Vivado Design Suite User Guide

Design Analysis and Closure Techniques

UG906 (v2012.2) AUGUST 20, 2012

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/20/12	2012.2	Initial Xilinx release.

Table of Contents

Chapter 1

Design Analysis

The *Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)* provides an introduction to:

- Understanding the design in the Vivado™ Integrated Design Environment (IDE), including:
    - Messages and reports
    - Design Rules Check (DRC) structure
- Understanding the design in the chip
- Analyzing the timing
- Analyzing Place and Route results
- Techniques to use when the design does not meet its timing goals

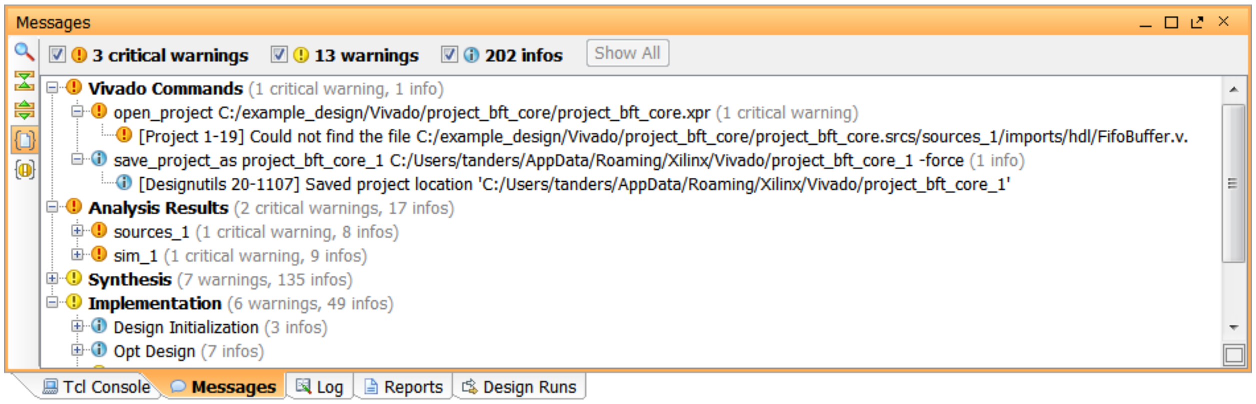
For more information about managing windows and using the Vivado IDE, see the *Vivado Design Suite User Guide: Using Vivado IDE (UG893)*.

Reports and Messages

The Vivado IDE generates reports and messages to inform you of events during various tool interactions. Most user-initiated actions generate messages. Review the messages to determine whether the tools are having difficulties in any sections of the design.

Messages and reports are stored in the Messages and Reports windows in the Results window area.

In the Messages window, the Vivado IDE groups messages in subfolders by the action that created the message. Use the icons on the local toolbar to order the messages by message ID or file.



```
Vivado Commands (1 critical warning, 1 info)
  open_project C:/example_design/Vivado/project_bft_core/project_bft_core.xpr (1 critical warning)
    [Project 1-19] Could not find the file C:/example_design/Vivado/project_bft_core/project_bft_core.srcs/sources_1/imports/hdl/FifoBuffer.v.
  save_project_as project_bft_core_1 C:/Users/tanders/AppData/Roaming/Xilinx/Vivado/project_bft_core_1 -force (1 info)
    [Designutils 20-1107] Saved project location 'C:/Users/tanders/AppData/Roaming/Xilinx/Vivado/project_bft_core_1'
Analysis Results (2 critical warnings, 17 infos)
  sources_1 (1 critical warning, 8 infos)
  sim_1 (1 critical warning, 9 infos)
Synthesis (7 warnings, 135 infos)
Implementation (6 warnings, 49 infos)
  Design Initialization (3 infos)
  Opt Design (7 infos)
```

Figure 1: Messages Window

Some messages include hyperlinks to a file or a design element to help in debugging. Click the file name or line number to view the source. Use the popup menu to copy messages in full.

Message Severities

Table 1: Message Severities

Icon	Severity	Message
ℹ	Info	General status of the process and feedback regarding design processing.
⚠	Warning	Design results may be sub-optimal because constraints or specifications may not be applied as intended.
❗	Critical Warning	Certain user input or constraints will not be applied, or are outside the best practices, which usually leads to an error later on in the flow. Examine their sources and constraints. Changes are highly recommended.
⛔	Error	An issue that renders design results unusable and cannot be resolved without user intervention. The design flow stops.

Message Filtering

Use message filtering to view the most severe message type for each action. To filter out a message type, click the check mark next to a message severity. If there is a Critical Warning or an Error during design load, the tool may take you to the message tab and filter out the other messages.

Message Types

There are two types of messages in the tool:

- Messages stored on disk
- Messages stored in memory

New Processes

When you run any of the following commands, the tool starts a new process:

- Run Synthesis
- Run Implementation
- `launch_runs` (TCL)

The process generates messages and reports that persist on disk until you reset the run. Messages that relate to a run appear when a project is open. The tool displays only the messages for the active run in the Messages window.

Messages Created During Design Load

Messages created while loading one of the following designs appear under the relevant design folder:

- Elaborated Design
- Synthesized Design
- Implemented Design

The folder appears when the design is loaded into memory.

Messages Created During Project Open

Messages may appear when you open a project in either the Vivado IDE graphical user interface or in the TCL console. These messages persist only while the design is open.

Message ID Numbers

Messages have unique ID numbers. Use this ID to search the documentation for information about the messages. You may have different message priorities than the tool defaults.

Changing Message Severity

To increase or reduce the severity of a message, run the `set_msg_severity` command. For example:

```
set_msg_severity "Common 17-81" "CRITICAL WARNING"
```

Reports

Reports include:

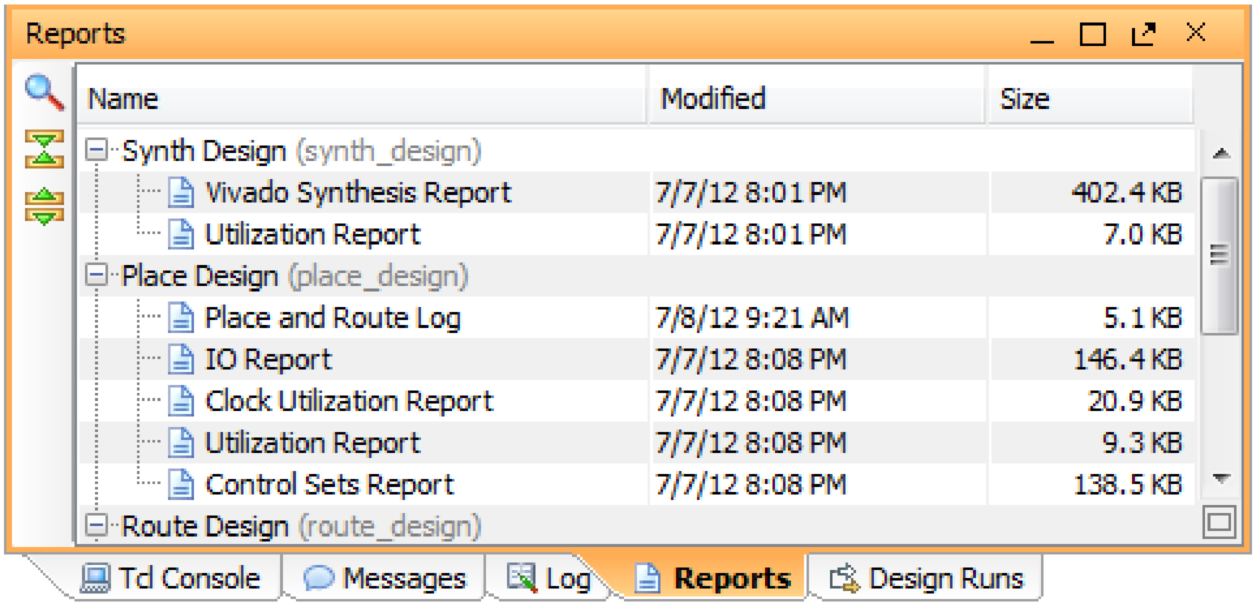
- Generated Reports
- Vivado Synthesis Report
- Control Sets Report
- DRC Report
- Route Status Report
- Timing Summary Report
- WebTalk Report

Report Sources

Reports result from a variety of actions in the Vivado IDE:

- When you load a design, many different reporting commands are available through the Tools menu.
- Running Synthesis or Implementation creates reports as part of the run.
- Other reports are generated when a design is loaded into memory.

The reports for the active Synthesis and Implementation runs appear in the Reports window. Double click a report to view it in the text viewer. Use the Reports tab of the Run Properties window to view reports of the run selected in the Design Runs window.



Name	Modified	Size
Synth Design (synth_design)		
Vivado Synthesis Report	7/7/12 8:01 PM	402.4 KB
Utilization Report	7/7/12 8:01 PM	7.0 KB
Place Design (place_design)		
Place and Route Log	7/8/12 9:21 AM	5.1 KB
IO Report	7/7/12 8:08 PM	146.4 KB
Clock Utilization Report	7/7/12 8:08 PM	20.9 KB
Utilization Report	7/7/12 8:08 PM	9.3 KB
Control Sets Report	7/7/12 8:08 PM	138.5 KB
Route Design (route_design)		

Figure 2: Reports Window

Generated Reports

Generated Reports include:

- Reports Generated During Synthesis
- Reports Generated During Placement
- Reports Generated During Route
- Reports Generated During Bitstream Generation

Reports Generated During Synthesis

Reports generated during synthesis include:

- Vivado Synthesis Report
- Utilization Report (Post Synthesis)

Reports Generated During Placement

Reports generated during placement include:

- Place and Route log
- IO Report
- Clock Utilization Report
- Utilization Report (Post Placement)
- Control Sets Report

Reports Generated During Route

Reports generated during route include:

- WebTalk Report
- DRC Report
- Route Status Report
- Timing Summary Report

Reports Generated During Bitstream Generation

Output from bitstream generation is written to the log file. The only report generated during bitstream generation is the WebTalk Report.

Vivado Synthesis Report

The Vivado Synthesis Report is the primary output from the Vivado Synthesis tool including:

- The files processed:
  - VHDL
  - Verilog
  - SystemVerilog
  - XDC
- Parameter settings per cell
- Nets with Multiple Drivers
- Undriven hierarchical pins
- Optimization information
- Black boxes
- Final Primitive count
- Cell usage by Hierarchy
- Runtime and memory usage

Review this report or the proper section of the messages tab for Errors, Critical Warnings and Warnings. The Synthesis tool can issue Critical Warnings and Warnings that become more serious later in the flow.

Utilization Report

The Utilization Report is generated during various steps in the flow by the `report_utilization` command.

The report includes the device used for the run and utilization for:

- Slice Logic
  - LUT
  - MuxFx
  - Register

- Memory
  - BlockRam
  - FIFO
- DSP48
- I/O resources
- Clocking Resources, including:
  - BUFGCTRL
  - BUFR
  - BUFHCE
  - MMCME2_ADV
  - PLLE2_ADV
- Specific Device Resources, including:
  - STARTUPE2
  - XADC
- Sorted Primitive count by type used
- Black Boxes

When run from the Tcl console, the report can include usage of a particular hierarchical cell when using the `-cells` option. When run from the Vivado IDE graphical user interface, this information appears in an interactive table.

The numbers may change at various points in the flow, when logic optimization commands change the netlist.

Place and Route Log

The Place and Route Log includes:

- Information about the location, netlist, and constraints used.
- Logic optimization task. The tool runs logic optimization routines by default to generate a smaller and faster netlist.
- The placement phases, plus a post-placement timing estimate (WNS and TNS only).
- The router phases , plus several timing estimates and a post-routing timing summary (WNS, TNS, WHS and THS only).
- Elapsed time and memory, for each implementation command and phases.

Review this report or the proper section of the messages tab for Errors, Critical Warnings and Warnings related to the design. The Placer generates warnings that may be elevated to Errors later in the flow. If using Stepwise runs, the log contains only the results for the last step.

IMPORTANT Review the Timing Summary Report to also see the Pulse Width timing summary, and get more information about any timing violations or missing constraints.

IO Report

The IO Report replaces the ISE PAD file. The IO Report lists:

- Pin Number

  All the pins in the device

- Signal Name

  The name of the user IO assigned to the pin

- Pin Usage

  The type of pad or buffer used by the pin

- Pin Name

  Name of the pin

- Direction

  Whether the pin is an input, output, inout, or unused

- IO Standard

  The IO standard for the User IO

  An asterisk (*) indicates that it is the default. This differs from the IO Ports window of the Vivado IDE.

- IO Bank Number

  The IO Bank where the pin is located

- Drive (mA)

  The drive strength in milliamps

- Slew Rate

  The Slew Rate configuration of the buffer: Fast or Slow

- Termination

  The off chip termination settings.

- IOB Delay

  Delay value set for this pin

- Voltage

  The values for various pins, including VCCO, VCCAUX, and related pins
- Constraint

  Displays Fixed if the pin has been constrained by the user
- IOB Sequential Element

  The Flip Flops packed into the IO Bank next to the given port
- Signal Integrity

  The Off-Chip Termination setting

Clock Utilization Report

The Clock Utilization Report helps you analyze the utilization of clock resources inside the device. It can be useful for debugging clock placement issues.

The Clock Utilization Report displays:

- The number of clocking primitives available, occupied and constrained
- Loading and skew per BUFG

  Look for nets with large maxdelay and skew.
- Loading and skew per MMCM

  Look for nets with unexpected loading, large maxdelay, and skew.

Regional Clocks Regional clock networks are clock networks independent of the global clock network. Unlike global clocks, the span of a regional clock signal (BUFR) is limited to one clock region. One I/O clock signal drives a single bank. These networks are especially useful for source-synchronous interface designs. The I/O banks in Xilinx® 7 series FPGA devices are the same size as a clock region.

Local Clocks Local clocks are clock networks routed onto general routing resources. Avoid local clocks where possible. They can experience very large clock skew, and are more susceptible to PVT variations. The tools may route the clock differently each time you rerun implementation.

The Locked column for the clocking resources shows whether you placed the clock, or whether the tools are free to place the clocking resource.

If there are too many global clocks, consider moving low fanout global clocks to other clocking resources, such as, for example, BUFH or BUFR.

Control Sets Report

A control set is the unique combination of a clock signal, a clock enable signal, and a set/reset signal. Each slice supports at most one control set which can be used by any

Flip Flop located in it. Flip flops with different control sets cannot be placed in the same slice.

The Control Sets Report lists the number of unique control sets in the design. Based on the placement of the designs, the tool displays the minimum number of register sites lost to the control set packing.

- Clock Signal

  The logical clock signal name

- Enable Signal

  The logical clock enable signal name

- Set/Reset Signal

  The logical set/reset signal name

- Slice Load Count

  The number of unique slices that contain cells connected to the control set

- BEL Load Count

  The number of cells connected to the control set

DRC Report

The DRC Report is generated by the router. Before the router runs, the tool checks for a common set of design issues. The report lists the checks used in the run. Review the Critical Warnings. The severity of a particular check may be increased later in the flow.

Route Status Report

The Route Status report displays:

- The number of nets in the design
- The number of nets that do not use routing resources outside of a tile, including:
  - CLB
  - BlockRam
  - IO Pad
- The number of nets that do need routing resources

For a design to be fully routed this report should show:

```
# of unrouted nets............ : 0 :
```

Timing Summary Report

The Timing Summary report shows whether a design met timing. This is the equivalent of the ISE TWR. The report includes the following sections.

Summary

Summary shows the following:

- Design name
- Part
- Speedfile
- Vivado IDE version
- Time generated
- Command line

Timer Settings

Timer Settings shows the configuration of the timing engine used for generating this report.

Check Timing

Check Timing shows missing or suspicious timing constraints.

Design Timing Summary

The Design Timing Summary shows whether the design met the constraints. When any TNS, THS, or TPWS is not 0ns, the design did not meet timing.

Clock Summary

The Clock Summary shows a report of each clock defined by the user or by the timing engine, its waveform, period, and requested frequency.

Intra Clock Table

The Intra Clock Table is a summary of the timing on paths within a clock domain.

Inter Clock Table

The Inter Clock Table is a summary of timing for each clock domain crossing.

Path Group Table

The Path Group Table is a timing summary for each path group other than the default clock ones. This includes all the asynchronous timing check reports (recovery/removal), as well as user defined groups (see the `group_path` command)

Timing Details

Timing Details provides the details of the N-worst timing paths reported for each clock pair in the Intra-Clock, Inter-Clock, and Path Group sections

After an implementation run finishes, you must manually open the report. The tool does not open it for you. Use the Report Timing Summary dialog box to create an interactive report in the Vivado IDE.

WebTalk Report

The WebTalk Report is generated during Route and Bitstream. The report collects information about how you use Xilinx parts. This helps Xilinx provide you with better software. No proprietary information is collected.

For more information, go to http://www.xilinx.com/ise/webtalk/

On Demand Reports

To generate other reports, you must first load the design in memory. Commonly used reports are available from the Flow Navigator.

Elaborated Design

- Report DRC
- Report Noise
- Report Power

Synthesized and Implemented Design

- Report Timing Summary
- Report Clock Networks
- Report Clock Interaction
- Report DRC
- Report Noise
- Report Utilization

Implemented Design has all the reports in Synthesized design plus Report Power.

You can also select **Tools > Timing** to generate timing-related reports. For example, when a Synthesized Design is loaded in memory, you have access to:

- Report Timing
- Report Timing Summary
- Report Pulse Width
- Report Clock Interaction

The following sections give more information on each of the reports, including:

- How to run the report.
- The information it provides.

TIP Not all reports are useful at all steps in the flow. RTL Analysis supports only a subset of the reports.

Report DRC

Report DRC runs common Design Rule checks to look for common design issues and errors.

Elaborated Design

The tool checks for DRCs related to IO and Clock Placement. The RTL netlist typically does not have all the IO Buffers, Clock Buffers, and other primitives the post synthesis designs have. Elaborated Design DRCs do not check for as many errors as later DRCs.

Synthesized Design and Implemented Design

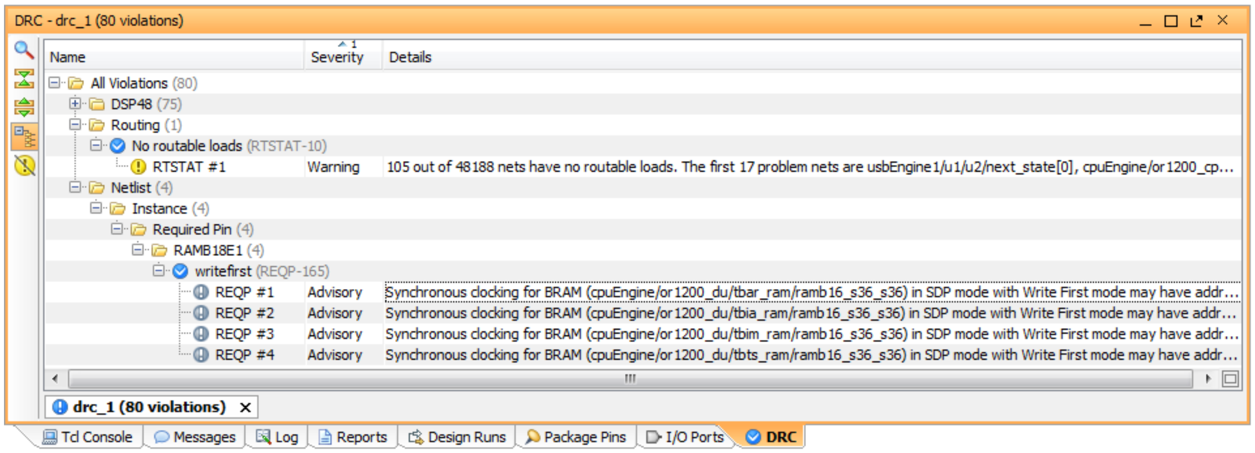
- Checks for DRCs related to the post synthesis netlist.
- Checks IO, BUFG, and other placement.
- Basic checks on the attributes wiring on MGTs, IODELAYs, and other primitives.
- The same DRCs run taking into account any available placement and routing.
- DRCs have the same four severities as Messages. Critical Warnings and Errors do not block the flow at this point.

Steps of the implementation flow also run the DRCs, which can stop the flow at critical points. The placer and router check for issues that block placement. Certain messages have a lower severity depending on the stage. These are DRCs flagging conditions that do not stop `opt_design`, `place_design`, or `route_design` from completing, but which can lead to issues on the board.

For example, some DRCs check that the user has manually constrained the package pin location and the IO standard for all design ports. If some of these constraints are missing, `place_design` and `route_design` issue critical warnings. However, these DRCs appear as an ERROR in `write_bitstream`. The tools will not program a part without these constraints.

The decreased severity earlier in the flow allows you to run the design through implementation iterations before the final pinout has been determined. You must run bitstream generation for a comprehensive DRC signoff.

The following figure (DRC Report) shows the Vivado IDE graphical user interface form of Report DRC.



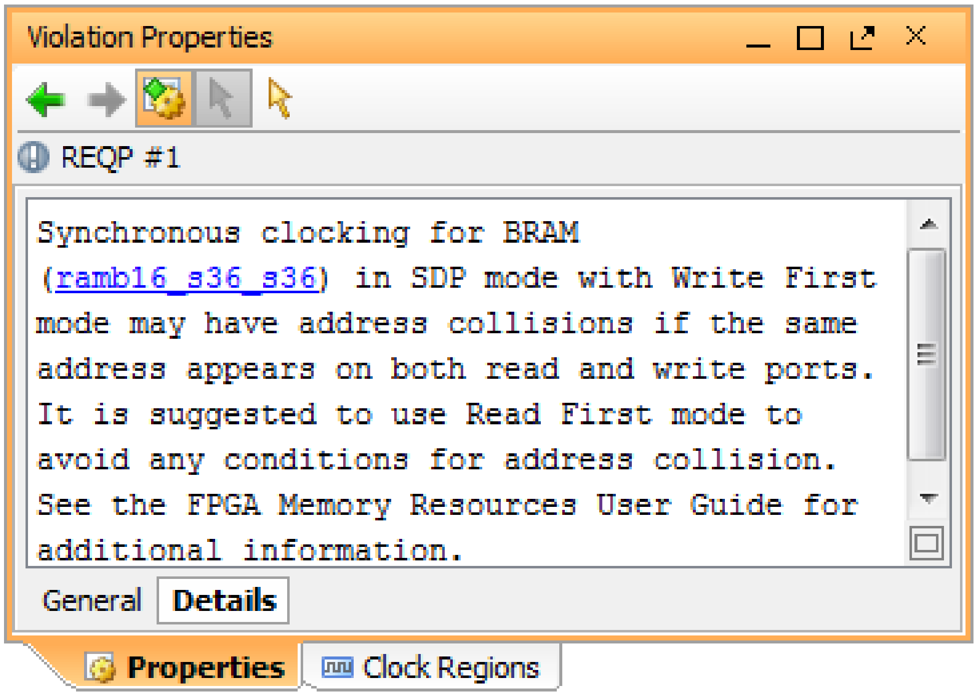
DRC - drc_1 (80 violations)

Name	Severity	Details
All Violations (80)		
DSP48 (75)		
Routing (1)		
No routable loads (RTSTAT-10)		
RTSTAT #1	Warning	105 out of 48188 nets have no routable loads. The first 17 problem nets are usbEngine1/u1/u2/next_state[0], cpuEngine/or1200_cp...
Netlist (4)		
Instance (4)		
Required Pin (4)		
RAMB18E1 (4)		
writefirst (REQP-165)		
REQP #1	Advisory	Synchronous clocking for BRAM (cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36) in SDP mode with Write First mode may have addr...
REQP #2	Advisory	Synchronous clocking for BRAM (cpuEngine/or1200_du/tbia_ram/ramb16_s36_s36) in SDP mode with Write First mode may have addr...
REQP #3	Advisory	Synchronous clocking for BRAM (cpuEngine/or1200_du/tbim_ram/ramb16_s36_s36) in SDP mode with Write First mode may have addr...
REQP #4	Advisory	Synchronous clocking for BRAM (cpuEngine/or1200_du/tbts_ram/ramb16_s36_s36) in SDP mode with Write First mode may have addr...

drc_1 (80 violations)

Figure 3: DRC Report

Click a DRC to open the properties for a detailed version of the message. Look in the Properties window to see the details. Most messages have a hyperlink for nets, cells, and ports referenced in the DRC.



Synchronous clocking for BRAM (ramb16_s36_s36) in SDP mode with Write First mode may have address collisions if the same address appears on both read and write ports. It is suggested to use Read First mode to avoid any conditions for address collision. See the FPGA Memory Resources User Guide for additional information.

Figure 4: DRC Properties

The DRC report is static. You must re-run Report DRC for the report to reflect design changes. The tool determines that the links are stale after certain design operations (such as deleting objects and moving objects), and invalidates the links.

Selecting an object from the hyperlink selects the object, but does not refresh the Properties window. To see the properties for the object, you must unselect and reselect it.

To create a DRC report in Tcl, run the command:

```
report_drc
```

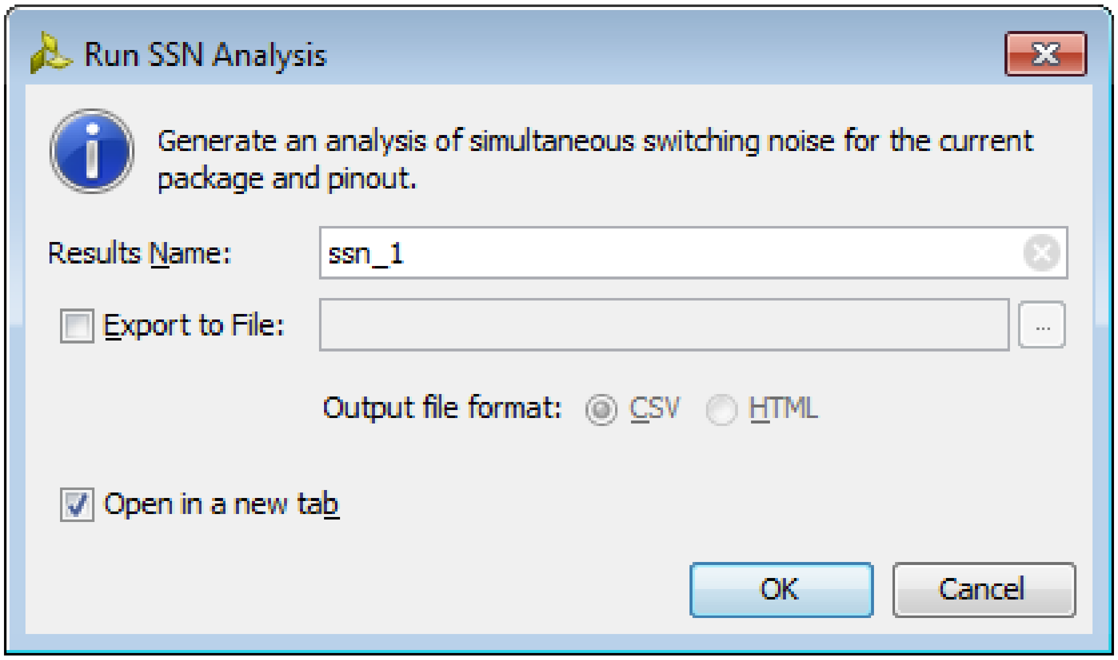
To send the results to a file:

```
report_drc -file myDRCs.txt
```

TIP For a more information on the `report_drc` command, run `report_drc -help`

Noise Report

Performs the Simultaneous Switching Noise (SSN) calculation for Xilinx 7 series FPGA devices. By default, the SSN report opens in a new tab in the Results window area of the Vivado IDE. You can export the results to a CSV or HTML file.


Run SSN Analysis

Generate an analysis of simultaneous switching noise for the current package and pinout.

Results Name: ssn_1

☐ Export to File:

Output file format: [✓ CSV] HTML

☑ Open in a new tab

Figure 5: Run SSN Dialog Box

The Noise Report has four sections:

- Summary
- Messages
- I/O Bank Details
- Links

Summary

The Summary section includes:

- When the report ran
- Number and percentage of applicable ports analyzed
- Status, including whether it passed
- Number of Critical Warnings, Warnings, and Info messages

Messages

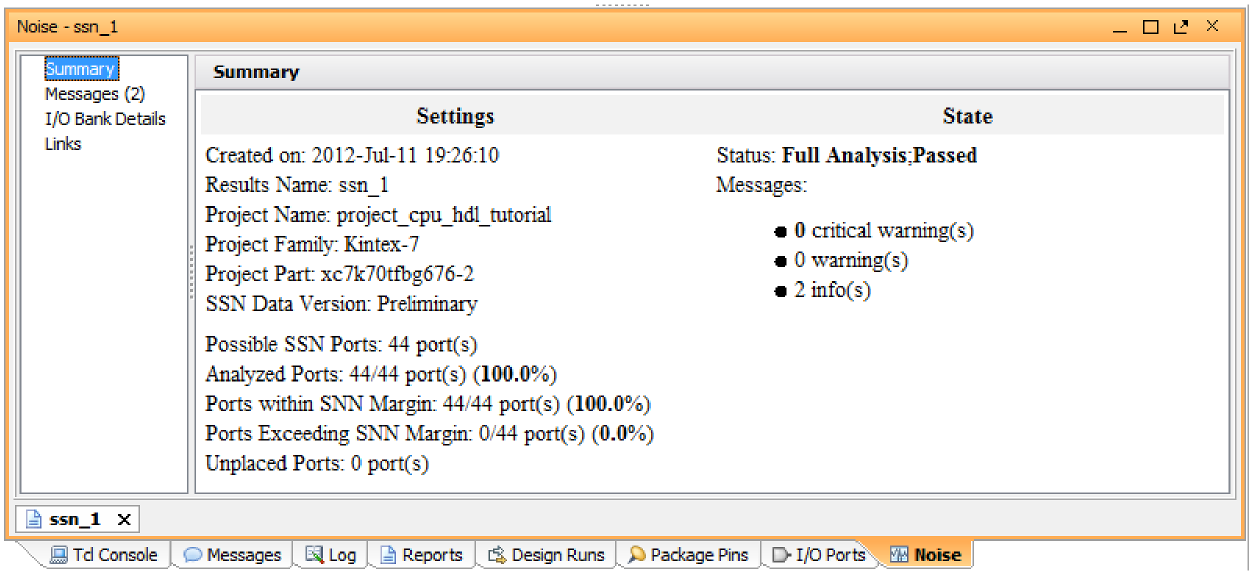
The Messages section includes a detailed list of the messages generated during the report.

I/O Bank Details

The I/O Bank Details section includes a list of Pins, Standards, and Remaining Margin.

Links

The Links section contains links to relevant documentation on www.xilinx.com/support.



Summary

Settings	State
Created on: 2012-Jul-11 19:26:10	Status: **Full Analysis;Passed**
Results Name: ssn_1	Messages:
Project Name: project_cpu_hdl_tutorial	● **0** critical warning(s)
Project Family: Kintex-7	● 0 warning(s)
Project Part: xc7k70tfbg676-2	● 2 info(s)
SSN Data Version: Preliminary	
Possible SSN Ports: 44 port(s)	
Analyzed Ports: 44/44 port(s) (**100.0%**)	
Ports within SNN Margin: 44/44 port(s) (**100.0%**)	
Ports Exceeding SNN Margin: 0/44 port(s) (**0.0%**)	
Unplaced Ports: 0 port(s)	

Figure 6: Noise Report

To create an HTML version of the report, select the option or run the following Tcl command:

```
report_ssn -format html -file myImplementedDesignSSN.html
```

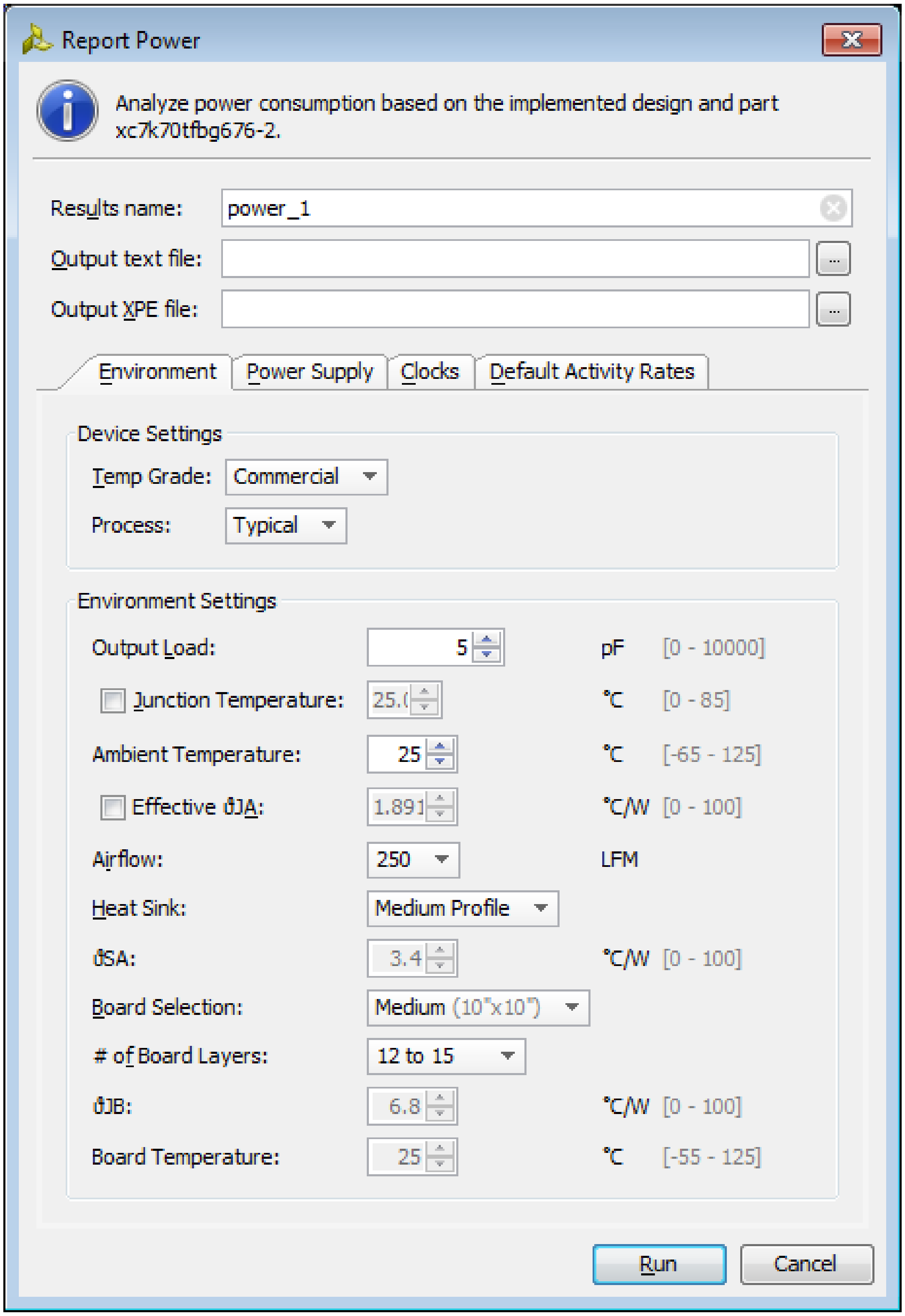
Power Report

Report Power is available when a Synthesized Design or Implemented Design is open.

The report estimates power consumption and junction temperature based on design inputs including:

- Thermal statistics, such as junction and ambient temperature values.
- Data on board selection, including number of board layers and board temperature.
- Data on the selection of airflow and the head sink profile used by the design.

- Reporting the FPGA current requirements from the different power supply sources.
- Allowing detailed power distribution analysis to guide power saving strategies and to reduce dynamic, thermal or off-chip power.



Report Power

Analyze power consumption based on the implemented design and part xc7k70tfbg676-2.

Results name:	power_1	
Output text file:		...
Output XPE file:		...

Device Settings

Temp Grade:	Commercial
Process:	Typical

Environment Settings

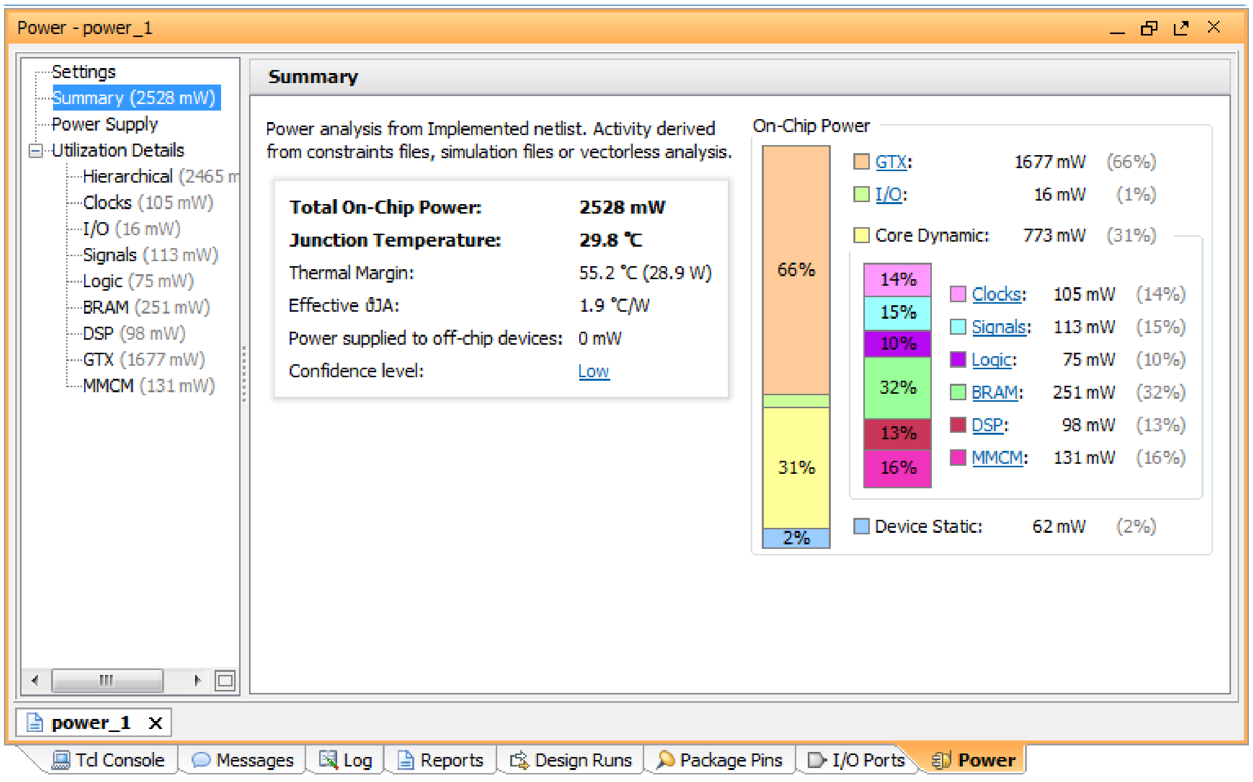
Output Load:	5	pF	[0 - 10000]
☐ Junction Temperature:	25.(	°C	[0 - 85]
Ambient Temperature:	25	°C	[-65 - 125]
☐ Effective θJA:	1.891	°C/W	[0 - 100]
Airflow:	250	LFM	
Heat Sink:	Medium Profile		
θSA:	3.4	°C/W	[0 - 100]
Board Selection:	Medium (10"x10")		
# of Board Layers:	12 to 15		
θJB:	6.8	°C/W	[0 - 100]
Board Temperature:	25	°C	[-55 - 125]

Figure 7: Report Power

Analyzing the Power Report

You can analyze the Power Report, shown in the figure above, based on:

- Settings
- Power total
- By hierarchy
- Per Voltage rail
- By block type



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**2528 mW**
Junction Temperature:	**29.8 ℃**
Thermal Margin:	55.2 ℃ (28.9 W)
Effective ϑJA:	1.9 ℃/W
Power supplied to off-chip devices:	0 mW
Confidence level:	Low

On-Chip Power

GTX:	1677 mW	(66%)	66%
I/O:	16 mW	(1%)	
Core Dynamic:	773 mW	(31%)	31%
Clocks:	105 mW	(14%)	14%
Signals:	113 mW	(15%)	15%
Logic:	75 mW	(10%)	10%
BRAM:	251 mW	(32%)	32%
DSP:	98 mW	(13%)	13%
MMCM:	131 mW	(16%)	16%
Device Static:	62 mW	(2%)	2%

Figure 8: Power Report

For more information on the power report and analyzing the results, see *Power Analysis and Optimization (UG907).*

Reporting Power in a Non-Project Flow

In the non-project flow, `report_power` is available after `link_design` or `synth_design`. The report uses the available placement and routing to give more accurate power numbers. To generate this report from the Tcl Console or a script , use the following command:

```
report_power
```

Report Timing Summary

Timing analysis is available anywhere in the flow after synthesis. You can review the Timing Summary report files automatically created by the Synthesis and Implementation runs. If your synthesized or implemented design is loaded in memory, you can also generate an interactive Timing Summary report from the Flow Navigator, under Synthesis or Implementation, or from the menu **Tools > Timing > Report Timing Summar**y. The equivalent Tcl command is *report_timing_summary*. For more information on the options of this command, see the *Vivado Design Suite Tcl Reference Guide (UG835*).

In a Synthesized Design, the Vivado IDE timing engine estimates the net delays based on connectivity and fanout. The accuracy of the delays is greater for nets between cells that are already placed by constraints.

In an Implemented Design, the net delays are based on the actual routing information. You must use the Timing Summary report for timing signoff if your design is completely routed, which you can verify by reading the Route Status report.

Report Timing Summary Dialog Box Options

In the Vivado IDE, the Report Timing Summary dialog box includes the following tabs:

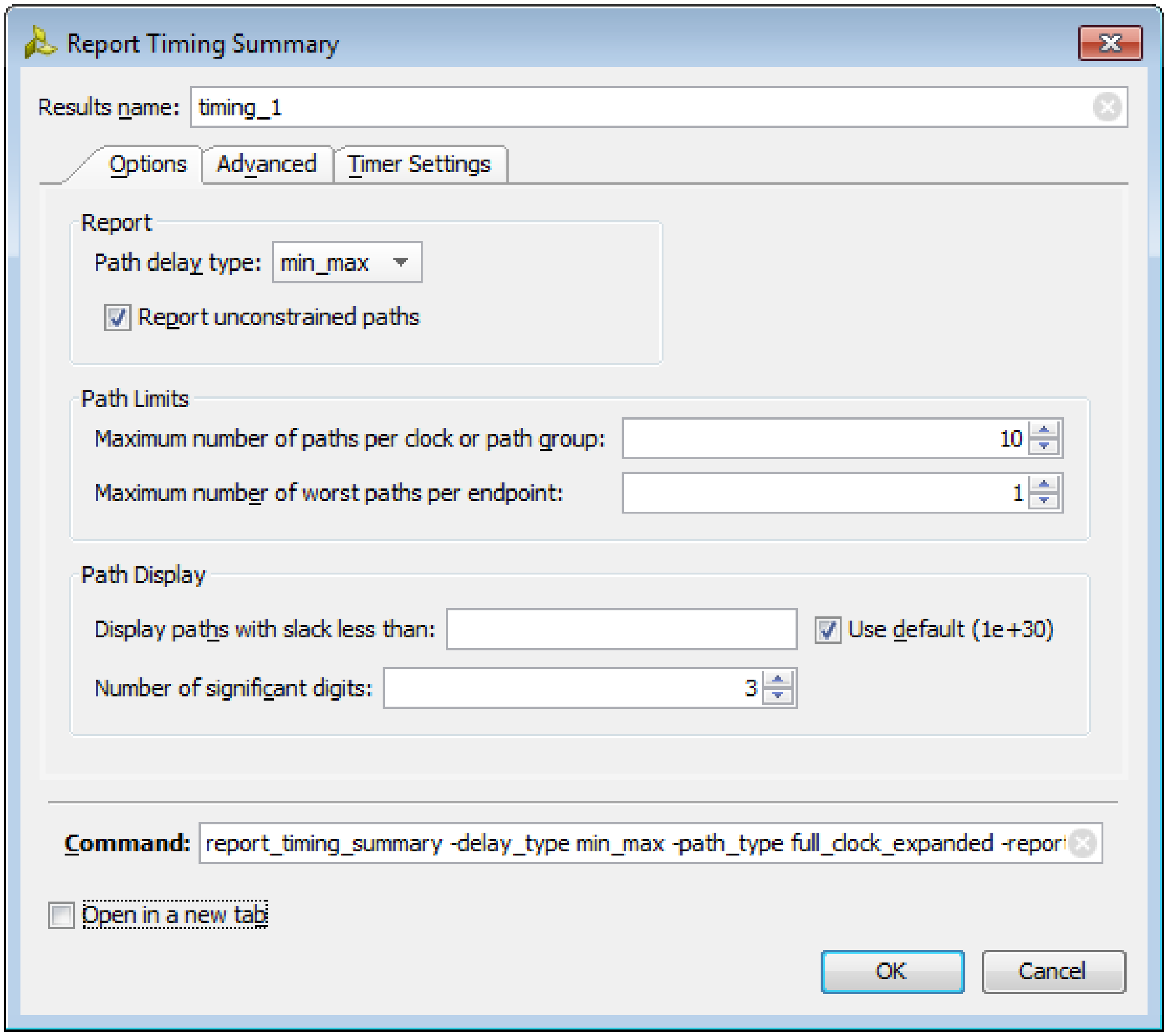
- Options
- Advanced
- Timer Settings

You can choose to open the results in a new window by specifying its name in the **Results name** field. If this field is empty, the report is printed in the Tcl Console. You can also delete the reports previously created by unchecking **Open in a new tab** at the bottom of the dialog box.

The equivalent *report_timing_summary* options are listed along the dialog options in the following section.

Options Tab

The following figure (Report Timing Summary Dialog Options Tab) shows the Options tab in the Report Timing Summary dialog box.



Results name: timing_1

Path delay type: min_max

[✓] Report unconstrained paths

Maximum number of paths per clock or path group: 10

Maximum number of worst paths per endpoint: 1

Display paths with slack less than: [✓] Use default (1e+30)

Number of significant digits: 3

Command: report_timing_summary -delay_type min_max -path_type full_clock_expanded -repor

Figure 9: Report Timing Summary Dialog Options Tab

The following options are available in the Options tab:

- **Reports**
  - **Path delay type**: sets the type of analysis to be run. For synthesized designs, only max delay analysis (setup/recovery) is performed by default. For implemented design, both min and max delay analysis (setup/hold, recover/removal) are performed by default. You can elect to run min delay analysis only (hold and removal) by selecting the "min" value.

    Equivalent Tcl option: -*delay_type*

  - **Path report format**: controls the amount of detail displayed for each reported timing path.

    Equivalent Tcl option: *-path_type*
  - **Report unconstrained paths**: generates information on paths that do not have timing requirements. This option is checked by default in the Vivado IDE, but is not turned on by default in the equivalent Tcl command report_timing_summary.

    Equivalent Tcl option: *-report_unconstrained*
- **Path Limits**
  - **Maximum number of paths per clock or path group**: controls the maximum number of paths reported per clock pair or path group.

    Equivalent Tcl option: *-max_paths*
  - **Maximum number of worst paths per endpoint**: controls the maximum number of paths potentially reported per path endpoint. This limit is bounded by the maximum number of paths per clock pair or path group.

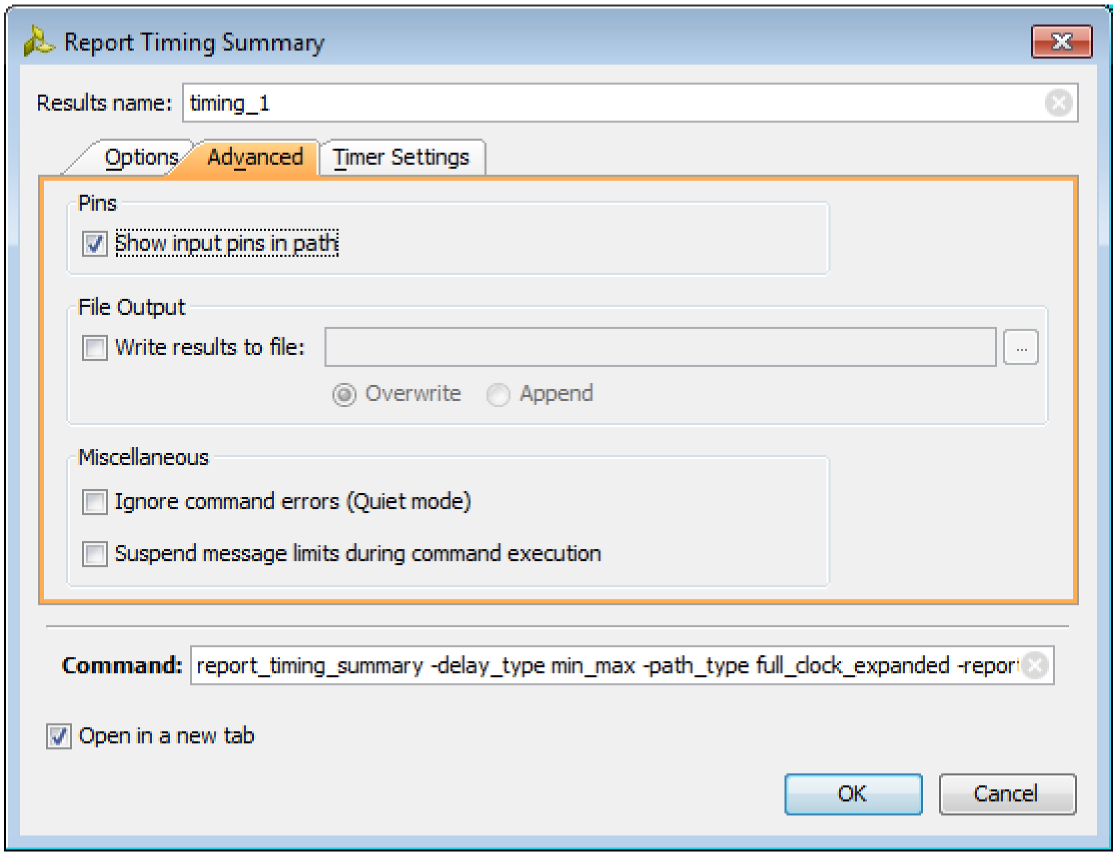
    Equivalent Tcl option: *-nworst*
- **Path Display**
  - **Display paths with slack less than**: filters the reported paths based on their slack value. This option does not affect the content of the summary tables.

    Equivalent Tcl option: *-slack_lesser_than*
  - **Number of significant digits**: controls the accuracy of the numbers displayed in the report.

    Equivalent Tcl option: *-significant_digits*

Advanced Tab

The following figure (Report Timing Summary Dialog Advanced Tab) shows the advanced tab in the Report Timing Summary dialog box.



Report Timing Summary

Results name: timing_1

Options | Advanced | Timer Settings

Pins

[✓] Show input pins in path

File Output

[ ] Write results to file:

(●) Overwrite ( ) Append

Miscellaneous

[ ] Ignore command errors (Quiet mode)

[ ] Suspend message limits during command execution

Command: report_timing_summary -delay_type min_max -path_type full_clock_expanded -repor

[✓] Open in a new tab

OK Cancel

Figure 10: Report Timing Summary Dialog Advanced Tab

The following options are available:

- **Pins**: show input pins in the path details. Xilinx recommends that you keep this option selected.

  Equivalent Tcl option: -*input_pins*

- **File Output**: redirects the output of the command to a file, and optionally append it to an existing report file.

  Equivalent Tcl options: -*file* and -*append*

- Miscellaneous:
  - **Ignore command errors:** force the tool to ignore any error.

    Equivalent Tcl option: -*quiet*

- **Suspend message limits during command execution**: turns on the verbose mode.

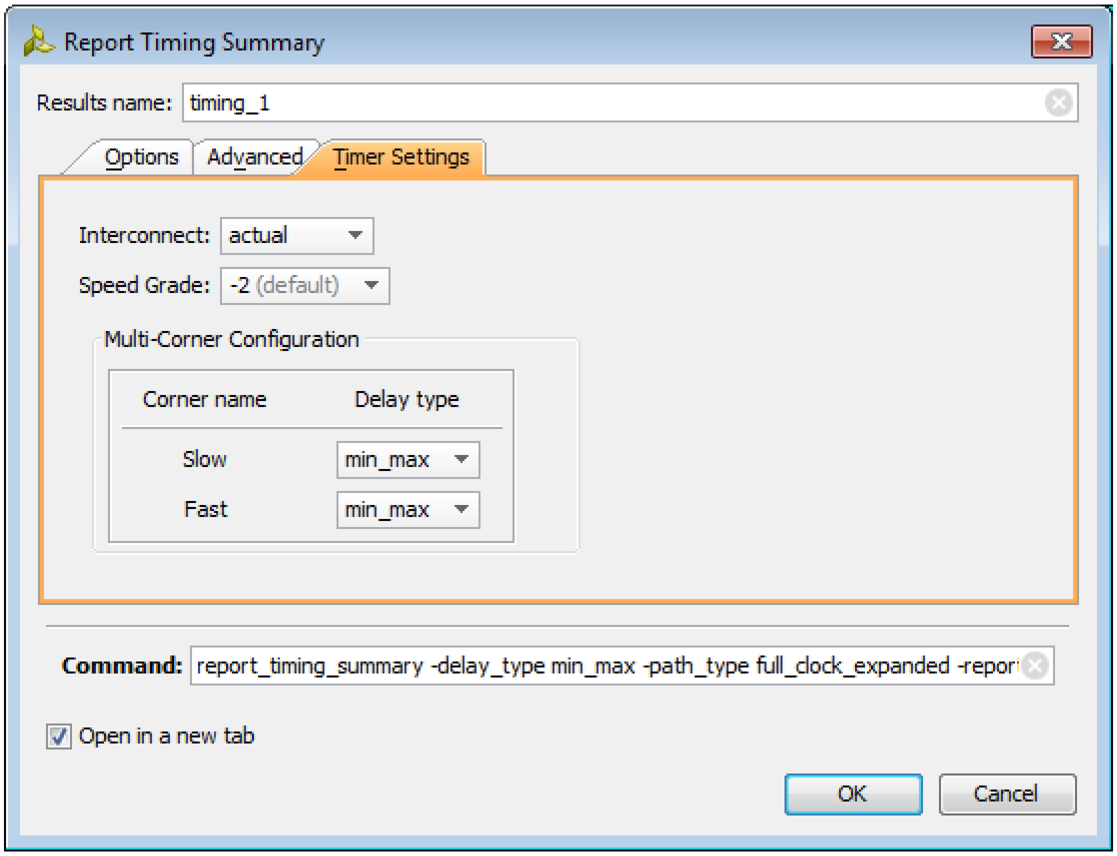
  Equivalent Tcl option: *-verbose*

Timer Settings Tab

To set the timer settings, use either the dialog box or the Tcl Console. These settings affect other timing-related commands run in the same Vivado IDE session, except the synthesis and implementation ones. The timer settings are not saved as tool preference, and the default values are restored for each new session.

RECOMMENDED Do not change the default values. Keeping the default values provides maximum timing analysis coverage with the most accurate delay values.

The following figure (Report Timing Summary Dialog Timer Settings Tab) shows the Timer Settings tab in the Report Timing Summary dialog box.


Report Timing Summary

Results name: timing_1

Interconnect: actual

Speed Grade: -2 (default)

Multi-Corner Configuration

Corner name	Delay type
Slow	min_max
Fast	min_max

Command: report_timing_summary -delay_type min_max -path_type full_clock_expanded -report

Open in a new tab [✓]

Figure 11: Report Timing Summary Dialog Timer Settings Tab

The following options are available in the Timer Settings tab:

- **Interconnect**: controls how the net delays are calculated. This option is automatically set to **Estimated** for post-synthesis designs and to **Actual** for post-implementation designs.
  - **Estimated:**

    For unplaced cells, the net delay value corresponds to the delay for the best possible placement, based on the nature of the driver and loads as well as the fanout. This net is labeled *unplaced* in the timing path report.

    For placed cells, the net delay depends on the distance between the driver and the load as well as the fanout. This net is labeled *estimated* in the timing path report.
  - **Actual:**

    For routed nets, the net delay corresponds to the actual hardware delay. This net is labeled *routed* in the timing path report.

    For unrouted nets between placed cells, the net delay depends on the distance between the driver and the load as well as the fanout. This net is labeled *estimated* in the timing report.
  - **None:**

    The net delays are forced to zero.

    Equivalent Tcl command: *set_delay_model*
- **Speed Grade**: sets the speed grade. By default, this option is set based on the part selected when creating a project or opening a design checkpoint. You can change this option to report timing on the same design database against another speed grade without re-running the complete implementation flow.

  Equivalent Tcl command: *set_speed_grade*
- **Multi-Corner Configuration**: specifies the analysis to run for each corner, whether setup (max), hold (min), or both

  RECOMMENDED Always keep both setup (max) and hold (min) analysis selected for both corners.

  Equivalent Tcl command: *config_timing_corner*s

Timing Summary Report Sections

The Report Timing Summary contains the following sections:

- General Information
- Timer Settings
- Design Timing Summary
- Check Timing
- Clock Summary
- Intra-Clock Paths
- Inter-Clock Paths
- Path Groups
- Unconstrained Paths

The comprehensive information contained in the Timing Summary Report is similar to the information provided by several reports available from the Vivado IDE (Report Clock Interaction, Report Pulse Width, Report Timing, Check Timing) and to some of the reports available in Tcl only (`report_clocks`). However, the Report Timing Summary also includes information that is unique to this report, such as Unconstrained Paths.

The following sections describe each section of the report in detail.

General Information

The General Information section provides information about the following:

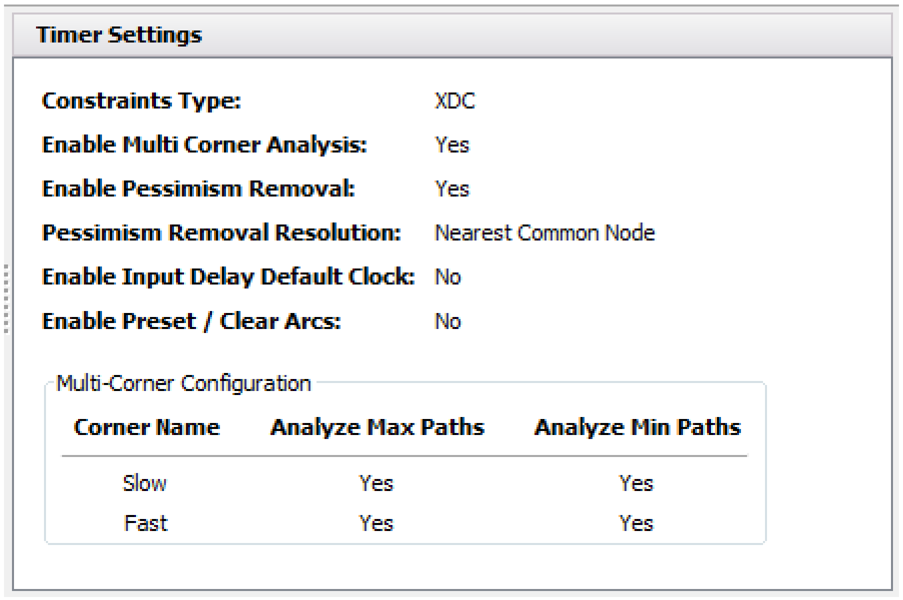
- Design name
- Selected device, package, and speed grade (with the speed file version)
- Vivado Design Suite release
- Current date
- Equivalent Tcl commands executed to generate the report

Timer Settings

The Timer Settings section contains details on the Vivado timing analysis engine settings used to generate the timing information in the report. The following figure (Timing Summary Report Timer Settings) shows an example of the Timer Settings section, which includes:

- Enable Multi-Corner Analysis and which analysis is enabled for each corner (Multi-Corner Configuration).

- Enable Pessimism Removal (and Pessimism Removal Resolution): this setting must always be enabled. It insures that the source and destination clocks of each path are reported with no skew at their common node.
- Enable Input Delay Default Clock: this option creates a default null input delay constraint on input ports with no user constraint. It is disabled by default.
- Enable Preset / Clear Arcs: this option enables timing path propagation through asynchronous pins. It does not affect recovery/removal checks and is disabled by default.



Timer Settings

Constraints Type:	XDC
Enable Multi Corner Analysis:	Yes
Enable Pessimism Removal:	Yes
Pessimism Removal Resolution:	Nearest Common Node
Enable Input Delay Default Clock:	No
Enable Preset / Clear Arcs:	No

Multi-Corner Configuration

Corner Name	Analyze Max Paths	Analyze Min Paths
Slow	Yes	Yes
Fast	Yes	Yes

Figure 12: Timing Summary Report Timer Settings

Design Timing Summary

The Design Timing Summary section provides a summary of the timing for the design, and combines the results of all other sections in one view.

RECOMMENDED Use this summary to efficiently check that all your timing constraints are met after route or to understand the status of your design at any point in the flow:

- **Setup**

  Setup includes all checks related to max delay analysis: setup, recovery and data check.

  - **WNS** (Worst Negative Slack): this value corresponds to the worst slack of all the timing paths for max delay analysis. It can be positive or negative.
  - **TNS** (Total Negative Slack): this value is the sum of all WNS violations, when considering only the worst violation of each timing path endpoint. Its value is 0ns when all timing constraints are met for max delay analysis, or negative when there are some violations.
  - **Failing Endpoints**: this is the total number of endpoints with a violation (WNS<0ns).

- **Hold**

  Hold includes all checks related to min delay analysis: hold, recovery and data check.

  - **WHS** (Worst Hold Slack): this value corresponds to the worst slack of all the timing paths for min delay analysis. It can be positive or negative.
  - **THS** (Total Hold Slack): this value is the sum of all WHS violations, when considering only the worst violation of each timing path endpoint. Its value is 0ns when all timing constraints are met for min delay analysis, or negative when there are some violations.
  - **Failing Endpoints**: this is the total number of endpoints with a violation (WHS<0ns).

- **Pulse Width**

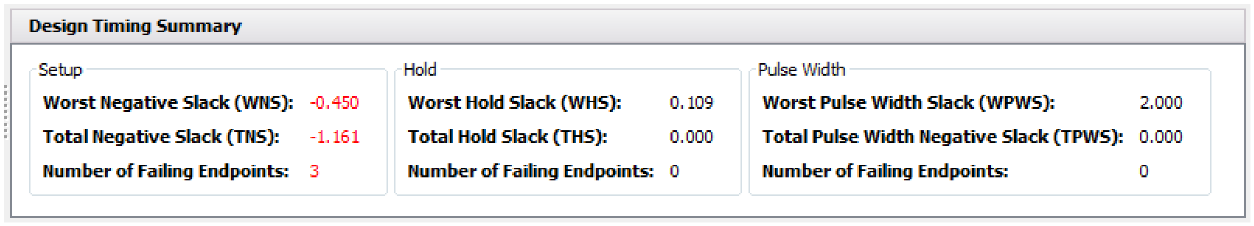
  Pulse width includes all checks related to pin switching limits:

  - Min low pulse width
  - Min high pulse width
  - Min period
  - Max period
  - Max skew (between two clock pins of a same leaf cell). The WPWS value corresponds to the worst pulse width slack for all the checks listed above with min and max and delays. The TWPS value is the sum of all WPWS violations, when considering only the worst violation of each timing path endpoint. Its value can only be 0ns or negative.

The three reported values are:

- **WPWS** (Worst Pulse Width Slack): this value corresponds to the worst slack of all the timing checks listed above when using both min and max delays.
- **TPWS** (Total Pulse Width Slack): this value is the sum of all WPWS violations, when considering only the worst violation of each pin in the design. Its value is 0ns when all related constraints are met, or negative when there are some violations.
- **Failing Endpoints**: this is the total number of pins with a violation (WPWS<0ns).

The following figure (Timing Summary Report Design Timing Summary) shows an example of the Design Timing Summary section.


Design Timing Summary

Setup

Worst Negative Slack (WNS):	-0.450
Total Negative Slack (TNS):	-1.161
Number of Failing Endpoints:	3

Hold

Worst Hold Slack (WHS):	0.109
Total Hold Slack (THS):	0.000
Number of Failing Endpoints:	0

Pulse Width

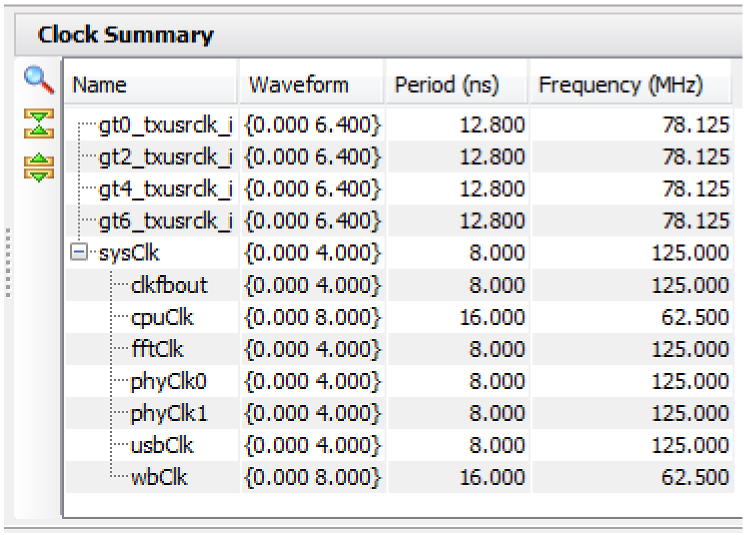
Worst Pulse Width Slack (WPWS):	2.000
Total Pulse Width Negative Slack (TPWS):	0.000
Number of Failing Endpoints:	0

Figure 13: Timing Summary Report Design Timing Summary

Clock Summary

The Clock Summary section contains information similar to that produced by the `report_clocks` command. It shows all the clocks that exist in the design (created by `create_clock`, `create_generated_clock`, or automatically by the tool). It also includes the properties for each clock, such as name, period, waveform, type, and target frequency. The indentation of names reflects the relationship between master and generated clocks.

The following figure (Timing Summary Report Clock Summary) shows an example of the Clock Summary section.



Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
gt0_txusrclk_i	{0.000 6.400}	12.800	78.125
gt2_txusrclk_i	{0.000 6.400}	12.800	78.125
gt4_txusrclk_i	{0.000 6.400}	12.800	78.125
gt6_txusrclk_i	{0.000 6.400}	12.800	78.125
sysClk	{0.000 4.000}	8.000	125.000
clkfbout	{0.000 4.000}	8.000	125.000
cpuClk	{0.000 8.000}	16.000	62.500
fftClk	{0.000 4.000}	8.000	125.000
phyClk0	{0.000 4.000}	8.000	125.000
phyClk1	{0.000 4.000}	8.000	125.000
usbClk	{0.000 4.000}	8.000	125.000
wbClk	{0.000 8.000}	16.000	62.500

Figure 14: Timing Summary Report Clock Summary

Check Timing

The Check Timing section contains useful information about missing timing constraints or paths with constraints issues that need to be reviewed. For complete timing signoff, all path endpoints must be constrained. You can also generate this report independently from the menu (**Tools > Timing > Check Timing**) or with the *check_timing* command.

The list of checks reported by default is as follows:

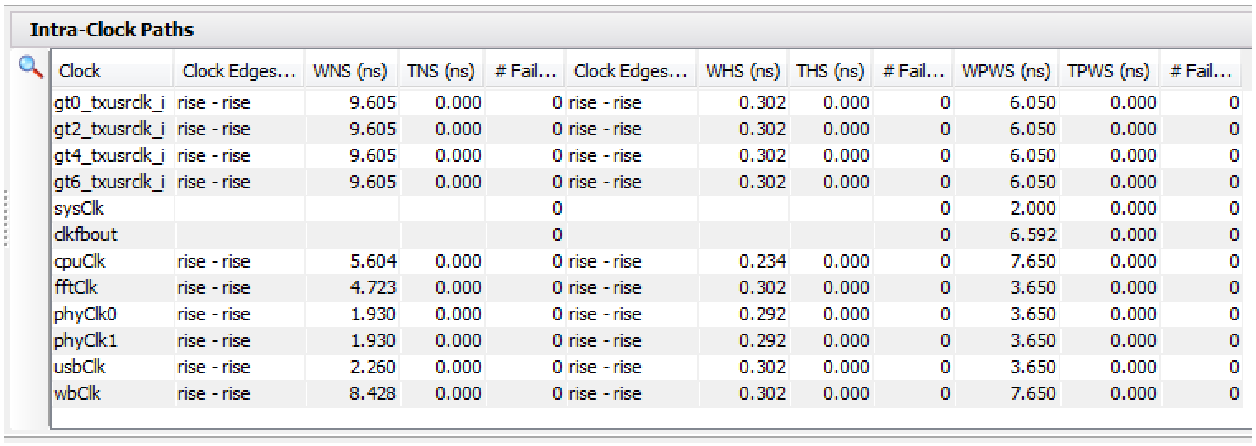
- `no_clock`: number of clock pins not reached by a defined timing clock. Constant clock pins are also reported.
- `unconstrained_endpoints`: number of path endpoints without a timing requirement. This number is directly related to missing clock definitions, which is also reported by the "no_clock" check.
- `no_input_delay`: number of input ports without at least one input delay constraint.
- `no_output_delay`: number of output ports without at least one output delay constraint.
- `multiple_clock`: number of clock pins reached by more than one timing clock. This usually happens when there is a clock multiplexer in one of the clock trees.
- `loops`: number of combinational loops found in the design. The loops are automatically broken by the Vivado timing engine in order to report timing.
- `generated_clocks`: number of generated clocks that refer to a master clock source which is not part of the same clock tree.

- `partial_input_delay`: number of input ports with only a min input delay or max input delay constraint. These ports are not reported by both setup and hold analysis.
- `partial_output_delay`: number of output ports with only a min output delay or max output delay constraint. These ports are not reported by both setup and hold analysis.
- `unexpandable_clocks`: clock pairs for which the Vivado timing engine could not find a common period multiplier over 1000 clock cycles. The paths between these clock pairs cannot be safely timed and the clock pairs must be treated as asynchronous.

You can find more information on constraints definition in the *Vivado Design Suite Using Constraints User Guide (UG903)*.

Intra-Clock Paths

The Intra Clock Paths section summarizes the worst slack and total violations of the timing paths with the same source and destination clocks. The following figure (Timing Summary Report Intra-Clock Paths) shows an example of the Intra-Clock Paths section.



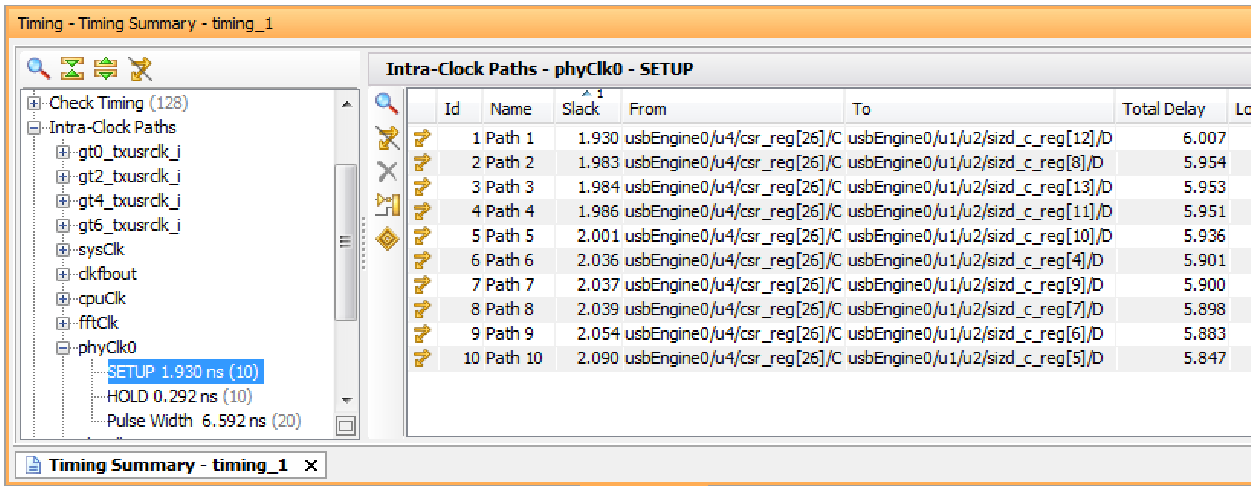
Intra-Clock Paths

Clock	Clock Edges…	WNS (ns)	TNS (ns)	# Fail…	Clock Edges…	WHS (ns)	THS (ns)	# Fail…	WPWS (ns)	TPWS (ns)	# Fail…
gt0_txusrclk_i	rise - rise	9.605	0.000	0	rise - rise	0.302	0.000	0	6.050	0.000	0
gt2_txusrclk_i	rise - rise	9.605	0.000	0	rise - rise	0.302	0.000	0	6.050	0.000	0
gt4_txusrclk_i	rise - rise	9.605	0.000	0	rise - rise	0.302	0.000	0	6.050	0.000	0
gt6_txusrclk_i	rise - rise	9.605	0.000	0	rise - rise	0.302	0.000	0	6.050	0.000	0
sysClk				0				0	2.000	0.000	0
clkfbout				0				0	6.592	0.000	0
cpuClk	rise - rise	5.604	0.000	0	rise - rise	0.234	0.000	0	7.650	0.000	0
fftClk	rise - rise	4.723	0.000	0	rise - rise	0.302	0.000	0	3.650	0.000	0
phyClk0	rise - rise	1.930	0.000	0	rise - rise	0.292	0.000	0	3.650	0.000	0
phyClk1	rise - rise	1.930	0.000	0	rise - rise	0.292	0.000	0	3.650	0.000	0
usbClk	rise - rise	2.260	0.000	0	rise - rise	0.302	0.000	0	3.650	0.000	0
wbClk	rise - rise	8.428	0.000	0	rise - rise	0.302	0.000	0	7.650	0.000	0

Figure 15: Timing Summary Report Intra-Clock Paths

From the index pane on the left, you can click the names under Intra-Clock Paths to view detailed information. For example, you can view the slack and violations summary for each clock and details about the N-worst paths for SETUP/HOLD/Pulse Width checks.

Next to the label for each analysis type, the worst slack value and the number of reported paths are shown. See the following figure (Timing Summary Report Intra-Clock Paths Details).


Timing - Timing Summary - timing_1

- Check Timing (128)
- Intra-Clock Paths
  - gt0_txusrclk_i
  - gt2_txusrclk_i
  - gt4_txusrclk_i
  - gt6_txusrclk_i
  - sysClk
  - clkfbout
  - cpuClk
  - fftClk
  - phyClk0
    - SETUP 1.930 ns (10)
    - HOLD 0.292 ns (10)
    - Pulse Width 6.592 ns (20)

Intra-Clock Paths - phyClk0 - SETUP

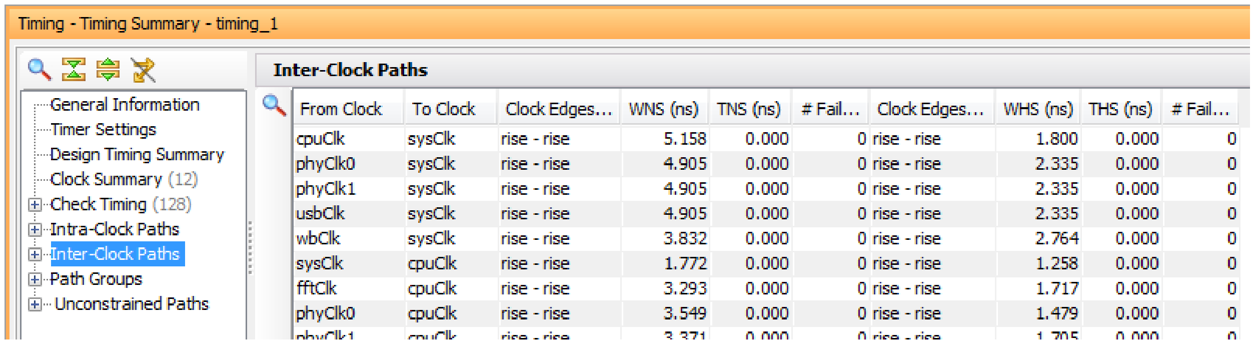
Id	Name	Slack	From	To	Total Delay
1	Path 1	1.930	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[12]/D	6.007
2	Path 2	1.983	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[8]/D	5.954
3	Path 3	1.984	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[13]/D	5.953
4	Path 4	1.986	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[11]/D	5.951
5	Path 5	2.001	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[10]/D	5.936
6	Path 6	2.036	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[4]/D	5.901
7	Path 7	2.037	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[9]/D	5.900
8	Path 8	2.039	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[7]/D	5.898
9	Path 9	2.054	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[6]/D	5.883
10	Path 10	2.090	usbEngine0/u4/csr_reg[26]/C	usbEngine0/u1/u2/sizd_c_reg[5]/D	5.847

Timing Summary - timing_1

Figure 16: Timing Summary Report Intra-Clock Paths Details

Inter-Clock Paths

Similar to the Intra-Clock Paths section, the Inter-Clock Paths section summarizes the worst slack and total violations of timing paths with different source and destination clocks. The following figure (Timing Summary Report Inter-Clock Paths Summary) shows an example of the Inter-Clock Paths section.


Timing - Timing Summary - timing_1

- General Information
- Timer Settings
- Design Timing Summary
- Clock Summary (12)
- Check Timing (128)
- Intra-Clock Paths
- Inter-Clock Paths
- Path Groups
- Unconstrained Paths

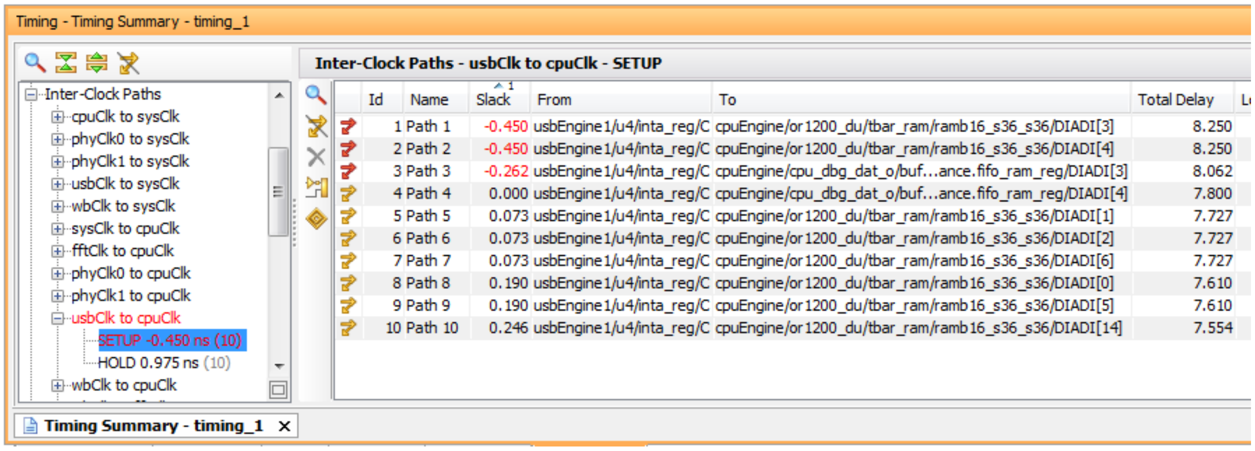
Inter-Clock Paths

From Clock	To Clock	Clock Edges...	WNS (ns)	TNS (ns)	# Fail...	Clock Edges...	WHS (ns)	THS (ns)	# Fail...
cpuClk	sysClk	rise - rise	5.158	0.000	0	rise - rise	1.800	0.000	0
phyClk0	sysClk	rise - rise	4.905	0.000	0	rise - rise	2.335	0.000	0
phyClk1	sysClk	rise - rise	4.905	0.000	0	rise - rise	2.335	0.000	0
usbClk	sysClk	rise - rise	4.905	0.000	0	rise - rise	2.335	0.000	0
wbClk	sysClk	rise - rise	3.832	0.000	0	rise - rise	2.764	0.000	0
sysClk	cpuClk	rise - rise	1.772	0.000	0	rise - rise	1.258	0.000	0
fftClk	cpuClk	rise - rise	3.293	0.000	0	rise - rise	1.717	0.000	0
phyClk0	cpuClk	rise - rise	3.549	0.000	0	rise - rise	1.479	0.000	0
phyClk1	cpuClk	rise - rise	3.371	0.000	0	rise - rise	1.705	0.000	0

Figure 17: Timing Summary Report Inter-Clock Paths Summary

From the index pane on the left, you can click the names under Inter-Clock Paths to view detailed information. For example, you can view the slack and violations summary for each clock and details about the N-worst paths for SETUP/HOLD/Pulse Width checks.

Next to the label of each analysis type, the worst slack value and the number of reported paths are shown. See the following figure (Timing Summary Report Inter-Clock Paths Details).



Timing - Timing Summary - timing_1

- Inter-Clock Paths
  - cpuClk to sysClk
  - phyClk0 to sysClk
  - phyClk1 to sysClk
  - usbClk to sysClk
  - wbClk to sysClk
  - sysClk to cpuClk
  - fftClk to cpuClk
  - phyClk0 to cpuClk
  - phyClk1 to cpuClk
  - usbClk to cpuClk
    - SETUP -0.450 ns (10)
    - HOLD 0.975 ns (10)
  - wbClk to cpuClk

Inter-Clock Paths - usbClk to cpuClk - SETUP

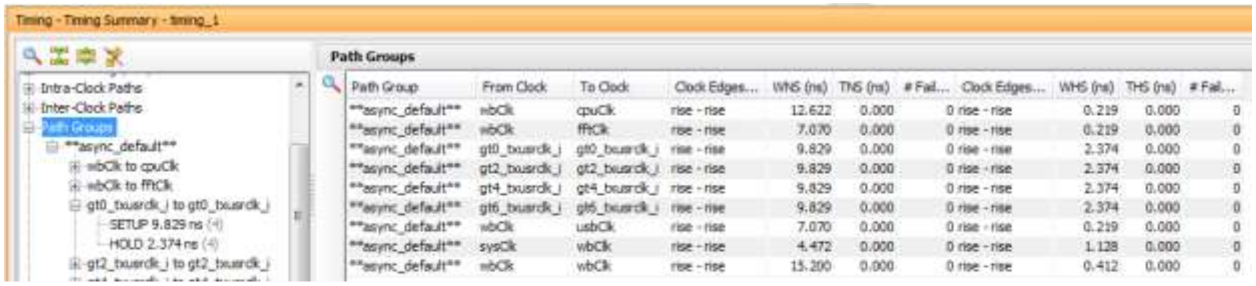
Id	Name	Slack	From	To	Total Delay
1	Path 1	-0.450	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[3]	8.250
2	Path 2	-0.450	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[4]	8.250
3	Path 3	-0.262	usbEngine1/u4/inta_reg/C	cpuEngine/cpu_dbg_dat_o/buf...ance.fifo_ram_reg/DIADI[3]	8.062
4	Path 4	0.000	usbEngine1/u4/inta_reg/C	cpuEngine/cpu_dbg_dat_o/buf...ance.fifo_ram_reg/DIADI[4]	7.800
5	Path 5	0.073	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[1]	7.727
6	Path 6	0.073	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[2]	7.727
7	Path 7	0.073	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[6]	7.727
8	Path 8	0.190	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[0]	7.610
9	Path 9	0.190	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[5]	7.610
10	Path 10	0.246	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[14]	7.554

Timing Summary - timing_1

Figure 18: Timing Summary Report Inter-Clock Paths Details

Path Groups

The Path Groups section shows default path groups and user-defined path groups. The following figure (Timing Summary Report Path Groups) shows an example of the Path Groups summary table which you can access by selecting **Path Groups** in the left pane.



Timing - Timing Summary - timing_1

- Intra-Clock Paths
- Inter-Clock Paths
- Path Groups
  - **async_default**
    - wbClk to cpuClk
    - wbClk to fftClk
    - gt0_txusrclk_i to gt0_txusrclk_i
      - SETUP 9.829 ns (4)
      - HOLD 2.374 ns (4)
    - gt2_txusrclk_i to gt2_txusrclk_i

Path Groups

Path Group	From Clock	To Clock	Clock Edges...	WNS (ns)	TNS (ns)	# Fail...	Clock Edges...	WHS (ns)	THS (ns)	# Fail...
async_default	wbClk	cpuClk	rise - rise	12.622	0.000	0	rise - rise	0.219	0.000	0
async_default	wbClk	fftClk	rise - rise	7.070	0.000	0	rise - rise	0.219	0.000	0
async_default	gt0_txusrclk_i	gt0_txusrclk_i	rise - rise	9.829	0.000	0	rise - rise	2.374	0.000	0
async_default	gt2_txusrclk_i	gt2_txusrclk_i	rise - rise	9.829	0.000	0	rise - rise	2.374	0.000	0
async_default	gt4_txusrclk_i	gt4_txusrclk_i	rise - rise	9.829	0.000	0	rise - rise	2.374	0.000	0
async_default	gt6_txusrclk_i	gt6_txusrclk_i	rise - rise	9.829	0.000	0	rise - rise	2.374	0.000	0
async_default	wbClk	usbClk	rise - rise	7.070	0.000	0	rise - rise	0.219	0.000	0
async_default	sysClk	wbClk	rise - rise	4.472	0.000	0	rise - rise	1.128	0.000	0
async_default	wbClk	wbClk	rise - rise	15.200	0.000	0	rise - rise	0.412	0.000	0

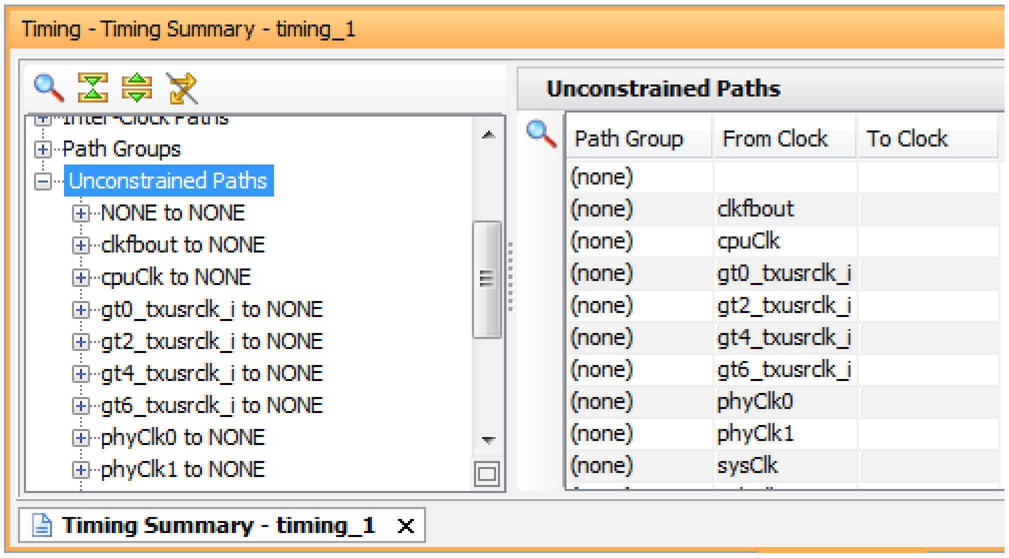
Figure 19: Timing Summary Report Path Groups

Note `**async_default**` is a path group automatically created by the Vivado timing engine. It includes all paths ending with an asynchronous timing check, such as recovery and removal. These two checks are respectively reported under SETUP and HOLD categories, which corresponds to max delay analysis and min delay analysis. Any groups you create using the `group_paths` command appear in this section as well. Any combination of source and destination clocks can be present in a path group.

Unconstrained Paths

The Unconstrained Paths section shows the logical paths that are not timed due to missing timing constraints or because you added timing exceptions to ignore some path. These paths are grouped by source and destination clock pairs. The clock name information shows empty (or NONE) when no clock can be associated with the path startpoint or endpoint.

The following figure (Timing Summary Report Unconstrained Paths) shows an example of the Unconstrained Groups section summary.


Timing - Timing Summary - timing_1

- Inter-Clock Paths
- Path Groups
- Unconstrained Paths
  - NONE to NONE
  - clkfbout to NONE
  - cpuClk to NONE
  - gt0_txusrclk_i to NONE
  - gt2_txusrclk_i to NONE
  - gt4_txusrclk_i to NONE
  - gt6_txusrclk_i to NONE
  - phyClk0 to NONE
  - phyClk1 to NONE

Unconstrained Paths

Path Group	From Clock	To Clock
(none)		
(none)	clkfbout	
(none)	cpuClk	
(none)	gt0_txusrclk_i	
(none)	gt2_txusrclk_i	
(none)	gt4_txusrclk_i	
(none)	gt6_txusrclk_i	
(none)	phyClk0	
(none)	phyClk1	
(none)	sysClk	

Timing Summary - timing_1

Figure 20: Timing Summary Report Unconstrained Paths

Reviewing Timing Path Details

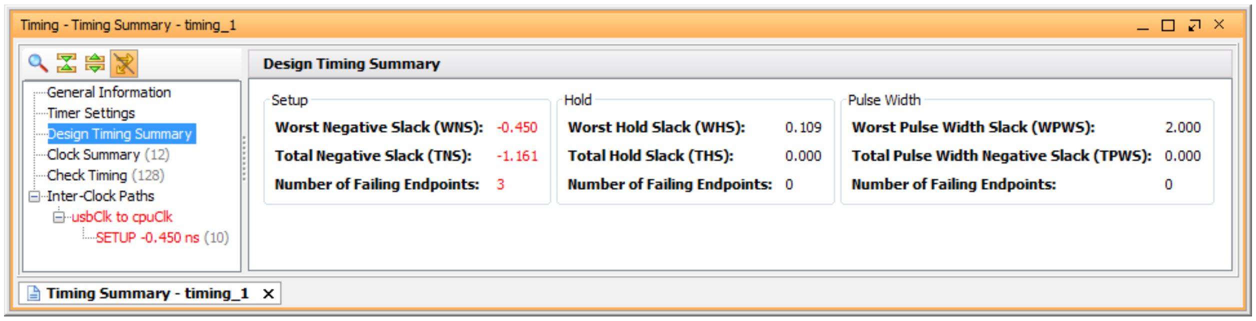
You can expand most of the sections to show paths organized by clock pairs. For each SETUP, HOLD and Pulse Width sub-section, you can see the N-worst reported paths. When selecting any of these paths, you can see more details in the Path Properties window (Report tab). You can also see the same details in a new window when you double click on the path. You can find more information on timing path details in Chapter 4, Timing Analysis.

To access more analysis views for each path, right click the path in the right pane, and select one of the following options from the popup menu:

- Open a Schematic of the path.
- Re-run timing analysis on this same path.
- Highlight the path in the Device and Schematic windows.

Filtering Paths With Violation

The report shows the slack value of failing paths in red. You can focus on these violations by activating the **Show only failing checks** mode (button:). The following figure (Timing Summary Report Violating Paths Filter) shows the Timing Summary window with only violating paths.


Timing - Timing Summary - timing_1

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0.450	Worst Hold Slack (WHS):	0.109	Worst Pulse Width Slack (WPWS):	2.000
Total Negative Slack (TNS):	-1.161	Total Hold Slack (THS):	0.000	Total Pulse Width Negative Slack (TPWS):	0.000
Number of Failing Endpoints:	3	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0

Figure 21: Timing Summary Report Violating Paths Filter

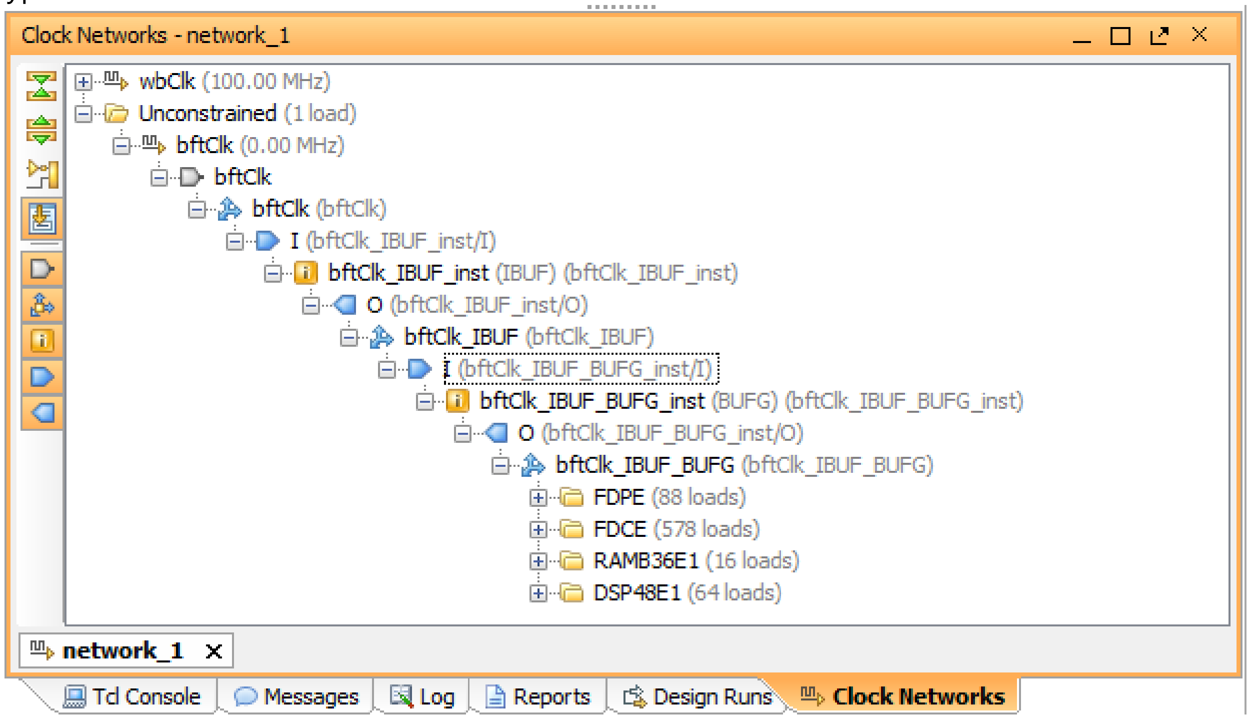
Report Clock Networks

Report Clock Networks provides a tree view of the clocks trees in the design.

- From IO port to load.
- Shows Clock names for clocks defined by the user or automatically by the tool.
- The full tree details only appear in the GUI. The text version of this report only shows the name of the clock roots.
- Used to find BUFG driving BUFG.
- Clocks driving not clock loads .

  There is a folder containing each primary clock and its generated clocks, if any, defined in the design. A separate folder shows each unconstrained clock root.

Each tree shows the clock network from source to endpoint with the endpoints sorted by type.



Clock Networks - network_1

```
wbClk (100.00 MHz)
Unconstrained (1 load)
  bftClk (0.00 MHz)
    bftClk
      bftClk (bftClk)
        I (bftClk_IBUF_inst/I)
          bftClk_IBUF_inst (IBUF) (bftClk_IBUF_inst)
            O (bftClk_IBUF_inst/O)
              bftClk_IBUF (bftClk_IBUF)
                I (bftClk_IBUF_BUFG_inst/I)
                  bftClk_IBUF_BUFG_inst (BUFG) (bftClk_IBUF_BUFG_inst)
                    O (bftClk_IBUF_BUFG_inst/O)
                      bftClk_IBUF_BUFG (bftClk_IBUF_BUFG)
                        FDPE (88 loads)
                        FDCE (578 loads)
                        RAMB36E1 (16 loads)
                        DSP48E1 (64 loads)
```

Figure 22: Clock Networks

Use the Filter Ports, Nets, Instances, and related buttons to reduce the amount of data displayed in the clock tree. Selecting an object in the tree and using the Trace to Source popup command produces a schematic of the clock path.

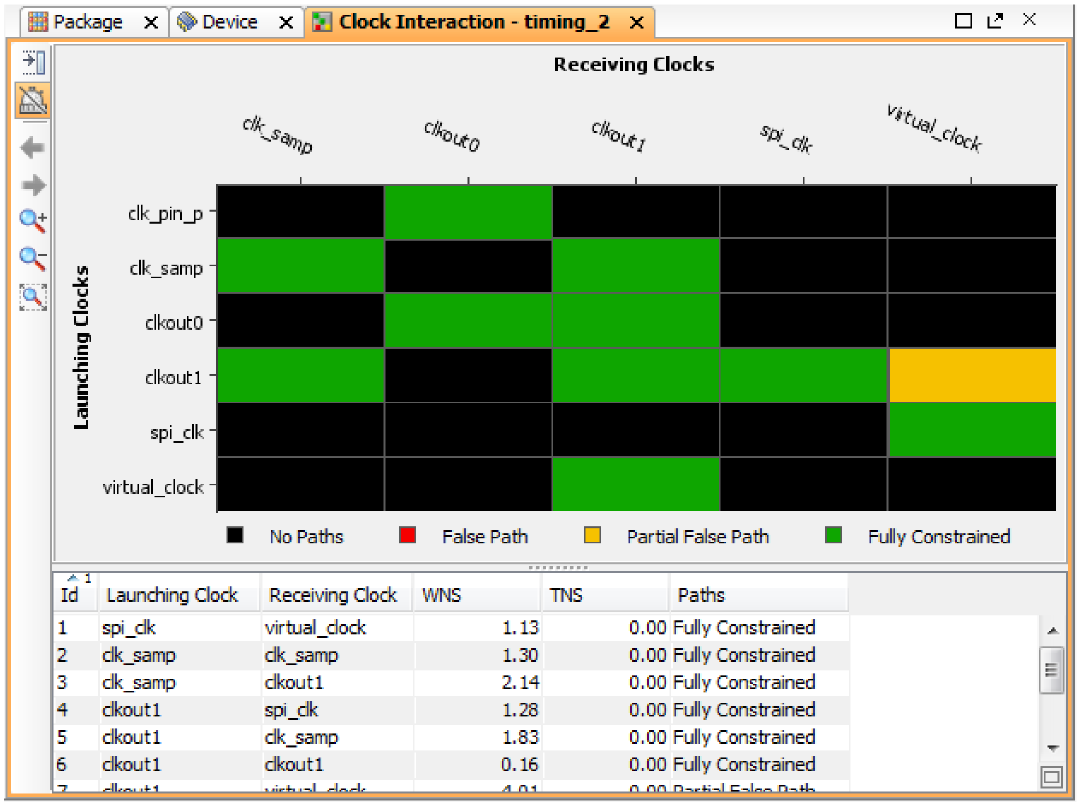
Run the following command to generate the report from the Tcl console and populate the GUI:

```
report_clock_networks -name {network_1}
```

Report Clock Interaction

Report Clock Interaction:

- Analyzes inter-clock timing paths.
- Shows them on a grid.
  - Green – Fully Constrained
  - Black – No paths
  - Red – User False path constraints cover all cross domain paths
  - Yellow – User False Path constraints cover some of the cross domain paths
- Helps to identify cases in which there may be data loss or metastability issues.



Id	Launching Clock	Receiving Clock	WNS	TNS	Paths
1	spi_clk	virtual_clock	1.13	0.00	Fully Constrained
2	clk_samp	clk_samp	1.30	0.00	Fully Constrained
3	clk_samp	clkout1	2.14	0.00	Fully Constrained
4	clkout1	spi_clk	1.28	0.00	Fully Constrained
5	clkout1	clk_samp	1.83	0.00	Fully Constrained
6	clkout1	clkout1	0.16	0.00	Fully Constrained
7	clkout1	virtual_clock	4.01	0.00	Partial False Path

Figure 23: Clock Interaction

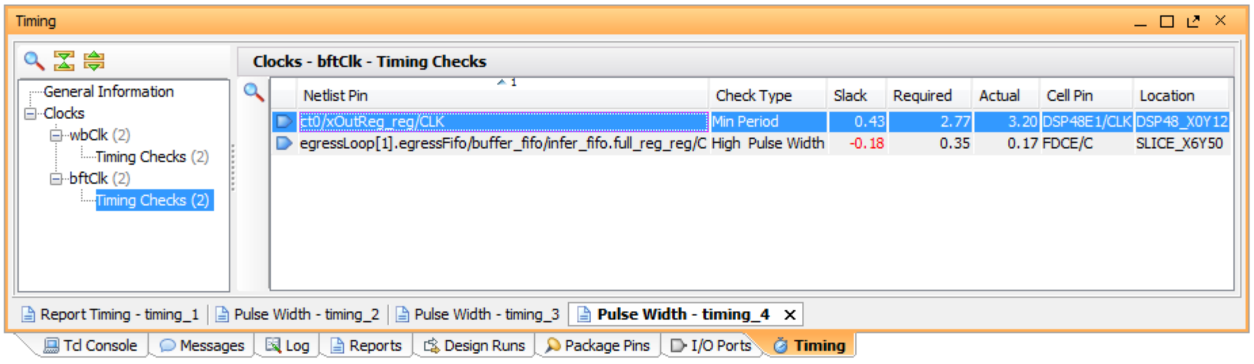
Run the following to create a Clock Interaction report to display in the Vivado IDE via Tcl:

```
report_clock_interaction -delay_type max -significant_digits 2 -name
timing_1
```

Report Pulse Width

The pulse width report checks that the design meets min period, max period, high pulse time, and low pulse time requirements for each instance clock pin. It also checks that the maximum skew requirement is met between two clock pins of a same instance in the implemented design (for example: PCIE clocks).

Note ISE implementation calls this check Component Switching Limits.



Clocks - bftClk - Timing Checks

Netlist Pin	Check Type	Slack	Required	Actual	Cell Pin	Location
ct0/xOutReg_reg/CLK	Min Period	0.43	2.77	3.20	DSP48E1/CLK	DSP48_X0Y12
egressLoop[1].egressFifo/buffer_fifo/infer_fifo.full_reg_reg/C	High Pulse Width	-0.18	0.35	0.17	FDCE/C	SLICE_X6Y50

Figure 24: Report Pulse Width

To generate this report from the Tcl console run:

```
report_pulse_width -name timing_1
```

Per Session Reports

The `vivado.exe` process creates the `vivado.log` and `vivado.jou` in the directory you were in when you started the process. The `vivado.exe` process initializes the LOG and JOU files before the Vivado IDE starts. Use **File> Open Log File** and **File > Open Journal File** to see the LOG and JOU files at any time.

Report Timing

You can use Report Timing to view specific timing paths at any point of the flow after synthesis when you need to investigate timing problems reported by Report Timing Summary or when you want to report the validity and the coverage of particular timing constraints. Report Timing does not cover Pulse Width reports.

You can run Report Timing from various places in the Vivado IDE, assuming a design is already loaded in memory:

- From the menu: **Tools > Timing > Report Timing**
- From the Clock Iteration Report: select a from/to clock pair, right click and select Report Timing to run a report from/to the selected clocks.
- From the Timing Report and Timing Report Summary paths list: select a path , right click and select Report Timing to run a report between the selected path startpoint endpoint.

The equivalent Tcl command is `report_timing`. When setting specific Report Timing options, you can see the equivalent `report_timing` command syntax in the Command field at the bottom of the dialog and in the Tcl Console after execution. The `report_timing` options are listed along with the dialog description in the following section.

Overall, the Report Timing options are identical to the Report Timing Summary options , plus a few additional options.

Targets Tab

Report Timing provides several filtering options that you must use in order to report a particular path or group of paths. The filters are based on the structure of a timing path:

- **`Start Points (From)`**: list of startpoints, such as sequential cell clock pins, sequential cells, input ports, bidirectional ports or source clock. If you combine several startpoints in a list, the reported paths will start from any of these netlist objects. The "Rise/Fall" filter selects a particular source clock edge.

  Equivalent Tcl option: -from, -rise_from, -fall_from.

- **`Through Point Groups (Through)`**: list of pins, ports, combinational cells or nets. You can combine several netlist objects in one list if you want to filter on paths that traverse any of them. You can also specify several "Through" options if you need to refine your filters and report paths that traverse all through points. The "Rise/Fall" filter applies to the data edge.

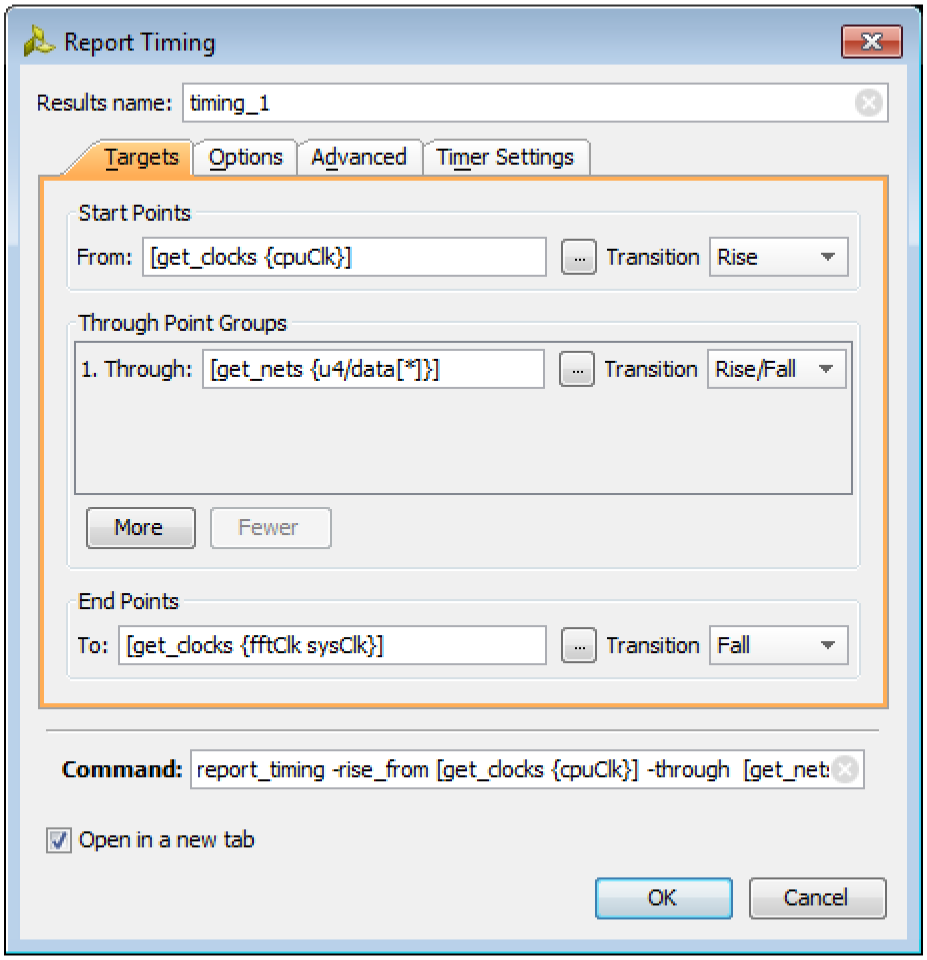
  RECOMMENDED Use the default value (Rise/Fall).

  Equivalent Tcl option: -through, -rise_through, -fall_through.

- **End Points (To)**: list of endpoints, such as input data pins of sequential cells, sequential cells, output ports, bidirectional ports or destination clock. If you combine several endpoints in a list, the reported paths will end with any of these netlist objects. In general, the "Rise/Fall" option selects a particular data edge. But if you specified a destination clock, it selects a particular clock edge.

  Equivalent Tcl option: *-to, -rise_to, -fall_to*.

The following figure (Report Timing Dialog Targets Tab) shows the Targets tab in the Report Timing dialog box, where the path filters have been set to returns path from the rising clock edge of *cpuClk*, through any if the *u4/data[*]* nets and to the falling edge of *fftClk* or *sysClk*.



Results name: timing_1

From: [get_clocks {cpuClk}] Transition Rise

1. Through: [get_nets {u4/data[*]}] Transition Rise/Fall

To: [get_clocks {fftClk sysClk}] Transition Fall

Command: report_timing -rise_from [get_clocks {cpuClk}] -through [get_net

☑ Open in a new tab

Figure 25: Report Timing Dialog Targets Tab

Options Tab

The following options are available in the Options tab:

- **Reports**
  - **Path delay type**: see Report Timing Summary, page 25.
  - **Do not report unconstrained paths**: by default, Report Timing reports paths that are not constrained if no path that matches the filters (from/through/to), is constrained. Check this box if you do not want to see unconstrained paths in your report.

    Equivalent Tcl option: - *no_report_unconstrained*.

- **Path Limits**
  - **Number of paths per group**: Report Timing Summary, page 25.
  - **Number of paths per endpoint**: Report Timing Summary, page 25.
  - **Limit paths to group**: filters on one or more timing path groups. Each clock is associated to a group. The Vivado IDE timing engine also creates a few default groups such as ***async_default*** which groups all the paths ending with a recovery or removal timing check.

    Equivalent Tcl option: -*group*.
- **Path Display**
  - **Display paths with slack greater than**: filters the reported paths based on their slack value.

    Equivalent Tcl option: -*slack_greater_than*
  - **Display paths with slack less than**: see Report Timing Summary, page 27.
  - **Number of significant digits**: see Report Timing Summary, page 27
  - **Sort paths by**: choose to display the reported paths by group (default) or by slack. When sorted by group, the N worst paths for each group and for each type of analysis (-delay_type min/max/min_max) are reported. The groups are sorted based on their individual worst path so that the group with the worst violation shows at the top of the list. When sorted by slack, the N worst paths per type of analysis are reported (all groups combined) and sorted by increasing slack.

    Equivalent Tcl option: -*sort_by*

Advanced Tab

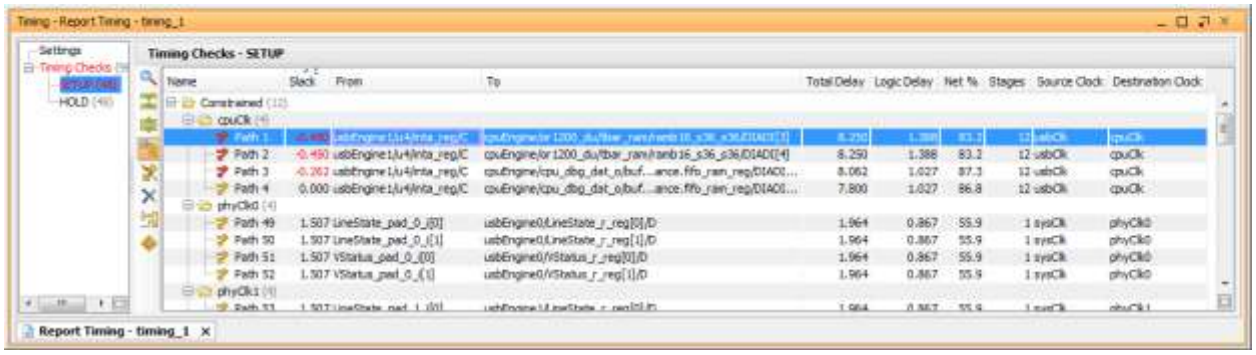
Same options as Report Timing Summary, page 28.

Timer Settings Tab

Same options as Report Timing Summary, page 29.

Reviewing Timing Path Details

After clicking **OK** to run the report command, a new window opens and you can start reviewing its content. You can see the N-worst paths reported for each type of selected analysis (min/max/min_max). The following figure (Report Timing Window) shows the Report Timing window where both min and max analysis (SETUP and HOLD) were selected, and N=4.


Timing - Report Timing - timing_1

Settings
Timing Checks
SETUP (48)
HOLD (48)

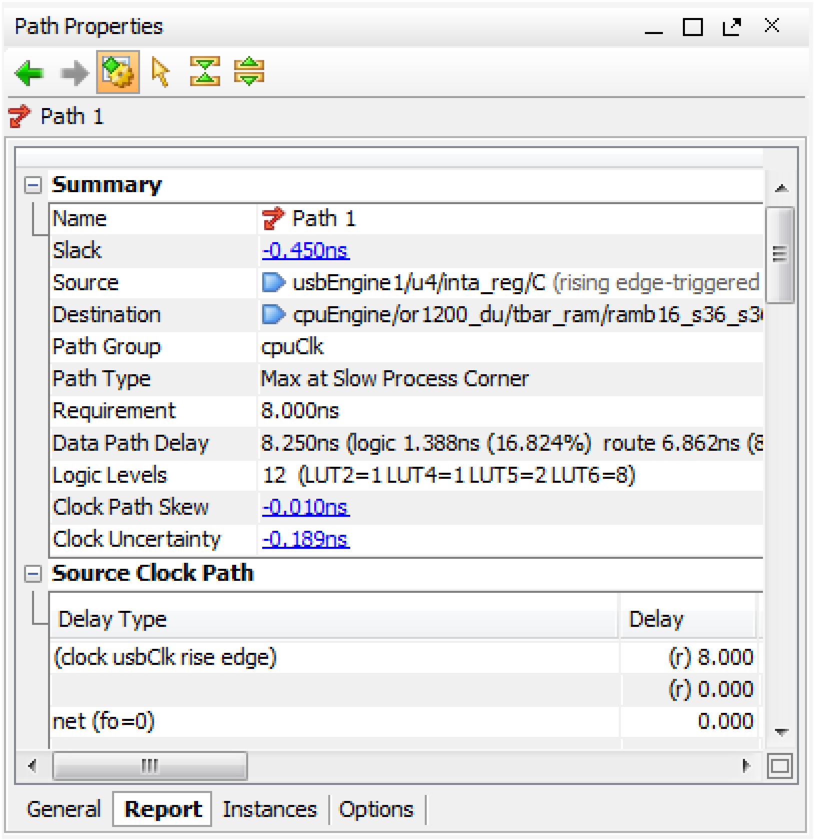
Timing Checks - SETUP

Name	Slack	From	To	Total Delay	Logic Delay	Net %	Stages	Source Clock	Destination Clock
Constrained (12)									
cpuClk (4)									
Path 1	-0.450	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[3]	8.250	1.388	83.2	12	usbClk	cpuClk
Path 2	-0.450	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s36/DIADI[4]	8.250	1.388	83.2	12	usbClk	cpuClk
Path 3	-0.262	usbEngine1/u4/inta_reg/C	cpuEngine/cpu_dbg_dat_o/buf....ance.fifo_ram_reg/DIADI...	8.062	1.027	87.3	12	usbClk	cpuClk
Path 4	0.000	usbEngine1/u4/inta_reg/C	cpuEngine/cpu_dbg_dat_o/buf....ance.fifo_ram_reg/DIADI...	7.800	1.027	86.8	12	usbClk	cpuClk
phyClk0 (4)									
Path 49	1.507	LineState_pad_0_i[0]	usbEngine0/LineState_r_reg[0]/D	1.964	0.867	55.9	1	sysClk	phyClk0
Path 50	1.507	LineState_pad_0_i[1]	usbEngine0/LineState_r_reg[1]/D	1.964	0.867	55.9	1	sysClk	phyClk0
Path 51	1.507	VStatus_pad_0_i[0]	usbEngine0/VStatus_r_reg[0]/D	1.964	0.867	55.9	1	sysClk	phyClk0
Path 52	1.507	VStatus_pad_0_i[1]	usbEngine0/VStatus_r_reg[1]/D	1.964	0.867	55.9	1	sysClk	phyClk0
phyClk1 (4)									
Path 53	1.507	LineState_pad_1_i[0]	usbEngine1/LineState_r_reg[0]/D	1.964	0.867	55.9	1	sysClk	phyClk1

Report Timing - timing_1

Figure 26: Report Timing Window

When selecting any of these paths, you can see more details in the Path Properties window (Report tab). The following figure (Timing Path Properties) shows a Path Properties window.


Path Properties

Path 1

Summary

Name	Path 1
Slack	-0.450ns
Source	usbEngine1/u4/inta_reg/C (rising edge-triggered
Destination	cpuEngine/or1200_du/tbar_ram/ramb16_s36_s3
Path Group	cpuClk
Path Type	Max at Slow Process Corner
Requirement	8.000ns
Data Path Delay	8.250ns (logic 1.388ns (16.824%) route 6.862ns (8
Logic Levels	12 (LUT2=1 LUT4=1 LUT5=2 LUT6=8)
Clock Path Skew	-0.010ns
Clock Uncertainty	-0.189ns

Source Clock Path

Delay Type	Delay
(clock usbClk rise edge)	(r) 8.000
	(r) 0.000
net (fo=0)	0.000

General | Report | Instances | Options

Figure 27: Timing Path Properties

You can also see the same details in a new window when you double click on the path. You can find more information on timing path details in Chapter 4, Timing Analysis. The figure above (Timing Path Properties) shows a Timing Path report window.

To access more analysis views for each path, you can right click on the path in the right pane, and select one of the following actions from the popup menu:

- View the timing path Schematic.
- Re-run timing analysis on the same startpoint and endpoint of the selected path.
- Highlight the path in the Device and Schematic windows.

Filtering Paths With Violation

The report shows the slack value of failing paths in red. You can focus on these violations by activating the "Show only failing checks" mode (button:).

Chapter 2

Logic Analysis

This chapter contains information on:

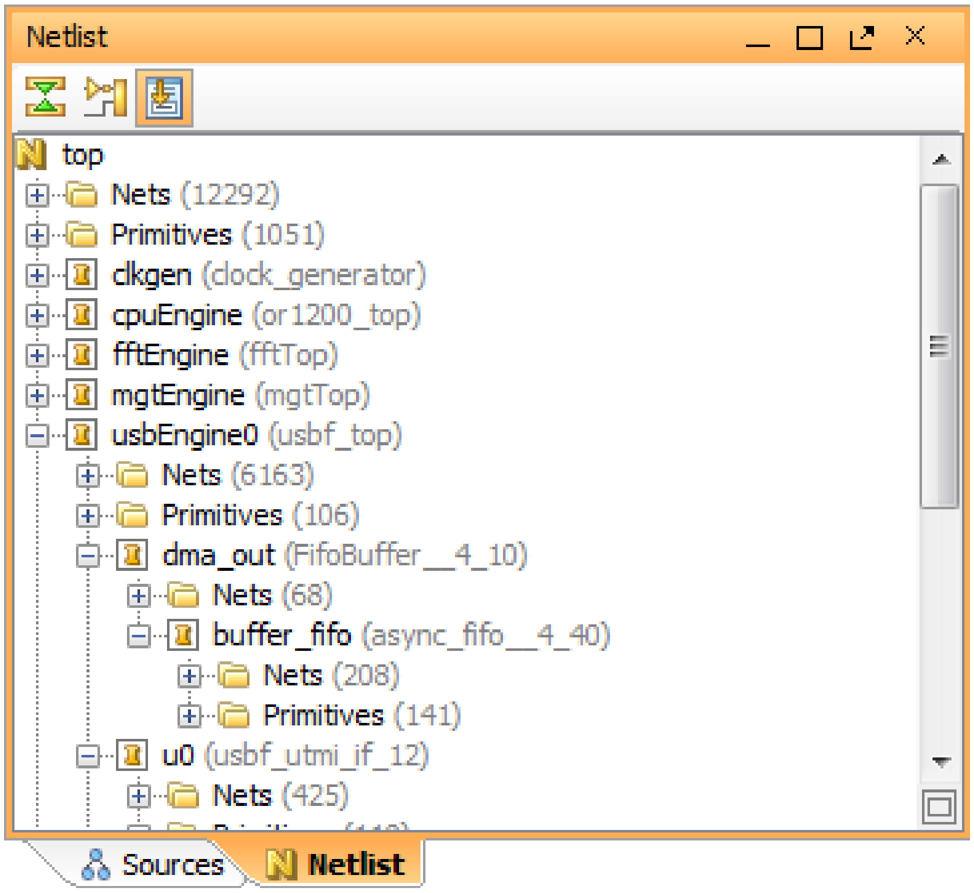
- The elements in a netlist
- The netlist connectivity
- Searching for gates and primitive types
- Design flow in a chip
- Utilization
- Common design issues using DRC

RTL Analysis

For information about RTL Analysis, see the *Vivado Design Suite User Guide: System-Level Design Entry (UG895).*

Netlist Window

The Netlist Window shows the design hierarchy as it is in the netlist, processed by the synthesis tools.



```
top
  Nets (12292)
  Primitives (1051)
  clkgen (clock_generator)
  cpuEngine (or1200_top)
  fftEngine (fftTop)
  mgtEngine (mgtTop)
  usbEngine0 (usbf_top)
    Nets (6163)
    Primitives (106)
    dma_out (FifoBuffer__4_10)
      Nets (68)
      buffer_fifo (async_fifo__4_40)
        Nets (208)
        Primitives (141)
    u0 (usbf_utmi_if_12)
      Nets (425)
```

Figure 28: Netlist Window

Depending on synthesis settings, the netlist hierarchy may be a one hundred percent match for the original RTL, or there may be no hierarchy. Generally, the synthesis tools default to preserving most of the user hierarchy while optimizing the edges, giving a smaller faster netlist. With the Synthesis tool defaults, the netlist hierarchy is recognizable, but the interfaces to the hierarchies may be modified. Some ports and levels of hierarchy may be missing.

Each level of hierarchy shows its hierarchy tree. At each level, the tool shows:

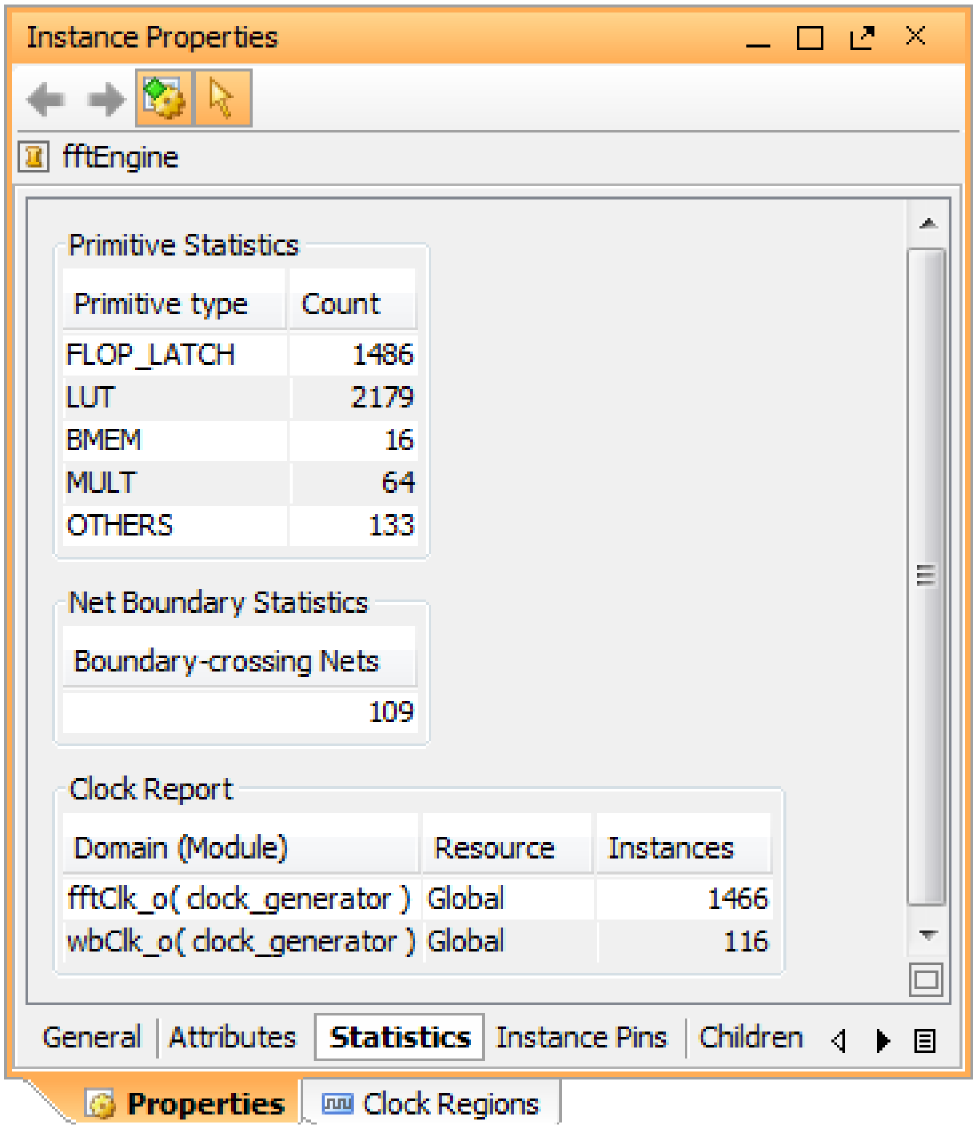
- A nets folder for any nets at that level
- A primitives folder if there are primitives at that level
- Any hierarchies instantiated at that level

Traversing the tree shows the whole branch. The icons next to the cells and primitives display information about the state of the design.

For more information, see Using the Netlist Window in the *Vivado Design Suite User Guide: Using Vivado IDE (UG893)*.

The Properties Window for each level of hierarchy shows utilization statistics including:

- Primitive usage for the whole hierarchical branch, grouped in higher level buckets
- The number of nets crossing the hierarchy boundary
- Clocks used in the hierarchy



fftEngine

Primitive Statistics

Primitive type	Count
FLOP_LATCH	1486
LUT	2179
BMEM	16
MULT	64
OTHERS	133

Net Boundary Statistics

Boundary-crossing Nets
109

Clock Report

Domain (Module)	Resource	Instances
fftClk_o(clock_generator)	Global	1466
wbClk_o(clock_generator)	Global	116

Figure 29: Instance Properties Window

If you floorplan the design, similar properties are displayed for the Pblock.

Hierarchy Window

Explore the hierarchy to understand resource usage. To open the Hierarchy Window, select **Tools > Show Hierarchy**.

The Hierarchy Window displays the hierarchy tree for the netlist. Each horizontal row displays a level of hierarchy inside the netlist. As you move down the rows, you move into deeper netlist hierarchy. Across the row, each level of hierarchy is sized relative to the other hierarchy at that level.

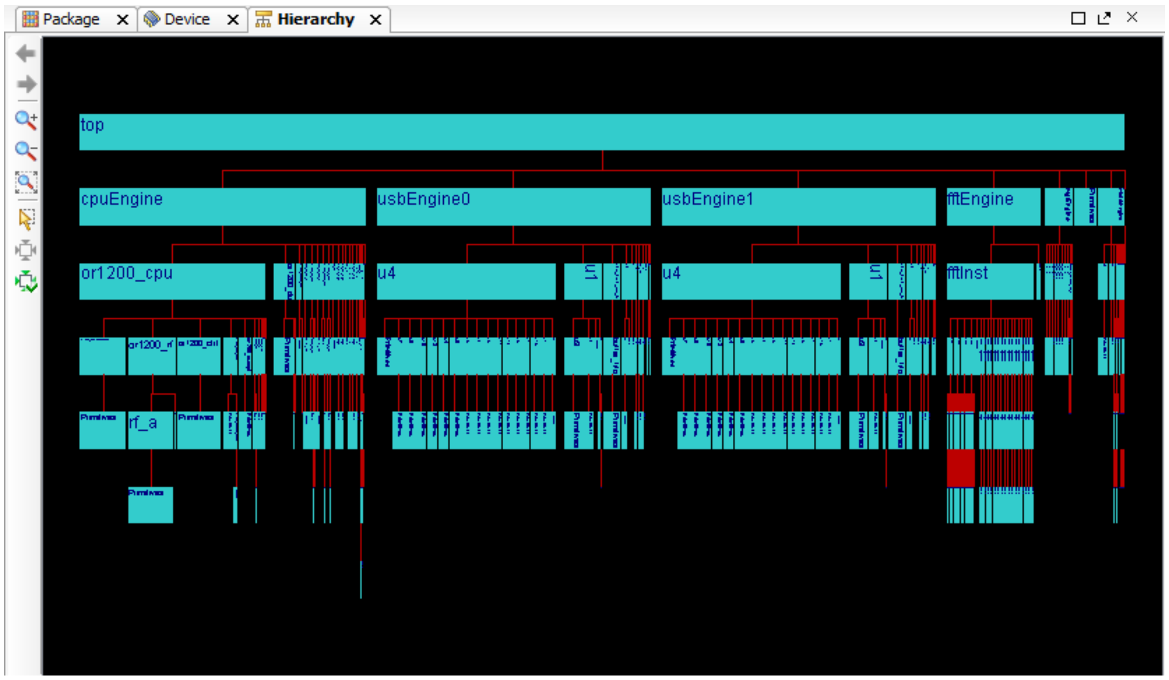


Figure 30: Hierarchy Window

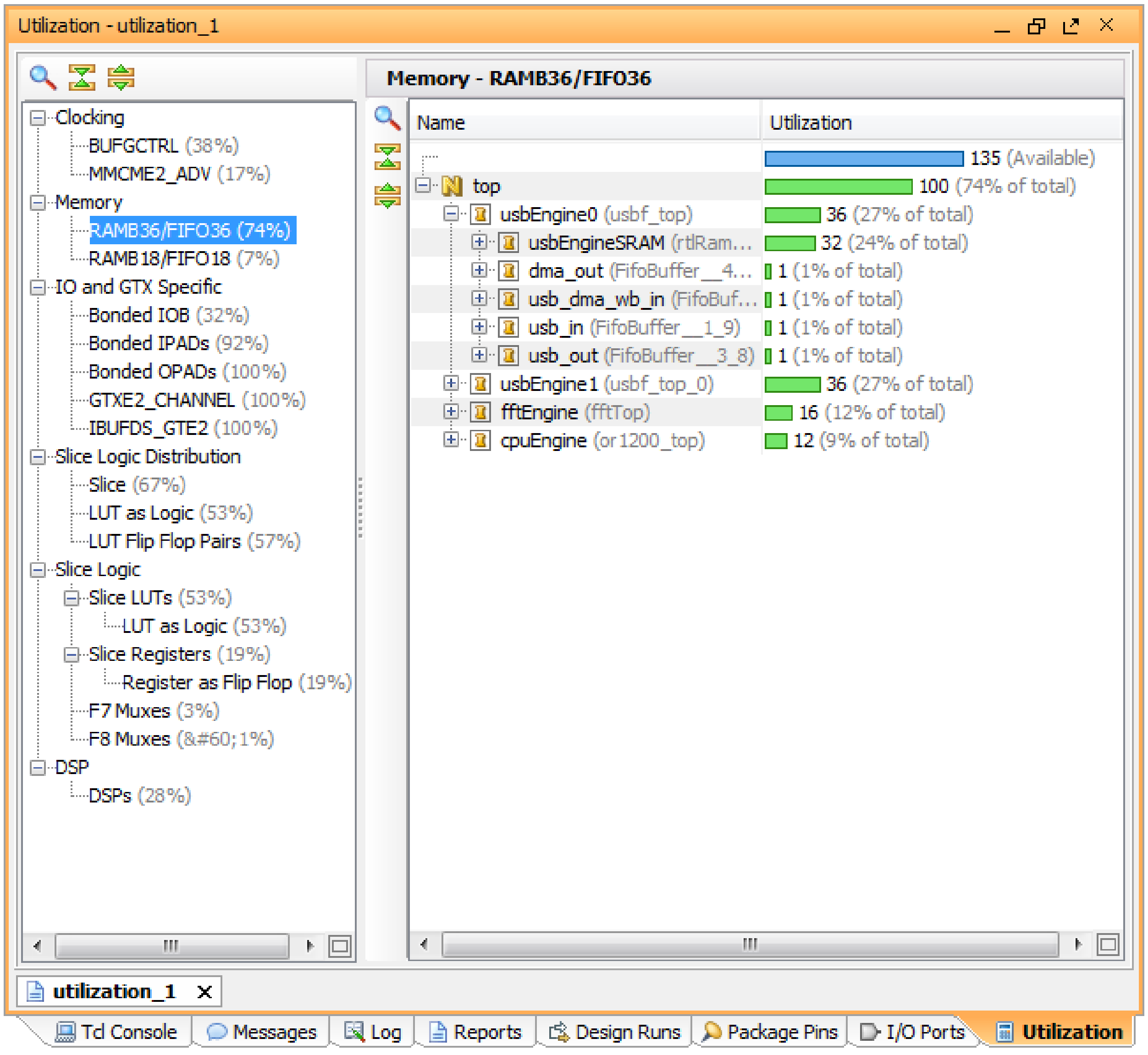
The figure above shows that `cpuEngine`, `usbEngine0`, and `usbEngine1`:

- Have most of the logic in the design.
- All use about the same number of resources.

The Utilization Report:

- Breaks apart the design based on resource type.
- Displays each resource type independently with consumption per level of hierarchy.

To view the Utilization Report, select **Tools > Report Utilization**.



Utilization - utilization_1

```
Clocking
    BUFGCTRL (38%)
    MMCME2_ADV (17%)
Memory
    RAMB36/FIFO36 (74%)
    RAMB18/FIFO18 (7%)
IO and GTX Specific
    Bonded IOB (32%)
    Bonded IPADs (92%)
    Bonded OPADs (100%)
    GTXE2_CHANNEL (100%)
    IBUFDS_GTE2 (100%)
Slice Logic Distribution
    Slice (67%)
    LUT as Logic (53%)
    LUT Flip Flop Pairs (57%)
Slice Logic
    Slice LUTs (53%)
        LUT as Logic (53%)
    Slice Registers (19%)
        Register as Flip Flop (19%)
    F7 Muxes (3%)
    F8 Muxes (&#60;1%)
DSP
    DSPs (28%)
```

Memory - RAMB36/FIFO36

Name	Utilization
	135 (Available)
top	100 (74% of total)
usbEngine0 (usbf_top)	36 (27% of total)
usbEngineSRAM (rtlRam…	32 (24% of total)
dma_out (FifoBuffer__4…	1 (1% of total)
usb_dma_wb_in (FifoBuf…	1 (1% of total)
usb_in (FifoBuffer__1_9)	1 (1% of total)
usb_out (FifoBuffer__3_8)	1 (1% of total)
usbEngine1 (usbf_top_0)	36 (27% of total)
fftEngine (fftTop)	16 (12% of total)
cpuEngine (or1200_top)	12 (9% of total)

Figure 31: Utilization Report

In this design, the two `usbEngine` blocks are the two biggest consumers of the RAMB36 and FIFO36 blocks. Click the **+** icon to view the consumption at sub-hierarchies.

Viewing the Schematic

The schematic is a graphical representation of the netlist. View the schematic to:

- View a graphical representation for the netlist.
- Review the gates, hierarchies, and connectivity.
- Trace and expand cones of logic.

- Analyze the design.
- Better understand what is happening inside the design.

At the RTL level in Elaborated Design, you see how the tool has interpreted your code. In Synthesize Design and Implemented Design, you see the gates generated by the synthesis tool.

To open the schematic, select **Tools > Schematic**. If nothing is selected, the gates, hierarchy, and connectivity appear at the top level of the design.

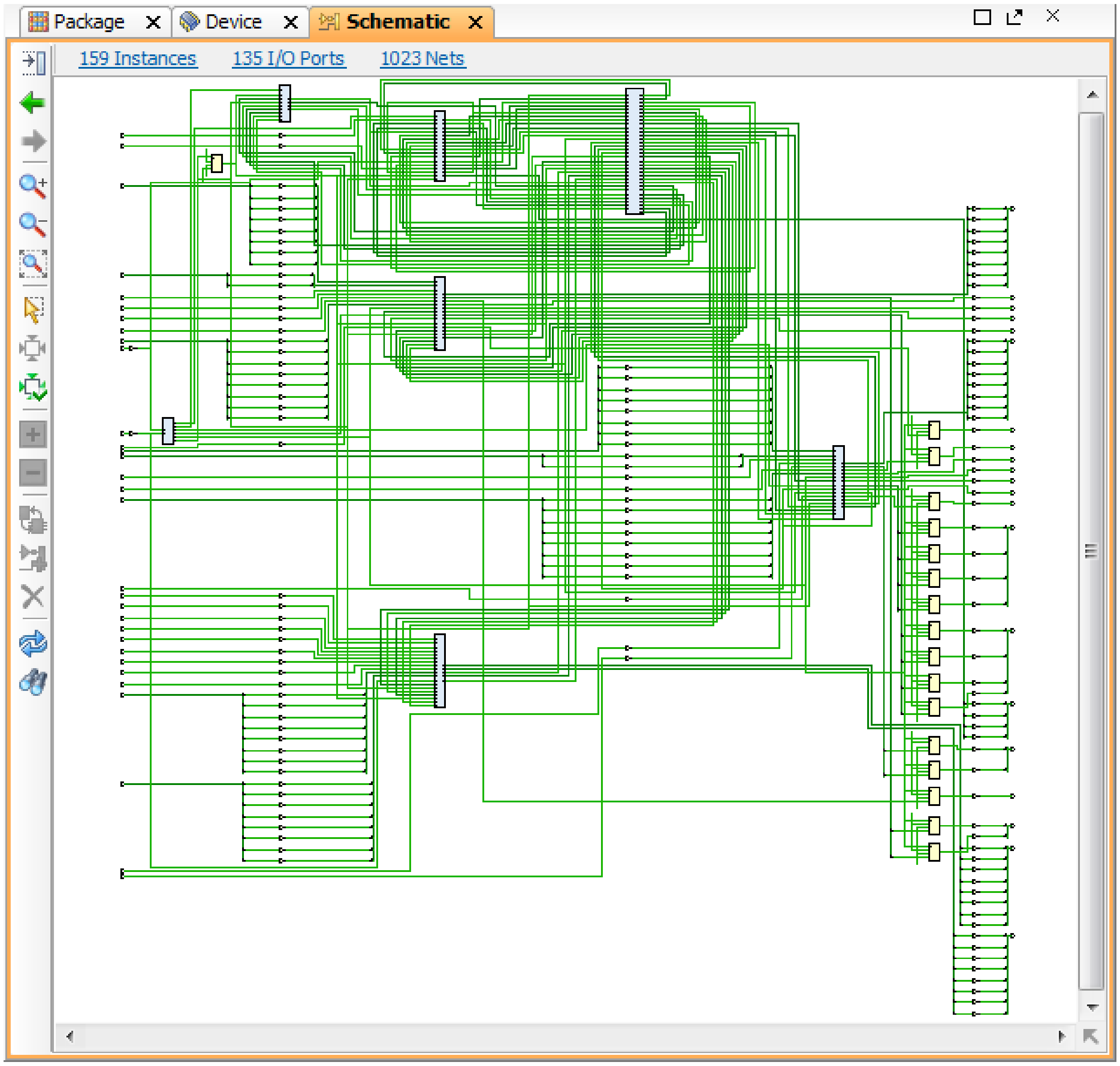
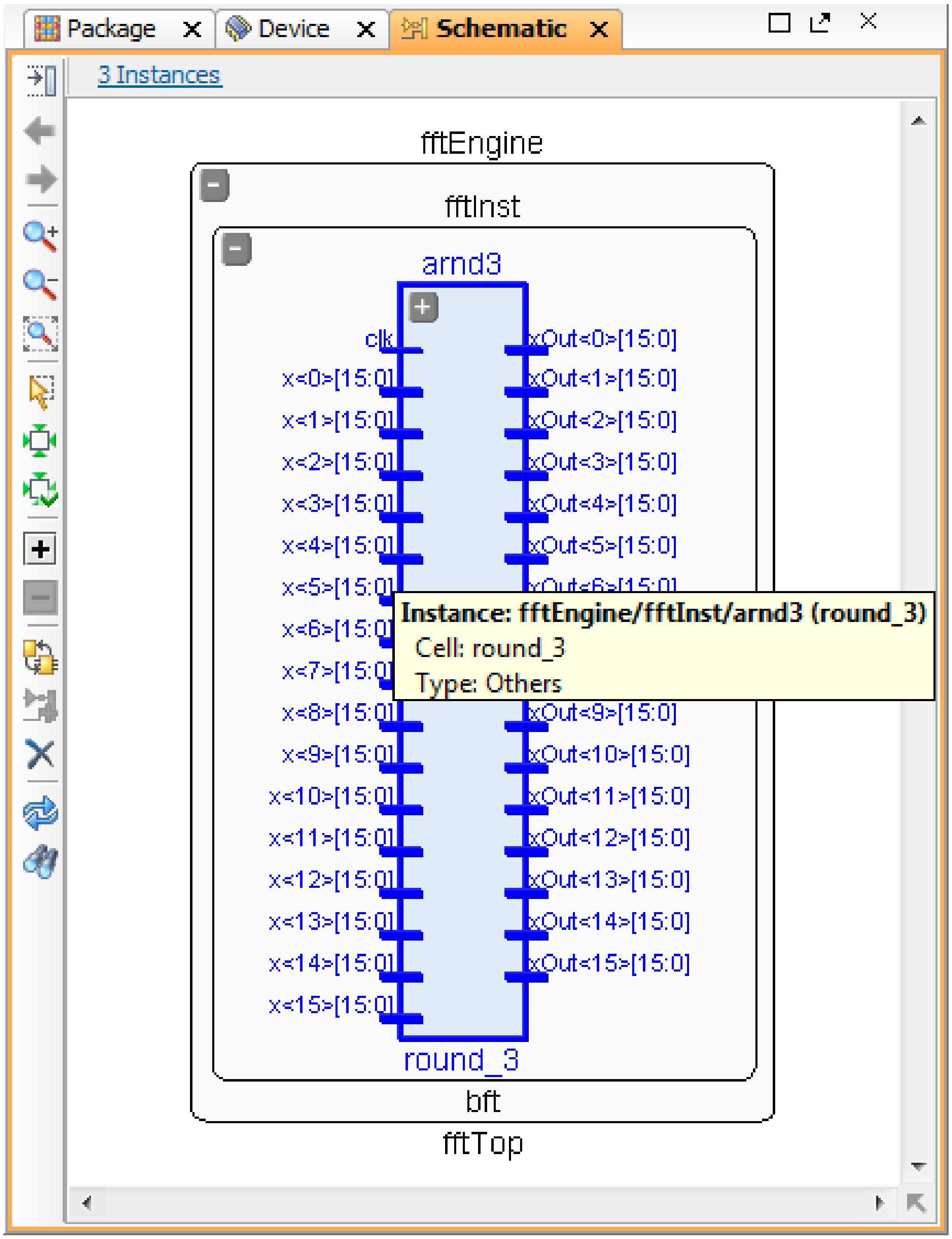


Figure 32: Top Level Schematic

For information about zooming and moving around the schematic, see the *Vivado Design Suite User Guide: Using Vivado IDE (UG893)*.

TIP The schematic is simpler if you only use a single level of hierarchy. The schematic populates with the selected element emphasized (blue). The ports for the single hierarchy display.



Instance: fftEngine/fftInst/arnd3 (round_3)
Cell: round_3
Type: Others

Figure 33: Schematic with Single Hierarchy Selected

You can trace the schematic in multiple ways:

- Click the **+** in the upper right to display the gates in the hierarchy.
- Double click a port or element to expand it.
- Use the schematic popup.

  For more information, see Using the Schematic Window in the *Vivado Design Suite User Guide: Using Vivado IDE (UG893)*.
- Use the <- -> arrows to switch between the previous and next schematic views.
- Select **Expand All** to see more logic and connectivity.
- Select **Collapse All** to simplify the schematic.

After implementation, the schematic is the easiest way to visualize the gates in a timing path. Select the path, then open the schematic with the gates and nets from that path.

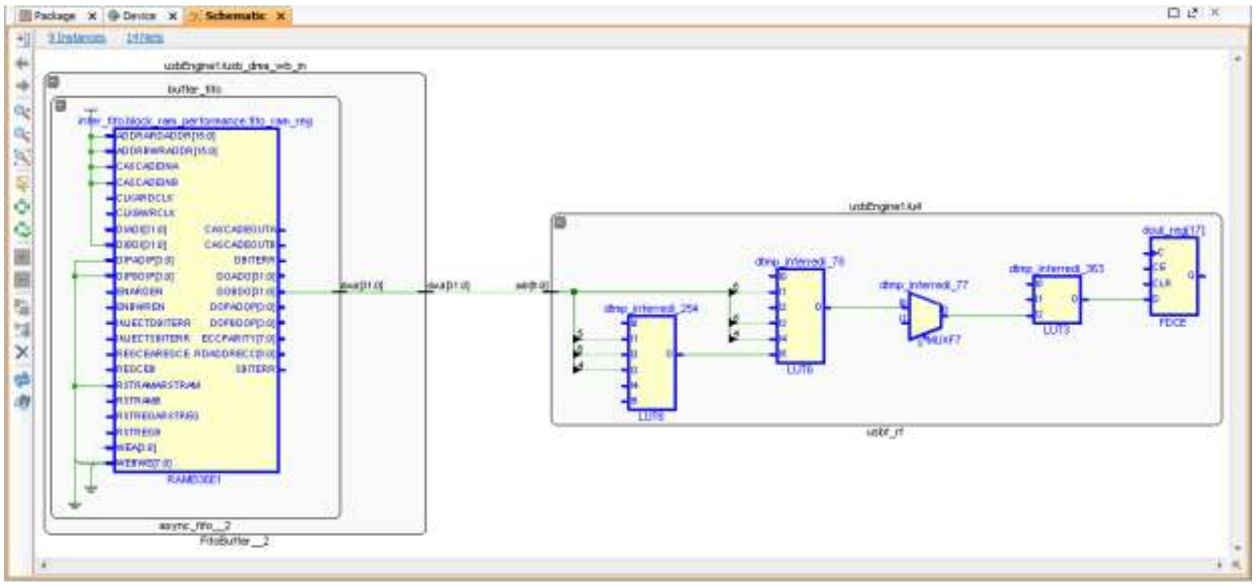


Figure 34: Schematic With Timing Path

To easily identify the relevant levels of hierarchy in the schematic, choose **Select Primitive Parents** from the popup menu.

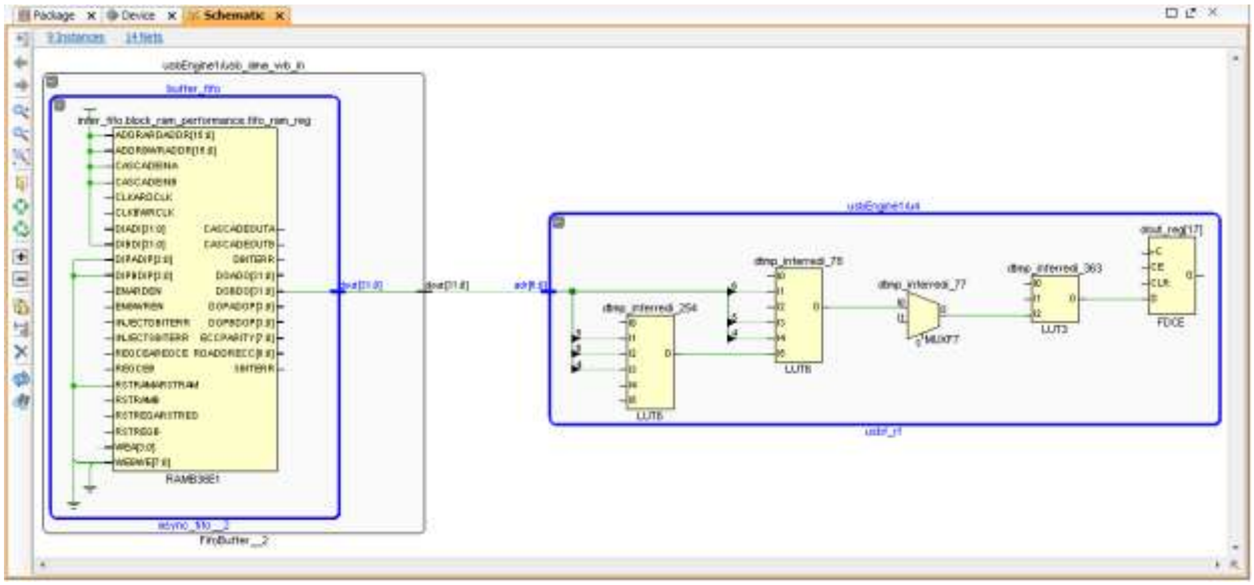


Figure 35: Timing Path with Select Primitive Parents

As you review the schematic, select the **Highlight** and **Mark** commands to track gates of interest. Color coding primitives (using either a mark or a highlight) makes it easier to track which logic was in the original path, and which logic was added.

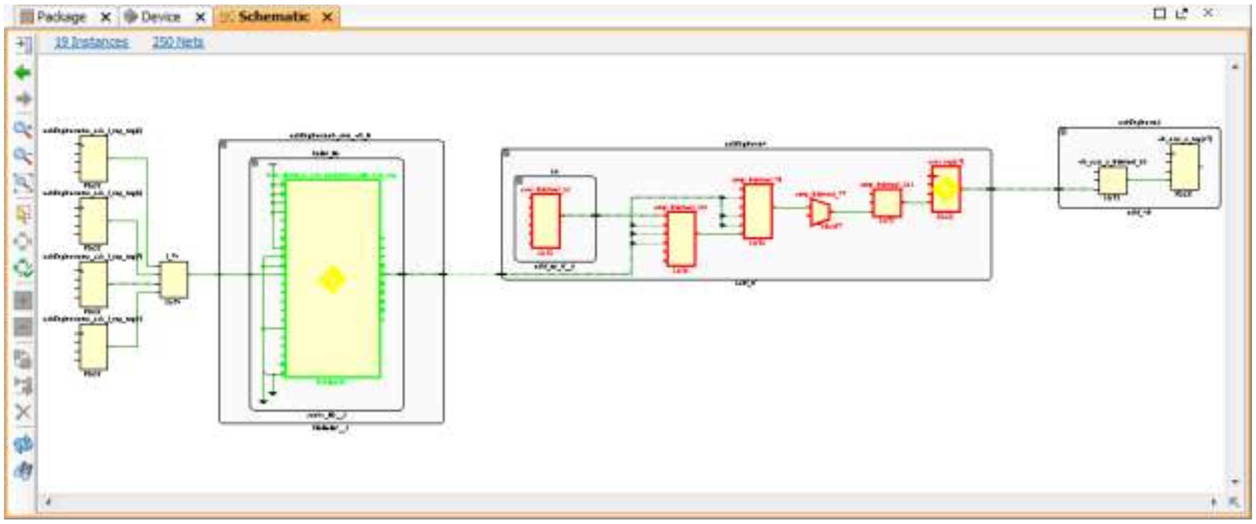


Figure 36: Schematic With Timing Path Marked

Searching for Objects Using Find

The Vivado™ IDE includes powerful find and search capabilities using **Edit > Find**.

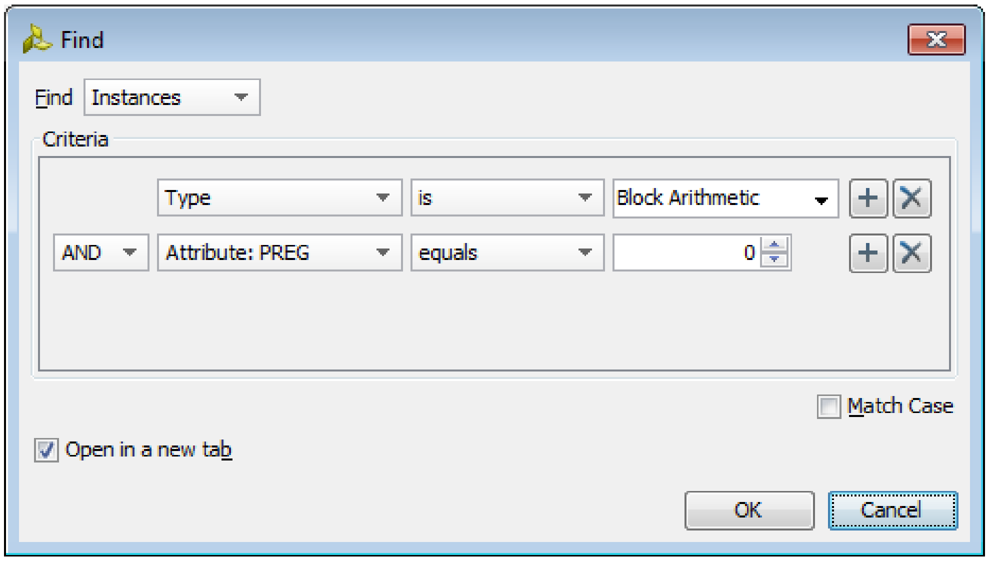


Figure 37: Edit Find Dialog Box

Find Criteria

Use **Edit > Find** to search the netlist for:

- Instances
- Nets
- I/O Ports
- Instance Pins
- Pblocks
- RPMs

Find Sub Criteria

There are multiple sub criteria for each of the Find criteria. For example, *instance* has the following sub criteria:

- Type
- Cell types
- Black boxes
- Primitives, including
  - I/O Buffer
  - Block Arithmetic
  - Block Memory
  - LUT
- Name
- Status
- Parent Pblock
- Module
- Primitive count
- Attribute

Review the other Find criteria for their sub fields.

Device-Specific Find Criteria

Device-specific Find criteria are:

- Arcs
- Nodes
- BEL Pins
- BELs
- Site Pins
- Sites
- Tiles
- I/O Banks
- Clock Regions

Find Examples

Select **Edit > Find** to find, for example:

- All unplaced IOs
- Only the tool-placed Global Clocks
- All nets with a fanout over 10,000
- All DSPs using the PREG embedded register

Complex Finds

To run a complex find:

1. Set the first search criterion.
2. Click **+**.
3. Add additional criteria.
4. Join the additional criteria with logical operators (and, or).

TCL Finds

Use the TCL **Find** commands when running from a script or in the TCL console.

For more information, see the *Vivado Design Suite User Guide: Using the Tcl Scripting Capabilities (UG894*).

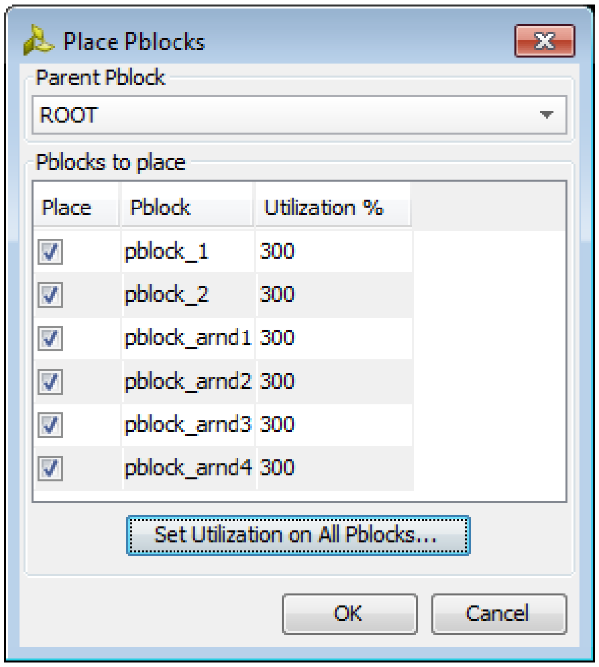
Design Data Flow – Top Level Floorplan

When you integrate RTL into a design, it helps to visualize the design inside the device. Graphically seeing how the blocks interconnect between themselves and the IO pinout after synthesis helps you to understand your design.

To view the interconnect, generate a top level floorplan using Pblocks on upper levels of hierarchy. To break apart the top level RTL into Pblocks, select **Tools > Floorplanning > Auto Create Pblocks**.

To place the blocks in the device, select **Tools > Floorplanning > Place Pblocks**. The tool sizes the Pblocks based on the slice count and target utilization.

Pblocks can be more than one hundred percent full during analysis, but not during implementation. Overfilling the Pblock makes them smaller on the device. This is a useful technique for understanding a full device.



Parent Pblock: ROOT

Place	Pblock	Utilization %
✓	pblock_1	300
✓	pblock_2	300
✓	pblock_arnd1	300
✓	pblock_arnd2	300
✓	pblock_arnd3	300
✓	pblock_arnd4	300

Figure 38: Place Pblocks Utilization

The top level floorplan shows which blocks communicate with IOs (Green Lines). Nets shared between two Pblocks are bundled together. The bundles change size and color based on the number of shared nets. Two top level floorplans are shown below.

Data Path Top Level Floorplan

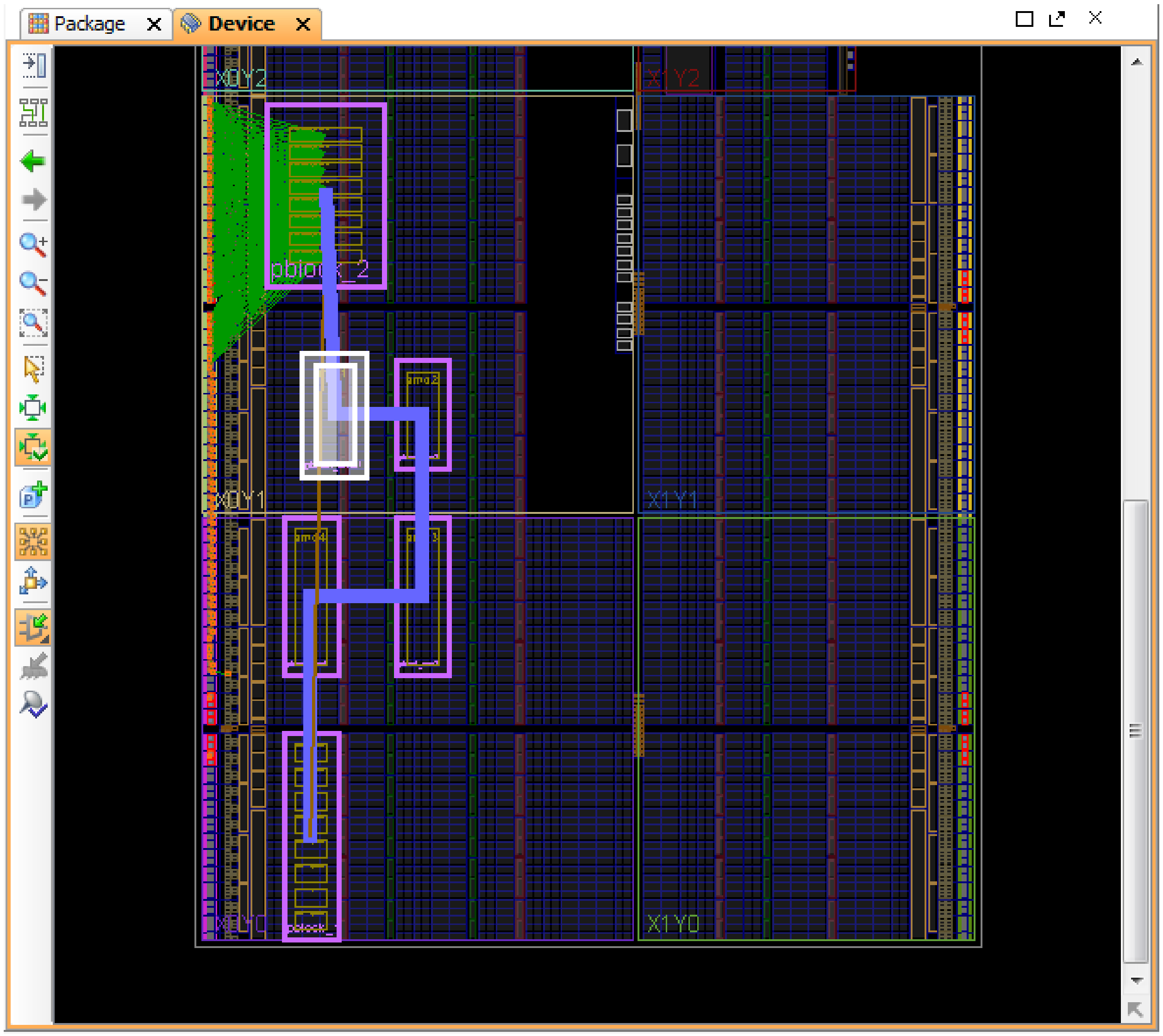


Figure 39: Data Path Top Level Floorplan

The Data Path Top Level Floorplan shows a data path. Each block communicates only to a drive block and a load block. The green lines show well-placed IOs that communicate with a single block. This design is easy to floorplan.

Control Path Floorplan

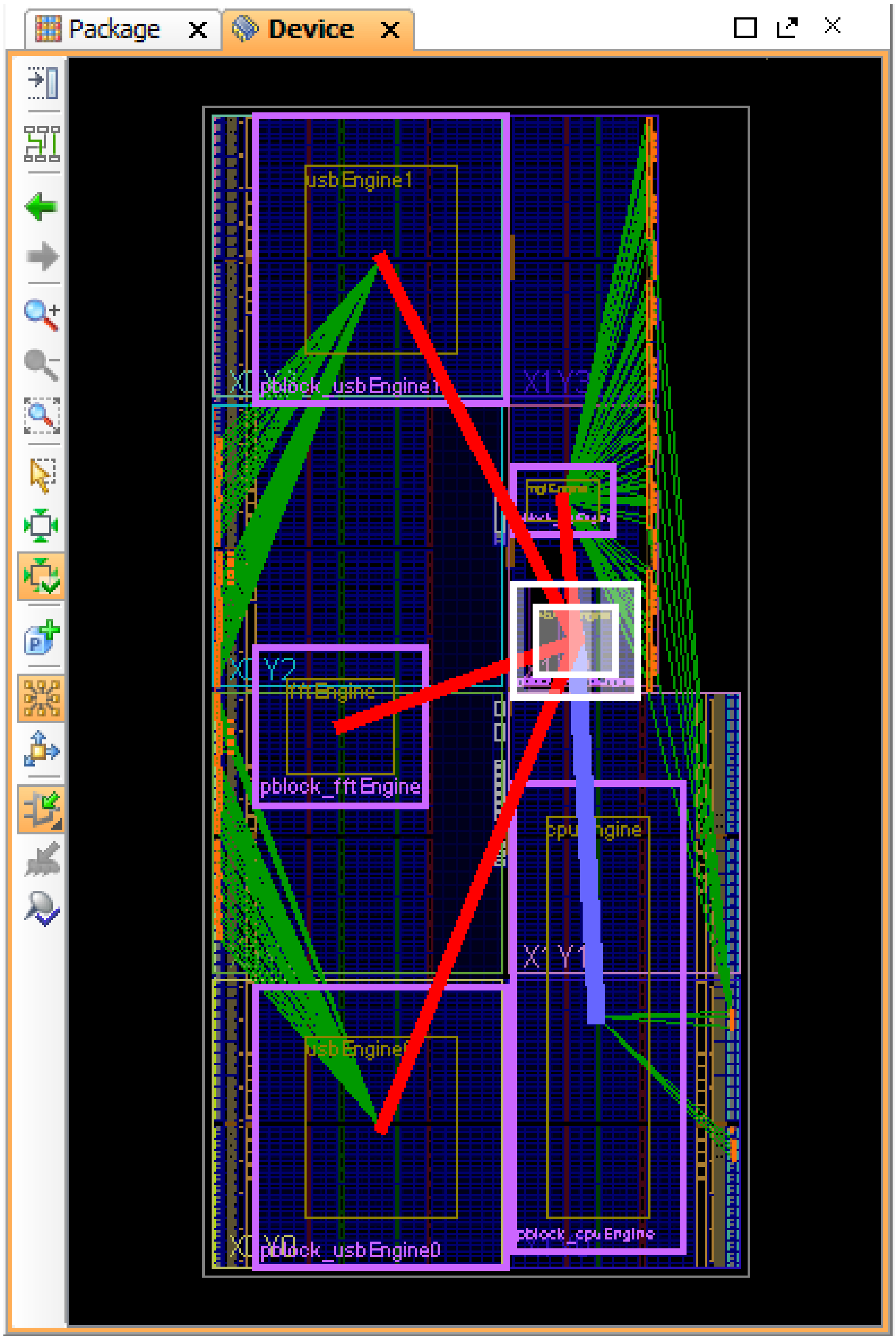


Figure 40: Control Path Floorplan

The Control Path Floorplan displays a design in which all the blocks communicate with a central block. The largest connection is between the central block and the block in the bottom right. The central block must spread out around the design to communicate with all the other loads. This design is hard to floorplan.

Reviewing the Floorplan

Consider device resources when reviewing the floorplan. The Pblock sizing does not take into account specialized device resources such as:

- Block RAM
- DSP48s
- MGTs
- ClockBuffers

Review the blocks with the floorplan and utilization in mind.

Analyzing Device Utilization Statistics

A common cause of implementation issues comes from not considering the logic and device layout implied by the pinout. Slice logic is uniform in most devices. However specialized resources such as the following impact logic placement:

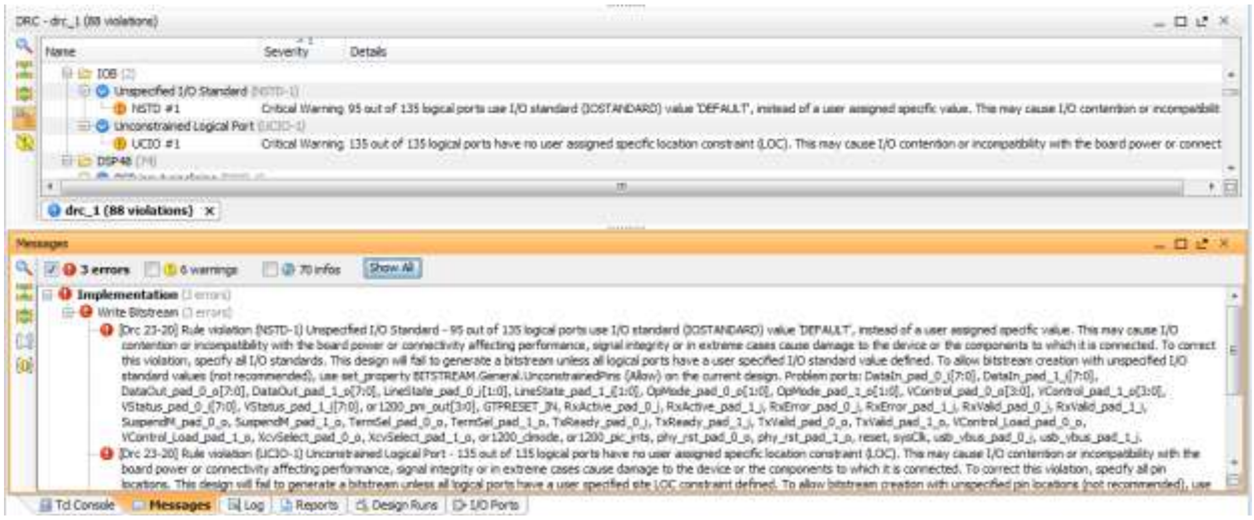
- IO
- High Performance Banks
- High Range Banks
- MGT
- DSP48
- Block RAM
- MMCM
- BUFG
- BUFR

Blocks that are large consumers of a certain specialized resource may have to spread around the device. Take this into account when designing the interface with the rest of the design. Use a combination of the following to find block resources:

- `report_utilization`
- netlist properties
- Pblock properties

DRCs

DRCs check the design and report on common issues. Run DRCs using the `report_drc` command. During implementation, the tools also run DRCS. The DRCs become more complete and comprehensive with placement and routing.



```
DRC - drc_1 (88 violations)
Name                                   Severity   Details
IOB (2)
  Unspecified I/O Standard (NSTD-1)
    NSTD #1                            Critical Warning  95 out of 135 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value. This may cause I/O contention or incompatibilit
  Unconstrained Logical Port (UCIO-1)
    UCIO #1                            Critical Warning  135 out of 135 logical ports have no user assigned specific location constraint (LOC). This may cause I/O contention or incompatibility with the board power or connect
DSP48 (78)
drc_1 (88 violations)
```

```
Messages
3 errors   6 warnings   70 infos   Show All
Implementation (3 errors)
  Write Bitstream (3 errors)
    [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard - 95 out of 135 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value. This may cause I/O
    contention or incompatibility with the board power or connectivity affecting performance, signal integrity or in extreme cases cause damage to the device or the components to which it is connected. To correct
    this violation, specify all I/O standards. This design will fail to generate a bitstream unless all logical ports have a user specified I/O standard value defined. To allow bitstream creation with unspecified I/O
    standard values (not recommended), use set_property BITSTREAM.General.UnconstrainedPins {Allow} on the current design. Problem ports: DataIn_pad_0_i[7:0], DataIn_pad_1_i[7:0],
    DataOut_pad_0_o[7:0], DataOut_pad_1_o[7:0], LineState_pad_0_i[1:0], LineState_pad_1_i[1:0], OpMode_pad_0_o[1:0], OpMode_pad_1_o[1:0], VControl_pad_0_o[3:0], VControl_pad_1_o[3:0],
    VStatus_pad_0_i[7:0], VStatus_pad_1_i[7:0], or1200_pm_out[3:0], GTPRESET_IN, RxActive_pad_0_i, RxActive_pad_1_i, RxError_pad_0_i, RxError_pad_1_i, RxValid_pad_0_i, RxValid_pad_1_i,
    SuspendM_pad_0_o, SuspendM_pad_1_o, TermSel_pad_0_o, TermSel_pad_1_o, TxReady_pad_0_i, TxReady_pad_1_i, TxValid_pad_0_o, TxValid_pad_1_o, VControl_Load_pad_0_o,
    VControl_Load_pad_1_o, XcvSelect_pad_0_o, XcvSelect_pad_1_o, or1200_clmode, or1200_pic_ints, phy_rst_pad_0_o, phy_rst_pad_1_o, reset, sysClk, usb_vbus_pad_0_i, usb_vbus_pad_1_i.
    [Drc 23-20] Rule violation (UCIO-1) Unconstrained Logical Port - 135 out of 135 logical ports have no user assigned specific location constraint (LOC). This may cause I/O contention or incompatibility with the
    board power or connectivity affecting performance, signal integrity or in extreme cases cause damage to the device or the components to which it is connected. To correct this violation, specify all pin
    locations. This design will fail to generate a bitstream unless all logical ports have a user specified site LOC constraint defined. To allow bitstream creation with unspecified pin locations (not recommended), use
```

Figure 41: Showing Critical Warnings and Error

RECOMMENDED Review the DRC messages, Critical Warnings, and Warnings early in the flow to prevent issues later.

At Synthesized Design, the optional Report DRC step reports a Critical Warning for the unconstrained IOs. The routed design DRC report reports the Critical Warnings. You must review the report. At `write_bitstream`, the DRC has been elevated to an Error. Review the DRC reports early to identify areas of the design that need modification.

Multiple Designs

A Design is an in-memory combination of a:

- Netlist
- Constraint set

  The constraint set is a set of XDC constraint files with the target file marked for new constraints.
- Part

The tool supports multiple designs open at the same time. When analyzing code, it can be useful to have the Elaborated Design open at the same time as the Implemented Design. It is also possible to have multiple designs open at the same step of the flow.

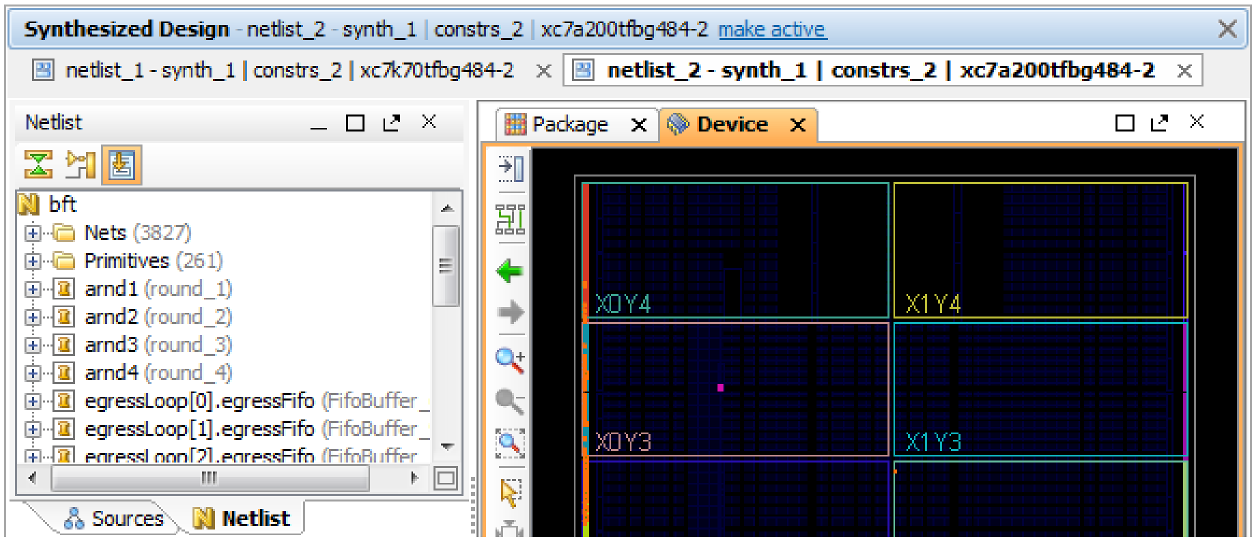

Synthesized Design - netlist_2 - synth_1 | constrs_2 | xc7a200tfbg484-2 make active
netlist_1 - synth_1 | constrs_2 | xc7k70tfbg484-2
netlist_2 - synth_1 | constrs_2 | xc7a200tfbg484-2
Netlist
bft
Nets (3827)
Primitives (261)
arnd1 (round_1)
arnd2 (round_2)
arnd3 (round_3)
arnd4 (round_4)
egressLoop[0].egressFifo (FifoBuffer_
egressLoop[1].egressFifo (FifoBuffer_
Sources
Netlist
Package
Device
X0Y4
X1Y4
X0Y3
X1Y3

Figure 42: Two Synthesized Designs

The **Make Active** link appears in designs using the non-default settings. Click **Make Active** to update the defaults.

Use this feature to compare:

- Multiple different synthesis runs
- Constraint file iterations
- Different parts for the same netlist
- Two different Place and Router results

This feature works best on a 64-bit operating system. A simple way to open multiple designs is to open two runs at the same step. A more advanced method involves opening the single netlist in multiple windows:

1. Right click Synthesized Design.

2. Choose **New Synthesized Design** from the popup menu.

Elaborated Design supports a similar method.

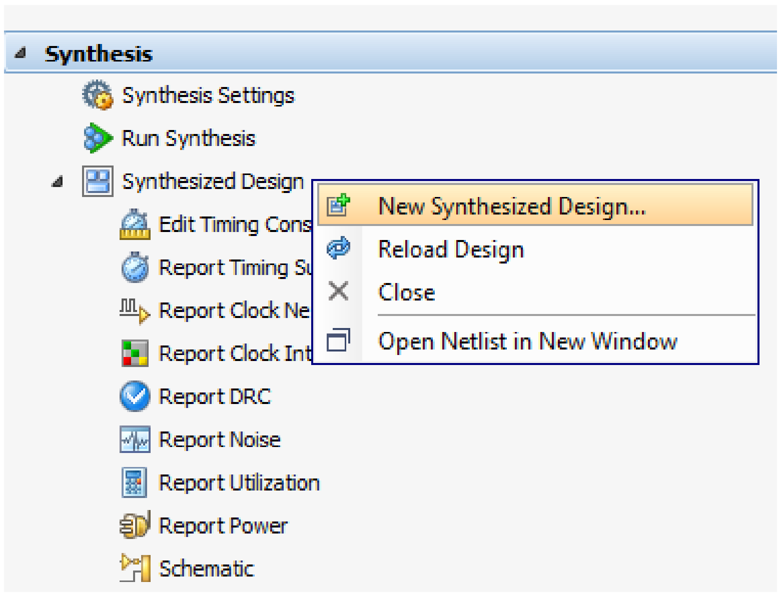


Figure 43: Two Designs Popup

Non-project mode has a similar state. An easy way to get into this state is to open multiple checkpoints without closing either. Use `close_design` to close a checkpoint before opening a second one. It is easiest to issue the `start_gui` command and manage the multiple designs using the Vivado IDE graphical user interface.

Multiple Windows

Users with multiple monitors can float windows or designs. To open an entire Design in a different window (for example, when you have multiple Synthesized Designs open and you want to view them simultaneously), right click the blue Design bar. Select **Open in New Window** from the popup menu.

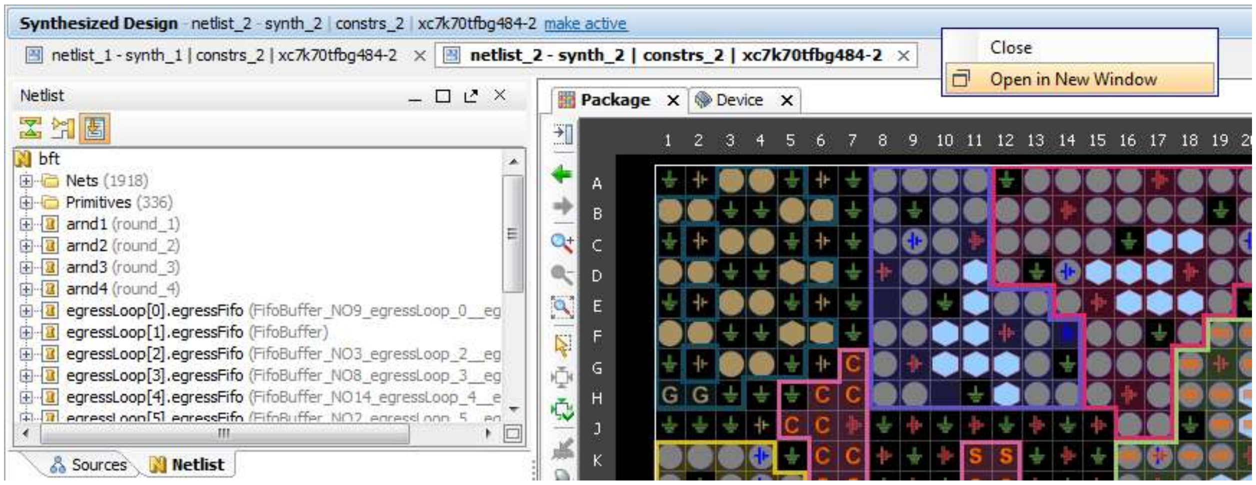

Synthesized Design - netlist_2 - synth_2 | constrs_2 | xc7k70tfbg484-2 make active
netlist_1 - synth_1 | constrs_2 | xc7k70tfbg484-2
netlist_2 - synth_2 | constrs_2 | xc7k70tfbg484-2
Close
Open in New Window
Netlist
bft
Nets (1918)
Primitives (336)
arnd1 (round_1)
arnd2 (round_2)
arnd3 (round_3)
arnd4 (round_4)
egressLoop[0].egressFifo (FifoBuffer_NO9_egressLoop_0__eg
egressLoop[1].egressFifo (FifoBuffer)
egressLoop[2].egressFifo (FifoBuffer_NO3_egressLoop_2__eg
egressLoop[3].egressFifo (FifoBuffer_NO8_egressLoop_3__eg
egressLoop[4].egressFifo (FifoBuffer_NO14_egressLoop_4__e
Sources
Netlist
Package
Device

Figure 44: Float a Design

For information on floating an individual window from a design, see Using Windows in the *Vivado Design Suite User Guide: Using the Vivado IDE (UG893)*.

Chapter 3

Analyzing Implementation Results

This chapter discusses techniques for reviewing a design after implementation to understand behavior inside the device, including:

- Reviewing placement for hierarchical blocks
- IOs
- Looking at connectivity
- Cross probing between views
- Reviewing detailed routing

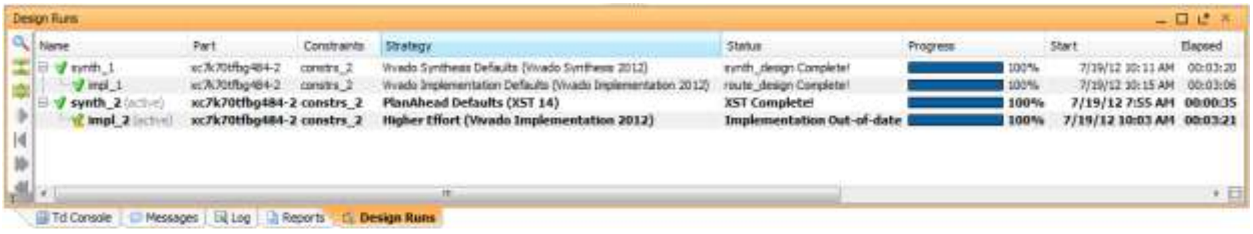
Design Runs Window

The Design Runs window displays the state of the current runs.

For more information, see Using the Design Runs Window in the *Vivado Design Suite User Guide: Implementation (UG904)*.

If the run is running, finished cleanly, or finished with errors, the Design Runs window appears when a run is done.

TIP If the run is not up to date, select **Force up to date** from the popup menu.



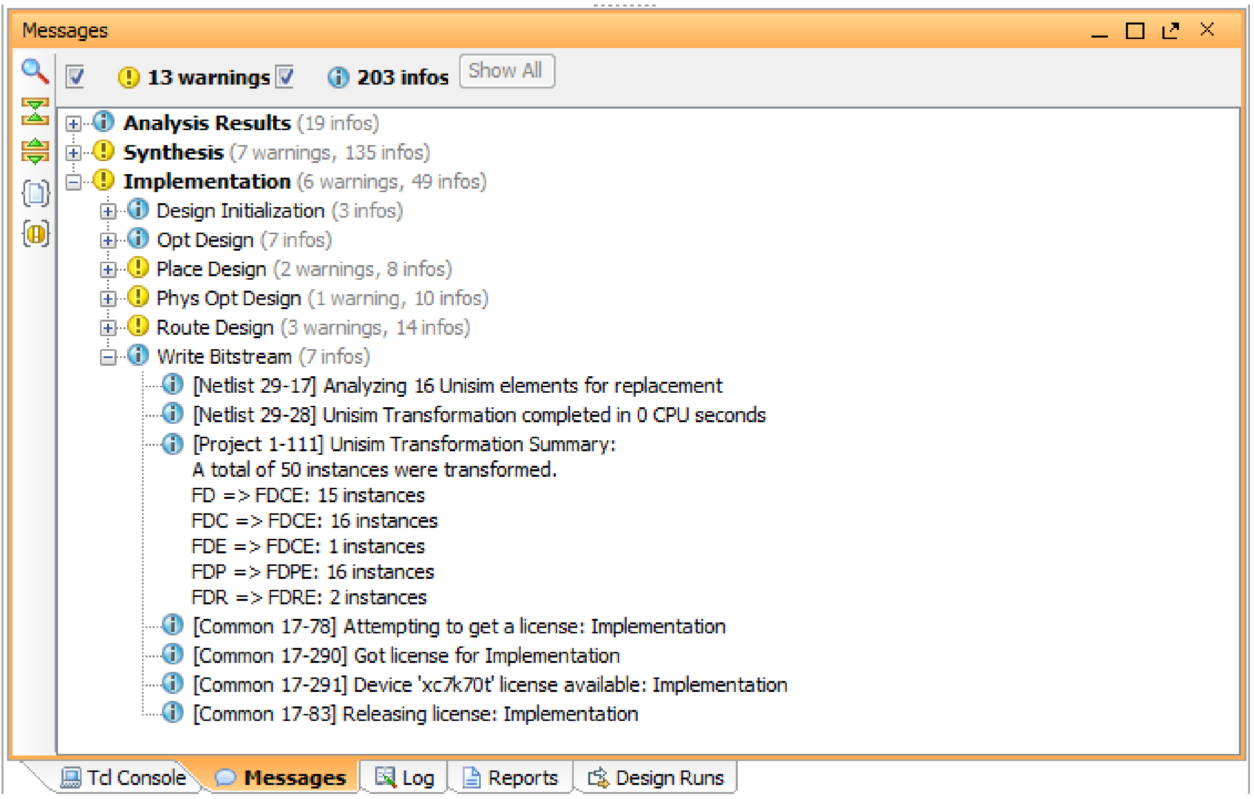
Name	Part	Constraints	Strategy	Status	Progress	Start	Elapsed
synth_1	xc7k70tfbg484-2	constrs_2	Vivado Synthesis Defaults (Vivado Synthesis 2012)	synth_design Complete!	100%	7/19/12 10:11 AM	00:03:20
impl_1	xc7k70tfbg484-2	constrs_2	Vivado Implementation Defaults (Vivado Implementation 2012)	route_design Complete!	100%	7/19/12 10:15 AM	00:03:06
synth_2 (active)	**xc7k70tfbg484-2**	**constrs_2**	**PlanAhead Defaults (XST 14)**	**XST Complete!**	**100%**	**7/19/12 7:55 AM**	**00:00:35**
impl_2 (active)	**xc7k70tfbg484-2**	**constrs_2**	**Higher Effort (Vivado Implementation 2012)**	**Implementation Out-of-date**	**100%**	**7/19/12 10:03 AM**	**00:03:21**

Figure 45: Design Runs Window

The Design Runs Window does the following:

- Displays the message `route_design Complete!`
- Gives a brief summary.
- Reports any design error.
- Does not report whether the design met timing.
- Does not report Critical Warnings or other design issues.

If you are using the Vivado™ IDE project flow, review the Messages tab for your active implementation. Messages are grouped by each step in the run flow. The critical information from the implementation steps appears in this view. See the message in the log file to view the context.



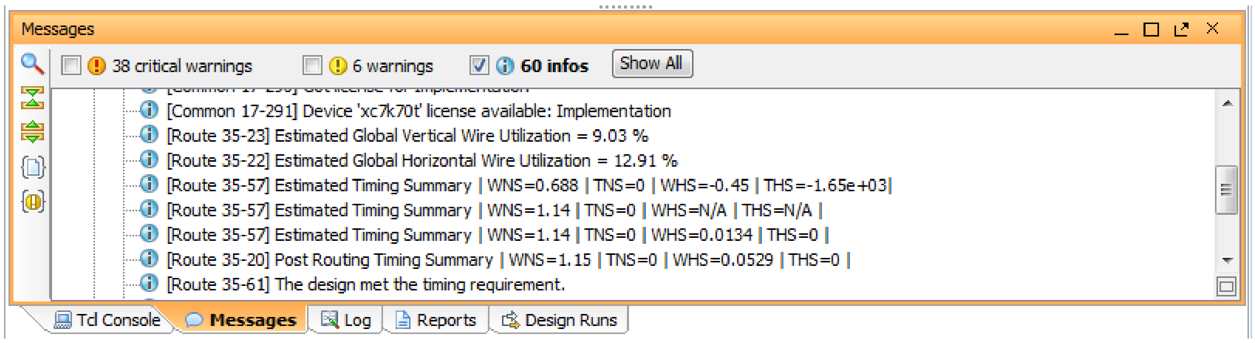
```
Analysis Results (19 infos)
Synthesis (7 warnings, 135 infos)
Implementation (6 warnings, 49 infos)
    Design Initialization (3 infos)
    Opt Design (7 infos)
    Place Design (2 warnings, 8 infos)
    Phys Opt Design (1 warning, 10 infos)
    Route Design (3 warnings, 14 infos)
    Write Bitstream (7 infos)
        [Netlist 29-17] Analyzing 16 Unisim elements for replacement
        [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
        [Project 1-111] Unisim Transformation Summary:
        A total of 50 instances were transformed.
        FD => FDCE: 15 instances
        FDC => FDCE: 16 instances
        FDE => FDCE: 1 instances
        FDP => FDPE: 16 instances
        FDR => FDRE: 2 instances
        [Common 17-78] Attempting to get a license: Implementation
        [Common 17-290] Got license for Implementation
        [Common 17-291] Device 'xc7k70t' license available: Implementation
        [Common 17-83] Releasing license: Implementation
```

Figure 46: Messages Grouped by Step

If there are warnings or errors, review one of the reports listed above for more information. Some messages crossprobe back to the design elements related to the message.

In order to analyze the design in the context of the message, you may need to open either:

- Implemented Design (using the netlist at the end of implementation), or
- Netlist Design (using the netlist before implementation)



```
[Common 17-291] Device 'xc7k70t' license available: Implementation
[Route 35-23] Estimated Global Vertical Wire Utilization = 9.03 %
[Route 35-22] Estimated Global Horizontal Wire Utilization = 12.91 %
[Route 35-57] Estimated Timing Summary | WNS=0.688 | TNS=0 | WHS=-0.45 | THS=-1.65e+03|
[Route 35-57] Estimated Timing Summary | WNS=1.14 | TNS=0 | WHS=N/A | THS=N/A |
[Route 35-57] Estimated Timing Summary | WNS=1.14 | TNS=0 | WHS=0.0134 | THS=0 |
[Route 35-20] Post Routing Timing Summary | WNS=1.15 | TNS=0 | WHS=0.0529 | THS=0 |
[Route 35-61] The design met the timing requirement.
```

Figure 47: Post Implementation Messages

The router gives a message if a design met timing or not.

IMPORTANT The router is not sign off timing. Only `report_timing_summary` is sign off timing.

Review the Timing Summary Report to determine if the design meets timing. If it did not meet timing, review the timing closure techniques discussed in *Chapter Five: Closure Techniques*.

In the non-project flow (provided that the Vivado IDE graphical user interface is open), messages generated during that executable session still show in the Messages Window. Otherwise, review the following:

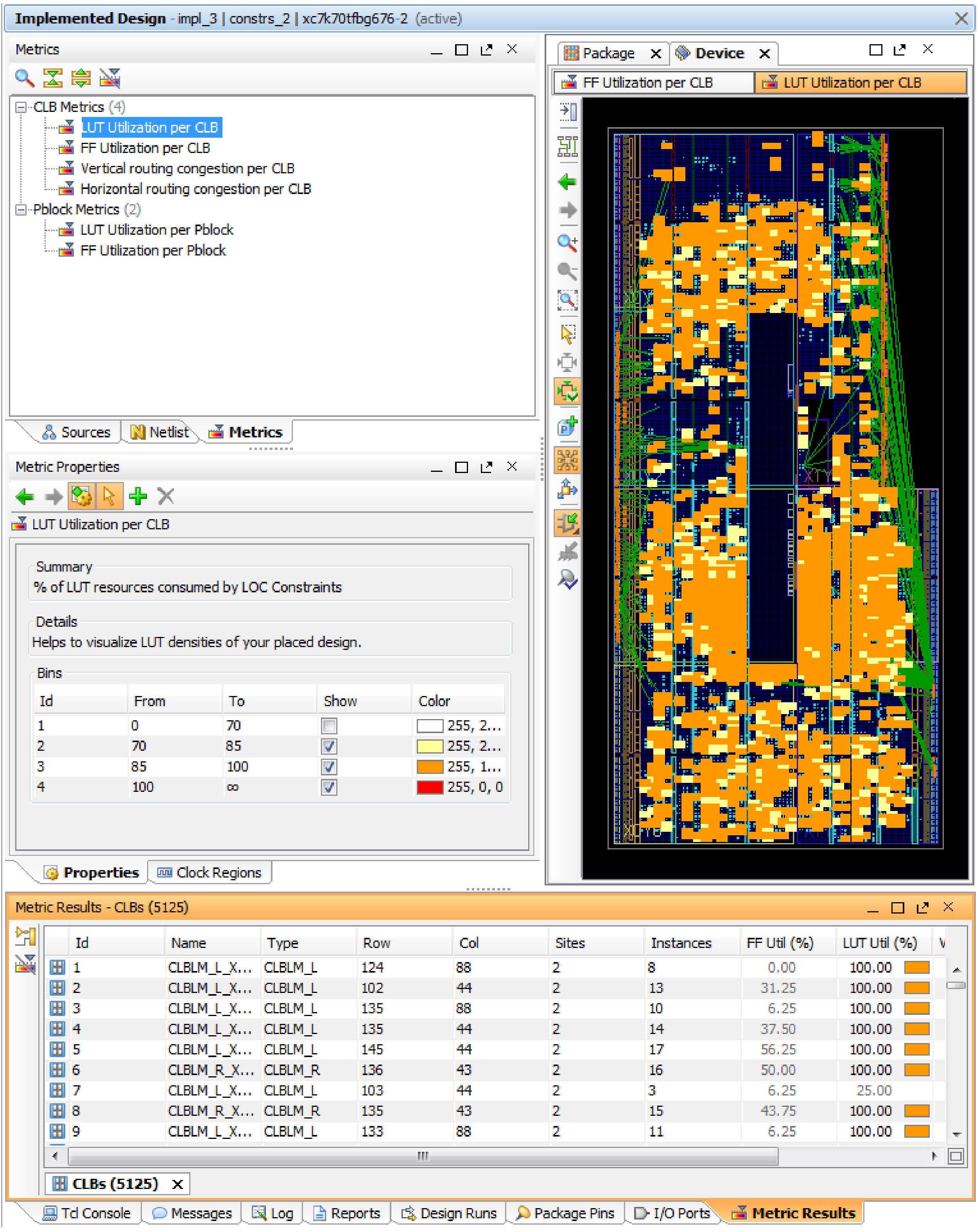
- Vivado IDE log
- Notes, Warnings and Errors

If you are running without the Vivado IDE graphical user interface, run the TCL `report_timing_summary` command to determine if the design met timing.

More Analysis

After implementation finishes, you may want to analyze the design to see how it interacts with the device.

The Vivado IDE has a number of metrics to help you determine logic and routing usage inside the device. The Metrics color code the device window based on a specified rule.



Implemented Design - impl_3 | constrs_2 | xc7k70tfbg676-2 (active)

Metrics

- CLB Metrics (4)
  - LUT Utilization per CLB
  - FF Utilization per CLB
  - Vertical routing congestion per CLB
  - Horizontal routing congestion per CLB
- Pblock Metrics (2)
  - LUT Utilization per Pblock
  - FF Utilization per Pblock

Metric Properties

LUT Utilization per CLB

Summary
% of LUT resources consumed by LOC Constraints

Details
Helps to visualize LUT densities of your placed design.

Bins

Id	From	To	Show	Color
1	0	70	☐	⬜ 255, 2...
2	70	85	☑	🟨 255, 2...
3	85	100	☑	🟧 255, 1...
4	100	∞	☑	🟥 255, 0, 0

Metric Results - CLBs (5125)

Id	Name	Type	Row	Col	Sites	Instances	FF Util (%)	LUT Util (%)
1	CLBLM_L_X...	CLBLM_L	124	88	2	8	0.00	100.00 🟧
2	CLBLM_L_X...	CLBLM_L	102	44	2	13	31.25	100.00 🟧
3	CLBLM_L_X...	CLBLM_L	135	88	2	10	6.25	100.00 🟧
4	CLBLM_L_X...	CLBLM_L	135	44	2	14	37.50	100.00 🟧
5	CLBLM_L_X...	CLBLM_L	145	44	2	17	56.25	100.00 🟧
6	CLBLM_R_X...	CLBLM_R	136	43	2	16	50.00	100.00 🟧
7	CLBLM_L_X...	CLBLM_L	103	44	2	3	6.25	25.00
8	CLBLM_R_X...	CLBLM_R	135	43	2	15	43.75	100.00 🟧
9	CLBLM_L_X...	CLBLM_L	133	88	2	11	6.25	100.00 🟧

Figure 48: Metrics

Metrics Requiring a Placed Design

Four metrics require a placed design in order to be accurate. They do not require a fully routed design.

- LUT Utilization per CLB

  Color codes slices based on placed LUT utilization.
- FF Utilization per CLB

  Color codes slices based on placed FF utilization.
- Vertical Routing Congestion per CLB

  Color codes the fabric based on a best case estimate of vertical routing usage.
- Horizontal Routing Congestion per CLB

  Color codes the fabric based on a best case estimate of horizontal routing usage.

Metrics in a Netlist Design With No Placement

Two metrics are applicable if there are Pblocks. They do not depend on placement.

- LUT Utilization per Pblock

  Color codes the Pblock based on an estimate of how the LUTs will be placed into the slices contained in the Pblock.
- FF Utilization per Pblock

  Color codes the Pblock based on an estimate of how the FFs will be packed into the slices contained in the Pblock.

More than one rule can be used at a time as shown in the figure above (Metrics). Both LUT Utilization per CLB and FF Utilization per CLB are on.

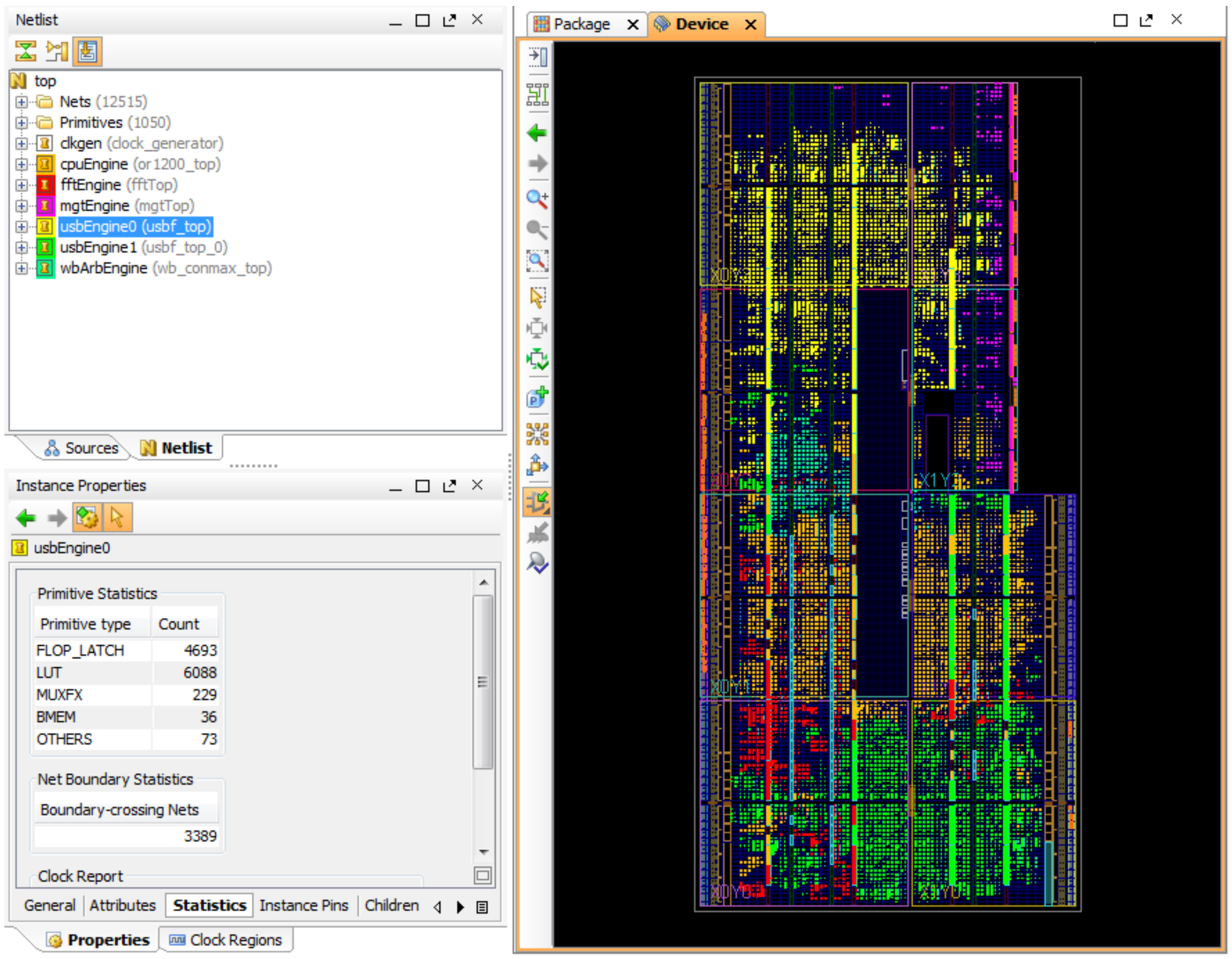
TIP If there are sections of the design with high utilization or high estimates of routing congestion, consider tweaking the RTL or placement constraints to reduce logic and routing utilization in that area.

Highlight Placement

Another way to review design placement is to analyze cell placement. The **Highlight Primitives** command helps in this analysis.

1. In the Netlist Window, select the levels of hierarchy to analyze.
2. From the popup menu, select **Highlight Primitives > Select a color**.
3. If you select multiple levels of hierarchy, select **Cycle Colors**.

The primitives that make up the cells are color coded in the Device window.



Primitive Statistics

Primitive type	Count
FLOP_LATCH	4693
LUT	6088
MUXFX	229
BMEM	36
OTHERS	73

Net Boundary Statistics

Boundary-crossing Nets
3389

Figure 49: Highlight Hierarchy

The color coding readily shows that `UsbEngine0` (in yellow):

- Uses a number of Block RAM and DSP48 cells.
- Is in the top clock region of the chip except where the DSPs bleed out.
- Is not highly intermingled with other logic (cells) in the design.

It is easy to see that the `fftEngine` (in red) and the `cpuEngine` (in brown) are intermingled. The two blocks primarily use different resources (DSP48 as opposed to slices). Intermingling makes best use of the device.

Show Connectivity

It can be useful to analyze a design based on connectivity. You may want to review the placement of all the logic driven by an input, a Block RAM, or a bank of DSPs. Use the **Show Connectivity** command for this purpose. **Show Connectivity** takes a set of cells or nets as a seed, and selects objects of the other type. Use this technique to build up and see cones of logic inside the design.

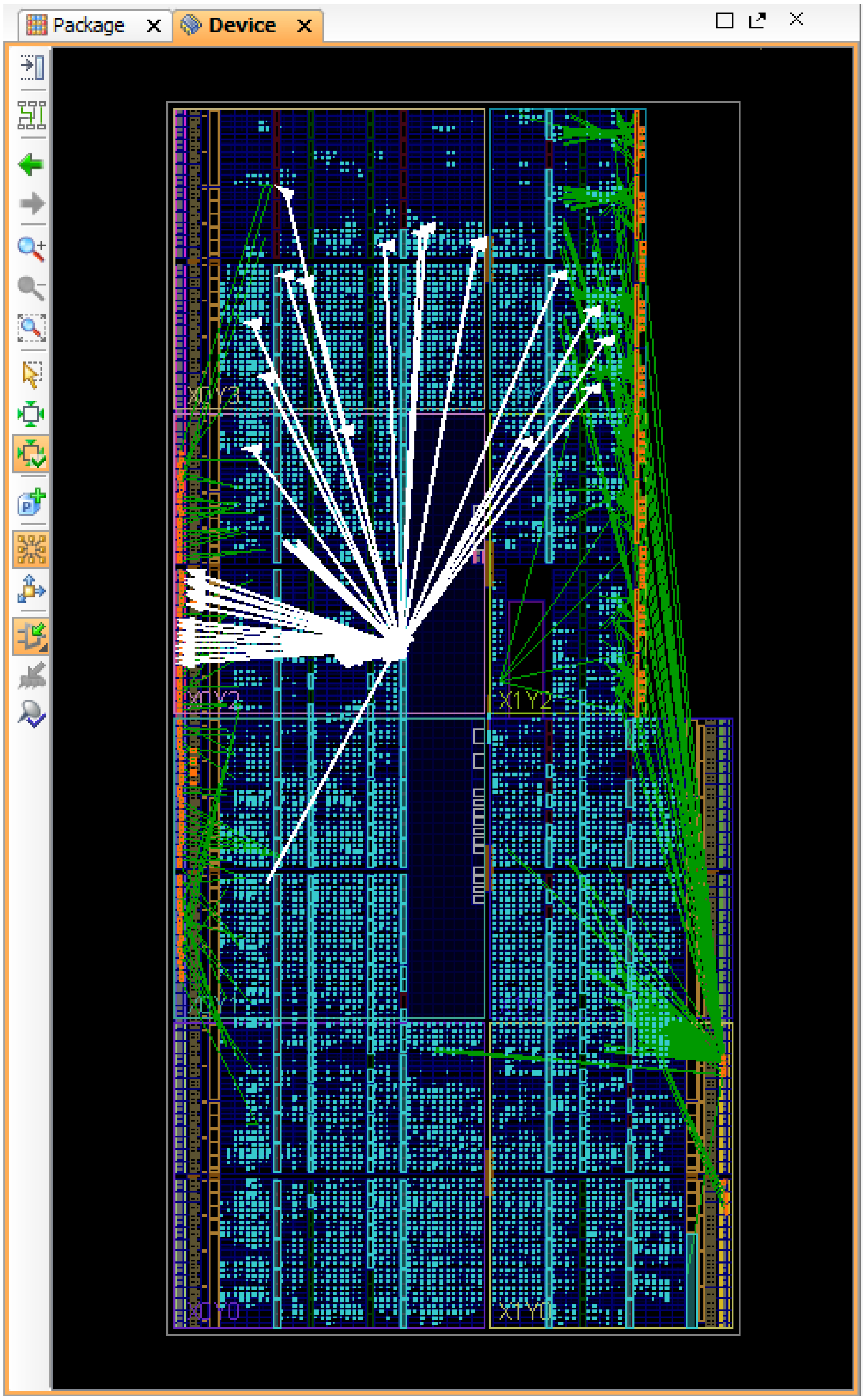


Figure 50: Show Connectivity

The figure above (Show Connectivity) shows a Block RAM driving logic inside the device including OBUFs. A synthesis pragma stops synthesis from placing the output flop in the Block RAM during memory inferencing.

Fixed and Unfixed Logic

The tool tracks two different types of placement:

- Elements placed by the user (shown in orange) are Fixed.
    - Fixed logic is stored in the XDC.
    - Fixed logic normally has a LOC constraint and may have a BEL constraint.
- Elements placed by the tool (shown in blue) are Unfixed.

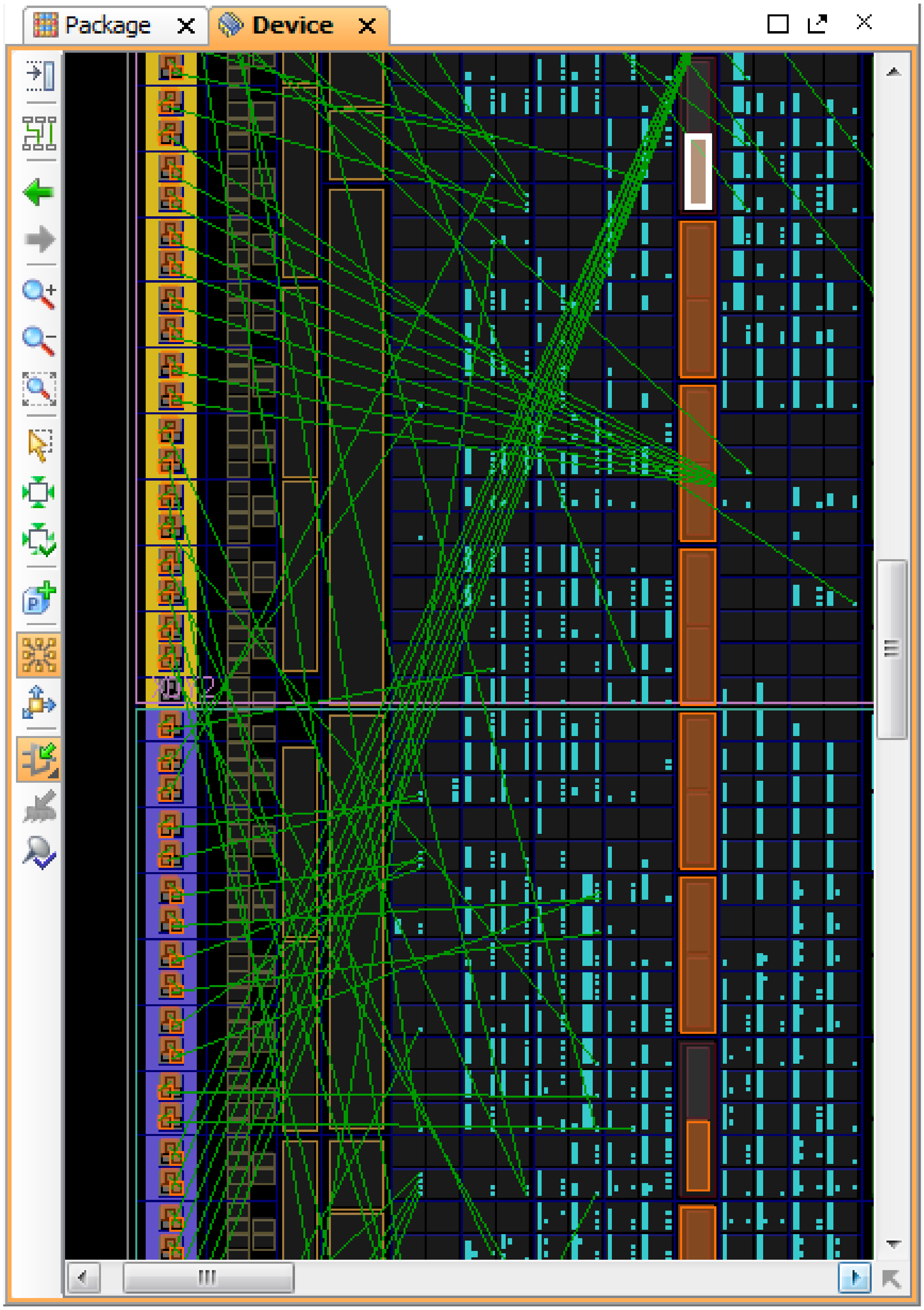


Figure 51: Fixed Unfixed

The IO and Block RAM placement is Fixed. The slice logic is Unfixed.

To fix logic:

1. Enter placement into the XDC.
2. Select **logic** in the Vivado IDE graphical user interface.
3. From the popup, select **Fix Instances**, or
4. Set the `is_loc_fixed` or `is_bel_fixed` properties in the TCL console.

In the XDC, the constraints look like the following.

- For a Flip Flop with a LOC and BEL

```
set_property BEL DFF [get_cells
{cpuEngine/cpu_iwb_dat_i/buffer_fifo/infer_fifo.wr_addr_tmp_reg[9]}]
set_property LOC SLICE_X7Y103 [get_cells
{cpuEngine/cpu_iwb_dat_i/buffer_fifo/infer_fifo.wr_addr_tmp_reg[9]}]
```

Flip Flops can be placed with only a LOC.

- For a Block RAM

```
set_property LOC RAMB36_X0Y21 [get_cells
usbEngine1/dma_out/buffer_fifo/infer_fifo.block_ram_performance.fifo_ram_reg]
```

- For an IO Pin

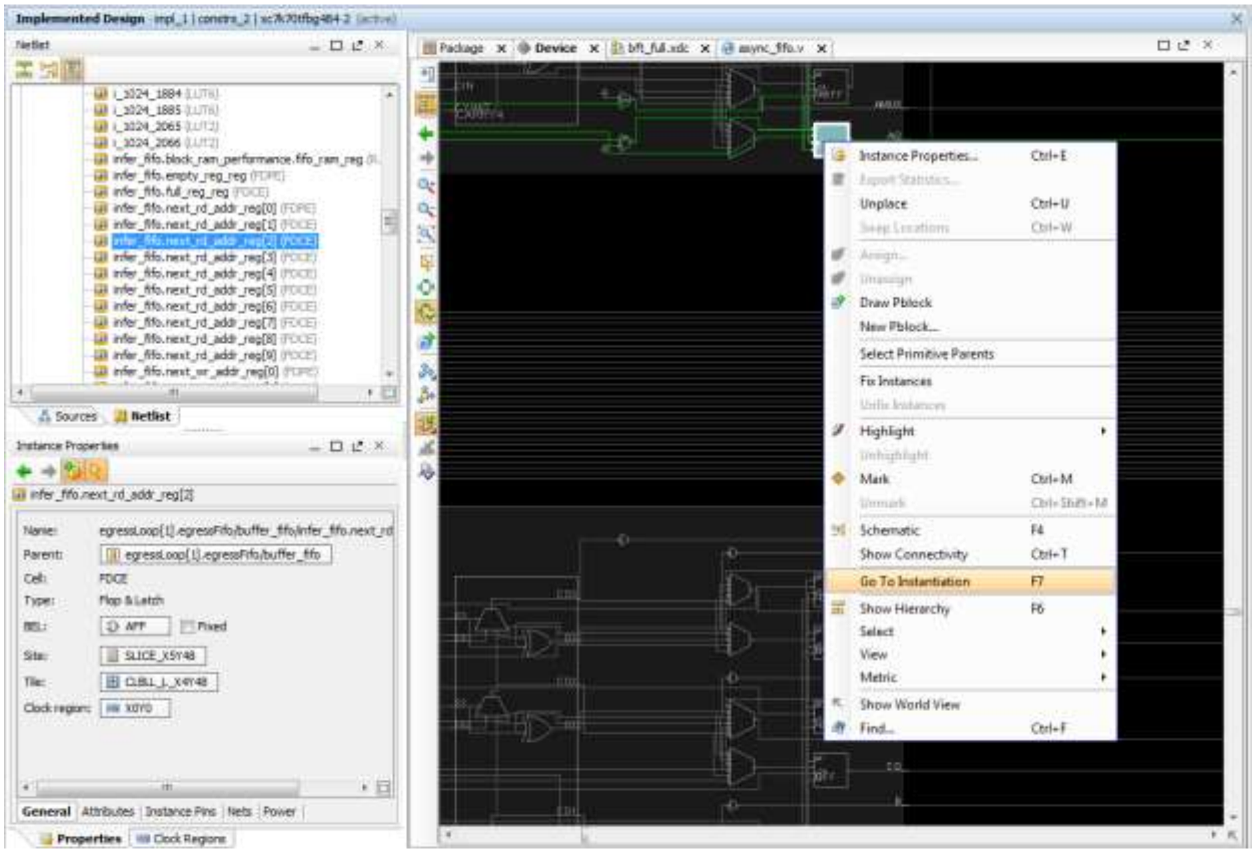
```
set_property PACKAGE_PIN E23 [get_ports {DataIn_pad_0_i[5]}]
```

Cross Probing

For designs synthesized with Vivado Synthesis, it is possible to cross probe back to the source files once the netlist design is in memory.

To cross probe:

1. Select the gate.
2. Select **Go to instantiation** from the popup menu.


Implemented Design - impl_1 | constrs_2 | xc7k70tfbg484-2 (active)

Netlist

```
i_1024_1884 (LUT6)
i_1024_1885 (LUT6)
i_1024_2065 (LUT2)
i_1024_2066 (LUT2)
infer_fifo.block_ram_performance.fifo_ram_reg
infer_fifo.empty_reg_reg (FDPE)
infer_fifo.full_reg_reg (FDCE)
infer_fifo.next_rd_addr_reg[0] (FDPE)
infer_fifo.next_rd_addr_reg[1] (FDCE)
infer_fifo.next_rd_addr_reg[2] (FDCE)
infer_fifo.next_rd_addr_reg[3] (FDCE)
infer_fifo.next_rd_addr_reg[4] (FDCE)
infer_fifo.next_rd_addr_reg[5] (FDCE)
infer_fifo.next_rd_addr_reg[6] (FDCE)
infer_fifo.next_rd_addr_reg[7] (FDCE)
infer_fifo.next_rd_addr_reg[8] (FDCE)
infer_fifo.next_rd_addr_reg[9] (FDCE)
infer_fifo.next_wr_addr_reg[0] (FDPE)
```

Instance Properties

infer_fifo.next_rd_addr_reg[2]

Name:	egressLoop[1].egressFifo/buffer_fifo/infer_fifo.next_rd
Parent:	egressLoop[1].egressFifo/buffer_fifo
Cell:	FDCE
Type:	Flop & Latch
BEL:	AFF ☐ Fixed
Site:	SLICE_X5Y48
Tile:	CLBLL_L_X4Y48
Clock region:	X0Y0

Figure 52: Xprobe Back To Source

Use crossprobing to determine which source is involved in netlist gates. Due to the nature of synthesis transforms, it is not possible to cross probe back to source for every gate in the design.

Viewing Routing

Turn on Routing Resources in the Device View to view the exact routing resources..

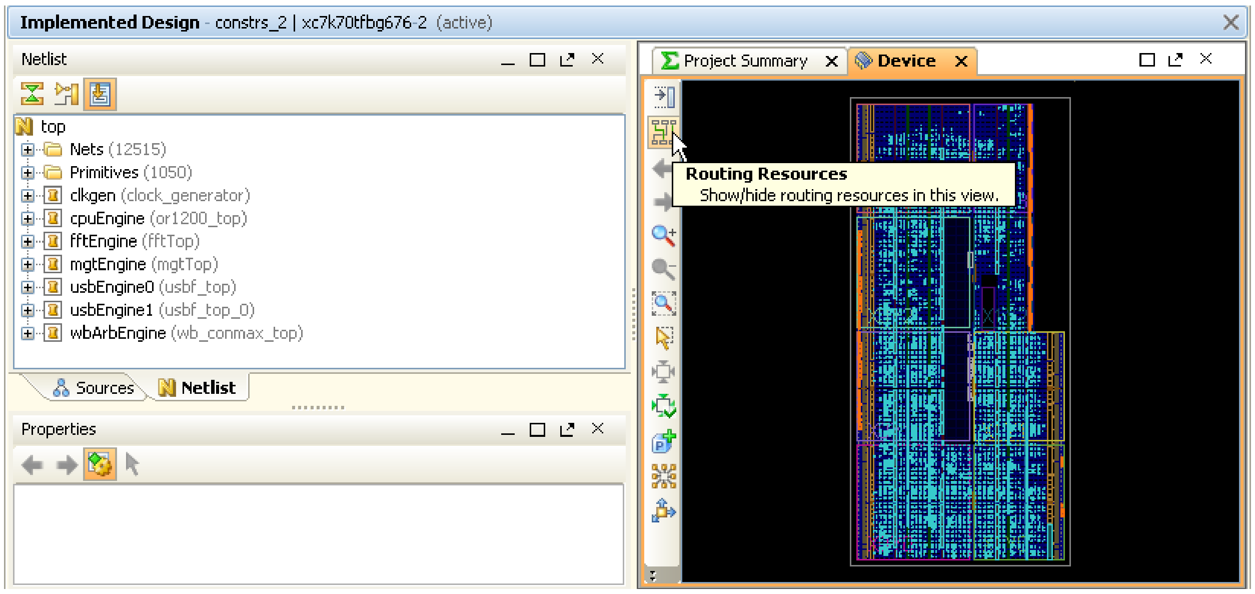


Figure 53: Enable Routing

Displaying Routing and Placement

Routing and placement are displayed in two different ways depending on the zoom level:

- When zoomed out
- At closer zoom levels

The two visualizations of the Device view minimize runtime and memory usage while showing the details of designs of all sizes.

Displaying Routing and Placement When Zoomed Out

When zoomed out, an abstract view is shown. The abstract view:

- Condenses the routes through the device.
- Shows lines of different thicknesses depending on the number of routes through a particular region.

Placement similarly displays a block for each tile with logic placed in it. The more logic in a tile, the larger the block representing that tile will be.

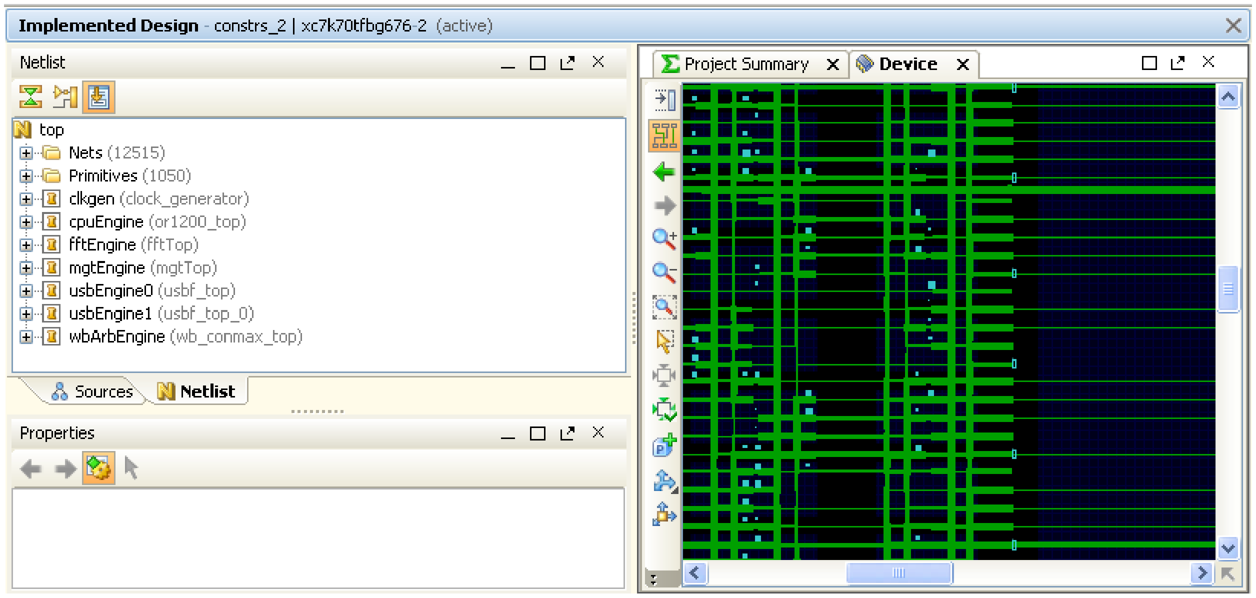

Implemented Design - constrs_2 | xc7k70tfbg676-2 (active)

Figure 54: Abstract View

Displaying Routing and Placement at Closer Zoom Levels

At closer zoom levels, the actual logic cells and routes are shown.

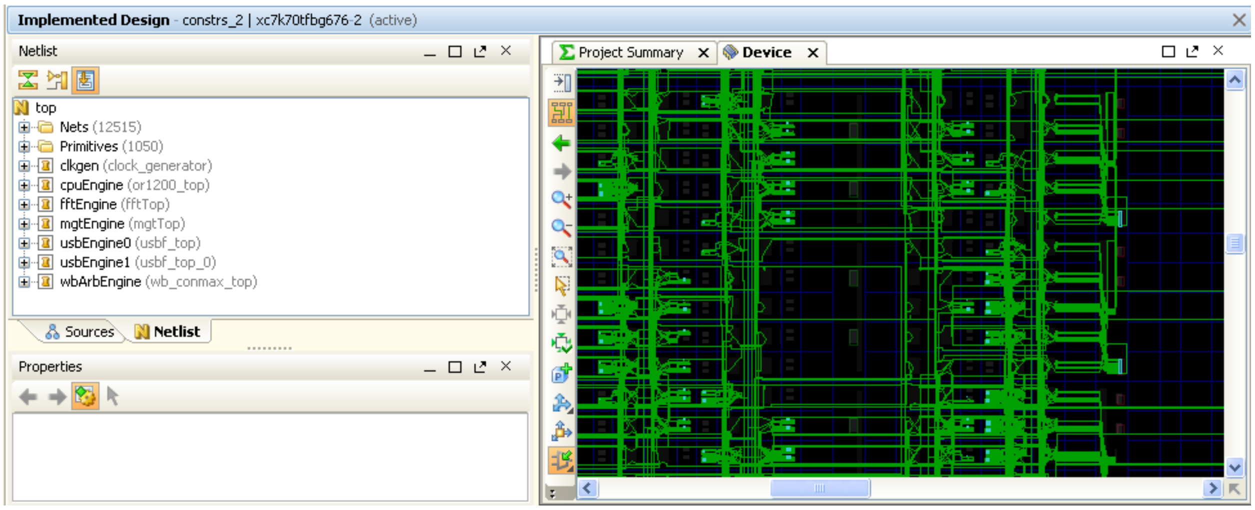

Implemented Design - constrs_2 | xc7k70tfbg676-2 (active)

Figure 55: Detailed View

Viewing Options

The Device View is customizable to show the device, and design, in a variety of ways. Most of these are controlled through the Device View Options slideout.

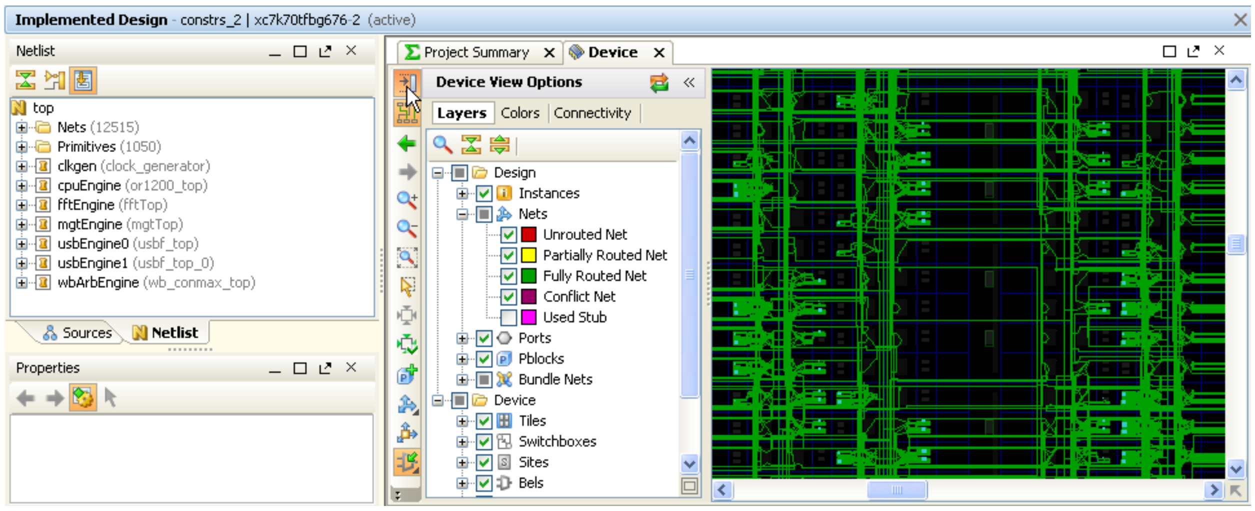


Figure 56: Layers

You can enable or disable the graphics for different design and device resources, as well as modify the display colors.

Navigating in the Device View

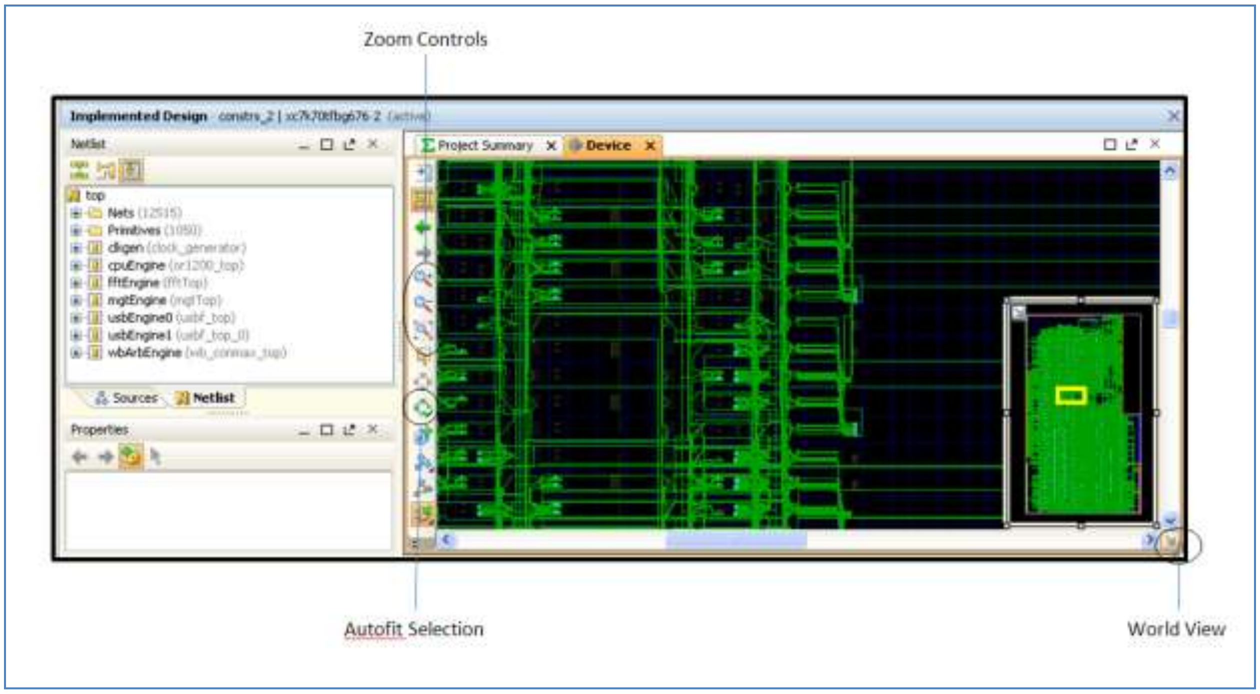


Figure 57: Navigate

Use the following tools to navigate in the Device View.

- Zoom Controls

  Standard Zoom In, Zoom Out, and Zoom Full tools.

- Autofit Selection

  Automatically zoom and pan to an object selected in any view outside of the device. Autofit Selection is particularly useful for cross probing.

- World View

  The World View shows where the currently visible portion of the device is on the overall device. You can move and resize the World View, as well as drag and resize the yellow box to zoom and pan.

- Control Hotkey

  Press **Ctrl** while clicking and dragging to pan the view.

Chapter 4

Timing Analysis

The Vivado™ Design Suite provides several reporting commands you can use to verify that your design meets all the timing constraints and is ready to be loaded on the application board. Report Timing Summary is the timing signoff report, equivalent to TRCE in the ISE™ Design Suite. It provides a comprehensive overview of all the timing checks and shows enough information to start analyzing and debugging any timing issue. For more information, see Chapter 1, Design Analysis.

You can generate this report in a window, write it to a file or simply print it in your log file. Whenever Report Timing Summary shows that your design does not meet timing or is missing some constraints, you can explore the details provided in the various sections of the summary and run more specific analysis depending on the nature of the problem. The other timing reports provide more details on a particular situation or to scope the analysis to some logic by using filters.

Verifying Timing Signoff

Before going into the details of timing analysis, it is important to understand which part of the timing reports indicates that your design is ready to run in hardware. Timing signoff is a mandatory step in the analysis of the implementation results , once your design is fully placed and routed. By default, when using projects in the Vivado Design Suite, the runs automatically generate the text version of Report Timing Summary. You can also generate this report interactively after loading the post-implementation design checkpoint in memory.

Timing signoff is the combination of two criteria:

- **Your design is fully constrained:**

  You can verify that your design is fully constrained by looking at the Check Timing section. Check Timing must show that:

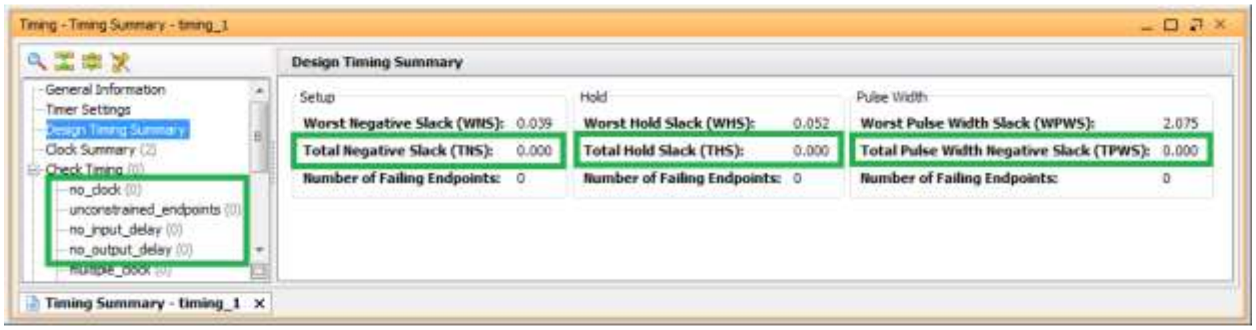
  - All non-constant clock pins are reached by a defined clock (`no_clock` check).
  - All internal path endpoints are timed (`unconstrained_endpoint` check).
  - All input and output ports are constrained (`no_input_delay`, `no_output_delay` checks).

  You can waive some of the missing constraints, at the risk of lowering the signoff quality of your design. You also need to review the result of the other checks to make sure that they are expected and they do not affect the signoff quality.

- **Your design meets timing:**
    - Total Negative Slack (TNS) is 0ns. This covers max delay analysis.
    - Total Hold Slack (THS) is 0ns. This covers min delay analysis.
    - Total Pulse Width Slack (TPWS) is 0ns. This is equivalent to Component Pin Switching Limit in ISE. It is performed with both min and max delays.

    The sum of TNS, THS and TPWS is equivalent to the ISE final Timing Score.

The following figure (Report Timing Summary Signoff in Vivado IDE) highlights in green the information you must look at first in order to verify that the timing signoff is clean.

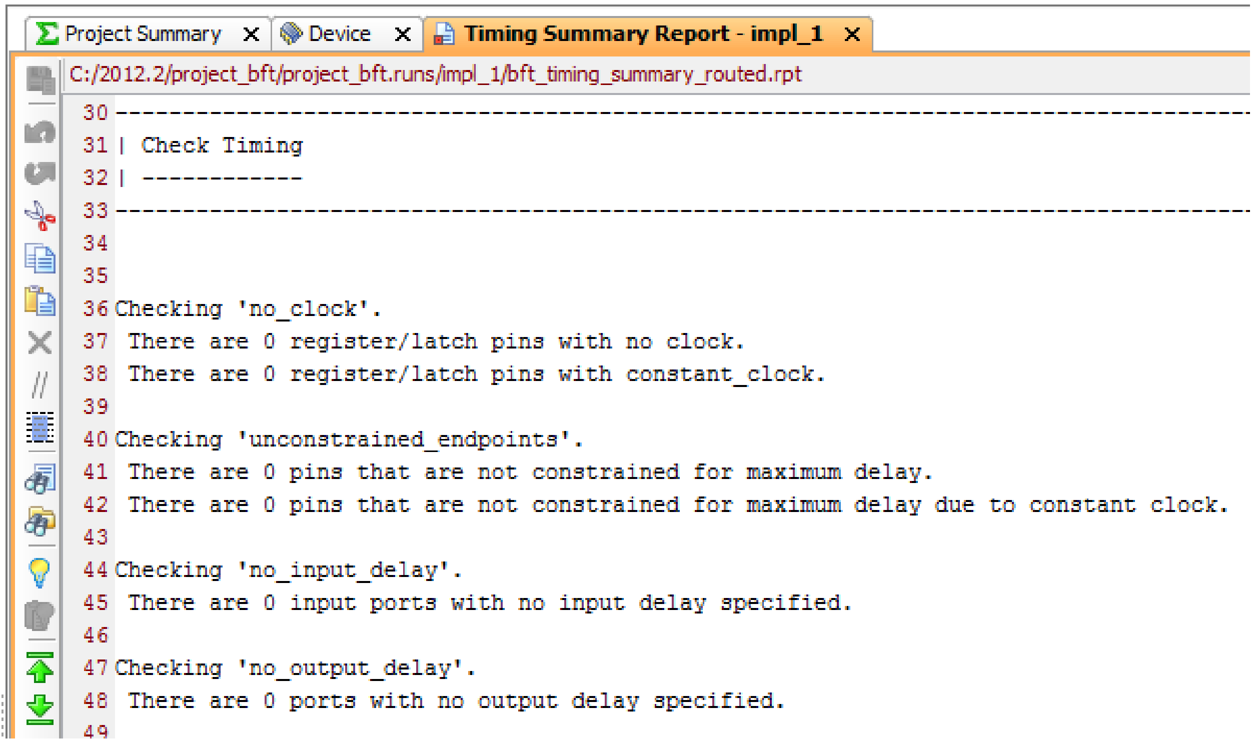


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.039	Worst Hold Slack (WHS):	0.052	Worst Pulse Width Slack (WPWS):	2.075
Total Negative Slack (TNS):	0.000	Total Hold Slack (THS):	0.000	Total Pulse Width Negative Slack (TPWS):	0.000
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0

Figure 58: Report Timing Summary Signoff in Vivado IDE

The following figure (Check Timing Signoff in Text Report) shows the Check Timing information to verify in the text report.

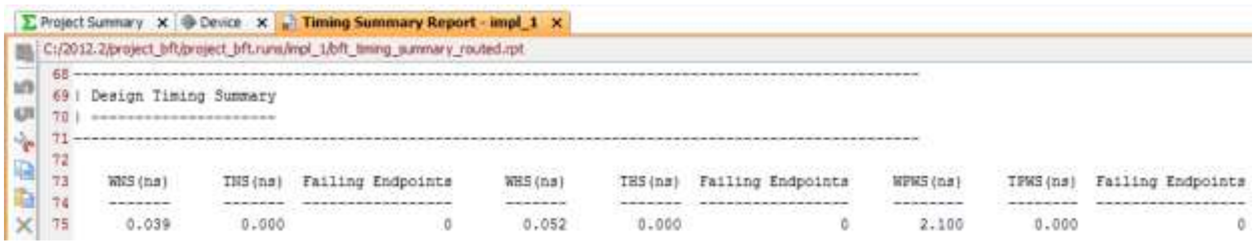


C:/2012.2/project_bft/project_bft.runs/impl_1/bft_timing_summary_routed.rpt

```
-------------------------------------------------------------------------------
| Check Timing
| ------------
-------------------------------------------------------------------------------


Checking 'no_clock'.
 There are 0 register/latch pins with no clock.
 There are 0 register/latch pins with constant_clock.

Checking 'unconstrained_endpoints'.
 There are 0 pins that are not constrained for maximum delay.
 There are 0 pins that are not constrained for maximum delay due to constant clock.

Checking 'no_input_delay'.
 There are 0 input ports with no input delay specified.

Checking 'no_output_delay'.
 There are 0 ports with no output delay specified.
```

Figure 59: Check Timing Signoff in Text Report

The following figure (Design Timing Summary Signoff in Text Report) shows the Design Timing Summary information to verify in the text report.


Project Summary
Device
Timing Summary Report - impl_1
C:/2012.2/project_bft/project_bft.runs/impl_1/bft_timing_summary_routed.rpt

```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  Failing Endpoints       WHS(ns)      THS(ns)  Failing Endpoints      WPWS(ns)      TPWS(ns)  Failing Endpoints
    -------      -------  -----------------       -------      -------  -----------------      --------      --------  -----------------
      0.039        0.000                  0         0.052        0.000                  0         2.100         0.000                  0
```

Figure 60: Design Timing Summary Signoff in Text Report

In the Vivado IDE, any slack violation is reported in red, while the missing constraints are not highlighted with a particular color. You can begin investigating any timing violation by reviewing the following sections:

- The Intra-Clock Paths, Inter-Clock Paths and Path Groups (**async_default**) sections provide information on setup/recovery and hold/removal violations.
- Intra-Clock Paths provides details on Pulse Width check violations.

For more information on the report windows, see Timing Summary Report in Chapter 1, Design Analysis.

To display violations only, click the **Show only failing checks** button.

If you used the default options, the Timing Summary report includes the details of the N-worst paths for each clock pair and for each type of analysis. The GUI default for N is 10 (ten), and the `report_timing_summary` command default for N is 1 (one). You can directly look into the timing path details without running another report by double-clicking on the path. If not enough paths are reported, you can either re-run Report Timing Summary with more paths (Tcl equivalent: *report_timing_summary –max_paths N*) or simply run Report Timing on a particular clock pair or a particular timing path.

Reading a Timing Path Report

The timing path report provides all the information needed to understand what causes a timing violation. It is separated into the four sections described below.

Timing Path Summary

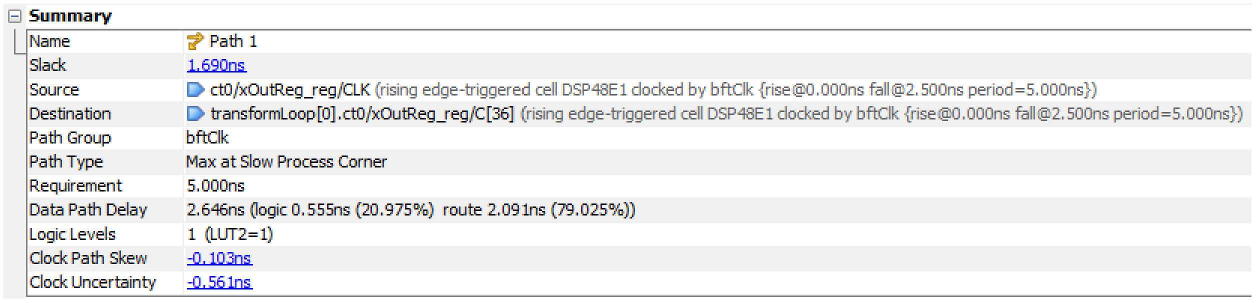
The Timing Path Summary is where the important information from the timing path details is gathered. You can review it to find out about the cause of a violation without having to analyze the details of the timing path. It includes slack, path requirement, datapath delay, cell delay, route delay, clock skew and clock uncertainty. It does not provide any information about cell placement. For more information about the terminology used for timing constraints and timing analysis, as well as learn how slack and path requirement are determined, see Timing Analysis in the *Vivado Design Suite Using Constraints User Guide (UG903)*.

The following figure (Timing Path Summary In Text Report) shows an example of the Timing Path Summary Header in a text report.

```
Slack (MET) :              1.690ns
  Source:                  ct0/xOutReg_reg/CLK
                             (rising edge-triggered cell DSP48E1 clocked by bftClk
                                                       {rise@0.000ns fall@2.500ns period=5.000ns})
  Destination:             transformLoop[0].ct0/xOutReg_reg/C[36]
                             (rising edge-triggered cell DSP48E1 clocked by bftClk
                                                       {rise@0.000ns fall@2.500ns period=5.000ns})
  Path Group:              bftClk
  Path Type:               Max at Slow Process Corner
  Requirement:             5.000ns
  Data Path Delay:         2.646ns  (logic 0.555ns (20.975%)  route 2.091ns (79.025%))
  Logic Levels:            1  (LUT2=1)
  Clock Path Skew:         -0.103ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    3.797ns
    Source Clock Delay      (SCD):    4.123ns
    Clock Pessimism Removal (CPR):    0.223ns
  Clock Uncertainty:       0.561ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE + UU
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.100ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
    User Uncertainty         (UU):    0.500ns
```

Figure 61: Timing Path Summary In Text Report

The following figure (Timing Path Summary in Vivado IDE) shows an example of the Timing Path Summary header in the Vivado IDE.



Summary

Name	Path 1
Slack	1.690ns
Source	ct0/xOutReg_reg/CLK (rising edge-triggered cell DSP48E1 clocked by bftClk {rise@0.000ns fall@2.500ns period=5.000ns})
Destination	transformLoop[0].ct0/xOutReg_reg/C[36] (rising edge-triggered cell DSP48E1 clocked by bftClk {rise@0.000ns fall@2.500ns period=5.000ns})
Path Group	bftClk
Path Type	Max at Slow Process Corner
Requirement	5.000ns
Data Path Delay	2.646ns (logic 0.555ns (20.975%) route 2.091ns (79.025%))
Logic Levels	1 (LUT2=1)
Clock Path Skew	-0.103ns
Clock Uncertainty	-0.561ns

Figure 62: Timing Path Summary in Vivado IDE

The Timing Path Summary header includes the following information:

- **`Slack`**: A positive slack indicates that the path meets the path requirement, which is derived from the timing constraints. The Slack equation depends on the analysis performed:
  - Max delay analysis (setup/recovery):

    `slack = data required time – data arrival time`
  - Min delay analysis (hold/removal):

    `slack = data arrival time – data required time`

Data required and arrival times are calculated and reported in the other sub-sections of the timing path report.

- **`Source`**: shows the path startpoint and the source clock that launches the data. The startpoint is usually the clock pin of a sequential cell or an input port. Whenever applicable, the second line displays the primitive and the edge sensitivity of the clock pin. It also provides the clock name and the clock edges definition (waveform and period).
- **`Destination`**: shows the path endpoint and the destination clock that captures the data. The endpoint is usually the input data pin of the destination sequential cell or an output port. Whenever applicable, the second line displays the primitive and the edge sensitivity of the clock pin. It also provides the clock name and the clock edges definition (waveform and period).
- **`Path Group`**: shows the timing group that the path endpoint belongs to. This is usually the group defined by the destination clock, except for asynchronous timing checks (recovery/removal) which are grouped in the ***async_default*** timing group. User-defined groups can also appear here. They are convenient for reporting purpose.
- **`Path Type`**: shows the type of analysis performed on this path. “Max” indicates that the maximum delay values are used to calculate the data path delay, which corresponds to setup and recovery analysis. “Min” indicates that the minimum delay values are used to calculate the data path delay, which corresponds to hold and removal analysis. This line also shows which corner was used for the report: Slow or Fast.
- **`Requirement`**: shows the timing path requirement, which is typically one clock period for setup/recovery analysis and 0ns for hold/removal analysis when the startpoint and endpoint are controlled by the same clock. When the path is between two different clocks, the requirement corresponds to the smallest positive difference between any source and destination clock edges. This value is overridden by timing exception constraints such as multicycle path, max delay and min delay. For more information on how timing path requirement is derived from the timing constraints, see Timing Analysis in the *Vivado Design Suite User Guide Using Constraints (UG903).*
- **`Data Path Delay`**: shows accumulated delay through the logic section of the path. The clock delay is excluded unless the clock is used as a data. The type of delay corresponds to what the “Path Type” line describes.
- **`Logic Levels`**: shows the number of each type of primitives included in the data section of the path, excluding the startpoint and the endpoint cells.

- **`Clock Path Skew`**: shows the insertion delay difference between the launch edge of the source clock and the capture edge of the destination clock, plus clock pessimism correction if any.
  - **`Destination Clock Delay (DCD)`**: shows the accumulated delay from the destination clock source point to the endpoint of the path. For max delay analysis (setup/recovery), the minimum cell and net delay values are used, while for min delay analysis (hold/removal), the maximum delay values are used.
  - **`Source Clock Delay (SCD`**): shows that accumulated delay from the clock source point to the startpoint of the path. For max delay analysis (setup/recovery), the maximum cell and net delay values are used, while for min delay analysis (hold/removal), the minimum delay values are used.
  - **`Clock Pessimism Removal (CPR)`**: shows the absolute amount of extra clock skew introduced by the fact that source and destination clocks are reported with different types of delay even on their common circuitry. After removing this extra pessimism, the source and destination clocks do not have any skew on their common circuitry. For a routed design, the last common clock tree node is usually located in the routing resources used by the clock nets and is not reported in the path details.
- **`Clock Uncertainty`**: shows the total amount of possible time variation between any pair of clock edges. The uncertainty comprises the computed clock jitter (system and discrete), the phase error introduced by certain hardware primitives and any clock uncertainty specified by the user in the design constraints (set_clock_uncertainty). The user clock uncertainty is additive to the uncertainty computed by the Vivado timing engine.
  - **`Total System Jitter (TSJ)`**: shows the combined system jitter applied to both source and destination clocks. You can modify the system jitter globally by using the `set_system_jitter` XDC command. The virtual clocks are ideal and therefore do not have any system jitter. For more information on System Jitter, see the *Vivado Design Suite User Guide: Using Constraints (UG903)*.
  - **`Total Input Jitter (TIJ)`**: shows the combined input jitter of both source and destination clocks. You can define the input jitter for each primary clock individually by using the `set_input_jitter` XDC command. *The Vivado IDE timing engine computes the generated clocks input jitter based on their master clock jitter and the clocking resources traversed.* By default, the virtual clocks are ideal and therefore do not have any jitter. For more information on jitter, see the *Vivado Design Suite User Guide: Using Constraints (UG903)*.
  - **`Discrete Jitter (DJ)`**: shows the amount of jitter introduced by hardware primitives like the MMCM or the PLL. The Vivado timing engine computes this value based on the configuration of these cells.

  - **`Phase Error (PE)`**: shows the amount of phase variation between two clock signals introduced by hardware primitives like the MMCM and the PLL. The Vivado timing engine automatically provides this value and adds it to the clock uncertainty
  - **`User Uncertainty (UU)`**: shows the additional uncertainty specified by the set_clock_uncertainty XDC command. See the *Vivado Design Suite User Guide: Using Constraints (UG903*) for more information on how to use this command.

Three other lines can appear in the Timing Path Summary depending on the timing constraints, the reported path and the target device:

- **`Inter-SLR penalty`**: shows the additional margin required for safely reporting paths that cross SLR boundaries in Xilinx® 7 series SSI devices only.
- **`Input Delay`**: shows the input delay value specified by the set_input_delay XDC command on the input port. This line does not show for paths that do not start from an input port.
- **`Output Delay`**: shows the output delay value specified by the set_output_delay XDC command on the output port. This line does not show for paths that do not end to an output port.
- **`Timing Exception`**: shows the timing exception that covers the path. Only the exception with the highest precedence is displayed, as it is the only one affecting the timing path requirement. For more information on timing exceptions and their precedence rules, see the V*ivado Design Suite User Guide: Using Constraints (UG903)*.

Timing Path Details

The second half of the report provides more details on the cells, pins, ports and nets traversed by the path. It is separated in three sections:

- **`Source Clock Path`**: shows the circuitry traversed by the source clock from its source point to the startpoint of the datapath. This section does not exist for a path starting from an input port.
- **`Data Path`**: shows the circuitry traversed by the data from the startpoint to the endpoint.
- **`Destination Clock Path`**: shows the circuitry traversed by the destination clock from its source point to the datapath endpoint clock pin.

The Source Clock Path and Data Path sections work together. They are always reported with the same type of delay: max delay for setup/recovery analysis, and min delay for hold/removal analysis. They share the accumulated delay which starts at the data launch edge time, and accumulates delay through both source clock and data paths. The final accumulated delay value is called the data arrival time.

The destination clock path is always reported with the opposite delay to the source clock and data paths. Its initial accumulated delay value is the time when the data capture edge is launched on the destination clock source point. The final accumulated delay value is called the data required time.

The last three lines of the report summarize how the slack is computed.

- For max delay analysis (setup/recovery):

  `slack = data required time – data arrival time`

- For min delay analysis (hold/removal):

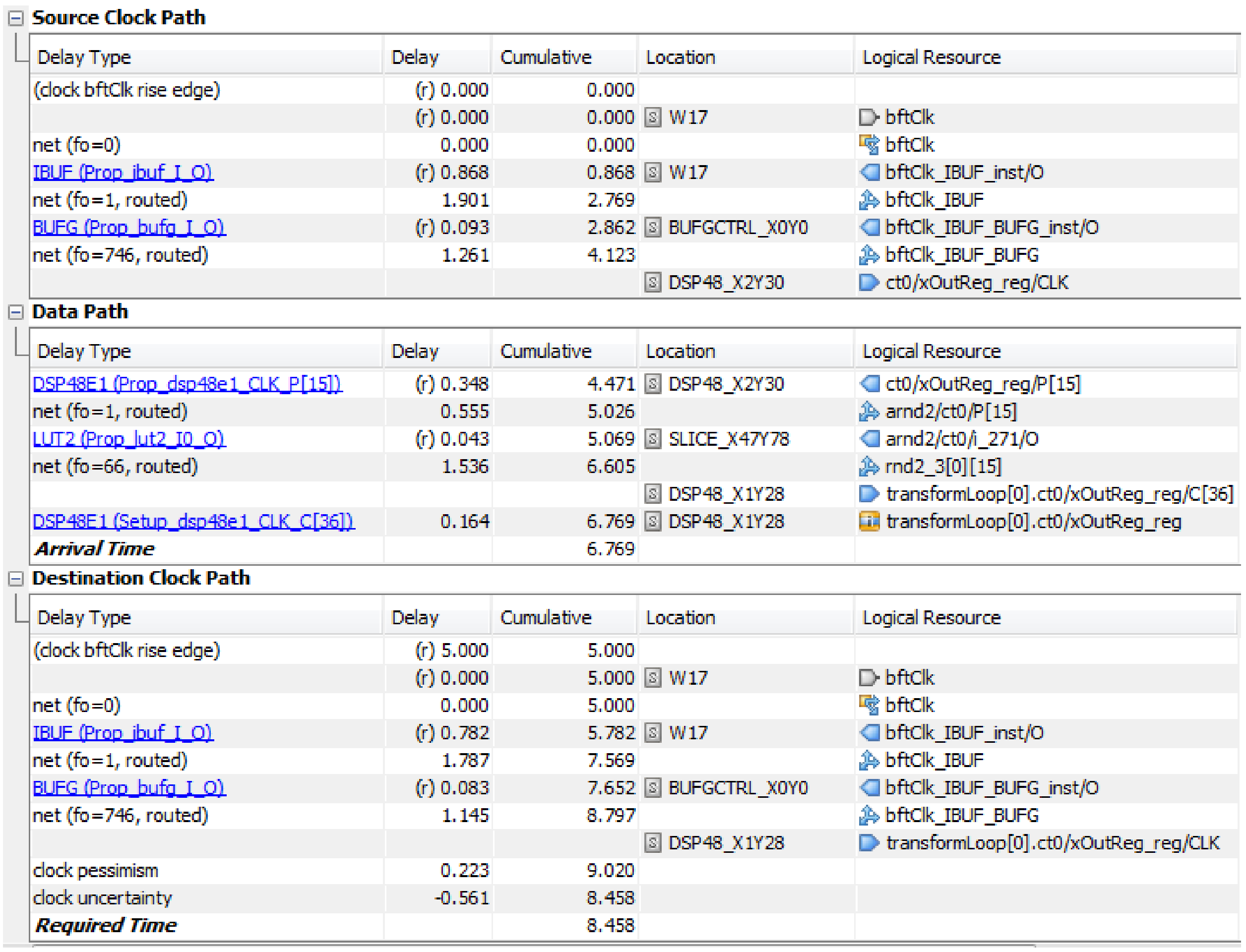
  `slack = data arrival time – data required time`

The following figure (Timing Path Details In Text Report) shows an example of the Source Clock, Data and Destination Clock Paths in the text report. The path is covered by a simple period constraint of 5ns, which is why the source clock launch edge starts at 0ns and the destination clock capture edge starts at 5ns.

```
  Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                       (clock bftClk rise edge)     0.000     0.000 r
  W17                                               0.000     0.000 r  bftClk
                       net (fo=0)                   0.000     0.000    bftClk
  W17                  IBUF (Prop_ibuf_I_O)         0.868     0.868 r  bftClk_IBUF_inst/O
                       net (fo=1, routed)           1.901     2.769    bftClk_IBUF
  BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)         0.093     2.862 r  bftClk_IBUF_BUFG_inst/O
                       net (fo=746, routed)         1.261     4.123    bftClk_IBUF_BUFG
  DSP48_X2Y30                                                     r  ct0/xOutReg_reg/CLK
  -------------------------------------------------------------------    -------------------
  DSP48_X2Y30          DSP48E1 (Prop_dsp48e1_CLK_P[15])
                                                    0.348     4.471 r  ct0/xOutReg_reg/P[15]
                       net (fo=1, routed)           0.555     5.026    arnd2/ct0/P[15]
  SLICE_X47Y78         LUT2 (Prop_lut2_I0_O)        0.043     5.069 r  arnd2/ct0/i_271/O
                       net (fo=66, routed)          1.536     6.605    rnd2_3[0][15]
  DSP48_X1Y28                                                     r  transformLoop[0].ct0/xOutReg_reg/C[36]
  DSP48_X1Y28          DSP48E1 (Setup_dsp48e1_CLK_C[36])
                                                    0.164     6.769    transformLoop[0].ct0/xOutReg_reg
  -------------------------------------------------------------------    -------------------

                       (clock bftClk rise edge)     5.000     5.000 r
  W17                                               0.000     5.000 r  bftClk
                       net (fo=0)                   0.000     5.000    bftClk
  W17                  IBUF (Prop_ibuf_I_O)         0.782     5.782 r  bftClk_IBUF_inst/O
                       net (fo=1, routed)           1.787     7.569    bftClk_IBUF
  BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)         0.083     7.652 r  bftClk_IBUF_BUFG_inst/O
                       net (fo=746, routed)         1.145     8.797    bftClk_IBUF_BUFG
  DSP48_X1Y28                                                     r  transformLoop[0].ct0/xOutReg_reg/CLK
                       clock pessimism              0.223     9.020
                       clock uncertainty           -0.561     8.458
  -------------------------------------------------------------------
                       required time                          8.458
                       arrival time                          -6.769
  -------------------------------------------------------------------
                       slack                                  1.690
```

Figure 63: Timing Path Details In Text Report

Figure 61: Timing Path Summary In Text Report shows the same example as Figure 62: Timing Path Summary in Vivado IDE in the Vivado IDE.



Source Clock Path

Delay Type	Delay	Cumulative	Location	Logical Resource
(clock bftClk rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	W17	bftClk
net (fo=0)	0.000	0.000		bftClk
IBUF (Prop_ibuf_I_O)	(r) 0.868	0.868	W17	bftClk_IBUF_inst/O
net (fo=1, routed)	1.901	2.769		bftClk_IBUF
BUFG (Prop_bufg_I_O)	(r) 0.093	2.862	BUFGCTRL_X0Y0	bftClk_IBUF_BUFG_inst/O
net (fo=746, routed)	1.261	4.123		bftClk_IBUF_BUFG
			DSP48_X2Y30	ct0/xOutReg_reg/CLK

Data Path

Delay Type	Delay	Cumulative	Location	Logical Resource
DSP48E1 (Prop_dsp48e1_CLK_P[15])	(r) 0.348	4.471	DSP48_X2Y30	ct0/xOutReg_reg/P[15]
net (fo=1, routed)	0.555	5.026		arnd2/ct0/P[15]
LUT2 (Prop_lut2_I0_O)	(r) 0.043	5.069	SLICE_X47Y78	arnd2/ct0/i_271/O
net (fo=66, routed)	1.536	6.605		rnd2_3[0][15]
			DSP48_X1Y28	transformLoop[0].ct0/xOutReg_reg/C[36]
DSP48E1 (Setup_dsp48e1_CLK_C[36])	0.164	6.769	DSP48_X1Y28	transformLoop[0].ct0/xOutReg_reg
Arrival Time		6.769		

Destination Clock Path

Delay Type	Delay	Cumulative	Location	Logical Resource
(clock bftClk rise edge)	(r) 5.000	5.000		
	(r) 0.000	5.000	W17	bftClk
net (fo=0)	0.000	5.000		bftClk
IBUF (Prop_ibuf_I_O)	(r) 0.782	5.782	W17	bftClk_IBUF_inst/O
net (fo=1, routed)	1.787	7.569		bftClk_IBUF
BUFG (Prop_bufg_I_O)	(r) 0.083	7.652	BUFGCTRL_X0Y0	bftClk_IBUF_BUFG_inst/O
net (fo=746, routed)	1.145	8.797		bftClk_IBUF_BUFG
			DSP48_X1Y28	transformLoop[0].ct0/xOutReg_reg/CLK
clock pessimism	0.223	9.020		
clock uncertainty	-0.561	8.458		
Required Time		8.458		

Figure 64: Timing Path Details in Vivado IDE

The information on the path is presented in five columns:

- `Location`: shows where the cell or port is placed on the device.
- `Delay Type`: shows the unisim primitive and the particular timing arc followed by the path. In case of a net, it shows the fanout (fo) and its status. A net can be:
    - `Unplaced`: the driver and load are not placed.
    - `Estimated`: the driver or the load or both are placed. A partially routed net is also reported as estimated.
    - `Routed`: driver and load are both placed, plus the net is fully routed.
- `Incr(ns) (text report) / Delay (IDE report)`: shows the value of the incremental delay associated to a unisim primitive timing arc or a net. It can also show of a constraint like input/output delay or clock uncertainty.
- `Path(ns) (text report) / Cumulative (IDE report)`: shows the accumulated delay after each segment of the path. On a given line, its value is the accumulated value from the previous + the incremental delay of the current line.

- **`Netlist Resource(s) (text report) / Logical Resource (IDE report)`**: shows the name of the netlist object traversed.

Each incremental delay is associated to an edge sense: **`r`** (*rising*) or **`f`** (*falling*). The initial sense of the edge is determined by the launch or capture edge selected for the path. It can be transformed by any cell along the path, depending on the nature of the timing arc. For example, a rising edge at the input of an inverter becomes a falling edge on the output. The edge sense can be very helpful to identify that an overly tight timing path requirement comes from a clock edge inversion along the source or destination clock tree.

Chapter 5

Closure Techniques

This chapter discusses techniques for timing closure including:

- Refining timing constraints
- Floorplanning
- Making the implementation tools work harder
- Automatic and manual ways to improve design timing

Check Input Constraints and Sources

Ensure that design and timing inputs are reasonable.

- Be sure that you have a good netlist.

  If you are using Vivado™ Synthesis, add synthesis-specific timing constraints to a synthesis XDC file. Synthesis is timing-driven, and can tune the gates to better meet timing.

- Manually review the clock trees in the Schematic window or the Clock Networks window.

  Be sure that the clock trees are reasonable. Designs can hit long clock skew when one BUFG drives a second BUFG or a LUT. The extra clock skew can lead to tight hold timing constraints, degrading system timing. The placer DRCs issue warnings for some clock tree issues.

- Consider the clocking resources when laying out the pinout and floorplanning the design.

  The clock regions in Xilinx® 7 series FPGFA devices support twelve global clocks per region. There are additional limitations on the placement of clock trees.

- Be sure that the clock periods are the ones the design needs to meet.

  If you are over constraining the design, the tools will try to hit artificially tight timing constraints at the cost of runtime. Overconstraining may lead to timing failures and higher power consumption. If the design is failing to meet artificially tight constraints, try the real constraints. You can modify timing constraints in the Vivado IDE without changing the placement and routing. Re-run `report_timing_summary` with the real clock periods.

- Determine if the failing paths are multi-cycle paths or false paths.

  For XDC, the tool assumes that all cross clock domain crossings are real paths. This is a change from ISE and UCF. Other paths may not be reachable due to the structure of the control logic. Enter False Path Timing constraints.
- Other control structures (for example, a clock enable driven by a state machine) generate multi-cycle paths. Data has multiple clock periods to get from the source to the destination. Enter Multi-cycle Timing constraints.
- You can re-run timing after Implementation with the improved timing constraints. You do not need to re-run implementation to see how changing the timing graph changes design timing. Review the next set of timing paths and continue to refine the timing constraints as needed.
- If you are still not meeting timing after modifying the timing constraints, re-run implementation. Implementation is timing-driven and can now focus on the real problem areas. If the original design had high Total Hold Slack (THS) in the router, you must re-run implementation.

TIP You may want to use **Save Constraints As** to create a new constraint set, preserving your original constraints.

- Review the timing path.

  Ensure that clock skew and jitter are reasonable.
- Review the logic.

  What is the logic delay compared to the period? If logic delay is a high percentage of the period you must resynthesize the design to reduce logic delay. Two ways to try to reduce logic delay are:
  - Consider re-writing the RTL to reduce route delay.

    If you see high fanout nets, consider replicating the drivers. When refactoring the high fanout net take placement into account. Each driver should communicate with localized loads rather than spread out loads.
  - If route delay is a high percentage of the timing path consider floorplanning.

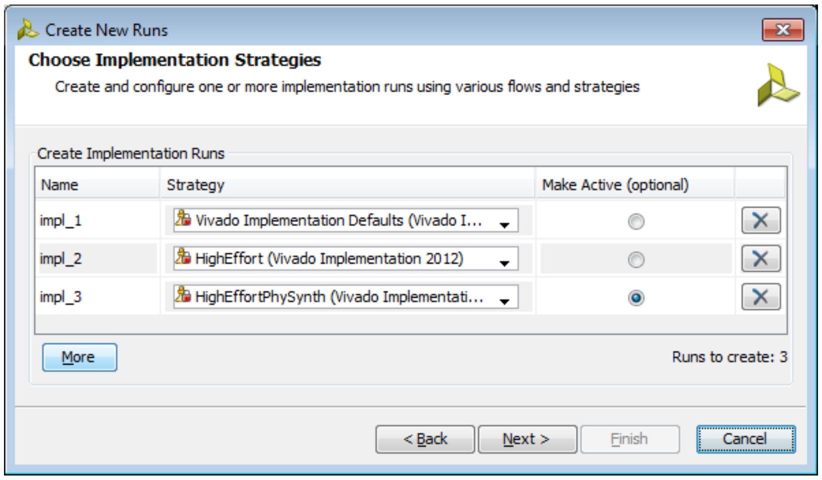
    The goal is to improve the timing of the critical paths by reducing route delay. Floorplanning does not change the logic that makes up the critical path. Review how the blocks in the design interconnect while floorplanning. You may want to restructure the RTL to better work inside the device. The synthesis tools do not know about placement. When they replicate high fanout nets, they generally do not take placement into account. Manually replicating and rewiring can lead to a better design. For example, if you have two memory interfaces that go on opposite corners of the chip, you may want to replicate the control signals in the RTL source. You may need to use synthesis attributes to stop optimization of logically equivalent registers.

Increasing Tool Effort

Ask the tools to work harder. Sometimes you can close timing on a design just by assigning more CPU cycles to implementation. Use the HighEffort Strategy. Implementation runs at a medium effort level by default. The `place_design` and `route_design` commands accept a `–effort_level high` switch. The tools will try different approaches to meet timing at the expense of runtime.

Consider using the HighEffortPhysSynth strategy. It turns up the effort level and turns on the optional `phys_opt_design` step. It works to improve timing at the expense of run time.

Use the **Flow > Create Runs** command to create and launch multiple runs using the different strategies.


Create New Runs
Choose Implementation Strategies
Create and configure one or more implementation runs using various flows and strategies
Create Implementation Runs
More
Runs to create: 3
< Back
Next >
Finish
Cancel

Name	Strategy	Make Active (optional)
impl_1	Vivado Implementation Defaults (Vivado I...	○
impl_2	HighEffort (Vivado Implementation 2012)	○
impl_3	HighEffortPhysSynth (Vivado Implementati...	●

Figure 65: Multiple Runs

Review the implementation results to find the strategy that works best for the design. Re-use the strategy for later runs.

TIP The optimal strategy may change between designs and software releases.

Manual Floorplanning

Floorplanning can help a design meet timing. Floorplan when a design:

- Does not meet timing consistently.
- Has never met timing.

Floorplanning is also helpful when you are working with design teams, and consistency is most important.

Floorplanning can improve the setup delay (TNS WNS) by reducing route delay. During implementation, the timing engine reports on Setup, Hold, and Component Switching Limits. Floorplanning can only improve Setup slack.

Manual floorplanning is easiest when the netlist has hierarchy. Design analysis is much slower when synthesis flattens the entire netlist. Set up synthesis to generate a hierarchical netlist.

- For XST, use:
    - `– keep_hierarchy=no` (default) and `–netlist_hierarchy = rebuilt` (non-default), or
    - The PlanAhead™ Defaults strategy
- For Vivado Synthesis use:
    - `–flatten_hierarchy = rebuilt`, or
    - The Vivado Synthesis Defaults strategy

Large hierarchical blocks with intertwined logical paths can be difficult to analyze. It is easier to analyze a design in which separate logical structures are in lower sub-hierarchies. Consider registering all the outputs of a hierarchical module. It is difficult to analyze the placement of paths that trace through multiple hierarchical blocks.

Floorplanning Basics

Not every design will always meet timing. You may have to guide the tools to a solution. Floorplanning allows you to guide the tools, either through high level hierarchy layout, or through detailed gate placement.

Consider floorplanning to increase performance by reducing route delay or increasing logic density on a non-critical block. Logic density is a measure of how tightly the logic is packed onto the chip.

Floorplanning can help you meet a higher clock frequency and improve consistency in the results.

There are multiple approaches to floorplanning, each with its advantages and disadvantages.

Detailed Gate-Level Floorplanning

Detailed gate-level floorplanning involves placing individual logic elements in specific sites on the device.

Advantages of Detailed Gate-Level Floorplanning

- Detailed gate-level floorplanning works with hand routing nets.
- Detailed gate-level floorplanning can extract the most performance out of the device.

Disadvantages of Detailed Gate-Level Floorplanning

- Detailed gate-level floorplanning is time consuming.
- Detailed gate-level floorplanning requires extensive knowledge of the device and design.
- Detailed gate-level floorplanning may need to be redone if the netlist changes.

RECOMMENDED Consider detailed gate-level floorplanning as a last resort.

Information Re-Use

Re-use information from a design that met timing.

Use this flow if the design does not consistently meet timing.

To re-use information:

- 1. Open two implementation runs:
  - One for a run that *is* meeting timing.
  - One for a run that is *not* meeting timing.

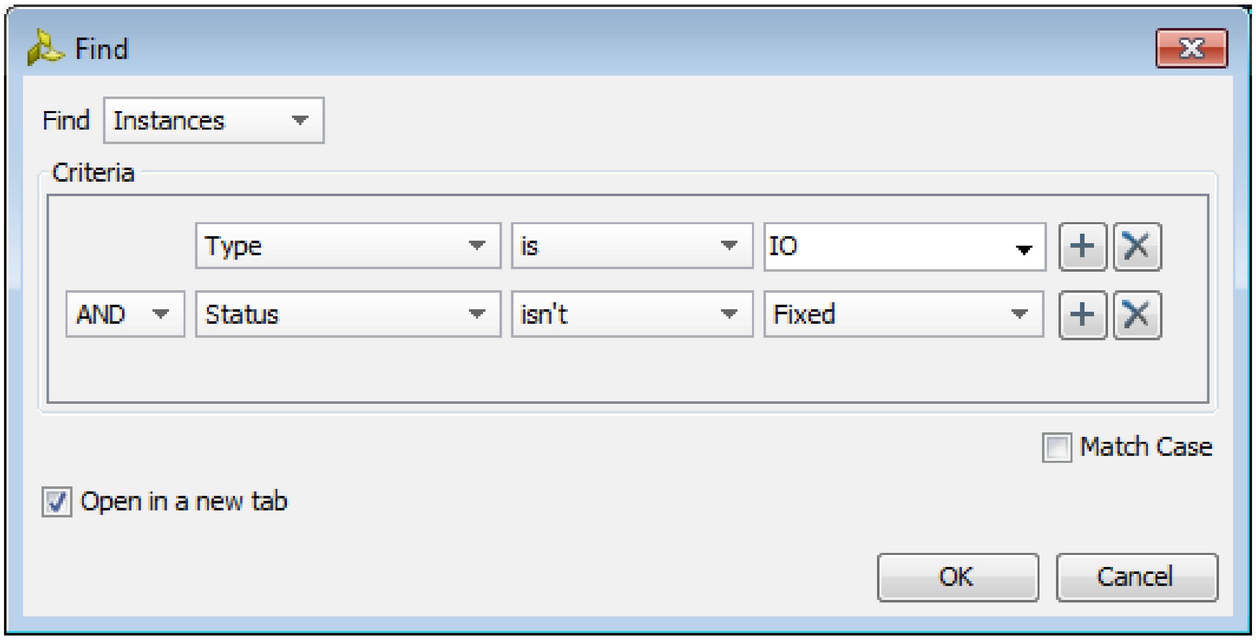
  TIP On a computer with multiple monitors, select **Open Implementation in New Window** to open a design in a new window. For more information, see Multiple Windows.
- 2. Look for the differences between the two designs.
  - Identify some failing timing paths from `report_timing_summary`.
  - On the design that is meeting timing, use `report_timing` in `min_max` mode to time those same paths on the design that meets timing.
- 3. Compare the timing results:
  - `-clock skew`
  - logic delay
  - placement
  - `route_delays`

4. If there are differences in the amount of logic delay between path end points, revisit the synthesis runs.

Review Element Placement

Review the placement of the elements in the design.

Compare two IO reports to review the IO placement and IO standards. Make sure all the IOs are placed. A simple search finds all IOs without fixed placement as shown in the following figure (IO Is Not Fixed).


Find

Find: Instances

Criteria

Type is IO

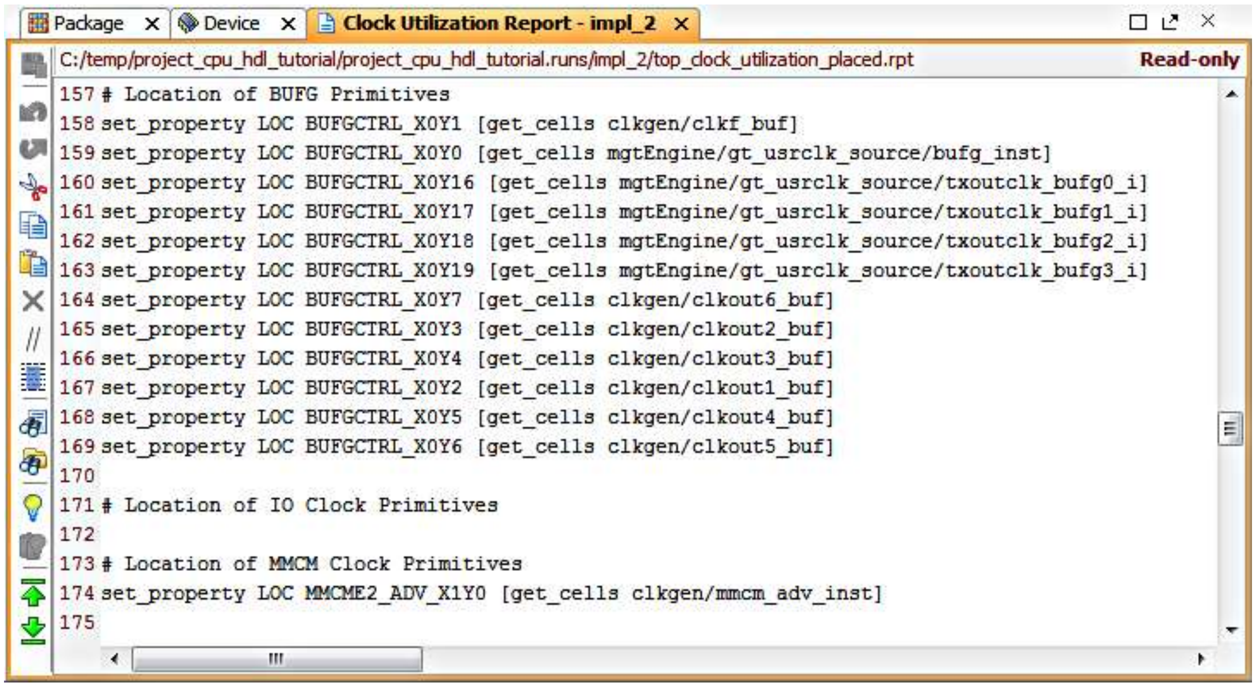
AND Status isn't Fixed

☐ Match Case

☑ Open in a new tab

Figure 66: IO Is Not Fixed

Consider placing all the clock primitives based on the run that met timing. The Clock Utilization Report lists the placement of the clock tree drivers, as shown in the following figure (Figure Clock Locations).

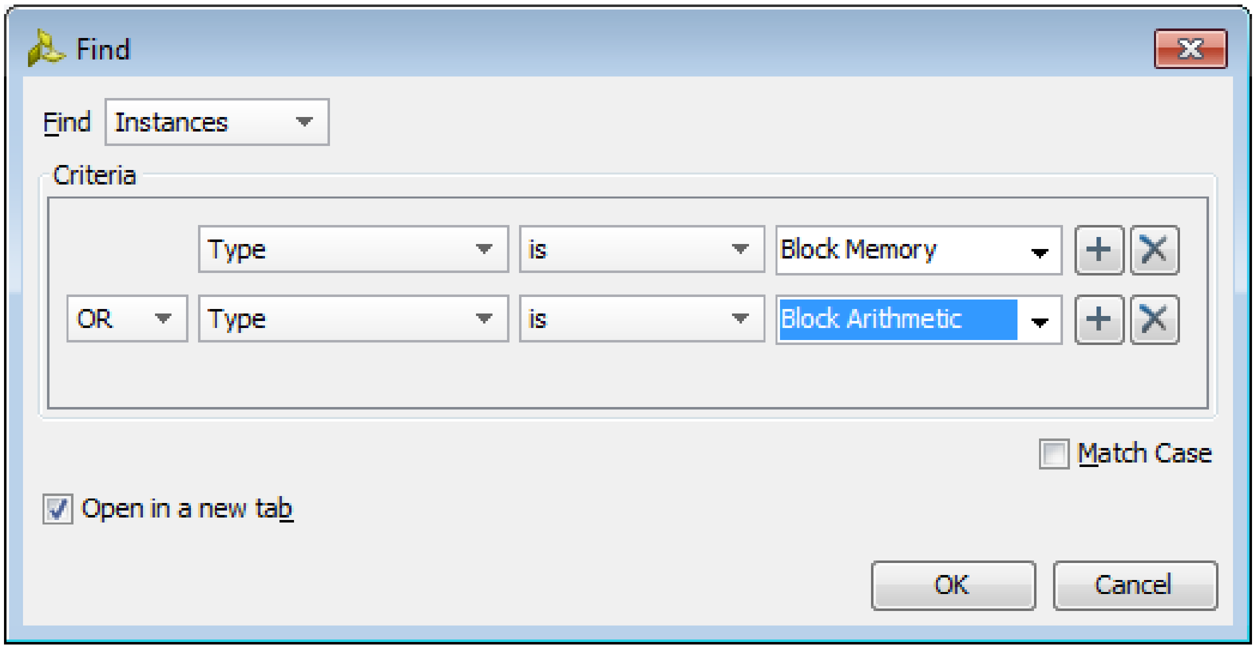


```
C:/temp/project_cpu_hdl_tutorial/project_cpu_hdl_tutorial.runs/impl_2/top_clock_utilization_placed.rpt
# Location of BUFG Primitives
set_property LOC BUFGCTRL_X0Y1 [get_cells clkgen/clkf_buf]
set_property LOC BUFGCTRL_X0Y0 [get_cells mgtEngine/gt_usrclk_source/bufg_inst]
set_property LOC BUFGCTRL_X0Y16 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg0_i]
set_property LOC BUFGCTRL_X0Y17 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg1_i]
set_property LOC BUFGCTRL_X0Y18 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg2_i]
set_property LOC BUFGCTRL_X0Y19 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg3_i]
set_property LOC BUFGCTRL_X0Y7 [get_cells clkgen/clkout6_buf]
set_property LOC BUFGCTRL_X0Y3 [get_cells clkgen/clkout2_buf]
set_property LOC BUFGCTRL_X0Y4 [get_cells clkgen/clkout3_buf]
set_property LOC BUFGCTRL_X0Y2 [get_cells clkgen/clkout1_buf]
set_property LOC BUFGCTRL_X0Y5 [get_cells clkgen/clkout4_buf]
set_property LOC BUFGCTRL_X0Y6 [get_cells clkgen/clkout5_buf]

# Location of IO Clock Primitives

# Location of MMCM Clock Primitives
set_property LOC MMCME2_ADV_X1Y0 [get_cells clkgen/mmcm_adv_inst]

```

Figure 67: Figure Clock Locations

The LOC constraints can easily copied into your XDC constraints file.

Many designs have met timing by reusing the placement of the Block RAMs and DSPs. Use the **Edit > Find** command to list the instances.



Find: Instances

Criteria

Type is Block Memory

OR Type is Block Arithmetic

Match Case [ ]

Open in a new tab [✓]

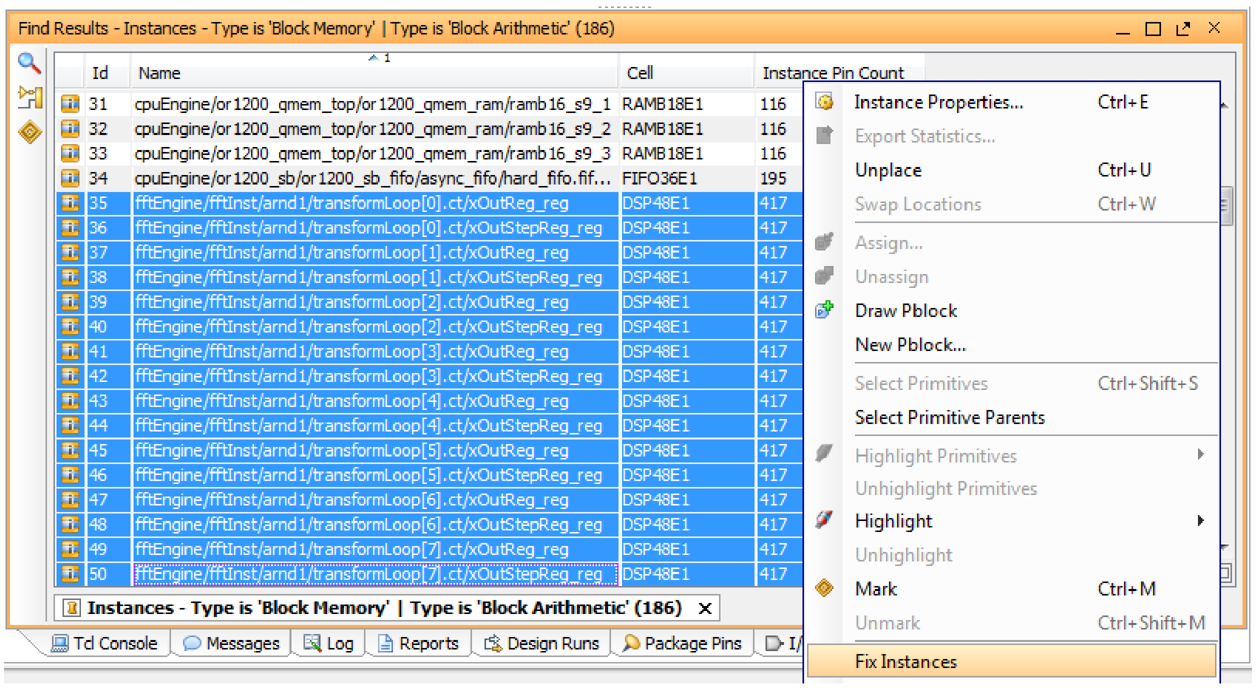
Figure 68: DSP or RAM

Fix the logic to add the placement constraints to your XDC.

1. Select the macros from the find results.

2. Right click.

3. Select **Fix Instances**.



Find Results - Instances - Type is 'Block Memory' | Type is 'Block Arithmetic' (186)

Id	Name	Cell	Instance Pin Count
31	cpuEngine/or1200_qmem_top/or1200_qmem_ram/ramb16_s9_1	RAMB18E1	116
32	cpuEngine/or1200_qmem_top/or1200_qmem_ram/ramb16_s9_2	RAMB18E1	116
33	cpuEngine/or1200_qmem_top/or1200_qmem_ram/ramb16_s9_3	RAMB18E1	116
34	cpuEngine/or1200_sb/or1200_sb_fifo/async_fifo/hard_fifo.fif...	FIFO36E1	195
35	fftEngine/fftInst/arnd1/transformLoop[0].ct/xOutReg_reg	DSP48E1	417
36	fftEngine/fftInst/arnd1/transformLoop[0].ct/xOutStepReg_reg	DSP48E1	417
37	fftEngine/fftInst/arnd1/transformLoop[1].ct/xOutReg_reg	DSP48E1	417
38	fftEngine/fftInst/arnd1/transformLoop[1].ct/xOutStepReg_reg	DSP48E1	417
39	fftEngine/fftInst/arnd1/transformLoop[2].ct/xOutReg_reg	DSP48E1	417
40	fftEngine/fftInst/arnd1/transformLoop[2].ct/xOutStepReg_reg	DSP48E1	417
41	fftEngine/fftInst/arnd1/transformLoop[3].ct/xOutReg_reg	DSP48E1	417
42	fftEngine/fftInst/arnd1/transformLoop[3].ct/xOutStepReg_reg	DSP48E1	417
43	fftEngine/fftInst/arnd1/transformLoop[4].ct/xOutReg_reg	DSP48E1	417
44	fftEngine/fftInst/arnd1/transformLoop[4].ct/xOutStepReg_reg	DSP48E1	417
45	fftEngine/fftInst/arnd1/transformLoop[5].ct/xOutReg_reg	DSP48E1	417
46	fftEngine/fftInst/arnd1/transformLoop[5].ct/xOutStepReg_reg	DSP48E1	417
47	fftEngine/fftInst/arnd1/transformLoop[6].ct/xOutReg_reg	DSP48E1	417
48	fftEngine/fftInst/arnd1/transformLoop[6].ct/xOutStepReg_reg	DSP48E1	417
49	fftEngine/fftInst/arnd1/transformLoop[7].ct/xOutReg_reg	DSP48E1	417
50	fftEngine/fftInst/arnd1/transformLoop[7].ct/xOutStepReg_reg	DSP48E1	417

Instances - Type is 'Block Memory' | Type is 'Block Arithmetic' (186)

Figure 69: Selecting the Logic to Fix

RECOMMENDED Analyze the placement based on hierarchy name and highlight before fixing the placement.

It is fairly easy to re-use the placement of:

- IOs
- Global Clock Resources
- BlockRam
- DSP macros

Re-using this placement helps to reduce the variability in results from one netlist revision to the next. These primitives generally have stable names. The placement is usually easy to maintain.

TIP Do not reuse the placement of general slice logic. Do not re-use the placement for sections of the design that are likely to change.

Hand Floorplanning

Consider gate floorplanning for a design that has never met timing, and in which changing the netlist or the constraints are not good options.

RECOMMENDED Try hierarchical floorplanning before considering gate level floorplanning.

Hierarchical Floorplanning

Hierarchical floorplanning allows you to place one or more levels of hierarchy in a region on the chip. This region provides guidance to the placer. The placer does the detailed placement.

Hierarchical floorplan creation is fast compared to gate floorplanning. A good floorplan can improve timing. The floorplan is resistant to design change.

The level of hierarchy acts as a container for all the gates. It will generally work if the netlist changes.

Hand Gate Placement

Hand gate placement can obtain the best performance from a device. When using this technique, designers generally use it only on a small block of the design. They may hand place a small amount of logic around a high speed IO interface, or hand place Block RAMs and DSPs. Hand placement can be slow.

All floorplanning techniques can require significant engineering time. They may require floorplan iterations. If any of the gate names change, the floorplan constraints must be updated.

In hierarchical floorplanning:

- Identify the lower levels of hierarchy that contain the critical path.
- Use the top level floorplan to identify where to place them.
- Implementation places individual gates.
- Has comprehensive knowledge of the gates and timing paths.
- Generally does a good job of fine grain placement.

When floorplanning, you should have an idea of final pinout. It is useful to have the IOs fixed. The IOs can provide anchor points for starting the floorplan. Logic that communicates to IOs migrates towards the fixed pins.

TIP Place blocks that communicate with IOs near their IOs. If the pinout is pulling a block apart, consider pinout or RTL modification.

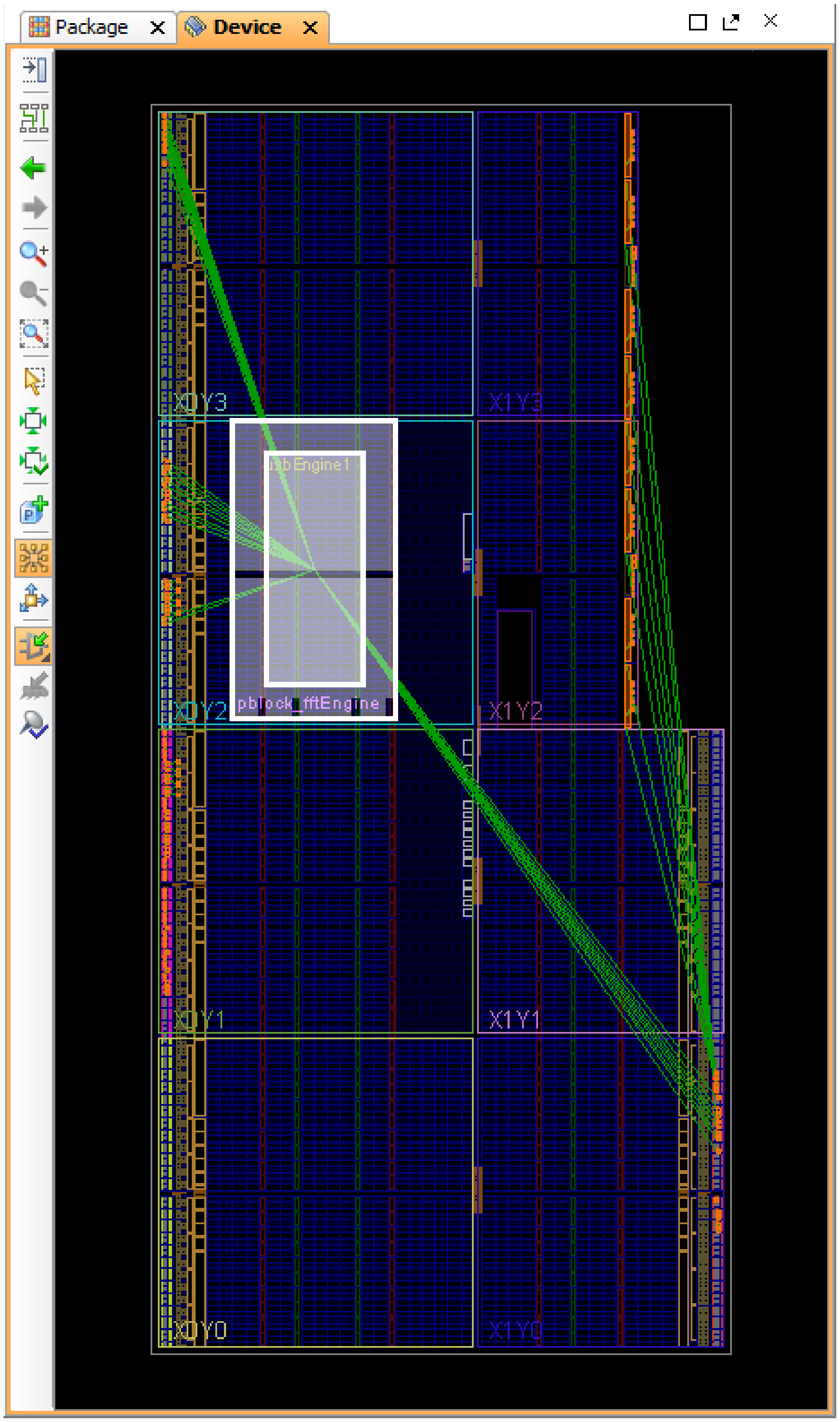


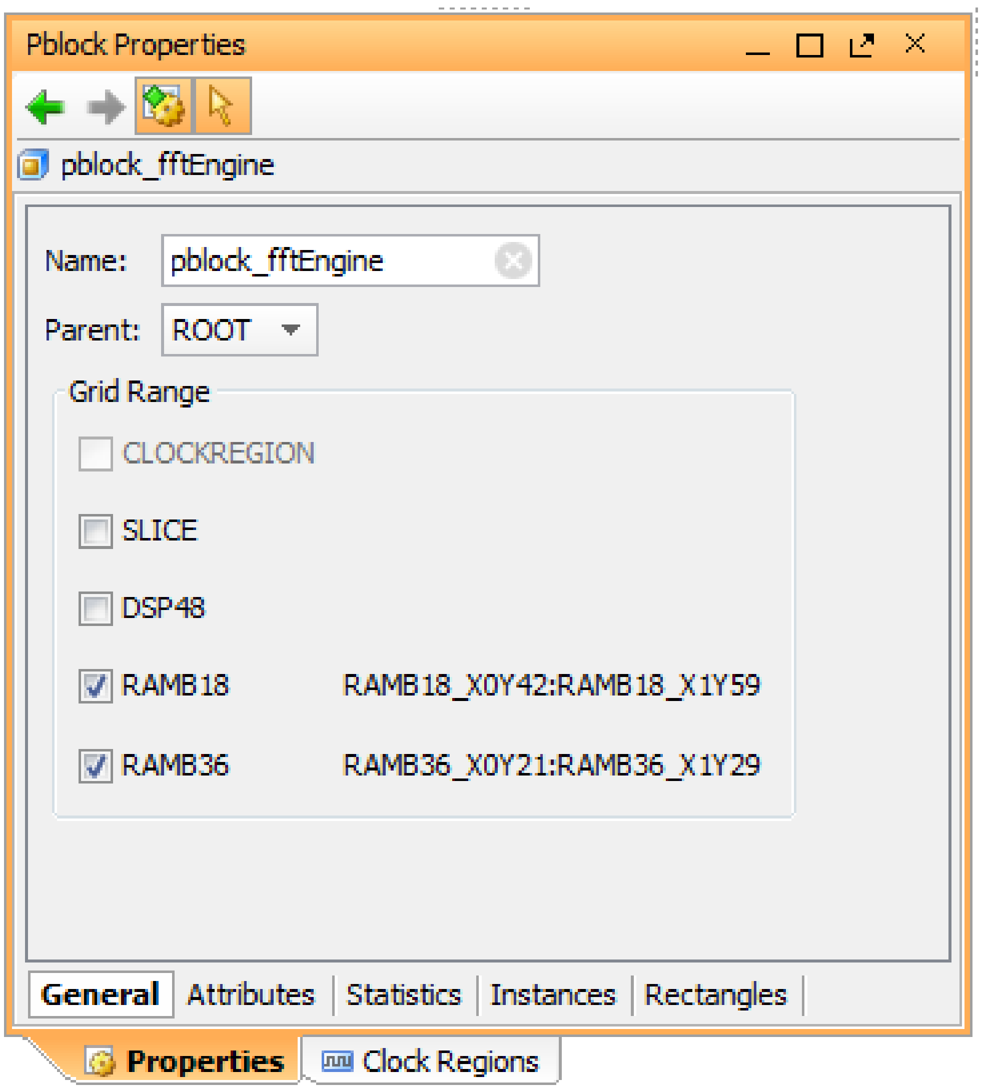
Figure 70: IO Components Pulling Design Apart

The floorplan shown in the figure above (IO Components Pulling Design Apart) may not help timing. Consider splitting the block apart, changing the source code, or constraining only the Block RAMs and DSPs.

The Pblock above is represented by the XDC constraints:

```
create_Pblock Pblock_fftEngine
add_cells_to_Pblock [get_Pblocks Pblock_fftEngine] [get_cells -quiet [list
usbEngine1]]
resize_Pblock [get_Pblocks Pblock_fftEngine] -add
{SLICE_X8Y105:SLICE_X23Y149}
resize_Pblock [get_Pblocks Pblock_fftEngine] -add {DSP48_X0Y42:DSP48_X1Y59}
resize_Pblock [get_Pblocks Pblock_fftEngine] -add {RAMB18_X0Y42:RAMB18_X1Y59}
resize_Pblock [get_Pblocks Pblock_fftEngine] -add {RAMB36_X0Y21:RAMB36_X1Y29}
```

There is one line assigning the level of hierarchy to the Pblock. There are four resource types each with its own grid. Logic that is not constrained by a grid can go anywhere in the device. To constrain just the Block RAMs in the level of hierarchy, disable the other Pblock grids.



Name: pblock_fftEngine

Parent: ROOT

Grid Range

☐ CLOCKREGION	
☐ SLICE	
☐ DSP48	
✓ RAMB18	RAMB18_X0Y42:RAMB18_X1Y59
✓ RAMB36	RAMB36_X0Y21:RAMB36_X1Y29

Figure 71: Pblock Grids

The resulting XDC constraints the simplified Pblock:

```
create_Pblock Pblock_fftEngine
add_cells_to_Pblock [get_Pblocks Pblock_fftEngine] [get_cells -quiet [list
usbEngine1]]
resize_Pblock [get_Pblocks Pblock_fftEngine] -add {RAMB18_X0Y42:RAMB18_X1Y59}
resize_Pblock [get_Pblocks Pblock_fftEngine] -add {RAMB36_X0Y21:RAMB36_X1Y29}
```

The Block RAMs will be constrained in the device, but the slice logic will be free to migrate to the IOs.

TIP When floorplanning logic, be careful not to floorplan hierarchy in such a manner that it crosses the central config block.

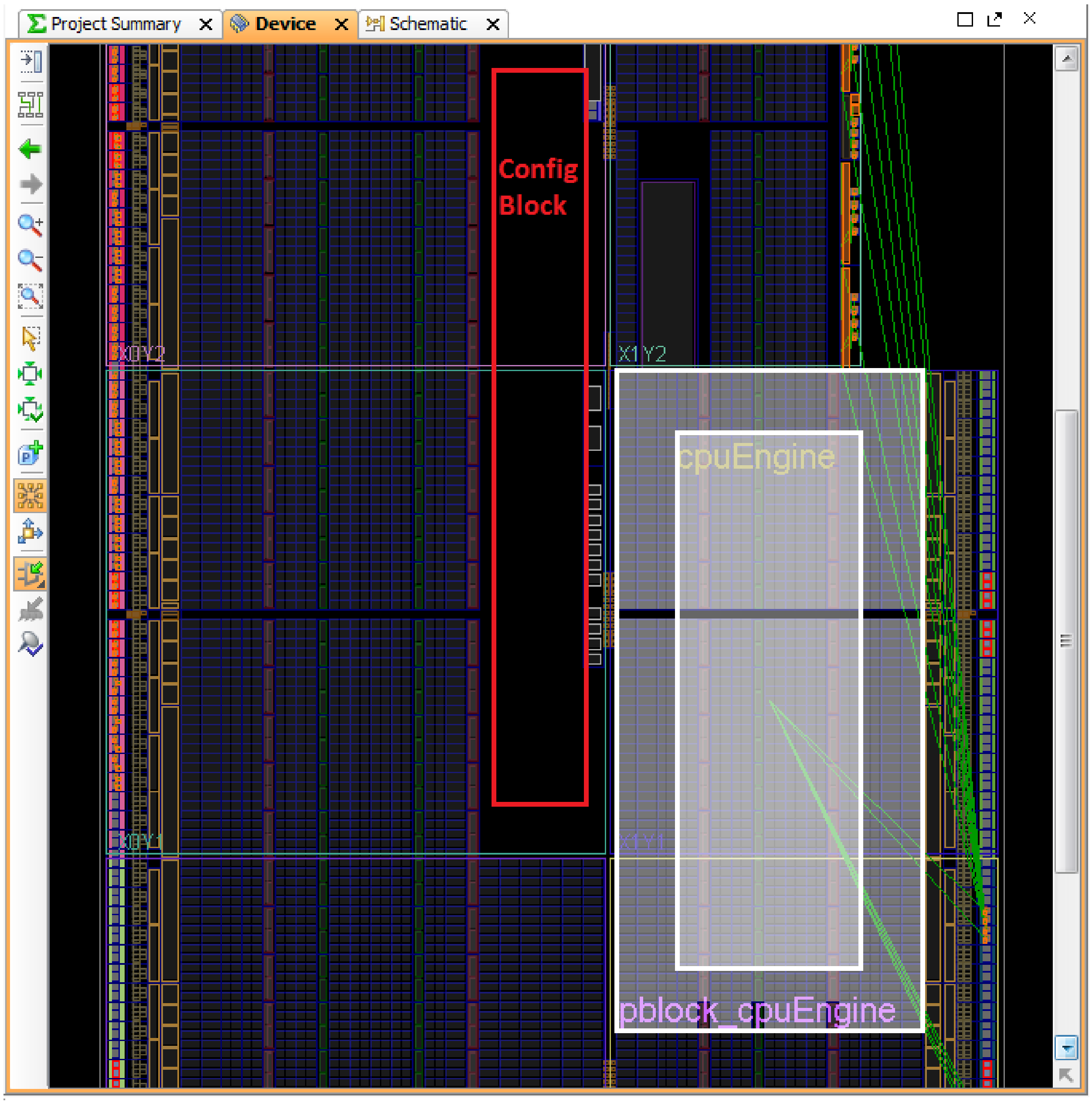


Figure 72: Avoiding the Config Block

RECOMMENDED Crossing the config block has a small routing penalty. Keep the Pblocks to one side or the other of the block.

Stacked Silicon Interconnect (SSI)

There are extra considerations for Stacked Silicon Interconnect (SSI) parts. The SSI parts are made of multiple die or Super Logic Regions (SLRs), joined together by an interposer. The interposer connections are called Super Long Lines (SLLs). There is some delay to cross from one SLR to another.

Keep the SLRs in mind when structuring the design, generating a pinout, and floorplanning. Minimize SLL crossings by keeping critical timing paths inside an SLR.

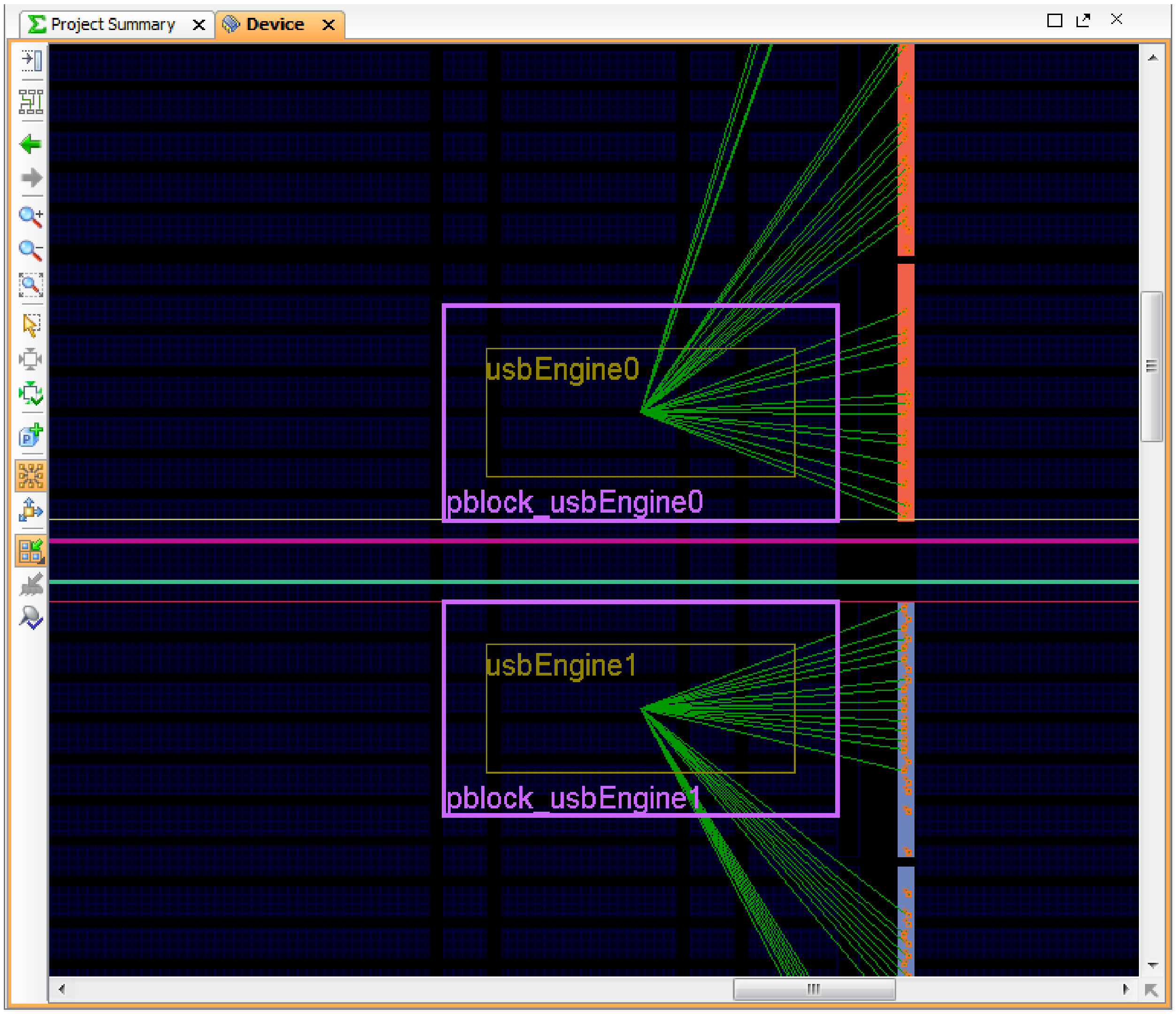


Figure 73: Minimize SLR Crossings

The IOs are placed in the same SLR as the relevant IO interface. You should also consider clock placement when laying out logic for SSI parts.

RECOMMENDED Let the placer try an automatic placement of the logic into the SSI parts before doing extensive partitioning. Analyzing the automatic placement may suggest floorplanning approaches you were not considering.

Modify Routing

Depending on results, you may want to make modifications to routes. The Device View allows you to Unroute, Route, and Fix Routing on any individual net. These commands are available from the context menu on a net.

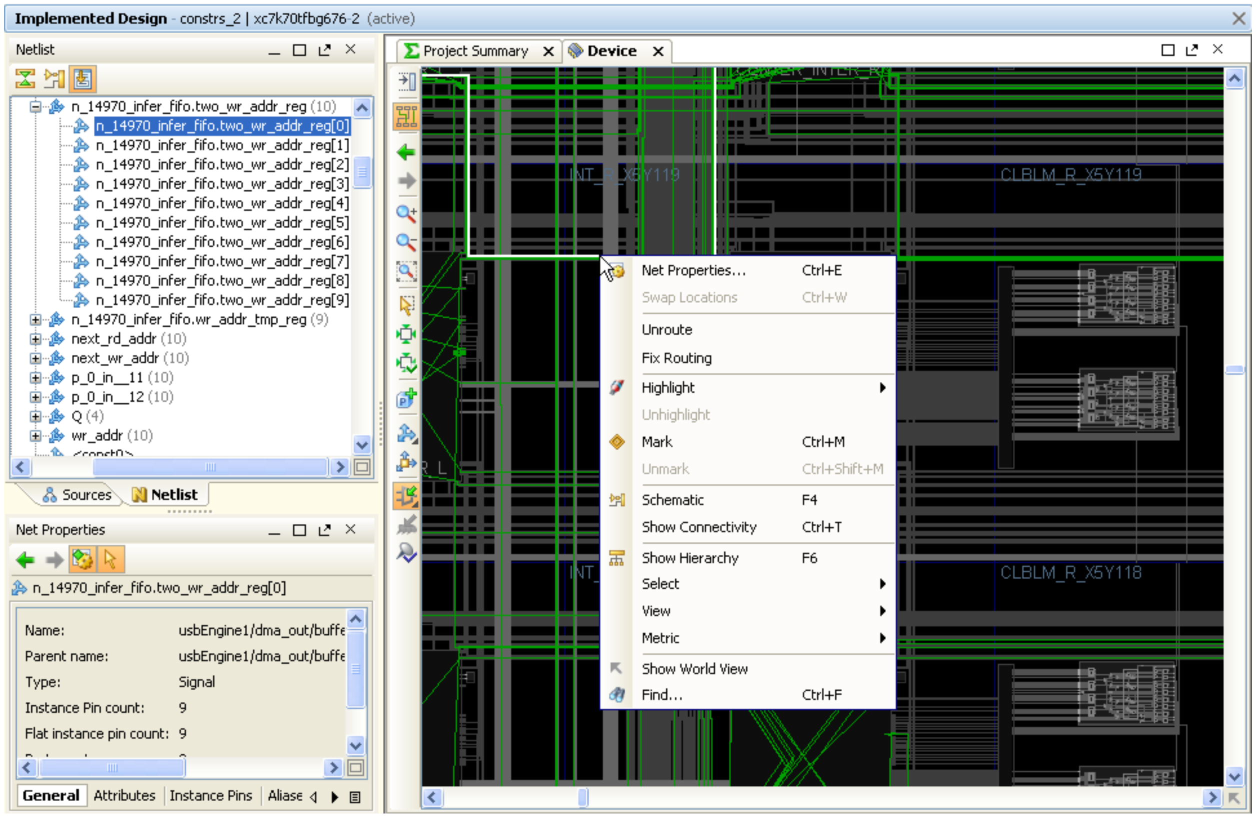
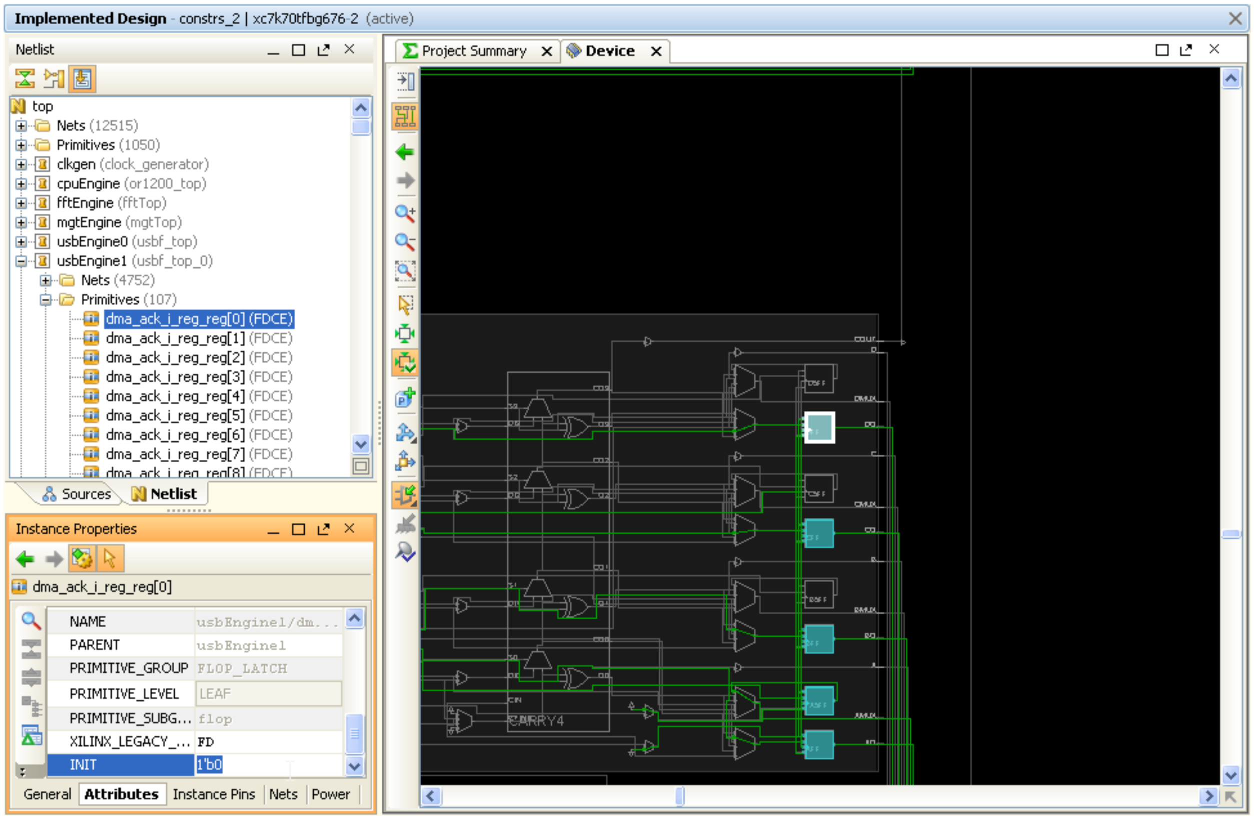

Implemented Design - constrs_2 | xc7k70tfbg676-2 (active)
Netlist
n_14970_infer_fifo.two_wr_addr_reg (10)
n_14970_infer_fifo.two_wr_addr_reg[0]
n_14970_infer_fifo.two_wr_addr_reg[1]
n_14970_infer_fifo.two_wr_addr_reg[2]
n_14970_infer_fifo.two_wr_addr_reg[3]
n_14970_infer_fifo.two_wr_addr_reg[4]
n_14970_infer_fifo.two_wr_addr_reg[5]
n_14970_infer_fifo.two_wr_addr_reg[6]
n_14970_infer_fifo.two_wr_addr_reg[7]
n_14970_infer_fifo.two_wr_addr_reg[8]
n_14970_infer_fifo.two_wr_addr_reg[9]
n_14970_infer_fifo.wr_addr_tmp_reg (9)
next_rd_addr (10)
next_wr_addr (10)
p_0_in__11 (10)
p_0_in__12 (10)
Q (4)
wr_addr (10)
Sources
Netlist
Net Properties
n_14970_infer_fifo.two_wr_addr_reg[0]
Name: usbEngine1/dma_out/buffe
Parent name: usbEngine1/dma_out/buffe
Type: Signal
Instance Pin count: 9
Flat instance pin count: 9
General
Attributes
Instance Pins
Aliase
Project Summary
Device
INT_L_X5Y119
CLBLM_R_X5Y119
CLBLM_R_X5Y118
Net Properties... Ctrl+E
Swap Locations Ctrl+W
Unroute
Fix Routing
Highlight
Unhighlight
Mark Ctrl+M
Unmark Ctrl+Shift+M
Schematic F4
Show Connectivity Ctrl+T
Show Hierarchy F6
Select
View
Metric
Show World View
Find... Ctrl+F

Figure 74: Net Context

Since only the commands applicable to a net are shown, not all commands are shown for all nets.

Modify Logic

Properties on logical objects that are not Read Only can be modified after Implementation in the Vivado IDE graphical user interface as well as Tcl. Use the Attributes Tab in the Properties window.



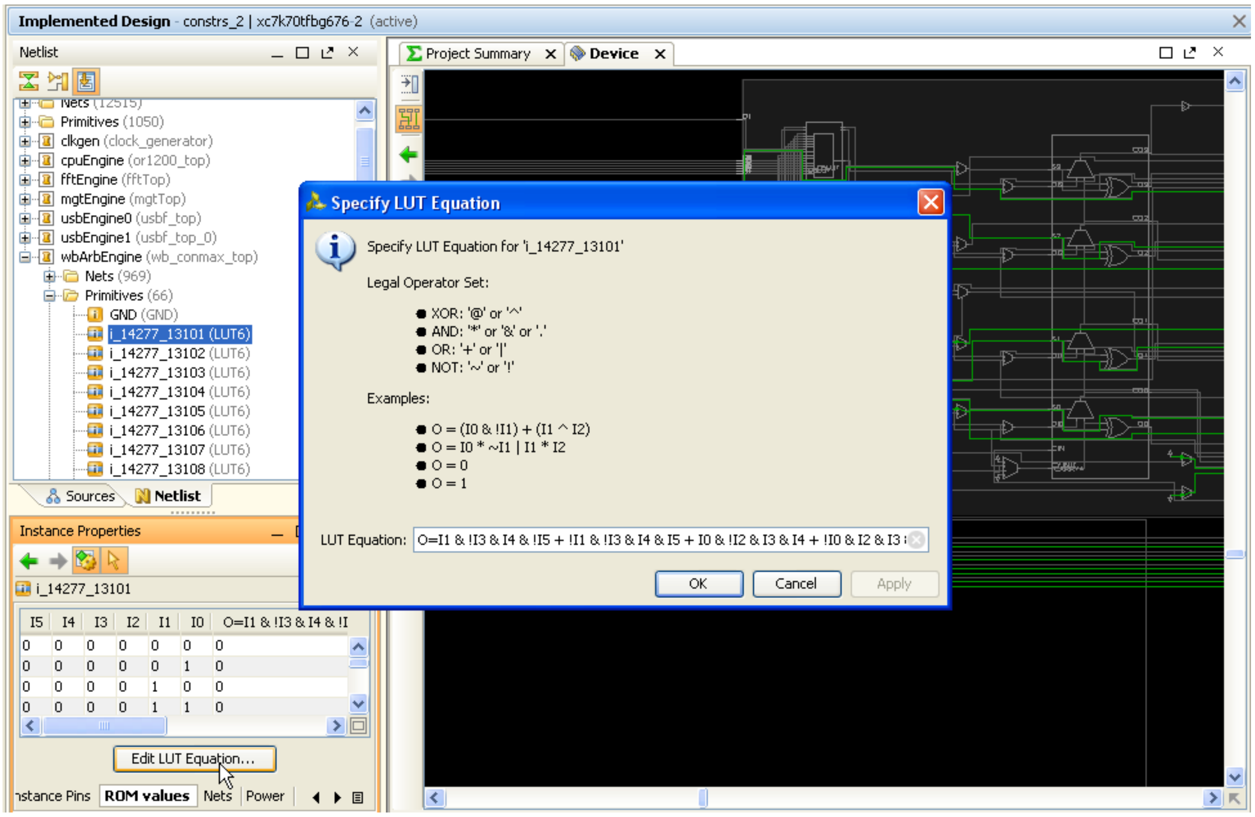
top
- Nets (12515)
- Primitives (1050)
- clkgen (clock_generator)
- cpuEngine (or1200_top)
- fftEngine (fftTop)
- mgtEngine (mgtTop)
- usbEngine0 (usbf_top)
- usbEngine1 (usbf_top_0)
  - Nets (4752)
  - Primitives (107)
    - dma_ack_i_reg_reg[0] (FDCE)
    - dma_ack_i_reg_reg[1] (FDCE)
    - dma_ack_i_reg_reg[2] (FDCE)
    - dma_ack_i_reg_reg[3] (FDCE)
    - dma_ack_i_reg_reg[4] (FDCE)
    - dma_ack_i_reg_reg[5] (FDCE)
    - dma_ack_i_reg_reg[6] (FDCE)
    - dma_ack_i_reg_reg[7] (FDCE)
    - dma_ack_i_reg_reg[8] (FDCE)

dma_ack_i_reg_reg[0]

NAME	usbEngine1/dm...
PARENT	usbEngine1
PRIMITIVE_GROUP	FLOP_LATCH
PRIMITIVE_LEVEL	LEAF
PRIMITIVE_SUBG...	flop
XILINX_LEGACY_...	FD
INIT	1'b0

Figure 75: Property Modify

These properties can include everything from Block RAM INITs to the clock modifying properties on MMCMs. There is also a special dialog box to set or modify INIT on LUT objects. This dialog box allows you to specify the LUT equation and have the tool determine the appropriate INIT.



Specify LUT Equation for 'i_14277_13101'

Legal Operator Set:

- XOR: '@' or '^'
- AND: '*' or '&' or '.'
- OR: '+' or '|'
- NOT: '~' or '!'

Examples:

- O = (I0 & !I1) + (I1 ^ I2)
- O = I0 * ~I1 | I1 * I2
- O = 0
- O = 1

LUT Equation: O=I1 & !I3 & I4 & !I5 + !I1 & !I3 & I4 & I5 + I0 & !I2 & I3 & I4 + !I0 & I2 & I3 &

I5	I4	I3	I2	I1	I0	O=I1 & !I3 & I4 & !I
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0

Figure 76: Equation Editor

Using Modifications

Once modified, a checkpoint is the recommended way to capture the design in memory. Since logical changes are not back annotated to the logical design, logical modifications must also be done in Source or XDC in order for them to impact the next run.

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx® Support website at:

www.xilinx.com/support

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this guide:

Vivado Design Suite 2012.2 Documentation,
www.xilinx.com/support/documentation/dt_vivado_vivado2012-2.htm