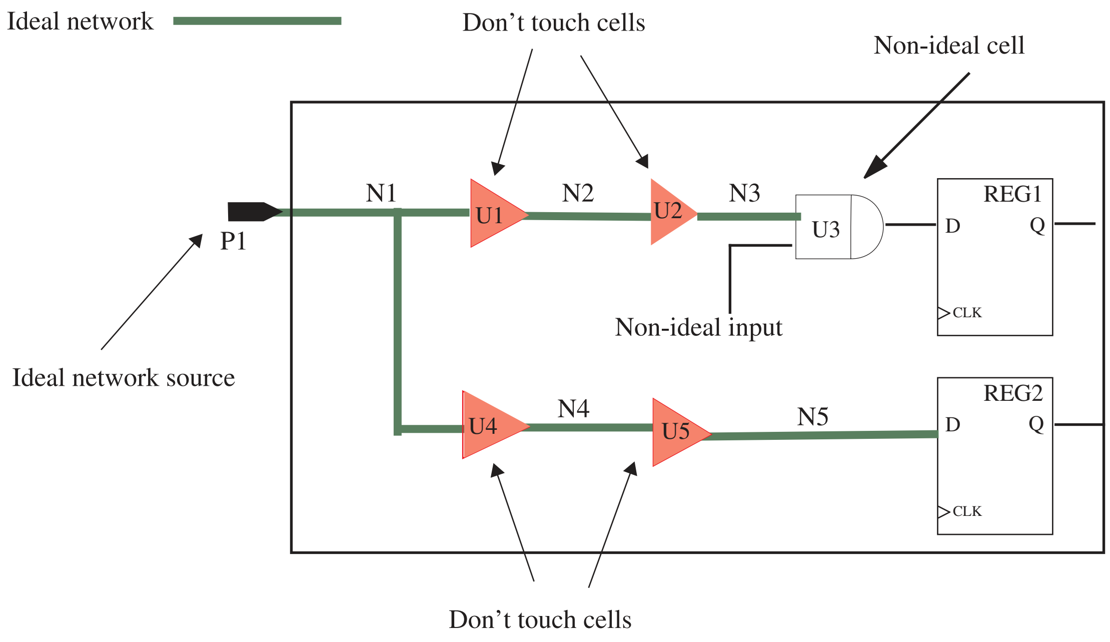


Figure 44 Propagation of the Ideal Network



Creating and Removing Ideal Networks

To create ideal networks, use the `set_ideal_network` command.

When you use this command, you must specify a list of ports, pins, or nets as the sources of the ideal network. If you specify a net, the net's global driver pins or ports are marked as ideal network sources. That is, the ideal network property is applied to the global driver pins or ports of the specified net. This ensures that the ideal network property is not lost even if the net is optimized away.

To prevent the ideal network from being propagated through logic gates, use the `-no_propagate` option.

The following command specifies port P1 as a source of an ideal network:

```
fc_shell> set_ideal_network [get_ports P1]
```

To report information about ports, pins, nets, and cells on ideal networks, use the `report_ideal_network` command. To remove an ideal network setting and restore cells, nets, and pins in the ideal network to their nonideal state, use the `remove_ideal_network` command.

Retrieving Ideal Objects

Use the following commands to retrieve ideal objects.

- `get_nets -filter "ideal_net == true"`

  This command returns ideal nets set by the `set_ideal_network -no_propagate` command.

- `get_pins -filter "ideal_network_source == true"`

  This command returns pins that are ideal network sources. Additionally, if you use the `ideal_network_options == 1` filter, the command returns only the ideal network source pins that were set by the `set_ideal_network -no_propagate` command.

- `get_ports -filter "ideal_network_source == true"`

  This command returns ports that are ideal network sources. Additionally, if you use the `ideal_network_options == 1` filter, the command returns only the ideal network source ports that were set by the `set_ideal_network -no_propagate` command.

Setting Ideal Latency and Ideal Transition Time

The default latency and transition values for ideal networks is zero. You can override these defaults by using the following commands:

- `set_ideal_latency`
- `set_ideal_transition`

Note:

The timing of ideal networks is updated whenever you execute either of these commands.

You can use these commands to set the ideal latency and ideal transition on the source pin of an ideal net or network and on any nonsource pin of an ideal network. The specified values override any library cell values or net delay values. For ideal networks, the ideal latency and transition values are propagated from the source pins to the network boundary pins.

The total ideal latency at any given point of an ideal network is the sum of the source pin ideal latency and all the ideal latencies of the leaf cell pins along the path to the given point. The ideal transition values specified at the various source and leaf cell pins are independent and noncumulative. The transition for an unspecified input pin is the ideal transition of the closest pin with a specified ideal transition value. This rule applies to boundary pins as well.

The `set_input_delay` command is applicable to ideal networks. This delay is treated as the off-block latency or source latency.

You can remove ideal latency and ideal transition values by using the following commands:

- `remove_ideal_latency`
- `remove_ideal_transition`

Ignoring Net Delays During Timing Analysis

By default, the Fusion Compiler tool considers both net and cell delays during timing analysis. To ignore net delays and consider only the cell delays during timing analysis, enable zero-interconnect delay calculation by setting the `time.delay_calculation_style` application option to `zero_interconnect`. To reset the delay calculation style to the default, set this application option to `auto`.

6

Performing Parasitic Extraction

Accurate timing analysis depends on accurate RC (parasitic) information for nets. The Fusion Compiler tool provides RC extraction capabilities at both preroute and postroute stages. At the preroute stage, the Fusion Compiler tool always performs native TLUPlus extraction. At the postroute stage, by default, the tool uses Fusion Signoff Extraction. The tool can also perform native TLUPlus extraction or StarRC In-Design extraction. In addition, the tool provides the capability of back-annotating detailed parasitic information.

The following topics describe the tasks related to extraction and back-annotation of parasitics information.

Performing TLUPlus Extraction

For unrouted, partially routed blocks, or global routed blocks, the Fusion Compiler tool always performs native TLUPlus extraction. For routed blocks, by default, the tool uses Fusion Signoff Extraction to perform StarRC extraction. You can also perform native TLUPlus extraction or StarRC In-Design extraction. To enable TLUPlus extraction, set the `extract.starrc_mode` application option to `none`.

The Fusion Compiler tool uses TLUPlus files for the parasitic technology information when performing native extraction. TLUPlus is a binary table that stores layer-specific RC coefficients.

The TLUPlus models enable accurate RC extraction results by including the effects of width, space, density, and temperature on the resistance coefficients. For details about modeling these effects in the Interconnect Technology Format (ITF) file, see the StarRC documentation.

For virtual route estimation, the Fusion Compiler tool calculates the RC coefficients by applying averaging techniques to the layer-specific RC coefficients because layers are not assigned at this stage of the design flow.

To associate parasitic information with a corner, you must

1. Read in the TLUPlus files by using the `read_parasitic_tech` command.

   For more information, see Reading TLUPlus Files.

2. Associate the TLUPlus files with a corner by using the `set_parasitic_parameters` command.

   For more information, see Associating TLUPlus Files With Corners.

The tool automatically performs extraction when it updates the timing information. You can update the timing information and perform extraction explicitly by running the `update_timing` command.

Reading TLUPlus Files

To associate parasitic information with a corner, you must read in the TLUPlus files by using the `read_parasitic_tech` command.

You must add the TLUPlus files to one of the following libraries:

- The cell library

  In general, the TLUPlus files should be in the cell libraries so they can be accessed by all blocks that use a specific technology.

- The design library

  If the TLUPlus files are not included in your cell libraries, add them to your design library.

You can read one or more TLUPlus files that are one of the following types:

- Emulation TLUPlus files, which are used before inserting metal fill
- Non-emulation TLUPlus files, which can be used before and after inserting metal fill

  When using non-emulated TLUPlus files, metal impact on parasitics is considered through real metal fill after metal fill insertion, or through virtual metal fill before metal fill insertion.

You must also specify the TLUPlus file by using the `-tlup` option. If you specify the TLUPlus files with a relative path or with no path, the Fusion Compiler tool uses the search path defined with the `search_path` variable to locate the files.

If the layer names in the TLUPlus file do not match the layer names in the technology file, you must define the mapping in a TLUPlus layer mapping file, which uses the syntax described in TLUPlus Layer Mapping File. To specify the name of the layer mapping file, use the `-layermap` option.

By default, the `read_parasitic_tech` command performs a basic check on the TLUPlus file and the corresponding layer mapping files. To disable this basic check or to perform a more advanced check, set the `-sanity_check` option to `none` or `advanced`.

After you read in a TLUPlus file, it is identified by its parasitic technology model name. By default, the parasitic technology model name is the base name of the specified TLUPlus file; however, you can specify a different name by using the `-name` option.

For example, to read in a TLUPlus file named my.tlup using a layer mapping file named my.layermap and store it in the current design library with a parasitic technology model name of para1, use the following command:

```
fc_shell> read_parasitic_tech -tlup my.tlup \
   -layermap my.layermap -name para1
```

Note:

The nxtgrd files generated with N-2017.12 or later version of the StarRC tool contain TLUPlus models, and the `read_parasitic_tech` command can read these files also.

TLUPlus Layer Mapping File

The TLUPlus layer mapping file uses the following syntax:

```
conducting_layers
tf_metal_layer_name1        ITF_metal_layer_name1
...
tf_metal_layer_namen        ITF_metal_layer_namen

via_layers
```

```
tf_via_layer_name1      ITF_via_layer_name1
...
tf_via_layer_namen      ITF_via_layer_namen
```

To include comments in the layer mapping file, start the line with an asterisk (*) or pound sign (#).

Associating TLUPlus Files With Corners

To associate parasitic information with a corner, you must associate the TLUPlus files with a corner by using the `set_parasitic_parameters` command.

- To associate the TLUPlus files with specific corners, user the `-corners` option.

  By default, the command associates the specified TLUPlus files with the current corner.
- To specify the TLUPlus file used for early delay calculations, use the `-early_spec` option.
- To specify the temperature used for early delay calculations, use the `-early_temperature` option.
- To specify the TLUPlus file used for late delay calculations, use the `-late_spec` option.
- To specify the temperature used for late delay calculations, use the `-late_temperature` option.
- To specify the library name that contains the parasitic specification, use the `-library` option.

  By default, the tool looks for the TLUPlus files in the current design library.

You can specify only one early and one late parasitic technology model per corner. However, you can change these settings at any time during the design flow, such as changing to non-emulation TLUPlus files after inserting metal fill.

To specify the TLUPlus files with the `-early_spec` and `-late_spec` options, use the parasitic technology model name assigned by the `read_parasitic_tech` command. For example, to use the TLUPlus file that has a parasitic technology model name of para1 for early delay calculations for the current corner, use the following command:

```
fc_shell> set_parasitic_parameters -early_spec para1
```

Performing Fusion Signoff Extraction

By default, the Fusion Compiler tool performs Fusion Signoff Extraction on routed designs using the StarRC extraction engine. This feature improves correlation with the StarRC tool, and it can be used with PrimeTime delay calculation to improve postroute correlation. This feature is enabled with the `extract.starrc_mode` application option set to `fusion_adv`.

After ECO routing, you can perform incremental Fusion extraction, which reduces extraction runtime by reusing RC data from the Galaxy Parasitic Database (GPD) for unaffected partitions. To enable incremental Fusion extraction, set the following application option before you route the design:

```
fc_shell> set_app_options -name extract.incremental_fusion -value 1
```

To set up a routed design for Fusion Signoff Extraction, perform the following steps before you run the `route_opt` command:

1. Specify the configuration by using the `set_starrc_in_design` command, as described in Specifying the Configuration for Fusion Signoff Extraction.
2. (Optional) Report the settings by using the `report_starrc_in_design` command.
3. (Optional) Define a multithreading configuration that applies to Fusion Signoff Extraction by using the `set_host_options -max_cores` command.

   For more information about the various settings you can specify for multicore processing, see the “Configuring Multithreading” topic in the *Fusion Compiler User Guide.*

The tool automatically performs extraction when it updates the timing information. You can update the timing information and perform extraction by explicitly running the `update_timing` command.

To use the parasitic information from the Fusion extraction, run the `update_timing -full` command before you run the `report_timing`, `report_qor`, or `write_parasitics` command.

Specifying the Configuration for Fusion Signoff Extraction

You must specify a configuration file for Fusion Signoff Extraction by using the `-config` option with the `set_starrc_in_design` command. The configuration file must contain the following StarRC settings:

- `CORNER_GRD_FILE`, which is a file that lists each corner and the corresponding StarRC nxtgrd file
- `MAPPING_FILE`, which is a StarRC layer mapping file

The following is an example configuration file for running Fusion Signoff Extraction in the implementation mode:

```
CORNER_GRD_FILE: corner_grd.mapping
MAPPING_FILE: star.mapping
```

Similar to the TLUPlus extraction, the Fusion Signoff Extraction honors extraction settings from the `set_extraction_options` command. To achieve good correlation with StarRC, you should check the Fusion Compiler and StarRC settings by running the Fusion Compiler and StarRC correlation checker. Modify the Fusion Compiler settings accordingly, using the `set_extraction_options` command.

Performing StarRC In-Design Extraction

StarRC In-Design extraction performs signoff extraction on routed designs using the StarRC capabilities. This feature improves correlation with the StarRC tool, and it can be used with PrimeTime delay calculation to improve postroute QoR. To enable this feature, set the `extract.starrc_mode` application option to `in_design`.

Note:

StarRC In-Design extraction requires a StarRC license.

To set up a routed design for StarRC In-Design extraction, perform the following steps before you run the `route_opt` command:

1. Specify the operational mode and configuration file by using the `set_starrc_in_design` command, as described in Specifying the Operational Mode and Configuration File for StarRC In-Design Extraction.
2. (Optional) Report the settings by using the `report_starrc_in_design` command.
3. (Optional) Define a distributed processing configuration that applies to StarRC In-Design by using the `set_host_options -target starrc` command.

   For more information about the various settings you can specify for distributed processing, see the “Configuring Distributed Processing” topic in the *Fusion Compiler User Guide.*
4. (Optional) Discover any potential setup issues by using the `check_starrc_in_design` command, as described in Discovering Setup Issues Before StarRC In-Design Extraction.

The tool automatically performs extraction when it updates the timing information. You can update the timing information and perform extraction by explicitly running the `update_timing` command.

To use the parasitic information from the StarRC extraction, run the `update_timing -full` command before you run the `report_timing`, `report_qor`, or `write_parasitics` command.

For more information about the StarRC tool, see the *StarRC User Guide and Command Reference*.

Specifying the Operational Mode and Configuration File for StarRC In-Design Extraction

You can specify the operational mode to run StarRC In-Design extraction by using one of the following two methods:

- To run in implementation mode, specify the `-mode icc2_centric` option with the `set_starrc_in_design` command.

  In this mode, which is the default, the Fusion Compiler tool runs StarRC In-Design extraction using the Fusion Compiler extraction settings.

- To run in signoff mode, specify the `-mode starrc_centric` option with the `set_starrc_in_design` command.

  In this mode, the Fusion Compiler tool runs StarRC In-Design extraction using the StarRC extraction settings.

You must specify a configuration file for StarRC extraction by using the `-config` option with the `set_starrc_in_design` command. The configuration file must contain the following StarRC settings:

- `SIGNOFF_IMAGE`, which is the location of the StarRC executable
- `CORNER_GRD_FILE`, which is a file that lists each corner and the corresponding StarRC nxtgrd file
- One of the following settings:
  - `MAPPING_FILE`, which is a StarRC layer mapping file

    Specify this setting when running StarRC In-Design extraction in the implementation mode.

  - `COMMAND_FILE`, which is a StarRC command file

    Specify this setting when running StarRC In-Design extraction in the signoff mode.

The following is an example configuration file for running StarRC In-Design extraction in the implementation mode:

```
SIGNOFF_IMAGE: /home/star_rcxt/bin/StarXtract
CORNER_GRD_FILE: corner_grd.mapping
MAPPING_FILE: star.mapping
```

The following is an example configuration file for running StarRC In-Design extraction in the signoff mode:

```
SIGNOFF_IMAGE: /home/star_rcxt/bin/StarXtract
CORNER_GRD_FILE: corner_grd.mapping
COMMAND_FILE: star.cmd
```

The following table shows the extraction settings used from the StarRC command file when running in the signoff mode, and the corresponding Fusion Compiler command and application option settings used when running in the implementation mode.

Table 11 Corresponding Extraction Settings Used in the Signoff and Implementation Modes

Setting used in signoff mode	Setting used in implementation mode
`METAL_FILL_POLYGON_HANDLING`	`set_extraction_options -real_metalfill_extraction`
`INSTANCE_PORT`	`set_extraction_options -include_pin_resistance`
`COUPLE_TO_GROUND`	`extract.enable_coupling_cap`
`REFERENCE_DIRECTION`	`set_extraction_options -reference_direction`
`COUPLING_THRESHOLD_OPERATION`	`set_extraction_options -enable_ccap_or_filtering`
`DPT`	`set_extraction_options -honor_mask_constraints`
`COUPLING_ABS_THRESHOLD`	`set_extraction_options -late_ccap_threshold`
`COUPLING_REL_THRESHOLD`	`set_extraction_options -late_ccap_ratio`

The following StarRC command file settings are ignored. Instead, they are derived from the Fusion Compiler design environment:

- `BLOCK`
- `NDM_DATABASE`
- `POWER_NETS`
- `SELECTED_CORNERS`
- `CORNERS_FILE`

The following StarRC settings are fixed and therefore are not overridden by the corresponding setting in the StarRC command file:

- `REDUCTION: NO_EXTRA_LOOPS`
- `EXTRA_GEOMETRY_INFO: NODE`
- `NETLIST_FORMAT: GPD`
- `SHORT_PINS: MIXED`
- `SKIP_CELLS: *`

- `POWER_EXTRACT: NO`
- `NETLIST_TYPE: RCC`
- `NETLIST_SELECT_NETS: *`
- `REDUCTION: NO_EXTRA_LOOPS`
- `NETLIST_NODE_SECTION: YES`
- `NETLIST_CONNECT_OPENS: *`
- `SIMULTANEOUS_MULTI_CORNER: YES`

The following settings in the StarRC command file are not supported:

- `DEF_ATTRIBUTE_FROM_LEF`
- `DEF_USE_PINS`
- `LEF_FILE`
- `LEF_USE_OBS`
- `MACRO_DEF_FILE`
- `TOP_DEF_FILE`

Discovering Setup Issues Before StarRC In-Design Extraction

You can use the `check_starrc_in_design` command to capture potential setup or StarRC errors before performing StarRC extraction.

Use the `-effort` option as follows:

- `-effort low` (default) checks paths to the following:
    - Configuration file
    - StarRC executable
    - Layer mapping file
    - Corner mapping file
- `-effort medium` checks the same paths and also writes the generated StarRC command file to the Check_StarRC_*moduleName*_*nnnnn* subdirectory. You can use

this file to help debug result inconsistencies between the Fusion Compiler and StarRC tools.

- `-effort high` checks the same paths, creates a StarRC command file, and then invokes the StarRC tool using the command file. All StarRC output files go in the Check_StarRC_*moduleName_nnnnn* subdirectory.

Specifying the Parasitic Scaling Factors for Extraction

Parasitic scaling factors allow you to adjust the resistance and capacitance values for specific conditions, such as minimum or maximum delay calculations and virtual-routed nets, detail-routed nets, and so on. By default, the parasitic scaling factors are 1.0 and the tool does not adjust the resistance and capacitance values.

To specify the parasitic scaling factors, use the `set_extraction_options` command. For example, to increase the capacitance values for maximum delay calculations by 10 percent and decrease the capacitance values for minimum delay calculations by 5 percent for the current corner, use the following command:

```
fc_shell> set_extraction_options \
   -late_cap_scale 1.1 -early_cap_scale 0.95
```

To reset a parasitic scaling factor, use the `set_extraction_options` command to set its value to 1.0. To report the parasitic scaling factors, use the `report_extraction_options` command.

The parasitic scaling factors that you specify with this command are saved with the block when you save it in the design library by using the `save_block` command.

Specifying Virtual Metal Fill for Extraction

Virtual metal fill (VMF) estimates the metal fill impact on parasitics and timing without actually inserting the metal fill. To enable virtual metal fill during extraction,

- Specify the virtual metal fill type.

  By default, extraction does not use virtual metal fill. To extract the virtual metal fill capacitance effect, you must specify which type of virtual metal fill to use: grounded or floating. To do this, use the `-virtual_metalfill_extraction` option with the `set_extraction_options` command.

  For example, to use floating virtual metal fill, use the following command:

  ```
  fc_shell> set_extraction_options \
     -virtual_metalfill_extraction floating
  ```

- Specify the virtual metal fill parameter file.

The virtual metal fill parameter file provides information about how metal fill is inserted.

To specify the virtual metal fill parameter file, use the `-virtual_metalfill_parameter_file` option with the `set_extraction_options` command.

For example, to use a parameter file named vmf.param, use the following command:

```
fc_shell> set_extraction_options \
   -virtual_metalfill_parameter_file vmf.param
```

For more information about the virtual metal fill parameter file formats that are supported, see Virtual Metal Fill Parameter File Formats.

Virtual Metal Fill Parameter File Formats

The virtual metal fill parameter file supports the following formats: basic VMF format, advanced VMF format (also known as StarRC VMF), the IC Validator .rh file, and the VMF option file.

Table 12 summarizes the support level of the VMF parameter file formats by the TLUPlus, Fusion Signoff, and StarRC In-Design extraction engines. Fusion Signoff extraction is the default post-route engine. Fusion Signoff extraction supports both basic VMF and advanced VMF file formats, while TLUPlus extraction only supports basic VMF and In-Design StarRC extraction only supports advanced VMF.

Table 12 Virtual metal fill parameter file formats

File format	**TLUPlus extraction**	**Fusion Signoff extraction**	**StarRC In-Design extraction**
Basic VMF	Yes	Yes (basic VMF only)	No
Advanced VMF (StarRC VMF)	Yes (only uses two parameters)	Yes (basic VMF and advanced VMF)	Yes
VMF option file	No	Yes (advanced VMF only)	Yes
IC Validator .rh file	No	Yes (advanced VMF only)	Yes

For more information about the VMF parameter file formats and extraction engine support, see the topics in this section.

- The Basic VMF File Format
- The Advanced VMF File Format

- The IC Validator .rh File
- The VMF Option File

The Basic VMF File Format

The basic VMF file format only requires the `fill_width` and `net_fill_space` parameters. The `fill_width` parameter specifies the width of the virtual metal fill shapes. The `net_fill_space` parameter specifies the spacing between route and virtual metal fill shapes. For example,

```
db_layer_nameA fill_widthA net_fill_spaceA
db_layer_nameA fill_widthA net_fill_spaceB
```

TLUPlus extraction, including RDE and Groute extraction, and basic-mode Fusion Signoff extraction (which is the default) can use the basic VMF file format. Basic VMF does not require a StarRC license.

To run basic VMF, use the `set_extraction_options` command to enable VMF and provide the VMF parameter file. For example,

```
fc_shell> set_extraction_options \
      -real_metalfill_extraction none  \
      -virtual_metalfill_extraction floating  \
      -virtual_metalfill_parameter_file vmf_parameter_file
```

The Advanced VMF File Format

The advanced VMF (also known as the StarRC VMF) file format is a detailed virtual metal fill parameter file which defines 15 parameters to more accurately model the metal file for each metal layer.

Fusion Signoff extraction and StarRC In-Design extraction can use advanced VMF. TLUPlus extraction can also use the advanced VMF format, however, only the `fill_width` and `net_fill_w_spacing` parameters are read and all other parameters are ignored.

Note:

Advanced VMF requires a StarRC license.

StarRC In-Design extraction and Fusion Signoff extraction convert the `set_extraction_options` VMF settings into the corresponding StarRC commands in the following advanced VMF file example.

```
VIRTUAL_METAL_FILL_PARAMETER_FILE: StarRC_vmf_parameter_file
METAL_FILL_POLYGON_HANDLING:      IGNORE|FLOATING|GROUNDED
VIRTUAL_METAL_FILL_POLYGON_HANDLING: IGNORE|FLOATING|GROUNDED
```

To run advanced VMF for Fusion Signoff extraction, you must enable it with the `extract.fusion_starrc_vfm` application option.

```
fc_shell> set_app_options -name extract.fusion_starrc_vmf -value advanced
```

Then, use the `set_extraction_options` command to enable VMF and provide the VMF parameter file. For example,

```
fc_shell> set_extraction_options \
      -real_metalfill_extraction none \
      -virtual_metalfill_extraction floating \
      -virtual_metalfill_parameter_file StarRC_vmf_parameter_file
```

The IC Validator .rh File

The IC Validator .rh file is generated after running the `signoff_create_metal_fill` command.

Fusion Signoff extraction and StarRC In-Design extraction can use the IC Validator .rh file.

The following is an example of the IC Validator .rh file.

```
m1_fill_width = ...;
m2_fill_width = ....;
...

m1_min_fill_length = ...;
m2_min_fill_length = ....;
...

m1_max_fill_length = ….;
m2_max_fill_length = ….;
…

m1_fill2fill_side_spacing = ….;
m2_fill2fill_side_spacing = ….;
…

m1_fill2fill_end_spacing = ….;
m2_fill2fill_end_spacing = ….;
…
```

Note:

If the VMF parameter file misses certain metal layers, a SX-3818 warning message appears stating that these layers will be disabled.

```
WARNING: Virtual Metall Fill flow for layer ‘xxx’ will be
 disabled as VMF
properties are not declared for such a layer. (SX-3813)
```

The VMF Option File

The VMF option file is also a supported parameter file format. Fusion Signoff extraction and In-Design StarRC extraction can use the VMF options file.

The following is an example of the VMF options file.

```
{
    "VMF_option_file_version": "1.0.0",
    "global": {
        "name": ""
    },
    "layer": {
        "name": "m9",
        "placement": {
            "name": "param_placement",
            "direction": "V",
            "fill_width": "0.05000",
            "min_fill_length": …
            …
         }
    },
…
}
```

Enabling Coupling Capacitance Extraction for Detailed Routed Nets

By default, when you perform extraction for a detail routed block, the tool does not extract the net-to-net coupling capacitances separately. The coupling effect is included in the total capacitance.

To extract coupling capacitances for detail routed nets, set the `extract.enable_coupling_cap` application option to `true`.

```
fc_shell> set_app_options -name extract.enable_coupling_cap \
   -value true
```

Disabling the Breaking of a Loop Structures in a Resistance Networks

During extraction, when the tool encounters a loop structure in a resistance network, it breaks the loop at the minimum resistance value. To improve the accuracy of pin-to-pin resistance and correlation with the StarRC tool, you can prevent the tool from breaking resistance loops by setting the `extract.cut_loop` application option to `false`. However, doing so can increase optimization runtime.

Writing Out the Extracted Parasitics

To write out the extracted parasitics information, use the `write_parasitics` command.

Note:

If you are using Fusion Signoff Extraction, the `write_parasitics` command requires a StarRC license.

When you use the `write_parasitics` command, you must specify an output file name by using the `-output` option.

By default, the `write_parasitics` command:

- Generates a SPEF file for each corner.

  The names of these SPEF files are derived based on the output file name you specify and the technology name, temperature, and the scaling factors specified by the `set_extraction_options` command. The tool also generates a file named XX.spef_scenario that lists the scenarios associated with each of the output SPEF files.

  To generate parasitic information:

  - For a specific corner, use the `-corner` option
  - In the Galaxy Parasitic Database (GPD) format, which is a binary file, use the `-format gpd` option

    GPD files can be used in the PrimeTime tool for timing analysis.

- Maps the net names to numbers and uses these numbers when writing the parasitic information for the nets.

  This reduces the file size. To override the mapping of net names, use the `-no_name_mapping` option.

- Generates a SPEF file for the top-level design.

  To generate separate SPEF files for each lower-level block, use the `-hierarchical` option.

When you run the `write_parasitics` command, if the parasitic information is out-of-date, the tool performs extraction. However, if you use the `-hierarchical` option, the tool does not check if extraction has been performed for the lower-level blocks. Therefore, run the `update_timing` command before you use the `write_parasitics` command with the `-hierarchical` option.

Back-Annotating Parasitics

You can read net parasitic data generated by an external tool in SPEF format, and use it during delay calculation during timing analysis. To do so, use the `read_parasitics` command and specify a SPEF file for each corner by using the `-corner_spef` option one or more times. If several corners share the same SPEF file, you can list multiple corners in one `-corner_spef` option setting.

By default, the parasitics in the SPEF file are applied to the top-level block. To read SPEF for lower-level blocks, use the `-block` option. When you use this option, the corners you specify with the `-corner_spef` option must be defined at the top level.

The following example applies:

- The TOP_CIC2.spef file for corners C1 and C2 and TOP_C3C4.spef file for corners C3 and C4 at the top level.
- The BLK_CIC2.spef file for corners C1 and C2 and BLK_C3C4.spef file for corners C3 and C4 at the block level.

```
fc_shell> read_parasitics -corner_spef {{C1 C2} TOP_C1C2.spef} \
     -corner_spef {{C3 C4} TOP_C3C4.spef}
fc_shell> read_parasitics -block BLK  \
   -corner_spef {{C1 C2} BLK_C1C2.spef} \
   -corner_spef {{C3 C4} BLK_C3C4.spef}
```

After the `read_parasitics` command is executed, the tool prints the number of nets that are not annotated. The tool performs extraction for any nets that are not annotated by the SPEF files.

In addition to SPEF format, you can also use the `read_parasitics` command to read data in Galaxy Parasitics Database (GPD) format by using one of the following options:

- `-gpd` *`gpd_path`* specifies a path to the GPD file for RC annotation.
- `-gpd_attach` reads parasitics from the GPD attached in the current block.

By reading parasitics through the GPD or GPD attachment, you can avoid performing extraction (through the `update_timing` command) when opening a block.

Note:

You must make sure that the input GPD has the same corners as the current block.

Feedback

7

Performing Analysis

The Fusion Compiler tool allows you to analyze and report a wide range of timing related information about the design. The following topics describe how to control and perform timing analysis:

- Controlling Delay Calculation
- Ignoring Net Delay During Timing Analysis
- Enabling Multiple-Input Switching Analysis
- Enabling Waveform Propagation Using CCS Models
- Identifying Infeasible Paths and Resolving the Infeasibility
- Reporting Timing Paths
- Performing Path-Based Analysis
- Reporting the Logical DRC Violations
- Generating a Global Summary of Timing Violations
- Reporting the QoR
- Generating and Analyzing a QoR Snapshot
- Reporting Cells With Large Delays
- Reporting the Delay of a Timing Arc
- Reporting the Clock or Data Arrival Time at Pins or Ports
- Checking the Timing Constraints and Settings
- Checking for Consistency in Timing Analysis or Extraction Settings
- Analyzing Timing Correlation Between the Fusion Compiler and PrimeTime Tools

Controlling Delay Calculation

The Fusion Compiler tool supports both the Elmore and Arnoldi delay calculation models. You cannot specify which of these timing models to use; the tool selects whichever model is the most appropriate, based on the state of the design.

If you are using cell libraries that are generated by Library Manager version O-2018.06 or later, the Fusion Compiler tool automatically uses the PrimeTime delay calculation engine when you perform postroute optimization. If your cell libraries are generated by a Library Manage version before O-2018.06, during postroute optimization, the Fusion Compiler tool issues a TIM-260 warning message recommending you to regenerate your cell libraries.

To ensure that the Fusion Compiler timing analysis settings are consistent with the corresponding PrimeTime settings, use the `check_consistency_settings` command as described in Checking for Consistency in Timing Analysis or Extraction Settings.

For more information about performing postroute optimization with the `route_opt` command, see the *Fusion Compiler Implementation User Guide*.

To generate cell libraries, use either the library configuration flow in the Fusion Compiler tool, as described in *"Creating Cell Libraries"* in the *Fusion Compiler Data Model User Guide*, or the library assembly flow in the Library Manager tool, as described in the *Library Manager User Guide*.

Ignoring Net Delay During Timing Analysis

You can ignore net delays during timing analysis by using the `time.delay_calculation_style` application option as follows:

```
fc_shell> set_app_options -name time.delay_calculation_style \
  -value zero_interconnect
```

Enabling Multiple-Input Switching Analysis

Library cells are usually characterized such that a change at a single input causes the output to change. However, in an actual design, if multiple inputs switch at the same time, the actual cell delay can be very different from the characterized value. Multiple-input switching mostly affects hold analysis.

You can account for multiple-input switching effects during hold analysis by

1. Specifying multiple-input switching coefficients for library cells by using the `set_multi_input_switching_coefficient` command.
2. Enabling multiple-input switching analysis by setting the `time.enable_multi_input_switching_analysis` application option to `true`.

When you enable this feature, by default, the tool considers the data arrival windows of the input pins, checks for overlaps, and derives the actual coefficients based on the coefficients specified for the library cells. You can ignore the data arrival windows of input pins and use coefficients specified for the library cells by setting the `time.enable_multi_input_switching_timing_window_filter` application option to `false`.

Enabling Waveform Propagation Using CCS Models

Waveform distortion effects can become significant at advanced geometry nodes. The Fusion Compiler tool offers an advanced gate-level waveform propagation analysis mode that captures the effects of such distortion on the next stage delay, including the Miller and long tail effects.

The waveform propagation mode uses CCS timing models and propagated piecewise linear waveforms in place of simplified equivalent waveforms. It produces more accurate results when the voltage waveform shapes differ significantly from the library characterization driver waveform shapes. The CCS modeling data must be available in the logic libraries to get more accurate results.

To enable waveform propagation, use the following steps:

1. Generate your cell libraries with version M-2016.12-SP2 or later of the library manager.
2. Use CCS receiver models for delay calculation.

```
fc_shell> set_app_options -name time.enable_ccs_rcv_cap \
    -value true
```

3. Use CCS-based waveform analysis on the design.

```
fc_shell> set_app_options \
    -name time.delay_calc_waveform_analysis_mode \
    -value full_design
```

Identifying Infeasible Paths and Resolving the Infeasibility

Infeasible paths are those paths that cannot meet timing constraints because the data arrival time is after the required time. They can occur because of invalid constraints such as

- Missing false path or multicycle path constraints
- Unreasonable input or output delay constraints even when cell and net delays are set to zero

Infeasible paths increase runtime because the tool continues to optimize these paths. Therefore, identifying and resolving the infeasibility can prevent this possible runtime increase.

- To identify the infeasible paths in the design, use the `compute_infeasible_path_overrides` command.
- To identify the infeasible paths and generate a script containing timing exceptions that resolve the infeasibility, use the `compute_infeasible_path_overrides -output` command.
- To identify the infeasible paths and automatically apply timing exceptions that resolve the infeasibility, use the `compute_infeasible_path_overrides -apply_override_constraints` command.

The following example checks infeasible paths and generates the out.tcl file that contains timing exceptions to resolve the infeasibility:

```
fc_shell> compute_infeasible_path_overrides -verbose -output out.tcl
*************************************************************************
Report : infeasible path
Design : top
...
*************************************************************************

start-                                       startpoint endpoint infeasi-
point    endpoint              scenario        arrival  required   bility
-------------------------------------------------------------------------
D1 inst_flip_flop1/Q2ff_reg/D  SETUP_SS_WC_test  2000.00  100.00  1900.00
D2 inst_flip_flop2/Q1ff_reg/D  SETUP_SS_WC_test  2000.00  100.00  1900.00
```

As shown in the following example, the out.tcl file contains path margins that are generated by the command:

```
#check_infeasible_timing_paths
#################################

set_path_margin -10000.00 -setup \
   -comment SYNOPSYS_INFEASIBLE_PATH_OVERRIDE \
   -scenarios SETUP_SS_WC_test -from D2 -to inst_flip_flop3/Q2ff_reg/D
set_path_margin -10000.00 -setup \
   -comment SYNOPSYS_INFEASIBLE_PATH_OVERRIDE \
   -scenarios SETUP_SS_WC_test -from D2 -to inst_flip_flop3/Q1ff_reg/D
```

This command generates path margins that are marked with a `SYNOPSYS_INFEASIBLE_PATH_OVERRIDE` comment. If you want to remove these generated path margins after applying them, use the following command:

```
fc_shell> remove_path_margin \
   -comment SYNOPSYS_INFEASIBLE_PATH_OVERRIDE
1
```

Reporting Timing Paths

To report information about the timing paths in a block, use the `report_timing` command.

By default, the command reports the path having the worst maximum (setup) delay across all scenarios for the current block that are enabled for setup analysis. To report the timing information for minimum (hold) delay analysis, use the `-delay_type` option.

To report the timing paths for

- All scenarios associated with specific modes, use the `-modes` option.
- All scenarios associated with specific corners, use the `-corners` option.
- Specific scenarios, use the `-scenarios` option.

To report maximum and minimum timing information for a scenario, you must enable the scenario for setup and hold analysis.

You can further limit the report to

- Include only specific path groups by using the `-groups` option
- Include only paths
  - Starting from specific points by using the `-from`, `-rise_from`, or `-fall_from` option
  - Ending at specific points by using the `-to`, `-rise_to`, or `-fall_to` option
  - Going through specific points by using the `-through`, `-rise_through`, or `-fall_through` option
  - Between a specific startpoint and endpoint pair by using the `-start_end_pair` option
- Include only a specific number of paths or paths per endpoint by using the `-max_paths` or `-nworst` option
- Include only specific paths that have a slack less than a specified value by using the `-slack_lesser_than` option.
- Exclude any timing paths that are

  - ◦ From, through, or to specific pins, ports, nets, or cell instances by using the `-exclude` option
  - ◦ Rising at specific pins, ports, nets, or cell instances by using the `-rise_exclude` option
  - ◦ Falling at specific pins, ports, nets, or cell instances by using the `-fall_exclude` option

Control the information that is printed about the timing paths or its objects by using the following options:

- Include nets by using the `-nets` option
- Include pin locations by using the `-physical` option
- Include attributes by using the `-attributes` option
- Include input pins by using the `-input_pins` option
- Include hierarchical pins by using the `-include_hierarchical_pins` option
- Include transition times by using the `-transition_time` option
- Include capacitances by using the `-capacitance` option
- Include information about the dominant timing exceptions (false paths, multicycle paths, and minimum and maximum delays) that apply to each timing path that is printed by using the `-exception` option

  Including information about the exceptions that apply to each timing path increases the runtime.
- Include operating condition process, voltage, and temperature by using the `-process`, `-voltage`, and `-temperature` options
- Include timing derates by using the `-derate` option
- Include crosstalk information when signal integrity is enabled, by using the `-crosstalk_delta` option
- Include the mean and standard deviation of the statistical time increment for each path element when parametric on-chip variation (POCV) analysis is enabled by using the `-variation` option
- Exclude the transparency window for latch based designs by using the `-skip_transparency_window` option

You can control the formatting of the report as follows:

- Specify the path reporting style, such as the full path, endpoints only, and so on, by using the `-path_type` option
- Order the report based on modes, corners, scenarios, path groups, or the entire block by using the `-report_by` option
- Sort the report based on paths, path groups, or slack by using the `-sort_by` option
- Specify the number of significant digits to display by using the `-significant_digits` option
- Prevent the splitting of lines, if columns over flow, by using the `-nosplit` option

The following is an example of the default output of the `report_timing` command.

```
fc_shell> report_timing
****************************************
Report : timing
        -path_type full
        -delay_type max
        -nworst 1
        -max_paths 1
        -report_by design
        -nets
        -physical
        -attributes
Design : my_design
.......
.......
.......
****************************************
...
 Startpoint: u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_/CLK (rising
 edge-triggered flip-flop)
  Endpoint: u0_3/p0/iu0/r_reg_E__OP2__31_/D (rising edge-triggered
 flip-flop)
  Mode: my_mode
  Corner: wc_corner
  Scenario: wc_scenario
  Path Group: pci_clk
  Path Type: max
  Launch clock: pci_clk  r
  Capture clock: pci_clk  r

Attributes
    b - black-box (unknown)
    s - size_only
    d - dont_touch
    u - dont_use
    g - generic
    h - hierarchical
    i - ideal
    n - noncombinational
    E - extracted timing model
    Q - Quick timing model

  Point                                Fanout  Cap  Trans  Incr   Path
------------------------------------------------------------------------
---
  clock pci_clk (rise edge)                                 0.00   0.00
  clock network delay (ideal)                               2.00   2.00
  u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_/CLK
   (SDFFX1_LVT)                                      0.00   0.00   2.00 r
  u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_/QN
   (SDFFX1_LVT)
                                          1    7.19  0.11   0.18   2.18 f
...
  u0_3/p0/iu0/U6660/S (MUX21X1_LVT)                  0.22   0.00   6.26 r
  u0_3/p0/iu0/U6660/Q (MUX21X1_LVT)       1    1.78  0.11   0.11   6.37 f
  u0_3/p0/iu0/r_reg_E__OP2__31_/D
   (SDFFX1_LVT)                                      0.11   0.00   6.37 f
  data arrival time                                                6.37

  clock pci_clk (rise edge)                                 5.00   5.00
  clock network delay (ideal)                               2.00   7.00
  clock uncertainty                                        -0.40   6.60
  library setup time                                       -0.20   6.40
  data required time                                               6.40
------------------------------------------------------------------------
---
  data required time                                               6.40
  data arrival time                                               -6.37
------------------------------------------------------------------------
---
  slack (MET)                                                      0.03
```

Controlling the Column Order in a Timing Report

The default report shows the startpoint, endpoint, path group, path type, the incremental and cumulative time delay values along the data and clock paths, the data required time at the path endpoint, and the timing slack for the path.

You can control the column order in the report by using the `time.report_timing_column_order` application option, as shown in the following example:

```
fc_shell> set_app_options -name time.report_timing_column_order \
   -value {point incr path cap fanout trans}
```

The valid values for this application option are `point`, `incr`, `path`, `cap`, `fanout`, `trans`, `derate`, `mean`, and `sensit`.

Reporting Unconstrained Timing Paths

By default, the `report_timing` and `get_timing_paths` commands do not report timing paths that are unconstrained. To report unconstrained timing paths, use the following application option setting before you run the `report_timing` or `get_timing_paths` command:

```
fc_shell> set_app_options -list {time.report_unconstrained_paths true}
```

Unconstrained paths have a slack value of infinity, and you can report unconstrained paths only by using path filtering as shown in the following example:

```
fc_shell> report_timing [filter_collection \
   [get_timing_paths -max_paths 200000 -to [all_outputs]] \
     slack=="INFINITY"]
```

Performing Path-Based Analysis

For faster runtime and lower memory usage, by default, the tool uses graph-based analysis (GBA), which merges the timing data at nodes with multiple paths going through them. Graph-based analysis is conservative and provides a safety margin for timing optimization.

Path-based analysis (PBA) can address the pessimism of graph-based analysis by analyzing and propagating the timing for each topological path. It is recommended for more accurate timing analysis at the postroute stage.

You can perform path-based analysis by using the `-pba_mode` option with the following commands:

- `report_timing`
- `report_global_timing`
- `report_qor`
- `get_timing_paths`

The supported values for the `-pba_mode` option are:

- `none`

  Graph-based analysis is performed, which is the default.

- `path`

  Graph-based analysis is used to identify the appropriate timing paths, and then path-based analysis used to report the timing of the selected paths.

- `exhaustive`

  Path-based analysis is used exhaustively to identify and report the appropriate timing paths, which might increase the runtime significantly. You can limit the number of paths analyzed by using the `time.pba_exhaustive_endpoint_path_limit` application option.

To specify that path-based analysis is used only for adjusting the timing derating values, set the `time.pba_derate_only_mode` application option to `true`.

Reporting the Logical DRC Violations

To report the logical DRC violations, use the `report_constraints` command. This command displays a summary of the constraint violations in the block, including the amount by which a constraint is violated, information about the design object that is the worst violator, and the weighted cost of the violation.

By default, the `report_constraints` command checks the following timing constraints:

- Minimum path delay, which includes violations of hold time on registers or ports with output delay as well as violations of the `set_min_delay` constraint setting.
- Maximum path delay, which includes violations of setup time on registers or ports with output delay as well as violations of the `set_max_delay` constraint setting.
- Maximum transition time
- Maximum capacitance
- Minimum capacitance

To restrict the report to a specific constraint, use one of the following options:

- `-min_delay` (minimum path delay)
- `-max_delay` (maximum path delay)
- `-max_transition` (maximum transition time)
- `-max_capacitance` (maximum capacitance)
- `-min_capacitance` (minimum capacitance)

By default, the report generated by the `report_constraints` command displays brief information about the worst violation for each constraint in the current block and the overall cost. To report all violations, rather than just the worst violations, use the `-all_violators` option.

The following is an example of the default output of the `report_constraints` command.

Example 1 Default Constraint Report

```
fc_shell> report_constraints
****************************************
Report : constraint
Design : placed
.......
.......
.......
****************************************
                                                     Weighted
    Group (min_delay/hold)       Cost     Weight       Cost     Scenario
    --------------------------------------------------------------------
    ...
    pci_clk                      0.00      1.00        0.00   wc_scenario
    FEEDTHROUGH                  0.00      1.00        0.00   wc_scenario
    REGIN                        0.00      1.00        0.00   wc_scenario
    REGOUT                       0.00      1.00        0.00   wc_scenario
    ...
    pci_clk                      1.12      1.00        1.12   bc_scenario
    FEEDTHROUGH                  0.00      1.00        0.00   bc_scenario
    REGIN                        0.01      1.00        0.01   bc_scenario
    REGOUT                       0.00      1.00        0.00   bc_scenario
    --------------------------------------------------------------------
    min_delay/hold                                     1.13

                                                     Weighted
    Group (max_delay/setup)      Cost     Weight       Cost     Scenario
    --------------------------------------------------------------------
    ...
    pci_clk                      0.44      1.00        0.44   wc_scenario
    FEEDTHROUGH                  0.00      1.00        0.00   wc_scenario
    REGIN                        0.00      1.00        0.00   wc_scenario
    REGOUT                       0.24      1.00        0.24   wc_scenario
    ...
    pci_clk                      0.00      1.00        0.00   bc_scenario
    FEEDTHROUGH                  0.00      1.00        0.00   bc_scenario
    REGIN                        0.00      1.00        0.00   bc_scenario
    REGOUT                       0.00      1.00        0.00   bc_scenario
    --------------------------------------------------------------------
    max_delay/setup                                    0.68

    Constraint                                         Cost
    -----------------------------------------------------
    min_delay/hold                                     1.13   (VIOLATED)
    max_delay/setup                                    0.68   (VIOLATED)
    max_transition                                   417.69   (VIOLATED)
    max_capacitance                               189895.48    (VIOLATED)
    min_capacitance                                  100.52   (VIOLATED)
1
```

Generating a Global Summary of Timing Violations

To generate a global summary of all timing violations in a block, use the `report_global_timing` command. The report shows the worst negative slack, the total negative slacks, and the number of violating endpoints.

The following example shows a default report:

```
fc_shell> report_global_timing
****************************************
Report : global_timing
...
...
****************************************

Setup violations

---------------------------------------------------------
        Total   reg->reg   in->reg  reg->out   in->out
---------------------------------------------------------
WNS      -20.34    -20.34      0.00     -1.19     -0.98
TNS    -6887.98  -6882.84      0.00     -3.17     -1.96
NUM         398       392         0         4         2
---------------------------------------------------------

Hold violations

---------------------------------------------------------
         Total  reg->reg   in->reg  reg->out   in->out
---------------------------------------------------------
WNS       -3.55     -3.55      0.00      0.00      0.00
TNS     -561.17   -561.17      0.00      0.00      0.00
NUM         418       418         0         0         0
---------------------------------------------------------
```

By default, the tool analyzes both setup and hold violations of all path groups in all active corners, modes, and scenarios. You can limit the analysis by using the `-delay_type`, `-groups`, `-corners`, `-modes`, and `-scenarios` options. For more information about these options and other options available for changing the granularity of the report, see the man page of the `report_global_timing` command.

Reporting the QoR

To generate a summary of the quality of results and other statistics of a block, use the `report_qor` command. It reports information about timing path group details and cell count and current block statistics, including combinational, noncombinational, and total area. The command also reports static power and design rule violations.

By default, the command reports the QoR for all active scenarios. To report the QoR for

- All scenarios associated with specific modes, use the `-modes` option.
- All scenarios associated with specific corners, use the `-corners` option.
- Specific scenarios, use the `-scenarios` option.

The following is an example of a report generated by the `report_qor` command.

Example 2 report_qor Report Example

```
fc_shell> report_qor
****************************************
Report : qor
Design : placed
.....
.....
****************************************

Information: Design Average RC value per unit length: (NEX-011)
Information: r = 1.792938 ohm/um, c = 0.079907 ff/um, cc = 0.000000
ff/um (X dir) (NEX-017)
Information: r = 1.785714 ohm/um, c = 0.093680 ff/um, cc = 0.000000
ff/um (Y dir) (NEX-017)

Scenario           'bc_scenario'
Timing Path Group  'pci_clk'
----------------------------------------
Worst Hold Violation:             -0.15
Total Hold Violation:             -8.79
No. of Hold Violations:             283
----------------------------------------
...

Scenario           'wc_scenario'
Timing Path Group  'pci_clk'
----------------------------------------
Levels of Logic:                     55
Critical Path Length:              2.79
Critical Path Slack:               0.03
Critical Path Clk Period:          3.00
Total Negative Slack:              0.00
No. of Violating Paths:               0
----------------------------------------
...

Cell Count
----------------------------------------
Hierarchical Cell Count:            694
Leaf Cell Count:                 116919
Buf/Inv Cell Count:               17031
CT Buf/Inv Cell Count:                0
```

```
----------------------------------------

Area
----------------------------------------
Combinational Area:           8365846.92
Noncombinational Area:         571218.74
Net Area:                              0
Net XLength:                           0
Net YLength:                           0
----------------------------------------
Cell Area:                    8937065.66
Design Area:                  8937065.66
Net Length:                            0

Design Rules
----------------------------------------
Total Number of Nets:             142890
Nets with Violations:                273
Max Trans Violations:                 12
Max Cap Violations:                  262
----------------------------------------
```

In the Cell Count section, the report shows the number of macros in the block. To qualify as a macro cell, a cell must not be hierarchical and must have the `is_macro_cell` attribute set on its library cell.

Generating and Analyzing a QoR Snapshot

You can generate detailed information about the quality of results (QoR) for the design by using the `create_qor_snapshot` command. This command measures and reports the quality of the design in terms of timing, design rules, area, and power, and stores the quality information into a set of snapshot files.

Subsequently, you can

- Retrieve and report the QoR information of the snapshot with the `report_qor_snapshot` command
- Selectively retrieve, sort, and display the needed information from the snapshot with the `query_qor_snapshot` command
- Remove the snapshot with the `remove_qor_snapshot` command

When you create a QoR snapshot with the `create_qor_snapshot` command, you must specify a name for the snapshot by using the `-name` option. You use this name when you report, query, or remove the snapshot.

By default, the tool creates the QoR snapshot for all scenarios of the block. To create it for a specific scenario, use the `-scenarios` option, as shown in the following example:

```
fc_shell> create_qor_snapshot -name func1 -scenarios func1

**********************************************************************
Report          : create_qor_snapshot (func1)
.......
.......
.......
Time unit       : 1.00ns
Resistance unit : 1.00MOhm
Capacitance unit: 1.00fF
Voltage unit    : 1.00V
Current unit    : 1.00uA
Power unit      : 1.00pW
Location        : /home/my_work_dir/placement_stage/snapshot
**********************************************************************
No. of scenario = 1
s1 = func1
--------------------------------------------------
WNS of each timing group:                       s1
--------------------------------------------------
SD_DDR_CLK                                    1.07
SDRAM_CLK                                     0.06
PCI_CLK                                      -1.76
SYS_2x_CLK                                    0.08
v_PCI_CLK                                    -0.71
SYS_CLK                                       0.06
--------------------------------------------------
Setup WNS:                                   -1.76
Setup TNS:                                 -102.47
Number of setup violations:                     95
Hold WNS:                                    -0.50
Hold TNS:                                   -27.81
Number of hold violations:                     125
Number of max trans violations:                  0
Number of max cap violations:                   28
Number of min pulse width violations:            0
--------------------------------------------------
Area:                                   444776.034
Cell count:                                  53399
Buf/inv cell count:                           7182
Std cell utilization:                         0.50
CPU(s):                                         64
Mem(Mb):                                       649
Host name:                                syn_host
--------------------------------------------------
```

```
Histogram:              s1
--------------------------------------------------
Max violations:         95
   above ~ -0.7   ---   40
    -0.6 ~ -0.7   ---   55
    -0.5 ~ -0.6   ---    0
    -0.4 ~ -0.5   ---    0
    -0.3 ~ -0.4   ---    0
    -0.2 ~ -0.3   ---    0
    -0.1 ~ -0.2   ---    0
       0 ~ -0.1   ---    0
--------------------------------------------------
Min violations:        125
  -0.06 ~ above   ---    0
  -0.05 ~ -0.06   ---    0
  -0.04 ~ -0.05   ---    0
  -0.03 ~ -0.04   ---    3
  -0.02 ~ -0.03   ---    4
  -0.01 ~ -0.02   ---   20
      0 ~ -0.01   ---   41
--------------------------------------------------
Snapshot (func1) is created and stored under
 "/home/my_work_dir/placement_stage/snapshot" directory
```

By default, the snapshot is stored in a directory named snapshot in the current working directory. To specify a different location, use the `time.snapshot_storage_location` application option.

When you create, report, or query a QoR snapshot with the `create_qor_snapshot`, `report_qor_snapshot`, or `query_qor_snapshot` command, you can display the results in the GUI by using the `-display` option.

The following example analyzes a snapshot and displays the paths that have a worst negative slack between –2.0 and –1.0:

```
fc_shell> query_qor_snapshot -name func1 -display \
   -filters "-wns -2.0,-1.0"
```

The following example analyzes a snapshot and displays the paths that have more than 10 levels of logic:

```
fc_shell> query_qor_snapshot -name func1 -display \
   -filters "-logic_level 10, "
```

Reporting Cells With Large Delays

To report the leaf cells in block that have large delays, you can use the `check_cell_delay` command. By default, this command reports all the cells with a cell delay larger than 30 percent of the path slack up to that cell. You can change this percentage delay threshold by using the `-delay_threshold` option.

Reporting the Delay of a Timing Arc

To get a detailed report about the delay calculation of a given cell or net timing arc along a timing path, use the `report_delay_calculation` command.

Use the `-from` and `-to` options to specify the “from” and “to” pins of the cell or net. These two pins can be the input and output pins of a cell to report the calculation of a cell delay, or can be the driver pin and a load pin of a net to report the calculation of a net delay.

By default, the command reports the delay calculation for the current scenario. You can specify the mode and corner to report with the `-mode` and `-corner` option, respectively, or you can specify the scenario to report by using the `-scenario` option.

The following is an example report generated by the `report_delay_calculation` command for the current scenario.

Example 3 Delay Calculation Report

```
fc_shell> report_delay_calculation \
   -from u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_/CLK \
   -to u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_/QN
****************************************
Report : delay_calculation
Module : placed
Mode   : my_mode
Corner : wc_mode
.......
.......
.......
****************************************

cap units: 1.00fF  res units: 1.00MOhm  time units: 1.00ns

Current delay calculation style: auto
Current min Elmore tau: 0.00s
Current min Arnoldi tau: 2.00ps

Cell arc:
Lib: /usr/libs/mylib.ndm:mylib_lvt
Cell: u0_3/p0/mul0/m0_U1_INT_SUMR_reg_1_ (FD1_LVT)
From-pin: CLK  To-pin: QN

Sense:   rising_edge

            Input                  Output  Stored   Stored         Delay
            slew    Load   Delay   slew    delay    output slew    derate
early rise: 0.29    7.27   0.22    0.11    0.22     0.11           0.99
```

```
 late rise: 0.29    7.27   0.22    0.11    0.22     0.11          1.01
early fall: 0.29    6.85   0.20    0.10    0.20     0.10          0.99
 late fall: 0.29    6.85   0.20    0.10    0.20     0.10          1.01

1
```

Reporting the Clock or Data Arrival Time at Pins or Ports

To obtain the arrival time of a data signal at a pin or port on a data net, query the `arrival_window` attribute by using the `get_attribute` command as shown in the following example:

```
fc_shell> get_attribute [get_pins U50/Y] arrival_window
{{{SYS_CLK} pos_edge {min_r_f 0.367955 0.361167} {max_r_f 0.385574
 0.383949}}}
```

For the arrival window, the tool reports the clock name, the edge type that launches the path, and the minimum and maximum rise and fall arrival times at the pin. If data launched from multiple clocks arrive at a pin, the tool reports a separate arrival window for the data launched by each clock.

To obtain the arrival time of a clock signal at a pin or port on the clock network, query the `clock_arrival_window` attribute by using the `get_attribute` command as shown in the following example:

```
fc_shell> get_attribute [get_pins CT_BUF_7/Y] clock_arrival_window
{{{SYS_CLK} pos_edge {min_r_f 1.03046 --} {max_r_f 1.03067 --}}}
```

For the clock arrival window, the tool reports the clock name, the active edge type, and the minimum and maximum rise or fall arrival times at the clock pin, based on the active edge of the clock. If multiple clocks or multiple active edges of the same clock arrive at the pin, the tool reports a separate arrival window for each clock or clock edge.

Checking the Timing Constraints and Settings

Paths that are incorrectly constrained might not appear in the violation reports, possibly causing you to overlook paths with violations. You can check for incorrectly specified or missing constraints or other timing issues by using the `check_timing` command.

Table 13 shows a list of checks that performed by the `check_timing` command.

- To perform all the checks, use the `-all` option.
- To include specific checks, in addition to the default checks, use the `-include` option.
- To exclude specific checks, from the default checks, use the `-exclude` option.

Table 13 Types of Checks Performed by the check_timing Command

Type of Check	Default	What the tool does
`gated_clock`	Yes	Issues a warning about any gated clocks that do not reach register clock pins.
`generated_clock`	Yes	Checks the generated clock network and issues a warning for any of the following conditions: Generated clocks with no path to the master clock Generated clocks with no path to the master clock that meets the sense relationship specified Multiple generated clocks that form a loop Generated clocks that are not expanded Generated clocks that have no period specified
`loops`	Yes	Issues a warning if a combinational feedback loops is found.
`no_clock`	Yes	Issues a warning if no clocks reach a register clock pin.
`no_input_delay`	Yes	Issues a warning if no clock-related delay is specified for an input port.
`unconstrained_ endpoints`	Yes	Issues a warning if register data pin or primary output is unconstrained.
`clock_crossing`	No	Issues a warning if a timing path crosses a clock domain.
`data_check`	No	Issues a warning if no clocked signal or multiple clocked signals reach the reference pin of a data check register.
`multiple_clock`	No	Issues an information message if multiple clocks reach a register pin.

By default, this command performs the timing checks for all active scenarios. To perform the timing checks for

- All scenarios associated with specific modes, use the `-modes` option.
- All scenarios associated with specific corners, use the `-corners` option.
- Specific scenarios, use the `-scenarios` option.

The following is an example output of the `check_timing` command.

```
fc_shell> check_timing
Warning: The register clock pin 'F1/CLK' has no fanin clocks. (TCK-002)
Warning: The register clock pin 'F2/CLK' has no fanin clocks. (TCK-002)
Warning: The register clock pin 'F3/CLK' has no fanin clocks. (TCK-002)
Warning: The reported endpoint 'OUT' is unconstrained. Reason: 'no
 check'. (TCK-001)
```

```
Warning: The reported endpoint 'F1/DATA' is unconstrained. Reason:
 'unclocked'. (TCK-001)
Warning: The reported endpoint 'F2/DATA' is unconstrained. Reason:
 'unclocked'. (TCK-001)
Warning: The reported endpoint 'F3/DATA' is unconstrained. Reason:
 'unclocked'. (TCK-001)
```

You can also perform the timing checks by using the `check_design -checks timing` command, as shown in the following example:

```
fc_shell> check_design -checks timing
****************************************
Report : check_design
Options: { timing
g }
Design : TOP
.......
.......
****************************************
Running atomic-check 'timing'
*** EMS Message summary ***
----------------------------------------------------------------------
Rule    Type   Count Message
 ----------------------------------------------------------------------
TCK-002  Warn   3      The register clock pin '%pin' has no fanin clocks.
-----------------------------------------------------------------------
Total 7 EMS messages : 0 errors, 3 warnings, 0 info.
-----------------------------------------------------------------------
*** Non-EMS message summary ***
-----------------------------------------------------------------------
Total 0 non-EMS messages : 0 errors, 0 warnings, 0 info.
-----------------------------------------------------------------------
Info: EMS database is saved to file 'check_design.ems'
```

In addition to generating a report, the `check_design` command generates an enhanced messaging system (EMS) database that you can view by using the message browser in the Fusion Compiler GUI.

Checking for Consistency in Timing Analysis or Extraction Settings

To achieve faster timing convergence, it is important to ensure that the settings used for timing analysis in the Fusion Compiler and PrimeTime tools are consistent. Similarly, the settings used for extraction is the Fusion Compiler and StarRC tools should also be consistent. To check for the consistency of these settings, use the `check_consistency_settings` command.

This command can help you identify the following:

- Inconsistencies in the settings for Fusion Compiler and PrimeTime timing analysis.
- Fusion Compiler and PrimeTime timing analysis settings that are different from the recommended settings for the respective tools.
- Inconsistencies in the settings for Fusion Compiler and StarRC extraction.
- Your Fusion Compiler and StarRC extraction settings that are different from the recommended settings for the respective tools

Before you run the `check_consistency_settings` command, you must specify options for consistency checking by using the `set_consistency_settings_options` command.

The following example sets up and checks the consistency between the Fusion Compiler and PrimeTime settings.

```
fc_shell> set_consistency_settings_options \
   -exec_path /global/SNPS_tools -tool pt \
   -script PT_top.tcl
fc_shell> check_consistency_settings  -tool pt \
   -output pt_consistency_check.rpt
```

The following is part of the output file generated for this example:

```
Error: The timing_crpr_threshold_ps PrimeTime setting should be set to 5.
 (CORR-800)
Error: The time.crpr_remove_clock_to_data_crp Fusion Compiler setting
 should be set to FALSE. (CORR-812)
Error: The time.enable_si(si_enable_analysis) setting is inconsistent
 between Fusion Compiler and PrimeTime. The Fusion Compiler value is
 true; the PrimeTime value is false. (CORR-803)
```

The following example sets up and checks the consistency between the Fusion Compiler and StarRC settings.

```
fc_shell> set_consistency_settings_options \
   -exec_path /global/SNPS_tools -tool starrc \
   -script run_starrc.tc -corner max
fc_shell> check_consistency_settings  -tool starrc \
   -output starrc_consistency_check.rpt
```

Analyzing Timing Correlation Between the Fusion Compiler and PrimeTime Tools

You can analyze timing correlation between the Fusion Compiler and PrimeTime tools by using the `analyze_timing_correlation` command.

By default, this command performs the following steps:

1. Based on the current Fusion Compiler settings, generates the files necessary for running the PrimeTime tool for each scenario of the current block.
2. Invokes the PrimeTime tool and generate the timing data corresponding to each scenario.
3. Generates the Fusion Compiler timing data for each scenario.
4. Generates the timing correlation report between the two tools.

With the `analyze_timing_correlation` command, you must specify

- A working directory for correlation analysis by using the `-work_dir` option.

  The working directory contains the intermediate files such as design netlist, constraints, parasitics, and Tcl scripts generated for running the PrimeTime tool, and also the timing data generated for correlation analysis.

- The path to the PrimeTime executable by using the `-pt_exec_path` option.

```
fc_shell> analyze_timing_correlation -work_dir run_correlation \
   -pt_exec_path /tools/SNPS/linux64/bin/pt_shell
```

The following is an example of a correlation report generated by this command:

```
Correlation Metric: slack
--------------------------------------------- --------------------------
Correlation Metric                            Value
--------------------------------------------- --------------------------
All Endpoints                                 99.952% at 3.00% Threshold
Violating Endpoints                           99.306% at 3.00% Threshold
Absolute Slack Difference on Average          2.664 ps
Absolute Slack Difference Standard Deviation 4.253 ps

---------- ------ ------ --- ----------
Context    WNS    TNS    NVE Unique NVE
---------- ------ ------ --- ----------
PT         -0.058 -1.476 132         39
FC         -0.039 -0.997 105         12
Difference -0.019 -0.480  27         27
```

This report shows

- The percentage of timing endpoints that correlate within a specified threshold, which defaults to 3 percent. You can change this threshold by using the `-pass_rate_threshold` option.
- The percentage of violating timing endpoints that correlate within the specified threshold.
- The average and standard deviation of the absolute difference between Fusion Compiler and PrimeTime slack for all endpoints.
- The worst negative slack (WNS), total negative slack (TNS), number of violating endpoints (NVE), and the unique number of violating endpoints for the both tools and the difference.