See all versions
of this document

Vivado Design Suite User Guide

Design Analysis and Closure Techniques

UG906 (v2023.1) May 16, 2023

AMD Adaptive Computing is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.



Table of Contents

Chapter 1: Introduction **6**
Navigating Content by Design Process 6

Chapter 2: Interactive Design Analysis in the IDE **7**
Using the Design Runs Window 7
Logic Analysis in the IDE 8
Methodology Analysis 20
Placement Analysis 21
Routing Analysis 30

Chapter 3: Viewing Reports and Messages **36**
Viewing and Managing Messages in the IDE 36
Vivado Generated Messages 39
Generating and Waiving Design Checks 42
Configurable Report Strategies 57

Chapter 4: Timing Analysis **63**
Terminology 63
Timing Paths 64
Timing Analysis Key Concepts 67
Reading a Timing Path Report 78
Verifying Timing Signoff 86

Chapter 5: Reports **88**
General Reports 88
Timing Reports 122
Design Closure Reports 210

Chapter 6: Synthesis Analysis and Closure Techniques **253**
Using the Elaborated View to Optimize the RTL 253
Decomposing Deep Memory Configurations for Balanced Power and Performance 256
Optimizing RAMB Utilization when Memory Depth is not a Power of 2 259

Optimizing RAMB Input Logic to Allow Output Register Inference....261
Improving Critical Logic on RAMB Outputs....265

Chapter 7: Implementation Analysis and Closure Techniques....270
Intelligent Design Runs....270
QoR Suggestions....281
Strategy Suggestions....293
Floorplanning....295
Determining if Hold-Fixing is Negatively Impacting the Design....310

Chapter 8: Performing NoC Quality of Service Analysis in Versal Devices....313
Vivado NoC QoS Reporting Examples....314

Appendix A: Timing Methodology Checks....317
TIMING-1: Invalid Clock Waveform on Clock Modifying Block....317
TIMING-2: Invalid Primary Clock Source Pin....319
TIMING-3: Invalid Primary Clock on Clock Modifying Block....320
TIMING-4: Invalid Primary Clock Redefinition on a Clock Tree....321
TIMING-5: Invalid Waveform Redefinition on a Clock Tree....323
TIMING-6: No Common Primary Clock Between Related Clocks....324
TIMING-7: No Common Node Between Related Clocks....325
TIMING-8: No Common Period Between Related Clocks....326
TIMING-9: Unknown CDC Logic....327
TIMING-10: Missing Property on Synchronizer....328
TIMING-11: Inappropriate Max Delay with Datapath Only Option....329
TIMING-12: Clock Reconvergence Pessimism Removal Disabled....330
TIMING-13: Timing Paths Ignored Due to Path Segmentation....331
TIMING-14: LUT on the Clock Tree....331
TIMING-15: Large Hold Violation on Inter-Clock Path....332
TIMING-16: Large Setup Violation....333
TIMING-17: Non-Clocked Sequential Cell....333
TIMING-18: Missing Input or Output Delay....334
TIMING-19: Inverted Generated Clock Waveform on ODDR....334
TIMING-20: Non-Clocked Latch....335
TIMING-21: Invalid COMPENSATION Property on MMCM....335
TIMING-22: Missing External Delay on MMCM....336
TIMING-23: Combinational Loop Found....336
TIMING-24: Overridden Max Delay Datapath Only....337

TIMING-25: Invalid Clock Waveform on Gigabit Transceiver (GT)..... 338
TIMING-26: Missing Clock on Gigabit Transceiver (GT).....338
TIMING-27: Invalid Primary Clock on Hierarchical Pin..... 339
TIMING-28: Auto-Derived Clock Referenced by a Timing Constraint..... 340
TIMING-29: Inconsistent Pair of Multicycle Paths.....340
TIMING-30: Sub-Optimal Master Source Pin Selection for Generated Clock.....341
TIMING-31: Multicycle Path on Phase-Shifted Clock.....341
TIMING-32: Bus Skew Constraint Applied on Too Many Signals.....342
TIMING-33: Invalid Bus Skew Constraint on Safely Timed Paths.....343
TIMING-34: Bus Skew Constraint with Unrealistic Value.....343
TIMING-35: No Common Node in Paths with the Same Clock..... 344
TIMING-36: Invalid Generated Clock due to No Edge Propagation.....345
TIMING-37: Bus Skew Constraint Applied on Signals with Fanout..... 345
TIMING-38: Bus Skew Constraint Applied on Multiple Clocks..... 346
TIMING-39: Invalid Bus Skew Constraint on Paths that have Too Many Levels of Logic..... 346
TIMING-40: Max Skew Violation between OSERDESE3 CLK and CLKDIV Pins Crossing SLRs..... 347
TIMING-41: Invalid Forwarded Clock Defined on an Internal Pin.....347
TIMING-42: Path Segmentation Detected in the Clock Tree.....348
TIMING-43: Min Period or Min Pulse Width Violation on Gigabit Transceiver (GT).....348
TIMING-44: Unreasonable User Intra-Clock Uncertainty.....349
TIMING-45: Unreasonable User Inter-Clock Uncertainty.....349
TIMING-46: Multicycle Path with Tied CE Pins.....350
TIMING-47: False Path, Asynchronous Clock Group or Max Delay Datapath Only Constraint between Synchronous Clocks 352
TIMING-48: Max Delay Datapath Only Constraint on Latch Input.....354
TIMING-49: Unsafe Enable or Reset Topology from Parallel BUFGCE_DIV.....354
TIMING-50: Unrealistic Path Requirement between Same-Level Latches..... 356
TIMING-51: No Common Phase between Related Clocks from Parallel CMBs 357
TIMING-52: No Common Phase between Related Clocks from Spread Spectrum MMCM.....357
TIMING-53: No Common Phase between Related Clocks from DPLL 358
TIMING-54: Scoped False Path, Clock Group, or Max Delay Datapath Only Constraint between Clocks 359
TIMING-55: Multiple Clocks Reaching a CMB Deskew Pin..... 359
TIMING-56: Missing Logically or Physically Excluded Clock Groups Constraint.....360

TIMING-57: Unsupported Configuration with PHASESHIFT_MODE and Digital Deskew....361

Appendix B: Report QoR Suggestion RTL Code Change Example....362

RQS_TIMING-201: Add an Output Register to RAM....362
RQS_TIMING-202: Add Extra Pipelining to Wide Multipliers....366
RQS_UTIL-10: Incomplete Case Statement Increasing Control Sets....369
RQS_UTIL-203: Large ROM Inferred using Distributed RAM....372
Reference Design Files....376

Appendix C: Additional Resources and Legal Notices....377

Finding Additional Documentation....377
Support Resources....378
References....378
Training Resources....378
Revision History....379
Please Read: Important Legal Notices....379

Introduction

This document covers how to drive the AMD Vivado™ Design Suite to analyze and improve your design, detailing the following topics:

- Using the Vivado integrated design environment (IDE) to view messages, design netlists, and cross-probe
- Methodology and DRC waivers
- Analyzing timing reports
- Generating all netlist, timing, and design closure reports
- Intelligent design runs, QoR suggestions, and ML strategies

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All AMD Versal™ adaptive SoC design process Design Hubs and the Design Flow Assistant materials can be found on the Xilinx.com website. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration.

Chapter 2

Interactive Design Analysis in the IDE

The following sections provide an introduction to design analysis in the AMD Vivado™ Design Suite IDE. The IDE is particularly relevant for project-based users but many of the features are also applicable to non-project-based users. This chapter covers the following techniques:

- Using the Design Runs window for quick analysis
- Cross-probing between windows
- Methodology analysis
- Placement and routing analysis

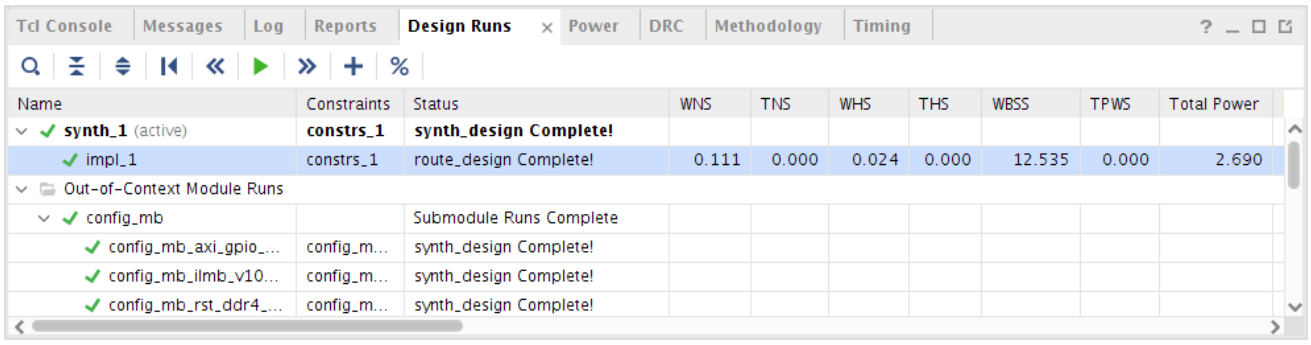
Using the Design Runs Window

The Design Runs window is a useful starting point for analysis. It displays the state of the current runs. For more information, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

The Design Runs window shows the status of runs (running, finished cleanly, or finished with errors).

TIP: *If the run is not up to date, select* ***Force Up-to-Date*** *from the pop-up menu.*

Figure 1: **Design Runs Window**



The Design Runs window columns show the following:

- The name of the run
- The target part
- The constraints set associated with a run
- The run strategy
- The status of the last completed step of a run
- The progress of a run
- The start time of a run
- The elapsed time of a run during execution or the final runtime of a completed run
- The timing score of a run: WNS, TNS, WHS, THS, WBSS, and TPWS (see Report Timing Summary for more information on these numbers). This is where you can quickly verify that a run meets timing. If it does not meet timing, you must start the analysis with the Timing Summary Report.

  ***Note*:** WBSS represents the Worst Bus Skew Slack reported by `report_bus_skew`.
- The number of nets that were not successfully routed
- The utilization of the design LUT, FF, block RAMs, DSP, and if applicable, UltraRAMs.
- The total power estimate
- A brief description of the run strategy
- Methodology check violations
- Available QoR suggestions
- The incremental mode of the design run

When you select a run, the right-click menu allows operations such as opening a run for further analysis and setting up run features such as incremental compile or QoR suggestions.

Logic Analysis in the IDE

Logic Analysis Features

This chapter discusses Logic Analysis Features, and includes:

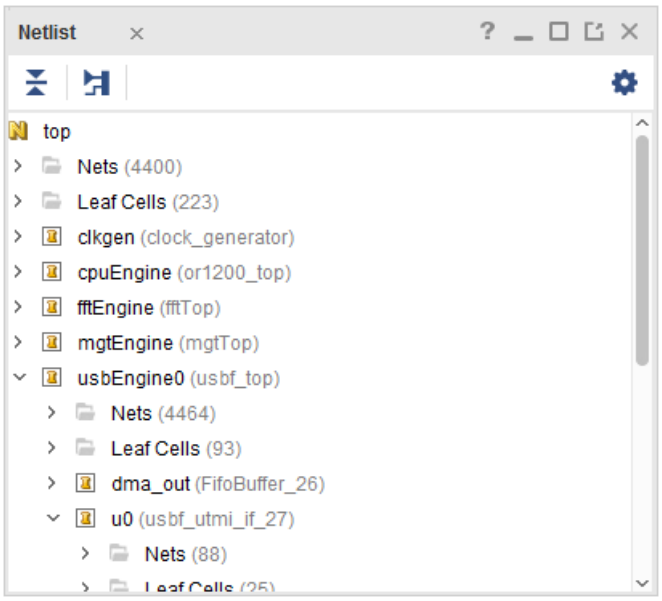
- Using the Netlist Window
- Using the Hierarchy Window
- Using the Schematic Window

- Searching for Objects Using the Find Dialog Box
- Analyzing Utilization Statistics
- Using Report DRC
- Methodology Analysis

Using the Netlist Window

The Netlist Window shows the design hierarchy as it is in the netlist, processed by the synthesis tools. It is useful for exploring the logical hierarchy of the design.

Figure 2: **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 defaults to preserving most of the user hierarchy while optimizing the logic. This results in a smaller and faster netlist.

With the synthesis tool defaults, the netlist hierarchy is recognizable, but the interfaces to the hierarchies may be modified. Some pins and levels of hierarchy may be missing.

The netlist hierarchy is represented as a folder tree. At each level, the tool shows:

- A Nets folder for any nets at that level
- A Leaf Cells folder if there are hardware primitive instances at that level
- A hierarchy folder for any hierarchies instantiated at that level

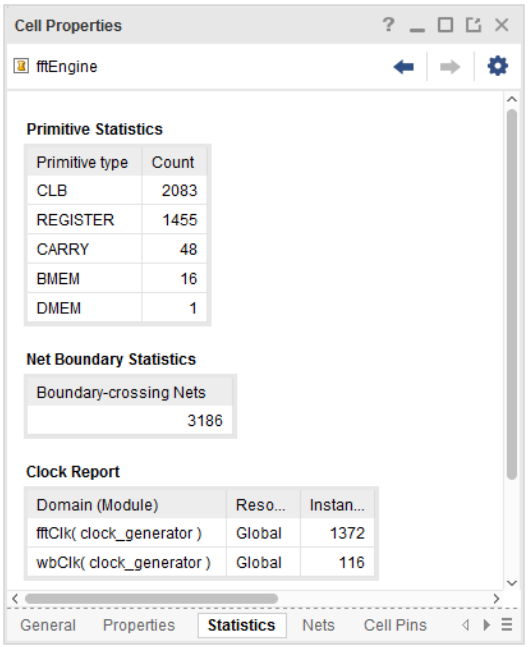
Expanding a hierarchy folder reveals the Nets, Leaf Cells, and hierarchies at that level. The icons next to the cells display information about the state of the design.

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

The Cell Properties Window for the selected hierarchy provides useful information filtered by the category buttons at the bottom of the window. Selecting the Statistics button shows utilization statistics including:

- Primitive usage for the whole hierarchical branch, grouped in higher level buckets
- The number of nets crossing the hierarchy boundary
- Each clock, whether it is on global routing and the number of its loads in the current hierarchical branch

Figure 3: **Cell Properties Window**



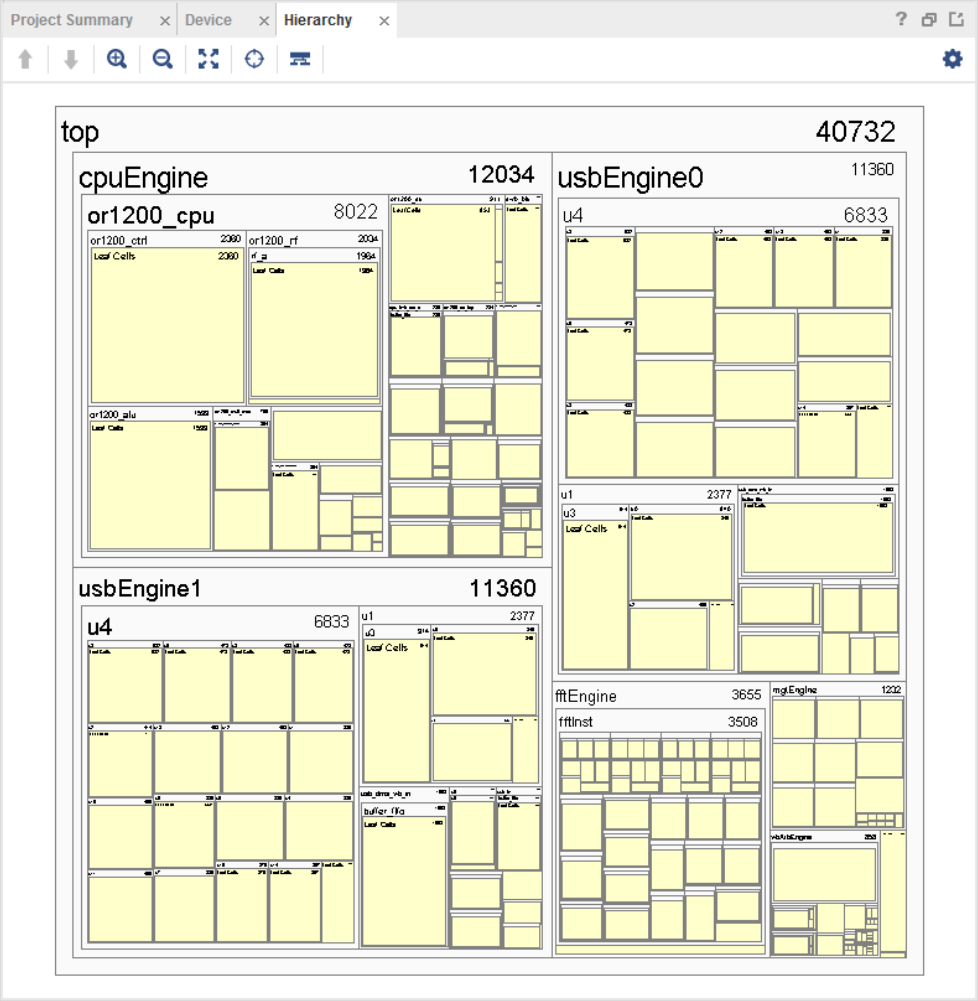
If you floorplan the design, similar properties are displayed for the Pblock.

Using the Hierarchy Window

Explore the hierarchy physically to understand the resource usage. To open the Hierarchy Window, select **Tools > Show Hierarchy**, or from the Netlist window, click **F6**.

As shown in the following figure, the Hierarchy Window displays a hierarchy map for the selected hierarchy. The hierarchy map displays the leaf cells as yellow blocks nested within rectangles corresponding to their parent hierarchy. Each level of the hierarchy is sized relative to the flat number of instances at that level compared to the total number of instances in the design.

Figure 4: **Hierarchy Window**



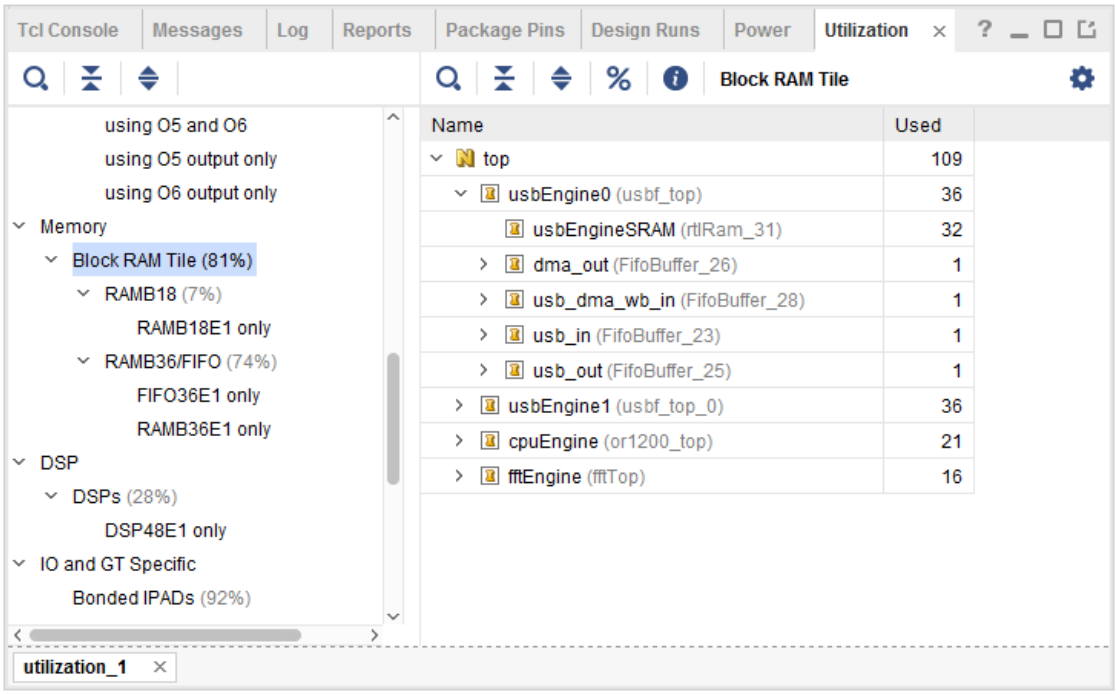
The previous figure shows that `cpuEngine`, `usbEngine0`, and `usbEngine1` have most of the logic in the design, and all use about the same number of resources.

Using the Utilization Report

The Utilization Report breaks down the design utilization based on resource type. The left panel summarizes usage by resource type and the right panel displays usage per hierarchy.

To view the Utilization Report, select **Reports → Report Utilization**. The following figure shows the Utilization Report.

Figure 5: **Utilization Report**



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

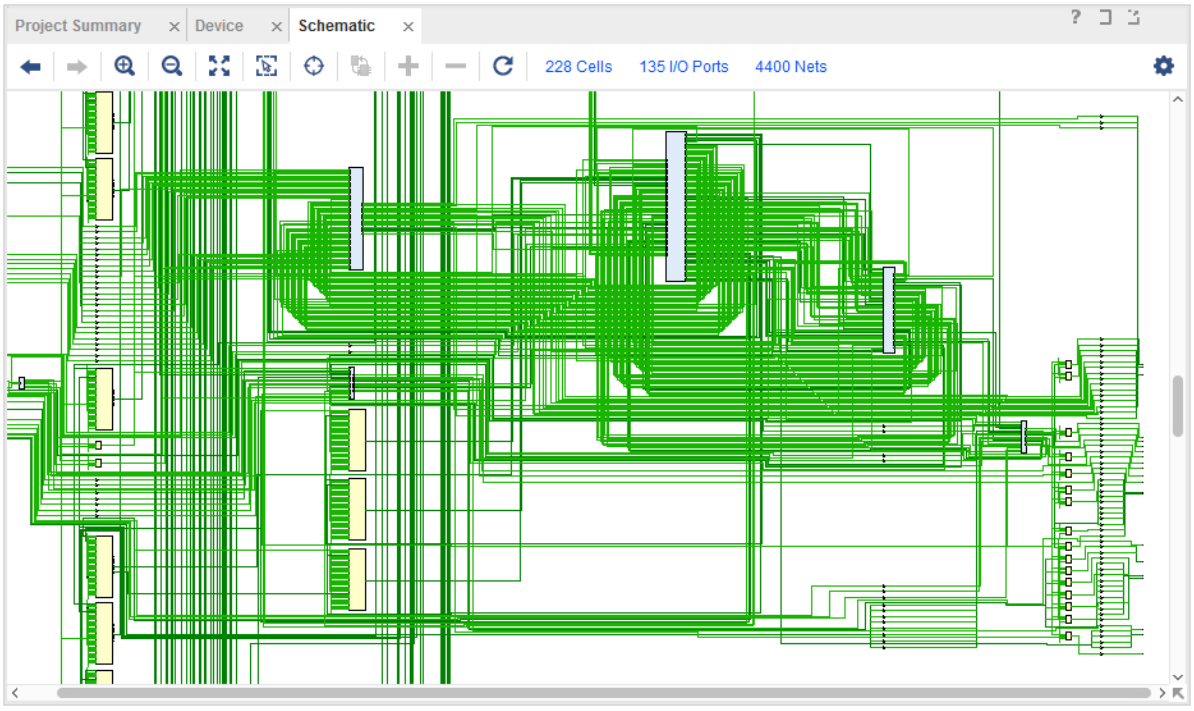
Using the Schematic Window

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.

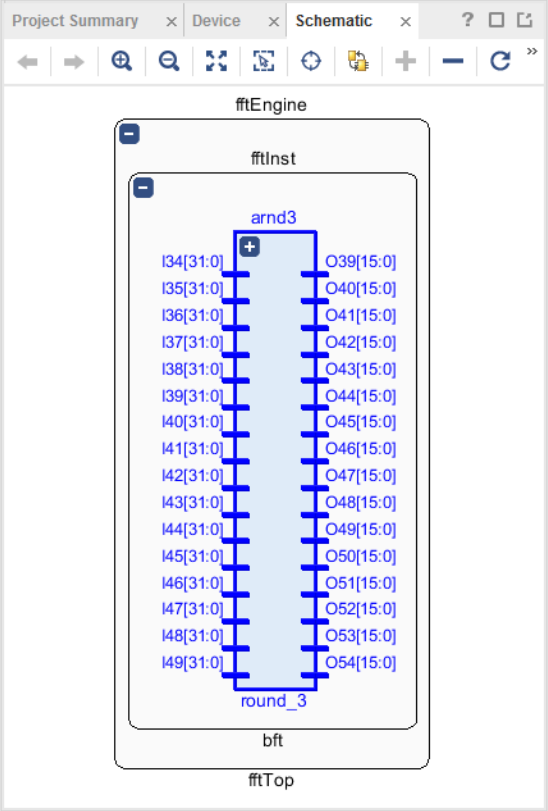
At the RTL level in Elaborated Design, you see how the tool has interpreted your code. In Synthesize Design and Implemented Design, you see the gates generated by the synthesis tool. To open the schematic, select **Tools > Schematic**. If nothing is selected, the gates, hierarchy, and connectivity appear at the top level of the design, as shown in the following figure.

Figure 6: **Top Level Schematic**



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

Figure 7: **Schematic with Single Hierarchy Selected**



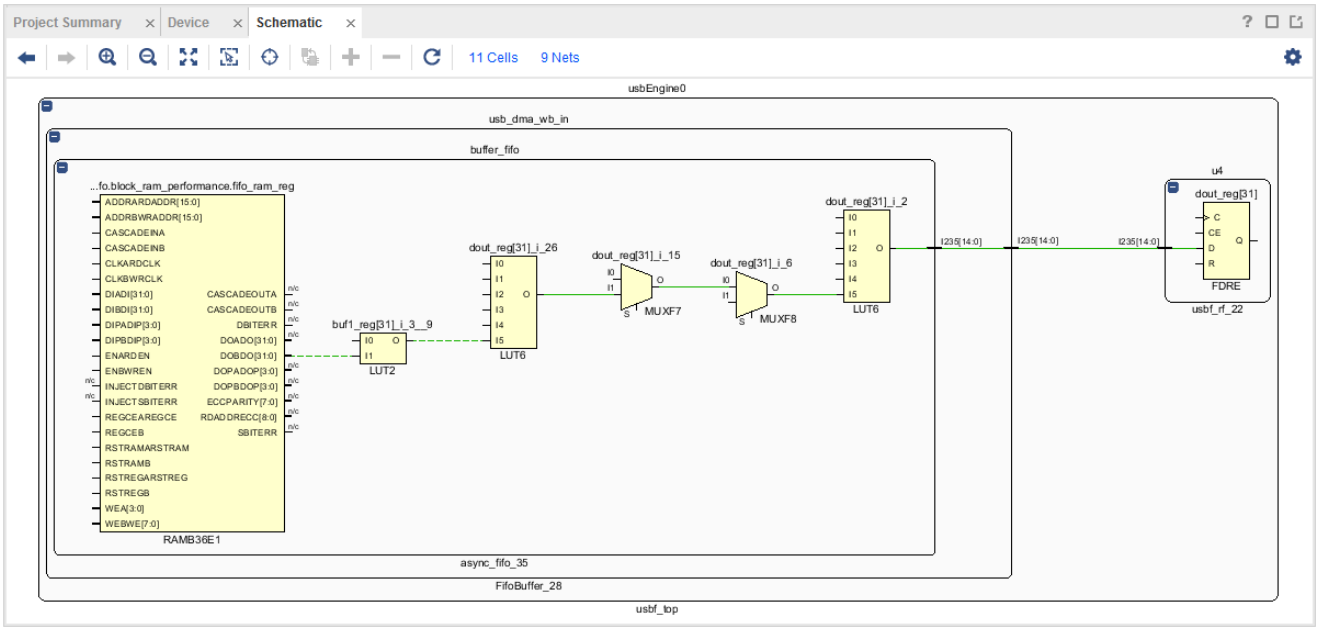
You can trace the schematic in multiple ways:

- Click the **+** (plus) icon in the upper left to display the gates in the hierarchy.
- Double-click a port or element to expand it.
- Right-click and select **Schematic** from the popup menu.
- Click the **<- ->** navigation arrows to switch between the previous and next schematic views.

For more information about schematics, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

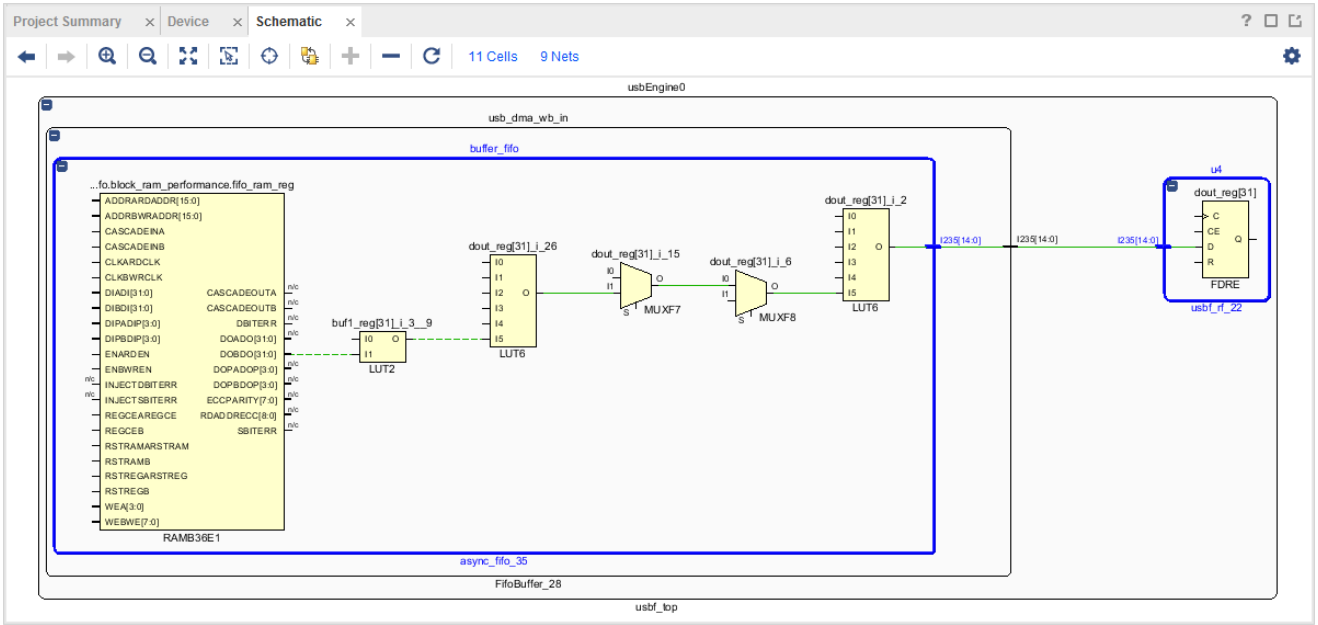
After implementation, the schematic is the easiest way to visualize the gates in a timing path. Select the path, then open the schematic with the gates and nets from that path.

Figure 8: **Schematic with Timing Path**



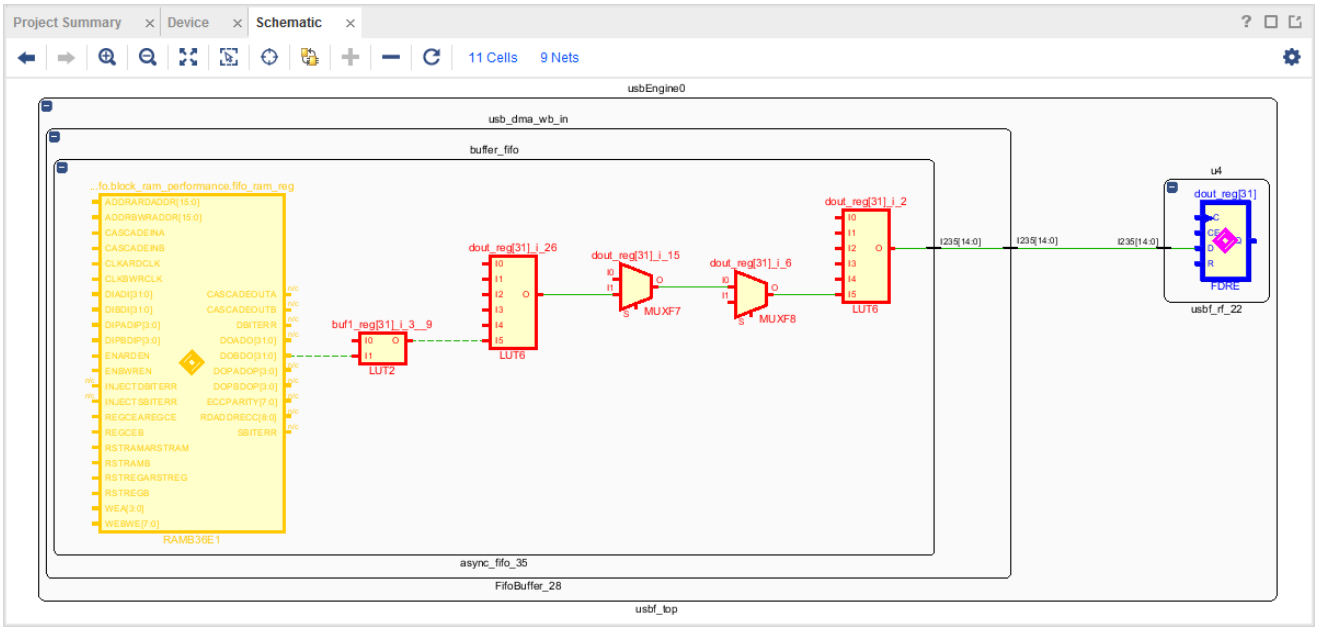
To identify the relevant levels of hierarchy of a selected cell in the schematic, choose **Select Leaf Cell Parents** from the popup menu.

Figure 9: **Timing Path with Select Primitive Parents**



As you review the schematic, select **Highlight** and **Mark** commands to track leaf cells of interest. Color coding cells (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 10: **Schematic with Timing Path Marked**

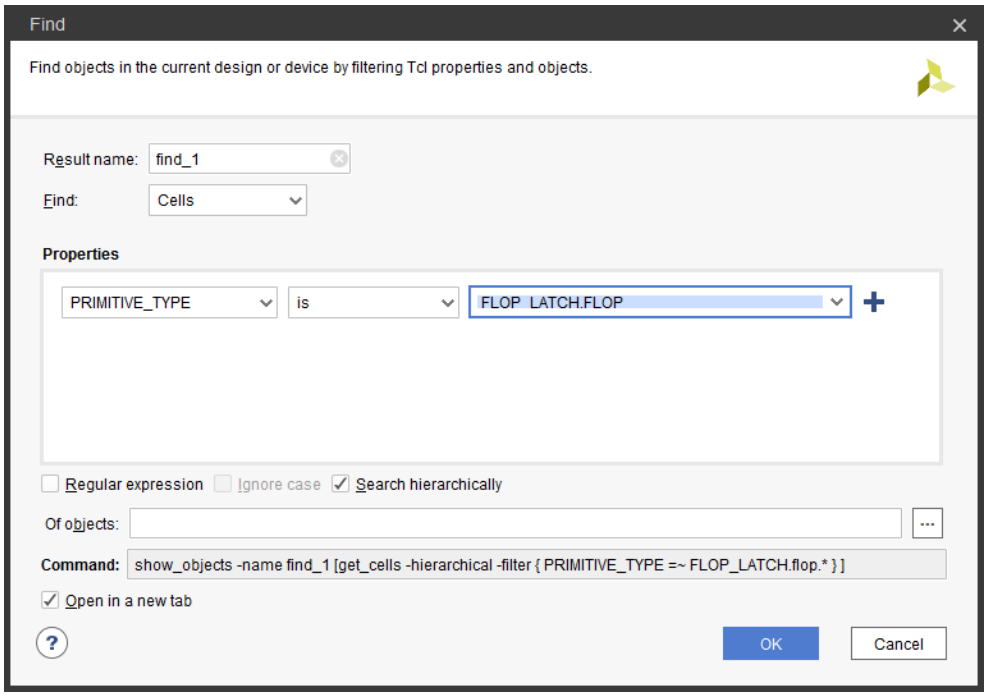


Searching for Objects Using the Find Dialog Box

The AMD Vivado™ IDE includes powerful find and search capabilities. To open the Find dialog box, select **Edit → Find**. See the following figure.

***Note*:** You can also open the Find window by pressing **Ctrl+F.**

Figure 11: **Find Dialog Box**



Find Criteria

The Find dialog box allows you to search the netlist for a wide range of criteria and properties, as shown in the following figures.

Figure 12: **Find Dialog Box Displaying Search Criteria**

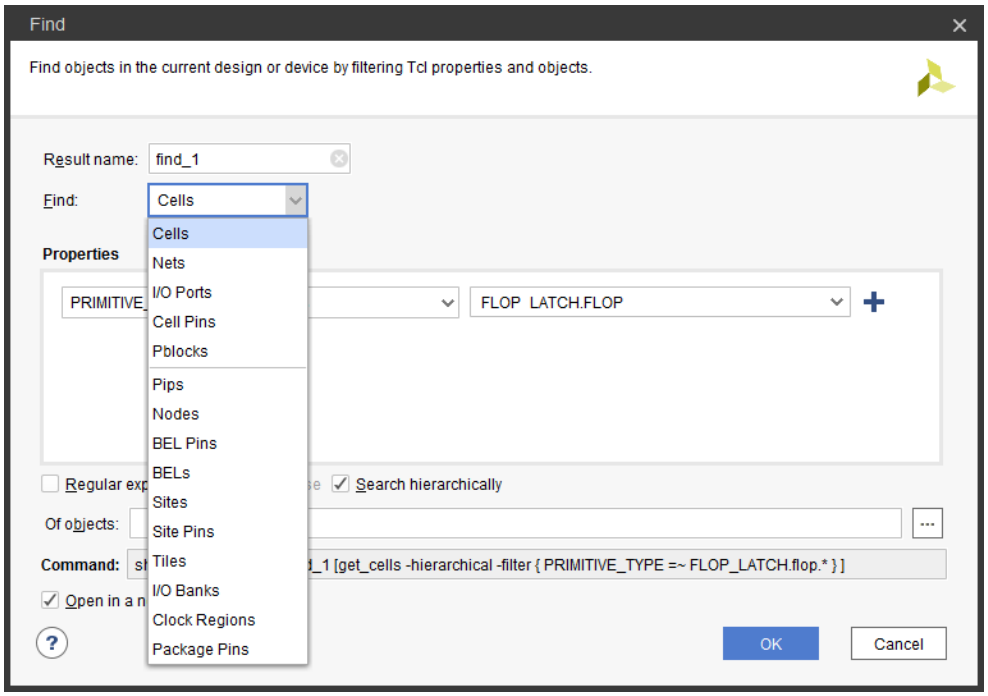
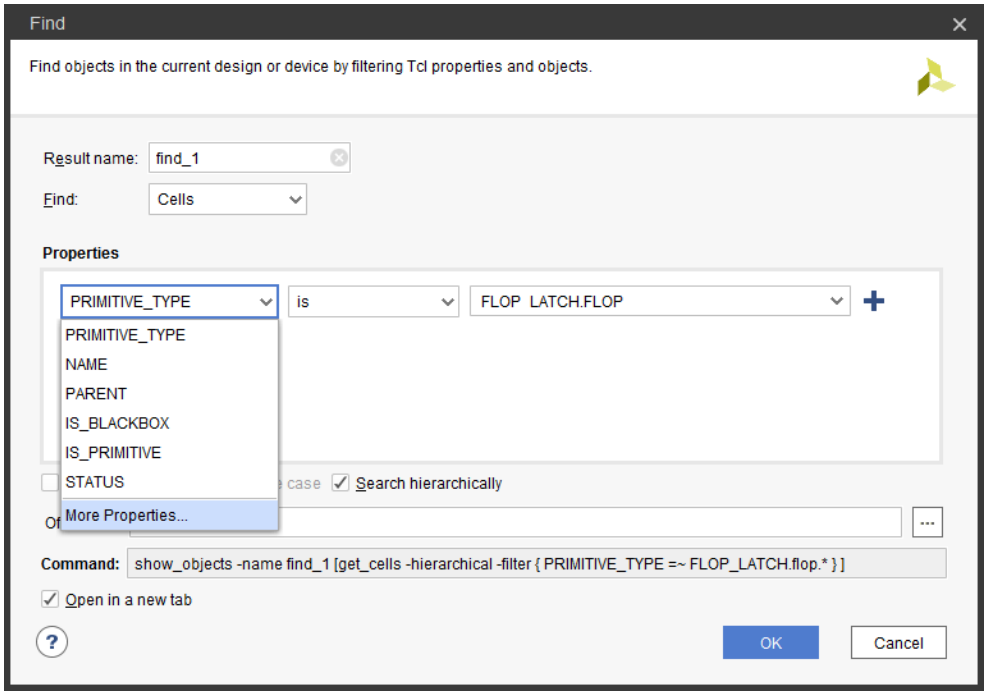


Figure 13: **Find Dialog Box Showing Properties Options**



Complex Finds

To run a complex find:

1. Set the first search criterion.
2. Click **+** (plus) next to the Properties drop-down options.
3. Add additional criteria.
4. Join the additional criteria with logical operators (AND, OR).

Find Examples

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

- All unplaced I/Os:

  Find: Cells, **Properties:** Primitive is IO + AND STATUS is UNPLACED

- All nets with a fanout over 10,000:

  Find: Nets, **Properties:** FLAT_PIN_COUNT is greater than 10000

- All DSPs using the PREG embedded register:

  Find: Cells, **Properties:** PRIMITIVE_TYPE is ARITHMETIC.DSP + AND PREG is greater than 0

Tcl Finds

From the script or Tcl Console, use the equivalent Tcl `get_*` command (such as `get_cells`) to query Vivado objects.



TIP: *The Tcl Console at the bottom of the AMD Vivado™ IDE shows the Vivado Design Suite Tcl commands run for each action executed in the GUI. From the Tcl Console, you can also enter Vivado Design Suite Tcl commands.*

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

For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835), or type `<command> -help`.

Methodology Analysis

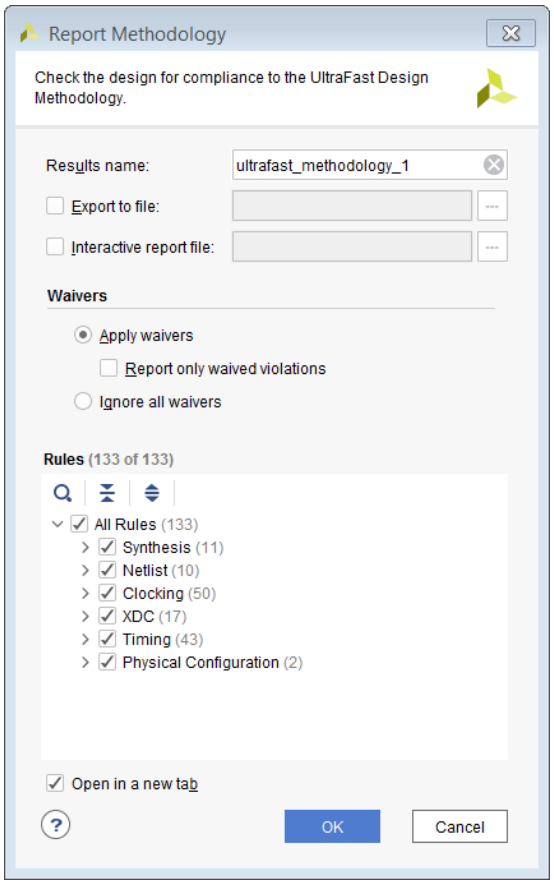
Due to the importance of methodology, the AMD Vivado™ tools provide the `report_methodology` command, which specifically checks for compliance with methodology DRCs. There are different types of DRCs depending on the stage of the design process. RTL lint-style checks are run on the elaborated RTL design; netlist-based logic and constraint checks are run on the synthesized design; and implementation and timing checks are run on the implemented design.

To run these checks at the Tcl prompt, open the design to be validated and enter following Tcl command:

```
report_methodology
```

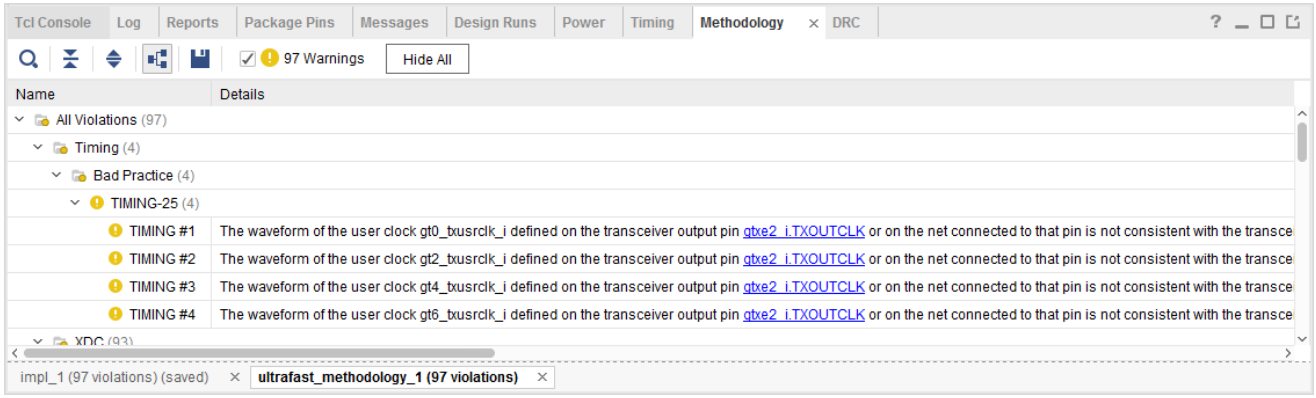
To run these checks from the IDE, open the design to be validated and run the Report Methodology command from the Flow Navigator in project mode, or from **Reports → Report Methodology**. The dialog box appears, as shown in the following figure.

Figure 14: **Report Methodology Dialog Box**



Violations (if there are any) are listed in the Methodology window, as shown in the following figure.

Figure 15: **Methodology Violations**



For more information on running design methodology DRCs, refer to this link in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895).

***Note*:** It is recommended to address all methodology violations with a special focus on Critical Warnings as they affect both timing closure and sign-off quality.

Placement Analysis

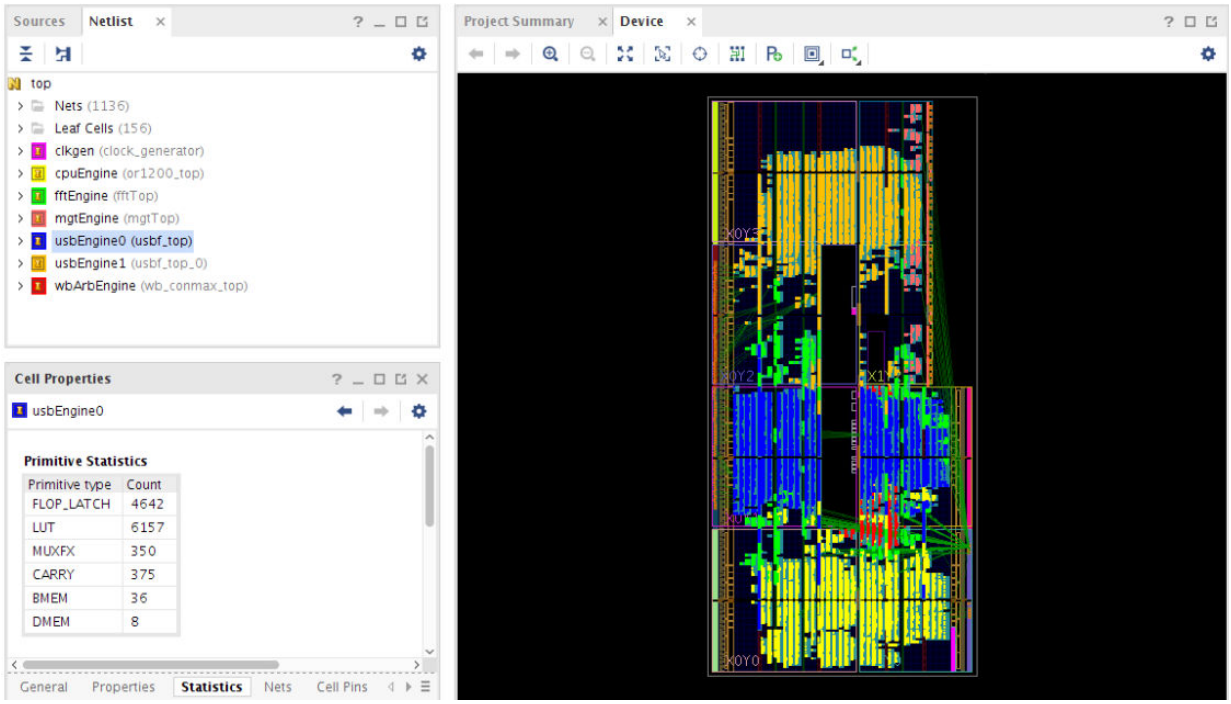
Highlighting Placement

Another way to review design placement is to analyze cell placement. TheHighlight Leaf Cells command helps in this analysis.

1. In the Netlist window, select the levels of hierarchy to analyze.
2. From the popup menu, select **Highlight Leaf Cells → Select a color**.
3. If you select multiple levels of hierarchy, select **Cycle Colors**.

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

Figure 16: **Highlight Hierarchy**



The color coding shows the placement of the key hierarchical blocks in the device. For instance, you can observe the following attributes of `usbEngine0` (in blue):

- It uses a number of block RAM and DSP48 cells.
- It is in the middle clock regions of the chip.
- It is intermingled with other logic (`fftEngine`) in the design.

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

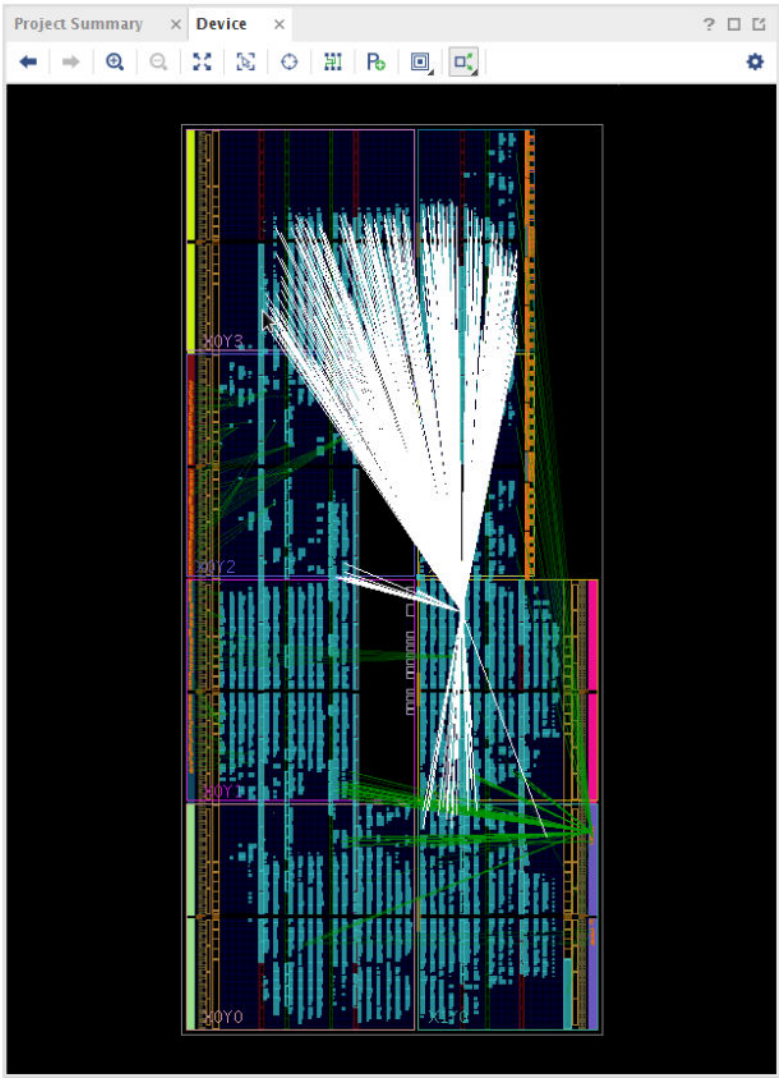
Showing Connectivity

It can be useful to analyze a design based on connectivity. Run **Show Connectivity** to review the placement of all logic driven by an input, a Block RAM, or a bank of DSPs. Show Connectivity takes a set of cells or nets as a seed, and selects objects of the other type.

TIP: *Use this technique to build up and see cones of logic inside the design.*

The following figure 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.

Figure 17: **Show Connectivity**



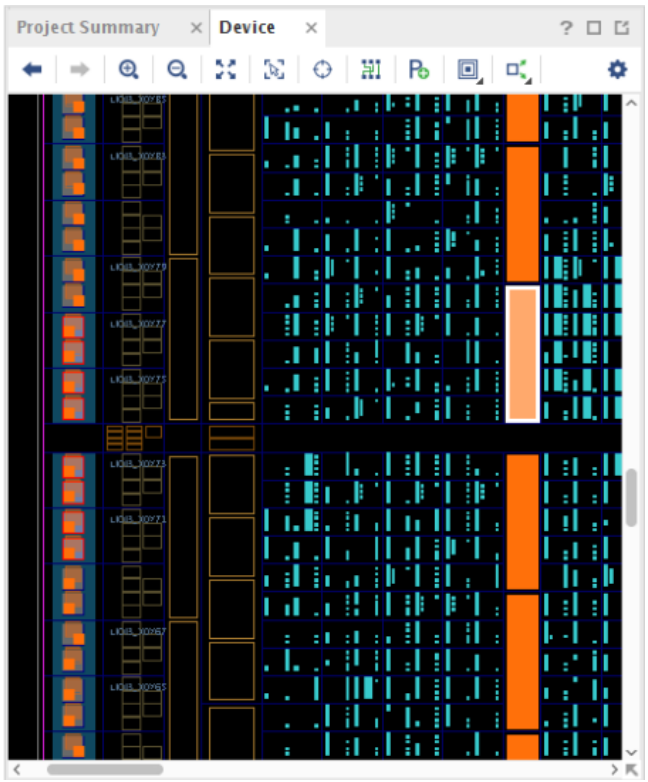
Fixed and Unfixed Logic

The AMD Vivado™ tools track 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 might have a BEL constraint.
- Elements placed by the tool (shown in blue) are Unfixed.

In the following figure, the I/O and Block RAM placement is Fixed. The slice logic is Unfixed.

Figure 18: **Fixed and Unfixed Placement**



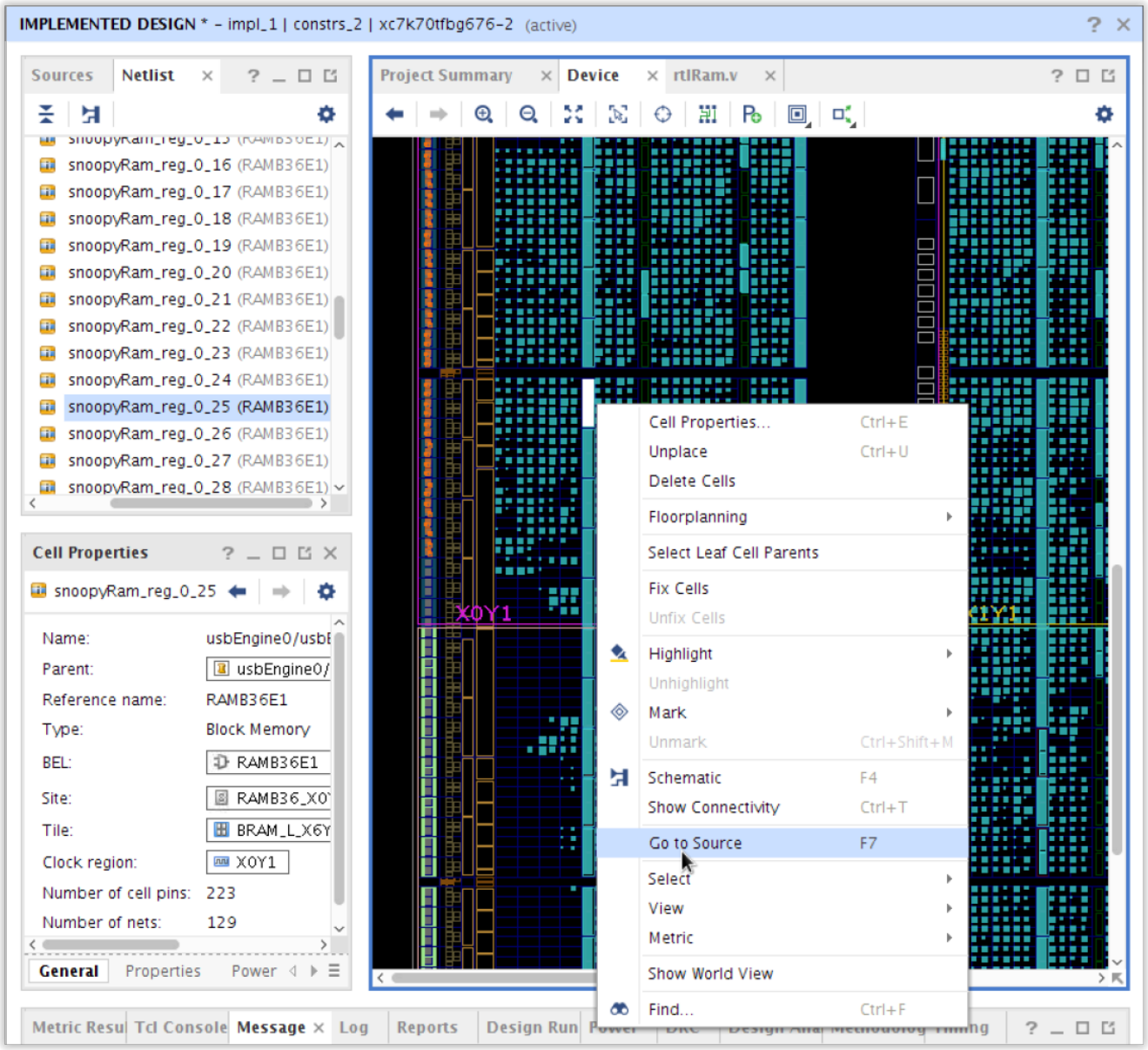
Cross-Probing to RTL Source

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 Source** from the popup menu, shown in the following figure.

Figure 19: **Cross-Probe Back To Source**



Use cross-probing 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 Metrics

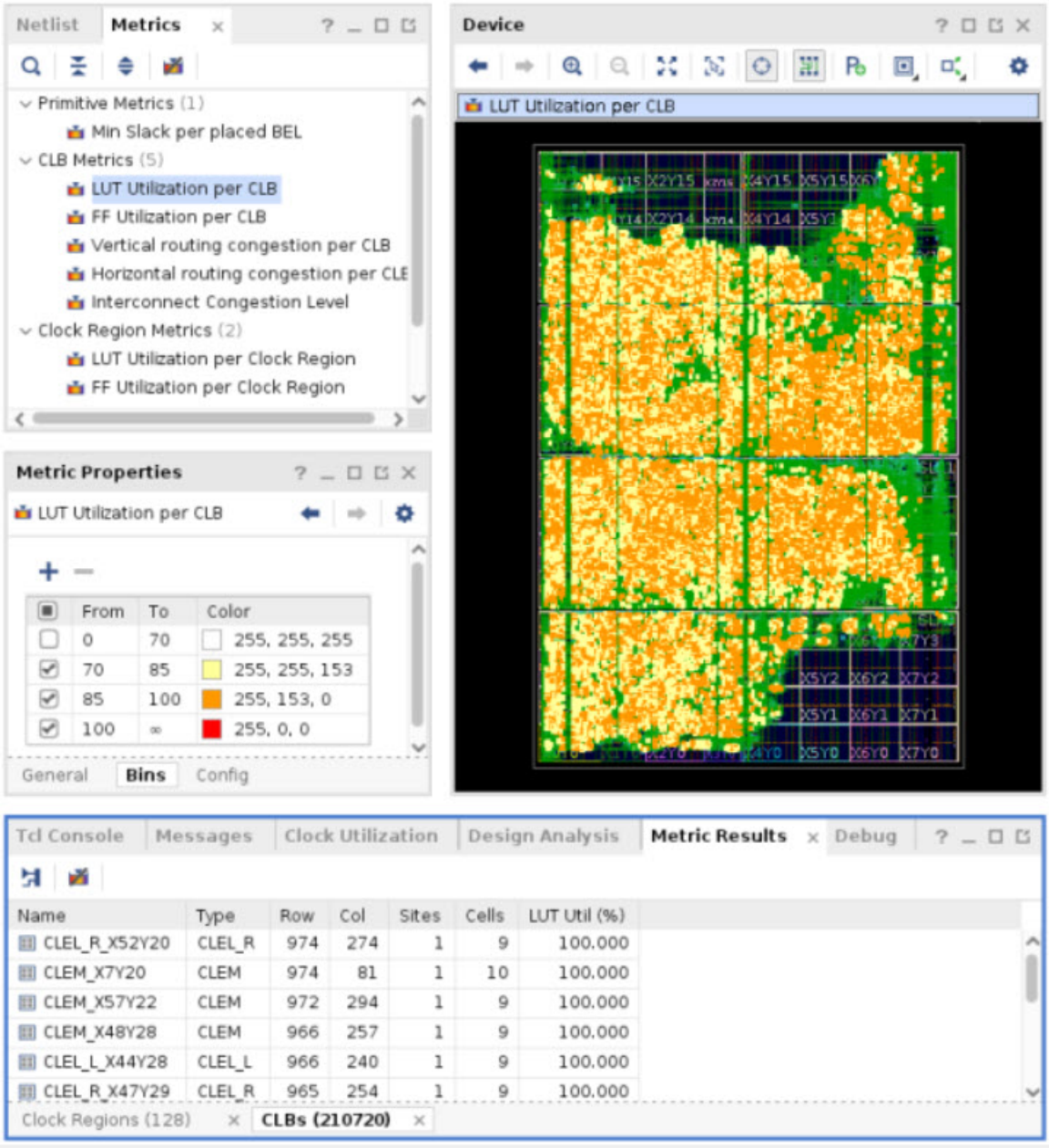
The Vivado Design Suite provides metrics to inform you about logic and routing inside the device. They provide an interactive way of analyzing a design rather than a static report.

To activate metrics, make sure that you have an open design. Select **Window → Metrics**. To enable a metric, right-click it and select **Show**. More than one metric can be added at a time. To disable a metric, right-click it and select **Hide** to remove it from the Device view. Select **Hide All Metrics** to hide all the enabled metrics in one click.

The following figure shows an example of the LUT Utilization per CLB metric:

- The Metrics window shows the list of available metrics that can be shown using the right-click menu.
- The Metric Properties window shows the color bins that the given metric has been graded by. You can modify the metric bins by enabling or disabling them, or by changing the colors. New bins can be added, or existing bins can be removed.
- In the Device window, the metric bins have been overlaid to show the physical location where the metric levels have a higher impact.
- The Metric Results window allows you to select and use the cross-probe with other design elements.

Figure 20: **LUT Utilization per CLB Metric**



Metrics in a Netlist Design with No Placement

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

- **LUT Utilization per Pblock:** This metric 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:** This metric color codes the Pblock based on an estimate of how the FFs will be packed into the slices contained in the Pblock.

Metrics Requiring a Placed Design

Four metrics require a placed design 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.

For UltraScale+ and newer architectures:

- Interconnect Congestion Level: Color codes the Interconnect Congestion Level based on a worst case estimate of routing usage over contiguous regions.

Timing Metrics

Timing metrics show a physical representation of the designs timing issues. Each bel is color coded based on the worst case negative slack (WNS) value on the timing path through the bel.

Utilization Metrics

Utilization metrics represent percentages of used resources and are graded by CLB or clock region usage. They are available for both LUT and FF utilization.

Congestion Metrics

Different congestion metrics should be used depending on the device family being targeted.

Congestion Metrics for 7 Series and UltraScale Device Families

For 7 series and UltraScale parts, a method similar to the placer estimation of congestion is used by way of the following metrics:

- Vertical routing congestion per CLB
- Horizontal routing congestion per CLB

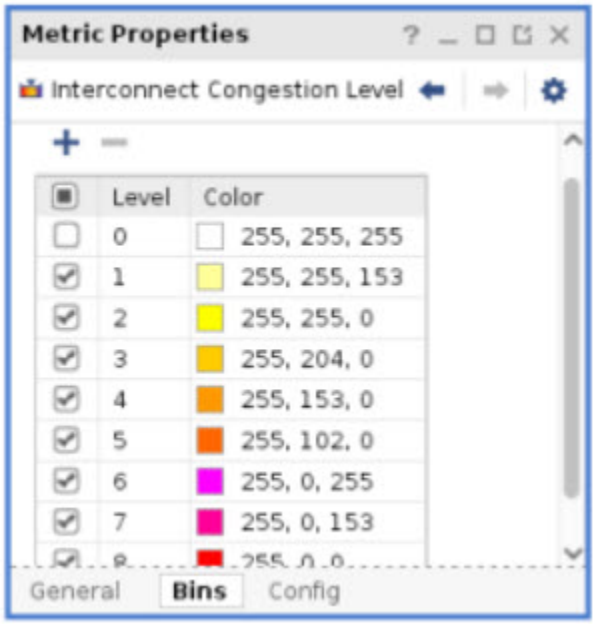
Both of these metrics show a demand-based model that estimates the requirement for routing going between two points. When the demand exceeds 100, routing in adjacent tiles is used. Congestion is proportional to the area of the die where demand is consistently exceeding 100%.

Note: This method is still available for newer device families, but it is not recommended for use.

Congestion Metrics for UltraScale+ and Newer Device Families

For UltraScale+ and newer device families, use the Interconnect Congestion Level metric. This metric has simplified representation and improved accuracy, and is better aligned to how the router estimates congestion. The color of the bins for this metric is based on the congestion levels.

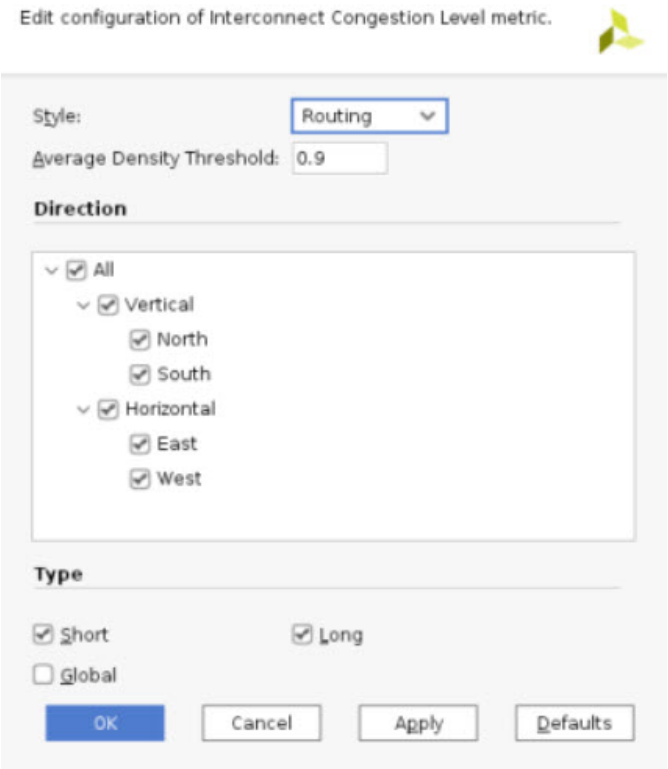
Figure 21: **Interconnect Congestion Level Metric Properties**



In the configuration window, the metric view can be configured to show the following:

- Estimated routing congestion (impacts the placer), actual routing congestion (impacts the router), or both
- Only items above the interconnect utilization threshold (0.9 = 90% usage)
- Any combination of specific routing types (Short, Long, or Global)
- Any combination of directions that the routing traverses

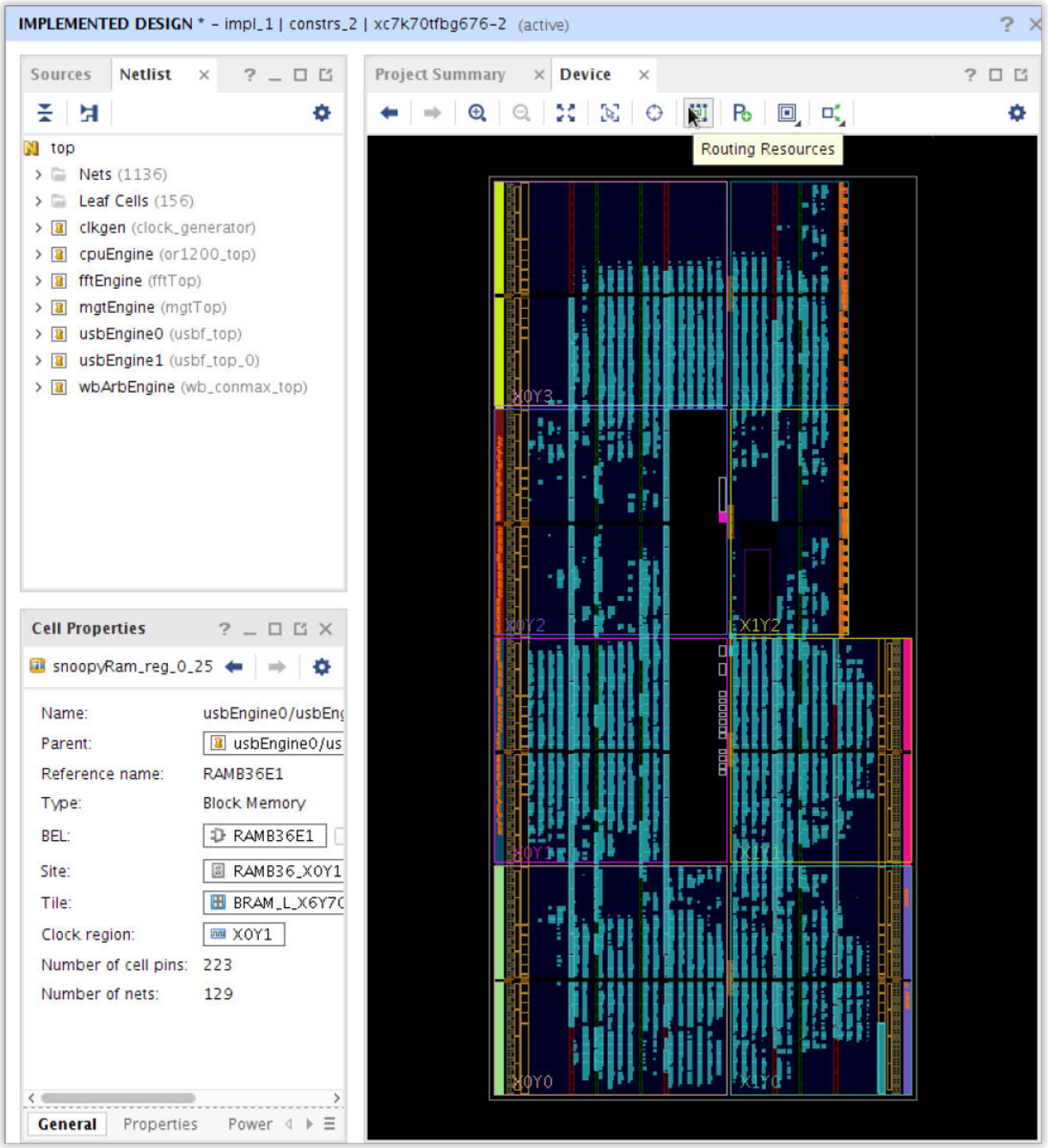
Figure 22: **Interconnect Congestion Level Configuration**



Routing Analysis

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

Figure 23: **Enable Routing**



Displaying Routing and Placement

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

- When zoomed out
- At closer zoom levels

TIP: *The two visualizations of the Device window minimize runtime and memory usage while showing the details of designs of all sizes.*

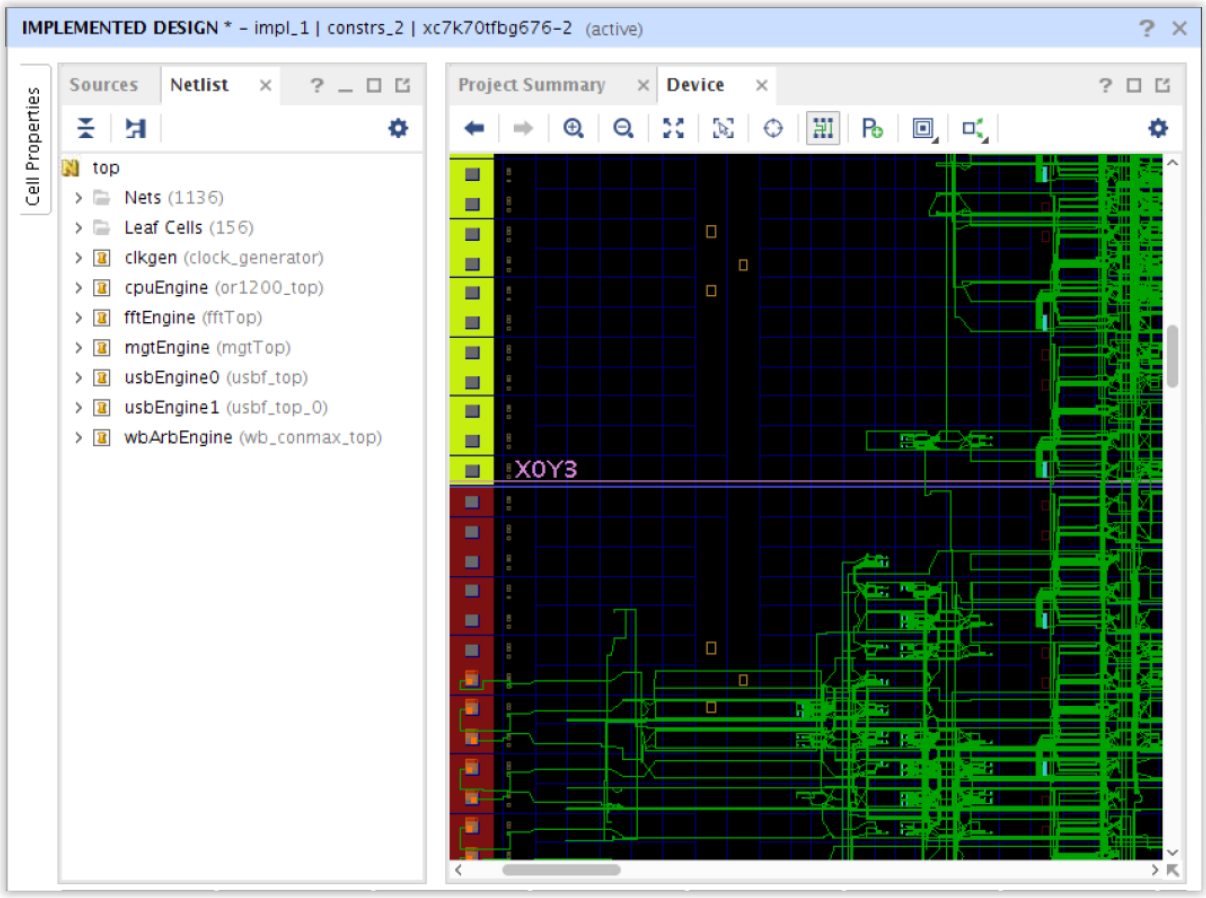
Displaying Routing and Placement when Zoomed Out

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.

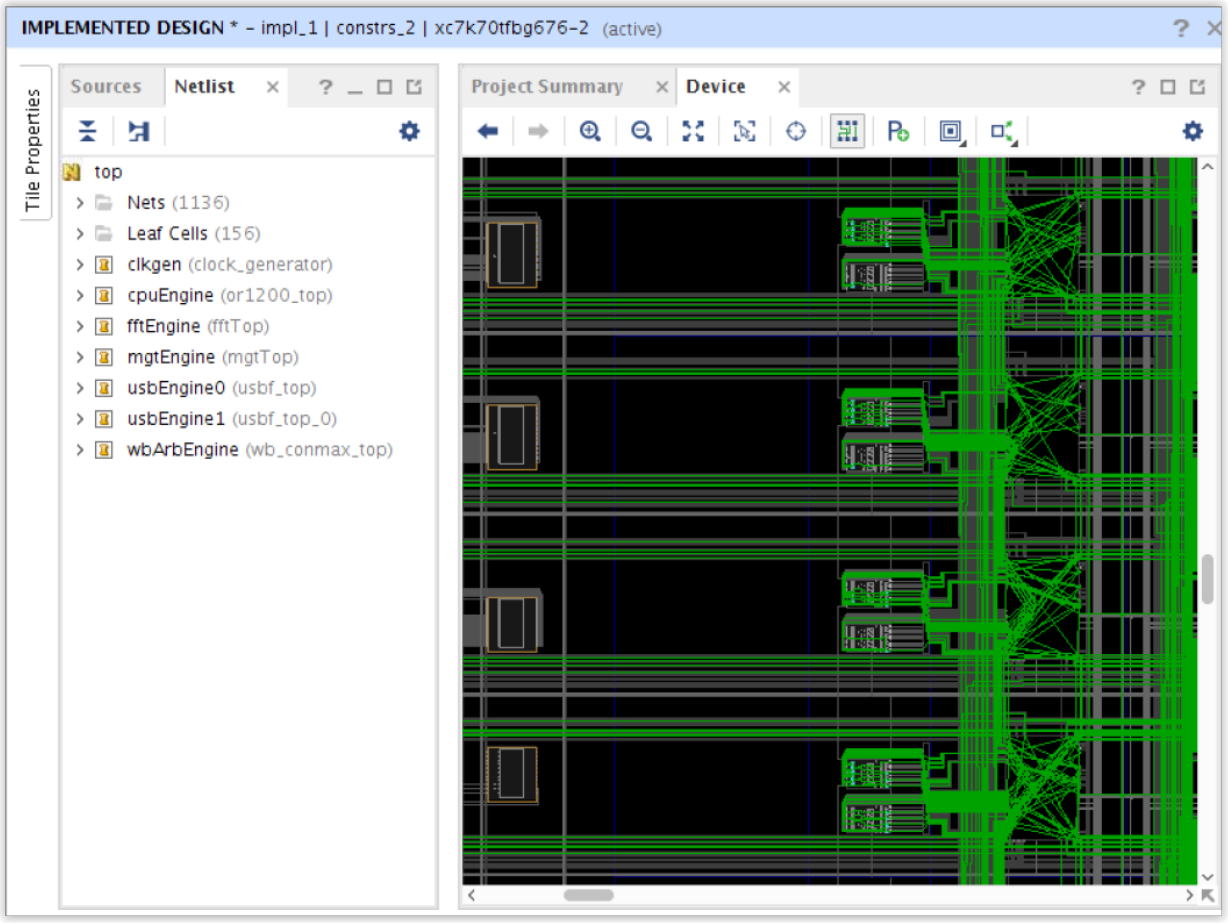
Figure 24: **Abstract View**



Displaying Routing and Placement at Closer Zoom Levels

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

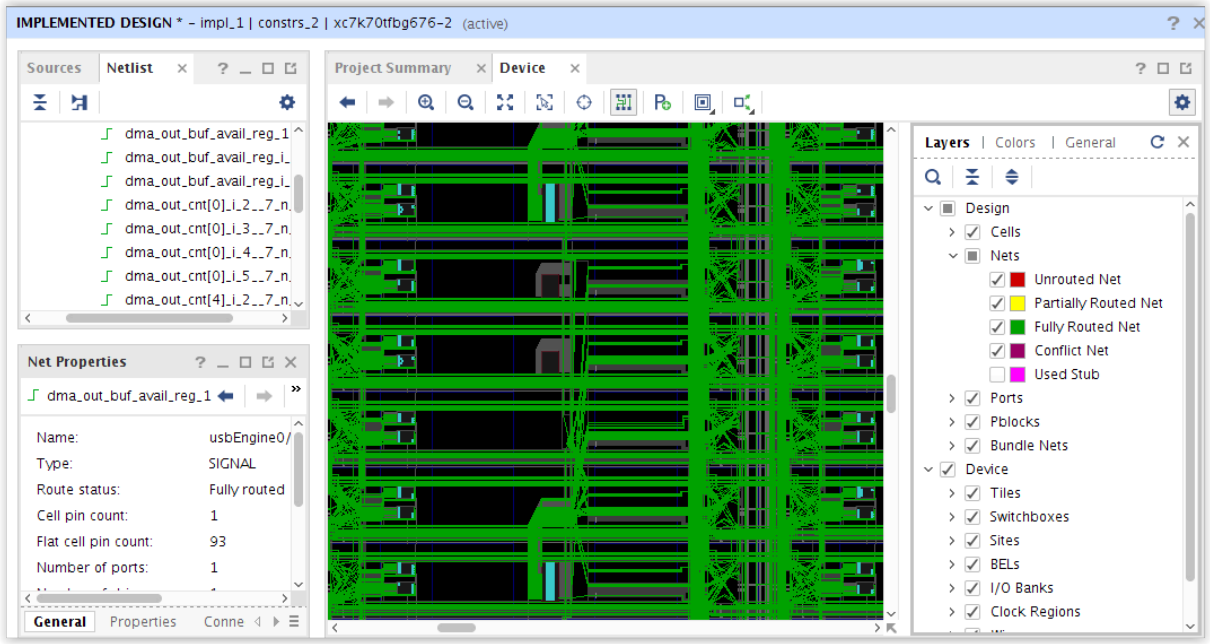
Figure 25: **Detailed View**



Viewing Options

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

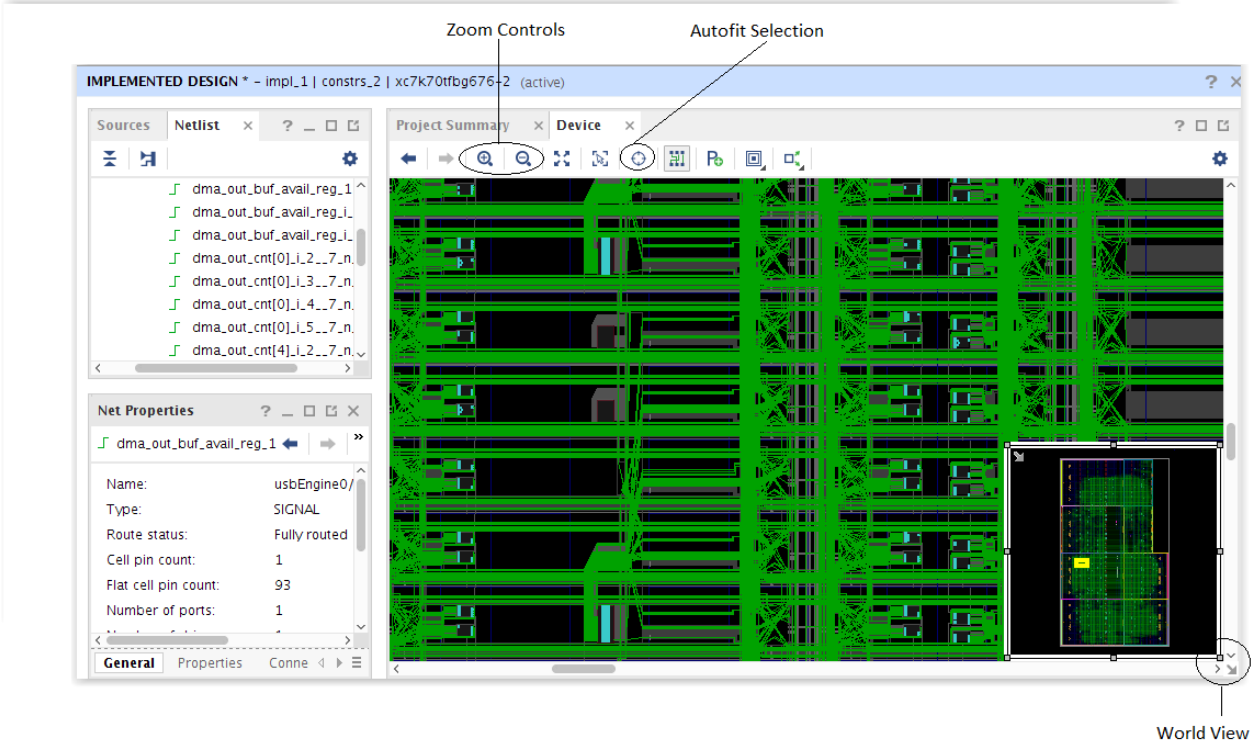
Figure 26: **Device Window Layers**



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

Navigating in the Device Window

Figure 27: **Navigating the Device Window**



Use the following tools to navigate in the Device window:

- Zoom Controls: Standard Zoom In, Zoom Out, and Zoom Full tools.
- Auto-fit 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. Use **Ctrl** and the mouse wheel to zoom in and out at the position of the cursor.

Viewing Reports and Messages

The AMD Vivado™ integrated design environment (IDE) generates reports and messages to inform you of the state of the design or design processes during various tool interactions. Reports are generated by you (or by the tool) at key steps in the design flow. The reports summarize specific information about the design.

The tool generates messages automatically at each step of the design process, and for many user actions.

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

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

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

For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835), or type `<command> -help`.

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.

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.

Viewing and Managing Messages in the IDE

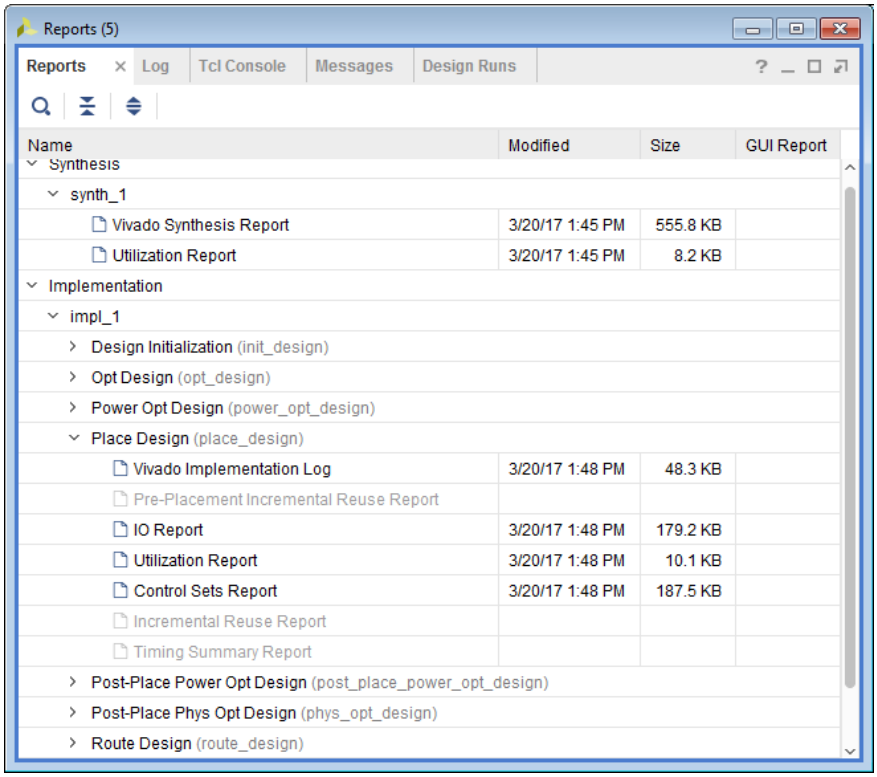
Messages provide brief status notes about specific elements of the design, or about errors that occurred in tool processes.

TIP: *Review the messages to determine whether the Vivado tools are having difficulty, or are encountering errors in any sections of the design.*

Using the Reports Window

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

Figure 28: **Reports Window**



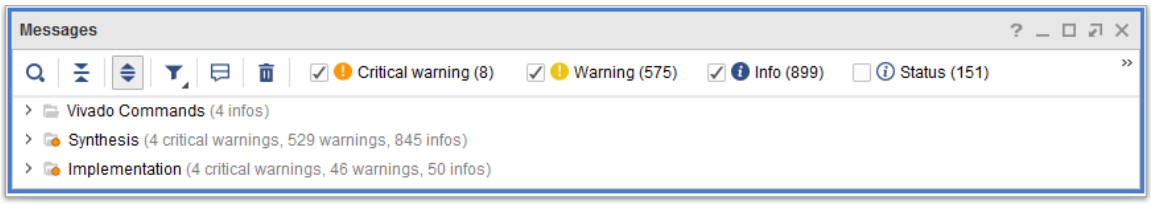
Using the Messages Window

There are two types of messages:

- Messages stored on disk
- Messages stored in memory

The Vivado Integrated Design Environment (IDE) groups messages in the Messages window by the action that created the message. Use the settings buttons on the toolbar menu to group the messages by message ID or file.

Figure 29: **Messages Window**



Some messages include hyperlinks to a file or a design element to help in debugging. Click the link to view the source.

TIP: *Use the popup menu to copy messages to paste into another window or document.*

Each message is labeled with a message ID and a message severity.

- Message ID: The message ID identifies different messages, allowing them to be grouped and sorted.
- Message Severity: The message severity describes the nature of the information presented.

Some messages require your attention and resolution before the design can be elaborated, synthesized, or implemented. Some messages are informational only. Informational messages provide details about the design or process, but require no user action.

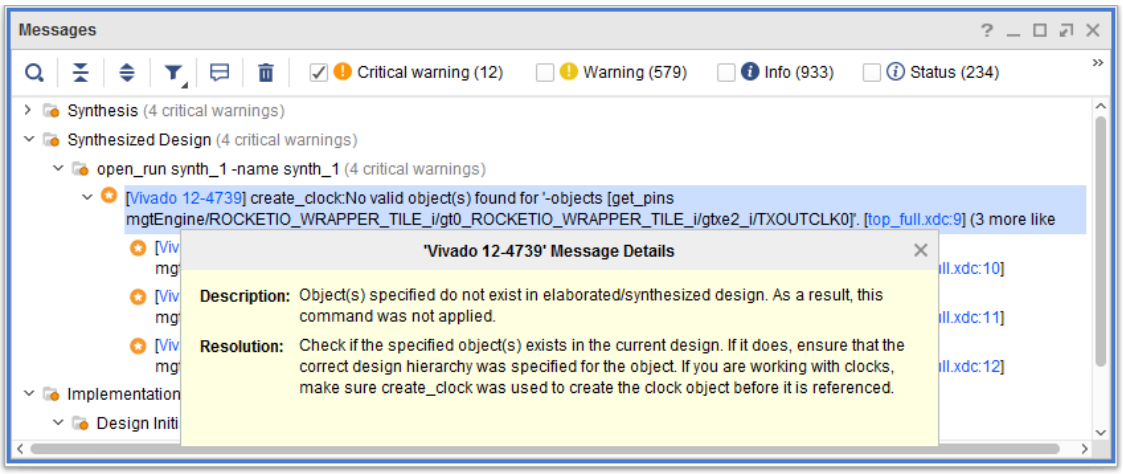
Table 1: **Message Severities**

Severity	Message
Status	Communicates general status of the design processing.
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.

RECOMMENDED: *Carefully review all errors and critical warnings issued by the tools when loading a design in memory, or from your active synthesis and implementation run. The messages provide information about problems that require your attention. Many messages include a longer description. along with resolution advice that can be displayed by clicking on the message ID.*

For an example, see the following figure. In this example, a primary clock constraint refers to a port that cannot be found in the design (first warning), so the clock is not created (first critical warning) and any other constraints that refer to this clock fail as well.

Figure 30: **Reviewing Errors and Critical Warning**



Filtering Messages

You can filter messages by severity.

To enable or disable the display of a specific message type:

1. Go to the Messages window.
2. Select (to enable) or deselect (to disable) the check box next to a message severity in the window header.

You can change the severity of a specific message ID. For example, you can decrease the severity of a message you do not believe is critical, or increase the severity of a message you think demands more attention.

To increase or decrease the severity of a message, use the `set_msg_config` Tcl command. For example:

```
set_msg_config -id "[Common 17-81]" -new_severity "CRITICAL WARNING"
```

For more information on the `set_msg_config` Tcl command, see this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Vivado Generated Messages

This section discusses Vivado generated messages.

Synthesis Log

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

- The files processed, which are:
  - VHDL
  - Verilog
  - System Verilog
  - 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

> **IMPORTANT!** *Review this report or 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.*

Implementation Log

The Vivado Implementation 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 an estimated post-routing timing summary (WNS, TNS, WHS and THS only).
- Elapsed time and memory for each implementation command and phases.

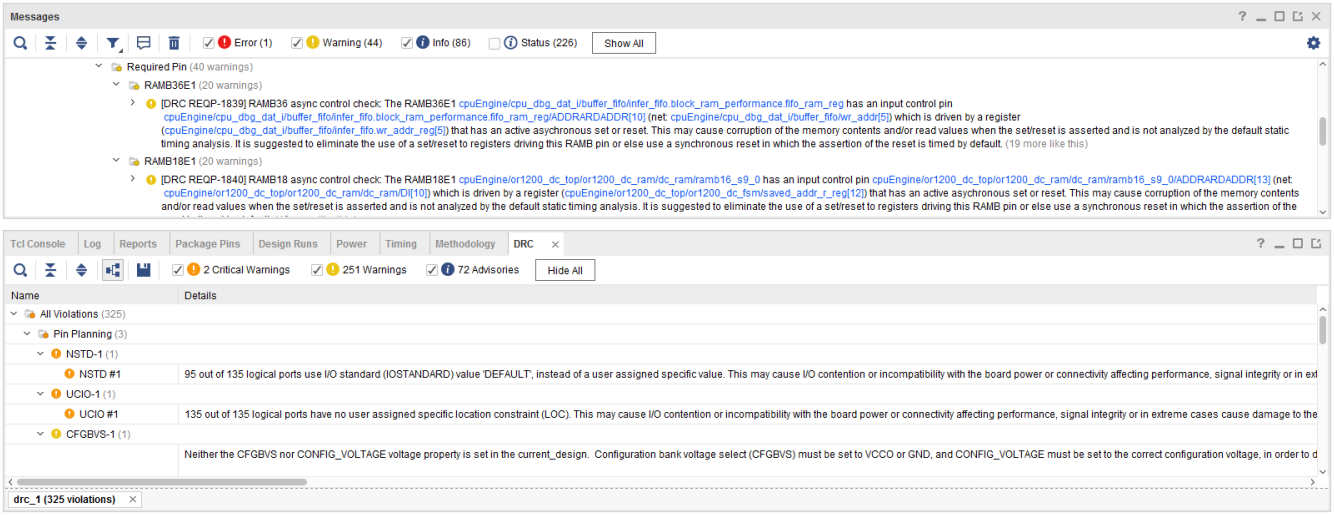
Review this report or the proper section of the messages tab for Errors, Critical Warnings and Warnings. The Placer generates warnings that may be elevated to Errors later in the flow. If using Stepwise runs, the log contains only the results for the last step.

IMPORTANT! *Review the Timing Summary Report to view: (1) the Pulse Width timing summary, and (2) additional information about timing violations or missing constraints.*

Using Report DRC

Design Rule Checks (DRCs) check the design and report on common issues. Since the 2016.1 release, DRCs are split into two different commands. The methodology DRCs have been moved to the `report_methodology` command, while all other DRCs are in the `report_drc` command. Run non-methodology DRCs using the `report_drc` command. During implementation, the tools also run DRCS. The DRCs become more complete and comprehensive with placement and routing.

Figure 31: **Showing Critical Warnings and Error**



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

Critical Warnings in early design stages become Errors later during the implementation flow and prevent bitstream creation. In the above example generated from a post Synthesized Design, the optional Report DRC step reports a Critical Warning for the unconstrained I/Os. The post-route design DRC report also reports these Critical Warnings. You must review the report because at `write_bitstream` the DRC is elevated to an Error. Review the DRC reports early to identify areas of the design that need modification.

WebTalk Report

The WebTalk Report is generated during Bitstream. This report helps AMD understand how its customers use AMD FPGA devices, software, and Intellectual Property (IP). The information collected and transmitted by WebTalk helps AMD improve features most important to customers. No proprietary information is collected. For more information, see https://www.xilinx.com/webtalk/.

Generating and Waiving Design Checks

The waiver mechanism provides the ability to waive CDC, DRC, and Methodology violations. After a violation is waived, it is not reported anymore by the `report_cdc`, `report_drc`, and `report_methodology` commands. Waived DRCs are also filtered out from the mandatory DRCs which are run as a prerequisite for the implementation commands such as `opt_design`, `place_design`, `phys_opt_design`, `route_design` or `write_bitstream`.

Waivers are XDC compatible and can be imported through the commands, `read_xdc` or `source`. They can be included in any XDC file or Tcl script, in Project or Non-Project modes. They can be created from the top-level or scoped to a hierarchical module. After waivers are added to the design, they are automatically saved inside the checkpoint and restored when the checkpoint is reloaded. The waivers can be written out with the commands, `write_xdc` and `write_waivers`.

Waivers provide tracking capabilities. Vivado tools record the user who creates the waivers, the date and time of creation, as well as a short description. This information is important for tracking purposes. It is recommended that all the waivers applied to the design are reviewed and verified to ensure that they are valid.

Waivers are first class objects that can be created, queried, reported on, and deleted. Waivers reference other first class objects returned by the Vivado `get_*` commands, such as pins, cells, nets, Pblocks, and sites. These objects must exist in the design prior to creating the waivers. Design objects that do not exist at the time of waiver creation are not covered by the waivers.

IMPORTANT! *Similar to other constraints, it is recommended to create the waivers on a post-synthesized design. Waivers that are created on post-implemented designs could reference design objects that do not exist inside the post-synthesis netlist. Such waivers are discarded if they are applied on the post-synthesis design.*

The waiver mechanism supports replication and deletion of netlist objects. When an object involved in a waiver is replicated, the replicated object is automatically added to the waiver. Similarly, when an object is deleted, any reference to it is automatically removed from the waivers. When the deleted object results in a waiver referencing an empty list of objects, the waiver is deleted from the in-memory design and is not saved in the subsequent checkpoints. The same mechanism applies to timing constraints and clock objects. When a clock gets deleted through logic optimization or by removing the timing constraints (`reset_timing`), any waiver referencing the clock object gets deleted and is not saved in the subsequent checkpoints.

***Note*:** The Vivado commands `rename_net`, `rename_cell`, `rename_port`, and `rename_pin` do not rename the design elements inside the waiver. If a waiver refers to a netlist element renamed by one of these commands, the waiver becomes invalid because it still refers to the design element's original name.

***Note*:** Custom Design Rule Checks cannot be waived. For more information about user written DRCs, refer to this link in the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894).

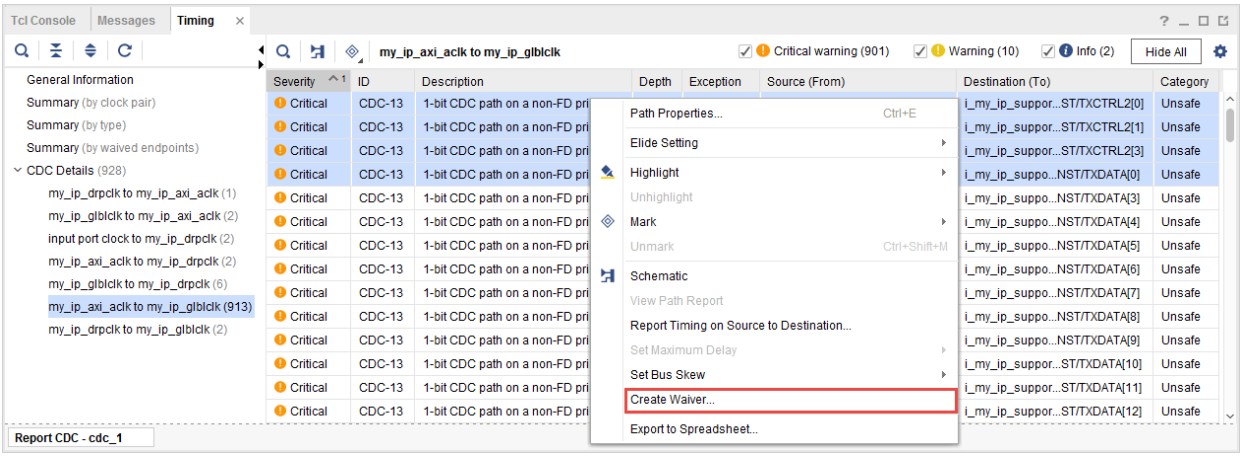
Creating a Waiver

Waivers can be created from the GUI, from a DRC, Methodology, or CDC violation object, or by manually specifying all the required arguments.

Creating Waivers from the GUI

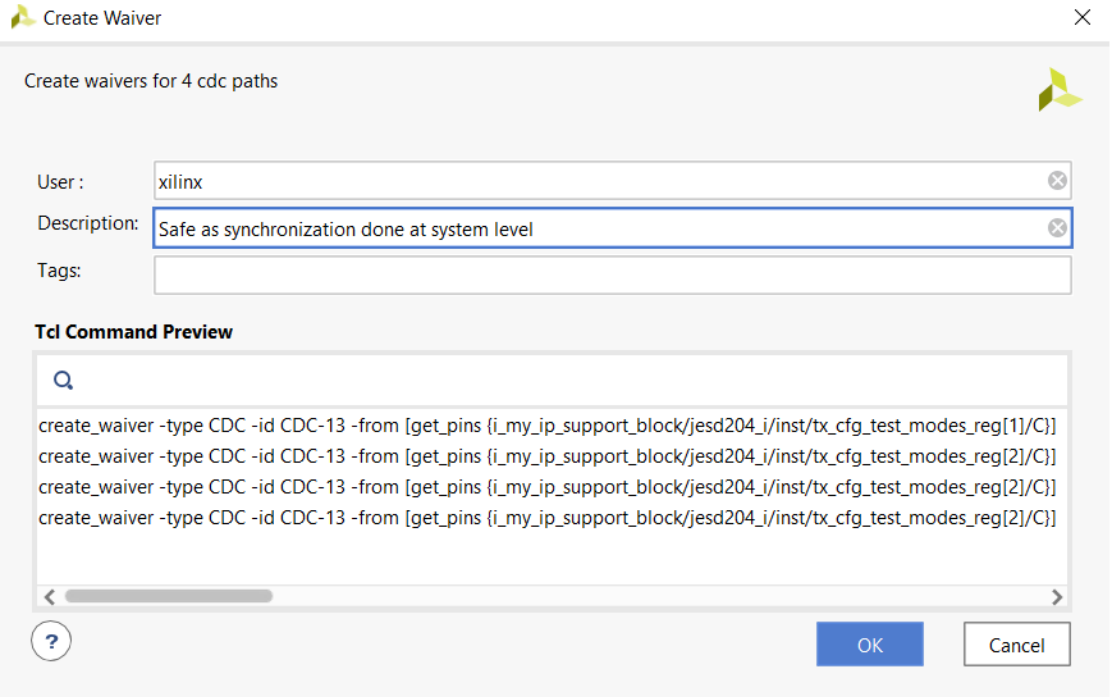
Waivers can be directly created from any Report DRC, Report Methodology, and Report CDC GUI result window. To waive violation(s) from the result window, select one or more violations and right-click to select Create Waiver from the context menu. In the following figure, four waivers are created from the four selected CDC violations.

Figure 32: **Waivers Created from CDC Violations**



Selecting Create Waiver opens the following widget.

Figure 33: **Create Waiver GUI**

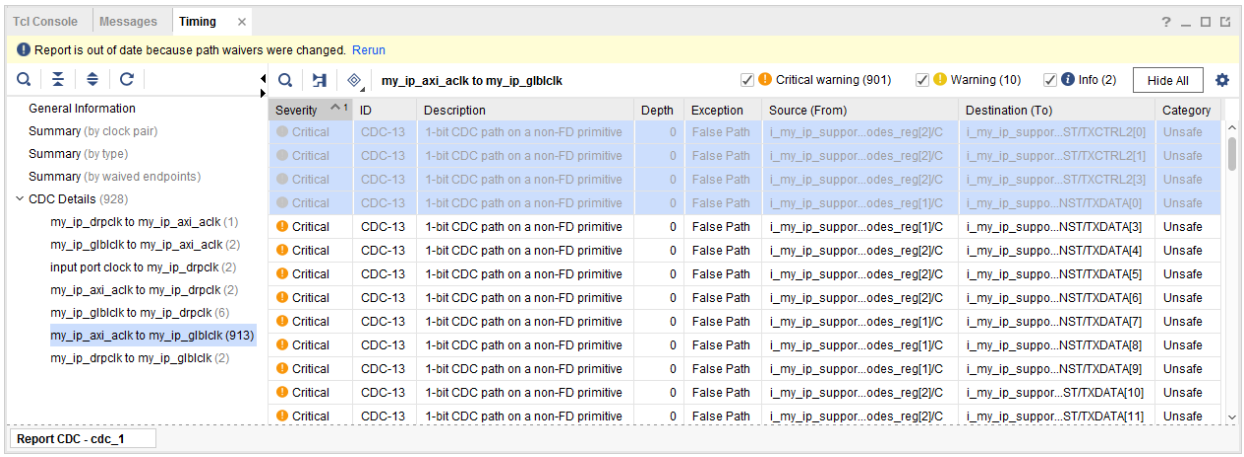


Even though Vivado tools populate the user name, the field is editable. The description is mandatory and it is recommended to provide some detailed information that can be reviewed by the design team. The field Tags is optional and can be used to provide an additional description through a string or a list of keywords. Its main purpose is documentation because it can be used, for instance, when searching through the XDC file or to filter waivers with the `get_waivers` command. The dialog box includes a preview of the Tcl commands that the GUI sends to the Tcl Console.

After the Create Waiver window is submitted, one `create_waiver` command is sent to the Tcl Console by the GUI for each violation that is waived. For DRC and Methodology violations, the `create_waiver` command generated by the GUI references the violation object but this is only a transitional form. The waiver engine converts the violation object into a fully descriptive waiver, referencing all the design objects involved in the violation. The waiver that is created never references the original violation object that it is built from. The timestamp is automatically added by the engine when the waiver is created.

After waivers are created from the GUI, the selected rows are grayed out as well as disabled and the report goes stale. This is a visual confirmation that some waivers have been created from this result window. After the GUI report is re-run, the waived violations are filtered out from the new result window.

Figure 34: **Disabled Rows after Waiver Creation**



***Note*:** The process to create a waiver from the GUI is the same for DRC and Methodology violations.

Creating Waivers from a Violation Object

The second method to create waivers is to use the DRC, Methodology, or CDC violation objects. This is the method the GUI uses when the `create_waiver` commands are sent to the Tcl Console.

The following syntax is used to create a waiver from one or more violation objects:

```
create_waiver -of_objects <ViolationObject(s)> -description <string> [-user
<name>]
```

The description is mandatory. When the user is not specified, the system uses the user id running the Vivado tools.

***Note*:** When multiple violation objects are specified, the system creates one waiver per violation. The waivers that are created do not reference the original violation objects. Instead, the waivers contain the list of strings and objects included in the violations.

The violation objects are returned through the `get_cdc_violations`, `get_drc_violations`, and `get_methodology_violations` commands. These commands only return objects when `report_cdc`, `report_drc`, and `report_methodology` have been run earlier. Use the command line option `-name` to get the list of violation objects from one of the GUI reports.

The following example code creates a waiver for all CDC-1 violations that have their startpoints inside the module `top/sync_1`:

```
report_cdc -name cdc_1
set vios [get_cdc_violations -name cdc_1 -filter {CHECK == CDC-1}]
foreach vio $vios {
    if {[regexp {^top/sync_1} [get_property STARTPOINT_PIN $vio]]} {
        create_waiver -of $vio -description {Safe by protocol}
    }
}
```

A waiver created from a violation object is built from all the objects and strings referenced inside the violation. This makes the waiver unique to that violation. If you want the waiver to cover multiple violations, you must export the waiver to an external file and edit the `create_waiver` command to replace single objects and strings with, for example, patterns and wildcards. See Creating DRC and Methodology Waivers for more information on using patterns and wildcards.

***Note*:** For a few DRC and Methodology checks some strings are automatically converted to wildcards, for example, UCIO-1, NSTD-1, TIMING-15, and TIMING-16. For TIMING-15 or TIMING-16, the Setup and Hold slack amount inside the violation is not relevant. When a waiver is created from a TIMING-15 or TIMING-16 violation object, the `create_waiver` command automatically replaces the string that represents the slack with a wildcard. This enables the waiver to waive the violation related to a specific object and regardless of the reported slack. Similar behavior applies to UCIO-1 and NSTD-1.

Creating Waivers from the Command Line

The waiver for a specific DRC and Methodology violation is unique. A violation is an aggregation of strings and/or various device and design objects, such as pins, cells, nets, Pblocks, sites, and tiles. Some DRC, Methodology, and CDC violations could just have one of those elements while others can have multiple elements. The order and content of all the strings and objects is important and must be preserved. When a waiver is created with the arguments specified in the wrong order, the waiver either never waives any violation or it could waive the wrong violation.

Before manually creating a waiver for a specific violation (for example, TIMING-14#1) or a class of violations (for example, TIMING-14), it is recommended to first create an example waiver from the GUI or from a violation object. Use the `write_waiver` or `write_xdc` commands to export the waiver. You can then relate the content of the waiver created by the system to the original violation and understand the order of strings and objects that need to be specified. You can extrapolate this information for other waivers with the same CDC, DRC, or Methodology ID (for example, TIMING-14).

There are two mandatory arguments for any CDC, DRC, and Methodology waiver:

- ID: The violation or check ID that is waived. The id is specified with `-id`. For example, CDC-1, TIMING-14, or PDRC-1569. Only one id can be specified at a time.
- Description: Support multi-lines string. Must provide enough information to be reviewed by the team. The description is specified with `-description`.

You can use the command line option `-type` to force the waiver type, CDC, DRC, or Methodology. A waiver created with the wrong type does not match any violation. For instance, to waive a CDC violation, the waiver type must be set as CDC. When the type is not specified, the system infers the type from the check id specified with `-id`. The user name can be overridden using the `-user` option. By default, the system uses the user id running Vivado Design Suite.

The waivers support the XDC scoping mechanism and the current instance can be changed before creating a waiver. In this case, the current instance information is saved along with the waiver and restored when the waivers are exported as XDC. When scoped waivers are created, it is recommended to use the command line option `-scope` to ensure that the wildcards are scoped.

A waiver is considered by the system as a duplicate if another waiver already exists with all the exact same arguments. To reduce the memory footprint and runtime, duplicate waivers are not saved and result in a message similar to the following:

```
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-13' is a duplicate and will not
be added
again.
```

Some DRC/Methodology checks such as RTSAT-* are read-only and cannot be waived. The list of DRC/Methodology checks that can be waived can be filtered by checking the property `IS_READ_ONLY` on the DRC/Methodology check objects. For example:

```
set allWaivableChecks [get_drc_checks -filter {!IS_READ_ONLY}]
set allWaivableChecks [get_methodology_checks -filter {!IS_READ_ONLY}]
```

The following message is an example of an error message generated by `create_waiver` when waiving a read-only check such as DRC RTSTAT-12:

```
ERROR: [Vivado_Tcl 4-934] Waiver ID 'RTSTAT-12' is READONLY and may not be
waived.
```

Creating DRC and Methodology Waivers

The number and types of additional arguments for `create_waiver` depends on the DRC and Methodology violation that needs to be waived. Few DRC and Methodology violations, such as TIMING-9, have no other arguments because the message is generic and non-specific. Other DRC and Methodology violations can include multiple strings and different types of objects.

***Note*:** It is not recommended to waive violations that do not reference any string or object such as TIMING-9 and TIMING-10.

The strings inside the violation are specified with `-string` inside the waiver. The device or design objects (pins, cells, nets, Pblocks, and sites) are specified with the `-objects` option. Each of these command line options should be specified as many times as the violation contains those elements.

When a waiver is created from the GUI or from a violation object, the waiver is defined to only waive that exact violation as it specifies all the strings and objects that make that violation unique. When the waiver is manually created, it is possible to widen the coverage of the waiver so that multiple violations can be waived from a single waiver. To expand a waiver to cover multiple violations:

- Use patterns for the `get_*` commands in place of a specific name
- Use a wildcard in place of a string or an object. Wildcards are special keywords such as '*' for 'any string' or '*PIN' for 'any pin' (refer to the following table). As long as an object of the same type matches the element found inside the violation for the same position, it is a match. When all the elements from the waiver match the violation, then the violation is waived.
- Specify a list of objects instead of a single object. As long as the object inside the violation matches one of the objects inside the list of objects at the same position inside the waiver, it is a match. When all the elements from the waiver match the violation, then the violation is waived.

For example: the command below waives a single TIMING-14 violation that references the cell `mux2_inst/mux_out_INST_0`:

```
create_waiver -id "TIMING-14" -description "Reviewed by the team" \
                  -objects [get_cells mux2_inst/mux_out_INST_0 ]
```

Suppose that the design has multiple cells `mux2_inst/mux_out_INST_*`, the above waiver could be modified to waive the TIMING-14 violations related to all those cells by using a pattern for the `get_cells` command:

```
create_waiver -id "TIMING-14" -description "Reviewed by the team" \
                   -objects [get_cells mux2_inst/mux_out_INST_* ]
```

The following table summarizes the keywords used as wildcards based on the object type.

Table 2: **Wildcard Keywords**

Object	Wildcard
Cell	*CELL
Net	*NET
Pin	*PIN
Port	*PORT
Site	*SITE
Tile	*TILE
BEL	*BEL
Package Bank	*PKGBANK
Clock Region	*CLKREGION
Clock	*CLOCK
Pblock	*PBLOCK

Table 2: **Wildcard Keywords** *(cont'd)*

Object	Wildcard
String	*

Note: `create_waiver -scope` forces the wildcards for pins and cells to be scoped to the current instance where the waiver is created. When creating scoped waivers, `-scope` ensures that wildcards for pins and cells do not match objects located at a higher level than the scope, which could result in waiving violations that must not be waived.

Creating CDC Waivers

CDC waivers are simpler to define because each CDC violation references only two pin(s) or port(s) objects for the source and destination elements. Use the command line options `-from`/`-to` for specifying the source and destination pins or ports. CDC waivers cannot be defined with `-string`/`-objects`.



IMPORTANT! *The CDC waivers are not sensitive to the source and destination clocks but only to the source and destination pins. As a result creating waivers from the GUI or from some CDC violation objects that reference the same source and destination pins but for different clock pairs results in a warning such as: WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-7' is a duplicate and will not be added again.*

The following command creates a CDC-1 waiver between the source pin `U_CORE/U00_TOP/sr_reg[3]/C` and the destination pin `U_CORE/U10/ar_reg[3]/CE`.

```
create_waiver -id {CDC-1} -description "CDC violations" \
-from [get_pins {U_CORE/U00_TOP/sr_reg[3]/C}] \
-to [get_pins {U_CORE/U10/ar_reg[3]/CE}]
```

If one of the command line options `-from` or `-to` is omitted, the waiver engine considers the missing option as a wildcard.

The following two commands are equivalent and waive all CDC-1 to the endpoint pin `U_CORE/U10/ar_reg[3]/CE`, regardless of the startpoint:

```
create_waiver -id {CDC-1} -description "CDC violations" \
-from {*PIN} \
-to [get_pins {U_CORE/U10/ar_reg[3]/CE}]
create_waiver -id {CDC-1} -description "CDC violations" \
-to [get_pins {U_CORE/U10/ar_reg[3]/CE}]
```

CDC Rules Precedence

By default, Report CDC only reports one violation per endpoint and per clock-pair. When multiple violations exist for an endpoint and for a specific clock-pair, only the CDC violation with the highest precedence is reported.

The CDC rules are sorted as shown in the table below from the highest to the lowest precedence.

Table 3: **CDC Rules Precedence**

Rule ID	CDC Topology	Severity	Category
CDC-18	Synchronized with HARD_SYNC Primitive	Info	Safe
CDC-13,14	1-bit and multi-bit CDC path on a non-FD primitive	Critical	Unsafe
CDC-17	MUX Hold Type	Warning	Safe
CDC-16	MUX Type	Warning	Safe
CDC-15	CE Type	Warning	Safe
CDC-26	LUTRAM read/write potential collision	Warning	Safe
CDC-7	Asynchronous Reset not synchronized	Critical	Unknown
CDC-1, 4	1-bit and Multi-bit CDC not synchronized	Critical	Unknown
CDC-12	Multi-Clock Fan-in	Critical	Unsafe
CDC-10	Combinatorial Logic detected between synchronizer	Critical	Unsafe
CDC-11	Fan-out from Launch Flop to destination domain	Critical	Unsafe
CDC-9	Asynchronous Reset synchronized with ASYNC_REG property	Info	Safe
CDC-6	Multi-bit synchronized with ASYNC_REG property	Warning	Unsafe
CDC-3	1-bit synchronized with ASYNC_REG property	Info	Safe
CDC-8	Asynchronous Reset synchronized with missing ASYNC_REG property	Warning	Safe
CDC-2,5	1-bit and multi-bit CDC synchronized with missing ASYNC_REG property	Warning	Safe

***Note*:** Some rules with severity Warning are listed above with a higher precedence than other critical rules because those rules cannot actually apply to the same endpoint due to different CDC topologies.

When an endpoint has multiple CDC violations, if the violation with the highest precedence is waived, the next violation is reported based on the precedence order.

To create waivers for a design, it could be convenient to report all the CDC violations for each endpoint in a single run, regardless of the rules precedence. Use the `report_cdc` command line option `-all_checks_per_endpoint` to generate an extensive report of all the CDC violations in the design.

***Note*:** `-all_checks_per_endpoint` is only available from the Tcl Console and is not supported in the Report CDC dialog window. However, the results of `-all_checks_per_endpoint` can be displayed in the Vivado IDE using the `-name` option.

Reporting the Waivers

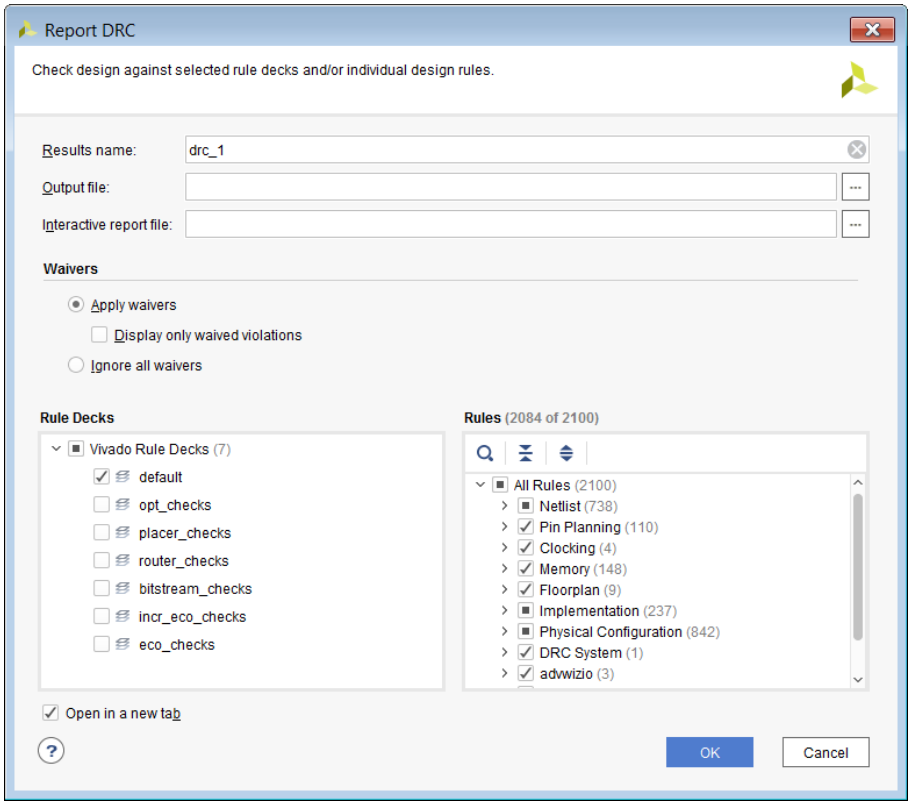
It is recommended to verify that only the expected violations have been waived. This must be done after the waivers are defined and before the final bitstream.

Report CDC, Report DRC, and Report Methodology commands support multiple reporting modes:

- By default, the `report_cdc`, `report_drc`, and `report_methodology` commands only report the violations that are not waived.
- Use `-waived` to force `report_cdc`, `report_drc`, and `report_methodology` commands to only report the violations that have been waived. The report must be reviewed to confirm that all the waived violations are expected.
- Use `-no_waiver` to force `report_cdc`, `report_drc`, and `report_methodology` commands to run without applying the waivers. In this mode, all violations are reported whether they are waived or not.

The three reporting modes are available from the command line and from the GUI Report dialog windows. The image below from Report DRC illustrates the selection of the reporting modes under the Waivers section. The same Waivers section is also available for Report CDC and Report Methodology widgets.

Figure 35: **Reporting Modes for Waivers**

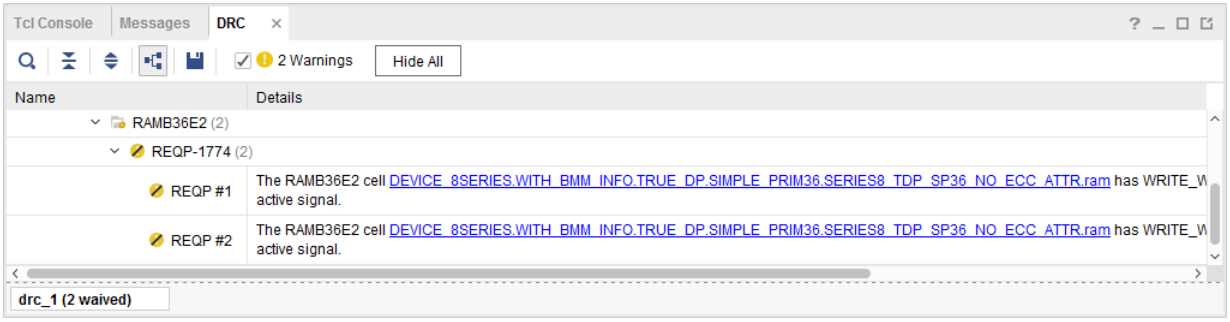


In the CDC, DRC, and Methodology GUI result windows, there are some visual differences when a result window contains waived violations, icons in front of each violations are different and the name of the result window includes the number of waived violations.

Note: Waivers cannot be created from the result window of waived CDC, DRC, and Methodology violations.

The example below illustrates a result window of waived DRCs with only two waived violations.

Figure 36: **Waived DRC Violations**



To get a summary report of all the waivers and the waived violations, use the `report_waivers` command. The report is only available from the Tcl Console.

`report_waivers` only reports statistics that are extracted by `report_cdc`, `report_drc`, and `report_methodology` commands. To get accurate statistics, it is necessary to run `report_cdc`, `report_drc`, and `report_methodology` before `report_waivers` and any time the waivers have been modified (added or deleted). The statistics are updated regardless of whether the reports are run from the command line or GUI. Failure to have updated statistics, `report_waivers` issues one or more of the following messages, depending on which of the CDC, DRC, or Methodology information is outdated:

```
WARNING: [Vivado_Tcl 4-972] Waiver counts for 'CDC' will be invalid because
report_cdc has not been run since waivers were changed; please run the
report_cdc
command.
WARNING: [Vivado_Tcl 4-972] Waiver counts for 'DRC' will be invalid because
report_drc has not been run since waivers were changed; please run the
report_drc
command.
WARNING: [Vivado_Tcl 4-972] Waiver counts for 'METHODOLOGY' will be invalid
because
report_methodology has not been run since waivers were changed; please run
the
report_methodology command.
```

The report from `report_waivers` includes a summary table and a detailed table for each of the CDC, DRC, and Methodology waivers. The table columns are:

- Total Vios: Total number of violations before the waivers apply. That is the number of violations that would be reported without waivers. Multi-bit rules are accounted for by the number of endpoints.
- Remaining Vios: Number of violations after the waivers apply. When no violation is waived, this number matches Total Vios. Multi-bit rules are accounted for by the number of endpoints.
- Waived Vios: Number of violations that are waived. When no violation is waived, this number is 0. Multi-bit rules are accounted for by the number of endpoints.
- Used Waivers: Number of waivers that have waived some violation(s). One waiver can waive multiple violations if it includes some patterns or wildcards.

- Set Waivers: Number of waivers that have been applied to the design. Ideally, the number of Used Waivers and Set Waivers should match. The numbers do not match when some waivers have been defined but they do not match any violation.

By default, only the rules that have some waivers defined are reported in the detailed tables. However, the first Summary table reports all the violations in the design.

Figure 37: **report_waivers Default Report**

```
-----------------
Table Of Contents
-----------------
1. REPORT SUMMARY
2. REPORT DETAILS (DRC: no waivers)
3. REPORT DETAILS (METHODOLOGY)
4. REPORT DETAILS (CDC)


-----------------
1. REPORT SUMMARY
-----------------
Waiver Type  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-----------  ----------  --------------  -----------  ------------  -----------
DRC          240         240             0            0             0
METHODOLOGY  166         162             4            4             4
CDC          929         923             6            6             6
Note: This report is based on the most recent report_drc/report_methodology/report_cdc runs.


---------------------------------
2. REPORT DETAILS (DRC)
---------------------------------
Rule  Severity  Description  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
----  --------  -----------  ----------  --------------  -----------  ------------  -----------


---------------------------------
3. REPORT DETAILS (METHODOLOGY)
---------------------------------
Rule     Severity  Description                 Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-------  --------  --------------------------  ----------  --------------  -----------  ------------  -----------
LUTAR-1  Warning   LUT drives async reset alert  49          45              4            4             4


---------------------------------
4. REPORT DETAILS (CDC)
---------------------------------
Rule    Severity  Description                                 Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
------  --------  ------------------------------------------  ----------  --------------  -----------  ------------  -----------
CDC-1   Critical  1-bit unknown CDC circuitry                  534         530             4            4             4
CDC-11  Critical  Fan-out from launch flop to destination clock  2          0               2            2             2

Note: Any 'Rule' which is flagged by '*' is an aggregating message and its counts are based on the number of objects represented,
      rather than the number of messages.
```

With the command line option `-show_msgs_with_no_waivers`, the detailed tables report all the checks that have some violations, regardless of whether a waiver exists or not for that particular rule.

Figure 38: **report_waivers -show_msgs_with_no_waivers**

```
-----------------
Table Of Contents
-----------------
1. REPORT SUMMARY
2. REPORT DETAILS (DRC: no waivers)
3. REPORT DETAILS (METHODOLOGY)
4. REPORT DETAILS (CDC)


-----------------
1. REPORT SUMMARY
-----------------
Waiver Type  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-----------  ----------  --------------  -----------  ------------  -----------
DRC          240         240             0            0             0
METHODOLOGY  166         162             4            4             4
CDC          929         923             6            6             6
Note: This report is based on the most recent report_drc/report_methodology/report_cdc runs.


---------------------------------
2. REPORT DETAILS (DRC)
---------------------------------
Rule       Severity          Description                                                                  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
---------  ----------------  ---------------------------------------------------------------------------  ----------  --------------  -----------  ------------  -----------
LOCE-1     Warning           Pblock ranges contradict LOC constraints on logic assigned to the Pblock     1           1               0            0             0
NSTD-1     Critical Warning  Unspecified I/O Standard                                                      113         113             0            0             0
RTSTAT-13  Critical Warning  Insufficient Routing                                                         1           1               0            0             0
UCIO-1     Critical Warning  Unconstrained Logical Port                                                   125         125             0            0             0


---------------------------------
3. REPORT DETAILS (METHODOLOGY)
---------------------------------
Rule       Severity  Description                      Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
---------  --------  -------------------------------  ----------  --------------  -----------  ------------  -----------
LUTAR-1    Warning   LUT drives async reset alert     49          45              4            4             4
TIMING-9   Warning   Unknown CDC Logic                1           1               0            0             0
TIMING-10  Warning   Missing property on synchronizer 1           1               0            0             0
TIMING-18  Warning   Missing input or output delay    115         115             0            0             0


---------------------------------
4. REPORT DETAILS (CDC)
---------------------------------
Rule    Severity  Description                                          Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
------  --------  ---------------------------------------------------  ----------  --------------  -----------  ------------  -----------
CDC-1   Critical  1-bit unknown CDC circuitry                          534         530             4            4             4
CDC-11  Critical  Fan-out from launch flop to destination clock        2           0               2            2             2
CDC-3   Info      1-bit synchronized with ASYNC_REG property           9           9               0            0             0
CDC-4   Critical  Multi-bit unknown CDC circuitry                      5           5               0            0             0
CDC-9   Info      Asynchronous reset synchronized with ASYNC_REG property  7           7               0            0             0
CDC-10  Critical  Combinational logic detected before a synchronizer  187         187             0            0             0
CDC-13  Critical  1-bit CDC path on a non-FD primitive                170         170             0            0             0
CDC-14  Critical  Multi-bit CDC path on a non-FD primitive            5           5               0            0             0
CDC-15  Warning   Clock enable controlled CDC structure detected      10          10              0            0             0

Note: Any 'Rule' which is flagged by '*' is an aggregating message and its counts are based on the number of objects represented,
      rather than the number of messages.
```

In addition to the above reports, `report_waivers` can export the list of waivers that have matched a CDC, DRC, or Methodology violation and those that did not match any violation. Use the options `-write_valid_waivers` to export waivers that have matched a violation and `-write_ignore_waivers` to export the waivers that did not match any violation. It is recommended to review the list of waivers that did not match any violation. When waivers that did not match are not expected, make sure that the waivers are correctly defined.

The options `-write_valid_waivers` and `-write_ignore_waivers` filter the waivers based on the information reported by the latest execution of `report_cdc`, `report_drc`, and `report_methodology` commands. When `report_drc` or `report_methodology` are run with a rule deck or a subset of checks, some waivers are ignored for the checks that have not been run. It is recommended to run all the DRC/Methodology checks before using `-write_valid_waivers` and `-write_ignore_waivers`. For example:

```
report_cdc -all_checks_per_endpoint
report_drc -checks [get_drc_checks]
report_methodology -checks [get_methodology_checks]
report_waivers -write_valid_waivers -file waivers_valid.xdc
report_waivers -write_ignored_waivers -file waivers_ignored.xdc
```

Exporting the Waivers

As part of the design constraints, the waivers are automatically saved inside the checkpoint and restored from the checkpoint. Waivers are saved inside both the plain XDC and binary constraints.

Use `write_xdc` and `write_waivers` commands to export the waivers as a standalone XDC file. The XDC can be reloaded inside Vivado tools with the `read_xdc` or `source` commands.

The `write_xdc` command exports all the waivers inside the XDC file along with all the design constraints. This includes the waivers defined by the user and also the AMD IP waivers. The constraints inside the XDC are in the same order as they have been applied to the design. To only export the waivers, use the command line option `-typewaiver`. For example:

```
write_xdc -type waiver -file waivers.xdc
```

IMPORTANT! *The IP waivers are identified with the option `create_waiver -internal`. User waivers must never use the option `create_waiver -internal`. This option is exclusively reserved for AMD IP waivers.*

The `write_waivers` command differs from `write_xdc` because it only exports the user waivers and provides more control and granularity. The AMD IP waivers are not exported through `write_waivers`. By default, all the user CDC, DRC, and Methodology waivers are exported. Use the option `-type` to only export CDC, DRC, or Methodology waivers.

For example, the command below exports all CDC waivers to the file `waivers_cdc.xdc`:

```
write_waivers -type CDC -file waivers_cdc.xdc
```

All the waivers for a particular check id can be exported with the option `-id`. The example below export all the waivers for the Methodology check TIMING-15:

```
write_waivers -id TIMING-15 -file waivers_timing_15.xdc
```

The following table summarizes the differences between `write_xdc` and `write_waivers` commands with regards to the user waivers and AMD IP waivers.

Table 4: **Exporting the Waivers**

Vivado Command	Export User Waivers	Export AMD IP Waivers
`write_xdc`	Yes	Yes
`write_waivers`	Yes	No

Other Waiver Commands

The `get_waivers` command returns a collection of waiver objects. Waivers can be returned by type, name, or pattern.

The following commands return all the DRC waivers:

```
get_waivers -type DRC
get_waivers -filter {TYPE == DRC}
```

The following commands return all the DRC DPIR-2 waivers:

```
get_waivers DPIR-2#*
get_waivers -filter {ID == DPIR-2}
get_waivers -filter {NAME =~ DPIR-2#*}
```

***Note*:** AMD IP waivers are not returned by the `get_waivers` command.

The `delete_waivers` command deletes user waiver objects. The collection of waiver objects must be built from `get_waivers`.

The following command deletes all the waivers:

```
delete_waivers [get_waivers]
```

The following command deletes all the CDC waivers:

```
delete_waivers [get_waivers -type CDC]
```

***Note*:** AMD IP waivers cannot be deleted.

Configurable Report Strategies

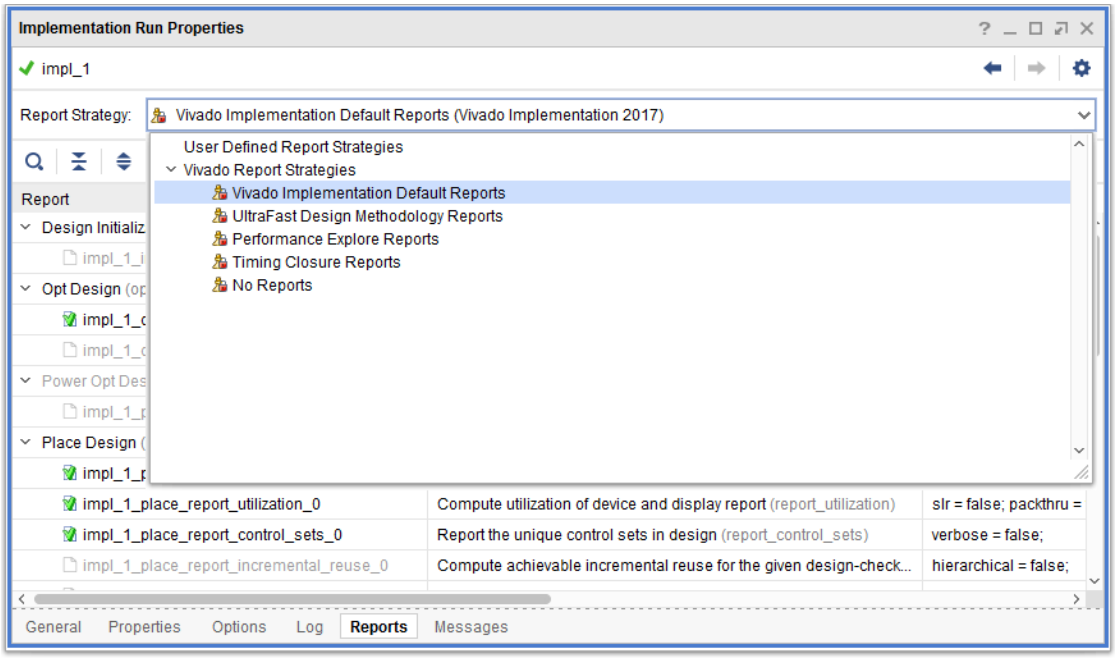
Configurable Report Strategies provide the ability to select which reporting commands are run after each step of the synthesis and implementation runs in the Vivado project mode. Depending on the design stage, the design complexity and the user preferences, a different set of reports need to be automatically generated and are commonly reviewed. Several pre-defined synthesis and implementation report strategies are available by default. In addition, the Vivado IDE allows the creation of new report strategies which are saved with the user preferences along with any other Vivado IDE settings.

Setting Run Report Strategies

By default, all synthesis and implementation runs will use their corresponding default Reports Strategy. To set a different Report Strategy:

1. Select a run in the Design Runs window.
2. Select the Reports tab in the Run Properties window.
3. Select the strategy from the Report Strategy drop down list.

Figure 39: **Selecting the Report Strategy for an Implementation Run**



Two Flow categories are available and each of them provide several pre-defined report strategies as well as any user-defined strategies.

Table 5: **Flows and Report Strategies**

Flow	Report Strategy	Note
Synthesis	Vivado Synthesis Default Reports	Only runs the utilization report at the end of synthesis
	No Reports	Best strategy to minimize runtime
Implementation	Vivado Implementation Default Reports	Runs same reports as in Vivado releases prior to 2017.3
	UltraFast Design Methodology Reports	Runs all reports recommended in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)
	Performance Explore Reports	Same as the default reports, plus additional timing reports after `phys_opt_design`
	Timing Closure Reports	Same as UltraFast Design Methodology Reports, plus several `report_design_analysis` and `report_qor_suggestions` reports
	No Reports	Best strategy to minimize runtime

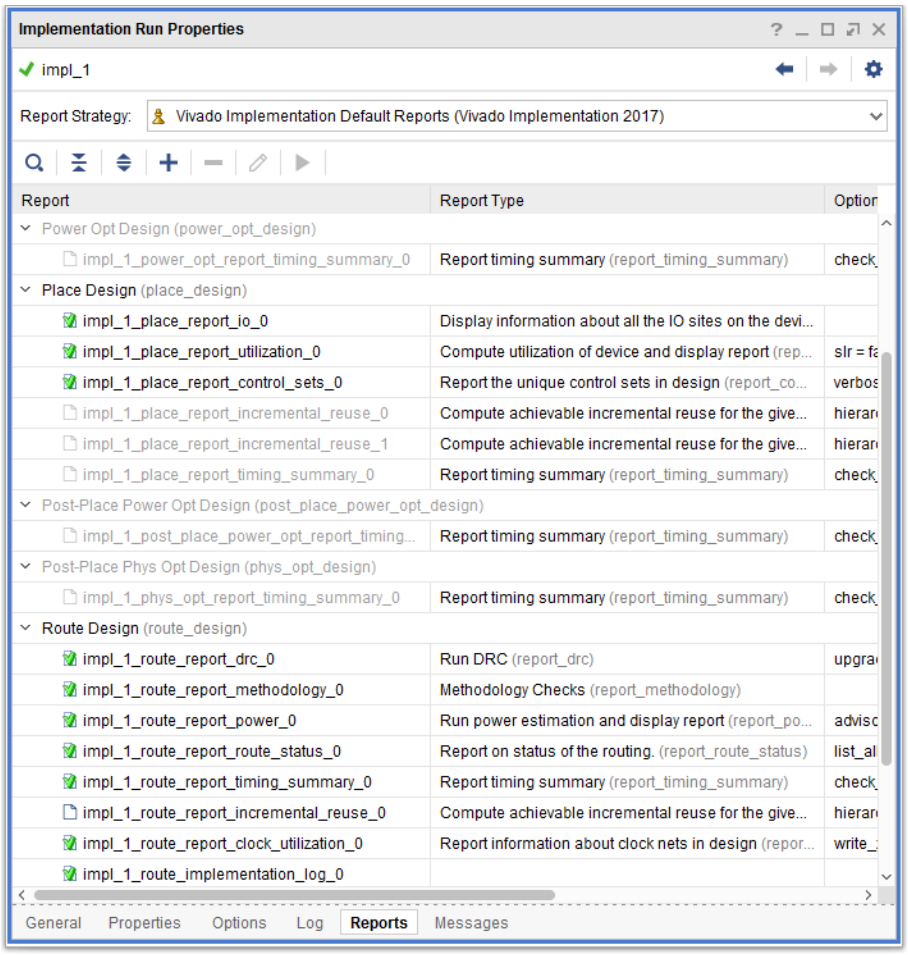
Before launching the run, some reports are greyed out due to one of the following reasons:

- They are associated with a disabled flow step
- They are disabled by the user

After the run has completed, all available reports have a green check mark and can be opened by double-clicking on them from the Reports tab. Some reports are not available for one of the following reasons:

- They are associated with a disabled flow step
- They are disabled by the user
- They are enabled in the Incremental Compile runs only

Figure 40: **Viewing Generated Run Reports**



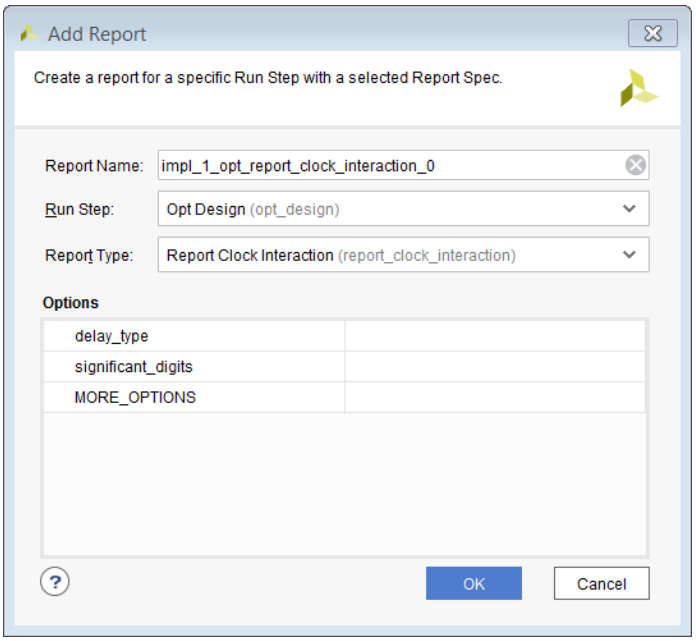
TIP: *Several timing summary reports are only generated when setting the legacy project property* `ENABLE_OPTIONAL_RUNS_STA`. *AMD reserves the right to eliminate this property in a future release. Example:* `set_property ENABLE_OPTIONAL_RUNS_STA 1 [current_project]`

Editing Run Report Strategies

After selecting the Report Strategy for a Run, you can decide to add, delete, or edit reports by selecting a report and clicking on the corresponding buttons in the Reports tab, or right-clicking over any existing report, in the Run Properties window:

- Add Report (+): Select the Run Step and the Report Type, then review and edit the report options. If an option is not visible, you can add it through the MORE_OPTIONS field. The default unique report name is based on the run name, the flow step, and the report command name.

Figure 41: **Adding a Report to a Run Report Strategy**



- Delete Report (-): This button deletes the selected report from the Run Report Strategy. This operation cannot be undone.
- Edit Report ( ): Edit the report name, enable or disable the report, or edit the report options.

To Enable or Disable a report, select the report, right-click on it, and use the contextual pop-up menu.

After a Run has completed, you can add new reports for a particular step or enable a report that was previously disabled. In this situation, you must click on the play button to generate the report.

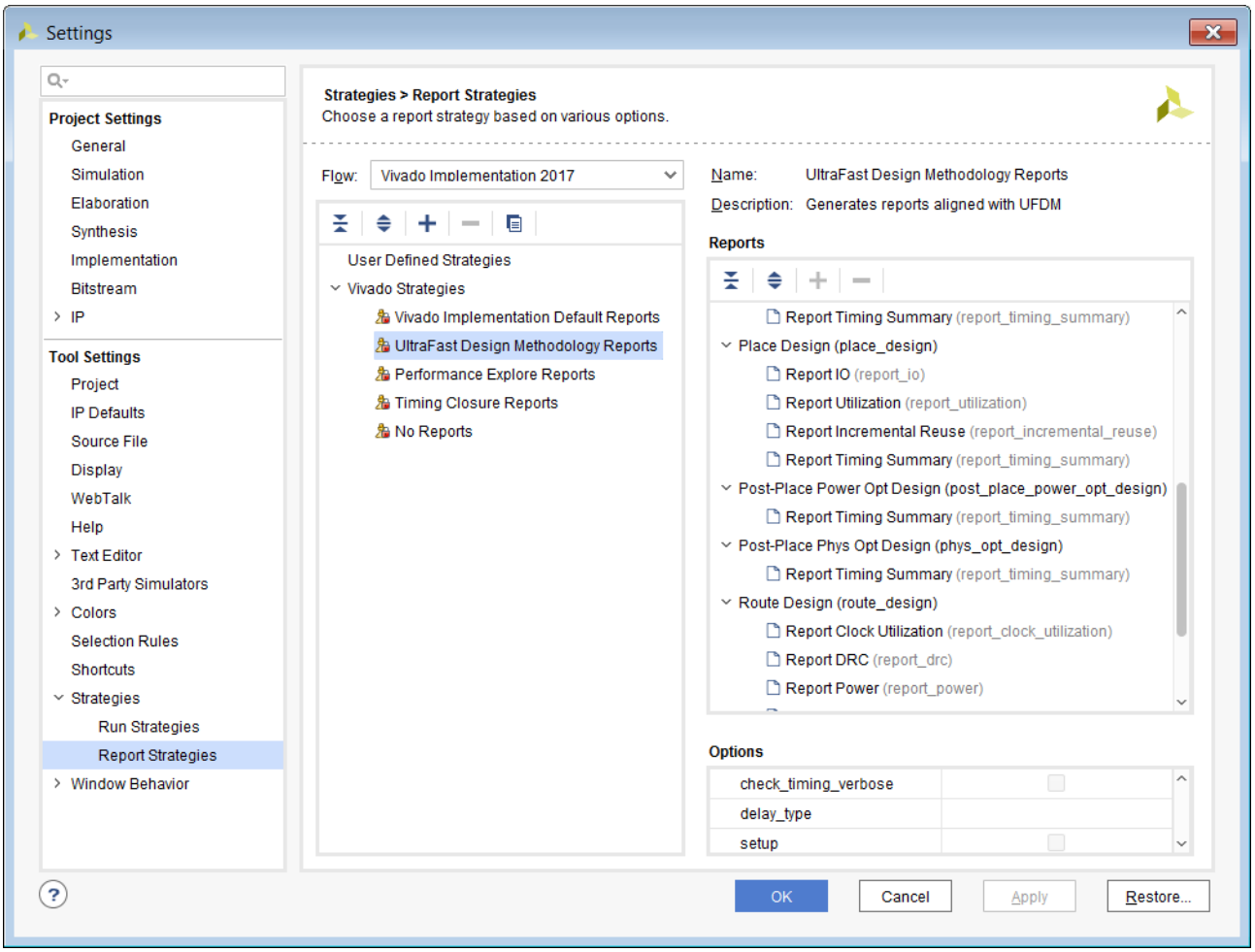
IMPORTANT! *When generating a new report after the run has completed, Vivado opens the checkpoint of the corresponding flow step in the background in order to generate the report file. This operation blocks most Vivado IDE functions, including the Tcl Console, until the report has been successfully generated. Generating the report can take a few minutes to over an hour depending on your design size and the complexity of the report.*

Any modification to a Report Strategy made for a specific run cannot be saved as a new report strategy. Instead, you must create new report strategies or modify any existing user-defined report strategies in the Project Settings window as explained in the next section.

Creating New Report Strategies

New Report Strategies must be created in the Vivado IDE Project Settings window, under Tool Settings and the subsection Strategies. Similarly, any existing user-defined Report Strategy can only be permanently modified in the Project Settings window. The pre-defined Report Strategies and the default association of a Run Strategy with a Report Strategy cannot be modified.

Figure 42: **Report Strategies Configuration in the Settings Window**



To create a new Report Strategy:

1. Click on "**+**" in the Strategy Window and perform the following steps:
   a. Specify the name of the strategy.
   b. Select the target Flow, Synthesis or Implementation.
   c. Provide a description (optional).
   d. Click **OK**.

Or,

2. Select an existing strategy, click on the **copy** button, and edit the strategy name. Then:
   a. Add a report using the "**+**" button.
3. Edit the report options by selecting the report and editing the options. If an option is not available, it can be added in the MORE_OPTIONS field.
4. Remove a report using the "**-**" button.
5. After all the edits are complete, click **OK**.

Each flow category provides several pre-defined Report Strategies. See Table 5: Flows and Report Strategies.

Chapter 4

Timing Analysis

The AMD Vivado™ integrated design environment (IDE) provides several reporting commands to verify that your design meets all 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. Report Timing Summary provides a comprehensive overview of all the timing checks, and shows enough information to allow you to start analyzing and debugging any timing issue. For more information, see Logic Analysis in the IDE.

You can generate this report in a window, write it to a file, or 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. The other timing reports provide more details on a particular situation and can scope the analysis to some logic by using filters and scoping capabilities.

Before adding timing constraints to your design, you must understand the fundamentals of timing analysis, and the terminology associated with it. This chapter discusses some of key concepts used by the Vivado IDE timing engine.

Terminology

- The *launch edge* is the active edge of the source clock that launches the data.
- The *capture edge* is the active edge on the destination clock that captures the data.
- The *source clock* is also referred to as the *launch clock*.
- The *destination clock* is also referred to as the *capture clock*.
- The *setup requirement* is the relationship between the launch edge and the capture edge that defines the most restrictive setup constraint.
- The *setup relationship* is the setup check verified by the timing analysis tool.
- The *hold requirement* is the relationship between the launch edge and capture edge that defines the most restrictive hold constraint.
- The *hold relationship* is the hold check verified by the timing analysis tool.

Timing Paths

Timing paths are defined by the connectivity between the instances of the design. In digital designs, timing paths are formed by a pair of sequential elements controlled by the same clock, or by two different clocks.

Common Timing Paths

The most common paths in any design are:

- Path from Input Port to Internal Sequential Cell
- Internal Path from Sequential Cell to Sequential Cell
- Path from Internal Sequential Cell to Output Port
- Path from Input Port to Output Port

Path from Input Port to Internal Sequential Cell

In a path from an input port to a sequential cell, the data:

- Is launched outside the device by a clock on the board.
- Reaches the device port after a delay called the input delay [Synopsys Design Constraints (SDC) definition].
- Propagates through the device internal logic before reaching a sequential cell clocked by the destination clock.

Internal Path from Sequential Cell to Sequential Cell

In an internal path from sequential cell to sequential cell, the data:

- Is launched inside the device by a sequential cell, which is clocked by the source clock.
- Propagates through some internal logic before reaching a sequential cell clocked by the destination clock.

Path from Internal Sequential Cell to Output Port

In a path from an internal sequential cell to an output port, the data:

- Is launched inside the device by a sequential cell, which is clocked by the source clock.
- Propagates through some internal logic before reaching the output port.
- Is captured by a clock on the board after an additional delay called the output delay (SDC definition).

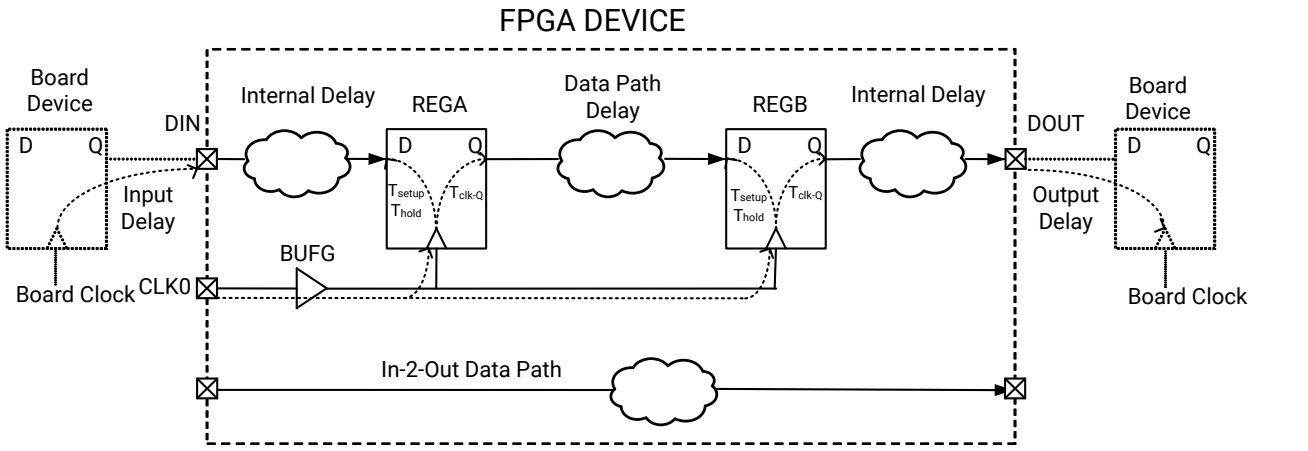
Path from Input Port to Output Port

In a path from an input port to output port, the data traverses the device without being latched. This type of path is also commonly called an *in-to-out path*. The input and output delays reference clock can be a virtual clock or a design clock.

Timing Paths Example

The following figure shows the paths described above. In this example, the design clock CLK0 can be used as the board clock for both DIN and DOUT delay constraints.

Figure 43: **Timing Paths Example**



Timing Path Sections

Each timing path is composed of three sections:

- Source Clock Path
- Data Path
- Destination Clock Path

Source Clock Path

The source clock path is the path followed by the source clock from its source point (typically an input port) to the clock pin of the launching sequential cell. For a timing path starting from an input port, there is no source clock path.

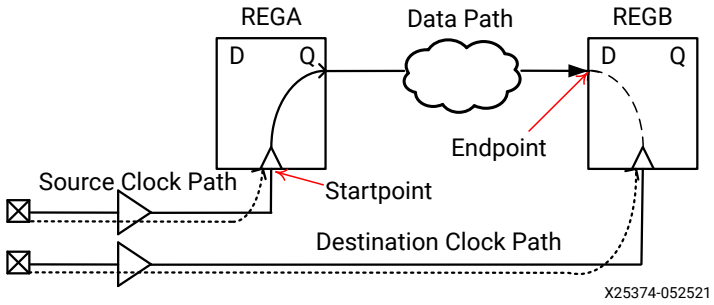
Data Path

The data path is the section of the timing path where the data propagates, between the path startpoint and the path endpoint. The following definitions apply: (1) A path startpoint is a sequential cell clock pin or a data input port; and (2) A path endpoint is a sequential cell data input pin or a data output port.

Destination Clock Path

The destination clock path is the path followed by the destination clock from its source point, typically an input port, to the clock pin of the capturing sequential cell. For a timing path ending at an output port, there is no destination clock path. Destination Clock Path shows the three sections of a typical timing path.

Figure 44: **Typical Timing Path**



Launch and Capture Edges

When transferring between sequential cells or ports, the data is:

- Launched by one of the edges of the source clock, which is called the launch edge.
- Captured by one of the edges of the destination clock, which is called the capture edge.

In a typical timing path, the data is transferred between two sequential cells within one clock period. In that case: (1) the launch edge occurs at 0ns; and (2) the capture edge occurs one period after.

The following section explains how the launch and capture edges define the setup and hold relationships used for timing analysis.

Timing Analysis Key Concepts

Max and Min Delay Analysis

Timing analysis is the static verification that a design's timing behavior will be predictable once loaded and run on hardware. It considers a range of manufacturing and environmental variations that are combined into delay models that are grouped by timing corners and corner variations. It is sufficient to analyze timing against all the recommended corners, and for each corner, to perform all the checks under the most pessimistic conditions. For example, a design targeted to a AMD FPGA must pass the four following analyses:

- Max delay analysis in Slow Corner
- Min delay analysis in Slow Corner
- Max delay analysis in Fast Corner
- Min delay analysis in Fast Corner

Depending on the check performed, the delays that represent the most pessimistic situation are used. This is the reason why the following checks and delay types are always associated:

- Max Delay with Setup and Recovery Checks
- Min Delay with Hold and Removal Checks

Max Delay with Setup and Recovery Checks

- The worst-case delays (slowest delays) of a given corner are used for the source clock path and data/reset path accumulated delay.
- The best-case delays (fastest delays) of the same corner are used for the destination clock path accumulated delay.

Min Delay with Hold and Removal Checks

- The best-case delays (fastest delays) of a given corner are used for the source clock path and data/reset path accumulated delay.
- The worst-case delays (slowest delays) of the same corner are used for the destination clock path accumulated delay.

When mapped to the various corners, these checks become:

- Setup/Recovery (Max Delay Analysis)
- Hold/Removal (Min Delay Analysis)

Setup/Recovery (Max Delay Analysis)

- source clock (Slow_max), datapath(Slow_max), destination clock (Slow_min)
- source clock (Fast_max), datapath(Fast_max), destination clock (Fast_min)

Hold/Removal (Min Delay Analysis)

- source clock (Slow_min), datapath (Slow_min), destination clock (Slow_max)
- source clock (Fast_min), datapath (Fast_min), destination clock (Fast_max)

Delays from different corners are never mixed on a same path for slack computation.

Most often, setup or recovery violations occur with Slow corner delays, and hold or removal violations occur with Fast corner delays. However, since this is not always true (especially for I/O timing) AMD recommends that you perform both analyses on both corners.

Setup/Recovery Relationship

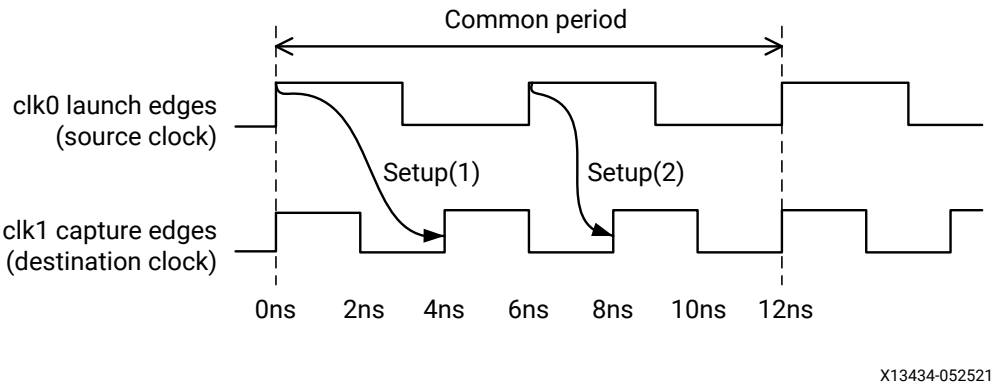
The setup check is performed only on the most pessimistic setup relationship between two clocks. By default, this corresponds to the smallest positive delta between the launch and capture edges. For example, consider a path between two flip-flops that are sensitive to the rising edge of their respective clock. The launch and capture edges of this path are the clock rising edges only.

The clocks are defined as follows:

- `clk0` has a period of 6 ns with first rising @ 0 ns and falling edge @ 3 ns.
- `clk1` has a period of 4 ns with first rising @ 0 ns and falling edge @ 2 ns.

As the following figure shows, there are two unique setup relationships: Setup(1) and Setup(2).

Figure 45: **Setup Relationships**



The smallest positive delta from `clk0` to `clk1` is 2 ns, which corresponds to Setup(2). The Common Period is 12 ns, which corresponds to the time between two simultaneous alignments of the two clocks.

TIP: *The relationships are established when considering the ideal clock waveforms, that is, before applying the insertion delay from the clock root to the flip-flop clock pin.*

IMPORTANT! *If the common period cannot be found over 1000 cycles of both clocks, the worst setup relationship over these 1000 cycles is used for timing analysis. For such case, the two clocks are called unexpandable, or clocks with no common period. The analysis will likely not correspond to the most pessimistic scenario. You must review the paths between these clocks to assess their validity and determine if they can be treated as asynchronous paths instead.*

Once the path requirement is known, the path delays, the clocks uncertainty and the setup time are introduced to compute the slack. The typical slack equation is:

Data Required Time (setup)	**=**	**capture edge time** **+ destination clock path delay** **- clock uncertainty** **- setup time**
Data Arrival Time (setup)	=	launch edge time + source clock path delay + datapath delay
Slack (setup)	=	Data Required Time - Data Arrival Time

As the equation shows, a positive setup slack occurs when the data arrives before the required time.

The recovery check is similar to the setup check, except that it applies to asynchronous pins such as preset or clear. The relationships are established the same way as for setup, and the slack equation is the same except that the recovery time is used instead of the setup time.

Hold/Removal Relationship

The hold check (also called hold relationship) is directly connected to the setup relationship. While the setup analysis ensures that data can safely be captured under the most pessimistic scenario, the hold relationship ensures that:

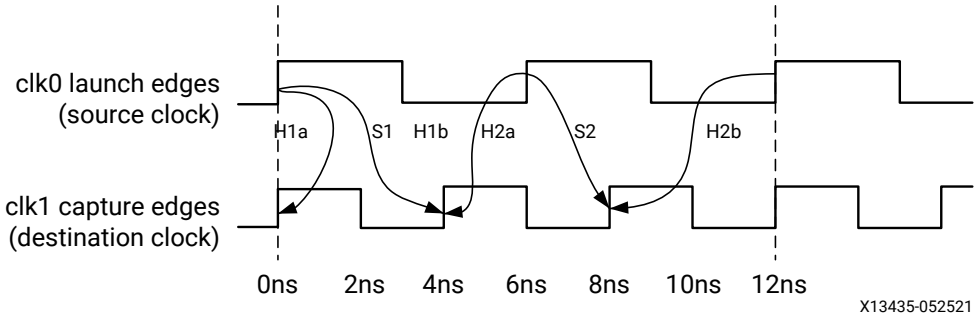
- The data sent by the setup launch edge cannot be captured by the active edge before the setup capture edge (H1a and H2a corresponding to setup edges S1 and S2 respectively in the following figure).
- The data sent by the next active source clock edge after the setup launch edge cannot be captured by the setup capture edge (H2a and H2b corresponding to setup edges S1 and S2 respectively in the following figure).

During hold analysis, the timing engine reports only the most pessimistic hold relationship between any two clocks. The most pessimistic hold relationship is not always associated with the worst setup relationship. The timing engine must review all possible setup relationships and their corresponding hold relationships to identify the most pessimistic hold relationship.

For example, consider the same path as in the setup relationship example. Two unique setup relationships exist.

The following figure illustrates the two hold relationships per setup relationship.

Figure 46: **Hold Relationships per Setup Relationship**



The greatest hold requirement is 0 ns, which corresponds to the first rising edge of both source and destination clocks.

Once the path requirement is known, the path delays, the clocks' uncertainty, and the hold time are introduced to compute the slack. The typical slack equation is:

Data Required Time (hold) = capture edge time + destination clock path delay + clock uncertainty + hold time

Data Arrival Time (hold) = launch edge time + source clock path delay + datapath delay

Slack (hold) = Data Arrival Time - Data Required Time

As the equation shows, the hold slack is positive when the new data arrives after the required time.

The removal check is similar to the hold check, except that it applies to asynchronous pins such as preset or clear. The relationships are established the same way as for hold, and the slack equation is the same except that the removal time is used instead of the hold time.

Path Requirement

The path requirement represents the difference in time between the capture edge and the launch edge of a timing path.

For example, when considering the same path and clocks as in the previous section, the following path requirements exist:

```
Setup Path Requirement (S1) = 1*T(clk1) - 0*T(clk0) = 4ns
Setup Path Requirement (S2) = 2*T(clk1) - 1*T(clk0) = 2ns
```

The corresponding hold relationships are:

- Corresponding to setup S1

```
Hold Path Requirement (H1a) = (1-1)*T(clk1) - 0*T(clk0) = 0ns
Hold Path Requirement (H1b) = 1*T(clk1) - (0+1)*T(clk0) = -2ns
```

- Corresponding to setup S2

```
Hold Path Requirement (H2a) = (2-1)*T(clk1) - 1*T(clk0) = -2ns
Hold Path Requirement (H2b) = 2*T(clk1) - (1+1)*T(clk0) = -4ns
```

The timing analysis is performed only with the two most pessimistic requirements. In the example above, these are:

- The setup requirement S2
- The hold requirement H1a

Clock Phase Shift

A clock phase-shift corresponds to a delayed clock waveform with respect to a reference clock due to special hardware in the clock path. In AMD FPGAs, clock phase-shift is usually introduced by the MMCM or PLL primitives, when their output clock property `CLKOUT*_PHASE` is non-zero.

MMCM/PLL Phase Shift Modes

During timing analysis, clock phase-shift can be modeled in two different ways by setting the MMCM/PLL PHASESHIFT_MODE property, as described in the following table.

Table 6: **MMCM/PLL PHASESHIFT_MODE Properties**

PHASESHIFT_MODE Property	Phase-Shift Modeling	Comment
WAVEFORM	Clock Waveform Modification	`set_multicycle_path -setup` constraints are usually needed to adjust the timing path requirement on clock-domain-crossing paths from or to the phase-shifted clock.
LATENCY	MMCM/PLL Insertion Delay	No additional multicycle path constraint is needed.

The default MMCM/PLL clock phase-shift mode varies across AMD FPGA families. However, the default mode can be overridden by the user on a per PLL/MMCM basis.

Table 7: **Default MMCM/PLL Clock Phase Shift Handling**

Technology	Default MMCM/PLL Clock Phase Shift Handling
7 series	Clock waveform modification (WAVEFORM)
AMD UltraScale™	Clock waveform modification (WAVEFORM)
AMD UltraScale+™	MMCM/PLL insertion delay (LATENCY)
AMD Versal™ adaptive SoC	MMCM/PLL insertion delay (LATENCY)

IMPORTANT! *The MMCM/PLL PHASESHIFT_MODE property does not affect the device configuration. It only impacts the slack computation by the Static Timing Analysis engine. Because the timing constraints might need to be adjusted between WAVEFORM/LATENCY modes, the tool QoR could subsequently be affected as well.*

IMPORTANT! *When a pin phase-shift is defined on any of the `CLKOUTx` pins and multiple clocks reach the input pins of the MMCM/PLL, the mode PHASESHIFT_MODE=LATENCY is invalid and triggers a "Warning Timing 38-437" warning. In such cases, the MMCM/PLL should be configured to use the mode PHASESHIFT_MODE=WAVEFORM.*

The use of PHASESHIFT_MODE=LATENCY is particularly convenient when introducing skew between two clocks to meet timing. No additional multicycle path constraint is needed for adjusting the timing path requirement when setting the clock phase-shift to negative, null, or positive.

For legacy designs migrated from 7 series or UltraScale to UltraScale+, when the property PHASESHIFT_MODE is not set on the MMCM/PLL, the default behavior applies and the MMCM/PLL clock phase-shift is modeled as delay latency instead of clock edge shift. In this case, all multicycle path constraints that were specified in the legacy designs to account for a clock phase-shift need to be reviewed and usually removed. Such constraints can easily be identified by running the methodology checks (`report_methodology`). TIMING-31 flags multicycle paths between clocks where one of the clocks is phase-shifted and generated by a MMCM/PLL with `PHASESHIFT_MODE` set to `LATENCY`.

The Clocking wizard and the High Speed SelectIO wizard both provide options to force the clock phase-shift modeling on each MMCM/PLL. The property PHASESHIFT_MODE is automatically saved inside the IP XDC.

Phase Shift in Timing Reports

A positive phase shift moves the source clock edge forward, delaying the clock edge. A negative phase shift moves the source clock edge backward. The modification of the clock waveform result in potentially different clock edges being used by the static timing analysis for the source and capture clocks.

In the examples below, clock `clkout0` (period 10 ns) is auto-derived by a MMCM.

- No phase shift

```
vivado% set_property CLKOUT0_PHASE 0.000 [get_cells qpll/plle2_adv_inst]
vivado% report_timing
...
(clock clkout0 rise edge) 0.000 0.000 r
...
        MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)
                                -5.411   5.903 r mmcm_inst/mmcm_adv_inst/
CLKOUT0
...
```

  The source clock edge is 0.0 ns.

- Positive phase shift of 12.0 with `PHASESHIFT_MODE=WAVEFORM`

```
vivado% set_property CLKOUT0_PHASE 12.000 [get_cells qpll/plle2_adv_inst]
vivado% report_timing
...
(clock clkout0 rise edge) 0.333 0.333 r
...
    MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)
                            -5.411   5.903 r mmcm_inst/mmcm_adv_inst/CLKOUT0
...
```

  The source clock edge is delayed by 0.333 ns (10 ns / 360 * 12.0).

- Positive phase shift of 12.0 with `PHASESHIFT_MODE=LATENCY`

```
vivado% set_property CLKOUT0_PHASE 12.000 [get_cells qpll/plle2_adv_inst]
vivado% report_timing
...
(clock clkout0 rise edge) 0.000 0.000 r
...
    MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)
                            -5.078   6.236 r mmcm_inst/mmcm_adv_inst/
CLKOUT0
...
```

  The MMCM insertion delay is increased by 0.333 ns (10 ns / 360 * 12.0). The source clock edge is 0.0 ns.

- Negative phase shift of -15.0 with `PHASESHIFT_MODE=WAVEFORM`

```
vivado% set_property CLKOUT0_PHASE -15.000 [get_cells qpll/plle2_adv_inst]
vivado% report_timing
...
(clock clkout0 rise edge) -0.417 -0.417 r
...
     MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)
                             -5.411   5.903 r mmcm_inst/mmcm_adv_inst/CLKOUT0
...
```

  The source clock edge is moved backward by -0.417 ns (10 ns / 360 * -15.0).

- Negative phase shift of -15.0 with `PHASESHIFT_MODE=LATENCY`

```
vivado% set_property CLKOUT0_PHASE -15.000 [get_cells qpll/plle2_adv_inst]
vivado% report_timing
...
(clock clkout0 rise edge) 0.000 0.000 r
...
     MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)
                           -5.828    5.486 r mmcm_inst/mmcm_adv_inst/
CLKOUT0
...
```

The MMCM insertion delay is decreased by 0.417 ns (10 ns / 360 * -15.0). The source clock edge is 0.0 ns.

Phase Shift in Clock Reports

The clock phase shift information is provided in the Clock Report (`report_clocks` command). When a MMCM/PLL clock is phase-shifted and the MMCM/PLL has the `PHASESHIFT_MODE` property set to `LATENCY`, then the auto-derived clock is marked with the attribute S (pin phase-shifted with Latency mode). In addition, the clock details under the section `Generated Clocks` of the clock report show the amount of pin phase-shift that is accounted in the MMCM/PLL insertion delay.

Note: Only the delay corresponding to the auto-derived clock phase-shift is reported. The amount of phase-shift coming from the MMCM/PLL block is not included in the auto-derived clock waveform definition.

In the example below, the MMCM has the property PHASESHIFT_MODE set to LATENCY. The auto-derived clock `clk_out1_clk_wiz_0` has no phase shift defined for the MMCM pin `CLKOUT0` but the clock `clk_out2_clk_wiz_0` has a -90 degrees phase shift defined for the MMCM pin `CLKOUT2`.

```
Attributes
  P: Propagated
  G: Generated
  A: Auto-derived
  R: Renamed
  V: Virtual
  I: Inverted
  S: Pin phase-shifted with Latency mode


Clock               Period(ns)  Waveform(ns)    Attributes  Sources
clk_in1             10.000      {0.000 5.000}   P           {clk_in1}
clk_out1_clk_wiz_0  10.000      {0.000 5.000}   P,G,A       {clknetwork/
inst/mmcme3_adv_inst/CLKOUT0}
clk_out2_clk_wiz_0  10.000      {0.000 5.000}   P,G,A,S     {clknetwork/
inst/mmcme3_adv_inst/CLKOUT2}


=====================================================
Generated Clocks
=====================================================

Generated Clock     : clk_out1_clk_wiz_0
```

```
Master Source        : clknetwork/inst/mmcme3_adv_inst/CLKIN1
Master Clock         : clk_in1
Multiply By          : 1
Generated Sources    : {clknetwork/inst/mmcme3_adv_inst/CLKOUT0}

Generated Clock      : clk_out2_clk_wiz_0
Master Source        : clknetwork/inst/mmcme3_adv_inst/CLKIN1
Master Clock         : clk_in1
Multiply By          : 1
Pin Phase Shift(ns) : -2.5  (-90 degrees)
Generated Sources    : {clknetwork/inst/mmcme3_adv_inst/CLKOUT2}
```

Clock Skew and Uncertainty

Skew and uncertainty both impact setup and hold computations and slack.

Skew Definition

Clock skew is the insertion delay difference between the destination clock path and the source clock path: (1) from their common point in the design; (2) to, respectively, the endpoint and startpoint sequential cell clock pins.

In the equation below:

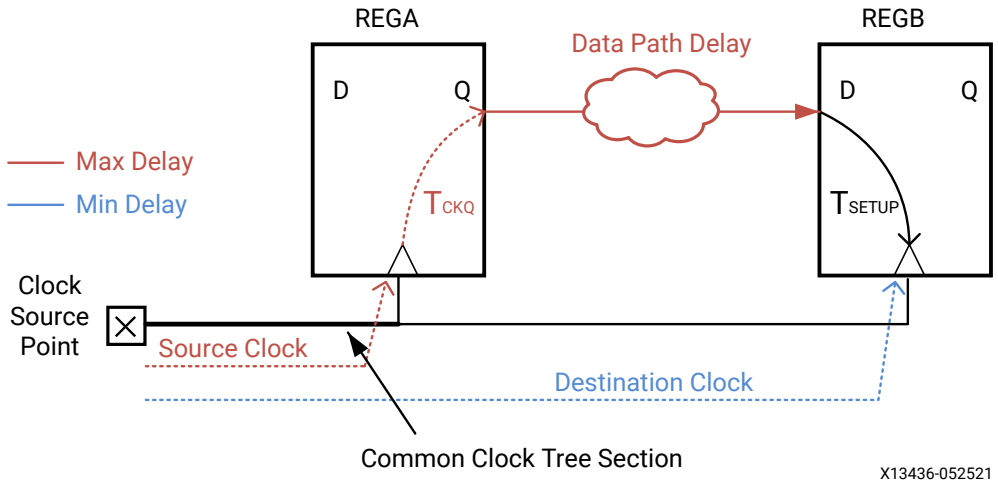
- `Tcj` is the delay from the common node to the endpoint clock pin.
- `Tci` is the delay from the common node to the startpoint clock pin:

```
Tskewi,j = Tcj- Tci
```

Clock Pessimism Removal

A typical timing path report shows the delay details of both source and destination clock paths, from their root to the sequential cell clock pins. As explained below, the source and destination clocks are analyzed with a different delay, even on their common circuitry.

Figure 47: **Common Clock Tree Section**



This delay difference on the common section introduces some additional pessimism in the skew computation. To avoid unrealistic slack computation, this pessimism is compensated by a delay called the Clock Pessimism Removal (CPR) value.

```
Clock Pessimism Removal (CPR) = common clock circuitry (max delay - min
delay)
```

The CPR is added or subtracted to the skew depending on the type of analysis performed:

- Max Delay Analysis (Setup/Recovery)

  CPR is added to the destination clock path delay.

- Min Delay Analysis (Hold/Removal)

  CPR is subtracted from the destination clock path delay.

The Vivado Design Suite timing reports clock skew for each timing path as shown below (hold analysis in this case):

- DCD - Destination Clock Delay
- SCD - Source Clock Delay
- CPR - Clock Pessimism Removal

```
Clock Path Skew: 0.301ns (DCD - SCD - CPR)
Destination Clock Delay (DCD): 2.581ns
Source Clock Delay (SCD): 2.133ns
Clock Pessimism Removal (CPR): 0.147ns
```

In many cases, the CPR accuracy changes before and after routing. For example, let's consider a timing path where the source and destination clocks are the same clock, and the startpoint and endpoint clock pins are driven by the same clock buffer.

Before routing, the common point is the clock net driver, that is, the clock buffer output pin. CPR compensates only for the pessimism from the clock root to the clock buffer output pin.

After routing, the common point is the last routing resource shared by the source and destination clock paths in the device architecture. This common point is not represented in the netlist, so the corresponding CPR cannot be directly retrieved by subtracting common clock circuitry delay difference from the timing report. The timing engine computes the CPR value based on device information not directly exposed to the user.

Optimistic Skew

AMD FPGA devices provide advanced clocking resources such as dedicated clock routing trees and Clock Modifying Blocks (CMB). Some of the CMBs have the capability to compensate the clock tree insertion delay by using a Phase Lock Loop circuit (present in PLL or MMCM primitives). The amount of compensation is based on the insertion delay present on the feedback loop of the PLL. In many cases, a PLL (or MMCM) drives several clock trees with the same type of buffer, including on the feedback loop. As the device can be large, the insertion delay on all these clock tree branches does not always match the feedback loop delay. The clocks driven by a PLL become over-compensated when the feedback loop delay is bigger than the source or destination clock delay. In this case, the sign of the CPR changes and it effectively removes skew optimism from the slack value. This is needed in order to ensure that there is no artificial skew at the common node of any timing path clocks during the analysis.

RECOMMENDED: *Always use the CPR compensation during timing analysis to preserve the slack accuracy and the overall timing signoff quality.*

Clock Uncertainty

Clock uncertainty is the total amount of possible time variation between any pair of clock edges. The uncertainty consists of the computed clock jitter (system, input, 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`).

For primary clocks, the jitter is defined by `set_input_jitter` and `set_system_jitter`. For clock generators such as MMCM and PLL, the tool computes the jitter based on user-specified jitter on its source clock and its configuration. For other generated clocks (such as flop based clock dividers), the jitter is the same as that of its source clock.

The user-specified clock uncertainty is added to the uncertainty computed by the AMD Vivado™ Design Suite timing engine. For generated clocks (such as from MMCM, PLL, and flop-based clock dividers), uncertainty specified by the user on source clock does not propagate through the clock generators.

For more information on jitter and phase error definitions, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

The clock uncertainty has two purposes:

- Reserve some amount of margin in the slack numbers for representing any noise on the clock that could affect the hardware functionality. Because the delay and jitter numbers are conservative, AMD does not recommend adding extra uncertainty to ensure proper hardware functionality.
- Over-constrain the paths related to a clock or a clock pair during one or several implementation steps. This increases the QoR margin that can be used to help the next steps to close timing on these paths. By using clock uncertainty, the clock waveforms and their relationships are not modified, so the rest of the timing constraints can still apply properly.

Pulse Width Checks

The pulse width checks are some rule checks on the signal waveforms when they reach the hardware primitives after propagation through the device. They usually correspond to functional limits dictated by the circuitry inside the primitive. For example, the minimum period check on a DSP clock pin ensures that the clock driving a DSP instance does not run at higher frequency than what is tolerated by the internal DSP.

The pulse width checks do not affect synthesis or implementation. Their analysis must be performed once before the bitstream generation like any other design rule check provided by the Vivado Design Suite.

When a pulse width violation occurs, it is due to an inappropriate clock definition (pulse width and period checks) or an inappropriate clock topology that induces too much skew (`max_skew` check). You must review the AMD FPGA data sheet of the target device to understand the operation range of the primitive where the violation occurs. In the case of a skew violation, you must simplify the clock tree or place the clock resources closer to the violating pins.

Reading a Timing Path Report

The timing path report provides the information needed to understand what causes a timing violation. The following sections describe the Timing Path Report.

The Timing Path Summary displays the important information from the timing path details. 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 Key Concepts.

Timing Path Summary Header Examples

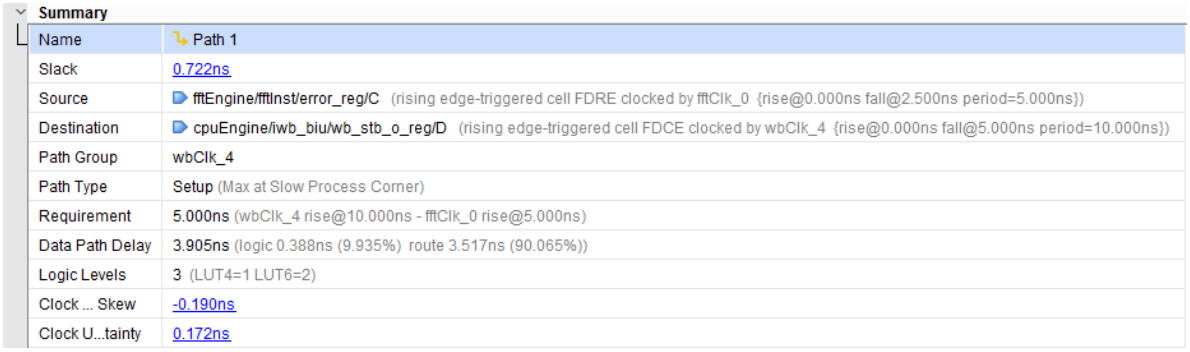
The following figure shows an example of the Timing Path Summary Header in a text report.

Figure 48: **Timing Path Summary Header in Text Report**

```
Slack (MET) :             0.722ns  (required time - arrival time)
  Source:                 fftEngine/fftInst/error_reg/C
                            (rising edge-triggered cell FDRE clocked by fftClk_0  {rise@0.000ns fall@2.500ns period=5.000ns})
  Destination:            cpuEngine/iwb_biu/wb_stb_o_reg/D
                            (rising edge-triggered cell FDCE clocked by wbClk_4  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             wbClk_4
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            5.000ns  (wbClk_4 rise@10.000ns - fftClk_0 rise@5.000ns)
  Data Path Delay:        3.905ns  (logic 0.388ns (9.935%)  route 3.517ns (90.065%))
  Logic Levels:           3  (LUT4=1 LUT6=2)
  Clock Path Skew:        -0.190ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    -1.471ns = ( 8.529 - 10.000 )
    Source Clock Delay      (SCD):    -2.117ns = ( 2.883 - 5.000 )
    Clock Pessimism Removal (CPR):    -0.836ns
  Clock Uncertainty:      0.172ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Discrete Jitter          (DJ):    0.077ns
    Phase Error              (PE):    0.120ns
```

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

Figure 49: **Timing Path Summary Header in Vivado IDE**



Timing Path Summary Header Information

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 subsections of the timing path report.

- Source

  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.

  When 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

  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

  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

  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

  The timing path requirement, when the startpoint and endpoint are controlled by the same clock, or by clocks with no phase-shift, is typically:

  - One clock period for setup/recovery analysis.
  - 0 ns for hold/removal analysis.

  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 the timing path requirement is derived from the timing constraints, Timing Paths.

- Data Path Delay

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

  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

  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)

  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
  - For min delay analysis (hold/removal), the maximum delay values are used.

- Source Clock Delay (SCD)

  The 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.
  - For min delay analysis (hold/removal), the minimum delay values are used.

- Clock Pessimism Removal (CPR)

  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

  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 IDE timing engine.

- Total System Jitter (TSJ)

The combined system jitter applied to both source and destination clocks. To modify the system jitter globally, use the `set_system_jitter` constraint. The virtual clocks are ideal and therefore do not have any system jitter.

- Total Input Jitter (TIJ)

  The combined input jitter of both source and destination clocks.

  To define the input jitter for each primary clock individually, use the `set_input_jitter` constraint. 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 clock uncertainty and jitter, see this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903).

- Discrete Jitter (DJ)

  The amount of jitter introduced by hardware primitives such as MMCM or PLL.

  The Vivado IDE timing engine computes this value based on the configuration of these cells.

- Phase Error (PE)

  The amount of phase variation between two clock signals introduced by hardware primitives such as MMCM or PLL.

  The Vivado IDE timing engine automatically provides this value and adds it to the clock uncertainty

- User Uncertainty (UU)

  The additional uncertainty specified by the `set_clock_uncertainty` constraint.

  For more information on how to use this command, see this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

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

- Inter-SLR Compensation

  The additional margin required for safely reporting paths that cross SLR boundaries in AMD 7 series SSI devices only.

- Input Delay

  The input delay value specified by the `set_input_delay` constraint on the input port. This line does not show for paths that do not start from an input port.

- Output Delay

  The output delay value specified by the `set_output_delay` constraint on the output port. This line does not show for paths that do not end to an output port.

- Timing Exception

  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 information on timing exceptions and their precedence rules, see Report Exceptions.

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 into three sections:

- Source Clock Path

  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

  The circuitry traversed by the data from the startpoint to the endpoint.

- Destination Clock Path

  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
- 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 final 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
```

Timing Path Details In Text Report

Following is an example of the Source Clock, Data and Destination Clock Paths in the text report. Because the path is covered by a simple period constraint of 5 ns, the source clock launch edge starts at 0 ns and the destination clock capture edge starts at 5 ns.

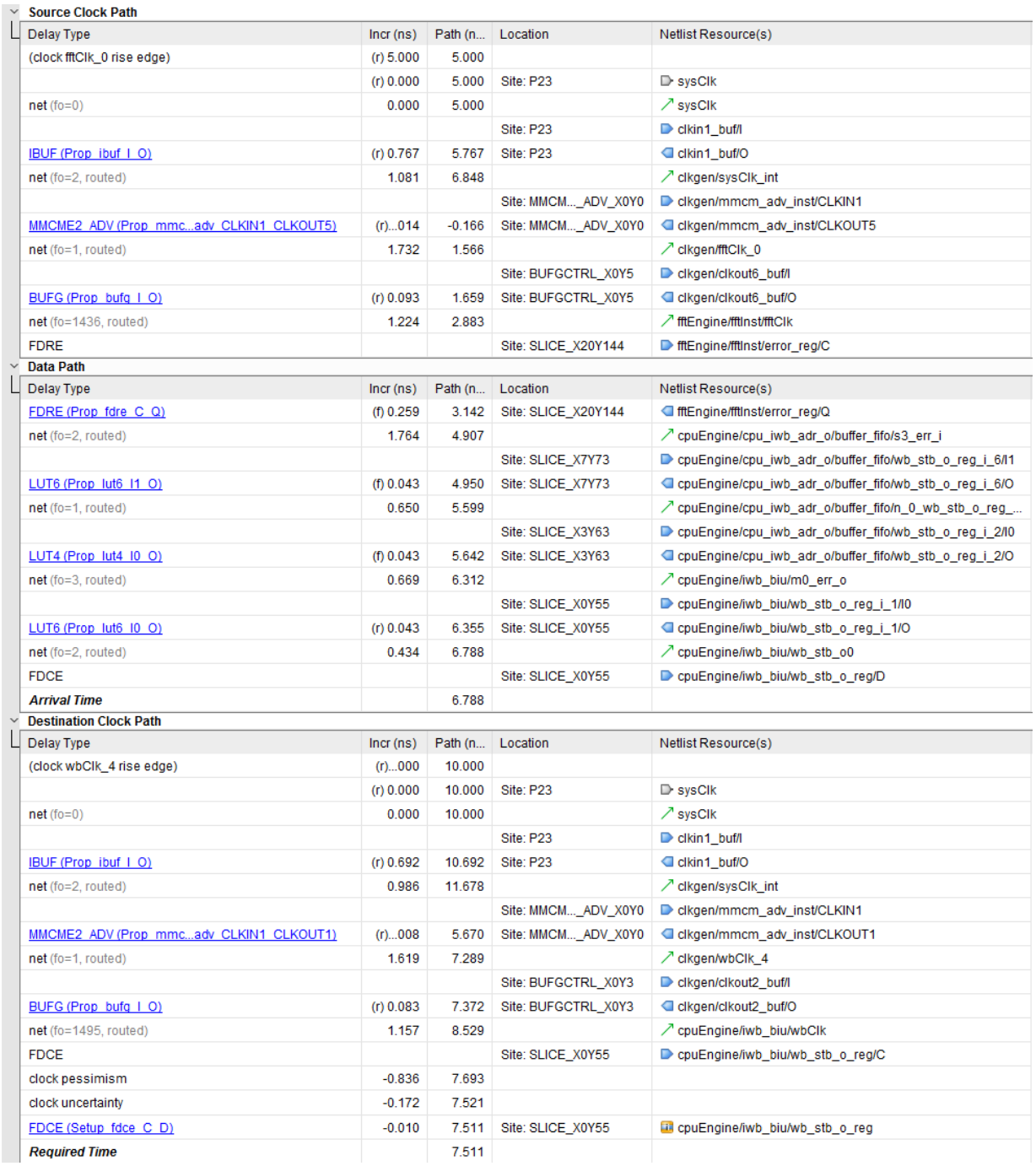
Figure 50: **Timing Path Details in Text Report**

```
  Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------------------------------    -------------------
                     (clock fftClk_0 rise edge)
                                                    5.000     5.000 r
  P23                                               0.000     5.000 r  sysClk (IN)
                     net (fo=0)                     0.000     5.000    sysClk
  P23                IBUF (Prop_ibuf_I_O)           0.767     5.767 r  clkin1_buf/O
                     net (fo=2, routed)             1.081     6.848    clkgen/sysClk_int
  MMCME2_ADV_X0Y0    MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT5)
                                                   -7.014    -0.166 r  clkgen/mmcm_adv_inst/CLKOUT5
                     net (fo=1, routed)             1.732     1.566    clkgen/fftClk_0
  BUFGCTRL_X0Y5      BUFG (Prop_bufg_I_O)           0.093     1.659 r  clkgen/clkout6_buf/O
                     net (fo=1436, routed)          1.224     2.883    fftEngine/fftInst/fftClk
  SLICE_X20Y144      FDRE                                         r  fftEngine/fftInst/error_reg/C
-------------------------------------------------------------------    -------------------
  SLICE_X20Y144      FDRE (Prop_fdre_C_Q)           0.259     3.142 f  fftEngine/fftInst/error_reg/Q
                     net (fo=2, routed)             1.764     4.907    cpuEngine/cpu_iwb_adr_o/buffer_fifo/s3_err_i
  SLICE_X7Y73        LUT6 (Prop_lut6_I1_O)          0.043     4.950 f  cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_6/O
                     net (fo=1, routed)             0.650     5.599    cpuEngine/cpu_iwb_adr_o/buffer_fifo/n_0_wb_stb_o_reg_i_6
  SLICE_X3Y63        LUT4 (Prop_lut4_I0_O)          0.043     5.642 f  cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_2/O
                     net (fo=3, routed)             0.669     6.312    cpuEngine/iwb_biu/m0_err_o
  SLICE_X0Y55        LUT6 (Prop_lut6_I0_O)          0.043     6.355 r  cpuEngine/iwb_biu/wb_stb_o_reg_i_1/O
                     net (fo=2, routed)             0.434     6.788    cpuEngine/iwb_biu/wb_stb_o0
  SLICE_X0Y55        FDCE                                         r  cpuEngine/iwb_biu/wb_stb_o_reg/D
-------------------------------------------------------------------    -------------------

                     (clock wbClk_4 rise edge)     10.000    10.000 r
  P23                                               0.000    10.000 r  sysClk (IN)
                     net (fo=0)                     0.000    10.000    sysClk
  P23                IBUF (Prop_ibuf_I_O)           0.692    10.692 r  clkin1_buf/O
                     net (fo=2, routed)             0.986    11.678    clkgen/sysClk_int
  MMCME2_ADV_X0Y0    MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT1)
                                                   -6.008     5.670 r  clkgen/mmcm_adv_inst/CLKOUT1
                     net (fo=1, routed)             1.619     7.289    clkgen/wbClk_4
  BUFGCTRL_X0Y3      BUFG (Prop_bufg_I_O)           0.083     7.372 r  clkgen/clkout2_buf/O
                     net (fo=1495, routed)          1.157     8.529    cpuEngine/iwb_biu/wbClk
  SLICE_X0Y55        FDCE                                         r  cpuEngine/iwb_biu/wb_stb_o_reg/C
                     clock pessimism               -0.836     7.693
                     clock uncertainty             -0.172     7.521
  SLICE_X0Y55        FDCE (Setup_fdce_C_D)         -0.010     7.511    cpuEngine/iwb_biu/wb_stb_o_reg
-------------------------------------------------------------------
                     required time                            7.511
                     arrival time                            -6.788
-------------------------------------------------------------------
                     slack                                    0.722
```

Timing Path Details in Vivado IDE

The Timing Path Details in the Vivado IDE, as shown in the he following figure, shows the same information as is shown in the text report, seen in the previous figure.

Figure 51: **Timing Path Details in Vivado IDE**



Source Clock Path

Delay Type	Incr (ns)	Path (n...	Location	Netlist Resource(s)
(clock fftClk_0 rise edge)	(r) 5.000	5.000		
	(r) 0.000	5.000	Site: P23	sysClk
net (fo=0)	0.000	5.000		sysClk
			Site: P23	clkin1_buf/I
IBUF (Prop_ibuf_I_O)	(r) 0.767	5.767	Site: P23	clkin1_buf/O
net (fo=2, routed)	1.081	6.848		clkgen/sysClk_int
			Site: MMCM..._ADV_X0Y0	clkgen/mmcm_adv_inst/CLKIN1
MMCME2_ADV (Prop_mmc...adv_CLKIN1_CLKOUT5)	(r)...014	-0.166	Site: MMCM..._ADV_X0Y0	clkgen/mmcm_adv_inst/CLKOUT5
net (fo=1, routed)	1.732	1.566		clkgen/fftClk_0
			Site: BUFGCTRL_X0Y5	clkgen/clkout6_buf/I
BUFG (Prop_bufg_I_O)	(r) 0.093	1.659	Site: BUFGCTRL_X0Y5	clkgen/clkout6_buf/O
net (fo=1436, routed)	1.224	2.883		fftEngine/fftInst/fftClk
FDRE			Site: SLICE_X20Y144	fftEngine/fftInst/error_reg/C

Data Path

Delay Type	Incr (ns)	Path (n...	Location	Netlist Resource(s)
FDRE (Prop_fdre_C_Q)	(f) 0.259	3.142	Site: SLICE_X20Y144	fftEngine/fftInst/error_reg/Q
net (fo=2, routed)	1.764	4.907		cpuEngine/cpu_iwb_adr_o/buffer_fifo/s3_err_i
			Site: SLICE_X7Y73	cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_6/I1
LUT6 (Prop_lut6_I1_O)	(f) 0.043	4.950	Site: SLICE_X7Y73	cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_6/O
net (fo=1, routed)	0.650	5.599		cpuEngine/cpu_iwb_adr_o/buffer_fifo/n_0_wb_stb_o_reg_...
			Site: SLICE_X3Y63	cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_2/I0
LUT4 (Prop_lut4_I0_O)	(f) 0.043	5.642	Site: SLICE_X3Y63	cpuEngine/cpu_iwb_adr_o/buffer_fifo/wb_stb_o_reg_i_2/O
net (fo=3, routed)	0.669	6.312		cpuEngine/iwb_biu/m0_err_o
			Site: SLICE_X0Y55	cpuEngine/iwb_biu/wb_stb_o_reg_i_1/I0
LUT6 (Prop_lut6_I0_O)	(r) 0.043	6.355	Site: SLICE_X0Y55	cpuEngine/iwb_biu/wb_stb_o_reg_i_1/O
net (fo=2, routed)	0.434	6.788		cpuEngine/iwb_biu/wb_stb_o0
FDCE			Site: SLICE_X0Y55	cpuEngine/iwb_biu/wb_stb_o_reg/D
Arrival Time		6.788		

Destination Clock Path

Delay Type	Incr (ns)	Path (n...	Location	Netlist Resource(s)
(clock wbClk_4 rise edge)	(r)...000	10.000		
	(r) 0.000	10.000	Site: P23	sysClk
net (fo=0)	0.000	10.000		sysClk
			Site: P23	clkin1_buf/I
IBUF (Prop_ibuf_I_O)	(r) 0.692	10.692	Site: P23	clkin1_buf/O
net (fo=2, routed)	0.986	11.678		clkgen/sysClk_int
			Site: MMCM..._ADV_X0Y0	clkgen/mmcm_adv_inst/CLKIN1
MMCME2_ADV (Prop_mmc...adv_CLKIN1_CLKOUT1)	(r)...008	5.670	Site: MMCM..._ADV_X0Y0	clkgen/mmcm_adv_inst/CLKOUT1
net (fo=1, routed)	1.619	7.289		clkgen/wbClk_4
			Site: BUFGCTRL_X0Y3	clkgen/clkout2_buf/I
BUFG (Prop_bufg_I_O)	(r) 0.083	7.372	Site: BUFGCTRL_X0Y3	clkgen/clkout2_buf/O
net (fo=1495, routed)	1.157	8.529		cpuEngine/iwb_biu/wbClk
FDCE			Site: SLICE_X0Y55	cpuEngine/iwb_biu/wb_stb_o_reg/C
clock pessimism	-0.836	7.693		
clock uncertainty	-0.172	7.521		
FDCE (Setup_fdce_C_D)	-0.010	7.511	Site: SLICE_X0Y55	cpuEngine/iwb_biu/wb_stb_o_reg
Required Time		7.511		

The information on the path is displayed in five columns when the standard flow is used or six columns when the Incremental Compile is used:

- Location

  Where the cell or port is placed on the device.

- Delay Type

  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 the load are not placed.
  - Estimated: The driver or the load or both are placed. A partially routed net is also reported as estimated.
  - Routed: The driver and the load are both placed, plus the net is fully routed.

- Incr(ns) (text report) / Delay (IDE report)

  The value of the incremental delay associated to a unisim primitive timing arc or a net. It can also show of a constraint such as input/output delay or clock uncertainty.

- Path(ns) (text report) / Cumulative (IDE report)

  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)

  The name of the netlist object traversed.

- Pin Reuse (Incremental Compile only)

  Indicates whether the path is being reused from the reference run. Applicable values are ROUTING, PLACEMENT, MOVED, and NEW.

Each incremental delay is associated to one of the following edge senses:

- `r` (rising)
- `f` (falling)

The initial sense of the edge is determined by the launch or capture edge used for the analysis. It can be inverted 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 helpful in identifying that an overly-tight timing path requirement comes from a clock edge inversion along the source or destination clock tree.

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.



IMPORTANT! *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.

IMPORTANT! *Report Timing Summary does not cover the bus skew constraints. To report the bus skew constraints, you must run the* `report_bus_skew` *command separately on the command line. There is no GUI support for this command.*

For a comprehensive Timing Signoff Verification methodology, see this link in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

Reports

This chapter discusses various report types:

- General Reports covers various netlist-based reports as well as power and DRC.
- Timing Reports covers all the reports that are related to timing constraints.
- Design Closure Reports covers reports that are used in the design closure phase. While these also contain timing and netlist-based analysis, there is a focus on how the two interact to cause timing closure challenges.

General Reports

Report Utilization

The Report Utilization Report helps you analyze the utilization of the design with different resources, at the hierarchical, user-defined Pblocks, or SLR level. You can generate the Utilization Report during various steps in the flow with the `report_utilization` Tcl command. (For details on Tcl command usage see the *Vivado Design Suite Tcl Command Reference Guide* (UG835).) The report details shown below are for UltraScale and UltraScale+ families. It includes the device used for the run and utilization for the following (additional items might appear in each category):

- Slice Logic
  - LUT
  - MuxFx
  - Register
  - Slice
  - LUT as Memory
  - LUT Flip-Flop pairs
  - LUT as Logic

- Memory
  - BlockRam
  - FIFO
- DSPs
- I/O Resources
- Clocking Resources
- Specific Device Resources. For example,
  - STARTUPE2
  - XADC
- Primitive type count sorted by usage
- Black Boxes
- Instantiated Netlists
- SLR Crossing Utilization

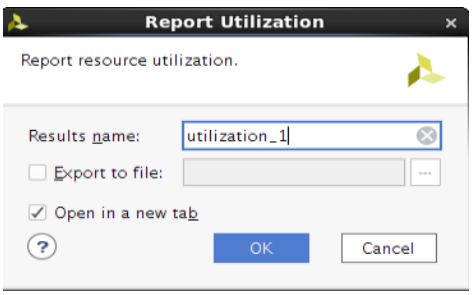
When run from the Tcl Console, the report can include usage of a particular hierarchical cell when using the `-cells` option. When run from the Vivado IDE, this information appears in an interactive table.

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

Running Report Utilization

To generate the Utilization Report from the Vivado IDE, select **Reports → Report Utilization**.

Figure 52: **Report Utilization Dialog Box**



Results Name Field

Specify the name of the result window in the Results Name field at the top of the Report Clock Utilization dialog box.

Equivalent Tcl command:

```
report_utilization -name utilization_1
```

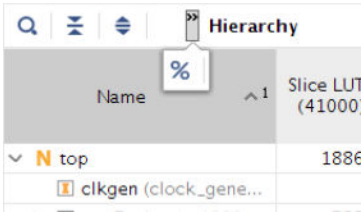
The following figure shows the detailed utilization report.

Figure 53: **Report Utilization**

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	F8 Muxes (10250)	Slice (10250)	LUT as Logic (41000)	LUT as Memory (13400)	LUT Flip Flop Pairs (41000)	Block RAM Tile (135)	DSPs (240)	Bon
top	18863	15635	884	174	6123	18854	9	7422	109	68	
clkgen (clock_gene...	0	0	0	0	0	0	0	0	0	0	
cpuEngine (or1200...	5335	3773	336	10	1719	5335	0	1261	21	4	
fftEngine (fftTop)	1542	1487	0	0	754	1541	1	459	16	64	
mgtEngine (mgtTop)	371	642	0	0	209	371	0	204	0	0	
usbEngine0 (usbf_t...	5659	4642	258	74	1796	5655	4	2624	36	0	
usbEngine1 (usbf_t...	5664	4642	258	74	1855	5660	4	2625	36	0	
wbArbEngine (wb_c...	317	430	32	16	192	317	0	80	0	0	

The utilization number or the utilization percentage can be toggled using the button in the report window.

Figure 54: **Report Utilization Percentage Toggling**



Show Utilization for Specific Cells

When selecting the `-cells` option, the generated report shows the utilization of the specified cells and their children.

```
-cells {cell_name_list}
```

Specific cells can be excluded from the targeted cell level:

```
-exclude_cells {cell_name_list}
```

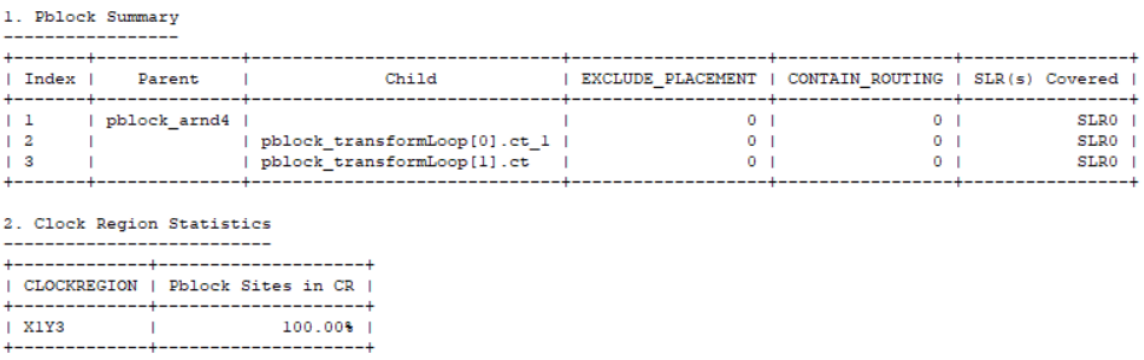
Show Utilization for Specific Pblocks

When selecting the `-pblocks` options, the utilization report reflects the specifics of the specified Pblock. Only one parent Pblock is allowed to be specified. Available resources reflect the parent Pblock ranges, and used resources are broken into parent Pblock, child Pblock, and non-assigned. This breakdown allows you to evaluate competing requirements for the specified parent Pblock resources. These command line options are only supported in Tcl mode.

```
-pblocks {Pblock}
-exclude_child_pblocks
-exclude_non_assigned
```

When specified, two additional tables are reported relating to the specified Pblock. The properties related to the parent and any child Pblocks are printed in the summary along with some information on their SLR placement. Clock region statistics are printed in the second table.

Figure 55: **Report Summary with Pblocks**



1. Pblock Summary

Index	Parent	Child	EXCLUDE_PLACEMENT	CONTAIN_ROUTING	SLR(s) Covered
1	pblock_arnd4		0	0	SLR0
2		pblock_transformLoop[0].ct_1	0	0	SLR0
3		pblock_transformLoop[1].ct	0	0	SLR0

2. Clock Region Statistics

CLOCKREGION	Pblock Sites in CR
X1Y3	100.00%

The pre-placement and post-placement utilization numbers can vary due to LUT combining and non-assigned cells that cannot be accounted for before placement.

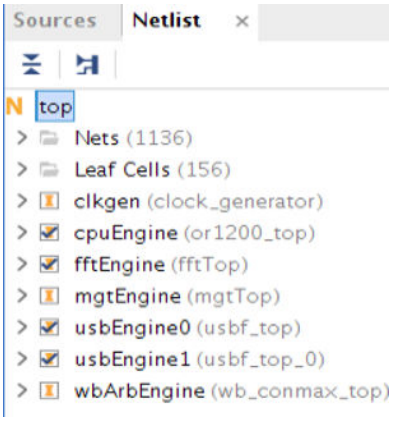
When using `-pblocks`, the utilization tables include the following columns:

- **Parent:** Assigned to parent Pblocks only.
- **Child:** Assigned to child Pblocks only.
- **Non-Assigned:** Total resources used in the area defined by the specified Pblocks, but are not assigned to the specified Pblocks or their child Pblocks.
- **Used:** Total resources used in the area defined by the specified Pblocks
- **Fixed:** Total resources fixed by LOC constraints in the area defined by the specified Pblocks
- **Prohibited:** Resources in the area defined that are prohibited from use, because of PROHIBIT constraints.
- **Available:** Total resources available in the area defined by the specified Pblocks.

- **Util%:** Used / Available

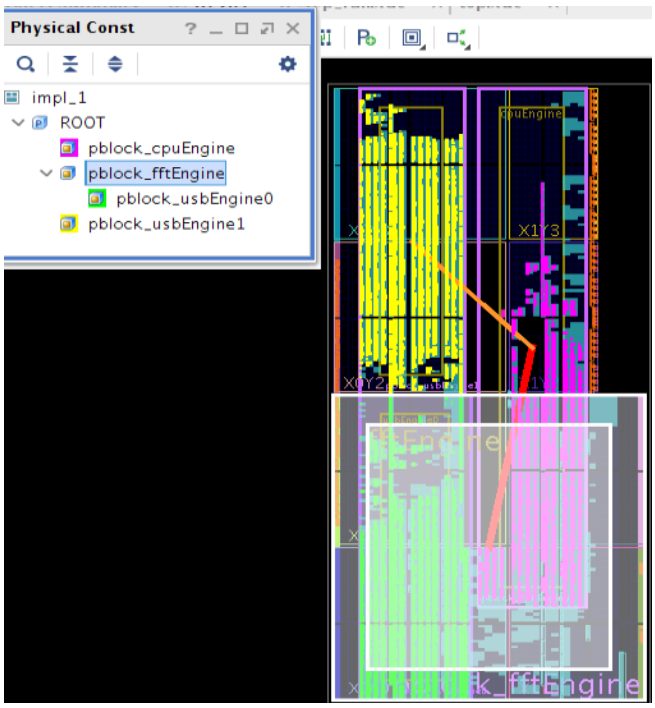
The following example can help you understand the report better. The following figure shows the example design hierarchy.

Figure 56: **Example Design Hierarchy**



The following figure shows the Pblock rectangles. The resources inside each Pblock are also highlighted from post-route netlist.

Figure 57: **Design Pblocks**



In this example:

- The *pblock_usbEngine1* Pblock does not have a child Pblock.

- The *pblock_fftEngine* Pblock has a child Pblock, *pblock_usbEngine0*.
- The *pblock_cpuEngine* Pblock overlaps with the *pblock_fftEngine*.

To generate a report for the entire design, run `report_utilization` without any option.

Figure 58: **Top-Level Utilization Report**

Site Type	Used	Fixed	Available	Util%
Slice LUTs	18863	0	41000	46.01
LUT as Logic	18854	0	41000	45.99
LUT as Memory	9	0	13400	0.07
LUT as Distributed RAM	0	0		
LUT as Shift Register	9	0		
Slice Registers	15635	0	82000	19.07
Register as Flip Flop	15635	0	82000	19.07
Register as Latch	0	0	82000	0.00
F7 Muxes	884	0	20500	4.31
F8 Muxes	174	0	10250	1.70

To generate a report for the *pblock_usbEngine1* Pblock, use the following command:

```
report_utilization -pblocks pblock_usbEngine1
```

Figure 59: **Utilization Report for Pblock *pblock_usbEngine1***

Site Type	Parent	Child	Non-Assigned	Used	Fixed	Available	Util%
Slice LUTs	5664	0	49	5713	0	9600	59.51
LUT as Logic	5660	0	49	5709	0	9600	59.47
LUT as Memory	4	0	0	4	0	4000	0.10
LUT as Distributed RAM	0	0	0	0	0		
LUT as Shift Register	4	0	0	4	0		
Slice Registers	4642	0	28	4670	0	19200	24.32
Register as Flip Flop	4642	0	28	4670	0	19200	24.32
Register as Latch	0	0	0	0	0	19200	0.00
F7 Muxes	258	0	0	258	0	4800	5.38
F8 Muxes	74	0	0	74	0	2400	3.08

To generate a report for the *pblock_fftEngine* Pblock, use the command below. In this case, the resource of the nested child Pblock, *pblock_usbEngine0*, is counted into the total used resources.

Note: If the property EXCLUDE_PLACEMENT is applied to the child Pblock, the child resources are isolated from the parent Pblock, both for Used and Available.

The overlapping Pblock *pblock_cpuEngine* has partial cells being placed in the *pblock_fftEngine* Pblock range and they are reported as Non-Assigned as external resources.

```
report_utilization -pblocks pblock_fftEngine
```

Figure 60: **Parent Pblock with Nested and Overlapping Child Pblocks**

Site Type	Parent	Child	Non-Assigned	Used	Fixed	Available	Util%
Slice	754	1796	1512	3731	0	5600	66.63
SLICEL	513	1081	1055	2458	0		
SLICEM	241	715	457	1273	0		
LUT as Logic	1541	5655	4649	11822	0	22400	52.78
using O5 output only	12	11	0	0			
using O6 output only	974	4651	3986	9588			
using O5 and O6	555	993	663	2234			
LUT as Memory	1	4	0	5	0	7200	0.07
LUT as Distributed RAM	0	0	0	0	0		
LUT as Shift Register	1	4	0	5	0		
using O5 output only	1	0	0	1			
using O6 output only	0	0	0	0			
using O5 and O6	0	4	0	4			
LUT Flip Flop Pairs	459	2624	954	4098	0	22400	18.29
fully used LUT-FF pairs	240	204	145	603			
LUT-FF pairs with one unused LUT output	219	2182	713	3125			
LUT-FF pairs with one unused Flip Flop	147	2315	772	3274			
Unique Control Sets	39	202	118	354			

To exclude some Pblocks or non-assigned resources, use the `-exclude_child_pblocks` or the `-exclude_non_assigned` switch. The following example shows the Non-Assigned column removed from the report.

```
report_utilization -pblocks [get_pblocks pblock_fftEngine] -
exclude_non_assigned
```

Figure 61: **Utilization Report Excluding the Non-Assigned Resources**

Site Type	Parent	Child	Used	Fixed	Available	Util%
Slice	754	1796	2395	0	5600	42.77
SLICEL	513	1081	1521	0		
SLICEM	241	715	874	0		
LUT as Logic	1541	5655	7184	0	22400	32.07
using O5 output only	12	11	11			
using O6 output only	974	4651	5613			
using O5 and O6	555	993	1560			
LUT as Memory	1	4	5	0	7200	0.07
LUT as Distributed RAM	0	0	0	0		
LUT as Shift Register	1	4	5	0		
using O5 output only	1	0	1			
using O6 output only	0	0	0			
using O5 and O6	0	4	4			
LUT Flip Flop Pairs	459	2624	3088	0	22400	13.79
fully used LUT-FF pairs	240	204	444			
LUT-FF pairs with one unused LUT output	219	2182	2403			
LUT-FF pairs with one unused Flip Flop	147	2315	2465			
Unique Control Sets	39	202	239			

The following table describes the content of the report for various scenarios.

Table 8: **Table Report with Pblock Assignments**

Case	Title	Description	Report
1	Report on the entire device (ROOT Pblock): `report_utilization`	EXCLUDE_PLACEMENT has no effect on the utilization report.	Util%: Used / Available.

Table 8: **Table Report with Pblock Assignments** *(cont'd)*

Case	Title	Description	Report
2	Report on the parent Pblock: `report_utilization -pblocks <parentPblockName>`	Child Pblock is nested within parent Pblock. No EXCLUDE_PLACEMENT property is specified for child Pblock.	Non-Assigned: Total cells placed within the parent Pblock bounds but not assigned to the parent or child Pblocks. Fixed: Total cells fixed within the parent Pblock bounds. Used: Parent + Child + Non-Assigned cells placed within the Parent Pblock bounds. Available: Total of physical resources within the Parent Pblock bounds. Util%: Used / Available.
		Child Pblock is nested within the parent Pblock. EXCLUDE_PLACEMENT property is specified for the child Pblock. Reported area corresponds to parent Pblock ranges minus child Pblock ranges.	Non-Assigned: Total cells placed within reported area, not assigned to Parent and Child Pblocks. Fixed: Total cells fixed within reported area. Used: Parent Pblock cells excluding the child Pblock cells. Available: Total of physical resources within reported area. Util%: Used / Available.
3	Report on overlapping Pblocks	Similar to the default Pblock report, except the Available becomes the union of the reported Pblocks. The EXCLUDE_PLACEMENT property is ignored.	Available: Union of the Pblocks' physical resources. Util%: Used / Available.

Show SLR Utilization

When selecting the `-slr` option, the generated report shows the SLR related utilization. Starting with AMD Vivado™ Design Suite 2018.3, the SLR utilization tables have been enhanced in the GUI and text report, which includes the following 4 different tables:

- SLR Connectivity
- SLR Connectivity Matrix
- SLR CLB Logic and Dedicated Block Utilization
- SLR IO Utilization

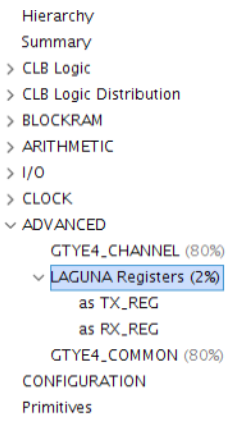
These tables are also shown by default when running `report_utilization`. The SLR connectivity table shows the LAGUNA registers utilized for both TX and RX directions on each SLR side.

Figure 62: **SLR Connectivity Report in GUI**

Connectivity	Used	Fixed	Available
SLR1 <-> SLR0	229		23040
SLR0 -> SLR1	0		
Using Both TX_REG and RX_REG	0	0	
Using RX_REG only	0	0	
Using TX_REG only	0	0	
SLR1 -> SLR0	229		
Using Both TX_REG and RX_REG	0	0	
Using RX_REG only	0	0	
Using TX_REG only	0	0	
SLR2 <-> SLR1	311		23040
SLR1 -> SLR2	0		
Using Both TX_REG and RX_REG	0	0	
Using RX_REG only	0	0	
Using TX_REG only	0	0	
SLR2 -> SLR1	311		
Using Both TX_REG and RX_REG	0	0	
Using RX_REG only	0	0	
Using TX_REG only	0	0	

The total used LAGUNA registers are also reported in the ADVANCED table for both directions.

Figure 63: **LAGUNA Register Report in GUI**



The SLR CLB Logic and Dedicated Block Utilization table shows the resource utilization for each SLR.

Figure 64: **Utilization in Each SLR**

Site Type	SLR0	SLR1	SLR2
Block RAM Tile	0	0	197
RAMB18	0	0	10
RAMB36/FIFO	0	0	192
CARRY8	0	2	1004
CLB	0	0	0
CLBL	3	43	11750
CLBM	3	0	1019
CLB LUTs	43	344	102149
LUT as Logic	19	344	93998
LUT as Memory	24	0	8151
LUT as Distributed RAM	0	0	5275
LUT as Shift Register	24	0	2876
CLB Registers	89	609	104659
DSPs	0	0	0
F7 Muxes	0	0	1302
F8 Muxes	0	0	242
F9 Muxes	0	0	0
MMCM	0	0	0
PLL	0	0	0
Unique Control Sets	7	51	4514
URAM	0	0	0

Show Hierarchical Information with Customized Options

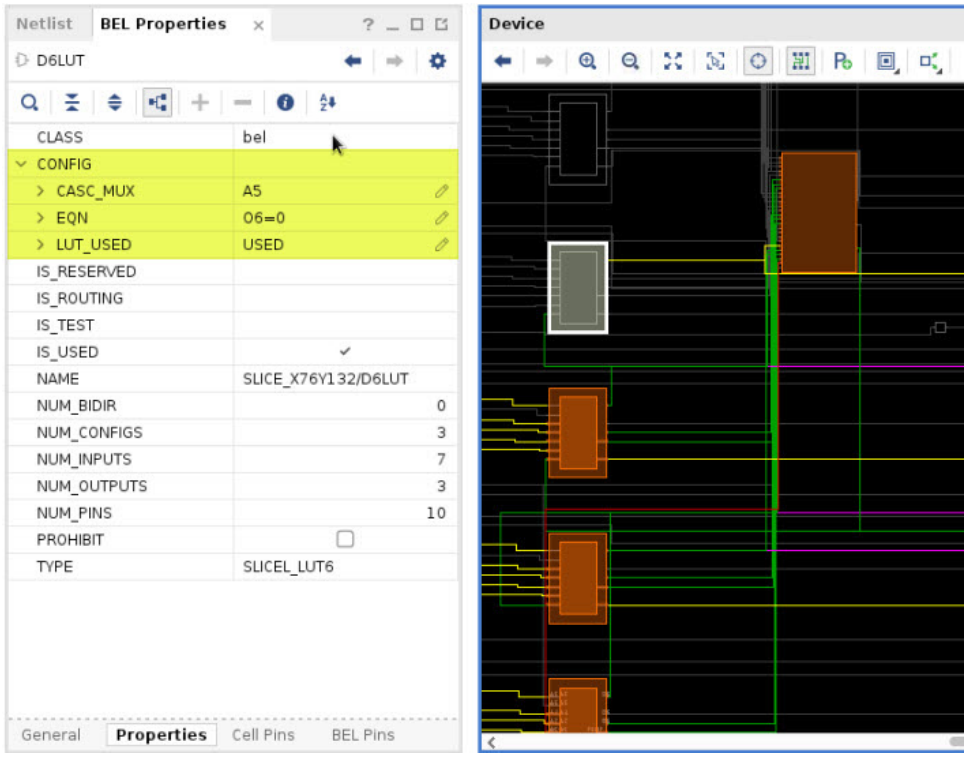
When selecting the following options, the report can be limited to certain levels of hierarchy. Specifies the depth of the hierarchy to report when reporting utilization according to the hierarchy. The default depth is 0, which means that by default, `-hierarchical` only reports the top-level.

```
-hierarchical
-hierarchical_depth <args>
-hierarchical_percentage
```

Showing Packthrus

Packthrus are used bels that do not have a netlist cell associated with them. They are typically added to legalize routing to reach elements that would be otherwise unroutable or to make routing easier. Packthru LUTs will use one output and up to one input. Other parts of the LUT are still usable if required. A register will only be used as a packthru when the site is not required for other uses.

In the following example, a packthru LUT is configured to drive a logic 0 into the LOOKAHEAD bel from the O6 output of the LUT. The O5 output is free and a smaller LUT can be added to use this BEL.




```
-packthru
```

Show Customized Table Report

When selecting the following options, the report can be customized to only address certain types of resources along with the hierarchical depth.

```
-spreadsheet_table <args>
-spreadsheet_depth
```

Showing Prohibit and Fixed Information

Information on prohibited sites and fixed cells is added to each line of the report when the text version of the report is generated. This happens when `report_utilization` is run at the Tcl Console without the `-name` option.

Prohibits change the calculation of the available resources (Available Resources = Total Resources – Prohibit).

Cells are fixed when they have the `IS_LOC_FIXED` property set to 1. This happens automatically when you set the `LOC` property on a cell.

Report I/O

The I/O Report replaces AMD ISE® Design Suite PAD file. The I/O Report lists:

- Pin Number: All the pins in the device
- Signal Name: The name of the user I/O assigned to the pin
- Bank Type: The bank type where the I/O is located (High Range, High Performance, Dedicated, etc.)
- Pin Name: Name of the pin
- Use: The I/O usage type (Input, Output, Power/Ground, Unconnected, etc.)
- I/O Standard: The I/O standard for the User I/O

  An asterisk (*) indicates that it is the default. This differs from the I/O Ports window of the Vivado IDE.
- I/O Bank Number: The I/O 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 on/off chip termination settings
- Voltage: The values for various pins, including VCCO, VCCAUX, and related pins
- Constraint: Displays Fixed if the pin has been constrained by the user
- Signal Integrity: The Signal Integrity of the pin

Report Clock Utilization

The Clock Utilization Report helps you analyze the utilization of clocking primitives and routing resources inside the device at the clock region level or at the clock net level. It can be useful for debugging clock placement issues and identify placement constraints to maximize the resource utilization. The Clock Utilization Report provides information on:

- The number of clocking primitives available and utilized, and their physical constraints
- The timing clock name and period associated with each clock net
- Each clock region clocking and fabric loads utilization
- Each clock net loads in each clock region

In addition, the Clock Utilization Report in the Vivado IDE supports netlist and device objects selection for highlighting placement information and creating schematics.

Report Clock Utilization Tables

The report presents the clocking topology and placement information organized by categories:

- Clock Primitive Utilization
- Global Clock Resources

- Global Clock Source Details
- Local Clock Resources
- Clock Regions utilization details
- Global Clocks placement details

Due to long netlist object names and to the large number of clock nets and clock primitives in typical designs, a short ID is given to specific clock resources:

- A unique Global Id "`g<n>`" for each net driven by a clock buffer
- A unique Source Id "`src<n>`" for clock generator, such as an MMCM or an input buffer.
- A unique Local Id "`<n>`" for clock nets not routed with global clock resources.

The Global Source and Local IDs simplify searching specific clock nets throughout the report. The original netlist object names are available in the last two columns of each table when applicable.

Clock Primitive Utilization Table

The Clock Primitive Utilization table shows the utilization summary for each clock primitive type and their physical constraints.

Figure 65: **Report Clock Utilization – Clock Primitive Utilization Table**

Clock Primitive Utilization

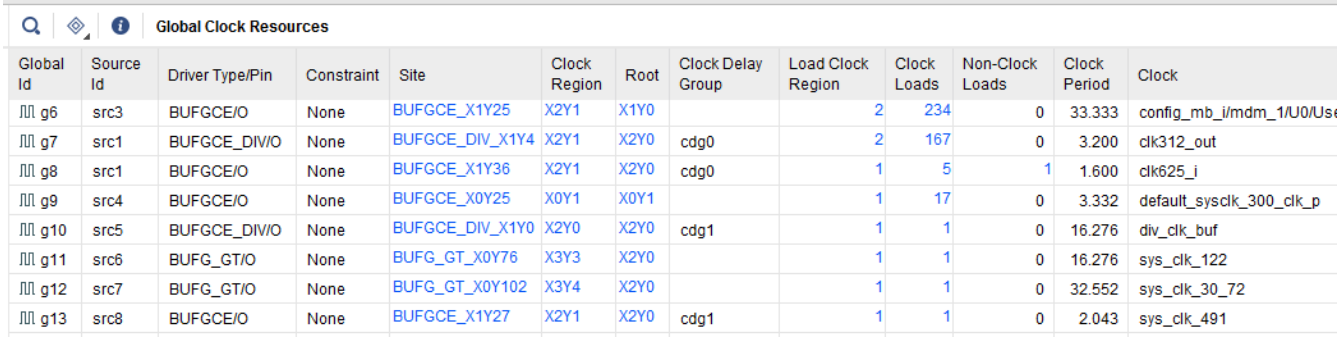
Type	Used	Available	LOC	Clock Region	Pblock
BUFGCE	11	240	0	0	0
BUFGCE_DIV	2	40	0	0	0
BUFGCTRL	1	80	0	1	0
BUFG_GT	2	120	0	0	0
MMCM	2	10	1	0	0
PLL	3	20	0	0	0

Note: The Clock Region constraints do not apply to 7 series devices.

Global Clock Resources Table

The Global Clock Resources table shows a summary for each clock net with important constraints and placement information, as shown in the following figure.

Figure 66: **Report Clock Utilization – Global Clock Resources Table**



Global Clock Resources

Global Id	Source Id	Driver Type/Pin	Constraint	Site	Clock Region	Root	Clock Delay Group	Load Clock Region	Clock Loads	Non-Clock Loads	Clock Period	Clock
g6	src3	BUFGCE/O	None	BUFGCE_X1Y25	X2Y1	X1Y0		2	234	0	33.333	config_mb_i/mdm_1/U0/Us
g7	src1	BUFGCE_DIV/O	None	BUFGCE_DIV_X1Y4	X2Y1	X2Y0	cdg0	2	167	0	3.200	clk312_out
g8	src1	BUFGCE/O	None	BUFGCE_X1Y36	X2Y1	X2Y0	cdg0	1	5	1	1.600	clk625_i
g9	src4	BUFGCE/O	None	BUFGCE_X0Y25	X0Y1	X0Y1		1	17	0	3.332	default_sysclk_300_clk_p
g10	src5	BUFGCE_DIV/O	None	BUFGCE_DIV_X1Y0	X2Y0	X2Y0	cdg1	1	1	0	16.276	div_clk_buf
g11	src6	BUFG_GT/O	None	BUFG_GT_X0Y76	X3Y3	X2Y0		1	1	0	16.276	sys_clk_122
g12	src7	BUFG_GT/O	None	BUFG_GT_X0Y102	X3Y4	X2Y0		1	1	0	32.552	sys_clk_30_72
g13	src8	BUFGCE/O	None	BUFGCE_X1Y27	X2Y1	X2Y0	cdg1	1	1	0	2.043	sys_clk_491

The columns in the Global Clock Resources table are listed in the following table.

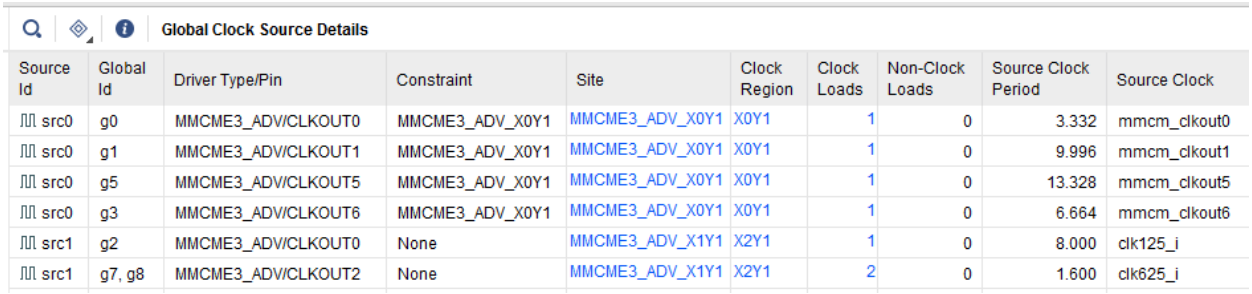
Table 9: **Global Clock Resources Table Details**

Column	Description
Global Id	Unique global clock net ID
Source Id	ID of the clock generating primitive connected to the clock buffer
Driver Type/Pin	Primitive pin connected to the clock net
Constraint	User physical constraint with highest precedence applied to the clock buffer. Priority rule is as follows: 1. `LOC` 2. `CLOCK_REGION`* 3. `PBLOCK` * Does not apply to 7 series.
Site	Clock buffer location set by the user or by the Vivado implementation tools.
Clock Region	Device clock region where the buffer is located. Does not apply to 7 series.
Root	Clock region where the clock net `CLOCK_ROOT` is located. Does not apply to 7 series.
Clock Delay Group	Name of the group of clock nets specified by the user to force routing matching by the AMD Vivado™ implementation tools. Does not apply to 7 series.
Load Clock Region	Number of clock regions where clock net loads are located.
Clock Loads	Number of cells connected via clock pin loads.
Non-Clock Loads	Number of non-clock pin loads, such as FDCE/CE pins for example.
Clock Period	Period in ns of the timing clock which propagates on the clock net. If several clocks propagate on the same clock net, the smallest clock period is reported.
Clock	Name of the timing clock which propagates on the clock net. If several clocks propagate on the same clock net, "Multiple" is reported.
Driver Pin	Logical name of the clock net driver pin.
Net	Logical name of the clock net segment connected to the clock driver pin.

Global Clock Source Details Table

The Global Clock Source Details table shows the global clock connectivity and timing clock information for each clock generator output. The following figure shows the connectivity of each output of an MMCM (`src0/src1`) to clock buffers. The output `CLKOUT0` of `src1` drives two global clocks g7 and g8.

Figure 67: **Report Clock Utilization – Global Clock Source Details Table**



Global Clock Source Details

Source Id	Global Id	Driver Type/Pin	Constraint	Site	Clock Region	Clock Loads	Non-Clock Loads	Source Clock Period	Source Clock
src0	g0	MMCME3_ADV/CLKOUT0	MMCME3_ADV_X0Y1	MMCME3_ADV_X0Y1	X0Y1	1	0	3.332	mmcm_clkout0
src0	g1	MMCME3_ADV/CLKOUT1	MMCME3_ADV_X0Y1	MMCME3_ADV_X0Y1	X0Y1	1	0	9.996	mmcm_clkout1
src0	g5	MMCME3_ADV/CLKOUT5	MMCME3_ADV_X0Y1	MMCME3_ADV_X0Y1	X0Y1	1	0	13.328	mmcm_clkout5
src0	g3	MMCME3_ADV/CLKOUT6	MMCME3_ADV_X0Y1	MMCME3_ADV_X0Y1	X0Y1	1	0	6.664	mmcm_clkout6
src1	g2	MMCME3_ADV/CLKOUT0	None	MMCME3_ADV_X1Y1	X2Y1	1	0	8.000	clk125_i
src1	g7, g8	MMCME3_ADV/CLKOUT2	None	MMCME3_ADV_X1Y1	X2Y1	2	0	1.600	clk625_i

The columns in the Global Clock Source Details table are listed in the following table.

Table 10: **Global Clock Source Details Columns**

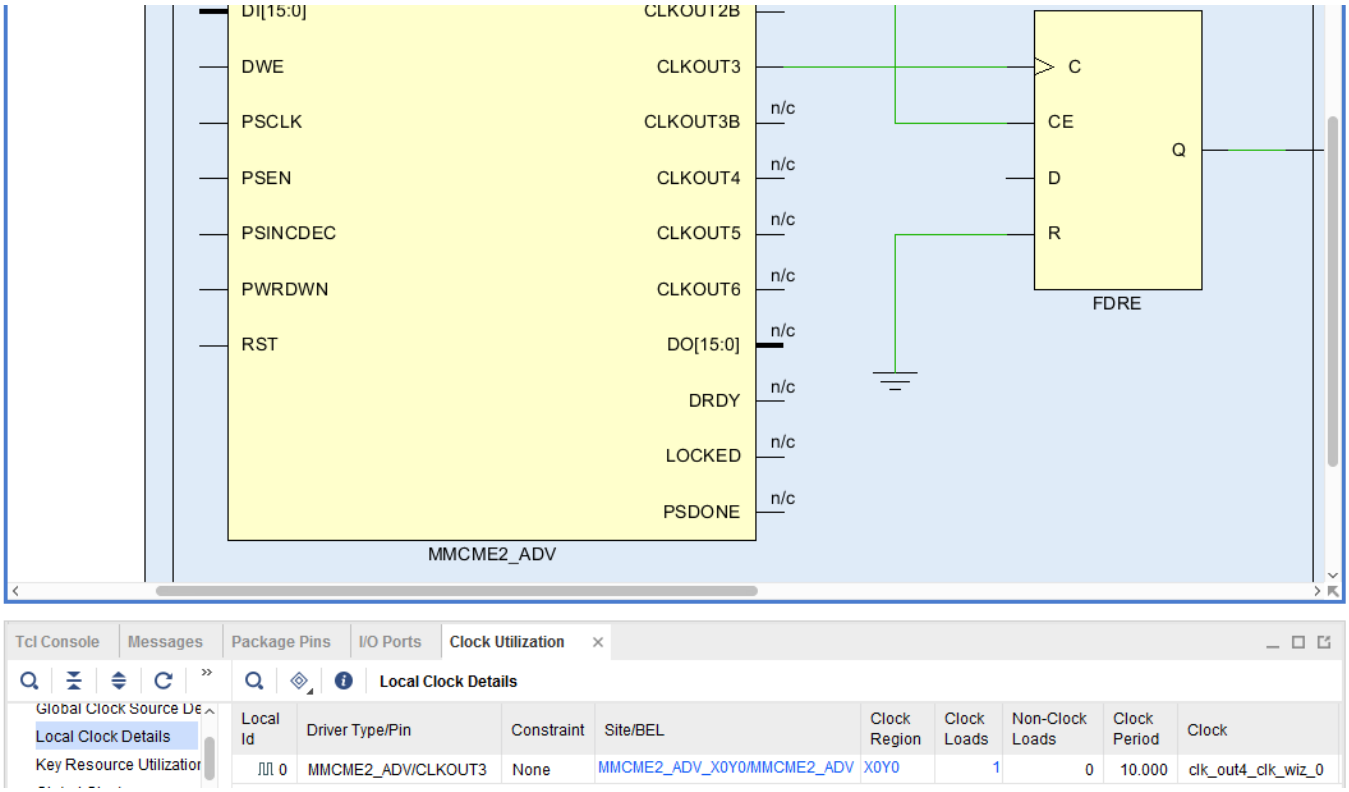
Column	Description
Source Id	ID of the clock generating primitive.
Global Id	Global clock ID(s) driven by the Global Clock source pin.
Driver Type/Pin	Output primitive pin which generates the clock.
Constraint	User physical constraint with highest precedence applied to the clock buffer. Priority rule is as follows: 1. `LOC` 2. `PBLOCK`
Site	Global clock source location set by the user or by the Vivado implementation tools.
Clock Region	Device clock region where the clock source is located.
Clock Loads	Number of clock pin loads connected to Global Clock source pin.
Non-Clock Loads	Number of non-clock pin loads, such as FDCE/CE pins for example.
Source Clock Period	Period in ns of the timing clock generated by the Global Clock Source pin. If several clocks propagate on the same clock net, the smallest clock period is reported.
Clock	Name of the timing clock generated by the Global Clock Source pin. If several clocks propagate on the same clock net, "Multiple" is reported.
Driver Pin	Logical name of the Global Clock Source pin.
Net	Logical name of the clock net segment connected to the Global Clock Source pin.

Local Clock Details Table

The Local Clock Details table is only reported when local clocks are found in the design. A local clock is a clock net routed with regular fabric routing resources instead of global clocking resources. This situation usually occurs when a clock net is not driven by a clock buffer. The information provided by the table is similar to the one found in the Global Clock Resources table.

The following figure shows a local clock net driven by a 7 series MMCM output which directly drives a register clock pin (FDRE/C).

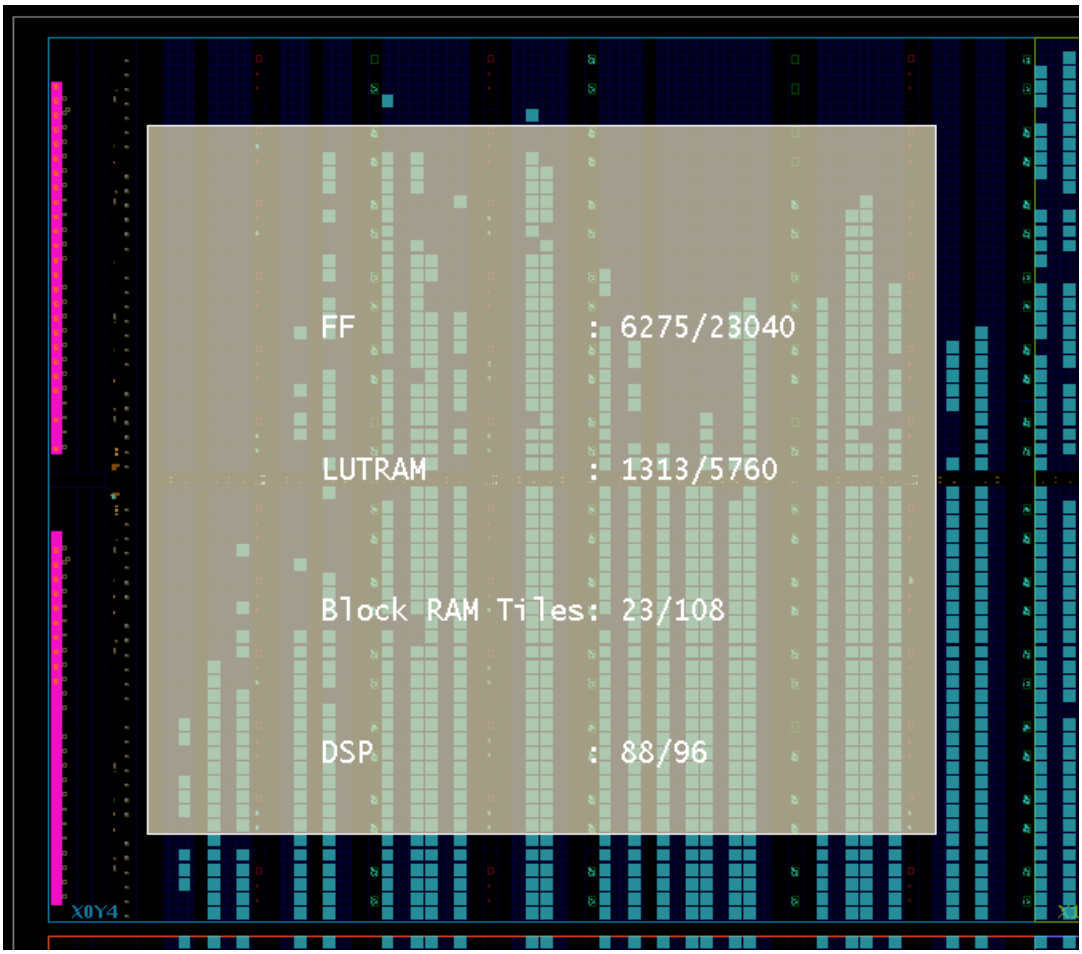
Figure 68: **Report Clock Utilization – Local Clock Example**



Clock Regions Tables

The Clock Regions section is only available for the UltraScale device families and includes several tables to cover primitive and routing resource utilization per clock region. In the Clock Utilization window, the Show Metrics In Device Window button can be used to select the resource types to be displayed over each clock region in the Device window, as shown in the following figure.

Figure 69: **Report Clock Utilization - Clock Region Resource Utilization Metrics in the Device Window**



The Clock Regions tables are:

- Clock Primitives: Utilization of each clock primitive type in each clock region.
- Load Primitives: Utilization of non-clock sequential primitives in each clock region.

For both Clock Primitives and Load Primitives table, the Global Clock column shows the number of global clock nets routed on the horizontal distribution layer with or without loads in the reported clock region. Clock nets routed on the vertical distribution layer with no branching to the horizontal layer in the reported clock region are not counted. Clock nets routed on the routing layer are not counted.

- Global Clock Summary: Shows the utilization of Global Clocks per clock region in a table which corresponds to the device clock region floorplan, as shown in the following figure. This table is only available in the text report.

Figure 70: **Report Clock Utilization – Global Clock Summary Example**

```
6. Clock Regions : Global Clock Summary
---------------------------------------

+----+----+----+----+----+
|    | X0 | X1 | X2 | X3 |
+----+----+----+----+----+
| Y4 |  4 |  5 |  5 |  3 |
| Y3 |  5 |  5 |  6 |  6 |
| Y2 |  5 |  9 | 11 |  7 |
| Y1 |  6 |  9 | 14 |  8 |
| Y0 |  6 |  6 | 13 | 12 |
+----+----+----+----+----+
```

- Routing Resource Utilization: Shows the global clock routing resource utilization per type and per clock region.

Key Resource Utilization Table

The Key Resource Utilization table is only available for 7 series devices and is equivalent to the combination of all Clock Regions tables for UltraScale devices. The Global Clock Summary table is also available in the text report only.

Global Clocks Tables

The Global Clocks tables report the type of loads in each clock region for each global clock net, as well as timing clock information and the clock track ID used to route the clock net. When sorting the table by Global ID in the Vivado IDE, the clock regions where each global clock net is routed can easily be identified and highlighted in the device by simply selecting the corresponding table rows.

The column description is the same as in the Clock Primitive Utilization, Global Clock Resources, and Global Clock Source Details tables.

For UltraScale devices, the Global ID of clock nets routed over a clock region without driving any loads are tagged with a "+" character, as shown in the following figure.

Figure 71: **Report Clock Utilization – Clock Region Cell Placement Example**

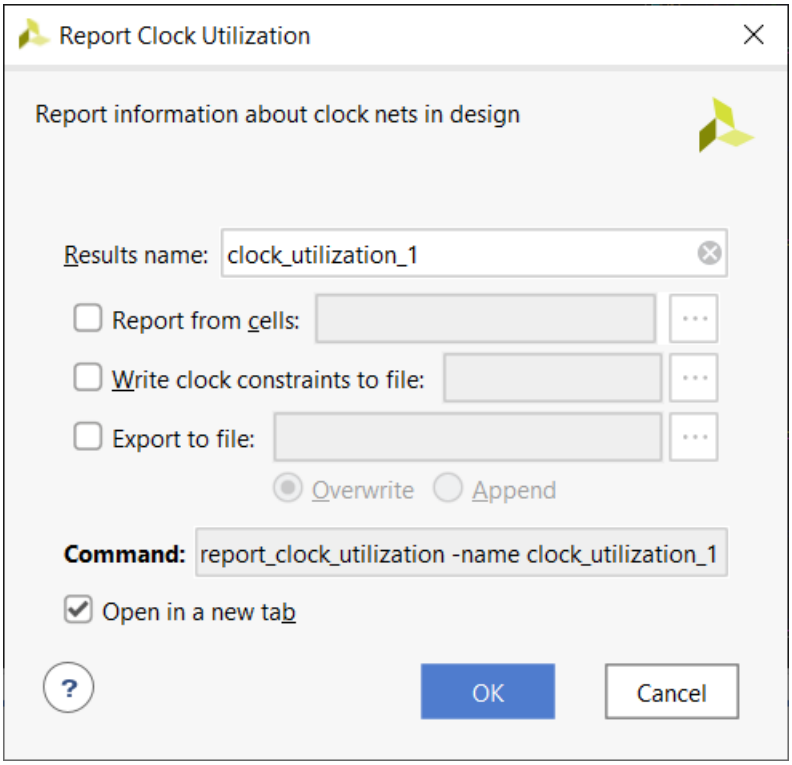
Clock Region Cell Placement

Global Id	Clock Region	Track	Driver Type/Pin	Constraint	Clock Loads	Non-Clock Loads	FF	LUTRAM	RAMB	URAM	DSP	GT	MMCM	PLL	Hard IP
g11+	X2Y1	4	BUFG_GT/O	None	0	0	0	0	0	n/a	0	0	0	0	0
g12+	X2Y1	6	BUFG_GT/O	None	0	0	0	0	0	n/a	0	0	0	0	0
g13+	X2Y1	3	BUFGCE/O	None	0	0	0	0	0	n/a	0	0	0	0	0
g14+	X2Y1	0	BUFGCTRL/O	X2Y4	0	0	0	0	0	n/a	0	0	0	0	0
g15	X2Y1	20	BUFGCE/O	None	0	1	0	0	0	n/a	0	0	1	0	0
g1	X2Y0	2	BUFGCE/O	None	2433	0	2425	0	8	n/a	0	0	0	0	0
g2	X2Y0	14	BUFGCE/O	None	3669	0	3627	40	2	n/a	0	0	0	0	0

Generating the Clock Utilization Report

To generate the Clock Utilization Report in the Vivado IDE, select **Reports → Report Clock Utilization**.

Figure 72: **Report Clock Utilization Dialog Box**



Equivalent Tcl command:

```
report_clock_utilization -name clock_utilization_1
```

Results Name Field

In the Results Name field at the top of the Report Clock Utilization dialog box, specify the name of the graphical window for the report.

Equivalent Tcl option:

```
-name <windowName>
```

Show Clock Roots Only

When selecting this option, the Global Clock Resources table only shows the clock root location for each clock net instead of the complete source, load, and timing clock details.

Equivalent Tcl option:

```
-clock_roots_only
```

Write Clock Constraints to File

Select this option and specify the name of a new constraints file to export the clock source and load physical constraints that correspond to the placement information of the design in memory.

Equivalent Tcl option:

```
-write_xdc <filename>
```

Export to File

You can write the results to a file in addition to generating a GUI report by selecting **Export to file** and specifying a file name in the field on the right. Click the **Browse** button to select a different directory.

Equivalent Tcl option:

```
-file <arg>
```

Select the **Overwrite** option to overwrite an existing file with the new analysis results. Select **Append** to append the new results.

Equivalent Tcl option:

```
-append
```

Report DRC

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.

IMPORTANT! *Review the Critical Warnings. The severity of a particular check may be increased later in the flow.*

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

Elaborated Design

The tool checks for DRCs related to I/O, Clock Placement, potential coding issues with your HDL, and XDC constraints. The RTL netlist typically does not have all the I/O 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 for I/O, 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 four severities: Info, Warning, Critical Warning, and Error. Critical Warnings and Errors do not block the design flow at this point.

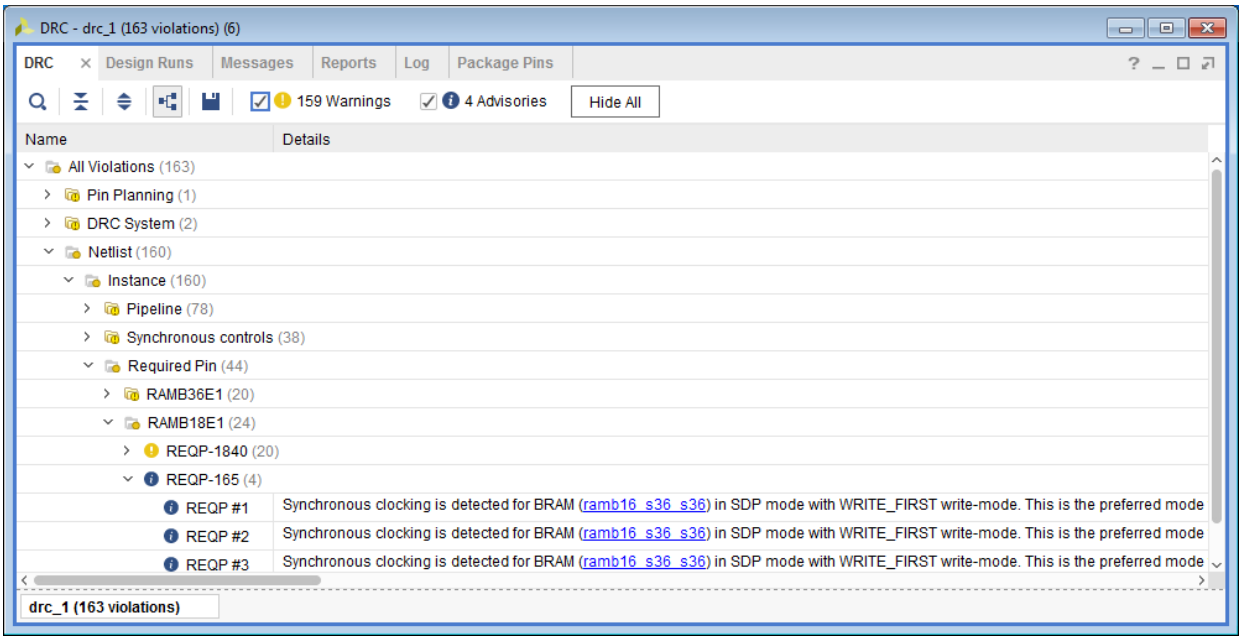
Steps of the implementation flow also run the DRCs, which can stop the flow at critical points. The placer and router check for issues that block placement. Certain messages have a lower severity depending on the stage. These are DRCs flagging conditions that do not stop `opt_design`, `place_design`,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

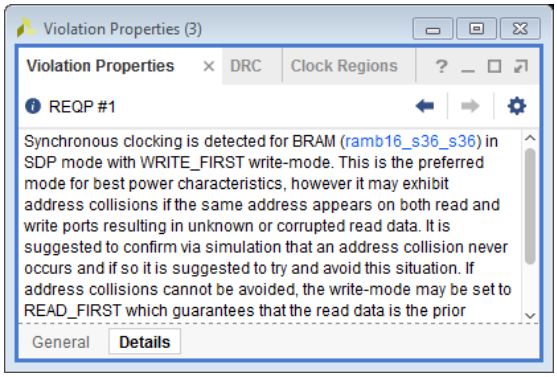
The following figure shows the Vivado IDE graphical user interface form of Report DRC.

Figure 73: **DRC Report**



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

Figure 74: **Violation Properties Dialog Box**



The DRC report is static. You must rerun 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 display the properties for the object, deselect and reselect it.

To create a DRC report in Tcl, run the command `report_drc`.

To write the results to a file, run the command `report_drc -file myDRCs.txt`.

TIP: *For more information on `report_drc`, run `report_drc -help`.*

Report Route Status

The Route Status Report is generated during the implementation flow and is available by using the `report_route_status` Tcl command.

The Route Status Report displays a breakdown of the nets in the design as follows:

- The total number of logical nets in the design
  - The number of nets that do not need routing resources
  - The number of nets that do not use routing resources outside of a tile. Examples include nets inside of a CLB, BlockRam, or I/O Pad.
  - The number of Nets without loads, if any exist
  - The number of routable nets that require routing resources

- The number of unrouted nets, if any exist
- The number of fully routed nets
- The number of nets with routing errors
- The number of nets with some unrouted pins, if any exist
- The number of nets with antennas/islands, if any exist
- The number of nets with resource conflicts, if any exist

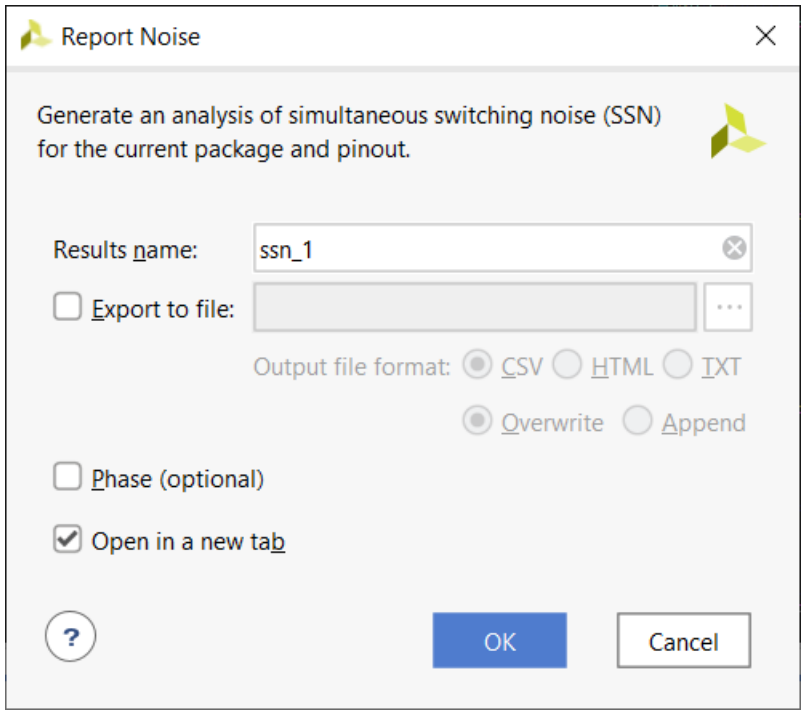
The following is an example of the Report Route Status for a fully routed design:

```
Design Route Status
                                                   :     # nets :
    ---------------------------------------------- : -----------:
    # of logical nets..........................    :        6137 :
        # of nets not needing routing..........    :         993 :
            # of internally routed nets........    :         993 :
        # of routable nets.....................    :        5144 :
            # of fully routed nets.............    :        5144 :
        # of nets with routing errors..........    :           0 :
    ---------------------------------------------- :------------ :
```

Report Noise

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

Figure 75: **Report Noise Dialog Box**



The Noise Report has four sections:

- Noise Report Summary Section
- Noise Report Messages Section
- Noise Report I/O Bank Details Section
- Noise Report Links Section

Noise Report Summary Section

The Summary section of the Noise Report 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

Noise Report Messages Section

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

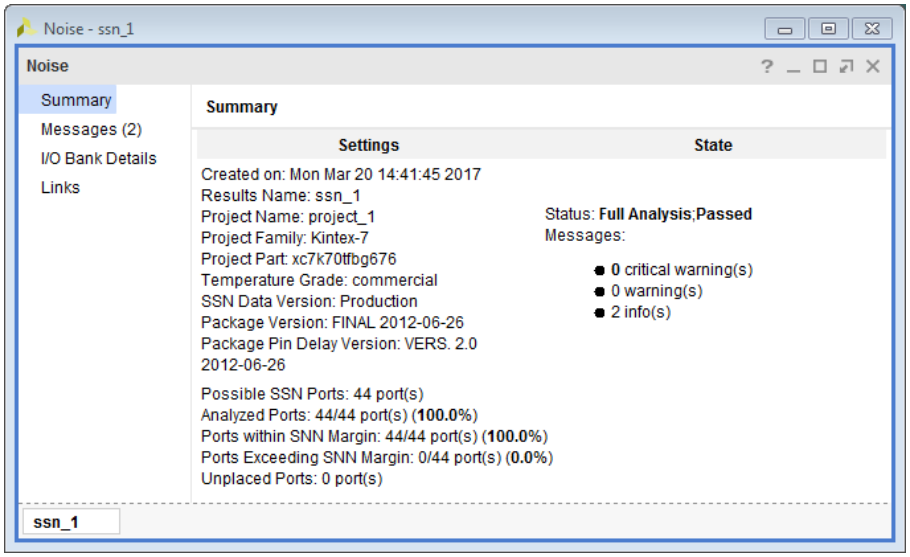
Noise Report I/O Bank Details Section

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

Noise Report Links Section

The Links section of the Noise Report contains links to documentation located online at www.xilinx.com/support.

Figure 76: **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
```

Report Power

The Power Report is generated after routing to report details of power consumption based on the current operating conditions of the device and the switching rates of the design. Power analysis requires a synthesized netlist or a placed and routed design.

- Use the `set_operating_conditions` command to set operating conditions.
- Use the `set_switching_activity` command to define switching activity.

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

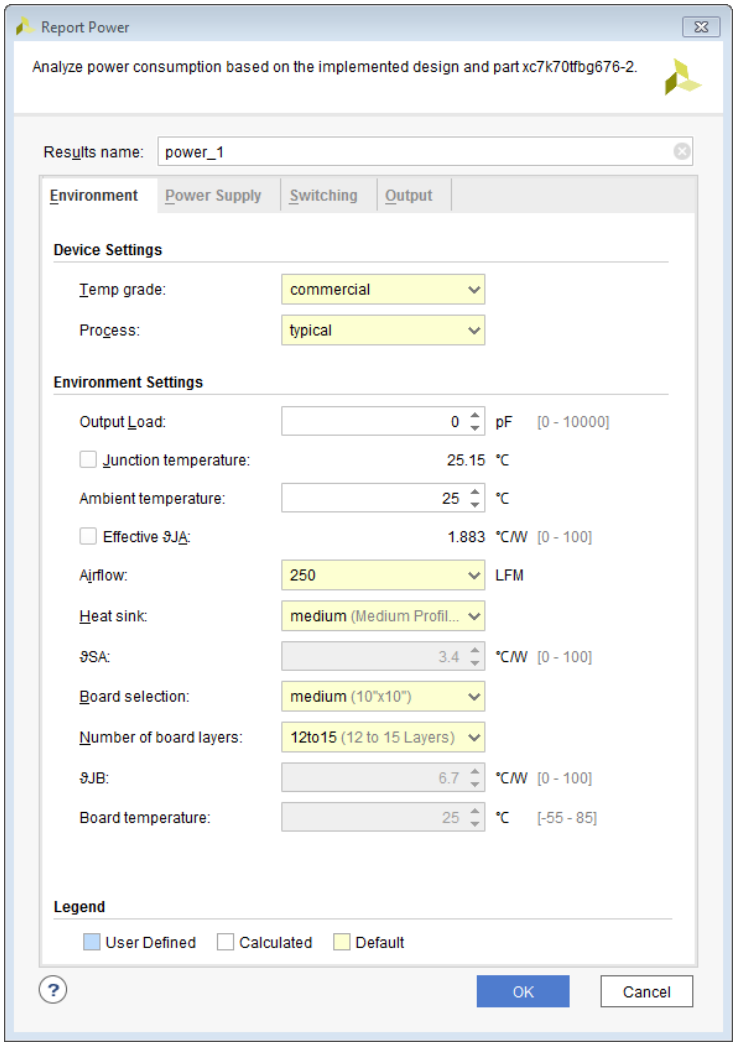
The Power Report estimates power consumption based on design inputs, including:

- Thermal statistics, such as junction and ambient temperature values.

  ***Note*:** You can set the junction temperature using the `-junction_temp` option of the `set_operating_condition` command. If you do not specify the temperature, the software computes it for you based on your design inputs.

- 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.
- Simulation activity files can be used to make power estimation more accurate.

Figure 77: **Report Power Dialog Box**

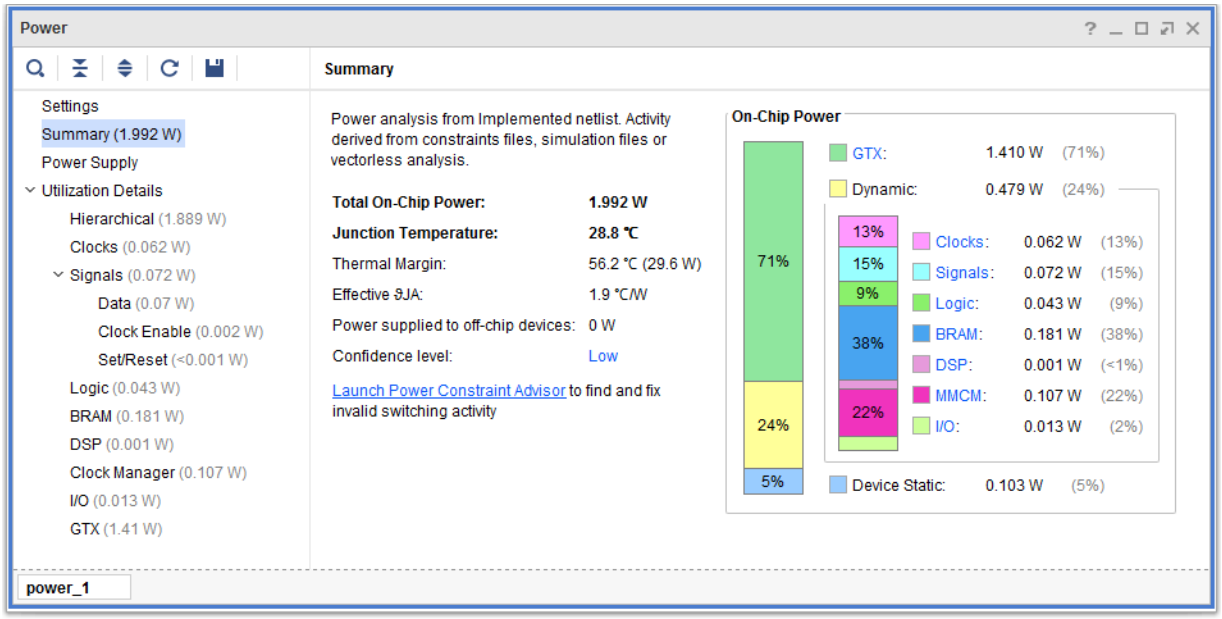


Analyzing the Power Report

Use the Report Power dialog box (see the following figure) to analyze power based on:

- Settings
- Power total
- Hierarchy
- Voltage rail
- Block type

Figure 78: **Power Report**



For more information on the power report and analyzing the results, see the *Vivado Design Suite User Guide: Power Analysis and Optimization* (UG907).

A text version of the power report is generated by default after route during the implementation process.

Reporting Power in a Non-Project Flow

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

Report RAM Utilization

The Ram Utilization Report helps you analyze the utilization of dedicated RAM blocks such as URAM and block RAM as well as Distributed RAM primitives. By default, the report considers the entire design but it can be limited to specific hierarchies using the `-cell` switch. The report can be generated after synthesis and any implementation step but is only available from the Tcl command line.

The RAM Utilization Report is most effective on memories inferred by Vivado synthesis because you can compare the RTL Memory Array with the actual physical implementation in the FPGA.

The report shows:

- The utilization of each memory primitive
- The size of the array and the dimensions (inference only)
- The type of memory
- The utilization of the memory primitives
- The required performance of the memory
- Optional pipeline usage of the memory (where applicable)
- Worst case logical paths that start and end at the memory
- Power efficiency items such as cascading and enable rate

The report can also be generated in CSV format. This is the preferred method when you need to manage and sort a large amount of data.

Running the RAM Utilization Report

The following syntax will run the report in its default mode and send the contents to a file `ram_util.rpt`.

```
report_ram_utilization -file ram_util.rpt
```

The following syntax will generate the report and a CSV file `ram_util.csv`.

```
report_ram_utilization -csv ram_util.csv -file ram_util.rpt
```

In order to report on all memories, including LUTRAM based memories, the `-include_lutram` switch must be used:

```
report_ram_utilization  -include_lutram
```

As path information can be runtime intensive, you can optionally add this to the report using the `-include_path_info` switch:

```
report_ram_utilization  -include_path_info
```

Report Control Sets

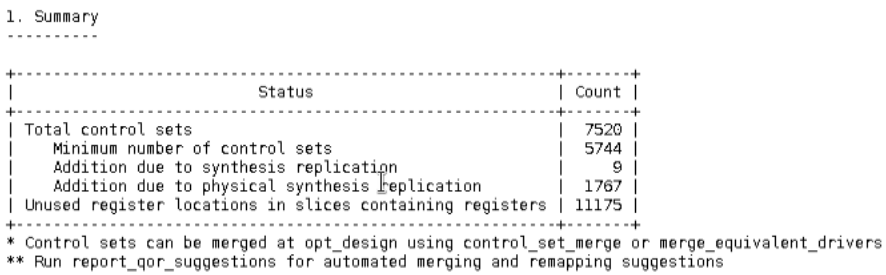
A control set is the unique combination of a clock signal, a clock enable signal, and a set/reset signal. Each slice supports a limited number of control sets in which a combination of flip flops located in it can use. Some control set sharing is permissible within a slice depending on the architecture being used. A user should be familiar with the Configurable Logic Block architecture for the targeted family to understand what are the compatibility rules.

There are two key areas reported:

1. The absolute number of control sets. There is a finite number of control sets in any given part. Exceeding the recommended number of control sets can have a negative impact on QoR.
2. The load profile of control sets. When control set reduction is required, reducing control sets with a low number of loads is most effective as it adds the least amount of logic to the design.

The following is an example of the Control Sets Report Summary. You should follow recommendations in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) regarding recommended control set count.

Figure 79: **Control Set Summary Table**



1. Summary

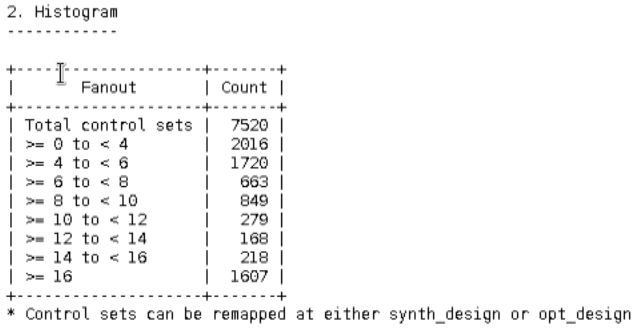
Status	Count
Total control sets	7520
Minimum number of control sets	5744
Addition due to synthesis replication	9
Addition due to physical synthesis replication	1767
Unused register locations in slices containing registers	11175

* Control sets can be merged at opt_design using control_set_merge or merge_equivalent_drivers
** Run report_qor_suggestions for automated merging and remapping suggestions

Typically nets replicated at synthesis are more likely to overlap and place a higher burden on routing resources. Typically nets replicated by physical synthesis overlap less and can be ignored when calculating maximum number of control sets.

When control set counts are above the recommended level, users should reduce the count by optimizing control sets with the lowest BEL count loads. A histogram summary is reported to give an overview:

Figure 80: **Control Set Histogram Table**



2. Histogram

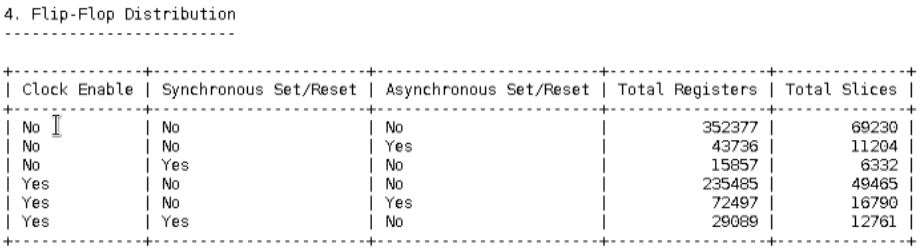
Fanout	Count
Total control sets	7520
>= 0 to < 4	2016
>= 4 to < 6	1720
>= 6 to < 8	663
>= 8 to < 10	849
>= 10 to < 12	279
>= 12 to < 14	168
>= 14 to < 16	218
>= 16	1607

* Control sets can be remapped at either synth_design or opt_design

Where more targeted information is required, the switches `-hierarchical` and `-hierarchical_depth` will help highlight specific hierarchies to target. Synthesis BLOCK_SYNTH.CONTROL_SET_THRESHOLD properties can be used to re-target control sets at a particular level of hierarchy.

The control set report also details the Flip Flop Distribution types that are used in the design. Asynchronous resets can not have their reset control re-targeted by Vivado.

Figure 81: **Control Set Flip Flop Distribution**



4. Flip-Flop Distribution

Clock Enable	Synchronous Set/Reset	Asynchronous Set/Reset	Total Registers	Total Slices
No	No	No	352377	69230
No	No	Yes	43736	11204
No	Yes	No	15857	6332
Yes	No	No	235485	49465
Yes	No	Yes	72497	16790
Yes	Yes	No	29089	12761

For a comprehensive list of all control sets in the design, use the `-verbose` switch. This lists the following information for each control set:

- 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

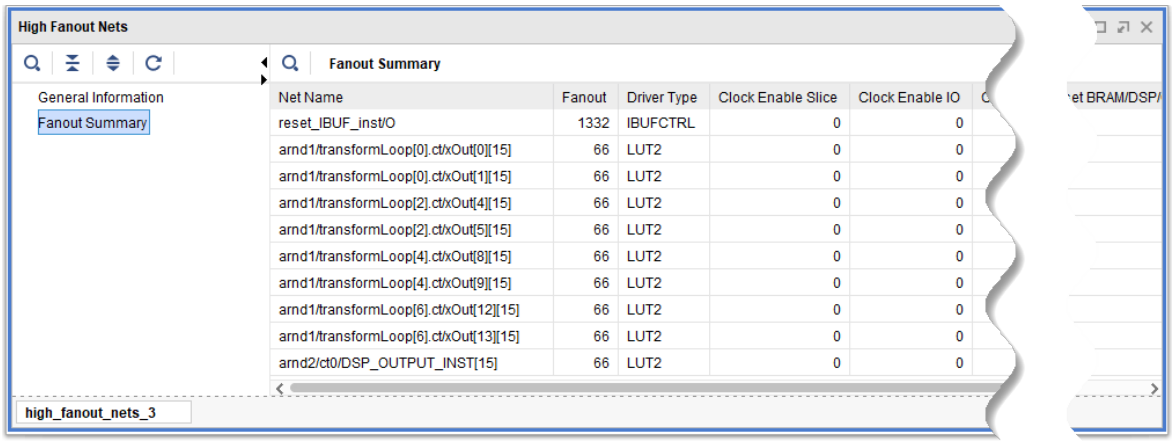
Report High Fanout Nets

The `report_high_fanout_nets` command analyzes the netlist and reports on the nets with the highest fanout. The command can be run on a post-synthesized, placed, or routed netlist. Before placement, however, the report is unable to convey clock region and SLR information. The report displays the following:

- Fanout
- Driver Type
- Load Types
- Clock Regions (where applicable)
- SLRs (where applicable)

The following figure shows an example of the report generated.

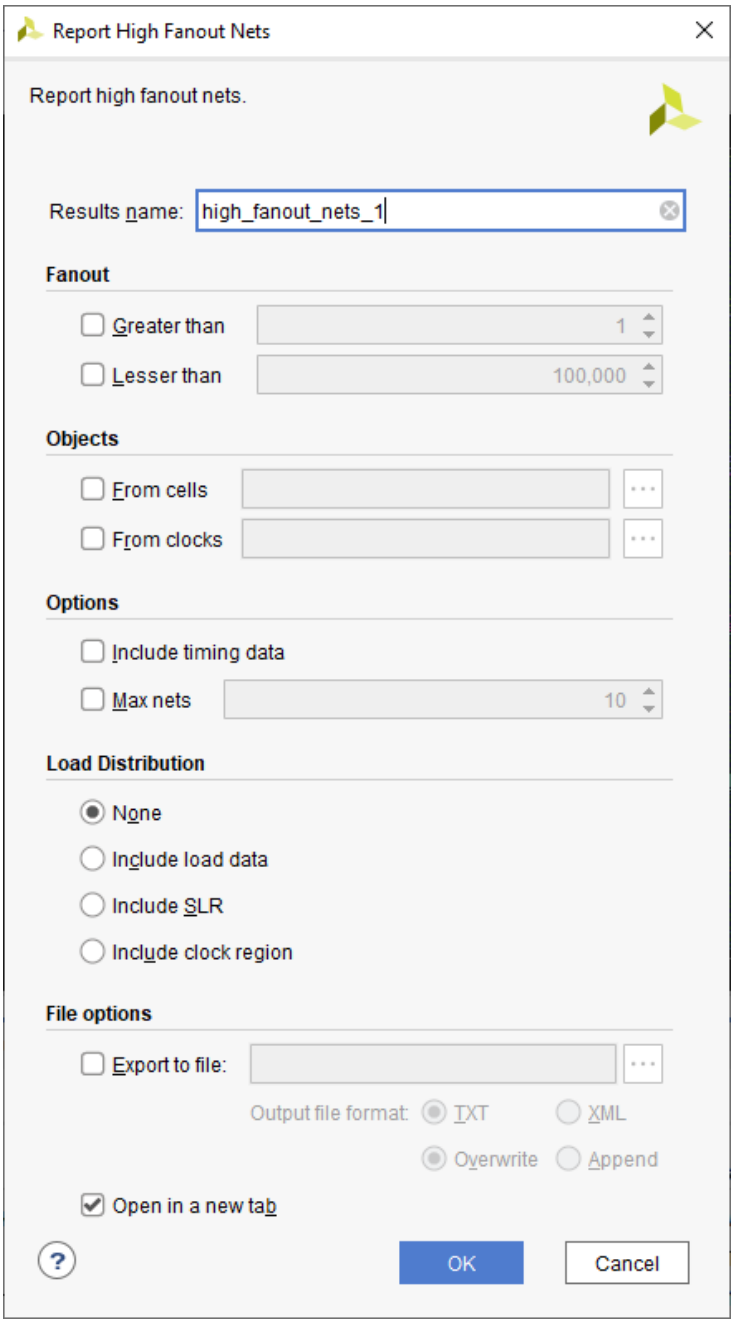
Figure 82: **High Fanout Net Report**



Generating the High Fanout Net Report

To generate the high fanout net report in the Vivado IDE on an open design, select **Reports→ Report High Fanout Nets**.

Figure 83: **Report High Fanout Nets Dialog Box**



Equivalent Tcl Command:

```
report_high_fanout_nets -name hfn_1
```

Report High Fanout Nets Options

Name

In the Results Name field at the top of the Report High Fanout Nets dialog box, specify the name of the graphical window for the report.

Equivalent Tcl option:

```
-name <windowName>
```

Fanout

Minimum and maximum limits can be set to report only nets with the specified fanout.

Equivalent Tcl commands:

```
-fanout_greater_than 1
```

```
-fanout_lesser_than 100000
```

Objects

It is possible to limit the report based on objects. You can scope the report to specific hierarchical cells. When selected, all the sources are in the scoped cell. You can also report on clocks. By default, clocks are not reported but these can be specified in the report using a switch.

Equivalent Tcl commands:

```
-cells [get_cells <hierarchical cell>]
```

```
-clocks [get_clocks <clock object>]
```

Max Nets

By default, 10 nets are reported. It is possible to change this by specifying the `-max_nets` switch. The worst case slack on the net can also be added to the report.

Equivalent Tcl commands:

```
-max_nets <n>
```

```
-timing
```

Load Distribution

Load types, clock regions, and SLRs can be added the report. If access to all is required, the command must be run multiple times.

Equivalent Tcl commands:

```
-load_types
```

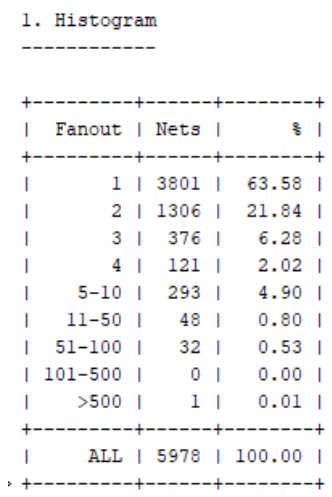
```
-clock_regions
```

```
-slr
```

Histogram

It is possible to show a histogram containing the fanout counts and the number of nets in a given bucket. This histogram can be used to determine if lower fanout nets represent a challenge if they are high in number. This table is only available in text mode. An example of the table is shown in the following figure.

Figure 84: **Histogram Table**



1. Histogram

Fanout	Nets	%
1	3801	63.58
2	1306	21.84
3	376	6.28
4	121	2.02
5-10	293	4.90
11-50	48	0.80
51-100	32	0.53
101-500	0	0.00
>500	1	0.01
ALL	5978	100.00

Equivalent Tcl command:

```
-histogram
```

***Note*:** Fanout is also shown in timing reports for each net and can be determined using the FLAT_PIN_COUNT property on nets.

To address high fanout issues, refer to this link and this link in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

Timing Reports

Report Timing

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

When run from the Tcl Console or the GUI, the timing report can be scoped to one or more hierarchical cells using the `-cells` option. When the report is scoped, only paths with the datapath section that start, end, cross, or are fully contained inside the cell(s) are reported.

Running Report Timing

If a design is already loaded in memory, you can run Report Timing from the menu, the Clock Interaction Report, or the Report Timing Summary paths list.

Running Report Timing from the Menu

To run Report Timing from the Menu, select **Reports → Timing → Report Timing**.

Running Report Timing from the Clock Interaction Report

To run Report Timing from the Clock Interaction Report:

1. Select a **from/to** clock pair.
2. Right-click and select **Report Timing** to run a report from or to the selected clocks.

Running Report Timing from a Paths List

To run Report Timing from a Paths List:

1. Select a path.
2. Right-click and select Report Timing to run a report between the selected path startpoint endpoint.

Equivalent Tcl command: `report_timing`

When setting specific Report Timing options, you can view the equivalent `report_timing` command syntax in:

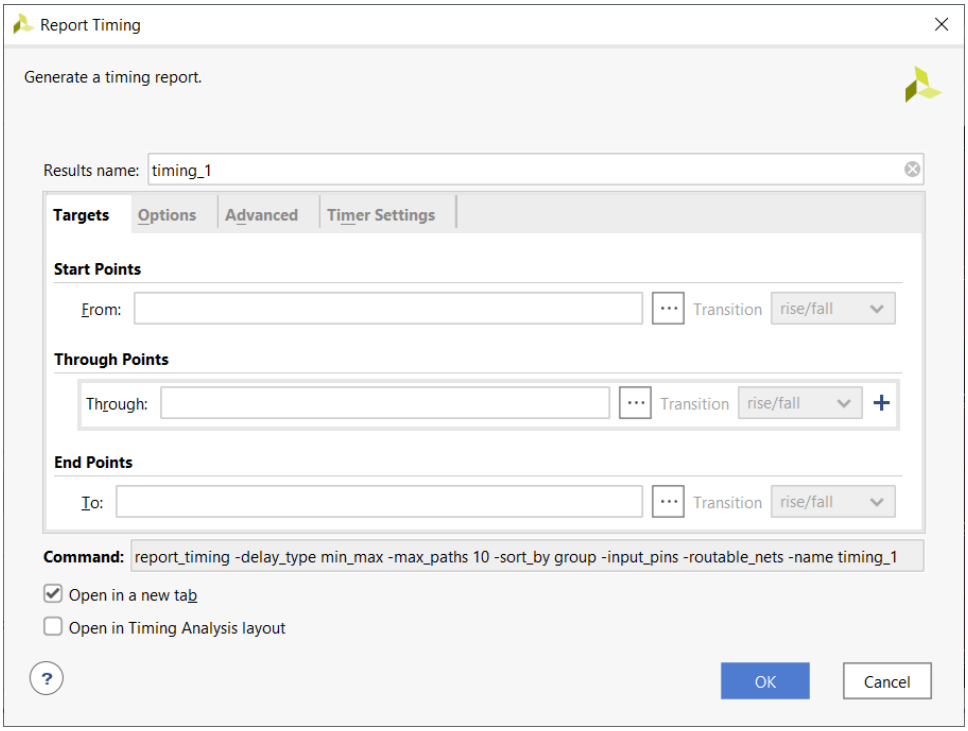
- The Command field at the bottom of the dialog box, and
- The Tcl Console after execution

The `report_timing` options are listed along with the dialog box description in the following section. Overall, the Report Timing options are identical to the Report Timing Summary options, plus a few additional filtering options.

Report Timing Dialog Box

Targets Tab

Figure 85: **Report Timing Dialog Box: 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.

- Startpoints (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 Points (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 to refine your filters and report paths that traverse all groups of through points in the same order as they are listed in the command options.

The Rise/Fall filter applies to the data edge.

RECOMMENDED: *Use the default value (Rise/Fall).*

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

- Endpoints (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 Targets tab in the Report Timing dialog box (see the previous figure) defines the paths from the rising clock edge of `usbClk_3`, through any of the `cpuEngine/or1200_cpu/sprs_dataout[*]` nets, to either edge of `cpuClk_5` or `sysClk`.

Options Tab

The Options tab contains the following options:

- Reports
- Path Limits
- Path Display

Reports

- Path delay type: See Report Section.
- 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 display unconstrained paths in your report.

  Equivalent Tcl option: `-no_report_unconstrained`

Path Limits

- Number of paths per group: See Report Timing Summary.
- Number of paths per endpoint: See Report Timing Summary.

- 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 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: Displays the reported paths based on their slack value.

  Equivalent Tcl option: `-slack_greater_than`

- Display paths with slack less than: See Report Timing Summary.
- Number of significant digits: See Report Timing Summary.
- Sort paths by: Displays 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. The group with the worst violation appears at the top of the list.

  When sorted by slack, the N worst paths per type of analysis are reported (all groups combined) and are sorted by increasing slack.

  Equivalent Tcl option: `-sort_by`

Advanced Tab

The Advanced tab has the same options as Report Timing Summary.

Timer Settings Tab

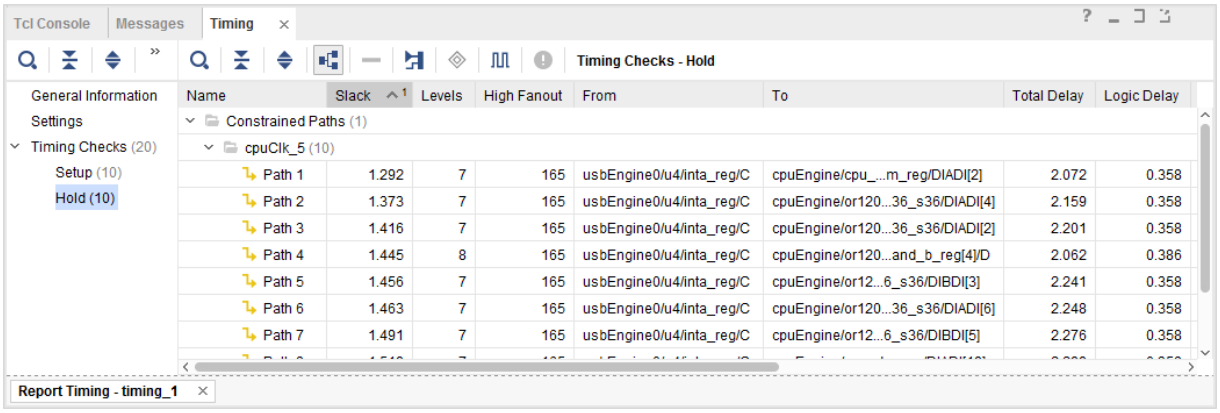
The Timer Settings tab has the same options as Report Timing Summary.

Reviewing Timing Path Details

After you click **OK** to run the report command, a new window opens. You can now review its content. You can view the N-worst paths reported for each type of selected analysis (`min/max/min_max`).

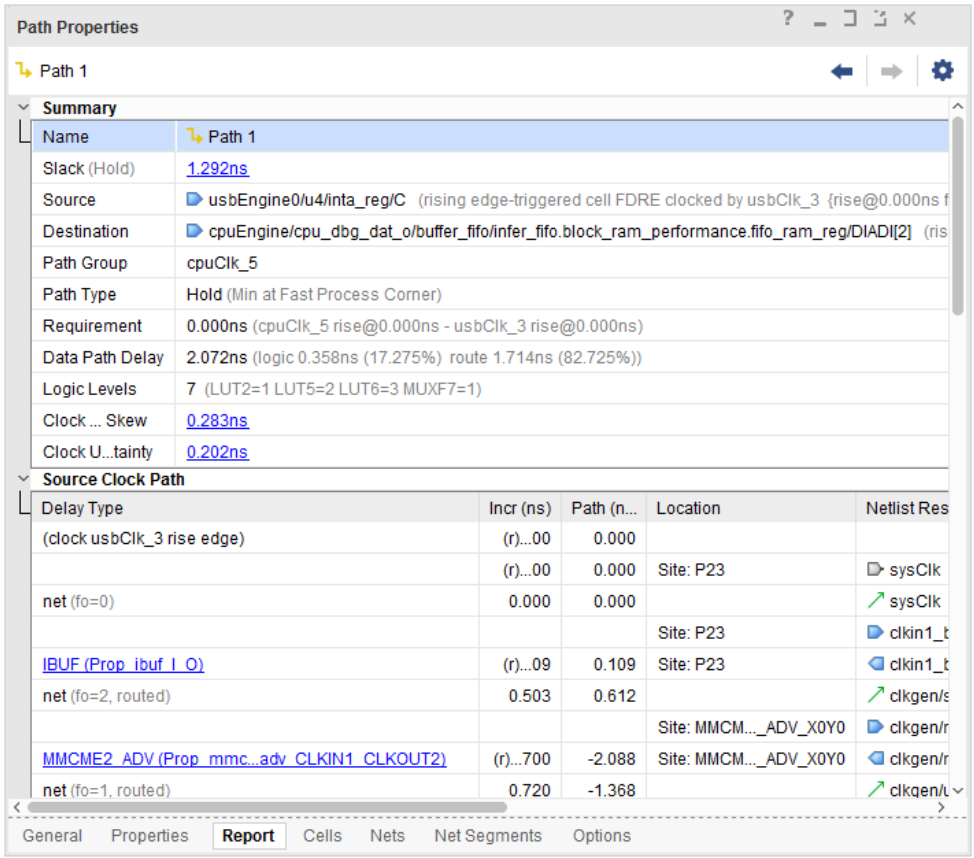
The following figure shows the Report Timing window in which both min and max analysis (SETUP and HOLD) were selected, and N=4.

Figure 86: **Report Timing Paths List**



Select any of these paths to view more details in the Path Properties window (Report tab).

Figure 87: **Timing Path Properties Window**



To view the same details in a new window, double click the path.

For more information on timing path details, see 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 actions:

- View the timing path Schematic.
- Rerun 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 displays the slack value of failing paths in red. To focus on these violations, click **Show only failing checks mode**.

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 following places:

- **Flow Navigator → Synthesis**
- **Flow Navigator → Implementation**
- **Reports → Timing → Report Timing Summary**

The equivalent Tcl command is `report_timing_summary`.

For more information on the `report_timing_summary` options, see this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

In a synthesized design, the AMD 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 the user. There can be larger clock skew on paths where some of the cells have been pre-placed, such as I/Os and GTs.

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 the design is completely routed. To verify that the design is completely routed, view the Route Status report.

When run from the Tcl Console or the GUI, the Timing Summary report can be scoped to one or more hierarchical cells using the `-cells` option. When the report is scoped, only paths with the datapath section that start, end, cross, or are fully contained inside the cell(s) are reported.

When run from the Tcl Console, the first section of the report is a summary of the methodology violations from the latest `report_methodology` run. This section is named "Methodology Summary" when `report_timing_summary` is run from the GUI. When `report_methodology` has not been run prior to `report_timing_summary`, the section is empty. If any design change has been implemented since the last `report_methodology` run, the violations summary might not be up to date.

Report Timing Summary Dialog Box

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

- Options Tab
- Advanced Tab
- Timer Settings Tab

The Results name field at the top of the Report Timing Summary dialog box specifies the name of the graphical report that opens in the Results window. The graphical version of the report includes hyperlinks that allow you to cross-reference nets and cells from the report to Device and Schematic windows, and design source files.

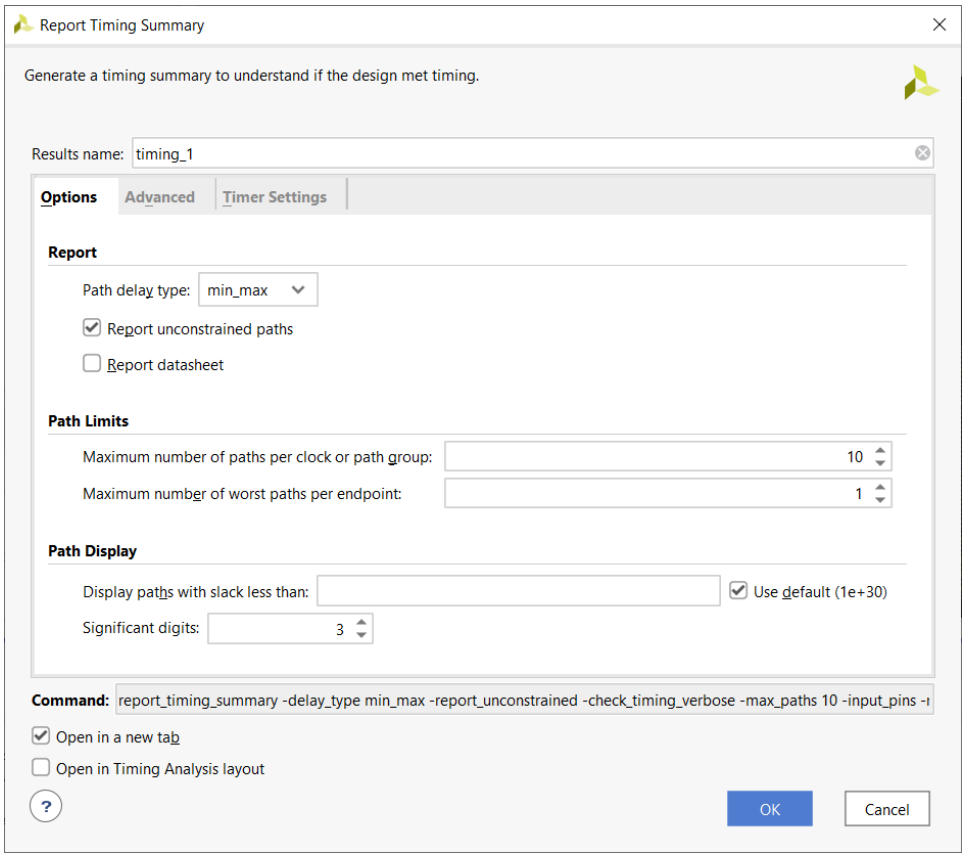
If this field is left empty, the report is returned to the Tcl Console, and a graphical version of the report is not opened in the Results window.

Equivalent Tcl option: `-name`

Options Tab

The Options tab in the Report Timing Summary dialog box is shown in the figure below.

Figure 88: **Report Timing Summary Dialog Box: Options Tab**



Report Section

The Report section of the Options tab of the Report Timing Summary dialog box includes:

- 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. To run min delay analysis only (hold and removal), select delay type `min`.

  Equivalent Tcl option: `-delay_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`

- Report datasheet

  Generates the design datasheet as defined in Report Datasheet, in this chapter.

Equivalent Tcl option: `-datasheet`

Path Limits Section

The Path Limits section of the Options tab of the Report Timing Summary dialog box includes:

- 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. Therefore, the total number of reported paths is still limited by the number of `-max_paths`.

  Equivalent Tcl option: `-nworst`

Path Display Section

The Path Display section of the Options tab of the Report Timing Summary dialog box includes:

- 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`

- Significant digits: Controls the accuracy of the numbers displayed in the report.

  Equivalent Tcl option: `-significant_digits`

Common Section

The following controls common to all three tabs are located at the bottom of the Report Timing Summary dialog box:

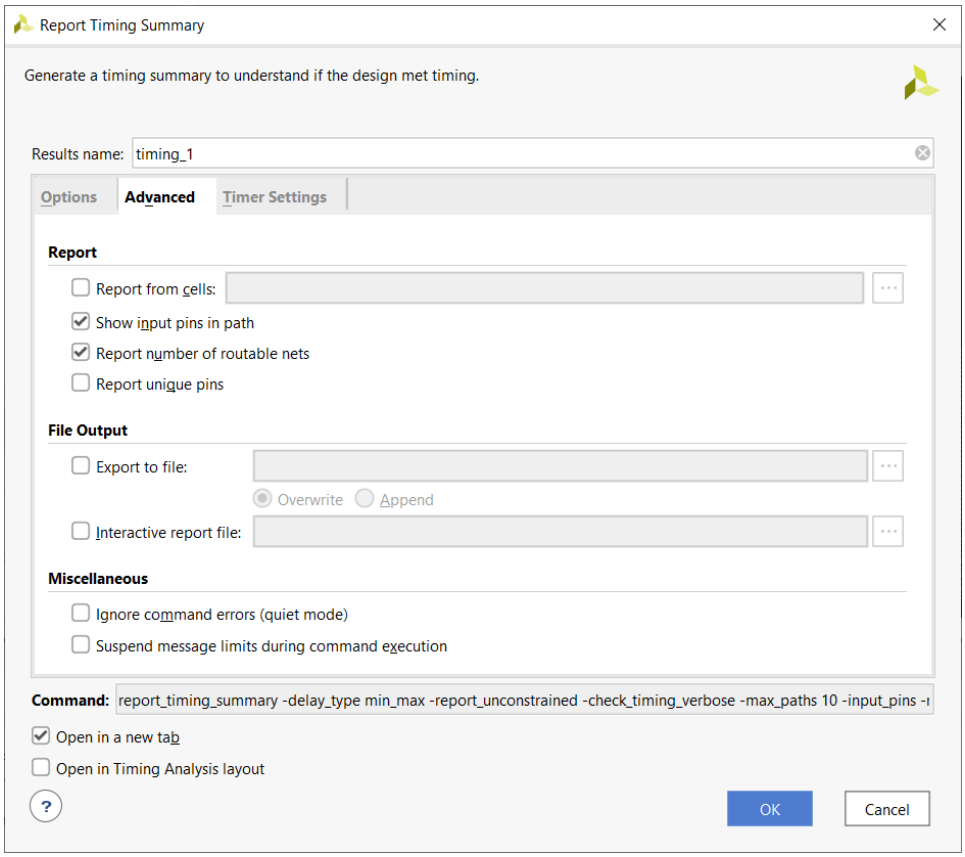
- Command: Displays the Tcl command line equivalent of the various options specified in the Report Timing Summary dialog box.
- Open in a New Tab: Opens the results in a new tab, or to replace the last tab opened by the Results window.
- Open in Timing Analysis Layout: Resets the current view layout to the Timing Analysis view layout.

  For more information on view layouts, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Advanced Tab

The Advanced tab in the Report Timing Summary dialog box is shown in the figure below.

Figure 89: **Report Timing Summary Dialog Box: Advanced Tab**



Report Section

- Report from cell: Enable to limit the timing reporting on the particular cell(s) of the design. Only paths with the datapath section that start, end, cross, or are fully contained inside the cell(s) are reported.

  Equivalent Tcl option: `-cells`

- Show input pins in path: Displays which input pin of the cell is used for the path.

  Equivalent Tcl option: `-input_pins`

  RECOMMENDED: *Keep this option selected to provide more information about all pins used in the path.*

- Report unique Pins: show only one timing path for each unique set of pins.

  Equivalent Tcl option: `-unique_pins`

File Output Section

- Write results to file: Writes the result to the specified file name. By default the report is written to the Timing window in the Vivado IDE.

Equivalent Tcl option: `-file`

- Overwrite/Append: When the report is written to a file, determines whether (1) the specified file is overwritten, or (2) new information is appended to an existing report.

  Equivalent Tcl option: `-append`

- Interactive report file: Writes the result in the AMD RPX format to the specified filename. The RPX file is an interactive report that contains all the report information and can be reloaded into memory in the Vivado Design Suite using the `open_report` command.

Miscellaneous Section

- Ignore command errors: Executes the command quietly, ignoring any command line errors and returning no messages. The command also returns TCL_OK regardless of any errors encountered during execution.

  Equivalent Tcl option: `-quiet`

- Suspend message limits during command execution: Temporarily overrides any message limits and return all messages.

  Equivalent Tcl option: `-verbose`

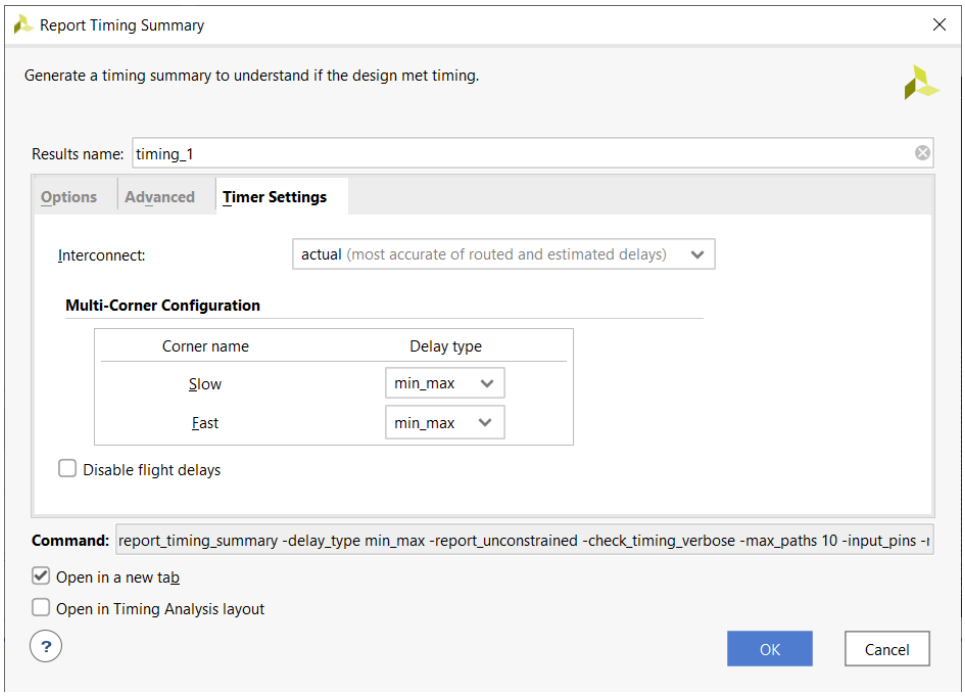
Timer Settings Tab

To set the timer settings, use either: (1) one of the Vivado IDE timing analysis dialog boxes; or, (2) one of the Tcl commands listed in this section. These settings affect other timing-related commands run in the same Vivado IDE session, except the synthesis and implementation commands.

The timer settings are not saved as a tool preference. The default values are restored for each new session. Do not change the default values. Keeping the default values provides maximum timing analysis coverage with the most accurate delay values.

The Timer Settings tab in the Report Timing Summary dialog box is shown in the figure below.

Figure 90: **Report Timing Summary Dialog Box: Timer Settings Tab**



Interconnect Setting

Controls whether net delays are calculated based on the estimated route distance between leaf cell pins, by the actual routed net, or excludes net delay from timing analysis. 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 of the best possible placement, based on the nature of the driver and loads as well as the fanout. A net between unplaced leaf cell pins 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 of the routed interconnect. This net is labeled `routed` in the timing path report.
- None: Interconnect delays are not considered in the timing report and net delays are forced to zero.

Equivalent Tcl command: `set_delay_model`

Multi-Corner Configuration Setting

Specifies the type of path delays to be analyzed for the specified timing corner. Valid values are `none`, `max`, `min`, and `min_max`. Select `none` to disable timing analysis for the specified corner.

RECOMMENDED: *Keep both setup (max) and hold (min) analysis selected for both corners.*

Equivalent Tcl command: `config_timing_corners`

Disable Flight Delays

Do not add package delays to I/O delay calculations.

Equivalent Tcl command: `config_timing_analysis`

Details of the Timing Summary Report

The Timing Summary Report contains the following sections:

- General Information Section
- Timer Settings Section
- Design Timing Summary Section
- Clock Summary Section
- Methodology Summary Section
- Check Timing Section
- Intra-Clock Paths Section
- Inter-Clock Paths Section
- Other Path Groups Section
- User-Ignored Paths Section
- Unconstrained Paths Section

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.

General Information Section

The General Information section of the Timing Summary Report 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

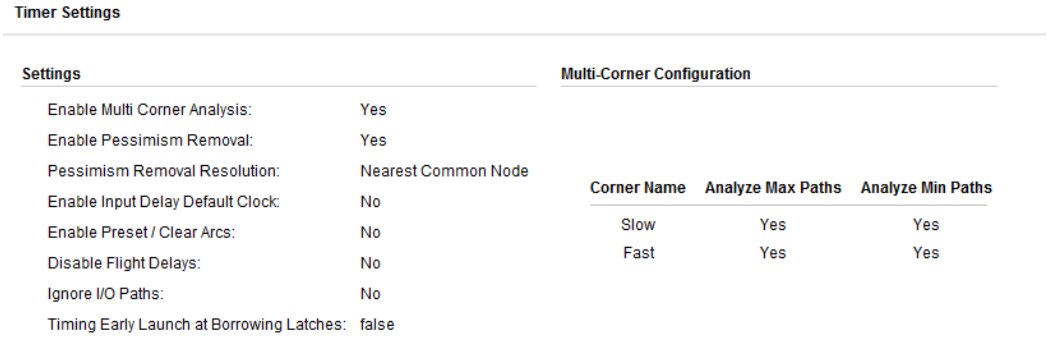
Timer Settings Section

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

- Enable Multi-Corner Analysis: This analysis is enabled for each corner (Multi-Corner Configuration).
- Enable Pessimism Removal (and Pessimism Removal Resolution): Ensures that the source and destination clocks of each path are reported with no skew at their common node.

  Note: This setting must always be enabled.
- Enable Input Delay Default Clock: Creates a default null input delay constraint on input ports with no user constraint. It is disabled by default.
- Enable Preset / Clear Arcs: Enables timing path propagation through asynchronous pins. It does not affect recovery/removal checks and is disabled by default.
- Disable Flight Delays: Disables package delays for I/O delay calculations.

Figure 91: **Timing Summary Report: Timer Settings**



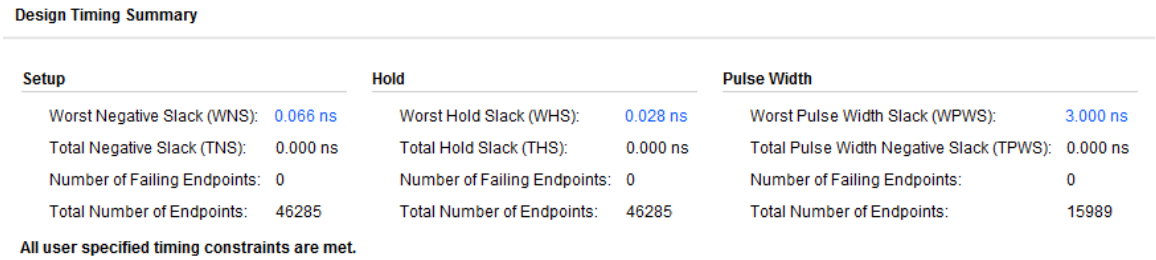
For additional information on default timer settings and how to change them, see `config_timing_analysis`, available from this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Design Timing Summary Section

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

RECOMMENDED: *Review the Design Timing Summary section to verify that all timing constraints are met after route, or to understand the status of the design at any point in the flow.*

Figure 92: **Design Timing Summary**



The Design Timing Summary section includes the following:

- Setup Area (Max Delay Analysis)
- Hold Area (Min Delay Analysis)
- Pulse Width Area (Pin Switching Limits)

Setup Area (Max Delay Analysis)

The Setup area of the Design Timing Summary section displays all checks related to max delay analysis: setup, recovery, and data check.

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

Hold Area (Min Delay Analysis)

The Hold area of the Design Timing Summary section displays all checks related to min delay analysis: hold, removal, and data check.

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

- Total Number of Endpoints: The total number of endpoints analyzed.

Pulse Width Area (Pin Switching Limits)

The Pulse Width area of the Design Timing Summary section displays 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, such as for PCIe or GT [UltraScale devices only]).

The reported values are:

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

Clock Summary Section

The Clock Summary section of the Timing Summary Report includes information similar to that produced by `report_clocks`:

- All the clocks in the design (whether created by `create_clock`, `create_generated_clock`, or automatically by the tool).
- The properties for each clock, such as name, period, waveform, and target frequency.



TIP: *The indentation of names reflects the relationship between master and generated clocks.*

Figure 93: **Timing Summary Report: Clock Summary**

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 5.000}	10.000	100.000
clkfbout	{0.000 5.000}	10.000	100.000
cpuClk_5	{0.000 10.000}	20.000	50.000
fftClk_0	{0.000 5.000}	10.000	100.000
phyClk0_2	{0.000 5.000}	10.000	100.000
phyClk1_1	{0.000 5.000}	10.000	100.000
usbClk_3	{0.000 5.000}	10.000	100.000

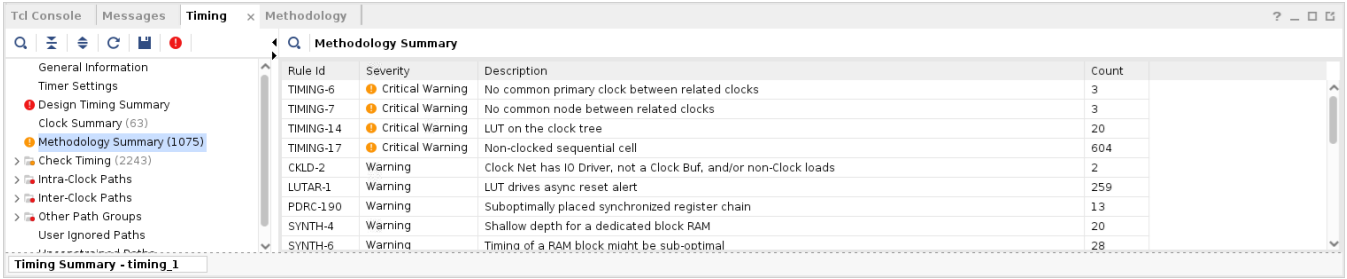
Methodology Summary Section

The Methodology Summary section of the Timing Summary report includes a table of methodology violations. The total number of violations is reported on the right hand side of the Methodology Summary. An icon in front of the category name indicates that an Error or Critical Warning needs to be reviewed. A red icon indicates that the highest severity is an Error, and an orange icon indicates when it is a Critical Warning. There is no icon for Warning or Advisory violations.

***Note*:** Report Timing Summary does *not* run Report Methodology. It only reports a summary of violations tracked from the latest run of `report_methodology` (inside the design in-memory or through reloading the checkpoint). To generate a detailed list of methodology violations as a standalone report, do one of the following:

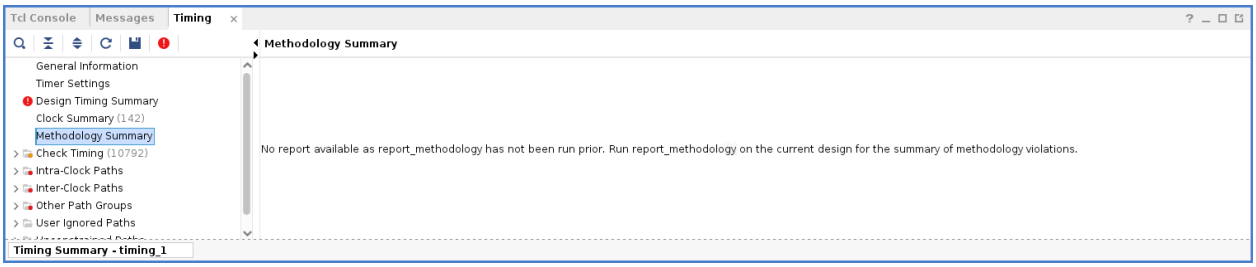
- Select **Reports → Report Methodology** from the menu.
- Run the Tcl `report_methodology` command.

Figure 94: **Methodology Summary**



If `report_methodology` has not been run prior to running Report Timing Summary, a message is shown explaining that no report is available, as shown in the following figure.

Figure 95: **Methodology Summary without Running Report Methodology**



For more information about Report Methodology, refer to Methodology Analysis.

Check Timing Section

The Check Timing section of the Timing Summary Report contains 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.

For more information on constraints definition, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Figure 96: **Timing Summary Report: Check Timing Section**

Check Timing

Timing Check	Count	Worst Severity
pulse_width_clock	8	Low
no_input_delay	1	Medium
no_clock	0	
constant_clock	0	
unconstrained_internal_endpoints	0	
no_output_delay	0	
multiple_clock	0	
generated_clocks	0	
loops	0	
partial_input_delay	0	
partial_output_delay	0	
latch_loops	0	

To generate Check Timing as a standalone report, do one of the following:

- Run the **Reports → Timing → Check Timing** menu command.
- Run the Tcl `check_timing` command.

When run from the Tcl console, the check_timing report can be scoped to one or more hierarchical cells using the `-cells` option. This option is not available from the Check Timing GUI.

The list of checks reported by default, as shown in the previous figure is:

- `pulse_width_clock`: Reports clock pins that have only a pulse width check associated with the pin, and no setup or hold check, no recovery, removal, or `clk` > Q check.
- `no_input_delay`: Number of non-clock input ports without any input delay constraints.
- `no_clock`: Number of clock pins not reached by a defined timing clock. Constant clock pins are also reported.
- `constant_clock`: Checks for clock signals connected to a constant signal (gnd/vss/data).
- `unconstrained_internal_endpoints`: Number of path endpoints (excluding output ports) without a timing requirement. This number is directly related to missing clock definitions, which is also reported by the `no_clock` check.
- `no_output_delay`: Number of non-clock output ports without at least one output delay constraint.
- `multiple_clock`: Number of clock pins reached by more than one timing clock. This can happen if there is a clock multiplexer in one of the clock trees. The clocks that share the same clock tree are timed together by default, which does not represent a realistic timing situation. Only one clock can be present on a clock tree at any given time.

  If you do not believe that the clock tree is supposed to have a MUX, review the clock tree to understand how and why multiple clocks are reaching the specific clock pins.
- `generated_clocks`: Number of generated clocks that refer to a master clock source which is not part of the same clock tree. This situation can occur when a timing arc is disabled on the logical path between the master clock and the generated clock source points. This check also applies to individual edges of the generated clocks when specified with the `-edges` option: the logical path unateness (inverting/non-inverting) must match the edge associations between the master and generated clocks.
- `loops`: Number of combinational loops found in the design. The loops are automatically broken by the Vivado IDE timing engine to report timing.
- `partial_input_delay`: Number of non-clock 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 non-clock output ports with only a min output delay or max output delay constraint. These ports are not reported by both setup and hold analysis.
- `latch_loops`: Checks for and warns of loops passing through latches in the design. These loops will not be reported as part of combinational loops, and will affect latch time borrowing computation on the same paths.

Intra-Clock Paths Section

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

Figure 97: **Timing Summary Report: Intra-Clock Paths Section**

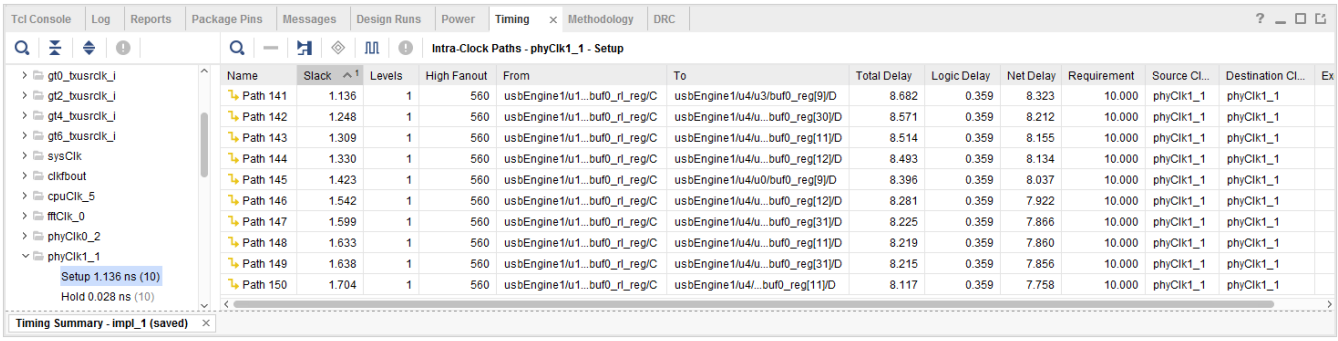
Intra-Clock Paths

Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)	WPWS (ns)	TPWS (ns)	Failing Endpoints (TPWS)	Total Endpoints (TPWS)
gt0_txusrclk_i	rise - rise	9.887	0.000	0	218	rise - rise	0.135	0.000	0	218	6.000	0.000	0	154
gt2_txusrclk_i	rise - rise	9.668	0.000	0	218	rise - rise	0.095	0.000	0	218	6.000	0.000	0	154
gt4_txusrclk_i	rise - rise	8.853	0.000	0	218	rise - rise	0.123	0.000	0	218	6.000	0.000	0	154
gt6_txusrclk_i	rise - rise	9.339	0.000	0	218	rise - rise	0.120	0.000	0	218	6.000	0.000	0	154
sysClk											3.000	0.000	0	10
clkfbout											8.592	0.000	0	3
cpuClk_5	rise - rise	9.754	0.000	0	5761	rise - rise	0.065	0.000	0	5761	9.600	0.000	0	3369
fftClk_0	rise - rise	3.658	0.000	0	8320	rise - rise	0.046	0.000	0	8320	4.358	0.000	0	1438
phyClk0_2	rise - rise	1.759	0.000	0	5958	rise - rise	0.076	0.000	0	5958	4.358	0.000	0	3787
phyClk1_1	rise - rise	1.136	0.000	0	5958	rise - rise	0.028	0.000	0	5958	4.358	0.000	0	3787
usbClk_3	rise - rise	1.114	0.000	0	2554	rise - rise	0.049	0.000	0	2554	4.600	0.000	0	1482
wbClk_4	rise - rise	9.921	0.000	0	2855	rise - rise	0.082	0.000	0	2855	9.600	0.000	0	1497

To view detailed information, click the names under Intra-Clock Paths in the left index pane. 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. The N-worst is defined using the `-max_paths` on the command line or the maximum number of paths per clock or path group (GUI).

The worst slack value and the number of reported paths are displayed next to the label for each analysis type. In the following figure, a Setup summary is selected under the Intra-Clock Paths section in the left index pane and a table listing all the paths related to that clock is displayed in the right pane.

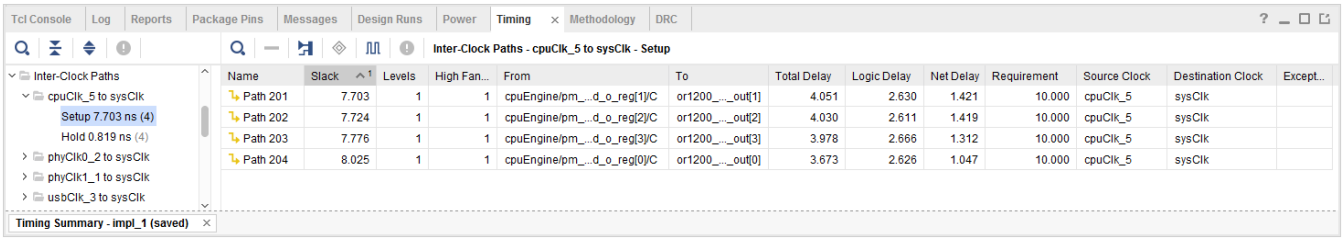
Figure 98: **Timing Summary Report: Intra-Clock Paths Details**



Inter-Clock Paths Section

Similar to the Intra-Clock Paths section, the Inter-Clock Paths section of the Timing Summary Report (shown in the following figure) summarizes the worst slack and total violations of the timing paths between different source and destination clocks.

Figure 99: **Timing Summary Report Inter-Clock Paths Details**

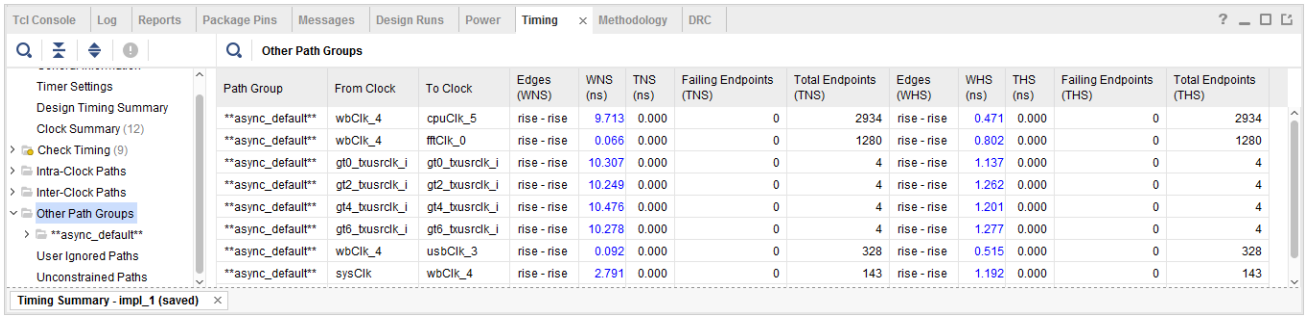


To view detailed information, click the names under Inter-Clock Paths in the left index pane. For example, you can view the slack and violations summary for each clock and details about the N-worst paths for SETUP/HOLD/Pulse Width checks. The N-worst is defined using the `-max_paths` on the command line or the maximum number of paths per clock or path group (GUI).

Other Path Groups Section

The Other Path Groups section of the Timing Summary Report displays default path groups and user-defined path groups. The following figure shows an example of the Other Path Groups summary table. To access this table, select **Other Path Groups** in the left pane.

Figure 100: **Timing Summary Report: Path Groups Section**

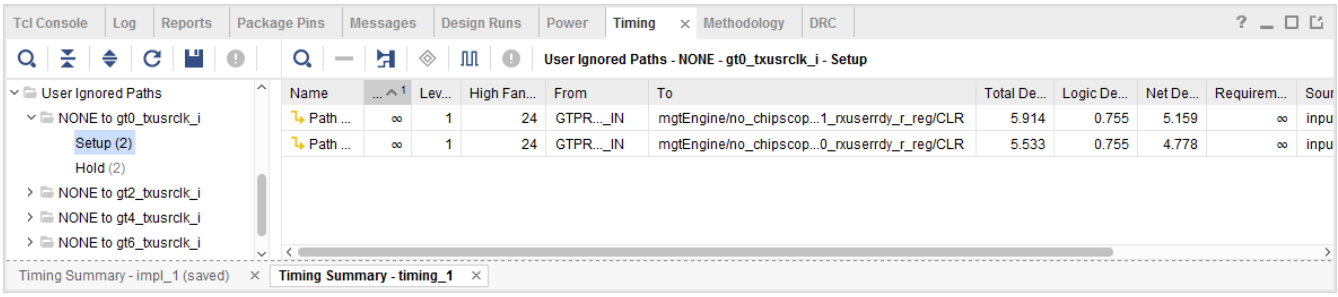


TIP: ***async_default** is a path group automatically created by the Vivado IDE timing engine. It includes all paths ending with an asynchronous timing check, such as recovery and removal. These two checks are respectively reported under SETUP and HOLD categories, which corresponds to max delay analysis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

User-Ignored Paths Section

The User-Ignored Paths Section of the Timing Summary Report (shown in the following figure) displays the paths that are ignored during timing analysis due to the `set_clock_groups` and `set_false_path` constraints. The reported slack is infinite.

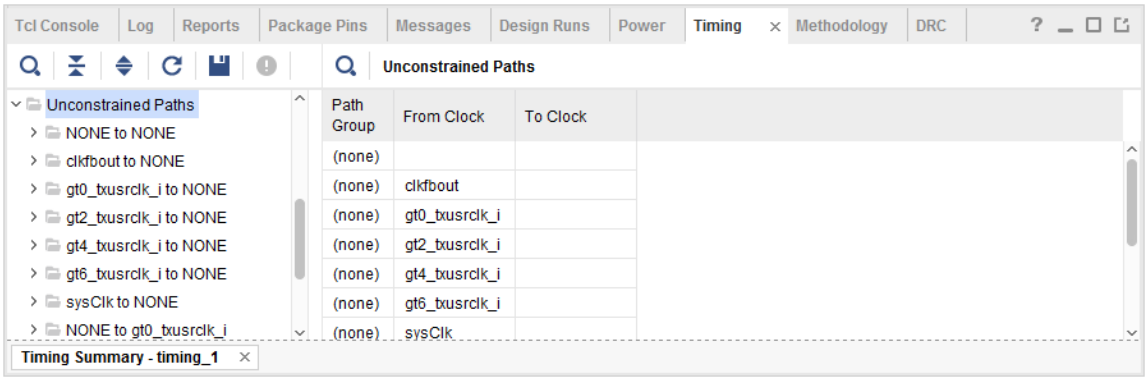
Figure 101: **Timing Summary Report: User-Ignored Paths Section**



Unconstrained Paths Section

The Unconstrained Paths section of the Timing Summary Report displays the logical paths that are not timed due to missing timing constraints. These paths are grouped by source and destination clock pairs. The clock name information displays as empty (or `NONE`) when no clock can be associated with the path startpoint or endpoint.

Figure 102: **Timing Summary Report: Unconstrained Paths Section**



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 view the N-worst reported paths. Select any of these paths to view more details in the Path Properties window (Report view).

To view the same details in a new window, double click the path.

For more information on timing path details, see Chapter 4: Timing Analysis.

To access more analysis views for each path:

1. Right click the path in the right pane.
2. Select one of the following options from the popup menu:
   - **Schematic**: Open a schematic of the path.

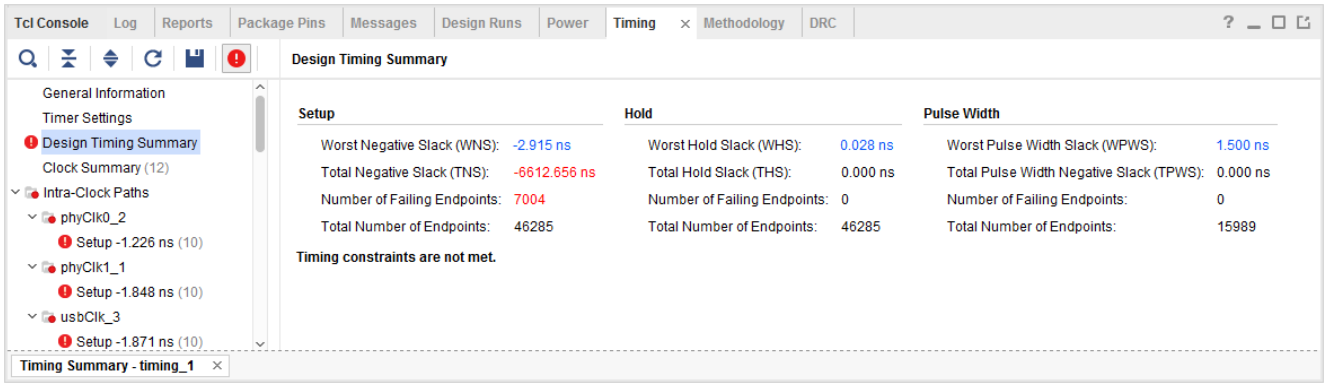
- **Report Timing on Source to Destination**: Rerun timing analysis on this same path.
- **Highlight**: Highlight the path in the Device and Schematic windows.

Filtering Paths with Violations

The report displays the slack value of failing paths in red. To focus on these violations, click the **Show only failing paths** button .

The following figure shows the Timing Summary window with only failing paths displayed.

Figure 103: **Timing Summary Report: Violating Paths Filter**



Report Bus Skew

The Bus Skew report covers all the bus skew constraints set in the design using `set_bus_skew`. The bus skew report is currently not included inside the timing summary report and you must manually generate the bus skew report in addition to the timing summary report for complete timing signoff.

When run from the Tcl Console, the bus skew report can be scoped to one or more hierarchical cells using the `-cells` option. When the report is scoped, only paths with the datapath section that start, end, or are fully contained inside the cell(s) are reported. This option is not available from the Report Bus Skew GUI.

Running Report Bus Skew

The bus skew report is available from the command line with the `report_bus_skew` Tcl command or from the GUI under **Reports → Timing → Report Bus Skew**. This command shares many options with the `report_timing` command to filter and format the output report. The bus skew constraints are reported in their order of definition. Use `-sort_by_slack` to sort the constraints in an ascending order, from the smallest to the largest slack.

Reviewing Bus Skew Path Details

The bus skew report includes two sections:

1. A bus skew report summary.
2. A bus skew report by constraint.

Bus Skew Report Summary Section

The Bus Skew Report Summary section reports all the `set_bus_skew` constraints defined in the design. The following information is reported for each constraint:

- **Id:** Constraint ID referred to later in the report. This information makes it easier to search for that constraint within the report.
- **From:** Pattern provided for the `set_bus_skew -from` option.
- **To:** Pattern provided for the `set_bus_skew -to` option.
- **Corner:** Corner (Slow or Fast) in which the worst bus skew was computed.
- **Requirement:** Bus skew target value.
- **Actual:** Worst bus skew computed across all the paths covered by the constraint.
- **Slack:** Difference between the worst bus skew and the constraint requirement.

In the following example, the design has only one bus skew constraint with a 1 ns requirement. The worst skew among all the paths covered by the constraint is 1.107 ns.

```
1. Bus Skew Report Summary
--------------------------

Id  Position  From                          To                            Corner  Requirement(ns)  Actual(ns)  Slack(ns)
--  --------  ----------------------------  ----------------------------  ------  ---------------  ----------  ---------
1   4         [get_pins {data1_reg[*]/C}]   [get_pins {data2_reg[*]/D}]  Slow             1.000       1.107     -0.107
```

Note: A bus skew violation (WBSS) indicates that there is more skew between the different bits of an asynchronous bus than expected. This can result in incorrect data on the bus being captured in the destination clock domain. In such cases, the different bits of the bus could reflect a state sent by the source clock domain at different clock cycles. When one or more WBSS violations are left after routing, try a different routing or placement directive. For hardware stability, it is recommended to ensure that there is no bus skew violation left in the design.

Bus Skew Report Per Constraint Section

The Bus Skew Report Per Constraint section provides more details for each of the `set_bus_skew` constraints. Each reported constraint includes two parts:

1. Detailed summary of the paths covered by the constraint.

2. Detailed timing paths for the paths reported in the Per Constraint summary.

The detailed summary table provides the following information:

- **From Clock:** Startpoints clock domain.
- **To Clock:** Endpoints clock domain.
- **Endpoint Pin:** Endpoint pin involved in the reported path.
- **Reference Pin:** Reference pin used to compute the skew. Each row of this table refers to the reference pin which results in the largest skew for that endpoint path.
- **Corner:** Fast/slow corner used to compute the worst skew to this endpoint.
- **Actual:** Computed skew. The skew is the difference between the relative delay for Endpoint Pin minus the relative delay for Reference Pin and minus the relative CRPR.
- **Slack:** The difference between the actual path skew and the requirement.

***Note*:** Both the `-from` and `-to` options must be specified when defining a bus skew constraint.

By default, only the endpoint with the worst bus skew is reported. To report multiple endpoints, the command line options `-max_paths` and `-nworst` can be used. They work similarly as for the `report_timing` command. For example, the combination of `-nworst 1 -max_path 16` reports, for each constraint, up to 16 endpoints with a single path per endpoint.

Figure 104: **Bus Skew Report Per Constraint**

```
2. Bus Skew Report Per Constraint
---------------------------------

Id: 1
set_bus_skew -from [get_pins {data1_reg[*]/C}] -to [get_pins {data_reg[*]/D}] 1.000
Requirement: 1.000ns

From Clock   To Clock   Endpoint Pin      Reference Pin     Corner  Actual(ns)  Slack(ns)
----------  ---------  ----------------  ----------------  ------  ----------  ---------
clk1         clk2       data_reg[1]/D     data_reg[2]/D     Slow         1.107     -0.107
clk1         clk2       data_reg[2]/D     data_reg[1]/D     Slow         1.107     -0.107
clk1         clk2       data_reg[3]/D     data_reg[2]/D     Slow         0.986      0.014
clk1         clk2       data_reg[0]/D     data_reg[2]/D     Slow         0.920      0.080
```

The detailed timing paths section provides a detailed timing path for each of the pin pairs reported in the Per Constraint summary table. The number of detailed paths that are reported is the same as the number of endpoints reported in the summary table and can be controlled with `-max_paths`/`-nworst` command line options.

The format for the detailed bus skew timing path is similar to a traditional timing path. Clock uncertainty, however, is not included because the bus skew analysis is done on the same clock edge. Instead, the worst of the clock uncertainty from the endpoint or reference paths is reported in the bus skew header. The launch time of the destination clock is always zero. For each slack, a timing path to the endpoint and a timing path to the reference pin are printed. When the clock or the datapath cross multiple SLRs, some inter-SLR compensation is applied during the slack computation to prevent unnecessary pessimism. Such compensation is then reported in the bus skew header.

The following detailed path was reported using the command line option `-path_type short` to collapse the clock network details. The path to the endpoint pin precedes the path to the reference pin. The path header summarizes the information from the two detailed paths, plus the requirement and the relative CRPR:

Figure 105: **Detailed Path Example**

```
Slack (MET) :             0.536ns  (requirement - actual skew)
  Endpoint Source:        datal_reg[1]/C
                            (rising edge-triggered cell FDRE clocked by clk1)
  Endpoint Destination:   datac_reg[1]/D
                            (rising edge-triggered cell FDRE clocked by clk2)
  Reference Source:       datal_reg[2]/C
                            (rising edge-triggered cell FDRE clocked by clk1)
  Reference Destination:  datac_reg[2]/D
                            (rising edge-triggered cell FDRE clocked by clk2)
  Path Type:              Bus Skew (Max at Slow Process Corner)
  Requirement:            1.000ns
  Endpoint Relative Delay:    1.068ns
  Reference Relative Delay:  -0.100ns
  Relative CRPR:              0.704ns
  Actual Bus Skew:            0.464ns  (Endpoint Relative Delay - Reference Relative Delay - Relative CRPR)

Endpoint path:
    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock clk1 rise edge)       0.000     0.000 r
                         propagated clock network latency
                                                      5.389     5.389
    SLICE_X47Y53         FDRE                         0.000     5.389 r  datal_reg[1]/C
    SLICE_X47Y53         FDRE (Prop_fdre_C_Q)         0.269     5.658 r  datal_reg[1]/Q
    SLICE_X46Y53         net (fo=1, estimated)        0.432     6.090 r  datac_reg[1]/D
  -------------------------------------------------------------------    -------------------

                         (clock clk2 rise edge)       0.000     0.000 r
                         propagated clock network latency
                                                      5.040     5.040
                         clock pessimism              0.000     5.040
    SLICE_X46Y53         FDRE (Setup_fdre_C_D)       -0.018     5.022    datac_reg[1]
  -------------------------------------------------------------------
                         data arrival                           6.090
                         clock arrival                          5.022
  -------------------------------------------------------------------
                         relative delay                         1.068

Reference path:
    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock clk1 rise edge)       0.000     0.000 r
                         propagated clock network latency
                                                      5.016     5.016
    SLICE_X47Y53         FDRE                         0.000     5.016 r  datal_reg[2]/C
    SLICE_X47Y53         FDRE (Prop_fdre_C_Q)         0.216     5.232 r  datal_reg[2]/Q
    SLICE_X46Y53         net (fo=1, estimated)        0.274     5.505 r  datac_reg[2]/D
  -------------------------------------------------------------------    -------------------

                         (clock clk2 rise edge)       0.000     0.000 r
                         propagated clock network latency
                                                      5.414     5.414
                         clock pessimism              0.000     5.414
    SLICE_X46Y53         FDRE (Hold_fdre_C_D)         0.191     5.605    datac_reg[2]
  -------------------------------------------------------------------
                         data arrival                           5.505
                         clock arrival                          5.605
  -------------------------------------------------------------------
                         relative delay                        -0.100
```

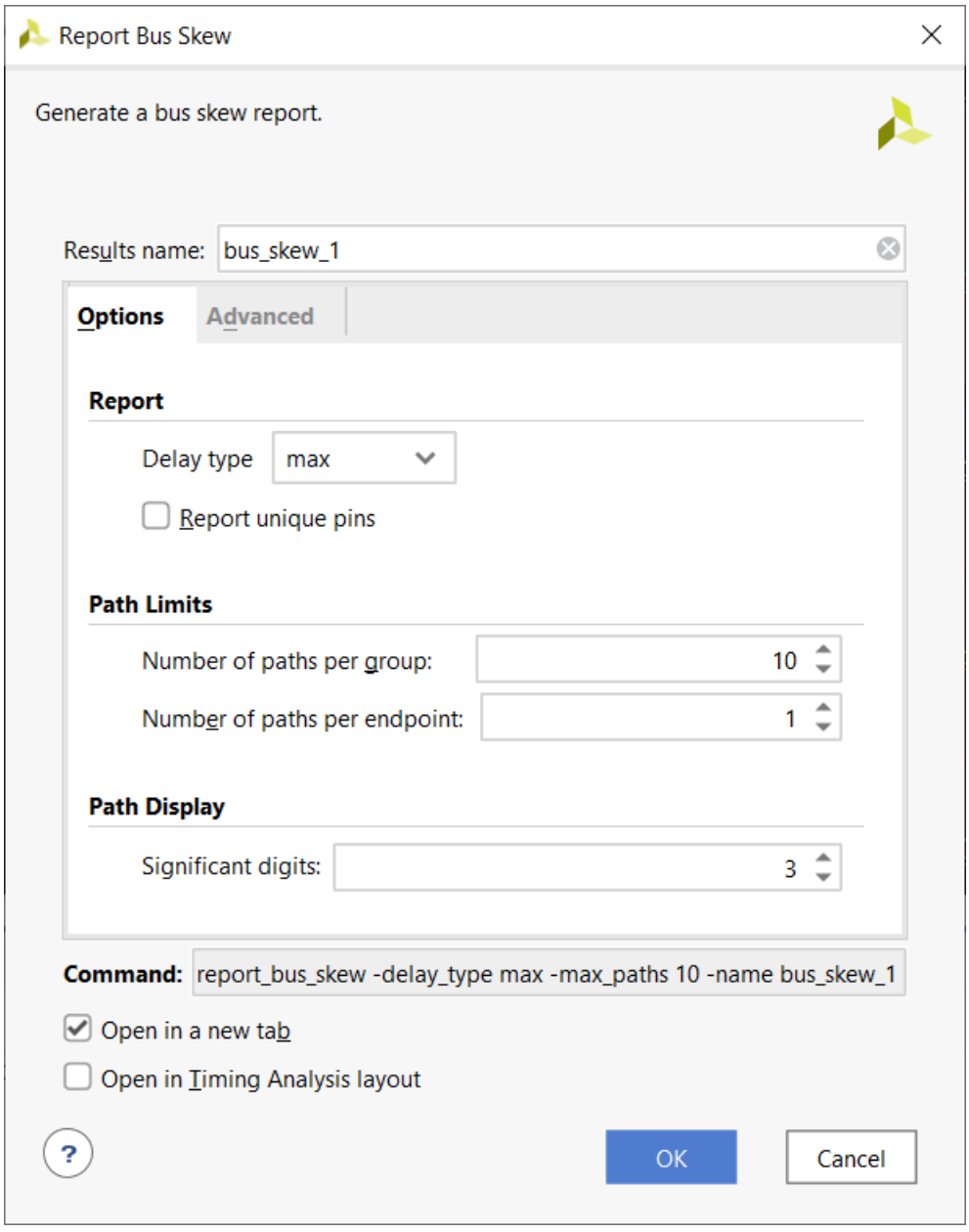
Report Bus Skew Dialog Box

In the AMD Vivado™ IDE, to open the Report Bus Skew dialog box, select **Reports → Timing → Report Bus Skew**.

Report Bus Skew Dialog Box: Options Tab

The Options tab in the Report Bus Skew dialog box is shown in the following figure.

Figure 106: **Report Bus Skew Dialog Box: Options Tab**



The Report Bus Skew Dialog Box Options tab includes the following sections:

- Report
- Path Limits
- Path Display

Report

- Delay type: See Report Section.
- Report unique pins: Show only one timing path for each unique set of pins.

  Equivalent Tcl option: `-unique_pins`.

Path Limits

- Number of paths per group: See Report Timing Summary.
- Number of paths per endpoint: See Report Timing Summary.

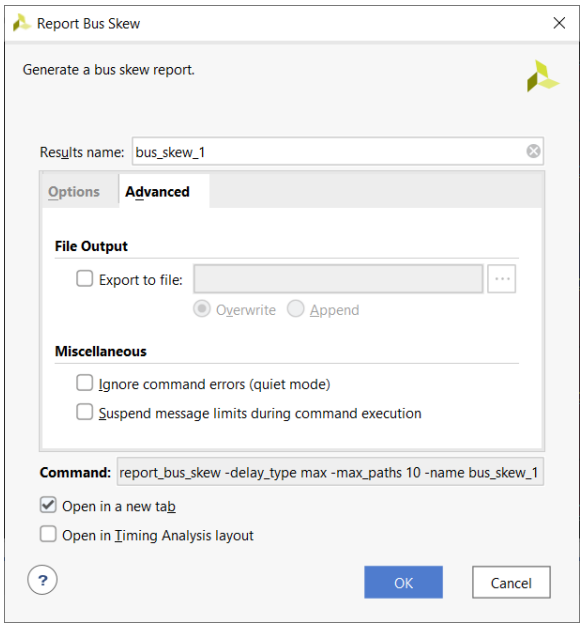
Path Display

- Number of significant digits: See Report Timing Summary.

Report Bus Skew Dialog Box: Advanced Tab

The Advanced tab in the Report Bus Skew dialog box is shown in the figure below.

Figure 107: **Report Bus Skew Dialog Box: Advanced Tab**



The Report Bus Skew Dialog Box Advanced tab includes the following sections:

- File Output
- Miscellaneous

File Output

- Write results to file: Writes the result to the specified file name. By default the report is written to the Timing window in the Vivado IDE.

  Equivalent Tcl option: `-file`.

- Overwrite/Append: When the report is written to a file, determines whether (1) the specified file is overwritten, or (2) new information is appended to an existing report.

  Equivalent Tcl option: `-append`.

Miscellaneous

- Ignore command errors: Executes the command quietly, ignoring any command line errors and returning no messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

  Equivalent Tcl option: `-quiet`.

- Suspend message limits during command execution: Temporarily overrides any message limits and returns all messages.

  Equivalent Tcl option: `-verbose`.

Details of the Timing Summary Report

The Bus Skew Report contains the following sections:

- General Information Section
- Summary Section
- Set Bus Skew Section

General Information Section

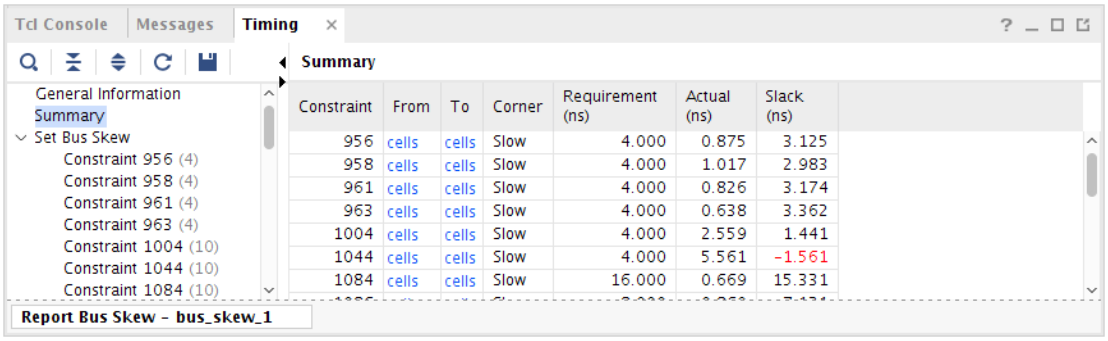
The General Information section of the Timing Summary Report provides information about the following:

- 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

Summary Section

This section provides a summary of all the Bus Skew constraints, with their requirements, the actual worst bus skew, and slack for each constraint. The summary table can be used to quickly see whether any of the Bus Skew constraints is violated.

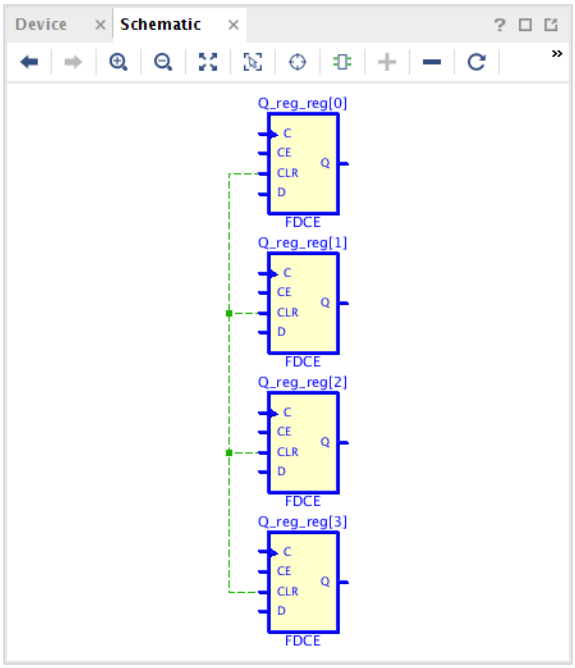
Figure 108: **Report Bus Skew: Summary Section**



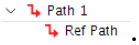
The Constraint table column represents the timing constraint position number that matches the position number reported inside the Timing Constraints Editor (TCE).

The From and To table columns include hyperlinks to the objects specified inside the `set_bus_skew` command. The objects can be cells or pins. It is possible to click on the hyperlinks to select all the startpoints or endpoints of a specific Bus Skew constraint. After the objects have been selected, the schematic can be opened with F4.

Figure 109: **Example Schematic of the Bus Skew Endpoints**



Set Bus Skew Section

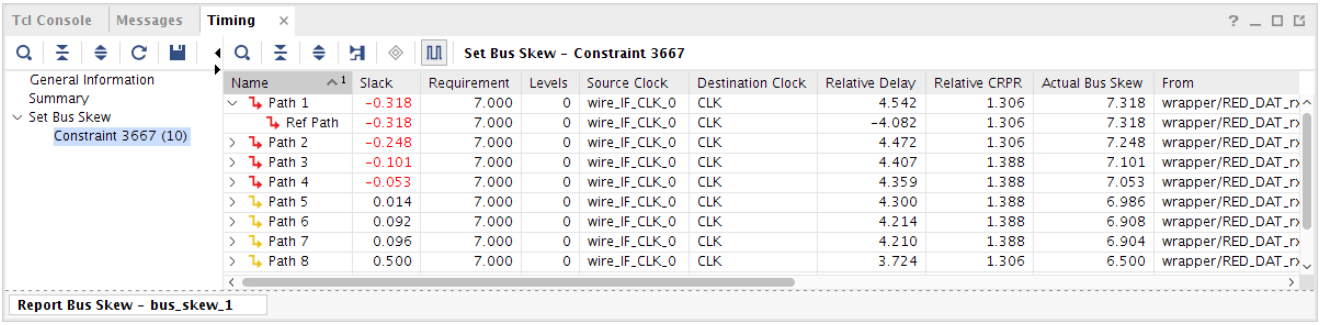
This section provides access to the detailed timing paths for each Bus Skew constraint. There is an associated reference path that can be expanded for each timing path endpoint 

.

The bus skew calculation is:

Actual Bus Skew = Endpoint Relative Delay - Reference Relative Delay - Relative CRPR

Figure 110: **Example of Detail Path for the First Endpoint and its Reference Path**



Name	Slack	Requirement	Levels	Source Clock	Destination Clock	Relative Delay	Relative CRPR	Actual Bus Skew	From
Path 1	-0.318	7.000	0	wire_IF_CLK_0	CLK	4.542	1.306	7.318	wrapper/RED_DAT_r
Ref Path	-0.318	7.000	0	wire_IF_CLK_0	CLK	-4.082	1.306	7.318	wrapper/RED_DAT_r
Path 2	-0.248	7.000	0	wire_IF_CLK_0	CLK	4.472	1.306	7.248	wrapper/RED_DAT_r
Path 3	-0.101	7.000	0	wire_IF_CLK_0	CLK	4.407	1.388	7.101	wrapper/RED_DAT_r
Path 4	-0.053	7.000	0	wire_IF_CLK_0	CLK	4.359	1.388	7.053	wrapper/RED_DAT_r
Path 5	0.014	7.000	0	wire_IF_CLK_0	CLK	4.300	1.388	6.986	wrapper/RED_DAT_r
Path 6	0.092	7.000	0	wire_IF_CLK_0	CLK	4.214	1.388	6.908	wrapper/RED_DAT_r
Path 7	0.096	7.000	0	wire_IF_CLK_0	CLK	4.210	1.388	6.904	wrapper/RED_DAT_r
Path 8	0.500	7.000	0	wire_IF_CLK_0	CLK	3.724	1.306	6.500	wrapper/RED_DAT_r

It is possible to select a path and look at the detailed timing path report inside the Property pane. The schematic of the path and/or reference path can be generated by clicking the schematic icon and pressing F4 (the endpoint path and its reference path can be selected together).

Report Clock Networks

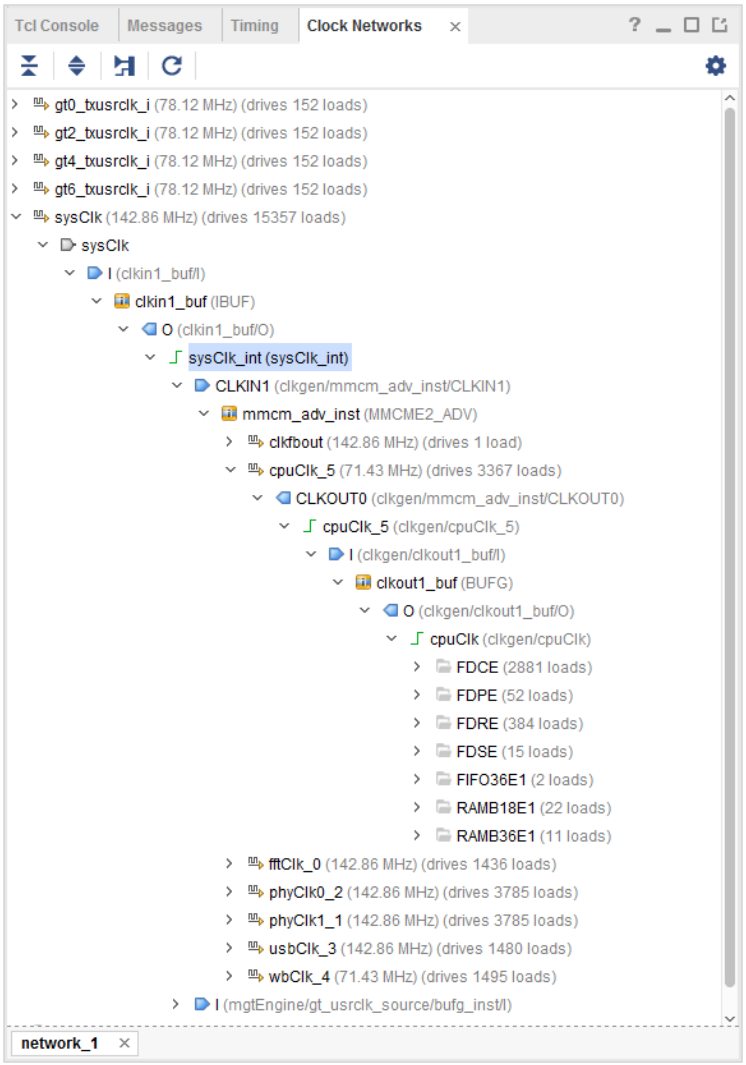
The Report Clock Network command can be run from:

- The Flow Navigator in the AMD Vivado™ IDE, or
- The Tcl command:

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

Report Clock Networks provides a tree view of the clock trees in the design. See the following figure. Each clock tree shows the clock network from source to endpoint with the endpoints sorted by type.

Figure 111: **Clock Networks**



The clock trees:

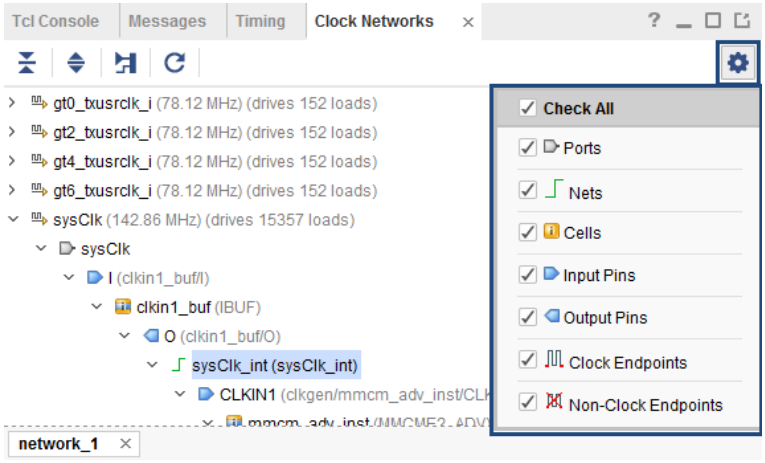
- Show clocks defined by the user or generated automatically by the tool.
- Report clocks from I/O port to load.

  Note: The full clock tree is only detailed in the GUI form of the report. The text version of this report shows only the name of the clock roots.
- Can be used to find BUFGs driving other BUFGs.
- Shows clocks driving non-clock loads.

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

Use the filter Ports, Nets, Instances, and related buttons to reduce the amount of data displayed in the clock tree. The filter options can be viewed by clicking on the icon.

Figure 112: **Clock Networks Filter**



To view a schematic of the clock path:

1. Select an object in the tree.
2. Run the **Trace to Source** popup command.

Report Clock Interaction

To view the Clock Interaction Report, select one of the following:

- **Reports → Timing → Report Clock Interaction**
- **Flow Navigator → Synthesis → Report Clock Interaction**
- **Flow Navigator → Implementation → Report Clock Interaction**

Equivalent Tcl command: `report_clock_interaction -name clocks_1`

When run from the Tcl Console, the interaction report can be scoped to one or more hierarchical cells using the `-cells` option. When the report is scoped, only paths with the datapath section that start, end, cross, or are fully contained inside the cell(s) are reported.

Report Clock Interaction Dialog Box

In the AMD Vivado™ IDE, the Report Clock Interaction dialog box includes the following:

- Results Name Field
- Command Field

- Open in a New Tab Check Box
- Options Tab
- Timer Settings Tab

Results Name Field

The Results name field at the top of the Report Clock Interaction dialog box specifies the name of the graphical report that opens.

Equivalent Tcl option: `-name`

Command Field

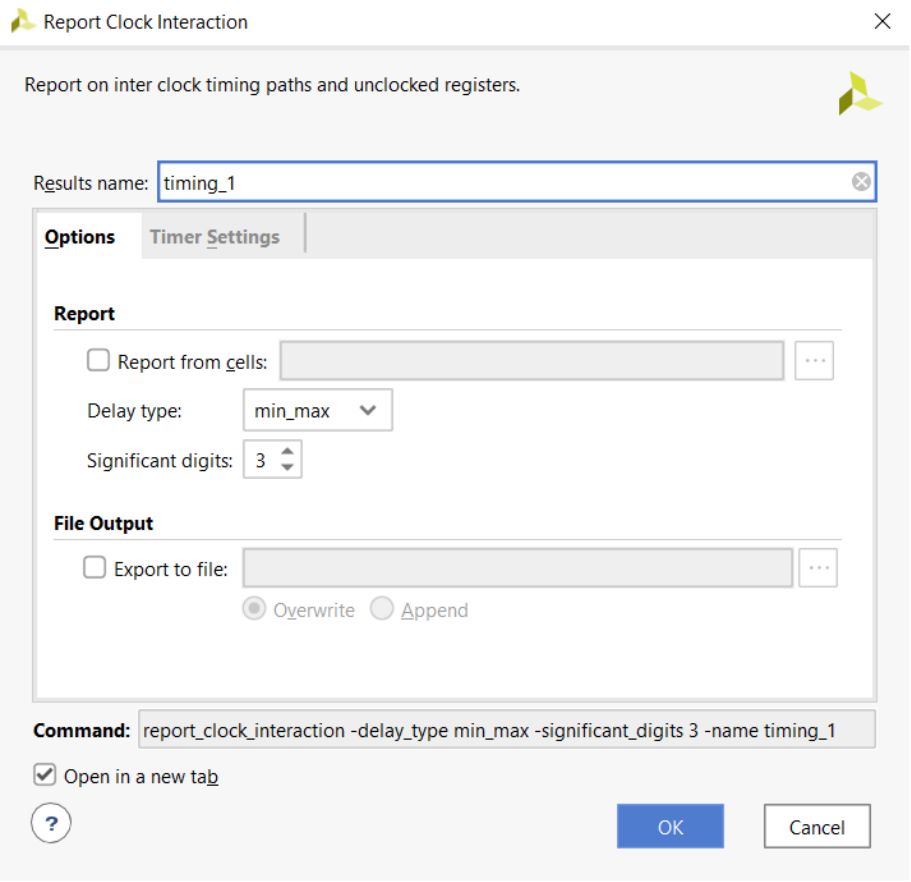
Use the Command field to display the Tcl command line equivalent of the various options specified in the Report Clock Interaction dialog box.

Open in a New Tab Check Box

Use the Open in a New Tab check box to either: open the results in a new tab, or replace the last tab opened by the Results window.

Options Tab

Figure 113: **Report Clock Interaction: Options Tab**



The Options tab of the Report Clock Interaction dialog box contains the following:

- Report from Cells Field
- Delay Type Field
- Significant Digits Field
- File Output Section

Report from Cells Field

Enable to limit the timing reporting on the particular cell(s) of the design. Only paths with the datapath section that start, end, cross, or are fully contained inside the cell(s) are reported.

Equivalent Tcl option: `-cells`

Delay Type Field

Use the Delay Type field to set the type of analysis to be run.

- For synthesized designs, only max delay analysis (setup/recovery) is performed by default.
- For implemented designs, both min delay and max delay analysis (setup/hold, recover/removal) are performed by default.

To run min delay analysis only (hold and removal), select delay type `min`.

Equivalent Tcl option: `-delay_type`

Significant Digits Field

Use the Significant Digits field to specify the number of significant digits in the reported values. The default is three.

Equivalent Tcl option: `-significant_digits`

File Output Section

The File Output section includes:

- **Write Results to File**: Use the Write Results to File field to write the result to a specified file. In the Vivado IDE, the report is displayed in the Clock Interaction window.

  Equivalent Tcl option: `-file`

- **Overwrite/Append**: Select the Overwrite/Append option buttons to determine whether, when the report is written to a file: (1) the specified file is overwritten, or (2) new information is appended to an existing report.

  Equivalent Tcl option: `-append`

Timer Settings Tab

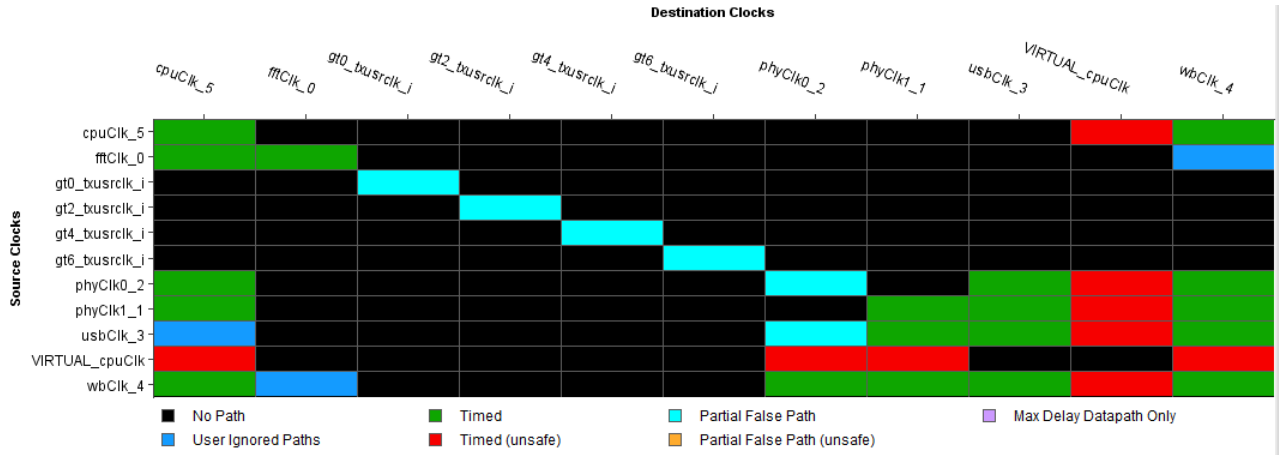
For details on this tab, see Timer Settings Tab .

Details of the Clock Interaction Report

The Clock Interaction report analyzes timing paths that cross from one clock domain (the source clock) into another clock domain (the destination clock). The Clock Interaction report helps to identify cases in which there may be data loss or metastability issues.

After you run the Report Clock Interaction command, the results open in the Clock Interaction window. As shown in the following figure, the Clock Interaction Report displays as a matrix of clock domains with the source clocks in the vertical axis and the destination clocks in the horizontal axis.

Figure 114: **Report Clock Interaction**



Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Clock Pair Classification	Inter-Clock Constraints
26	phyClk1_1	cpuClk_5	rise - rise	1.901	0.000	0	61	7.000	rise - rise	1.040	0.000	Clean	Timed
31	fftClk_0	cpuClk_5	rise - rise	1.063	0.000	0	61	7.000	rise - rise	1.488	0.000	Clean	Timed
34	VIRTUAL_c...	cpuClk_5	rise - rise	-1.372	-196.991	532	834		rise - rise	1.089	0.000	Virtual Clock	Timed (unsafe)
13	wbClk_4	fftClk_0				0	1635					Ignored	Asynchronous Groups
33	fftClk_0	fftClk_0	rise - rise	0.673	0.000	0	8320	7.000	rise - rise	0.046	0.000	Clean	Timed
1	gt0_txusrclk_i	gt0_txusrclk_i	rise - rise	9.923	0.000	0	222	12.800	rise - rise	0.135	0.000	Clean	Partial False Path
2	gt2_txusrclk_i	gt2_txusrclk_i	rise - rise	9.734	0.000	0	222	12.800	rise - rise	0.095	0.000	Clean	Partial False Path
3	gt4_txusrclk_i	gt4_txusrclk_i	rise - rise	8.856	0.000	0	222	12.800	rise - rise	0.123	0.000	Clean	Partial False Path

Matrix Color Coding

The tiles of the matrix are color coded. The colors of the matrix are determined by the background color of the graphical editors as defined under **Tools → Settings → Colors → Clock Interaction Chart** or by selecting the gear on the Clock Interactions tab. For more information, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893). To hide the legend, click the **?** button on the toolbar on the left of the matrix.

- **No Path (black):** There are no timing paths that cross from the source clock to the destination clock. In this case, there is no clock interaction and nothing to report.
- **Timed (green):** The source clock and destination clock have a synchronous relationship, and are safely timed together. This state is determined by the timing engine when the two clocks have a common primary clock and a simple period ratio.
- **User Ignored Paths (dark blue):** User-defined false path or clock group constraints cover all paths crossing from the source clock to the destination clock. When the interaction report is run for hold analysis only (`-delay_type min`) and the source clock and destination clock are covered with a `set_max_delay -datapath_only` constraint, the Clock Pair Classification is reported as *ignored* (the GUI category is User Ignored Paths) and the Inter-Clock Constraints are reported as *Auto Generated False Path* due to the implicit false path derived from the max delay datapath only.

- **Partial False Path (light blue):** User-defined false path constraints cover some of the timing paths crossing from the source clock to the destination clock where the source clock and destination clock have a synchronous relationship.
- **Timed (Unsafe) (red):** The source clock and destination clock have an asynchronous relationship. In this case, there is no common primary clock or no expandable period. For more information on asynchronous and unexpandable clocks, see this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903).
- **Partial False Path (Unsafe) (orange):** This category is identical to Timed (Unsafe), except that at least one path from the source clock to the destination clock is ignored due to a false path exception.
- **Max Delay Datapath Only (gray):** A `set_max_delay -datapath_only` constraint covers all paths crossing from the source clock to the destination clock

> **IMPORTANT!** *The color of a cell in the matrix reflects the state of the constraints between clock domains, not the state of the timing paths' worst slack between the domains. A green cell does not indicate that the timing is met, only that all timing paths that cross clock domains are properly timed, and that their clocks have a known phase relationship.*

Clock Pair Classification

The Clock Pair Classification column provides information about the missing common primary clock, missing common node, missing common phase, and missing common period between two clocks, as well as the presence of a virtual clock.

The possible values, from the highest to the lowest priority, are listed below. As soon as a condition is detected, the report command does not perform the remaining checks.

- **Ignored:** When the clock pair is entirely covered by a clock group, a false path, or a max delay datapath only, the analysis is ignored.

  Note: When a clock pair is covered by a max delay datapath only, the Inter-Clock Constraints are reported as Max Delay Datapath Only during setup analysis and as Auto Generated False Path during hold analysis (`-delay_type min`).

- **Virtual Clock:** At least one of the clocks is virtual, and common primary clock or common node checks do not apply.
- **No Common Clock:** The two clocks do not have a common primary clock.
- **No Common Period:** The periods of the two clocks are not expandable.
- **Partial Common Node:** The two clocks appear synchronous, but a subset of the crossing paths does not have a common node and cannot be safely timed.
- **No Common Node:** The two clocks appear synchronous, but the crossing paths do not have a common node.

- **No Common Phase:** The two clocks do not have a known phase relationship.
- **Clean:** None of the above conditions applies.

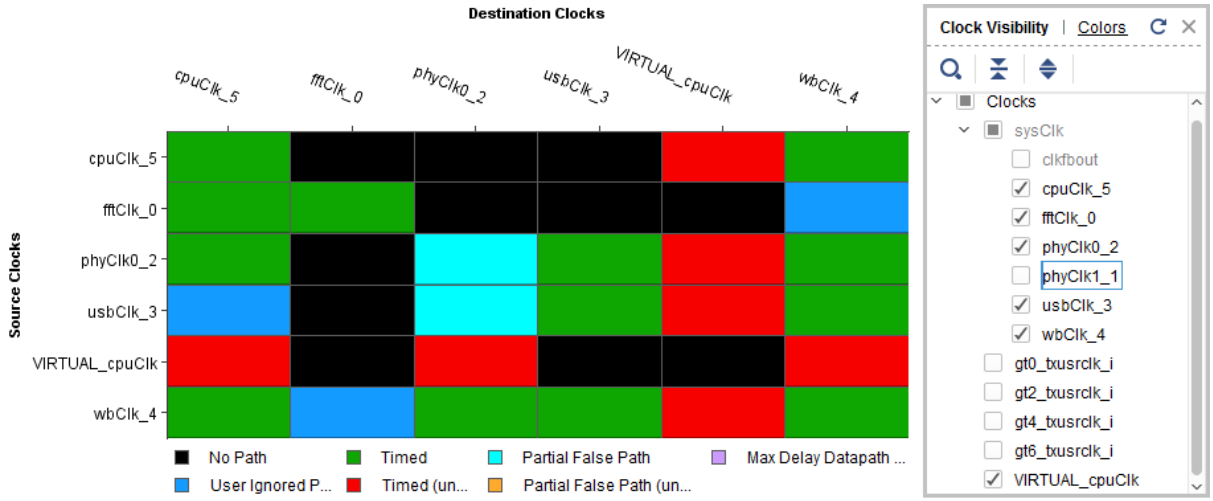
Filtering the Clocks

To filter the source clocks displayed in the Clock Interaction report:

1. Click on the settings button to display the Clock Visibility.
2. Select the source clocks to display. The list of destination clocks that are displayed in the table is automatically derived from the selected source clocks.

The Clock Visibility filter reduces the matrix complexity by limiting the number of clocks, but does not reduce the number of clock interactions reported in the table below the matrix. You can also show and hide the clocks that do not directly time a logical path in the design by clicking the Hide Unused Clocks button in the toolbar. Because these clocks do not contribute to WNS/TNS/WHS/THS computation, they are hidden by default.

Figure 115: **Clock Interaction View Layers**



Clock Pairs Slack Table

The table below the matrix provides a comprehensive overview of the timing slack for setup/recovery and/or for hold/removal for source/destination clock pair. It also shows useful information about path requirement of the worst paths, common primary clock and constraints status. See Details of the Clock Interaction Report. This provides details not displayed in the matrix above.

Sorting the Data

To sort the data in the table in increasing or decreasing values, single click multiple times on a column header.

Selecting Cells and Rows

Selecting a cell in the matrix cross-selects a specific row of the table below.

Selecting a row from the table highlights a cell in the matrix above.

Table Columns

The table columns are:

- **ID:** A numeric ID for the source/destination clock pair being displayed.
- **Source Clock:** The clock domain from which the path originates.
- **Destination Clock:** The clock domain within which the path terminates.
- **Edges (WNS):** The clock edges used to calculate the worst negative slack for max delay analysis (setup/recovery).
- **WNS (Worst Negative Slack):** The worst slack calculated for various paths crossing the specified clock domains. A negative slack indicates a problem in which the path violates a required setup (or recovery) time.
- **Total Negative Slack:** The sum of the worst slack violation for all the endpoints that belong to paths crossing the specified clock domains.
- **Failing Endpoints:** The number of endpoints in the crossing paths that fail to meet timing. The sum of the violations corresponds to TNS.
- **Total Endpoints (TNS):** The total number of endpoints in the crossing paths.
- **Path Req (WNS):** The timing path requirement corresponding to the path reported in the WNS column. There can be several path requirements between any clock pairs if both rising and falling edges are active for at least one of the two clocks, or some timing exceptions have been applied on paths between the two clocks. The value reported in this column is not always the most challenging requirement.

  For more information, see Path Requirement.
- **Clock Pair Classification:** Provides information about the common node and common period between the clock pair. From highest to lowest precedence: Ignored, Virtual Clock, No Common Clock, No Common Period, Partial Common Node, No Common Node, No Common Phase, and Clean. See Clock Pair Classification.
- **Inter-Clock Constraints:** Shows the constraints summary for all paths between the source clock and destination clock. The possible values are listed in the Matrix Color Coding. Following are example definitions of these constraints:

```
set_clock_groups -async -group wbClk -group usbClk
set_false_path -from [get_clocks wbClk] -to [get_clocks cpuClk]
```

When the min delay analysis is also selected (hold/removal), the following columns also appear in the table:

- **Edges (WHS):** The clock edges used to calculate the worst hold slack.
- **WHS (Worst Hold Slack):** The worst slack calculated for various paths crossing the specified clock domains. A negative slack indicates a problem in which the path violates a required hold (or removal) time.
- **THS (Total negative Hold Slack)::** The sum of the worst slack violation for all the endpoints that belong to paths crossing the specified clock domains for min delay analysis (hold/removal).
- **Failing Endpoints (THS):** The number of endpoints in the crossing paths that fail to meet timing. The sum of the violations corresponds to THS.
- **Total Endpoints (THS):** The total number of endpoints in the crossing paths for min delay analysis (hold/removal).
- **Path Req (WHS):** The timing path requirement corresponding to the path reported in the WHS column. Like with WNS, there can be several possible path requirements for min delay analysis between two clocks, and the value reported in this column does not always correspond to the most challenging ones.

  For more information, see Chapter 4: Timing Analysis.

One or multiple clock pairs can be selected from the table. Report Timing between a selected source/destination clock pair can be run from the popup menu.

Exporting the Table

Run the Export to Spreadsheet command to export the table to an XLS file for use in a spreadsheet.

Report Pulse Width

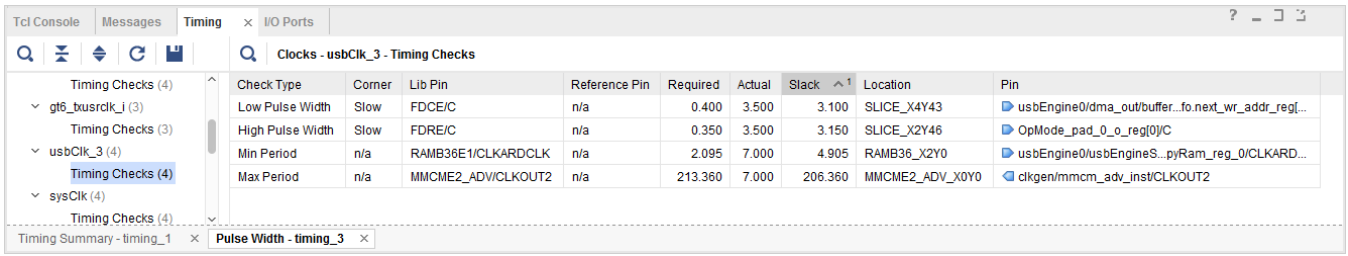
The Pulse Width Report (shown in the figure below) checks that the design meets min period, max period, high pulse time, and low pulse time requirements for each instance clock pin. It also checks that the maximum skew requirement is met between two clock pins of a same instance in the implemented design (for example, PCIe® clocks). The pulse width slack equations do not include jitter or clock uncertainty.

Equivalent Tcl command: `report_pulse_width`

When run from the Tcl Console, the pulse width report can be scoped to one or more hierarchical cells using the `-cells` option. When the report is scoped, only pins included inside the cell(s) are reported. This option is not available from the Report Pulse Width GUI.

***Note*:** AMD Integrated Software Environment (ISE) Design Suite implementation calls this check Component Switching Limits.

Figure 116: **Report Pulse Width**



Report Datasheet

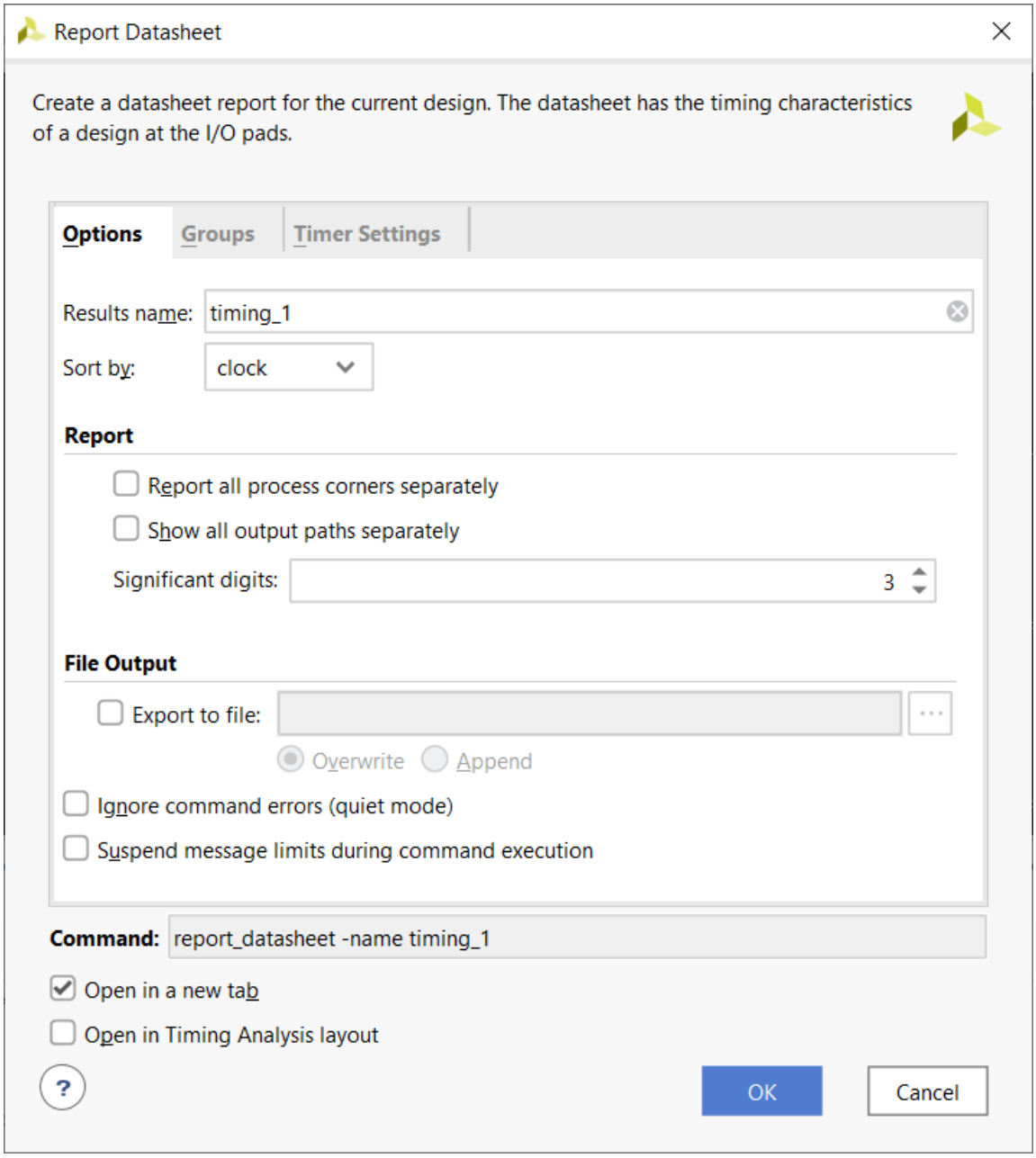
The Report Datasheet command reports the operating parameters of the FPGA for use in system-level integration.

Report Datasheet Dialog Box

In the AMD Vivado™ IDE, select **Reports → Timing → Report Datasheet** to open the Report Datasheet dialog box. See the following figure.

Report Datasheet Dialog Box: Options Tab

Figure 117: **Report Datasheet Dialog Box: Options Tab**



The Report Datasheet Dialog Box Options tab includes the following:

- Results name: Specifies the name for the returned results of the Report Datasheet command. The report opens in the Timing window of the Vivado IDE with the specified name.

  Equivalent Tcl option: `-name`

- Sort by: Sorts the results by port name or by clock name.

Equivalent Tcl option: `-sort_by`

- Report all process corners separately: Reports the data for all defined process corners in the current design.

  Equivalent Tcl option: `-show_all_corners`

- Significant digits: Specifies the number of significant digits in the reported values. The default is three.

  Equivalent Tcl option: `-significant_digits`

- Write results to file: Write the result to the specified file name. By default the report is written to the Timing window in the Vivado IDE.

  Equivalent Tcl option: `-file`

- Overwrite / Append: When the report is written to a file, determines whether the specified file is overwritten, or new information is appended to an existing report.

  Equivalent Tcl option: `-append`

- Ignore command errors: Executes the command quietly, ignoring any command line errors and returning no messages. Returns `TCL_OK` regardless of any errors encountered during execution.

  Equivalent Tcl option: `-quiet`

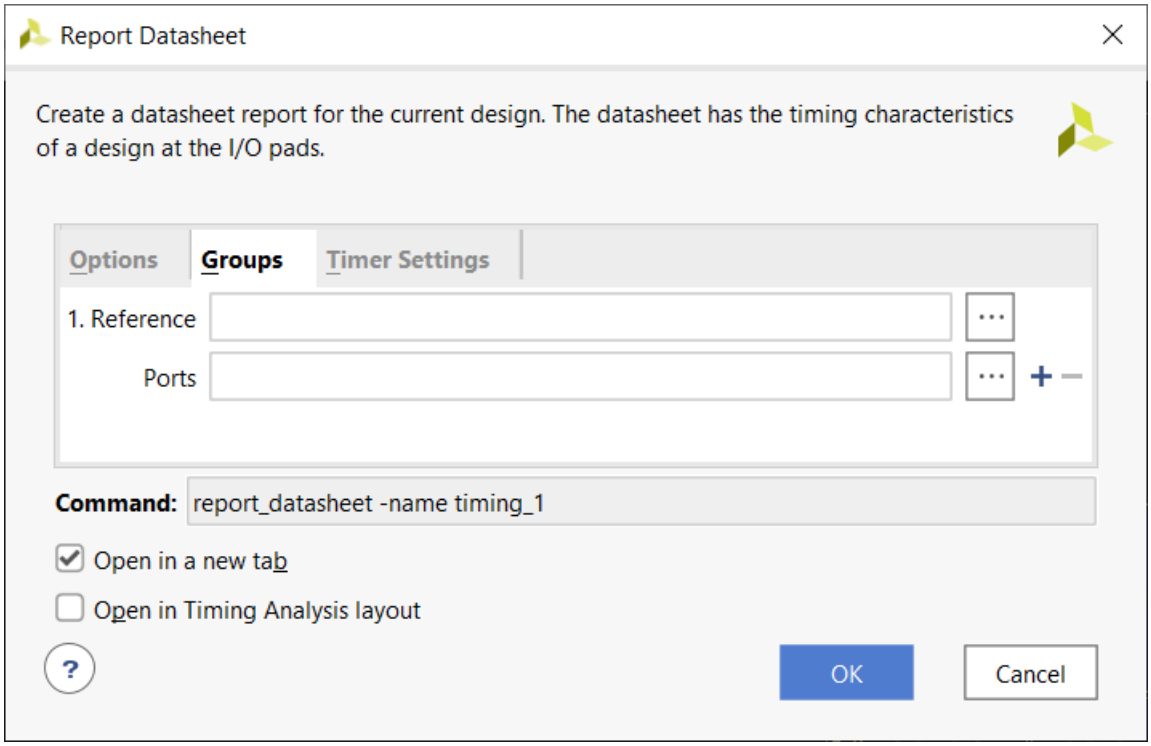
- Suspend message limits: Temporarily overrides any message limits. Returns all messages from this command.

  Equivalent Tcl option: `-verbose`

- Command: Displays the Tcl command line equivalent of the various options specified in the Report Datasheet dialog box.
- Open in a new tab: Opens the results in a new tab, or replaces the last tab opened by the Results window.
- Open in Timing Analysis layout: Resets the current view layout to the Timing Analysis view layout.

Report Datasheet Dialog Box: Groups Tab

Figure 118: **Report Datasheet Dialog Box: Groups Tab**



The Report Datasheet dialog box Groups tab allows you to define your own custom group of ports for analysis by specifying the reference port and additional ports to report. When Groups are not specified, the timer automatically finds the group of output ports based on the launching clock, and reports skew based on that clock.

The Report Datasheet dialog box Groups tab includes:

- Reference: Specifies the reference port for skew calculation. In most cases, this will be a clock port of a source synchronous output interface.

  Equivalent Tcl option: `-group`
- Ports: Defines additional ports to report.
  - Notice the **+** and **-** (plus and minus) buttons to the right of the Ports field.
  - The **+** (plus) button specifies multiple groups, each with its own reference clock port allowing you to define a new group of ports, including a new reference port.
  - The **-** (minus) button removes additional groups of ports as needed.

Report Datasheet Dialog Box: Timer Settings Tab

For details on this tab, see Timer Settings Tab .

Details of the Datasheet Report

General Information

This section provides details of the design and AMD device, and the tool environment at the time of the report.

- Design Name: The name of the design
- Part: The target AMD part and speed file information
- Version: The version of the Vivado tools used when the report was generated
- Date: The date and timestamp of the report
- Command: The command line used to generate the report

Input Ports Setup/Hold

The report displays worst-case setup and hold requirements for every input port with regard to the reference clock. The internal clock used to capture the input data is also reported.

Max/Min Delays for Output Ports

Shows worst-case maximum and minimum delays for every output port with regard to the reference clock. The internal clock used to launch the output data for is also reported.

Setup Between Clocks

For every clock pair, the worst-case setup requirements are reported for all clock edge combinations.

Setup/Hold for Input Buses

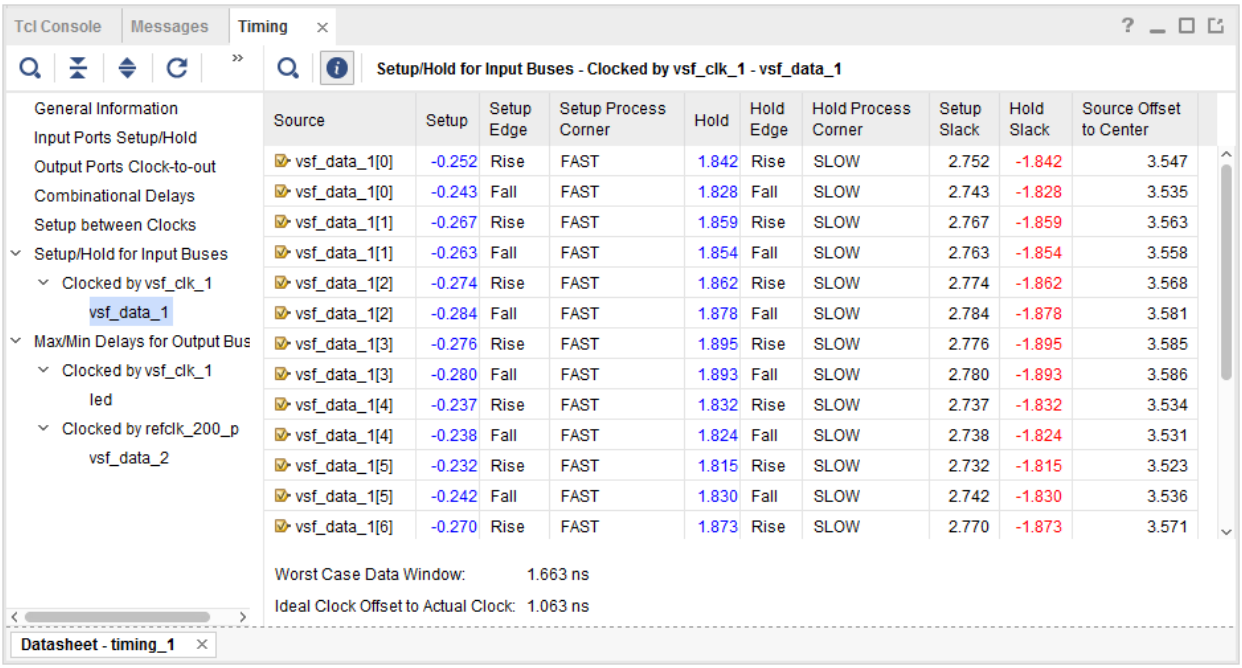
Input buses are automatically inferred and their worst-case setup and hold requirements are displayed. Worst case data window for the entire bus is the sum of the largest setup and hold values. If the input ports are constrained, the slack is also reported.

An optimal tap point is reported for input clocks with IDELAY defined. The optimal tap point can be used to configure IDELAY for balanced setup and hold slack.

The source offset is the delta between two windows. The first window is defined by the setup and hold time of the input port with regard to the clock. The second window is derived from the input delay and the clock period. If the input clock is offset with this value, then it will be in the center of the window.

The following figure reports a design in which a DDR input bus, `vsf_data[0:9]`, has a worst case data window of 1.663 ns. The ideal clock offset is 1.063 ns.

Figure 119: **Setup and Hold Delays for Input Buses**



***Note*:** The optimal tap point can be specified by using the Tcl command:

```
set_property IDELAY_VALUE 13 [get_cells idelay_clk]
```

Max/Min Delays for Output Buses

Output buses are automatically inferred and their worst case maximum and minimum delays are displayed. The bus skew is also reported. For bus skew calculation, one bit is considered as the reference and the offset of every other bit is calculated with respect to this reference bit. The worst offset is the skew for the entire bus.

Max/Min Delays for Groups

For Source Synchronous Output Interfaces, the output skew is desired with regard to the forwarded clock. A custom group report can be generated by specifying the reference port as the forwarded clock port. This table looks similar to "max/min delays for output buses" except the reference port is used as the reference bit for calculating source offset and bus skew.

***Note*:** This section might be hidden if empty.

As an example, for a DDR output skew calculation, if multiple bits (for example, `rldiii_a[0-19]`, `rldiii_ba[0-3]`, `rldiii_ref_n`, `rldiii_we_n`) should be grouped together with regard to the forwarded clock port (`rldiii_ck_n[0]`), the following command can be used:

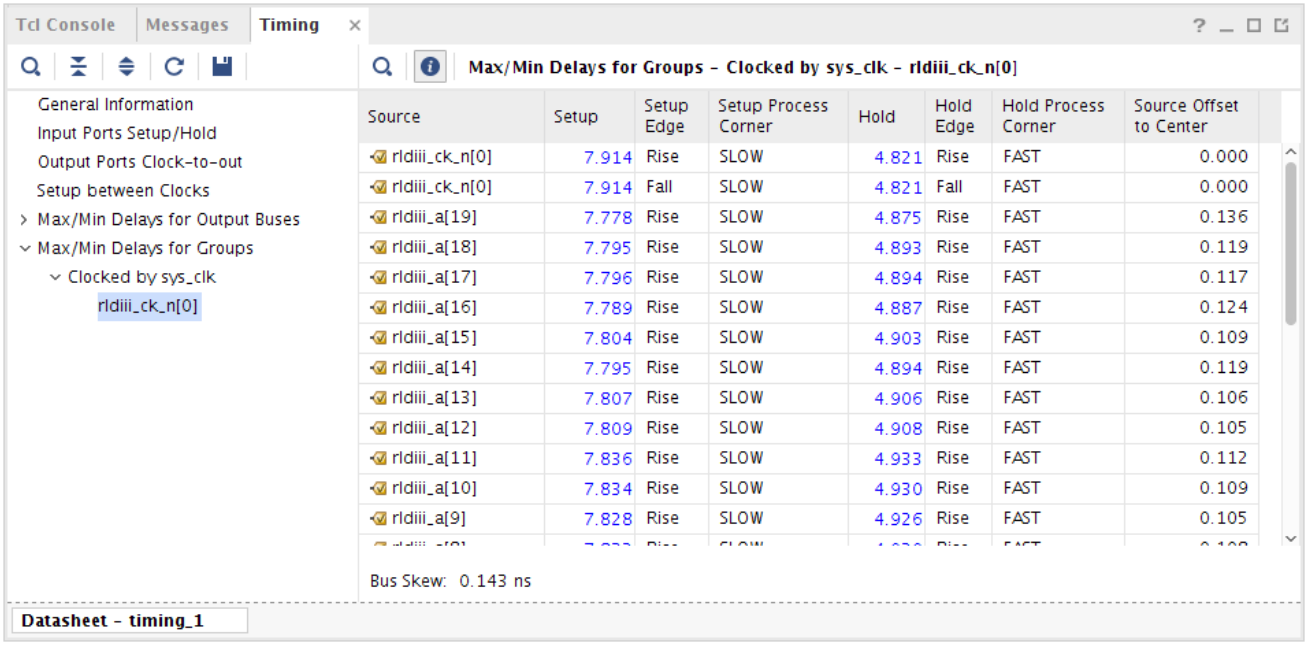
```
report_datasheet -group [get_ports {rldiii_ck_n[0] rldiii_a[*] rldiii_ba[*]
rldiii_ref_n rldiii_we_n}] -name timing_1
```

The first port in the group list is considered the reference pin.

For all these sections, the worst case data is calculated from multi-corner analysis. If `-show_all_corners` is used, the worst case data is reported for each corner separately.

The following figure shows the report data sheet for this example.

Figure 120: **Report Data Sheet Max/Min Delay Example**



Report Exceptions

You can use the Report Exceptions command anywhere in the flow after the synthesis. The Report Exception command reports the following information:

- All the timing exceptions that have been set in the design and that are affecting timing analysis
- All the timing exceptions that have been set in the design but that are being ignored as they are overridden by other timing exception

The timing exceptions analyzed by the Report Exception command are (in the order of precedence):

- clock groups
- false paths
- max/min delays
- multicycle paths

The Report Exception is a powerful command to help debugging issues related to timing exceptions. Some designs have timing constraints with complex timing exceptions. Because the timing exceptions have different priorities, it can quickly become difficult to understand which timing exceptions might be partially or fully ignored by other exception(s). The Report Exception reports timing exceptions that are partially overridden, as well as those that are totally overridden. It also provides a hint to the overriding constraint(s).

For more information about the `report_exceptions` command line options, refer to this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835). For more information about the timing exception priority order, refer to this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903).

The `report_exceptions` command has several modes of operation:

- Reporting the timing exceptions affecting the timing analysis
- Reporting the timing exceptions being ignored
- Reporting the timing exceptions coverage
- Reporting the invalid objects specified for the `-from`/`-through`/`-to` command line options
- Writing out the timing exceptions with only the valid objects
- Writing out the timing exceptions merged by the timing engine

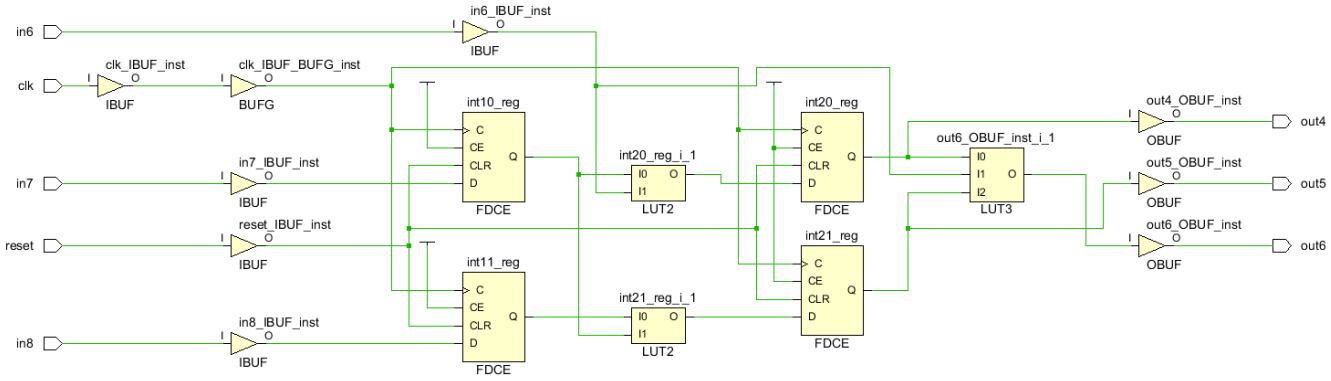
  Note: Even though Clock Groups are not strictly timing exceptions, they are covered by the command `report_exceptions` because they can override other timing exceptions.

  Note: Using the `report_exceptions` command with the `-from`/`-through`/`-to` options only report timing exceptions that have been defined with the same `-from`/`-through`/`-to` command line options. The specified patterns can be different but there must be at least one object (cell, net, pin, or port) matching inside each of the `-from`/`-through`/`-to` for it to be reported as an exception.

Example: Reporting the Timing Exceptions Affecting the Timing Analysis

This example describes how to take the design, shown in the following figure, through some timing exceptions. The design is fully constrained (`clk` and input/output delays defined relative to `clk`).

Figure 121: **Fully Constrained Design for Timing Exception Example**



The first mode of operation of the Report Exception command is `report_exceptions`.

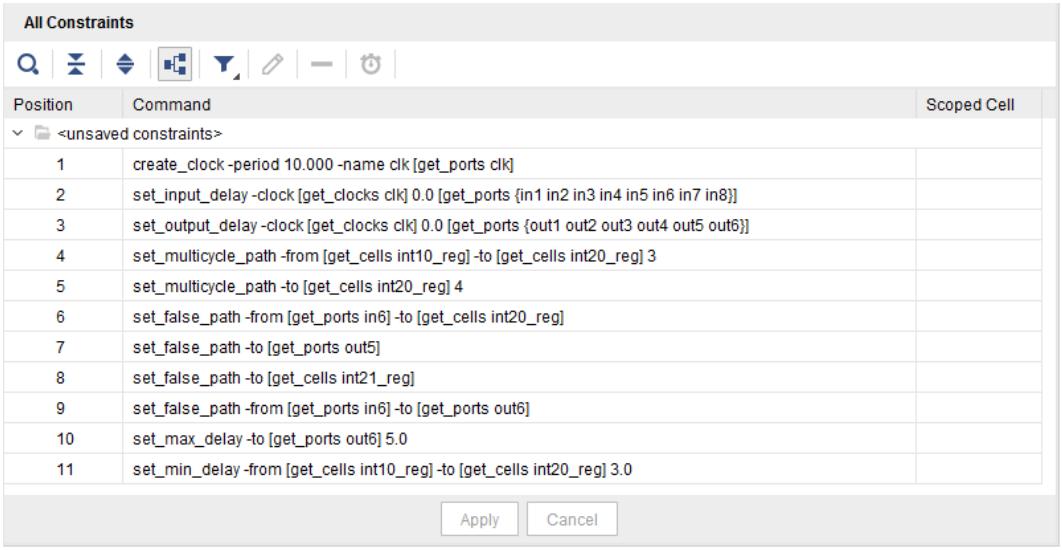
1. Select **Window → Timing Constraints**.
2. In the Timing Constraints window, add the following timing exceptions to the design:

```
set_multicycle_path 3 -from [get_cell int10_reg] -to [get_cell int20_reg]
```

```
id="ab439753">set_multicycle_path 4 -to [get_cell int20_reg]
set_false_path -from [get_ports in6] -to [get_cell int20_reg]
set_false_path -to [get_ports out5]
set_false_path -to [get_cell int21_reg]
set_false_path -from [get_ports in6] -to [get_ports out6]
set_max_delay 5 -to [get_ports out6]
set_min_delay 3 -from [get_cells int10_reg] -to [get_cell int20_reg]
```

The Timing Constraints window displays the timing constraints applied to the design, as shown in the following figure.

Figure 122: **Timing Constraints Window Displaying Timing Constraint Changes**



The actual Exception Report (`report_exceptions`) is shown in the following figure.

Figure 123: **Report Exception**

```
Position  From                 Through       To                   Setup          Hold           Status
--------  -------------------  ------------  -------------------  -------------  -------------  ----------------------------------------
4         [get_cells int10_reg]  *             [get_cells int20_reg]  cycles=3       -
5         *                    *             [get_cells int20_reg]  cycles=4       -              Partially overridden path by MCP 4 - FP 6
6         [get_ports in6]      *             [get_cells int20_reg]  false          false
7         *                    *             [get_ports out5]       false          false
8         *                    *             [get_cells int21_reg]  false          false
9         [get_ports in6]      *             [get_ports out6]       false          false
10        *                    *             [get_ports out6]       max=5          -              Partially overridden path by FP 9
11        [get_cells int10_reg]  *             [get_cells int20_reg]  -              min=3
```

The Exceptions Report contains the following information:

- The Position column indicates the constraint position number. This is the same position number reported by the Timing Constraint window (shown previously).
- The From/Through/To columns indicate the patterns or objects specified with `-*from/-*through/-*to` command line options (including all the `rise/fall` versions of those options). An asterisk is displayed when the associated option was not specified.
- The Setup/Hold columns indicate whether the constraint applies to setup check, hold check, or both. The naming conventions for the Setup/Hold columns are shown in the following table:

Table 11: **Setup/Hold Column Naming Conventions**

Short Name	Timing Exception
`cycles=`	`set_multicycle_path`
`false`	`set_false_path`
`max=`	`set_max_delay`

Table 11: **Setup/Hold Column Naming Conventions** *(cont'd)*

Short Name	Timing Exception
max_dpo=	set_max_delay -datapath_only
min=	set_min_delay
clock_group=	set_clock_group

- The Status column reports a message when a constraint is partially overridden by another timing exception. The naming conventions for the Status column are shown in the following table:

Table 12: **Status Column Naming Conventions**

Short	Timing Exception
MCP	multicycle path
FP	false path
MXD	max delay
MND	min delay
CG	clock group

Note: The clock group is only reported in the **Status** column of the `report_timing -ignored` command when a clock group constraint overrides another timing exception.

In this example, there are two messages regarding partially overridden constraints:

- The timing constraint position 5 (`set_multicycle_path 4 -to [get_cell int20_reg]` based on the Timing Constraints window) is partially overridden by the multicycle constraint position 4 (`set_multicycle_path 3 -from [get_cell int10_reg] -to [get_cell int20_reg]`) and by the false path constraint position 6 (`set_false_path -from [get_ports in6] -to [get_cell int20_reg]`).
- The timing constraint position 10 (`set_max_delay 5 -to [get_ports out6]`) is partially overridden by the false path position 9 (`set_false_path -from [get_ports in6] -to [get_ports out6]`).

Reporting Overridden Scoped Timing Exceptions

The second mode of operation of the Report Exception command is to report the list of scoped timing exceptions that are overridden. The report is generated using the `-scope_override` command line option.

```
report_exceptions -scope_override
```

Note: This mode is only available for the text report from the Tcl Console.

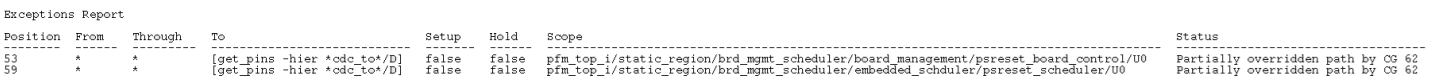
The command line option `-scope_override` reports the scoped timing exceptions that are partially or fully overridden by top-level constraints. It can, for example, flag IP constraints overridden by top-level user constraints. However, this option does not report the scoped constraints that are overridden by other scoped constraints (either from the same scope or from a different scope).

The report includes, for each overridden timing exception, the following information:

- Constraint position number
- Patterns for the `-from`/`-through`/`-to` command line options
- Type of constraint overriding the Setup and/or Hold
- Scope of the constraint
- A list of position numbers of top-level constraints that are partially or fully overriding the constraint

The following figure shows the Exceptions Report.

Figure 124: **Exceptions Report**



Exceptions Report

Position	From	Through	To	Setup	Hold	Scope	Status
53	*	*	[get_pins -hier *cdc_to*/D]	false	false	pfm_top_i/static_region/brd_mgmt_scheduler/board_management/psreset_board_control/U0	Partially overridden path by CS 62
59	*	*	[get_pins -hier *cdc_to*/D]	false	false	pfm_top_i/static_region/brd_mgmt_scheduler/embedded_schduler/psreset_scheduler/U0	Partially overridden path by CS 62

Reporting the Timing Exceptions Being Ignored

Another mode of operation of the Report Exception command is as follows:

```
report_exceptions -ignored
```

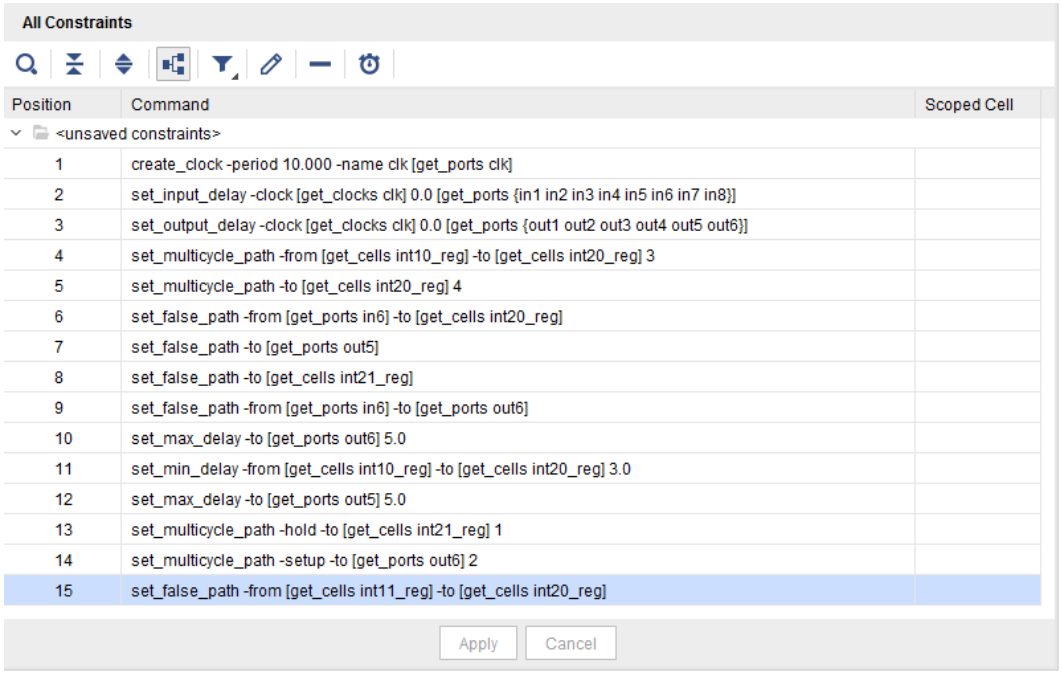
To illustrate, add the following timing exceptions on the top of the previous ones:

```
set_max_delay 5 -to [get_ports out5]
set_multicycle_path 1 -hold -to [get_cell int21_reg]
set_multicycle_path 2 -setup -to [get_ports out6]
set_false_path -from [get_cell int11_reg] -to [get_cell int20_reg]
```

All those exceptions are either already covered by a timing exception from the previous section (reporting the timing exceptions affecting the timing analysis) or target a non-existing path (there is no physical connection between the registers `int11_reg` and `int20_reg`).

After adding these four constraints, the Timing Constraints window looks like the following figure.

Figure 125: **Timing Constraints Window**



The Exceptions Report (`report_exceptions -ignored`) is as shown in the following figure:

Figure 126: **Exceptions Report**

```
Exceptions Report

Position  From                  Through       To                    Setup          Hold           Status
--------  --------------------  ------------  --------------------  -------------  -------------  ----------------------------------------
12        *                     *             [get_ports out5]      max=5          -              Totally overridden path by FP 7
13        *                     *             [get_cells int21_reg] -              cycles=1       Totally overridden path by FP 8
14        *                     *             [get_ports out6]      cycles=2       -              Totally overridden path by FP 9 - MXD 10
15        [get_cells int11_reg] *             [get_cells int20_reg] false          false          Non-existent path
```

***Note*:** The Status column provides some explanations why the timing exceptions are being ignored.

Reporting the Timing Exceptions Coverage

The Vivado tools can generate a detailed coverage of each valid timing exception applied to the design. All the timing exceptions are reported, including those that are fully overridden or that do not have a path between startpoints and endpoints.

The exceptions coverage report is generated using the `-coverage` command line option:

```
report_exceptions -coverage
```

The report includes, for each valid timing exception, the following information:

- Constraint position number.

- Number of objects selected by the `-from/-through/-to` command line options.
- The coverage, expressed as a percentage, between the number of pins reached by the timing exception compared to the number of pins specified by the `-from/-through/-to` command line options.

  ***Note*:** When cells objects are specified, Vivado tools expand the cells into valid pins objects. This cell-to-pin conversion tends to bring the coverage down because typically the timing exception only reaches a subset of pins.

The following figure shows the exceptions coverage report.

Figure 127: **Exceptions Coverage Report**

Exceptions Report

Position	Type	Setup	Hold	From	Through	To	Endpoints	From (%)	Through (%)	To (%)
4	Multicycle Path	cycles=3	-	1 cells		1 cells	1	100.00		33.33
5	Multicycle Path	cycles=4	-			1 cells	2			66.67
6	False Path	false	false	1 ports		1 cells	1	100.00		33.33
7	False Path	false	false			1 ports	1			100.00
8	False Path	false	false			1 cells	2			66.67
9	False Path	false	false	1 ports		1 ports	1	100.00		100.00
10	Max Delay	max=5	-			1 ports	1			100.00
11	Min Delay	-	min=3	1 cells		1 cells	1	20.00		25.00
12	Max Delay	max=5	-			1 ports	1			100.00
13	Multicycle Path	-	cycles=1			1 cells	2			66.67
14	Multicycle Path	cycles=2	-			1 ports	1			100.00
15	False Path	false	false	1 cells		1 cells	0	0.00		0.00

Warning: the percentages reported indicate the number of pins covered by the exception relative to the number of pins specified implicitly or explicitly.

When a timing exception does not have a path between the startpoints and endpoints, the coverage report shows 0.0. In the above example, timing exception position 15 does not have a timing path. This matches the result from `report_exceptions -ignored` where constraint position 15 is reported as Non-Existent Path.

A coverage reports can assist in writing effective timing exceptions. The following figure shows another example of a coverage report for the following `set_multicycle_path` constraint:

```
set_multicycle_path -setup 2 -from [all_registers] -to [get_cells
cpuEngine/or1200_cpu/or1200_ctrl/ex_insn_reg[*]]
```

Figure 128: **Multicycle Path Coverage**

Exceptions Report

Position	Type	Setup	Hold	From	Through	To	Endpoints	From (%)	Through (%)	To (%)
54	Multicycle Path	cycles=2	-	15901 cells		21 cells	63	0.95		100.00

In the example shown in the previous figure, the coverage for the `-from` option is only 0.95% for 15901 cells objects returned by `all_registers`. The efficiency of the constraint can be improved by refining the list of objects specified for the `-from` option to only those objects that have a path to the cells `cpuEngine/or1200_cpu/or1200_ctrl/ex_insn_reg[*]`.

Reporting the Ignored Objects

The Report Exception command can generate a list of invalid startpoints and endpoints for each of the timing exception constraints. Invalid startpoints and endpoints are ignored by the Vivado tool because timing paths can neither originate from those startpoints nor end on those endpoints. The ignored pins are reported by `report_exceptions -ignored_objects`.

***Note*:** Invalid startpoints and endpoints with a Max Delay or Min Delay constraint are not ignored but result in path segmentation.

***Note*:** Startpoint or endpoint pins tied to POWER or GROUND are reported in the list of ignored objects.

To illustrate, set the following timing constraints on the small example design:

```
create_clock -period 10.000 -name clk [get_ports clk]
set_false_path -from [get_cells int10_reg] -to [get_cells int20_reg]
set_false_path -from [get_pins int11_reg/*] -to [get_pins int21_reg/*]
```

***Note*:** When the second False Path constraint is entered, the Vivado tool generates Warning Constraints 18-402 because some startpoints and endpoints are invalid.

WARNING: [Constraints 18-402] set_false_path: 'int11_reg/CE' is not a valid startpoint.

Resolution: A valid start point is a main or generated clock pin or port, a clock pin of a sequential cell, or a primary input or inout port. Please validate that all the objects returned by your query belong to this list.

- The first `set_false_path` constraint uses the `get_cells` command. The Vivado tool converts the cell(s) from `get_cells` into pins using only valid startpoint or endpoint pins. This ensures that the constraint refers only to valid objects.
- The second `set_false_path` constraint uses the `get_pins` command and forces all the register pins for `-from` and `-to`. This results in several invalid pins for both `-from` and `-to`.

The following figure shows the report from `report_exceptions -ignored_objects`.

Figure 129: **Ignored Objects**

```
Exceptions Report

Position  Exception   Ignored Startpoints  Ignored Endpoints
--------  ----------  -------------------  -----------------
3         False Path  int11_reg/Q          int21_reg/Q
3         False Path  int11_reg/CE         int21_reg/C
3         False Path  int11_reg/CLR        int21_reg/CE
3         False Path  int11_reg/D
```

Exporting the Valid Exceptions

The Report Exception command can export the list of timing exceptions. Only the constraints that cover at least one path are exported. Only valid startpoints and endpoints pins are exported while the patterns used to specify the timing exceptions are expanded inside the Vivado Design Suite Timer memory. This report can be used in conjunction with the coverage report to help refine the patterns and collections of objects used to define the timing exceptions.

***Note*:** Timing constraints `set_clock_group` and `set_bus_skew` are not exported.

The following figure illustrates `report_exceptions -write_valid_exceptions` on the two False Path constraints explained in the section Reporting the Ignored Objects.

Figure 130: **Valid Exceptions**

```
# position: 2
set_false_path \
    -from [get_pins {int10_reg/C}] \
    -to [get_pins {int20_reg/D}]

# position: 3
set_false_path \
    -from [get_pins {int11_reg/C}] \
    -to [get_pins {int21_reg/D}]
```

Exporting the Merged Exceptions

The Report Exception command can export the list of timing exceptions as seen by the STA engine. The Vivado timing engine internally merges the timing exceptions to reduce memory and runtime. If the number of merged timing exceptions is different from the number of timing exceptions specified for the design, then this could mean that the timing exceptions are not optimally defined. The merged timing exceptions are reported with `report_exceptions -write_merged_exceptions`.

***Note*:** Timing constraints `set_clock_group` and `set_bus_skew` are not exported.

***Note*:** Invalid startpoints and endpoints are not filtered out when the merged timing exceptions are exported.

The following figure illustrates the `report_exceptions -write_merged_exceptions` on the two False Path constraints explained in the section Reporting the Ignored Objects. The second False Path includes all the registers pins because the pattern for `-from`/`-to` for the `get_pins` command is `int21_reg/*`.

Figure 131: **Merged Exceptions**

```
set_false_path \
    -from [get_pins {int10_reg/C}] \
    -to [list [get_pins {int20_reg/CE}] \
          [get_pins {int20_reg/CLR}] \
          [get_pins {int20_reg/D}]]

set_false_path \
    -from [list [get_pins {int11_reg/C}] \
          [get_pins {int11_reg/CE}] \
          [get_pins {int11_reg/CLR}] \
          [get_pins {int11_reg/D}] \
          [get_pins {int11_reg/Q}]] \
    -to [list [get_pins {int21_reg/C}] \
          [get_pins {int21_reg/CE}] \
          [get_pins {int21_reg/CLR}] \
          [get_pins {int21_reg/D}] \
          [get_pins {int21_reg/Q}]]
```

Report Exceptions in the Vivado IDE

Report Exceptions Dialog Box

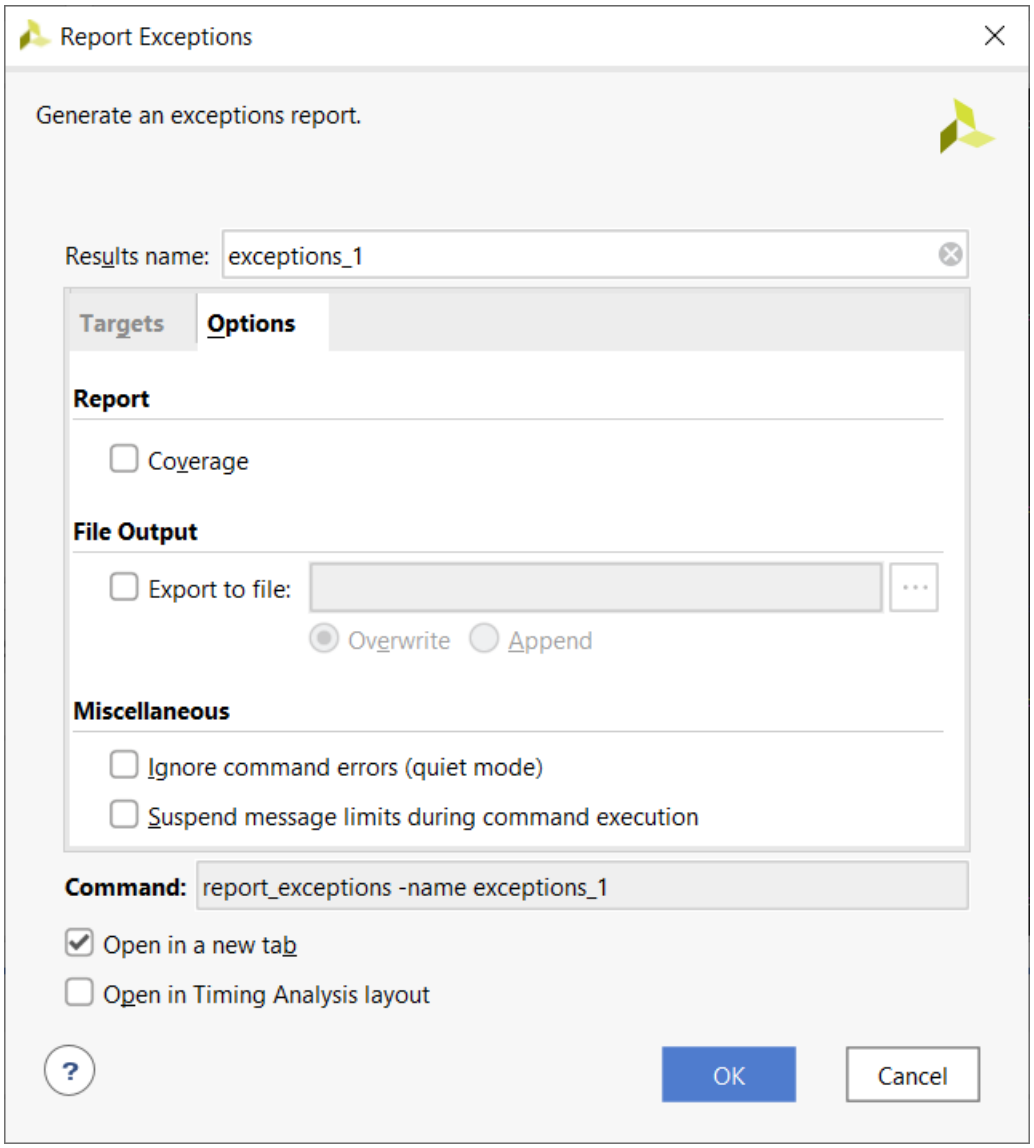
In the AMD Vivado™ IDE, select **Reports → Timing → Report Exceptions** to open the Report Exceptions dialog box.

The report generated from Report Exceptions GUI consolidates multiple tables in a single run. Normally, several runs of `report_exceptions` with different command line options would be required to generate such a report. As a result, the runtime of Report Exceptions through the GUI can be higher than running `report_exceptions` through the Tcl Console.

Report Exceptions Dialog Box: Options Tab

The Options tab in the Report Exceptions dialog box is shown in the figure below.

Figure 132: **Report Exceptions Dialog Box: Options Tab**



The Report Exceptions Dialog Box Options tab includes the following sections:

Report Section

Coverage: Reports the timing exceptions coverage through additional columns inside the detailed tables.

Note: This option can result in a significant increase in the runtime.

Equivalent Tcl option: `-coverage`

File Output Section

- Write results to file: Writes the result to the specified file. By default the report is written to the Timing window in the Vivado IDE.

  Equivalent Tcl option: `-file`

- Overwrite or Append: When the report is written to a file, determines whether the specified file is overwritten or new information is appended to an existing report.

  Equivalent Tcl option: `-append`

Miscellaneous Section

- Ignore command errors: Executes the command quietly, ignores any command line errors, and does not return messages. The command also returns `TCL_OK` regardless of any errors encountered during execution.

  Equivalent Tcl option: `-quiet`

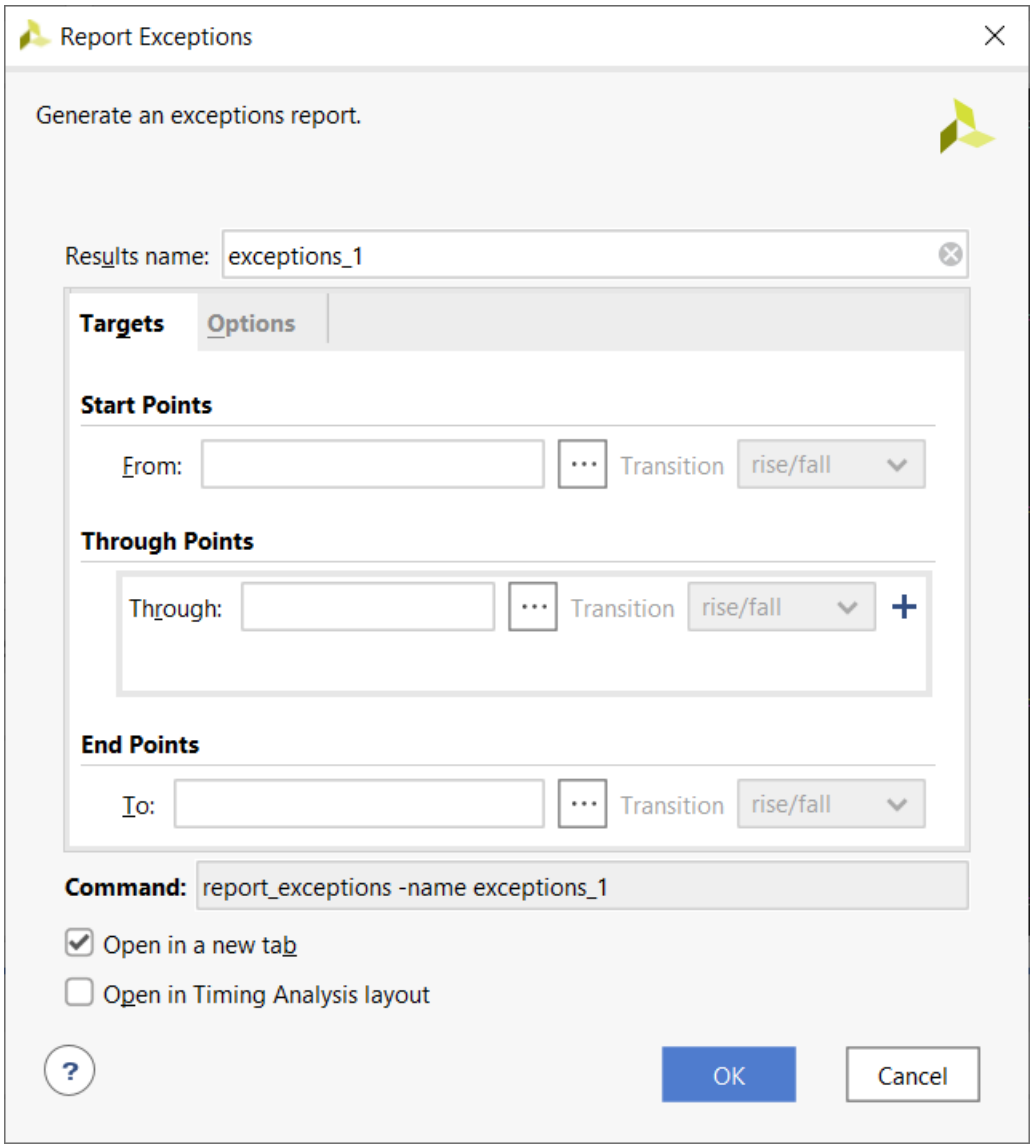
- Suspend message limits during command execution: Temporarily overrides any message limits and returns all messages.

  Equivalent Tcl option: `-verbose`

Report Exceptions Dialog Box: Targets Tab

The Targets tab in the Report Exceptions dialog box is shown in the figure below.

Figure 133: **Report Exceptions Dialog Box: Targets Tab**



Report Exceptions provides several filtering options that can be used to report a particular timing exception or group of timing exceptions:

- Start Points: See Targets Tab.
- Through Points: See Targets Tab.
- End Points: See Targets Tab.

When the filtering options are used, only the timing exceptions that are strictly defined according to those options are reported.

Details of the Exceptions Report

The Exceptions Report contains the following sections:

General Information Section

The General Information section of the Exceptions Report provides information about the following:

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

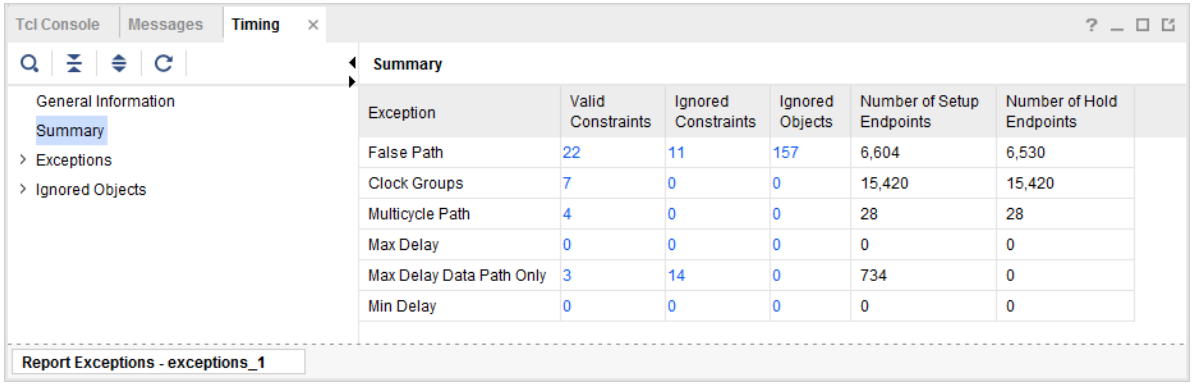
Summary Section

This section provides a summary of all the timing exceptions and clock group constraints. For each constraint type, the number of valid constraints, ignored constraints, number of ignored objects, and number of covered setup and hold endpoints are reported. This table provides more information than the summary table available when `report_exceptions` is run from the command line (`report_exceptions -summary`).

To get the detailed information for each exception type, the summary table provides hyperlinks to the Exceptions or Ignored Objects sections. The Valid Constraints and Ignored Constraints link to the same Exceptions detailed table.

Note: An exception is considered ignored when there is no physical path that connects the `-from`, `-through`, or `-to` or when the constraint is totally overridden by another constraint.

Figure 134: **Report Exceptions: Summary Section**

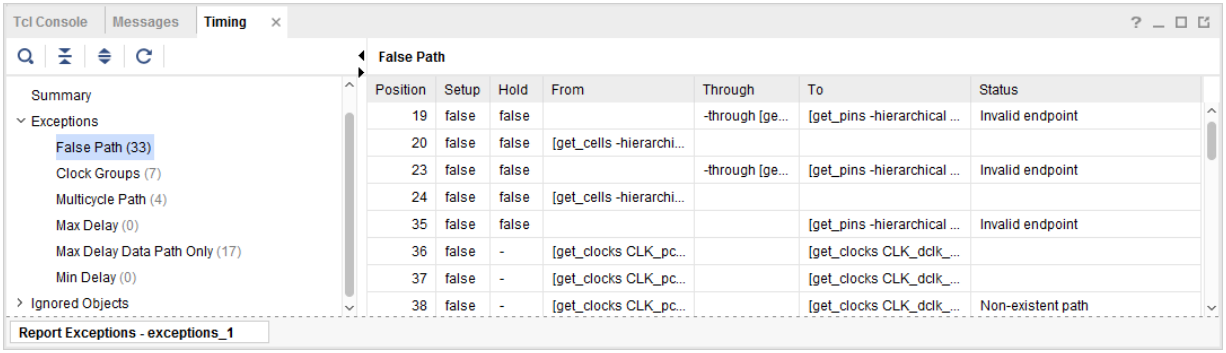


Exceptions Section

This section provides access to the detailed table of each timing exception. A category is available for each type of timing exceptions and the categories are hyperlinked from the Summary table. The format of the detailed tables depends on whether the Coverage option has been selected in the GUI or not.

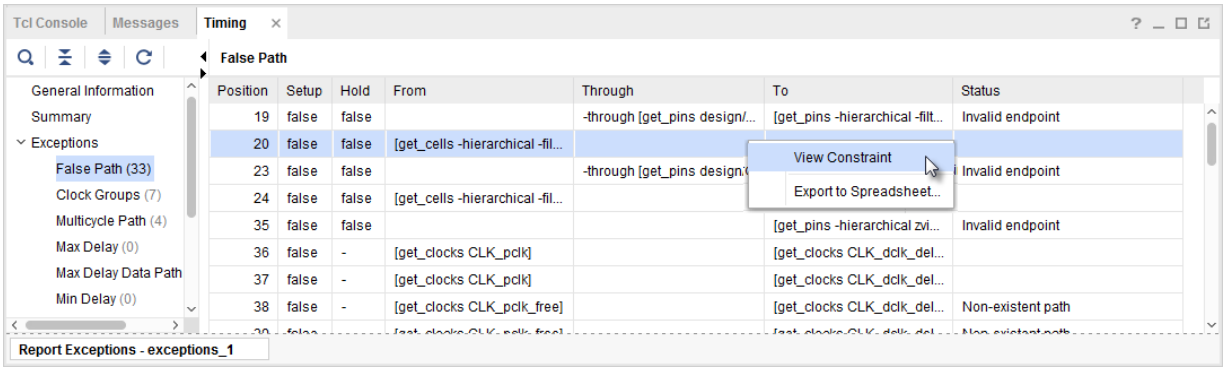
Below is an example of a detailed table without the Coverage option selected.

Figure 135: **Report Exceptions: Detailed Table without Coverage**



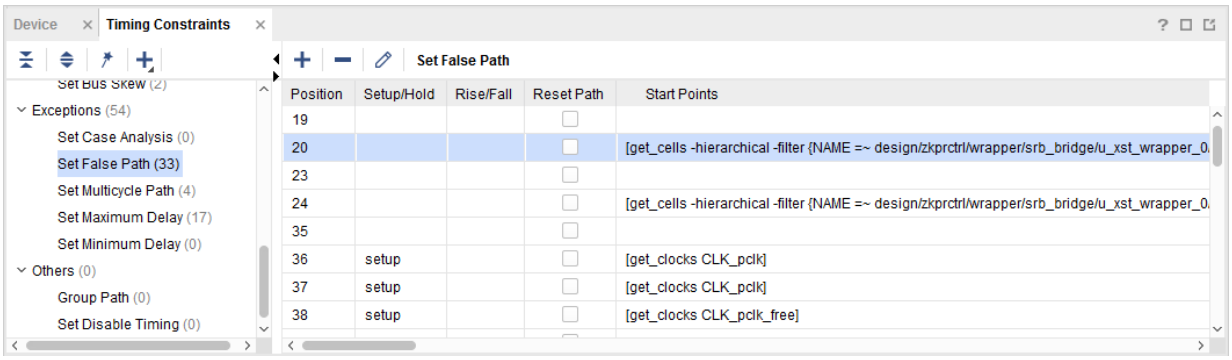
The table column Position represents the timing constraint position number that matches the position number reported by the Timing Constraints Editor (TCE). You can double-click on a row to be redirected to the selected constraint inside TCE. An alternative is to right-click on the row and select View Constraint from the pop-up menu.

Figure 136: **Report Exceptions Context Menu**



The image below shows the selected constraint inside the Timing Constraint Editor (TCE).

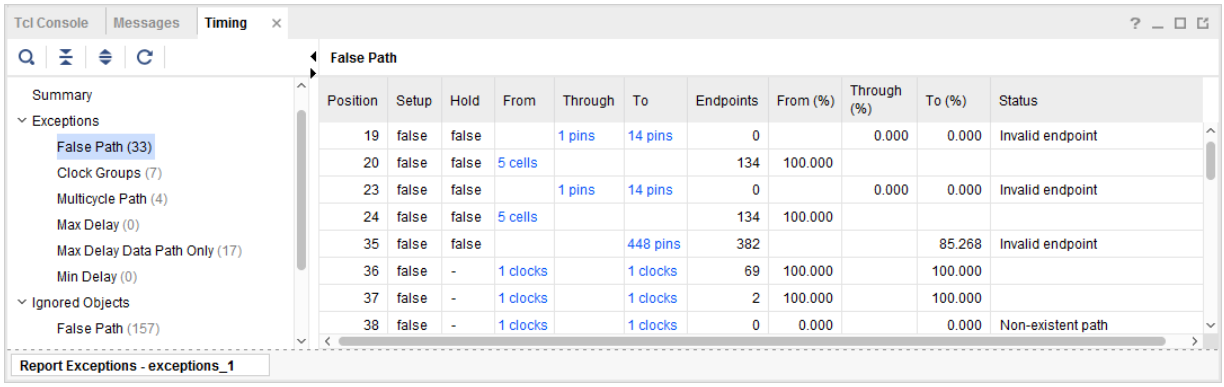
Figure 137: **Timing Exception inside Timing Constraint Editor**



The columns From, Through, and To report the original patterns used to define the timing exception. You can also refer to the constraint position number inside the TCE to review the same patterns.

The following figure shows an example of a detailed table with the Coverage option selected inside the Report Exception GUI.

Figure 138: **Report Exceptions: Detailed Table with Coverage**



The table column Position represents the timing constraint position number as described above.

When the Coverage option has been selected, the table columns From, To, and Through include hyperlinks to the design objects targeted by the timing constraints. The objects can be cells, nets, pins, ports, or clocks. It is possible to click on the blue hyperlinks to select the objects. After the objects have been selected, the schematic can be opened with the **F4** key. In addition, the coverage information adds the table columns From (%), To (%), and Through (%) that indicate the coverage percentage.

The table column Status reports the status of the constraint such as Invalid endpoint, Partially overridden path, Non-existent path, or Totally overridden. The same status is reported when `report_exception` is run on the command line:

- Non-existent path: The exception is considered invalid (does not impact timing analysis).

- Totally overridden: The exception is considered invalid (does not impact timing analysis).

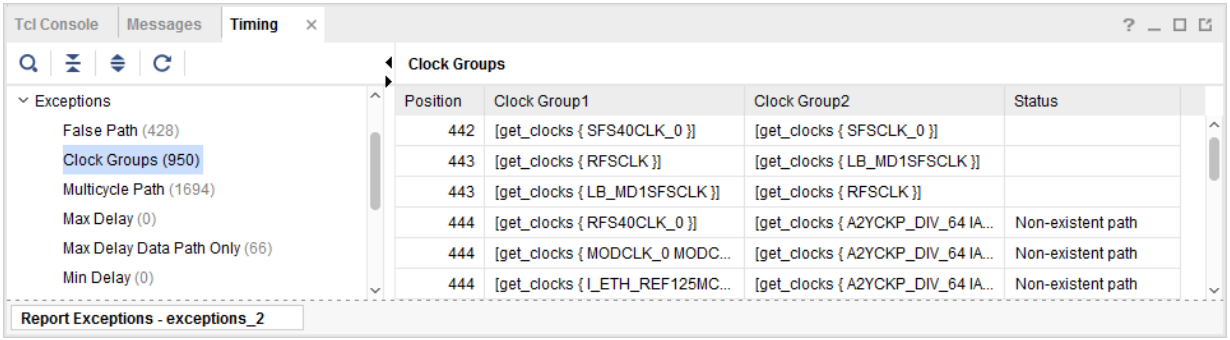
  Note: The coverages are calculated in the following order: From, Through, and To. The coverage computed for a level depends on the previous level. When the calculated coverage is 0% for a given level, all sub-sequent levels inherit the 0% coverage.

  Note: A constraint with 0% coverage can be considered invalid because it does not impact timing analysis.

  Note: Pins tied to VCC/GND are reported as invalid pins.

The Clock Groups are not defined with `-from`, `-through`, and `-to` and therefore, the detailed table is different.

Figure 139: **Detailed Table for Clock Groups**



When a Clock Group constraint involves multiple groups and each group has multiple clocks, the table includes one row per clock-pair for all the possible combinations of clock-pairs. In this scenario, the constraint spans over multiple rows and each row refers to the same constraint position number.

The constraint position number 443 in the above design is defined as:

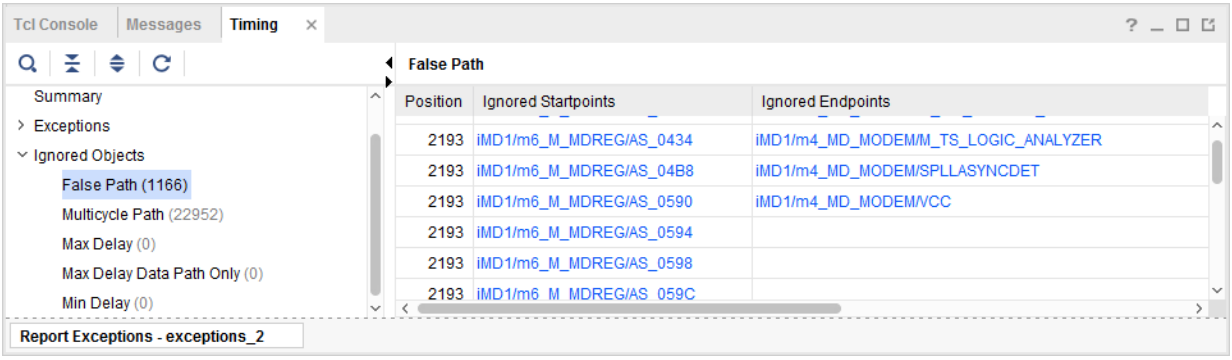
```
set_clock_groups -physically_exclusive -group RFSCLK -group LB_MD1SFSCLK
```

The constraint spans over two rows because some timing paths exist from clock `RFSCLK` to clock `LB_MD1SFSCLK` and from clock `LB_MD1SFSCLK` to clock `RFSCLK`.

Ignored Objects Section

This section reports the ignored startpoint and endpoints, organized by constraint type. This is equivalent to running `report_exceptions -ignored_objects` from the Tcl Console.

Figure 140: **Report Exceptions: Ignored Objects**



The table column Position represents the timing constraint position number that matches the position number reported inside the TCE. You can double-click a row to be redirected to the selected constraint inside the TCE. An alternative is to right-click on the row and select View Constraint in the popup menu.

The table columns Ignored Startpoints and Ignored Endpoints report the ignored pins. A pin is ignored when it is not a valid startpoint or endpoint, depending on which of the `-from` and `-to` pattern the pin was specified. A constraint can span over multiple rows, depending on the number of pins that are reported. Use the hyperlinks to select the design objects. After selection, the properties can be reviewed in the Property page and the schematic opened by pressing the **F4** key.

Report Clock Domain Crossings

The Clock Domain Crossings (CDC) report performs a structural analysis of the clock domain crossings in your design. You can use this information to identify potentially unsafe CDCs, which will lead to metastability or data coherency issues. While the CDC report is similar to the Clock Interaction Report, the CDC report focuses on structures and their timing constraints, but does not provide information related to timing slack.

When run from the Tcl Console, the CDC report can be scoped to one or more hierarchical cells using the `-cells` option. If the CDC is scoped, checks are reported when either the source or destination pins are inside the list of hierarchical cells. The scoping option is not available from the Report CDC GUI.

Overview

Before generating the CDC report, you must ensure that the design has been properly constrained and there are no missing clock definitions. Report CDC only analyzes and reports paths where both source and destination clocks have been defined. Report CDC performs structural analysis:

- On all paths between asynchronous clocks.

- Only on paths between synchronous clocks that have the following timing exceptions:
  - Clock groups
  - False Path
  - Max Delay Datapath Only

  Synchronous clock paths with no such timing exception are assumed to be safely timed and are not analyzed by the CDC engine. The Report CDC operates without taking into consideration any net or cell delays.

Terminology

The terminology for *safe*, *unsafe*, and *endpoints* is different in the context of Cross Domain Crossing (CDC) and inter-clock timing analysis.

In the context of CDC, an asynchronous crossing is safe when proper synchronization circuitry is used to prevent metastability. For example, a safe single-bit CDC can be implemented by a synchronizer, which is a chain of registers with same clock and control signals. A safe multi-bit CDC can be implemented with a MUX Hold circuitry or a Clock Enabled Controlled circuitry.

Conversely, an unsafe CDC is when the CDC analysis engine does not recognize a known safe synchronization circuit on an asynchronous CDC path.

The number of endpoints reported for CDC between two clock domains can be different than the number of endpoints reported by the timing analysis commands. For example, an asynchronous reset synchronizer involves multiple timing path endpoints. However, the synchronization circuitry is reported as a single element and therefore as a single CDC endpoint. Similarly, a multi-bit CDC contains multiple single bit crossings but is reported as a single CDC endpoint. However, the same bus is reported as multiple timing endpoints by other timing reports.



IMPORTANT! *Because* `report_clock_interaction` *and* `report_cdc` *have different purposes, the number of endpoints reported by each command must not be compared. In the context of* `report_clock_interaction`*, safe/unsafe refers to the ability for the timing analysis engine to provide a slack that matches the worst situation in hardware. For* `report_cdc`*, safe/unsafe refers to the type of CDC circuitry implemented in the design.*

Running Report Clock Domain Crossings

When you run Report CDC from the Vivado IDE, it provides all the details for the CDC paths between the specified clocks by default. When you run Report CDC from the Tcl Console, however, it only prints the Summary by Clock Pairs table. You must specify the `-details` option to report all the details as in the GUI mode. Reporting the details can create very long files or log files.

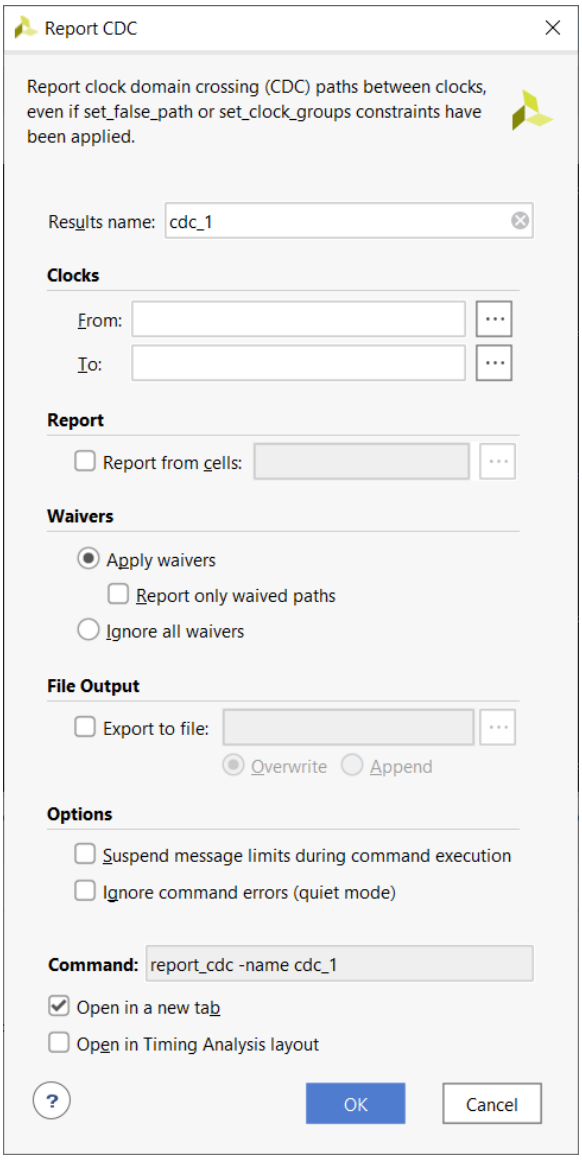
To run the Report Clock Domain Crossings in the Vivado IDE, select **Reports → Timing → Report CDC**.

Equivalent Tcl command: `report_cdc -name cdc_1`

In the Vivado IDE, the Report CDC dialog box includes the following fields, as shown in the following figure:

- Results Name Field
- Clocks Field (From/To)
- File Output Field
- Options Field

Figure 141: **Report CDC Dialog Box**



Results Name Field

In the Results Name field at the top of the Report Clock Domain Crossings dialog box, specify the name of the graphical window for the report.

Equivalent Tcl option: `-name <windowName>`

Clocks Field (From/To)

The Clocks To and From fields allow you to optionally specify the source and/or destination clocks on which to run the CDC analysis. You can use the From/To options to control the scope of Report CDC to specific clocks and result in more readable reports. Click the Browse button  to the right to open a search dialog box to aid in finding clock objects.

Equivalent Tcl option: `-from <clockNames> -to <clockNames>`

File Output Field

The File Output field allows you to optionally specify a file into which to write the results. You can overwrite the file or append to it.

Equivalent Tcl option: `-file <fileName> -append`

Options Field

The Options field allows you to:

- Suspend message limits during command execution

  Equivalent Tcl option: `-verbose`

- Ignore command errors

  Equivalent Tcl option: `-quiet`

Understanding the Clock Domain Crossings Report Rules

Report CDC tries to match each CDC path to a known CDC topology. Each CDC topology is associated with one or several CDC rules, as presented in Table 13: CDC Rules and Description. Note that you cannot modify the severity of the rules as with DRCs and Messages. Simplified schematics and descriptions of the CDC Topologies being detected are included in Simplified Schematics of the CDC Topologies.

The CDC topologies are analyzed based on some precedence rules. Table 14: CDC Rules and Precedence (Highest to Lowest) shows the CDC rules ordered from the highest to the lowest precedence. By default, only one CDC violation is reported at most per endpoint and if multiple violations exist on a particular endpoint, the CDC rule with the highest precedence is reported and masks any lower precedence CDC violation. For example, since CDC-15 has a higher precedence than CDC-10, a safe CDC-15 detected on a register masks an unsafe CDC-10 on the pin D of the same register.

Note: The default behavior can be overridden with the following command line option:

```
-all_checks_per_endpoint
```

This option forces the tool to report all the Info, Warning, and Critical checks that apply on the endpoints, regardless of the rules of precedence. However, unsafe rules on a register are not reported if at least one safe rule on the same register is detected.

Table 13: **CDC Rules and Description**

CDC Topology	CDC Rule	Severity	Description
Single-bit CDC	CDC-1	Critical	A single-bit CDC path is not synchronized or has unknown CDC circuitry.
	CDC-2	Warning	A single-bit CDC path is synchronized with a 2+ stage synchronizer but the ASYNC_REG property is missing on all or some of the synchronizer flip-flops.
	CDC-3	Info	A single-bit CDC path is synchronized with a 2+ stage synchronizer and the ASYNC_REG property is present.
Multi-bit CDC	CDC-4	Critical	A multi-bit bus CDC path is not synchronized or has unknown CDC circuitry.
	CDC-5	Warning	A multi-bit bus CDC path is synchronized with a 2+ stage synchronizer but the ASYNC_REG property is missing on all or some of the synchronizer flip-flops.
	CDC-6	Warning	A multi-bit bus CDC path is synchronized with a 2+ stage synchronizer and the ASYNC_REG property is present.
Asynchronous Reset	CDC-7	Critical	An asynchronous signal (clear or preset) is not synchronized or has unknown CDC circuitry.
	CDC-8	Warning	An asynchronous signal (clear or preset) is synchronized but the ASYNC_REG property is missing on all or some of the synchronizer flip-flops.
	CDC-9	Info	An asynchronous reset is synchronized and the ASYNC_REG property is present.
Combinatorial Logic	CDC-10	Critical	Combinatorial logic has been detected in the fanin of a synchronization circuit.
Fanout	CDC-11	Critical	A fanout has been detected before a synchronization circuit.
Multi-Clock Fanin	CDC-12	Critical	Data from multiple clocks are found in the fanin of a synchronization circuit.
non-FD primitive	CDC-13	Critical	CDC detected on a non-FD primitive.
CE-controlled CDC	CDC-15	Warning	Clock Enable controlled CDC.
Mux-controlled CDC	CDC-16	Warning	Multiplexer controlled CDC.
Mux Data Hold CDC	CDC-17	Warning	Multiplexer data holding CDC.
HARD_SYNC primitive	CDC-18	Info	A signal is synchronized with a HARD_SYNC primitive.
LUTRAM-to-FD CDC	CDC-26	Warning	LUTRAM read/write potential collision.

Table 14: **CDC Rules and Precedence (Highest to Lowest)**

CDC Topology	CDC Rule
HARD_SYNC primitive	CDC-18
Non-FD primitive	CDC-13
Mux Data Hold CDC	CDC-17
Mux-controlled CDC	CDC-16
CE-controlled CDC	CDC-15

Table 14: **CDC Rules and Precedence (Highest to Lowest)** *(cont'd)*

CDC Topology	CDC Rule
LUTRAM-to-FD CDC	CDC-26
Asynchronous Reset	CDC-7
Single-bit CDC not synchronized	CDC-1
Multi-bit CDC not synchronized	CDC-4
Multi-Clock Fanin	CDC-12
Combinatorial Logic	CDC-10
Fanout	CDC-11
Asynchronous Reset synchronized with property	CDC-9
Single-bit CDC synchronized with property	CDC-3
Multi-bit CDC synchronized with property	CDC-6
Asynchronous Reset synchronized without property	CDC-8
Single-bit CDC synchronized without property	CDC-2
Multi-bit CDC synchronized without property	CDC-5

Reviewing the Clock Domain Crossings Report Sections

In the GUI mode, three sections are generated by default:

- Summary by Clock Pair
- Summary by Type
- Detailed Report

The summary sections provide a convenient overview of the issues that need review and possibly a change in the design. These sections can be used to navigate to the violations of highest Severity where additional information is contained within the Detailed Report section.

***Note*:** By default, only the Summary by clock pair section is generated when running the report in text mode.

Summary by Clock Pair

In the Summary (by clock pair) section, useful information about the number of CDC paths between two clocks are presented, along with the severity of the most critical issue found among these paths. The table includes the following columns:

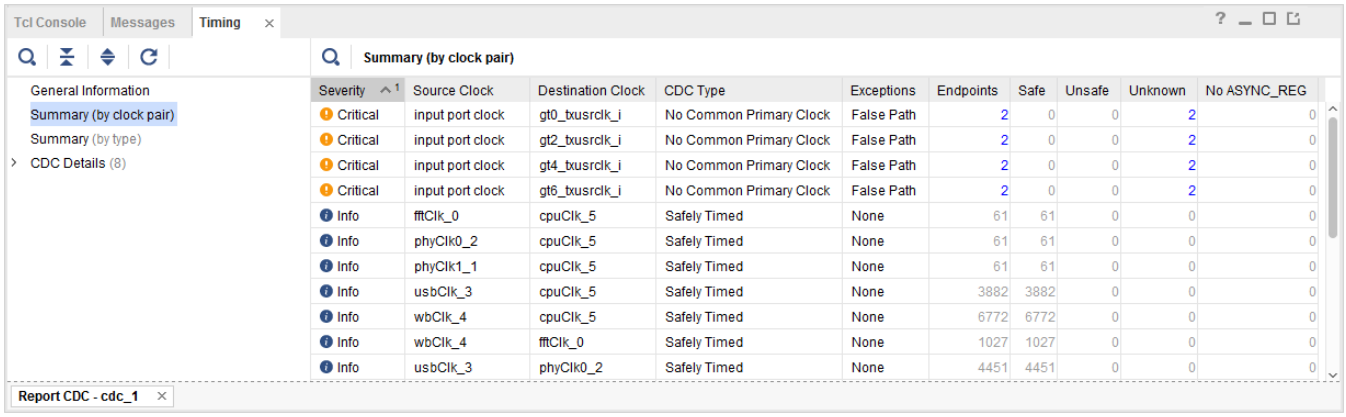
- **Severity:** Reports the worst severity of all CDC paths from/to the listed clocks. Values are Info, Warning, or Critical.
- **Source Clock:** Shows the name of the CDC source clock.
- **Destination Clock:** Shows the name of the CDC destination clock.

- **CDC Type:** Reflects the relationship between two clocks and their dominant timing exception, if any. Possible types are:
  - **Safely Timed:** All CDC paths are safely timed because the clocks are synchronous and accurate timing is not prevented by a timing exception.
  - **User Ignored:** All CDC paths are covered by `set_false_path` or `set_clock_groups`.
  - **No Common Primary Clock:** The CDC clocks are asynchronous and at least 1 CDC path is normally timed between two clocks that do not have a common primary clock.
  - **No Common Period:** The CDC clocks are asynchronous and at least 1 CDC path is normally timed between two clocks that do not have a common period. For the definition of clocks with no common period, refer to Timing Analysis Key Concepts.
  - **No Common Phase:** The CDC clocks are asynchronous because there is no known phase relationship between the two clocks.
- **Exceptions:** The timing exceptions applied to the CDC (if any) are:
  - **None:** No clock group/false path/max delay datapath only timing exceptions exist on the CDC paths. Other timing exceptions such as multicycle paths, min delay, and max delay are not reported by `report_cdc`.
  - **Asynch Clock Groups:** The `set_clock_groups -asynchronous` exception was applied to the CDC clocks.
  - **Exclusive Clock Groups:** The `set_clock_groups -exclusive` exception was applied to the CDC clocks.
  - **False Path:** The `set_false_path` exception was applied to from/to the CDC clocks or to all CDC paths.
  - **Max Delay Datapath Only:** The `set_max_delay -datapath_only` exception was applied to all CDC paths. Note that Max Delay Datapath Only is reported when at least one CDC path is only covered by `set_max_delay -datapath_only`, while all other CDC paths are ignored due to `set_false_path` constraints.
  - **Partial Exceptions:** A mix of `set_false_path` and `set_max_delay -datapath_only` constraints are applied to some of the CDC paths, and at least one CDC path is normally timed.
- **Endpoints:** The total number of CDC path endpoints. This is the sum of safe, unsafe, and unknown endpoints. In this context, an endpoint is a sequential cell input data pin. An FD cell can be counted several times depending on the D, CE, and SET/RESET/CLEAR/PRESET connectivity. For some CDC topologies, only one endpoint is counted while there are effectively several paths crossing the clock domain boundary to reach the CDC structure. For example, in an asynchronous reset synchronizer, several CLEAR pins are connected to the crossing net, but only the first pin of the synchronizer chain is counted.

- **Safe:** The number of safe CDC path endpoints. Safe endpoints are endpoints on CDC paths identified as:
  - Asynchronous clocks with known safe CDC structures
  - Synchronous clocks with exceptions and known safe CDC structures
  - Synchronous clocks without exceptions that are safely timed regardless of the CDC structure
  - CDC synchronized with `HARD_SYNC` macro
- **Unsafe:** The number of CDC path endpoints that are recognized as having an unsafe structure. The unsafe endpoints are CDC-10, CDC-11, CDC-12 and CDC-13.
  - Combinatorial logic topology
  - Fanout topology
  - Multi-clock fan-in topology
  - Non-FD primitive topology
- **Unknown:** The number of unknown CDC path endpoints. No CDC structure can be matched on these endpoints or an unknown CDC circuitry has been detected (CDC-1, CDC-4 and CDC-7).
- **No ASYNC_REG:** The number of identified synchronizers that are missing the `ASYNC_REG` property on at least one of the two first FD cells of the chain.

The following figure shows an example of a Summary (by clock pair) section.

Figure 142: **Summary by Clock Pair Section**

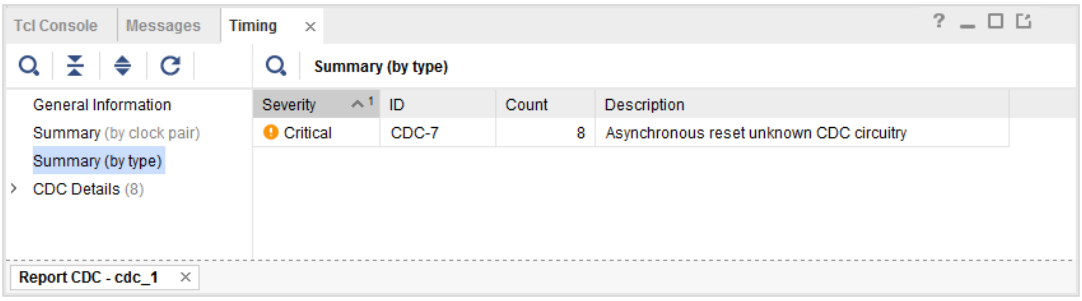


Severity	Source Clock	Destination Clock	CDC Type	Exceptions	Endpoints	Safe	Unsafe	Unknown	No ASYNC_REG
Critical	input port clock	gt0_txusrclk_i	No Common Primary Clock	False Path	2	0	0	2	0
Critical	input port clock	gt2_txusrclk_i	No Common Primary Clock	False Path	2	0	0	2	0
Critical	input port clock	gt4_txusrclk_i	No Common Primary Clock	False Path	2	0	0	2	0
Critical	input port clock	gt6_txusrclk_i	No Common Primary Clock	False Path	2	0	0	2	0
Info	fftClk_0	cpuClk_5	Safely Timed	None	61	61	0	0	0
Info	phyClk0_2	cpuClk_5	Safely Timed	None	61	61	0	0	0
Info	phyClk1_1	cpuClk_5	Safely Timed	None	61	61	0	0	0
Info	usbClk_3	cpuClk_5	Safely Timed	None	3882	3882	0	0	0
Info	wbClk_4	cpuClk_5	Safely Timed	None	6772	6772	0	0	0
Info	wbClk_4	fftClk_0	Safely Timed	None	1027	1027	0	0	0
Info	usbClk_3	phyClk0_2	Safely Timed	None	4451	4451	0	0	0

Summary by Type

The Summary by Type table is convenient for quickly reviewing the nature of CDC structures found in the current report. An example is shown in the following figure.

Figure 143: **Summary by Type Table**



The Summary by Type table includes the following columns:

- Severity: Reports the severity of the CDC Rule as Info, Warning, or Critical.
- ID: Unique identification number of the CDC Rule, as listed in Table 13: CDC Rules and Description.
- Count: Number of occurrences of the CDC Rule in the entire report.
- Description: Short description of the CDC Rule.

When analyzing the summary tables, it is important to start with the highest severity. Severity levels are:

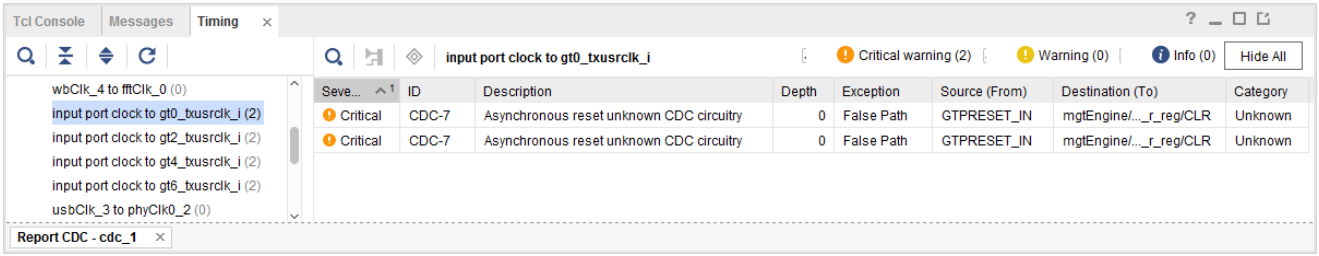
- Critical: This severity is for CDC paths with unknown or unsafe CDC Structures. You must review each individual path to either fix the structure by modifying the RTL, or waive the issue. The path details are generated by default when using the Vivado IDE, and only when `-details` is used with `report_cdc` on the command line.
  - There is some combinatorial logic on the crossing net or several source clocks are found in the fanin of the crossing net. This can degrade the Mean Time Between Failures (MTBF) characteristics.
  - There is a fanout on the crossing net to the same destination clock domain. This can lead to data coherency problems.
- Warning: This severity is for CDC paths with known CDC Structures that are safe but non-ideal due to one of the following reasons:
  - At least one of the two first synchronizer flip-flops does not have the `ASYNC_REG` property set to `1` (or `true`)
  - The CDC structure identified requires functional correctness that the CDC engine cannot verify. These structures are Clock Enable Controlled, MUX Controlled, and MUX Data-Hold controlled CDC topologies.
- Info: This severity indicates that CDC structures are all safe and properly constrained.

Detailed Report

The Report CDC details can be viewed by looking at the CDC Details section in the report. You can use the detailed report to view the schematic of the selected path (by pressing the F4 key), view the timing report, or generate a new timing report by right-clicking on the individual entry.

You can use the timing reports and schematics to review unexpected CDC paths in the design, to identify incorrect or missing timing exceptions, and to find missing `ASYNC_REG` properties. An example of the CDC Detailed Report is shown in the following figure.

Figure 144: **CDC Detailed Report**



The CDC Detailed Report table includes the following columns:

- Severity: Reports the severity of the CDC Rule as Info, Warning, or Critical.
- ID: The unique identification number of the CDC Rule, as listed in Table 13: CDC Rules and Description.
- Description: A short description of the CDC Rule.
- Depth: The number of synchronizer stages found (only applies to synchronizer topologies).
- Exception: The timing exception applied to the CDC path.
- Source (From): The CDC timing path startpoint.
- Destination (To): The CDC timing path endpoint.
- Category: Displays Safe, Unsafe, Unknown, etc.
- Source Clock (From): The name of the source clock.

  ***Note*:** This column displays only when you click **CDC Details** (left column of Timing-Report CDC window)

- Destination Clock (To): The name of the destination clock.

  ***Note*:** This column displays only when you click **CDC Details** (left column of Timing-Report CDC window)

> **IMPORTANT!** *The CDC report can flag issues in some of the AMD IPs because the CDC engine does not recognize all possible CDC topologies and does not provide a built-in waiver mechanism. More information can be found in each AMD IP Product Guide.*

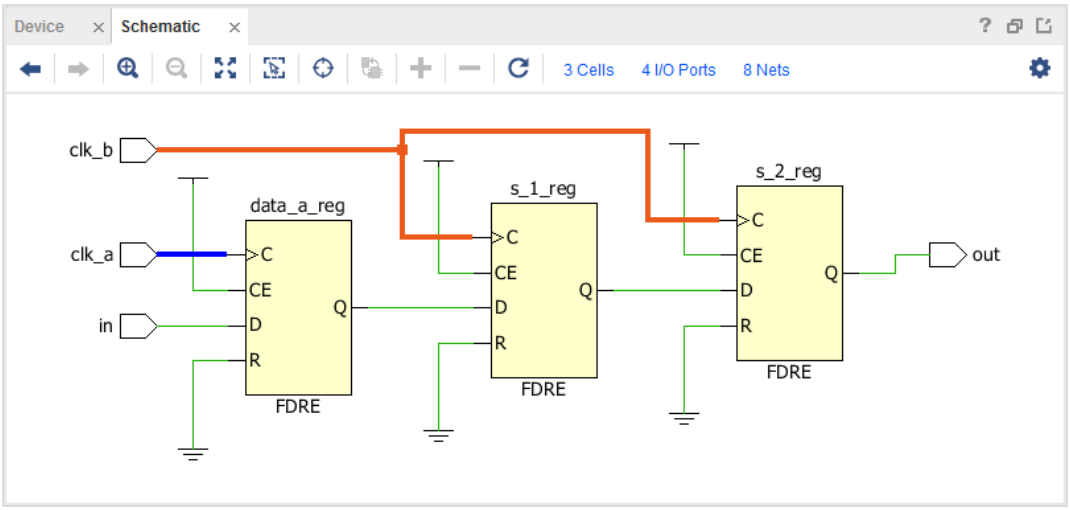
Simplified Schematics of the CDC Topologies

Simplified schematics of the CDC Topologies along with brief descriptions are shown in the following sections. In all schematics, the Source Clock net (typically `clk_a`) is highlighted in blue and the Destination Clock net (typically `clk_b`) is highlighted in orange.

Single-Bit Synchronizer

The simplified topology of a Single-bit synchronizer is shown in the following figure. The `ASYNC_REG` property must be set on at least the first two flip flops of the synchronization chain. The synchronizer depth is defined by the number of chained flip-flops that share the same control signals.

Figure 145: **Simplified Topology of a Single-Bit Synchronizer**

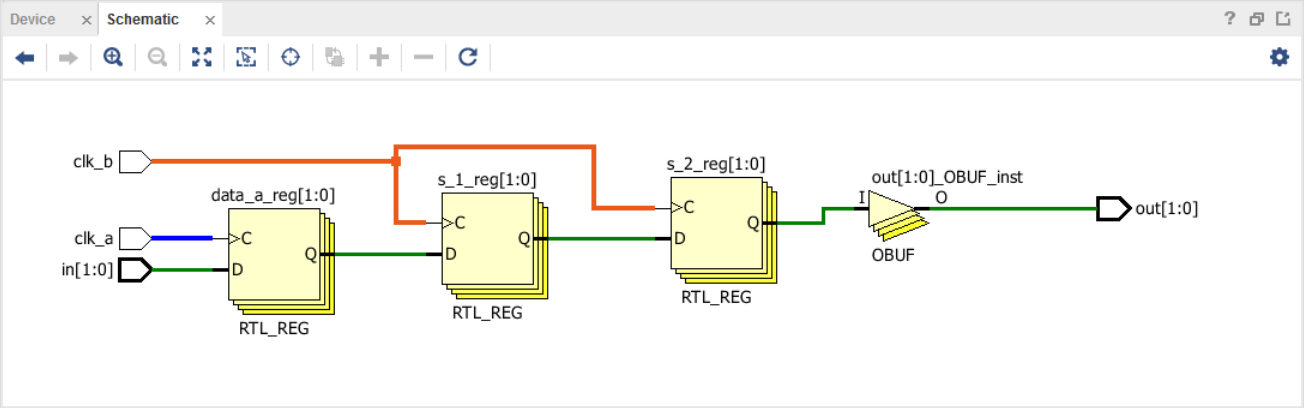


If the `CLEAR` or `PRESET` pins of the flip-flops are also connected to an asynchronous source, the synchronizer is only reported as a single-bit synchronizer and not as an asynchronous reset synchronizer.

Multi-Bit Synchronizer

The multi-bit synchronizer topology that is detected is equivalent to multiple single-bit synchronizers grouped together based on the startpoint-endpoint names and matching CDC rules. In this context, a bus is defined by the startpoint and endpoint cell names and not by the net names. The expected bus name format is `baseName[index]`. Also the startpoint and endpoint indexes must match. The following figure shows an example of a multi-bit synchronizer that is 2 bits wide.

Figure 146: **Multi-Bit Synchronizer with 2-Bit Width**



If all bits of a CDC bus do not match the same CDC rule, the bus is reported as single bits or bus segments with continuous indexes that match a same CDC rule.

It is essential to understand that having a register-based synchronizer on a bus does not make the domain crossing safe for the bus. This is the reason the CDC rule CDC-6 is a Warning, as the tool cannot decide whether or not the topology is adequate for the design. It is up to the designer to confirm the safety of the CDC.

If the bus is Gray coded, it is safe to use a register-based synchronizer on all the bits of the bus as long as the adequate timing constraints have been set on the bus to make sure that no more than one data at a time can be captured by the receiving domain.

If the bus is not Gray coded, other synchronizer topologies should be used instead, such as CE Controlled CDC or MUX Controlled CDC.

Asynchronous Reset Synchronizer

The synchronization of an asynchronous reset is shown in the following figure for CLEAR-based synchronization, and in the subsequent figure for PRESET-based synchronization. The FF1 cell is respectively connected to the synchronized clear or preset signals and their deassertion can safely be timed against `clk_a`. Note that flip-flops with `CLEAR` and `PRESET` cannot be mixed within an asynchronous reset synchronizer.

Figure 147: **CLEAR-Based Asynchronous Reset Synchronizer**

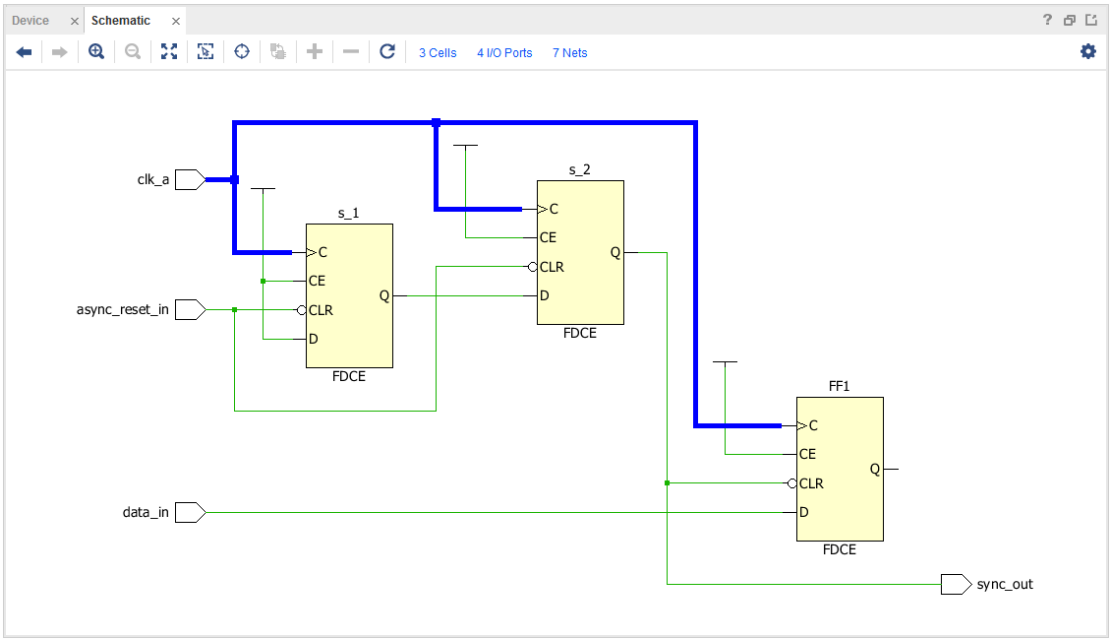
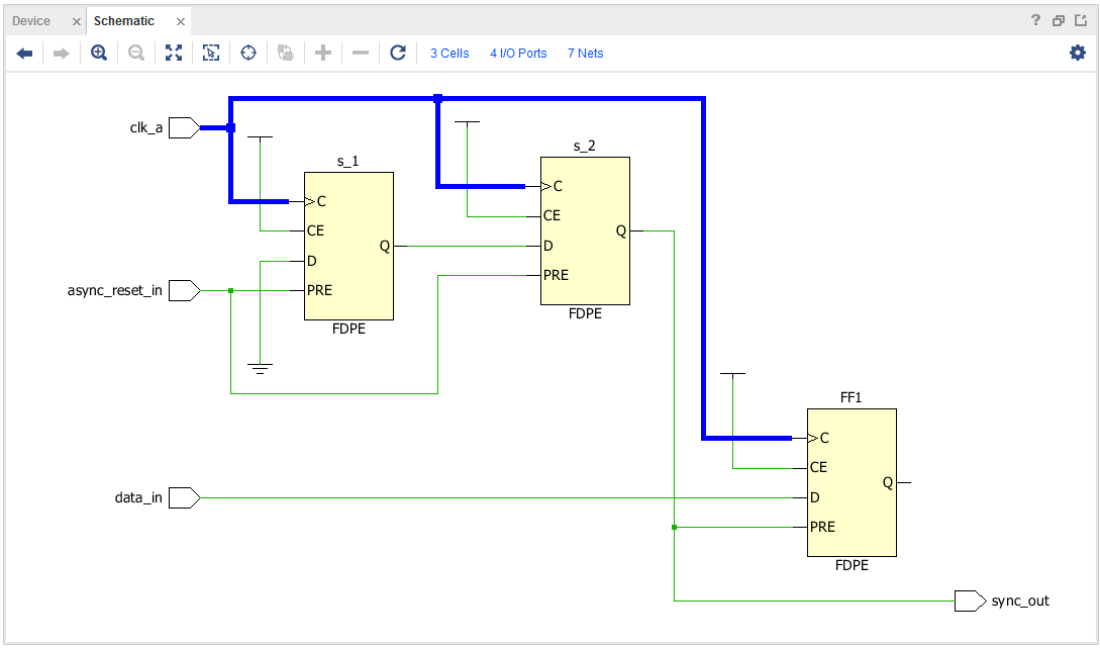


Figure 148: **PRESET-Based Asynchronous Reset Synchronizer**



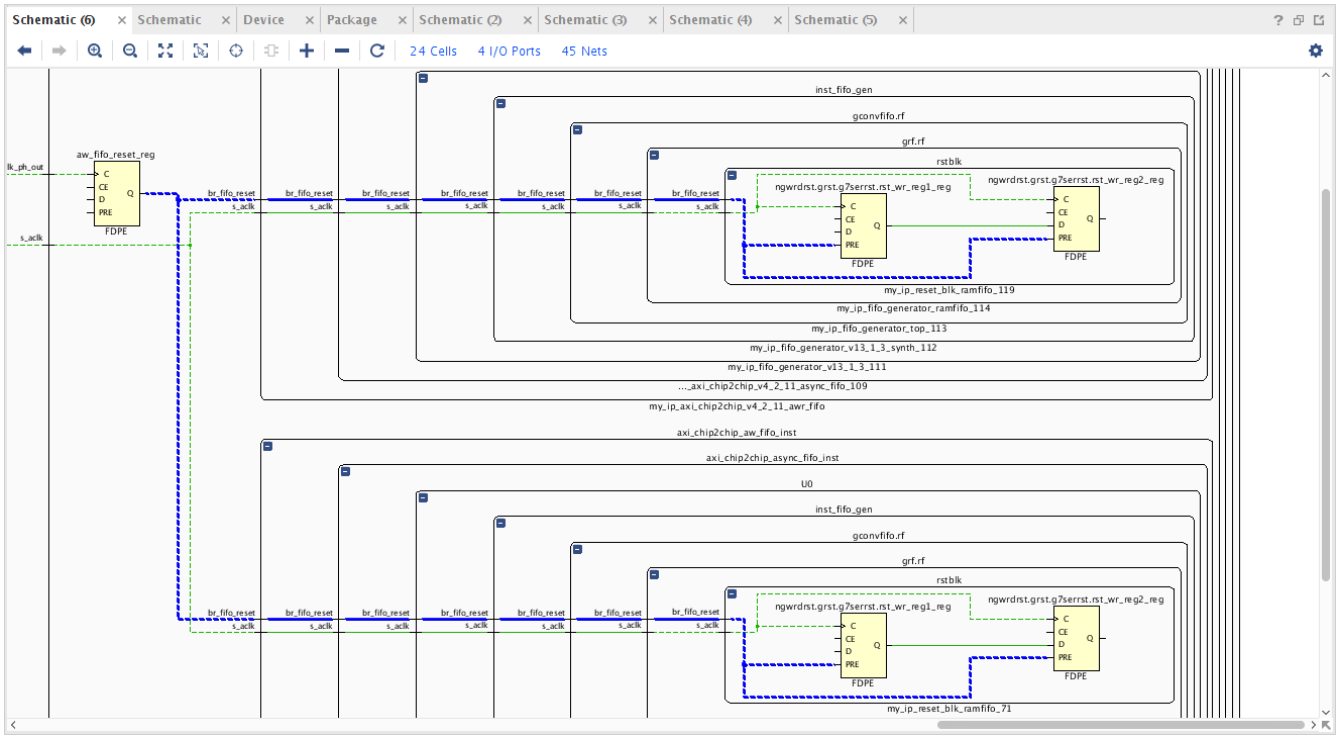
The general recommendation is to avoid multiple synchronizations of the reset signal inside the destination clock domain. This means that there should not be any fanout of the reset from the source clock domain into the destination clock domain. This recommendation prevents the destination clock domain to come out of reset at different time which could put the design in an unknown state. Failing to follow this recommendation results in a critical CDC-11 Fan-out from launch flop to destination clock violation.

However, there are scenarios involving the FIFO Generator IP where it is safe to have multiple synchronizations of the reset signal inside the destination clock domain. The FIFO Generator enters the reset state asynchronously and comes out synchronously. It applies true synchronous reset to block RAM though the FIFO receives the asynchronous reset. There will not be a situation where some part of logic is out of reset and some part is still in reset as long as its `wr_rst_busy` signal is used by the design to hold the data flow.

The AXI interface uses 5 FIFO Generator IPs to synchronize the reset in each of the destination clock domains and is another example of a reset circuitry that is safe by construction. In those scenarios when it is safe to synchronize the reset signal multiple times, the CDC-11 violations can be ignored.

The following figure illustrates an example of safe reset synchronization involving two FIFO Generators in the same destination clock domain.

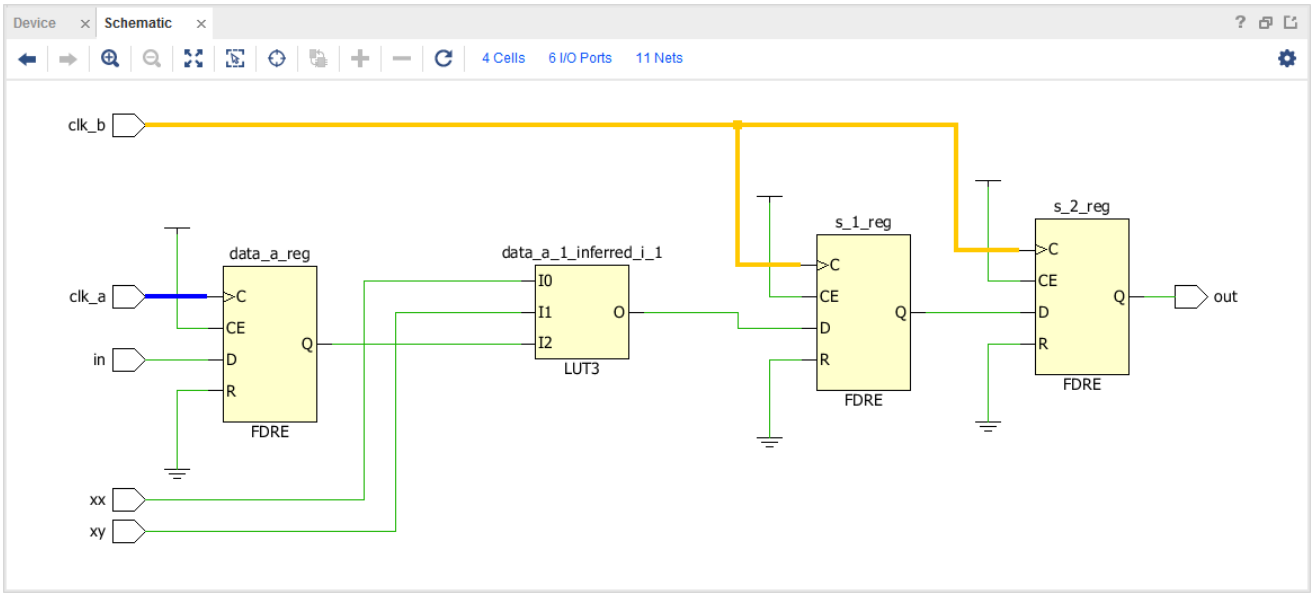
Figure 149: **Safe Reset Synchronization Example**



Combinatorial Logic

In the combinatorial logic simplified example presented in the following figure, a logic function represented by the LUT3 is placed between the CDC from `clk_a` to `clk_b` synchronizers.

Figure 150: **Combinatorial Logic Simplified Example**

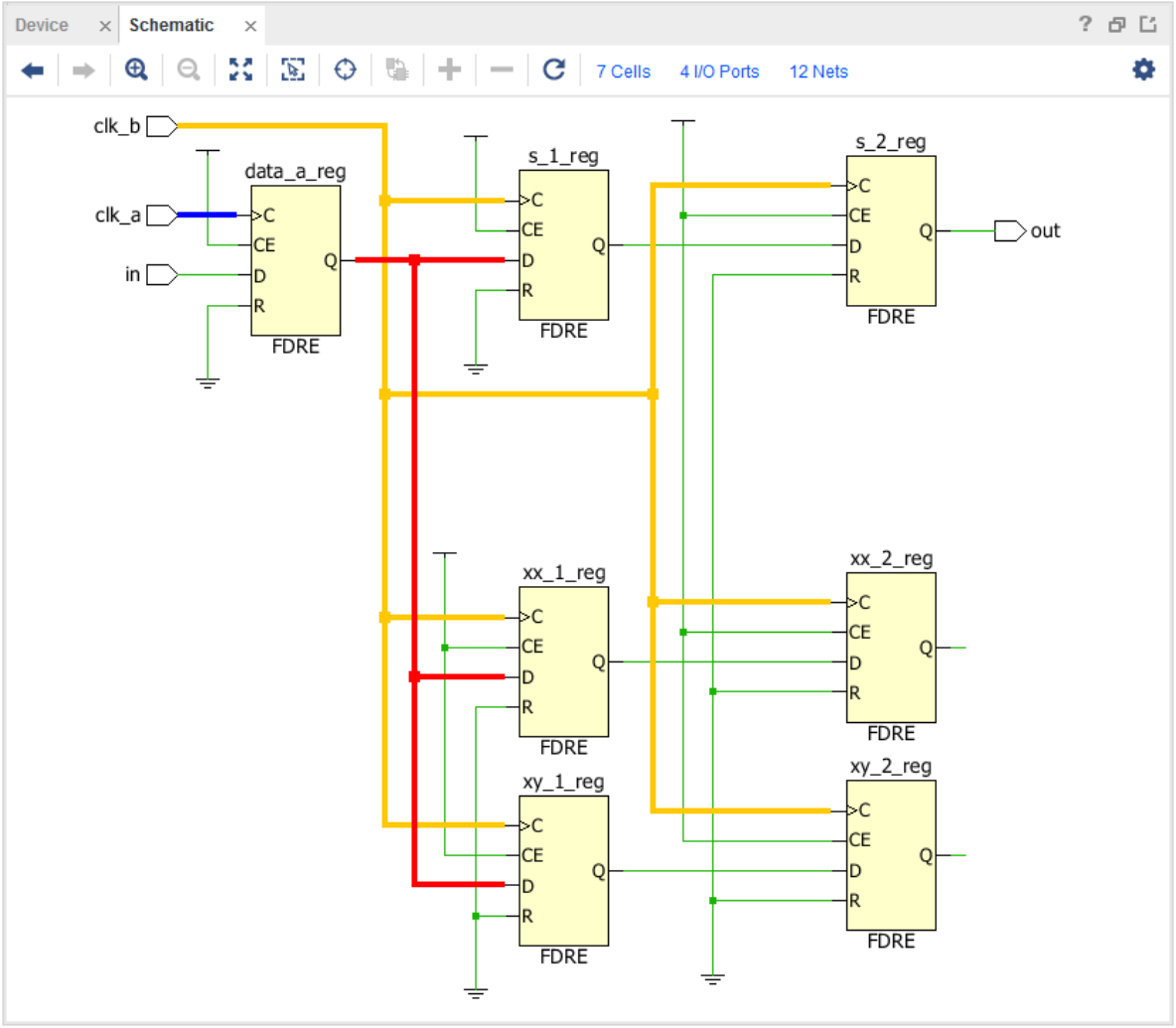


This structure is traditionally not recommended due the potential occurrence of glitches on the output of the combinatorial logic, which is captured by the synchronizer and propagated downward to the rest of the design.

Fanout

In the simplified Fanout example shown in the following figure, the source flip-flop drives a net that is synchronized three times in the `clk_b` domain highlighted in red. This structure is not recommended as it can lead to data coherency issues in the destination clock domain because the latency through the synchronizers is bounded but not cycle-accurate.

Figure 151: **Simplified Fanout Example**

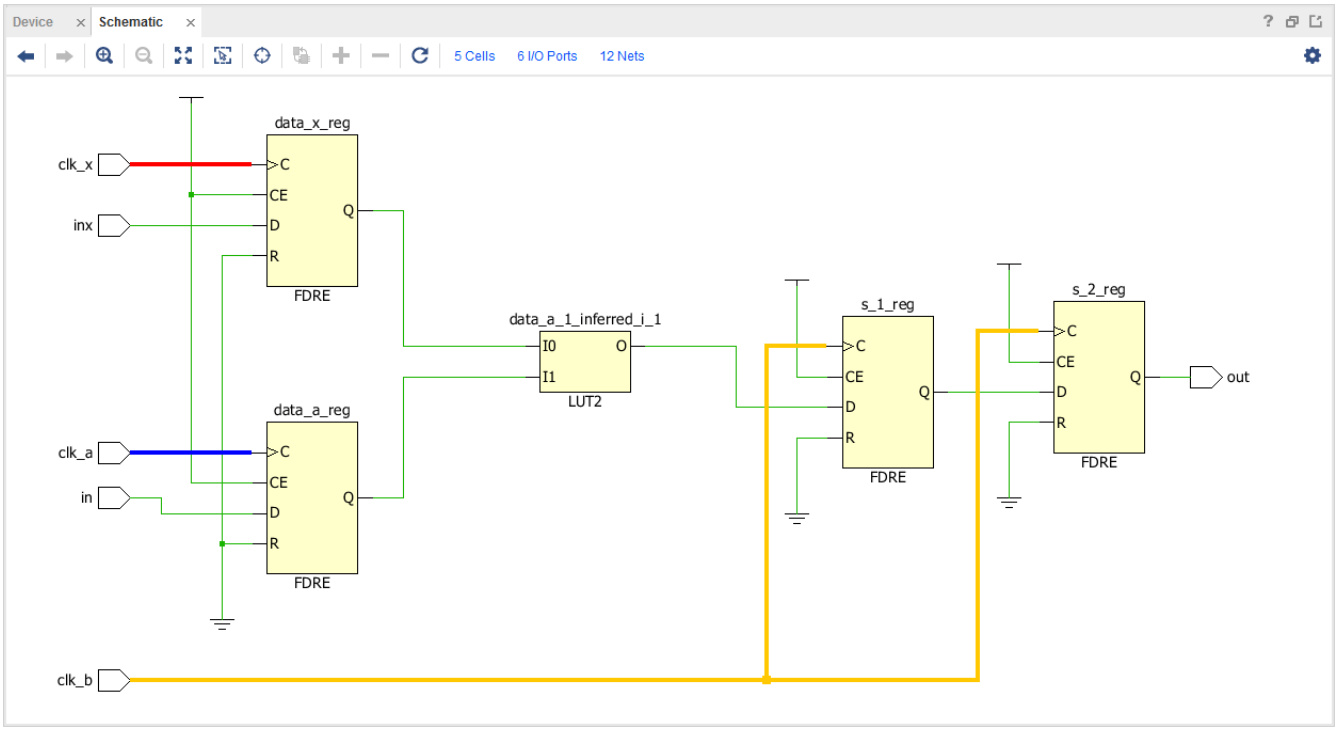


Note: A fanout of N to N different clock domains is not a CDC problem and does not trigger a CDC-11 violation. Refer to the section Asynchronous Reset Synchronizer for examples of safe fanout on reset signal.

Multi-Clock Fanin

In the Multi-Clock Fanin example shown in the following figure, both `clk_a` and `clk_x` are transferring data through combinatorial logic (LUT2) to the synchronizer circuit in the `clk_b` domain. It is recommended to first synchronize the source data from `clk_a` and `clk_x` individually before combining them via some interconnect logic | FPGA logic. This improves the MTBF characteristics of the overall CDC structure, and it prevents glitches to propagate to the destination clock domain.

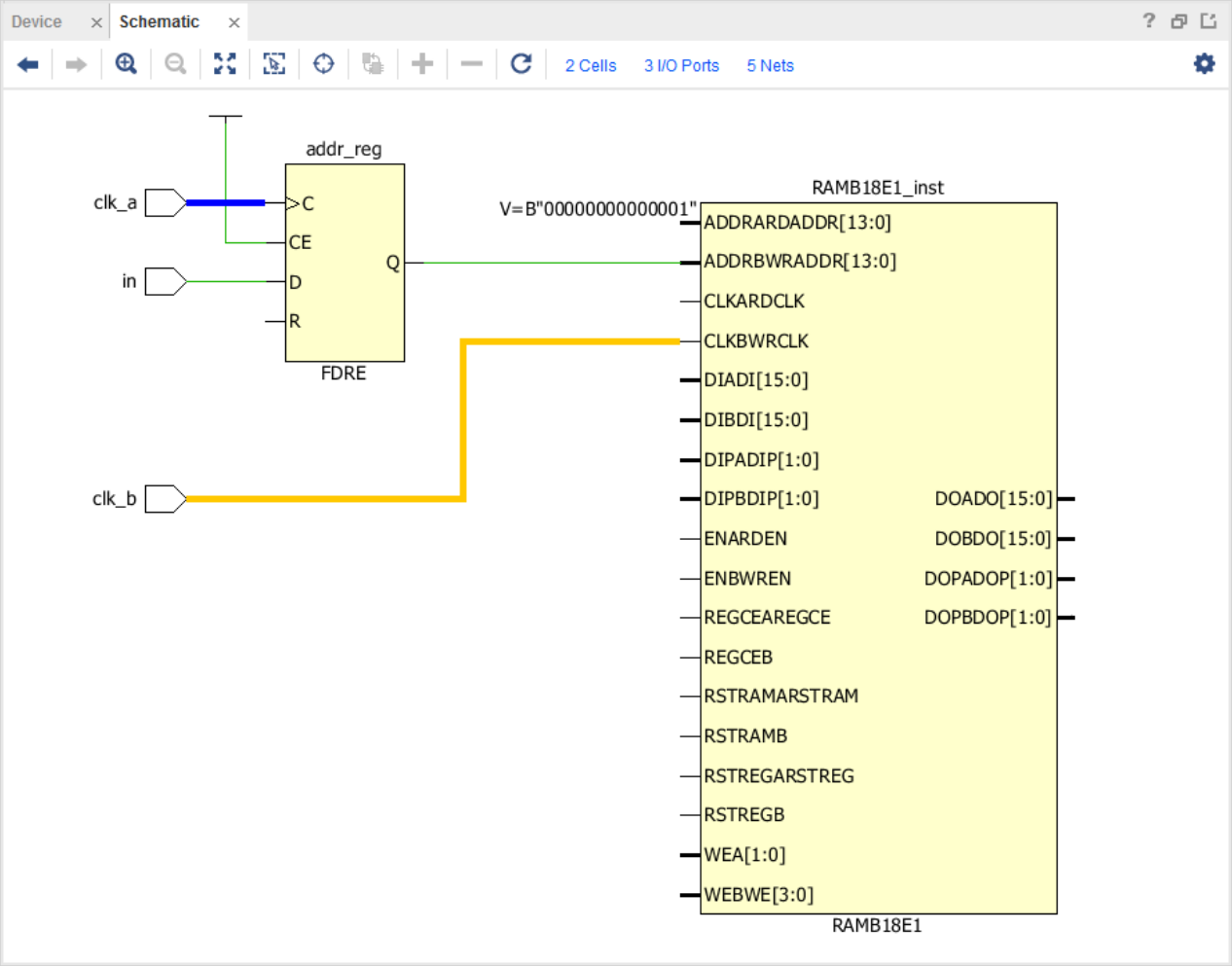
Figure 152: **Multi-Clock Fanin Example**



Non-FD Primitive

In the Non-FD Primitive example presented in the following figure, a CDC is occurring between a FDRE and a RAMB while no synchronization logic exists inside the `RAMB` primitive. Even if a single stage flip-flop connected to `clk_b` is inserted in front of the `RAMB`, it is still considered an inadequate synchronizer due to the routing distance between the FDRE and RAMB cells.

Figure 153: **Non-FD Primitive Example**

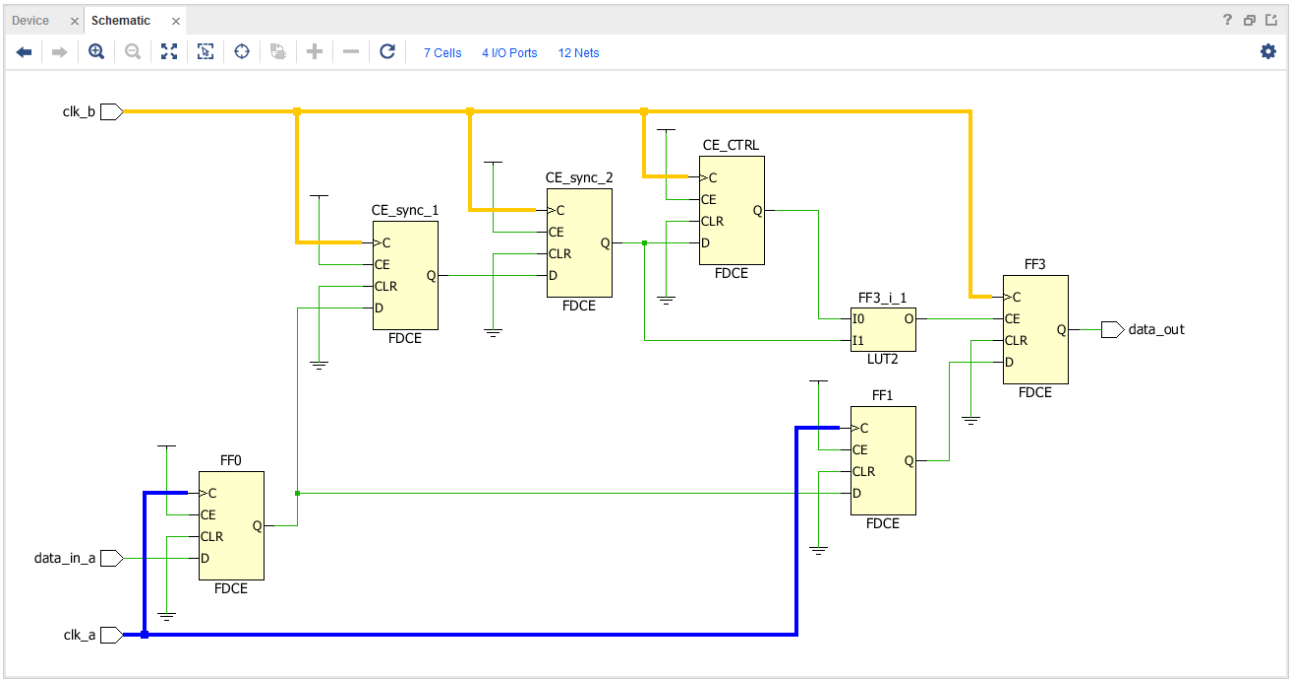


***Note*:** This rule does not include the `HARD_SYNC` macro, which is detected and covered by CDC-18.

CE-Controlled CDC

In the CE-controlled CDC example shown in the following figure, the clock enable signal is synchronized in the destination `clk_b` domain before being used to control the crossing flip-flops.

Figure 154: **CE-Controlled CDC Example**

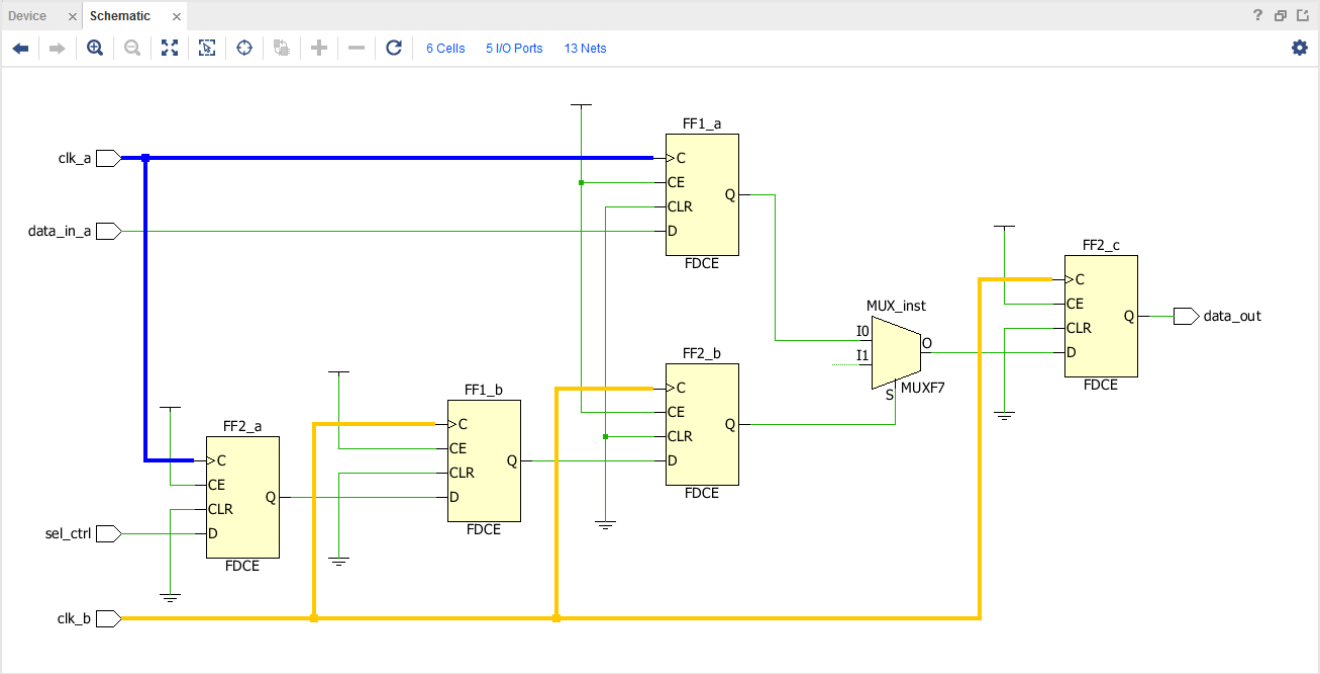


The CDC engine only checks that the signal connected to FF3/CE is also launched by `clk_b`. There is no restriction on how the clock enable signal is synchronized or on the circuitry driving the CE pin, as long as it is separately reported as a safe CDC path. Also, you are responsible for constraining the latency from the `clk_a` domain to FF3, which is usually done by a `set_max_delay -datapath_only` constraint.

Mux-Controlled CDC

In the Mux-controlled CDC example, shown in the following figure, the multiplexer select signal is synchronized to the destination clock domain, `clk_b`.

Figure 155: **Mux-Controlled CDC Example**

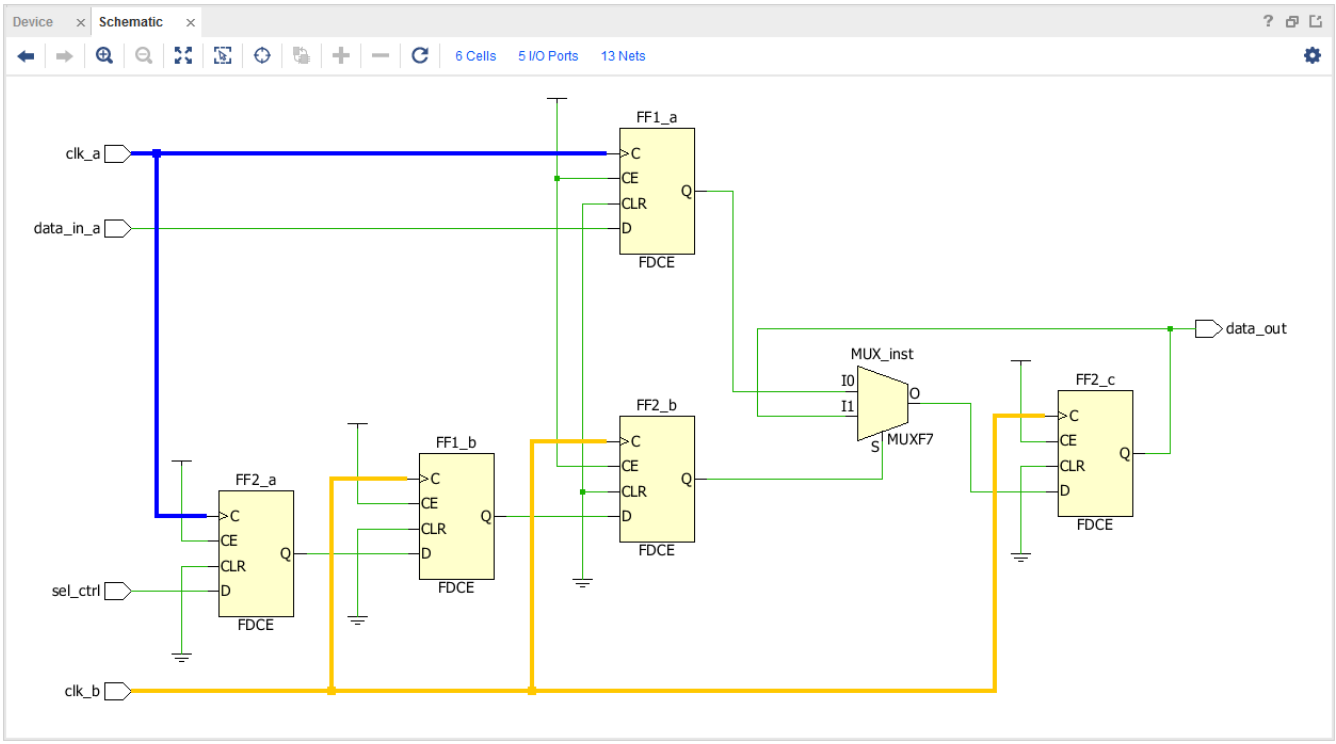


Similar to CE-controlled CDC, there is no restriction on how the select signal is synchronized as long as it is reported as safe individually and the user is responsible for constraining the crossing delay on `FF2_c`.

Mux Data Hold CDC

In the Mux Data Hold CDC example, presented in the following figure, the multiplexer select signal is synchronized to the destination clock domain `clk_b` and the `data_out` is fed back to the multiplexer.

Figure 156: **Mux Data Hold CDC Example**



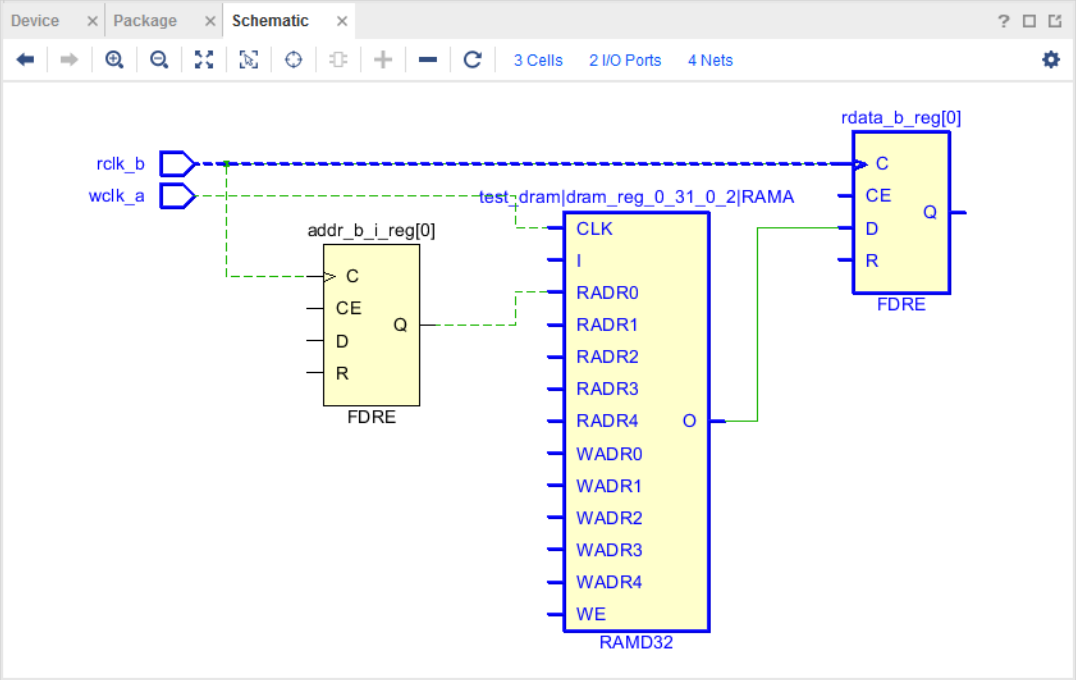
Similar to CE-controlled CDC, there is no restriction on how the multiplexer select signal is synchronized as long as it is reported as safe individually and the user is responsible for constraining the crossing delay on `FF2_c`.

LUTRAM Read/Write Potential Collision

In the LUTRAM Read/Write Potential Collision example below, the data is written inside the LUTRAM with the write clock and the output of the LUTRAM is captured by the read clock. When the read and write addresses are different, there is no CDC path between the write and the read clocks. However, when the write and read addresses are the same, then there is a CDC path between the write clock and the read clock.

To avoid a CDC path between the write and read clocks, it is necessary to ensure that the logic around the LUTRAM can never generate the same read and write addresses during active read and write operations. When this condition is ensured, the CDC violation related to this topology can be waived. AMD's FIFO Generator IP, for example, has a built-in logic that prevents any read/write collision.

Figure 157: **LUTRAM Read/Write Potential Collision**



Design Closure Reports

Report QoR Assessment

The `report_qor_assessment` command generates a report which provides the following:

- An assessment score that is indicative of how likely your design is to meet performance targets
- Flow guidance on the recommended next steps
- A summary of utilization and performance metrics
- A summary on methodology checks critical to QoR (only available in text version)
- Information on the availability of ML strategies

Overall Assessment Summary

The summary contains both the QoR Assessment Score and Flow Guidance information.

The assessment score predicts how likely the design is to achieve timing goals at the given point in the implementation flow. The earlier in the flow you run the command, the greater the benefit as there is more compile time to save. There is a small compromise on accuracy but the score should not be greater than one away from the final post-route score. It is generated by analyzing a complex set of design metrics in areas such as UltraFast methodology, device utilization, control sets, clocking, setup slack, and hold slack. In addition, device-specific characteristics are also considered and the design stage. For example, after `synth_design`, clocking netlist structures are closely examined, after `place_design` clock skew has a greater weighting as accuracy increases and after `route_design` it will consider other new factors such as if the design has been able to fully route. The scoring range is from 1 to 5. When the score is less than 5, use `report_qor_suggestions` to improve the score.

Score details are provided in the following table:

Table 15: **Report QoR Assessment Scoring**

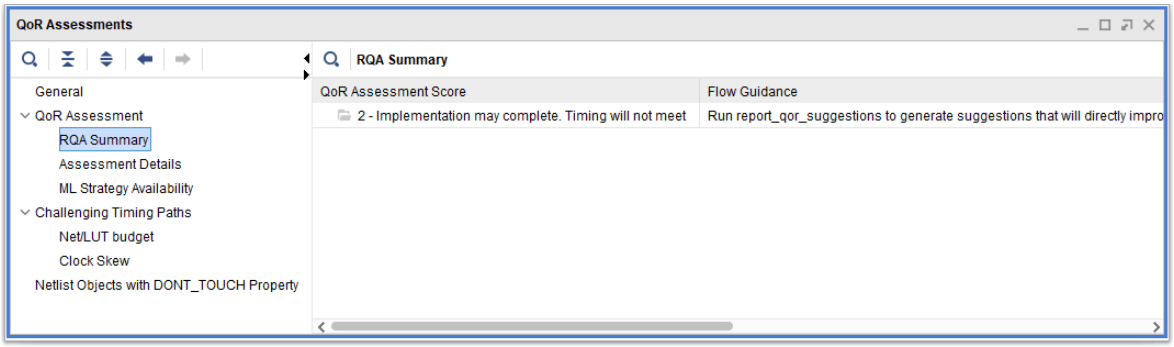
Score	Meaning
1	Design will likely not complete implementation.
2	Design will complete implementation but will not meet timing.
3	Design will likely not meet timing.
4	Design will likely meet timing.
5	Design will meet timing.

Flow guidance is part of the overall assessment summary. It dynamically updates depending on the current status of the design. It provides information about:

- Whether you need to address methodology issues
- Whether the design improves using QoR suggestions
- Whether to use ML strategies or incremental compile

The following figure shows an example design with a QoR Assessment Score of 2.

Figure 158: **Overall Assessment Summary**

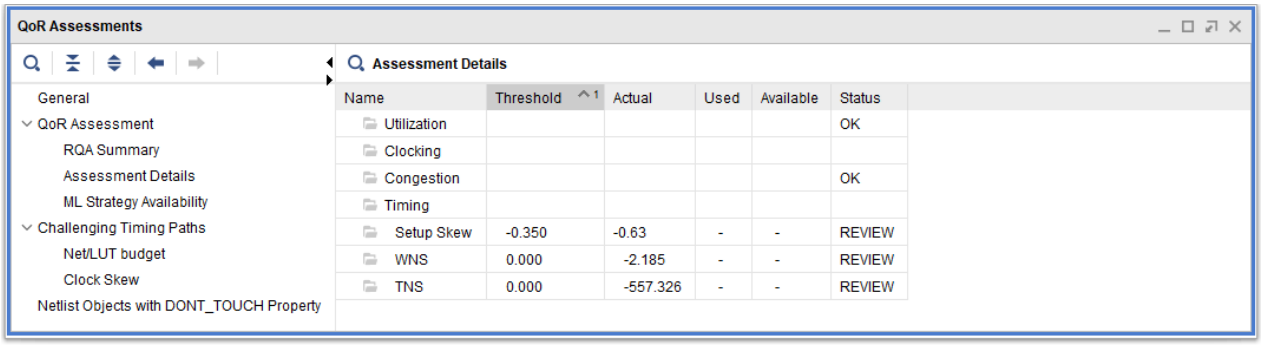


QoR Assessment Details

The QoR Assessment Details table, shown in the following figure, gives a convenient design overview that highlights issues in the following areas that form the basis of the RQA score.

- Utilization
- Netlist
- Clocking
- Congestion
- Timing

Figure 159: **QoR Assessment Details**



The table shows design characteristics broken into five categories. Each category is marked OK when there are no sub-items marked REVIEW. When sub-items are marked REVIEW, the failing item is displayed with its threshold and current value. The thresholds are not hard limits and can be exceeded, but going over these limits can make timing closure difficult. Pay particular attention when thresholds are significantly exceeded or when many categories are exceeding their thresholds. Items marked with an * do not directly contribute to the score but may be critical to whether the design will meet timing and should be reviewed.

Utilization checks are performed on the whole device, at the SLR level and the Pblock level. Running `report_qor_suggestions` can help reduce utilization.

Netlist checks are performed on the netlist structure and non timing constraints. These will identify items with DONT_TOUCH properties, high fanout nets with poor driver profiles and other design features that could add difficulty to the implementation tools.

Clocking shows whether there is high clock skew on setup or hold paths. Details of failing clock skew paths are automatically added to the report in the Vivado IDE. In text mode, add `-csv_output_dir <directory>` to generate the timing paths in CSV format. Running `report_qor_suggestions` can give automated fixes to many clock skew issues.

Congestion looks into the netlist for profiles that can contribute to routing congestion. Congested region information is not available before placement but some netlist items are available. You might wish to evaluate congestion by running place and route before fixing these items. Run `report_qor_suggestions` to generate suggestions that reduce congestion targeted at cells in the congested area.

Timing looks at the 100 worst case paths per clock group. It will look at:

- WNS, TNS, WHS, and THS to determine whether it is likely the design can close timing.
- Net budget checks: for routable nets, a conservative net delay is added instead of the estimated delay.
- LUT budget checks: for LUTs, the delay is swapped for a conservative LUT delay instead of using the estimated delay.

LUT and net budget checks allow an estimate that is less than ideal. Resolve paths that exceed the slack to reduce the number of issues seen later in the design flow. Examine the **Challenging Timing Paths** section in the Vivado IDE or generate a CSV file to see more information on these paths.

On a routed design, additional features are checked to see if the design can close using the last mile directive that is leveraged internally within the Intelligent Design Runs feature. This will check, whether timing paths can meet timing based on WNS, WHS, pre and post path slack and the primitives involved in the worst case timing paths.

Methodology Checks

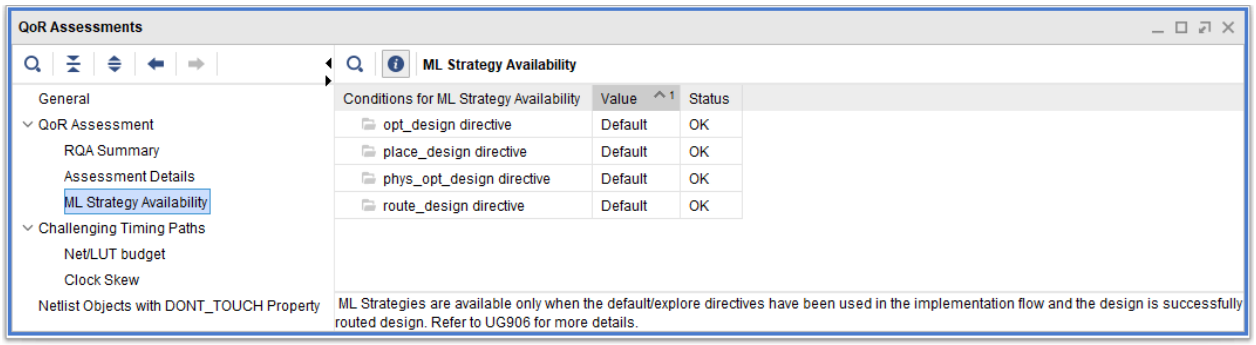
A limited number of methodology checks (from `report_methodology`) are run to ensure that a stable foundation is in place for QoR suggestions to be effective. If methodology checks are already generated, the cached results are reused unless there are changes in the design. If methodology checks are required to be run, this results in an increase in run time. Methodology checks can be disabled using the `-exclude_methodology_checks` switch.

***Note*:** In the Vivado IDE, this section is not generated. Instead, use the Report Methodology report to access methodology information. This information is generated by default for most implementation reporting strategies.

ML Strategy Availability

ML strategies are not generated by `report_qor_suggestions` if the required implementation flow in the reference run has not been run. The ML Strategy Availability table, shown in the following figure, checks off each required implementation step.

Figure 160: **ML Strategy Availability**



The flow requirements are as follows:

- The `opt_design` command must be run with the directive `Explore` or `Default`.

  Note: More than one call to `opt_design` is allowed, but the final call *must* meet this condition.

- The remaining implementation directives must be either all set to `Default` or all set to `Explore`. Mix and match of these implementation steps is not permitted.
- The `phys_opt_design` command must be enabled
- The design must be routed.

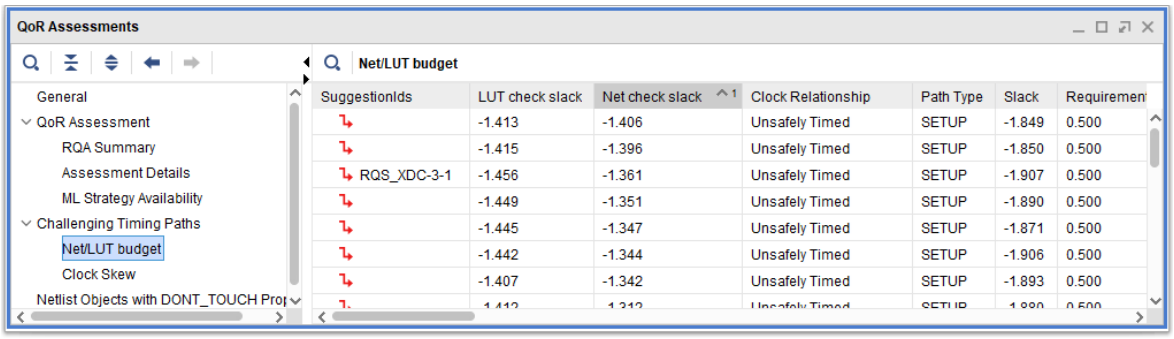
Challenging Timing Paths

The Challenging Timing Paths section lists key properties of timing paths that have failed checks in the Assessment Details section. By default, the command evaluates 100 failing paths per clock group. It analyzes the following factors:

- Net budget
- LUT budget
- Clock skew

The following figure shows an example of the Net/LUT budget report.

Figure 161: **Net/LUT Budget Report**



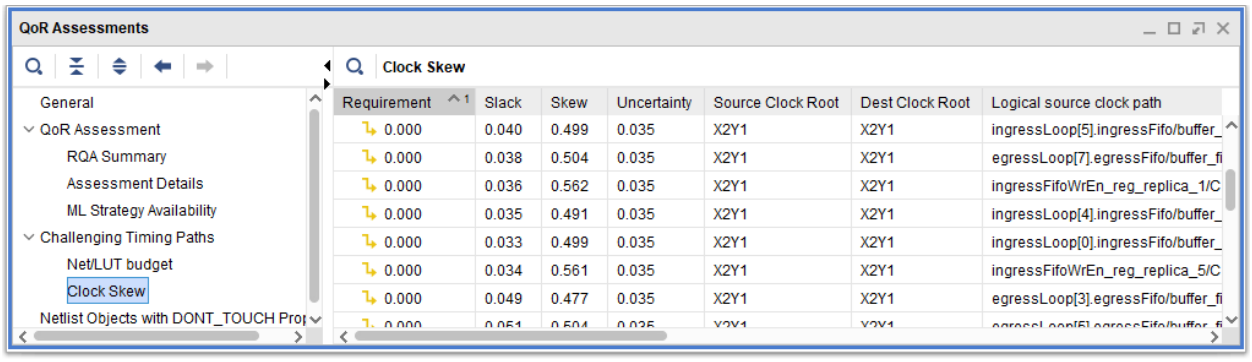
For these checks, estimated net or LUT delays are replaced with more typical values expected for the device and a new budget is calculated. Other penalties are added to the paths when they start or end at block RAMs, DSPs, or other hard blocks. There are different penalties when it is not possible to leverage clock tree skew to improve margin. The new slack based on the penalized path is shown in the LUT Check Slack and Net Check Slack columns.

Path-related QoR suggestions are shown in the SuggestionsID column. Items where suggestions do not exist should be investigated and potentially recoded. Where suggestions exist, applying the suggestion might resolve the issue without requiring code edits.

The Clock Skew section, shown in the following figure, reports on items relating to the clock skew:

- Skew value
- Source and destination clock names
- Clock root on both source and destination clocks
- Uncertainty

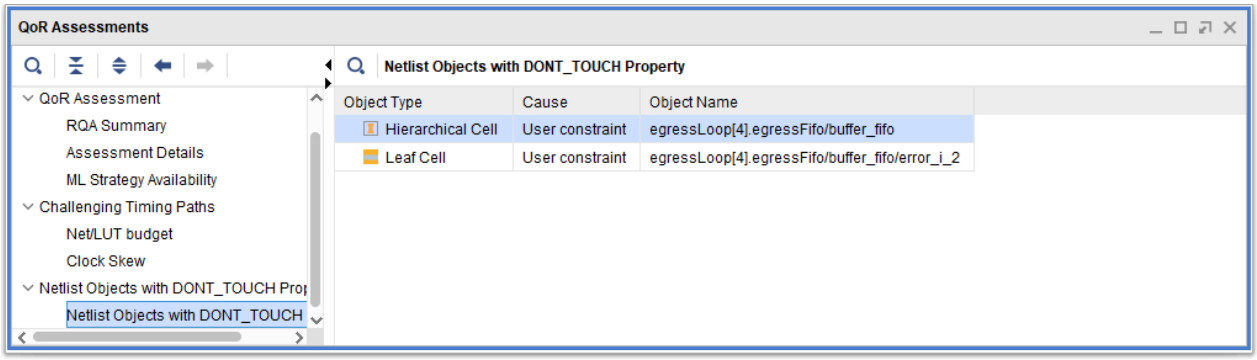
Figure 162: **Clock Skew Report**



Netlist Objects With DONT_TOUCH Properties

Netlist objects with DONT_TOUCH properties can prevent optimizations from being carried out that improve design performance. The following section of the RQA report shows which objects have the DONT_TOUCH property set on them:

Figure 163: **Netlist Objects with DONT_TOUCH Property**



The RQA report shows the following objects as well as their cause:

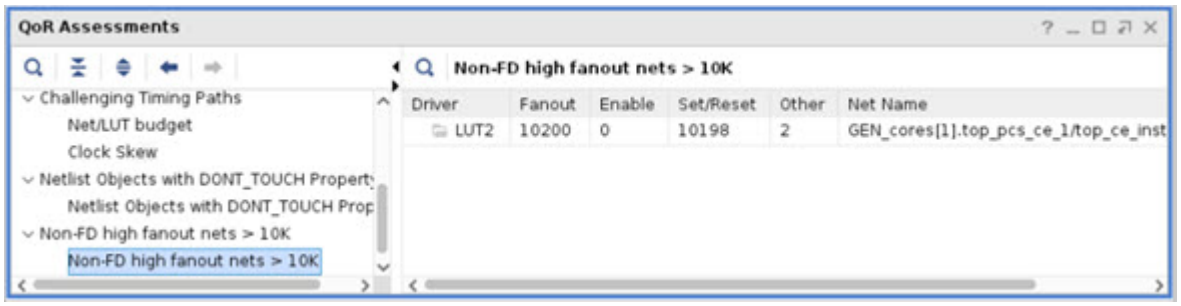
- Hierarchical cells
- Leaf cells
- Nets

***Note*:** DONT_TOUCH properties prevent the tools from optimizing paths and can be added through the use of other properties automatically by Vivado. Removing DONT_TOUCH properties should be done with care. For instance, the DFX flow uses `DONT_TOUCH` to prevent cross boundary optimizations between the static and reconfigurable module, so should not be removed. A DONT_TOUCH property added as a result of MARK_DEBUG, by contrast, is *not* critical to the flow, but it does mean that the signal is not available for hardware probing if optimized.

Non-FD High Fanout Nets > 10K

This section reports signals that have a fanout greater than 10K but are not driven by a flip flop. Nets that drive a high fanout should have a driver profile that allows replication to be carried out without impacting the pre-path timing. This allows the replicated cell to be more freely placed next to the loads. If there are source constraints, the placement of replicated cells get clustered together and reduce the effectiveness of the technique.

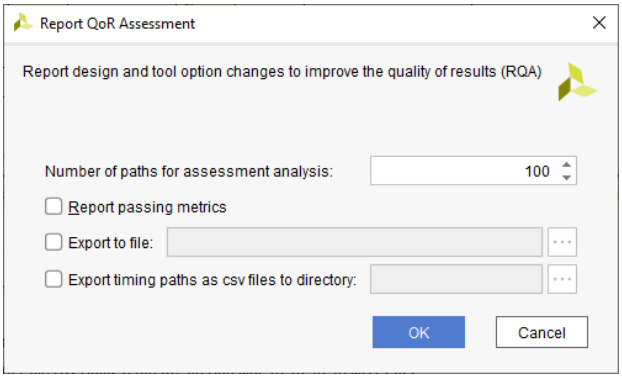
In the details section, further details such as the load types, information on the driver profile and the net name are captured. The timing information on this signal is not captured. If the path is a false path or has a very low requirement, it can be ignored.



Generating the QoR Assessment Report

The `report_qor_assessment` command can be accessed on an open design in the Vivado IDE by clicking **Reports → Report QoR Assessment**.

Figure 164: **Report QoR Assessment Dialog Box**



The equivalent command at the Tcl Console is as follows:

```
report_qor_assessment -name report_qor_assessment
```

To change the timing path limit from the default of 100, change the **Number of paths for suggestion analysis** shown in the dialog box. The equivalent Tcl command uses the `-max_paths <N>` switch, where N is an integer:

```
report_qor_assessment -max_paths <N>
```

The Report passing metrics option expands the details table to show all the metrics that were checked but passed. The equivalent Tcl command uses the `-full_assessment_details` switch:

```
report_qor_assessment -full_assessment_details
```

The option to generate supporting CSV files helps improve the ability to navigate a high volume of data when compared to the text report. To generate supporting CSV files, check the box and specify a directory. The CSV files are used to do the following:

- Show the timing paths contributing to the RQA score, and if suggestions are available to help address them
- Show cells and nets containing DONT_TOUCH properties that can prevent optimization

The equivalent Tcl command uses the `-csv_output_dir` switch:

```
report_qor_assessment -csv_output_dir <directory>
```

Other Command Options

- **`-exclude_methodology_checks`:** By default, methodology checks are run (or, if the command has been run previously, they are captured from the internal cache). This option allows methodology checks to be completely omitted. It offers a compile time saving when the command has not been run before. Because methodology checks are largely static throughout the implementation run, it is recommended to use this option when calling `report_qor_assessment` at multiple stages of the implementation run.

Auto Termination of Runs

When the QoR Assessment feature is run during a project-based implementation run, a score is generated between 1 and 5 that indicates if the design is unlikely (low) or likely (high) to close timing.

It is also possible to do an early termination of runs that are predicted to be low quality, allowing server resources to be freed up earlier than if the run had continued to the end. Early termination of a run occurs in the following circumstances:

- When the MIN_RQA_SCORE run property is set. For example when set to 3, design runs with RQA scores of 1 or 2 are terminated.
- The report_qor_assessment command is called in the Timing Closure Report Strategy after opt_design. If additional calls are required, a custom report strategy is recommended.

An example of setting the MIN_RQA_SCORE property to 1 on run `impl_1` is as follows:

```
set_property MIN_RQA_SCORE 3 [get_runs impl_1]
```

Note: This feature does not have a direct correlation in non-project mode. An easy way to generate the RQA score from script can be to call `report_qor_assessment` and then to call `report_design_analysis -qor_summary -json <filename>.json`.

Report QoR Suggestions

The `report_qor_suggestions` command is the principal command used when working with QoR suggestion objects. QoR suggestion objects create both commands and properties to improve the timing performance of the design (see QoR Suggestions for more information).

The `report_qor_suggestions` command performs two tasks:

1. Reports on QoR suggestion objects
2. Generates new QoR suggestion objects

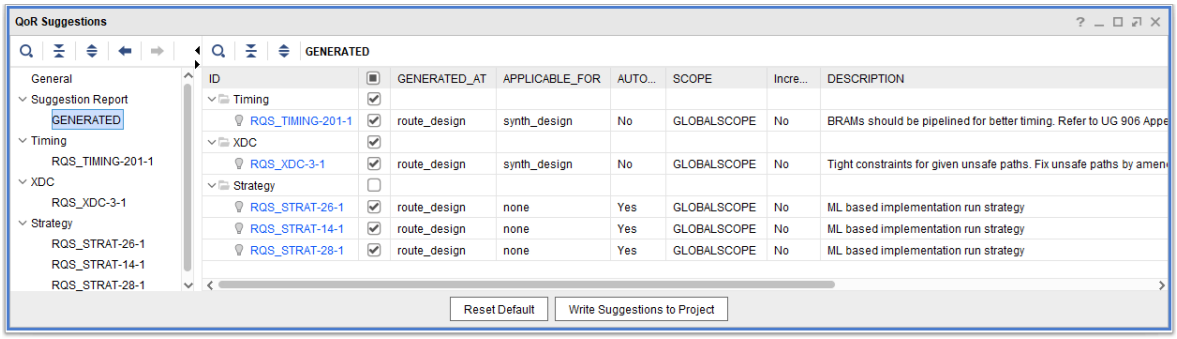
The command can be run on a design at any stage after synthesis.

Also captured in this section are some details on creating a (.rqs) file from the report using the `write_qor_suggestions` command.

QoR Suggestion Report

The QoR Suggestions report is split into a summary section with each suggestion at the top, and details on those suggestions in the bottom half of the report. The following table shows an example of the report generated.

Figure 165: **Example report_qor_suggestions Report**



In the report under Suggestion Report, there is a list of all suggestions. These are presented in four categories. They can be considered in the following pairs:

- **GENERATED and EXISTING:**
  - Generated suggestions are newly generated at the current stage of the flow.
  - Existing suggestions might have come from earlier in the flow, or by reading in an RQS file.
- **APPLIED and FAILED TO APPLY:**
  - Applied suggestions are suggestions that have been enabled and the APPLICABLE_FOR stage has been passed. They have been successfully applied.
  - Failed to apply suggestions have been enabled and passed through the APPLICABLE_FOR stage but were not successfully applied. Examine the log file to understand why suggestions have not been not applied. Applied suggestions are suggestions that have been enabled and the APPLICABLE_FOR stage has been passed. They have been successfully applied.

The lower half of the report contains details on the suggestions generated. It is split into the following categories in which `report_qor_suggestions` analyzes the design:

- Clocking
- Congestion
- Utilization
- Timing
- Netlist
- XDC
- Strategy

When looking at GENERATED suggestions, the detailed section should provide sufficient information so that you can infer why the suggestion has been reported. It is possible to cross-probe from the details section for GENERATED suggestions. The following cross-probing methods are useful:

- Selecting objects to highlight objects in other windows (for example, Device view)
- Pressing **F4** to show schematics of selected objects
- Right-clicking on objects to generate a timing report

When looking at EXISTING suggestions, it is possible that the objects have been modified and do not exist (for example, `opt_design` might remove objects from the netlist). For this reason, cross-probing is not always available when selecting EXISTING suggestions.

For each suggestion, additional columns in the report provide useful information about how the suggestion should work. The details of these columns are shown in the following table.

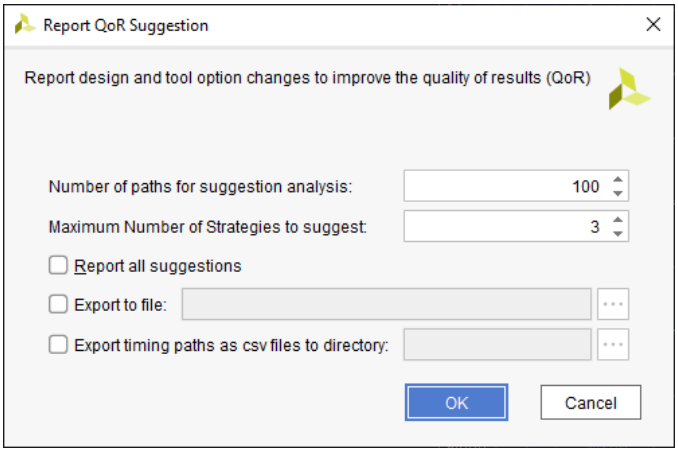
Table 16: **Additional Information Columns**

Attribute	Values	Description
GENERATED_AT	Design stage (for example, `opt_design`)	Stage at which the suggestion was generated.
APPLICABLE_FOR	Design stage (for example, `opt_design`)	Stage which must be rerun with the suggestion enabled.
SOURCE	An RQS file, or `current_run` if the suggestion was generated in the current run.	Where the suggestion source is.
AUTOMATIC	Yes, No	Describes if Vivado tools can automatically execute the suggestion or it is a manual suggestion.
SUGGESTION_SCOPE	GLOBALSCOPE, OOC top module	Used to enable OOC synthesis to automatically scope synthesis suggestions.

Generating the QoR Suggestion Report

The `report_qor_suggestions` command can be accessed in the AMD Vivado™ IDE using the Report QoR Suggestions option from the Reports pulldown menu.

Figure 166: **Report QoR Suggestions Dialog Box**



The equivalent command at the Tcl Console is as follows:

```
report_qor_suggestions -name qor_suggestions_1
```

To change the timing path limit from the default of 100, change the **Number of paths for suggestion analysis** shown in the dialog box. This will expand the number of suggestions but they will be on timing paths that might not have been optimized. The equivalent Tcl command-line option is as follows:

```
-max_paths <N>
```

To change the number of ML Strategies generated, change the **Maximum Number of Strategies to suggest** shown in the dialog box. The equivalent Tcl command-line option is as follows:

```
-max_strategies <N>
```

To expand the analysis to report suggestions that do not violate threshold criteria, select the **Report all suggestions** checkbox. The behavior is as follows:

- **Timing suggestions:** Offer suggestions on timing paths regardless of whether timing is met.
- **Utilization suggestions:** Offer suggestions on a resource that is not critical.
- **Congestion suggestions:** Offer suggestions on timing met designs at post-route stage.

The equivalent Tcl command-line option is as follows:

```
-report_all_suggestions
```

To generate supporting CSV files showing failing timing paths associated with their suggestions, check the box and specify a directory. The CSV files can make the navigation of timing paths significantly easier to manage than the table in the text report. A second file containing a DONT_TOUCH report is also generated. The equivalent Tcl command-line option is as follows:

```
-csv_output_dir <directory>
```

***Note*:** DONT_TOUCH properties prevent the tools from optimizing paths and can be added through the use of other properties automatically by Vivado. Removing DONT_TOUCH properties should be done with care. For instance, the DFX flow uses DONT_TOUCH to prevent cross boundary optimizations between the static and reconfigurable module, so should not be removed. A DONT_TOUCH property added as a result of MARK_DEBUG, by contrast, is not critical to the flow but means that the signal is not available for hardware probing if optimized.

Other Command Options

- **`-of_objects <suggestion objects>`:** Enables reporting of specific suggestions. When run in this mode, `report_qor_suggestions` does not generate new suggestions. This command executes quickly and can be used to see what suggestions are in an RQS file after it has been read. An example of its use is as follows:

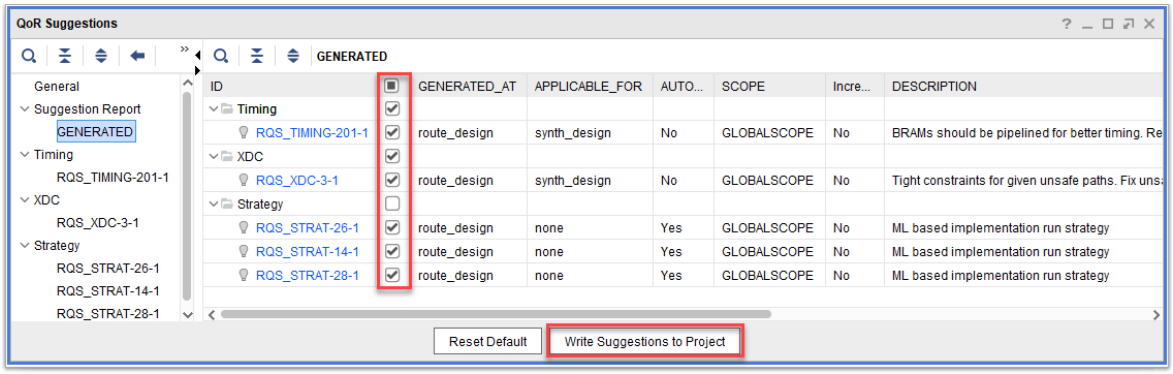
  ```
  report_qor_suggestions -of_objects [get_qor_suggestions <objectNames>]
  ```

- **`-cells <cellName>`:** Changes the top level cell for the analysis performed. The default value is the top level of the design.

Writing the Suggestion Object File

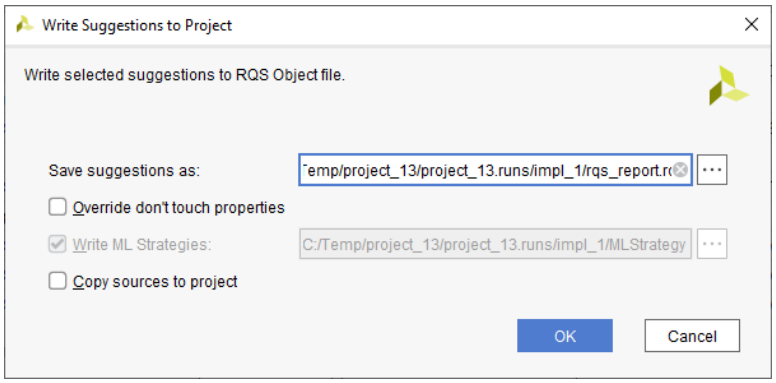
After the QoR suggestion report has been created, an RQS (`.rqs`) file containing the suggestions must be generated that can be fed into the suggestion run. To do this, select the suggestions to be included in the run and then write the QoR suggestion file. This is illustrated in the following figure.

Figure 167: **Select/Write Suggestions**



The following figure shows the dialog box when writing suggestions.

Figure 168: **Write Suggestions to Project Dialog Box**



The equivalent Tcl command is as follows:

```
write_qor_suggestions filename.rqs
```

Some suggestions require you to authorize them by acknowledging that DONT_TOUCH properties must be overridden. The equivalent Tcl option is as follows:

```
-disable_dont_touch
```

Strategy suggestions are handled differently to standard suggestions. Writing ML strategies allows the creation of multiple strategy RQS files. For more details, refer to Strategy Suggestions. The equivalent Tcl option is as follows:

```
-strategy_dir <directory>
```

Report Design Analysis

The Design Analysis report provides information on timing path characteristics, design interconnect complexity, and congestion. You can use this information to make design or constraint changes that improve QoR and possibly alleviate routing congestion.

Running Report Design Analysis

You can run Report Design Analysis from the Tcl Console or the Vivado IDE. Report Design Analysis generates three categories of reports:

- **Timing:** Reports timing and physical characteristics of timing paths.
- **Complexity:** Analyzes the design for routing complexity and LUT distribution.
- **Congestion:** Analyzes the design for routing congestion.

To run Report Design Analysis in the Vivado IDE, select **Reports → Report Design Analysis**.

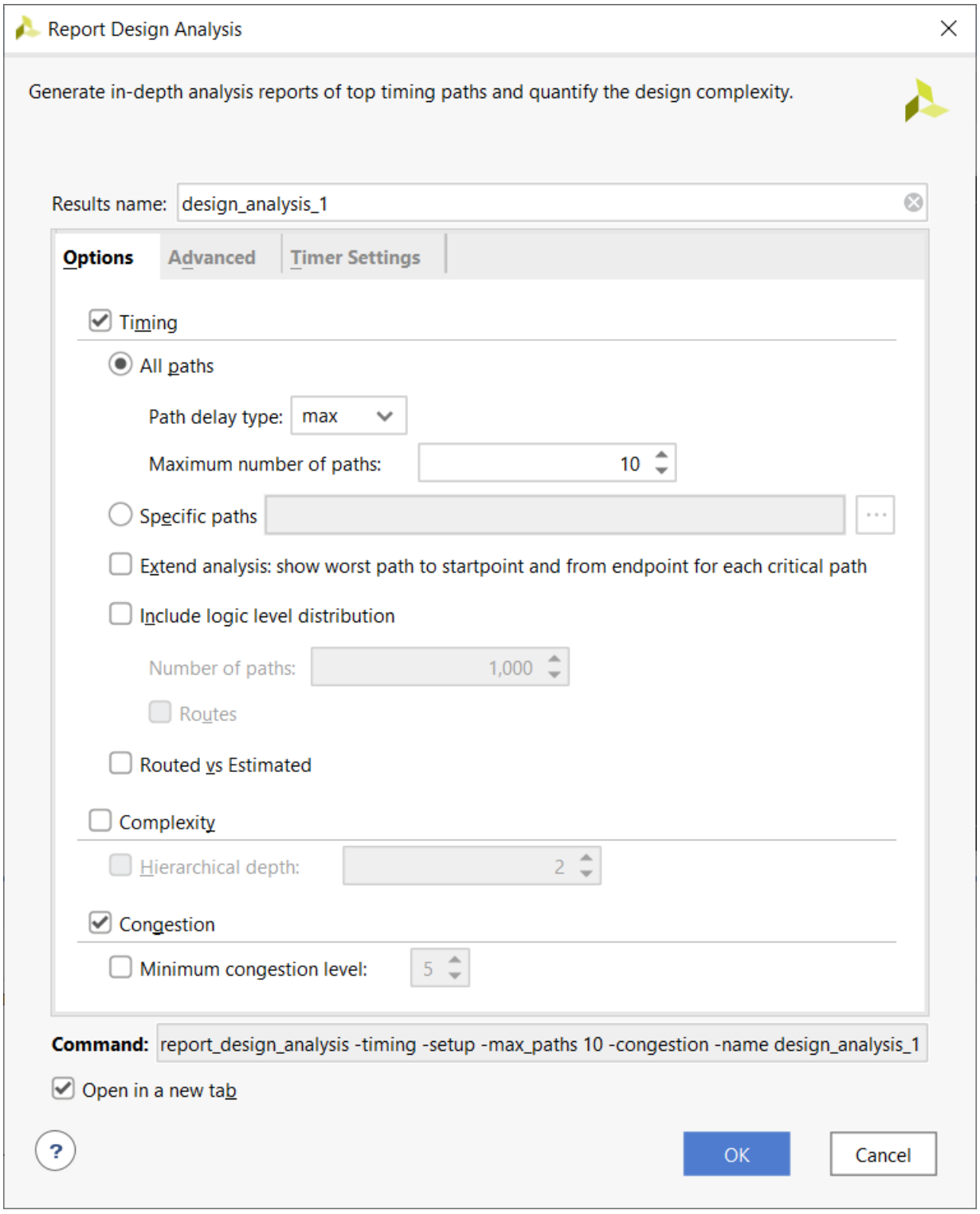
Equivalent Tcl command: `report_design_analysis -name design_analysis_1`

***Note*:** There are some Report Design Analysis options that are only available when running the `report_design_analysis` Tcl command. You can use the `-name` option to view the results of this Tcl command in the GUI.

In the Vivado IDE, the Report Design Analysis dialog box (shown in the following figure) includes the following:

- Results Name Field
- Options Tab
- Advanced Tab
- Timer Settings Tab

Figure 169: **Report Design Analysis Dialog Box**



Results Name Field

In the Results Name field at the top of the Report Design Analysis dialog box, specify the name of the graphical window for the report.

Equivalent Tcl option: `-name <windowName>`

Options Tab

In the Options tab (shown in the previous figure), the following fields are available:

- Timing
- Complexity
- Congestion

Timing Field

The Timing field allows you to report timing and physical characteristics of timing paths.

Equivalent Tcl option: `-timing`

You have the option to generate reports for all paths or specific timing paths. If you select the All Paths option you can specify the path delay type: max for setup, min for hold or min_max for setup and hold.

Equivalent Tcl option: `-setup/-hold`

You can also specify the maximum number of paths per clock group (default is 10).

Equivalent Tcl option: `-max_paths <arg>`

When you select the Specific Paths option, analysis is performed on the specified path objects. Click the **Browse** button (on the right) to open a search dialog box to aid in finding path objects. For more information about `get_timing_paths`, refer to this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Equivalent Tcl option: `-of_timing_paths <args>`

Select the **Extend Analysis** option to perform an extended analysis for each path of interest by also reporting the worst path to the startpoint and the worst path from the endpoint.

Equivalent Tcl option: `-extend`

***Note*:** When running the Extend Analysis option (Tcl option `-extend`) for hold path analysis, the tool generates a report showing the setup and hold characteristics of the paths with the same start and endpoints to show if hold fixing is impacting setup timing.

Include logic-level distribution information by selecting that option and specifying the number of paths to be used. If you are also analyzing all paths, the number of paths selected overrides the maximum number of paths per clock group. If you are analyzing specific paths, logic-level distribution information is limited to the specified paths.

Equivalent Tcl option: `-logic_level_distribution -logic_level_dist_paths <arg>`

Complexity Field

The complexity field allows you to report the complexity of the design netlist, which is a measure of the connectivity density throughout the hierarchy. See Complexity Report.

Equivalent Tcl option: `-complexity`

Select the **Cells to Analyze** option in the Advanced tab to specify the hierarchical cells to use for the complexity analysis. Click the **Browse** button (on the right) to open a search dialog box to aid in finding cell objects.

Equivalent Tcl option: `-cells <args>`

When you select the hierarchical depths option, you can select the levels of hierarchy to examine. You can examine at the top level by default, or at the level of the cells specified by the `-cells` option.

Equivalent Tcl option: `-hierarchical_depth <arg>`

Congestion Field

The Congestion field toggles the `-congestion` Tcl switch ON and OFF.

Select the **Minimum congestion level** option to specify the minimum congestion to show router congestion in the design. The default minimum congestion level is 5 if not specified. The value must be between 3 and 8.

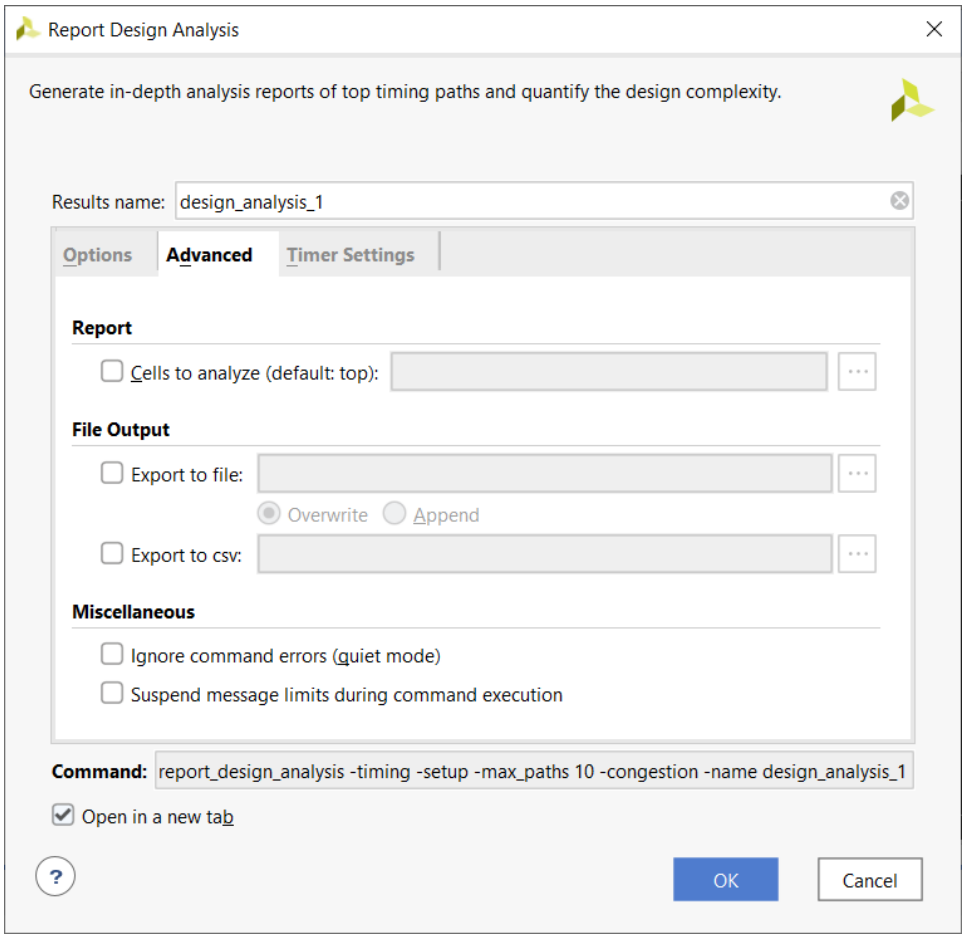
Equivalent Tcl option: `-min_congestion_level <args>`

***Note*:** The congestion report is not generated/displayed if there are no router congested regions greater than or equal to the threshold level (`-min_congestion_level`). In that case, re-run the command with a lower threshold (values between 3 and 8).

Advanced Tab

The Advanced tab is shown in the following figure.

Figure 170: **Advanced Tab**



The following fields are available in the Advanced tab:

- **Report:** Select the **Cells to Analyze** option in the Advanced tab to specify the hierarchical cells to use. Click the **Browse** button on the right to open a search dialog box and find cell objects. This option limits timing and complexity analysis within the report.
- **File Output:** Write the results to a file in addition to generating a GUI report. Select **Export to file** and specify a file name in the field to the right. Click the **Browse** button to select a different directory.

  Equivalent Tcl option: `-file <arg>`

  Select the **Overwrite** option to overwrite an existing file with the new analysis results.

  Select **Append** to append the new results.

  Equivalent Tcl option: `-append`
- **Miscellaneous:** The Miscellaneous field provides options to ignore command errors and suspend message limits during command execution.

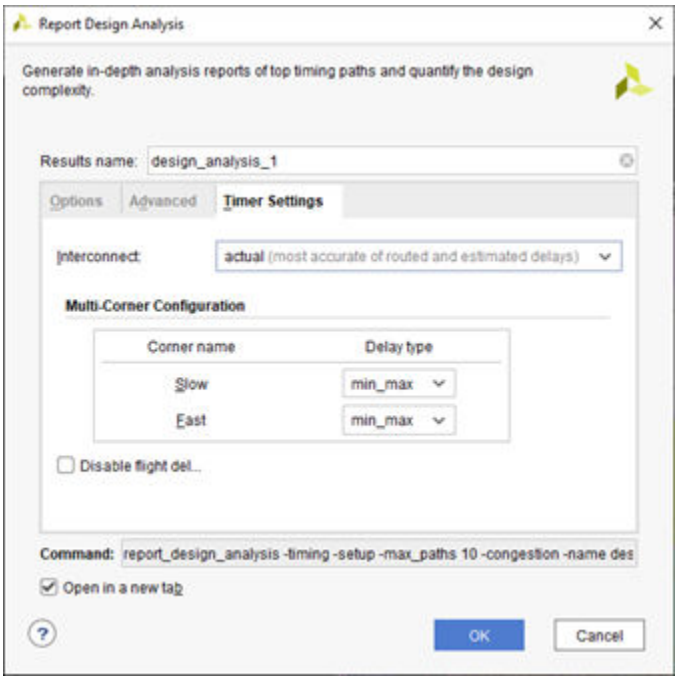
Equivalent Tcl option: `-quiet/-verbose`

Timer Settings Tab

In the timer settings tab (shown in the following figure), the following fields and options are available.

- Interconnect Option
- Multi-Corner Configuration Field
- Disable Flight Delays Option

Figure 171: **Time Settings Tab**



Interconnect Option

You can select the interconnect model to be used in your analysis of timing paths:

- **actual:** This model provides the most accurate delays for a routed design.
- **estimated:** This model includes an estimate of the interconnect delays based on the placement and connectivity of the design onto the device prior to implementation. Estimated delay can be specified even if the design is fully routed.
- **none:** No interconnect delay in the timing analysis is included. Only the logic delay is applied. This can help identify paths where the logic delay exceeds or takes up a significant percentage of the timing path requirement.

Equivalent Tcl command:

```
set_delay_model -interconnect <arg>
```

For more information about `set_delay_model`, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Multi-Corner Configuration Field

You can limit the default four-corner analysis performed by the Vivado timing analysis engine, as appropriate, using the options available in this field.

Equivalent Tcl command: `config_timing_corners -corner <arg> -delay_type <arg>`

For more information about `config_timing_corners`, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

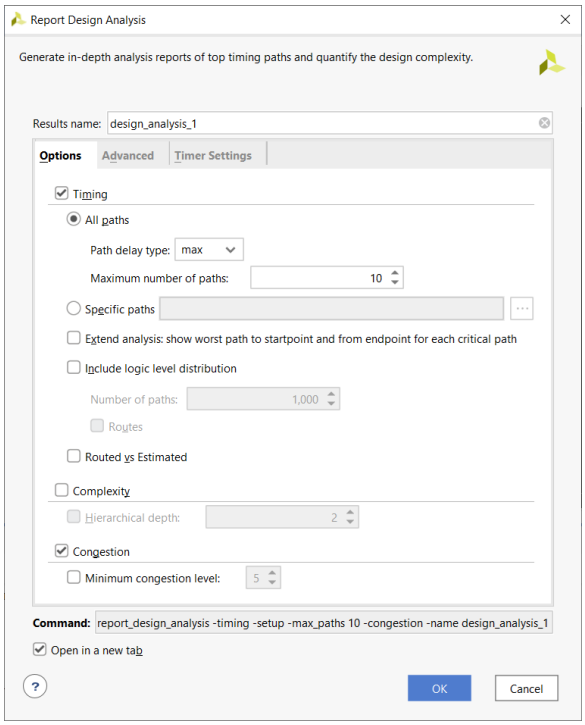
Disable Flight Delays Option

You can select this option to disable the addition of package delays to I/O timing calculations.

Equivalent Tcl command: `config_timing_analysis -disable_flight_delays <arg>`

For more information about `config_timing_analysis`, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Figure 172: **Report Design Analysis, Timer Settings Tab**



Command Line Only Options

The following timing options are only available from the Tcl command line and can be used with the `-name` option to generate a GUI report.

- **`csv <filename>.csv`:** Generates a CSV file with timing path options. This can be useful when sorting through a large number of paths.
- **`-routed_vs_estimated`:** This option reports the estimated versus actual routed delays side-by-side for the same path. Some fields within the Timing Category in the report are prefaced with "Estimated" or "Routed" for comparison.
- **`-return_timing_paths`:** Returns timing path objects to enable the further analysis of paths of specific logic levels within a specified clock domain. Must be used with both the `-end_point_clock` and `-logic_levels` options.
- **`-end_point_clock <arg>`:** Used to limit the logic level distribution report section to the timing paths with the specified end point clock.
- **`-logic_levels <arg>`:** Can limit the timing paths sent to the logic level histogram algorithm by logic levels. Only a single value can be specified.
- **`-min_level <arg>`:** Groups all timing paths with logic levels, or routes, less than the value specified into a single bin. When specified, the <arg> value passed must be at least 1.

- `-max_level <arg>`: Group all timing paths with logic levels, or routes, greater than the value specified into a single bin. When specified, the <arg> value must be greater than the value of -min_level.

The following complexity options are only available from the command line and can be used with the `-name` option to generate a GUI report.

- `-bounding_boxes <arg>`: This option performs the complexity analysis of the specified bounding boxes. For example:

```
-bounding_boxes { "CLE_M_X21Y239:CLEL_R_X28Y254"
"CLEL_R_X18Y171:CLE_M_X26Y186" }
```

Note: A space is required between the open bracket `{` and the start of the bounding box, as shown in the previous example.

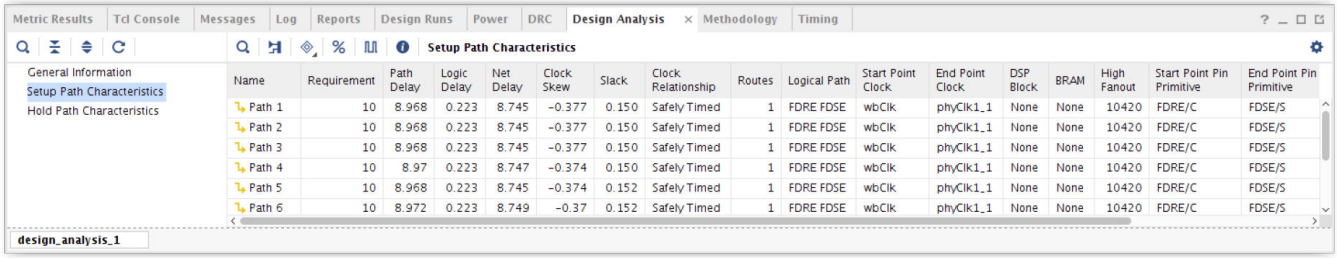
Timing Path Characteristics Report

The following figure shows example output after running the Report Design Analysis in Timing Mode to show the path characteristics of only the ten worst setup paths in the design. Generate the report from the Vivado IDE by selecting **Reports → Report Design Analysis**, or use the following Tcl command:

```
report_design_analysis -name <arg>
```

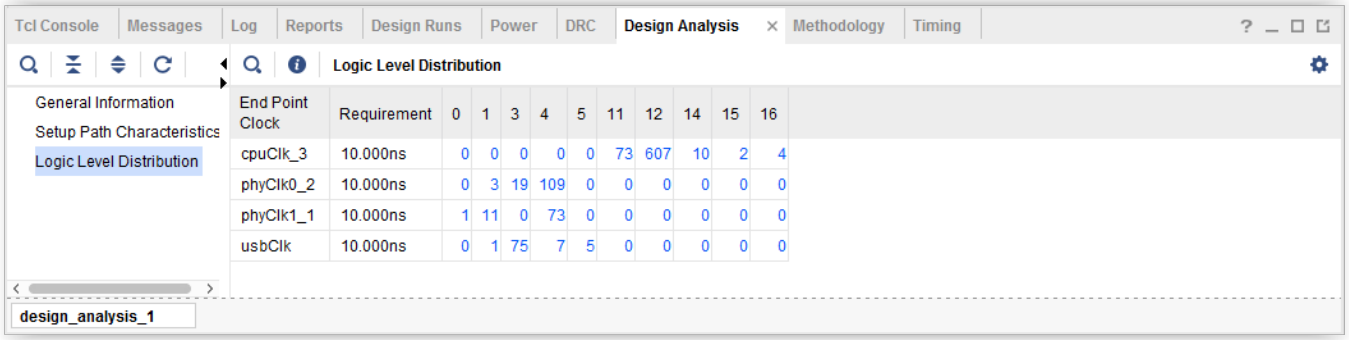
TIP: *To create hold path characteristics, select* ***Path delay type: min*** *in the Options tab of the Report Design Analysis dialog box or add* `-hold` *to the Tcl command. For more information on Tcl command syntax, see the Vivado Design Suite Tcl Command Reference Guide (UG835) .*

Figure 173: **Example Setup Path Characteristics**



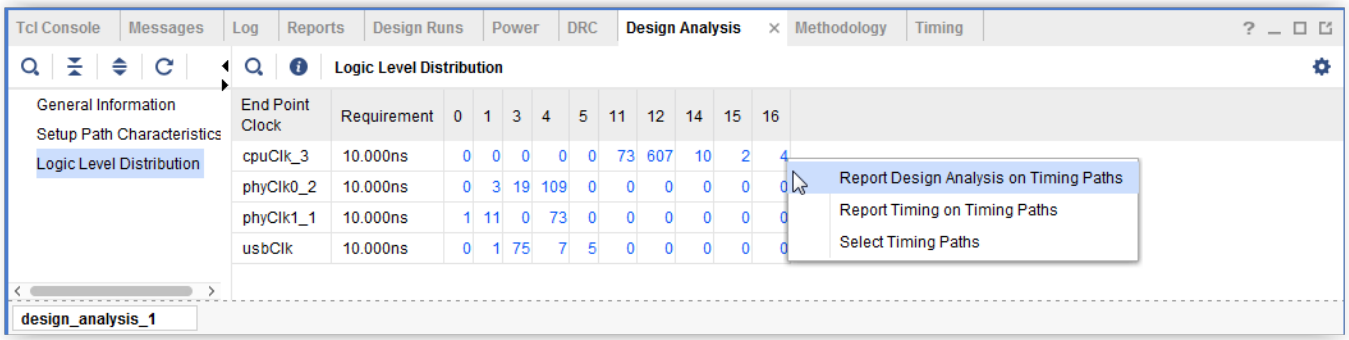
Report Design Analysis can also provide a Logic Level Distribution table for the worst timing paths. The default number of paths analyzed for the Logic Level Distribution table is 1,000 and can be changed in the Report Design Analysis dialog box. The Logic Level Distribution table is not generated by default but can be generated when you select **Include logic level distribution** in the Report Design Analysis dialog box Options tab. An example of the Logic Level Distribution table is shown in the following figure.

Figure 174: **Example of Logic Level Distribution Report**



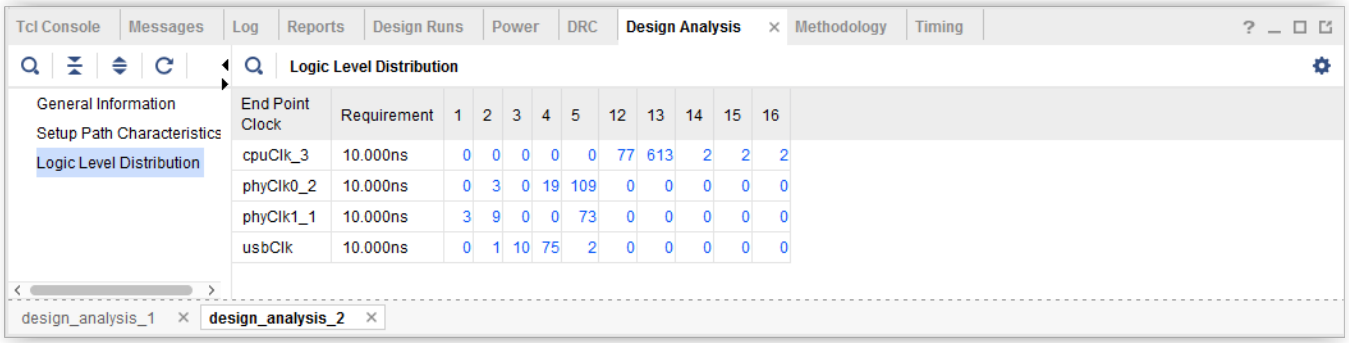
The logic level distribution GUI has been enhanced to include hyperlinks for the individual bins. By clicking on these hyperlinks, you can run `report_design_analysis` or `report_timing` on paths or select timing path objects as shown in the following figure.

Figure 175: **Report Design Analysis on a Selected Path**



The command line option `-routes` can be used with `-logic_level_distribution` to generate a report based on the number of routes instead of the number of logic levels.

Figure 176: **Example of Logic Level Distribution Report with -routes**

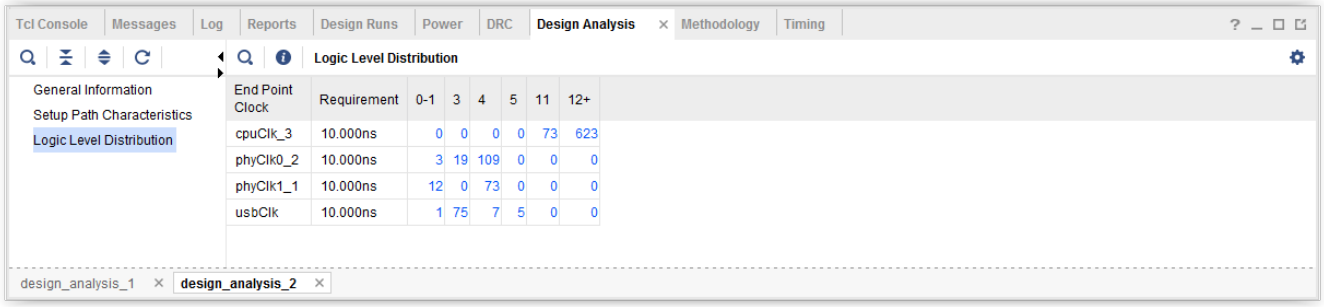


The command line options `-min_level` and `-max_level` can be used with `-logic_level_distribution` to control the bins.

All the paths with logic levels less than `-min_level` are placed in a single bin, and all the paths with logic levels greater than `-max_level` are placed in a single bin.

Create an individual bin for each logic level where at least one path exists in between the levels. For example, if a design has paths with logic levels of 0, 1, 3, 4, 5, 11, 12, 14, 15, 16 (see Timing Path Characteristics Report) using `-min_level` 3 and `-max_level` 11, `report_design_analysis` reports using the 0-2, 3, 4, 5, 11, 12+ bins.

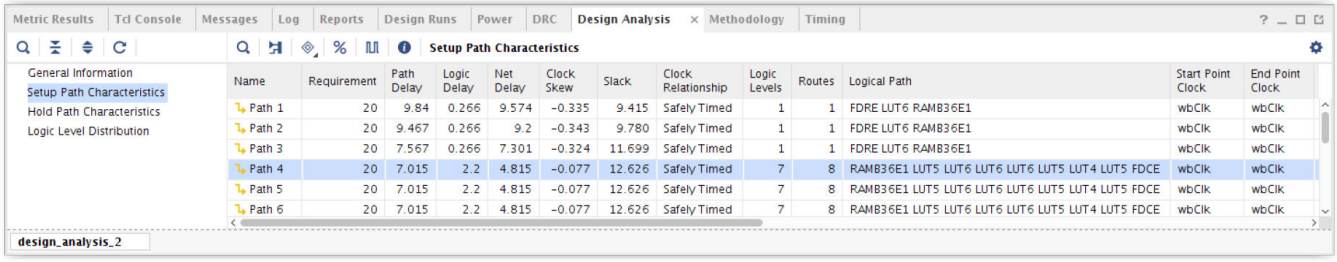
Figure 177: **Example of Logic Level Distribution Report with -min_level and -max_level**



Analyzing Specific Paths

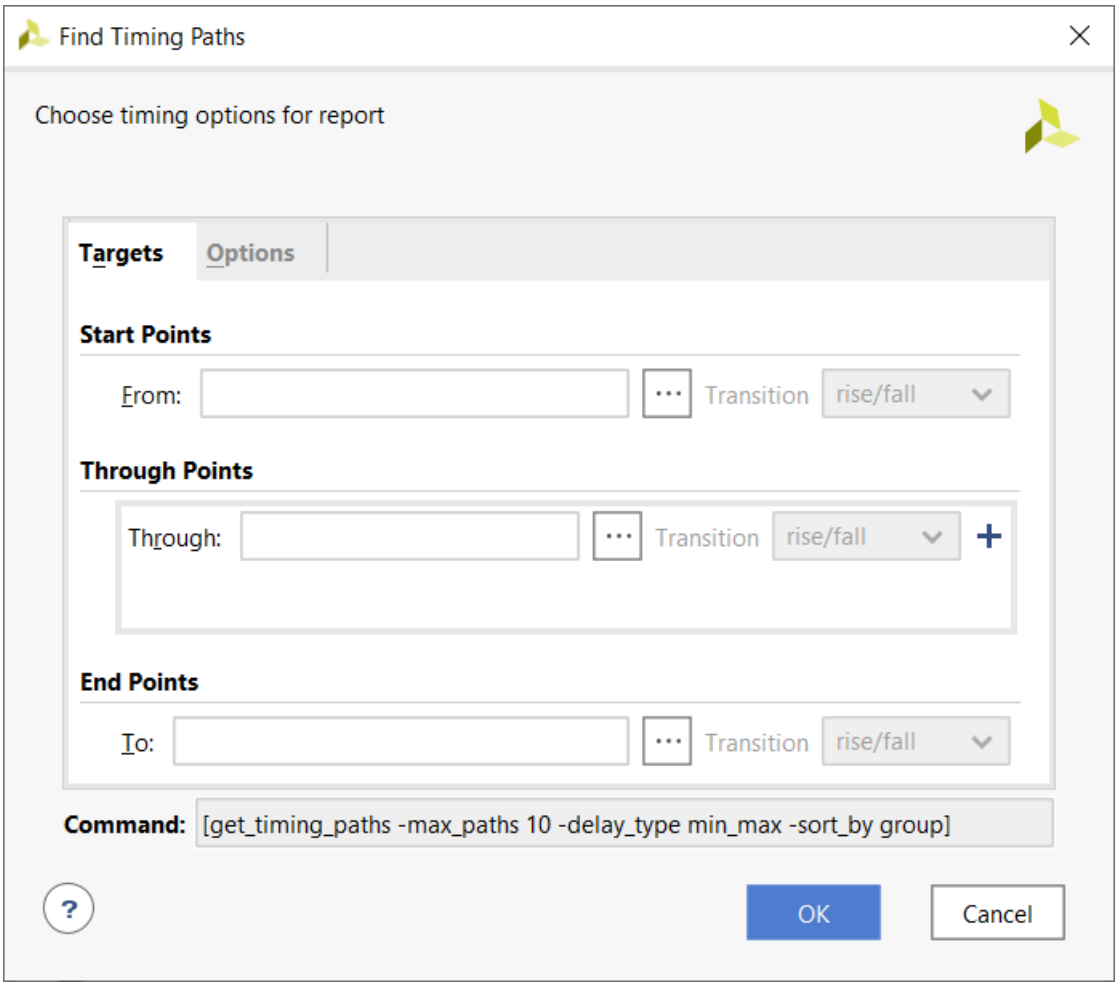
Analyzing Specific Paths shows an example report from Report Design Analysis in Timing Mode with specific paths selected.

Figure 178: **Example of Specific Timing Path Characteristics**



In this case, the Path Characteristics and the Logic Level Distribution tables (if selected) are limited to the specified path. To specify the paths, click the **Browse** button to the right of the Specific paths selection in the Report Design Analysis dialog box. This opens the Find Timing Paths dialog box (shown in the following figure).

Figure 179: **Find Timing Paths Dialog Box**



Analyzing the Worst Path along with Preceding and Following Worst Paths

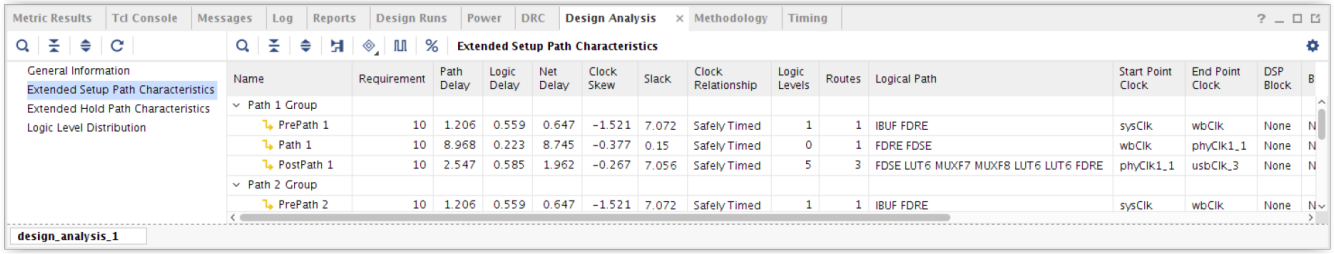
The figure below shows an example report from Report Design Analysis in Timing Mode with the Extend analysis option selected.

Note: The Extend Analysis for All Paths option is currently only available for setup analysis.

The Path Characteristics are reported on the worst setup path along with the worst setup path to the startpoint cell (`PrePath`) and the worst setup path from the endpoint cell (`PostPath`). The `-extend` option incurs higher runtime as several timing analyses are required to collect the characteristics of all reported paths.

Equivalent Tcl Command: `report_design_analysis -extend`

Figure 180: **Extended Path Characteristics of the Worst Setup Path**



Name	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes	Logical Path	Start Point Clock	End Point Clock	DSP Block
Path 1 Group													
PrePath 1	10	1.206	0.559	0.647	-1.521	7.072	Safely Timed	1	1	IBUF FDRE	sysClk	wbClk	None
Path 1	10	8.968	0.223	8.745	-0.377	0.15	Safely Timed	0	1	FDRE FDSE	wbClk	phyClk1_1	None
PostPath 1	10	2.547	0.585	1.962	-0.267	7.056	Safely Timed	5	3	FDSE LUT6 MUXF7 MUXF8 LUT6 LUT6 FDRE	phyClk1_1	usbClk_3	None
Path 2 Group													
PrePath 2	10	1.206	0.559	0.647	-1.521	7.072	Safely Timed	1	1	IBUF FDRE	sysClk	wbClk	None

Reading and Interpreting Timing Path Characteristics Reports

The path characteristics fall into four main categories: timing, logic, physical, and property. You can find the definition of each characteristics in the command long help.

Tcl Command: `report_design_analysis -help`

Alternatively, you can find the same information in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Category 1: Timing

- Timing Analysis: The Path Type and Requirement detail the timing analysis type (SETUP or HOLD) along with the timing path requirement. The Slack indicates whether or not the timing path requirement is met based on the timing analysis as dictated by the timing constraints. The Timing Exception indicates if any timing exceptions such as multicycle path or max delay have been applied to the timing path.

  Checking the path requirement is often the first step in debugging missing or incorrect timing constraints:

  - Paths with setup requirement under 4 ns must be reviewed to verify their validity in the design, especially for clock domain crossing paths.
  - Paths with setup requirement under 2 ns are difficult to meet and must be avoided in general, especially for the older architectures.
  - In general, when small setup requirements are present, check for missing timing exception constraints and also check the source and destination clock edges. The timing analysis always assumes the smallest positive difference between source and destination clock edges unless overridden by a timing exception constraint.
  - Positive hold path requirements need to be reviewed as they are not common and are difficult to meet. When positive hold path requirements are present, check for missing multicycle path constraints for hold analysis that might have only been applied to the path for setup analysis. Also check the relationship between source and destination clocks for correctness.

- Datapath: The Path Delay, Logic Delay, and Net Delay detail the total datapath delay along with its breakdown into delay contribution by logic cells and nets.
  - If the Logic Delay makes up an unusually high proportion of the total datapath delay, for example 50% or higher, it is advised to examine the datapath logic depth and types of cells on the logic path, and possibly modify the RTL or synthesis options to reduce the path depth or use cells with faster delays.
  - If the Net Delay dominates the total path delay for a setup path where the Requirement is reasonable, it is advised to analyze some of the physical characteristics and property characteristics of the path listed in this section. Specific items to look at include the High Fanout and Cumulative Fanout characteristics to understand if some nets of the path have a high fanout that could potentially be causing a placement problem. Also check the Hold Fix Detour characteristic to understand if hold fixing has occurred on the path.



IMPORTANT! *The LUT input pins have different delay characteristics. The physical pins (or site pins) of higher index are faster than the pins of lower index. Be aware of the difference in 7 series and UltraScale device LUT delay reporting. In 7 Series devices, the variable portion of LUT delay is reported as part of the net delay in front of the LUT. In UltraScale devices, the variable portion of LUT delay is reported as logic delay. Therefore, the 7 Series device* `Net Delay/Logic Delay` *ratio will be larger than the ratio for UltraScale devices.*

- Clocks: The Start Point Clock, End Point Clock, Clock Relationship, and Clock Skew detail information regarding the timing path clocks. The Start Point Clock and Endpoint Clock list the respective source clock and destination clock for the timing path.
  - Check that the Clock Relationship is correct and expected. For intra-clock paths or synchronous clock domain crossing paths, the relationship is labeled as "Safely Timed." You must verify that the Requirement and Clock Skew are reasonable. For asynchronous clocks, the relationship is labeled as "No Common Primary Clock", "No Common Period", "No Common Node", or "No Common Phase". Asynchronous clock domain crossing paths must be covered by timing exceptions (check the Timing Exception value).
  - Check that the Clock Skew is reasonable. When analyzing clock skew, check the clock tree structure for cascaded clock buffers. In 7 series devices, check for different clock buffer types for the source and destination clocks. In UltraScale devices, it might be necessary to examine the placement and routing of the clock nets because it depends on logic loads placement. The crossing of a Clock Region boundary or an I/O Column can result in higher clock skew; this is expected.

***Note*:** Almost all of the Timing Characteristics provided by `report_design_analysis` are available in a timing report.

Category 2: Logic

- Path: The Start Point Pin Primitive, End Point Pin Primitive, Start Point Pin, End Point Pin, Logic Levels, Logical Path, and Routes provide some basic information about the timing path.
  - The Start Point Pin Primitive and End Point Pin Primitive are the reference pin names of the timing path start point and end point. Check that the Start Point Pin Primitive and End Point Pin Primitive are expected timing path start and endpoints. The Start Point Pin and End Point Pin identify the actual timing path pin startpoints and endpoints that would show in the header of a typical timing report.

  Check for endpoint pins such as `CLR`, `PRE`, `RST`, and `CE` that could potentially be part of high-fanout nets for control signals such as asynchronous resets and clock enable signals. Also check the type of cell, because some primitives like block RAMs and DSPs have larger Clock-to-Q delay and setup/hold requirements than other cells. Their presence in the path can potentially consume a significant portion of the path timing budget.

  - The Logic Levels and Logical Path detail the number of logic levels and the types of primitives in the datapath. Routes indicates the number of routable nets in the datapath. You can use this information to quickly check if a high number of logic levels is mostly due to `LUT`s or to a mix of `LUT`/`CARRY`/`MUXF` cells. `CARRY` and `MUXF` cells are usually connected to nets with dedicated routes that have null or very small delays, while LUT inputs always need to be routed through the fabric.

  When the path mostly contains LUTs, it is also important to check their size. Try to understand why there are several smaller LUTs (non-LUT6) that are chained and what prevents synthesis from targeting LUT6 only, which can reduce the logic levels. There can be properties like `KEEP`/`DONT_TOUCH`/`MARK_DEBUG` or mid-to-high fanout nets in the path that also impact mapping efficiency.

  Based on the outcome of your analysis, you can either modify the RTL source, add/modify attributes in the RTL, or use different synthesis settings to reduce the number of LUTs on the path. Also, you can use the option `-remap` of the `opt_design` command to re-optimize LUT mapping and possibly eliminate some smaller LUTs.

- Cells: Presence of DSP block(s) and BRAM(s) in the datapath. Timing is more difficult to meet on paths from RAMBs or DSPs with no output registers and with several logic levels. You should consider modifying your design to use the RAMB or DSP output registers if these paths are having difficulty meeting the timing requirements.

Category 3: Physical

- **Architectural Boundary Crossings:** The IO Crossings and SLR Crossings identify whether the path is crossing architectural resources such as IO Columns or SLR boundaries.

  The crossing of many architectural columns does not always represent a problem. Check for high net delay or large skew in conjunction with the crossing of many architectural columns. If many architectural column crossings appear to be the cause of timing issues across multiple implementation runs for a particular module, consider minimal floorplanning using Pblocks to reduce the crossings of the architectural column(s) or SLR boundary.

- **Path Placement Restrictions: Pblocks:** Excessive floorplanning can sometimes prevent the tool from achieving the optimal results. Paths that cross multiple Pblocks can sometimes experience timing issues.
  - If the path crosses multiple Pblocks, examine the location of the Pblocks and the impact on the timing path placement.
  - If the Pblocks are adjacent, consider creating a single Pblock that is a super-set of each individual Pblock. This could potentially improve timing by being less restrictive on the placer.

  If physical requirements dictate that the Pblocks are placed far apart, consider pipelining between the Pblocks to help meet timing requirements.
- **Placement Box: Bounding Box Size, Clock Region Distance, Combined LUT Pairs:** If the Bounding Box Size or Clock Region Distance of the timing path is too large, try using directives in `place_design`. In UltraScale devices, be especially aware of the Clock Region Distance and its possible impact on timing path Clock Skew.
- **Net Fanout and Detour:**
  - High Fanout shows the highest fanout of all nets in the datapath, and Cumulative Fanout corresponds to the sum of all datapath net fanouts.

    If High Fanout and Cumulative Fanout are large, the timing violations are very likely due to the fanout impact on routing and net delay.

    If physical optimization was run and did not reduce the fanout, check for `MARK_DEBUG` and `DONT_TOUCH` constraints preventing replication.

    If replication is desired on the net prior to implementation, you can use the `MAX_FANOUT` constraint in synthesis, either inside the RTL or in an XDC file. Due to reliance on placement for good timing for high fanout nets, it is usually not recommended to have synthesis perform replication and it is best to rely on post-placement physical optimization (`phys_opt_design`) for replication. You can also increase the physical optimization effort to also optimize paths with a small positive slack by using different directives such as `Explore`, `AggressiveExplore`, or `AggressiveFanoutOpt`.

    If fanout reduction is desired on a specific net during implementation, you can force the replication using the command: `phys_opt_design -force_replication_on_nets <netName>`
  - When the Hold Fix Detour is asserted, the routing on the datapath is delayed to meet the path hold time requirement. If the path is failing setup, check for excessive skew between the source and destination clocks. Check also for proper timing constraints between the source and destination clocks in case the hold path requirement is positive (it should be zero or negative in most common cases).

    Note: The `HOLD_DETOUR` attribute is set during specific hold fixing stages in the router. Other phases might have an impact on hold timing but the property will not be set; for example, if there is a routing detour due to congestion.

Category 4: Property

- LUT Combining: Combined LUT Pairs indicates that there are combined LUT pairs present in the path. While combining LUT pairs can reduce logic utilization, it can also restrict the placement solutions and can create congestion due to high pin density. If LUT combining appears to be an issue in the design, it is recommended to disable LUT combining in synthesis by using the `-no_lc` option.
- Optimization Blocking: Mark Debug and Dont Touch can quickly identify whether there are any nets or cells in the path that the tool is not allowed optimize.
  - The default behavior of setting the `MARK_DEBUG` property is to also to set the `DONT_TOUCH` property. Consider setting `DONT_TOUCH` to `FALSE` to allow for optimization.
  - `DONT_TOUCH` disables optimizations such as cell or net replication. Evaluate the need for `DONT_TOUCH` constraints and remove them if possible. When a net enters a hierarchical cell with `DONT_TOUCH`, the portion of the net inside the hierarchical cell cannot be replicated. If `DONT_TOUCH` is used to prevent logic trimming, check the design for correctness. One simple example would be logic removed due to unconnected outputs.
- Fixed Placement and Routing: The Fixed Loc, Fixed Route can quickly identify whether there are any fixed placement or fixed routing constraints that might be impacting the timing path slack.
  - Using cell location constraints can help stabilizing QoR for a difficult design. If timing can no longer be met after modifying the design, you can try removing the placement constraints to give more flexibility to the placer.
  - Having fixed routes prevents the router from optimizing the net delays to meet timing. A timing path with locked routing usually shares nets with other paths that can be negatively impacted by this constraint. Use fixed routes only when necessary and when it does not affect interacting paths. Always be aware that changes to other physical constraints such as Pblocks might require the fixed cell locations or fixed routes to also be updated.

Category 5: Dynamic Function eXchange Designs

For Dynamic Function eXchange (DFX) designs, each cell in the logical path is prefixed to identify the cell as belonging to a reconfigurable partition (RP#:), or to the static region of the design (S:).

- **DFX Path Type:** Specifies the path as being completely in the static region, completely in a reconfigurable partition (RP), or as crossing the boundary between regions. The delay elements for the timing path are also broken down between the regions.
- **DFX Boundary Nets:** Reports the number of times a timing path crosses between either a static and reconfigurable module (RM) or between two RMs in the netlist.
- **Boundary Fanout:** Reports the fanout of a boundary path at the PPLOC to its downstream loads.

Complexity Report

The complexity report shows the Rent Exponent, Average Fanout, and distribution of the types of leaf cells of the top-level design and/or of hierarchical cells that contain more than 1000 leaf cells. The Rent exponent is the relationship between the number of ports and the number of cells of a netlist partition when recursively partitioning the design with a min-cut algorithm. It is computed with similar algorithms as the ones used by the placer during global placement. Therefore it can provide a good indication of the challenges seen by the placer, especially when the hierarchy of the design matches well the physical partitions found during global placement.

The Rent Exponent is defined by the Rent's rule:

$$\text{ports} = \text{constant} \times \text{cells}^{\text{Rent}}$$

$$\log(\text{ports}) = \text{Rent} \times \log(\text{cells}) + \text{constant}$$

A design with higher Rent exponent corresponds to a design where the groups of highly connected logic also have strong connectivity with other groups. This usually translates into a higher utilization of global routing resources and an increased routing complexity. The Rent exponent provided in this report is computed on the unplaced and unrouted netlist.

After placement, the Rent exponent of the same design can differ as it is based on physical partitions instead of logical partitions. The post-placement Rent exponent is not reported by the Report Design Analysis command as it is recommended to analyze the congestion reports once the design is placed instead.

Report Design Analysis runs in Complexity Mode when you do either of the following:

- Check the Complexity option in the Report Design Analysis dialog box Options tab.
- Execute the `report_design_analysis` Tcl command and use any of the options shown in the following table.

Table 17: **Options that Run Report Design Analysis in Complexity Mode**

Tcl Option	Description
`-complexity`	Must be specified to run the report design analysis in Complexity Mode.
`-cells <arg>`	Specifies the hierarchical cells to use when analyzing the complexity.
`-hierarchical_depth <arg>`	The levels of hierarchy to examine at the top level by default or at the level of the cells specified by the `-cells` option.

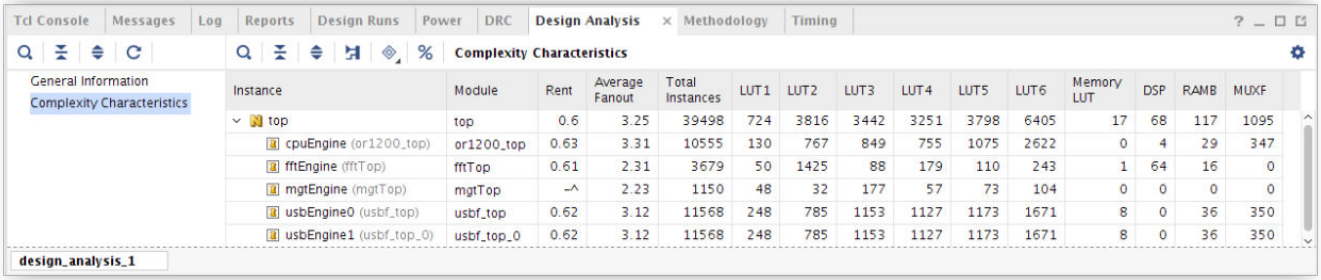
Analyzing the Design Complexity at the Top Level

The following figure shows an example report from Report Design Analysis in Complexity Mode that reports up to one level of hierarchy from the top module.

Tcl Command:

```
report_design_analysis -complexity -hierarchical_depth 1
```

Figure 181: **Complexity Analysis at the Top Level and Hierarchical Depth of 1**



Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF
top	top	0.6	3.25	39498	724	3816	3442	3251	3798	6405	17	68	117	1095
cpuEngine (or1200_top)	or1200_top	0.63	3.31	10555	130	767	849	755	1075	2622	0	4	29	347
fftEngine (fftTop)	fftTop	0.61	2.31	3679	50	1425	88	179	110	243	1	64	16	0
mgtEngine (mgtTop)	mgtTop	~^	2.23	1150	48	32	177	57	73	104	0	0	0	0
usbEngine0 (usbf_top)	usbf_top	0.62	3.12	11568	248	785	1153	1127	1173	1671	8	0	36	350
usbEngine1 (usbf_top_0)	usbf_top_0	0.62	3.12	11568	248	785	1153	1127	1173	1671	8	0	36	350

Reading and Interpreting a Complexity Report

The Complexity Characteristics table from the previous example shows the Rent exponent and average fanout for each level of hierarchy below the top level. The typical ranges to consider when reviewing these metrics are the following:

- Rent exponent:
  - Between 0.0 and 0.65: The complexity is considered low to normal and does not highlight any potential problems.
  - Between 0.65 and 0.85: The complexity is considered high, especially when the total number of instances is above 25k.
  - Above 0.85: The complexity is very high, and if the number of instances is also high, the design can potentially fail during implementation.
- Average fanout:
  - Below 4: It is considered normal.
  - Between 4 and 5: The implementation tools can show difficulty to place the design without congestion. In the case of a SSI device, if the total number of instances is above 100k, the placer can have problems finding a placement solution that fits in 1 SLR or that is spread over 2 SLRs.
  - Above 5: The design can potentially fail during implementation.

You must treat high Rent exponents and/or high average fanouts for larger modules with higher importance. Smaller modules, especially under 10k total instances, can have higher Rent exponent and average fanout, and yet be simple to place and route successfully. For this reason, the Total Instances column must always be reviewed along with the Rent exponent and average fanout.

The complexity characteristics might not always predict routing congestion. Other factors such as I/O location constraints, floorplanning, and macro primitive location in the target device can limit the placement solution space and introduce congestion. The effect of such constraints is better analyzed by the congestion reports available after placement.

Other items to consider when interpreting the Complexity Characteristics table:

- A higher percentage of LUT6s in a module usually increases the average fanout and potentially the Rent exponent.
- A high number of RAMB and DSPs can increase the Rent exponent because these primitives have a large amount of connectivity.
- The hierarchical instances with higher Rent exponents or higher average fanouts are not always a problem because the placer operates on a flat netlist and can break these instances into easier groups of logic to place. This report provides an indication of where a netlist problem can possibly exist if a module stands out clearly.

When a large module exhibits a high Rent exponent and/or average fanout that is causing congestion and timing issues, consider the following actions:

- Reduce the connectivity of the module. Preserving the hierarchy to prevent cross-boundary optimization in synthesis can reduce the use of LUT6s and consequently reduce the netlist density.
- Try to disable LUT combining in synthesis.
- Use a Congestion Strategy during Implementation or SpreadLogic placement directive that can potentially help to relieve congestion. If the design is targeting an SSI Device, consider trying several SSI placement directives.
- Use simple floorplanning at the SLR level for SSI devices, or at the clock region level in general, to keep congested groups of logic separate, or to guide global placement towards a solution similar to a previously found good placement.

Congestion Report

The Congestion reports show the congested areas of the device and the name of design modules present in these areas. Congestion can potentially lead to timing closure issues if the critical paths are placed inside or next to a congested area.

Analyzing the Design Congestion

To run Report Design Analysis in Congestion Mode, the Congestion option must be specified in the Options tab of the Report Design Analysis dialog box, and the design must be placed and/or routed. Running Report Design Analysis with Congestion Mode on an unplaced design results in nothing being reported.

Report Design Analysis produces three congestion tables:

- Placer Final Congestion Reporting
- Router Initial Congestion Reporting
- SLR Net Crossing Reporting

Maximum Congestion Reports

These tables report all the windows with the same maximum congestion level seen in a particular direction. The columns are defined as follows:

- Direction: The direction of the congested routing resources (North, South, West, or East).
- Congestion Level: The maximum congestion level in CLB tiles.
- Congestion: Indicates the estimated routing resource utilization in the defined window. This value can be greater than 100%.
- Congestion Window: Indicates the bounding CLB tiles where the congestion for the identified Direction is present. The CLB coordinates correspond to the lower left and upper right corners of the window.

  TIP: *The Congestion Window column is only available in the text report. In the GUI report, you can select the congestion window, which highlights the congested area in the Device window.*

- Cell Names: Indicates the parent instance that contains the hierarchical cells involved in the Congestion Window, up to the three largest contributors along with their contribution percentage.

  TIP: *In the GUI report, you can select the hyperlinked cell names to highlight the respective leaf cells in the congestion window.*

- Avg LUT Input: This is the average LUT inputs of the LUTs in the Window.
- COMBINED LUTs %: Indicates the percentage of LUTs combined in the window.
- LUT usage %: The percentage of LUT utilization in the Window.
- LUTRAM usage %: The percentage of LUTRAM utilization in the Window.
- Flop usage %: The percentage of FD (including LD) utilization in the Window.
- MUX usage %: The percentage of MUXF utilization in the Window.
- RAMB usage %: The percentage of RAMB utilization in the Window.
- URAM usage %: The percentage of URAM utilization in the Window.
- DSP usage %: The percentage of DSP utilization in the Window.
- CARRY usage %: The percentage of CARRY utilization in the Window.
- SRL usage %: The percentage of SRL utilization in the Window.

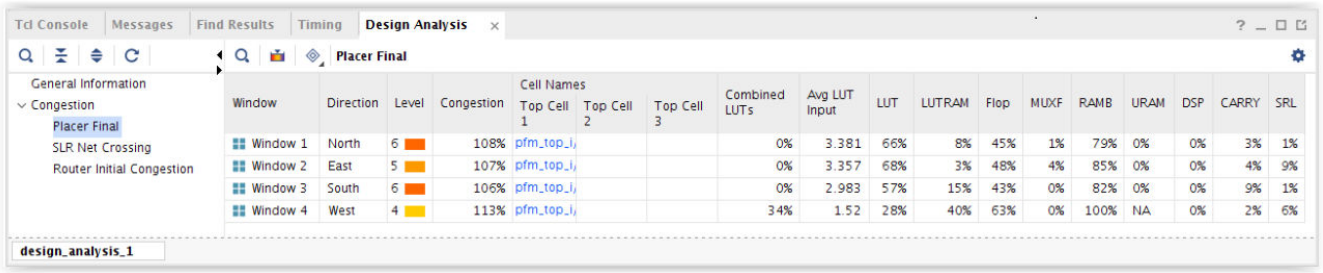
Placer Final Congestion Reporting

When analyzing the Placer Final Congestion Reporting Table of your design for Congestion and Timing QoR, look for the following:

- If a high level of LUT usage exists, examine the instances that have a high percentage of LUT6s in the Complexity report.
- In case of high RAMB or DSP utilization in the congested area, check for Pblock constraints that might be limiting the available placement area of the reported modules. Use various targeted placement directives to relieve congestion such as the BlockPlacement or SpreadLogic directives. In some cases, it might be beneficial to reuse the RAMB or DSP placement from a previous run that showed low congestion and resulted in good Timing QoR.

The following figure shows an example of the Placer Final Congestion Reporting table. Using this report, you can examine areas of the device defined by the Congestion window along with the modules residing in that window. The resource usage percentages gives an indication of the types of resources located in the congested area.

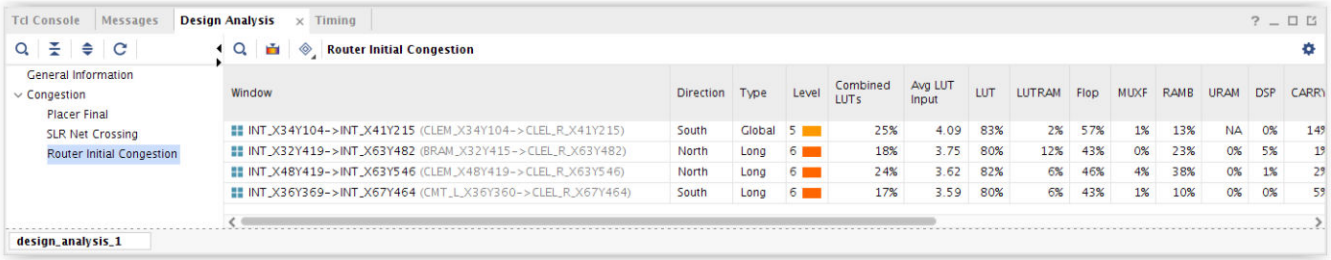
Figure 182: **Example Placer Final Congestion Reporting Table**



Router Initial Congestion Reporting

Router Initial Congestion (named Initial Estimated Router Congestion for 7 series FPGAs) is only available when the router has been run. It shows the routing congestion initially faced by the router during the early stages of routing.

Figure 183: **Example of Router Initial Congestion Reporting Table**



When the congestion level is 5 or higher, `report_design_analysis` generates a congestion table that provides details about the nature of congestion and region(s) associated with the highest congestion in a particular direction and type.

- Global congestion is estimated similar to placer congestion and is based on all types of interconnects.
- Long congestion only considers long interconnect utilization for a given direction.
- Short congestion considers all other interconnect utilization for a given direction.

Any congestion area greater than 32x32 (level 5) is likely to impact QoR and routability. Congestion on long interconnects increases usage of short interconnects and results in longer routed delays. Congestion on short interconnects usually induces longer runtimes and if their window size is big, it is also likely to cause QoR degradation.

When analyzing the Router Initial Congestion table, look for the following:

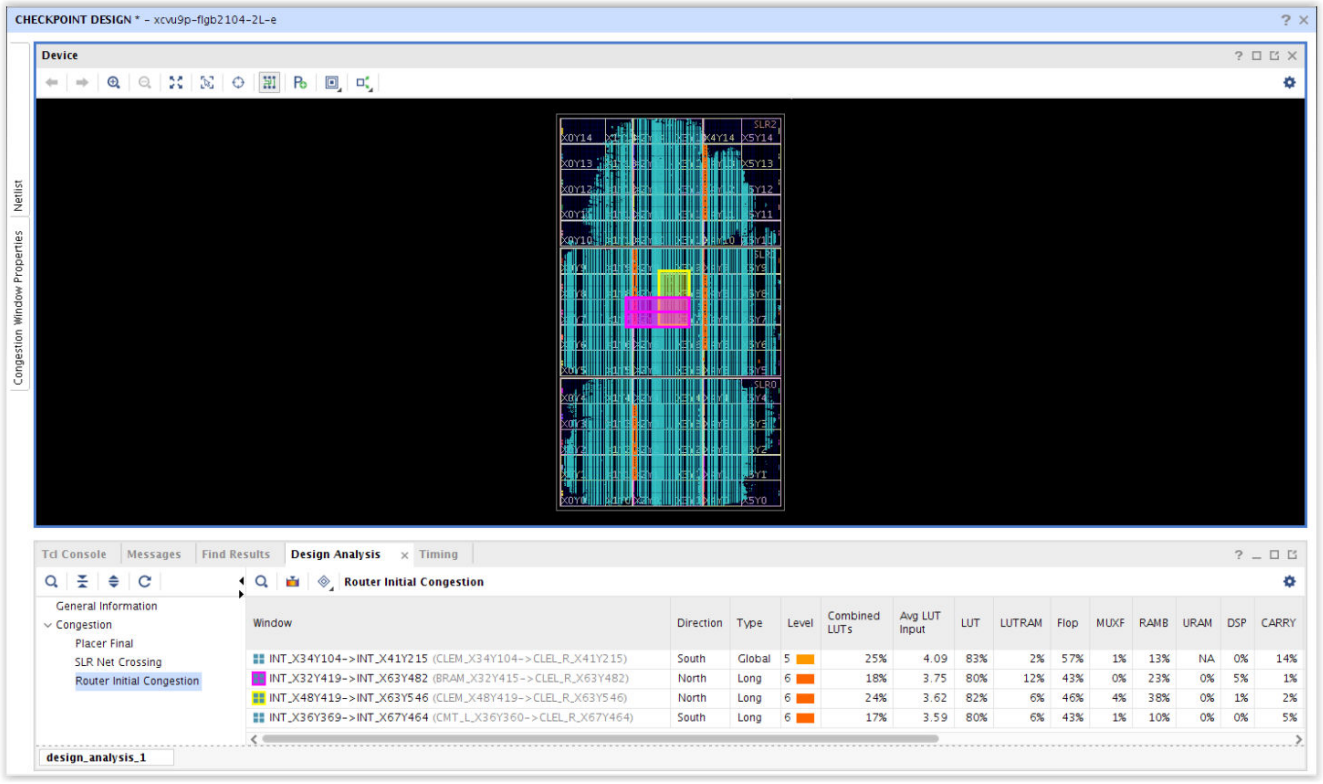
- If the congestion level is greater than 6, the design is unlikely to meet timing and might fail during routing.
- If the congestion level is 4 or 5, then identify the module(s) located in the congested area(s). You can apply a congestion alleviation technique on these modules or rerun the placer with different directives, such as *SpreadLogic*.
- If the congestion level is 3 or less, the congestion is probably not a cause for concern unless the design has a very tight timing budget.

The previous figure illustrates an example of the Router Initial Congestion, where regions with congestion level 5 or more are reported. To generate a congestion report with a lower congestion threshold, use the switch `-min_congestion_level`. The default minimum congestion level is 5. The values must be between 3 and 8.

In addition to the region where the design has maximum congestion in a given direction and type, the congestion report also contains additional regions with the maximum congestion level in that given direction and type (if any). These regions can have some amount of overlap or they can be present in different regions of the device.

The following figure illustrates an example of where the design has a congestion level 6 for North (Direction) Long (Type) in more than one region.

Figure 184: **Example of Router Initial Congestion Reporting Table**

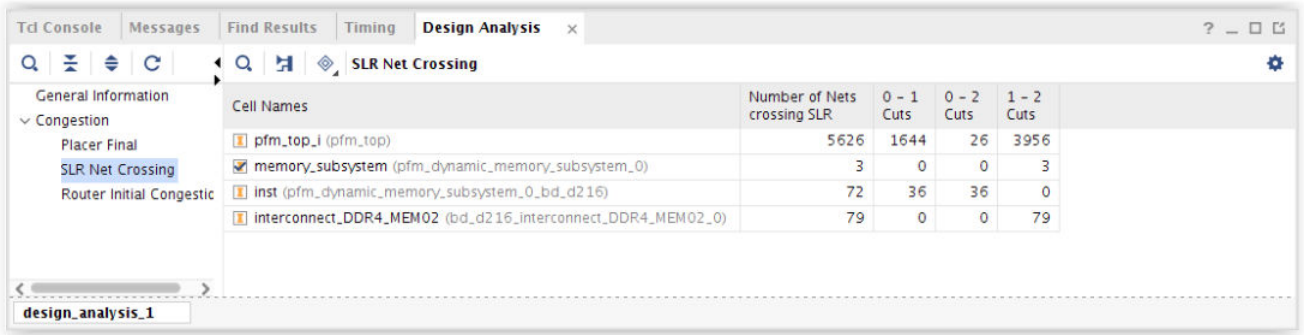


SLR Net Crossing Reporting

The SLR Net Crossing Reporting is only applicable to SSI Devices and reports the number of nets contained in a module that cross the SLR boundaries. For each module, the table provides further details of which SLRs are crossed by the nets. The following figure shows an example of the SLR Net Crossing Reporting table.

Note: When a net has loads in multiple SLRs, it is only counted once for the furthest cut. For example, a net driven from SLR0 to loads in SLR1, SLR2, and SLR3 is only counted once under the 0-3 cuts, with SLR3 being the "furthest fanout" from SLR0. This counting method enables to sum the number of nets under each column (0-1 Cuts, 1-2 Cuts, and so on) to match to total number of nets crossing, as each net is only counted once.

Figure 185: **Example SLR Net Crossing Reporting Table**



When analyzing the SLR Net Crossing Reporting Table of your design for Congestion and Timing QoR, look for the following:

1. When using SSI Devices, the SSI placement directives can be beneficial for both timing and congestion.
2. If a particular module that is crossing SLRs is consistently experiencing timing issues across multiple implementation runs using various placement directives, attempt light Pblocking to constrain the module to a single SLR.

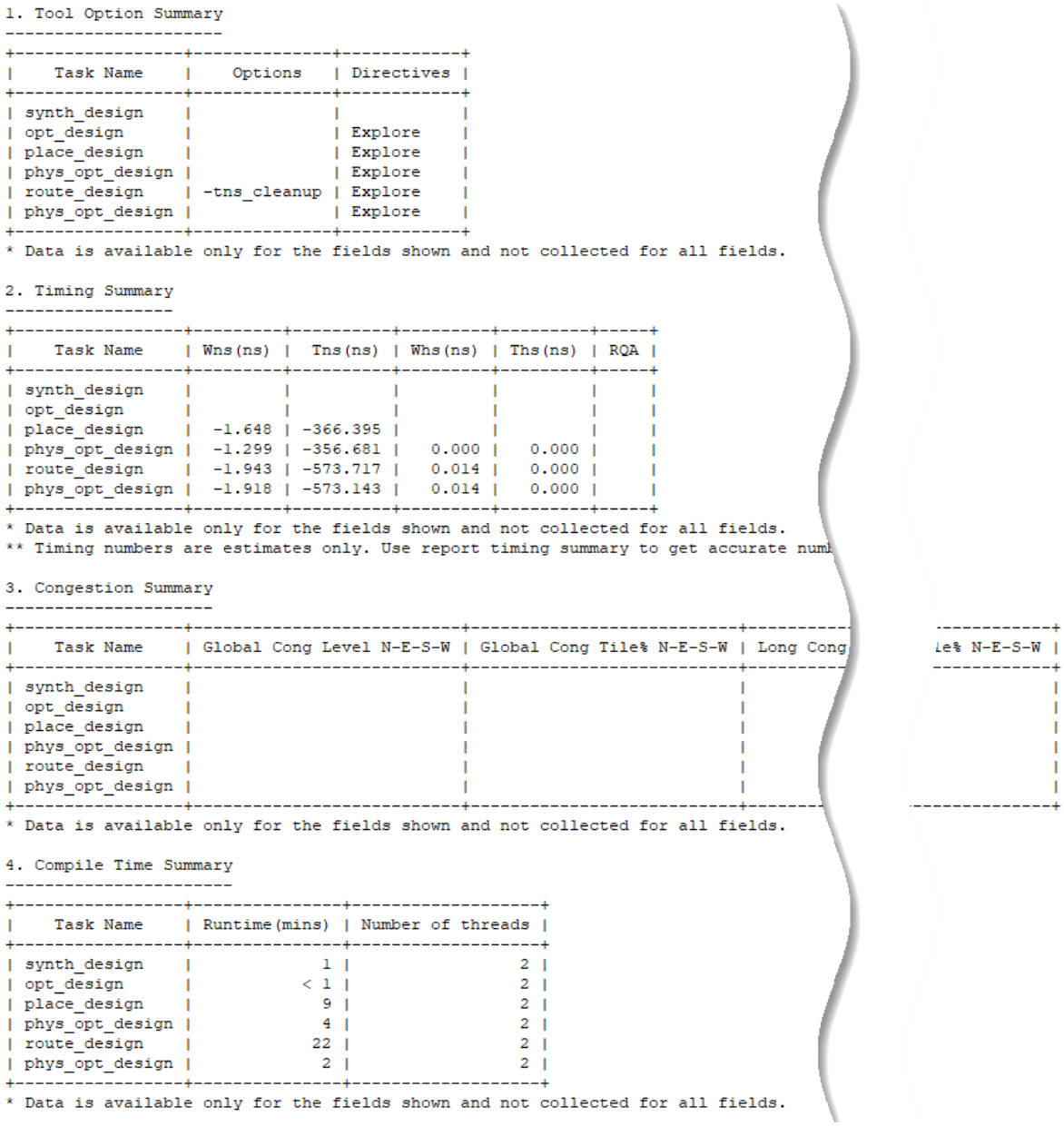
Design QoR Summary

The command-line option `-qor_summary` can be used to generate a QoR summary for each step of the flow. This option is only available from the Tcl Console. This option can be generated in two formats, either text based report or JSON format.

To generate the text-based format, run the following command:

```
report_design_analysis -qor_summary
```

Figure 186: **Report Design Analysis QoR Summary**



1. Tool Option Summary

Task Name	Options	Directives
synth_design		
opt_design		Explore
place_design		Explore
phys_opt_design		Explore
route_design	-tns_cleanup	Explore
phys_opt_design		Explore

* Data is available only for the fields shown and not collected for all fields.

2. Timing Summary

Task Name	Wns(ns)	Tns(ns)	Whs(ns)	Ths(ns)	RQA
synth_design					
opt_design					
place_design	-1.648	-366.395			
phys_opt_design	-1.299	-356.681	0.000	0.000	
route_design	-1.943	-573.717	0.014	0.000	
phys_opt_design	-1.918	-573.143	0.014	0.000	

* Data is available only for the fields shown and not collected for all fields.
** Timing numbers are estimates only. Use report timing summary to get accurate numb

3. Congestion Summary

Task Name	Global Cong Level N-E-S-W	Global Cong Tile% N-E-S-W	Long Cong	Le% N-E-S-W
synth_design				
opt_design				
place_design				
phys_opt_design				
route_design				
phys_opt_design				

* Data is available only for the fields shown and not collected for all fields.

4. Compile Time Summary

Task Name	Runtime(mins)	Number of threads
synth_design	1	2
opt_design	< 1	2
place_design	9	2
phys_opt_design	4	2
route_design	22	2
phys_opt_design	2	2

* Data is available only for the fields shown and not collected for all fields.

The summary report can also be generated in JSON format. In JSON format, the table is presented in flat as a single table which makes parsing it easier. It is the only section of this report that can be output in this format, but it offers an easier way to parse some key design metrics. To generate the summary report in JSON format, run the following command:

```
report_design_analysis -qor_summary -json <json filename>
```

Using Report Design Analysis

Two main categories of QoR problems are usually encountered:

- Timing Violations
- Congestion

Timing Violations

While analyzing and fixing the worst timing violation usually helps the overall QoR improvement, you must also review the other critical paths as they often contribute to the timing closure challenge. You can use the following command to report the 50 worst setup timing paths:

```
report_design_analysis -max_paths 50 -setup
```

The following figure shows an example of the Setup Path Characteristics table generated by this command.

Figure 187: **Setup Path Characteristics**

Paths	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes	Logical Path
Path #1	10.000	9.022	0.223(3%)	8.799(97%)	-0.498	0.066	Safely Timed	0	1	FDRE FDCE
Path #2	10.000	9.079	0.223(3%)	8.856(97%)	-0.496	0.070	Safely Timed	0	1	FDRE FDCE
Path #3	10.000	9.079	0.223(3%)	8.856(97%)	-0.496	0.070	Safely Timed	0	1	FDRE FDCE
Path #4	10.000	9.189	0.223(3%)	8.966(97%)	-0.385	0.070	Safely Timed	0	1	FDRE FDCE
Path #5	10.000	9.189	0.223(3%)	8.966(97%)	-0.385	0.070	Safely Timed	0	1	FDRE FDCE
Path #6	10.000	9.156	0.223(3%)	8.933(97%)	-0.385	0.070	Safely Timed	0	1	FDRE FDCE
Path #7	10.000	9.156	0.223(3%)	8.933(97%)	-0.385	0.070	Safely Timed	0	1	FDRE FDPE
Path #8	10.000	8.921	0.223(3%)	8.698(97%)	-0.525	0.071	Safely Timed	0	1	FDRE FDRE
Path #9	10.000	8.899	0.223(3%)	8.676(97%)	-0.524	0.071	Safely Timed	0	1	FDRE FDRE
Path #10	10.000	8.899	0.223(3%)	8.676(97%)	-0.524	0.071	Safely Timed	0	1	FDRE FDRE
Path #11	10.000	8.936	0.223(3%)	8.713(97%)	-0.510	0.072	Safely Timed	0	1	FDRE FDSE
Path #12	10.000	8.936	0.223(3%)	8.713(97%)	-0.510	0.072	Safely Timed	0	1	FDRE FDSE
Path #13	10.000	8.982	0.223(3%)	8.759(97%)	-0.440	0.072	Safely Timed	0	1	FDRE FDSE
Path #14	10.000	9.015	0.223(3%)	8.792(97%)	-0.499	0.072	Safely Timed	0	1	FDRE FDCE

From the table, you can isolate which characteristics are introducing the timing violation for each path:

- High logic delay percentage (Logic Delay)
  - Are there many levels of logic? (Logic Levels)
  - Are there any constraints or attributes that prevent logic optimization? (Don't Touch, Mark Debug)
  - Does the path include a cell with high logic delay such as RAMB or DSP?
  - Is the path requirement too tight for the current path topology? (Requirement)
- High net delay percentage (Net Delay)
  - Are there any high fanout nets in the path? (High Fanout, Cumulative Fanout)
  - Are the cells assigned to several Pblocks that can be placed far apart? (Pblocks)

    - Are the cells placed far apart? (Bounding Box Size, Clock Region Distance)
    - For SSI devices, are there nets crossing SLR boundaries? (SLR Crossings)
    - Are one or several net delay values a lot higher than expected while the placement seems correct? See the section on Congestion.
- Missing pipeline register in a RAMB or DSP cell (when present in the path)
    - Check the path to see if pipeline register is enabled for RAMBs or DSP cells
- High skew (<-0.5 ns for setup and >0.5 ns for hold) (Clock Skew)
    - Is it a clock domain crossing path? (Start Point Clock, End Point Clock)
    - Are the clocks synchronous or asynchronous? (Clock Relationship)
    - Is the path crossing I/O columns? (IO Crossings)

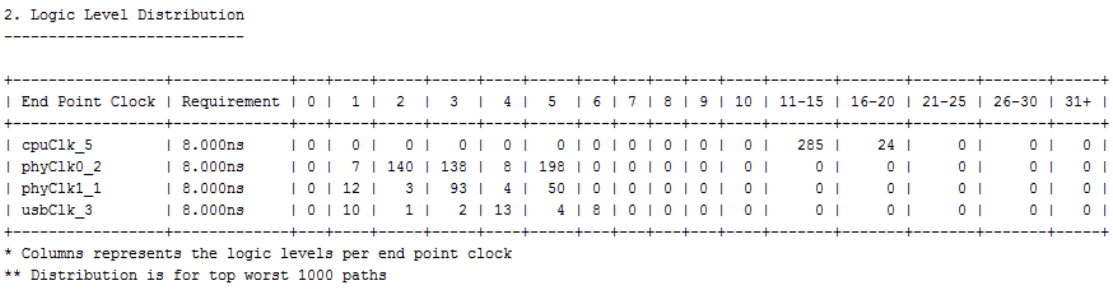
    For visualizing the details of the timing paths and their placement/routing in the AMD Vivado™ IDE, you must use the following command:

```
report_timing -max_paths 50 -setup -input_pins -name worstSetupPaths
```

The paths are sorted by slack and appear in the same order as in the Setup Path Characteristics table (shown in the previous figure).

The `report_design_analysis` command also generates a Logic Level Distribution table for the worst 1000 paths that you can use to identify the presence of longer paths in the design. The longest paths are usually optimized first by the placer to meet timing, which will potentially degrade the placement quality of shorter paths. You must always try to eliminate the longer paths to improve the overall QoR. The following figure shows an example of the Logic Level Distribution for a design with only one clock.

Figure 188: **Logic Level Distribution Table**



2. Logic Level Distribution

End Point Clock	Requirement	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
cpuClk_5	8.000ns	0	0	0	0	0	0	0	0	0	0	0	285	24	0	0	0
phyClk0_2	8.000ns	0	7	140	138	8	198	0	0	0	0	0	0	0	0	0	0
phyClk1_1	8.000ns	0	12	3	93	4	50	0	0	0	0	0	0	0	0	0	0
usbClk_3	8.000ns	0	10	1	2	13	4	8	0	0	0	0	0	0	0	0	0

* Columns represents the logic levels per end point clock
** Distribution is for top worst 1000 paths

Based on what you find, you can improve the netlist by changing the RTL or using different synthesis options, or you can modify the timing and physical constraints.

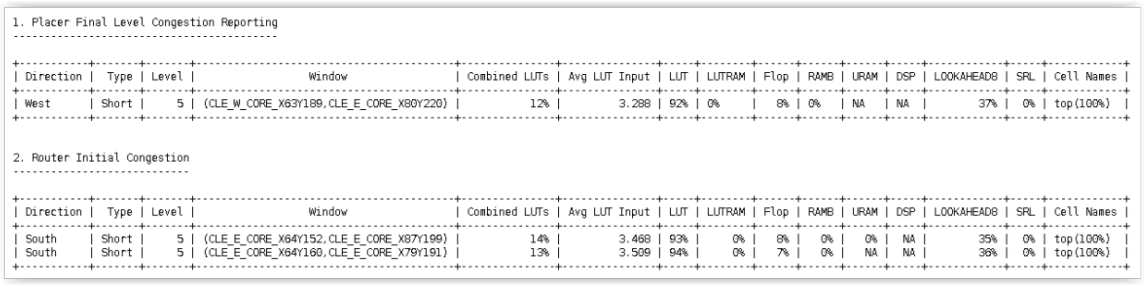
Congestion

The `report_design_analysis` command reports several congestion tables which show the congested area seen by the placer and router. You can generate these tables using the following command in the same Vivado tools session where the placer and router were run:

```
report_design_analysis -congestion
```

The following figure shows an example of the congestion tables which are equivalent to placer final and router initial congestion.

Figure 189: **Estimated Congestion Tables**



1. Placer Final Level Congestion Reporting

Direction	Type	Level	Window	Combined LUTs	Avg LUT Input	LUT	LUTRAM	Flop	RAMB	URAM	DSP	LOOKAHEAD8	SRL	Cell Names
West	Short	5	(CLE_W_CORE_X63Y189,CLE_E_CORE_X80Y220)	12%	3.28B	92%	0%	8%	0%	NA	NA	37%	0%	top(100%)

2. Router Initial Congestion

Direction	Type	Level	Window	Combined LUTs	Avg LUT Input	LUT	LUTRAM	Flop	RAMB	URAM	DSP	LOOKAHEAD8	SRL	Cell Names
South	Short	5	(CLE_E_CORE_X64Y152,CLE_E_CORE_X87Y199)	14%	3.46B	93%	0%	8%	0%	0%	NA	35%	0%	top(100%)
South	Short	5	(CLE_E_CORE_X64Y160,CLE_E_CORE_X79Y191)	13%	3.50B	94%	0%	7%	0%	NA	NA	36%	0%	top(100%)

The names provided for the Module Names correspond to the hierarchical cells present in each reported Tile. You can retrieve the complete name using the following command:

```
get_cells -hier <moduleName>
```

Once the hierarchical cells present in the congested area are identified, you can use congestion alleviating techniques to try reducing the overall design congestion.

Chapter 6

Synthesis Analysis and Closure Techniques

This chapter covers analyzing synthesis results using the elaborated view, with examples.

Using the Elaborated View to Optimize the RTL

When analyzing the timing results after any implementation step with `report_timing`, `report_timing_summary`, or `report_design_analysis`, you must review the structure of critical paths to understand if they can be mapped to logic primitives more efficiently by modifying the RTL, using synthesis attributes, or using different synthesis options. This is especially important for paths with high number of logic levels, which stress the implementation tools and limit the overall design performance.

Whenever you find a critical path with a high number of logic levels, you must question whether the functionality of the path requires so many logic levels or not. It is usually not easy to determine the optimal number of logic levels because it depends on your knowledge of the design and your knowledge of RTL optimization in general. It is a complex task to look at the post-synthesis optimized netlist and identify where the problem comes from in the RTL and how to improve it.

In project mode, the Vivado IDE helps simplifying the analysis by providing a powerful cross probing mechanism between the synthesized or implemented design and the elaborated design. Do the following to cross-probe the synthesized/implemented design and the elaborated design:

1. Open both the synthesized/implemented design and the elaborated design in memory.
2. Select the timing path in the synthesized/implemented design view and show its schematics by pressing the **F4** key.
3. Select the Elaborated Design in the Flow Navigator pane. The RTL cells that correspond to the timing path are also selected, so that you can open the RTL schematics (by pressing the **F4** key) to view the same path in the elaborated view or trace from the endpoint pin back to the startpoint cell.
4. Review the RTL logic traversed by the path, especially the size of the operators or vectors.

Example

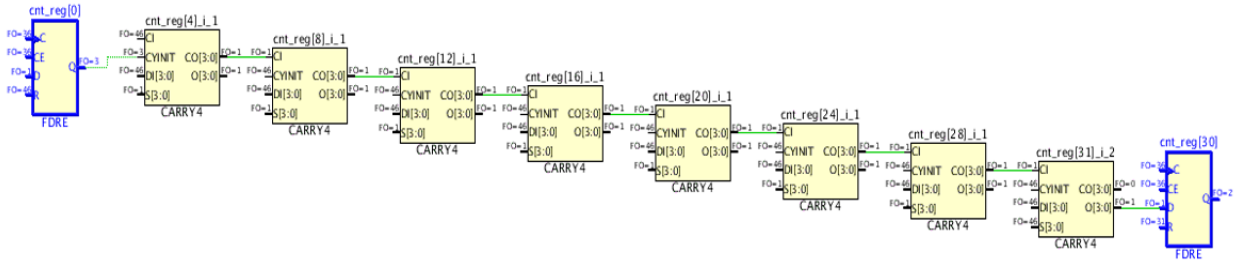
In the following example, a user has written a counter as follows:

Figure 190: **Simple Counter VHDL Example**

```
signal cnt : integer := 0;

process (clk)
begin
    if(clk'event and clk = '1') then
        if(cnt = 16) then
            cnt <= 0;
        else
            cnt <= cnt + 1;
        end if;
        if(cnt = 8) then
          dout <= din0;
        else
          dout <= din1;
        end if;
    end if;
end process;
```

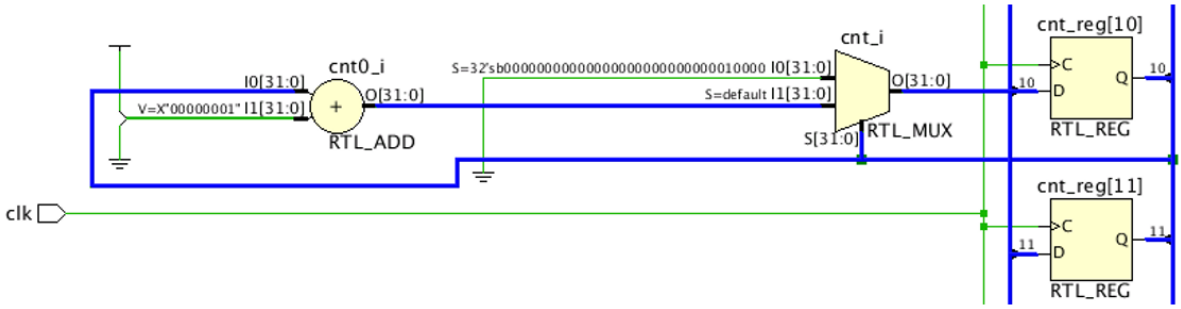
The signal `cnt` counts from 0 to 16, which requires a 5-bit vector to encode. The post-route critical schematics is shown in the following figure. The endpoint is the bit 30 of the `cnt` signal.

Figure 191: **cnt Counter Post-Route Critical Path Schematic**



After selecting the startpoint and endpoint cells of the critical path, you can visualize the equivalent path in the elaborated view by opening a schematics of the selected cells and expanding the logic from the endpoint pin back to the startpoint, as shown in the following figure.

Figure 192: **cnt Counter in the Elaborated View**



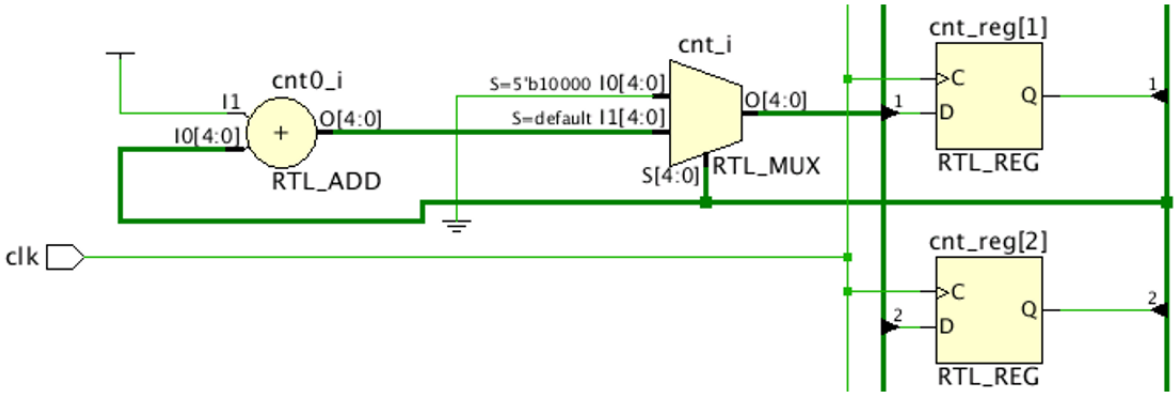
The elaborated view shows that the adder-input has been sized to 32 bit, because the signal `cnt` is declared as an integer. In this particular example, the 32-bit operator is retained throughout the synthesis optimizations. The elaborated view gives a good hint of what is happening and you can change the RTL as follows in order to get a better optimized netlist and timing QoR. As the counter increments from 0 to 16, you can define a range for the signal `cnt` which forces the adder-inputs to be 5 bits wide instead of 32 bits wide.

Figure 193: **Simple Counter VHDL example with Integer Range**

```
signal cnt : integer range 0 to 16 := 0;

process (clk)
begin
    if(clk'event and clk = '1') then
        if(cnt = 16) then
            cnt <= 0;
        else
            cnt <= cnt + 1;
        end if;
        if(cnt = 8) then
          dout <= din0;
        else
          dout <= din1;
        end if;
    end if;
end process;
```

The change made to the RTL code will subsequently impact the synthesis optimization, which you can verify using the elaborated view instead of going through the entire compilation flow:

Figure 194: **cnt Counter in the Elaborated View after RTL Improvement**

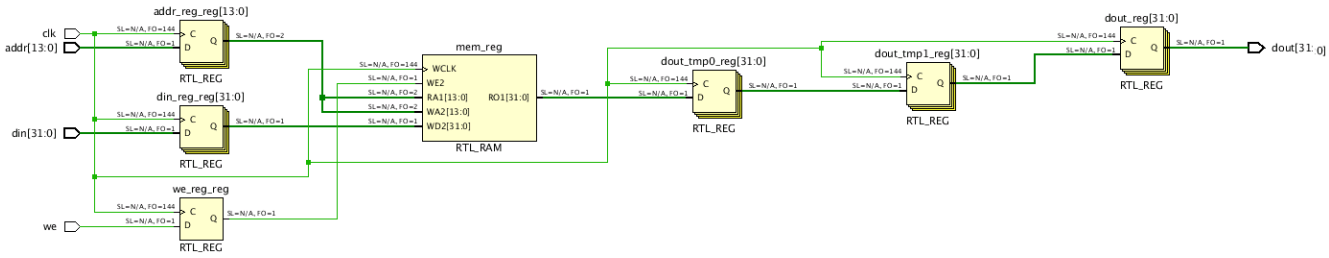


Decomposing Deep Memory Configurations for Balanced Power and Performance

In deep memory configurations, the synthesis attribute `RAM_DECOMP` can be used for better memory decomposition and reduced power consumption. This attribute can be set in the RTL. When the `RAM_DECOMP` attribute is applied to a memory, the memory is setup in a wider configuration (of primitives) instead of a deep and narrow configuration.

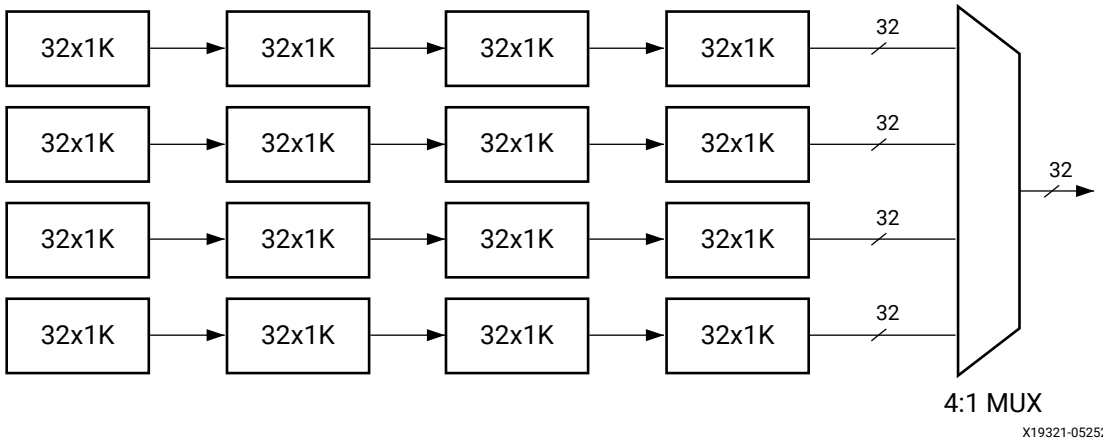
When the `CASCADE_HEIGHT` attribute is used along with the `RAM_DECOMP` attribute, the synthesis inference has more granular control on cascading thereby providing balanced power and performance. This approach requires additional address decoding logic but reduces the number of block RAMs accessed at any given point in time, which helps reduce power consumption. The memory configuration (32 × 16K) in the following figure shows an example of how the memory is decomposed when the `RAM_DECOMP` and `CASCADE_HEIGHT` attributes are set.

Figure 195: **32 × 16K Memory Configuration**



If the attributes `RAM_DECOMP` = power and `CASCADE_HEIGHT` = 4 are applied, 16 RAMB36E2 are inferred and the memory is decomposed as shown in the following figure.

Figure 196: **Generated Structure for 32 × 16K Memory Configuration using RAM_DECOMP and CASCADE_HEIGHT Attributes**



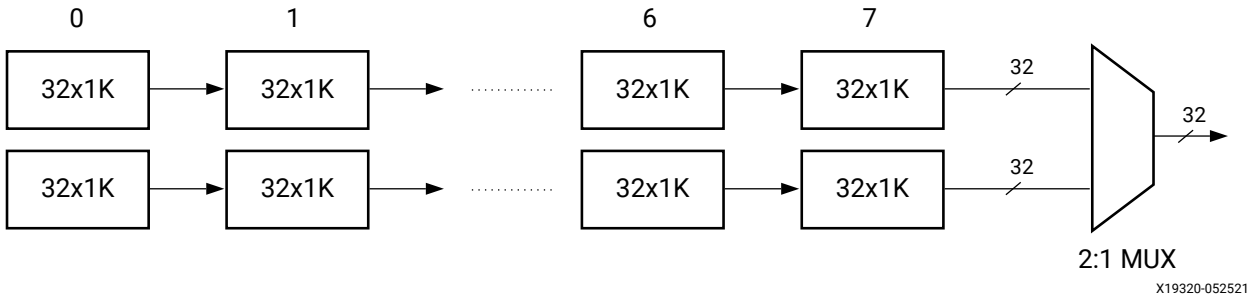
The base primitive used here is 32 × 1K and four block RAMs are cascaded with a built-in feature to form a 32 × 4K configuration. Four such parallel structures create a 16K deep memory. The outputs are multiplexed to generate the output data.

Figure 197: **RTL Code Snippet for 32 × 16K Memory Configuration using RAM_DECOMP and CASCADE_HEIGHT Attributes**

```
module test
(
input  clk,
input  we,
input  [13:0] addr,
input  [31:0] din,
output reg [31:0] dout
);

(* ram_style = "block", ram_decomp = "power", cascade_height = 4 *) reg  [31:0] mem [(16*1024)-1:0];
reg  [13:0] addr_reg;
reg  [31:0] dout_tmp0;
reg  [31:0] dout_tmp1;
reg  [31:0] din_reg;
reg         we_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    din_reg  <= din;
    we_reg   <= we;
    dout_tmp0 <= mem[addr_reg];
    dout_tmp1 <= dout_tmp0;
    dout <= dout_tmp1;
    if (we_reg)
        mem[addr_reg] <= din_reg;
end

endmodule
```

If only the `RAM_DECOMP` = power attribute is applied, 16 RAMB36E2 are inferred and the memory is decomposed as shown in the following figure.

Figure 198: **Generated Structure for 32 × 16K Memory Configuration using `RAM_DECOMP` Attribute**



The base primitive used here is 32 × 1K and eight block RAMs are cascaded with a built-in feature to form a 32 × 8K configuration. Two such parallel structures create a 16K deep memory. The outputs are multiplexed to generate the output data. The multiplexer is a 2:1 MUX.

Figure 199: **RTL Code Snippet for 32 × 16K Memory Configuration using `RAM_DECOMP` Attribute**

```
module test
(
input  clk,
input  we,
input  [13:0] addr,
input  [31:0] din,
output reg [31:0] dout
);

(* ram_style = "block", ram_decomp = "power"*) reg  [31:0] mem [(16*1024)-1:0];
reg  [13:0] addr_reg;
reg  [31:0] dout_tmp0;
reg  [31:0] dout_tmp1;
reg  [31:0] din_reg;
reg         we_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    din_reg  <= din;
    we_reg   <= we;
    dout_tmp0 <= mem[addr_reg];
    dout_tmp1 <= dout_tmp0;
    dout <= dout_tmp1;
    if (we_reg)
        mem[addr_reg] <= din_reg;
end

endmodule
```

The overall power savings are similar for both the memory decomposition examples, shown in Figure 196 and Figure 198, because only one block RAM is active at any given point in time. However, in terms of performance, a four-level deep cascaded block RAM chain (Figure 196) provides better performance than an eight-level deep cascaded block RAM chain (Figure 198).

Optimizing RAMB Utilization when Memory Depth is not a Power of 2

The following test case can be used to observe the log file generated by the synthesis tool and see if there is any improvement that can be done to the RTL to guide the tool in a better way. The following code snippet shows a 40K-deep 36-bit wide memory description in VHDL. The address bus requires 16 bits.

Figure 200: **40K x 36 bits Memory RTL Example**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity test is
    Port ( clk   : in  STD_LOGIC;
           addr  : in  STD_LOGIC_VECTOR (15 downto 0);
           din   : in  STD_LOGIC_VECTOR (35 downto 0);
           we    : in STD_LOGIC;
           dout  : out  STD_LOGIC_VECTOR (35 downto 0));
end test;

architecture rtl of test is
signal addr_reg : STD_LOGIC_VECTOR(15 downto 0);
type mem_type is array (0 to 40959) of STD_LOGIC_VECTOR(35 downto 0);
signal mem : mem_type;
begin

process (clk)
begin
  if (rising_edge(clk)) then
    addr_reg <= addr;
    if(we='1') then
      mem(to_integer(unsigned(addr_reg))) <= din;
    end if;
    dout <= mem(to_integer(unsigned(addr_reg)));
  end if;
end process;

end rtl;
```

Using the `report_utilization` command post-synthesis, you can see that 72 block RAMs are generated by the synthesis tool, as shown in the following figure.

Figure 201: **Number of Block RAMs Generated by Synthesis in the Utilization Report**

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	72	0	1030	6.99
RAMB36/FIFO*	72	0	1030	6.99
RAMB36E1 only	72			
RAMB18	0	0	2060	0.00

If you calculate the number of block RAMs that are supposed to be inferred for the 40K x 36 configuration, you would end up with fewer block RAMs than the synthesis tool generated.

The following shows the manual calculation for this memory configuration:

- 40K x 36 can be broken in two memories: (32K x 36) and (8K x 36)
- An address decoder based on the MSB address bits is required to enable one or the other memory for read and write operations, and select the proper output data.
- The 32K x 36 memory can be implemented with 32 RAMBs: 4 * 8 * (4K x 9)
- The 8K x 36 memory can be implemented with 8 RAMBs: 8 * (1K x 36)
- In total, 40 RAMBs are required to optimally implement the 40K x 36 memory.

To verify that the optimal number of RAMBs have been inferred, the synthesis log file includes a section that details how each memory is configured and mapped to FPGA primitives. As shown in the following figure, memory depth is treated as 64K, which gives a clue that non-power of 2 depths are not handled in an optimal way.

Figure 202: **RAM Configuration and Mapping Section in the Synthesis Log**

```
---------------------------------------------------------------------------------
Start ROM, RAM, DSP and Shift Register Reporting
---------------------------------------------------------------------------------

Block RAM:
+------------+-----------+-----------------------+---+---+-----------------------+---+---+---------+--------+--------+------------------+
|Module Name | RTL Object | PORT A (Depth x Width) | W | R | PORT B (Depth x Width) | W | R | OUT_REG | RAMB18 | RAMB36 | Hierarchical Name |
+------------+-----------+-----------------------+---+---+-----------------------+---+---+---------+--------+--------+------------------+
|test        | mem_reg    | 64 K x 36(READ_FIRST)  | W | R |                       |   |   | Port A  | 0      | 72     | test/extram__2    |
+------------+-----------+-----------------------+---+---+-----------------------+---+---+---------+--------+--------+------------------+
```

The synthesis tool has used 64K x 1 (2 block RAMs with cascade feature), 36 such structures because of 36-bit data. So in total, you have 36 x 2 = 72 block RAMs. The following figure shows the code snippet that forces synthesis to infer the optimal number of RAMBs.

Figure 203: **Optimized 40 K x 36 bits Memory RTL Example**

```
architecture rtl of test is
    signal addr_reg : std_logic_vector(15 downto 0);
    type ram_type_0 is array (0 to 32767) of std_logic_vector(35 downto 0);
    type ram_type_1 is array (0 to 8191) of std_logic_vector(35 downto 0);
    signal RAM_0 : ram_type_0;
    signal RAM_1 : ram_type_1;
    signal dout_0 : std_logic_vector(35 downto 0);
    signal dout_1 : std_logic_vector(35 downto 0);
begin

    process (clk)
    begin
        if clk'event and clk = '1' then
           addr_reg <= addr;
           if we = '1' and addr_reg(15) = '0' then
               RAM_0(to_integer(unsigned(addr_reg(14 downto 0)))) <= din;
           end if;
           dout_0 <= RAM_0(to_integer(unsigned(addr_reg(14 downto 0))));
        end if;
    end process;

    process (clk)
    begin
        if clk'event and clk = '1' then
           if we = '1' and addr_reg(15) = '1' then
               RAM_1(to_integer(unsigned(addr_reg(12 downto 0)))) <= din;
           end if;
           dout_1 <= RAM_1(to_integer(unsigned(addr_reg(12 downto 0))));
        end if;
    end process;

    dout <= dout_1 when addr_reg(15) = '1' else dout_0;

end rtl;
```

Optimizing RAMB Input Logic to Allow Output Register Inference

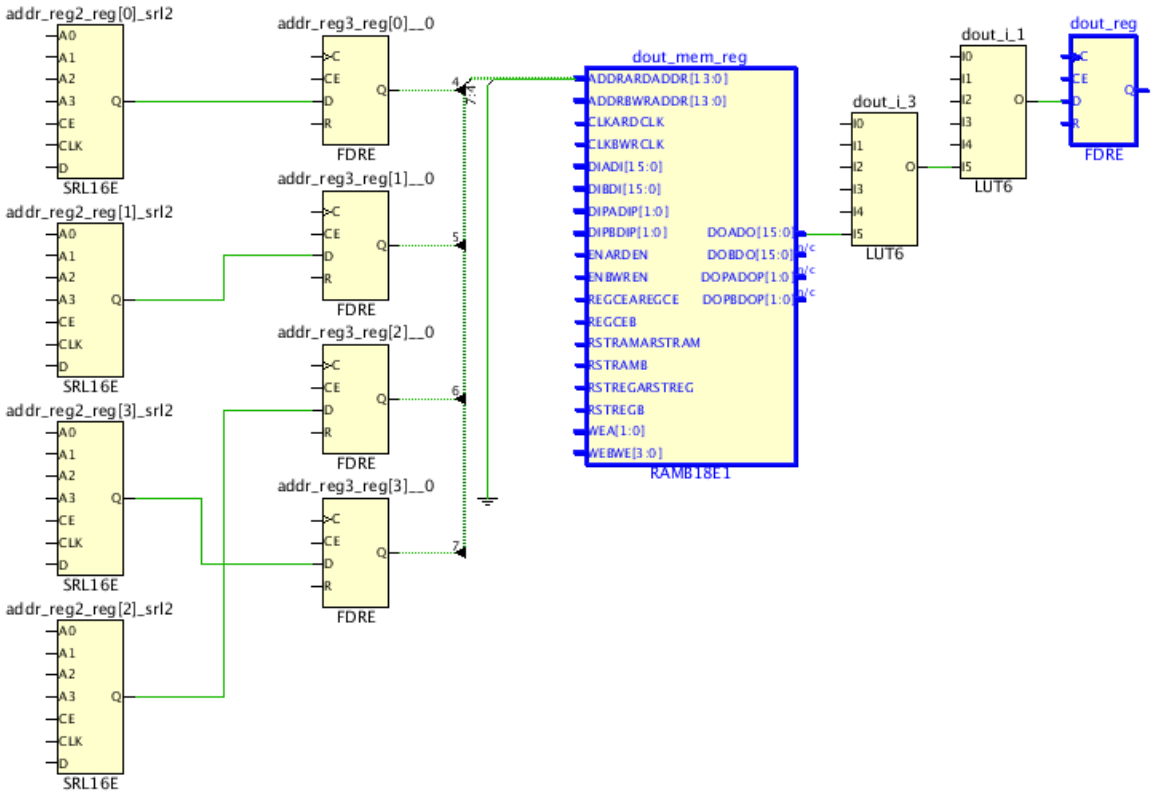
The following RTL code snippet generates a critical path from block RAM (actually it is a ROM) with multiple logic levels ending at a flip-flop (FF). The RAMB cell has been inferred without the optional output registers (DOA-0), which adds over 1 ns extra delay penalty to the RAMB output path.

Figure 204: **Memory RTL Code Without Inferred RAMB Output Register**

```
module test (input clk,input [3:0] addr, output reg dout, dout_shift);

(* rom_style = "block" *) reg [15:0] mem [0:15];

reg [3:0] addr_reg0;
reg [3:0] addr_reg1;
reg [3:0] addr_reg2;
reg [3:0] addr_reg3;

reg [15:0] dout_mem;

initial
begin
   $readmemh("init.txt",mem);
end

always@(posedge clk)
begin
    addr_reg0 <= addr;
    addr_reg1 <= addr_reg0;
    addr_reg2 <= addr_reg1;
    addr_reg3 <= addr_reg2;
end

always@(posedge clk)
begin
    dout_mem <= mem[addr_reg3];
    dout <= |dout_mem;
    dout_shift <= |addr_reg3;
end

endmodule
```

The critical path for the above RTL code is shown by the tool, such as in the following figure.

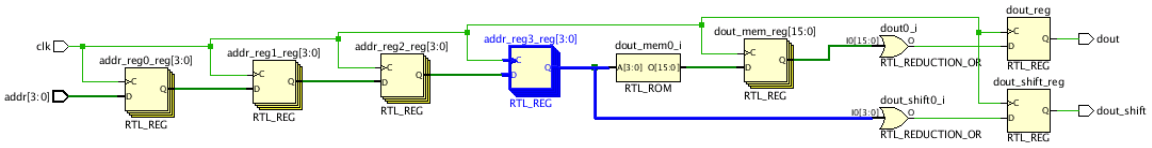
Figure 205: **Critical Path from RAMB Without Output Register Enabled**



It is good practice to review the critical paths after synthesis and after each implementation step in order to identify which groups of logic need to be improved. For long paths or any paths that do not take advantage of the FPGA hardware features optimally, go back to the RTL description, try to understand why the synthesized logic is not optimal, and modify the code to help the synthesis tool improve the netlist.

Vivado has a powerful embedded debugging mechanism that you can use to start off with elaborated view. The elaborated view helps to identify where the problem could be, instead of manually searching through the RTL code. See the elaborated view shown in the following figure for the above RTL code snippet.

Figure 206: **Elaborated View of RTL Code Snippet**



The elaborated view gives a good hint about the inefficient structure for the given test case. In this case, the problem comes from the address register fanout (addr_reg3_reg), which drives the memory address as well as some glue-logic, highlighted in blue.

RAMB inference by the synthesis tool requires a dedicated address register in the RTL code, which is not compatible with the current address register fanout. As a consequence, the synthesis tool re-times the output register in order to allow the RAMB inference instead of using it to enable the RAMB *optional* output register.

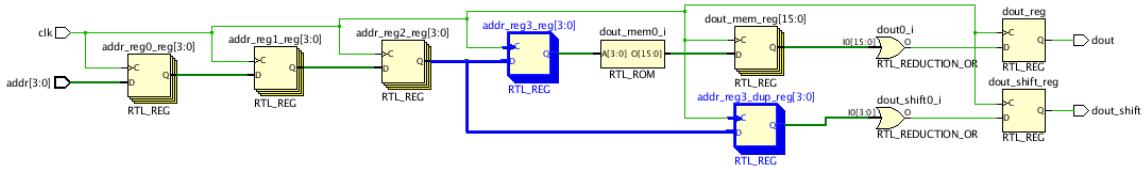
By replicating the address register in the RTL code so that the memory address and the interconnect logic | FPGA logic are driven by separate registers, the RAMB will be inferred with the output registers enabled.

The RTL code and elaborated view after manual replication are shown in the following figures:

Figure 207: **RTL Code with the Replicated Address Register**

```
module test (input clk,input [3:0] addr, output reg dout, dout_shift);

(* rom_style = "block" *) reg [15:0] mem [0:15];

reg [3:0] addr_reg0;
reg [3:0] addr_reg1;
reg [3:0] addr_reg2;
reg [3:0] addr_reg3;
(* KEEP = "true" *) reg [3:0] addr_reg3_dup;

reg [15:0] dout_mem;

initial
begin
   $readmemh("init.txt",mem);
end

always@(posedge clk)
begin
    addr_reg0 <= addr;
    addr_reg1 <= addr_reg0;
    addr_reg2 <= addr_reg1;
    addr_reg3 <= addr_reg2;
    addr_reg3_dup <= addr_reg2;
end

always@(posedge clk)
begin
    dout_mem <= mem[addr_reg3];
    dout <= |dout_mem;
    dout_shift <= |addr_reg3_dup;
end

endmodule
```

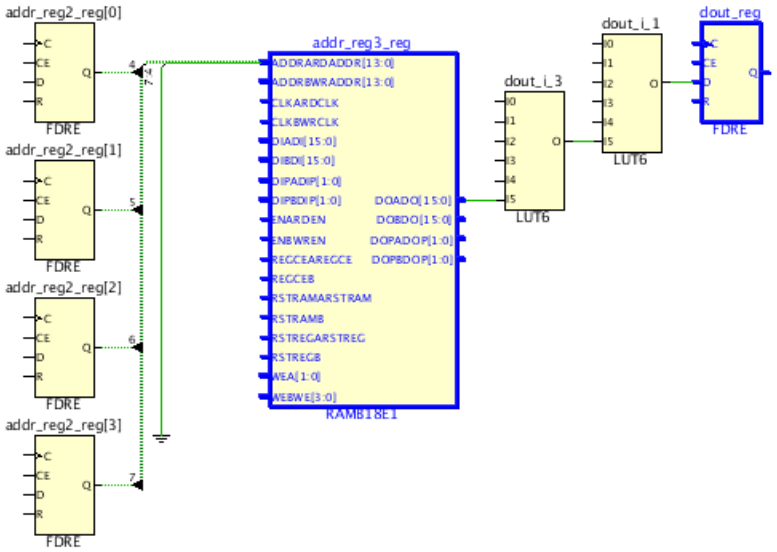
Figure 208: **Elaborated View of the Replicated Address Register**



The critical path for the modified RTL code can be seen in the following figure. Notice the following:

- The `addr_reg2_reg` register is connected to the address pin of the block RAM.
- The `addr_reg3_reg` register has been absorbed in the Block RAM.
- The RAMB output register is enabled, which significantly reduces the datapath delay on the RAMB outputs.

Figure 209: **Critical Path for the Modified RTL Code**



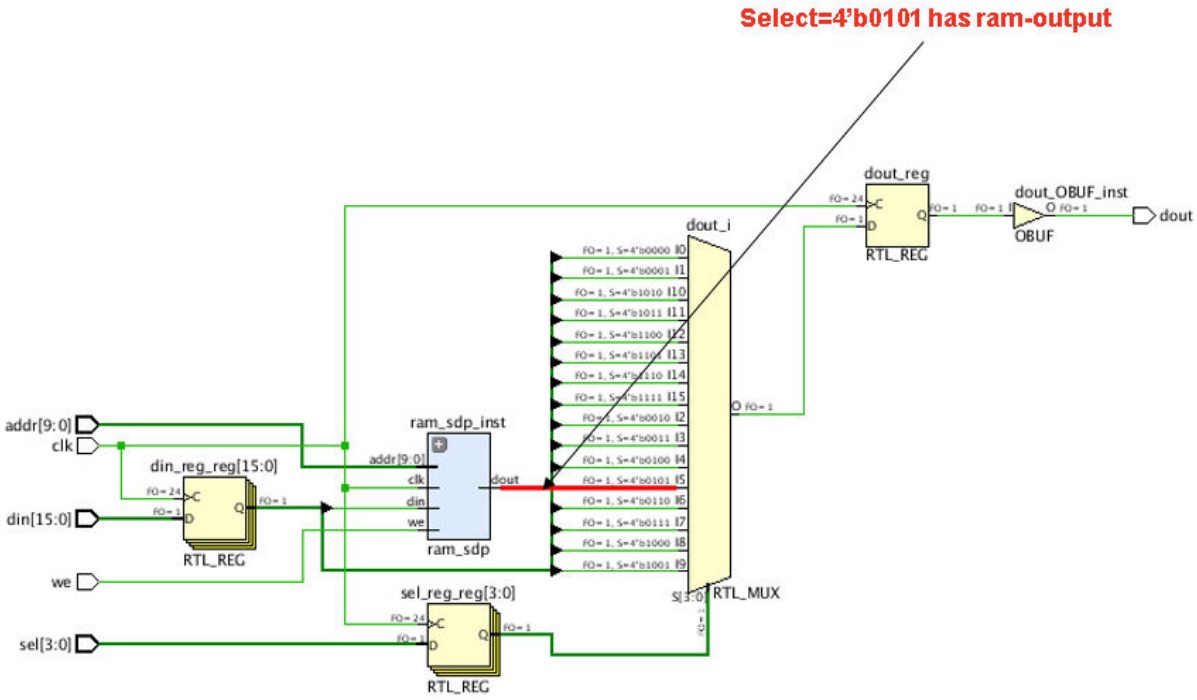
Improving Critical Logic on RAMB Outputs

The following test case highlights about improving critical paths through restructuring, such as when pushing macro (block RAM) closer to the destination register.

The following figure shows a 16x1 Multiplexer with only one input to the Multiplexer coming from block RAM and the rest of the inputs being fed by registers.

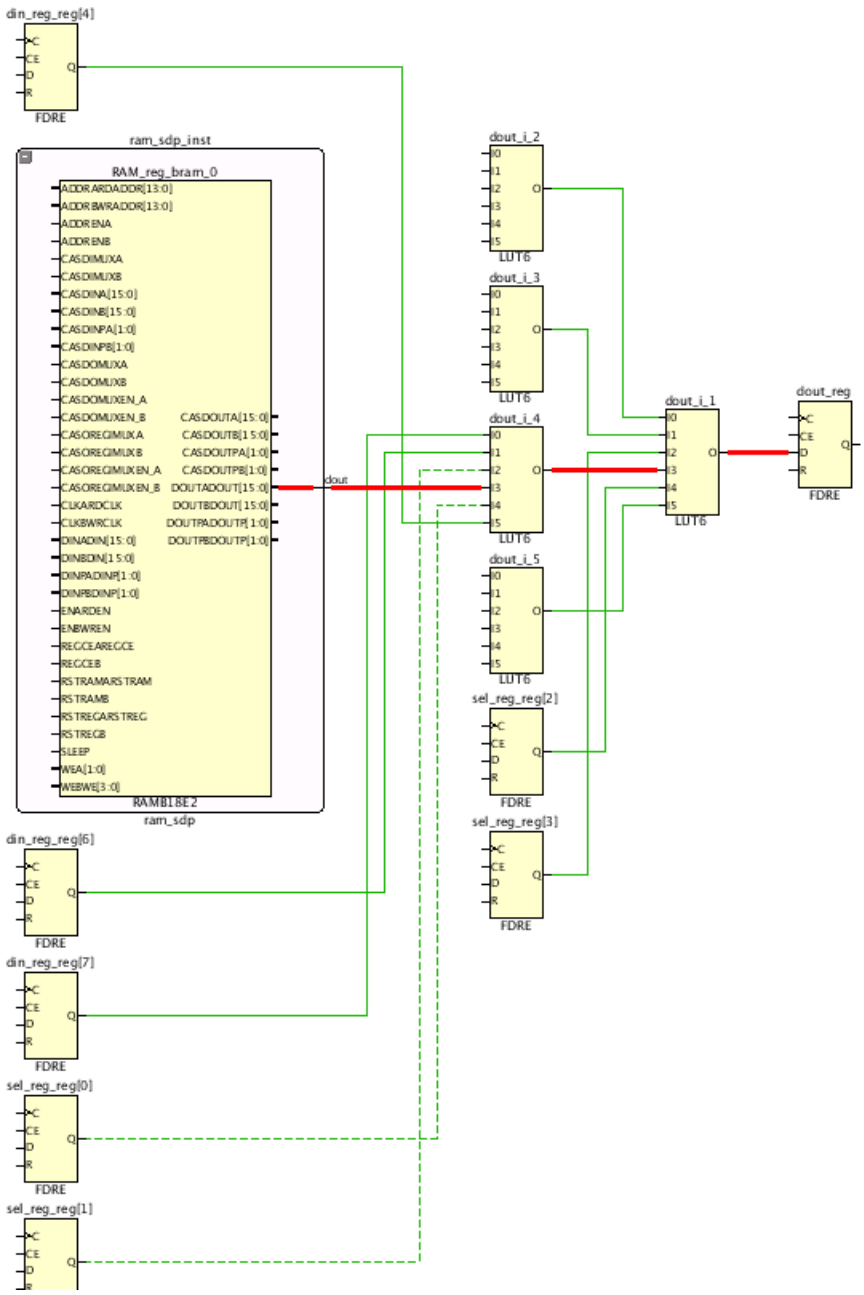
Critical path: block RAM-> 2 Logic levels -> FF.

Figure 210: **16x1 Multiplexer Connected to Block RAM Outputs**



The following figure shows the critical path where the block RAM to FF path is highlighted in red. There are 2 logic levels from block RAM->FF as well as FF->FF. Because block RAM CLK->Q delay is higher for block RAM, block RAM->FF is critical.

Figure 211: **Critical RAMB-LUT-FF Path**



Next, look at the RTL code snippet shown in the following figure to see whether there is a way to restructure the logic.

Figure 212: **RTL Code Snippet**

```
ram_sdp ram_sdp_inst (.clk(clk), .we(we), .addr(addr), .din(din_reg[5]), .dout(dout_from_ram));

always@(posedge clk)
begin
sel_reg <= sel;
din_reg <= din;
case(sel_reg)
    4'd0: dout <= din_reg[0];
    4'd1: dout <= din_reg[1];
    4'd2: dout <= din_reg[2];
    4'd3: dout <= din_reg[3];
    4'd4: dout <= din_reg[4];
    4'd5: dout <= dout_from_ram;
    4'd6: dout <= din_reg[6];
    4'd7: dout <= din_reg[7];
    4'd8: dout <= din_reg[8];
    4'd9: dout <= din_reg[9];
    4'd10: dout <= din_reg[10];
    4'd11: dout <= din_reg[11];
    4'd12: dout <= din_reg[12];
    4'd13: dout <= din_reg[13];
    4'd14: dout <= din_reg[14];
    4'd15: dout <= din_reg[15];
endcase
end
```

The optimal way to restructure the logic is to rewrite the above code snippet by breaking the 16x1 Multiplexer into two multiplexers. You can exempt the condition of select value 4'd5 and use it as an enabling condition for the 2x1 Multiplexer as shown in the following figure, creating this cascade Multiplexer structure results in FF->FF with 3 logic levels, but block ;RAM->FF is reduced to 1 logic level. This way, the block RAM->FF path has been improved, which helps the downstream tools for better placement because RAMB placement is more challenging than LUT and FF placement. In general, fewer long paths around Macro primitives such RAMB, URAM, and DSP will yield better QoR for any given design.

Figure 213: **Cascade Multiplexer Structure to Reduce RAMB Output Logic Levels**

```
ram_sdp ram_sdp_inst (.clk(clk), .we(we), .addr(addr), .din(din_reg[5]), .dout(dout_from_ram));

(* KEEP = "true" *) reg dout_nxt;

always@*
begin
dout_nxt <= dout;
case(sel_reg)
    4'd0: dout_nxt <= din_reg[0];
    4'd1: dout_nxt <= din_reg[1];
    4'd2: dout_nxt <= din_reg[2];
    4'd3: dout_nxt <= din_reg[3];
    4'd4: dout_nxt <= din_reg[4];
    //4'd5: dout_nxt <= dout_from_ram;
    4'd6: dout_nxt <= din_reg[6];
    4'd7: dout_nxt <= din_reg[7];
    4'd8: dout_nxt <= din_reg[8];
    4'd9: dout_nxt <= din_reg[9];
    4'd10: dout_nxt <= din_reg[10];
    4'd11: dout_nxt <= din_reg[11];
    4'd12: dout_nxt <= din_reg[12];
    4'd13: dout_nxt <= din_reg[13];
    4'd14: dout_nxt <= din_reg[14];
    4'd15: dout_nxt <= din_reg[15];
endcase

end

always@(posedge clk)
begin
sel_reg <= sel;
din_reg <= din;
if(sel_reg == 4'd5)
    dout <= dout_from_ram;
else
    dout <= dout_nxt;
end
```

Chapter 7

Implementation Analysis and Closure Techniques

This chapter covers techniques that can be used for timing closure. It should be considered in addition to the timing closure techniques described in the *UltraFast Design Methodology Timing Closure Quick Reference Guide* (UG1292) and *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

- Intelligent design runs is an automated timing closure flow that solves complex timing closure issues and requires little user knowledge to use.
- The QoR suggestion object flow enhances QoR automatically for the user by applying properties.
- The ML strategy flow helps the user select optimal tool options for a given design.
- Floorplanning is a complex technique that can guide the placer to improve placer results that can result in improved timing paths and congestion.
- Determining if a design has hold issues can be a key part in determining a timing closure strategy. This section describes how to diagnose if hold is an issue.

Intelligent Design Runs

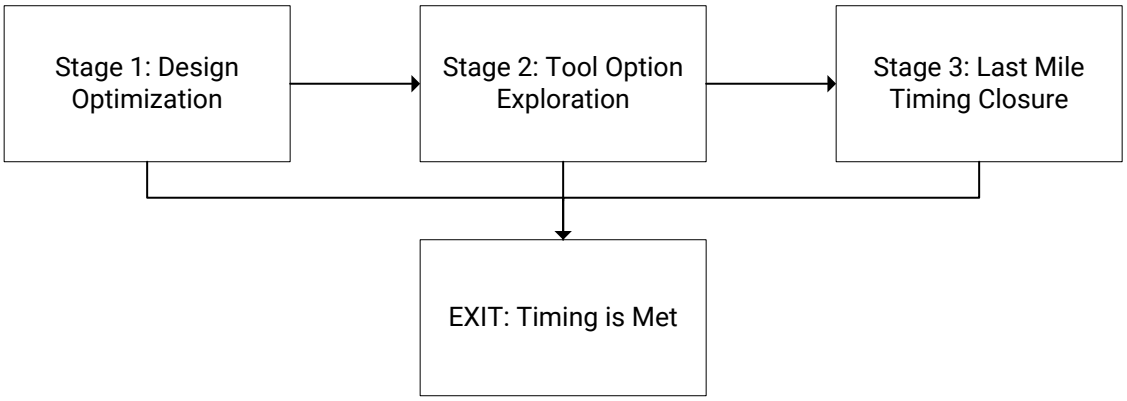
An intelligent design run (IDR) is a special type of implementation run that uses a complex flow to attempt to close timing. Because an IDR can be aggressive, a compile time of approximately 3.5 times that of a standard run can be expected.

The IDR presents a simple user interface around complex timing closure features, and achieves results at least on par with FPGA experts for a high percentage of designs.

Overview

The Intelligent Design Run for timing closure is an aggressive timing closure implementation run with the sole objective of closing timing. Both power and compile time are not considered, but some power optimizations might be possible with utilization savings. It is split into three stages, as shown in the following simplified diagram.

Figure 214: **IDR Overview**



The flow is fully automated, and there is no user control over which stages can be run. Before attempting to close timing using IDR, a design should be clean of methodology issues. Run `report_methodology` and fix or waive all critical warnings and warnings.

The details of each of the stages are as follows.

- **Stage 1: Design Optimization:** In the Design Optimization stage QoR Suggestions are generated and applied. Compile time is typically up to two and a half times the compile time of a standard implementation run. The reasons for this are as follows:
  - To generate accurate data for analysis, the implementation tools must be run to post-place or post-route. To apply the suggestions, the design run must be reset and the implementation tools rerun.
  - By allowing the impact of QoR suggestions to be realized before conducting a new analysis and generating new suggestions, the impact of design issues is not overestimated, resulting in the maximum QoR impact.
- **Stage 2: Tool Option Exploration:** This stage uses ML strategies to predict the best tool options to use.
- **Stage 3: Last Mile Timing Closure:** This stage leverages post-route `phys_opt_design`, incremental implementation using a Last Mile directive, and incremental QoR suggestions to close timing. To enter this stage, a design must have an RQA score of 3 or greater, and have WNS between -0.250 and 0.000. If these criteria are not met, this stage is skipped and the flow exits.

The flow can exit in any of the following conditions:

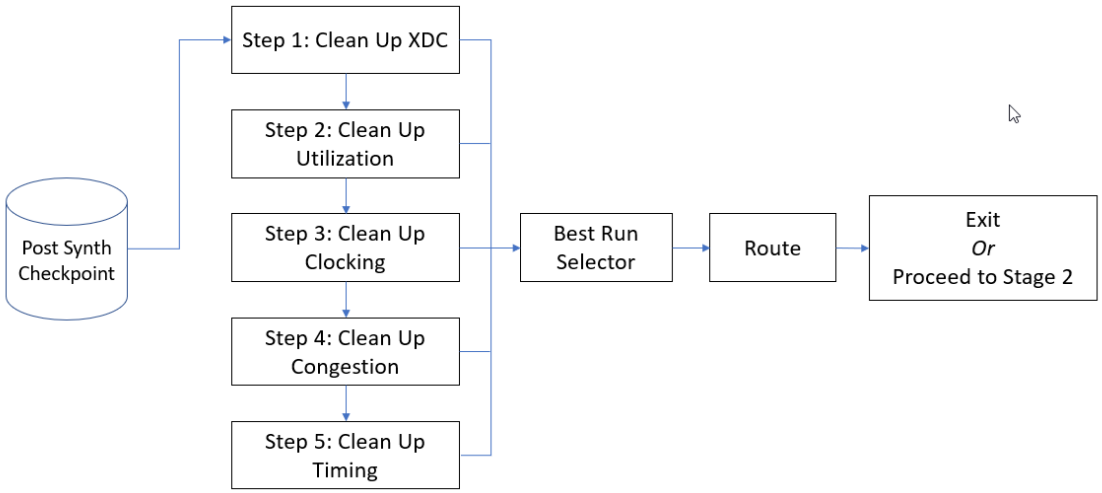
- At any stage, if timing is met and the design is fully routed.
- At stage 1, if:
  - The design fails initial timing checks.

    - The design fails initial utilization checks.
    - The tool option to exit when failed methodology checks is activated and there are failed methodology checks with IDs TIMING-6/7/8/13.
    - The design fails to route.
    - There are no predicted ML strategies.
- At stage 2, if the Last Mile criteria are not met.
- At the end of stage 3, if the Last Mile algorithms have been exhausted and no further improvements can be made.

Stage 1: Design Optimization

The Design Optimization stage is split into sequentially executed steps. These steps are shown in the following diagram.

Figure 215: **Design Optimization Steps**



Within any given step, multiple implementation commands such as `opt_design`, `place_design`, and `route_design` can be run and QoR suggestions can be generated. For each step, there is a target suggestion list; if any of the generated suggestions appear on the target suggestion list, the design is reset to the required design stage for the suggestion to be successfully applied. If there are no suggestions available on the target suggestion list for a given step, the step is skipped.

The details of the Design Optimization steps are as follows.

- **Clean Up XDC:** The design is checked for any causes that generate an implementation error and timing that is impossible to fix. If an error is picked up, the flow exits. No suggestions are generated or applied at this stage.

- **Clean Up Utilization:** Suggestions that reduce utilization without timing penalty are sought. Other non-utilization-based suggestions might be applied if they can be detected and fixed early in the flow.
- **Clean Up Clocking:** The design is run to `place_design` to generate accurate clock skew timing numbers. If suggestions exist, the flow is reset.

  ***Note*:** If there are no suggestions in the Clean Up Utilization and Clean Up Clocking stages, a special stage called First Pass is reported. This is used as a baseline reference to compare with subsequent stages. To reduce compile time, it is not generated if there are suggestions.

- **Clean Up Congestion :** In this step, congestion suggestions are generated after running a limited part of the router to more accurately see the congestion in the design. These suggestions are applied if they exist.

  ***Note*:** The log file shows that `route_design` failed when generating this congestion information, but this can be ignored because a full route is not intended at this point.

- **Clean Up Timing:** This phase generates QoR Suggestions based on timing paths that fail timing from the preceding phase's checkpoint and the placer is rerun.

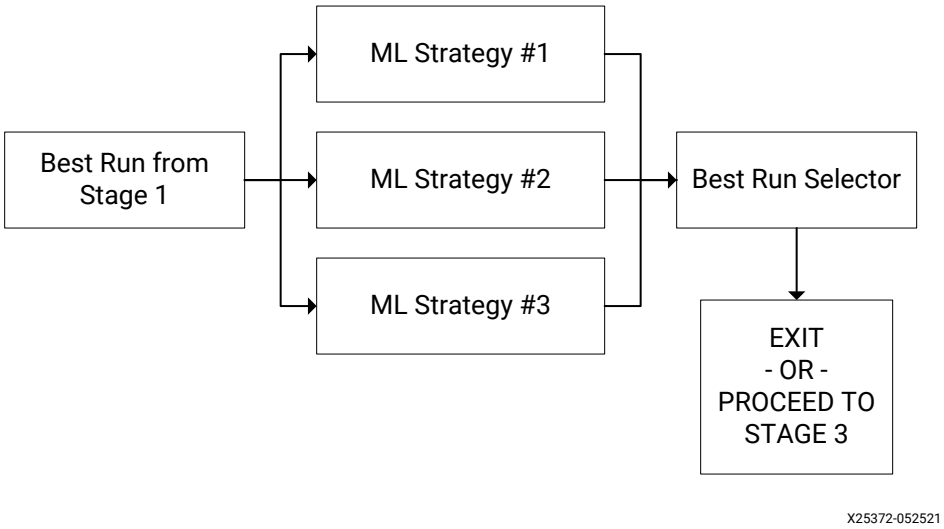
At the end of stage 1 (Design Optimization), it is decided whether to exit or to take the best run to stage 2 (Tool Option Exploration).

To examine the modifications the IDR has made to the design, the QoR suggestion report shows which suggestions are GENERATED and APPLIED at each step. Multiple checkpoints are also generated which can be accessed easily from the design run directory.

Stage 2: Tool Option Exploration

In the Tool Option Exploration stage, the goal is to get the maximum QoR return from tool options. To achieve this, three implementation runs using tool options predicted by ML strategies are run. This is shown in the following diagram:

Figure 216: **Tool Exploration Implementation Runs**



QoR suggestions are carried forward from the best run in stage 1. If later steps in stage 1 showed a QoR degradation, their suggestions are dropped. By running three ML strategy runs (where three or more are available), QoR fluctuations from any single run are smoothed out.

The ML strategy phase uses standard implementation runs that can be run in parallel if compute resources allow. As such, when used in parallel, the compile time of this phase is approximately one implementation run. When the implementation runs are finished, the best run selector determines which run to take forward. The flow behavior is as follows:

- Exit if timing is met.
- Proceed to stage 3 if Last Mile Timing Closure criteria are met.
- Exit if Last Mile Timing Closure criteria are not met.

Stage 3: Last Mile Timing Closure

The Last Mile Timing Closure stage takes the best implementation run result from either of the previous two stages and attempts to close timing on it. The QoR gain in this phase might be small compared to compile time. The design must have met the Last Mile Timing Closure requirements for the stage to be run.

The Last Mile directive continues on from an existed routed checkpoint and attempts to work on paths that are failing. Timing closure is achieved in approximately 20% of designs with a WNS < -0.100 ns.

The goal of the Last Mile Timing Closure stage is to close timing on the design. This is slightly different when compared to the default tool flow, which aims to achieve the best WNS possible and timing closed WHS. Algorithms must strike a balance between trying to improve timing but not altering the place and route results significantly. To achieve this, the Last Mile incremental directive and QoR suggestions are used to close timing. Suggestions with the APPLIED property are reused from the reference run and suggestions that have the INCREMENTAL_FRIENDLY property set are applied. After routing is complete, `phys_opt_design` can be run to further attempt to close timing.

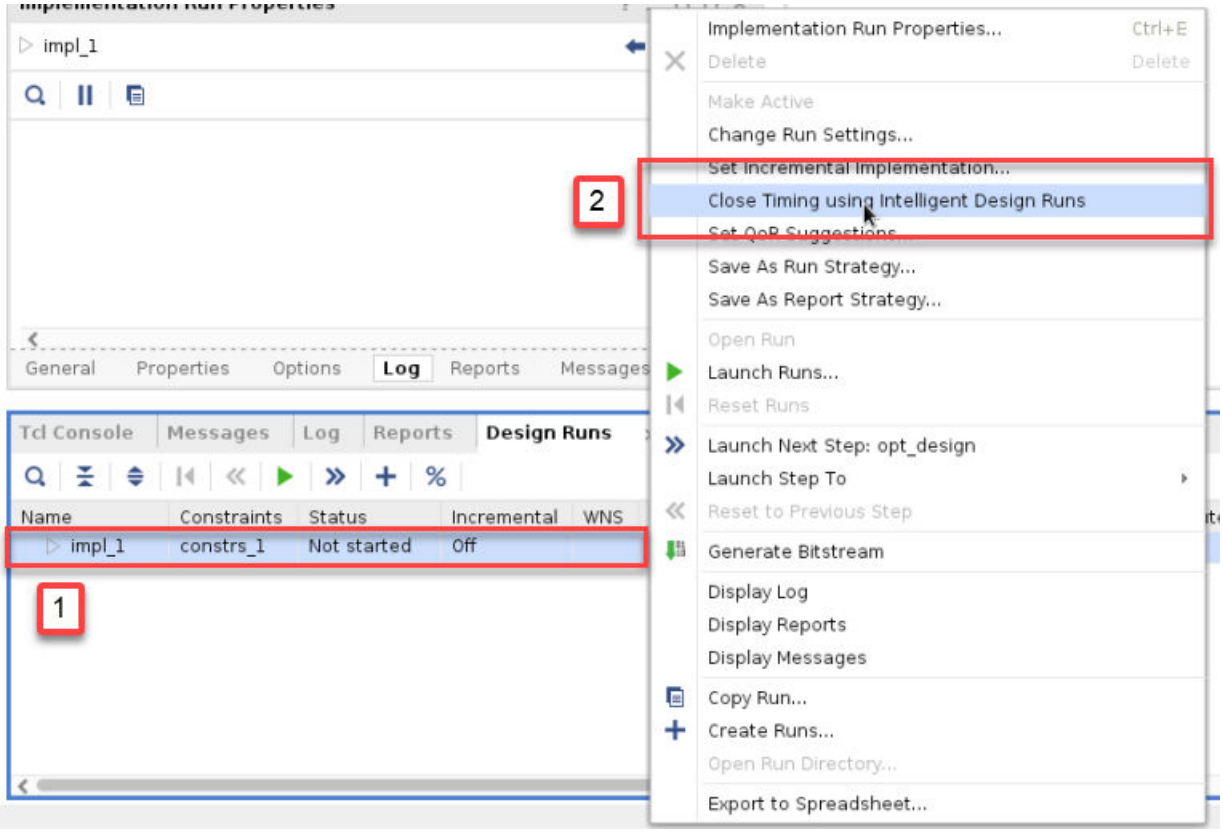
To enter the Last Mile Timing Closure stage from stage 1 or 2, the design must have the following:

- A fully routed run
- A WNS > -0.250
- An RQA score of 3 or greater

Creating an Intelligent Design Run

The intelligent design run (IDR) is created from a standard implementation run. In the Design Runs window, right-click on the implementation run and select **Close Timing using Intelligent Design Runs** as shown in the following figure.

Figure 217: **Close Timing Using Intelligent Design Runs Command**



The equivalent Tcl commands to create an intelligent design run are as follows:

```
create_run -flow {Vivado IDR Flow 2021} -parent_run <synth runName> <idr
runName>
set_property REFERENCE_RUN impl_1 [get_runs <idr runName>]
```

The REFERENCE_RUN property is used to copy Tcl hooks from an implementation run. Tcl hooks are applied at each implementation phase of the run. For example, if there is a pre-`opt_design` Tcl hook, it is executed every time before the `opt_design` command is called. This property is examined when the run is reset, so subsequent changes to the implementation run Tcl hooks are picked up. If you wish to add a Tcl hook to an IDR, first create an implementation run, add the Tcl hook, then create a new IDR.

Note: Pre- and post- `init_design` Tcl hooks are not currently supported.

Because directives are controlled by the IDR, there is no value in creating an IDR from a run that has an identical netlist, identical constraints, and identical Tcl hooks. There is consequently a restriction where only one IDR can be created from any given implementation run. If more IDRs are desired, alter the synthesis options to create a different netlist or modify the floorplan.

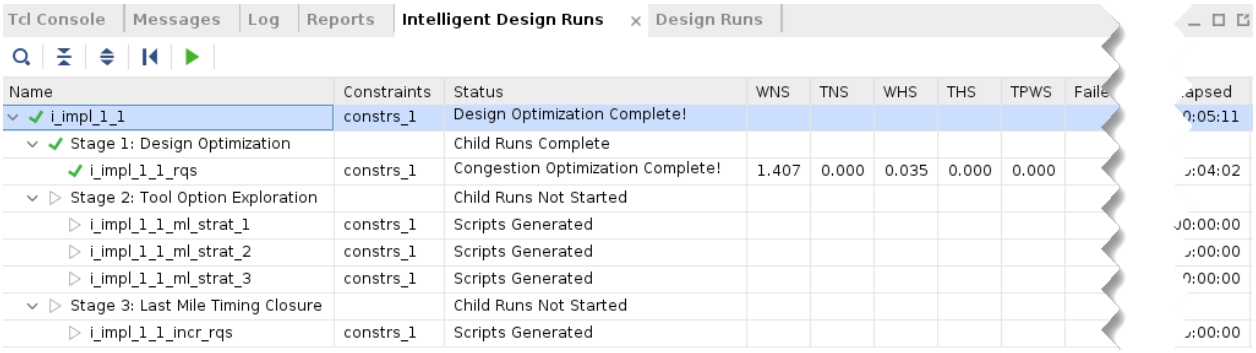
Note: It is not possible to create an IDR run directly from a synthesis run.

Flow Control within Intelligent Design Runs Window

The Intelligent Design Runs window, shown in the following figure, provides two functions:

- A context-sensitive right-click menu to access flow control and design analysis options
- Access to metrics such as WNS, TNS, WHS, and THS for the top-level and sub-level IDR runs

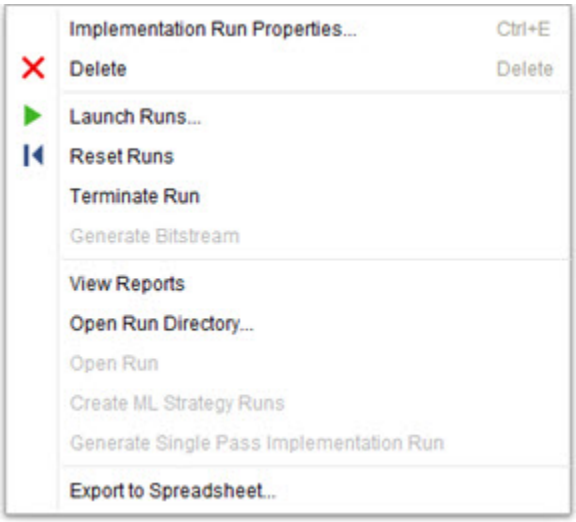
Figure 218: **Intelligent Design Runs Tab**



The metrics represent the best run metrics for that stage of the IDR. They are updated regularly throughout the flow.

The right-click menu is context sensitive, targeting the flow stage that is selected. Right-clicking at the top level gives the following options (which are a superset of right-clicking on any of the lower stages):

Figure 219: **Flow Control and Design Analysis Options Menu**



The menu options are described below.

- **Implementation Run Properties:** Opens the run properties for the IDR run. This is reduced from a normal run.
- **Delete:** Deletes the run.

  Note: Only one IDR run can be created from any given implementation run. To create a second run, the first must be deleted beforehand, or a second implementation run should be created and an IDR created from that.
- **Launch Runs:** Launches the runs.
- **Reset Runs:** Resets the IDR and deletes all files.
- **Terminate Run:** Terminates the selected run but does not delete the files in the run directory. This option is only available when a run is in progress.
- **Generate Bitstream:** Launches the run through to bitstream generation if the run has not started. If the IDR is complete but a bitstream is not generated, the option generates a bitstream from the best run routed checkpoint of the completed IDR. This option is unavailable while a run is in progress.
- **View Reports:** Opens the Intelligent Design Runs Reports window.
- **Open Run Directory:** Opens the run directory. Intermediate checkpoints and text reports can be accessed using this option.
- **Open Run:** Opens the selected run or the best run from the selected stage for design analysis. Only available on routed checkpoints.
- **Create ML Strategy Runs:** This option becomes available after the completion of stage 1 when a design still has a timing failure. ML strategies are automatically created and ready to use along with any APPLIED QoR suggestions. Selecting this option creates three runs that are equivalent to running stage 2. In cases where the design is changing, this option is preferred to using a single pass run that contains Last Mile, because it can better handle larger changes.
- **Generate Single Pass Implementation Run:** Creates a standard implementation run that sets up the RQS file and incremental checkpoints (if required) to create the same result achieved in IDR. This option is only available on a successfully completed IDR.

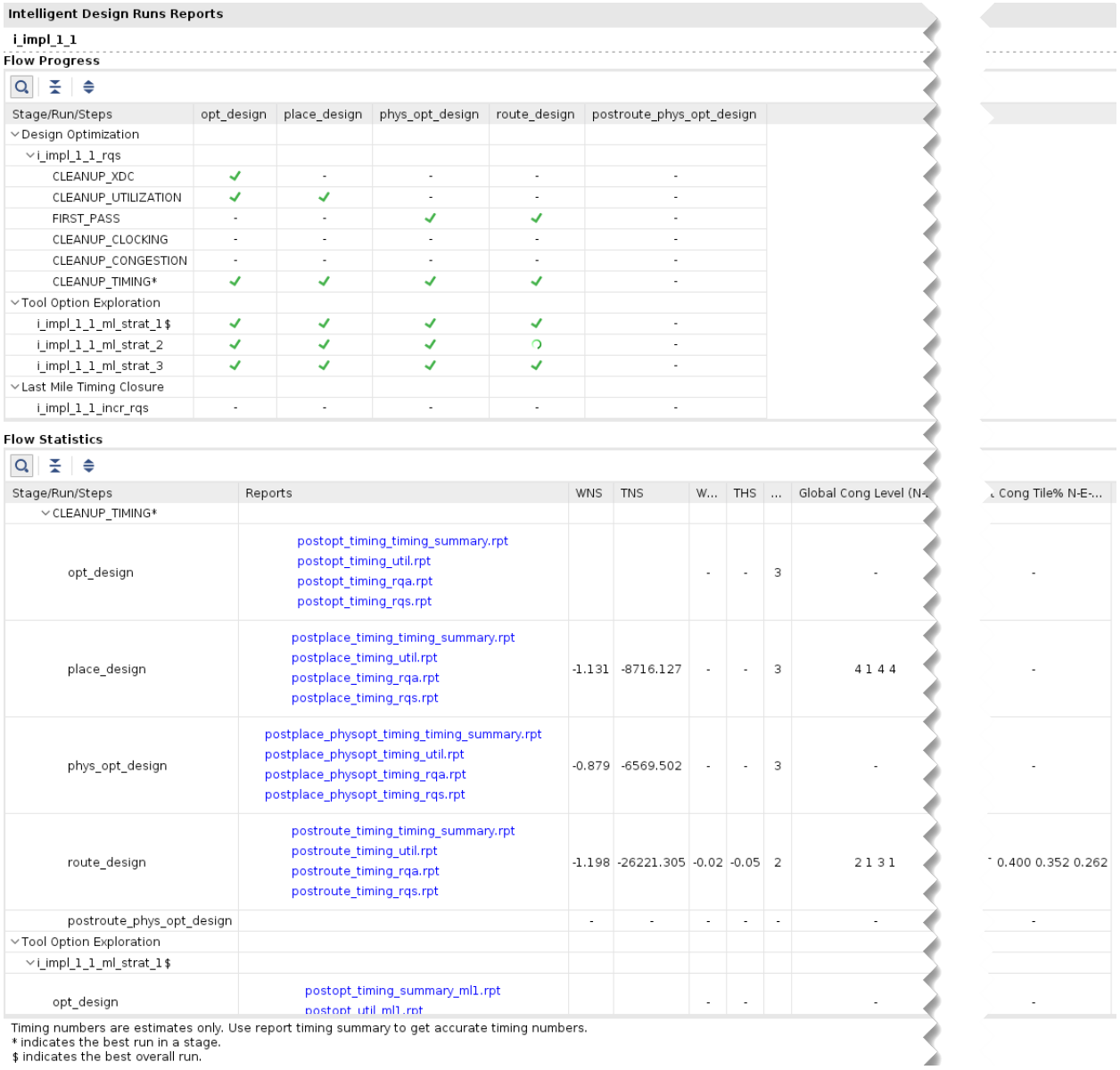
Intelligent Design Run Reports

The Intelligent Design Run Reports window, shown in the following figure, is split into two sections:

- **Flow Progress:**
  - Shows which stages have been run and which stages are currently running.
  - Indicates the best overall run ($) and the best run within a stage (*).
- **Flow Statistics:**

- Shows collected statistics on the design for timing and congestion information.
- Hyperlinks to the reports generated in the IDR.

Figure 220: **Intelligent Design Run Reports Window**



The following data is captured in an IDR:

- RQA score (captured throughout)
- Congestion (collected at post-place and initial route stages)
- Timing (collected at post-place, post-phys opt, and post-route stages only)

***Note*:** When stages have not been run or no data is collected for a particular metric, it is represented with a hyphen (-).

***Note*:** The timing numbers collected are typically estimates, and might differ slightly from running `report_timing_summary`.

The reports generated are fixed. There is no direct user control over the reports generated. Extra reports should be added using Tcl hooks. Alternatively, they can be generated after opening the checkpoints in the run directory. The equivalent text report is generated automatically in the top-level run directory. It has a fixed name of `idr_flow_summary.rpt`.

Intelligent Design Runs Recommendations for Non-Project Users

A Vivado project is required to use the intelligent design runs feature. This is because of the run management required. The following instructions explain the easiest way to create a post-synthesis project. They are applicable to users of the following flows:

- Non-project implementation runs
- Synthesis using earlier versions of Vivado or third party synthesis tools

The easiest way to access the intelligent design runs feature is to add a single full design DCP source to a post-synthesis project. This will provide the full netlist as well as all design constraints. When this project is created, the IDR can be launched following the steps in the Creating an Intelligent Design Run section.

1. Generate a single checkpoint from your existing implementation run. To do this, locate the `opt_design` call in the implementation run Tcl script, and write a checkpoint before this stage. The following is an example:

   ```
   write_checkpoint -force <PreOptDesign>.dcp
   opt_design -directive Explore ; ## FOR EXAMPLE ONLY ##
   ```

   Without user intervention, the earliest a full design checkpoint is written is after `opt_design`. However, the ideal case is to write a checkpoint *before* `opt_design` is run. In this case, a pre-`opt_design` Tcl hook should be used to write the checkpoint. Add the `write_checkpoint` line to a Tcl script and insert it either after `init_design` (`link_design`) or before `opt_design` if there is a conflict:

   ```
   add_files -fileset utils_1 -norecurse ./test.tcl
   set_property STEPS.INIT_DESIGN.TCL.POST [get_files ./test.tcl -of
   [get_fileset utils_1] ] [get_runs <ImplRun>]
   ```

   or

   ```
   set_property STEPS.OPT_DESIGN.TCL.PRE [get_files ./test.tcl -of
   [get_fileset utils_1] ] [get_runs <ImplRun>]
   ```

2. When the checkpoint is available, create a post-synthesis project (this can be done easily using the New Project wizard). The equivalent Tcl code for project creation is shown here:

```
create_project <ProjectName> <ProjectDirectory> -part <PartName>
set_property design_mode GateLvl [current_fileset]
add_files -norecurse <PreOptDesign>.dcp
```

For more information on setting up projects, refer to this link in *Vivado Design Suite User Guide: Design Flows Overview* (UG892).

Supported Families and Design Flows

The following table summarizes the supported device families and design flows for IDR in this release.

Table 18: **IDR Support Summary**

Item	Support Status
Supported Families	UltraScale, UltraScale+,Versal
Flows	Project Mode
DFX	Yes (excluding Versal Last Mile)
Vitis	No

QoR Suggestions

QoR suggestions are used to improve the ability of a design to meet timing through the following tactics:

- Adding switches to commands such as `opt_design`
- Adding properties to design objects such as cells and nets
- Full implementation strategies

The `report_qor_suggestions` command generates a report in either the AMD Vivado™ IDE or a text-based report. It can be used for the following purposes:

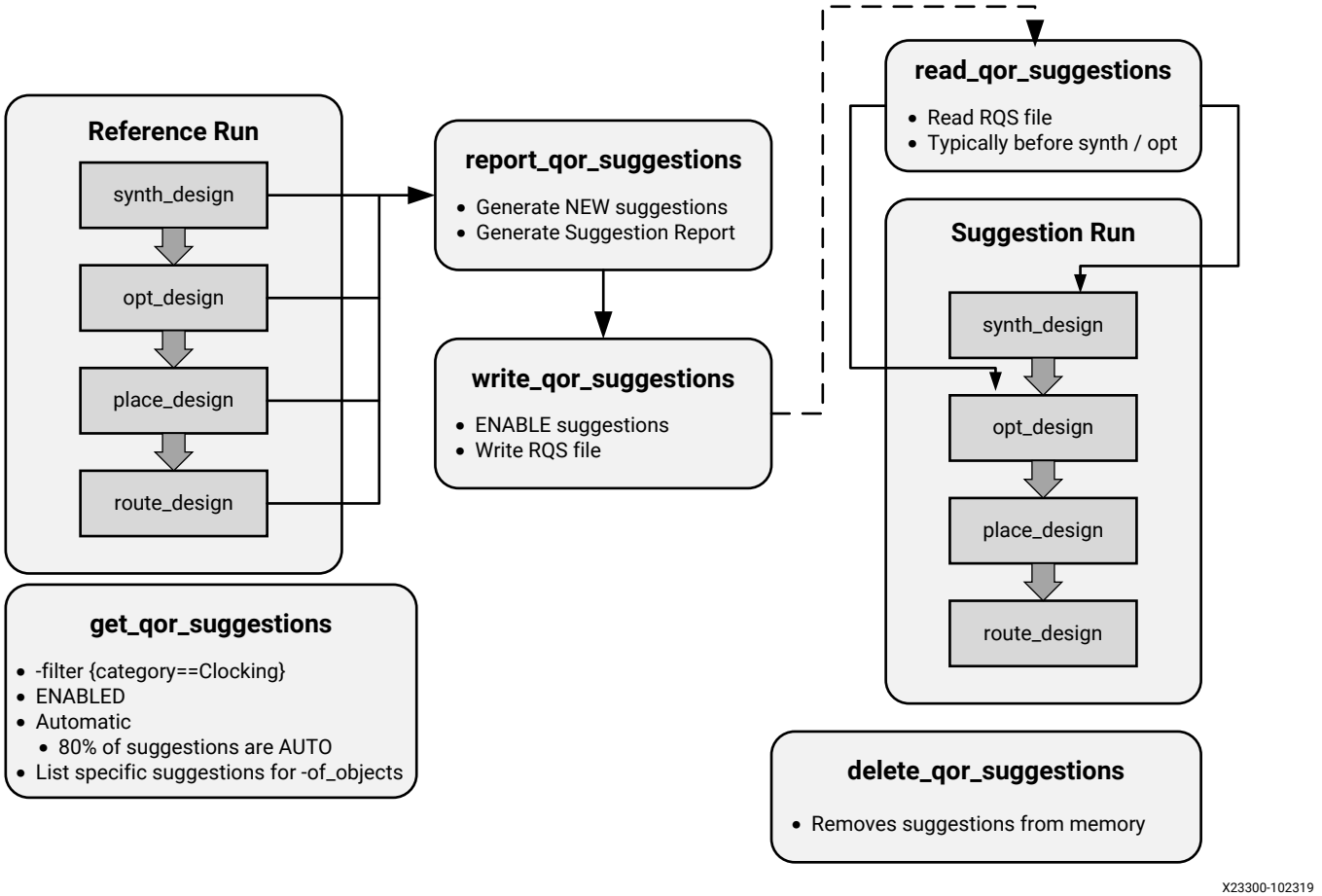
- Generating and viewing new suggestions on the current design in memory
- Viewing existing suggestions that have been read in using the `read_qor_suggestions` command

The `report_qor_suggestions` command can be run on a design loaded in memory at any time after synthesis. The suggestion objects generated consider many design characteristics and generate suggestions in the following categories:

- Clocking
- XDC
- Netlist
- Utilization
- Congestion
- Timing
- Strategies

The generated suggestions must then be fed back into the flow to take effect. Design stages are typically required to be rerun, as is shown in the following figure:

Figure 221: **Suggestion Flow**



Prior to generating new suggestions, a design must be loaded into memory. `report_qor_suggestions` can be run at any stage of the flow after synthesis. The returned suggestions are ordered based on importance, with the most important listed at the top of the report.

It only reports suggestions required to improve the QoR of the design. Sometimes placement or routing information is required before a suggestion can be issued. In addition, there are restrictions to ensure only suggestions that contain necessary design changes are generated.

- Netlist suggestions are generated based on analysis of the netlist. They identify common netlist structures that lead to timing failures later in the flow but they do not directly look at timing paths so can be generated on timing closed designs.
- Clocking suggestions generally need to be generated after placement, but there are some exceptions to this when accurate information is available before placement. With a few exceptions, they require a failing timing path.
- Timing suggestions are generated by examining the top 100 failing timing paths per clock group.
- Utilization suggestions are generated when it judges that the resource targeted by the suggestion is overused and does not result in an increase of a critical resource. These can be reported at any design stage.
- Congestion is only reported after placement. If a design is routed and timing is met, it does not report congestion suggestions as it is proven these are not having an impact on timing closure.
- Strategies, the final category, contains implementation strategies. These are generated using machine learning algorithms that analyze many design characteristics. The flow when using these objects is slightly different to that described above and is outlined in more detail later in this chapter.

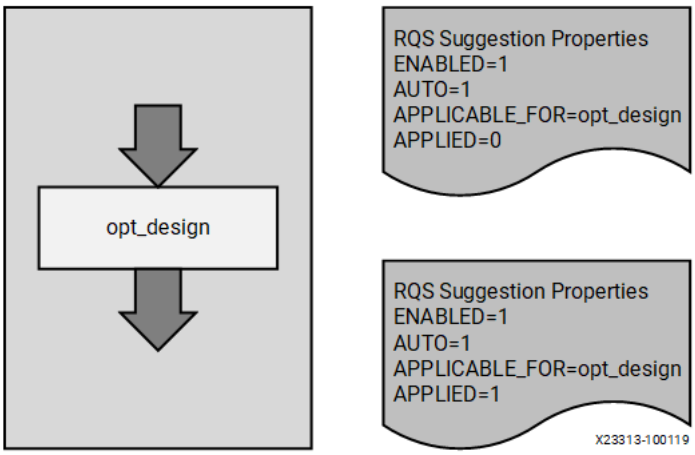
Executing Suggestions

Suggestions are executed in the suggestion run when the following criteria are met:

- The suggestion is ENABLED.
- The APPLICABLE_FOR stage must be run.
- The suggestion must be AUTO.

When a suggestion is executed, the APPLIED setting updates as is shown in the following figure:

Figure 222: **Suggestion Execution**



In the implementation flow, if a property contained in a suggestion is not applied correctly to the associated cell or net, FAILED_TO_APPLY will be set to 1. If a suggestion is partially applied, a new suggestion will be generated and the suggestion will be broken out into suggestions that have been applied and and those that failed to apply.

***Note*:** FAILED_TO_APPLY will not be set if the implementation tools reject the property later in the flow. FAILED_TO_APPLY is set only when the given property cannot be applied to the object. The application of the property is not tracked.

Suggestions can be executed in the same run as they were generated in if the APPICABLE_FOR stage is after the stage they were generated at. To do this, you must manually enable the suggestion first.

```
set_property ENABLED 1 [get_qor_suggestions <SuggID>]
```

When using this method, you must remember to write this suggestion to the RQS file when the run is complete to use this moving forward.

All Related Commands

There are five related commands when working with QoR suggestion objects:

Table 19: **All Related Commands**

Command	Function
report_qor_suggestions	Generates new suggestions. Reports on existing suggestions.
write_qor_suggestions	Writes suggestion objects to a file. Suggestions are ENABLED automatically during this process.
read_qor_suggestions	Reads suggestion objects from a file.
get_qor_suggestions	Returns QoR suggestion objects.

Table 19: **All Related Commands** *(cont'd)*

Command	Function
delete_qor_suggestions	Removes QoR suggestions from memory.
get_qor_checks	Returns a list of all the suggestions that can be generated by Vivado.

Applying the Suggestions

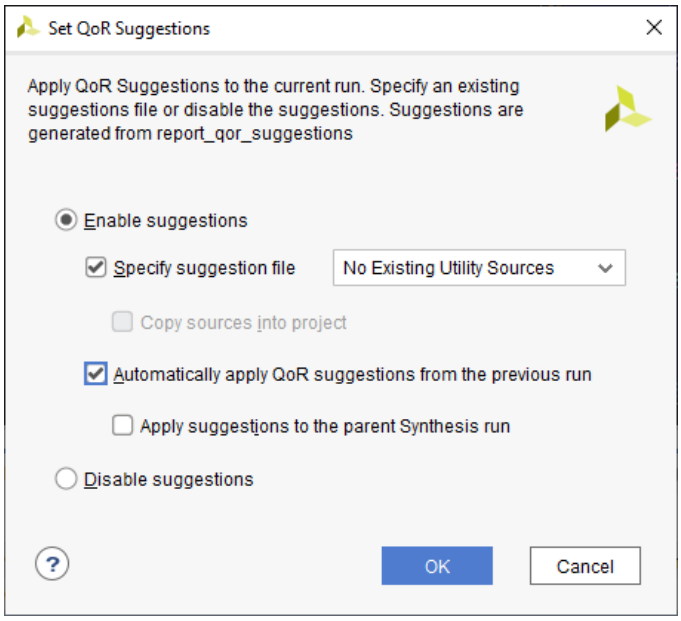
Suggestions are applied in the flow by reading in the suggestion object file and then running the commands. The following section covers additional features that make it easier to work with suggestion object files in project mode.

Project Mode

In project mode, after suggestions are written to an RQS file, the file is automatically added to the utility sources fileset in the project (`utils_1`). Store this file somewhere other than in the run directory, because it will be deleted when the run is reset. It should be in a unique directory per run to allow multiple runs to have their own suggestion file. The recommended location is: `<project_dir>/<project>.srcs/utils_1/<run_name>`.

When you are in a project, right-click on a run from the Design Runs window and select **Set QoR Suggestions**. It might be required to add this to both synthesis and implementation runs.

Figure 223: **Set QoR Suggestions Window**



Select **Enable Suggestions**. This allows you to select a suggestion file, the automatic run, or a combination of both. Selected suggestion files are added to the `utils_1` fileset if they are not added already.

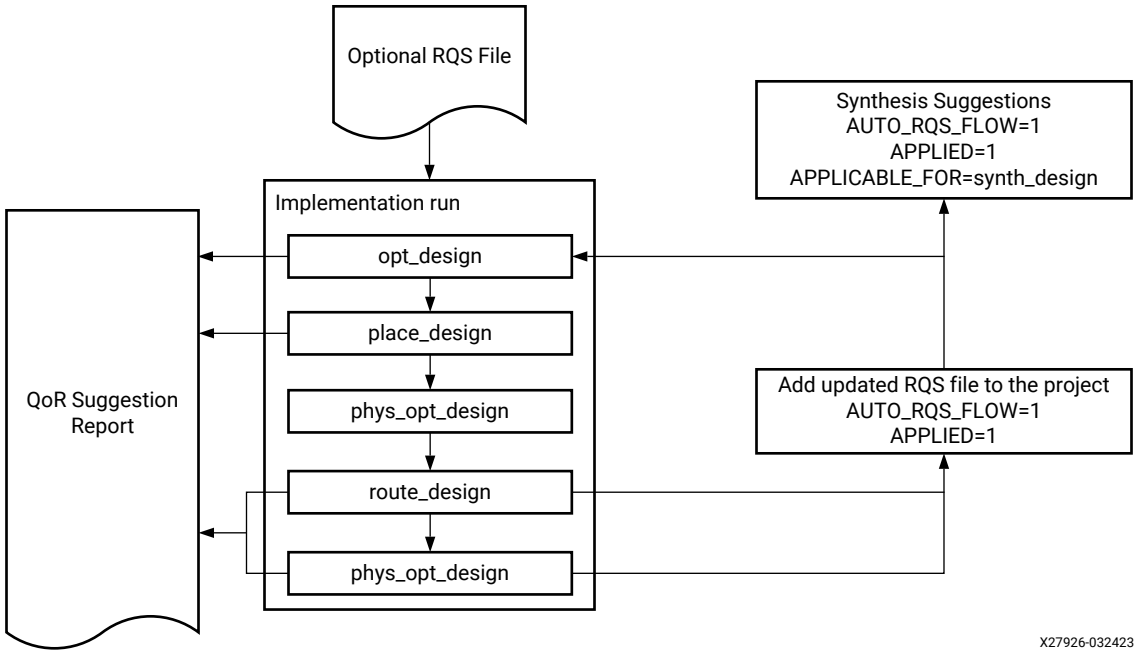
The equivalent Tcl commands for this flow are shown in the following code snippet:

```
write_qor_suggestions -of_objects [get_qor_suggestions \
{<NAME_1> <NAME_2>}] -file <fn.rqs>
add_files -fileset utils_1 <fn>.rqs
set_property RQS_FILES <fn>.rqs [get_runs <run name>]
```

Automatically Generating and Applying Suggestions

The automatic QoR suggestion flow generates and applies QoR suggestions for project-based runs. It is the easiest way to generate and apply QoR suggestions on a design that is readily changing. When automatic QoR suggestions are enabled, the standard implementation flow is adjusted as shown in the following figure.

Figure 224: **Implementation Flow with Automatic QoR Suggestions**



The initial run can be set to take a user-selected RQS file, or it can run without an RQS file. If a user RQS file is selected, it remains in place until new suggestions are generated. Any suggestions in the file are written into a `<top>_routed.rqs` file, removing the requirement to keep the original file.

The `report_qor_suggestions` command is run after `opt_design`, `place_design` and either `route_design` or post route `phys_opt_design` if that command is enabled.

Most suggestions are written to an RQS file at the end of the implementation run so that they can be applied on the next implementation run. The suggestions that get written to the RQS file are:

- New suggestions with the property AUTO_RQS_FLOW==1
- Older suggestions with the property APPLIED==1

There is also a suggestion RQS_CLOCK-1 that can be generated at `opt_design` and applied in the same run.

To examine the suggestions, the `report_qor_suggestions` report is available in the Reports tab for each of the points where the command is run.

When the run is reset, suggestions are copied from the implementation run directory to another location and added to the `utils_1` fileset. This location is part of the sources directory by default but can be modified to any other location.

Suggestions can also be applied to the parent synthesis run if one exists. Only one child implementation run can supply synthesis suggestions. If more than one child implementation run is selected, the latest one is used.

The following properties are used on the first class *run* object:

Table 20: **Auto QoR Suggestion Flow Run Properties**

Property	Values	Description
RQS_FILES	RQS file name	Allows you to input user RQS file.
AUTO_RQS	• **1:** On • **0:** Off	Enable the automatic RQS flow.
AUTO_RQS.DIRECTORY	<directory>	Allows the specification of an alternative directory to copy the RQS file back to.
AUTO_RQS.SUGGESTION_RUN	Implementation run name that generates the RQS file	Applied to the synthesis parent run.

An example of the use of these properties in Tcl is shown in the following code snippet:

```
set_property AUTO_RQS 1 [get_runs impl_1]
set_property RQS_FILES C:/temp/test.rqs [get_runs impl_1]
set_property AUTO_RQS.DIRECTORY C:/project_name/sources/rqs/impl_1
[get_runs impl_1]
set_property AUTO_RQS.SUGGESTION_RUN impl_1 [get_runs synth_1]
```

Automatic QoR Suggestions and ML Strategies

If ML strategies exist before the run is started, the RQS file can be set using the RQS_FILES property. At the end of the flow, strategy suggestions are added to the new RQS file.

If ML strategies do not exist, they are automatically generated for the user in the `<run_name>/MLStrategy` directory during the suggestion generation process.

Automatic QoR Suggestions and the Incremental Flow

When the incremental flow is enabled, only new incremental-friendly suggestions can be applied in the subsequent steps. This limits the amount of change that a suggestion can achieve, so large timing improvements are unlikely to be seen.

If suggestions are applied at `opt_design` and the incremental flow is initiated after, larger impact suggestions can be applied. It will likely be necessary to disable the suggestion because it might cause timing degradation.

If the automatic incremental flow decides to use the default flow because of a poor quality reference checkpoint, all suggestions can be applied.

Disabling Unwanted Suggestions

Because there is limited control over the suggestions that are automatically added to the suggestion file, if a suggestion is causing worse timing results, it can be disabled using Tcl scripts at the pre-`opt_design` step. The following code snippet is an example of disabling the `RQS_TIMING-1` suggestion:

```
set_property ENABLED 0 [get_qor_suggestions RQS_TIMING-1*]
```

Non-Project Mode

The creation of the RQS file is the same as project mode. This writes the suggestion file that must then be added to the run using `read_qor_suggestions`. The `read_qor_suggestions` command should be run either before `synth_design` or before `opt_design`.

The equivalent Tcl commands for this flow are as follows:

```
read_vhdl <some_file>.vhd
read_qor_suggestions all_enabled_suggestions.rqs
synth_design -top <top> -part <part>
opt_design
...
route_design
report_qor_suggestions -file design_rqs_routed.rpt
write_qor_suggestions -force all_enabled_suggestions.rqs
```

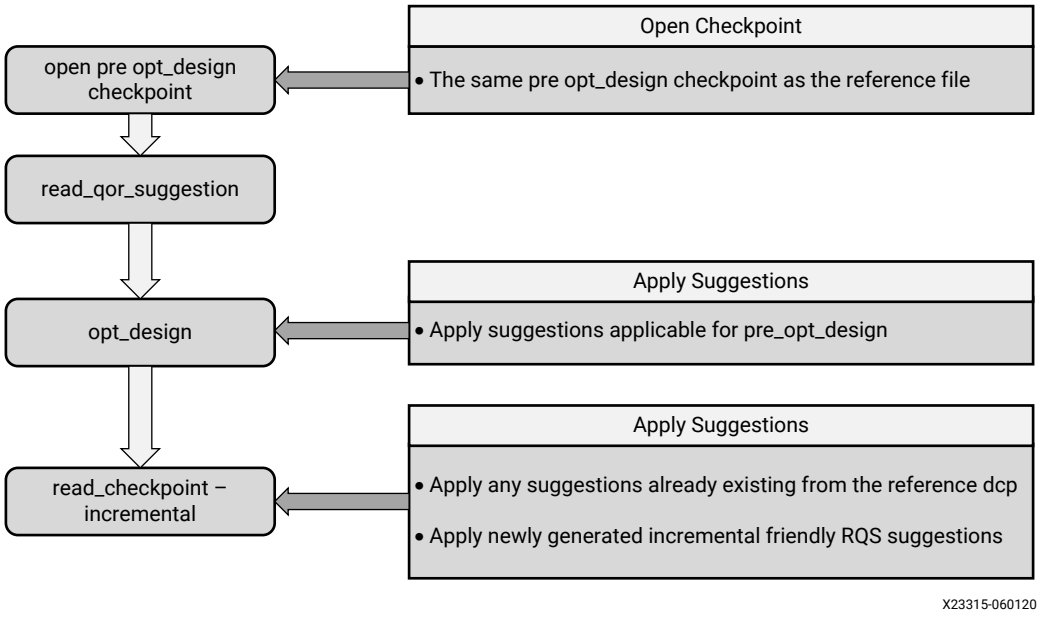
Note: The `report_qor_suggestions` command cannot be run until a design is loaded.

RQS in the Incremental Flow

When your design is very close to timing closure (typically the WNS is less than -250ps), enable the incremental flow with RQS suggestions. This allows you to achieve timing closure and save iteration time by taking advantage of both the incremental flow and RQS suggestions. `report_qor_assessment` indicates when to use this flow in the Flow Guidance section.

In the run, the suggestions generated from the reference routed DCP are read in before running the incremental flow commands. The rest of the flow is applied automatically for you. Vivado understands which suggestions to apply at what stage in the flow by differentiating which suggestions are newly GENERATED and which ones were APPLIED in the reference run. This is shown in the following figure:

Figure 225: **Incremental Flow**



In the case where the incremental flow is run, the suggestions that have been applied in the reference are read from the reference DCP and applied regardless of whether they are enabled or not. The ENABLED property is ignored because it is important to replicate the reference checkpoint as closely as possible.

Next, incremental friendly suggestions from the RQS file are applied. These must be enabled, which is done automatically during `read_qor_suggestions`. The suggestions are applied during `read_checkpoint -incremental` and not at the APPLICABLE_FOR stages. Therefore, suggestions should not be read or enabled after this stage because they will be ignored. Any new non-incremental friendly suggestions in the RQS file will be ignored (existing non-incremental friendly suggestions applied in the reference will be applied).

Special care should be taken when applying suggestions that are applicable for `opt_design`. Because this is before the flow is aware it is in incremental mode, it is not possible to manage these suggestions automatically. You must ensure that the existing suggestions that are applied in the reference are also applied in the incremental run and that no new suggestions are applied. If it is desirable to apply these suggestions, the reference should be updated.

In the case where the incremental flow reverts to the default flow, usually due to a negative change, all suggestions will be executed from the RQS file. For this reason, before launching the next incremental run, you must export *all* suggestions to the RQS file and not only the incremental friendly ones.

Before adopting this flow, note the following prerequisites:

- The device part for the reference run and the incremental run should match.
- The reference checkpoint should be a post-route checkpoint.
- The same directive should be used for `opt_design` in the reference and incremental runs.
- The design should not have major design issues such as high congestion, unbalanced clocking, or an RQA score of less than 4.
- The suggestions should be regenerated from the reference checkpoint.
- Newly generated suggestions will only be applied if they are *incremental friendly*. If suggestions are not incremental friendly, they will only be executed if the flow reverts back to the default flow. If this does not happen they will be ignored.
- Newly generated suggestions must be generated from the reference checkpoint. This check ensures suggestions do not impact paths that have their timing resolved (for example, at post-route `phys_opt_design`).

An example of the commands required to run the flow is shown here:

- **Reference:**

```
# Generate RQS suggestions from the reference DCP
open_checkpoint reference_routed.dcp
report_qor_suggestions -file postroute_rqs.rpt
write_qor_suggestions -force ./post_route.rqs
```

- **Incremental:**

```
# RQS-Incremental Run:
open_checkpoint <pre_opt.dcp>
read_qor_suggestions ./post_route.rqs
# opt_design directive must be same as the reference run
opt_design -directive {same directive as reference run}
read_checkpoint -incremental reference_routed.dcp
# place_design is running in TimingClosure mode
place_design
```

```
# phys_opt_design is optimized for incremental
phys_opt_design
# route_design is running in TimingClosure mode
route_design
write_checkpoint postroute.dcp
```

Automatic Removal of Suggestions

To prevent suggestions from accruing in number excessively, Vivado Design Suite carries out automatic management of suggestions. It will delete suggestions that are identical to previously generated suggestions at the time new suggestions are generated.

Viewing Suggestions in Tcl or Text Format

Suggestion objects are stored in binary: as a consequence, the only way to read the suggestion is to load the design, read the suggestions, and run `report_qor_suggestions`. Support for viewing and executing suggestions in Tcl is available for users who do not wish to use the object flow.

To write out suggestions in Tcl, use the following command:

```
write_qor_suggestions -tcl_output_dir <outputDir>
```

Running this command outputs one or more Tcl files to the stated directory. This option is not available in the Vivado IDE.

For the Tcl scripts generated for the implementation commands, add them to implementation runs using the Tcl Pre or Post hooks, typically before opt_design is best to use. They can be read using Tcl source command in a non project flow. For the synthesis Tcl scripts, they must be added to the constraints set in project mode (update the file filter to Tcl in order to see them), or in non project mode accessed using `read_xdc -unmanaged <Tcl file>`.

When the objects are in Tcl, it is the user's responsibility to maintain the Tcl to remove objects that are no longer required and apply only the suggestions that they want to apply. Any newly generated Tcl scripts will need to be appended to the existing Tcl scripts.

Suggestions that are entered using Tcl are no longer reported by `report_qor_suggestions`.

Supported Families and Flows

The following table summarizes the supported device families and design flows for QoR suggestions in this release.

Table 21: **QoR Suggestions Support Summary**

Item	Support Status
Supported Families	UltraScale, UltraScale+, Versal adaptive SoC
Flows	Project mode, non-project mode
DFX	Yes
IP OOC Synthesis	No
Vitis	No

Supported Suggestions

There are over 100 QoR suggestions that Vivado can generate, with over 80 being able to automatically execute without user edits required to constraints or RTL. To see a list of all the suggestions that can be generated, the `get_qor_checks` command can be run. The properties attached to the returned objects can be inspected. The objects returned by this command are not generated suggestions and the objects themselves can not be written.

The `-family` switch can be used to show only suggestions that are applicable to a certain family. The following is an example to see the suggestions available for the family of a currently opened design:

```
get_qor_checks -family [get_property FAMILY [get_parts [get_property PART
[current_design]]]]
```

The `-filter` switch allows users to filter based on object properties. Useful properties to filter on are:

- CATEGORY
- AUTO
- INCR_FRIENDLY

The output of this command should be processed using Tcl. An example of how to view the IDs and description of all the AUTO checks is as follows:

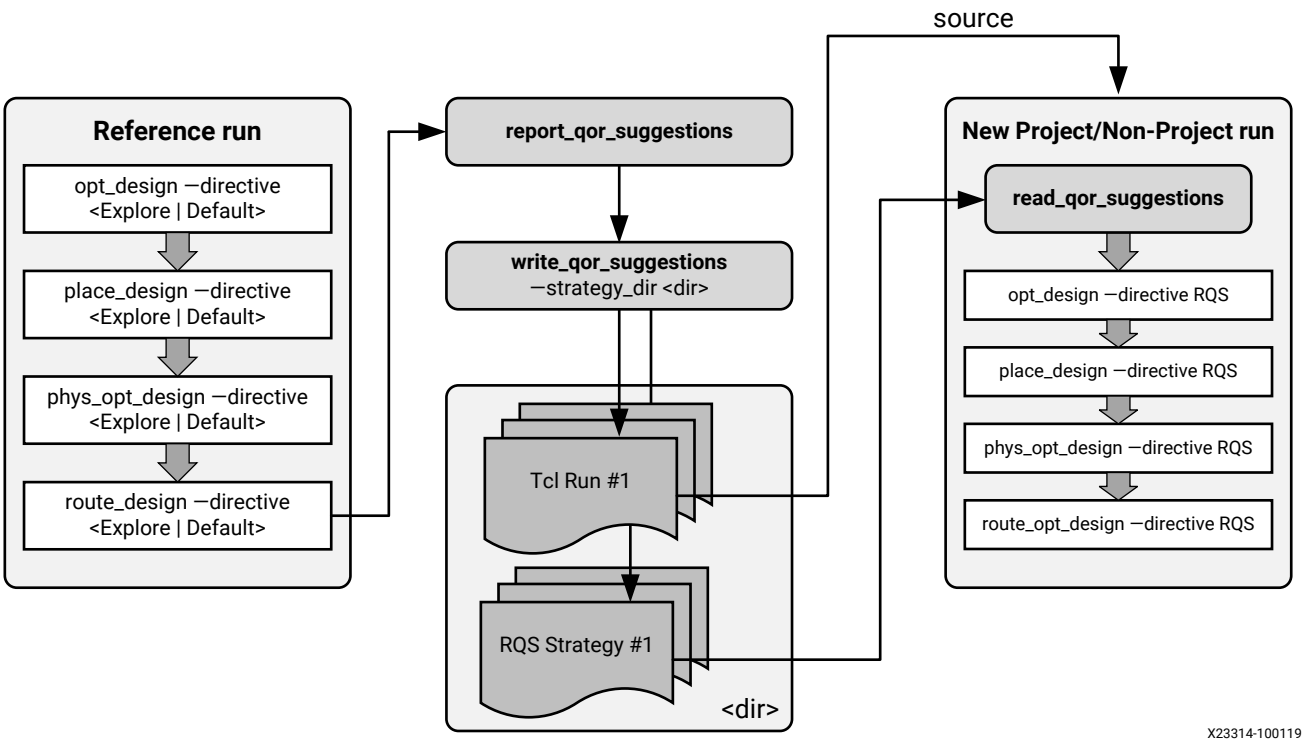
```
foreach sugg [lsort -dict [get_qor_checks -filter {AUTO==1}]] {
  set ID [get_property ID $sugg]
  set DESCRIPTION [get_property DESCRIPTION $sugg]
  puts "[format %-16s $ID]: $DESCRIPTION "
}
```

Strategy Suggestions

Typical timing closure strategies involve running many implementation strategies and picking the best one to take to the lab. ML strategies are an alternative to this, requiring you to run only three strategies to achieve a similar QoR benefit. They use machine learning to examine features from a post-route design to predict the performance of different strategies on the same design. The best three strategies are captured in RQS files generated by `report_qor_suggestions` (and `write_qor_suggestions`) and can be applied moving forward. Considerably less server power is required as a result.

When the directive is set to RQS on implementation commands, the command references the RQS file for the directive and other tool command options. The flow is shown in the following figure:

Figure 226: **Strategy Suggestion Flow**



There are four key points to this flow:

1. The `report_qor_suggestions` command must be run on a fully routed design that is generated using either `Default` or `Explore` directives. For complete details about the requirements, see ML Strategy Availability.

2. The `write_qor_suggestions -strategy_dir <dir>` command generates the required RQS files in the directory specified. By default, three strategies are generated. For each strategy generated, a single RQS file contains all the suggestion objects as well as the strategy suggestions object. The RQS file specified using `write_qor_suggestions -file <fn>.rqs` can be discarded as the information is replicated in each strategy RQS file.

   ***Note*:** To generate more strategies, increase the number using the following command:

   ```
   report_qor_suggestions -max_strategies <n>
   ```

3. The generated RQS file must be read in to the new implementation run.
4. The directive RQS must be set and the script must contain a call to `opt_design`, `place_design`, `phys_opt_design` and `route_design`.

Project Application of Strategy Suggestions

In project mode, the process of generating and applying suggestions is captured in the following steps:

1. Generate QoR suggestion files containing strategies (and, optionally, other QoR suggestions).
2. Create the new implementation runs, read the QoR suggestion file, and set the directives to RQS.

There are various ways integrated into the IDE that accomplish step 1. They each differ based on the flow used and the other QoR suggestions that are added to the RQS file. The following table describes how each feature works and its impact on non-strategy QoR suggestions.

Table 22: **Procedures to Generate ML Strategies**

Flow	RQS File Creation	Other QoR Suggestions
Standard implementation run	Right-click in the Design Runs window.	APPLIED suggestions are added by default. Newly generated AUTO suggestions are added optionally.
With auto RQS enabled	Automatic.	APPLIED suggestions are added by default. New AUTO suggestions are added by default.
Report QoR suggestions from an open design	Strategies are written automatically containing selected suggestions during the writing of suggestions.	Individually selected by the user.
Intelligent design runs	Automatic	Includes APPLIED suggestions from stage 1 IDR. Equivalent to stage 2 runs.
Report QoR suggestions in post-route Tcl hook	Manual	Individually selectable by the user. Use `-strategy_dir ./MLStrategy`.

Each of these options leads to a single way to create and set up the runs in step 2. In each of these items, an `MLStrategy` directory is created in the run directory. It contains three RQS files. When these files are seen, the `create_rqs_runs` option is enabled in the Design Runs right-click menu.

When run, this option automatically creates three new implementation runs and links them with the reference run. Only three ML strategies are allowed per reference. If you want to recreate them, delete them first and then rerun. Minor design changes should not require regeneration of the strategies.

The equivalent Tcl command to create new ML strategy runs from an implementation run `impl_1` is as follows:

```
create_rqs_runs -referene_run [get_runs impl_1]
```

Non-Project Application of Strategy Suggestions

In *non-project* mode, an example Tcl script is provided in the `-strategy_dir` directory. This script shows how the RQS file must be read, as well as the directives for the implementation commands set to RQS. These scripts are intended to be an example on a design loaded into memory in the pre-`opt_design` stage. They do not contain any reporting or writing of checkpoints.

Floorplanning

This section discusses Floorplanning and includes:

- About Floorplanning
- Understanding Floorplanning Basics
- Using Pblock-Based Floorplanning
- Locking Specific Logic to Device Sites
- Floorplanning With Stacked Silicon Interconnect (SSI) Devices

About Floorplanning

Floorplanning can help a design meet timing. AMD recommends that you floorplan when a design does not meet timing consistently, or 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 slack (TNS, WNS) by reducing the average route delay. During implementation, the timing engine works on resolving the worst setup violations and all the hold violations. 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 Vivado synthesis use:

- `synth_design -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.

Understanding 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.

You will achieve the greatest improvements by fixing the worst problems or the most common problems. For example if there are outlier paths that have significantly worse slack, or high levels of logic, fix those paths first. The **Reports → Timing → Create Slack Histogram** command can provide a view of outlier paths. Alternatively, if the same timing endpoint appears in several negative slack paths, improving one of the paths might result in similar improvements for the other paths on that endpoint.

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 leaf cells 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: *Use detailed gate-level floorplanning as a last resort.*

Information Reuse

Reuse information from a design that met timing. Use this flow if the design does not consistently meet timing. To reuse information:

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

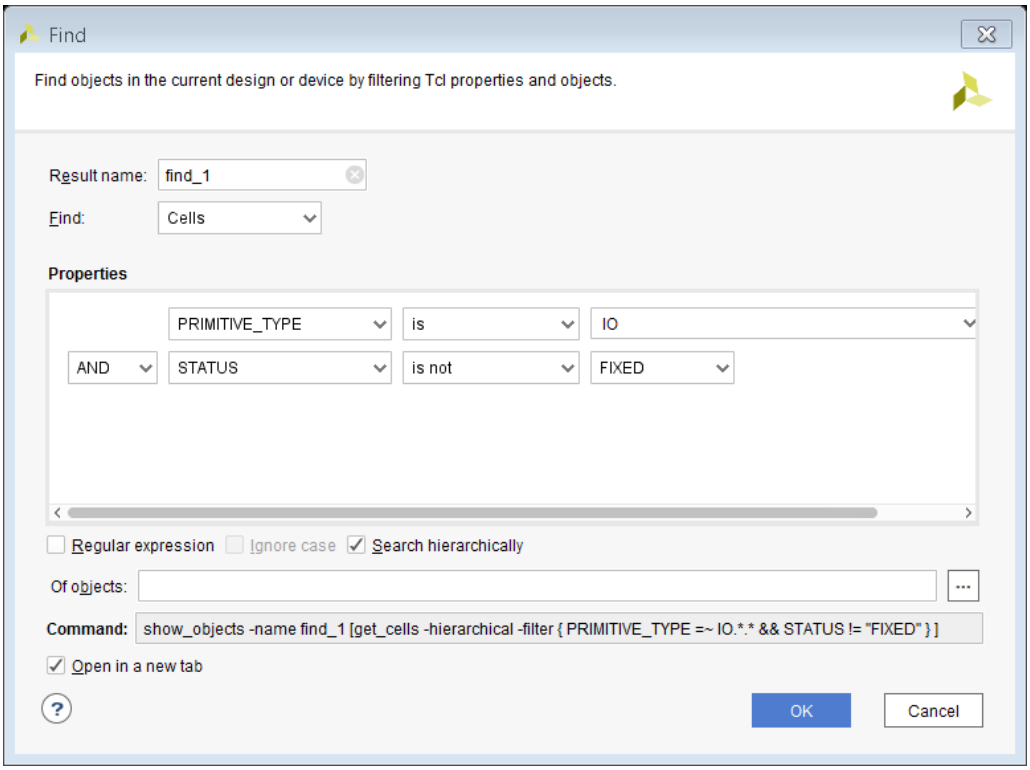
      TIP: *On a computer with multiple monitors, select Open Implementation in New Window to open a design in a new window.*

2. Look for the differences between the two designs.
   a. Identify some failing timing paths from `report_timing_summary`.
   b. On the design that is meeting timing, run `report_timing` in `min_max` mode to time those same paths on the design that meets timing.
3. Compare the timing results:
   a. Clock skew
   b. Datapath delay
   c. Placement
   d. Route delays
4. If there are differences in the amount of logic delay between path end points, revisit the synthesis runs.

Review I/O and Cell Placement

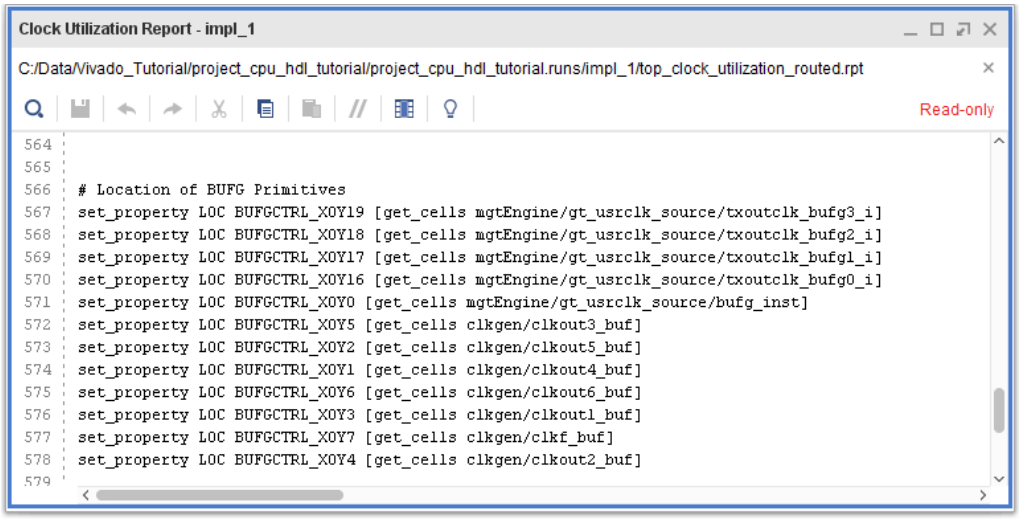
Review the placement of the cells in the design. Compare two I/O reports to review the I/O placement and I/O standards. Make sure all the I/Os are placed. A simple search finds all I/Os without fixed placement as shown in the following figure.

Figure 227: **I/O Is Not Fixed**



If clock skew has changed between the runs, consider re-using the clock primitive placement from the run that met timing. The Clock Utilization Report lists the placement of the clock tree drivers, as shown in the following figure.

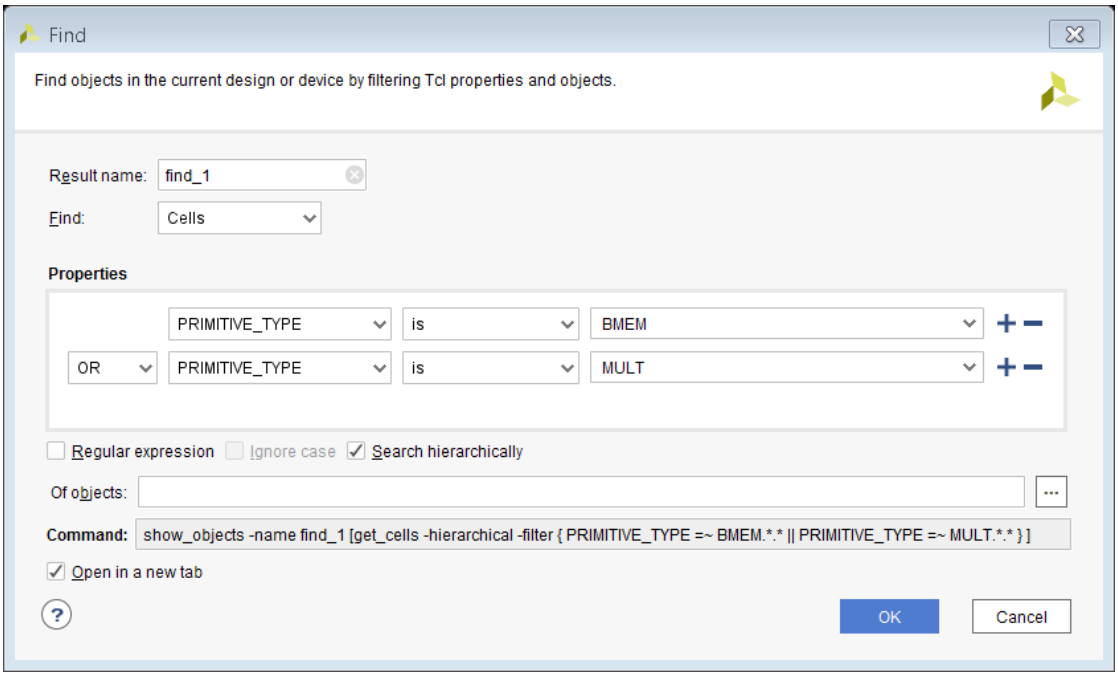
Figure 228: **Clock Locations**



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

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

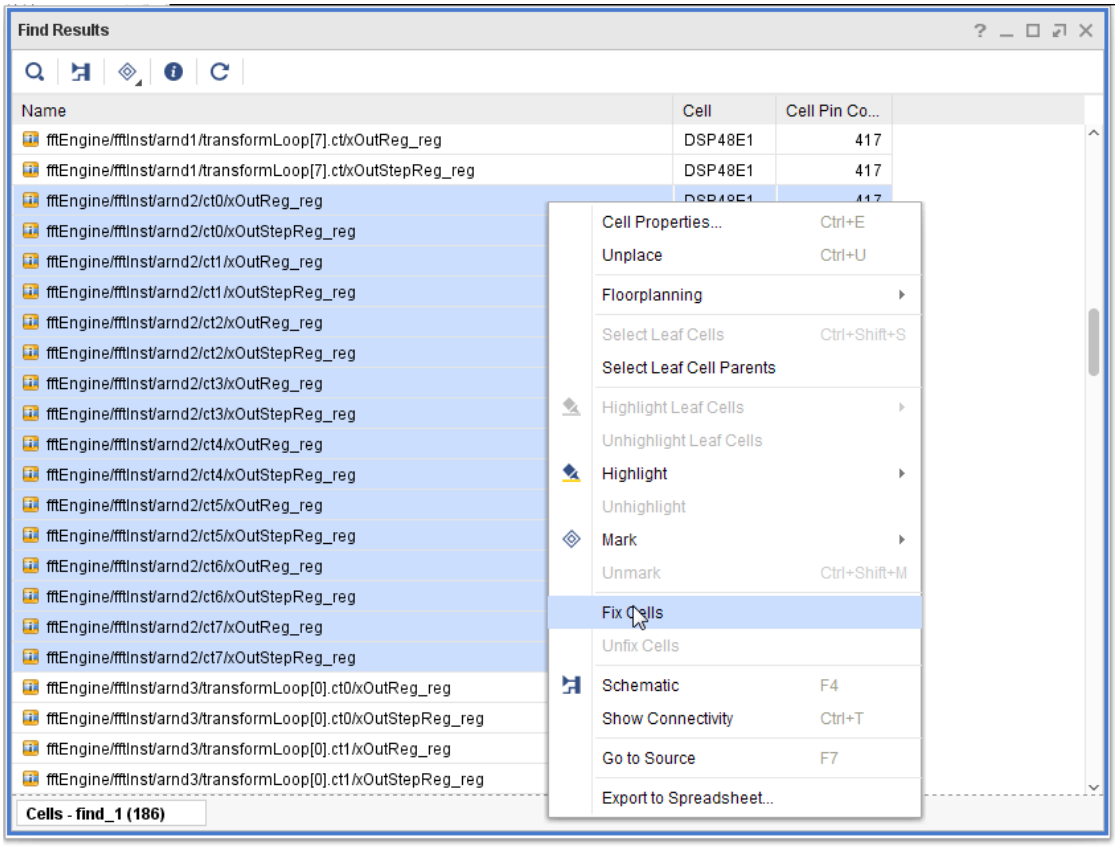
Figure 229: **DSP or RAM**



Adding Placement Constraints

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

1. Select the macros from the find results.
2. Right click and select **Fix Cells** (shown in the following figure).



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

Reusing Placement

It is fairly easy to reuse the placement of I/Os, Global Clock Resources, Block RAM macros, and 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 reuse the placement for sections of the design that are likely to change.*

Reusing Placement with Incremental Compile

Incremental Compile allows reuse of place and route data from a previous run. To set it up, simply reference an existing placed or routed DSP before `place_design`. It is possible to reuse a full design, a hierarchy level, or a cell type like DSPs or block RAMs. Incremental Compile also automatically handles changes made to parts of a design.

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

Floorplanning Techniques

Consider gate-level 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 at a global level, and the placer does the detailed placement. Hierarchical floorplanning has the following advantages over gate-level floorplanning:

- Hierarchical floorplan creation is fast compared to gate-level 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.

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 cells.
- Has comprehensive knowledge of the cells and timing paths.
- Generally does a good job of fine grain placement.

Manual Cell Placement

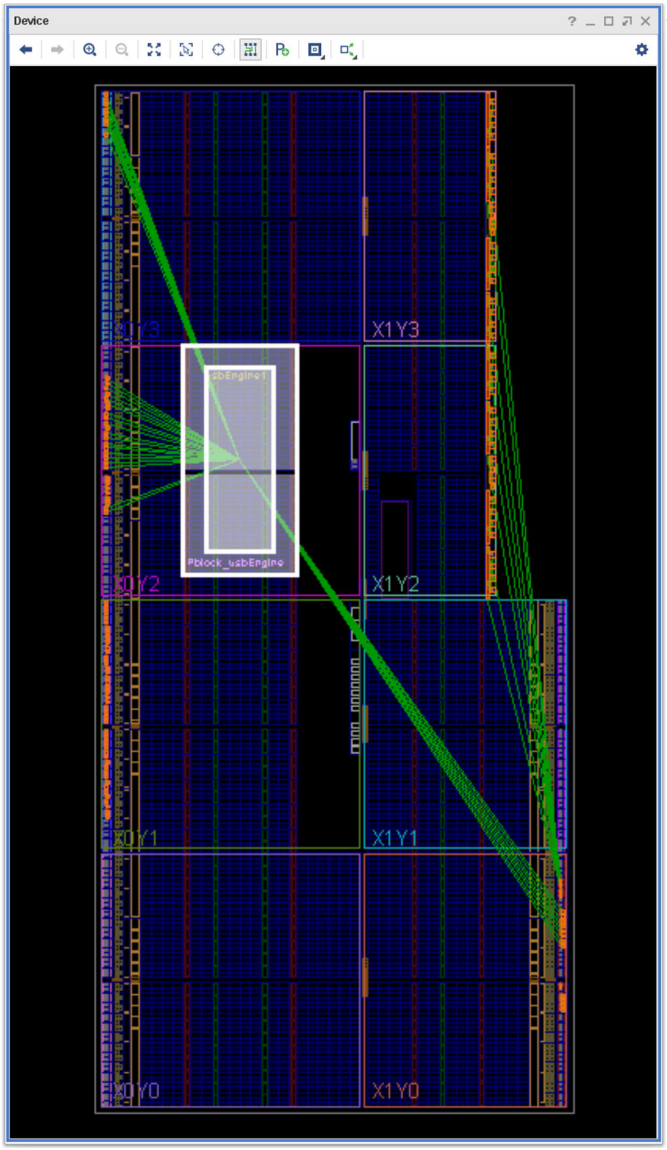
Manual cell placement obtains the best performance from a device. When using this technique, designers generally use it only on a small block of the design. They may hand place a small amount of logic around a high speed I/O interface, or hand place Block RAMs and DSPs. Manual placement can be slow.

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

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

TIP: *Place blocks that communicate with I/Os near their I/Os. If the pinout is pulling a block apart, consider pinout or RTL modification.*

Figure 230: **I/O Components Pulling Design Apart**



The floorplan shown in the previous figure might not help timing. Consider splitting the block apart, changing the source code, or constraining only the Block RAMs and DSPs. Also consider unplacing I/O registers if external timing requirements allow.

The Pblock mentioned in this section is represented by the XDC constraints:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells -quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add
{SLICE_X8Y105:SLICE_X23Y149}
```

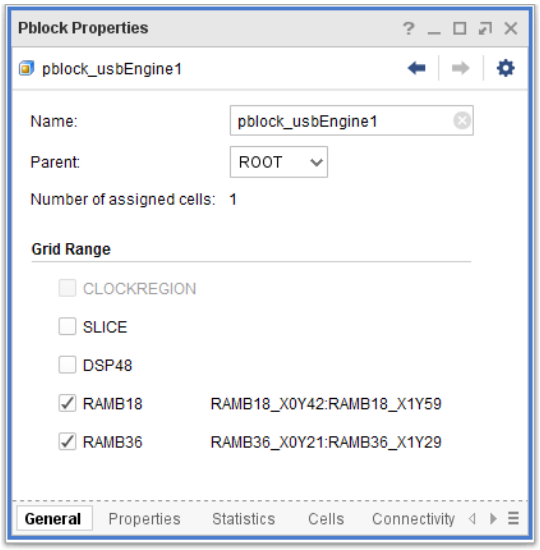
```
resize_pblock [get_pblocks Pblock_usbEngine] -add {DSP48_X0Y42:DSP48_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB18_X0Y42:RAMB18_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB36_X0Y21:RAMB36_X1Y29}
```

***Note*:** By default, the Pblocks are treated as soft throughout the implementation flow after the floorplanning stage. To force a hard Pblock, the Pblock property IS_SOFT should be set to 0:

```
set_property IS_SOFT 0 [get_pblocks Pblock_usbEngine]
```

The first line creates the Pblock. The second line (`add_cells_to_pblock`) assigns the level of hierarchy to the Pblock. There are four resource types (SLICE, DSP48, RAMB18, RAMB36) 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.

Figure 231: **Pblock Grids**



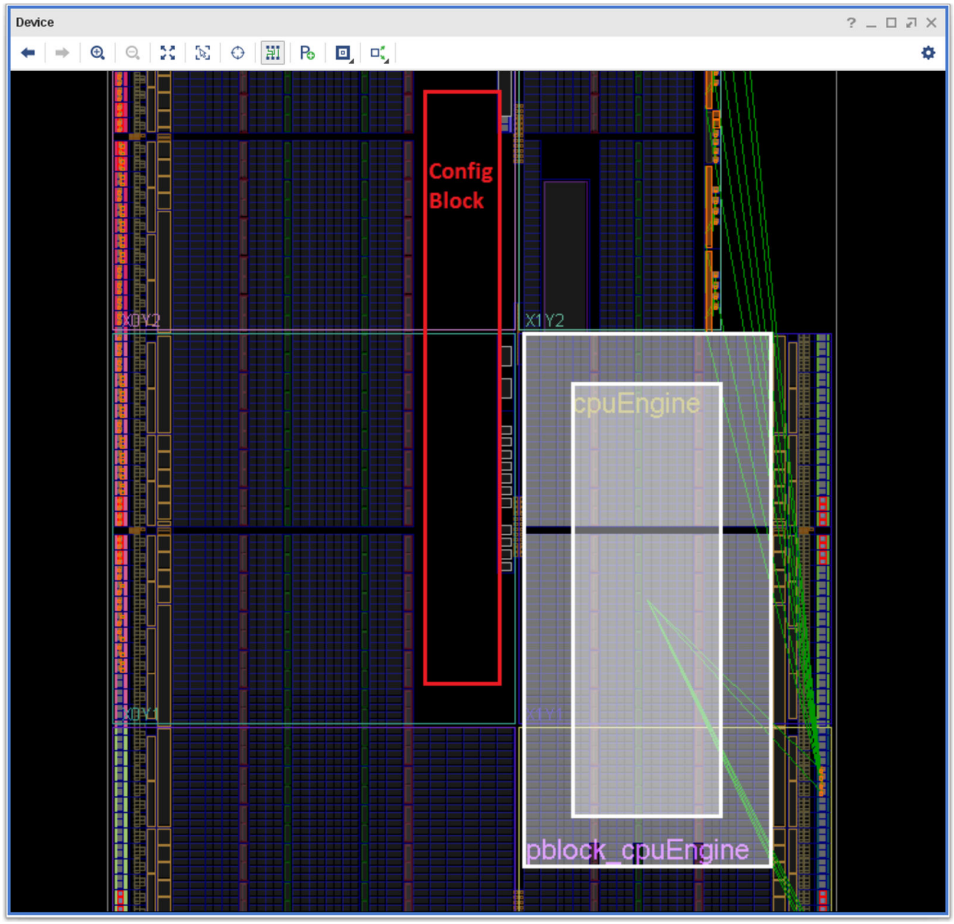
The resulting XDC commands define the simplified Pblock:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells -quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB18_X0Y42:RAMB18_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB36_X0Y21:RAMB36_X1Y29}
```

The Block RAMs are constrained in the device, but the slice logic is free to be placed anywhere on the device.

TIP: *When placing Pblocks, be careful not to floorplan hierarchy in such a manner that it crosses the central config block.*

Figure 232: **Avoiding the Config Block**



Using Pblock-Based Floorplanning

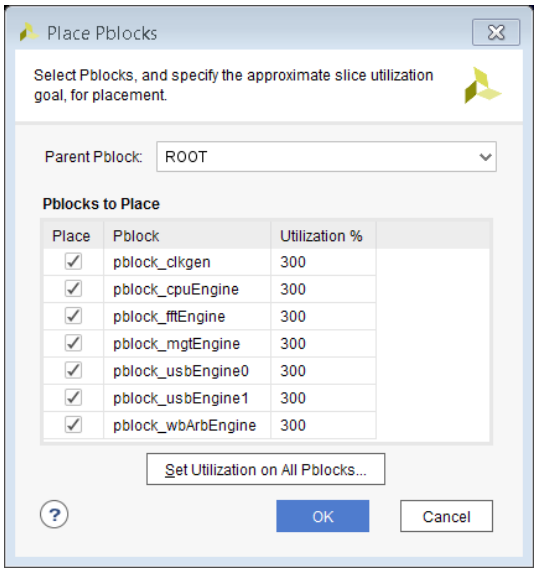
When you integrate RTL into a design, it helps to visualize the design inside the device. Graphically seeing how the blocks interconnect between themselves and the I/O pinout after synthesis helps you to understand your design.

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 getting an overview of the relative size of your design top-level blocks, and how they will occupy the device.

Figure 233: **Place Pblocks Utilization**

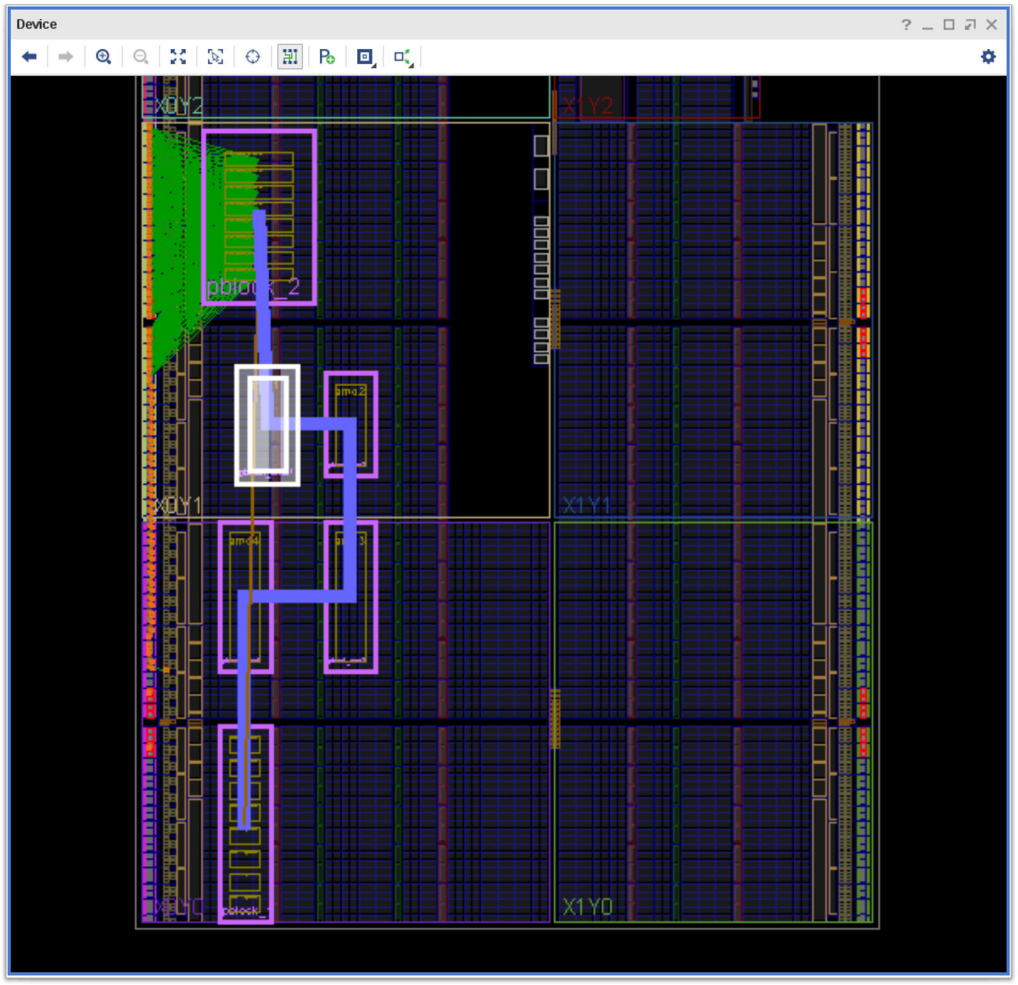


Top-Level Floorplan

The top-level floorplan shows which blocks communicate with I/Os (green lines). Nets connecting two Pblocks are bundled together. The bundles change size and color based on the number of shared nets. Two top-level floorplans are shown in the following figures.

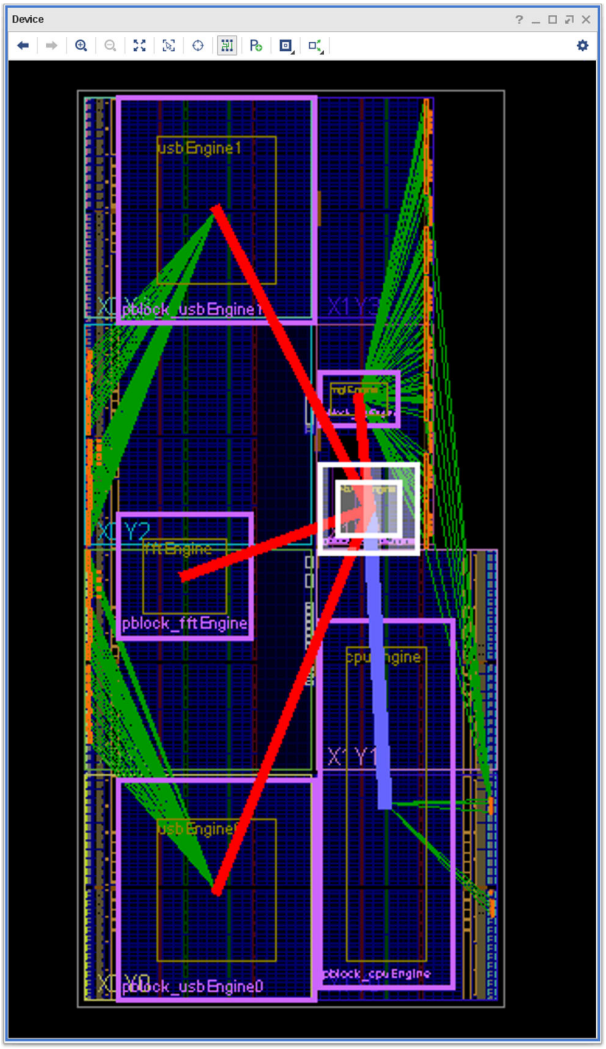
The Data Path Top Level Floorplan shows how the data flows between the top-level blocks of the design. Each block communicates only with two neighbors. The green lines show well-placed I/Os that communicate with a single block.

Figure 234: **Data Path Top Level 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.

Figure 235: **Control Path Floorplan**



Analyzing Utilization Statistics

A common cause of implementation issues is not considering the explicit and implicit physical constraints. The pinout, for example, becomes an explicit physical constraint on logic placement. Slice logic is uniform in most devices. The following specialized resources, however, represent implicit physical constraints because they are only available in certain locations, and impact logic placement:

- I/Os
- Gigabit transceivers
- DSP slices
- Block RAM
- Clock management blocks such as MMCM

- Clock buffers such as BUFG

Blocks that are large consumers of these specialized resources might have to be spread around the device, physically constraining placement and routing when designing the interface with the rest of the design. Additionally, Pblocks are explicit physical constraints used to define allowable placement areas for specified logic. Use a combination of the following methods to analyze block resource usage on the device:

- Report utilization
- Netlist properties
- Pblock properties

Locking Specific Logic to Device Sites

You can place cells on specific locations on the FPGA, such as placing all the I/O ports on an AMD 7 series FPGA design. AMD recommends that you place the I/Os before attempting to close timing.

The I/O placement can impact the cell placement in the FPGA fabric. Hand placing other cells in the fabric can help provide some consistency to clock logic and macro placement, with the goal of more consistent implementation runs.

Table 23: **Constraints Used to Place Logic**

Constraint	Use	Notes
LOC	Places a gate or macro at a specific site.	SLICE sites have subsites called BEL sites.
BEL	Specifies the subsite in the slice to use for a basic element.	

Fixed and Unfixed Cells

Fixed and Unfixed apply to placed cells. They describe the way in which the Vivado tools view placed cells in the design.

For more information about Fixed and Unfixed Cells, refer to this link in the *Vivado Design Suite User Guide: Implementation* (UG904).

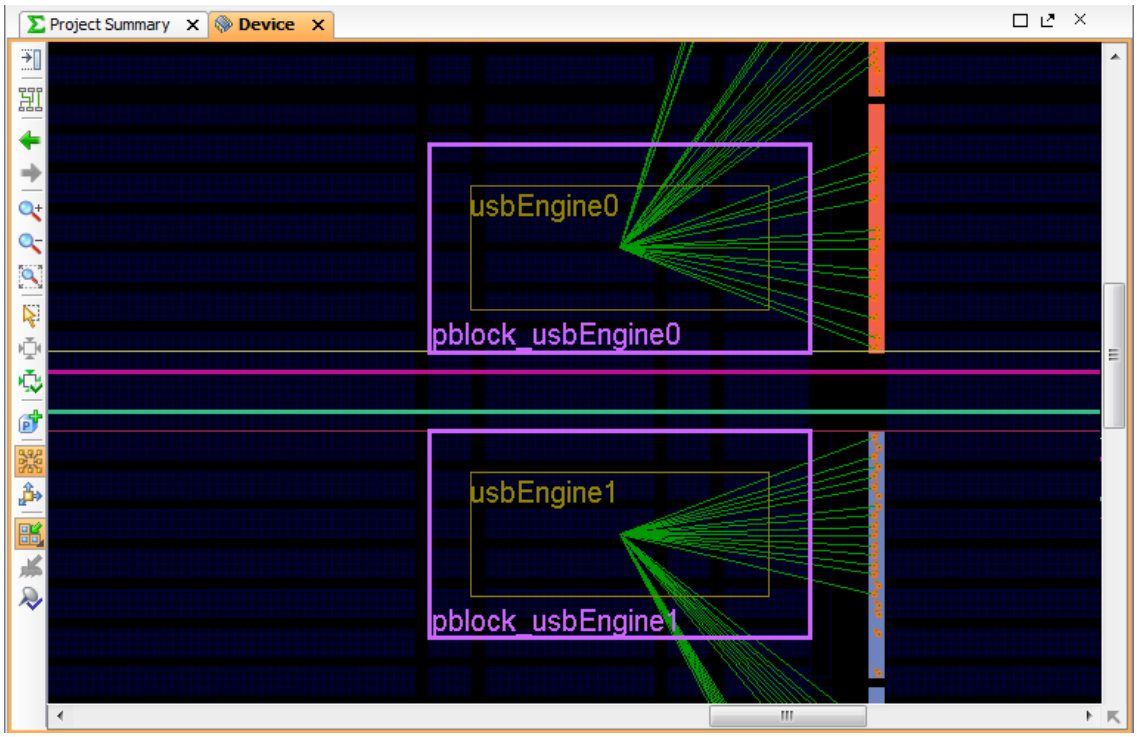
RECOMMENDED: *After the I/Os are placed, use a hierarchical Pblock floorplan as a starting point for user-controlled placement. Hand placing logic should be used when Pblocks have been found not to work.*

Floorplanning With Stacked Silicon Interconnect (SSI) Devices

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

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

Figure 236: **Minimize SLR Crossings**



The I/Os must be placed in the same SLR as the relevant I/O interface circuitry. You must also carefully 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.*

Determining if Hold-Fixing is Negatively Impacting the Design

The Vivado Design Suite router prioritizes fixing hold over setup. This is because your design may work in the lab if you are failing setup by a small amount. There is always the option of lowering the clock frequency. If you have hold violations, the design will most likely not work.

In most cases, the router can meet the hold timing without affecting the setup. In some cases (mostly due to errors in the design or the constraints), the setup time will be significantly affected. Improper hold checks are often caused by improper `set_multicycle_path` constraints in which the `-hold` was not specified. In other cases, large hold requirements are due to excessive clock skew. In this case, AMD recommends that you review the clocking architecture for that particular circuit. For more information, see this link in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

This may occur if your design meets setup timing post placement, but fails setup post route. You can use the `report_design_analysis` command with the `-show_all` option to view path delay due to routing detours added by the router to fix hold violations. The following figure shows an example of report_design_analysis report with the Hold Fix Detour column indicating the delay (in ps) added to the timing path by the router due to hold fixing.

Figure 237: **Report Design Analysis with Hold Fix Detour**

Paths	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Hold Fix Detour	Combir
Path #1	2.034	1.813	0.134(8%)	1.679(92%)	-0.179	-0.118	467	
Path #2	2.034	1.813	0.134(8%)	1.679(92%)	-0.179	-0.118	467	
Path #3	2.034	1.813	0.134(8%)	1.679(92%)	-0.179	-0.118	467	
Path #4	2.034	1.813	0.134(8%)	1.679(92%)	-0.179	-0.118	467	
Path #5	2.034	1.853	0.135(8%)	1.718(92%)	-0.256	-0.117	567	
Path #6	2.034	1.811	0.134(8%)	1.677(92%)	-0.179	-0.117	466	

TIP: *Analyze the estimated hold timing post place and identify any unusually large hold violations (over 500ps).*

If you suspect that hold fixing is affecting timing closure, you can use one of the following to determine if this is the case:

- Method 1: Routing without hold fixing
- Method 2: Run report_timing -min on Worst Failing Setup Path

Method 1: Routing without hold fixing

1. Read the post-placed checkpoint into Vivado Design Suite.

2. Add a constraint to disable all hold checking:

```
set_false_path -hold -to [all_clocks]
```

> **CAUTION!** *This constraint is for test purposes only. Never do this for designs that will be put into production or delivered to another designer. You must remove this constraint before the production design.*

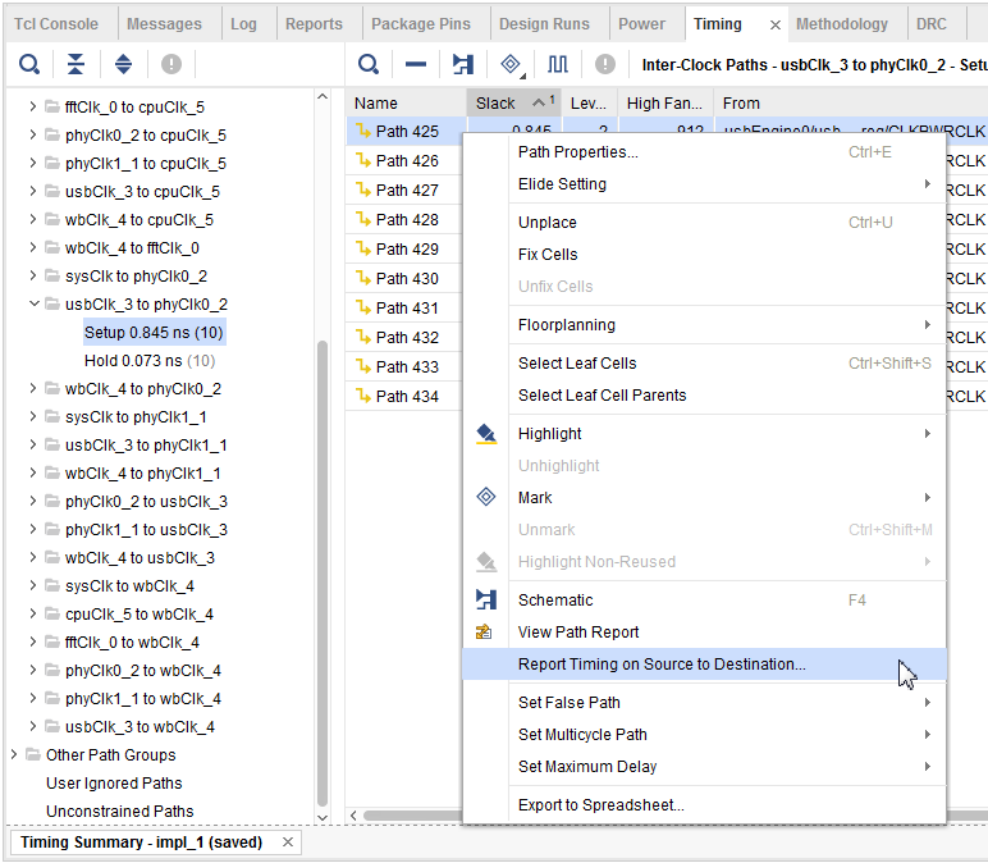
3. Run `route_design` and `report_timing_summary`.

If there is a significant difference between the WNS with and without the hold checks, the hold violations might be too large, and the setup paths are being affected.

Method 2: Run report_timing -min on Worst Failing Setup Path

To determine whether the worst failing setup path is due to hold fixing, review the hold timing of that path. In the Vivado IDE, right click and report timing on source to destination. As opposed to doing the setup timing analysis, it is important to look at the hold timing. Once you have the hold report, verify the requirement and ensure that additional delay was not added on the path to be able to meet hold.

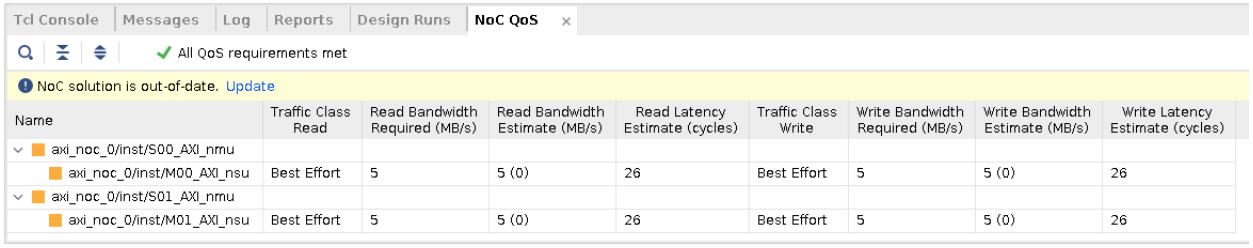
Figure 238: **Running Timing Report on Specific Paths**



Performing NoC Quality of Service Analysis in Versal Devices

The Quality of Service (QoS) Report in AMD Vivado™ compares the estimated QoS from the current Network on Chip (NoC) solution generated by the NoC Compiler to the QoS Requirements specified in the AXI NoC IP and/or the AXI4-Stream NoC IP. If at any time the NoC solution becomes out-of-date, it is required to call the NoC Compiler and generate a new NoC solution to update the QoS Report. The QoS Report will show in its banner that it is out-of-date (see the following figure). Clicking the Update link will call the NoC Compiler to regenerate the NoC solution.

Figure 239: **Vivado NoC QoS Report that is out-of-date**



Name	Traffic Class Read	Read Bandwidth Required (MB/s)	Read Bandwidth Estimate (MB/s)	Read Latency Estimate (cycles)	Traffic Class Write	Write Bandwidth Required (MB/s)	Write Bandwidth Estimate (MB/s)	Write Latency Estimate (cycles)
axi_noc_0/inst/S00_AXI_nmu								
axi_noc_0/inst/M00_AXI_nsu	Best Effort	5	5 (0)	26	Best Effort	5	5 (0)	26
axi_noc_0/inst/S01_AXI_nmu								
axi_noc_0/inst/M01_AXI_nsu	Best Effort	5	5 (0)	26	Best Effort	5	5 (0)	26

***Note*:** In IP integrator, the `validate_bd_design` Tcl Command can be used to update the NoC solution and in the Implementation Tools/Flows the `update_noc_qos` Tcl command can be used.

Actions that can cause the NoC solution to become out-of-date include:

- In IP integrator:
  - Updates to the AXI NoC IP QoS Requirements in IPI
  - Changes to the IPI Block Design
  - Changes to the NoC Master Unit/NoC Slave Unit (NMU/NSU) assignments in the NoC view
- In Implementation Tools/Flow:
  - Unplacing NMU/NSU instances
  - XDC physical constraints impacting NMU/NSU placement
  - Changes to logical net connections to NMU/NSU

For each connection specified in the NoC, the NoC QoS report reports the following information for both Read and Write Transactions:

- Traffic Class: The traffic class specified on the NoC in IP integrator for the connection
- Bandwidth Required: The bandwidth specified in MegaBytes per second on the NoC IP integrator for the connection
- Bandwidth Estimate: The bandwidth estimated in MegaBytes per second by the NoC Compiler for the current NoC solution
- Latency Estimate: The structural latency estimated in NoC Clock Cycles by the NoC Compiler for the current NoC solution

> **IMPORTANT!** *There are two very important items to note for the Latency Estimate in the QoS Report. The Latency Estimate is a structural latency, not a dynamic latency. It reports the minimum round-trip time for a transaction. The Latency Estimate is measured in NoC Clock Cycles in the QoS report whereas the AMD Performance Traffic Generator IP reports Latency in AXI Clock Cycles in simulation. Due to the structural estimates of the QoS Report in Vivado, it is required that your design be validated with actual traffic stimulus for design signoff.*

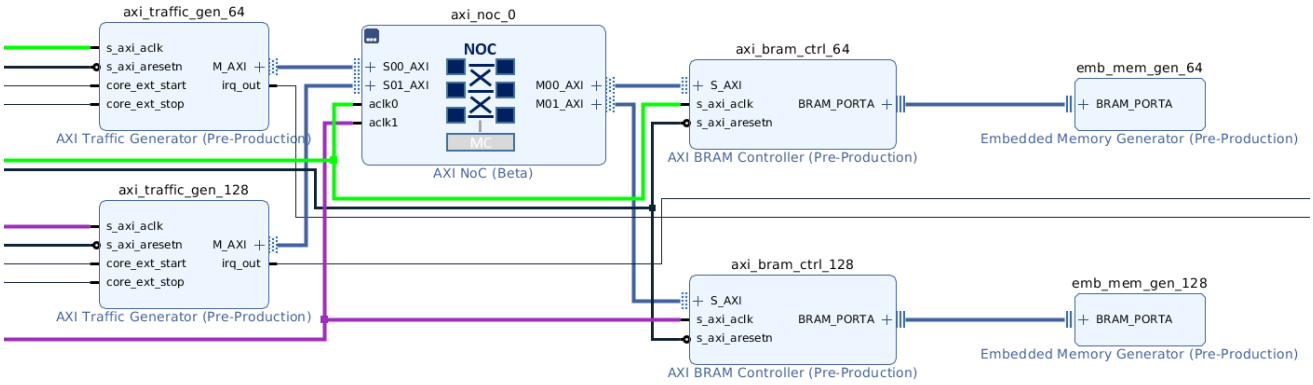
Vivado NoC QoS Reporting Examples

The following example shows a portion of an IP integrator Design with two AXI Traffic Generators and two AXI block RAM Controllers along with associated embedded memory. This example is used to show how you can adjust your datawidth and PL clock frequency to achieve the same bandwidth. The `axi_traffic_gen_64`, `axi_bram_ctrl_64`, and `emb_mem_gen_64` all have 64-bit datawidths and connect to the 200 MHz clock highlighted in green. The `axi_traffic_gen_128`, `axi_bram_ctrl_128`, and `emb_mem_gen_128` all have 128-bit datawidths and connect to the 100 MHz clock highlighted in purple.

The `axi_traffic_gen_64` has a connection through the NoC to the `axi_bram_ctrl_64` while the `axi_traffic_gen_128` has a connection through the NoC to the `axi_bram_ctrl_128`.

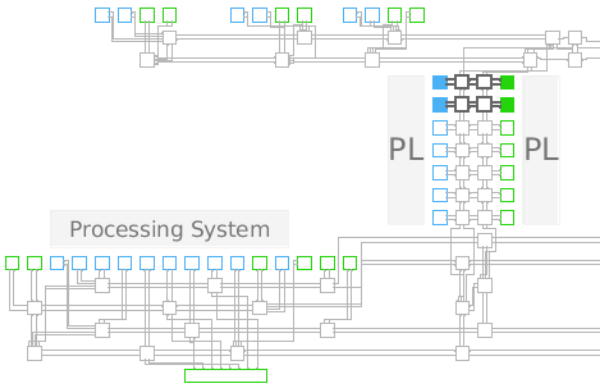
The required bandwidth for both read/write on each connection has been set to 1000 MB/sec on the NoC.

Figure 240: **Example IPI Block Design**



The initial NoC Solution after the design is validated in IP integrator is shown in the following figure and each NoC Connection is routed through the horizontal NoC.

Figure 241: **Initial NoC Solution**



The QoS Report (see the following figure) shows that the bandwidth requirements have been met and each connection is reporting a structural latency of 26 NoC Clock Cycles.

Figure 242: **Initial NoC QoS Report**

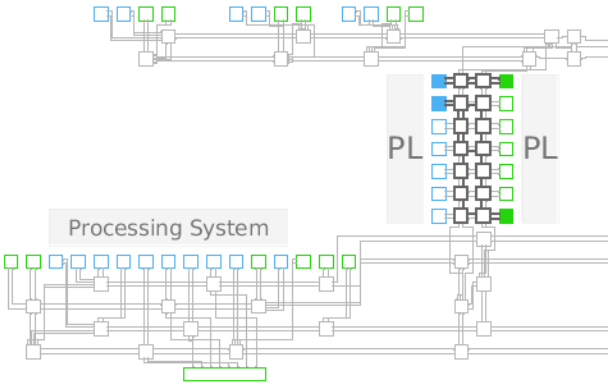
Tcl Console | Messages | Log | Reports | Design Runs | NoC QoS

All QoS requirements met

Name	Traffic Class Read	Read Bandwidth Required (MB/s)	Read Bandwidth Estimate (MB/s)	Read Latency Estimate (cycles)	Traffic Class Write	Write Bandwidth Required (MB/s)	Write Bandwidth Estimate (MB/s)	Write Latency Estimate (cycles)
axi_noc_0/inst/S00_AXI_nmu								
axi_noc_0/inst/M00_AXI_nsu	Best Effort	1000	1000 (0)	26	Best Effort	1000	1000 (0)	26
axi_noc_0/inst/S01_AXI_nmu								
axi_noc_0/inst/M01_AXI_nsu	Best Effort	1000	1000 (0)	26	Best Effort	1000	1000 (0)	26

The NoC View in IP integrator allows for NMU/NSU assignment and the NoC solution can be updated to observe the changes in the QoS Report. The `axi_noc_0/inst/M01_AXI_nsu` has been moved in the NoC view away from the `axi_noc_0/inst/S01_AXI_nmu` to create a longer path through additional NoC Switches. The resulting NoC view is shown in the following figure.

Figure 243: **NoC Solution with manual NSU assignment**



For the resulting NoC QoS, the bandwidth has been maintained but the structural latency for the path has increased from 26 NoC clock cycles to 46 NoC clock cycles as shown in the following figure.

Figure 244: **NoC QoS with manual NSU assignment**

Tcl Console | Messages | Log | Reports | Design Runs | NoC QoS

All QoS requirements met

Name	Traffic Class Read	Read Bandwidth Required (MB/s)	Read Bandwidth Estimate (MB/s)	Read Latency Estimate (cycles)	Traffic Class Write	Write Bandwidth Required (MB/s)	Write Bandwidth Estimate (MB/s)	Write Latency Estimate (cycles)
axi_noc_0/inst/S00_AXI_nmu								
axi_noc_0/inst/M00_AXI_nsu	Best Effort	1000	1000 (0)	26	Best Effort	1000	1000 (0)	26
axi_noc_0/inst/S01_AXI_nmu								
axi_noc_0/inst/M01_AXI_nsu	Best Effort	1000	1000 (0)	46	Best Effort	1000	1000 (0)	46

Appendix A

Timing Methodology Checks

This appendix details each methodology check carried out by the tool, explaining what triggers each check and how to resolve the item.

TIMING-1: Invalid Clock Waveform on Clock Modifying Block

Invalid clock waveform for clock <clock_name> specified at a <cell_type> output <pin_name> that does not match the Clock Modifying Block (CMB) settings. The waveform of the clock is <VALUE>. The expected waveform is <VALUE>.

Description

The Vivado Design Suite automatically derives clocks on the output of a CMB based on the CMB settings and the characteristics of the incoming master clock. If the user defines a generated clock on the output of the CMB, Vivado does not auto-derive a generated clock on the same definition point (net or pin). The DRC warning is reporting that the user-defined generated clock does not match the expected auto-derived clock that Vivado would automatically create. This could lead to hardware failures because the timing constraints for the design do not match what happens on the device.

Resolution

If the user-defined generated clock is unnecessary, remove the constraint and use the auto-derived clock instead. If constraint is necessary, verify that the generated clock constraint matches the auto-derived clock waveform or modify the CMB properties to match the expected clock waveform. If the intention is to force the name of the auto-derived clock, the recommendation is to use the `create_generated_clock` constraint with only the `-name` option defined and the name of the object where the clock is defined (typically output pin of CMB). See the *Vivado Design Suite User Guide: Using Constraints* (UG903) for additional information about creating generated clocks and restrictions of the auto-derived clocks renaming constraint.

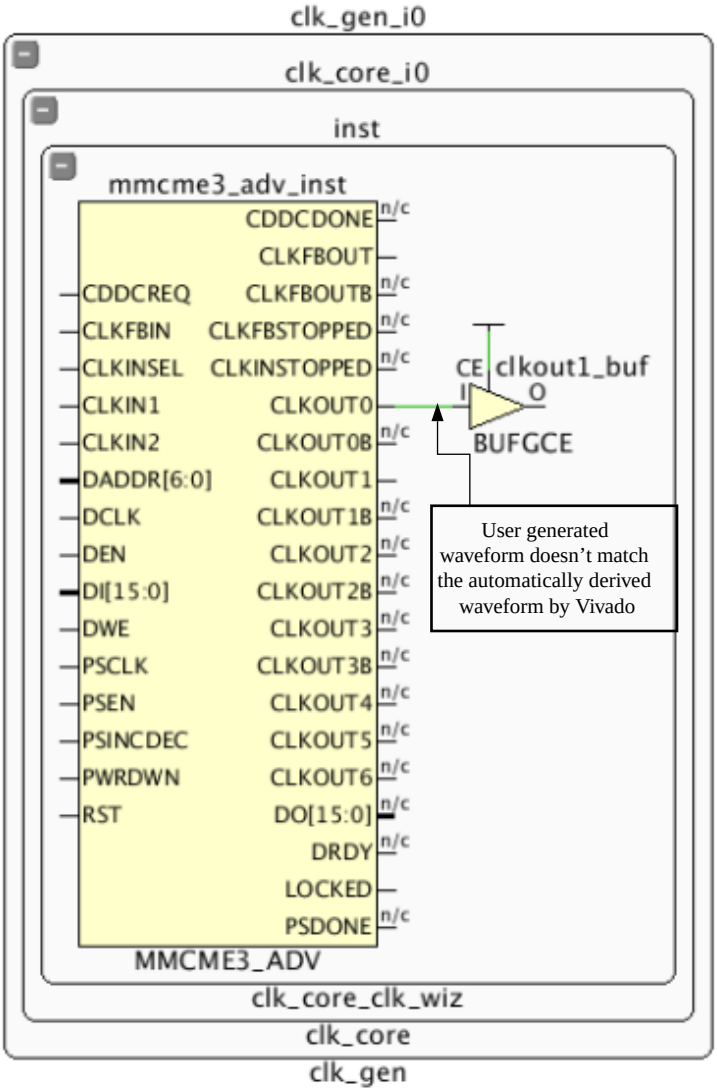
Example

In the following figure, a `create_generated_clock` constraint was defined on the MMCM instance pin `CLKOUT0`, but doesn't match the auto-derived waveform generated by Vivado from the MMCM attribute settings.

To just rename the auto-derived clock, use the following constraint right after the master clock definition in your constraint files:

```
create_generated_clock -name clkName [get_pins clk_gen_i0/clk_core_i0/inst/
mmcme3_adv_inst/CLKOUT0]
```

Figure 245: **Invalid Clock Waveform on Clock Modifying Block**



TIMING-2: Invalid Primary Clock Source Pin

A primary clock <clock_name> is created on an inappropriate pin <pin_name>. It is recommended to create a primary clock only on a proper clock root (input port or primitive output pin with no timing arc).

Description

A primary clock must be defined on the source of the clock tree. For example, this would be the input port of the design. When a primary clock is defined in the middle of a logic path, timing analysis can become inaccurate because it ignores the insertion delay prior to the primary clock source point, which prevents proper skew computation. Therefore, a primary clock created on an internal driver pin should be discouraged. The consequence could be a failure in hardware.

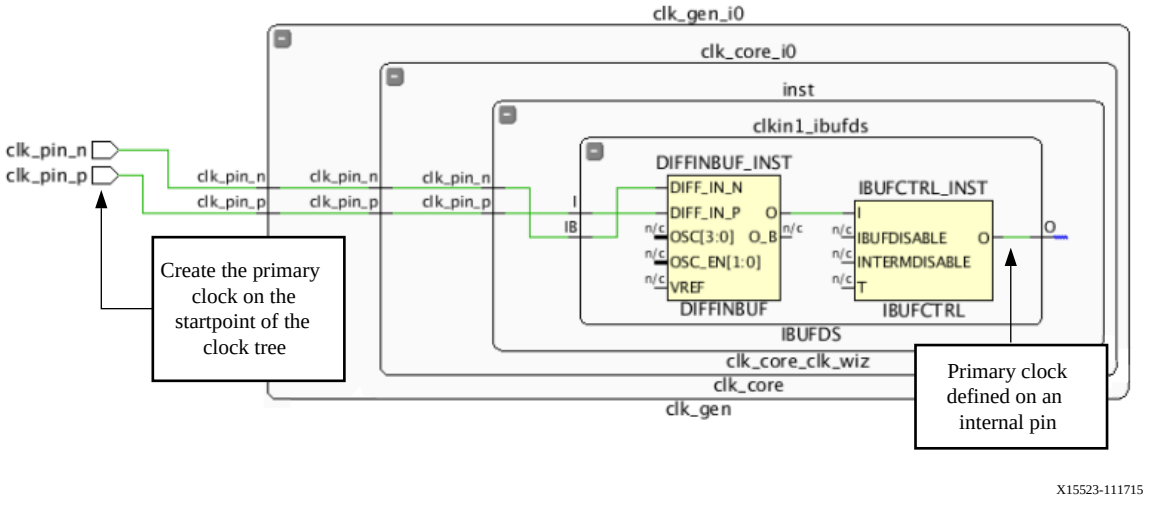
Resolution

Modify the `create_clock` constraint to use the actual clock tree source.

Example

In the following figure, the primary clock definition, the `create_clock` constraint, was placed on the output pin of the IBUFCTRL instance. If the clock `clk_pin_p` is used to time an input or output port path, the slack will be inaccurate because part of the clock tree insertion delay will be missing. The primary clock definition for the differential input buffer should be placed on the top-level port `clk_pin_p`.

Figure 246: **Invalid Primary Clock on Internal Pin**



TIMING-3: Invalid Primary Clock on Clock Modifying Block

A primary clock `<clock_name>` is created on the output pin or net `<pin/net_name>` of a Clock Modifying Block.

Description

Vivado automatically derives clocks on the output of a CMB based on the CMB settings and the characteristics of the incoming master clock. If the user defines a primary clock on the output of the CMB, Vivado does not auto-derive a clock on the same output. This DRC is reporting that a primary clock was created on the output of the CMB, which breaks the relationship with the incoming clock and prevents proper clock insertion delay computation. This is not recommended because it can lead to inaccurate timing analysis and incorrect hardware behavior.

Resolution

Modify the constraints to remove the `create_clock` constraint on the output of the CMB. If the intention is to force the name of the auto-generated clock, AMD recommends using the `create_generated_clock` constraint with only the `-name` option and the CMB output pin. See the *Vivado Design Suite User Guide: Using Constraints* (UG903) for additional information about creating generated clocks.

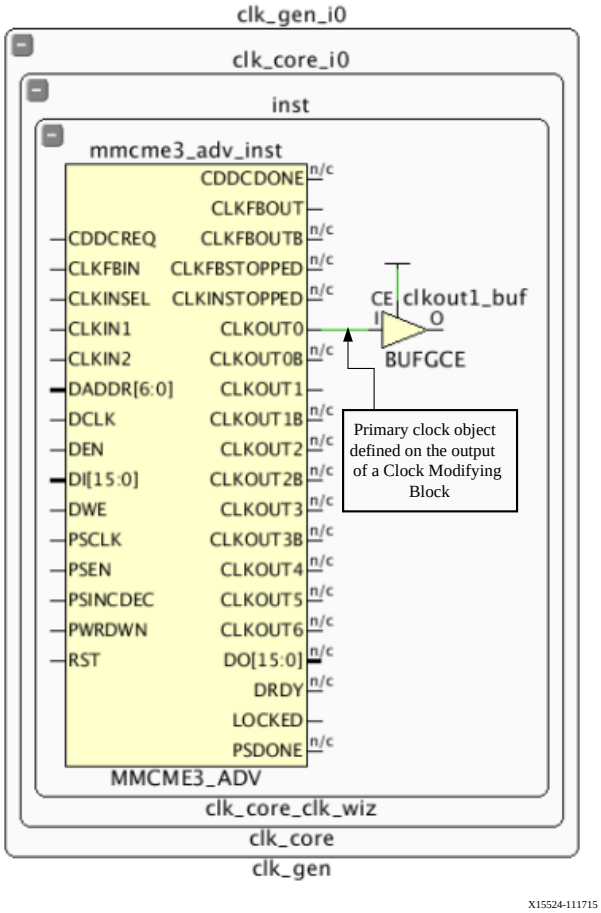
Example

In the following figure, a `create_clock` constraint was defined on the MMCM instance pin `CLKOUT0`. This overrides the automatically derived clock created by Vivado and loses any relationship with the incoming clock.

To just rename the auto-derived clock, use the following constraint right after the master clock definition in your constraint files:

```
create_generated_clock -name clkName [get_pins clk_gen_i0/clk_core_i0/inst/
mmcme3_adv_inst/CLKOUT0]
```

Figure 247: **Invalid Primary Clock on Clock Modifying Block**



TIMING-4: Invalid Primary Clock Redefinition on a Clock Tree

Invalid clock redefinition on a clock tree. The primary clock <clock_name> is defined downstream of clock <clock_name> and overrides its insertion delay and/or waveform definition.

Description

A primary clock must be defined on the source of the clock tree. For example, this would be the input port of the design. When a primary clock is defined downstream that overrides the incoming clock definition, timing analysis can become inaccurate because it ignores the insertion delay prior to the redefined primary clock source point, which prevents proper skew computation. It is not recommended as the consequence could be incorrect timing analysis which might lead to a failure in hardware.

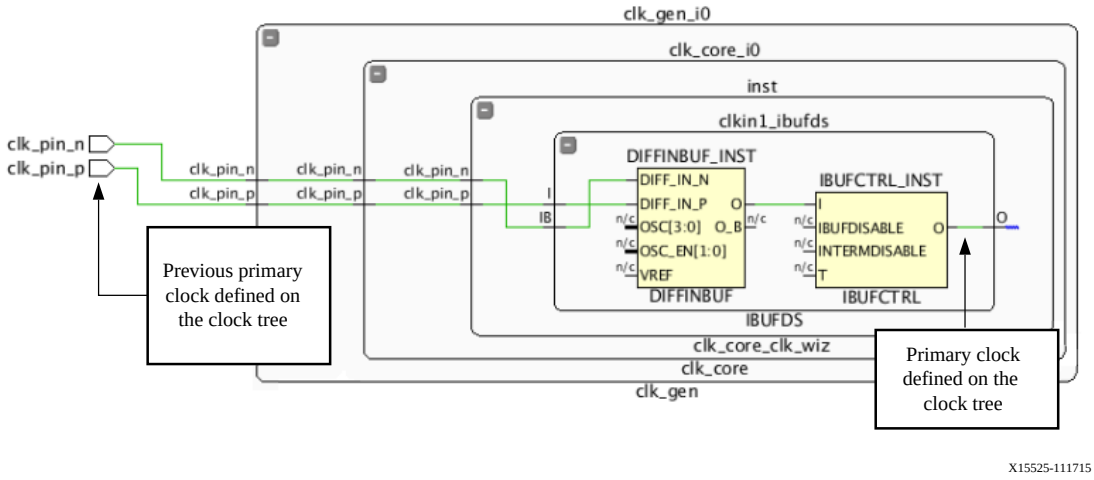
Resolution

Remove the `create_clock` constraint on the downstream object and allow the propagation of the upstream clock or create a generated clock referencing the upstream primary clock.

Example

In the following figure, the primary clock was correctly defined on the top-level port `clk_pin_p`. However, a `create_clock` constraint was used to redefine the primary clock on the output of the IBUFCTRL output. This new clock will ignore all delays prior to the IBUFCTRL.

Figure 248: **Invalid Primary Clock Redefinition on a Clock Tree**



TIMING-5: Invalid Waveform Redefinition on a Clock Tree

Invalid inverted waveform on a clock tree. The generated clock `<clock_name>` is defined downstream of clock `<clock_name>` and has an inverted waveform definition compare to the incoming clock.

Description

A generated clock should be defined in relation to the incoming clock. The DRC warning is reporting that the generated clock has an invalid definition, such as a different period, phase shift, or inversion compared to the incoming clock.

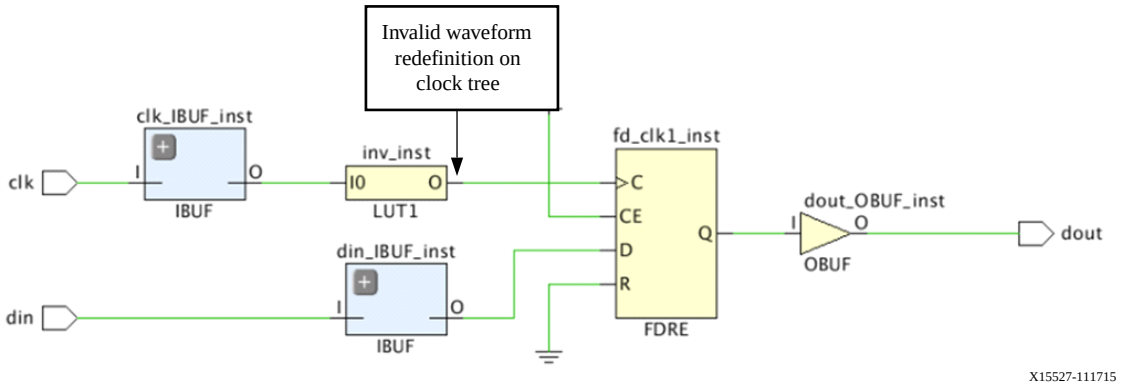
Resolution

Modify the `create_generated_clock` constraint to define a proper waveform definition that matches the incoming clock definition. For more details about creating a proper generated clock constraint, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Example

In the following figure, a `create_generated_clock` was created on the output of the LUT1 inverter, but the `-invert` switch was not applied.

Figure 249: **Invalid Waveform Redefinition on a Clock Tree**



TIMING-6: No Common Primary Clock Between Related Clocks

The clocks <clock_name> and <clock_name> are related (timed together) but they have no common primary clock. The design could fail in hardware even if timing is met. To find a timing path between these clocks, run the following command:

```
report_timing -from [get_clocks <CLOCK_NAME1>] -to [get_clocks
<CLOCK_NAME2>]
```

Description

The two clocks reported are considered related and timed as synchronous by default even if they are not derived from a common primary clock and do not have a known phase relationship. The DRC warning is reporting that the timing engine cannot guarantee that these clocks are synchronous.

Resolution

The resolution depends on whether the two clock domains are asynchronous or synchronous. In the case of the clocks being asynchronous, the paths between the two domains should be covered by a timing exception (such as set_max_delay -datapath_only, set_clock_groups, or set_false_path). The DRC will be resolved once all the paths between these two domains have full exception coverage.

Example

In the case of the clocks being synchronous, you can define one timing clock on both clock source objects if originally both clocks have the same waveform (see the first example below).

Example 1: `create_clock -period 10 -name clk1 [get_ports <clock-1-source> <clock-2-source>]`

If the two clocks have different waveforms, you can define the first clock as a primary clock and the second clock as a generated clock, with the first clock specified as the master clock (see Example 2 below).

Example 2: `create_clock -period 10 -name clk1 [get_ports <clock-1-source>]`

If the clocks are related, but have a clock period ratio of 2, the solution is to create a primary clock on the one source, and create a generated clock on the second source:

```
create_generated_clock -source [get_ports <clock-1-source>] -name clk2
-divide_by 2 [get_ports <clock-2-source>]
```

TIMING-7: No Common Node Between Related Clocks

The clocks <clock_name> and <clock_name> are related (timed together) but they have no common node. The design could fail in hardware. To find a timing path between these clocks, run the following command:

```
report_timing -from [get_clocks <CLOCK_NAME1>] -to [get_clocks
<CLOCK_NAME2>]
```

Description

The two clocks reported are considered related and timed as synchronous by default. The DRC warning is reporting that the timing engine cannot guarantee that these clocks are synchronous in hardware, since it could not determine a common node between the two clock trees.

Resolution

The resolution depends on whether the two clock domains are asynchronous or synchronous. In the case of the clocks being asynchronous, the paths between the two domains should be covered by a timing exception (such as set_max_delay -datapath_only, set_clock_groups, or set_false_path).

In the case of the clocks being synchronous, this DRC warning can be waived.

When the violation is reported during the OOC (Out-Of-Context) synthesis of a module and if the two clocks are known to have a common node at the top-level, the TIMING-7 violation can be prevented by the steps outlined below:

1. Define one of the clocks as a primary clock on the first input clock port.
2. Define the second clock as a generated clock on the second input clock port. This clock should reference the primary clock defined in step 1.
3. Define the property HD.CLK_SRC on the two input clock ports.

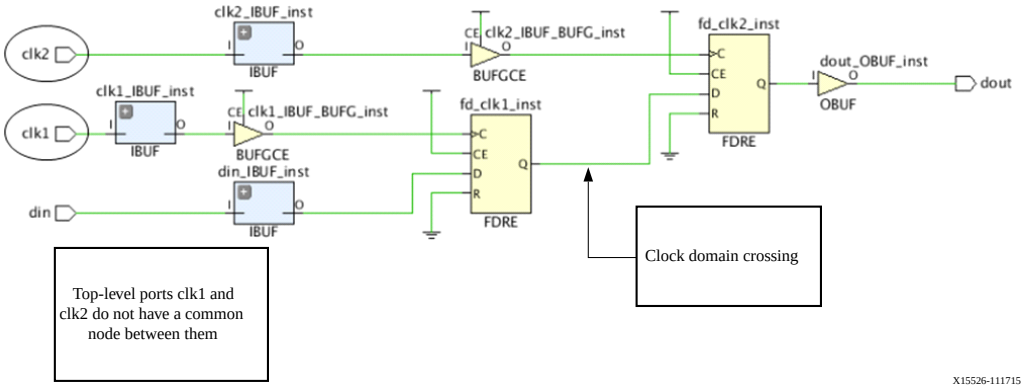
Example

In the following figure, a synchronous clock domain crossing (CDC) exists between the `clk1` and `clk2` domains. Both `clk1` and `clk2` are determined to be synchronous in Vivado by default. However, because `clk1` and `clk2` are input ports, there is no common node relationship between the two clocks. For this case, Vivado Design Suite cannot guarantee that the two clocks are synchronous.

However, if the module is synthesized Out-Of-Context and `clk1` and `clk2` have a common node at the top-level, the TIMING-7 violation can be suppressed during the OOC synthesis by defining, for example, the following constraints:

```
   create_clock -period 3.000 [get_ports clk1]
   set_property HD.CLK_SRC BUFGCTRL_X0Y2 [get_ports clk1]
   create_generated_clock -divide_by 2 -source [get_ports clk1] \
[get_ports clk2]
   set_property HD.CLK_SRC BUFGCTRL_X0Y4 [get_ports clk2]
```

Figure 250: **No Common Node Between Related Clocks**



TIMING-8: No Common Period Between Related Clocks

The clocks `<clock_name>` and `<clock_name>` are found related (timed together) but have no common (expandable) period.

Description

The two clocks reported are considered related and timed as synchronous by default. However, the timing engine was unable to determine a common period after expanding the waveform of both clocks over 1000 cycles. In such a case, the worst setup relationship over these 1000 cycles is used for timing analysis. However, the timing engine cannot ensure this is the most pessimistic case. This typically occurs with clocks with an odd fractional period ratio.

Resolution

As the waveforms do not allow safe timing analysis between the two clocks, it is recommended to treat these clocks as asynchronous. Therefore, the paths between the two clock domains should be covered by a timing exception (such as set_max_delay -datapath_only, set_false_path, or set_clock_groups).

TIMING-9: Unknown CDC Logic

One or more asynchronous Clock Domain Crossing has been detected between two clock domains through a `set_false_path`, `set_clock_groups`, or a `set_max_delay -datapath_only` constraint. However, no double-registers logic synchronizer has been found on the side of the capture clock. It is recommended to run `report_cdc` for a complete and detailed CDC coverage. Also, consider using `XPM_CDC` to avoid critical severities.

Description

The purpose of the DRC is to ensure that inter-clock domains constrained with timing exceptions have been designed with safe asynchronous clock domain crossing circuitry. For more details on recognized safe topologies, see Report Clock Domain Crossings.

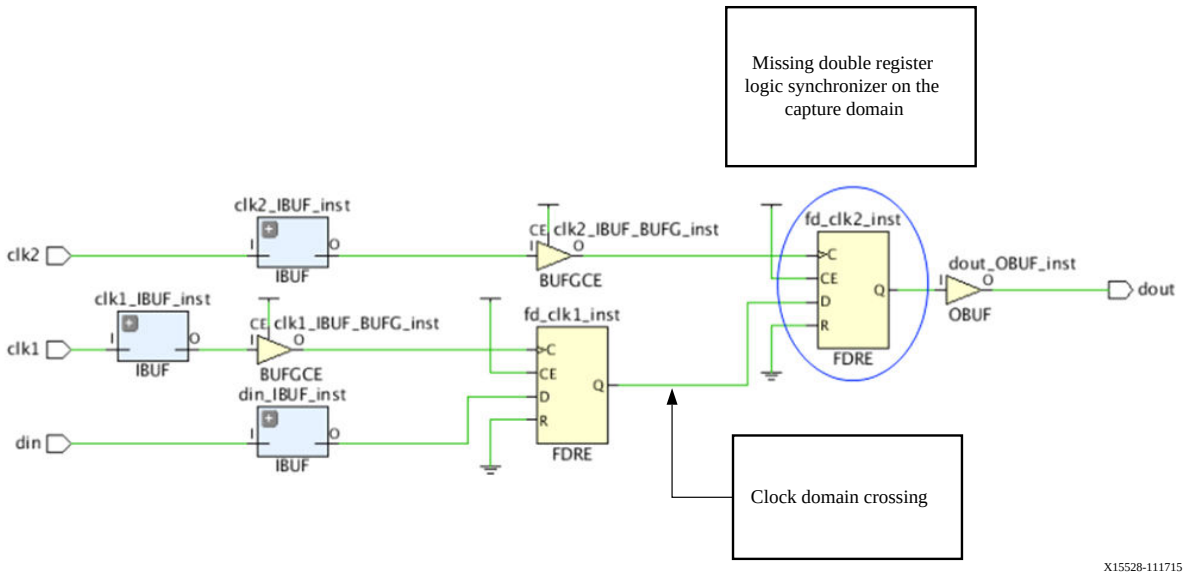
Resolution

The recommendation is to make the appropriate design to have a proper synchronization for the inter-clock paths. To do this, add, at minimum, a double-register logic synchronizer. In the case a FIFO or higher-level protocol is already defined on the path, this DRC can be safely ignored. For a detailed list of CDC violations, run `report_cdc`.

Example

In the following figure, an asynchronous clock domain exists between `clk1` and `clk2`. However, the `clk2` capture domain does not contain a double register logic synchronizer to synchronize the data.

Figure 251: **Missing Synchronizer**



TIMING-10: Missing Property on Synchronizer

One or more logic synchronizer has been detected between two clock domains but the synchronizer does not have the property ASYNC_REG defined on one or both registers. It is recommended to run `report_cdc` for complete and detailed CDC coverage.

Description

Synchronizer registers must have their ASYNC_REG property set to `TRUE` to preserve the cells through any logic optimization during synthesis and implementation, and to optimize their placement for the best mean time between failure (MTBF) statistics.

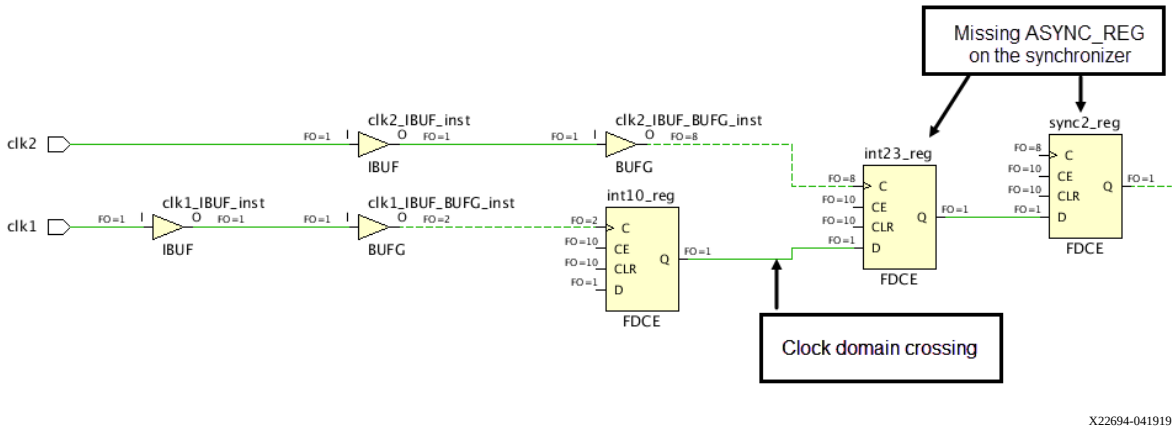
Resolution

The solution is to add the ASYNC_REG constraint to each stage of the logic synchronizer. For a detailed list of CDC violations, run `report_cdc`. To find out more information on the `ASYNC_REG` constraint, refer to the *Vivado Design Suite Properties Reference Guide* (UG912). The TIMING-10 violation is triggered when at least one of the first two synchronizer registers is missing the ASYNC_REG property.

Example

In the following figure, an asynchronous clock domain exists between `clk1` and `clk2` and is properly synchronized with a double register logic synchronizer. However, each register of the synchronizer needs to have the ASYNC_REG property applied to increase the timing slack and lower MTBF.

Figure 252: **Missing Property on Synchronizer**



TIMING-11: Inappropriate Max Delay with Datapath Only Option

A max delay constraint with `-datapath_only` has been applied between `<pin_name>` and `<pin_name>`. The startpoint(s) and endpoint(s) either belong to the same clock domain or belong to two clock domains that can safely be timed together. It is only recommended to use the `-datapath_only` option on paths between clocks that do not have a known phase relationship. This DRC is waived when a synchronizer is found on the path endpoint.

Description

The `set_max_delay` with the `-datapath_only` option is used to remove the clock skew from the setup slack computation and to ignore hold timing. The `set_max_delay -datapath_only` command is used to constrain asynchronous signals timing paths that: (1) do not have a clock relationship; but which (2) require maximum delay. It is not recommended to use this constraint on synchronous paths.

Resolution

The solution is to modify the `set_max_delay -datapath_only` constraint so that it does not cover synchronous timing paths. Refer to the startpoint and endpoint cells listed in the message to find the associated `set_max_delay` constraint.

TIMING-12: Clock Reconvergence Pessimism Removal Disabled

Description

The Clock Reconvergence Pessimism Removal (CRPR) mode has been disabled. It is not recommended to perform timing analysis in this mode as over-pessimistic clock tree delays could result in impossible timing closure.

The CRPR feature is used to remove artificially induced pessimism that is derived from the usage of the maximum and minimum delay along the common portion of the clock network. If the CRPR is disabled, it might be difficult to close timing.

Resolution

The recommendation is to enable the CRPR analysis to ensure the design has accurate timing information. The Tcl command to enable the CRPR analysis is `config_timing_pessimism -enable`.

TIMING-13: Timing Paths Ignored Due to Path Segmentation

Some timing paths are not reported due to path segmentation on pin(s) <pin_name>. To prevent path segmentation, all the min and max delay constraints should be defined with a list of valid startpoints and endpoints.

Description

Path segmentation occurs when a timing path is broken into a smaller path to be timed. When max and min delay constraints are defined on pins that are invalid startpoints (and respectively, endpoints), the timing engine breaks the timing arcs going through the node so that the node becomes a valid startpoint (and respectively, endpoint). It is highly recommended to avoid path segmentation as it might have unexpected consequences. This might result in incorrect timing analysis and hardware failures.

Resolution

Avoid path segmentation whenever possible by carefully choosing valid startpoints and endpoints in the `set_max_delay` and `set_min_delay` constraints. For additional information on path segmentation and using the min/max delay constraints, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-14: LUT on the Clock Tree

The LUT <cell_name> has been found on the clock tree. It is not recommended to have LUT cells on the clock path.

Description

A LUT on the clock path might cause excess skew because the clock must be routed on general routing resources through the fabric. In addition to excess skew, these paths are more susceptible to PVT variations. It is highly recommended to avoid local clocks whenever possible.

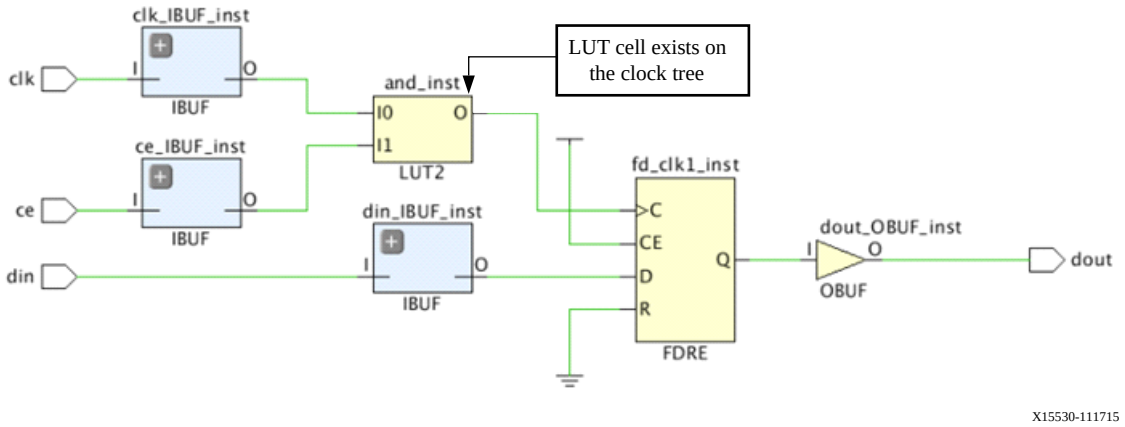
Resolution

The solution is to change the design to remove the LUT located on the clock tree. Synthesis can create this situation in many cases such as clock gating and inversion. In the case of an inversion LUT1 cell, the LUT might be absorbed into the downstream SLICE after `opt_design`. Investigate the case to ensure that the situation is still valid after `opt_design` is complete.

Example

In the following figure, a LUT is used to gate the clock with a clock enable signal. The LUT on the path can cause excess skew, which is undesirable.

Figure 253: **LUT on the Clock Tree**



TIMING-15: Large Hold Violation on Inter-Clock Path

There is a large inter-clock skew of `<value` ns between `<cell_name>` (clocked by `<clock_name>`) and `<cell_name>` (clocked by `<clock_name>`) that results in large hold timing violation(s) of `<VALUE>` ns. Fixing large hold violations during routing might impact setup slack and result in more difficult timing closure.

Description

The DRC warning is reporting that the large hold violation due to the inter-clock skew will most likely be difficult to close timing during implementation. It is recommended to investigate the large inter-clock skew greater than 1.0 ns to ensure proper constraints or design topology.

Resolution

Investigate whether the large inter-clock skew on the timing path should be timed or is related to non-optimal timing constraints. If the large skew occurs due to an unconstrained CDC path, add the necessary timing exception. If the violation occurs due to a logic associated with the clock tree, investigate the topology of path for improvements to more easily close timing.

TIMING-16: Large Setup Violation

There is a large setup violation of `<value>` ns between `<cell_name>` (clocked by `<clock_name>`) and `<cell_name>` (clocked by `<clock_name>`). Large setup violations at the end of those stages might be difficult to fix during the post-placement implementation flow and could be the result of non-optimal XDC constraints or non-optimal design architecture.

Description

This DRC warning reports setup violations that will most likely be difficult to close timing during implementation. It is recommended to investigate setup violations greater than 1.0 ns to ensure proper constraints or design topologies.

Resolution

Investigate whether the large setup violation is a timing path that should be timed or is related to non-optimal timing constraints. If the setup violation occurs due to an unconstrained CDC path, add the necessary timing exception. If the violation occurs due to a significant amount of combinational logic, investigate the topology of the path for improvements to more easily close timing.

TIMING-17: Non-Clocked Sequential Cell

The clock pin `<pin_name>` is not reached by a timing clock.

Description

The DRC reports the list of sequential cells unconstrained by a timing clock which affect the resulting timing analysis for the reported cells. It is highly recommended that all clocks be properly defined in order to get the maximum timing path coverage with the best accuracy. The consequence could be missing timing analysis, which might lead to hardware failures.

Resolution

The resolution is to create the missing primary or generated clock on the clock tree driving the unconstrained sequential cells.

TIMING-18: Missing Input or Output Delay

An <input/output> delay is missing on <port_name> relative to clock(s) <clock_name>.

Description

IO timing is in reference to a timing path that includes an external device. The input and output delays specify the paths delay of the ports relative to a clock edge at the interface of the design. It is highly recommended to add input/output delay constraints to ensure that the FPGA interface can meet the timing of the external devices.

Note: A TIMING-18 violation is reported for any missing clock edge. If a clock is inverted during the clock network propagation (including on the primitive clock pin) and an input or output delay is expected but missing for the falling clock edge, a TIMING-18 violation is generated although the missing clock edge is not specified inside the message.

Resolution

Add the required input and output delay constraints in correspondence with required board application.

TIMING-19: Inverted Generated Clock Waveform on ODDR

The waveform of the generated clock <clock_name> is inverted compared to the waveform of the incoming clock <clock_name>.

Description

A generated clock on a forwarded clock port should be defined in relation to the incoming clock. The DRC warning is reporting that the generated clock on the forwarding clock port has an invalid waveform, such as an inversion, compared to the incoming source clock. This might lead to hardware failures as the timing analysis of the ports associated with the forwarded clock do not match what happens on the device.

Resolution

Modify the `create_generated_clock` constraint to define a proper waveform that matches the incoming clock definition. For more details about creating a proper generated clock constraint, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-20: Non-Clocked Latch

The latch `<cell_name>` cannot be properly analyzed because its control pin `<pin_name>` is not reached by a timing clock.

Description

This DRC is reporting the list of latch cells not constrained by a timing clock which affect the resulting timing analysis. It is highly recommended that all clocks be properly defined in order to get the maximum timing path coverage with the best accuracy. The consequence could be incomplete timing analysis coverage, which might lead to hardware failures.

Resolution

The resolution is to create the primary or generated clock on the source of the clock tree driving the unconstrained control pins on the latch cells.

TIMING-21: Invalid COMPENSATION Property on MMCM

The MMCM `<cell_name>` has an invalid COMPENSATION property value relative to the connection of its feedback loop. If the feedback loop goes outside the FPGA, the property should be set to EXTERNAL. If the feedback loop is internal to the FPGA, the property should be set to ZHOLD.

Description

MMCM compensation modes define how the MMCM feedback is configured for delay compensation of the output clocks. Depending on the MMCM use case, the feedback path should match a specific topology. This DRC warning is reporting that the topology of the MMCM use case doesn't match the COMPENSATION property value. This might lead to unintended behavior in hardware because the timing analysis does not match.

Resolution

The recommendation is to leave the default value of AUTO to the COMPENSATION property of the MMCM in the design. The Vivado IDE will automatically select the appropriate compensation value based on the circuit topology. For additional information on the compensation property and the input delay compensation, refer to the Clocking Resources User Guide for your specific architecture.

TIMING-22: Missing External Delay on MMCM

The MMCM `<cell_name>` has an external feedback loop but no external delay has been specified between `FBOUT` and `FBIN`. It is recommended to specify an external delay with set_external_delay between the two ports connected to the pins `FBOUT` and `FBIN` with an external feedback loop.

Description

The MMCM can be configured for external deskew where the feedback board trace matches the trace to the external components. The external delay value is used in the calculation of the MMCM compensation delay. This could lead to hardware failures, especially on the IO paths, because the timing analysis of the MMCM compensation does not match what happens on the device.

Resolution

Add a `set_external_delay` constraint between the external feedback input and output port for the defined external trace delay. For additional information on the `set_external_delay` command, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Example

```
set_external_delay -from <output_port> -to <input_port>
<external_delay_value>
```

TIMING-23: Combinational Loop Found

A timing loop has been detected on a combinational path. A timing arc has been disabled between `<cell_name1>` and `<cell_name2>` to break the timing loop.

Description

Combinational timing loops are created when the output of combinatorial logic is fed back to its input, resulting in a timing loop. This loop unnecessarily increases the number of cycles by infinitely going around the same path and cannot be timed. To resolve the timing loop, the Vivado IDE disables the timing arc on the cell in the loop.

Resolution

If you did not intend to create a combinational feedback loop, correct the issue by modifying the design source files (RTL). But because the timing loop is expected, use the `set_disable_timing` command to break the timing loop where it makes the most sense (usually the feedback path) instead of letting Vivado timing break it at a random location.

TIMING-24: Overridden Max Delay Datapath Only

A `set_clock_groups` or a `set_false_path` between clocks `<clock_name>` and `<clock_name>` overrides a `set_max_delay -datapath_only` (see constraint position <#> in the Timing Constraints window in the Vivado IDE). It is not recommended to override a `set_max_delay -datapath_only` constraint. Replace the `set_clock_groups` or `set_false_path` between clocks with point-to-point `set_false_path` constraints.

Description

The DRC warning only occurs when a `set_max_delay -datapath_only` constraint is overridden by a `set_clock_groups` or `set_false_path` constraint between clocks. If a point-to-point `set_false_path` constraint overrides a `set_max_delay -datapath_only` constraint, the DRC will not be reported.

Resolution

The solution is to replace the `set_clock_groups` or `set_false_path` between clocks with point-to-point false path constraints to avoid incorrectly overriding a `set_max_delay -datapath_only` constraint.

TIMING-25: Invalid Clock Waveform on Gigabit Transceiver (GT)

The waveform of the clock `<clock_name>` defined on the transceiver output pin `<pin_name>` or on the net connected to that pin is not consistent with the transceiver settings or the reference clock definition is missing. The auto-derived clock period is <PERIOD> and the user-defined clock period is <PERIOD>.

Description

For UltraScale devices, Vivado automatically derives clocks on the output of a GT based on the GT settings and the characteristics of the incoming master clock. For 7 series devices, Vivado does not automatically derive the GT clocks; it is your responsibility to create the appropriate primary clocks on the GT's output pins. The DRC warning is reporting that the user-defined clock does not match the expected auto-derived clock that Vivado would automatically create. This could lead to hardware failures as the timing constraints for the design do not match what happens on the device.

Resolution

If the user-defined generated clock is unnecessary, remove the constraint and use the auto-derived clock instead. If constraint is necessary, verify that the generated clock constraint matches the auto-derived clock waveform or modify the GT properties to match the expected clock waveform. If the intention is to force the name of the auto-derived clock, the recommendation is to use the `create_generated_clock` constraint with only the `-name` option defined and the name of the object where the clock is defined (typically output pin of GT). See the *Vivado Design Suite User Guide: Using Constraints* (UG903) for additional information about creating generated clocks and restrictions of the auto-derived clocks renaming constraint.

TIMING-26: Missing Clock on Gigabit Transceiver (GT)

The output clock pin `<pin_name>` does not have clock defined. Create a primary clock on the `<port_name>` input port to let Vivado auto-derive the missing GT clocks.

Description

For UltraScale devices, Vivado automatically derives clocks on the output of a GT based on the GT settings and the characteristics of the incoming master clock. The DRC warning is reporting that Vivado is unable to auto-derive the output clock of the GT due to the missing primary clock on the input port. The consequence is that the downstream logic connected to the GT related clocks will not be timed.

Resolution

Create a primary clock on the recommended input port to the GT.

TIMING-27: Invalid Primary Clock on Hierarchical Pin

A primary clock `<clock_name>` is created on an inappropriate internal pin `<pin_name>`. It is not recommended to create a primary clock on a hierarchical pin when its driver pin has a fanout connected to multiple clock pins.

Description

If the driver is traversed by a clock and a new clock is defined downstream on a hierarchical pin, the cells downstream of the hierarchal pin will have different timing analysis compared to the cells on the fanout of the driver pin. If any synchronous paths exist between the driver clock and the hierarchical pin clock, skew will be inaccurate and timing signoff will be invalid. This situation can result in hardware failure.

Resolution

Remove the primary clock definition on the hierarchical pin, or if the downstream clock is absolutely needed, use a generated clock constraint with the driver clock specified as master clock instead.

TIMING-28: Auto-Derived Clock Referenced by a Timing Constraint

The auto-derived clock `<clock_name>` is referenced by name inside timing constraint (see constraint position <#> in the Timing Constraint window in the Vivado IDE). It is recommended to reference an auto-derived clock by the pin name attached to the clock: `get_clocks -of_objects [get_pins <PIN_NAME>]`.

Description

An auto-derived clock should be referenced by the source pin object. The auto-derived clock name might change during development due to modifications to the netlist or constraints. Unless it has been renamed, referencing an auto-derived clock by name should be discouraged, because the consequence could be invalidated constraints in subsequent runs after the design has been modified.

Resolution

Modify the constraint to reference the auto-derived clock by the pin name attached to the clock using `[get_clocks -of_objects [get_pins <PIN_NAME>]]`. Alternatively, use the `create_generated_clock` constraint to force the name of the auto-derived clock. An auto-derived clock can be renamed even after being referenced by some timing constraints. For more details about using a generated clock constraint to force a clock name, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-29: Inconsistent Pair of Multicycle Paths

Setup and hold multicycle path constraints should typically reference the same `-start` pair for SLOW-to-FAST synchronous clocks or `-end` pair for FAST-to-SLOW synchronous clocks (see constraint positions <#> in the Timing Constraint window in Vivado IDE).

Description

By default, the `set_multicycle_path` constraint is used to modify the path requirement multipliers with respect to the source clock for hold or the destination clock for setup. For certain use cases, the path requirement must be multiplied with respect to a specific clock edge.

Resolution

For both setup and hold, modify the `set_multicycle_path` constraints to reference the destination clock (`-end`) for SLOW-to-FAST synchronous clocks and the source clock (`-start`) for FAST-to-SLOW synchronous clocks. See the *Vivado Design Suite User Guide: Using Constraints* (UG903) for additional information about properly setting multicycle paths between clocks.

TIMING-30: Sub-Optimal Master Source Pin Selection for Generated Clock

The generated clock `<clock_name>` has a suboptimal master source pin selection, timing can be pessimistic.

Description

A generated clock should reference the clock that is propagating in its direct fanin, although the `create_generated_clock` command lets you specify any reference clock. This DRC warning is reporting that the generated clock is associated to a master clock defined farther upstream than the incoming master clock. In this situation, timing analysis can be more pessimistic and apply additional clock uncertainty on the paths between the master clock and the generated clock. This can lead to slightly more difficult timing closure. It is recommended to associate the generated clock to the master clock source pin that the generated clock is derived.

Resolution

Modify the `create_generated_clock` constraint to reference the master clock source pin from which the generated clock is directly derived in the design.

TIMING-31: Multicycle Path on Phase-Shifted Clock

The MMCM (or PLL) generated clock `<clock_name>` is phase shifted and is involved in one or several multicycle path constraints for setup only. Because the MMCM (or PLL) property `PHASESHIFT_MODE` is set to `LATENCY`, the legacy multicycle path constraint(s) might no longer be required.

Description

In 7 series and UltraScale, a phase-shifted output of an MMCM or PLL is modeled by default as a change in the clock waveform (`PHASESHIFT_MODE=WAVEFORM`). This signifies that the edges of the clock waveform are shifted to account for the pin phase shift.

In UltraScale+, the pin phase shift is modeled by default as a latency propagation delay through the clock modifying block (`PHASESHIFT_MODE=LATENCY`) with no change in the clock waveform.

In `WAVEFORM` mode, multicycle constraints might be needed to adjust the timing path requirement due to the phase shift introduced between the source and destination clock waveforms. In `LATENCY` mode, these multicycle path constraints are no longer required.

For more information on phase shift modes, see MMCM/PLL Phase Shift Modes.

Resolution

Evaluate whether the multicycle constraint is required. If the multicycle path constraint is added to adjust the path requirement in `WAVEFORM` mode, the constraint most likely needs to be removed in `LATENCY` mode. The pin phase shift behavior can also be reverted by setting `PHASESHIFT_MODE=WAVEFORM` on the clock modifying block.

TIMING-32: Bus Skew Constraint Applied on Too Many Signals

The bus skew constraint is set on too many signals (> 2500 paths for UltraScale/UltraScale+ or 1000 paths for 7 series). See the constraint position `<position>` in the timing constraint window in the Vivado IDE. The first endpoint covered by the constraint is `<object>`.

Description

Vivado tries to meet the bus skew constraint by adding extra delay to the individual signals through some routing detours. When the number of signals in the same bus skew constraint is very high, a significant number of routing detours might need to be added. These detours can lead to routing congestion or a failure to meet the bus skew constraint. It can also have a significant impact on `route_design` runtime.

Resolution

Review the bus skew constraint and verify that all the signals should be part of it. After confirming this, consider redesigning this CDC interface to reduce the number of signals.

TIMING-33: Invalid Bus Skew Constraint on Safely Timed Paths

Bus skew constraint set on paths that are safely timed (see the constraint position `<position>` in the timing constraint window in the Vivado IDE). The constraint `set_bus_skew` is recommended on asynchronous paths, or on paths between synchronous clock domains but with a `set_false_path` exception. The first endpoint covered by the constraint is `<object>`.

Description

A bus skew constraint is set on a safely timed crossing that is either a synchronous clock-domain-crossing path or an intra-clock domain path. The `set_bus_skew` constraint should only be used on asynchronous paths. Although `set_bus_skew` is generally recommended to be used in conjunction with `set_max_delay -datapath_only` to control the relative placement between the source and destination registers, the constraint could also be used on a synchronous path when it is covered by a false path exception.

Resolution

Review the bus skew constraint and verify that the constraint is applied on asynchronous clock domain crossing paths. If the paths are expected to be asynchronous but are reported as synchronous by Vivado, review the timing constraints for missing clock group, false path, or max delay datapath only.

TIMING-34: Bus Skew Constraint with Unrealistic Value

There is a bus skew constraint with an unrealistic value (less than the minimum of half a source and destination clock period) or a <1 ns requirement (see constraint position `<position>` in the timing constraint window in Vivado IDE). Setting an aggressive value results in increased runtime. The first endpoint covered by the constraint is `<object>`.

Description

The bus skew constraint is set with a value lower than half the source or destination clock period, or it is less than 1 ns. The bus skew timing can be met by adding routing delay during `route_design`. Trying to meet an aggressive value is runtime intensive and might result in unnecessary routing congestion. It might also be impossible for `route_design` to meet such a target.

Resolution

Review the value set. Set it to higher than at least half the minimum clock period of the source or destination clock, or to 1 ns.

TIMING-35: No Common Node in Paths with the Same Clock

The clock `<clock_name>` has paths without a common node. The first path is found between `<startpoint>` and `<endpoint>`. Review the clock constraints.

Description

An intra-clock path can have no common node between the source and destination clock pins when the clock is defined on more than one object (port/pin/net). In such cases, the path is unsafe to time and should be considered asynchronous because the source and destination clock pins are driven by different clock source pins.

Resolution

Review the `create_clock` constraint and the clock source pin definition. If the clock is intended to be defined on multiple source objects, some asynchronous constraints such as false path or max delay datapath only should be set to cover the paths clocked by different clock source objects.

TIMING-36: Invalid Generated Clock due to No Edge Propagation

There is no rising or falling edge propagation between master clock <clock_name> to generated clock <clock_name>.

Description

There is no rising or falling edge from the master clock source pin that can propagate to the source pin of the generated clock. As a result, there is no known clock relationship between the master clock and the generated clock, and the sign-off timing cannot be accurate.

Resolution

Review the master clock used to define the generated clock. There should be a physical path between the master clock and the generated clock. If there is a physical path, ensure that all the timing arcs along the path are enabled and that the rising and falling clock edges can propagate.

TIMING-37: Bus Skew Constraint Applied on Signals with Fanout

Bus skew constraint is not recommended on signals that have a fanout (see constraint position <position> in the Timing Constraint window in the Vivado IDE). The first endpoint covered by the constraint is <object>.

Description

To avoid any hardware failure from potential logic reconvergence inside the destination clock domain, it is necessary to avoid any fanout on the clock domain crossing paths. A signal should only fan out inside the destination clock domain after the synchronization logic.

Resolution

Review the design and clock domain crossing paths to remove the fanout inside the asynchronous clock domain crossing paths.

TIMING-38: Bus Skew Constraint Applied on Multiple Clocks

Multiple clocks are involved on the source or destination of a bus skew constraint (see constraint position `<position>` in the Timing Constraint window in the Vivado IDE). It is recommended to have only one source clock and one destination clock per bus skew constraint. The first endpoint covered by the constraint is `<object>`.

Description

For proper bus skew timing analysis, it is recommended to have only one source clock and one destination clock that reach the sequential elements specified inside the `set_bus_skew` constraint.

Resolution

Review the CDC crossing to ensure that all the source elements and all the destination elements are only reached by one clock. If multiple clocks reach the source and/or destination registers, the clocks should be defined as logically or physically exclusive.

TIMING-39: Invalid Bus Skew Constraint on Paths that have Too Many Levels of Logic

Bus skew constraint has been applied to paths with more than one logic level (see constraint position `<MESSAGE_STRING>` in the Timing Constraint window in the Vivado IDE). The first endpoint covered by the constraint is `<NETLIST_ELEMENT>`.

Description

To avoid any hardware failure from logic reconvergence inside the destination clock domain, it is necessary to avoid any logic on the clock domain crossing paths. Each register from the source clock domain should drive only one register inside the destination clock domain with no logic in between.

Resolution

Review the design and remove the combinational logic inside the asynchronous CDC crossing.

TIMING-40: Max Skew Violation between OSERDESE3 CLK and CLKDIV Pins Crossing SLRs

A max skew violation exists between the CLK and CLKDIV pins on the OSERDESE3 instance <NETLIST_ELEMENT>, and the clock networks are crossing SLRs to reach the clock pins. It is not recommended to drive the OSERDESE3 CLK and CLKDIV from clock sources placed in another SLR. Check your design.

Description

The CLK and CLKDIV pins of an OSERDESE3 have a specific skew requirement that must not be exceeded. Having the clock sources for CLK and CLKDIV in a different SLR than OSERDESE3 can result in a clock path delay with max skew violations.

Resolution

Review the clocking topology for the OSERDESE3 and ensure that both clock sources for CLK and CLKDIV pins are placed inside the same SLR as the OSERDESE3.

TIMING-41: Invalid Forwarded Clock Defined on an Internal Pin

The forwarded clock <clock_group> is defined on the pin <netlist_element> instead of the port <netlist_element>.

Description

The forwarded clock is defined on a leaf pin connected to the output port instead of being defined on the output port itself. For proper I/O timing computation, the forwarded clock should be defined on the output port.

Resolution

Review the generated clock constraint and move the forwarded clock definition to the output port instead of the internal leaf pin.

TIMING-42: Path Segmentation Detected in the Clock Tree

The <message_string> delay constraint position <message_string> is blocking the propagation of clock <clock_group> on pin <netlist_element>.

Description

A path segmentation is detected on the clock tree due to an invalid startpoint or endpoint in a min/max delay timing constraint. When such a situation occurs, the tool is forced to disable the timing arc to this invalid pin which blocks the clock propagation. This results in an inaccurate timing signoff and a design that could fail in hardware.

Resolution

Review the min/max delay constraint that is applied to the clock tree and only specify valid startpoints and endpoints. For more information, see this link in *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-43: Min Period or Min Pulse Width Violation on Gigabit Transceiver (GT)

The GT pin <instance/pin> has MIN_PERIOD or MIN_PULSE_WIDTH violations. The Power Analysis Report is inaccurate for the GT instance.

Description

The minimum period checks on the GT clock pin ensure that the clock driving the GT instance does not run at a higher frequency than is tolerated by the hardware inside the primitive. In cases of violation, the design could fail in hardware and the power analysis report will be inaccurate.

Resolution

To resolve this violation, check the AC and DC characteristics datasheet of the relevant device family to find the maximum permitted frequency for this primitive pin.

Because this is a silicon-level limitation, you need to tune down the frequency to resolve the violation. Refer to this blog for more information.

TIMING-44: Unreasonable User Intra-Clock Uncertainty

A user clock uncertainty of <delay> ns is defined on clock <clock_name> (see constraint position <position> in the Timing Constraint window in the Vivado IDE). High user clock uncertainty might adversely impact timing closure. Consider reviewing how much user clock uncertainty is required.

Description

It is not recommended to define an overly high user clock uncertainty between two clocks because it impacts timing closure, compile time, and QoR. It could also impact power consumption and make it impossible to close timing.

Resolution

Review the user inter-clock uncertainty and reduce it to the minimum value required.

RECOMMENDED: *Do not overconstrain beyond 0.5 ns. Overconstraining the design can result in increased power for the implementation as well as an increase in runtime. Refer to this link in UltraFast Design Methodology Guide for FPGAs and SoCs (UG949) for more information.*

TIMING-45: Unreasonable User Inter-Clock Uncertainty

A user clock uncertainty of <delay> ns is defined between clocks <clock_name> and <clock_name> (see constraint position <position> in the Timing Constraint window in the Vivado IDE). High user clock uncertainty might adversely impact timing closure. Consider reviewing how much user clock uncertainty is required.

Description

It is not recommended to define an overly high user intra-clock uncertainty because it impacts timing closure, compile time, and QoR. It can also impact power consumption and make it impossible to close timing.

Resolution

Review the user intra-clock uncertainty and reduce it to the minimum value required. It is recommended not to over-constrain beyond 0.5 ns. Refer to this link in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for more information.

TIMING-46: Multicycle Path with Tied CE Pins

One or more multicycle paths are defined between registers <cell_name1> and <cell_name2> with a direct connection and the `CE` pins connected to `VCC` (see constraint position `<position>` in the Timing Constraint window in the Vivado IDE). This might result in an inaccurate path requirement.

Description

The source and destination registers do not have their `CE` pin controlled by a dynamic signal (`CE` pin tied to `VCC`). Because there is a direct datapath connection between the source and destination registers, the data on this path is captured based on a single-cycle path requirement. The multicycle path defined on the path does not represent the behaviour in hardware and might result in the design failing in hardware.

Resolution

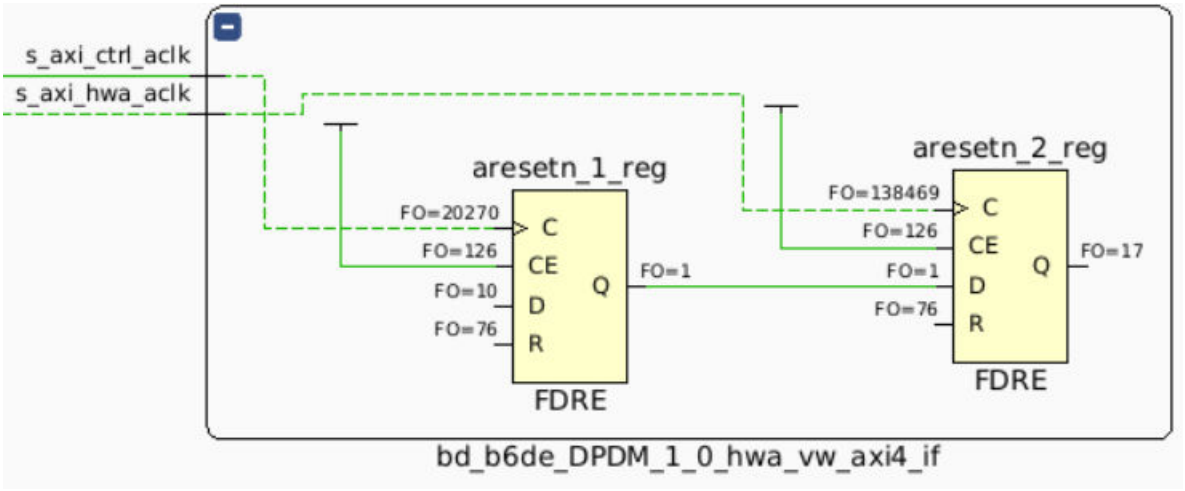
The path and the timing constraint should be reviewed. If the `CE` pins are expected to be tied, the multicycle path needs to be removed. If the path is expected to be a multicycle path, the `CE` pin of the source and/or destination register(s) should be driven by a dynamic signal that toggles according to the multicycle constraint that is provided.

Example

The following constraint is written by the user and is applied on the datapath in between the two registers as shown in the following schematic, where the `CE` pin of the destination register is connected to VCC:

```
set_multicycle_path -setup -end -from [get_clocks -of [get_ports -
scoped_to_current_instance s_axi_ctrl_aclk]] -to [get_clocks -of [get_ports
-scoped_to_current_instance s_axi_hwa_aclk]] 2
```

Figure 254: **Multicycle Path with Tied CE Pins Example**



Check whether data is changed at every clock edge outside the scope of the multicycle path. If it does, make sure to add the required logic tied to the `CE` pin.

The following figure shows a flop-to-flop path that has the proper clock enable logic tied to the `CE` pin, making the flop enable on alternating clock cycles:

```
set_multicycle_path 2 -setup -from [get_pins data0_reg/C] -to [get_pins
data1_reg/D]
```

Figure 255: **Flop-to-Flop Path**

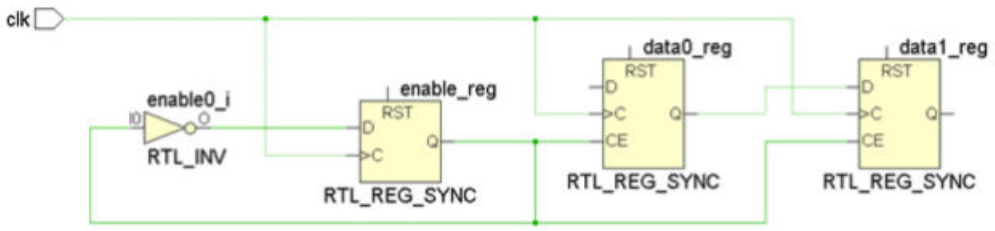
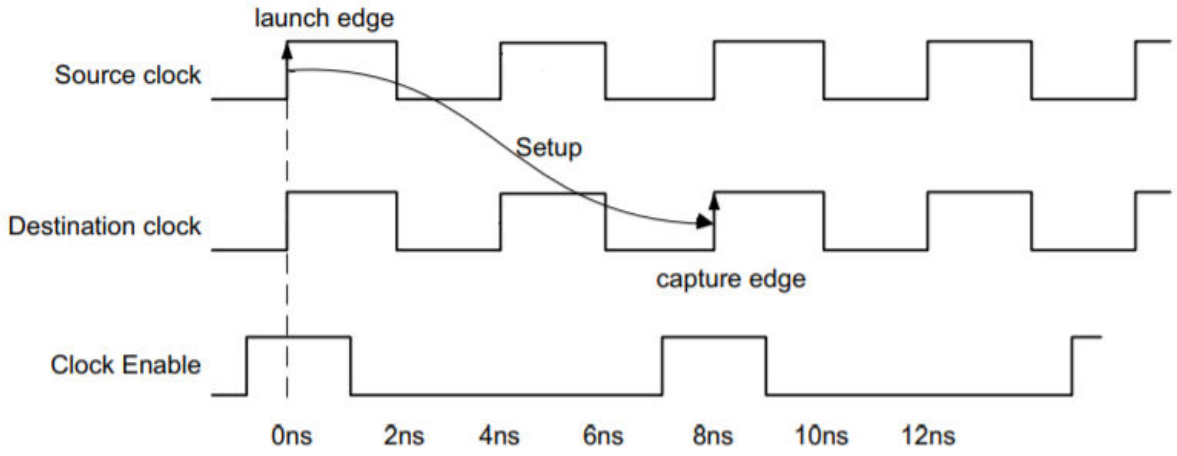


Figure 256: **Timing Diagram**



TIMING-47: False Path, Asynchronous Clock Group or Max Delay Datapath Only Constraint between Synchronous Clocks

A `<message_string>` timing constraint is set between synchronous clocks `<clock_group>` and `<clock_group>` (see the constraint position `<message_string>` in the Timing Constraint window in the Vivado IDE). Masking entire synchronous clock domains using `set_false_path`, `set_clock_groups`, or `set_max_delay -datapath_only` might result in failure in hardware.

Description

There should not be a `false_path`, `set_clock_group -asynchronous`, or `set_max_delay -datapath_only` constraint on a synchronous clock domain crossing. With such constraints, the paths will not be properly timed, the sign-off timing will be inaccurate, and the design could fail in hardware.

Resolution

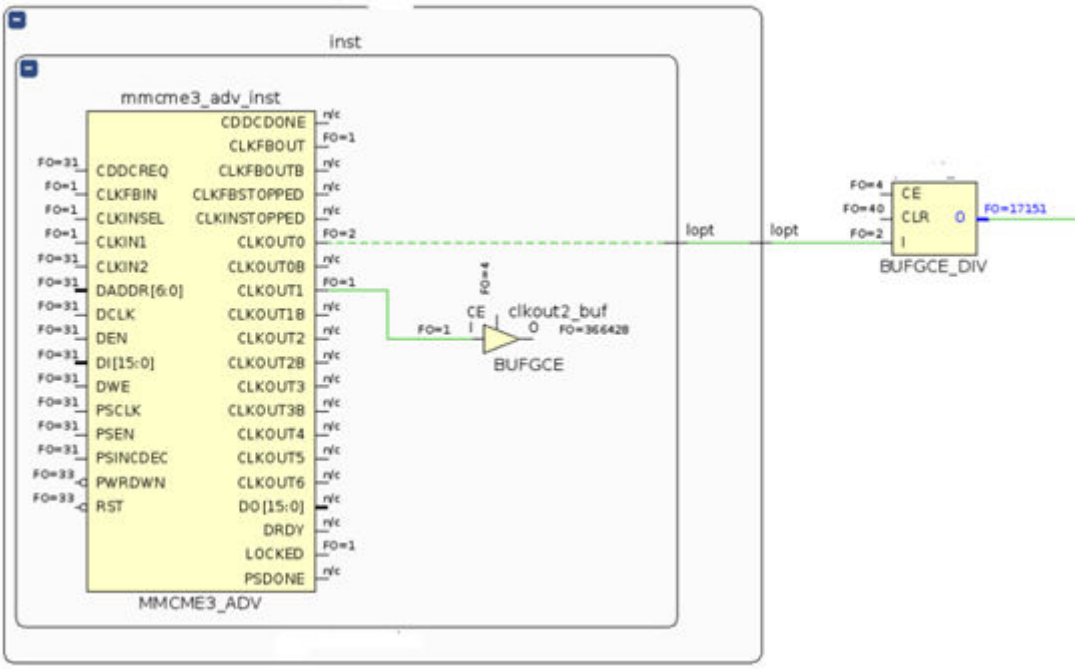
Remove the `false_path`, `set_clock_group -asynchronous`, or `set_max_delay -datapath_only` constraints that are applied between the synchronous clocks. If the clocks are expected to be asynchronous, add an asynchronous clock constraint with the proper synchronization circuitry for the asynchronous clock domain crossing.

To understand which clock domain crossing is synchronous or asynchronous, follow the clock interaction report.

Example

In the following example, two MMCM outputs are used to generate the clocks. The BUFGCE_DIV output has a divisional clock, `clk1`, which has the master clock coming from `CLKOUT0`. `CLKOUT1` has a generated clock, `clk2`. If you add a constraint such as `set_clock_groups -asynchronous -group [get_clocks clk1] -group [get_clocks clk2]`, there will be a TIMING-47 warning on it.

Figure 257: **False Path, Asynchronous Clock Group, or Max Delay Datapath Only Constraint between Synchronous Clocks**



Because both `clk1` and `clk2` are coming from the same MMCM, they are considered to be synchronous to each other. Data crossing to these domains is also considered to be under the synchronous CDC. There is therefore no need to add `set_clock_groups -asynchronous`.

TIMING-48: Max Delay Datapath Only Constraint on Latch Input

A max delay datapath only constraint has been detected on the input of latch <pin_name> (see constraint position <position> in the Timing Constraint window in the Vivado IDE). This constraint is typically used on asynchronous clock domain crossings and can trigger unrealistic latch time borrowing which impacts the QoR of the downstream timing paths.

Description

A max delay constraint should not be applied on the input of latch because the constraint changes the requirement of the path and could cause unrealistic latch time borrowing that does not match the behavior in hardware. This could result in a hardware failure.

Resolution

Verify the timing constraint against the hardware behaviour and remove the max delay datapath only constraint applied on the latch input if it does not match the behavior in hardware.

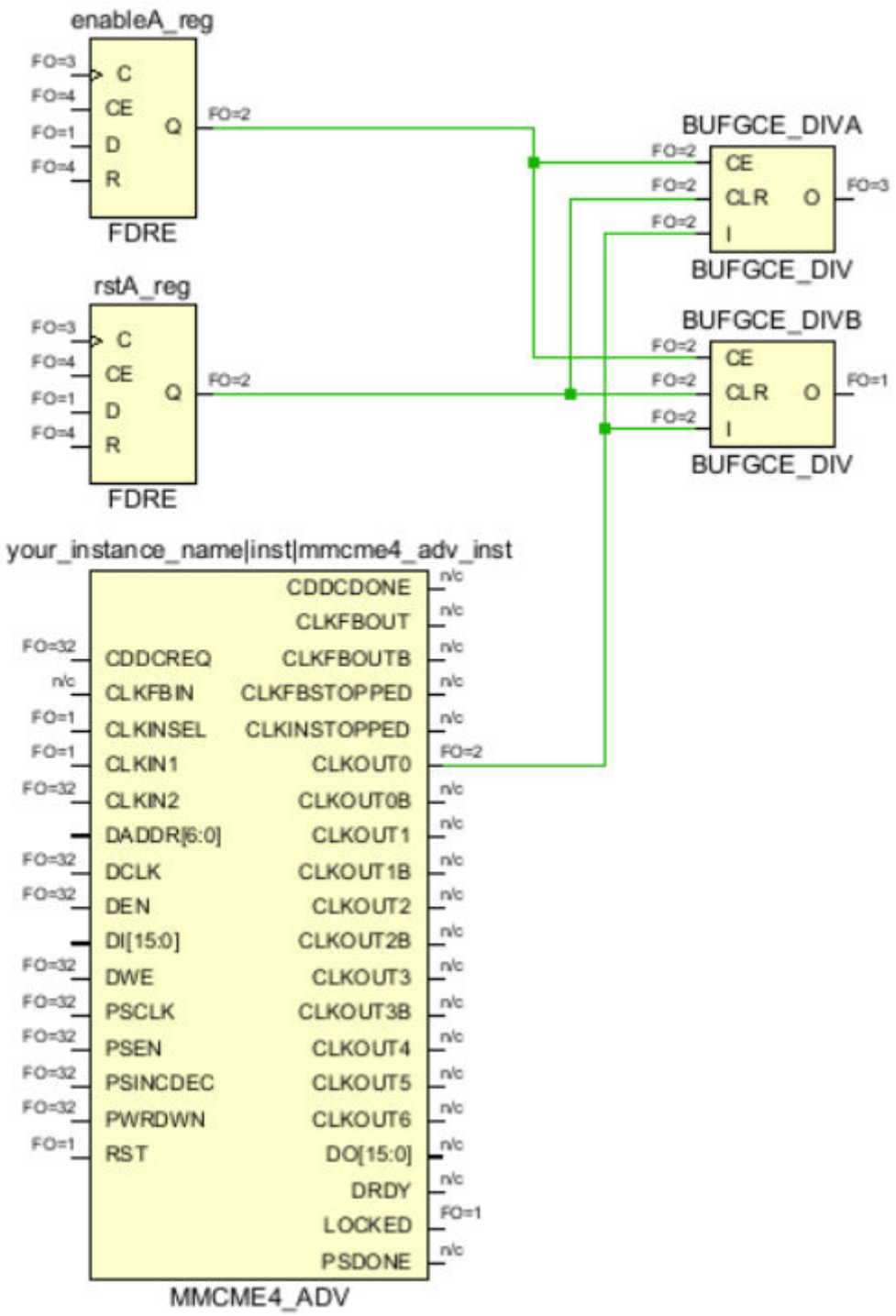
TIMING-49: Unsafe Enable or Reset Topology from Parallel BUFGCE_DIV

To ensure safe timing on paths with clocks driven by parallel BUFGCE_DIV cells with a BUFGCE_DIVIDE property set to a value greater than 1 on both, both buffers <buffer> and <buffer> must use the same enable signal (CE) and the same clear signal (CLR). The clear signal must not be connected to power or ground. Otherwise, the divided clock might become phase-shifted to one another in hardware. It is recommended to use the safe clock startup reset circuitry to reset both BUFGCE_DIV buffers at the same time.

Description

Clocking topologies with parallel BUFGCE_DIV buffers that have BUFGCE_DIVIDE>1 must have common control signals to prevent a phase ambiguity that might occur from the internal state of the BUFGCE_DIV. Such a situation could lead to the clock buffer being reset during different clock cycles and the outputs of the BUFGCE_DIV having an unknown clock relationship. The design could fail in hardware. The following diagram shows the correct implementation of the circuit.

Figure 258: **Correct Implementation of BUFGCE_DIV Buffers**



Resolution

The `CLR` pin of the parallel BUFGCE_DIV buffers should be tied to or be driven by the same signal. Use safe clock startup reset circuitry for seamless operation on hardware. This can be enabled in the clocking wizard IP. See this link in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for more information.

TIMING-50: Unrealistic Path Requirement between Same-Level Latches

There is a timing path between the source pin `<pin>` and the destination pin `<pin>`. The two latches are `<positive|negative>` level-sensitive with a 0 ns path requirement. The 0 ns path requirement is coming from the conservative latch analysis and could severely impact the runtime due to a time borrowing calculation. Such topology is not recommended unless there is a multicycle path constraint to adjust the path requirement to a realistic value (at least half a clock period).

Description

The Vivado timing engine calculates the conservative requirement of 0 ns between two latches that have the same sensitivity level. Because time borrowing happens on the second latch, such topology can significantly impact the runtime of timing analysis and implementation.

Resolution

This is an uncommon topology because cascaded latches typically involve latches that are sensitive to opposite levels to prevent data from crossing multiple latches during the same clock cycle. Review the topology of the latches. If this is the expected connectivity, confirm the behavior in hardware and use a multicycle path between the cascaded latches to increase the path requirement to a realistic value.

TIMING-51: No Common Phase between Related Clocks from Parallel CMBs

The clocks <clock_name> and <clock_name> are timed together but have no phase relationship. The design could fail in hardware. The clocks originate from two parallel clock modifying blocks and at least one of the MMCM, PLL, or XPLL input clock dividers is not set to 1. To be safely timed, all MMCMs, PLLs, or XPLLs involved in parallel clocking must have the clock divider set to 1.

Description

When the clocking topology involves clocks derived from parallel MMCMs or PLLs, it is only safe to time when the clock divider (DIVCLK_DIVIDE) of each CMB is set to 1. Any other value for DIVCLK_DIVIDE results in the loss of the clock relationship between the parallel CMBs and the clock domain crossing becomes asynchronous.

Resolution

If the clock domain crossing between the parallel PLLs is expected to be synchronous, change the clock dividers (DIVCLK_DIVIDE) to 1. If the clock domain crossing is expected to be asynchronous, change the timing constraints to make the clock domain crossing asynchronous and add the proper synchronization circuitry in the destination clock domain. See this link in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for more information.

TIMING-52: No Common Phase between Related Clocks from Spread Spectrum MMCM

Spread spectrum is enabled for MMCM <cell>. In this mode, it is unsafe to time between <clock_name> and <clock_name> due to the variations of the VCO frequency.

Description

In spread spectrum mode, due to the variations of the VCO frequency over time, it is unsafe to time between two different outputs of the same MMCM.

Resolution

Confirm that spread spectrum mode is required for this MMCM. If this mode is required, change the clocking topology to use a single output of the MMCM and use a clock divider to divide the clock outside of the MMCM. If you cannot change the clocking topology, the crossing between the multiple clocks of the MMCM must be considered asynchronous and the proper circuitry must be added to transfer data between these clock domains.

TIMING-53: No Common Phase between Related Clocks from DPLL

The clocks `<clock_name>` and `<clock_name>` are timed together but have no phase relationship. The design could fail in hardware. One of the clocks originates from the DPLL `<cell>` that is not using the phase detector, or its incoming clock is not connected to the `CLKIN_DESKEW` pin. Under these conditions, it is unsafe to time between the master clock of the DPLL and its auto-derived clock (or downstream generated clocks) because the relationship between the clocks is unknown.

Description

This check applies to Versal adaptive SoC only.

In Versal adaptive SoC devices, it is only safe to time between the DPLL input clock and one of its output clocks when the phase detector (`CLKOUTx_PHASE_CTRL=01`) is used and the `CLKIN_DESKEW` pin is connected to the `CLKIN` pin. In all other cases, the phase relationship between the master clock and the auto-derived clocks is unknown and the clock domain crossing between them should be considered asynchronous: any path in the crossing should have the proper synchronization circuitry in the destination clock domain.

Resolution

If the paths between the master clock and the auto-derived clock are expected to be synchronous, review the DPLL settings and connectivity to use the proper deskew configuration. If not, the clock domain crossing should be considered asynchronous and the design should have the proper logic synchronization in the destination clock domain.

TIMING-54: Scoped False Path, Clock Group, or Max Delay Datapath Only Constraint between Clocks

A scoped <message_string> timing constraint is set between clocks <clock_group> and <clock_group> (see the constraint position <message_string> in the Timing Constraint window in the Vivado IDE). It is not recommended to define such scoped constraints between clocks because the constraint impacts timing paths outside of the scope.

Description

Timing clocks propagate globally and cannot be restricted to a hierarchical module. It is not recommended to define a scoped timing constraint such as clock group, false path, and max delay datapath only between clocks because the constraint impacts the sign-off timing at a global level for the entire design. To avoid confusion and unexpected coverage beyond the scope of the hierarchical module, such constraints should be defined at the top level.

Resolution

Review the timing constraint and move it to the top level if the intent is to apply the constraint to the entire design. If the intent is only to apply the constraint under the scope of the instance, the timing constraint and the topology of the design might need to be changed. For more information, see this link in *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-55: Multiple Clocks Reaching a CMB Deskew Pin

Multiple clocks propagate to the deskew pin <pin_name> causing timing analysis to be inaccurate and potentially resulting in hardware failures. The timing constraints should be updated so that a single clock propagates to the deskew pin. The list of clocks reaching the pin is <clock_names>.

Description

This is check applies to Versal adaptive SoC only.

For proper timing analysis, it is not supported to have multiple clocks that propagate to the deskew pin of the MMCM/XPLL/DPLL. Failure to comply results in inaccurate signoff timing with a risk of a hardware failure.

Resolution

Review the clocking topology and timing constraints to ensure that only one clock propagates to the MMCM/XPLL/DPLL deskew pin.

TIMING-56: Missing Logically or Physically Excluded Clock Groups Constraint

Multiple clocks are user generated or auto-derived on the source pin(s) `<pin_names>` but are not logically or physically exclusive with respect to one another. To have the static timing analysis match the behavior in hardware, there cannot be multiple clocks generated on the same pin(s). In such cases, the clocks should be defined as physically or logically exclusive. The list of clocks generated on the source pin(s) is: `<clock_names>`.

Description

When the output pin of a clocking primitive has multiple generated or auto-derived clocks present, these clocks cannot be available on hardware simultaneously. To match static timing analysis with hardware behavior, provide physically or logically exclusive clock group constraints between these clocks. Otherwise, some clock pairs will be timed although they do not exist in hardware.

Resolution

Ensure that the proper physically or logically exclusive clock group constraint is provided between the clocks generated on the source pin based on clock tree topology. For more information, see this link in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-57: Unsupported Configuration with PHASESHIFT_MODE and Digital Deskew

The clock modifying block `<netlist_element>` is configured for digital deskew and has `PHASESHIFT_MODE=WAVEFORM`. This combination is unsupported, and timing analysis will proceed by treating it as if `PHASESHIFT_MODE = LATENCY`. Change the specified cell configuration to `PHASESHIFT_MODE= LATENCY` and ensure that no timing constraints are written against the expectation of `PHASESHIFT_MODE=WAVEFORM`.

Description

This check applies to Versal adaptive SoC only.

When the MMCM/XPLL/DPLL is configured with digital deskew, the pin phase shift can only be modeled as a latency delay through the CMB and not as a change in the clock waveform. This clocking element is configured with `PHASESHIFT_MODE=WAVEFORM`, which cannot be honored by the timer. The violation does not impact the signoff timing accuracy because the tool enforces the latency modeling.

Resolution

To suppress this violation, change the pin phase shift modeling to `PHASESHIFT_MODE=LATENCY`. Review any existing timing exception to support the original intent of `WAVEFORM` modeling.

Report QoR Suggestion RTL Code Change Example

This section details some of the QoR suggestion triggers that require RTL edits. It gives a brief description of each one and an example of the modification required.

RQS_TIMING-201: Add an Output Register to RAM

Adding an output register to a RAM improves the clock to out time of the RAM read data path. This provides more flexibility to the place and route tools to place the RAM optimally as well as an option to place the register in the fabric instead of the RAM to optimize the critical path.

Output registers can be easily inferred by the synthesis tool. They must either have a synchronous reset or no reset.

Verilog Code Example

Figure 259: **Before**

```
module single_sdp_ram #(
    parameter C_DATA_WIDTH = 16,
    parameter C_ADDR_WIDTH = 10) (
    input CLKA,
    input WEA,
    input [C_ADDR_WIDTH-1:0] ADDRA,
    input [C_DATA_WIDTH-1:0] DINA,
    input ENB,
    input [C_ADDR_WIDTH-1:0] ADDRB,
    output [C_DATA_WIDTH-1:0] DOUTB,
    input CLKB
    );

  reg [C_DATA_WIDTH-1:0] ram_i [2**C_ADDR_WIDTH-1:0];
  reg [C_DATA_WIDTH-1:0] ram_data = {C_DATA_WIDTH{1'b0}};

  always @(posedge CLKA)
    if (WEA)
      ram_i[ADDRA] <= DINA;

  always @(posedge CLKB)
    if (ENB)
      ram_data <= ram_i[ADDRB];

  // 1 clock cycle read latency at the cost of a longer clock-to-out timing
   assign DOUTB = ram_data;

endmodule
```

Figure 260: **After**

```
module single_sdp_ram #(
    parameter C_DATA_WIDTH = 16,
    parameter C_ADDR_WIDTH = 10) (
    input CLKA,
    input WEA,
    input [C_ADDR_WIDTH-1:0] ADDRA,
    input [C_DATA_WIDTH-1:0] DINA,
    input ENB,
    input [C_ADDR_WIDTH-1:0] ADDRB,
    output [C_DATA_WIDTH-1:0] DOUTB,
    input CLKB
    );

  reg [C_DATA_WIDTH-1:0] ram_i [2**C_ADDR_WIDTH-1:0];
  reg [C_DATA_WIDTH-1:0] ram_data = {C_DATA_WIDTH{1'b0}};

  always @(posedge CLKA)
    if (WEA)
      ram_i[ADDRA] <= DINA;

  always @(posedge CLKB)
    if (ENB)
      ram_data <= ram_i[ADDRB];

  // 2 clock cycle read latency with improved clock-to-out timing
  reg [C_DATA_WIDTH-1:0] doutb_reg = {C_DATA_WIDTH{1'b0}};
  always @(posedge CLKB)
   doutb_reg <= ram_data;

  assign DOUTB = doutb_reg;

endmodule
```

VHDL Code Example

Figure 261: **Before**

```
-- 2D Array Declaration for RAM signal
type ram_type is array (2**C_ADDR_WIDTH-1 downto 0) of std_logic_vector (C_DATA_WIDTH-1 downto 0);
signal RAM_DATA : std_logic_vector(C_DATA_WIDTH-1 downto 0) ;

...
process(CLKA)
begin
    if(CLKA'event and CLKA = '1') then
        if(WEA = '1') then
            RAM(to_integer(unsigned(ADDRA))) <= DINA;
        end if;
    end if;
end process;

process(CLKB)
begin
    if(CLKB'event and CLKB = '1') then
        if(ENB = '1') then
            RAM_DATA <= RAM(to_integer(unsigned(ADDRB)));
        end if;
    end if;
end process;

-- Read latency of 1 but slower clock to out time
DOUTB <= RAM_DATA;
```

Figure 262: **After**

```
-- 2D Array Declaration for RAM signal
type ram_type is array (2**C_ADDR_WIDTH-1 downto 0) of std_logic_vector (C_DATA_WIDTH-1 downto 0);
signal RAM_DATA : std_logic_vector(C_DATA_WIDTH-1 downto 0) ;
signal DOUTB_REG : std_logic_vector(C_DATA_WIDTH-1 downto 0) := (others => '0');
...
process(CLKA)
begin
    if(CLKA'event and CLKA = '1') then
        if(WEA = '1') then
            RAM(to_integer(unsigned(ADDRA))) <= DINA;
        end if;
    end if;
end process;

process(CLKB)
begin
    if(CLKB'event and CLKB = '1') then
        if(ENB = '1') then
            RAM_DATA <= RAM(to_integer(unsigned(ADDRB)));
        end if;
    end if;
end process;

-- Read latency of 2 but faster clock to out time
process(CLKB)
begin
    if(CLKB'event and CLKB= '1') then
        DOUTB_REG <= RAM_DATA;
    end if;
end process;
```

RQS_TIMING-202: Add Extra Pipelining to Wide Multipliers

Wide multipliers (where at least one port is greater than the maximum width supported by the DSP slice in the given architecture) need extra pipelines to achieve the maximum operating frequency of the DSP slice. The number of pipeline stages require changes depending on the width required.

By adding extra stages to the output of wide multipliers in the RTL, synthesis moves them to optimal positions which makes recoding very simple.

Verilog Code Example

Figure 263: **Before**

```
module wide_mulitplier #(parameter DATA_WIDTH=30) (
    input clk,
    input [DATA_WIDTH-1:0] a,
    input [DATA_WIDTH-1:0] b,
    output [2*DATA_WIDTH-1:0] p
    );


    reg [DATA_WIDTH-1:0] a_r;
    reg [DATA_WIDTH-1:0] b_r;
    reg [2*DATA_WIDTH-1:0] m_r;

    always @ (posedge clk)
    begin
        a_r <= a;
        b_r <= b;
        m_r <= a_r * b_r;
    end

    assign p = m_r;

endmodule
```

Figure 264: **After**

```
module wide_mulitplier #(parameter DATA_WIDTH=30) (
    input clk,
    input [DATA_WIDTH-1:0] a,
    input [DATA_WIDTH-1:0] b,
    output [2*DATA_WIDTH-1:0] p
    );


    reg [DATA_WIDTH-1:0] a_r;
    reg [DATA_WIDTH-1:0] b_r;
    reg [2*DATA_WIDTH-1:0] m_r;
    reg [2*DATA_WIDTH-1:0] m_2r;
    reg [2*DATA_WIDTH-1:0] m_3r;
    reg [2*DATA_WIDTH-1:0] m_4r;

    always @ (posedge clk)
    begin
        a_r <= a;
        b_r <= b;
        m_r <= a_r * b_r;
        // Add more pipeline stages after the multiplier
        m_2r <= m_r;
        m_3r <= m_2r;
        m_4r <= m_3r;
    end

    assign p = m_4r;

endmodule
```

VHDL Code Example

Figure 265: **Before**

```
entity wide_multiplier is
    Generic ( DATA_WIDTH : integer := 30);
    Port ( clk : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
           b : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
           p : out STD_LOGIC_VECTOR (2*DATA_WIDTH-1 downto 0));
end wide_multiplier;

architecture Behavioral of wide_multiplier is

signal a_r : SIGNED(DATA_WIDTH-1 downto 0);
signal b_r : SIGNED(DATA_WIDTH-1 downto 0);
signal m_r : SIGNED(2*DATA_WIDTH-1 downto 0);

begin

process (clk)
begin
    if rising_edge (clk) then
        a_r <= SIGNED(a);
        b_r <= SIGNED(b);
        m_r <= a_r * b_r;
    end if;
end process;
p <= STD_LOGIC_VECTOR(m_r);


end Behavioral;
```

Figure 266: **After**

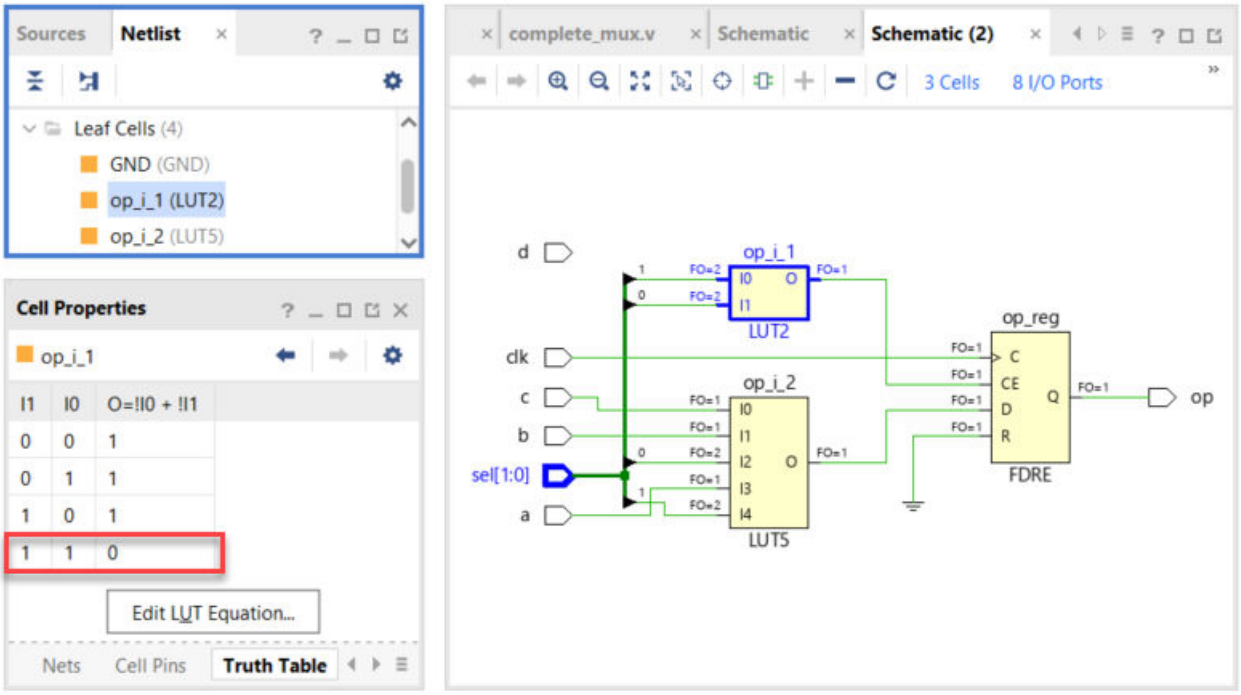
```
entity wide_multiplier is
    Generic ( DATA_WIDTH : integer := 30);
    Port ( clk : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
           b : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
           p : out STD_LOGIC_VECTOR (2*DATA_WIDTH-1 downto 0));
end wide_multiplier;

architecture Behavioral of wide_multiplier is

signal a_r : SIGNED(DATA_WIDTH-1 downto 0);
signal b_r : SIGNED(DATA_WIDTH-1 downto 0);
signal m_r : SIGNED(2*DATA_WIDTH-1 downto 0);
signal m_2r : SIGNED(2*DATA_WIDTH-1 downto 0);
signal m_3r : SIGNED(2*DATA_WIDTH-1 downto 0);
signal m_4r : SIGNED(2*DATA_WIDTH-1 downto 0);

begin

process (clk)
begin
    if rising_edge (clk) then
        a_r <= SIGNED(a);
        b_r <= SIGNED(b);
        m_r <= a_r * b_r;
        -- Add extra pipes after the multiplier
        m_2r <= m_r;
        m_3r <= m_2r;
        m_4r <= m_3r;
    end if;
end process;
p <= STD_LOGIC_VECTOR(m_4r);

end Behavioral;
```

RQS_UTIL-10: Incomplete Case Statement Increasing Control Sets

When case statements are incomplete, the output is forced to remember the previous state which infers a clock enable. It is recommended to define all states. For additional states, either define the output as a previously used signal or a constant 1 or constant 0 in order not to increase logic.

The following figure shows the inference of a CE LUT when state "11" is undefined.

Figure 267: **CE Inference Incomplete Case**



Verilog Code Example

Before:

```
module incomplete_mux(
input clk,
input a,b,c,d,
input [1:0] sel,
output reg op
);

always @(posedge clk)
  begin
  case (sel)
    2'b00 : op <= a;
    2'b01 : op <= b;
    2'b10 : op <= c;
    default: ;
  endcase
end
endmodule
```

After:

```
module incomplete_mux(
input clk,
input a,b,c,d,
input [1:0] sel,
output reg op
);
```

```
always @(posedge clk)
  begin
  case (sel)
    2'b00 : op <= a;
    2'b01 : op <= b;
    2'b10 : op <= c;
    2'b11 : op <= a; // ADDED
    default: ;
  endcase
end
endmodule
```

VHDL Code Example

Before:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity incomplete_mux is
Port ( clk : in STD_LOGIC;
       a,b,c,d : in STD_LOGIC;
       sel : in STD_LOGIC_VECTOR (1 downto 0);
       op : out STD_LOGIC);
end incomplete_mux;

architecture Behavioral of incomplete_mux is

begin

process (clk)
begin
  if rising_edge(clk) then
    case sel is
      when "00" => op <= a;
      when "01" => op <= b;
      when "10" => op <= c;
      when others => null;
    end case;
  end if;
end process;

end Behavioral;
```

After:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity incomplete_mux is
Port ( clk : in STD_LOGIC;
       a,b,c,d : in STD_LOGIC;
       sel : in STD_LOGIC_VECTOR (1 downto 0);
       op : out STD_LOGIC);
end incomplete_mux;

architecture Behavioral of incomplete_mux is

begin
```

```
process (clk)
begin
  if rising_edge(clk) then
    case sel is
      when "00" => op <= a;
      when "01" => op <= b;
      when "10" => op <= c;
      when "11" => op <= a;
    end case;
  end if;
end process;

end Behavioral;
```

RQS_UTIL-203: Large ROM Inferred using Distributed RAM

ROMs whose array depth is significantly over 64 bits are better inferred into Block RAM. The synthesis tool tries to do this by default but sometimes it is unable to do so due to coding or constraint restrictions.

The primary reason for not inferring a Block RAM is a missing output register. Block RAMs only support a synchronous read, but Distributed RAMs do not have this requirement. The second reason when reading the array or a `ROM_STYLE` attribute forcing the type of resource that must be inferred.

By making a simple modification, you can expect improvements in LUT utilization, timing, and where applicable, congestion.

Verilog Code Example

Figure 268: **Before**

```
module sp_rom (clk, en, addr, dout);
input clk;
input en;
input [5:0] addr;
output  [83:0] dout;

reg [83:0] data;

// Combinatoral read process prevents Block RAM usage
always_comb
begin
  data <= 84'h11223344;
    case(addr)
      6'b000000: data <= 84'hFF200A; 6'b100000: data <= 84'hFF2222;
      6'b000001: data <= 84'hFF0300; 6'b100001: data <= 84'hFF4001;
      ...
      6'b011111: data <= 84'hFF0102; 6'b111111: data <= 84'hFF400D;
      default: data <= 84'hFF00111;
    endcase
end

assign dout = data;

endmodule
```

Figure 269: **After**

```
module sp_rom (clk, en, addr, dout);
input clk;
input en;
input [5:0] addr;
output  [83:0] dout;

reg [83:0] data;

always @(posedge clk) // Add an output register to help this ROM get
                      //  inferred as block RAM.
begin
  data <= 84'h11223344;
  if (en)
    case(addr)
      6'b000000: data <= 84'hFF200A; 6'b100000: data <= 84'hFF2222;
      6'b000001: data <= 84'hFF0300; 6'b100001: data <= 84'hFF4001;
      ...
      6'b011111: data <= 84'hFF0102; 6'b111111: data <= 84'hFF400D;
      default: data <= 84'hFF00111;
    endcase
end

assign dout = data;

endmodule
```

VHDL Code Example

Figure 270: **Before**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sp_rom is
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR (9 downto 0);
           dout : out STD_LOGIC_VECTOR (15 downto 0));
end sp_rom;

architecture Behavioral of sp_rom is

signal data : STD_LOGIC_VECTOR(15 downto 0);

begin

-- Unregistered ROM read prevents Block RAM usage
process (addr, en)
begin
    if (en = '1') then
      case (TO_INTEGER(UNSIGNED(addr))) is
         when 0 => data <= X"3423";
         when 1 => data <= X"ED77";
         ...
         when 1023 => data <= X"CD34";
         when others => data <= X"0111";
      end case;
   end if;
end process;

dout <= data;

end Behavioral;
```

Figure 271: **After**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sp_rom is
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR (9 downto 0);
           dout : out STD_LOGIC_VECTOR (15 downto 0));
end sp_rom;

architecture Behavioral of sp_rom is

signal data : STD_LOGIC_VECTOR(15 downto 0);

begin

-- Register process to enable inference of Block RAM
process (clk)
begin
  if rising_edge (clk) then
    if (en = '1') then
      case (TO_INTEGER(UNSIGNED(addr))) is
        when 0 => data <= X"3423";
        when 1 => data <= X"ED77";
        ...
        when 1023 => data <= X"CD34";
        when others => data <= X"0111";
      end case;
    end if;
  end if;
end process;

dout <= data;

end Behavioral;
```

Reference Design Files

Download the reference design files associated with this appendix from the AMD website.

Appendix C

Additional Resources and Legal Notices

Finding Additional Documentation

Documentation Portal

The AMD Adaptive Computing Documentation Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Documentation Portal, go to https://docs.xilinx.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs webpage.

Note: For more information on DocNav, see the Documentation Navigator webpage.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
2. *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894)
3. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
4. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)
5. *Vivado Design Suite User Guide: Using Constraints* (UG903)
6. *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)
7. *Vivado Design Suite User Guide: Implementation* (UG904)
8. *Vivado Design Suite User Guide: Power Analysis and Optimization* (UG907)
9. *7 Series FPGAs Clocking Resources User Guide* (UG472)
10. *Vivado Design Suite Properties Reference Guide* (UG912)
11. All Vivado Design Suite Documentation

Training Resources

AMD provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Designing FPGAs Using the Vivado Design Suite 1
2. Designing FPGAs Using the Vivado Design Suite 2
3. Designing FPGAs Using the Vivado Design Suite 3
4. Designing FPGAs Using the Vivado Design Suite 4
5. Vivado Design Suite QuickTake Video Tutorials
6. Vivado Design Suite QuickTake Video: Advanced Clock Constraints and Analysis
7. Vivado Design Suite QuickTake Video: Analyzing Implementation Results

8. Vivado Design Suite QuickTake Video: Timing Analysis Controls
9. Vivado Design Suite QuickTake Video: Cross Clock Domain Checking - CDC Analysis

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/16/2023 Version 2023.1	
Automatically Generating and Applying Suggestions	Updated screenshots for RQS Auto Flow and general updates throughout the document.

Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2012–2023 Advanced Micro Devices, Inc. AMD, the AMD Arrow logo, Versal, Vitis, Vivado, and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.