



# Timing Analyzer Quick-Start Tutorial

---

## Intel® Quartus® Prime Pro Edition

Updated for Intel® Quartus® Prime Design Suite: **17.1**



**Online Version**



**Send Feedback**

**UG-TMQSTANZR**

ID: **683588**

Version: **2017.12.01**

## Contents

---

<b>Timing Analyzer Quick-Start Tutorial (Intel® Quartus® Prime Pro Edition).....</b>	<b>3</b>
Step 1: Open the Project and Compile.....	3
Step 2: Specify Clock Constraints.....	4
Step 3: View Clock Timing Analysis.....	6
Step 4: Declare False Paths.....	7
Step 5: View Post-Fit Timing Results.....	7
Step 6: Specify Input and Output Delay Constraints.....	9
Step 7: Report Clocks and Top Failing Paths.....	11
Document Revision History.....	12

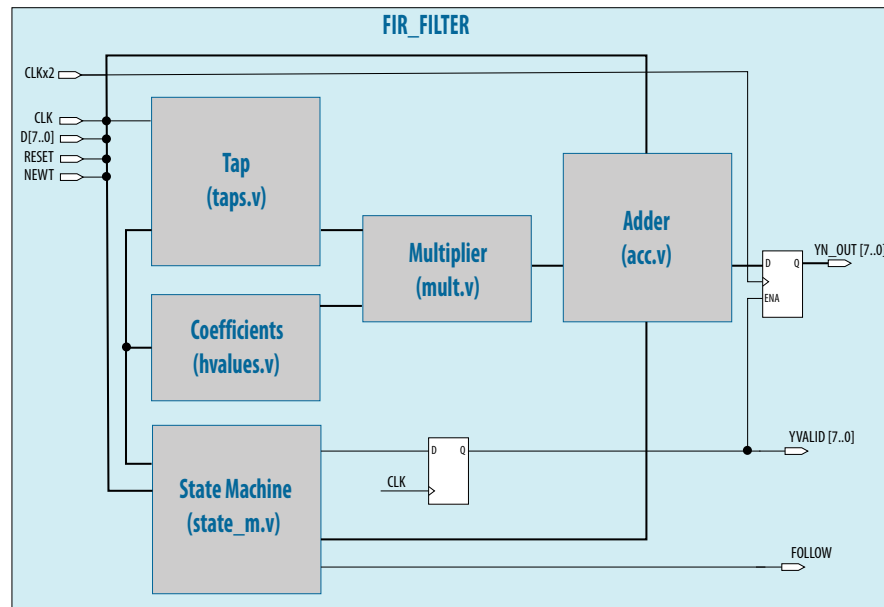
## Timing Analyzer Quick-Start Tutorial (Intel® Quartus® Prime Pro Edition)

This tutorial demonstrates how to specify timing constraints and perform static timing analysis with the Intel® Quartus® Prime Timing Analyzer. The Timing Analyzer validates the timing performance of all logic in your design using industry-standard constraint, analysis, and reporting methodology. The Intel Quartus Prime software generates timing analysis data by default during design compilation.

Running timing analysis involves running the Compiler, specifying timing constraints, and viewing timing analysis reports. The following steps describe this process in detail.

**Note:** This Quick-Start requires a basic understanding of timing analysis concepts and the Intel Quartus Prime design flow, as the [Intel Quartus Prime Pro Edition Foundation Online Training](#) describes.

**Figure 1. fir\_filter Design Schematic**

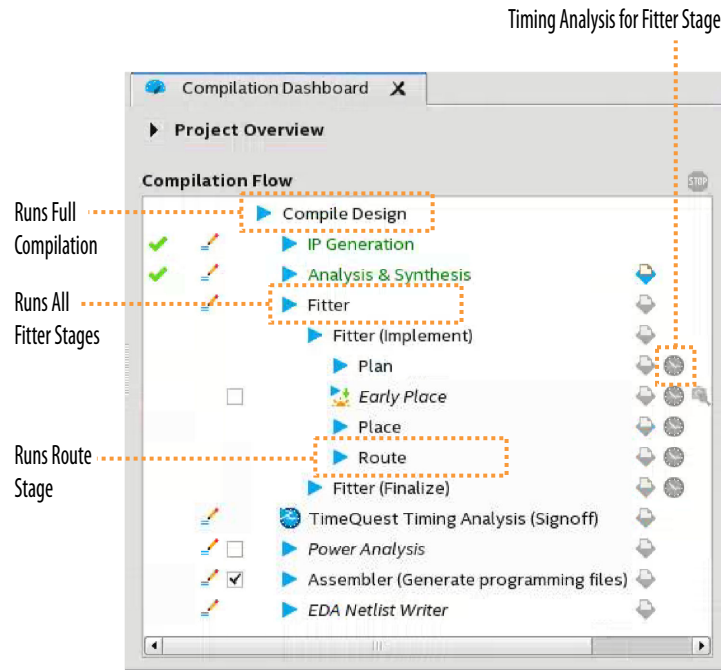


### Step 1: Open the Project and Compile

This tutorial uses a simple `fir_filter` design to demonstrate the Intel Quartus Prime Timing Analyzer.

The Intel Quartus Prime software installation includes the sample `fir_filter` project in the `quartus/qdesigns/fir_filter/` directory. The following steps describe opening the example project and running initial compilation to elaborate the design hierarchy, synthesize logic, and generate a node netlist for application of constraints.

1. Launch the Intel Quartus Prime Pro Edition software.
2. To open the example design project, click **File ► Open Project**, and open the `quartus/qdesigns/fir_filter/fir_filter.qpf` project file.
3. To view the top-level design schematic, click **Open** on the **Tasks** pane, select the `filtref.bdf` file, and click **Open**. The `filtref.bdf` schematic appears in the Block Editor.
4. Perform one of the following from the Compilation Dashboard. (To display the Dashboard if closed, click **Processing ► Compilation Dashboard**).
  - To run the Fitter, click **Fitter** (or any Fitter stage) on the Compilation Dashboard.
  - To run a full compilation, click **Compile Design** on the Compilation Dashboard.

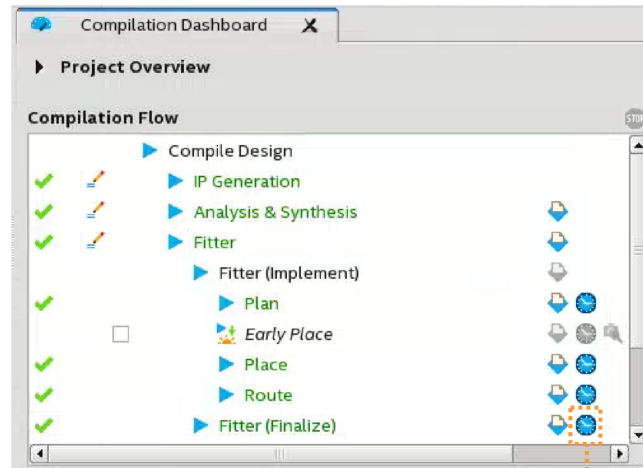


## Step 2: Specify Clock Constraints

The Intel Quartus Prime Timing Analyzer supports the industry standard Synopsys Design Constraints (`.sdc`) format for specifying timing constraints.

The `fir_filter` design example already includes a default `filtref.sdc` file. You can modify these constraints in the Timing Analyzer GUI, or in the `.sdc` file directly. Use the following steps to modify the clock constraints in the `.sdc` file:

1. To launch the Timing Analyzer, click the **Timing Analyzer** icon for the Fitter stage you run in the Compilation Dashboard. The **Create Timing Netlist** dialog box opens automatically when the Timing Analyzer launches.



Launches Timing Analyzer

2. In the **Create Timing Netlist** dialog box, select the compilation **Snapshot**, and the **Delay model** of the timing netlist, and then click **OK**.
3. In the Timing Analyzer, click **File > Open**, and then select the `filtref.sdc` file in the project directory. The file opens in the Intel Quartus Prime Text Editor.
4. In the `.sdc` file, locate the following two `create_clock` constraints:

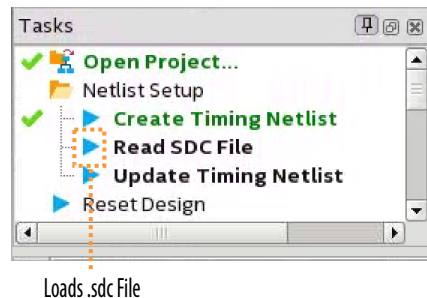
```
*****
# Create Clock
*****
create_clock -name {clk} -period 4.000 -waveform { 0.000 2.000 } /
[get_ports {clk}]
create_clock -name {clkx2} -period 4.000 -waveform { 0.000 2.000 } /
[get_ports {clkx2}]
```

5. Replace the existing clock constraints with the following clock constraints. The replacement defines a 50 MHz (50/50 duty cycle) clock for `clk`, and a 100 MHz (60/40 duty cycle) clock for `clkx2`.

```
*****
# Create Clock
*****
create_clock -name clk -period 20 [get_ports {clk}]
create_clock -name clkx2 -period 10.000 -waveform {0 6} /
[get_ports {clkx2}]
```

**Note:** For help entering `.sdc` constraints, right-click anywhere in the text file, point to **Insert Constraint**, and then select the constraint to insert at the cursor location.

- To save the .sdc file changes, click **File ► Save**.
- To load the updated .sdc constraints, double-click **Read SDC File** in the **Tasks** pane.



- To update the timing netlist, double-click **Update Timing Netlist**.

### Step 3: View Clock Timing Analysis

Follow these steps to confirm the clock constraints and view timing analysis results.

- In the **Tasks** pane, double-click **Report SDC** in the **Diagnostic** reports. The Create Clock report shows your new clock constraints.

Create Clock						
	SDC Command	Name	Period	Waveform	Targets	Location
1	create_clock	clk	20.000	{ 0.000 10.000 }	[get_ports {clk}]	filtref.sdc:41
2	create_clock	clkx2	10.000	{ 0.000 6.000 }	[get_ports {clkx2}]	filtref.sdc:42

- To generate clock timing performance summary report, double-click **Report Clocks** on the **Tasks** pane.

Clocks Summary						
	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Base	20.000	50.0 MHz	0.000	10.000
2	clkx2	Base	10.000	100.0 MHz	0.000	6.000

- To verify the validity of all clock-to-clock transfers, double-click **Report Clock Transfers** on the **Tasks** pane.

Setup Transfers						
	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths
1	clk	clk	13823	0	0	0
2	clk	clkx2	16	0	0	0

The Setup Transfers report indicates that a clock-to-clock transfer exists between `clk` and `clkx2`. The **RR Paths** column lists the number of instances where `clk` clocks the source node, and `clkx2` clocks the destination node. These clock transfers actually do not require analysis because they are false paths in the design. The next step demonstrates declaring these false paths.

## Step 4: Declare False Paths

Follow these steps to declare false paths for the clock transfers between the `clk` and `clkx2` clock signals:

1. In the Timing Analyzer, click **File > Open**, and open `filtref.sdc`.
2. In the `.sdc` file, locate the following `get_false_path` constraint:

```
*****
# Set False Path
*****
set_false_path -from [get_clocks {clk clkx2}] -through /
               [get_pins -compatibility_mode *] -to [get_clocks {clk clkx2}]
```

3. To declare false paths for all clock transfers between the `clk` and `clkx2` clock signals, replace the existing false path constraint with the following:

```
*****
# Set False Path
*****
set_false_path -from [get_clocks clk] /
               -to [get_clocks clkx2]
```

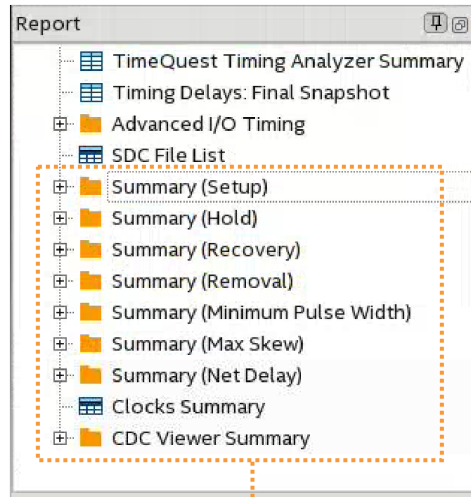
4. Save and close the `.sdc` file.
5. To load the new `.sdc` constraints, double-click **Read SDC File** in the **Tasks** pane.
6. To update the timing netlist, double-click **Update Timing Netlist**.
7. To confirm the false path assignment in the reports, double-click **Setup Transfers** in the **Diagnostic** reports. The **RR Paths** column indicates that the clock domain is now a "false path," reflecting your constraint.

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths
1	clk	clk	13823	0	0	0
2	clk	clkx2	false path	0	0	0

## Step 5: View Post-Fit Timing Results

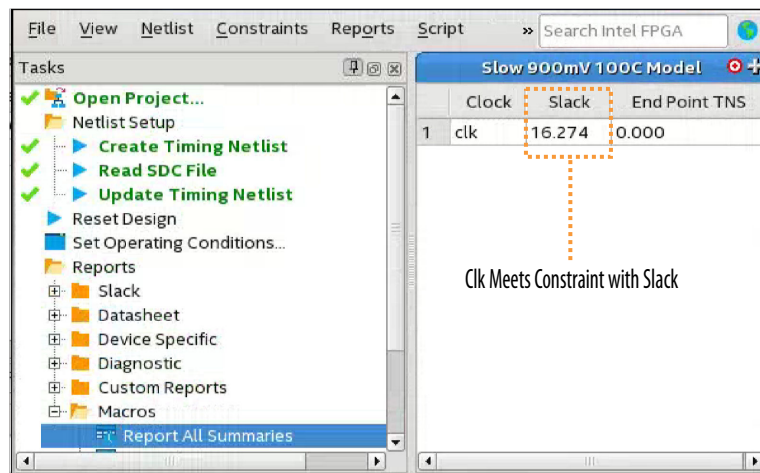
After specifying clock and false path constraints, follow these steps to run full compilation and view post-fit timing analysis results. The Compiler attempts to meet your timing constraints when implementing your design, and reports paths that do not achieve the requirement.

1. In the Compilation Dashboard, click **Compile Design**. The Compiler runs all stages in full compilation.
2. Click the **Timing Analyzer** icon for **Fitter (Finalize)** in the Compilation Dashboard. The **Create Timing Netlist** dialog box appears automatically when the Timing Analyzer opens. Retain the default netlist settings, and then click **OK**.
3. In the **Tasks** pane, click **Update Timing Netlist**. You can now generate timing analysis for the final snapshot.
4. In the **Tasks** pane, double-click **Report All Summaries** in the **Macros** reports. The Timing Analyzer generates all summary reports in the **Report** pane.



Report All Summary Reports

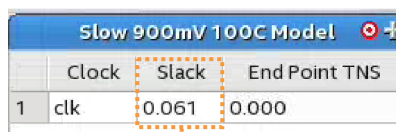
- To verify that there are no violations of the clock constraints, double-click a report in the **Summary (Setup)** folder. The clock setup check ensures that each register-to-register transfer does not violate your .sdc clock constraints. The **Slack** column indicates that `clk` meets the requirement with some margin. The **End Point TNS** column reports the total negative slack (TNS) for the clock domain. Use this value to determine the number of failing paths in the clock domain.



*Note:* The `clkx2` clock does not appear in the **Summary (Setup)** reports because of the false path constraints between `clk` and `clkx2`. The `fir_filter` design also contains no register-to-register paths with a destination register path clocked by `clkx2`.

- Double-click the report under the **Summary (Setup)** folder. The **Summary (Hold)** report indicates that the `clk` clock node meets the timing constraints by some margin.

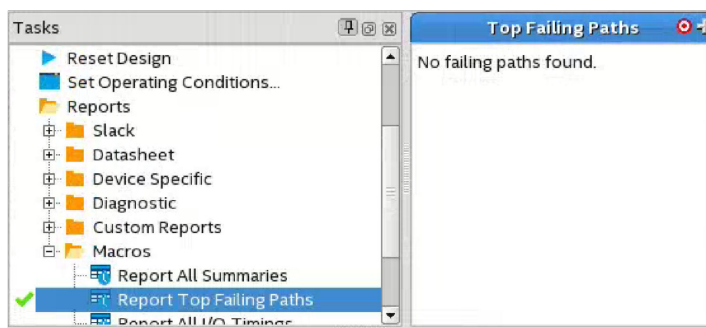




	Clock	Slack	End Point TNS
1	clk	0.061	0.000

Clk Meets Constraint with Slack

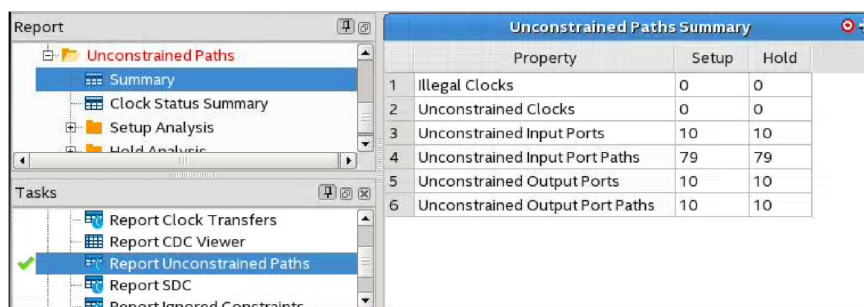
- Double-click **Report Top Failing Paths** in the **Macros** reports. The report indicates "No failing paths found."



## Step 6: Specify Input and Output Delay Constraints

Accurate timing analysis requires constraining all input and output ports. Follow these steps to identify unconstrained paths and apply input and output delay constraints to the ports.

- To identify unconstrained path in the design, double-click **Report Unconstrained Paths** the **Diagnostic** reports. The report lists the details of all unconstrained paths.



	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	10	10
4	Unconstrained Input Port Paths	79	79
5	Unconstrained Output Ports	10	10
6	Unconstrained Output Port Paths	10	10

- Repeat the steps in [Step 2: Specify Clock Constraints](#) on page 4 to open the .sdc file for edit.
- In the .sdc file, locate the following section:

```
*****
# Set Input Delay
*****
```

- To insert the set\_input\_delay constraint, right-click under the # Set Input Delay comment, and then click **Insert Constraint > Set Input Delay**.

5. In the **Set Input Delay** dialog box, specify the following options for the constraint:

Option	Setting
Clock name	clk
Use falling clock edge	Off
Delay value	2
Targets	<ul style="list-style-type: none"> <li>Collection—get_ports</li> <li>Filter—d* newt reset</li> <li>SDC Command</li> </ul> <pre>[get_ports {d[0] d[1] d[2] d[3] d[4] / d[5] d[6] d[7] newt reset}]</pre>

6. Click **Insert**. The following constraint appears at the insertion point:

```
set_input_delay -clock { clk } 2 [get_ports /
{d[0] d[1] d[2] d[3] d[4] d[5] d[6] d[7] newt reset}]
```

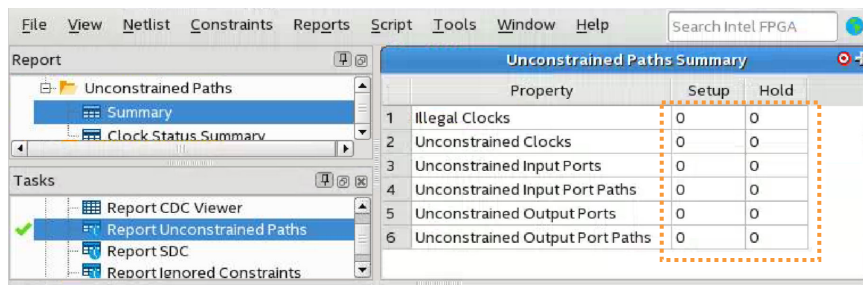
7. Repeat steps 4 through 6 to add the following set\_output\_delay constraint in the # Set Output Delay section of the .sdc file:

Option	Setting
Clock name	clk
Use falling clock edge	Off
Delay value	1.5
Targets	<ul style="list-style-type: none"> <li>Collection—get_ports</li> <li>Filter—d* newt reset</li> <li>SDC Command</li> </ul> <pre>[get_ports {follow yn_out[0] yn_out[1] yn_out[2] / yn_out[3] yn_out[4] yn_out[5] yn_out[6] / yn_out[7] yvalid}]</pre>

8. Click **Insert**. The following constraint appears at the insertion point. All input and output ports now have constraints:

```
set_output_delay -clock { clk } 1.5 [get_ports /
{follow yn_out[0] yn_out[1] yn_out[2] yn_out[3] yn_out[4] /
yn_out[5] yn_out[6] yn_out[7] yvalid}]
```

9. Save and close the .sdc file.
10. Double-click **Read SDC File**, and then double-click **Update Timing Netlist** in the **Tasks** pane. You can now verify there are no remaining unconstrained paths.
11. Double-click **Report Unconstrained Paths** in the **Diagnostic** reports. The report shows zero unconstrained paths.



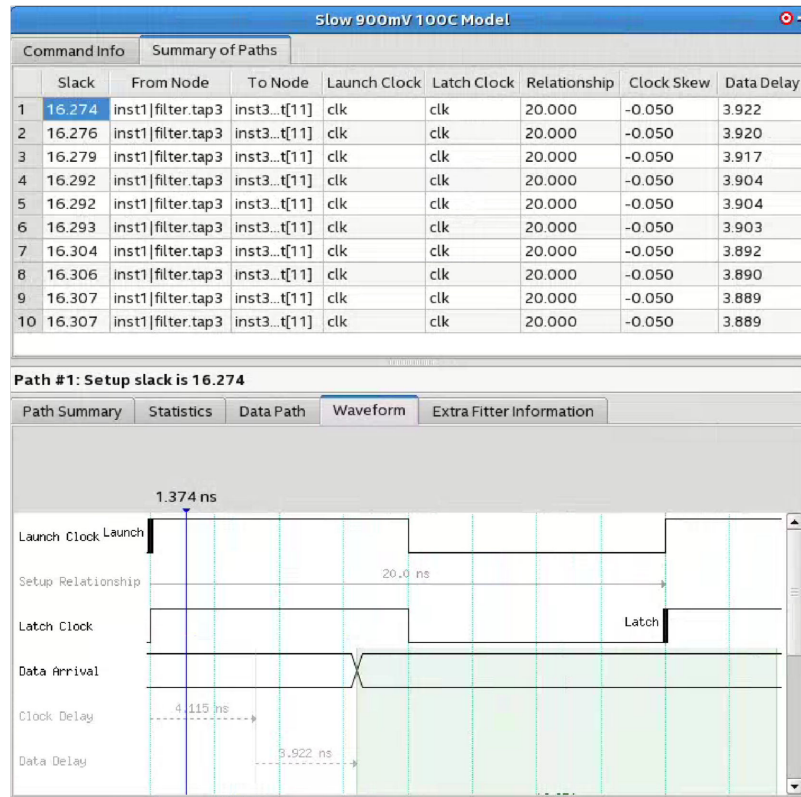
## Step 7: Report Clocks and Top Failing Paths

After constraining all paths, follow these steps to report top failing timing paths:

1. Under **Custom Reports**, double-click **Report Timing**, and then specify the following options. Retain the default settings for the other options.

Option	Setting
To clock	clk
Targets To:	acc:inst3 result*
Analysis type	Setup
Report number of paths	10
Tcl command	report_timing -setup -npaths 10 -detail full_path -panel_name {Report Timing} -multi_corner

2. Click **Report Timing**. The Slow 900mV 100C Model report shows the 10 worst paths between the clk and acc:inst3|result nodes.
3. To report the top failing paths in the design, double-click **Report Top Failing Paths** in the **Macros** reports.



Continue optimizing the design and running timing analysis until the design meets all requirements.

## Document Revision History

Date	Version	Changes Made
2017.12.01	17.1.0	<ul style="list-style-type: none"> <li>Updated for Intel Quartus Prime Pro Edition and Intel Cyclone® 10 devices.</li> <li>Updated steps to use .sdc file rather than GUI for constraint entry.</li> <li>Consolidated related steps.</li> <li>Updated all screenshots.</li> <li>Updated for latest Intel branding and style conventions.</li> </ul>
2017.12.05	1.2	Converted to new Template, updated for most recent devices.
December 2009	1.1	Updated figures in Chapter 2. Updated chapter for Intel Quartus Prime software 9.1 functionality.
May 2006	1.0	Initial Release