Vivado Design Suite User Guide

Design Analysis and Closure Techniques

UG906 (v2025.2) December 10, 2025

AMD

Table of Contents

Chapter 1

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. You can access the AMD Versal™ adaptive SoC design processes on the Design Hubs page. You can also use the Design Flow Assistant to better understand the design flows and find content that is specific to your intended design needs. 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

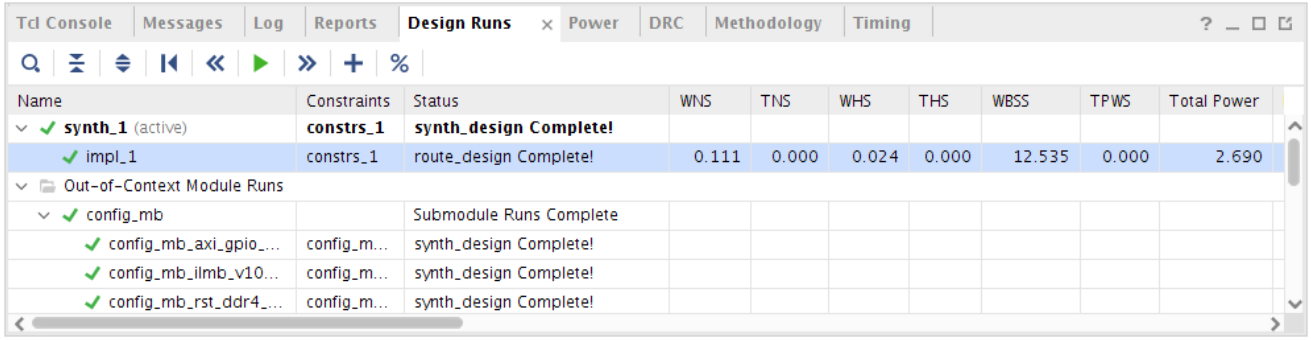
The following sections introduce you to design analysis in the AMD Vivado™ Design Suite IDE. The IDE supports project-based workflows, but many features also apply to non-project-based workflows. In this chapter, you learn how to do the following:

- Use the Design Runs window for quick analysis
- Perform interactive logic analysis using windows, search tools, and reports
- Perform methodology analysis
- Analyze placement and routing

Using the Design Runs Window

Start your analysis with the Design Runs window. This window shows the current state of each run, including whether it is running, completed successfully, or completed with errors. For details about schematic-based analysis, see Using the Schematic Window in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Figure 1: **Design Runs Window**



The Design Runs window includes the following columns:

- **Name:** Name of the run and target part.

- **Constraints:** Constraints set used in the run.
- **Status:** Status of the last completed step.
- **WNS, TNS, WHS, THS, WBSS, TPWS:** Timing score of the run used to quickly check timing results. If timing is not met, begin analysis with the Timing Summary Report. See Report Timing Summary for more information on these numbers.

  Note: WBSS represents the Worst Bus Skew Slack reported by `report_bus_skew`.
- **Total Power:** Total power estimate

It also provides the following information:

- The run strategy.
- 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 number of nets that were not successfully routed.
- The utilization of the design LUT, FF, block RAMs, DSP, and if applicable, UltraRAMs.
- A brief description of the run strategy.
- Methodology check violations.
- Available QoR suggestions.
- The incremental mode of the design run.

To analyze a run, right-click the run and choose from several actions such as opening the run or setting up incremental compile and QoR suggestions.

Logic Analysis in the IDE

Logic Analysis Features

Use logic analysis features to explore and evaluate your design in the Vivado IDE. This section includes the following topics.

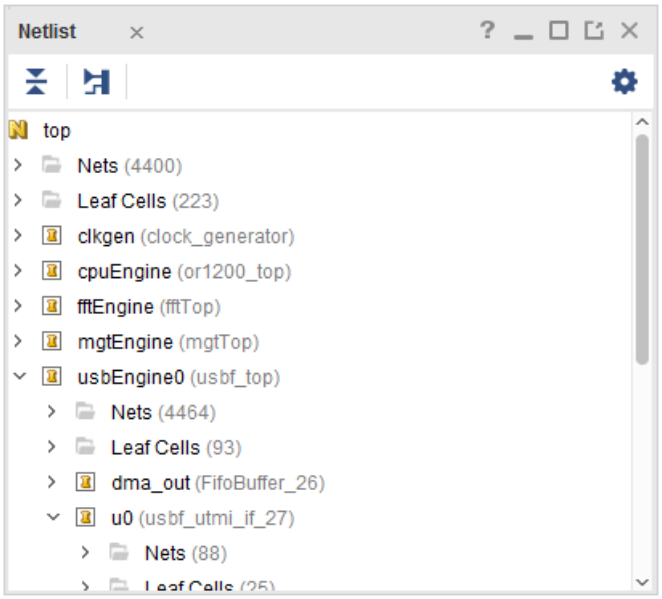
- Using the Netlist Window
- Using the Hierarchy Window
- Using the Schematic Window
- Searching for Objects Using the Find Dialog Box

- Analyzing Utilization Statistics
- Using Report DRCs
- Methodology Analysis

Using the Netlist Window

Use the Netlist window to view the design hierarchy as it exists in the netlist, processed by the synthesis tools. This window helps you explore the logical hierarchy of the design.

Figure 2: **Netlist Window**



Depending on your synthesis settings, the netlist hierarchy can match the original RTL, or contain no hierarchy. When run with the default settings, synthesis preserves your RTL hierarchy while optimizing logic, resulting in an optimal netlist for implementation.

***Note*:** This results in the netlist hierarchy remaining recognizable, but the interfaces to the hierarchies can be modified. Some pins and levels of hierarchy can be missing after optimization across hierarchical boundaries. The netlist hierarchy appears as a folder tree. At each level, you see the following:

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

When you expand a hierarchy folder, you reveal the Nets, Leaf Cells, and hierarchies at that level. The icons next to the cells show information about the state of the design.

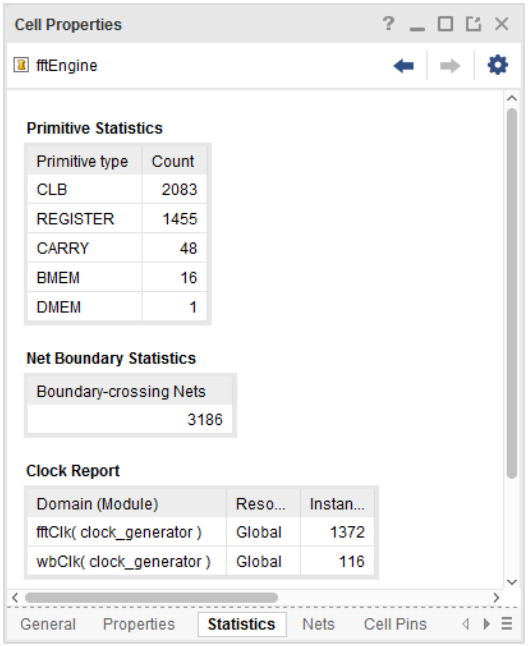
For more information, see Using the Netlist Window in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) and KEEP_HIERARCHY property and `-flatten_hierarchy` command switch in the *Vivado Design Suite User Guide: Synthesis* (UG901).

Cell Properties

Use the Cell Properties window for the selected hierarchy to view detailed information. Filter the information using the category buttons at the bottom of the window. When you select the Statistics button, you see the following:

- Primitive usage for the full hierarchical branch, grouped in higher-level buckets
- Number of nets crossing the hierarchy boundary
- Clock details, including whether the clock uses 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 appear for the Pblock.

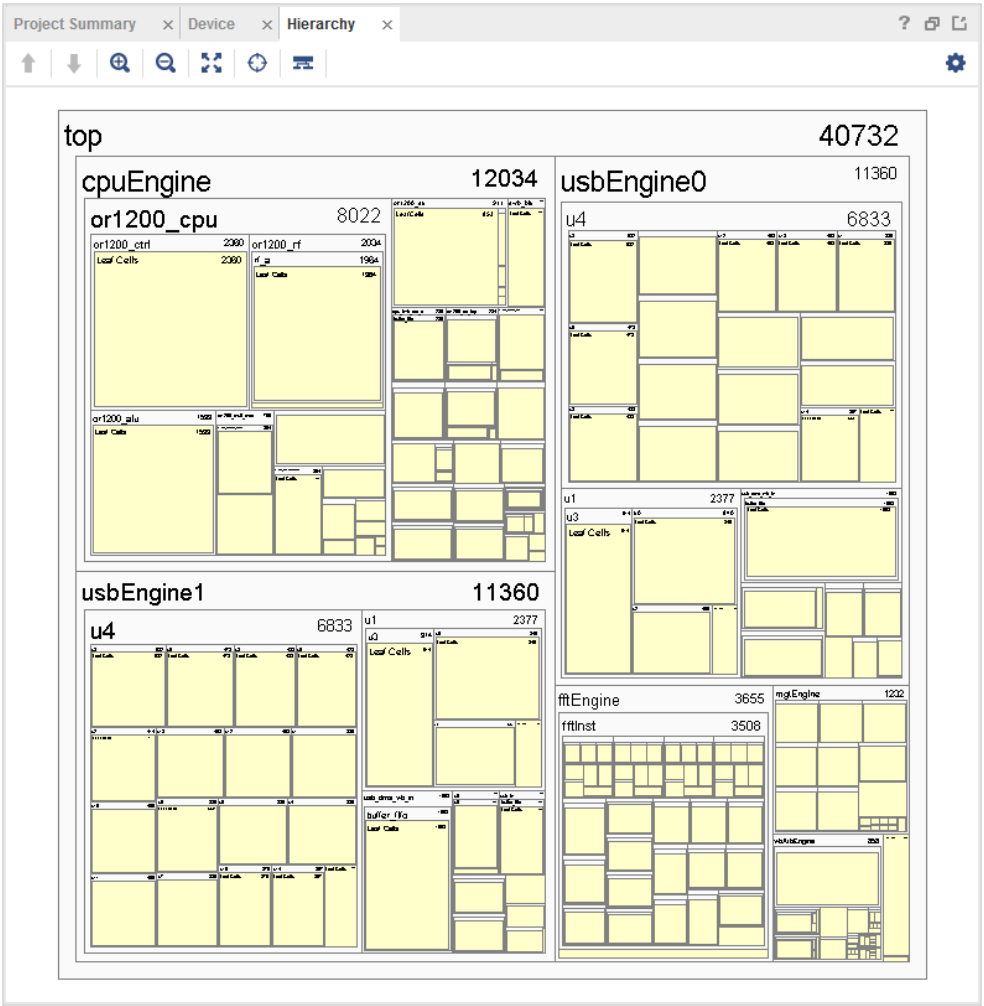
Using the Hierarchy Window

Explore the hierarchy physically to understand resource usage. The Hierarchy window displays a hierarchy map for the selected hierarchy. The map shows leaf cells as yellow blocks nested within rectangles representing their parent hierarchy. Each level is sized relative to the flat number of instances at that level compared to the total number of instances in the design, or when a hierarchy is selected in the netlist window, the total number of instances below the selected hierarchy.

Use the following steps to view the hierarchy map.

1. Select **Tools → Show Hierarchy**, or
2. From the Netlist window, press **F6**.

Figure 4: **Hierarchy Window**

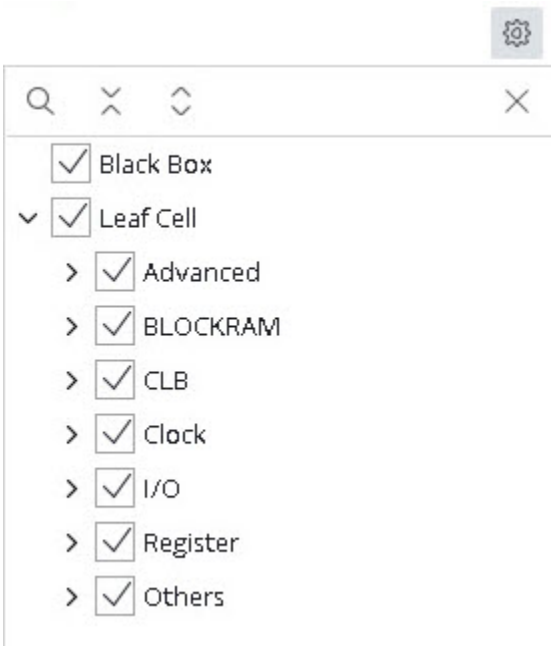


The figure shows that cpuEngine, usbEngine0, and usbEngine1 contain most of the logic in the design and use roughly the same amount of resources.

With default settings, all primitives are considered. To see the size of a specific resource, do the following:

1. Click the cog
2. Open the settings
3. Select the resources you want to view

Figure 5: **Hierarchy View Primitive Type Selection**

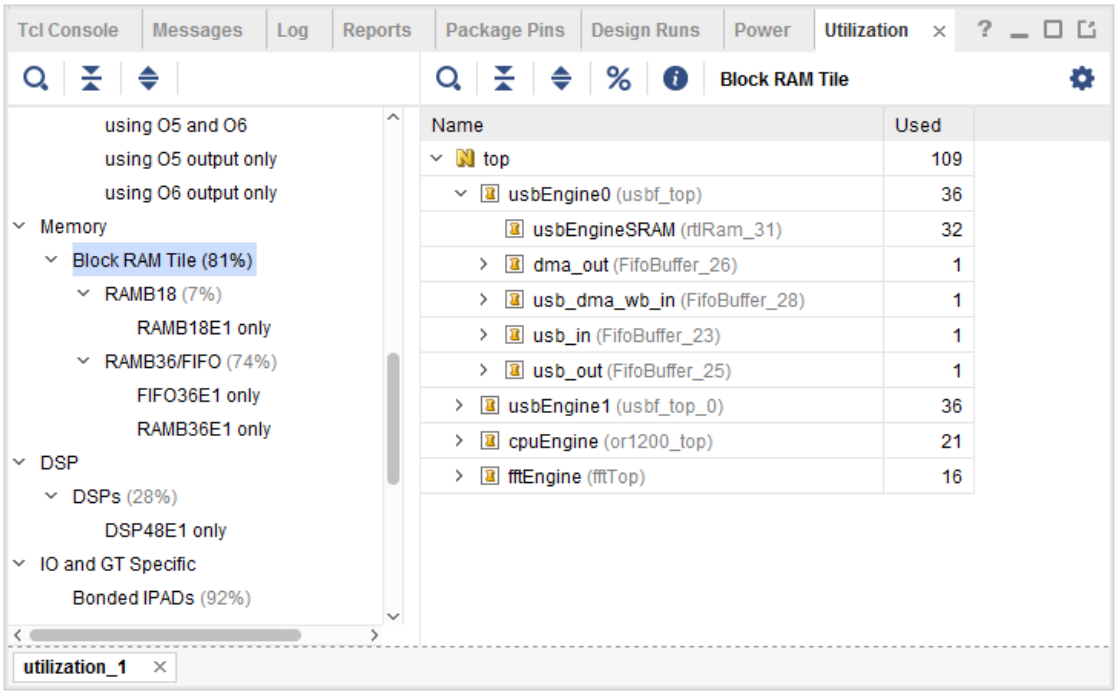


Using the Utilization Report

The Utilization Report breaks down design utilization by resource type. The left panel summarizes usage by resource type, while the right panel shows usage per hierarchy.

To view the Utilization Report, select **Reports → Report Utilization**.

Figure 6: **Utilization Report**



In this design, the two usbEngine blocks are the largest consumers of the RAMB36 and FIFO36 blocks. Click the arrows next to each block in the Utilization Report to view consumption at the sub-hierarchy level.

Using the Schematic Window

The schematic is a graphical representation of the netlist that includes the following:

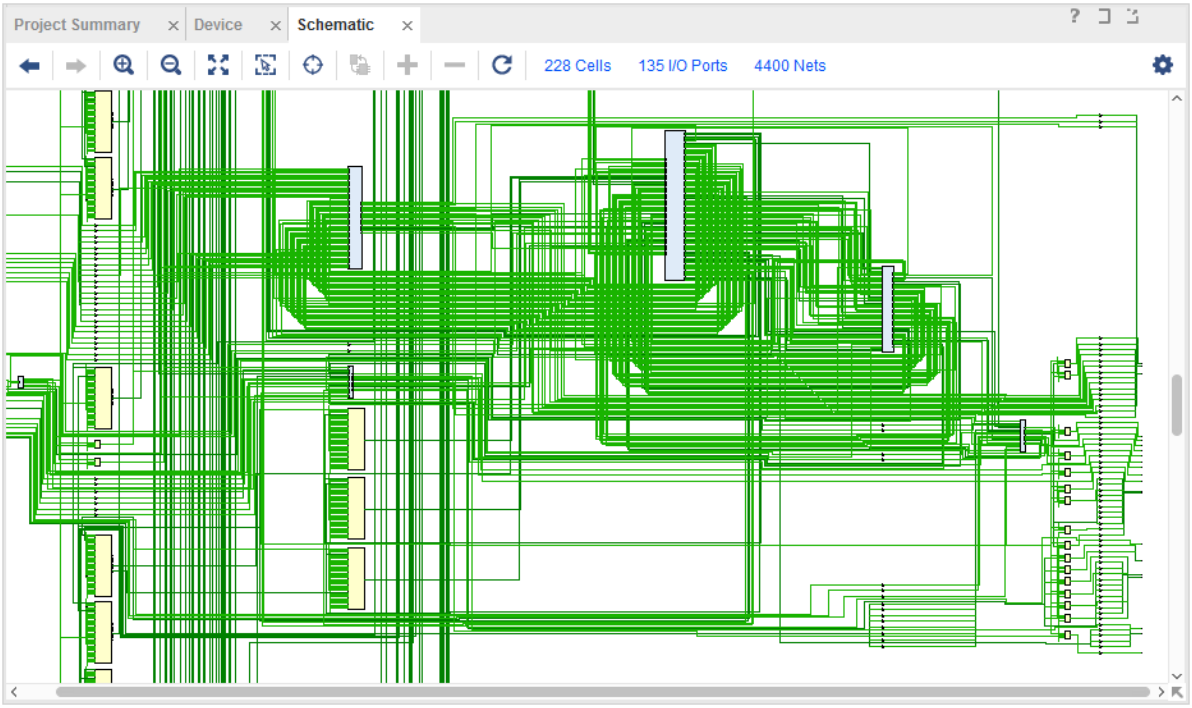
- View a graphical representation for the netlist.
- Review gates, hierarchies, and connectivity.
- Trace and expand cones of logic.
- Analyze the design.
- Better understand what is happening inside the design.

At the RTL level in Elaborated Design, the schematic shows how the tool has interpreted your code. In Synthesize Design and Implemented Design, it displays the netlist generated by the synthesis tool.

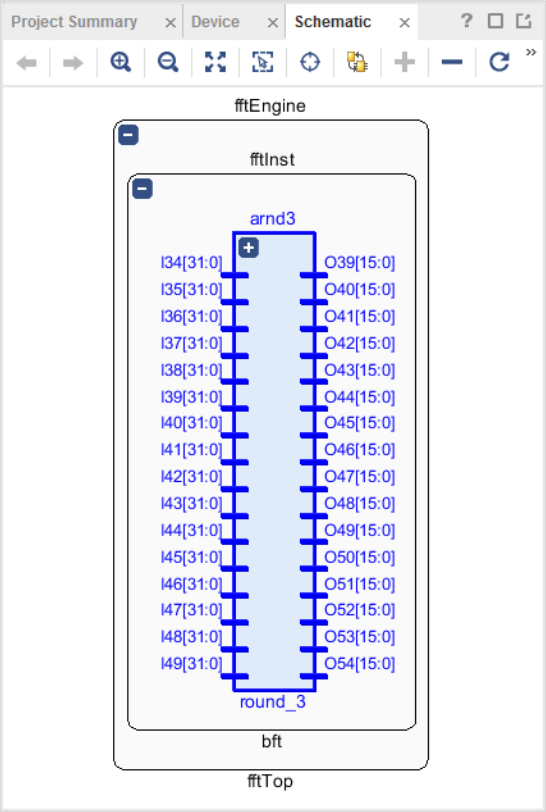
Use the following steps to open and interact with the schematic:

1. Select **Tools → Schematic**.

2. If nothing is selected, the schematic displays the cells, hierarchy, and connectivity at the top level of the design.



TIP: *You can create a simpler schematic by first selecting cell, net, pin, or port objects, then generating the schematic from those selected items.*

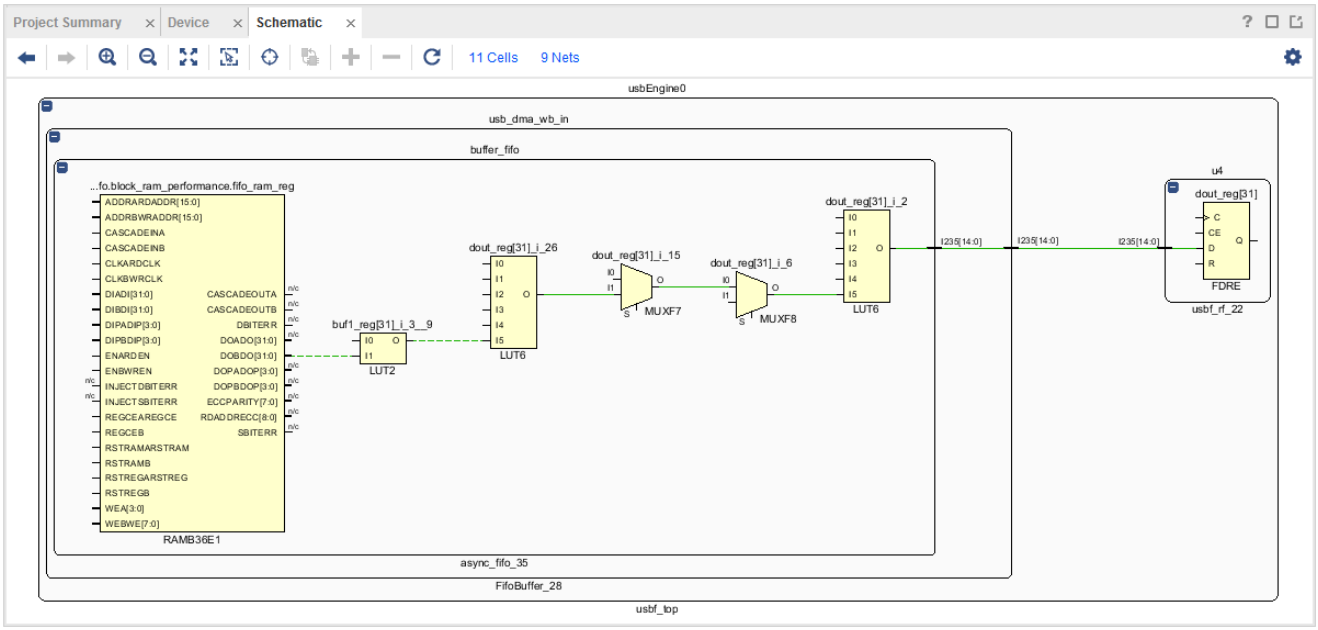


3. Use the following methods to navigate the schematic:
   - Click **+** to expand the gates in the hierarchy.
   - Double-click ports or cells to trace connectivity.
   - Right-click and select **Schematic** from the popup menu.
   - Use ← and → to switch views.

After implementation, the schematic is the easiest way to visualize the cells in a timing path. Select the path, then open the schematic to view the related cells and nets.

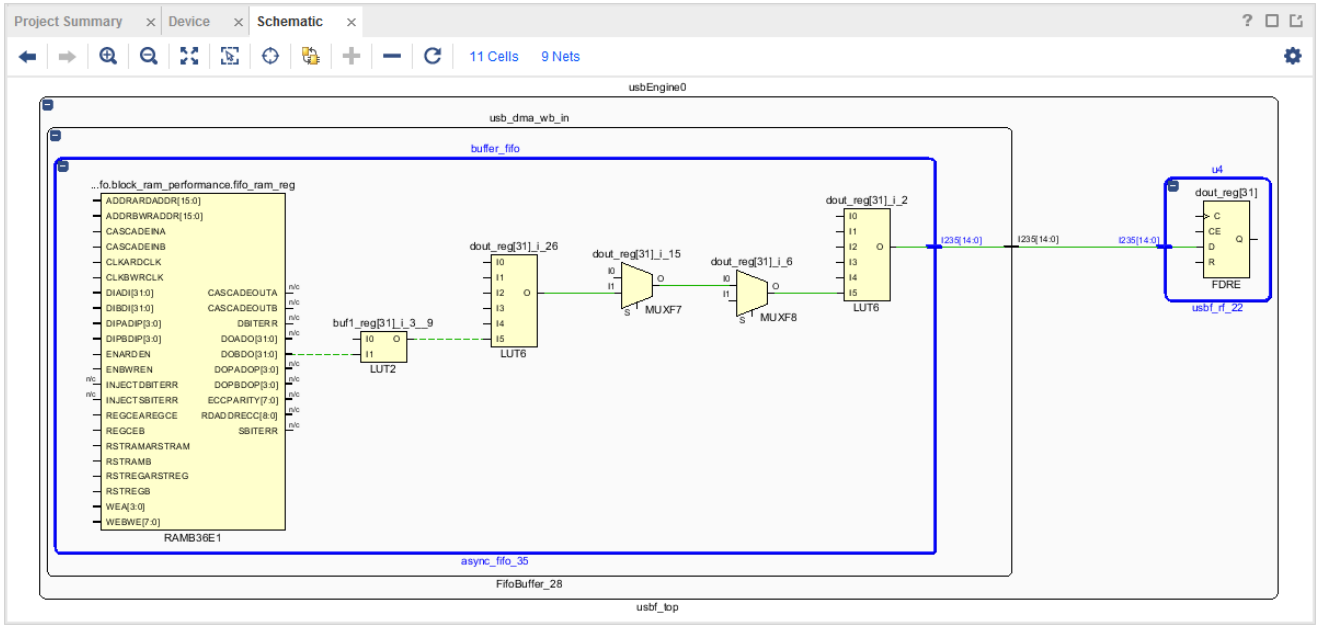
For more details, see *Using the Schematic Window* in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Figure 7: **Schematic with Timing Path**



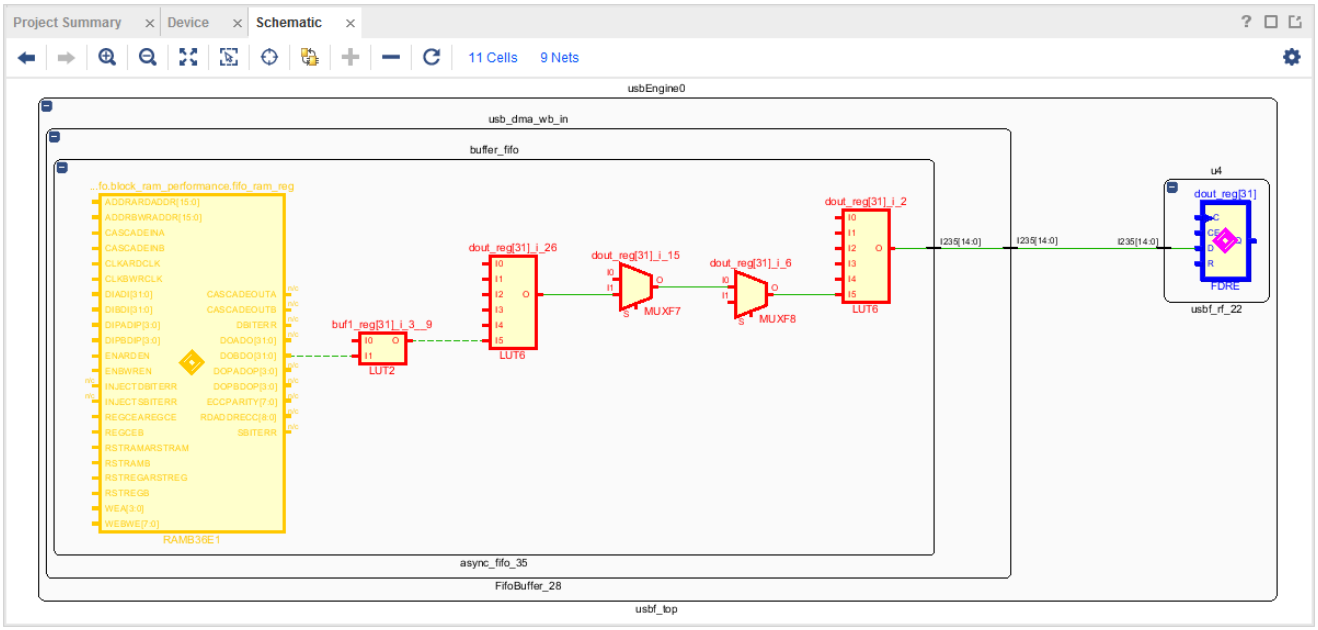
To analyze a selected cell in the schematic, follow these steps:

1. Right-click the cell and select **Select Leaf Cell Parents** from the popup menu to identify the relevant levels of hierarchy.



2. Use the **Highlight** or **Mark** commands to track leaf cells of interest.
3. Apply color coding to the selected cells to clearly distinguish between logic from the original path and added logic.

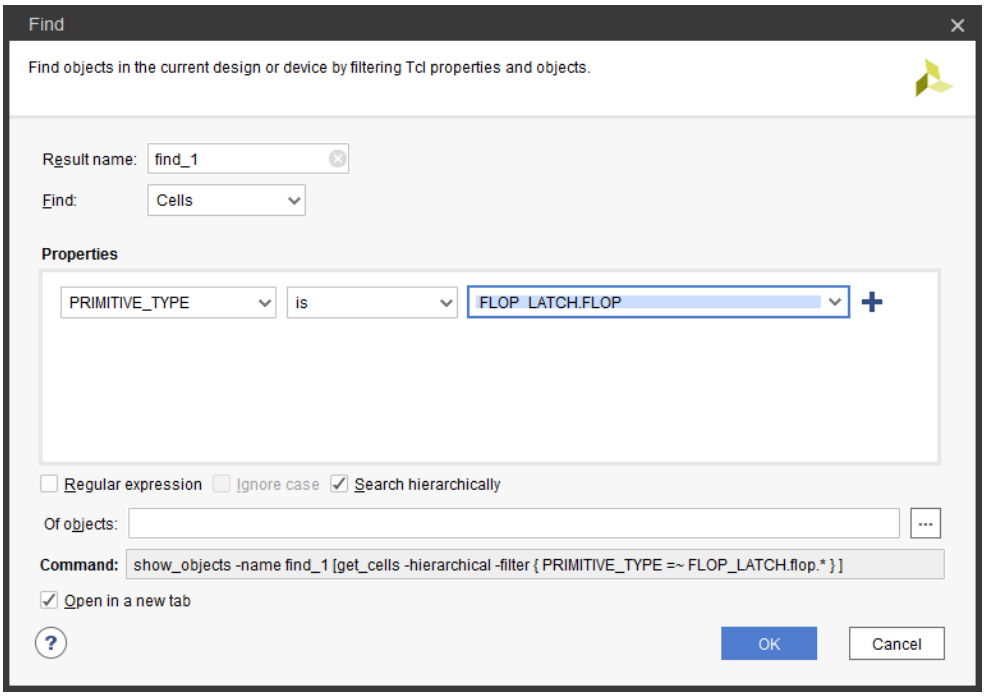
Figure 8: **Schematic with Timing Path Marked**



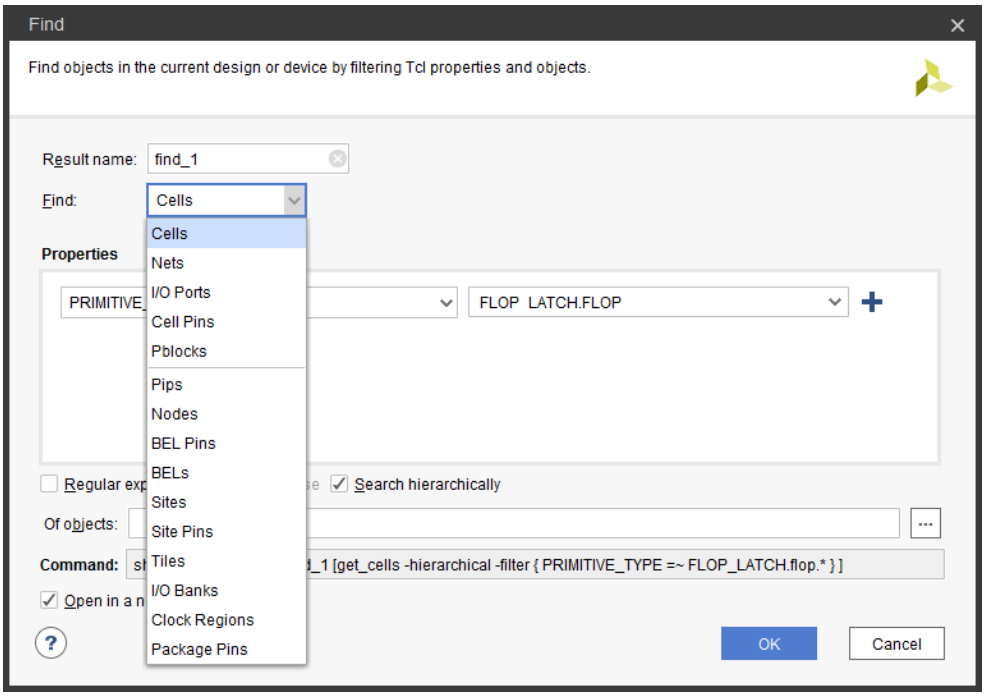
Searching for Objects Using the Find Dialog Box

The AMD Vivado™ IDE includes powerful find and search capabilities. Use the following steps to search for objects using the Find dialog box:

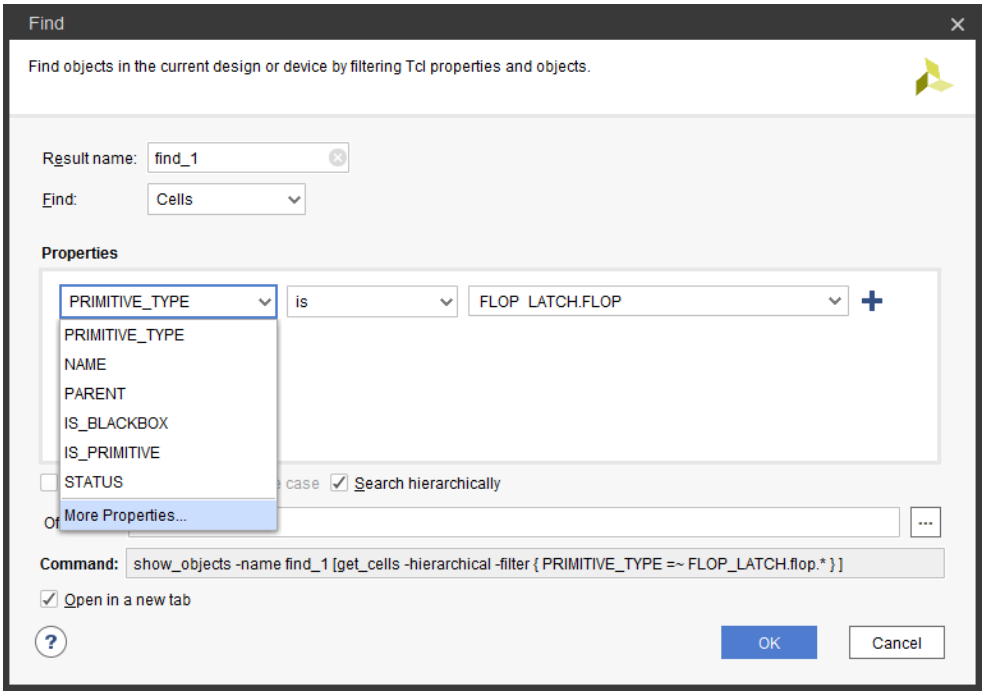
1. Select **Edit → Find** or press **Ctrl+F**.



2. Click the drop-down arrow next to the **Find** field to select the object type to search.



3. Set properties or filters using the drop-down menus.



   a. (Optional) Click the + icon to add more criteria.

   b. (Optional) Combine criteria with logical operators such as AND and OR.

Examples

Find All Unplaced I/Os

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

Find All Nets with a Fanout Over 10,000

- **Find**: Nets
- **Properties**: FLAT_PIN_COUNT > 10000

All DSPs Using the PREG Embedded Register

- **Find**: Cells
- **Properties**: PRIMITIVE_TYPE is ARITHMETIC.DSP + AND PREG > 0

Tcl Finds

From a script or the 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 displays the Tcl command options the Vivado Design Suite runs for each GUI action. You can also enter your own Vivado Tcl command options directly in the Tcl Console,*

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

For more information on Tcl command options, type `<command> -help` in the Tcl Console. See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for more information.

Methodology Analysis

Because methodology is critical, the AMD Vivado™ tools provide the `report_methodology` command to check for compliance with methodology design rule checks (DRCs). These checks vary by design stage:

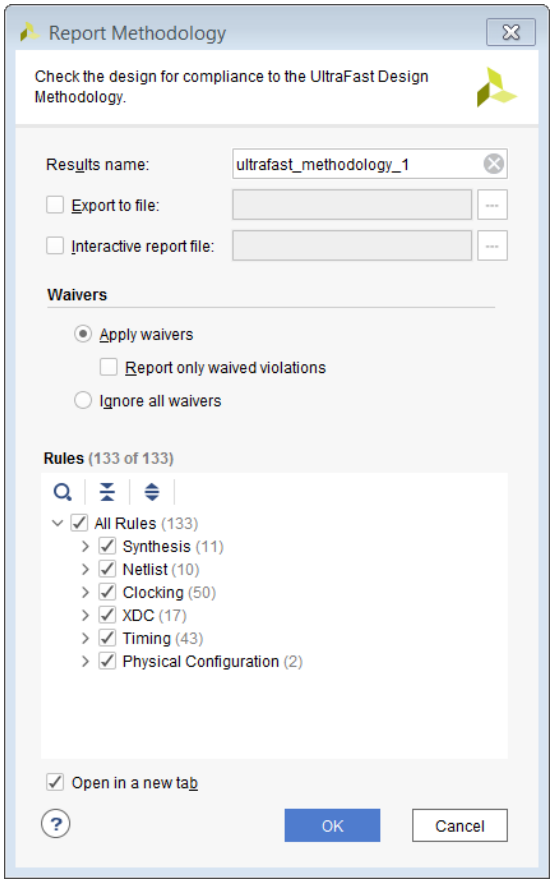
- RTL lint-style checks run on the elaborated RTL design.
- Netlist-based logic and constraint checks run on the synthesized design.
- Implementation and timing checks run on the implemented design.

You can run methodology checks using either Vivado IDE or Tcl.

1. Open the design you want to validate.
2. In the Vivado IDE Flow Navigator, click **Report Methodology**. Select **Reports → Report Methodology**.
3. Or, in Tcl use the following command:

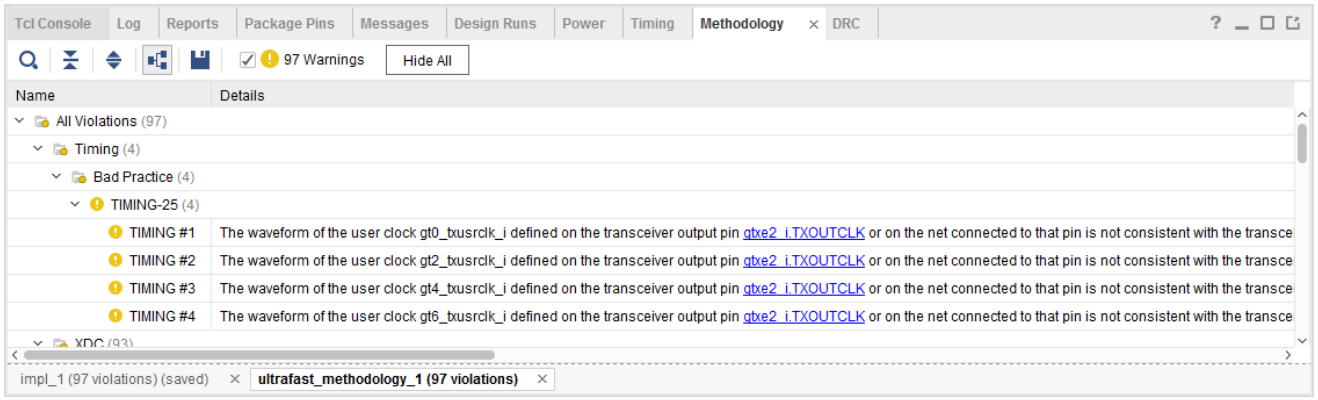
```
report_methodology
```

Figure 9: **Report Methodology Dialog Box**



Violations, if present, appear in the Methodology window, as shown in the following figure.

Figure 10: **Methodology Violations**



For details on running design methodology DRCs, refer to Running Methodology Checks in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895).

***Note*:** Address all methodology violations, with special attention to Critical Warnings, as they impact both timing closure and sign-off quality.

Placement Analysis

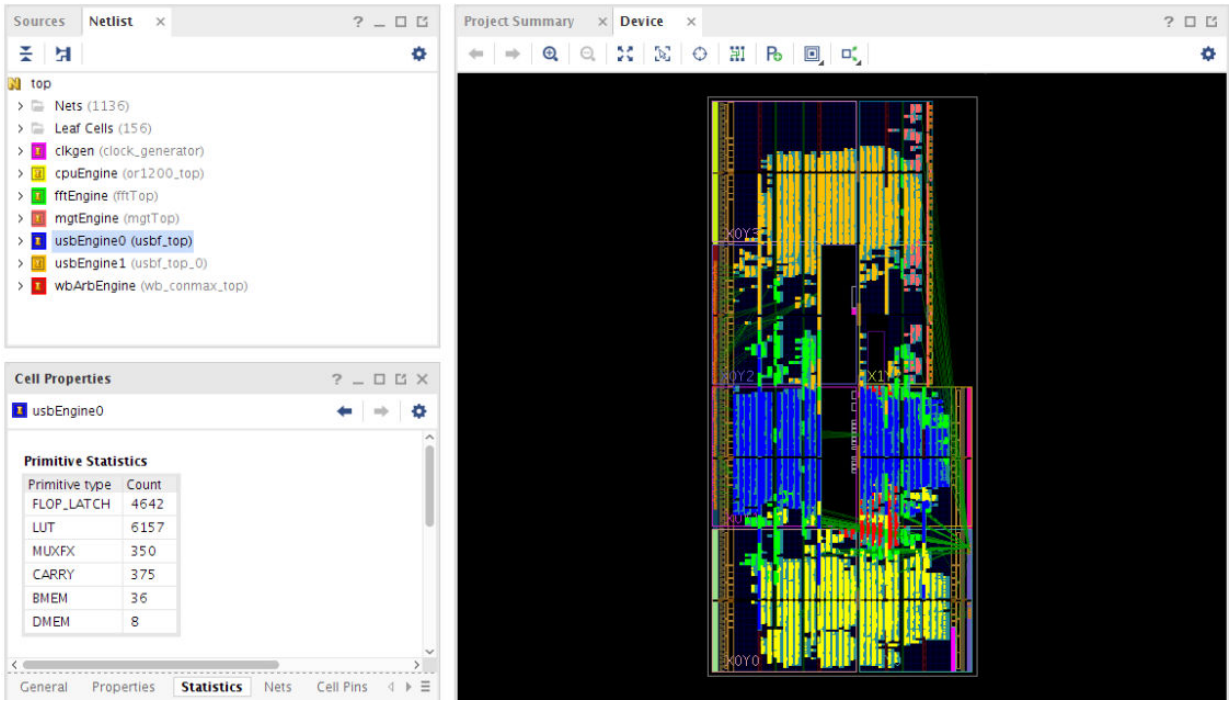
Highlighting Placement

Another way to review design placement is to analyze cell placement. The Highlight 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 11: **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

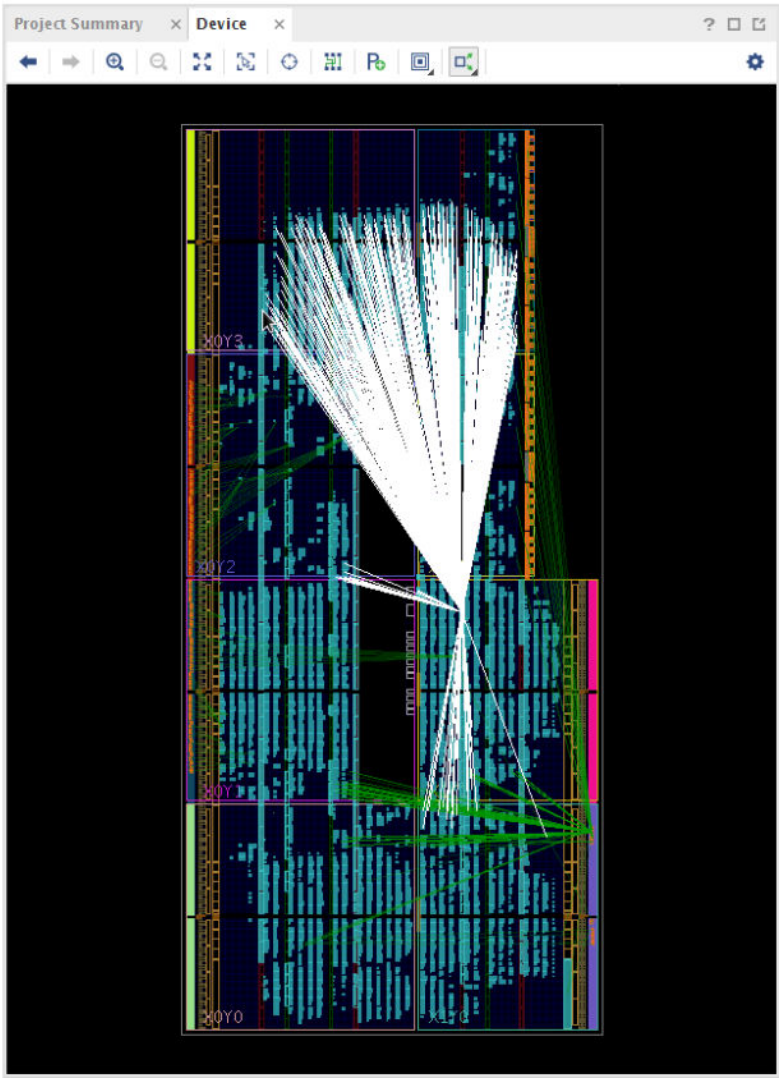
It can be useful to analyze a design based on connectivity. Use Show Connectivity to review the placement of all logic driven by a selected cell or net.

1. Select one or more cells or nets.
2. Run Show Connectivity to expand the selection by including connected cells or nets.

TIP: *Use this method to build and view cones of logic inside the design.*

The following figure shows a block RAM driving logic inside the device including OBUFs. A synthesis pragma prevents synthesis from placing the output flop in the block RAM during memory inferencing.

Figure 12: **Show Connectivity**



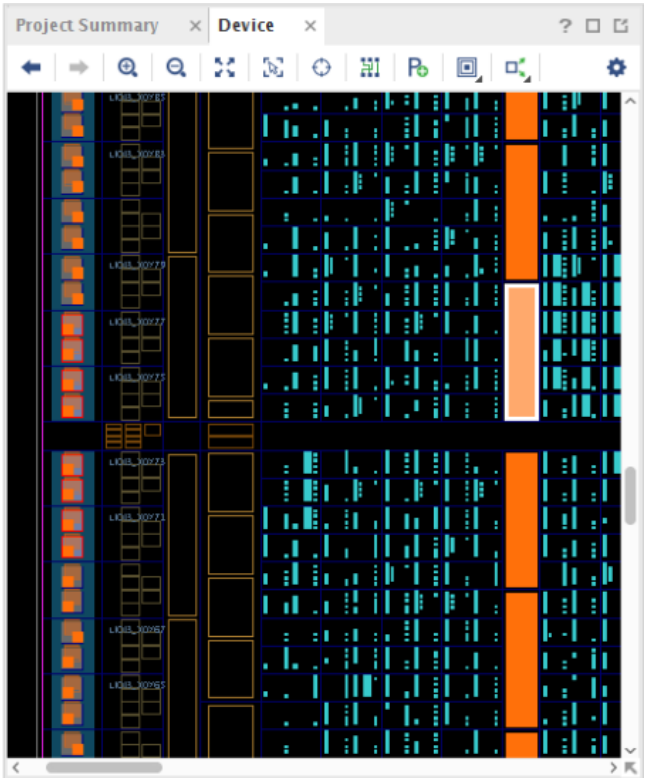
Fixed and Unfixed Logic

The Vivado tools track two different types of placement:

- Fixed elements (shown in orange) are placed by you.
  - Vivado stores fixed logic in the XDC.
  - Fixed logic usually has a LOC constraint and can include a BEL constraint.
- Unfixed elements (shown in blue) placed by the tool.

In the following figure, the I/O and block RAM placement is fixed. The slice logic is unfixed.

Figure 13: **Fixed and Unfixed Placement**



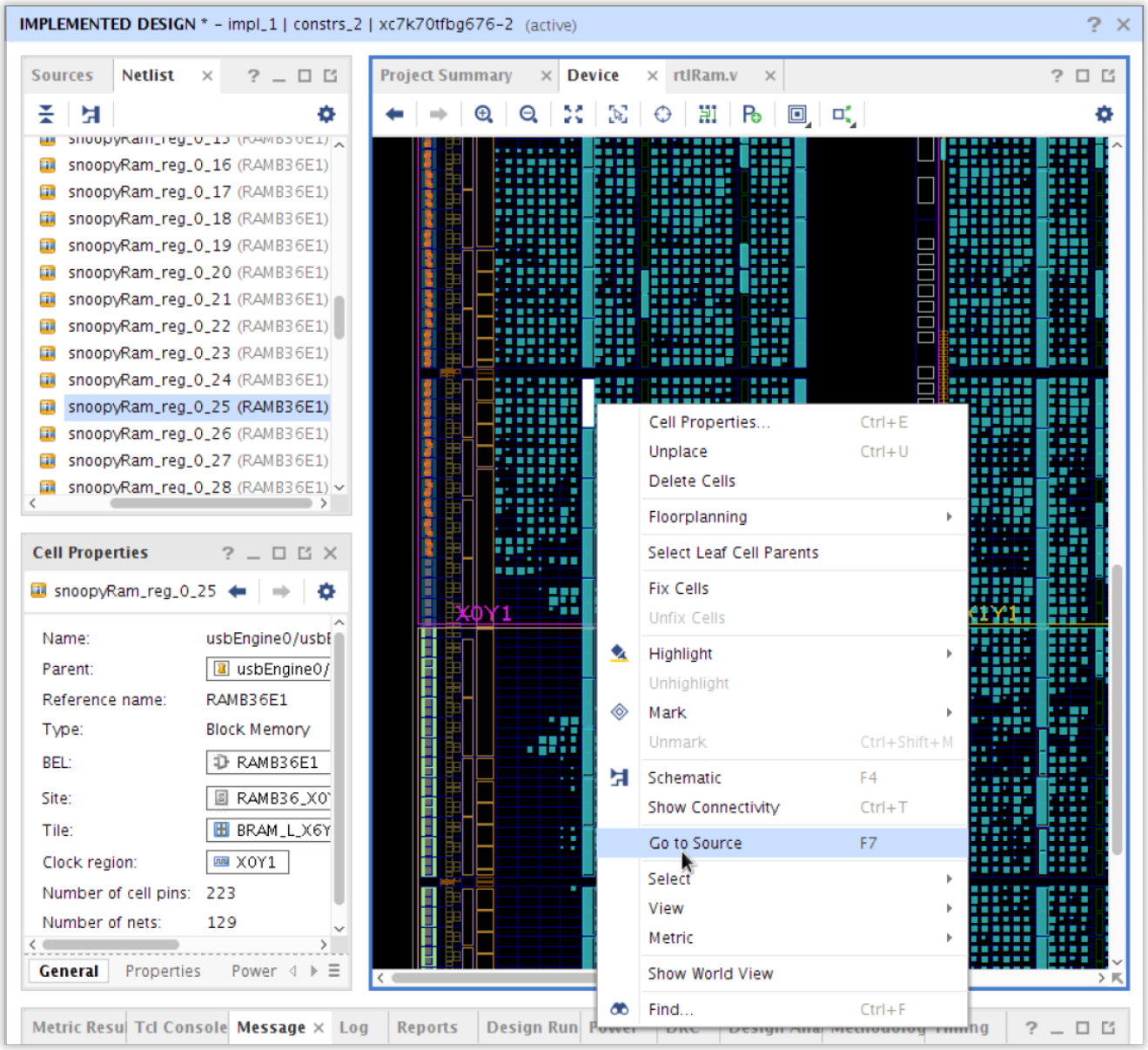
Cross-Probing to RTL Source

You can cross-probe to the source files after the netlist design is loaded into memory for designs synthesized with Vivado synthesis.

Use the following instructions to cross-probe:

1. Select the object (for example cell, hierarchical cell, net, or port).
2. Select **Go to Source** from the popup menu, shown in the following figure.

Figure 14: **Cross-Probe Back To Source**



Use cross-probing to identify which source corresponds to specific gates in the netlist. Because of synthesis transformations, you cannot cross-probe every gate back to its exact source in the design.

Viewing Metrics

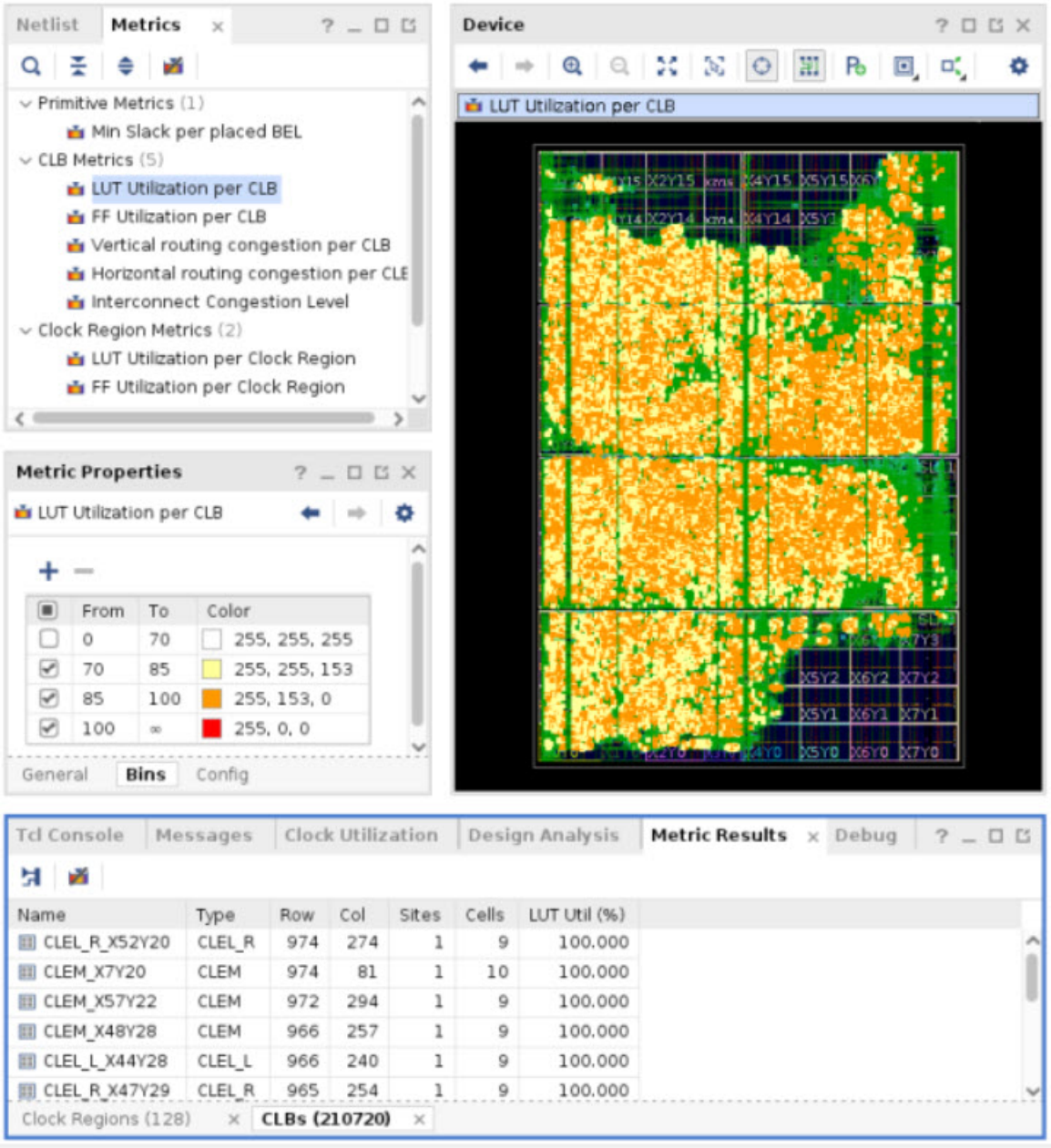
The Vivado Design Suite provides metrics to inform you about logic and routing inside the device. They provide an interactive way of analyzing a design rather than a static report.

Use the following instructions to enable and configure metrics in the Device window:

1. Make sure a design is open.
2. Select **Window → Metrics**.
3. To enable a metric, right-click and select **Show**. More than one metric can be added at a time.
4. To disable a metric, right-click it and select **Hide** to remove it from the Device view.
5. Select **Hide All Metrics** to clear all selected metrics at the same time.

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

Figure 15: **LUT Utilization per CLB Metric**



- **Metrics Window:** Lists available metrics that you can display using the right-click menu.
- **Metric Properties Window:** Shows the color bins used to grade the selected metric. You can enable or disable bins, change their colors, add new bins, or remove existing ones.
- **Device Window:** Overlays the metric bins showing the physical location where the metric levels have a higher impact.
- **Metric Results Window:** Enables you to select results and cross-probe them with other design elements.

Metrics in a Netlist Design with No Placement

Use the following metrics when Pblocks exist. These metrics do not depend on placement:

- **LUT Utilization per Pblock:** This metric color codes the Pblock based on an estimate of how the LUTs pack into the slices within the Pblock.
- **FF Utilization per Pblock:** This metric color codes the Pblock based on an estimate of how the FFs pack into the slices within the Pblock.

Metrics Requiring a Placed Design

Use the following metrics when the design is placed. These metrics do not require full routing to provide accurate results:

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

  ***Note*:** This metric is for AMD UltraScale+™ and newer architectures.

Timing Metrics

Use timing metrics to view a physical representation of your design's timing issues. Each BEL is color coded based on the worst-case negative slack (WNS) value on the timing path through that BEL.

Utilization Metrics

Use utilization metrics to view the percentage of used resources, graded by CLB or clock region usage. These metrics are available for both LUT and FF utilization.

Congestion Metrics

Use different congestion metrics based on the device family you target.

Congestion Metrics for 7 Series and UltraScale Device Families

For 7 series and AMD UltraScale™ parts, use a method similar to the placer's congestion estimation with the following metrics:

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

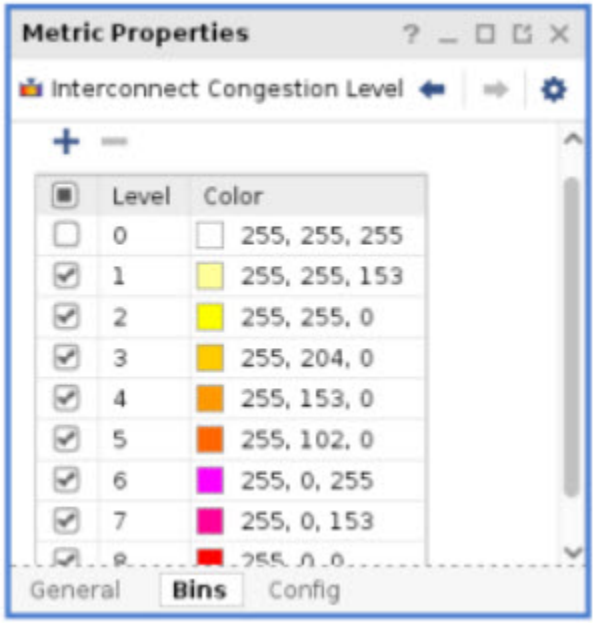
Both metrics use a demand-based model to estimate routing requirements between two points. When demand exceeds 100, the tool uses routing in adjacent tiles. Congestion increases in areas of the die where demand consistently exceeds 100%.

***Note*:** You can use this method for newer device families, but it is not recommended.

Congestion Metrics for UltraScale+ and Newer Device Families

For UltraScale+ and newer device families, use the Interconnect Congestion Level metric. This metric provides a simplified view with improved accuracy and better alignment to how the router estimates congestion. The bin colors reflect the congestion levels.

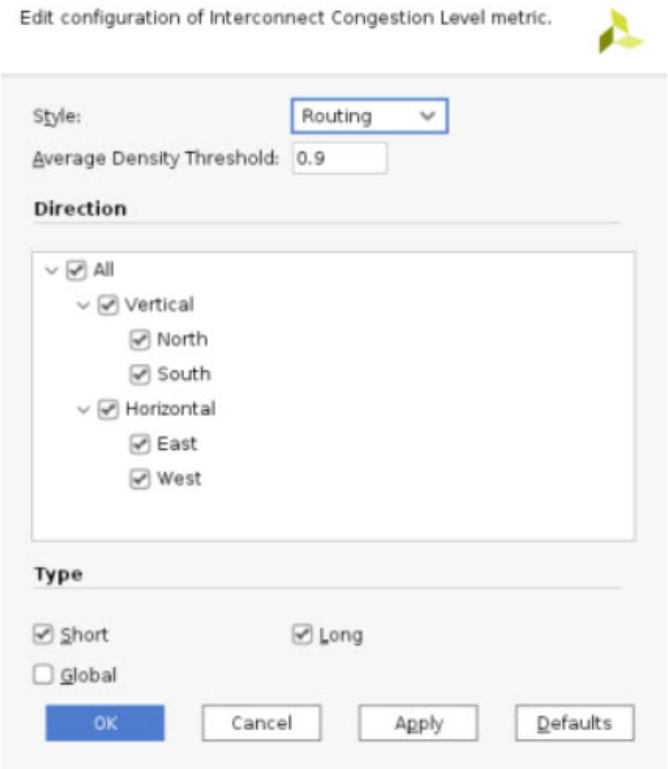
Figure 16: **Interconnect Congestion Level Metric Properties**



In the configuration window, you can configure the metric view to show the following information:

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

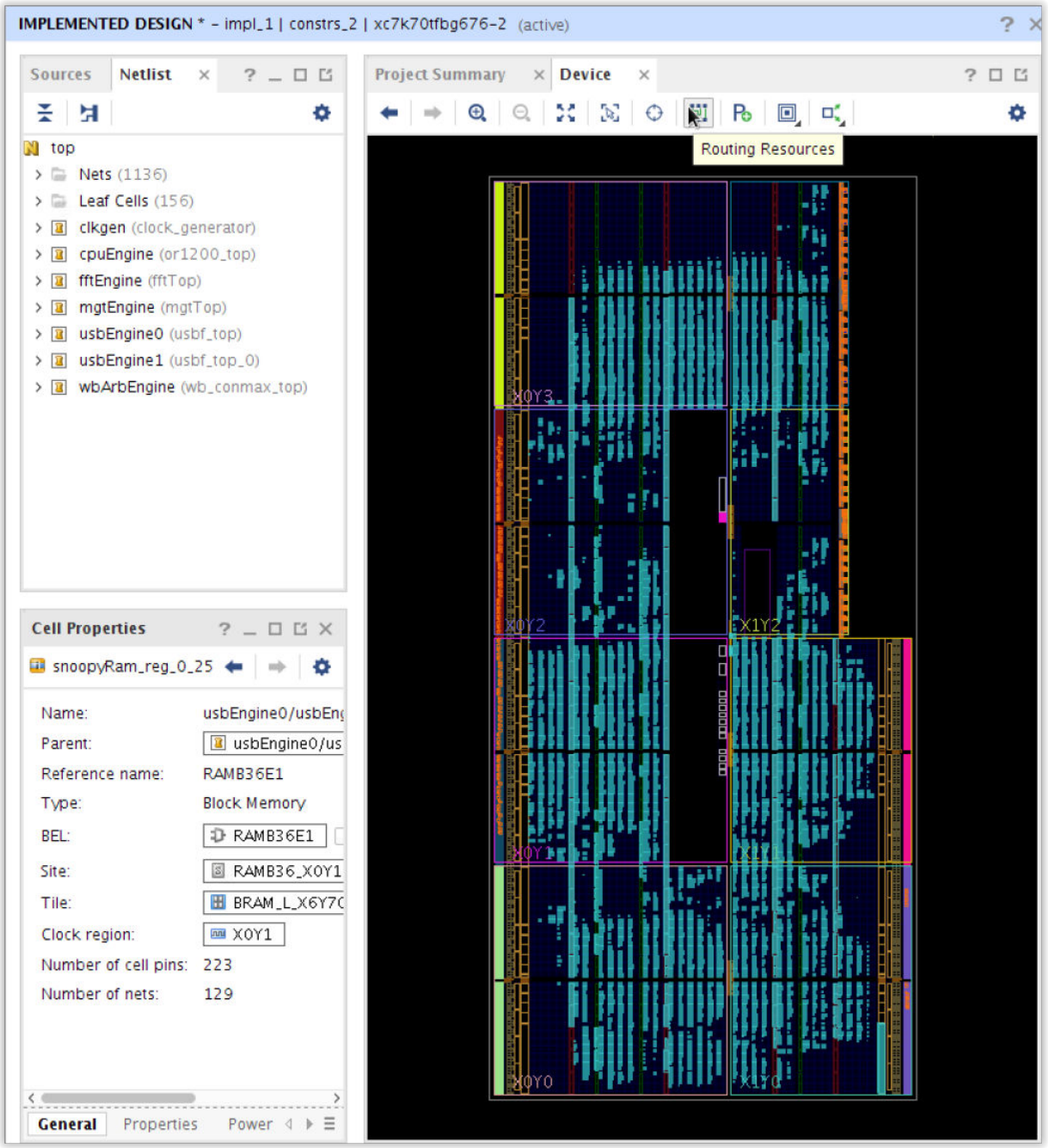
Figure 17: **Interconnect Congestion Level Configuration**



Routing Analysis

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

Figure 18: **Enable Routing**



Displaying Routing and Placement

Routing and placement display different levels of detail based 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.*

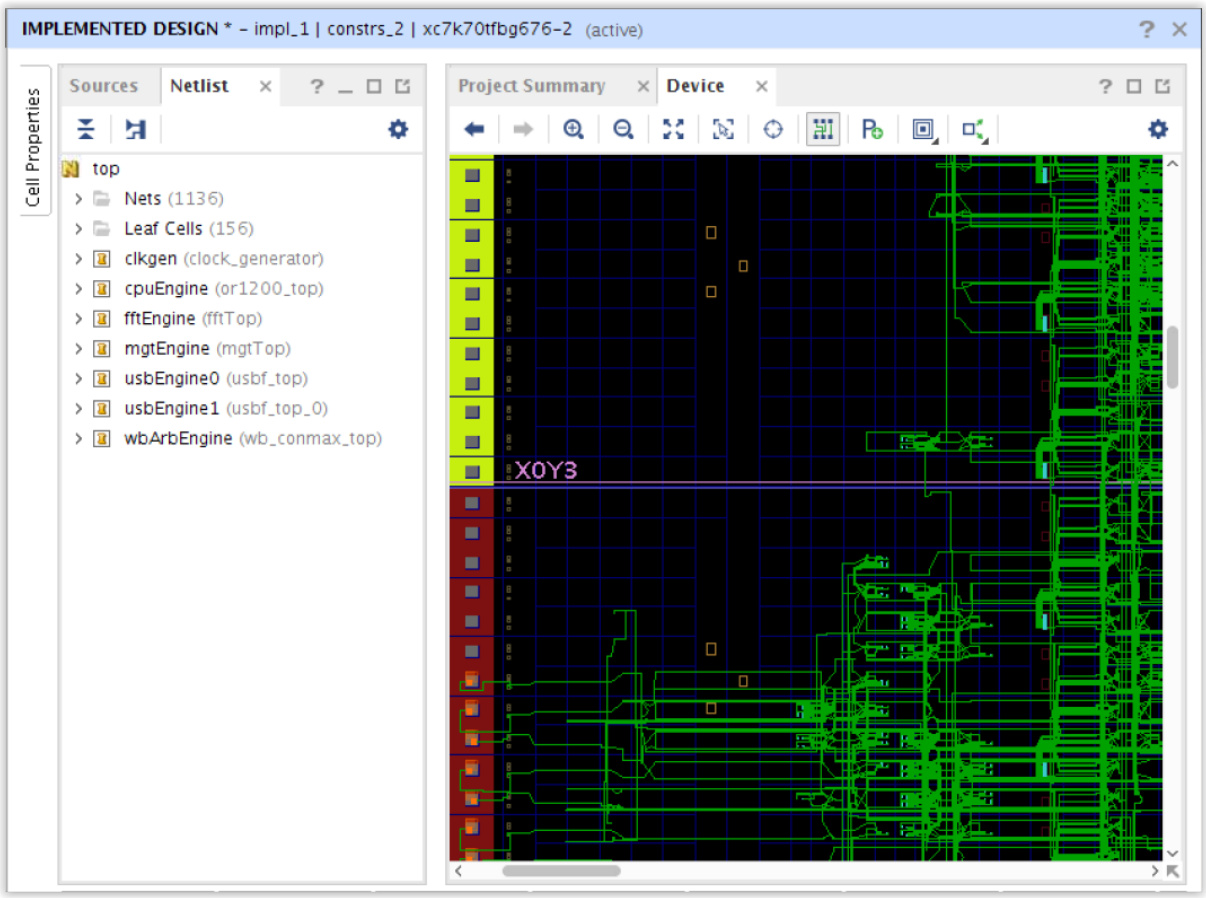
Displaying Routing and Placement when Zoomed Out

When you zoom out, the Device window shows an abstract view. In this view, you can observe the following:

- Routes through the device appear condensed.
- Line thickness varies based on the number of routes passing through a region.

Placement also appears abstract. Each tile with placed logic displays as a block, and the more logic a tile contains, the larger its block appears.

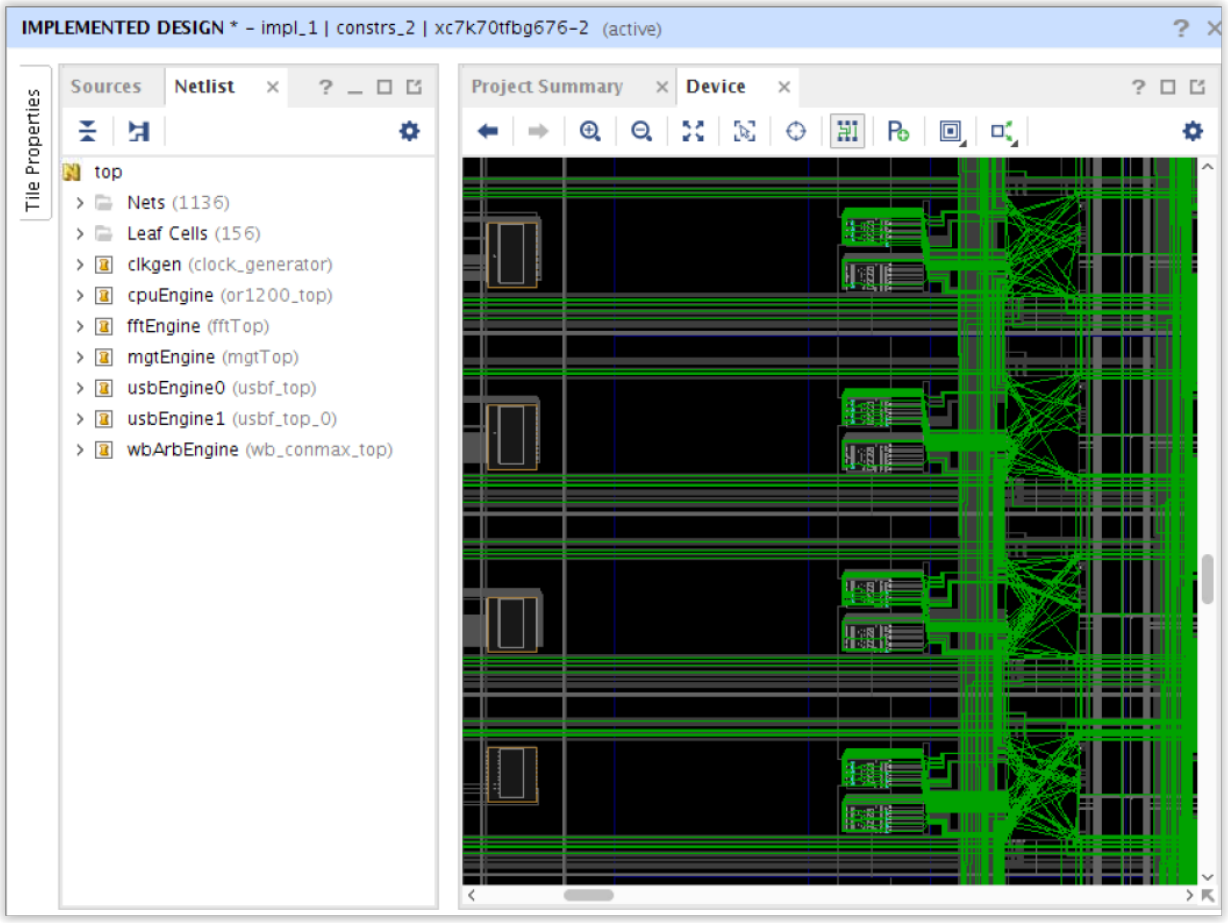
Figure 19: **Abstract View**



Displaying Routing and Placement at Closer Zoom Levels

At closer zoom levels, you see the actual logic cells and routing paths.

Figure 20: **Detailed View**

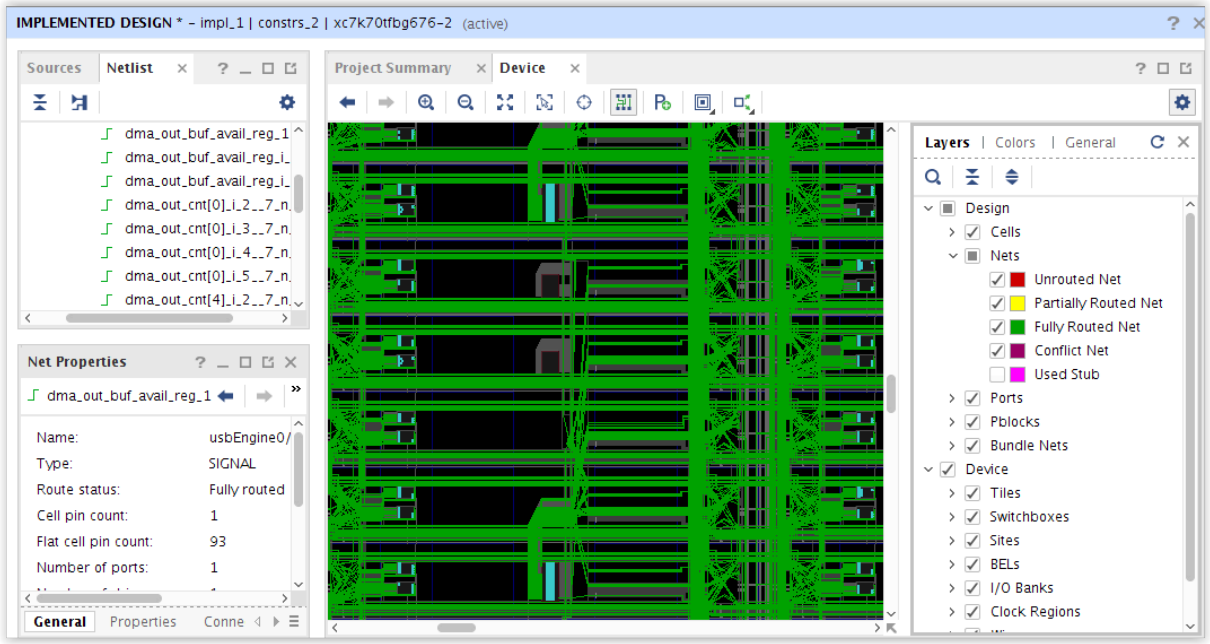


Viewing Options

You can customize the Device window to view the device and design in different ways, and use the Device Options slideout to control the following settings:

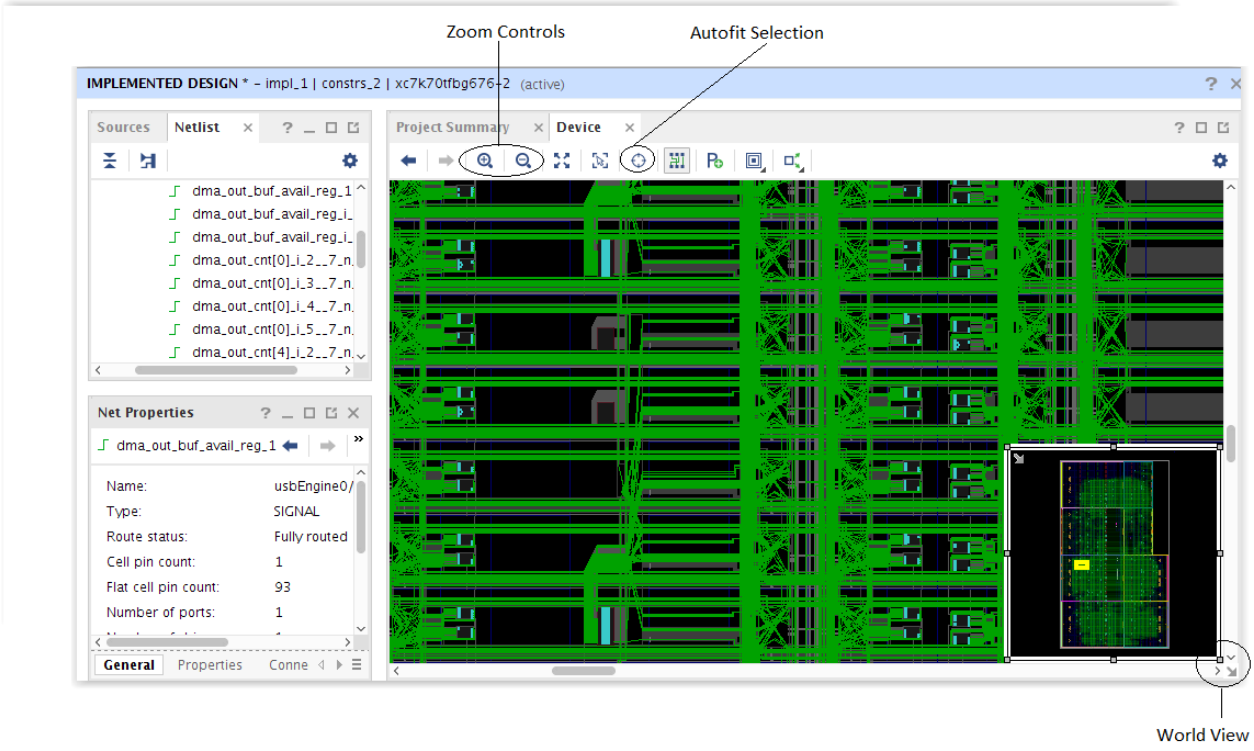
- Enable or disable graphics for various design and device resources
- Modify the display colors

Figure 21: **Device Window Layers**



Navigating in the Device Window

Figure 22: **Navigating the Device Window**



Use the following instructions to navigate the Device window:

1. Use the zoom controls to zoom in, out, or fit to full view.
2. Use auto-fit selection to focus on selected objects.

   Note: AutofitSelection is particularly useful for cross-probing.
3. Use the World View to view the overall device location. You can move and resize the World View, as well as drag and resize the yellow box to zoom and pan.
4. Press **Ctrl** while doing one of the following:
   - Click and drag to pan the view.
   - Use the mouse wheel to zoom in and out at the position of the cursor.

Dataflow Analysis

High-bandwidth buses can affect routing congestion, timing, and power because of their width. Use dataflow analysis to inspect how data moves through these buses. This analysis simplifies the netlist to make navigation easier while preserving key placement anchor points.

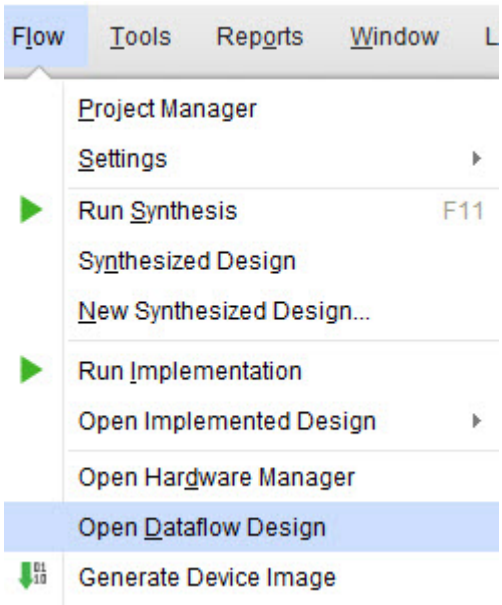
When you create a dataflow design, the following applies:

1. Vivado generates a simplified netlist from the full design netlist.
2. The dataflow viewer displays this stripped-down netlist.
3. The dataflow netlist keeps bus nets above a user-defined width.
4. It removes bus nets below this threshold, scalar nets, and lower-level cells.

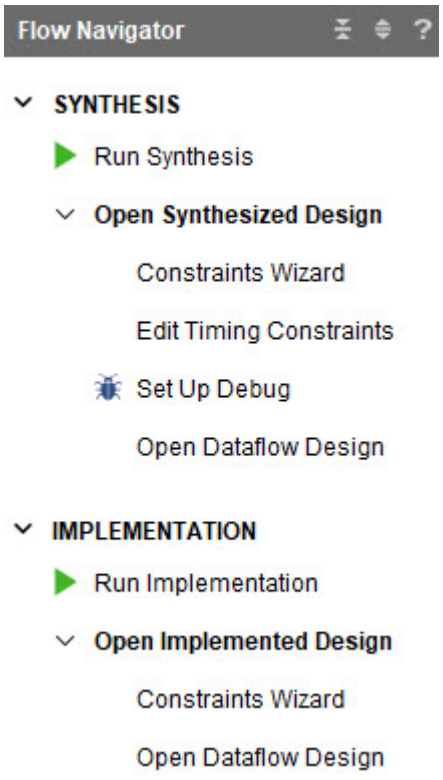
Creating a Dataflow Design

Use the following instructions to create a dataflow design.

1. Open a synthesized or implemented design.
2. Open the dataflow design by using one of the following options:
   - In the interface, click **Flow → Open Dataflow Design**



   - In the Flow Navigator, select **Open Dataflow Design** under either **Open Synthesized Design** or **Open Implemented Design**. This option carries the flow forward as needed. The tool generates the netlist based on the active run.



- From the Tcl interface, use the command `create_dataflow_design`.

Make sure a design is open before running this command. When a design is placed or routed, cell locations that remain in the dataflow netlist are copied and shown in the device view.

***Note*:** Location changes made after netlist creation do not update automatically. To reflect those changes, regenerate the dataflow netlist.

You can simplify the netlist further by increasing the value of the `-min_bus_width` option. The default is 16. Setting it to a higher value removes smaller buses and produces a more simplified netlist. The following is an example:

```
create_dataflow_design -min_bus_width 128
```

Dataflow Netlist Generation

The dataflow netlist is a subset of your original design netlist. When you generate the dataflow netlist, Vivado trims out the following:

- Scalar nets
- Bus nets below the `-min_bus_width` threshold (default:16)
- Registers
- LUTs that do not significantly influence the datapath

The dataflow netlist includes the following:

- Bus nets equal to or above the `-min_bus_width` threshold
- Significant cells that impact placement include the following items:
  - BlockRAMs
  - UltraRAMs
  - LUTRAMs
  - DSP slices
  - Hard IP
  - Gigabit transceivers
  - NoC
- Hierarchical cells
- Clock nets

Navigating the Dataflow Netlist

Use the schematic function as the primary way to navigate a dataflow design. You can explore it using the same techniques as in a parent design, including the following methods:

- Select an object, right-click, and select **Schematic**, or press **F4**, to open a new canvas with the selected object.
- Drag objects from the Netlist window and drop them onto an existing schematic canvas.
- Double-click a pin object to expand the net to its loads if it is an output pin, or to its source if it is an input pin.
- Right-click a pin object and select **expand cone to...** to expand the net to its loads or sources.
- Double-click a cell to expand all nets connected to it, showing their sources or loads.

To simplify navigation, the following features differ from parent design navigation:

- Vivado removes internal primitives from the schematic view and displays the macro level above. This improves navigation, especially through DSP and LUTRAM cells.
- When you select cascaded cells, such as DSP or RAM cascade chains, Vivado automatically selects all primitives in the chain within the DataFlow Viewer (DFV). This helps you quickly navigate to the chain output and move to the next cell.

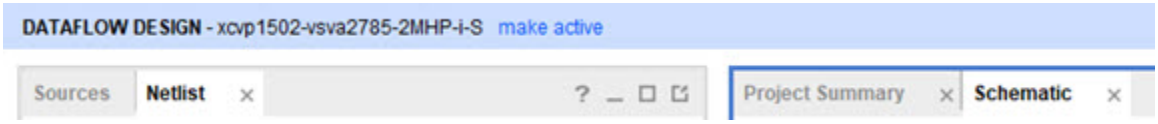
Cross-Probing Between Windows Within the Dataflow Viewer

When you select the Dataflow Design, Vivado sets the dataflow netlist as the active netlist. As a result the following happens:

- The Netlist window shows the dataflow netlist
- Any open schematic windows display the elements of the dataflow netlist
- The Hierarchy window sizing reflects the number of primitives in the dataflow netlist
- Other features, such as **Edit → Find**, work on the dataflow netlist

***Note*:** The blue banner at the top of the Vivado IDE shows the selected design. When the dataflow design is selected, the banner includes DATAFLOW DESIGN, as shown in the following figure.

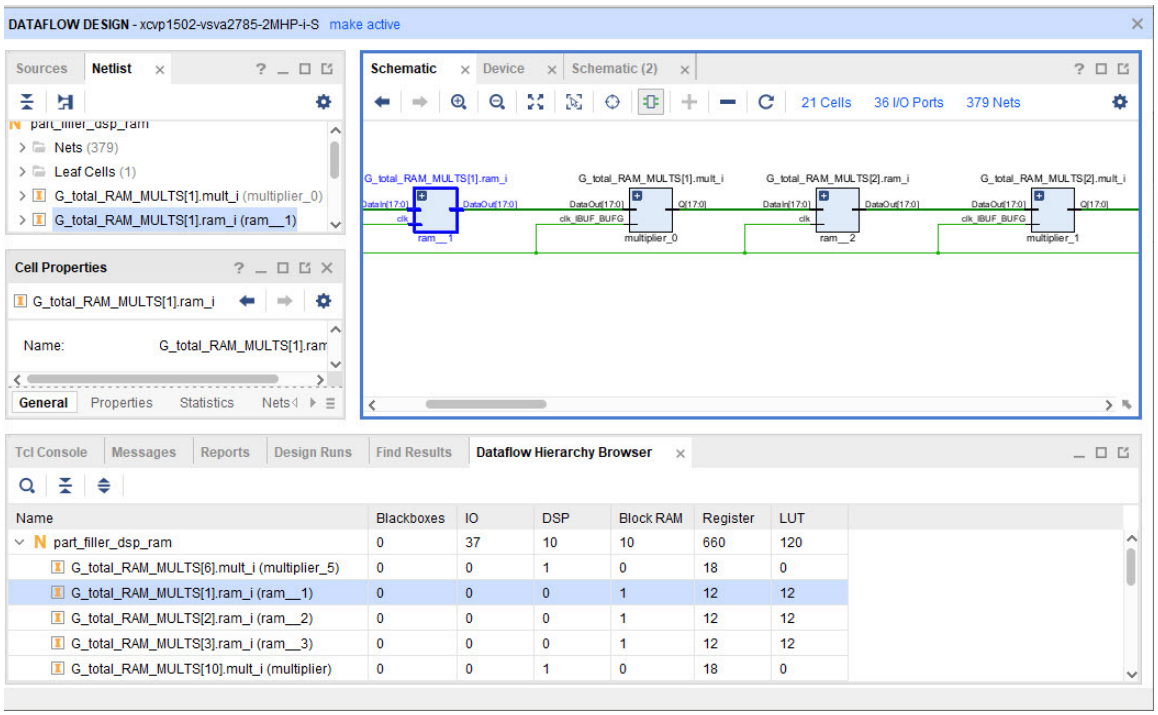
Figure 23: **BLUE BANNER Dataflow Design**



When you select a cell in one window, Vivado selects it in all windows. This cross-probing feature lets you select an object in one context and continue your investigation in another window.

The Dataflow Hierarchy browser instead works differently and shows you resource utilization from the original netlist. This removes the need to cross probe back to the original netlist to find resource utilization on LUTs and registers.

Figure 24: **CROSS PROBING Dataflow Design**



Cross-Probing Between Parent and Child Designs

Cross-probing between the dataflow design and the parent design might be necessary for several reasons:

- Cross-probing the parent netlist to access details that are trimmed from the dataflow netlist, such as:
  - Cells that were removed
  - The number of pipeline stages between key placement cells
  - The placement of cells not available in the dataflow netlist
- Certain analyses are not supported in the dataflow viewer, including:
  - Timing reports
  - Congestion analysis using `report_design_analysis`

Cell cross-probing is the most effective method to do this. Select a cell in one design and switch to the other design. Net correlation between the parent and dataflow designs is typically not possible because net names often change during netlist generation.

When cross-probing from the dataflow design to the parent netlist, all leaf cells in the dataflow viewer exist in the parent netlist. Because the dataflow netlist is a subset of the parent netlist, this direction of cross probing is usually successful.

Avoid cross-probing from the parent netlist to dataflow design using leaf cells like registers or LUTs, as they aren't present in the dataflow netlist. Instead, use hierarchical cells or leaf cells such as BlockRAMs UltraRAMs, DSPs or other advanced primitives.

Use the following instructions to switch between designs:

1. Use the Design Switch icon at the top of the screen.
2. Click the icon to switch quickly between the parent design and the last used dataflow design.
   - The icon becomes available only after you create the dataflow design.
   - The icon changes depending on the switch direction:
     -  denotes switching to the DFV
     -  denotes switching from the DFV

Dataflow Paths

Use the `get_dataflow_paths` command to return a list of paths from the dataflow netlist that include two or more cells connected together. This command helps you target your analysis on specific cells and their connecting logic.

Use the following instructions to display dataflow paths in the Vivado IDE:

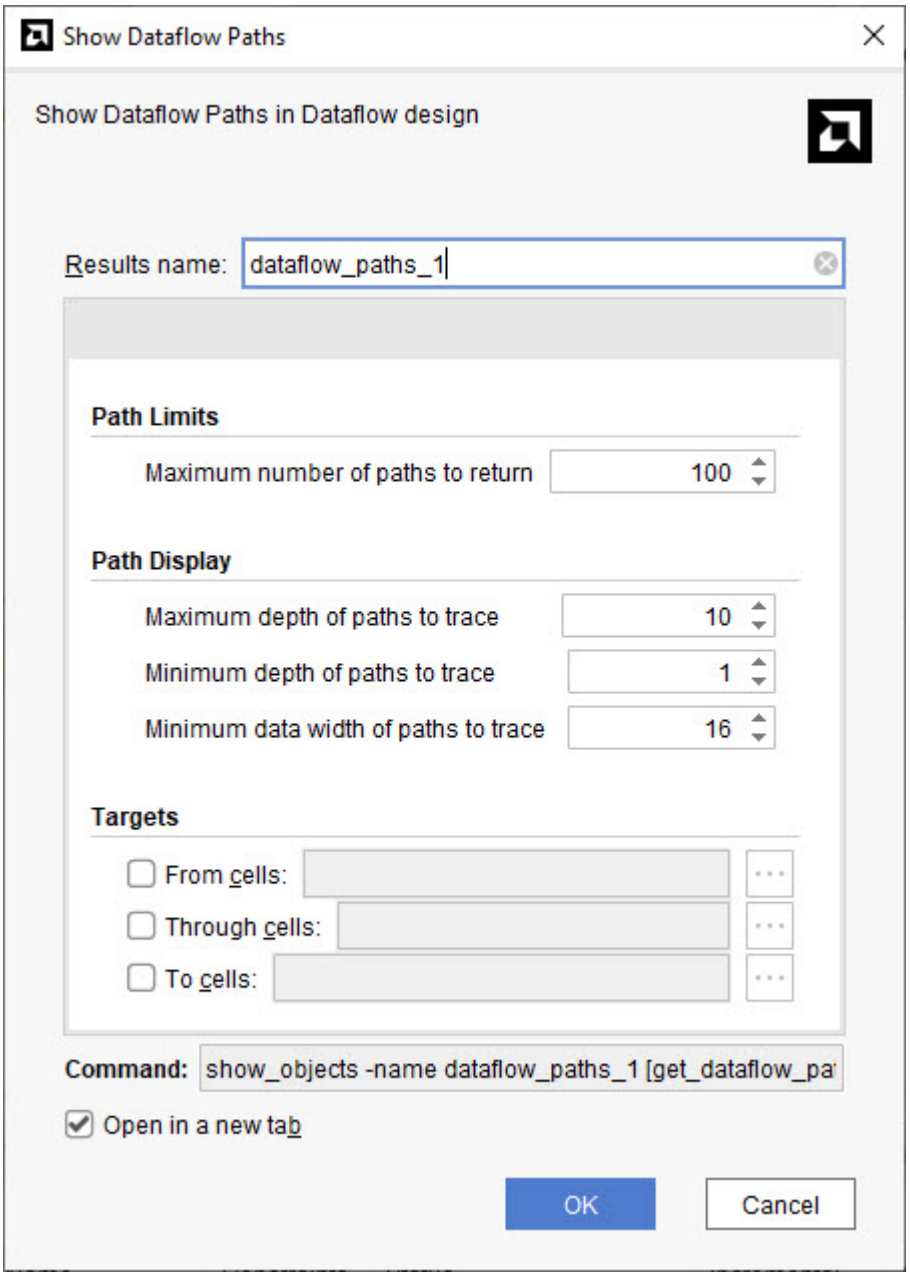
1. Run the `show_objects` Tcl command. For example:

```
show_objects -name dataflow_paths_1 [get_dataflow_paths]
```

2. View the results in the Vivado interface. You can select and cross probe the returned paths in the Schematic, Netlist, and Device windows.

Use the following instructions to generate dataflow paths using the GUI:

1. Select **Tools → Show Dataflow Paths**.
2. Configure the settings in the dialog box:
   - Control how many paths to return
   - Set the minimum and maximum number of cells to trace through
   - Define the minimum data width of buses in a path
   - Target paths that start, end, or pass through specific cells
3. Or, right-click a leaf cell and select **Show Dataflow Paths → From/Through/To Cell**



4. Vivado opens the dialog box with the selected cell name prefilled.

`get_dataflow_paths`

The `get_dataflow_paths` command prioritizes results. Vivado scans the dataflow netlist using the `-max_depth` value (default: 10) and returns up to the number of paths set by `-max_paths` (default: 100). When more paths exist than allowed, Vivado returns them in this order:

1. Paths equal to `max_depth`

2. Paths shorter than `max_depth`, sorted by depth
3. Remaining paths in alphabetical order of the startpoint

To customize the results in Tcl, use the following switches with `get_dataflow_paths`:

- **-from <cells>:** Set startpoints
- **-to <cells>:** Set endpoints
- **-through <cells>:** Set midpoints
- **-max_depth <integer>:** Set the longest path to search (default: 10)
- **-min_depth <integer>:** Set the minimum path depth (used only if `max_paths` is not reached)
- **-max_paths <integer>:** Set the number of paths to return (default: 100)
- **-min_width <integer>:** Set the minimum bus width to include (default: 1)

Set these options to control the number, depth, and width of the paths returned.

Disabled Commands within a Dataflow Design

The dataflow viewer is only for netlist analysis of a dataflow-optimized netlist. Because of this, many commands are not supported with the dataflow design.

Unsupported commands include, but are not limited to, the following:

- Examples of timing commands include:
  - `report_timing`
  - `report_clock_interaction`
  - `create_clock`
  - `set_max_delay`
  - Any timing constraint or timing report
- Examples of ECO commands include:
  - `create_cell`
  - `create_net`
  - `create_pin`
  - `connect_net`
- Examples of implementation commands include:
  - `synth_design`

    - `opt_design`
    - `place_design`
    - `phys_opt_design`
    - `route_design`
    - `write_device_image`
- Examples of reporting commands include:
    - `report_high_fanout_nets`
    - `report_control_sets`
    - `report_clock_utilization`
    - `report_ram_utilization`
    - `report_design_analysis`
    - `report_qor_suggestions`
    - `report_qor_assessment`

Viewing Reports and Messages

The AMD Vivado™ Integrated Design Environment (IDE) generates reports and messages. Reports are used to analyze the design in Vivado. Messages are used more to debug when something does not behave as expected.

Reports can be generated in the Vivado IDE, at the Tcl prompt, during a design implementation or synthesis run by inclusion in a run script or by leveraging reporting strategies in project mode runs.

Many reports have restrictions on when they can be run. The restrictions are different for different reports so refer to the individual command documentation for more information. The following are examples of common restrictions:

- Require a design to be opened
- Require the design to be a particular device architecture
- Require a particular flow such as DFX to be used
- Require the design to be in given state such as placed

Messages are extracted from the log file when commands are run. It is important to understand what command has written the message and severity of the message.

Viewing and Managing Messages in the IDE

Messages provide brief status updates about specific design elements or errors that occur during tool processes.

TIP: *Review the messages to check whether the Vivado tools are experiencing difficulties or encountering errors in any part of the design.*

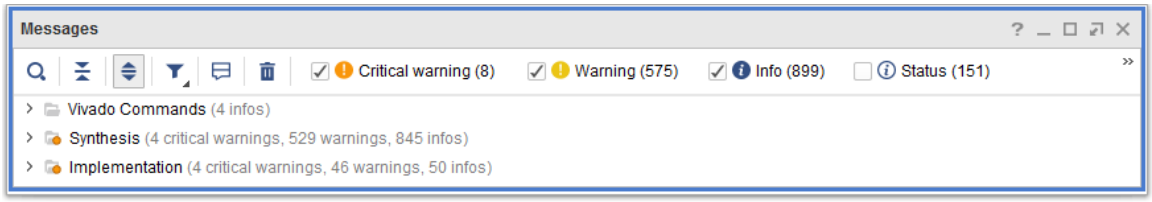
Using the Messages Window

The Vivado (IDE) displays two types of messages:

- Messages stored on disk
- Messages stored in memory

The Messages window groups messages by the action that generated them. You can use the toolbar settings to regroup messages by Message ID or File.

Figure 25: **Messages Window**



Some messages include hyperlinks to a file or design element. Click these links to view the source or select the design element.

TIP: *Right-click a message to copy and paste it into another window or document.*

Each message includes:

- **Message ID:** Helps you identify, group, and sort messages.
- **Message Severity:** Indicates how serious the message is and whether it requires your attention.

Some messages require your attention and resolution before you can elaborate, synthesize, or implement the design. Others are informational only and provide details about the design or process without requiring any action from you.

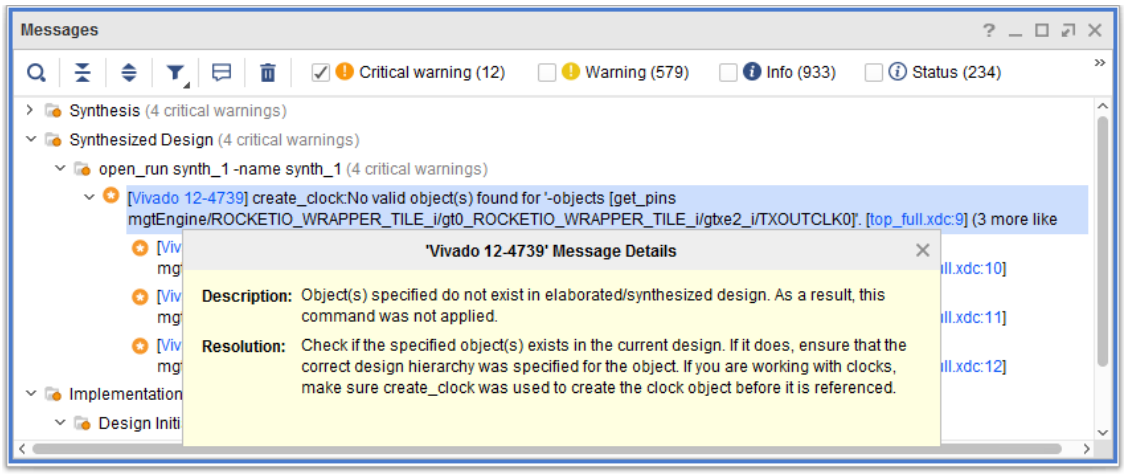
Table 1: **Message Severity**

Severity	Message
Status	Communicates general status of the design processing.
Info	Provides feedback about the process or design.
Warning	Indicates that constraints or specifications are not applied as intended. Design results can be sub-optimal.
Critical Warning	Signals user input or constraints that were not applied or fall outside best practices. These often lead to errors later. Review and correct them.
Error	Indicates a problem that stops the design flow. You must address the issue before proceeding.

RECOMMENDED: *Carefully review all errors and critical warnings when loading a design or reviewing active synthesis and implementation runs. Many messages include detailed descriptions and resolution tips you can view by clicking the* ***Message ID****.*

For example, the following figure shows a primary clock constraint refering to a missing port, and the tool issued a warning. It then generates a critical warning when it cannot create the clock. All dependent constraints fail as a result.

Figure 26: **Reviewing Errors and Critical Warning**



Filtering Messages

You can filter messages by severity to manage which types of messages are displayed.

Use the following instructions 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 also change the severity of a specific message ID. For example, lower the severity for a message you consider non-critical, or raise it for one that needs more attention.

To change the severity, use the `set_msg_config` Tcl command. For example:

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

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

Vivado Generated Messages

This section explains the messages that the Vivado tools generate during the design flow.

Synthesis Log

The Vivado synthesis log is the primary output from the synthesis tool. It includes details such as:

- Files processed:
  - 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 log or the Messages tab for errors, critical warnings, and warnings. The synthesis tool can issue messages that cause serious issues later. Address them early to avoid downstream problems.*

Implementation Log

The Vivado implementation log includes:

- Information about the location, netlist, and constraints used.
- Logic optimization task. The tool runs logic optimizations by default to reduce utilization and improve design performance.
- 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.

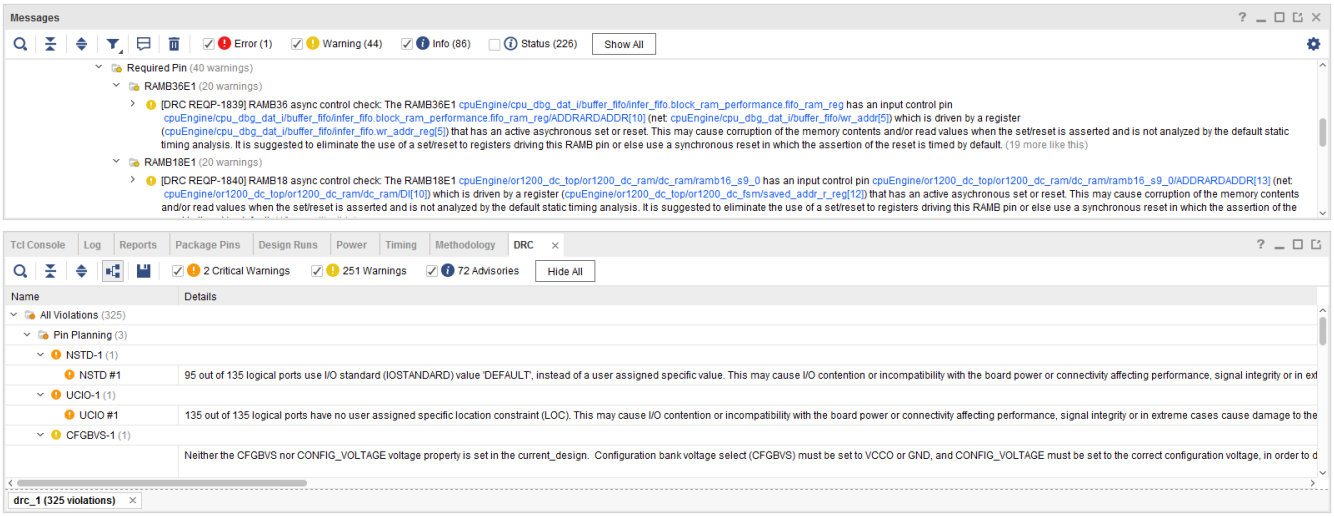
Review this report or the proper section of the messages tab for errors, critical warnings, and warnings. The placer generates warnings that can be elevated to errors later in the flow. If using stepwise runs, the log contains only the results for the last step.

> **IMPORTANT!** *Review this log or the messages tab for errors, critical warnings, and warnings. The synthesis tool can issue messages that cause serious issues later in the design flow. Address them early to avoid downstream problems.*

Using Report DRCs

Design rule checks (DRCs) examine the design and report common issues. During implementation, the tools run DRCs again. After placement and routing, the DRCs become more complete and comprehensive.

Figure 27: **Showing Critical Warnings and Error**



Review the DRC messages, critical warnings, and warnings early in the flow to avoid issues later.

Critical warnings during early design stages become errors later during implementation and block bitstream creation. In the previous example from a post-synthesized design, the optional report DRC step reports a critical warning for unconstrained I/Os. The post-route DRC report also lists these critical warnings. At the `write_bitstream` stage, these critical warnings escalate to error-level severity and prevent bitstream generation.

Review the DRC reports early to identify and correct areas in the design that require modification.

Generating and Waiving Design Checks

The waiver mechanism allows you to waive CDC, DRC, and Methodology violations in your design. After you waive a violation, it no longer appears in the results of the `report_cdc`, `report_drc`, and `report_methodology` commands.

Waived DRCs are excluded from mandatory checks that run before implementation steps such as:

- `opt_design`

- `place_design`
- `phys_opt_design`
- `route_design`
- `write_bitstream`

Use the following instructions to create and apply waivers:

1. Create waivers at the top level or scope them to a specific hierarchical module.
2. Use `read_xdc` or `source` to import waivers into your design. You can include them in any XDC file or Tcl script in both Project and Non-Project modes.
3. The tool automatically saves waivers inside design checkpoints and restores them when checkpoints are reloaded.
4. Use `write_xdc` or `write_waivers` to export waivers.

Vivado records the following metadata for each waiver:

- User who created it
- Date and time of creation
- Short description

Review and validate all waivers to ensure they remain appropriate and relevant to the current design state.

Waivers are first-class objects that you can create, query, report on, and delete. They reference other first-class objects returned by Vivado `get_*` commands, such as pins, cells, nets, Pblocks, and sites.

Make sure the referenced objects exist in the design before you create the waiver. If the design objects do not exist at the time of waiver creation, the waiver does not apply to them.

IMPORTANT! *Create waivers on a post-synthesized design. Avoid creating waivers on post-implemented designs because the referenced objects might not exist in the post-synthesis netlist. Invalid waivers are discarded if applied at the wrong stage to non-existing objects.*

The waiver mechanism supports replication and deletion of netlist objects.

- When an object involved in a waiver is replicated, Vivado automatically adds the replicated object to the waiver.
- When a referenced object is deleted, Vivado removes it from the waiver.
- If the deletion leaves the waiver with no valid references, Vivado removes it from the in-memory design and does not save it in future checkpoints.

This behavior also applies to timing constraints and clock objects. If a clock is deleted through logic optimization or by removing timing constraints using `reset_timing`, Vivado deletes any waiver that references the clock. That waiver is not saved in future checkpoints.

Note: The following commands do not update waiver references:

- `rename_net`
- `rename_cell`
- `rename_port`
- `rename_pin`

If you rename a referenced object, the waiver becomes invalid because it still points to the original name.

Note: You cannot waive Custom Design Rule Checks. For more information, refer to Creating Custom Design Rules Checks (DRCs) in the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894).

Creating a Waiver

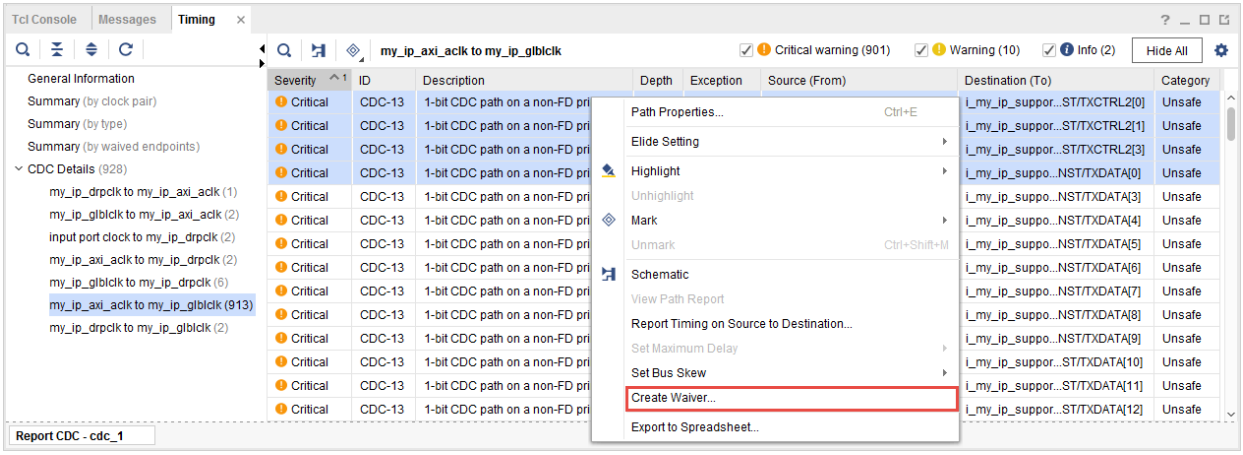
You can create waivers in the GUI, from a DRC, Methodology, or CDC violation object, or by manually specifying all the required arguments.

Creating Waivers from the GUI

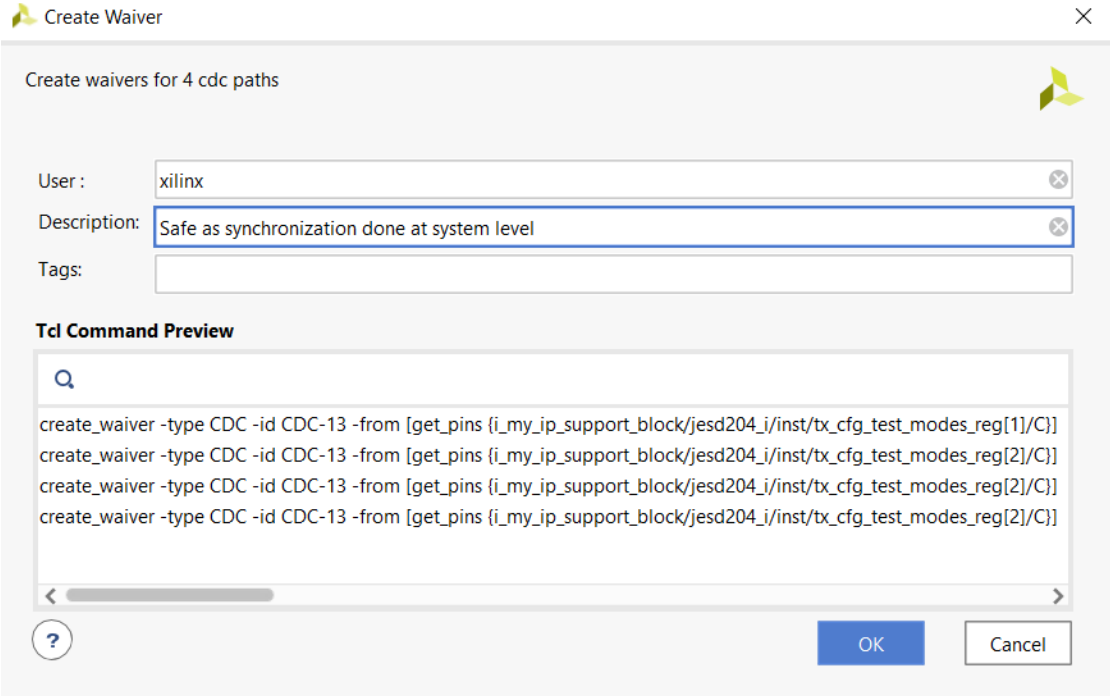
You can create waivers directly from any Report DRC, Report Methodology, or Report CDC GUI result window.

Use the following instructions to create a waiver from a report window:

1. Select one or more violations in the result window.
2. Right-click and select **Create Waiver** to open the create waiver creation widget.



3. Fill out the form.

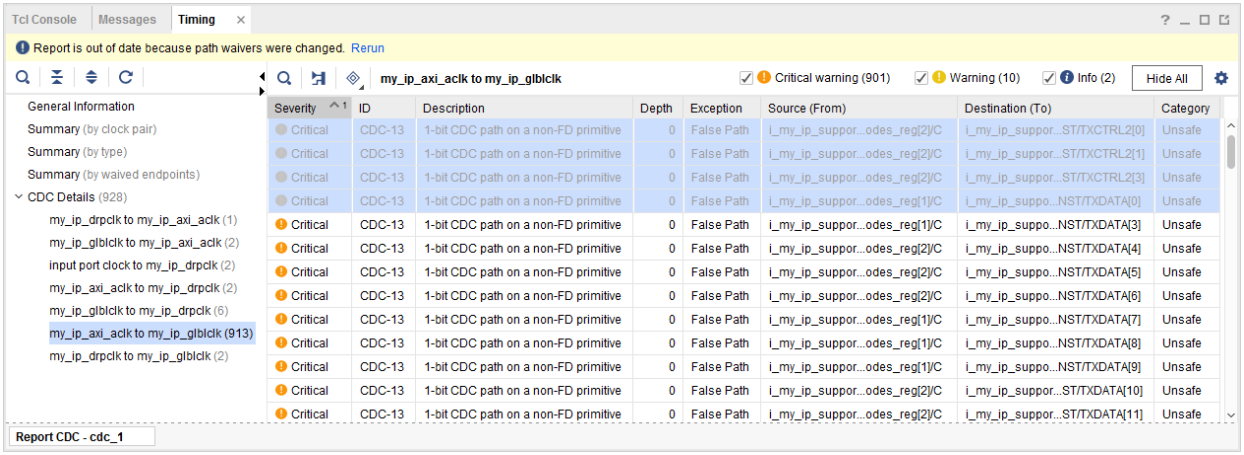


- **User:** The Vivado tools auto-fill this field, but you can edit it.
- **Description:** This field is mandatory. Provide clear, detailed information for review by the design team.
- **Tags:** Use this field to add a string or list of keywords. Tags help with documentation and make it easier to search the XDC file or filter waivers using the `get_waivers` command.

4. Submit the form. After you submit the form, Vivado sends one `create_waiver` command to the Tcl Console for each selected violation.

   ***Note*:** When waivers are created for DRC and methodology violations, the Tcl command `create_waiver` generated by the GUI references the violation object, but only as transitional form. The waiver engine then converts this into a fully descriptive waiver that includes all the related design objects. The final waiver does not rely on the original violation object. A timestamp is automatically added when the waiver is created.

5. Verify waiver creation by observing the following indicators:
   - The selected rows in the report are grayed out and disabled.
   - The report becomes stale, indicating waivers were applied.
   - After re-running the report, waived violations are filtered out.



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

Creating Waivers from a Violation Object

The second way to create waivers is by using DRC, Methodology, or CDC violation objects. This is the method Vivado uses in the GUI when it sends `create_waiver` commands to the Tcl Console.

Use the following syntax to create a waiver from one or more violation objects:

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

- `-description` is required.
- If you omit the `-user` option, Vivado uses the user ID of the person running the tools.

***Note*:** When you specify multiple violation objects, Vivado creates one waiver for each violation. These waivers do not directly reference the original violation objects. Instead, they capture the list of strings and objects included in each violation.

Use the following commands to get violation objects:

- `get_cdc_violations`
- `get_drc_violations`
- `get_methodology_violations`

  These commands only return results if you have already run `report_cdc`, `report_drc`, or `report_methodology`.

To retrieve violations from a specific named report, use the `-name option`.

Example: Waiving CDC-1 Violations in a Specific Module

```
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}
    }
}
```

This code creates a waiver for each CDC-1 violation that has a startpoint in the module `top/sync_1`.

Customizing Waivers to Cover Multiple Violations

Vivado builds waivers from the objects and strings in each violation, making each waiver unique. To apply a waiver to multiple violations, follow these steps:

1. Export the waiver to an external file.
2. Edit the `create_waiver` command manually.
3. Use patterns and wildcards to generalize the single objects and strings.

For guidance on using patterns and wildcards, see Creating DRC and Methodology Waivers.

Special Handling for Some Checks

Vivado automatically converts certain strings to wildcards for specific checks, including the following:

- `UCIO-1`
- `NSTD-1`
- `TIMING-15`
- `TIMING-16`

For `TIMING-15` and `TIMING-16`, Vivado ignores the setup and hold slack amount. The `create_waiver` command replaces the string representing slack with a wildcard. This allows the waiver to apply to the specific object regardless of the reported slack value. Similar wildcard behavior applies to `UCIO-1` and `NSTD-1`.

Creating Waivers from the Command Line

Each DRC, Methodology, or CDC violation is unique. It includes one or more of the following elements: strings and design or device objects such as pins, cells, nets, Pblocks, sites, and tiles.

1. Preserve the order of elements. The order and content of all strings and objects in a violation are critical. Providing arguments in the wrong order when creating a waiver can fail to match the violation or mistakenly waive the wrong one.

2. Before creating a manual waiver for a specific violation (such as `TIMING-14#1`) or a class of violations (such as all `TIMING-14`), follow these steps:
   a. Create an example waiver using the GUI or a violation object.
   b. Export the waiver using the `write_waiver` or `write_xdc` command.
   c. Review the exported content to understand the correct order of strings and objects.
   d. Use this format as a reference for other waivers with the same check ID.
3. Include the following for every CDC, DRC, or Methodology waiver:
   - **`-id`:** Specifies the violation or check ID (for example, `CDC-1`, `TIMING-14`, `PDRC-1569`). Only one ID can be specified per waiver.
   - **`-description`:** A multi-line string that provides enough information for team review.
4. Use the recommended arguments:
   - **`-type`:** Forces the waiver type (`CDC`, `DRC`, or `Methodology`). A waiver with the wrong type does not match any violation. For example, to waive a CDC violation, set the type to `CDC`. When the type is not specified, the system infers the type from the check id specified with `-id`.
   - **`-user`:** Overrides the default user name, which is the user ID running Vivado Design Suite.
5. When you use scoped waivers, use the `-scope` command to ensure wildcards are scoped. 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.
6. Avoid duplicate waivers. Vivado treats a waiver as a duplicate if another waiver exists with the 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.
```

7. Identify read-only checks. Some checks, such as those starting with `RTSTAT-*`, are read-only and cannot be waived. To filter out read-only checks, use the `IS_READ_ONLY` property:

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

   If you try to waive a read-only check such as DRC RTSTAT-1, Vivado returns an error:

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

Creating DRC and Methodology Waivers

The number and type of additional arguments you need for the `create_waiver` command depend on the specific DRC or Methodology violation. Some violations, such as `TIMING-9`, are generic and do not reference any specific strings or objects. These have no additional arguments. Other violations include one or more strings and design objects. You must specify these elements in the waiver.

***Note*:** Do not waive violations like `TIMING-9` or `TIMING-10` that contain no strings or objects.

1. Define the waiver elements.
   a. Use `-string` to specify the string values found in the violation.
   b. Use `-objects` to specify design elements such as pins, cells, nets, Pblocks, and sites.
   c. Repeat each of these options for every element the violation references.
2. Create the waiver.
   - Creating a waiver via the GUI or a specific violation object includes all relevant strings and objects, restricting the waiver to that specific violation.
   - When creating waivers manually, you can broaden the waiver to apply to multiple similar violations.
3. Expand the waiver coverage manually. Use these tips to create a broader waiver that covers multiple violations:
   - Use patterns with `get_*` commands instead of specific object names.
   - Use wildcards in place of strings or objects. For example:
     - `*` matches any string.
     - `*PIN` matches any pin.
   - Use a list of objects instead of a single object. If any object of the same type in the list matches the element found inside the violation for the same position, the match succeeds.

   A violation is waived only when all elements in the waiver match the corresponding elements in the violation.

Use the following wildcard keywords to match design objects by type when creating waivers:

Table 2: **Wildcard Keywords**

Object Type	Wildcard
Cell	`*CELL`
Net	`*NET`
Pin	`*PIN`
Port	`*PORT`
Site	`*SITE`

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

Object Type	Wildcard
Tile	*TILE
BEL	*BEL
Package Bank	*PKGBANK
Clock Region	*CLKREGION
Clock	*CLOCK
Pblock	*PBLOCK
String	*

Examples

- Waive 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 ]
```

- Waive multiple similar violations:

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

Note: `create_waiver -scope` forces wildcards for pins and cells to stay within the current instance. This prevents wildcards from matching higher-level objects and unintentionally waiving other violations.

Creating CDC Waivers

CDC waivers are simpler to define because each CDC violation references only two objects: a source pin or port and a destination pin or port. Use the `-from` and `-to` options to specify the source and destination pins or ports. Do not use `-string` or `-objects` with CDC waivers.

IMPORTANT! *CDC waivers are based only on the source and destination pins. They do not consider the source and destination clocks. Creating a waiver from Vivado or CDC violations with the same pins for different clock pairs triggers a warning message.*

```
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-7' is a duplicate and will
not be added again.
```

Example: CDC-1 Waiver Between Two Pins

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 you omit the `-from` or `-to` option, Vivado treats the missing option as a wildcard. The following two commands are equivalent and waive all CDC-1 violations 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 reports only one violation per endpoint and per clock-pair. If multiple violations exist for a single endpoint and clock pair, only the violation with the highest precedence is reported.

CDC rules are ordered by precedence. The rule with the highest precedence is reported first. If you waive that violation, the next violation in the precedence list is reported.

The following table shows the CDC rules sorted from highest to 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 marked as Warning appear above Critical rules in precedence because their rules do not apply to the same endpoint due to different CDC topologies.

When an endpoint has multiple CDC violations, waiving the highest precedence causes the next violation to be reported in order.

Viewing All Violations Per Endpoint

To simplify waiver creation, you can generate a report that includes all CDC violations for each endpoint, regardless of rule precedence.

1. Run the `report_cdc` command with the `-all_checks_per_endpoint` option to include every CDC rule that applies to an endpoint:

   ```
   report_cdc -all_checks_per_endpoint -name full_cdc_report
   ```

   ***Note*:** This option is only available from the Tcl Console. It is not supported in the Report CDC dialog window.

2. You can view the results of `-all_checks_per_endpoint` in the Vivado IDE by using the `-name` option.

Reporting the Waivers

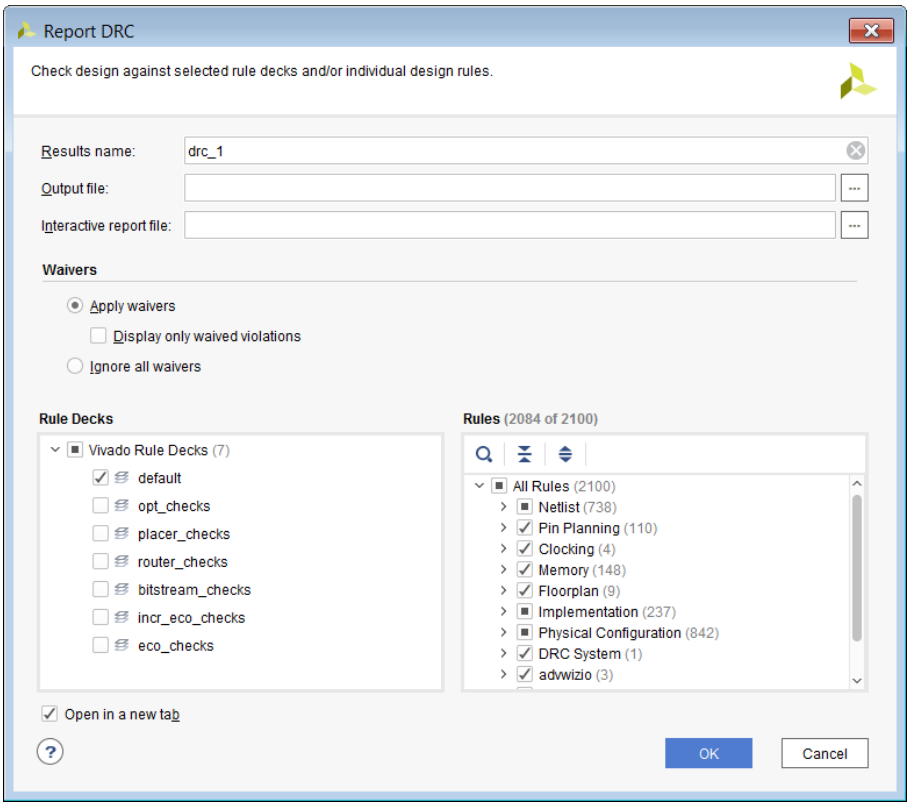
After you define waivers and before you generate the final bitstream, verify that only the expected violations are waived.

The `report_cdc`, `report_drc`, and `report_methodology` commands support multiple reporting modes. By default, these commands report only violations that are not waived.

1. Use `-waived` to force `report_cdc`, `report_drc`, and `report_methodology` commands to report only the violations that are waived. Review the results to confirm that all waived violations are expected.
2. Use `-no_waiver` to report all violations, whether waived or not.

These reporting modes are available from the Tcl Console and the GUI Report dialog windows. The following image is from Report DRC and shows how to select a reporting mode under the Waivers section. The same Waivers section is also available for Report CDC and Report Methodology widgets.

Figure 28: **Reporting Modes for Waivers**



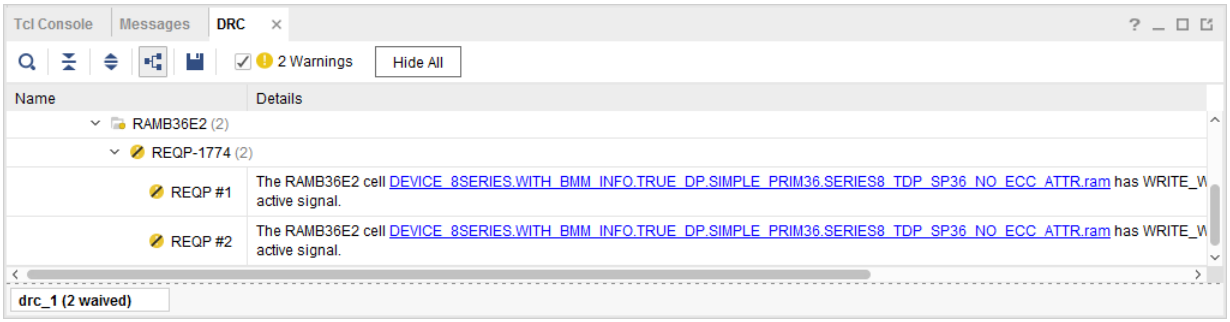
In the CDC, DRC, and Methodology GUI result windows:

- A different icon appears next to waived violations.
- The result window name includes the number of waived violations.

***Note*:** You cannot create waivers from a result window that shows waived CDC, DRC, or Methodology violations.

The following example shows a DRC result window with two waived violations.

Figure 29: **Waived DRC Violations**



Summarizing Waivers

Use the `report_waivers` command with the following instructions to generate a summary report of all waivers and the violations they waive. This report is available only from the Tcl Console.

1. Run `report_cdc`, `report_drc`, and `report_methodology` before running `report_waivers`. The `report_waivers` command only reports statistics that those three commands extract.
2. Rerun `report_cdc`, `report_drc`, and `report_methodology` any time you modify waivers (add or delete). This ensures the statistics are accurate.
3. Run `report_waivers` from the Tcl Console. This command generates a summary of all defined waivers and the violations they affect waive. You cannot run `report_waivers` from the GUI.
   - If the statistics are not updated, `report_waivers` issues one or more warnings depending on which set of waiver 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 as follows:

- **Total Vios:** Total number of violations before the waivers apply. This is the number of violations reported without waivers. For multi-bit rules, the total is based on the number of endpoints.
- **Remaining Vios:** Number of violations after the waivers apply. If no violations are waived, this number matches *Total Vios*. For multi-bit rules, this count is also based on the number of endpoints.
- **Waived Vios:** Number of violations that are waived. If no violations are waived, this number is 0. For multi-bit rules, this count is based on the number of endpoints.
- **Used Waivers:** Number of waivers that waive at least one violation. A single waiver can apply to multiple violations if it uses patterns or wildcards.

- **Set Waivers:** Total number of waivers applied to the design. Ideally, this number matches *Used Waivers*. If the numbers differ, some waivers do not match any violations.

By default, the detailed tables include only rules that have waivers defined. However, the first summary table includes all violations in the design, whether waived or not.

Figure 30: **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.
```

Expanding the Report and Exporting Waiver Lists

Follow these steps to include all violations in the report, waived or not waived, and to export waiver lists:

1. Add the `-show_msgs_with_no_waivers` option to the `report_waivers` command to include all rules with violations in the detailed tables, even if waivers exist.

```
-----------------
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.
```

2. Use the `-write_valid_waivers` option to export waivers that match at least one CDC, DRC, or Methodology violation.
3. Use the `-write_ignore_waivers` option to export waivers that do not match any violation.
4. Review the list of waivers that do not match any violation. If a waiver does not match and you did not expect that, check the waiver definition for errors.
5. Understand that `-write_valid_waivers` and `-write_ignore_waivers` filter waivers based on data from the latest execution of `report_cdc`, `report_drc`, and `report_methodology`.
6. If you run `report_drc` or `report_methodology` with a rule deck or subset of checks, Vivado ignores waivers for rules not included in the run.

7. Run all DRC and 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, waivers are automatically saved inside the checkpoint and restored from the checkpoint. Waivers are saved in both plain XDC and binary constraint formats.

1. Use `write_xdc` and `write_waivers` to export waivers into standalone XDC files.
2. Use `read_xdc` or `source` to reload exported XDC files.
3. Use `write_xdc` to export all waivers and design constraints in the order they apply to the design. This includes user-defined waivers and AMD IP waivers.

   Note: To export only waivers, add the `-type waiver` option:

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

   IMPORTANT! *The IP waivers are created with the option `create_waiver -internal`. Never use `create_waiver -internal` in your own waivers. This option is reserved for AMD IP waivers.*

4. Use `write_waivers` to export user CDC, DRC, and Methodology waivers. This option gives you more control and granularity.

   Note: AMD IP waivers are not exported with this command.

5. Use the `-type` option to export a specific category of waivers. For example, export only CDC waivers:

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

6. Use the `-id` option to export waivers for a specific check. For example, export all waivers for 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

Table 4: **Exporting the Waivers** *(cont'd)*

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

Other Waiver Commands

Return Waiver Objects

The get_waivers command returns a collection of waiver objects. You can filter waivers by type, name, or pattern.

Use the following command to return all DRC waivers:

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

Use the following command to return all DRC DPIR-2 waivers:

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

Note: The get_waivers command does not return AMD IP waivers.

Remove User Waivers

The delete_waivers command deletes objects returned by get_waivers.

Use the following command to delete all user waivers:

```
delete_waivers [get_waivers]
```

Use the following command to delete all CDC waivers:

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

Note: You cannot delete AMD IP waivers.

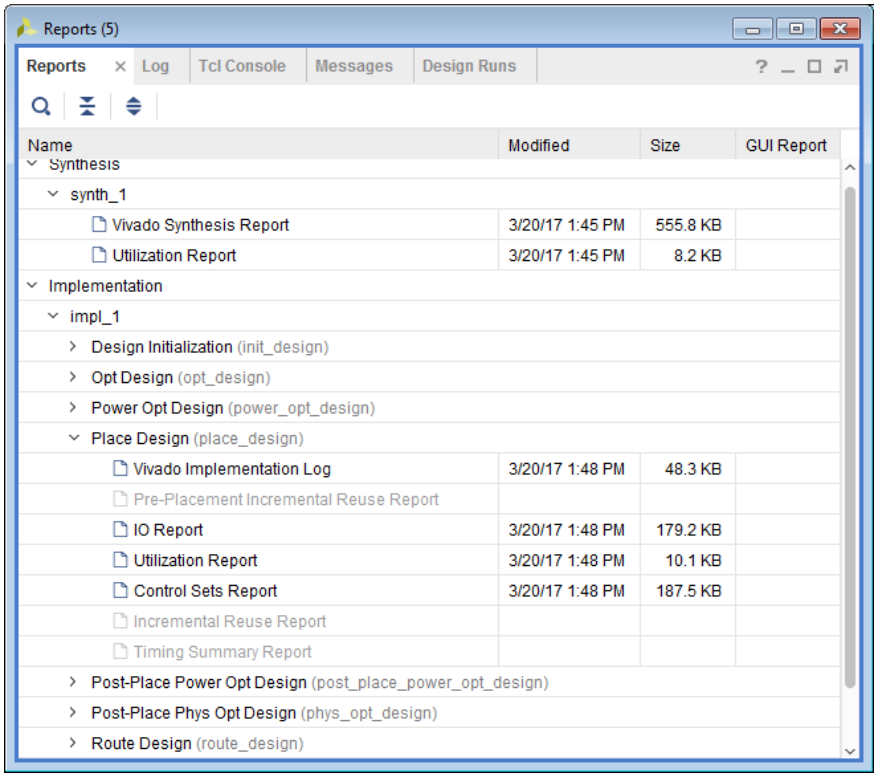
Using the Reports Window

Project mode users can view reports for the active synthesis and implementation runs in the Reports window. Reports for the other non-active runs can be accessed by following these steps:

1. In the Design Runs window, select the run you want to inspect.
2. Open the Run Properties window and select the Reports tab to view the associated reports.

3. Double-click any report to open it in the text viewer.

Figure 31: **Reports Window**



Configurable Report Strategies

Configurable Report Strategies let you choose which reporting commands run automatically after each step of the synthesis and implementation runs in Vivado project mode.

Depending on the design stage, design complexity, and your preferences, you can choose different sets of reports. Vivado provides several predefined synthesis and implementation report strategies by default.

You can also create your own custom report strategies. Vivado saves these with your user preferences and other IDE settings.

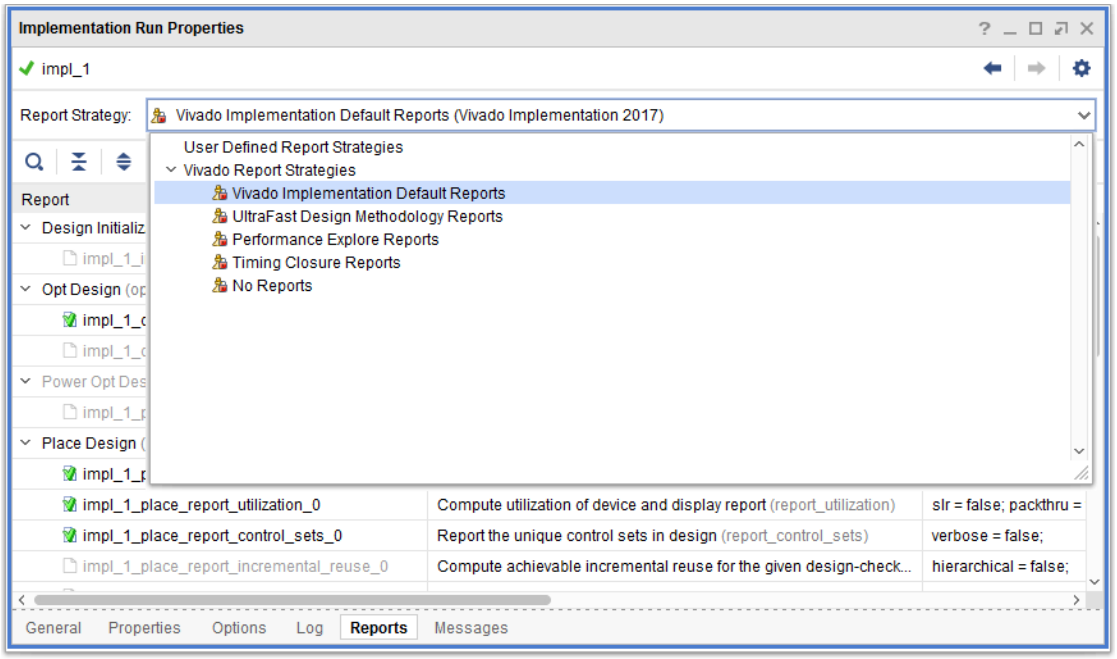
Setting Run Report Strategies

By default, Vivado uses the corresponding default report strategy for each synthesis and implementation run. To change the report strategy, use the following steps:

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 32: **Selecting the Report Strategy for an Implementation Run**



Vivado offers two flow categories, each with several predefined and user-defined report strategies:

Table 5: **Flows and Report Strategies**

Flow	Report Strategy	Note
Synthesis	Vivado Synthesis Default Reports	Runs only the utilization report after synthesis.
	No Reports	Minimizes runtime by skipping report generation.
Implementation	Vivado Implementation Default Reports	Matches the reports used in Vivado versions before 2017.3.
	UltraFast Design Methodology Reports	Runs all reports recommended in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).
	Performance Explore Reports	Includes default reports plus extra timing reports after `phys_opt_design`.
	Timing Closure Reports	Includes UltraFast reports plus `report_design_analysis` and `report_qor_suggestions`.
	No Reports	Minimizes runtime by skipping all reports.

Before launching the run, some reports appear grayed out for one of the following reasons:

- They are tied to a disabled flow step
- You have manually disabled them

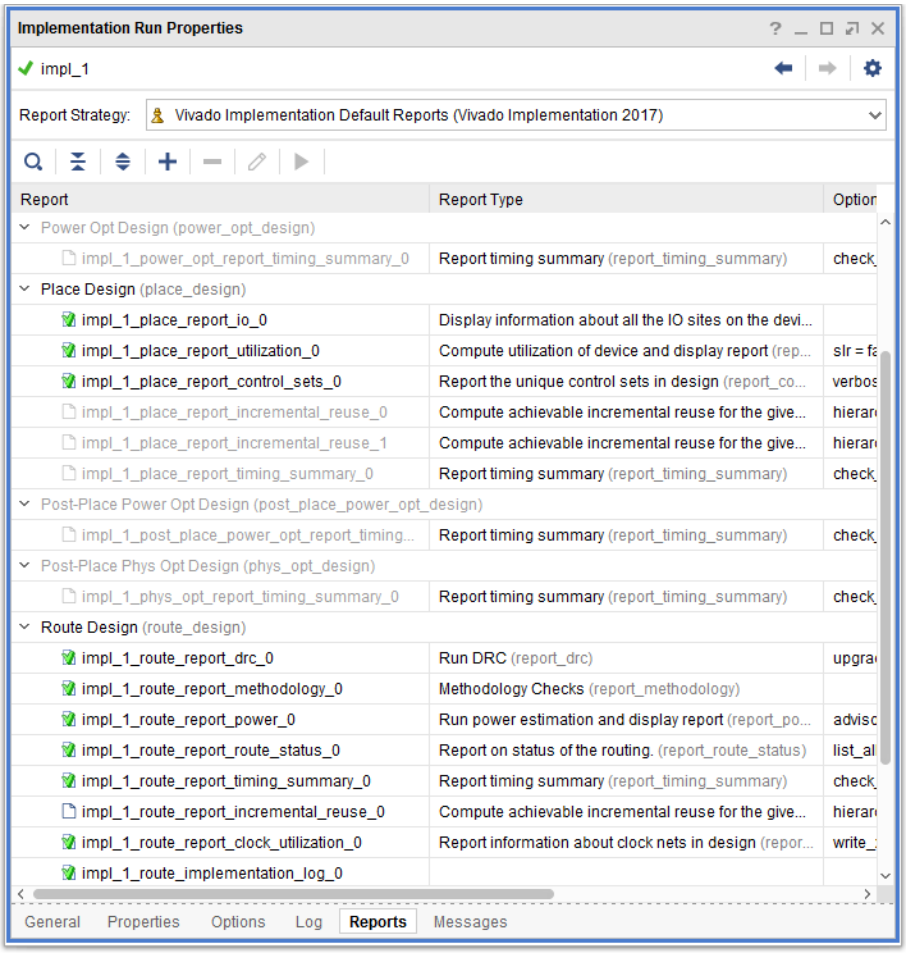
After the run completes, the following occurs:

- Reports with a green check mark are available.
- You can open any of these by double-clicking the report name in the Reports tab.

Some reports can still be unavailable with the following conditions:

- They are associated with a disabled flow step.
- You manually disabled them.
- They are only enabled in Incremental Compile runs.

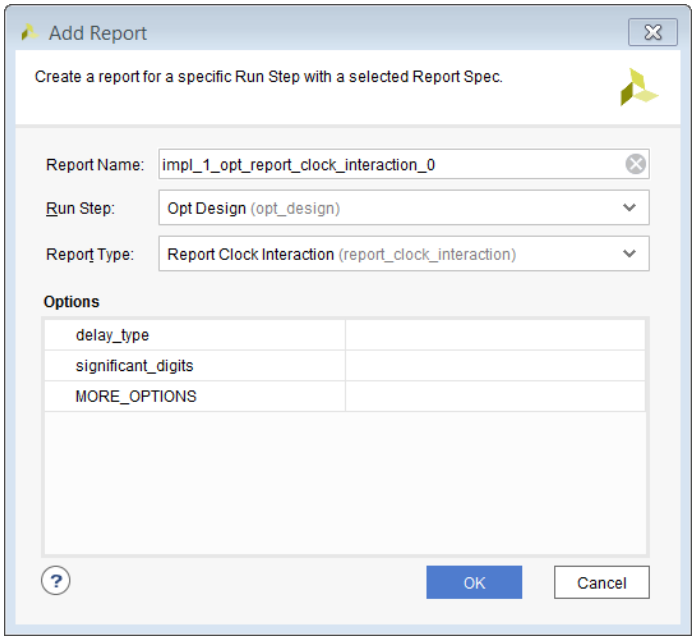
Figure 33: **Viewing Generated Run Reports**



Editing Run Report Strategies

After choosing a report strategy for a run, use the Reports tab in Run Properties to add, delete, or edit reports.

1. Add a report:
   a. Select the report and click **Add Report (+)**, or right-click an existing report and select **Add Report**.
   b. Choose the Run Step and Report Type, then review and edit the report options.
   c. If an option does not appear, add it using the MORE_OPTIONS field.
   d. Vivado assigns a default unique report name based on the run name, flow step, and report command name.



2. Delete a report:
   a. Select the report and click **Delete Report (-)**.
   b. The selected report is removed from the Run Report Strategy.
   c. This action cannot be undone.
3. Edit a report:
   a. Select the report and click **Edit Report**, or right-click and use the context menu.
   b. You can edit the report name, enable or disable the report, or change report options.
4. Enable or disable a report:
   a. Select the report and right-click to use the context menu.

You can add a report to a specific step or enable a previously disabled report after the run finishes. Click **Play** to generate the report.

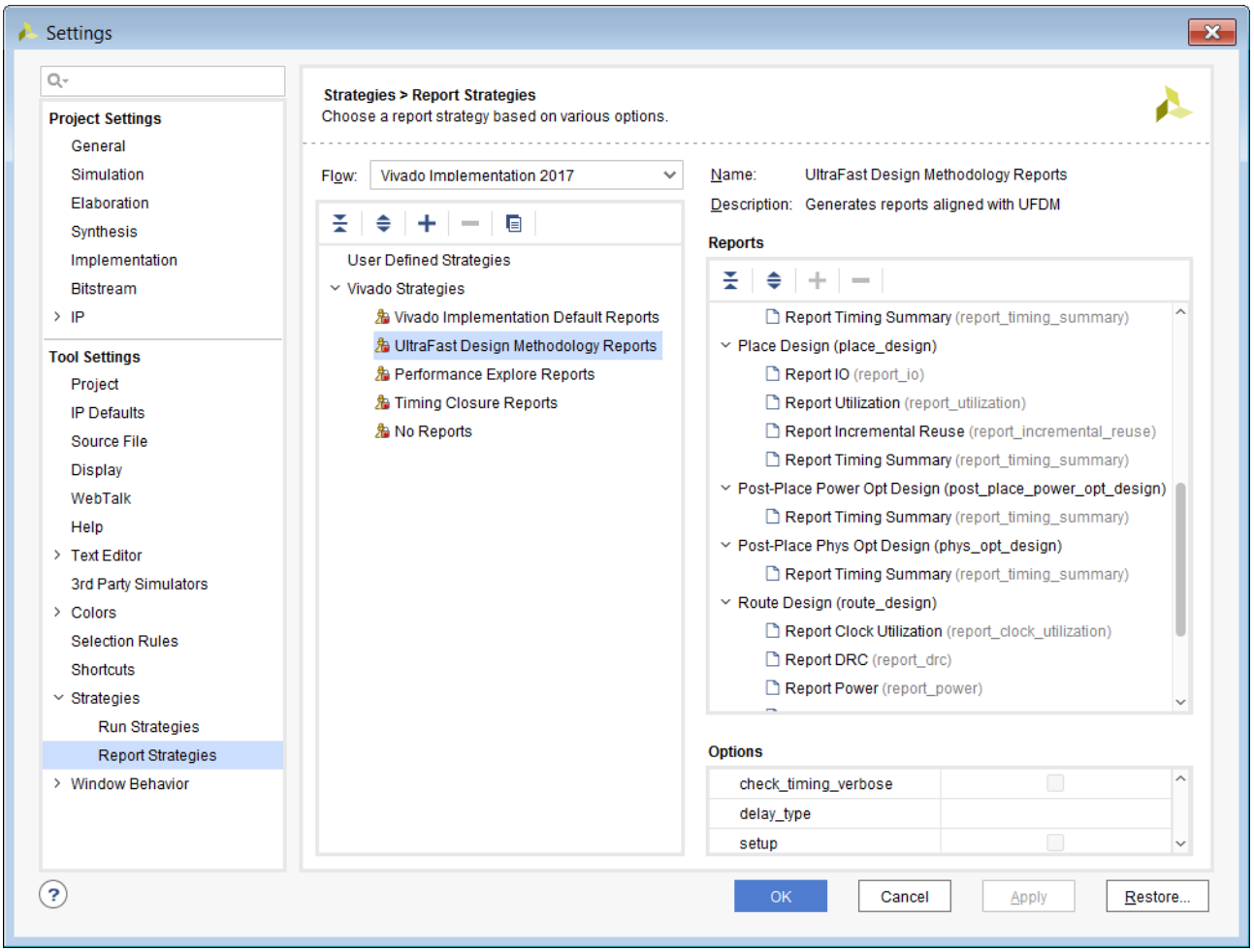
IMPORTANT! *When generating a report after the run completes, Vivado opens the checkpoint for the corresponding flow step in the background. This process blocks most Vivado IDE functions, including the Tcl Console, until the report finishes. Report generation can take anywhere from a few minutes to over an hour depending on design size and report complexity.*

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

Creating New Report Strategies

You must create and modify user-defined report strategies in the Vivado IDE Project Settings window in **Tool Settings → Strategies**. You cannot modify predefined report strategies or change the default association between a run strategy and a report strategy.

Figure 34: **Report Strategies Configuration in the Settings Window**



Use the following instructions to create a new Report Strategy:

1. In the Strategy window, click **+**
2. Complete the following fields:
   a. Enter the name of the strategy.
   b. Select the target flow: **Synthesis** or **Implementation**.
   c. (Optional) Add a description.
3. Click **OK** to create the strategy.

Use the following instructions to copy and modify an existing strategy:

1. Select an existing strategy.
2. Click **Copy**.
3. Edit the strategy name.

4. Click **+** to add a report.
5. Select a report and edit the options.

   ***Note*:** If an option is not available, enter it in the MORE_OPTIONS field.
6. Click **-** to remove a report you no longer want.
7. After all the edits are complete, click **OK**.

The Vivado IDE includes several predefined strategies. See Table 5: Flows and Report Strategies for a list.

Chapter 4

Timing Analysis

The AMD Vivado™ Integrated Design Environment (IDE) provides several reporting commands to help you verify that your design meets all timing constraints and is ready to load onto the application board.

Use the Report Timing Summary for timing signoff. This report gives you a comprehensive overview of all timing checks and includes enough details to help you begin analyzing and debugging any timing issues. For additional insight, refer to the Logic Analysis in the IDE.

You can:

- View the report in a window.
- Write it to a file.
- Print it to your log file.

If the Report Timing Summary shows that your design fails to meet timing or is missing constraints, do the following:

- Review the detailed sections of the summary.
- Use the information to run more specific analysis.

You can use other timing reports to:

- Focus on a particular issue.
- Narrow the scope of analysis using filters and scoping capabilities.

Before adding timing constraints, take time to understand the fundamentals of timing analysis and the key terminology used by the Vivado IDE timing engine. This chapter introduces those core concepts.

Terminology

- **Launch edge:** The active edge of the source (launch) clock that launches the data.
- **Capture edge:** The active edge of the destination (capture) clock that captures the data.
- **Source clock:** Also called the launch clock.

- **Destination clock:** Also called the capture clock.
- **Setup requirement:** The timing relationship between the launch edge and capture edge that defines the most restrictive setup constraint.
- **Setup relationship:** The setup check verified by the timing analysis tool.
- **Hold requirement:** The timing relationship between the launch edge and capture edge that defines the most restrictive hold constraint.
- **Hold relationship:** The hold check verified by the timing analysis tool.

Timing Paths

Timing paths follow the connectivity between instances in your design. In digital designs, a timing path forms between two sequential elements controlled either by the same clock or by two different clocks.

Common Timing Paths

The most common timing paths in a design include:

- 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 this path, data follows these steps:

1. A clock on the board launches the data outside the device.
2. The data reaches the device input port after a delay, known as the input delay (as defined by Synopsys design constraints (SDC)).
3. The data then propagates through the internal logic of the device and reaches a sequential cell that is clocked by the destination clock.

Internal Path from Sequential Cell to Sequential Cell

In this path, data follows these steps:

1. A sequential cell inside the device, clocked by the source clock, launches the data.

2. The data propagates through internal logic.
3. It reaches another sequential cell that is clocked by the destination clock.

Path from Internal Sequential Cell to Output Port

In this path, data follows these steps:

1. A sequential cell inside the device, clocked by the source clock, launches the data.
2. The data propagates through internal logic and reaches the output port.
3. A clock on the board captures the data after an additional delay, known as the output delay (as defined by SDC).

Path from Input Port to Output Port

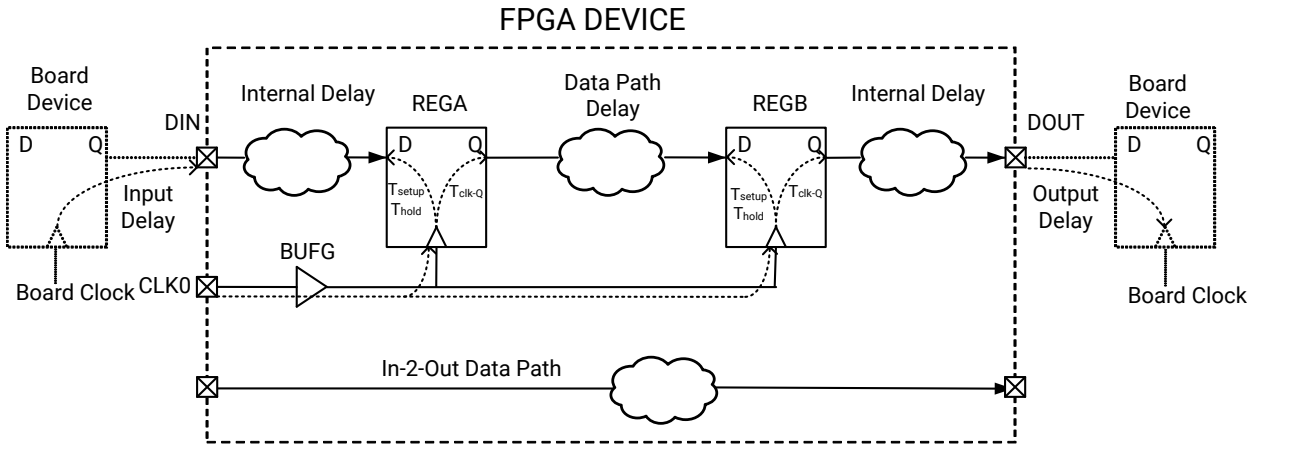
In this path, data traverses the device without being latched inside the device. This is commonly called an *in-to-out* path.

- The input clock and output clock can be either a virtual clock or a design clock.
- Input and output delays are defined relative to that clock.

Timing Paths Example

The following figure illustrates the paths previously described. In this example, the design clock `CLK0` serves as the board clock for both DIN and DOUT delay constraints.

Figure 35: **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 runs from its source point, usually an input port, to the clock pin of the launching sequential cell.

Data Path

The data path covers the path where data propagates between the startpoint and the endpoint.

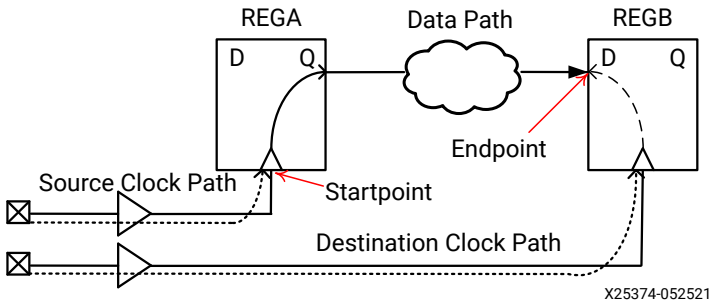
- **Startpoint:** Either a sequential cell clock pin or a data input port.
- **Endpoint:** Either a sequential cell data input pin or a data output port.

Destination Clock Path

This is the path the destination clock follows from its source (typically an input port) to the clock pin of the capturing sequential cell. If the timing path ends at an output port, there is no destination clock path.

The figure Destination Clock Path shows these three sections in a typical timing path.

Figure 36: **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, data transfers between two sequential cells within one clock period:

1. The launch edge occurs at 0 ns.
2. The capture edge occurs one clock period later.

The following section explains how these edges define the setup and hold relationships used in timing analysis.

Timing Analysis Key Concepts

Max and Min Delay Analysis

Timing analysis is the static verification that ensures your design behaves predictably when loaded and run on hardware. It accounts for manufacturing and environmental variations, which are captured in delay models grouped by timing corners and corner variations.

To verify your design, you need to analyze timing against all recommended corners. For each corner, perform all checks under the most pessimistic conditions. Your design must pass the following four analyses:

- Max delay analysis in the slow corner
- Min delay analysis in the slow corner
- Max delay analysis in the fast corner
- Min delay analysis in the fast corner

Each check uses delays that reflect the most pessimistic scenario. The tool always associates the following checks with specific delay types:

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

Max Delay with Setup and Recovery Checks

- Use the worst-case (slowest) delays of the selected corner for the source clock path and the data or reset path accumulated delays.
- Use the best-case (fastest) delays of the same corner for the destination clock path accumulated delay.

Min Delay with Hold and Removal Checks

- Use the best-case (fastest) delays of the selected corner for the source clock path and data or reset path accumulated delays.
- Use the worst-case (slowest) delays of the same corner for the destination clock path accumulated delay.

When these checks are mapped to the various corners, they correspond to the following:

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

Setup/Recovery (Max Delay Analysis)

Use the following combinations for 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)

Use the following combinations for 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)

IMPORTANT!

- Delays from different corners are never mixed on the same path when calculating slack.
- Most setup or recovery violations occur with slow corner delays.
- Most hold or removal violations occur with fast corner delays. However, because this is not always true, especially for I/O timing, AMD recommends performing both max and min delay analyses on both corners.

Setup/Recovery Relationship

The setup check uses the most pessimistic setup relationship between two clocks. By default, this is the smallest positive delta between the launch and capture edges.

For example, if two flip-flops are triggered by rising clock edges, both the launch and capture edges are rising.

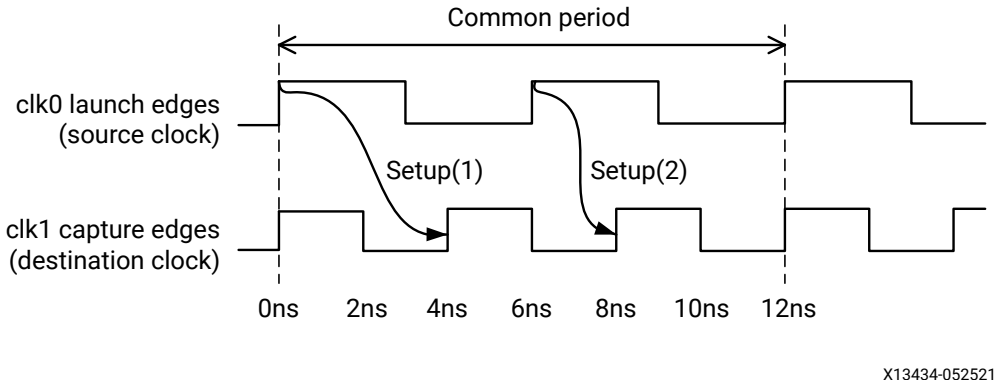
The clocks are defined as follows:

- `clk0` has a 6 ns period with the first rising at 0 ns and falling edge at 3 ns.

- `clk1` has a 4 ns period with the first rising at 0 ns and falling edge at 2 ns.

As shown in the following figure, two unique setup relationships exist: Setup(1) and Setup(2). The smallest positive delta from `clk0` to `clk1` is 2 ns, which corresponds to Setup(2). The common period is 12 ns, which is the time between two simultaneous alignments of the two clocks.

Figure 37: **Setup Relationships**



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

IMPORTANT! *If no common period is found after 1000 cycles of both clocks, the tool uses the worst setup relationship observed over these 1000 cycles. In such scenario, these clocks are considered unexpandable or without a common period. The resulting timing analysis is not always the most pessimistic case. Review the paths between these clocks to confirm if they need to be treated as asynchronous.*

After the tool establishes the path requirement, it computes setup slack using path delays, clock uncertainty, and setup time.

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 slack means data arrives before the required time.

The recovery check works the same way as the setup check but applies to asynchronous control pins, such as preset or clear. It uses the same clock relationships and slack equation, but substitutes recovery time in place of setup time.

Hold/Removal Relationship

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

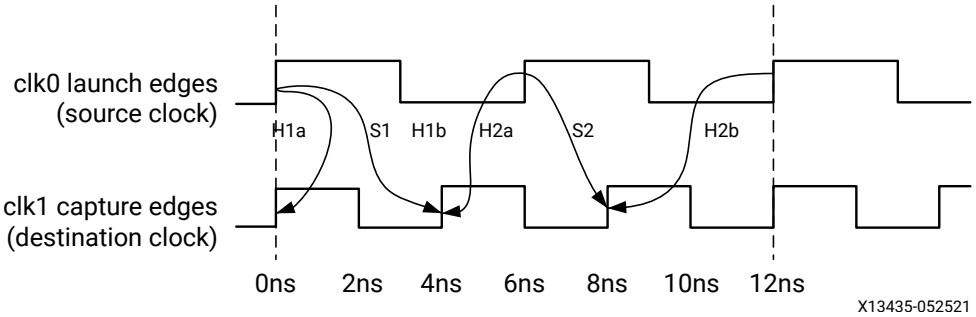
- Data from the setup launch edge is not captured by any active edge before the setup capture edge.
  - H1a and H2a edges match setup edges S1 and S2 in the following figure.
- Data from the next active source clock edge after the setup launch edge is not captured by the setup capture edge.
  - H2a and H2b edges match setup edges S1 and S2 in the following figure.

During hold analysis, the timing engine reports only the most pessimistic hold relationship between any two clocks. This is not always the same as the worst setup relationship. The timing engine evaluates all possible setup relationships and their corresponding hold relationships to find the most pessimistic hold case.

For example, consider the same path used in the setup relationship example. Two unique setup relationships exist. The figure shows two hold relationships for each setup case. The greatest hold requirement is 0 ns, which occurs at the first rising edge of both source and destination clocks.

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

Figure 38: **Hold Relationships per Setup Relationship**



After the path requirement is known, the tool uses the path delays, clock uncertainty, and hold time to calculate 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 works like the hold check but applies to asynchronous control pins, such as preset or clear. It uses the same clock relationship rules and slack equation, replacing hold time with removal time.

Path Requirement

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

Using the same clocks and path as in the previous section, the setup path requirements are as follows:

```
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 path requirements are as follows:

- For 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
  ```

- For 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
  ```

Timing analysis uses only the two most pessimistic path requirements. In this example, the most pessimistic paths are as follows:

- Setup requirement S2
- Hold requirement H1a

Clock Phase Shift

A clock phase shift occurs when a clock waveform is delayed relative to a reference clock because of special hardware in the clock path. In AMD FPGAs, this shift is typically introduced by MMCM or PLL primitives when the output clock property `CLKOUT*_PHASE` is set to a non-zero value.

MMCM/PLL Phase Shift Modes

During timing analysis, you can model clock phase shift in two ways by setting the `PHASESHIFT_MODE` property on the MMCM or PLL. The two options are:

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

PHASESHIFT_MODE Property	Phase-Shift Modeling	Comment
WAVEFORM	Models the phase shift by modifying the clock waveform.	Apply `set_multicycle_path -setup` constraints to adjust the timing requirements for clock domain crossing paths from or to the phase-shifted clock.
LATENCY	Models the phase shift as insertion delay through the MMCM or PLL.	Does not require additional multicycle path constraints.

The default phase-shift mode varies by AMD FPGA family, but you can override the default on each MMCM or PLL as needed.

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 `PHASESHIFT_MODE` property affects only the static timing analysis engine's slack computation. It does not change the actual device configuration. Switching between WAVEFORM and LATENCY modes can require updates to your timing constraints, which can affect design timing.*

IMPORTANT! *If a phase shift is defined on any of the `CLKOUTx` pins and multiple clocks reach the input pins of the MMCM/PLL, using `PHASESHIFT_MODE=LATENCY` is invalid. This triggers the warning: Timing 38-437. In this case, set `PHASESHIFT_MODE=WAVEFORM`.*

Using `PHASESHIFT_MODE=LATENCY` is especially helpful when you want to introduce skew between two clocks to meet timing. You do not need to add multicycle path constraints when the phase shift is negative, zero, or positive.

Migrating a legacy design from 7 series or UltraScale to UltraScale+ without setting `PHASESHIFT_MODE` defaults phase shift modeling to latency. In that case, you should review and typically remove any multicycle path constraints that were originally used to account for phase shift. You can identify these constraints by running `report_methodology`. Look for TIMING-31, which flags multicycle paths between clocks when one is phase-shifted and generated by an MMCM/PLL with `PHASESHIFT_MODE=LATENCY`.

Both the clocking wizard and the high speed SelectIO wizard allow you to force the clock phase-shift modeling on each MMCM/PLL. The `PHASESHIFT_MODE` property is automatically saved in the IP XDC file.

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. When you modify the clock waveform, you can cause the static timing analysis to use different clock edges for the source and capture clocks.

In the following examples, the MMCM auto-derives the clock `clkout0` with a period of 10 ns.

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 at 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 ($10\ \text{ns} / 360 \times 12.0$).

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 ($10\ \text{ns} / 360 \times 12.0$). The source clock edge remains at 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 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 decreased by 0.417 ns (10 ns / 360 × –15.0). The source clock edge remains at 0.0 ns.

Phase Shift in Clock Reports

You can find clock phase shift information in the Clock Report, which you generate using the `report_clocks` command. When an MMCM/PLL clock is phase-shifted with `PHASESHIFT_MODE` set to `LATENCY`, the tool marks the auto-derived clock with attribute `S`.

In the Generated Clocks section of the clock report, you see the pin phase shift the tool accounts for in the MMCM/PLL insertion delay.

***Note*:** The report only includes the delay from the auto-derived clock phase shift. It doesn't include the phase shift the tool includes in the MMCM/PLL insertion delay.

In the following example below, the MMCM has `PHASESHIFT_MODE` set to `LATENCY`. The auto-derived clock `clk_out1_clk_wiz_0` has no phase shift defined for the MMCM pin `CLKOUT0`, while `clk_out2_clk_wiz_0` includes a –90 degree phase shift defined for `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 affect both setup and hold computations and slack.

Skew Definition

Clock skew is the difference in insertion delay between the destination clock path and the source clock path. You measure it from their common point in the design to the endpoint and startpoint sequential cell clock pins, respectively.

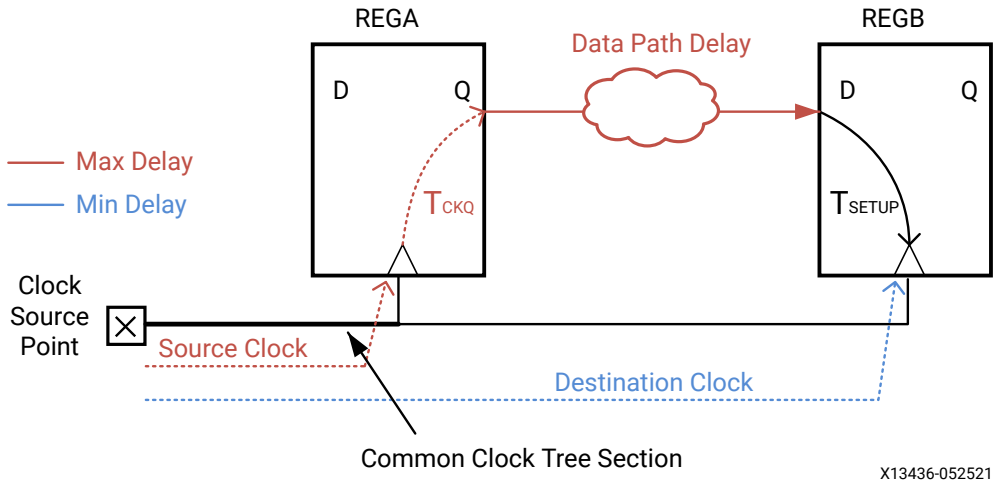
In the equation $Tskew_{i,j} = Tcj - Tci$, Tcj and Tci are defined as follows:

- **Tcj:** The delay from the common node to the endpoint clock pin.
- **Tci:** The delay from the common node to the startpoint clock pin.

Clock Pessimism Removal

A typical timing path report shows delay details for both source and destination clock paths, from their root to the sequential cell clock pins. As illustrated in the following figure, the tool analyzes the source and destination clocks with different delays, even on their common circuitry.

Figure 39: **Common Clock Tree Section**



This delay difference on the shared section introduces additional pessimism into the skew computation. To avoid unrealistic slack, the tool compensates for this pessimism using a delay called clock pessimism removal (CPR).

$$\text{CPR} = \text{common clock circuitry (max delay} - \text{min delay)}$$

The timing engine either adds or subtracts the CPR from the skew, depending on the type of analysis:

- **Max Delay Analysis (Setup/Recovery):** The tool adds CPR to the destination clock path delay.
- **Min Delay Analysis (Hold/Removal):** The tool subtracts CPR from the destination clock path delay.

The Vivado Design Suite timing reports include clock skew for each timing path. In the following example (from a hold analysis), the report provides:

- Destination clock delay (DCD)
- Source clock delay (SCD)
- Common pessimism removal (CPR)

```
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, CPR accuracy changes before and after routing. For example, consider a timing path where source and destination clocks are the same and driven by one buffer.

- Before routing, the tool treats the clock buffer output pin as the common point. In this case, CPR compensates only for pessimism from the clock root to the buffer output.

- After routing, the tool identifies the common point as the last shared routing resource between the source and destination clock paths in the device architecture. This point does not appear in the netlist, so CRP cannot be directly retrieved by subtracting clock delays from the timing report. The timing engine computes the CPR value based on device information not directly available to the user.

Optimistic Skew

AMD FPGA devices give you access to advanced clocking resources, including dedicated clock routing trees and clock modifying blocks (CMBs). Some CMBs can compensate for clock tree insertion delay using a phase-locked loop (PLL) circuit, which is present in PLL or MMCM primitives.

The amount of compensation depends on the insertion delay in the PLL feedback loop. In many cases, a PLL or MMCM drives several clock trees using the same type of buffer, including the feedback loop. Because the device can be large, the insertion delay across these clock tree branches doesn't always match the feedback loop delay.

You end up with over-compensated clocks when the feedback loop delay is greater than the source or destination clock delay. In this case, the sign of the CPR changes, and the tool removes skew optimism from the slack value. This adjustment ensures timing analysis doesn't introduce artificial skew at the common node clock paths.

Clock Uncertainty

Clock uncertainty is the total possible time variation between any pair of clock edges. It includes computed jitter, phase error, and any user-defined uncertainty from `set_clock_uncertainty`.

For primary clocks, define jitter using `set_input_jitter` and `set_system_jitter`. For clock generators such as MMCMs and PLLs, the tool calculates jitter from the source clock jitter and clock modifying block (CMB) configuration. For generated clocks, such as flop-based dividers, jitter matches the source clock.

IMPORTANT! *The AMD Vivado™ Design Suite supports clock-based uncertainty, not pin-based. If a clock propagates through multiple CMBs, like parallel `BUFGCEs`, while retaining the same clock name, its jitter remains constant, and is unaffected by resources in the clock's path and unaffected by the clock's fanout.*

The tool adds the user-specified uncertainty to its internal computation. For generated clocks (such as from MMCM, PLL, and flop-based clock dividers), the user-specified clock uncertainty does not propagate through the clock generator.

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:

- Reserving some margin during slack calculations to account for clock noise that can affect hardware behavior. Because the delay and jitter values are already conservative, AMD does not recommend adding extra uncertainty to guarantee hardware functionality.
- Over-constraining selected paths: You can use clock uncertainty to make paths related to specific clocks or clock pairs more restrictive during one or more implementation steps. This helps improve quality of results (QoR) and timing closure. When you apply clock uncertainty, the tool does not alter the clock waveforms or their relationships, so the rest of your timing constraints remain valid.

Pulse Width Checks

Pulse width checks are rule checks on signal waveforms as they reach hardware primitives after propagating through the device. These checks usually reflect functional limits set by the internal circuitry of the primitive. For example, the minimum period check on a DSP clock pin ensures that the clock driving a DSP instance doesn't exceed the maximum frequency tolerated by the DSP.

Pulse width checks do not affect synthesis or implementation. Their analysis must be performed at least one time before generating the bitstream like any other design rule check in the Vivado Design Suite.

A pulse width violation usually comes from an incorrect clock definition, failing pulse width or period checks. It can also result from excessive skew in the clock topology, failing `max_skew`. Review the AMD FPGA data sheet for your target device to understand the valid operating range for the primitive where the violation happens.

If the violation is due to skew, simplify the clock tree or place clock resources closer to the affected pins to reduce the skew.

Reading a Timing Path Report

The timing path report gives you the information needed to understand what causes a timing violation. The following sections describe the components of the timing path report.

The Timing Path Summary displays the most important information from the timing path details. You can review it to identify the cause of a violation without analyzing every detail of the path. The summary includes slack, path requirement, datapath delay, cell delay, route delay, clock skew, and clock uncertainty. It does not include any information about cell placement.

For more details on timing constraint terminology, timing analysis concepts, and how the tool determines slack and path requirements, refer to Timing Analysis Key Concepts.

Timing Path Summary

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 type.
  - Max delay analysis (setup/recovery) is as follows:

    ```
    slack = data required time - data arrival time
    ```

  - Min delay analysis (hold/removal) is as follows:

    ```
    slack = data arrival time - data required time
    ```

  Vivado calculates and reports the data required and arrival times in other sections of the timing path report.

- **Source:** This field shows the path startpoint and the source clock that launches the data. The startpoint is usually the clock pin of a sequential cell or an input port. When applicable, the second line shows the primitive type and the edge sensitivity of the clock pin. It also includes the clock name and the clock edge definition (waveform and period).
- **Destination:** This field shows the path endpoint and the destination clock that captures the data. The endpoint is usually the input data pin of a sequential cell or an output port. When applicable, the second line shows the primitive type and the edge sensitivity of the clock pin. It also includes the clock name and the clock edge definition (waveform and period).
- **Path Group:** This field shows the timing group that includes the path endpoint. The group usually comes from the destination clock. For asynchronous timing checks such as recovery and removal, the path appears in the `**async_default**` timing group. User-defined groups can also appear and are useful for reporting purposes.
- **Path Type:** This field shows the type of analysis used for the path.
  - Max: Uses maximum delay values to calculate the data path delay for setup and recovery analysis.
  - Min: Uses minimum delay values to calculate the data path delay for hold and removal analysis.

  The field also shows the timing corner used in the report: Slow or Fast.

- **Requirement:** This field shows the timing path requirement. When the startpoint and endpoint use the same clock or clocks with no phase shift, the requirement is typically the following:
  - One clock period for setup and recovery analysis
  - 0 ns for hold and removal analysis

For paths between two different clocks, the requirement equals the smallest positive difference between any source and destination clock edges. Timing exception constraints such as multicycle path, max delay, or min delay override this value.

For more information on how Vivado derives the timing path requirement from constraints, see Timing Paths.

- **Data Path Delay:** This field shows the accumulated delay through the logic section of the path. It does not include clock delay unless the clock is used as data. The delay type matches the analysis shown in the Path Type field.
- **Logic Levels:** This field shows the number of each type of primitive in the data path. It excludes the startpoint and 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):** This field shows the accumulated delay from the destination clock source point to the endpoint of the path.
  - For max delay analysis (setup/recovery), Vivado uses the minimum cell and net delay values
  - For min delay analysis (hold/removal), Vivado uses the maximum delay values
- **Source Clock Delay (SCD):** This field shows the accumulated delay from the clock source point to the startpoint of the path.
  - For max delay analysis (setup/recovery), Vivado uses the maximum cell and net delay values
  - For min delay analysis (hold/removal), Vivado uses the minimum delay values
- **Clock Pessimism Removal (CPR):** The extra skew introduced when source and destination clocks use different delay types on shared circuitry. Vivado removes this pessimism to eliminate skew on the shared path. In routed designs, the last common clock tree node usually lies in routing resources and does not appear in path details.
- **Clock Uncertainty:** This field shows the total possible time variation between any pair of clock edges. The value includes:
  - Computed clock jitter (system and discrete)
  - Phase error from hardware primitives
  - User-defined clock uncertainty from the `set_clock_uncertainty` constraint

  Vivado adds the user-defined clock uncertainty to the computed uncertainty from the timing engine.
- **Total System Jitter (TSJ):** This field shows the combined system jitter applied to both the source and destination clocks. To modify system jitter globally, use the `set_system_jitter` constraint. Virtual clocks are ideal and do not include any system jitter.

- **Total Input Jitter (TIJ):** This field shows the combined input jitter for the source and destination clocks. Use the `set_input_jitter` constraint to define input jitter for each primary clock. The Vivado IDE timing engine calculates the input jitter for generated clocks based on the master clock's jitter and the clocking resources it traverses. By default, virtual clocks are ideal and do not include jitter.

  For more details, see the Clock Latency Jitter and Uncertainty section in the *Vivado Design Suite User Guide: Using Constraints* (UG903).

- **Discrete Jitter (DJ):** This field shows the jitter introduced by hardware primitives such as MMCM or PLL. The Vivado IDE timing engine computes this value based on how these components are configured.

- **Phase Error (PE):** This field shows the amount of phase variation between two clock signals introduced by hardware primitives such as MMCM or PLL. The Vivado IDE timing engine automatically calculates this value and adds it to the overall clock uncertainty.

- **User Uncertainty (UU):** This field shows the additional uncertainty defined by the `set_clock_uncertainty` constraint. For usage details, see 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:** Appears only for AMD SSI devices and shows the margin required to safely report timing paths that cross super logic region (SLR) boundaries. Its presence depends on timing constraints, the reported path, and the target device.

- **Input Delay:** This field shows the input delay value set by the `set_input_delay` constraint on the input port. It does not appear for paths that do not start from an input port.

- **Output Delay:** This field shows the output delay value set by the `set_output_delay` constraint on the output port. It does not appear for paths that do not end at an output port.

- **Timing Exception:** This field shows the timing exception that applies to the path. Only the exception with the highest precedence appears, as it determines the timing path requirement.

  For details on timing exceptions and how precedence is applied, see Report Exceptions.

Timing Path Summary In Text Report

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

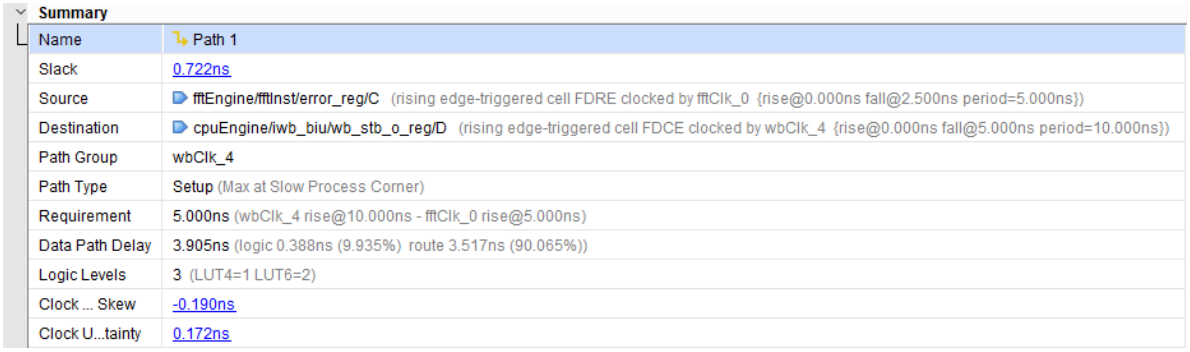
Figure 40: **Timing Path Summary Header in Text Report**

```
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
```

Timing Path Summary in Vivado IDE

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

Figure 41: **Timing Path Summary Header in Vivado IDE**



Timing Path Details

The second half of the timing path report gives you detailed information about the cells, pins, ports, and nets that the path traverses. This section is divided into three parts:

- **Source Clock Path:** This shows the circuitry the source clock traverses from its source point to the startpoint of the datapath. You do not see this section for a path that starts from an input port.
- **Data Path:** This shows the circuitry the data traverses from the startpoint to the endpoint.
- **Destination Clock Path:** This shows the circuitry the destination clock traverses from its source point to the clock pin at the datapath endpoint.

The Source Clock Path and Data Path sections operate together and use the same type of delay:

- Maximum delay for setup and recovery analysis
- Minimum delay for hold and removal analysis

These two sections share the accumulated delay, starting at the data launch edge time and continuing through both the source clock path and the data path.

The destination clock path always uses the opposite type of delay compared to the source clock and data paths. Its accumulated delay begins at the time when the data capture edge is launched at the source point of the destination clock. The final value is the data required time.

The final lines of the report summarize how the tool computes slack:

- 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

The following example shows the source clock, data, and destination clock paths in the text report. With a simple 5 ns period constraint, the tool sets the source clock launch edge at 0 ns and the destination clock capture edge at 5 ns.

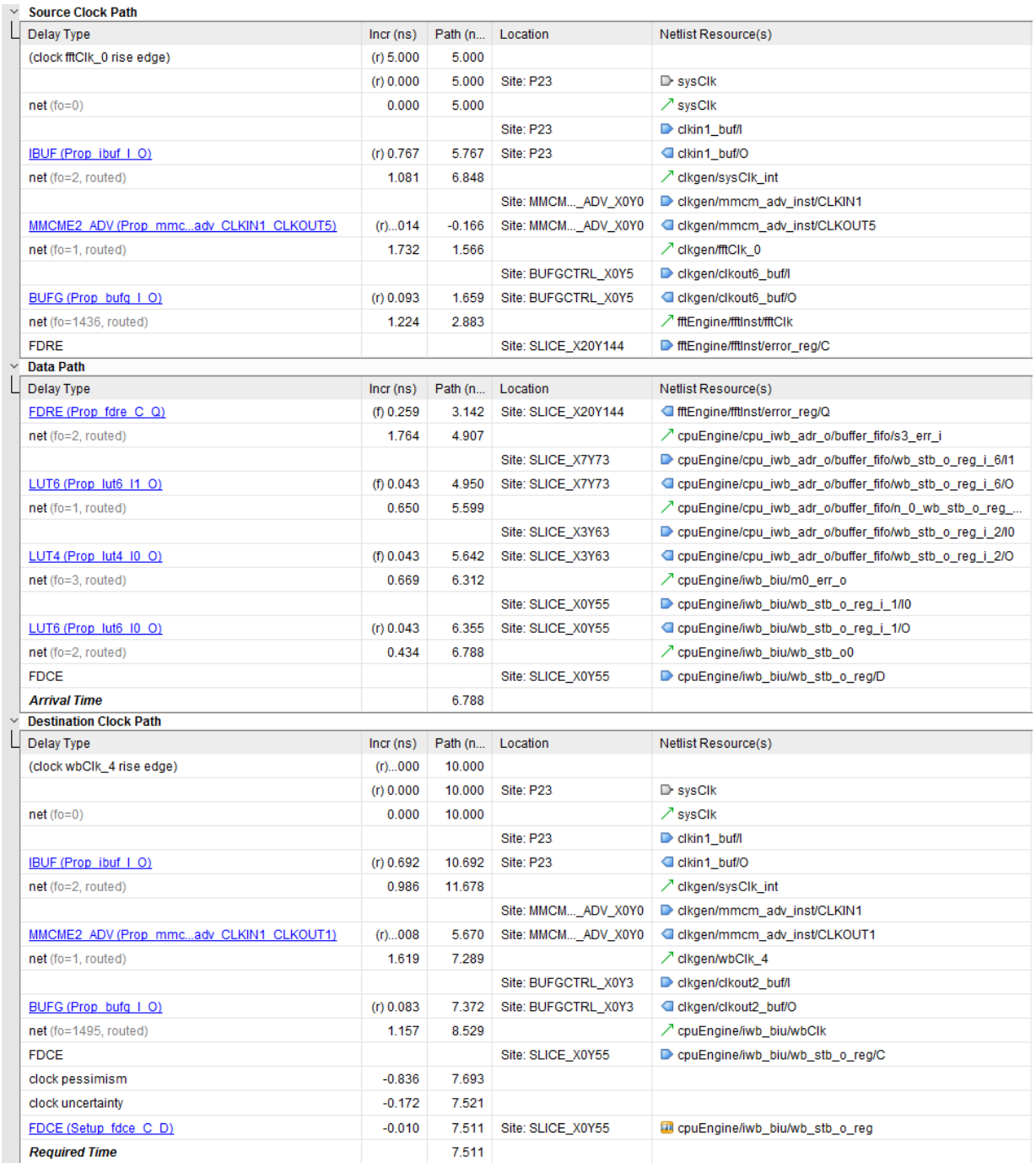
Figure 42: **Timing Path Details in Text Report**

```
  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, shown in the following figure, present the same information as the text report in the previous figure.

Figure 43: **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 tool displays the path information in five columns when you use the standard flow, or six columns when you use Incremental Compile:

- **Location:** Shows where the cell or port is placed on the device.

- **Delay Type:** Describes the UniSim primitive and the specific timing arc that the path follows. For a net, this column shows the fanout (fo) and its placement or routing status. A net can be Unplaced, Estimated, or Routed.
  - Unplaced: The driver and the load are not placed.
  - Estimated: The driver, the load, or both are placed. A partially routed net is also reported as estimated.
  - Routed: The driver and the load are both placed, and the net is fully routed.
- **Incr (ns) (text report) / Delay (IDE report):** Displays the incremental delay associated with a UniSim primitive timing arc or a net. It can also show the delay from a constraint, such as input/output delay or clock uncertainty.
- **Path (ns) (text report) / Cumulative (IDE report):** Displays the accumulated delay after each segment of the path. On a given line, this value equals the previous cumulative delay plus the current incremental delay.
- **Netlist Resource(s) (text report) / Logical Resource (IDE report):** Shows the name of the netlist object that the path traverses.
- **Pin Reuse (Incremental Compile only):** Indicates whether the path is reused from the reference run. Possible values include `ROUTING`, `PLACEMENT`, `MOVED`, and `NEW`.

Each incremental delay is associated with either rising or falling edge senses.

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

The tool determines the initial edge sense based on the launch or capture edge used for analysis. The edge sense can change if an element along the path inverts the signal. For example, a rising edge at the input of an inverter becomes a falling edge at its output. Use edge sense to identify tight path requirements from clock edge inversion along the source or destination clock tree.

Verifying Timing Signoff

Before reviewing timing analysis details, you need to understand which part of the timing reports confirms that your design is ready to run in hardware.

IMPORTANT! *Timing signoff is a required step when analyzing implementation results after placement and routing complete.*

By default, the Vivado Design Suite generates the text version of the Report Timing Summary during project runs. You can also generate this report manually after loading the post-implementation design checkpoint into memory.

IMPORTANT! *The Report Timing Summary does not include bus skew constraints. To report bus skew, run the* `report_bus_skew` *command separately from the command line. This command is not available in the GUI.*

For a comprehensive Timing Signoff Verification methodology, see Analyzing and Resolving Timing Violations in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) and the *Versal Adaptive SoC System Integration and Validation Methodology Guide* (UG1388).

Reports

This chapter explains the different types of reports you can generate.

- General Reports covers various netlist-based reports, along with power analysis and design rule checks (DRC).
- Timing Reports covers all reports related to timing constraints.
- Timing Closure Reports covers reports used in the design closure phase. Although they include both timing and netlist-based analysis, they focus on how these two aspects interact to create timing closure challenges.

This chapter describes reporting commands you run with Tcl, from the console on an open design or from a run script. For instructions on adding these commands to project runs, see Configurable Report Strategies.

TIP: *For more help with the command line and options for any reporting command, type* `report_name -help`. *For example,* `report_drc -help`.

General Reports

Report Utilization

The utilization report helps you analyze how your design uses resources at the hierarchical level, within user-defined Pblocks, or across super logic regions (SLRs). You can generate this report at various stages in the flow using the `report_utilization` Tcl command. For detailed syntax and options, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

The following details apply to AMD UltraScale™ and AMD UltraScale+™ families. Each report includes the target device and shows utilization for the following categories:

- Netlist Logic
  - LUT
  - MuxFx
  - Register

    - LUT as memory
    - LUT flip-flop pairs
    - LUT as logic
    - Carry logic

    Additional items might appear in each category depending on the design.
- CLB Distribution: Provides details on how the logical netlist maps to physical sites after placement. This includes:
    - Slice
    - LUT combining information:
        - Use of O5 and O6 for UltraScale+ and earlier
        - Dual output LUT for AMD Versal™
    - Packing details for LUT and register pair usage
    - Control sets
- BLOCKRAM
    - BlockRAM
    - UltraRAM
    - FIFO
- ARITHMETIC
    - DSP resources
- 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

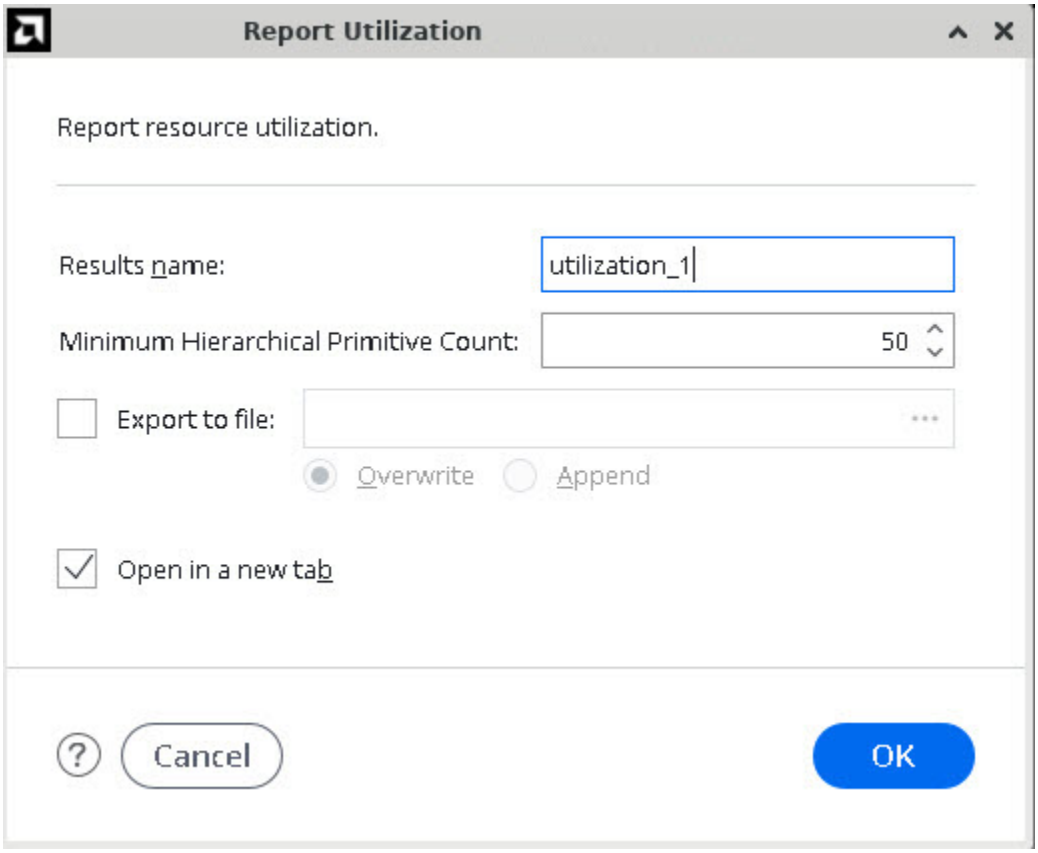
When you run the command from the Tcl Console, you can scope the resource usage for a specific hierarchical cell using the `-cells` option. When you run it from the AMD Vivado™ Integrated Design Environment (IDE), the tool displays this information in an interactive table.

Resource numbers might change throughout the flow as logic optimization commands modify the netlist.

Running Report Utilization

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

Figure 44: **Report Utilization Dialog Box**



Results Name Field

At the top of the Report Utilization dialog box, enter a name for the result window in the Results Name field.

The equivalent Tcl option is:

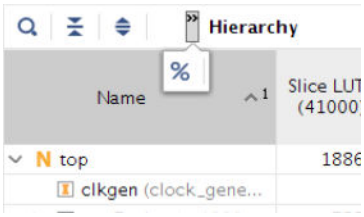
```
report_utilization -name utilization_1
```

The following figure shows the detailed utilization report.

Figure 45: **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	

Figure 46: **Report Utilization Percentage Toggling**



When you run `report_utilization` in the Vivado IDE, the tool uses Hierarchy mode. This mode reports utilization for each hierarchy in the design. The report includes only hierarchical cells that contain more than the number of leaf cells defined by the `-hierarchical_min_primitive_count` option. The default value is 50. To include all hierarchies, reduce this value to zero.

You can toggle between showing utilization numbers and percentages using the button in the report window.

Show Utilization for Specific Cells

Use the `-cells` option to show the utilization of specific cells and their children.

```
-cells {cell_name_list}
```

Use the following option to exclude specific cells from the targeted cell level:

```
-exclude_cells {cell_name_list}
```

Show Utilization for Specific Pblocks

When you use the `-pblocks` option, the utilization report shows details for the specified Pblock. You can specify only one parent Pblock. The report lists available resources based on the parent Pblock range and splits used resources into parent Pblock, child Pblocks, and non-assigned logic. This breakdown helps you evaluate how logic competes for resources within the parent Pblock.

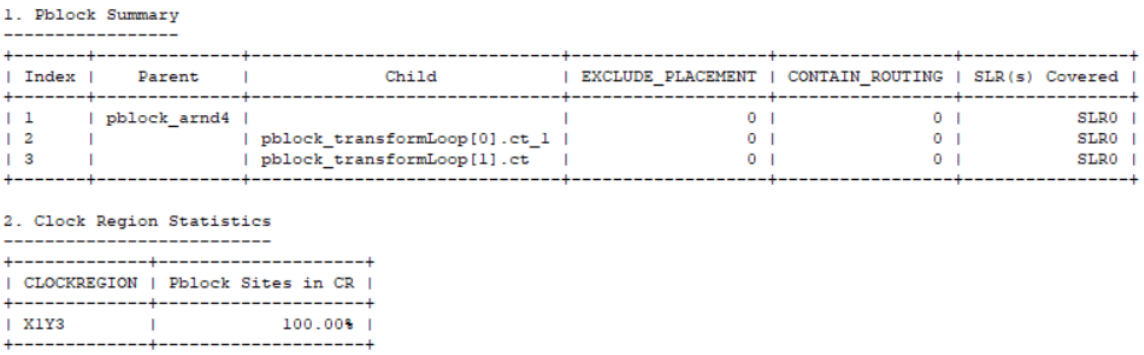
The following options are available only in Tcl mode:

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

When you include these options, the report adds two extra tables:

- The first table summarizes the parent and child Pblock properties, including SLR placement details.
- The second table shows clock region statistics for the specified Pblock.

Figure 47: **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% |
+-------------+--------------------+
```

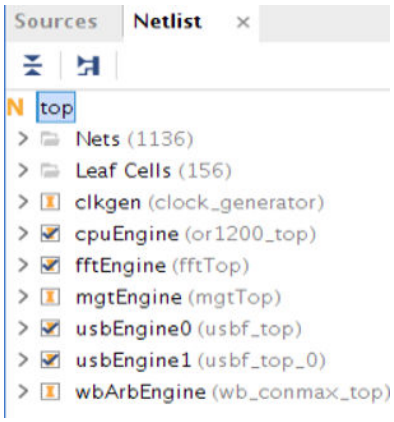
The pre-placement and post-placement utilization numbers can differ due to LUT combining and non-assigned cells that are not accounted for before placement.

When you use the `-pblocks` option, the utilization tables includes the following columns:

- **Parent:** Resources assigned only to the specified parent Pblock
- **Child:** Resources assigned only to child Pblocks of the specified parent
- **Non-Assigned:** Resources used within the specified Pblock area but not assigned to the parent or any child Pblocks
- **Used:** Total resources used in the specified Pblock area
- **Fixed:** Total resources fixed by LOC constraints in the area defined by the specified Pblocks
- **Prohibited:** Resources in the specified area that are blocked from use due to PROHIBIT constraints
- **Available:** Total resources available in the specified Pblock area
- **Util%:** Ratio of Used to Available

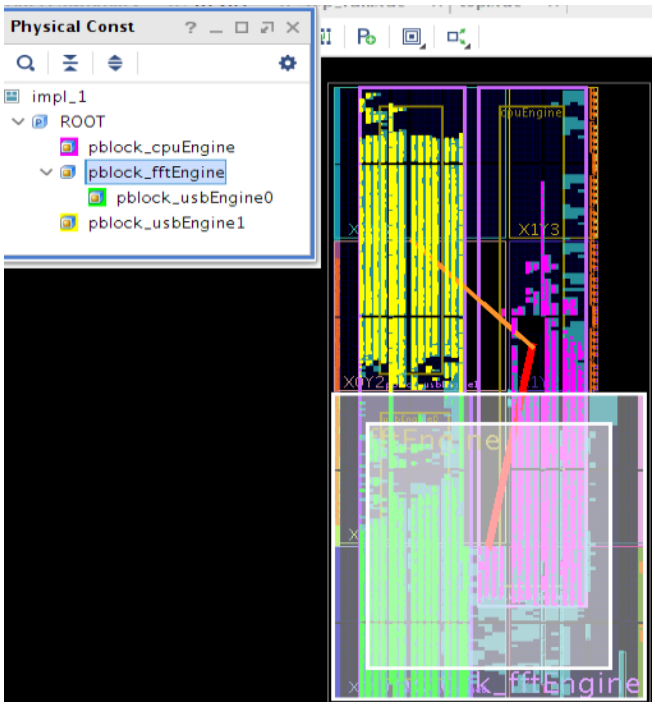
The following example illustrates how to interpret these columns using a sample design hierarchy.

Figure 48: **Example Design Hierarchy**



The following figure shows the Pblock rectangles, along with post-route netlist resource highlights inside each Pblock.

Figure 49: **Design Pblocks**



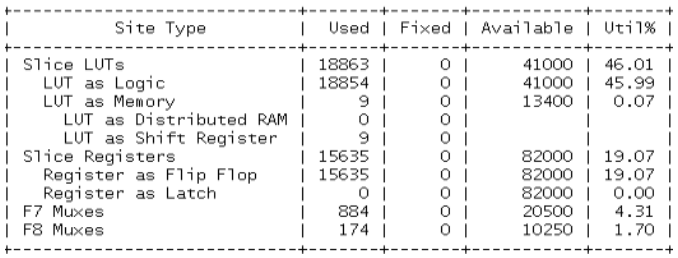
In this example, observe the following:

- The *pblock_usbEngine1* Pblock does not have a child Pblock.
- The *pblock_fftEngine* Pblock contains a child Pblock named *pblock_usbEngine0*.
- The *pblock_cpuEngine* Pblock overlaps with *pblock_fftEngine*.

Use the following command to generate a report for the entire design:

```
report_utilization
```

Figure 50: **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

Run the following command to generate a report for the *pblock_usbEngine1* Pblock:

```
report_utilization -pblocks pblock_usbEngine1
```

Figure 51: **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

Run the following command to generate a report for the `pblock_fftEngine` Pblock:

```
report_utilization -pblocks pblock_fftEngine
```

***Note*:** In this case, resources assigned to the nested child Pblock `pblock_usbEngine0` are included in the Used column.

***Note*:** If the `EXCLUDE_PLACEMENT` property is applied to the child Pblock, its resources are excluded from the parent Pblock in both the Used and Available columns.

The overlapping Pblock `pblock_cpuEngine` has partial cells placed in the range of `pblock_fftEngine`. These are reported as Non-Assigned because they come from an external source.

Figure 52: **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 specific Pblocks or non-assigned resources from the report, use the `-exclude_child_pblocks` or `-exclude_non_assigned` switch. The following command removes the Non-Assigned column from the report:

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

Figure 53: **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 how the `report_utilization` command behaves under different Pblock 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% is calculated as: 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>`	No EXCLUDE_PLACEMENT property on child Pblock	• **Non-Assigned:** Total cells placed within the parent Pblock bounds but not assigned to the parent or its child Pblocks • **Fixed:** Total cells fixed within the parent Pblock bounds • **Used:** Sum of parent, child, and non-assigned cells placed within the parent Pblock bounds • **Available:** Total physical resources within the parent Pblock bounds • **Util%:** Used/Available
		EXCLUDE_PLACEMENT property is set on the child Pblock. Reported area corresponds to parent Pblock ranges minus child Pblock ranges.	• **Non-Assigned:** Total cells placed within the reported area but not assigned to the parent or child Pblocks • **Fixed:** Total cells fixed within the reported area • **Used:** Only cells assigned to the parent Pblock (child Pblock cells are excluded) • **Available:** Physical resources in the reported area (parent minus child) • **Util%:** Used/Available
3	Report on overlapping Pblocks	Similar to the default Pblock report, except Available becomes the union of the reported Pblocks. EXCLUDE_PLACEMENT is ignored.	• **Available:** Union of the physical resources covered by the overlapping Pblocks • **Util%:** Used/Available

Show SLR Utilization

Vivado reports SLR utilization by default. When you use the `-slr` option, the report displays only the SLR-related utilization data. The SLR utilization section includes four tables:

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

For UltraScale families, the SLR connectivity table shows the number of SLR crossing registers used for both TX and RX directions on each SLR side. For other device families, the table shows only the number of crossings used, because those devices do not include dedicated crossing registers.

Figure 54: **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	

Grid vs Stacked SLR Crossing Reporting

For multi-SLR single column devices, the post-place estimate of SLR crossing resources is accurate. Some minor cell movement can occur across SLRs to improve timing, but the tool accounts for paths crossing multiple SLRs in each individual crossing segment. For example, a connection from SLR0 to SLR2 is treated as SLR0-SLR1-SLR2, and the tool includes a route in each segment.

For multi-SLR multi-column devices, the post-place estimate assumes that the router chooses adjacent SLRs when possible. However, multiple routing solutions can exist. For example, in a 2×2 SLR device, a connection from SLR0 to SLR1 can take either a direct route (SLR0-SLR1) or an indirect route (SLR0-SLR3-SLR2-SLR1). If adjacent SLR usage exceeds 100%, the tool might choose a slower alternate route to complete the connection.

Diagonal routes between SLRs, such as SLR0–SLR2 or SLR1–SLR3, are reported as one vertical and one horizontal segment.

Figure 55: **SLR Connectivity Report for a 2x2 SLR Crossing Place**

1. SLR Connectivity

	Used	Fixed	Available	Util%
Horizontal SLLs Used	18011		32400	55.59
SLR3 <-> SLR0	0		17112	0.00
SLR0 -> SLR3	0			0.00
SLR3 -> SLR0	0			0.00
SLR2 <-> SLR1	11		16200	0.07
SLR1 -> SLR2	0			0.00
SLR2 -> SLR1	11			0.07
SLR0 <-> SLR2	18000			
SLR2 -> SLR0	18000			
SLR0 -> SLR2	0			
SLR1 <-> SLR3	0			
SLR3 -> SLR1	0			
SLR1 -> SLR3	0			
Vertical SLLs Used	18003		34224	52.60
SLR3 <-> SLR2	0		17112	0.00
SLR2 -> SLR3	0			0.00
SLR3 -> SLR2	0			0.00
SLR1 <-> SLR0	3		17112	0.02
SLR0 -> SLR1	1			<0.01
SLR1 -> SLR0	2			0.01
SLR0 <-> SLR2	18000			
SLR2 -> SLR0	18000			
SLR0 -> SLR2	0			
SLR1 <-> SLR3	0			
SLR3 -> SLR1	0			
SLR1 -> SLR3	0			
Total SLLs Used	0		66624	

After routing is complete, the report shows only the actual routing used.

The SLR CLB Logic and Dedicated Block Utilization table displays the resource utilization for each SLR. You can toggle between utilization values and percentages using the % icon in the GUI to support SLR partition analysis.

Figure 56: **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 you select the following options, you can limit the utilization report to specific levels of hierarchy.

Use `-hierarchical_depth` to specify how deep the report analyzes the hierarchy. The default depth is 0, which means the tool reports only the top-level hierarchy.

The tool includes only those hierarchies that contain more leaf cells than the threshold set by `-hierarchical_min_primitive_count`. The default threshold is 50.

The following are the available options.

- **`-hierarchical`**: Enables reporting by hierarchy.
- **`-hierarchical_depth <value>`**: Sets how deep into the hierarchy the report explores
- **`-hierarchical_percentage`**: Displays utilization as a percentage within each hierarchy level.
- **`-hierarchical_min_primitive_count <integer>`**: Filters out hierarchies with fewer leaf cells than the specified number.

These options help you control the level of detail in hierarchical utilization analysis.

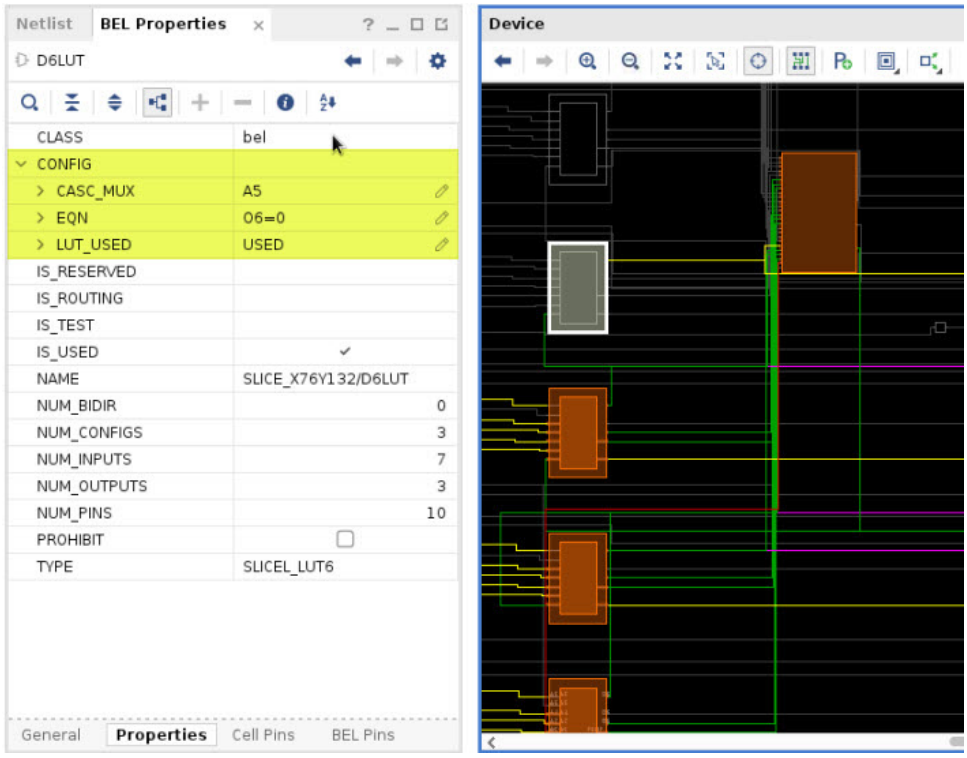
Showing Packthrus

Packthrus are used BELs that do not have an associated netlist cell. The tool typically adds them to legalize routing, allow access to otherwise unroutable elements, or simplify routing. You can include packthru information in the report by using the `-packthru` option.

A packthru LUT uses one output and up to one input. The rest of the LUT remains available for use if needed. A register functions as a packthru only when the site is not needed for any other purpose.

In the following example, a packthru LUT drives a logic 0 into the LOOKAHEAD BEL from the LUT's O6 output. The O5 output remains available, and the tool can map a smaller LUT to use this BEL.

Figure 57: **BEL Properties**



Show Customized Table Report

When you select the following options, you can customize the report to focus on specific resource types and control the hierarchy depth shown in the results:

- `-spreadsheet_table <args>`: Limits the report to specific resource types that you define.
- `-spreadsheet_depth`: Sets how deep into the hierarchy the report explores when displaying data in spreadsheet format.

Use these options to tailor the utilization report output for targeted analysis or presentation.

Showing Prohibit and Fixed Information

When you run `report_utilization` in the Tcl Console without the `-name` option, the text report includes prohibited sites and fixed cells on each line.

Prohibited sites affect how the tool calculates available resources, using the following formula:

$$\text{Available resources} = \text{total resources} - \text{prohibited resources}$$

A cell is considered fixed when it has the `IS_LOC_FIXED` property set to 1. This property is automatically applied when you assign a `LOC` constraint to the cell.

Report I/O

When you generate the I/O Report, the tool lists the following information:

- **Pin Number:** Lists all the pins in the device
- **Signal Name:** Shows the name of the user I/O assigned to each pin
- **Bank Type:** Indicates the bank type where the I/O is located (for example, `High Range`, `High Performance`, `Dedicated`)
- **Pin Name:** Displays the name of the pin
- **Use:** Describes the I/O usage type (for example, `Input`, `Output`, `Power/Ground`, `Unconnected`)
- **I/O Standard:** Shows the I/O standard for the user I/O; an asterisk (*) indicates the default standard. This field might differ from what is shown in the I/O ports window of the Vivado IDE
- **I/O Bank Number:** Identifies the I/O Bank where the pin is located
- **Drive (mA):** Displays the drive strength in milliamps
- **Slew Rate:** Shows the slew rate configuration of the buffer (`Fast` or `Slow`)
- **Termination:** Lists the on-chip or off-chip termination settings
- **Voltage:** Displays voltage values for relevant pins, including `VCCO`, `VCCAUX`, and related pins
- **Constraint:** Displays `Fixed` if you have constrained the pin
- **Signal Integrity:** Reports the signal integrity value for each pin

Report Clock Utilization

The Clock Utilization report helps you analyze how the design uses clocking primitives and routing resources at either the clock region level or the clock net level. You can use this report to debug clock placement issues and identify placement constraints that help maximize resource utilization.

The Clock Utilization Report provides the following information:

- The number of clocking primitives available and used, along with their physical constraints
- The timing clock name and period associated with each clock net
- Clocking and fabric load utilization for each clock region
- The number of loads on each clock net in each clock region

In the Vivado IDE, the Clock Utilization Report also supports selection of netlist and device objects, allowing you to highlight placement details and generate schematics.

Report Clock Utilization Tables

The report presents the clocking topology and placement details grouped into the following categories:

- Clock primitives used in the design
- Global clock resources
- Global clock source details
- Local clock resources
- Clock region-specific clock and load information
- Placement details for global clocks

Because netlist object names are often long and designs can include many clock nets and clocking elements, the report assigns short IDs for easier navigation.

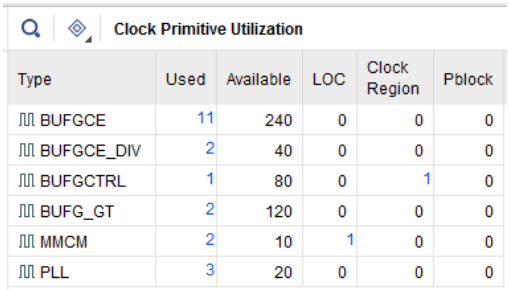
- A unique global ID (`g<n>`) for each net driven by a clock buffer
- A unique source ID (`src<n>`) for each clock generator, such as an MMCM or input buffer
- A unique local ID (`<n>`) for clock nets that do not use global clock routing

These IDs help you quickly locate specific clock nets throughout the report. When available, the original netlist object names appear in the final columns of each table.

Clock Primitive Utilization Table

The Clock Primitive table shows a summary of how each clock primitive type is used, along with any physical constraints applied to those primitives.

Figure 58: **Clock Primitive Utilization Table**



Clock Primitive Utilization

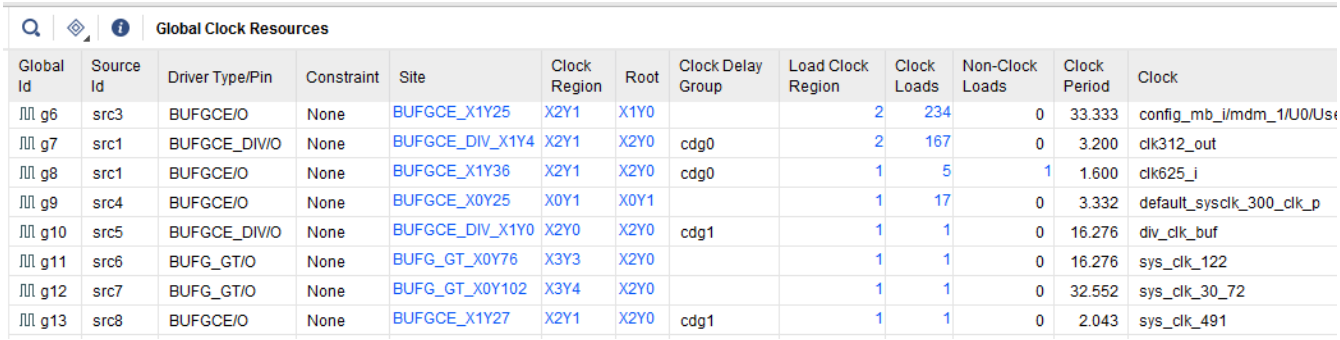
Type	Used	Available	LOC	Clock Region	Pblock
BUFGCE	11	240	0	0	0
BUFGCE_DIV	2	40	0	0	0
BUFGCTRL	1	80	0	1	0
BUFG_GT	2	120	0	0	0
MMCM	2	10	1	0	0
PLL	3	20	0	0	0

***Note*:** Clock region constraints do not apply to 7 series devices.

Global Clock Resources Table

The Global Clock Resources table provides a summary for each clock net, including key constraint and placement details, as shown in the following figure.

Figure 59: **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 table includes the following columns:

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

Column	Description
Global Id	Unique identifier for each global clock net
Source Id	Identifier for the clock-generating primitive connected to the clock buffer
Driver Type/Pin	Primitive pin that drives the clock net
Constraint	Highest-priority user-defined physical constraint applied to the clock buffer, following this order: 1. `LOC` 2. `CLOCK_REGION` (not applicable to 7 series) 3. `PBLOCK`
Site	Location of the clock buffer, set by you or by the Vivado implementation tools
Clock Region	Device clock region where the buffer is located. Does not apply to 7 series.

Table 9: **Global Clock Resources Table Details** *(cont'd)*

Column	Description
Root	Device clock region where the clock net `CLOCK_ROOT` is placed (not applicable to 7 series)
Clock Delay Group	User-defined group of clock nets to enforce matching of the clock routing (not applicable to 7 series)
Load Clock Region	Number of clock regions containing loads driven by the clock net
Clock Loads	Number of cells connected through clock pins
Non-Clock Loads	Number of cells connected through non-clock pins, such as CE pins on FDCE elements
Clock Period	Period in nanoseconds of the timing clock on the net; if multiple clocks propagate, the smallest period is shown
Clock	Name of the timing clock on the net; if multiple clocks propagate, the report displays `Multiple`
Driver Pin	Logical name of the clock driver pin
Net	Logical name of the clock net segment connected to the driver pin
`GCLK_DESKEW`	Indicates if deskew calibration is enabled (Versal only)
`CLOCK_EXPANSION_WINDOW`	Reports the `CLOCK_EXPANSION_WINDOW` property after placement (Versal only)
Clock Low Fanout	Indicates if the `CLOCK_LOW_FANOUT` property is applied

Global Clock Source Details Table

The Global Clock Source Details table shows how each clock generator output connects to global clocks and provides timing clock information. The following figure illustrates how the outputs of an MMCM instance (src0 and src1) connect to clock buffers. In the example, the `CLKOUT2` output of `src1` drives two global clocks: `g7` and `g8`.

Figure 60: **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 table includes the following columns:

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

Column	Description
Source Id	Identifier of the clock-generating primitive
Global Id	Identifiers of the global clocks driven by the source pin
Driver Type/Pin	Output primitive pin that generates the clock

Table 10: **Global Clock Source Details Columns** *(cont'd)*

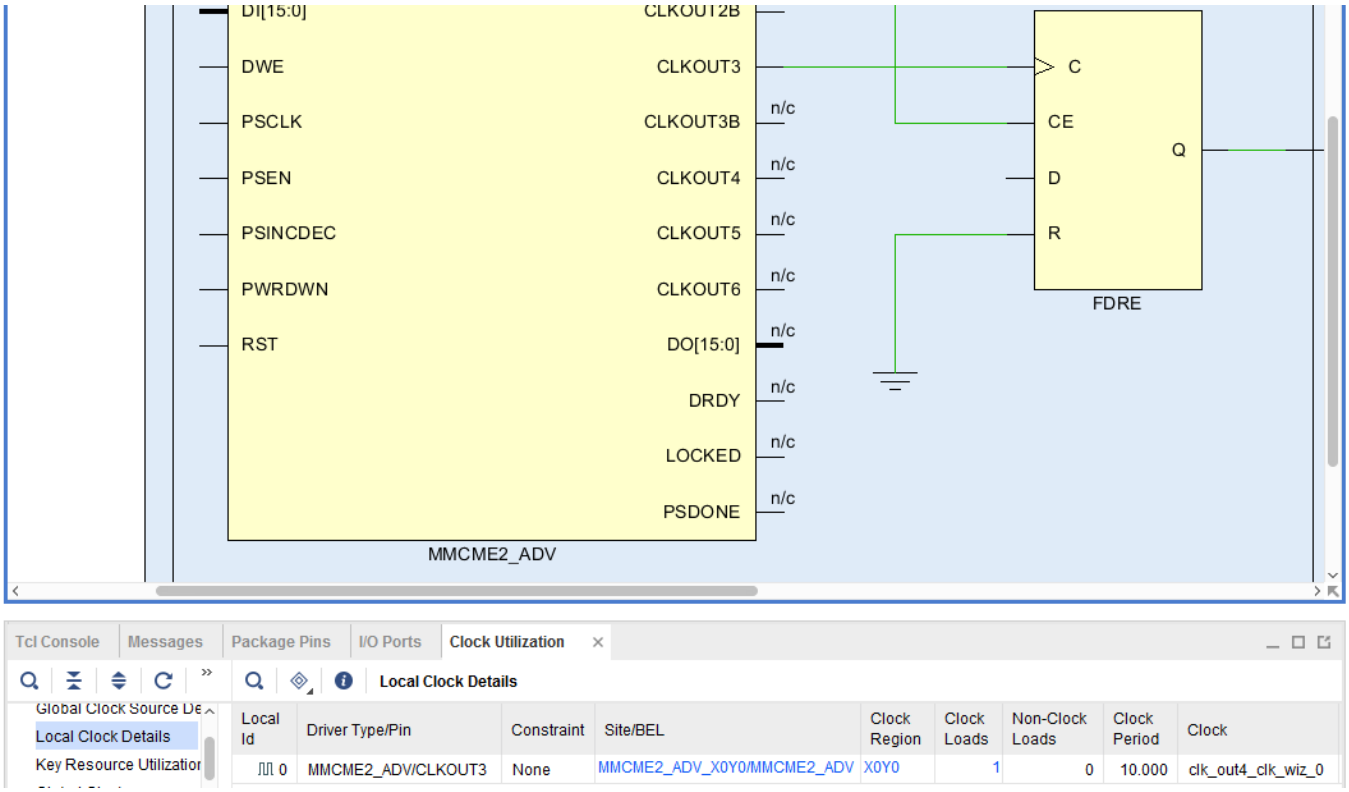
Column	Description
Constraint	Highest-priority user-defined constraint applied to the clock buffer, using the following order: 1. `LOC` 2. `PBLOCK`
Site	Clock source location set by you or by the Vivado implementation tools
Clock Region	Clock region of the device where the source is located
Clock Loads	Number of clock pin loads connected to the source pin
Non-Clock Loads	Number of non-clock pin loads connected to the source pin, such as CE pins on FDCE elements
Source Clock Period	Period in nanoseconds of the timing clock generated by the source pin. If multiple clocks propagate, the report shows the smallest period
Clock	Name of the timing clock generated by the source pin. If multiple clocks propagate, the report displays `Multiple`
Driver Pin	Logical name of the clock source pin
Net	Logical name of the clock net segment connected to the source pin

Local Clock Details Table

The Local Clock Details table appears only when the design includes local clocks. A local clock is a clock net that uses regular fabric routing resources instead of global clock routing. This typically occurs when a clock net is not driven by a clock buffer. The information in this table is similar to what you see in the Global Clock Resources table.

The following figure shows an example where a local clock net, driven by the output of a 7 series MMCM, connects directly to a register clock pin (FDRE/C).

Figure 61: **Local Clock Example**

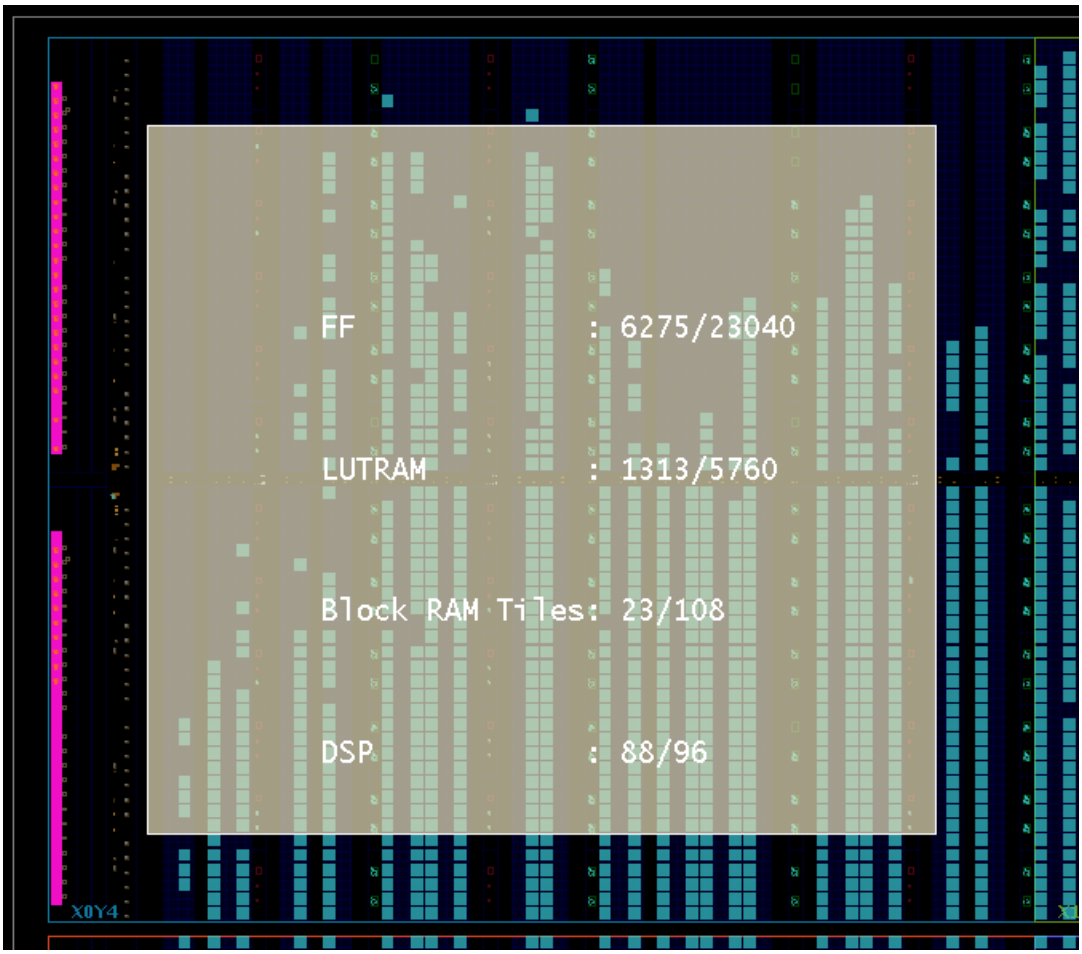


Clock Regions Tables

The Clock Regions section is available only for UltraScale device families. It includes several tables that show the design uses primitive and routing resources in each clock region.

In the Clock Utilization window, you can click the **Show Metrics In Device Window** button to select which resource types to display for each clock region in the Device view, as shown in the following figure.

Figure 62: **Clock Region Resource Utilization Metrics in the Device Window**



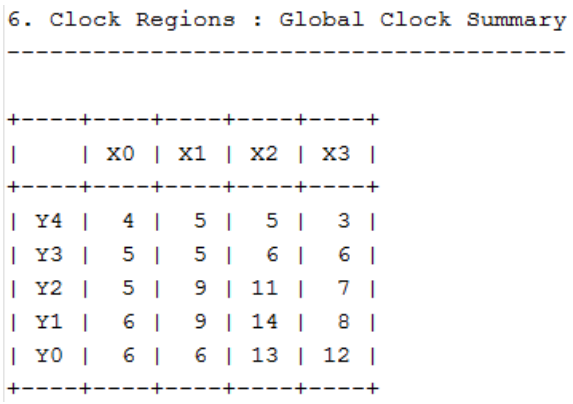
The Clock Regions section contains the following tables:

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

For both tables, the Global Clock column shows the number of global clock nets routed on the horizontal distribution layer, regardless of whether they drive loads in the region. Clock nets routed only on the vertical distribution layer, without branching into the horizontal layer, are not included. Nets routed entirely on the general-purpose routing layer are also excluded.

The global clock summary displays the number of global clocks present in each clock region, formatted to match the device's clock region floorplan. This table is available only in the text report. The Routing Resource Summary lists the global clock routing resources used in each clock region, broken down by resource type and location.

Figure 63: **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

Key Resource Utilization Table

The Key Resource table is available only for 7 series devices. It provides the same type of information you find in the Clock Regions tables for UltraScale devices, combined into a single view.

The Global Clock Summary table is also available only in the text report.

Global Clocks Tables

The Global Clocks tables show the type of loads present in each clock region for every global clock net. These tables also include timing clock information and the clock track ID used to route each clock net.

When you sort the table by Global ID in the Vivado IDE, you can identify the clock regions where each global clock net is routed. You can also highlight those regions in the Device view by selecting the corresponding table rows.

The column descriptions match those used in the Clock Primitive, Global Clock Resources, and Global Clock Source Details tables.

The Global Id of any clock net that crosses a clock region without driving loads in that region is marked with a plus sign (+), as shown in the following figure.

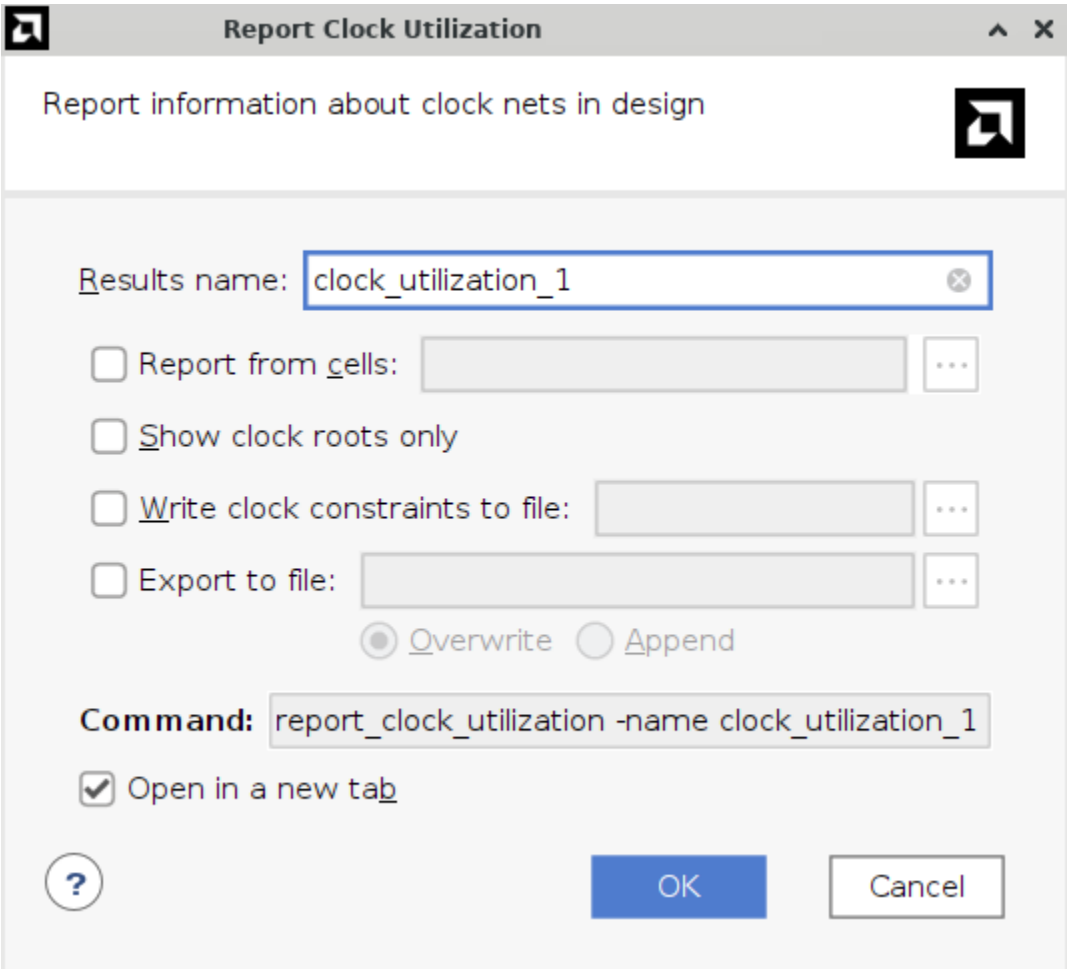
Figure 64: **Clock Region Cell Placement Example**

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

1. In the Vivado IDE, select **Reports → Report Clock Utilization**.



2. In the Results Name field, enter a name for the graphical report window.
   - The equivalent Tcl command option is:

```
report_clock_utilization -name clock_utilization_1
```

3. (Optional) Select **Show Clock Roots Only** to limit the Global Clock Resources table to the clock root location for each clock net. The report does not include the full source, load, or timing clock details.
   - The equivalent Tcl command option is:

     ```
     -clock_roots_only
     ```

4. (Optional) Enable **Write Clock Constraints to File**, then enter a file name. This exports the clock source and load constraints based on the current placement in memory.
   - The equivalent Tcl command option is:

     ```
     -write_xdc <filename>
     ```

5. (Optional) Enable **Export to File** to save the report to a file in addition to displaying it in the GUI.
   - Enter the file name in the provided field or click **Browse** to select a directory.
   - The equivalent Tcl command option is:

     ```
     -file <filename>
     ```

6. (Optional) Choose how to handle the output file:
   - Select **Overwrite** to replace the file if it exists.
   - Select **Append** to add the new results to the end of the file.
   - The equivalent Tcl command option is:

     ```
     -append
     ```

7. Click **OK** to generate the report.

Report DRC

Report DRC runs common design rule checks (DRCs) to look for common design issues and errors.

IMPORTANT! *Review the Critical Warnings. The severity of a particular check might increase later in the flow.*

Elaborated Design

The tool runs DRCs on the elaborated design to check for issues related to the following:

- I/O
- Clock placement
- Potential coding problems in your HDL
- XDC constraints

Because the RTL netlist excludes many I/O buffers, clock buffers, and other primitives found in post-synthesis designs, the tool checks fewer conditions at this stage. Elaborated design DRCs are less comprehensive than those run later in the flow.

Synthesized Design and Implemented Design

DRCs become more comprehensive as your design moves through the implementation flow. They also build as you progress through the flow. You can run post-synthesis checks after placement and routing. The following are examples of checks:

- After synthesis:
  - Netlist connectivity checks
  - Missing pin connections
- During placement:
  - Clock network checks
  - Location related checks
  - Utilization threshold checks
- During routing:
  - Routing requirement checks
    - For example, the cascade pin must be driven by another cascade pin
- After routing:
  - Bitstream checks to limit any device damage such as I/Os without a LOC property

DRCs have four levels of security: Info, Warning, Critical Warning, and Error. At this stage, Critical Warnings do not block the design flow, but any DRC violation with an error-level severity stops the flow.

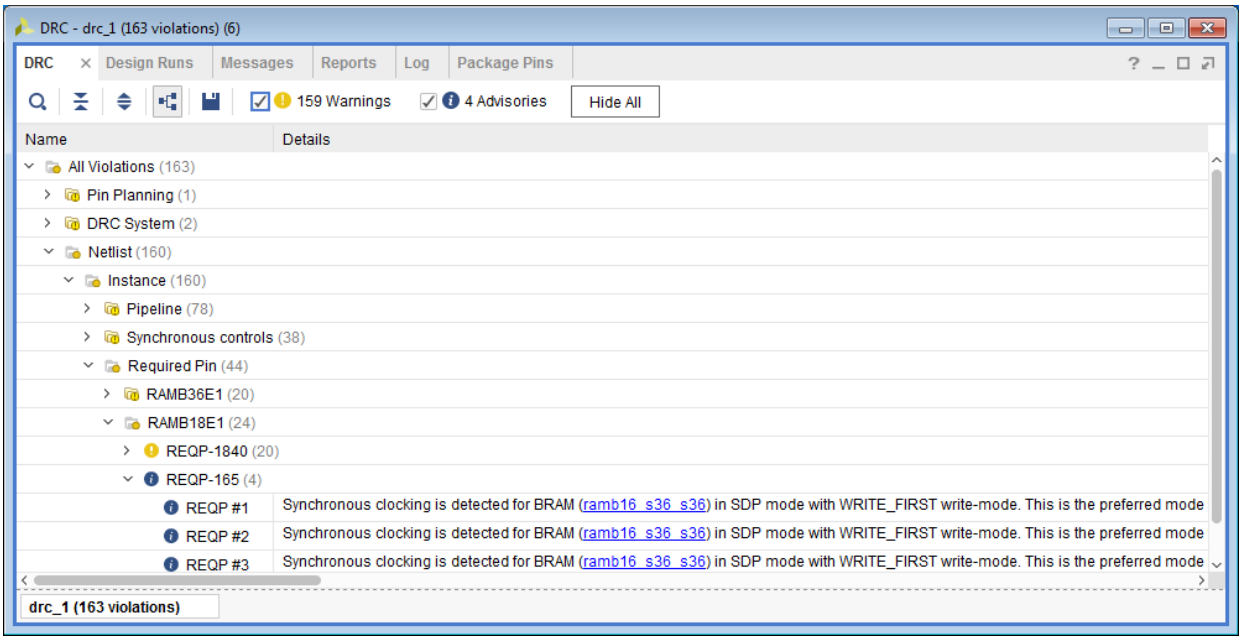
Steps of the implementation flow also run the DRCs, which can stop the flow at critical points. The placer and router check for issues that block placement. Certain messages have a lower severity depending on the stage. These are DRCs flagging conditions that do not stop `opt_design`, `place_design`,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

Report DRC checks that the user has constrained the package pin location and the I/O standard for every design ports. If these constraints are missing, `place_design` and `route_design` issue critical warnings. However, these DRCs appear as an ERROR in `write_bitstream`. The tools do not generate a bitstream 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.

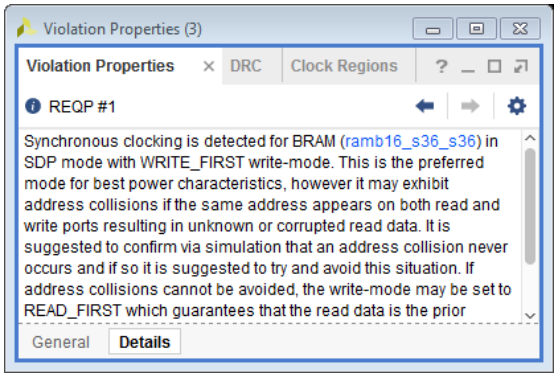
The following figure shows the Vivado IDE graphical user interface form of Report DRC.

Figure 65: **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 66: **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`.

Report Route Status

You can generate the Route Status Report during the implementation flow by using the `report_route_status` Tcl command.

This report gives you a breakdown of the nets in your design, including:

- Total number of logical nets
  - Number of nets that do not need routing resources
  - Number of nets that use only internal routing (within a tile, such as inside a CLB, Block RAM, or I/O pad)
  - Number of nets without loads, if any
  - Number of routable nets that require routing resources
  - Number of unrouted nets, if any
  - Number of fully routed nets
  - Number of nets with routing errors
  - Number of nets with some unrouted pins, if any
  - Number of nets with antennas or islands, if any
  - Number of nets with resource conflicts, if any

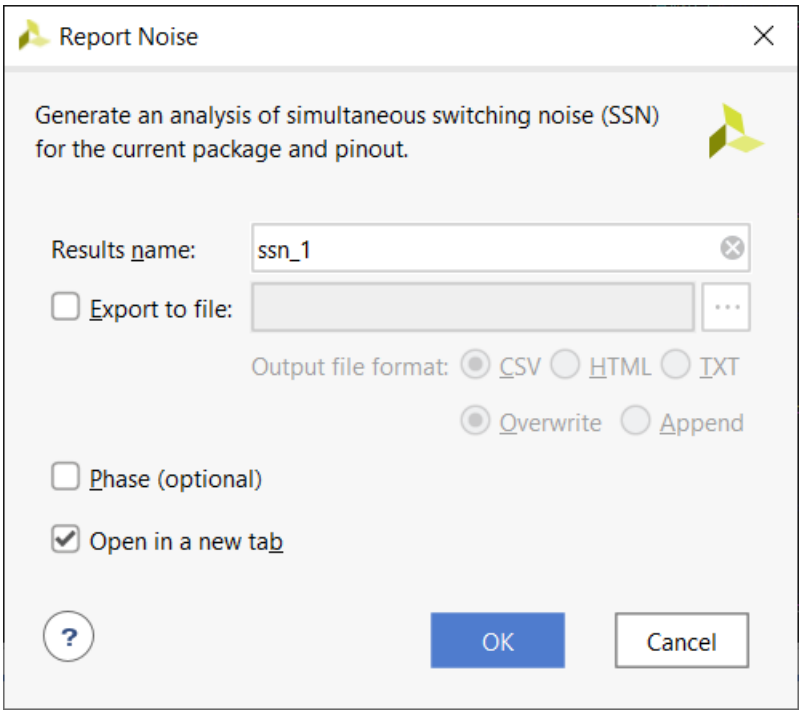
  Here is an example of a Route Status Report for a fully routed design:

```
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

Use the report noise command to perform the simultaneous switching noise (SSN) calculation for AMD. The noise report opens in a new tab in the lower pane of the Vivado IDE. You can export the results to a CSV or HTML file.

Figure 67: **Report Noise Dialog Box**

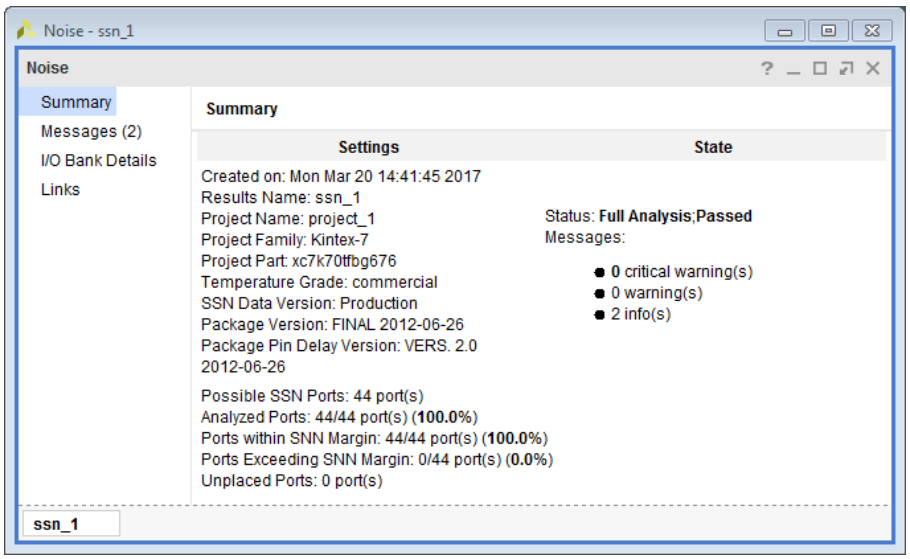


The noise report has four sections:

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

Noise Report Summary Section

Figure 68: **Noise Report**



The summary section of the noise report shows the following:

- The time and date the report ran
- The number and percentage of applicable ports analyzed
- The overall status, including whether the design passed
- The count of Critical Warnings, Warnings, and Info messages

Noise Report Messages Section

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

Noise Report I/O Bank Details Section

The I/O Bank Details section lists the pins, I/O standards, and remaining noise margin for each bank.

Noise Report Links Section

The Links section of the Noise Report contains links to documentation located on the AMD Support page.

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

Generate the Power Report after routing to analyze power consumption based on the device's operating conditions and the design's switching rates. Power analysis requires a synthesized netlist or a placed and routed design.

Use the following instructions to prepare for power analysis:

- Use `set_operating_conditions` to define the device's operating environment
- Use `set_switching_activity` to specify the switching behavior of the design

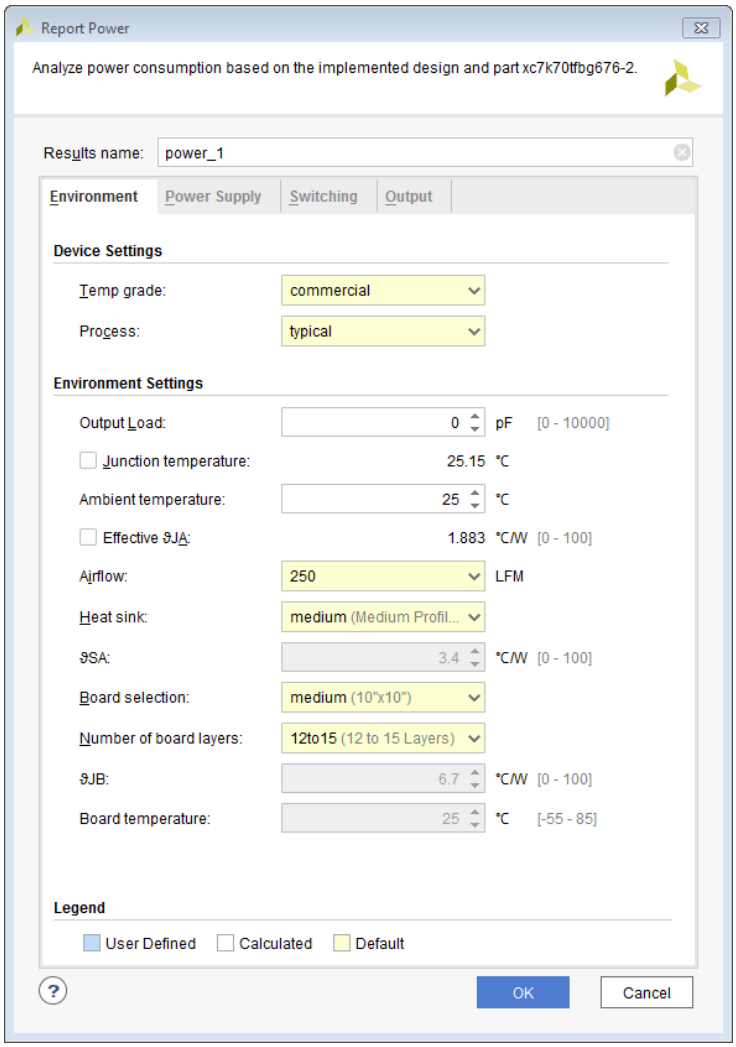
You can run the report power command when a synthesized design or an implemented design is open.

The Power Report estimates power consumption using inputs, such as the following:

- Thermal statistics, including junction and ambient temperature values.

  ***Note*:** You can set the junction temperature using the `-junction_temp` option of the `set_operating_condition` command. If not specified, the tool calculates it based on your design inputs.
- Board selection details, including number of board layers and board temperature
- Airflow and heatsink profile information
- FPGA current requirements by power supply source
- Power distribution analysis to support power-saving strategies
- Simulation activity files to improve power estimation accuracy

Figure 69: **Report Power Dialog Box**

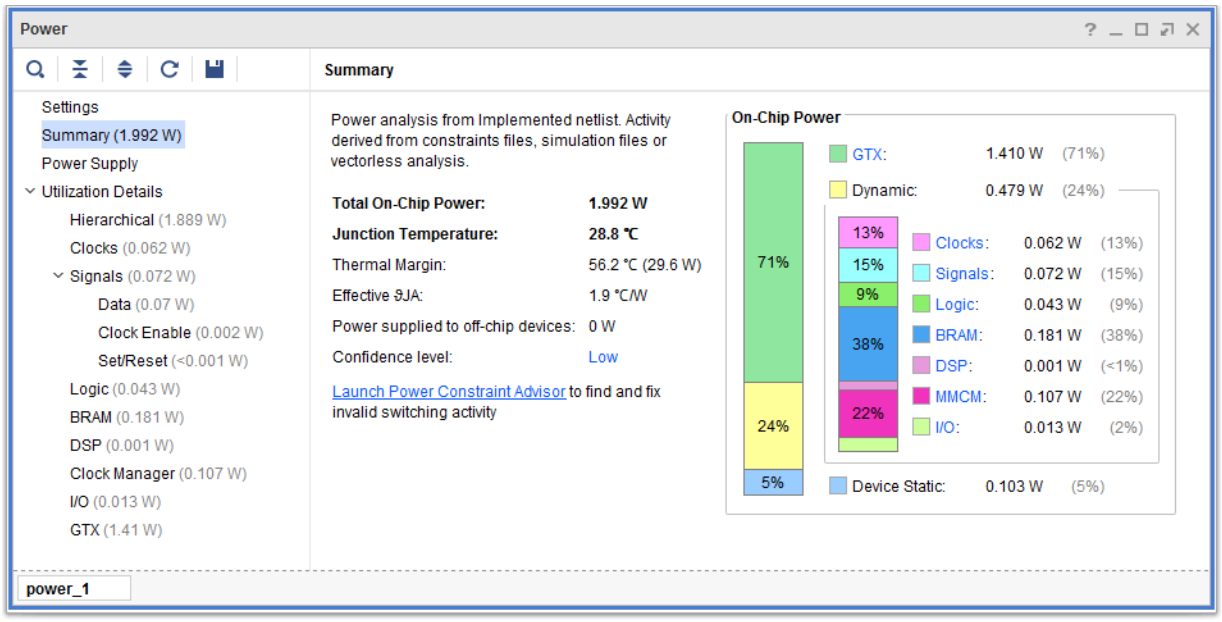


Analyzing the Power Report

Use the Report Power dialog box to view power data by the following:

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

Figure 70: **Power Report**



In Project mode, a text version of the power report is generated automatically after route during implementation. For more information, see the *Vivado Design Suite User Guide: Power Analysis and Optimization* (UG907).

Report RAM Utilization

The Ram Utilization report helps you analyze the usage of dedicated RAM blocks such as UltraRAM and block RAM and 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 FPGA implementation.

The report shows the following information:

- The total bit, width, and depth utilization of each memory primitive
- The size of the array and the dimensions (inference only)
- The type of memory
- Optional pipeline usage of the memory (where applicable)
- 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 runs 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 generates 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
```

Report Control Sets

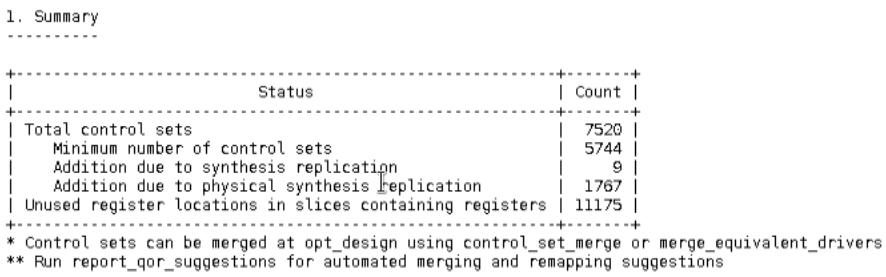
A control set is the unique combination of a clock signal, a clock enable signal, and a set/reset signal. Each slice supports a limited number of control sets, and the allowed combinations vary by architecture. Some sharing is possible within a slice, but you must understand the configurable logic block (CLB) architecture of your target device to follow compatibility rules.

The Control Sets report highlights the following key areas:

- **Absolute control set count:** Each device supports a finite number of control sets. Exceeding the recommended count can negatively affect quality of results (QoR).
- **Load profile of control sets:** When you need to reduce control sets, target those with low load counts. These introduce the least overhead when merged or optimized.

The following is an example of the Control Sets report summary.

Figure 71: **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

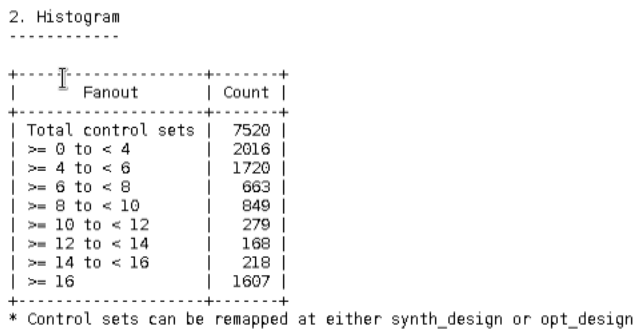
* Control sets can be merged at opt_design using control_set_merge or merge_equivalent_drivers
** Run report_qor_suggestions for automated merging and remapping suggestions

Nets replicated during synthesis tend to overlap and increase routing resource usage. Nets replicated by physical synthesis generally overlap less and can be ignored when counting control sets.

If your design exceeds the recommended control set limit, do the following:

- Optimize control sets with the lowest basic element of logic (BEL) load counts.
- Use the histogram summary to assess distribution and identify reduction opportunities.

Figure 72: **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

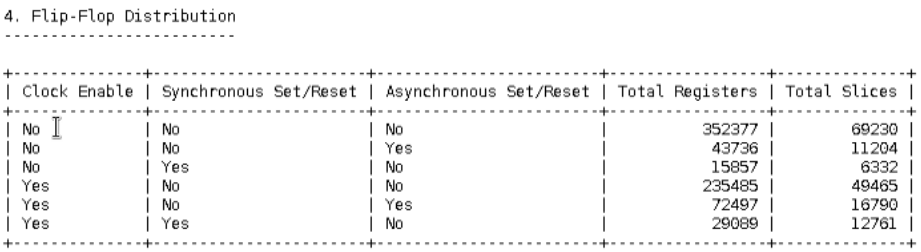
* Control sets can be remapped at either synth_design or opt_design

You can refine the report using the following switches:

- `-hierarchical` and `-hierarchical_depth`: scope control set data by design hierarchy
- `BLOCK_SYNTH.CONTROL_SET_THRESHOLD`: re-target control sets at specific hierarchy levels

The report also shows the types of flip-flop distribution in use. Vivado cannot re-target asynchronous resets.

Figure 73: **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

To view a complete list of all control sets, use the `-verbose` switch. This adds the following fields per 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

Refer to *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for guidance on recommended control set usage.

Report High Fanout Nets

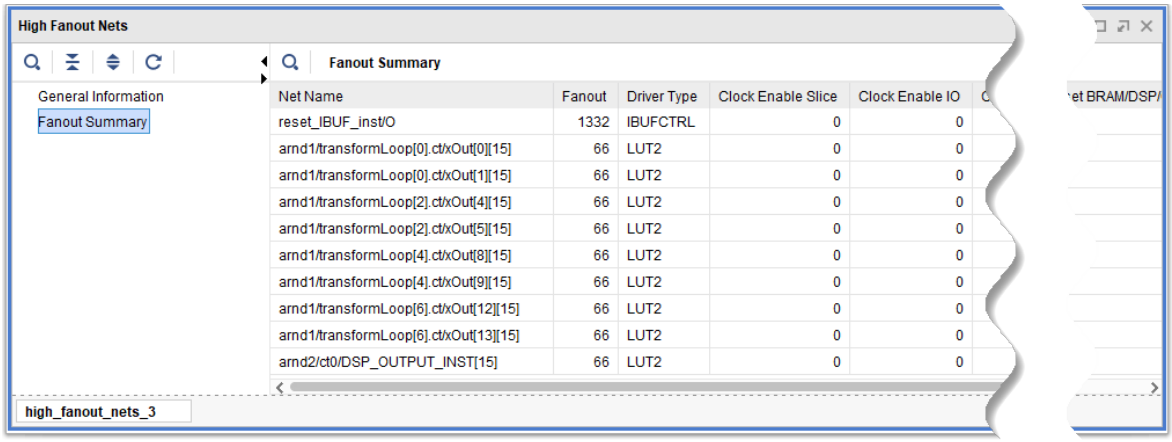
Use the `report_high_fanout_nets` command to analyze nets with the highest fanout in your design. You can run this command on a post-synthesized, placed, or routed netlist. Before placement, the report does not include clock region and SLR information.

The report shows the following:

- Fanout
- Driver type
- Load types
- Clock regions (if applicable)
- SLRs (if applicable)

The following figure shows an example of the report generated.

Figure 74: **High Fanout Net Report**



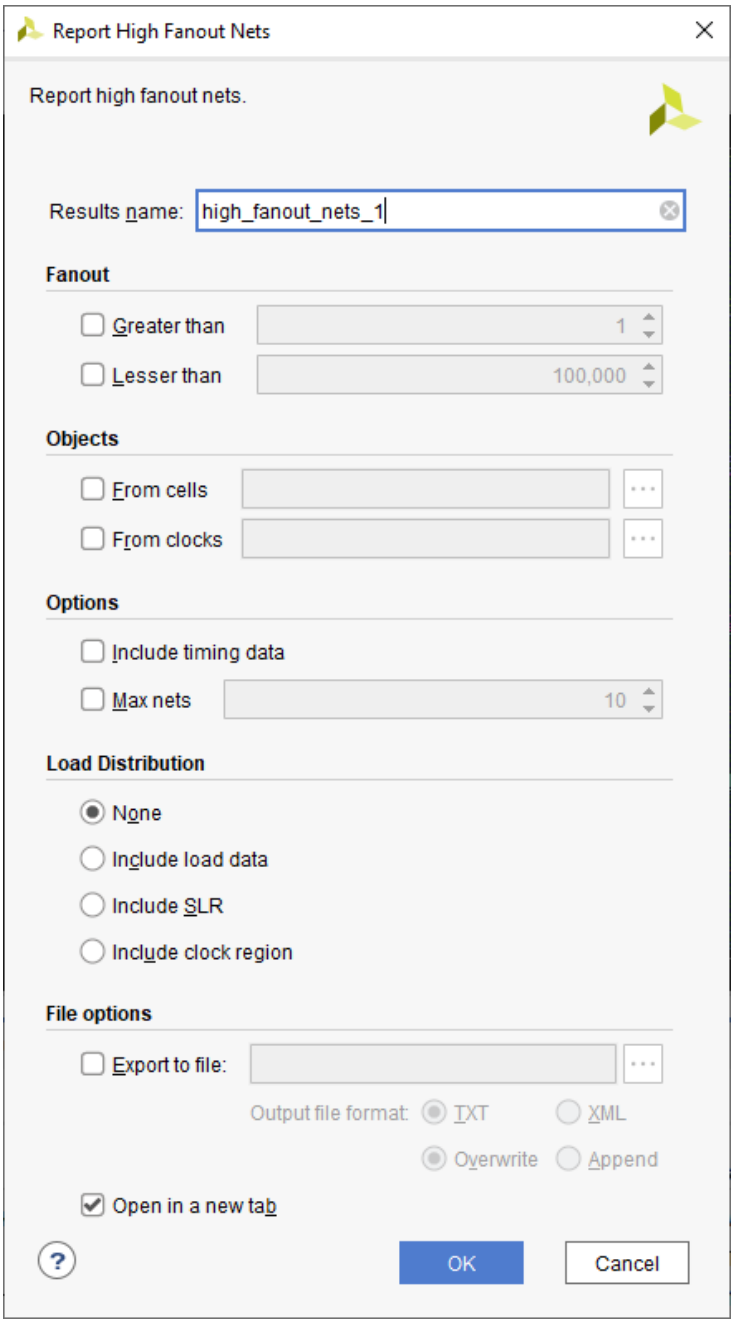
Generating the High Fanout Net Report

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

The equivalent Tcl command option is:

```
report_high_fanout_nets -name hfn_1
```

Figure 75: **Report High Fanout Nets Dialog Box**



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.

The equivalent Tcl command option is:

```
-name <windowName>
```

Fanout

You can set minimum and maximum limits to only report nets within a fanout range.

The equivalent Tcl command options where <n> is the number you specify, are:

```
-fanout_greater_than <n>
```

```
-fanout_lesser_than <n>
```

Objects

Hierarchical cells:

You can limit the the High Fanout nets report by hierarchical cell objects. Only nets with sources inside the specified cell are reported.

The equivalent Tcl command option is:

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

Clocks:

By default clocks are excluded from the high fanout report. Use `-clocks` to add reporting of clocks to the report.

The equivalent Tcl command option is:

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

Max Nets

By default, the report includes 10 nets. You can change this number by using the `-max_nets` switch. To include the worst-case slack for each net, add the `-timing` switch to the command.

The equivalent Tcl command options are:

```
-max_nets <n>
```

```
-timing
```

Load Distribution

You can add load types, clock regions, and SLRs to the report. To include all three, run the command multiple times, each with a different switch.

The equivalent Tcl command options are:

```
-load_types
```

```
-clock_regions
```

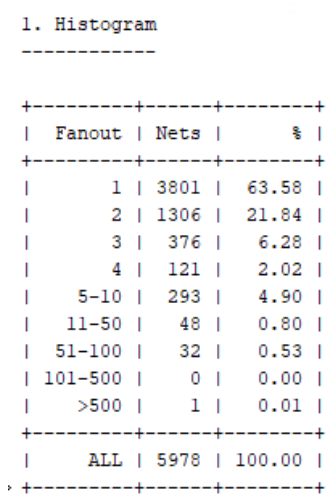
```
-slr
```

Histogram

You can display a histogram that shows fanout counts and the number of nets in each bucket. Use this histogram to determine whether a high number of lower fanout nets could pose a problem. This table is available only in text mode.

To generate it, use the `-histogram` switch.

Figure 76: **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

The equivalent Tcl command option is:

```
-histogram
```

***Note*:** You can also view fanout in timing reports for each net. To check fanout directly, use the `FLAT_PIN_COUNT` property on the net.

To address high fanout issues, refer to Optimizing High Fanout Nets and High Fanouts in Critical Paths in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) or the *Versal Adaptive SoC System Integration and Validation Methodology Guide* (UG1388).

Report SLR Crossing

Run the super logic region (SLR) crossing report to conduct a detailed analysis of the SLR crossing nets and their timing paths. Only run it on placed designs that use parts with more than one SLR. You can use it to review the following:

- Utilization of super long lines (SLLs) between different SLRs
- Utilization of different sites in each SLR
- Timing performance per clock on paths that cross SLR
- Nets with high fanout that cross SLR and their load distribution in each SLR
- The design hierarchy of the drivers to see which hierarchies are on the boundary of a crossing and the driver's primitive type

For information on the following tables, refer to Show SLR Utilization from `report_utilization`.

1. SLR connectivity
2. SLR connectivity matrix
3. SLR CLB logic and dedicated block utilization

SLR Crossings by Logic Level

You use the SLR Crossing by Logic Level table to analyze the performance of SLR crossings on timing paths. The table presents the data on a clock-by-clock basis. It contains the following information:

- The clock group name based on endpoint clock
- The requirement of the paths listed in the row. A clock group can have more than one row if there are different requirements for the timing paths caused by for example, a timing exception.
- The WNS of the crossing SLR paths. This might not be the WNS of the clock group as a whole.
- Calibrated deskew usage
- The clock vtree template used
- The clock root
- The clock expansion window
- Logic level bins

The following is an example of the report table with the clock information columns removed :

```
4. SLR Crossings by Logic Levels
--------------------------------

+--------+-------------+--------+-------------++------+------+
| Clock  | Requirement | WNS    | GCLK_DESKEW || 0    | 1    |
+--------+-------------+--------+-------------++------+------+
| clka   | 8.000       | 0.650  | CALIBRATED  ||  434 |    0 |
| clka   | 4.000       | 1.054  | CALIBRATED  ||   63 |  125 |
| clkc   | 3.000       | 0.022  | Off         || 5131 |  205 |
+--------+-------------+--------+-------------++------+------+
```

***Note*:** Because this report is timing-path based, count *n* paths that cross an SLR on the same SLL when the fanout is greater than one. The number of timing paths is adjusted based on the number of crossings, but you might not capture every timing path that crosses an SLR. The timing paths used in the report are ordered by WNS with the worst slack paths being used.

For AMD UltraScale™ and AMD UltraScale+™ designs, first evaluate the Requirement and WNS columns to determine whether the timing requirements are both realistic and achieved. If the requirements are realistic but timing is not met, use the Logic Levels section to gain insight into possible issues. In that case, evaluate the other tables to examine the design partition.

For AMD Versal™ designs, when WNS is less than 0.000, add the additional step of reviewing the reported clocking structures. Evaluate whether you can optimize the clocking structure for SLR crossings. Perform this step before reviewing any partitions. See *Versal Adaptive SoC System Integration and Validation Methodology Guide* (UG1388) for more information.

You can conduct further analysis on the specific paths by following the instructions provided in the table footer.

SLR Crossing by Fanout

The SLR Crossing by Fanout table shows the fanout of nets that cross between SLRs. It is ordered based on highest fanout first. It gives the following information:

- The total fanout of the net
- Driver cell type
- The SLR load distribution and an indicator of SLR driving region
- The net name

The following shows an example of a report:

```
5. SLR Crossings by Fanout
--------------------------

+--------+-------------+------+------+------+------+------------+
| Fanout | Driver Type | SLR0 | SLR1 | SLR2 | SLR3 | Net Name   |
+--------+-------------+------+------+------+------+------------+
|    378 | FDRE        |    0 |   9* |  369 |    0 | net_name1  |
+--------+-------------+------+------+------+------+------------+
* Indicates driver region
```

In this example, the driver is in SLR1, as indicated by the *. There are nine loads in the same SLR, and 369 loads in SLR2, giving a total fanout of 378. The net is called `net_name1`.

When your net has a high fanout, you can experience some difficulty optimizing for both fanout and SLR crossing. This depends on a combination of the design structure (for example, whether the net is driven by a register such as FDRE or FDCE), the timing requirement, and the tool options you select. You can resolve most issues by increasing the effort levels for `place_design` and `phys_opt_design`, and by using `FORCE_MAX_FANOUT` properties to further replicate.

Timing penalties incur on every timing path that has the source in one SLR and the load(s) in another. As long as the paths are timed, the challenge increases when more loads are in a different SLR from the driver, compared to a net with the same fanout profile but with most loads placed in the same SLR.

To find timing information on the net, use the `report_timing` command. For example:

```
report_timing -name slr_xing_hfn -through [get_nets net_name1] -setup
```

Note: See the *Versal Adaptive SoC System Integration and Validation Methodology Guide* (UG1388) for guidance on using properties to replicate the nets based on SLR load placement.

SLR Crossing by Hierarchy

The SLR Crossing by Hierarchy table shows you the hierarchy of the driving cells for nets that cross an SLR, along with the driving primitive type. You can use it to identify hierarchies that source a large number of crossings, helping you formulate actions that have the maximum impact.

The following is an example of the report:

```
6. SLR Crossings by Hierarchy of Driver
---------------------------------------
+--------------+--------+-------+------------+--------+-----------------+
| # Crossings | DCMAC  | FDCE  | GTME5_QUAD | SRL16E | Hierarchy       |
+--------------+--------+-------+------------+--------+-----------------+
|      20 (0) |  8 (0) | 5 (0) |      5 (0) |  2 (0) | TOP             |
|      18 (1) |  8 (1) | 5 (0) |      5 (0) |  0 (0) |   lev1A         |
|     17 (17) |  7 (7) | 5 (5) |      5 (5) |  0 (0) |     lev1A/Lev2A |
|       2 (2) |  0 (0) | 0 (0) |      0 (0) |  2 (2) |   lev1B         |
+--------------+--------+-------+------------+--------+-----------------+
```

Each line in the table represents a hierarchy level in your design. The number *not* in parenthesis indicates the total number of crossing nets in the current hierarchy and any levels below it.

In this design, there are 20 crossing nets: 18 from lev1A and two from lev1B. Of the crossing nets within lev1A, 17 are sourced from the lower-level hierarchy lev1A/lev2A (as indicted by the number in parenthesis), and one is sourced from within lev1A.

You can use the table to identify primitives that are undesirable drivers for SLR crossings. In high-performance designs, drivers of SLR crossings are typically registers (such as FDRE or FDCE) because they can be easily replicated and provide the fastest timing profile. Other primitive types can lack placement flexibility or replication capability, which limits the tool's ability to resolve timing problems on SLR crossing paths.

***Note*:** Follow the instructions in the footnote to further examine the nets driven by a given primitive in the schematic or netlist windows.

Generating the SLR Crossing Report

You can run the `report_slr_crossing` command in Tcl only. Run it after placement on designs that use devices with more than one SLR.

To generate a basic report with default settings and write it to the file `slr_crossing.rpt`.

```
report_slr_crossing -file slr_crossing.rpt
```

The `-file` option can be combined with any option.

The SLR Crossing by Fanout table can be limited by both the `-max_nets <n>` and `-fanout_greater_than <m>` switches.

- The `-max_nets <n>` switch limits the number of reported nets when the number of nets exceeding the minimum fanout specified is higher than max nets value. (Default: 20)
- The `-fanout_greater_than <m>` switch limits the number of nets reported when the number of nets meeting the meeting the fanout threshold is lower than the value of max nets. (Default: 10)

For example, to generate a report for nets with fanout greater than 100 and limit the output to 20 nets:

```
report_slr_crossing -fanout_greater_than 100 -max_nets 20
```

The `-cells <list_of_hierarchical_cell_names>` switch limits the report to a given hierarchy list and the hierarchies below each cell specified. When you run in this mode:

- For tables SLR Connectivity, SLR Connectivity Matrix, and SLR CLB Logic and Dedicated Block Utilization, the report generates one table per cell.
- For tables SLR Crossings by Logic Levels, SLR Crossings by Fanout, and SLR Crossings by Hierarchy of Driver, information is combined for all specified cells.

The following example generates the report for cells `cellA` and `cellB`:

```
report_slr_crossing -cells [get_cells [list cellA cellB]]
```

Additional options include the following:

- **`-append`:** Append to an existing file. By default, the command creates a new file and overwrites it if it exists.
- **`-quiet`:** Suppress any errors generated during report generation.
- **`-return_string`:** Return the report as a string that you can pass to a Tcl variable. By default, the report is written to the Tcl Console.

Timing Reports

Report Timing

Use the Report Timing command to view specific timing paths at any point after synthesis. This report helps you investigate timing issues flagged in the Report Timing Summary or verify the coverage and validity of specific timing constraints. It does not cover pulse width checks.

You can scope the report to one or more hierarchical cells using the `-cells` option. When scoped, the report includes only paths whose datapath starts, ends, crosses, or is fully contained within the specified cells.

Running Report Timing

If a design is already loaded in memory, you can run Report Timing from one of the following:

- The menu: Select **Reports → Timing → Report Timing**

- The Clock Interaction Report:
  1. Select a from/to clock pair
  2. Right-click and select **Report Timing**
- A Paths List:
  1. Select a path
  2. Right-click and select **Report Timing**

     The equivalent Tcl option is `report_timing`.

When you select specific options in the dialog, Vivado shows the equivalent Tcl syntax in the following areas:

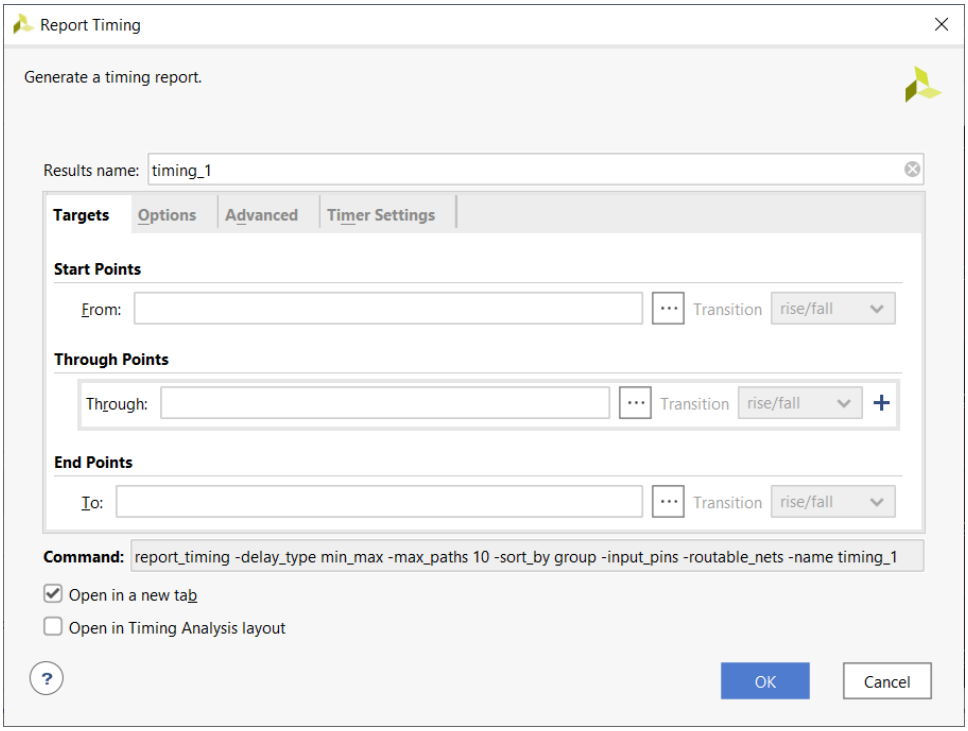
- The Command field at the bottom of the dialog box
- The Tcl Console after you run the command

Report Timing Dialog Box

Targets Tab

Use this tab to filter timing paths based on path structure.

Figure 77: **Report Timing Dialog Box: Targets Tab**



Report Timing gives you several filtering options that you must use to report a specific path or group of paths. These filters are based on the structure of each timing path. You can define the startpoints, through points, and endpoints to narrow the scope of the report and focus on the paths you need to analyze.

- **Startpoints (From):** List sequential cell clock pins, sequential cells, input ports, bidirectional ports, or source clocks. You can specify multiple objects. The equivalent Tcl command options are: `-from`, `-rise_from`, `-fall_from`
- **Through Points (Through):** List pins, ports, combinational cells, or nets. You can list multiple objects and use multiple `-through` filters to specify order. The equivalent Tcl command options are: `-through`, `-rise_through`, `-fall_through`
- **Endpoints (To):** List destination points such as input pins of sequential cells, output ports, bidirectional ports, or destination clocks.The equivalent Tcl command options are: `-to`, `-rise_to`, `-fall_to`

Options Tab

The Options tab contains the following settings:

- **Reports:** Select the path delay type. See Report Section for more information.

  Enable **Do not report unconstrained paths** to exclude unconstrained paths.

  The equivalent Tcl command is `-no_report_unconstrained`.

- **Path Limits:** Set the number of paths per group or endpoint. See Report Timing Summary for more information.

  Use `-group` to limit paths to specific groups.

  The equivalent Tcl command option is `-group`.

- **Path Display:** Filter by slack value using one of the following:

  - `-slack_greater_than`
  - `-slack_lesser_than` (Report Timing Summary)

  Select the number of significant digits. See Report Timing Summary for more information.Sort paths by group or by slack:

  - When you sort by group, the report shows the N worst paths for each group and each analysis type (`-delay_type min`, `max`, or `min_max`). Groups are ordered based on their worst path, with the group containing the most severe violation shown first.
  - When you sort by slack, the report shows the N worst paths across all groups for each analysis type. These paths are sorted by increasing slack.

  The equivalent Tcl command option is `-sort_by`.

Advanced Tab

The Advanced tab contains the same options available in Report Timing Summary.

Timer Settings Tab

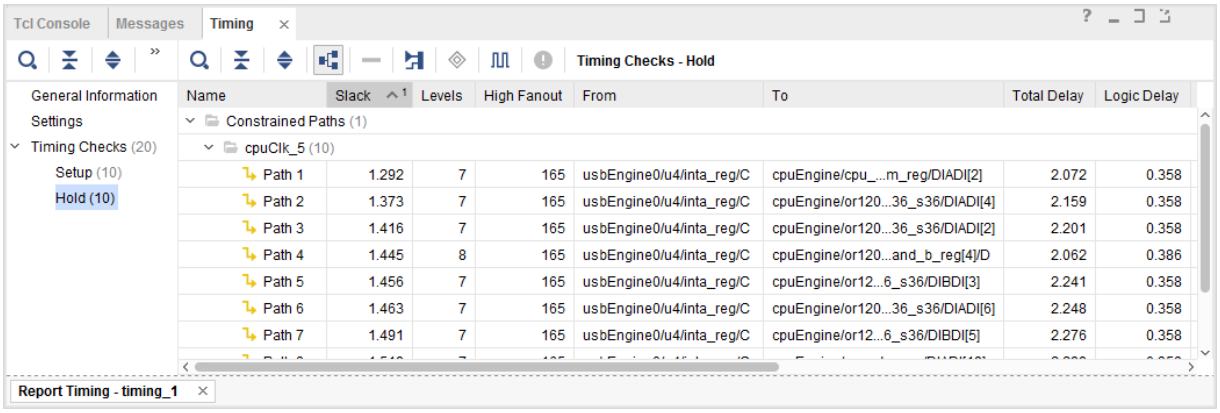
The Timer Settings tab matches the options in Report Timing Summary.

Reviewing Timing Path Details

After you click **OK** to run the report command, a new window opens showing the timing report. You can review the N-worst paths reported for each selected analysis type (min, max, or min_max).

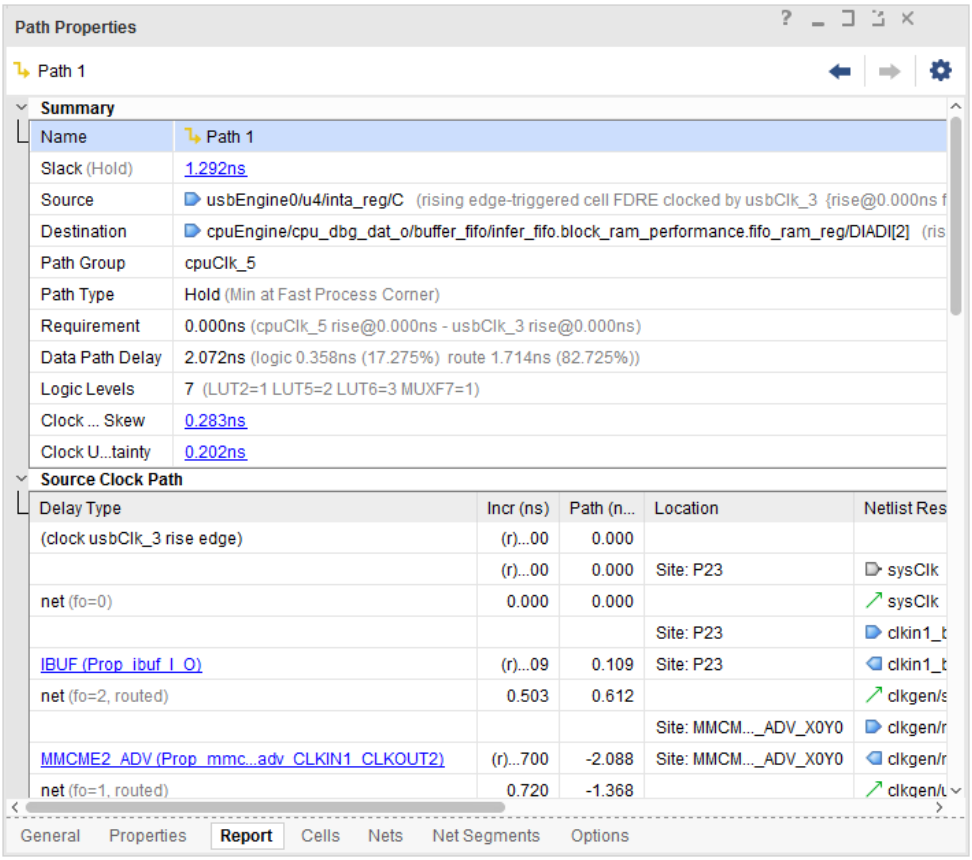
The following figure shows the Report Timing window in which both min and max analysis (SETUP and HOLD) were selected, and N=4.

Figure 78: **Report Timing Paths List**



Double-click a path to view more details in the Path Properties window (Report tab).

Figure 79: **Timing Path Properties Window**



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 for the same startpoint/endpoint
- Highlight the path in the Device and Schematic windows.

Filtering Paths with Violation

Failing paths show slack values in red. To focus only on these, click **Show only failing checks** mode.

Report Timing Summary

Timing analysis is available at any point in the flow after synthesis. You can review the Timing Summary report files that are automatically generated during the Synthesis and Implementation runs.

Use the following instructions to generate a Timing Summary report:

1. Load a synthesized or implemented design into memory.
2. In the Vivado IDE, do one of the following:
   - Click **Flow Navigator → Synthesis**
   - Click **Flow Navigator → Implementation**
   - Select **Reports → Timing → Report Timing Summary**
3. In Tcl, run the command `report_timing_summary`.
4. Review the interactive report that appears.

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

In a synthesized design, the AMD Vivado™ IDE timing engine estimates net delays based on connectivity and fanout. You get more accurate delay estimates for nets between cells that you have already placed. However, you might see larger clock skew on paths where some cells, such as I/Os and GTs, are only pre-placed.

In an implemented design, the tool calculates net delays based on actual routing information. You must use the Timing Summary report for timing signoff after the design is completely routed.

To verify that routing is complete, open the Route Status report.

Use the following instructions to generate a scoped Timing Summary report:

1. Run the report from the Tcl Console or GUI.
2. Use the `-cells` option to scope the report to one or more hierarchical cells.
3. Review the report. The only paths that are reported are the ones where the datapath section starts, ends, crosses, or is fully contained within the specified cells.

When you run `report_timing_summary` from the Tcl Console, the first section of the report shows a summary of methodology violations from the latest `report_methodology` run. When you run the report from the GUI, this section appears as Methodology Summary.

- If you have not run `report_methodology` before generating the Timing Summary, this section appears empty.
- If the design has changed because the last `report_methodology` run, the violations summary might be outdated. Run `report_methodology` again to ensure accuracy.

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 the Device and Schematic windows, and design source files.

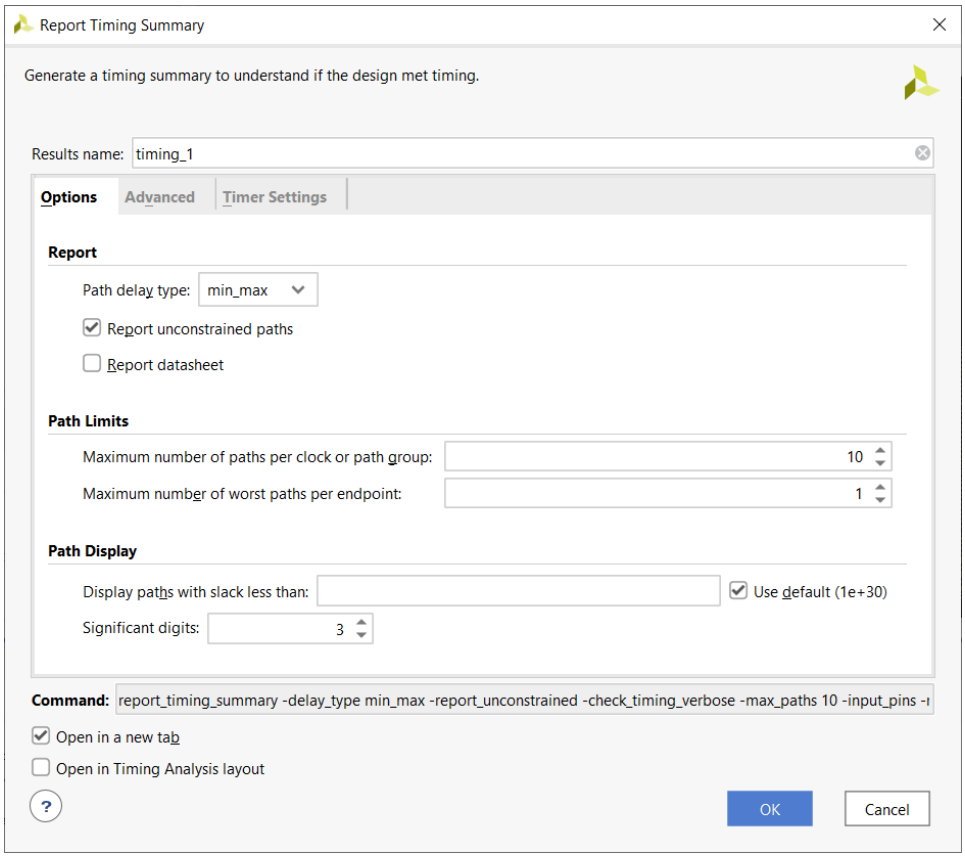
If this field is left empty, the report is returned to the Tcl Console, and a graphical version of the report is not opened in the Results window.

The equivalent Tcl option is `-name`.

Options Tab

Use the Options tab to control which timing data is included in the report and how it is displayed.

Figure 80: **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`.

  The equivalent Tcl option is `-delay_type`.

- **Report unconstrained paths:** Generates information on paths that do not have timing constraints. This option is enabled by default in the Vivado IDE, but is disabled by default in the equivalent `report_timing_summary` Tcl command option.

  The equivalent Tcl option is `-report_unconstrained`.

- **Report datasheet:** Generates the design datasheet as defined in Report Datasheet.

  The equivalent Tcl option is `-datasheet`.

Path Limits Section

- **Maximum number of paths per clock or path group:** Controls how many paths to report per clock group.

  The equivalent Tcl option is `-max_paths`.

- **Maximum number of worst paths per endpoint:** Sets a cap on how many of the worst paths to report per endpoint. This value is still limited by `-max_paths`.

  The equivalent Tcl option is `-nworst`.

Path Display Section

- **Display paths with slack less than:** Filter reported paths by slack. This does not affect the summary tables.

  The equivalent Tcl option is `-slack_lesser_than`.

- **Significant digits:** Set the precision for displayed values.

  The equivalent Tcl option is `-significant_digits`.

Common Section

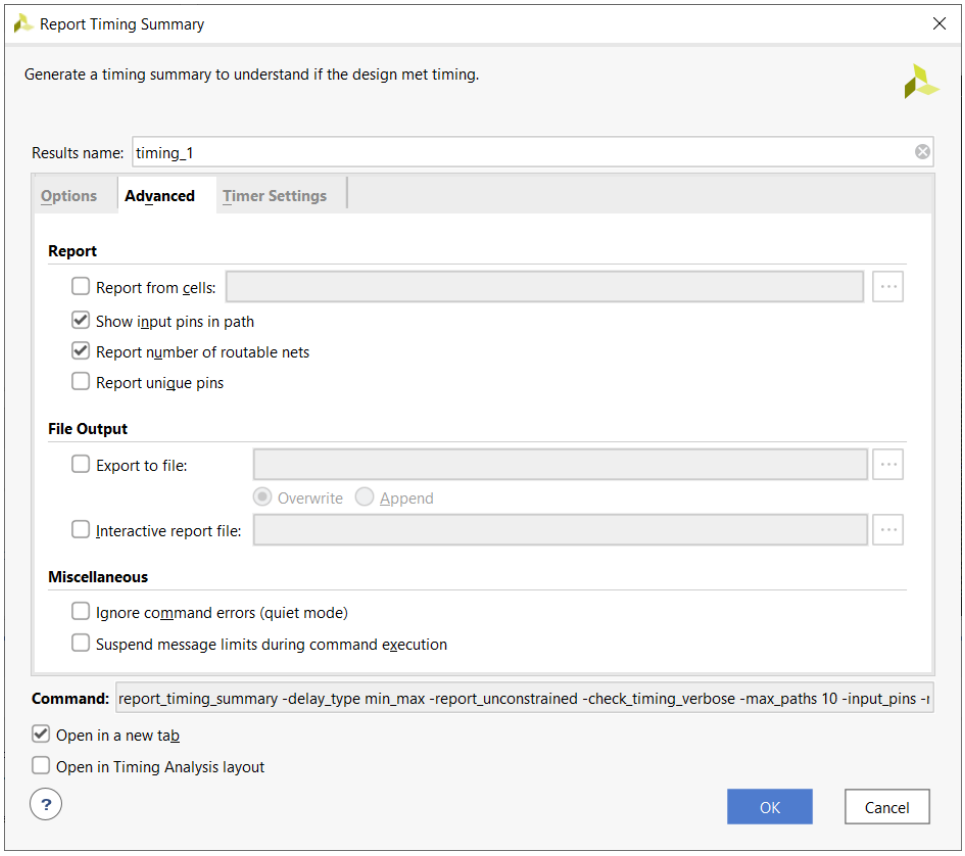
The following controls, which are common to all three tabs, are located at the bottom of the Report Timing Summary dialog box:

- **Command:** Shows the Tcl equivalent of selected GUI options.
- **Open in a New Tab:** Opens the report in a new tab or replaces the most recent one.
- **Open in Timing Analysis Layout:** Switches the layout to the default Timing Analysis view.

  For layout information, see *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Advanced Tab

Figure 81: **Report Timing Summary Dialog Box: Advanced Tab**



Report Section

- **Report from cells:** Limit the report to timing paths that start, end, cross, or are fully contained within selected cells.

  The equivalent Tcl option is: `-cells`

- **Show input pins in path:** Display the input pin used in each cell on the path.

  The equivalent Tcl option is: `-input_pins`

  Note: Keep this enabled for complete pin details.

- **Report number of routable nets:** Add the number of nets in the design that require routing resources to the report. This count gives context about net complexity and can help you identify routing pressure that might impact timing.

- **Report unique pins:** Show only one timing path per unique pin set.

  The equivalent Tcl option is: `-unique_pins`

File Output Section

- **Export to file:** Exports the report to a file. By default, results display in the Timing window.

  The equivalent Tcl option is `-file`.

- **Overwrite/Append:** Choose whether to overwrite or append to the file.

  The equivalent Tcl option is `-append`.

- **Interactive report file:** Saves the report in AMD RPX format to reopen later with `open_report`.

Miscellaneous Section

- **Ignore command errors (quiet mode):** Run quietly and ignore command-line errors. The command returns `TCL_OK` even if it encounters errors during execution.

  The equivalent Tcl option is `-quiet`.

- **Suspend message limits during command execution:** Return all messages, ignoring any message limits.

  The equivalent Tcl option is `-verbose`.

Timer Settings Tab

To set the timer settings, use one of the following options:

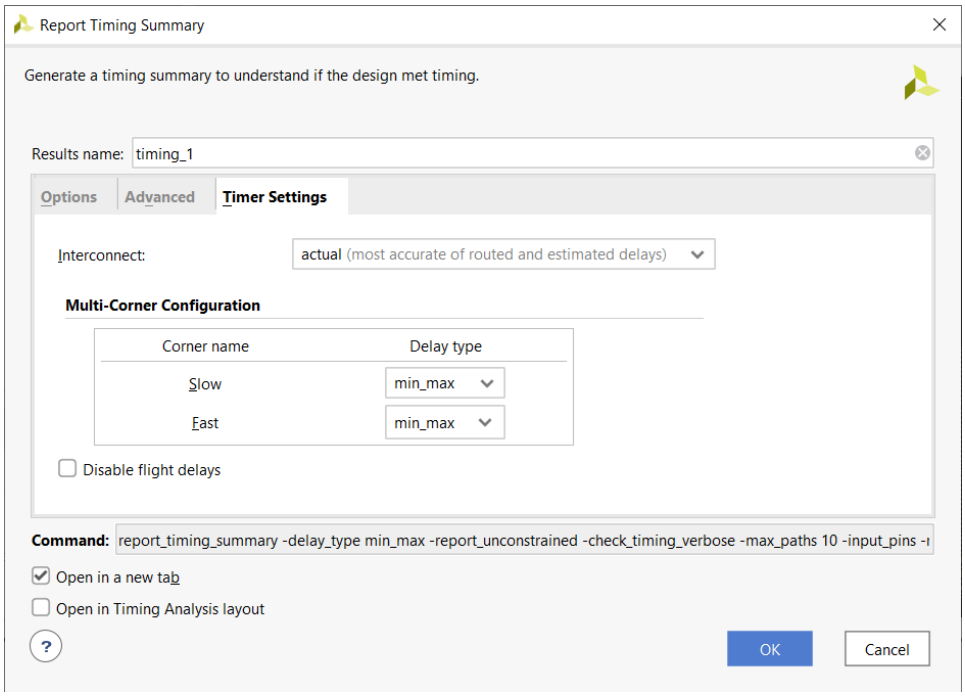
- A Vivado IDE timing analysis dialog box
- A Tcl command option listed in this section

These settings affect all timing-related commands in the current Vivado session, except for synthesis and implementation. The tool does not save timer settings as a persistent preference. When you start a new session, Vivado resets them to their default values.

Do not change the default timer settings. Keeping the defaults ensures the widest coverage and most accurate delay values during timing analysis.

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

Figure 82: **Report Timing Summary Dialog Box: Timer Settings Tab**



Interconnect

Use this setting to control how Vivado calculates net delays in the timing report. You can base delays on estimated placement, actual routing, or exclude them entirely. By default, Vivado sets this option to Estimated for post-synthesis designs and Actual for post-implementation designs.

The equivalent Tcl command is `set_delay_model`.

Estimated

1. For unplaced cells, Vivado estimates delay based on ideal placement, driver characteristics, load type, and fanout. Nets are labeled *unplaced* in the report.
2. For placed cells, delay is based on the actual distance between the driver and load and is labeled *estimated* in the report.

Actual

For routed designs, delay reflects the real hardware delay of the routed net. These nets are labeled *routed* in the report.

None

Vivado excludes interconnect delays and sets all net delays to zero.

Multi-Corner Configuration

Use this setting to specify which types of path delays to analyze for each timing corner. Valid options are as follows:

- `none`: Disable timing analysis for the selected corner
- `max`: Analyze setup and recovery checks (max delay)
- `min`: Analyze hold and removal checks (min delay)
- `min_max`: Analyze both min and max delays

RECOMMENDED: *Keep both `max` (setup) and `min` (hold) analysis enabled for all corners to ensure complete coverage.*

The equivalent Tcl command is `config_timing_corners`.

Disable Flight Delays

Use this option to exclude package delays from I/O delay calculations. This setting affects how I/O timing is evaluated in the report.

The equivalent Tcl command is `config_timing_analysis`.

Details of the Timing Summary Report

The Timing Summary Report includes 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

This report gives you a comprehensive overview of your design's timing status. It combines data from multiple sources, including reports available in the Vivado IDE, such as Report Clock Interaction, Report Pulse Width, Report Timing, and Check Timing. It also provides Tcl-only reports like `report_clocks`.

In addition to consolidating this information, the Timing Summary Report also includes unique sections, such as Unconstrained Paths, that help identify missing or incomplete constraints.

General Information Section

The General Information section of the Timing Summary Report shows basic details about the design and reporting environment. It includes:

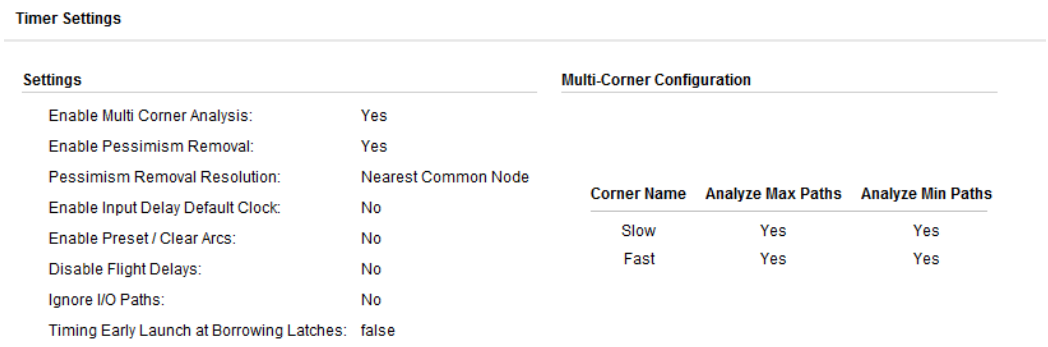
- The design name
- The selected device, package, and speed grade (including the speed file version)
- The Vivado Design Suite release version
- The current date when the report was generated
- The equivalent Tcl command options Vivado used to generate the report

Use this section to confirm the target settings and tool version used for timing analysis.

Timer Settings Section

The Timer Settings section shows the timing engine settings used to generate the report. These options determine how Vivado performs timing analysis and affect the accuracy and coverage of the results.

Figure 83: **Timing Summary Report: Timer Settings**



This section includes the following settings:

- **Enable Multi-Corner Analysis:** Enables timing analysis for all configured timing corners.
- **Enable Pessimism Removal (and Pessimism Removal Resolution):** Removes clock skew between the source and destination clocks at their common node.

  Note: This setting must always remain enabled.

- **Enable Input Delay Default Clock:** Applies a default null input delay to input ports that lack user-defined constraints. This option is disabled by default.

- **Enable Preset / Clear Arcs:** Allows timing analysis through asynchronous control pins like preset and clear. It does not apply to recovery or removal checks. This option is disabled by default.
- **Disable Flight Delays:** Excludes package delays from I/O delay calculations.

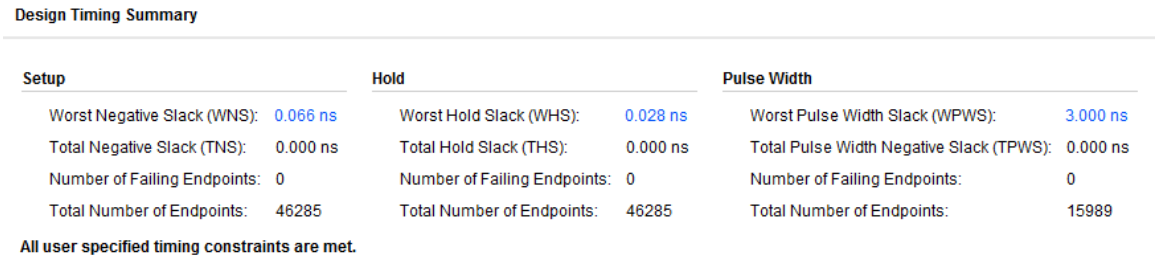
For details on how to configure these settings, see `config_timing_analysis` in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Design Timing Summary Section

The Design Timing Summary section provides a high-level summary of your design's timing status. It aggregates results from all other sections into a single view.

RECOMMENDED: *Review this section after implementation to confirm that all timing constraints are met, or at any stage to understand the current timing status.*

Figure 84: **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 shows all results related to max delay analysis, including setup, recovery, and data checks.

It includes the following values:

- **Worst Negative Slack (WNS):** The worst slack across all timing paths analyzed for max delay. This value can be positive or negative.
- **Total Negative Slack (TNS):** The total of all WNS violations, based on the worst violation per endpoint.
  - Value is 0 ns when all setup-related constraints are met

  - Value is negative when violations exist
- **Number of Failing Endpoint:** The number of endpoints with WNS violations (slack less than 0 ns).
- **Total Number of Endpoints:** The total number of endpoints checked during max delay analysis.

Hold Area (Min Delay Analysis)

The Hold area of the Design Timing Summary shows all results related to min delay analysis, including hold, removal, and data checks.

It includes the following values:

- **Worst Hold Slack (WHS):** The worst slack across all timing paths analyzed for min delay. This value can be positive or negative.
- **Total Hold Slack (THS):** The total of all WHS violations, based on the worst violation per endpoint.
  - Value is 0 ns when all hold-related constraints are met
  - Value is negative when violations exist
- **Number of Failing Endpoints:** The number of endpoints with WHS violations (slack less than 0 ns).
- **Total Number of Endpoints:** The total number of endpoints checked during min delay analysis.

Pulse Width Area (Pin Switching Limits)

The Pulse Width area of the Design Timing Summary shows results for all checks related to pin switching limits. These checks include the following:

- Minimum low pulse width
- Minimum high pulse width
- Minimum period
- Maximum period
- Maximum skew (between clock pins of the same leaf cell, such as PCIe® or GT on AMD UltraScale™ devices)

The report provides the following values:

- **Worst Pulse Width Slack (WPWS):** The worst slack across all pulse width-related checks, using both min and max delays.
- **Total Pulse Width Slack (TPWS):** The sum of all WPWS violations, based on the worst violation per pin.
  - Value is 0 ns when all pulse width constraints are met

  - Value is negative when violations exist
- **Number of Failing Endpoints:** The number of pins with WPWS violations (slack less than 0 ns).
- **Total Number of Endpoints:** The total number of pins analyzed for pulse width checks.

Clock Summary Section

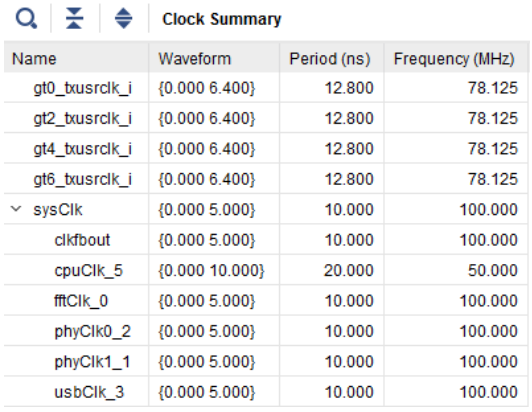
The Clock Summary section of the Timing Summary Report includes information similar to what you see in the `report_clocks` command.

It includes the following information:

- All clocks in the design, whether defined using `create_clock`, `create_generated_clock`, or generated automatically by the tool
- Properties of each clock including the following:
  - Name
  - Period
  - Waveform
  - Target frequency

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

Figure 85: **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 shows a table of methodology violations. The total number of violations appears on the right hand side of the table.

An icon next to each category helps you quickly identify the severity:

- A red icon indicates that the highest severity is an Error

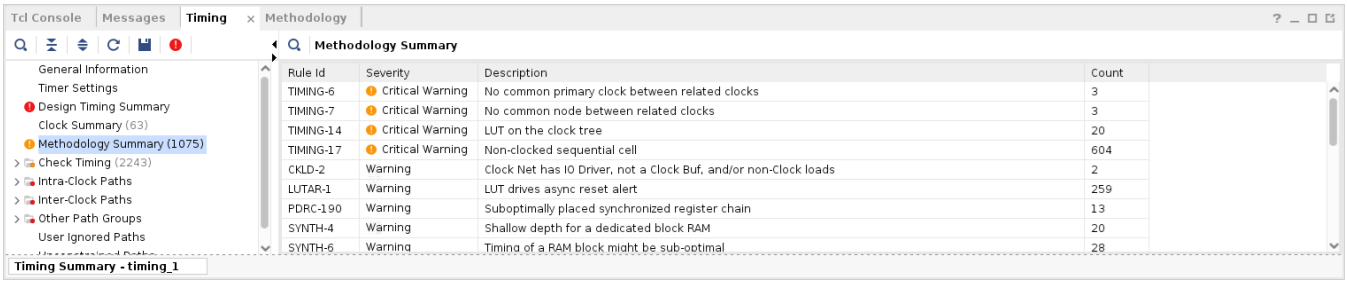
- An orange icon indicates a Critical Warning
- No icon appears for Warnings or Advisories

***Note*:** Report Timing Summary does not run `report_methodology`. It only displays a summary of violations from the most recent `report_methodology` run, either from the current in-memory design or a loaded checkpoint.

Use the following instructions to generate a full list of methodology violations:

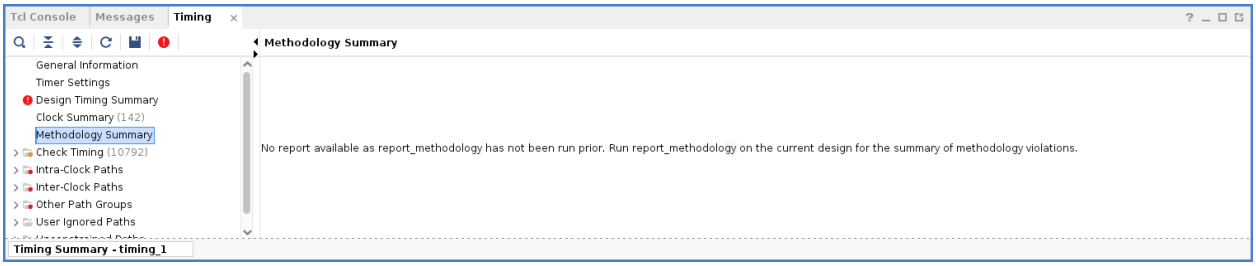
- Select **Reports → Report Methodology** from the menu.
- Or run the `report_methodology` Tcl command.

Figure 86: **Methodology Summary**



If you have not run `report_methodology` before generating the Timing Summary Report, Vivado displays a message stating that no methodology report is available, as shown in the following figure.

Figure 87: **Methodology Summary without Running Report Methodology**



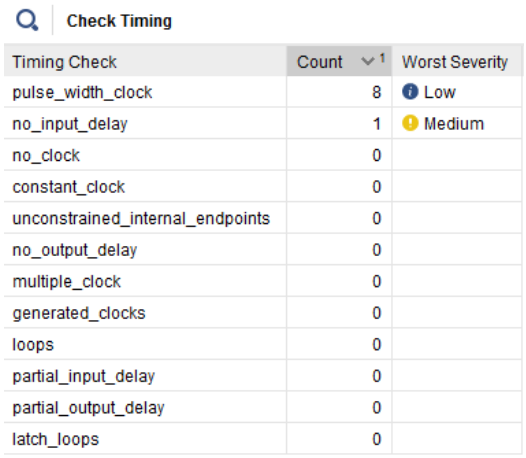
For more information about Report Methodology, refer to Methodology Analysis.

Check Timing Section

The Check Timing section of the Timing Summary Report lists issues related to missing or incomplete timing constraints. For complete timing signoff, you must constrain all path endpoints.

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

Figure 88: **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:

- In the Vivado IDE, select the **Reports → Timing → Check Timing** menu command.
- Run the `check_timing` Tcl command.

When running from the Tcl Console, you can use the `-cells` option to scope the report to specific hierarchical cells. This option is not available in the GUI.

Common checks reported in this section (as shown in the previous figure) include the following:

- **`pulse_width_clock`:** Reports clock pins that have only a pulse width check, with no setup, hold, recovery, removal, or clk-to-Q check.
- **`no_input_delay`:** Reports non-clock input ports that are missing input delay constraints.
- **`no_clock`:** Reports clock pins not reached by a defined timing clock. This also includes constant-driven clock pins.
- **`constant_clock`:** Flags clock signals that are connected to a constant (such as GND, VSS, or static data).
- **`unconstrained_internal_endpoints`:** Reports path endpoints (excluding output ports) with no timing requirement. These often result from missing clock definitions and can be related to `no_clock`.
- **`no_output_delay`:** Reports non-clock output ports without at least one output delay constraint.
- **`multiple_clock`:** Reports clock pins reached by more than one timing clock. This usually indicates a multiplexer in the clock tree. Only one clock can drive a clock tree at a time. Review the design if multiple clocks are unexpectedly present.

- `generated_clocks`: Reports generated clocks with a master source that is not in the same clock tree. This can happen when a timing arc is disabled between the master clock and the generated clock's source. If you use the `-edges` option, the logical unateness (inverting/non-inverting) must match the edge relationship.
- `loops`: Reports combinational loops found in the design. Vivado breaks these loops automatically for timing analysis.
- `partial_input_delay`: Reports input ports that have only a min or max input delay constraint, but not both. These paths are excluded from either setup or hold analysis.
- `partial_output_delay`: Reports output ports that have only a min or max output delay constraint. These are also excluded from complete timing analysis.
- `latch_loops`: Reports loops that pass through latches. These are not included in regular combinational loop checks and can affect latch time borrowing calculations.

Use this section to identify and resolve constraint issues that could compromise the accuracy of your timing analysis.

Intra-Clock Paths Section

The Intra-Clock Paths section of the Timing Summary Report displays the worst slack and total violations for timing paths that share the same source and destination clock.

Figure 89: **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 by running `-max_paths` on the command line or the maximum number of paths per clock or path group.

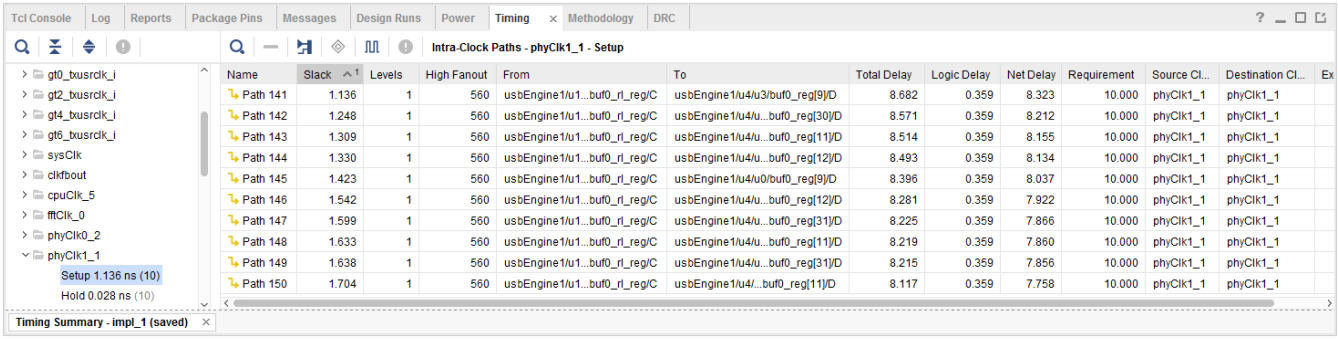
1. In the left index pane, click any entry under Intra-Clock Paths.
2. Review the slack and violation summary for each clock.
3. Examine the N-worst paths reported for each analysis type:
   - Setup

- Hold
- Pulse Width

The N-worst paths are determined by the `-max_paths` value in the Tcl command option or by the maximum number of paths per clock or path group set in the GUI.

Next to each analysis label, you can see the worst slack value and the number of reported paths. When you select a Setup summary (or Hold or Pulse Width), the report displays a table on the right showing all paths related to that clock.

Figure 90: **Timing Summary Report: Intra-Clock Paths Details**



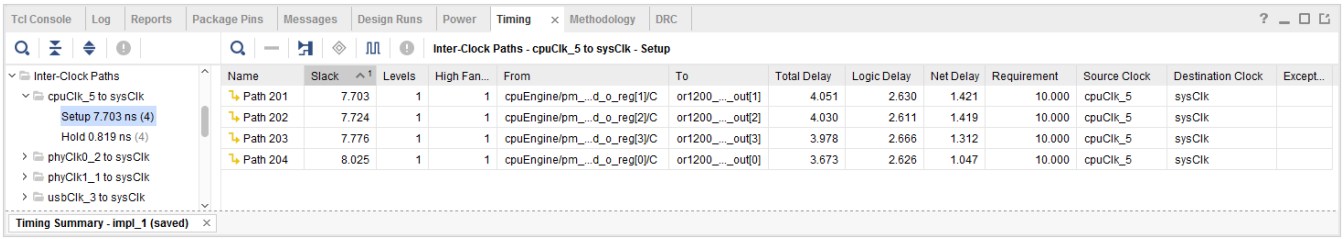
Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Cl...	Destination Cl...
Path 141	1.136	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u3/buf0_reg[9]/D	8.682	0.359	8.323	10.000	phyClk1_1	phyClk1_1
Path 142	1.248	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[30]/D	8.571	0.359	8.212	10.000	phyClk1_1	phyClk1_1
Path 143	1.309	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[11]/D	8.514	0.359	8.155	10.000	phyClk1_1	phyClk1_1
Path 144	1.330	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[12]/D	8.493	0.359	8.134	10.000	phyClk1_1	phyClk1_1
Path 145	1.423	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u0/buf0_reg[9]/D	8.396	0.359	8.037	10.000	phyClk1_1	phyClk1_1
Path 146	1.542	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[12]/D	8.281	0.359	7.922	10.000	phyClk1_1	phyClk1_1
Path 147	1.599	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[31]/D	8.225	0.359	7.866	10.000	phyClk1_1	phyClk1_1
Path 148	1.633	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[11]/D	8.219	0.359	7.860	10.000	phyClk1_1	phyClk1_1
Path 149	1.638	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/u...buf0_reg[31]/D	8.215	0.359	7.856	10.000	phyClk1_1	phyClk1_1
Path 150	1.704	1	560	usbEngine1/u1...buf0_rl_reg/C	usbEngine1/u4/...buf0_reg[11]/D	8.117	0.359	7.758	10.000	phyClk1_1	phyClk1_1

Use this section to quickly identify the most critical timing paths within the same clock domain.

Inter-Clock Paths Section

The Inter-Clock Paths section of the Timing Summary Report shows the worst slack and total violations for timing paths between different source and destination clocks.

Figure 91: **Timing Summary Report Inter-Clock Paths Details**



Name	Slack	Levels	High Fan...	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Except...
Path 201	7.703	1	1	cpuEngine/pm_...d_o_reg[1]/C	or1200_..._out[1]	4.051	2.630	1.421	10.000	cpuClk_5	sysClk	
Path 202	7.724	1	1	cpuEngine/pm_...d_o_reg[2]/C	or1200_..._out[2]	4.030	2.611	1.419	10.000	cpuClk_5	sysClk	
Path 203	7.776	1	1	cpuEngine/pm_...d_o_reg[3]/C	or1200_..._out[3]	3.978	2.666	1.312	10.000	cpuClk_5	sysClk	
Path 204	8.025	1	1	cpuEngine/pm_...d_o_reg[0]/C	or1200_..._out[0]	3.673	2.626	1.047	10.000	cpuClk_5	sysClk	

To view detailed information:

1. In the left index pane, click any entry under Inter-Clock Paths.
2. Review the slack and violation summary for each clock pair.
3. Examine the N-worst paths reported for each analysis type:
   - Setup
   - Hold

- Pulse Width

The N-worst paths are determined by the `-max_paths` value in the Tcl command option or by the maximum number of paths per clock or path group set in the GUI.

Use this section to analyze cross-clock domain timing interactions and identify any violations that might require constraint adjustments or design changes.

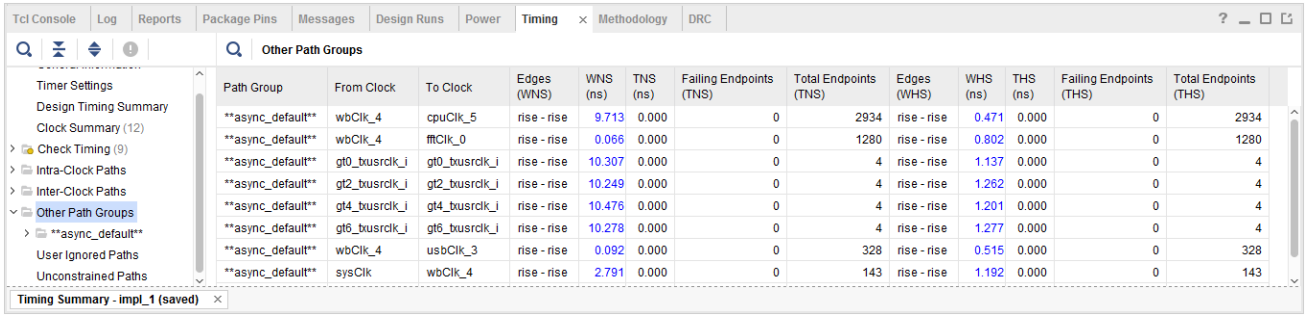
Other Path Groups Section

The Other Path Groups section of the Timing Summary Report shows both default and user-defined path groups.

To view this section:

1. In the left index pane, select **Other Path Groups**.
2. Review the summary table that lists each path group.

Figure 92: **Timing Summary Report: Path Groups Section**



Path Group	From Clock	To 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)
async_default	wbClk_4	cpuClk_5	rise - rise	9.713	0.000	0	2934	rise - rise	0.471	0.000	0	2934
async_default	wbClk_4	fftClk_0	rise - rise	0.066	0.000	0	1280	rise - rise	0.802	0.000	0	1280
async_default	gt0_txusrclk_i	gt0_txusrclk_i	rise - rise	10.307	0.000	0	4	rise - rise	1.137	0.000	0	4
async_default	gt2_txusrclk_i	gt2_txusrclk_i	rise - rise	10.249	0.000	0	4	rise - rise	1.262	0.000	0	4
async_default	gt4_txusrclk_i	gt4_txusrclk_i	rise - rise	10.476	0.000	0	4	rise - rise	1.201	0.000	0	4
async_default	gt6_txusrclk_i	gt6_txusrclk_i	rise - rise	10.278	0.000	0	4	rise - rise	1.277	0.000	0	4
async_default	wbClk_4	usbClk_3	rise - rise	0.092	0.000	0	328	rise - rise	0.515	0.000	0	328
async_default	sysClk	wbClk_4	rise - rise	2.791	0.000	0	143	rise - rise	1.192	0.000	0	143

This section includes any groups you define with the `group_path` command, along with default groups.

TIP: *The `**async_default**` group is automatically created by the Vivado IDE timing engine. It includes all paths that end with asynchronous checks, such as recovery and removal:*

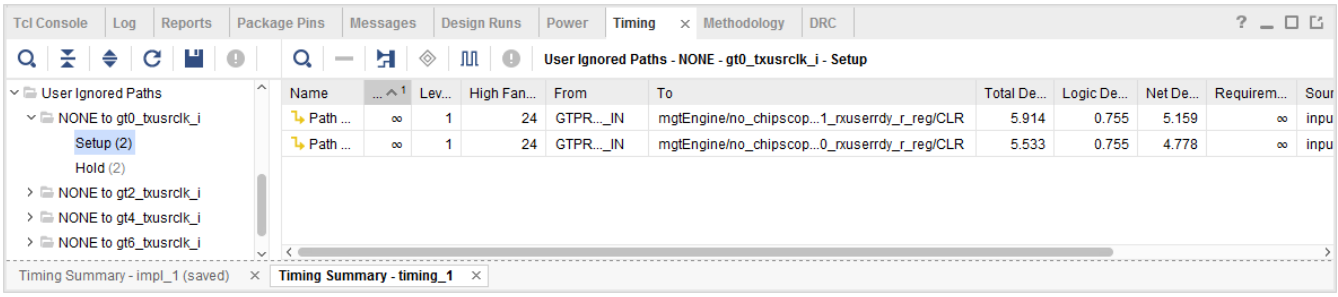
- Recovery checks appear under the SETUP category (max delay analysis).
- Removal checks appear under the HOLD category (min delay analysis).

Any groups you create with the `group_path` command also appear in this section. A path group can include any combination of source and destination clocks.

User-Ignored Paths Section

The User-Ignored Paths section of the Timing Summary Report displays paths that are excluded from timing analysis due to `set_clock_groups` and `set_false_path` constraints. These paths are not analyzed, and their reported slack is infinite.

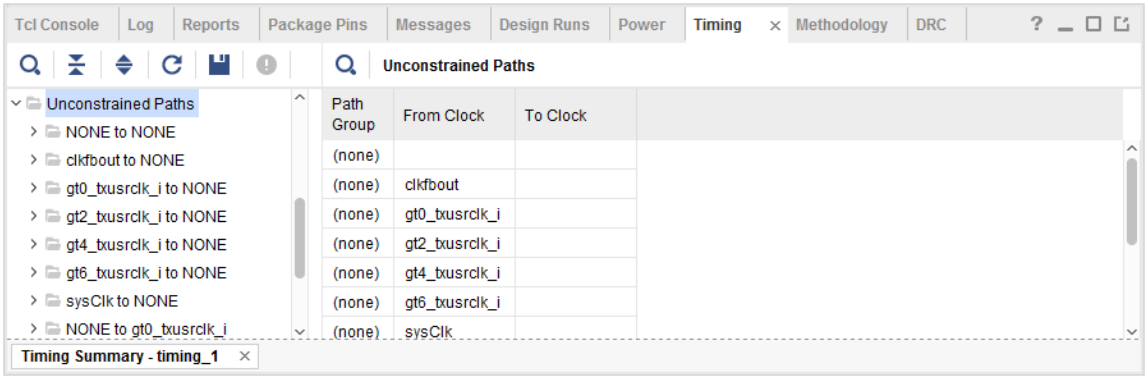
Figure 93: **Timing Summary Report: User-Ignored Paths Section**



Unconstrained Paths Section

The Unconstrained Paths section of the Timing Summary Report lists logical paths that are not timed because of missing timing constraints. These paths are grouped by their source and destination clock pairs. If no clock is associated with the path startpoint or endpoint, the clock name appears as empty or NONE.

Figure 94: **Timing Summary Report: Unconstrained Paths Section**



Reviewing Timing Path Details

Use the following instructions to review timing path details in the Timing Summary Report:

1. Expand any Setup, Hold, or Pulse Width sub-section to views paths organized by clock pairs.
2. Select a path to display its details in the Path Properties window.
3. Double-click a path to open the details in a new window.

Use the following instructions 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 the selected path.

- **Highlight:** Highlight the path in the Device and Schematic windows.

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

Filtering Paths with Violations

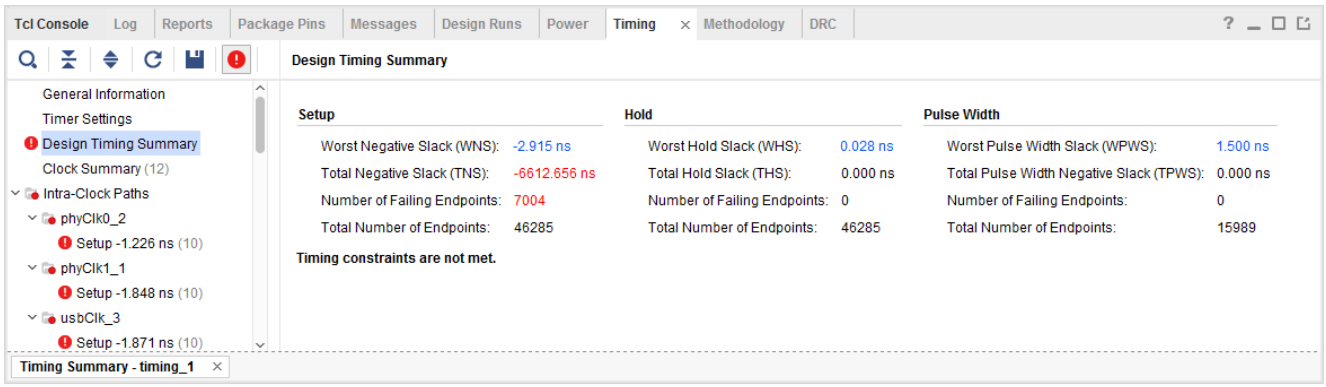
Use the following instructions to filter paths with violations in the Timing Summary Report:

1. Look for slack values displayed in red. These indicate failing paths.
2. Click the **Show only failing paths** button to hide all non-violating paths.
3. Review the filtered list to focus only on the paths that require attention.

This view helps you prioritize timing issues by highlighting only the critical violations.

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

Figure 95: **Timing Summary Report: Violating Paths Filter**



Report Bus Skew

The Bus Skew report shows all bus skew constraints set using the `set_bus_skew` command. This report is not included in the Timing Summary, so you must generate it separately to complete timing signoff.

Running Report Bus Skew

You can run the Bus Skew report in two ways:

1. From the Tcl Console use the `report_bus_skew` command. This method gives you access to advanced options, including:
   - `-cells` to scope the report to one or more hierarchical cells. The report includes only paths where the datapath section starts, ends, or is fully contained within the specified cells.

- `-sort_by_slack` to sort the constraints in ascending order of slack, from smallest to largest.

2. From the Vivado IDE, select **Reports → Timing → Report Bus Skew**.

   Note: The IDE does not support the `-cells` option.

The command shares many filtering and formatting options with `report_timing`, allowing you to customize the output. Constraints appear in the order you defined them unless you apply the `-sort_by_slack` option that 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:

- **Bus Skew Report Summary:** This section provides an overview of all bus skew constraints in the design.
- **Bus Skew Report by Constraint:** This section lists each bus skew constraint along with its associated path details.

Bus Skew Report Summary Section

The Bus Skew Report Summary lists all `set_bus_skew` constraints defined in the design. For each constraint, the report includes the following:

- **Id:** A unique identifier used throughout the report to help you locate and track each constraint
- **From:** The pattern you specified with the `set_bus_skew -from` option
- **To:** The pattern you specified with the `set_bus_skew -to` option
- **Corner:** The analysis corner (Slow or Fast) where the report found the worst bus skew
- **Requirement:** The skew limit you defined for the constraint
- **Actual:** The worst bus skew value found across all paths covered by the constraint
- **Slack:** The difference between the actual skew and the requirement

In the following example, the design has one bus skew constraint with a 1 ns requirement and the worst skew observed is 1.107 ns, the report flags it as a violation.

```
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) means the timing difference between bits of an asynchronous bus exceeds the expected limit. This can cause incorrect data capture in the destination clock domain, where different bits could reflect a state sent by the source clock domain at different clock cycles. If any WBSS violations remain after routing, try using a different placement or routing directive. To maintain hardware stability, make sure no bus skew violations remain in the design.

Bus Skew Report Per Constraint Section

The Bus Skew Report Per Constraint section provides detailed information for each `set_bus_skew` constraint. Each constraint includes two parts:

- A summary of the paths covered by the constraint.
- A list of detailed timing paths corresponding to the summary entries.

The summary table includes the following information:

- **From Clock:** The clock domain of the startpoints
- **To Clock:** The clock domain of the endpoints
- **Endpoint Pin:** The endpoint pin in the reported path
- **Reference Pin:** The reference pin used to compute the skew. Each row shows the reference pin that caused the largest skew for that endpoint
- **Corner:** The fast or slow analysis corner used to compute the worst skew
- **Actual:** The computed skew, calculated as the difference between the relative delay to the endpoint pin and the reference pin, minus the relative CRPR
- **Slack:** The difference between the actual skew and the constraint requirement

Note: You must specify both the `-from` and `-to` options when defining a bus skew constraint.

By default, the report includes only the endpoint with the worst bus skew. To include more endpoints, use the `-nworst` and `-max_paths` command-line options. These options function similarly to those in the `report_timing` command. For example, `-nworst 1 -max_paths 16` reports up to 16 endpoints per constraint, with one path per endpoint.

Figure 96: **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 second part of the report provides a detailed timing path for each pin pair listed in the summary. The number of detailed paths matches the number of endpoints and is also controlled by the `-max_paths` and `-nworst` options.

The format of the detailed timing paths is similar to a traditional timing path report, but with a few differences:

- Clock uncertainty is not included, because the analysis assumes the same clock edge
- The report shows the worst-case clock uncertainty from the endpoint or reference path in the header
- The launch time for the destination clock is always zero
- For each slack value, the report shows both the timing path to the endpoint and the timing path to the reference pin
- If the clock or datapath crosses multiple SLRs, Vivado applies inter-SLR compensation to reduce pessimism. The report includes this compensation in the header

If you use the `-path_type short` option, the report collapses the clock network details. In this format, the path to the endpoint pin appears first, followed by the path to the reference pin. The header summarizes information from both paths, the requirement, and the relative CRPR.

Figure 97: **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

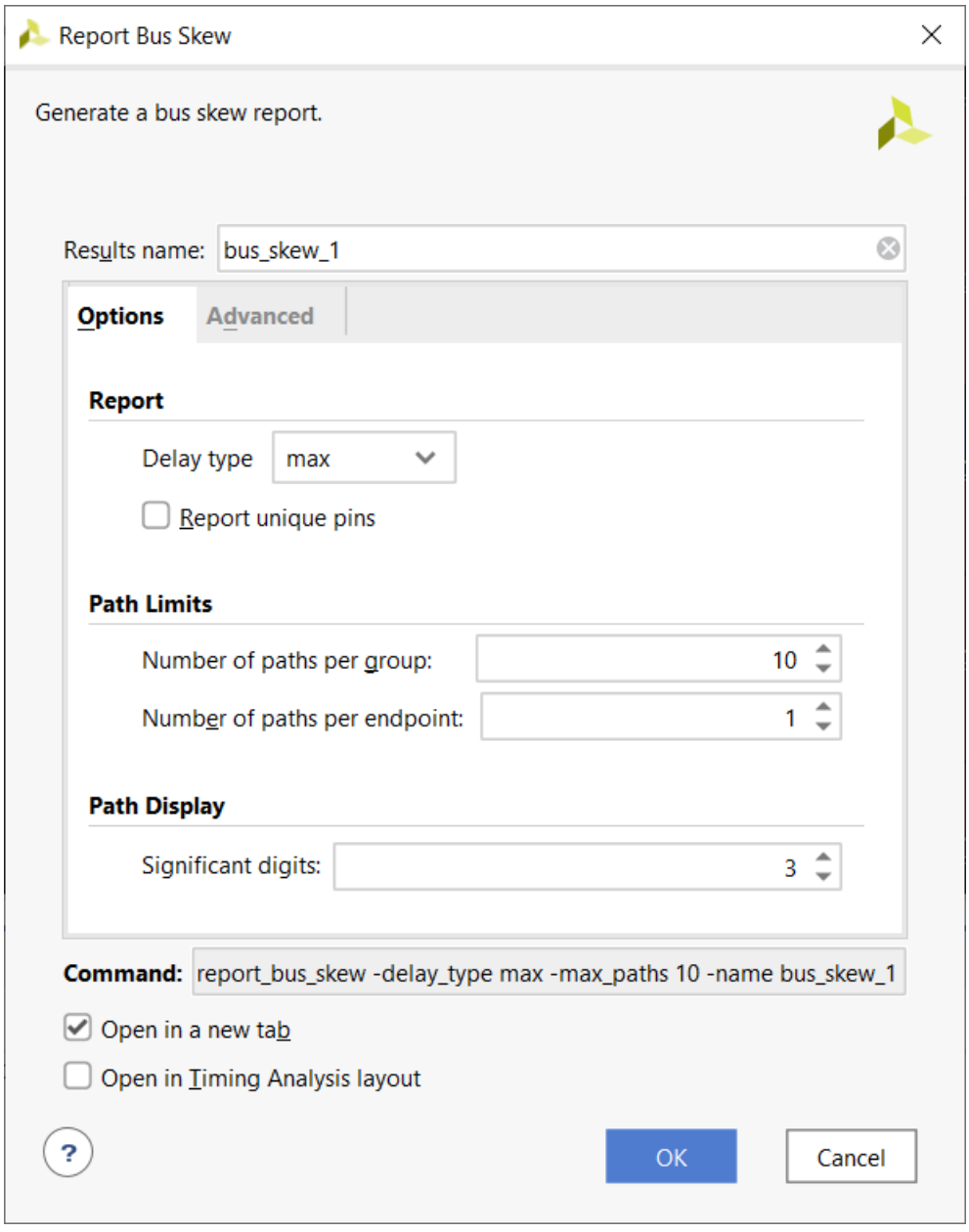
To open the Report Bus Skew dialog box, select **Reports → Timing → Report Bus Skew**.

The dialog box includes two tabs:

- Options Tab
- Advanced Tab

Options Tab

Figure 98: **Report Bus Skew Dialog Box: Options Tab**



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

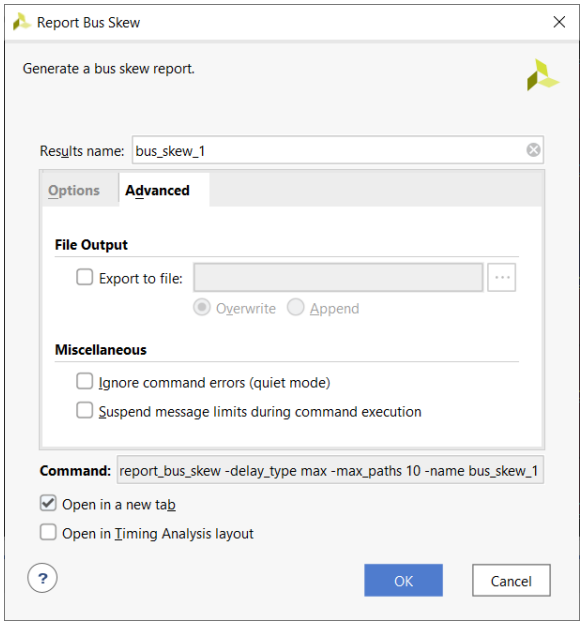
- **Report:**

- Delay type: Sets the type of delay used in analysis, such as max, min, or both. Determines how the tool calculates and reports path delays. See the Report Section for details on each type.
- Report unique pins: Limits the report to one timing path per unique combination of input and output pins. Use this to reduce duplicate or redundant paths in the report.
  - Equivalent Tcl option: `-unique_pins`

- **Path Limits:**
  - Number of paths per group: Sets the maximum number of timing paths to report for each clock group or constraint group. See the Report Timing Summary section for more details on how grouping is handled.
  - Number of paths per endpoint: Sets the number of timing paths reported for each endpoint pin. This helps you focus on the worst-case paths for each destination. See the Report Timing Summary for context.
- **Path Display:** Number of significant digits: Controls how many significant digits are displayed for delay and slack values in the report. See the Report Timing Summary section for examples and recommendations.

Advanced Tab

Figure 99: **Report Bus Skew Dialog Box: Advanced Tab**



The Advanced tab of the Report Bus Skew dialog box includes the following sections:

- **File Output:** Export to file: Saves the report to a specified file. By default, the report appears in the Timing window of the Vivado IDEEquivalent Tcl option: `-file`

- **Miscellaneous:**
  - Ignore command errors: Runs the command quietly without displaying error messages. The tool returns `TCL_OK` even if it encounters errors
    - Equivalent Tcl option: `-quiet`
  - Suspend message limits during command execution: Temporarily disables any message limits so that all messages are returned during execution
    - 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

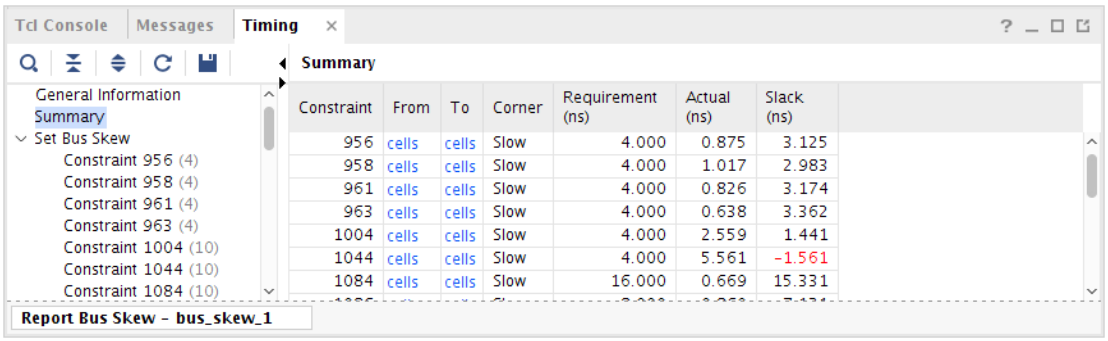
General Information Section

This section shows the basic context for the report, including:

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

Summary Section

Figure 100: **Report Bus Skew: Summary Section**



This section provides a table that summarizes all `set_bus_skew` constraints. For each constraint, it shows:

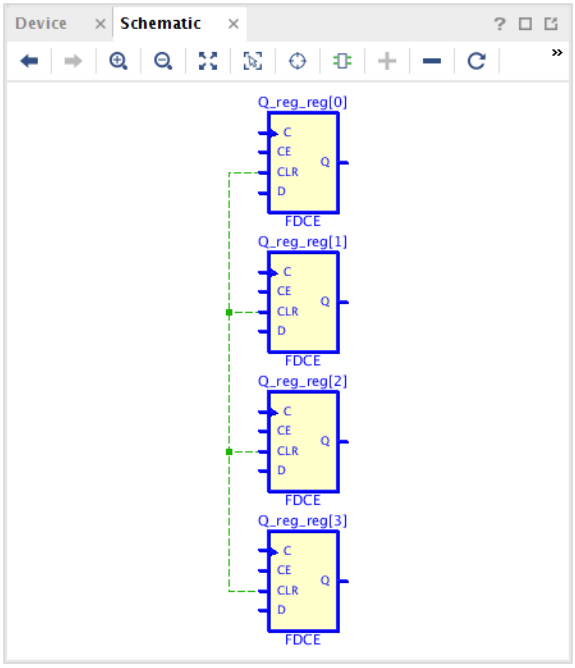
- The requirement
- The worst actual bus skew observed
- The calculated slack

Use this table to quickly identify any bus skew violations.

Additional Table Features

- The Constraint column shows the position number of the timing constraint. This number matches the one in the timing constraints editor (TCE)
- The From and To columns include hyperlinks to the specified objects (cells or pins) from the `set_bus_skew` command
- You can click these links to select all startpoints or endpoints associated with a specific constraint. After you select it, press F4 to open the schematic

Figure 101: **Example Schematic of the Bus Skew Endpoints**



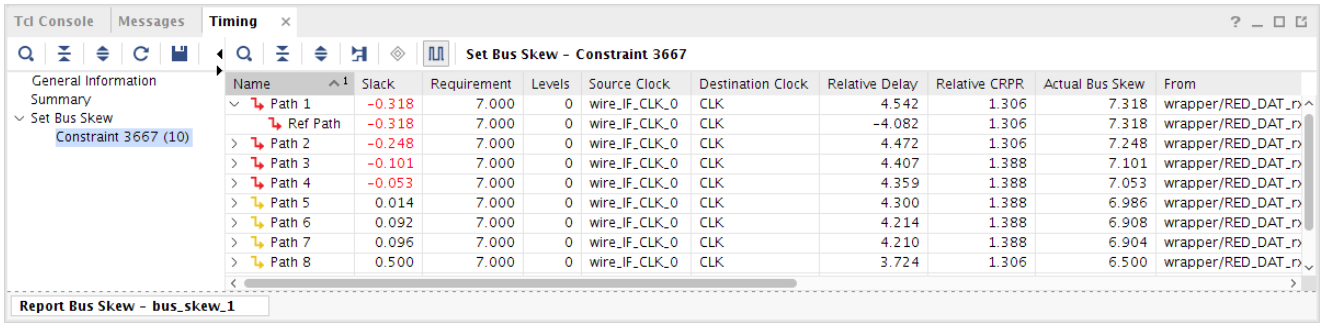
Set Bus Skew Section

This section lists the detailed timing paths for each bus skew constraint. Each endpoint has an associated reference path that you can expand. 

Vivado calculates the bus skew using the following formula:

$$\text{Actual bus skew} = \text{endpoint relative delay} - \text{reference relative delay} - \text{relative CRPR}$$

Figure 102: **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

Use the following instructions to examine a path:

1. Select the path in the report
2. View the detailed timing path in the Properties pane
3. Click the schematic icon or press F4 to generate the schematic for the endpoint path, the reference path, or both together

Report Clock Networks

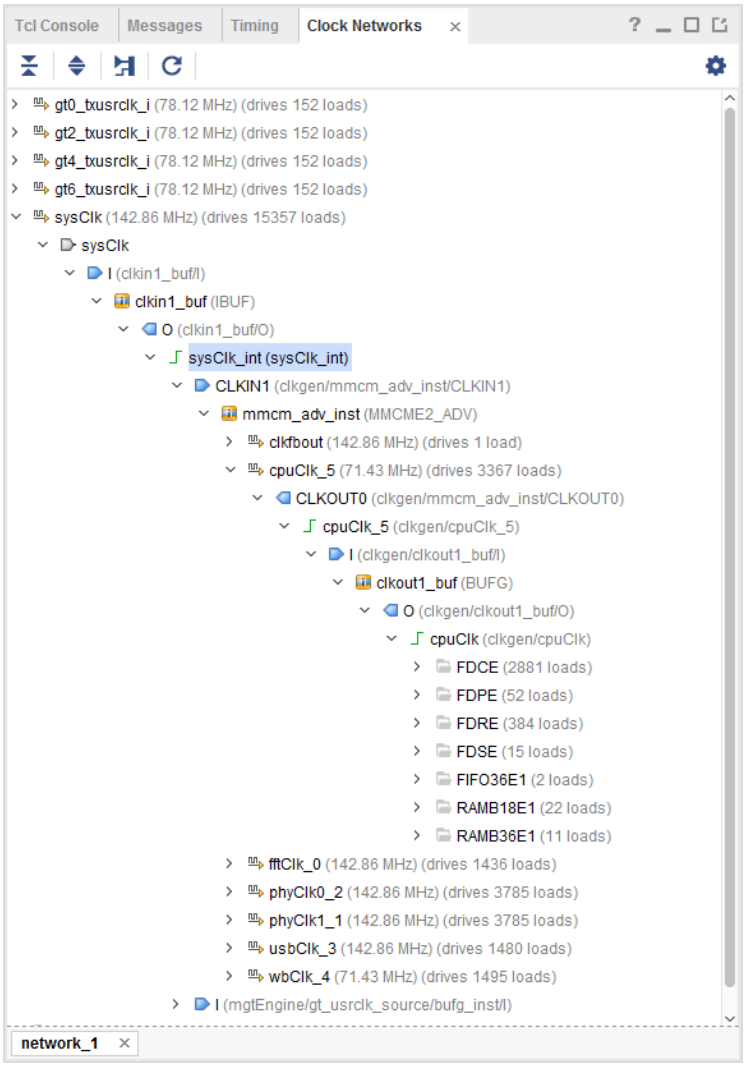
The Report Clock Network command can be run from one of the following:

- The Flow Navigator in the AMD Vivado™ IDE
- The Tcl command option:

```
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 103: **Clock Networks**



The clock trees have the following options:

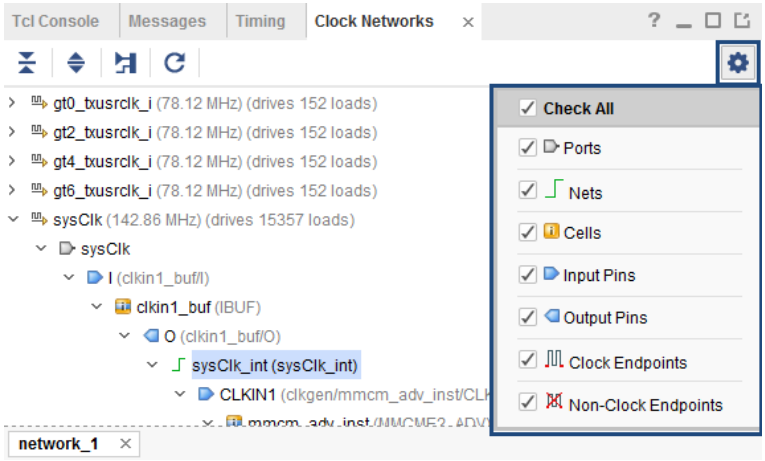
- They show clocks defined by the user or ones that are generated automatically by the tool.
- They can report clocks from I/O port to load.

  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.
- They can be used to find BUFGs driving other BUFGs.
- They show 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 104: **Clock Networks Filter**



Use the following instructions 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, choose one of the following options in the AMD Vivado IDE:

- **Reports → Timing → Report Clock Interaction**
- **Flow Navigator → Synthesis → Report Clock Interaction**
- **Flow Navigator → Implementation → Report Clock Interaction**

You can also run the equivalent Tcl command option: `report_clock_interaction -name clocks_1`

From the Tcl Console, use the `-cells` option to scope the report to one or more hierarchical cells. The report includes only those paths whose datapath section starts, ends, crosses, or is fully contained inside the specified cells.

Report Clock Interaction Dialog Box

In the AMD Vivado™ IDE, the Report Clock Interaction dialog box includes the following:

Results Name Field

Use this field to name the graphical report.

Equivalent Tcl option: `-name`

Command Field

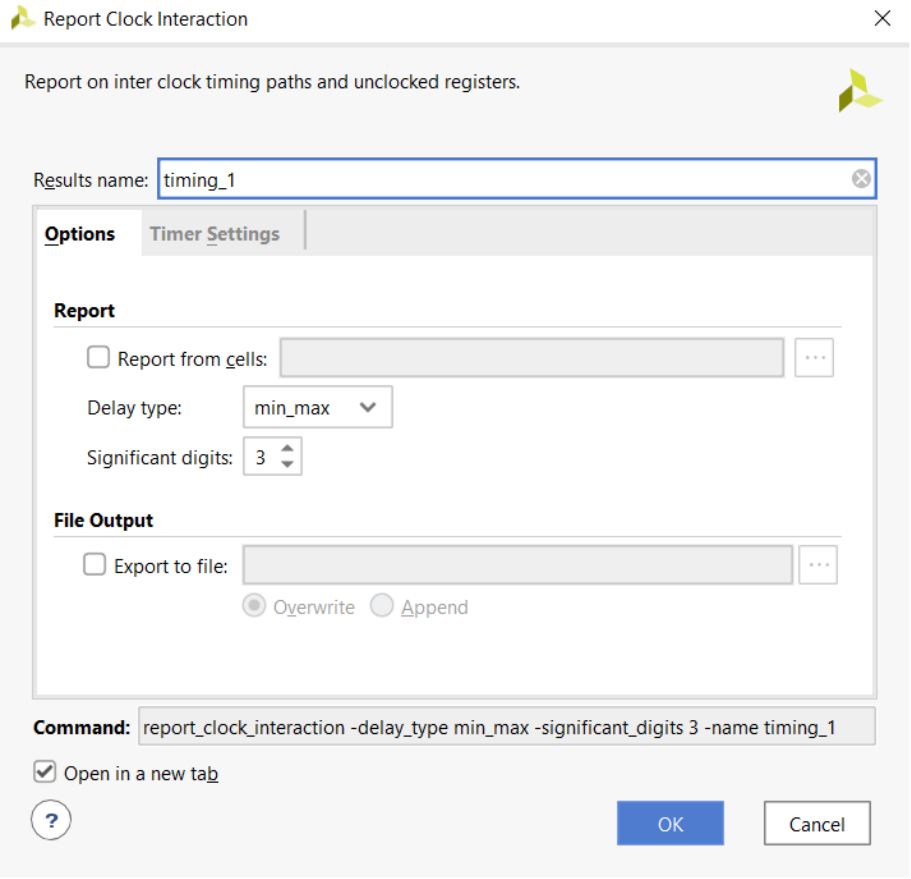
Displays the Tcl command equivalent for the selected dialog box options.

Open in a New Tab Check Box

Check this box to open the report in a new tab or leave it unchecked to replace the last opened tab in the Results window.

Options Tab

Figure 105: **Report Clock Interaction: Options Tab**



The Options tab contains the following fields:

- **Report from cells:** Use this to limit the report to selected cells in the design. The report includes only those paths that start, end, cross, or are fully contained inside the selected cells. The equivalent Tcl option is: `-cells`
- **Delay type:** Use this field to choose the type of delay analysis:
  - In synthesized designs, only max delay analysis (setup and recovery) runs by default
  - In implemented designs, both max and min delay analyses (setup, hold, recovery, and removal) run by default
  - To run only min delay analysis (hold and removal), select `min`

  The equivalent Tcl option is: `-delay_type`
- **Significant digits:** Sets the number of significant digits displayed in the report values. The default is three. The equivalent Tcl option is: `-significant_digits`

- **File Output:** Use these options to save the report:
  - Write Results to File: Specify a file to write the report results. The report still appears in the Clock Interaction window.
    - The equivalent Tcl option is: `-file`
  - Overwrite/Append: Choose whether to overwrite the existing file or append new results to it.
    - The equivalent Tcl option is: `-append`

Timer Settings Tab

For details on this tab, see Timer Settings Tab.

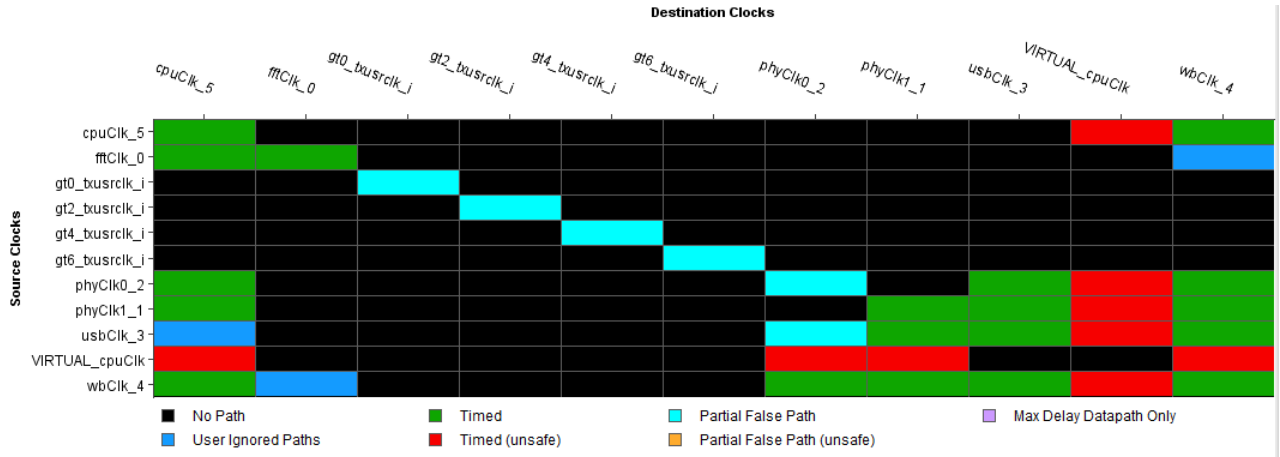
Details of the Clock Interaction Report

The Clock Interaction report analyzes timing paths that cross from one clock domain (the source clock) into another clock domain (the destination clock). Use this report to identify potential issues such as data loss or metastability.

After you run the Report Clock Interaction command, the results appear in the Clock Interaction window. The report displays as a matrix, with:

- Source clocks listed along the vertical axis
- Destination clocks listed along the horizontal axis

Figure 106: **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

This matrix format helps you quickly spot interactions between clock domains that might need further review.

Matrix Color Coding

The matrix in the Clock Interaction report uses color-coded tiles to show the relationship between source and destination clocks.

To view and customize the color coding in the Clock Interaction matrix:

1. Open the Clock Interaction report.
2. Locate the color-coded matrix showing source clocks (vertical) and destination clocks (horizontal).
3. To adjust background colors do one of the following:
   - Go to **Tools → Settings → Colors → Clock Interaction Chart**.
   - Click the gear icon on the Clock Interaction tab to access color settings directly.

   For more information, see Specifying Colors in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).
4. To hide the color legend, click the question mark icon on the left side of the matrix toolbar.

Color Meanings

- **Black: No Path:** No timing paths exist between the source and destination clocks. There is no clock interaction to report.
- **Green: Timed:** The clocks are synchronous. The timing engine confirms they share a common primary clock and have a simple period ratio.
- **Dark Blue: User Ignored Paths:** All paths between the source clock to the destination clock are excluded by user-defined false path or clock group constraints. If you run hold-only analysis (`-delay_type min`) and apply a `set_max_delay -datapath_only` constraint, the clock pair is classified as *Ignored* in the report and as *Auto Generated False Path* in the Inter-Clock Constraints column.
- **Light Blue: Partial False Path:** Some paths between the source clock to the destination clock are ignored due to false path constraints, but the clocks are still synchronous.
- **Red: Timed (Unsafe):** The clocks are asynchronous. They do not share a common primary clock or they have an unexpandable period. See *Vivado Design Suite User Guide: Using Constraints* (UG903) for more on asynchronous clock groups.
- **Orange: Partial False Path (Unsafe):** Similar to Timed (Unsafe), but at least one crossing path is ignored by a false path exception.
- **Gray: Max Delay Datapath Only:** All crossing paths are covered by a `set_max_delay -datapath_only` constraint.

IMPORTANT! *The matrix color shows the constraint relationship between clocks, not whether timing is met. A green tile means the clocks are properly timed and that they have a known phase relationship, not that the worst slack is positive.*

Clock Pair Classification

The Clock Pair Classification column explains why the tool considers two clocks synchronous or asynchronous. The tool stops checking as soon as it detects one of the conditions below, listed in order of priority:

- **Ignored:** A clock group, false path, or max delay datapath-only constraint covers the entire clock pair.

  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:** One or both clocks are virtual, so common primary clock or node checks do not apply
- **No Common Clock:** The clocks do not share a common primary clock
- **No Common Period:** The periods 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 clocks lack a known phase relationship
- **Clean:** None of the above conditions apply.

Filtering the Clocks

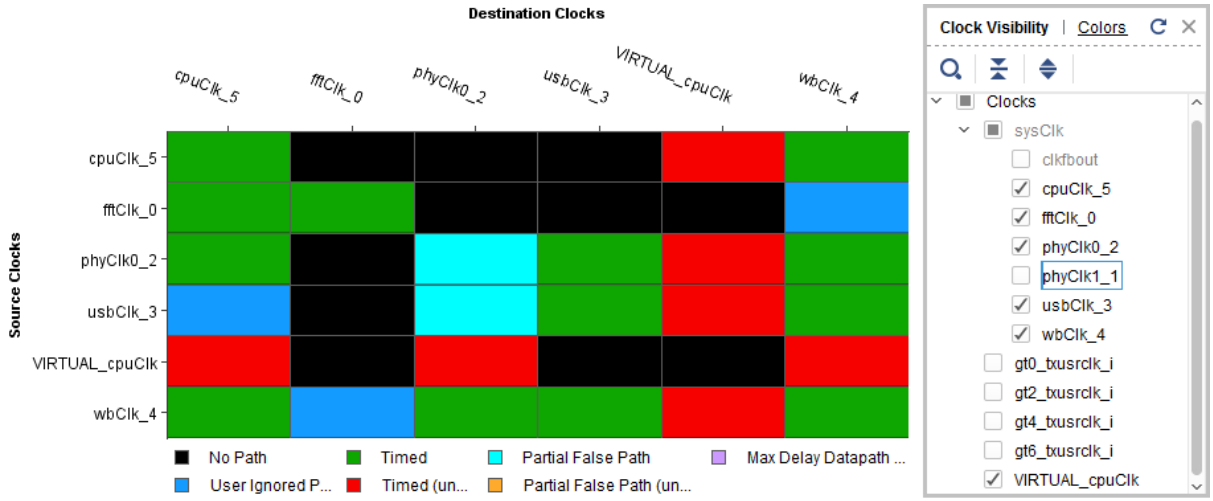
To filter the source clocks displayed in the Clock Interaction report:

1. Click the settings button to display the Clock Visibility panel.
2. Select the source clocks you want to view. The matrix automatically updates the list of destination clocks to match.

You can also simplify the view by hiding clocks that do not time any logical paths:

- Click the Hide Unused Clocks button on the toolbar
- These clocks are excluded by default because they do not affect WNS, TNS, WHS, or THS calculations

Figure 107: **Clock Interaction View Layers**



Clock Pairs Slack Table

The table underneath the matrix provides a detailed view of timing slack and other metrics for each source and destination clock pair. It includes setup/recovery (max delay) and hold/removal (min delay) information, along with constraint status and clock relationships that are not shown in the matrix. For more information, see Details of the Clock Interaction Report.

Using the Table

1. To sort the data by column:
   - Click a column header one time to sort by increasing value
   - Click again to reverse the sort order
2. To explore clock pair data:
   - Select a cell in the matrix to highlight the corresponding row in the table
   - Select a row in the table to highlight the related cell in the matrix
3. To export the table use the **Export to Spreadsheet** command to save the data as an XLS file
4. To run a timing report right-click on a selected clock pair and choose Report Timing for that source/destination pair

Table Columns

The table includes the following columns:

- **ID:** A unique identifier for each source/destination clock pair
- **Source Clock:** The originating clock domain
- **Destination Clock:** The receiving clock domain

Setup/Recovery (Max Delay) Columns

- **Edges (WNS):** Clock edges used to calculate worst negative slack
- **WNS (Worst Negative Slack):** The worst slack across all paths between the clock pair
- **Total Negative Slack:** The sum of all negative slack values across failing endpoints
- **Failing Endpoints:** The number of endpoints that fail to meet timing (contributes to TNS)
- **Total Endpoints (TNS):** The total number of endpoints analyzed for the clock pair
- **Path Req (WNS):** The required time for the path with the worst slack. Multiple requirements can exist within a clock pair depending on edge polarity or timing exceptions

  For more information, see Path Requirement.
- **Clock Pair Classification:** Describes the clock relationship. See Clock Pair Classification for details.
- **Inter-Clock Constraints:** Lists the constraints between the clocks, such as `set_false_path` or `set_clock_groups`. The possible values are listed in 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]
```

Hold/Removal (Min Delay) Columns

These appear only if min delay analysis is enabled:

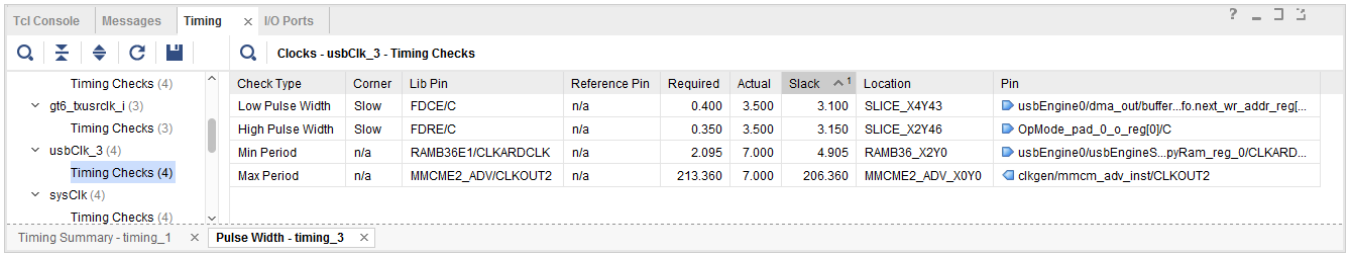
- **Edges (WHS):** Clock edges used to calculate worst hold slack
- **WHS (Worst Hold Slack):** The worst hold or removal slack for the clock pair
- **THS (Total Negative Hold Slack):** The sum of hold slack violations across endpoints
- **Failing Endpoints (THS):** The number of endpoints failing hold/removal timing
- **Total Endpoints (THS):** Total number of endpoints analyzed for hold/removal timing
- **Path Req (WHS):** Required time for the path with the worst hold slack. For more information, see Chapter 4: Timing Analysis.

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

Figure 108: **Report Pulse Width**



The Pulse Width Report checks whether your design meets the following requirements for each instance clock pin:

- Minimum period
- Maximum period
- High pulse time
- Low pulse time

It also verifies that the maximum skew requirement is met between two clock pins of the same instance in the implemented design (such as PCIe clocks).

The report does not include jitter or clock uncertainty in the slack calculations.

To generate this report:

1. Use the Tcl command: `report_pulse_width`
2. Use the `-cells` option to scope the report to specific hierarchical cells
   - Only pins within the selected cells are included
   - This option is not available in the Report Pulse Width GUI

***Note*:** In the AMD Integrated Software Environment (ISE) Design Suite, this check was previously called Component Switching Limits.

Report Datasheet

Use the Report Datasheet command to view the FPGA's operating parameters for system-level integration.

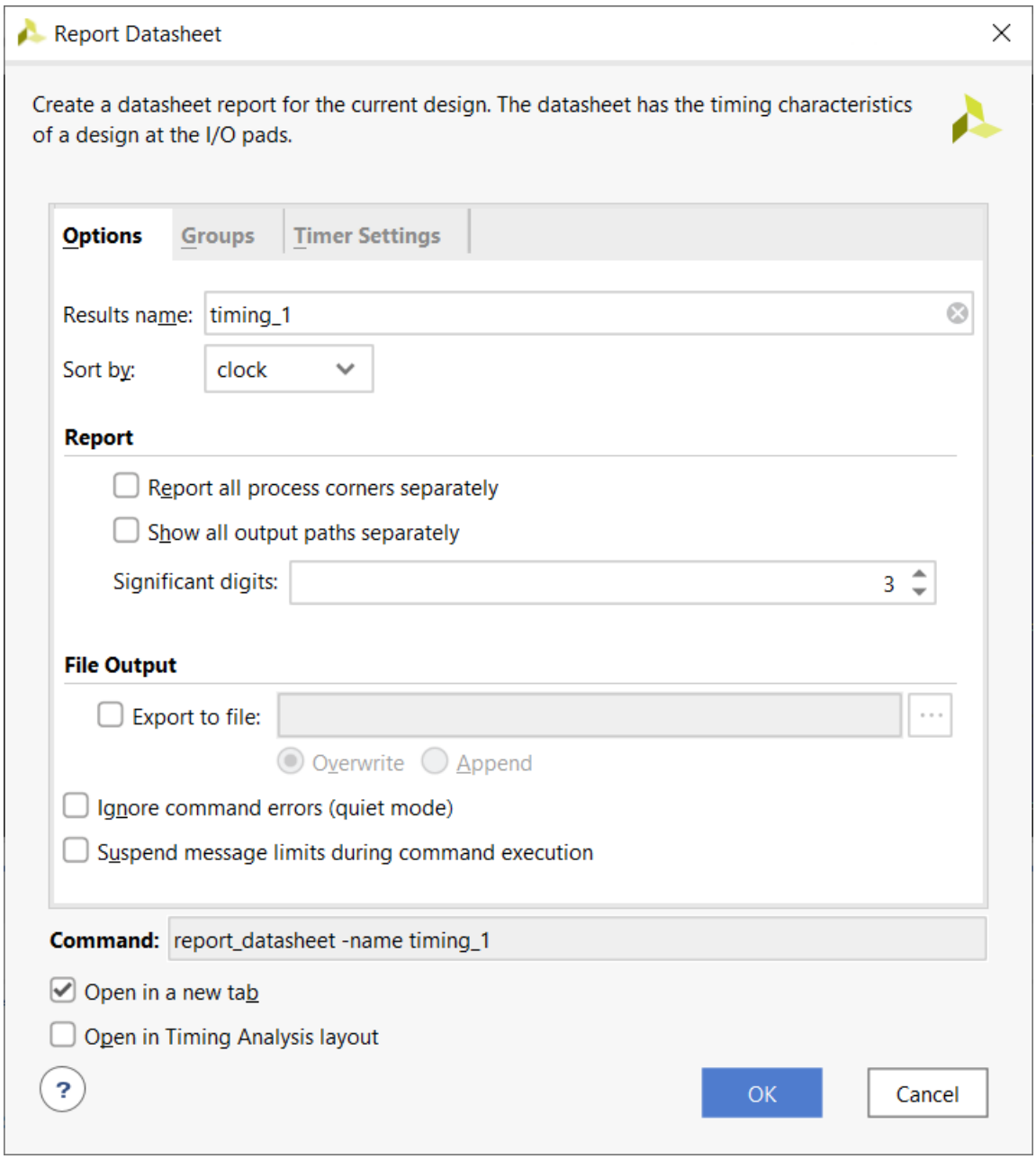
To open the Report Datasheet dialog box in the Vivado IDE:

1. Go to Reports
2. Select **Timing**
3. Click **Report Datasheet**

You can also run the equivalent Tcl command: `report_datasheet`

Report Datasheet Dialog Box: Options Tab

Figure 109: **Report Datasheet Options**



Use the Options tab to control report output and formatting. The available settings include the following:

- **Results name:** Enter a name for the report. The report opens in the Timing window under this name.

  The equivalent Tcl command option is: `-name`

- **Sort by:** Choose to sort results by port name or clock name.

  The equivalent Tcl command option is: `-sort_by`

- **Report all process corners separately:** Enable to display data for all process corners defined in the design

  The equivalent Tcl command option is: `-show_all_corners`

- **Show all output paths separately:** Report max and min values for paths using output enable, showing both `ON` and `OFF` states

  The equivalent Tcl command option is: `-show_oe_timing`

- **Significant digits:** Set the number of digits displayed in numerical results (default is 3)

  The equivalent Tcl command option is: `-significant_digits`

- **Export to file:** Specify a file name to save the report. By default, results appear in the Timing window

  The equivalent Tcl command option is: `-file`

- **Overwrite / Append:** Choose whether to overwrite the file or append to it

  The equivalent Tcl command option is: `-append`

- **Ignore command errors:** Run the command quietly, ignoring any errors and returning `TCL_OK`

  The equivalent Tcl command option is: `-quiet`

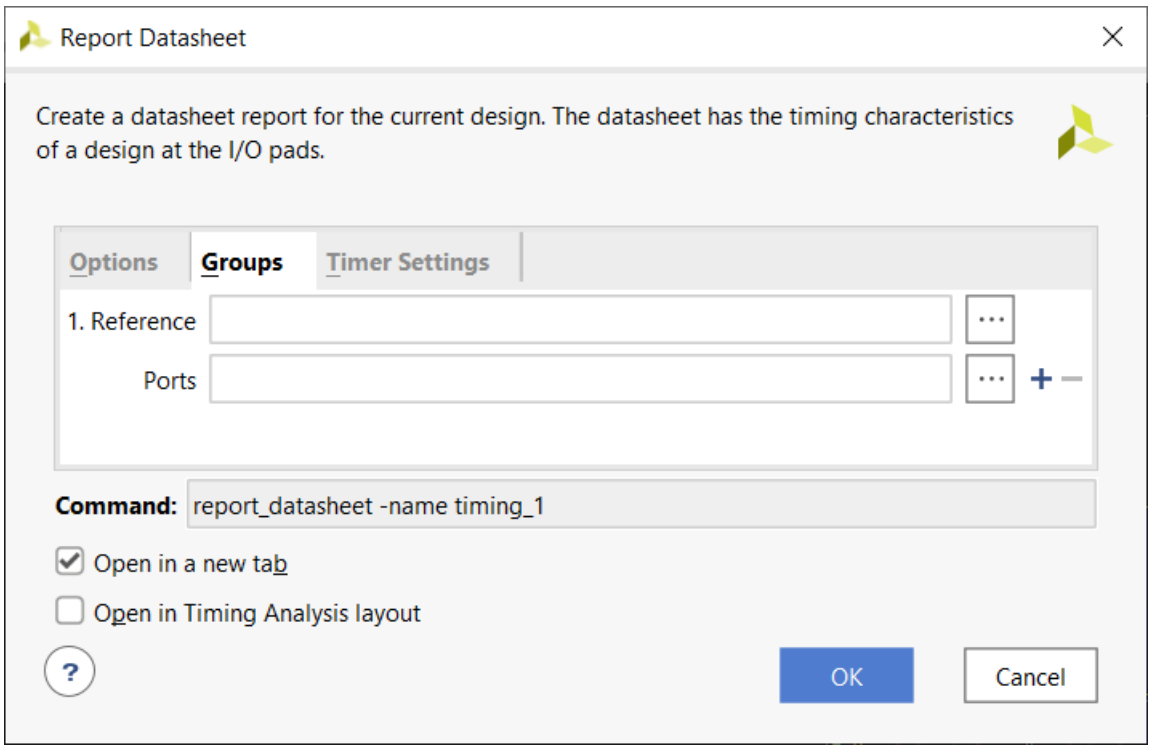
- **Suspend message limits during command execution:** Display all messages, overriding any output limits

  The equivalent Tcl command option is: `-verbose`

- **Command:** Displays the equivalent Tcl command option for the current settings
- **Open in a new tab:** Choose to open the results in a new tab or replace the most recent one
- **Open in Timing Analysis layout:** Switches the current view to the Timing Analysis layout

Report Datasheet Dialog Box: Groups Tab

Figure 110: **Report Datasheet: Groups Tab**



Use the Groups tab to define custom port groups for skew analysis. If you do not define any groups, the timer automatically identifies output port groups based on the launching clock and calculates skew from that clock.

The Groups tab includes the following:

- **Reference:** Specify the reference port used for skew calculation. Typically, this is the clock port of a source-synchronous output interface.

  The equivalent Tcl option is: `-group`

- **Ports:** Add additional ports to include in the same group.
  - Click the + (plus) button to create a new group with its own reference port
  - Click the – (minus) button to remove an existing group of ports

- **Reference:** Specifies the reference port for skew calculation. In most cases, this is a clock port of a source synchronous output interface.

  The equivalent Tcl option is: `-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

The Datasheet report provides system-level timing information based on multi-corner analysis. It includes setup, hold, delay, and skew metrics across inputs, outputs, clocks, and buses.

General Information

This section shows design and tool context at the time the report was generated:

- **Design Name:** Name of the design
- **Part:** Target AMD part and speed file
- **Version:** Vivado tool version used
- **Date:** Timestamp when the report was created
- **Command:** Tcl command used to generate the report

Input Ports Setup/Hold

For each input port, the report lists worst-case setup and hold requirements relative to the reference clock. It also shows the internal clock used to capture the input data.

Max/Min Delays for Output Ports

Displays the worst-case maximum and minimum output delays for each output port. The report includes the internal clock used to launch the output data.

Setup Between Clocks

For each clock pair, the report shows the worst-case setup requirement across all clock edge combinations.

Setup/Hold for Input Buses

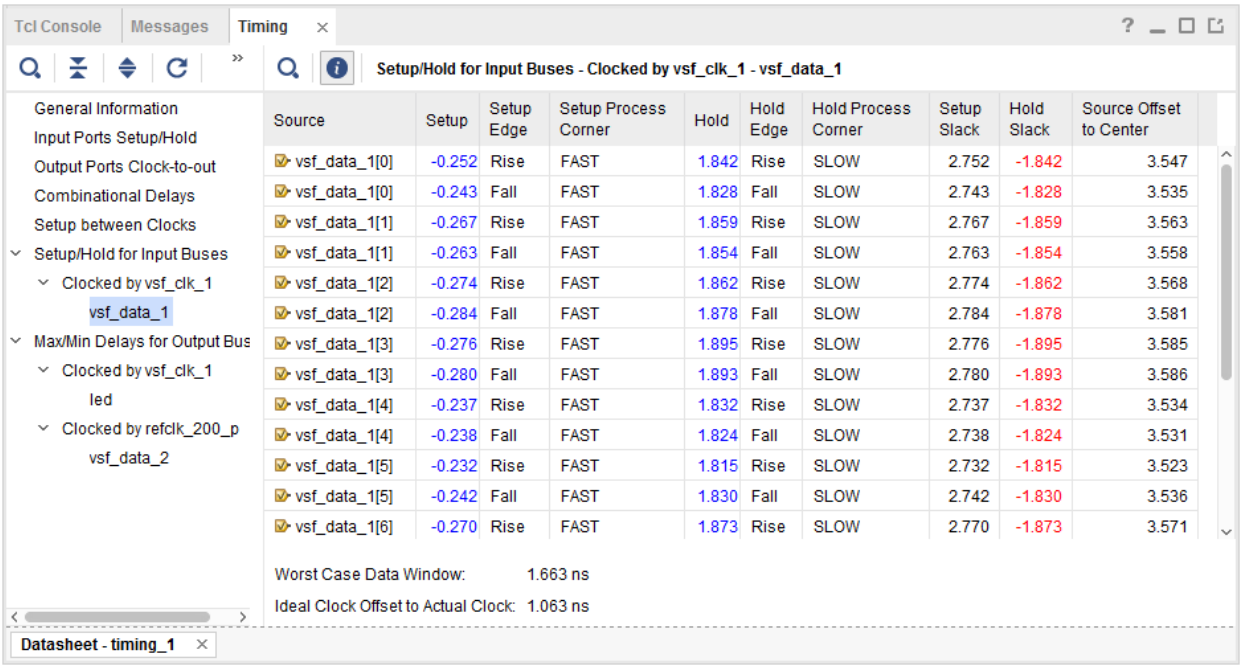
Vivado automatically detects input buses and reports the following:

- Worst-case setup and hold values

- Worst Case Data window = largest setup + largest hold
- Slack (if input ports are constrained)
- Optimal tap point (if IDELAY is used) for balanced setup and hold. The optimal tap point can be used to configure IDELAY for balanced setup and hold slack.
- Source offset = difference between the setup/hold window and the window defined by input delay and clock period. If the input clock is offset with this value, then it is 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 111: **Setup and Hold Delays for Input Buses**



***Note*:** The optimal tap point can be specified by using the following Tcl command:

```
set_property IDELAY_VALUE 13 [get_cells idelay_clk]
```

Max/Min Delays for Output Buses

Vivado detects output buses and reports the following:

- Worst-case maximum and minimum delay
- Bus skew, calculated by using one bit as a reference and measuring offset of all other bits. The worst offset is the skew for the entire bus.

Max/Min Delays for Groups

For source-synchronous output interfaces, use the Groups tab to define a custom group with a forwarded clock as the reference port. Vivado reports the following:

- Skew and delay relative to the reference port
- A structure similar to the output bus section, but based on your defined reference port

Note: If no groups are defined, this section might be hidden.

For example, you are grouping ports for DDR output skew calculation. To calculate output skew for a DDR interface, you can group multiple signals relative to a forwarded clock port. For example, if the group includes the following and the reference (forwarded) clock is `rldiii_ck_n[0]`, use the Tcl command below.

- `rldiii_a[0-19]`
- `rldiii_ba[0-3]`
- `rldiii_ref_n`
- `rldiii_we_n`

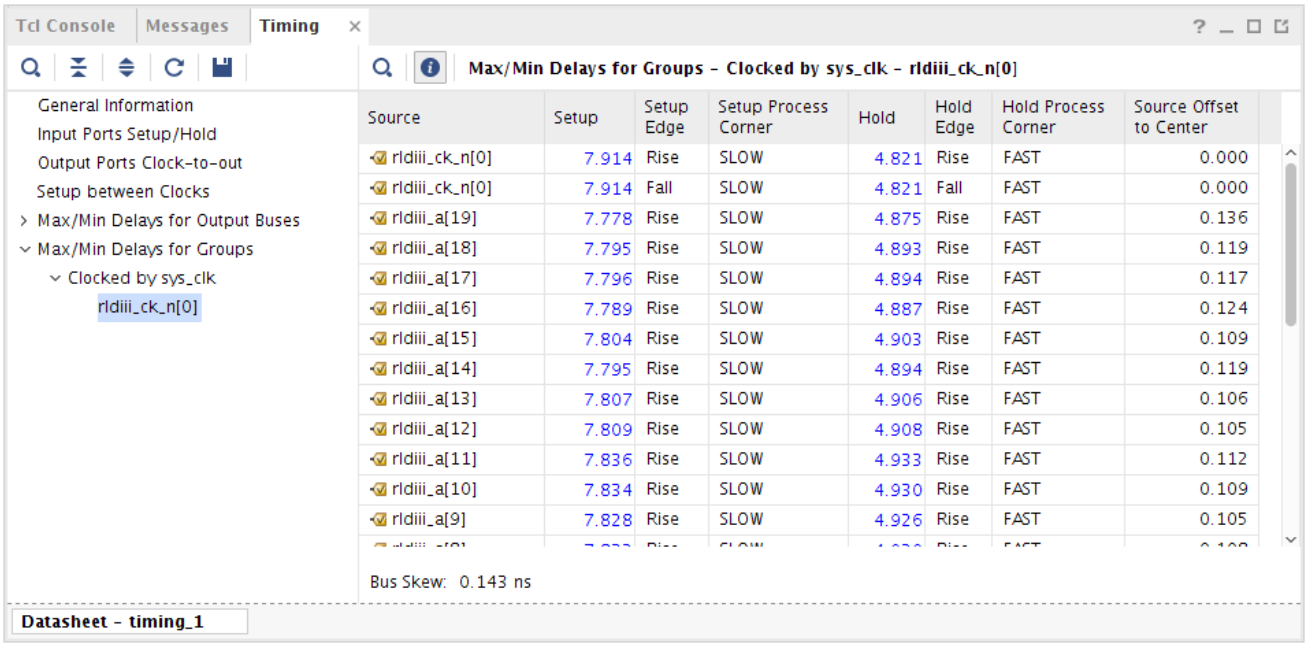
```
report_datasheet -group [get_ports {rldiii_ck_n[0] rldiii_a[*] rldiii_ba[*]
rldiii_ref_n rldiii_we_n}] -name timing_1
```

The first port in the group list is treated as the reference pin for skew and delay calculations.

Vivado uses multi-corner analysis to calculate worst-case data across all defined process corners. If you include the `-show_all_corners` option, the report shows the worst-case data separately for each corner.

The resulting datasheet report reflects timing behavior under a range of operating conditions. The following figure shows an example output for the DDR interface group.

Figure 112: **Report Data Sheet Max/Min Delay Example**



Report Exceptions

You can run the Report Exceptions command at any point after synthesis. Use it to identify and debug timing exceptions in your design.

This report includes the following:

- All timing exceptions that currently affect timing analysis
- All timing exceptions that are set but ignored because they are overridden by other exceptions

The command analyzes timing exceptions in the following order of precedence:

1. `set_clock_groups`
2. `set_false_path`
3. `set_max_delay` / `set_min_delay`
4. `set_multicycle_path`

Even though clock groups are not strict timing exceptions, they are included because they can override other exceptions.

What the Report Shows

The report helps you to do the following:

- Identify which exceptions are applied and which are ignored
- Understand if an exception is partially or completely overridden
- Identify what other constraint is overriding a particular exception
- Debug complex exception behavior where multiple constraints interact

Command Modes

The `report_exceptions` command supports several modes, including the following:

- Report active exceptions affecting timing
- Report ignored exceptions
- Report timing exception coverage
- Report invalid objects in `-from`, `-through`, or `-to`
- Write only valid exceptions to output file
- Write merged exceptions as interpreted by the timing engine to output file

Important Notes

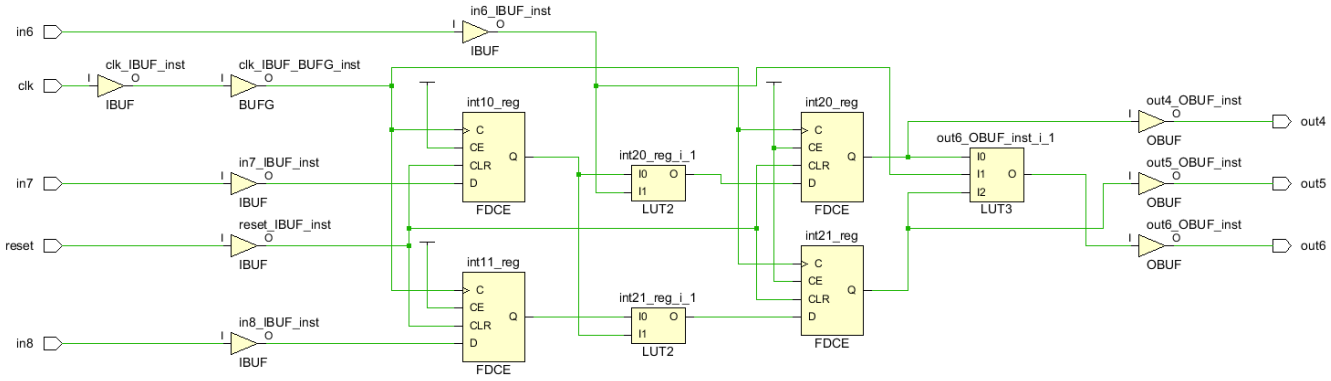
- If you use the `-from`, `-through`, or `-to` options, the report includes only those exceptions defined with the same combination.
- The patterns can differ, but there must be at least one object (cell, net, pin, or port) that matches in each of the `-from`, `-through`, and `-to` groups for the tool to recognize it as a valid exception.

For more information about the `report_exceptions` command line options, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835). For more information about the timing exception priority order, refer to XDC Precedence in the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Example: Reporting the Timing Exceptions Affecting the Timing Analysis

This example shows how to apply timing exceptions and use the `report_exceptions` command to identify how those constraints affect the design. The design is fully constrained, with clock and I/O delays defined relative to `clk`.

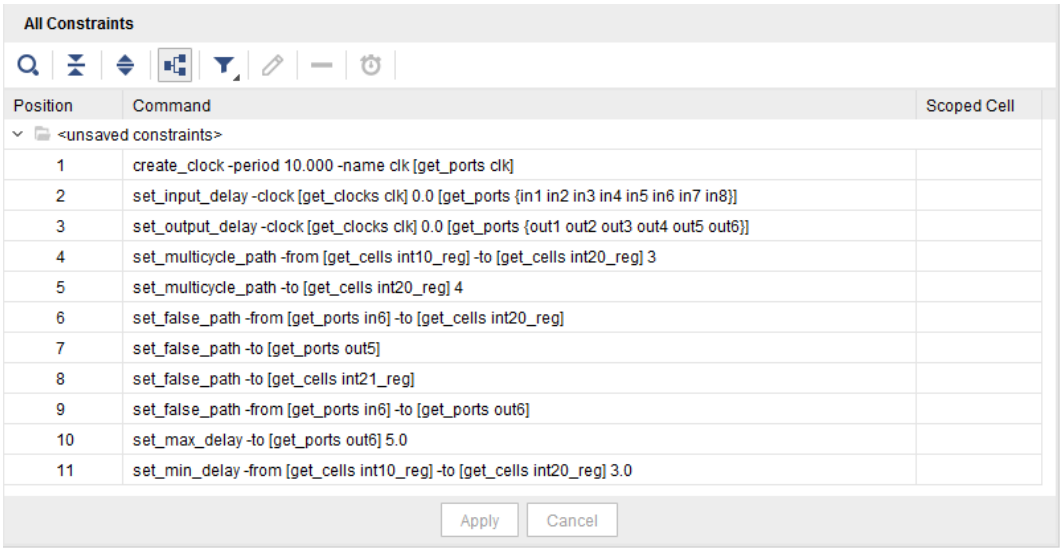
Figure 113: **Fully Constrained Design for Timing Exception Example**



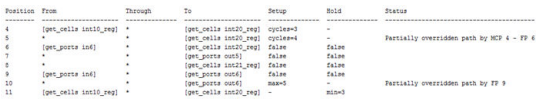
1. Select **Window → Timing Constraints**.
2. In the Timing Constraints window, add the following timing exceptions to the design:

```
set_multicycle_path 3 -from [get_cell int10_reg] -to [get_cell int20_reg]
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_cells 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 now displays each applied constraint and assigns a position number to each.



3. Run the `report_exceptions` Tcl command in the console.



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	

4. Review the Report Exceptions Output. The report shows the following:
   - **Position:** The constraint's position number, which matches the order in the Timing Constraints window
   - **From, Through, and To:** The pattern or object used with the `*-from`, `*-through`, or `*-to` options (an asterisk means the option wasn't used)
   - **Setup and Hold:** Which checks the constraint applies to, including the following:
     - `cycles=` for `set_multicycle_path`
     - `false` for `set_false_path`
     - `max=` for `set_max_delay`
     - `max_dpo=` for `set_max_delay -datapath_only`
     - `min=` for `set_min_delay`
     - `clock_group=` for `set_clock_group`
   - **Status:** Describes whether the constraint is partially or fully overridden, and which type of exception caused it.
     - `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.

Examples from the Report

- Position 5: `set_multicycle_path 4 -to [get_cell int20_reg]` is partially overridden by the following exceptions:
  - Position 4: `set_multicycle_path 3 -from [get_cell int10_reg] -to [get_cell int20_reg]`
  - Position 6: `set_false_path -from [get_ports in6]-to [get_cell int20_reg]`
- Position 10: `set_max_delay 5 -to [get_ports out6]` is partially overridden by the following exceptions:
  - Position 9: `set_false_path -from [get_ports in6] -to [get_ports out6]`

These messages help you understand how higher-priority exceptions affect lower-priority ones. Use this report to troubleshoot unintended overrides in complex constraint sets.

Reporting Overridden Scoped Timing Exceptions

The second mode of operation for the `report_exceptions` command identifies scoped timing exceptions that are partially or fully overridden by top-level constraints. Use the `-scope_override` option to generate this report.

```
report_exceptions -scope_override
```

***Note*:** This mode is only available for the text report from the Tcl Console. It does not generate a graphical report in the Vivado IDE.

This mode highlights scoped constraints, such as those created in IP cores, that are overridden by constraints at the top level of your design. However, it does not report scoped constraints that are overridden by the following:

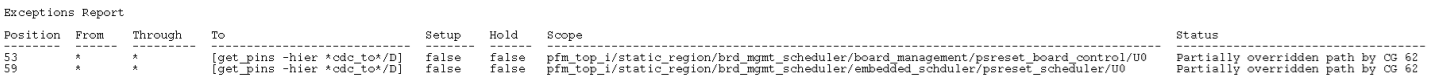
- Other scoped constraints from the same scope
- Scoped constraints from a different scope

For each overridden timing exception, the report includes the following:

- Constraint position number
- Patterns specified with `-from`, `-through`, and `-to` options
- Type of constraint overriding setup and/or hold
- Scope in which the overridden constraint is defined
- List of top-level constraint position numbers that are overriding the scoped constraint

Use this mode to detect and resolve conflicts between scoped IP constraints and user-defined top-level constraints.

Figure 114: **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 CG 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 CG 62
```

Reporting the Timing Exceptions Being Ignored

Use the `-ignored` option with the `report_exceptions` command to list all timing exceptions that the timing engine ignores.

```
report_exceptions -ignored
```

To illustrate how this mode works, add the following timing exceptions in addition to those used previously:

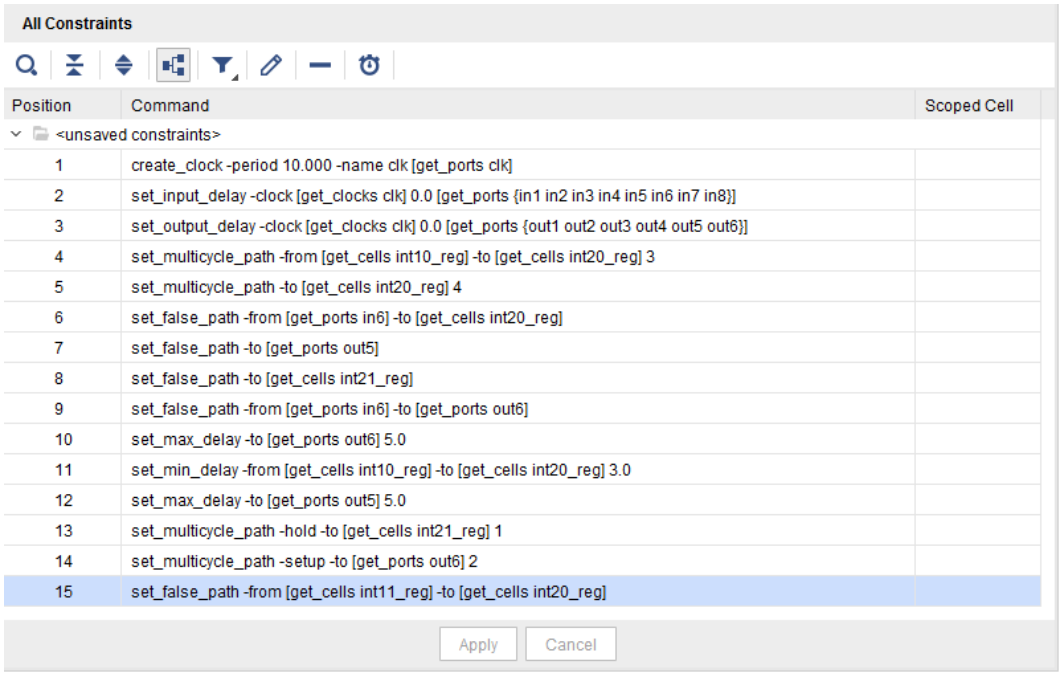
```
set_max_delay 5 -to [get_ports out5]
set_multicycle_path 1 -hold -to [get_cells int21_reg]
set_multicycle_path 2 -setup -to [get_ports out6]
set_false_path -from [get_cells int11_reg] -to [get_cells int20_reg]
```

These constraints are ignored for one of the following reasons:

- They are already covered by a higher-priority or more specific constraint
- They refer to a non-existent path (for example, no physical connection between `int11_reg` and `int20_reg`)

After adding these four constraints, the Timing Constraints window looks like the following figure.

Figure 115: **Timing Constraints Window**



Run `report_exceptions -ignored` to view which constraints are being ignored and why.

Figure 116: **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
```

The following is displayed in the output:

- The Status column explains the reason a constraint is ignored
- The report helps you identify redundancy or conflicts in your constraint file
- Use this to clean up constraints or refine your timing exception strategy

This mode is especially helpful when working with large or complex designs where overlapping constraints might silently override each other.

Reporting the Timing Exceptions Coverage

Use the `-coverage` option with the `report_exceptions` command to generate a detailed coverage report 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.

```
report_exceptions -coverage
```

For each valid timing exception, the report shows the following:

- Constraint position number
- Number of objects selected by the `-from`, `-through`, and `-to` options
- Coverage percentage, which compares the number of pins affected by the constraint to the total number of pins specified

  Note: If you use cells (instead of pins) in your `-from`, `-through`, or `-to` options, Vivado expands those to pins. This expansion often reduces the reported coverage, because not all pins within the cells are tageted by the exception.

The following figure shows the exceptions coverage report.

Figure 117: **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 has no valid path between the specified startpoints and endpoints, the coverage report shows 0.0%. For example, if constraint position 15 shows 0.0% coverage, it indicates that no path exists. This matches what you see in the `report_exceptions -ignored` report, where the same constraint appears as a Non-Existent Path.

You can use the coverage report to improve how you write timing exceptions. For instance, consider the following constraint:

```
set_multicycle_path -setup 2 -from [all_registers] -to [get_cells
cpuEngine/or1200_cpu/or1200_ctrl/ex_insn_reg[*]]
```

If the coverage for the `-from` option is only 0.95%, even though `all_registers` returns 15,901 cells, it means that only a small subset of those registers have paths to the destination.

To improve efficiency and clarity, refine the `-from` list to include only the registers that have paths to `ex_insn_reg[*]`. This prevents over-constraint and improves the accuracy of your timing exceptions.

Figure 118: **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, the `-from` option returns 15,901 cell objects using `all_registers`, but the coverage is only 0.95%. This low percentage indicates that most of the specified registers do not have timing paths to the `cpuEngine/or1200_cpu/or1200_ctrl/ex_insn_reg[*]` cells.

To improve the efficiency of this constraint, refine the `-from` list to include only the registers that actually have paths to the destination cells. This reduces unnecessary coverage and results in a more accurate and effective timing exception.

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.
```

To resolve this, choose a valid start point as a main or generated clock pin or port, a clock pin of a sequential cell, or a primary input/inout port. 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 119: **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

You can use the `report_exceptions -write_valid_exceptions` command to export only the timing exceptions that cover at least one valid timing path. This export includes the following:

- Only valid startpoint and endpoint pins
- Expanded object collections, as resolved in memory by the Vivado timer
- Constraints that are actively applied to timing paths

Usage Notes

1. The export excludes timing constraints created with `set_clock_groups` and `set_bus_skew`.

2. Use this report with the coverage report (`-coverage`) to refine the object patterns used in your timing exceptions.
3. This helps ensure your constraints are both effective and targeted to meaningful paths.

Example

The following figure illustrates `report_exceptions -write_valid_exceptions` from the case described in Reporting the Ignored Objects, where two `set_false_path` constraints were applied:

Figure 120: **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

Use the `report_exceptions -write_merged_exceptions` command to export the list of timing exceptions as processed by the static timing analysis (STA) engine.

The Vivado timing engine merges similar or overlapping timing exceptions internally to reduce memory usage and improve runtime. This merged view reflects how the STA engine actually interprets and applies your constraints.

***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 shows the results of running `report_exceptions -write_merged_exceptions` on the two `set_false_path` constraints described in the Reporting the Ignored Objects section. In this example, the second `set_false_path` constraint includes all pins of the register because the `-from` and `-to` options use the pattern `int21_reg/*` with the `get_pins` command. The merged report reflects how the Vivado timing engine combines these constraints for analysis.

Figure 121: **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

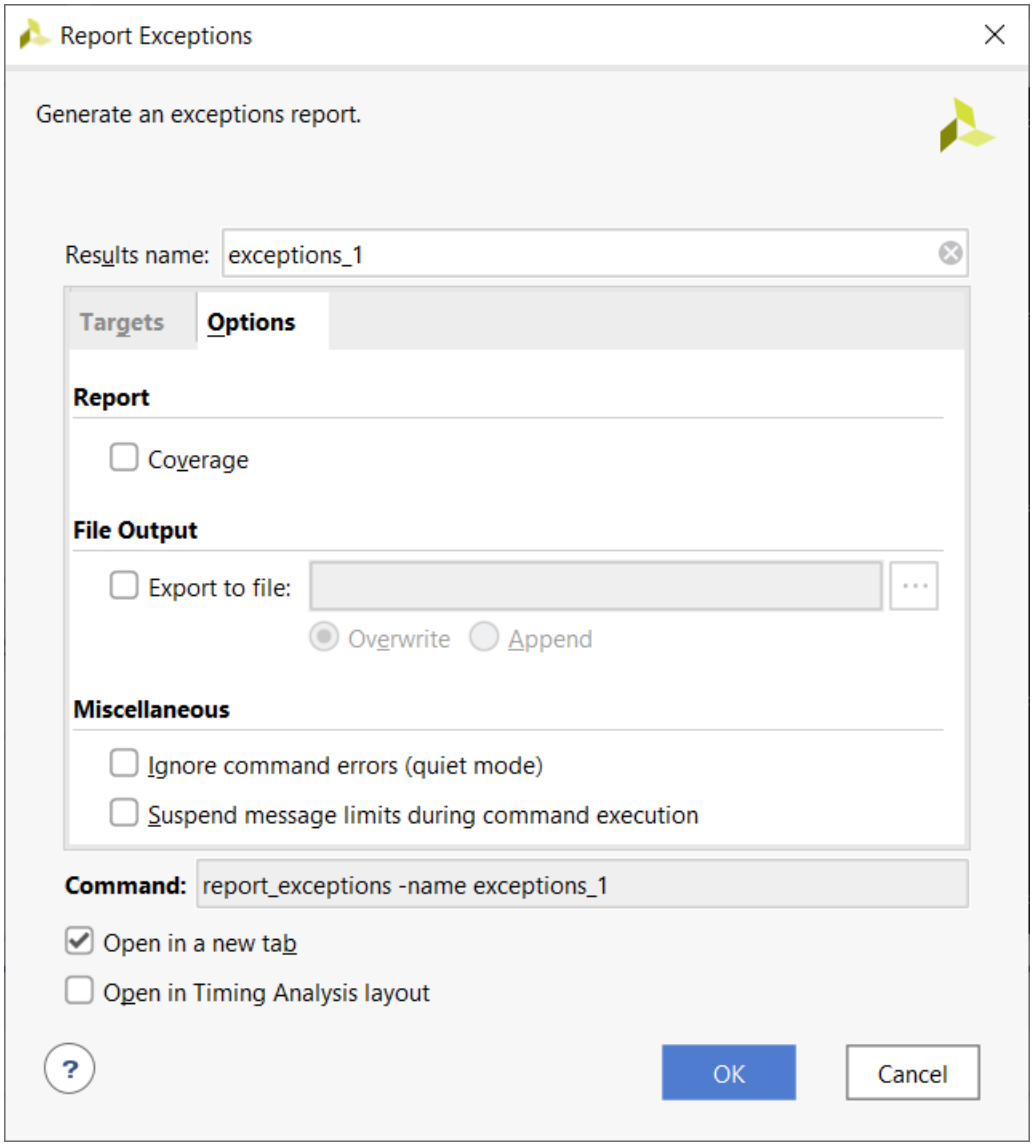
Report Exceptions Dialog Box

To open the Report Exceptions dialog box in the Vivado IDE, select **Reports → Timing → Report Exceptions**.

The GUI version of the report consolidates multiple tables into a single run. This saves you from running several separate `report_exceptions` commands with different options. However, because it gathers more data in one pass, the GUI report can take longer to generate compared to running the Tcl command in the console.

Report Exceptions Dialog Box: Options Tab

Figure 122: **Report Exceptions Dialog Box: Options Tab**



The Options tab in the Report Exceptions dialog box includes the following sections:

- **Report:** Coverage: Adds exception coverage data to the detailed tables in the report. This option can increase report runtime significantly.

  The equivalent Tcl option is: `-coverage`

- **File Output:** Export to file: Saves the report to a specified file. By default, Vivado writes the report to the Timing window.

  The equivalent Tcl option is: `-file`

- **Overwrite or Append:** Choose whether to overwrite the file or append new content to it.

  The equivalent Tcl option is: `-append`

- **Miscellaneous:** Ignore command errors: Runs the command quietly, suppresses error messages, and returns `TCL_OK` even when errors occur.

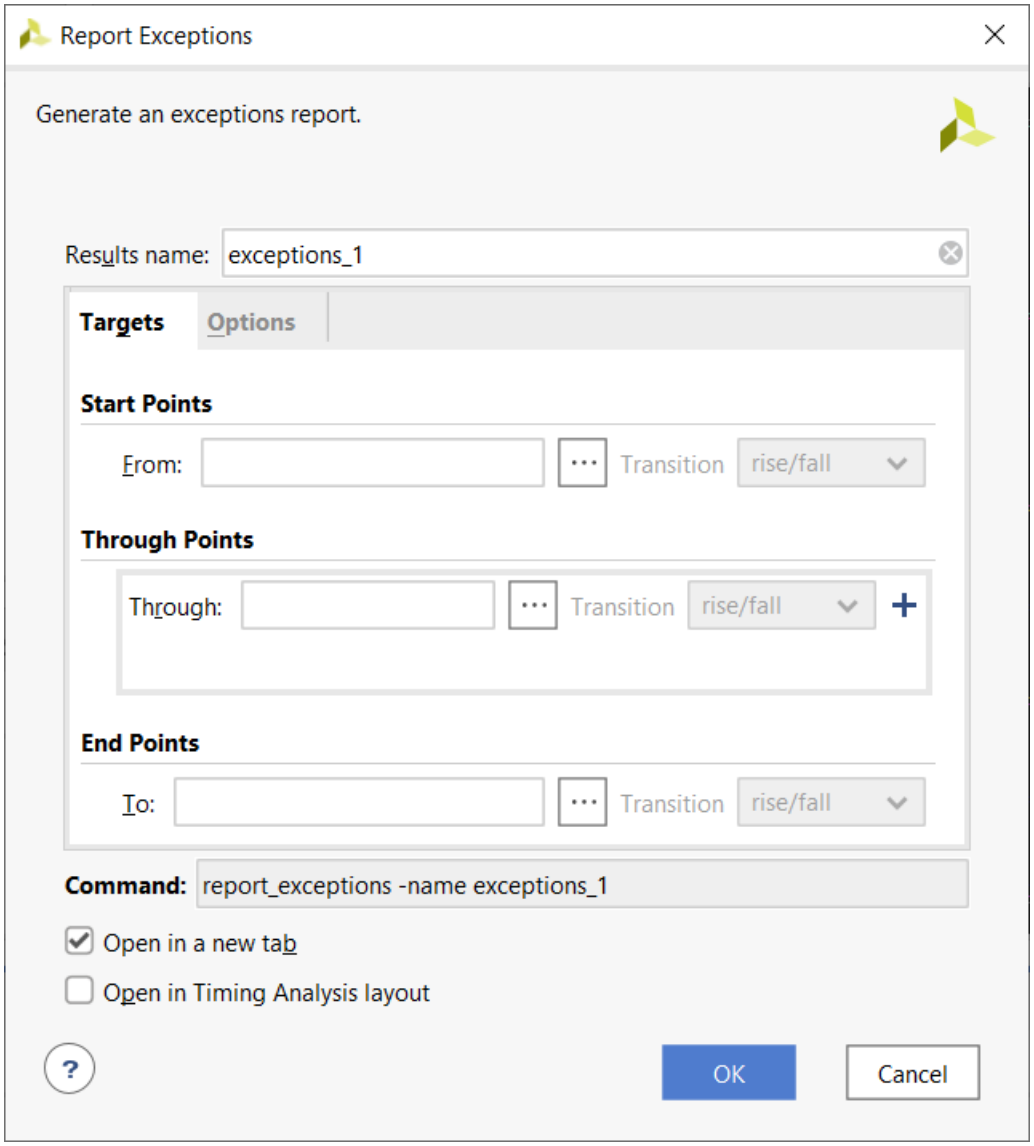
  The equivalent Tcl option is: `-quiet`

- **Suspend message limits during command execution:** Temporarily disables message limits so that all messages are returned.

  The equivalent Tcl option is: `-verbose`

Report Exceptions Dialog Box: Targets Tab

Figure 123: **Report Exceptions: Targets Tab**



Report Exceptions provides several filtering options that let you target specific timing exceptions or groups of exceptions. You can filter by the following:

- Start Points
- Through Points
- End Points

Descriptions of these options can be found in the Targets Tab section.

When you apply any of these filters, the report includes only the timing exceptions that are strictly defined with the specified start, through, and end points.

Details of the Exceptions Report

The Exceptions Report in Vivado provides a complete view of your timing exceptions, their status, and their coverage. It includes the following sections:

- General Information
- Summary
- Exceptions
- Ignored Objects Section

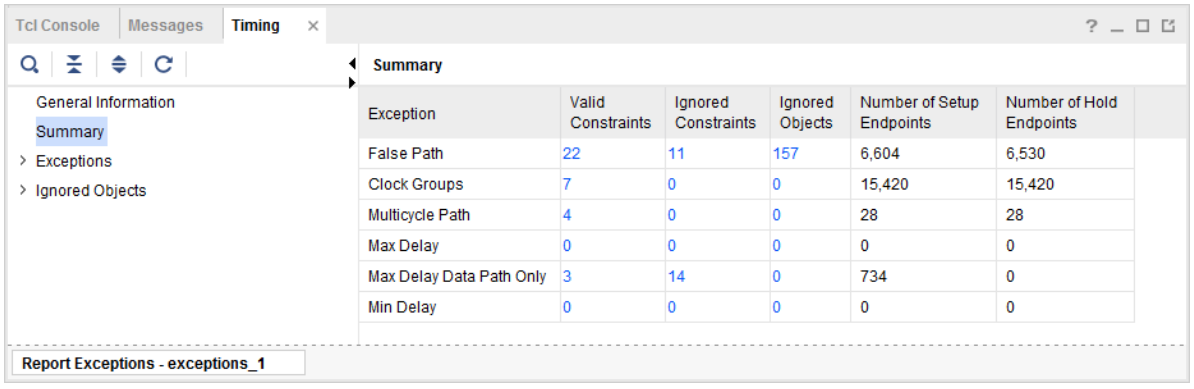
General Information

This section shows the context in which the report was generated, including the following:

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

Summary

Figure 124: **Report Exceptions: Summary Section**



This section summarizes all timing exceptions and clock group constraints. For each constraint type, it reports the following:

- Number of valid constraints
- Number of ignored constraints

- Number of ignored objects
- Number of covered setup and hold endpoints

This summary gives more detail than the `report_exceptions -summary` Tcl command. You can click hyperlinks in the table to view the corresponding entries in the Exceptions or Ignored Objects sections. Valid and ignored constraints link to the same detailed Exceptions table.

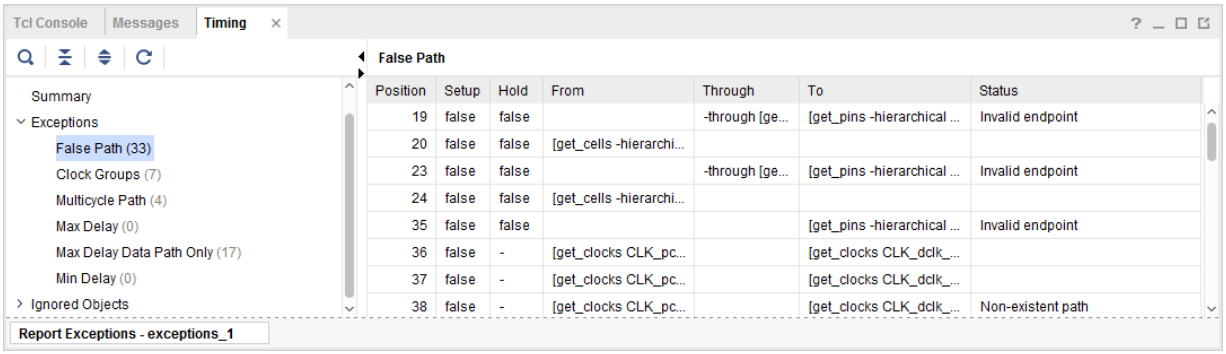
***Note*:** An exception is considered ignored if there is no valid path between the `-from`, `-through`, or `-to` objects, or if the exception is fully overridden.

Exceptions

This section contains detailed tables for each type of timing exception. You can access these tables through the links in the Summary table. The format of the detailed tables depends on whether the Coverage option has been selected or not.

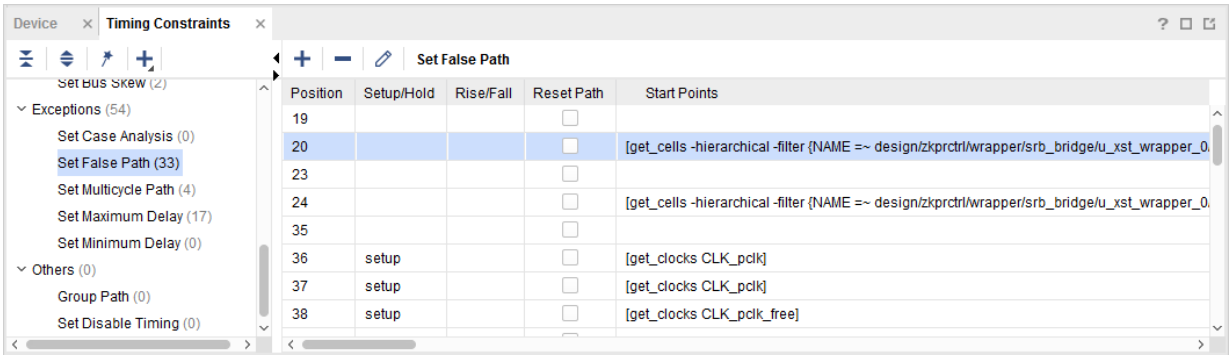
Without Coverage Enabled

Figure 125: **Report Exceptions: Detailed Table Without Coverage**



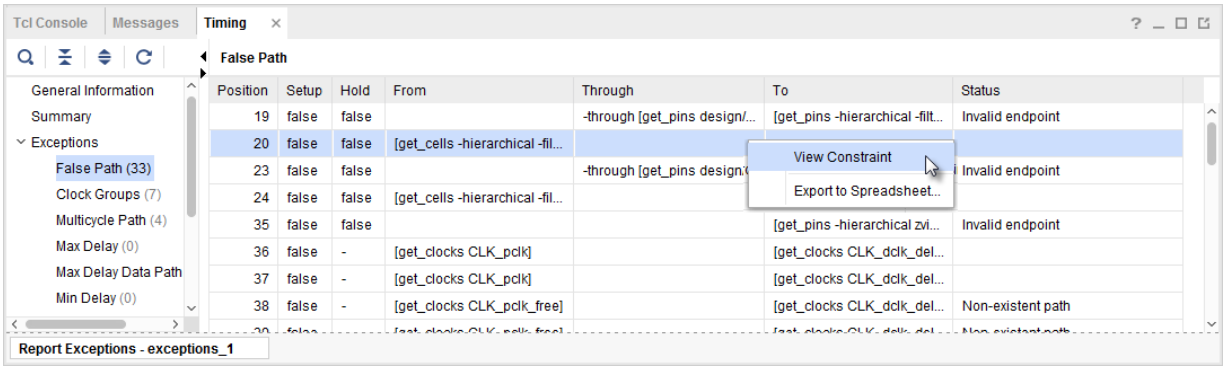
The Position column shows the constraint's position number, matching the timing constraints editor (TCE).

Figure 126: **Timing Exception Inside Timing Constraint Editor**



To go to the corresponding constraint in the TCE, double-click a row or right-click the row and select **View Constraint**.

Figure 127: **Report Exceptions Context Menu**

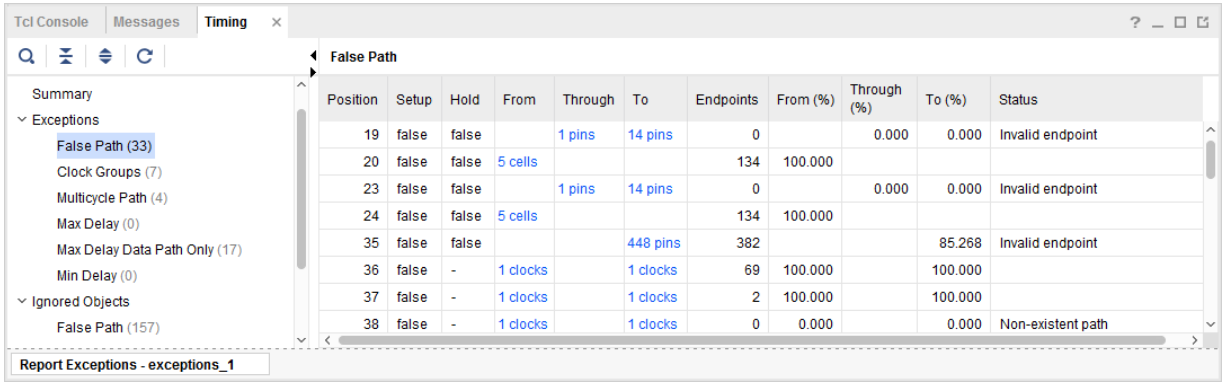


The From, Through, and To columns display the original patterns used. The same patterns are reported inside TCE.

With Coverage Enabled

The following figure shows an example of a detailed table with the Coverage option selected inside the Report Exception GUI.

Figure 128: **Report Exceptions: Detailed Table with Coverage**



- The From, Through, and To columns include clickable hyperlinks to the targeted objects (cells, nets, pins, ports, or clocks)
- Press **F4** to open the schematic after selecting one or more objects
- Coverage-specific columns (From (%), Through (%), To (%)) show the percentage of objects actually affected
- The Status column explains the state of the constraint, such as the following:
  - Invalid endpoint

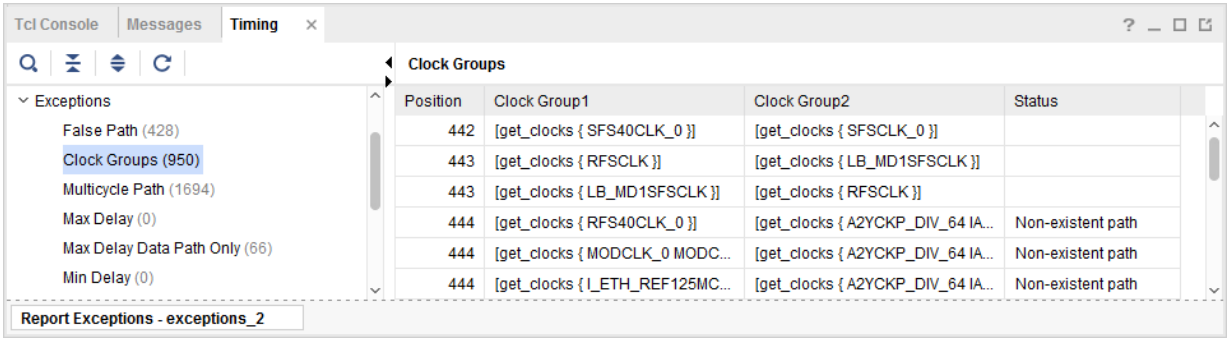
- Partially overridden
- Non-existent path
- Totally overridden

***Note*:**

- Vivado calculates coverage in the order: From → Through → To
- If coverage is 0% at one level, all levels below it show 0%
- A constraint with 0% coverage is considered invalid
- Pins tied to VCC or GND are reported as invalid

Clock Groups

Figure 129: **Detailed Table for Clock Groups**



Clock group constraints are not defined using `-from`, `-through`, or `-to`, so their table format is different.

- Each row lists a unique clock pair within the constraint
- The same constraint can span multiple rows, depending on how many clock pairs are covered
- All rows share the same Position value

The constraint position number 443 in the previous figure is defined as follows:

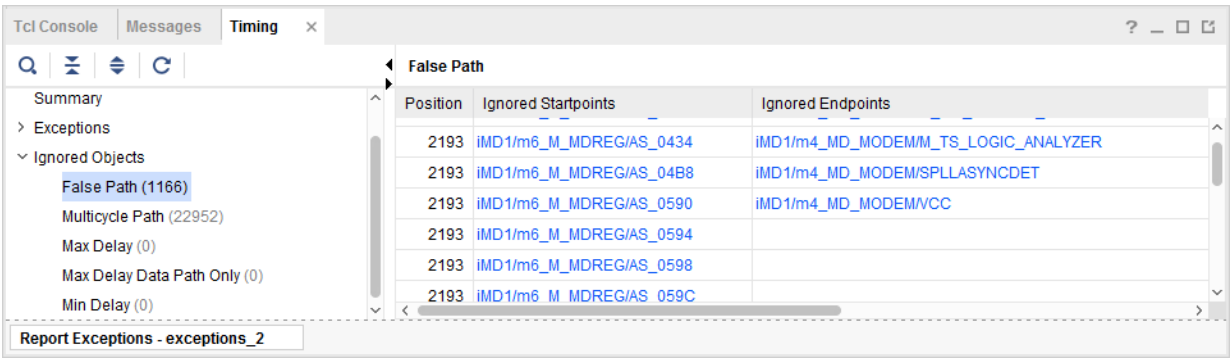
```
set_clock_groups -physically_exclusive -group RFSCLK -group LB_MD1SFSCLK
```

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 lists invalid startpoints and endpoints that Vivado excluded from each constraint. This is equivalent to running `report_exceptions -ignored_objects` from the Tcl Console.

Figure 130: **Report Exceptions: Ignored Objects**



- The Position column matches the constraint's entry in the TCE
- Double-click or right-click a row to jump to or view the constraint
- Ignored Startpoints and Ignored Endpoints columns show which pins were rejected
- A constraint can span multiple rows based on the number of ignored pins
- Use the hyperlinks to select the invalid objects. After selection, you can review them in the Properties pane or press **F4** to open the schematic

This section helps you identify and correct object queries that return invalid or unsupported pins.

Report Clock Domain Crossings

The Clock Domain Crossings (CDC) report performs a structural analysis of all clock domain crossings in your design. Use this report to identify potentially unsafe CDCs that could lead to metastability or data coherency issues.

While the CDC report is similar to the Clock Interaction Report, it focuses on the structure of crossings and their associated constraints. It does not provide timing slack information.

Use the `report_cdc` command to generate the CDC report from the Tcl Console.

To scope the report to one or more hierarchical cells, use the `-cells` option:

```
report_cdc -cells [get_cells path_to_instance]
```

When scoped, the report includes any CDC in which either the source or destination pin is located within the specified cell(s).

***Note*:** The `-cells` option is **not available** in the Report CDC GUI. Use the Tcl Console for scoped analysis.

Overview

Before generating the CDC report, make sure your design is fully constrained and all clocks are defined. The `report_cdc` command only analyzes paths where both the source and destination clocks are defined.

What Report CDC Analyzes

- All paths between asynchronous clocks
- Only paths between synchronous clocks that include one of the following timing exceptions:
  - `set_clock_groups`
  - `set_false_path`
  - `set_max_delay -datapath_only`

If a synchronous clock pair does not have one of these exceptions, the tool assumes the path is safely timed and skips CDC analysis for that path.

IMPORTANT! *The CDC engine performs structural checks only. It does not consider net or cell delays in the analysis.*

Terminology

The terms safe, unsafe, and endpoints have different meanings in the context of cross domain crossing (CDC) analysis versus inter-clock timing analysis.

In CDC analysis:

- An asynchronous crossing is considered safe when proper synchronization circuitry is used to prevent metastability.
  - A safe single-bit CDC uses a synchronizer, typically a chain of registers clocked and controlled (CE) by the same signals.
  - A safe multi-bit CDC uses structures such as MUX hold circuitry or Clock Enabled Controlled circuitry.
- An asynchronous crossing is considered unsafe when the CDC analysis engine does not detect a recognized safe synchronization structure on the path.

The number of endpoints reported in a CDC report between two clock domains can differ from what is shown in timing analysis reports:

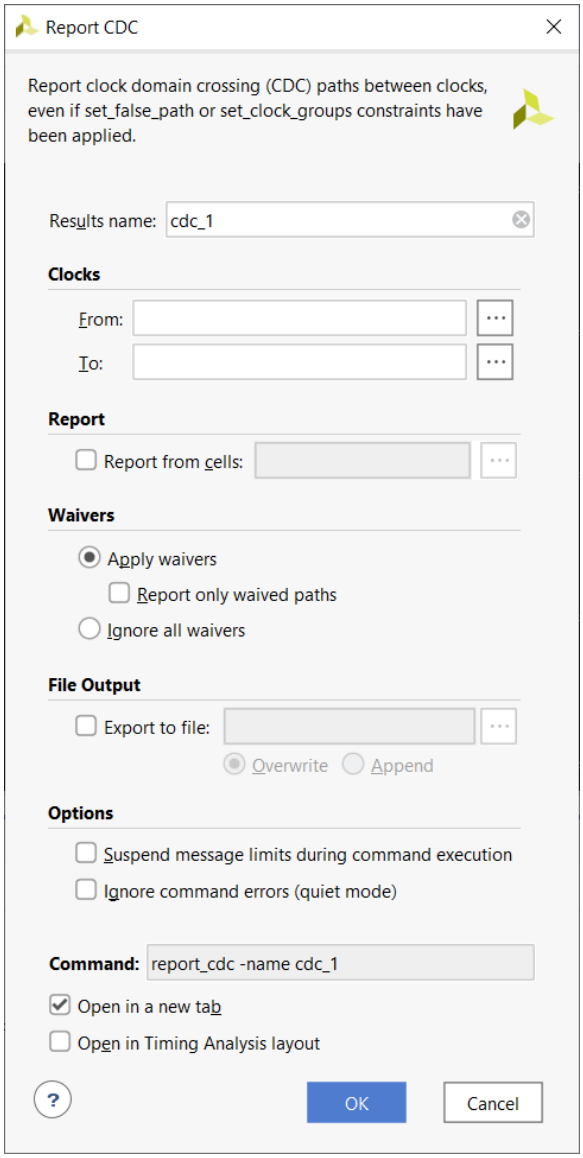
- An asynchronous reset synchronizer can include multiple timing endpoints but is reported as a single CDC endpoint.
- A multi-bit CDC contains several single-bit crossings but appears as one CDC endpoint. However, other timing reports list each bit as a separate timing path endpoint.

IMPORTANT! *Do not compare the number of endpoints between* `report_clock_interaction` *and* `report_cdc`.

- `report_clock_interaction` classifies crossings as safe or unsafe based on whether the timing engine can calculate slack that reflects the worst-case hardware behavior.
- `report_cdc` classifies crossings as safe or unsafe based on whether known CDC synchronization circuits are present in the design.

Running Report Clock Domain Crossings

Figure 131: **Report CDC Dialog Box**



You can run the Report CDC command from either the Vivado IDE or the Tcl Console. When you run Report CDC from the Vivado IDE, it provides all the details for the CDC paths between the specified clocks by default.

1. Select **Reports → Timing → Report CDC** or use the `report_cdc -name cdc_1` in the Tcl Console.
2. In the Report CDC dialog box, fill in the following fields:
   - **Results Name:** Enter a name for the report. This name appears in the Timing window or in the file output.Equivalent Tcl option: `-name <windowName>`
   - **Clocks (From/To):** Use the Clocks From and To fields to optionally specify the source and/or destination clocks for the CDC analysis. This narrows the scope to specific clocks and creates a more readable report. Click the Browse button to search for clock objects. Equivalent Tcl options: `-from <clockNames>` and `-to <clockNames>`
   - **File Output:** Specify a file to write the results. You can choose whether to overwrite the file or append to it. Equivalent Tcl options: `-file <fileName>` and `-append`
   - **Options:** Set additional behaviors for the report:
     - Suspend message limits during execution

       Equivalent Tcl option: `-verbose`
     - Ignore command errors

       Equivalent Tcl option: `-quiet`

***Note*:** By default, the Tcl version only prints the Summary by Clock Pairs table. To include full CDC path details, add the `-details` option:

```
report_cdc -details
```

***Note*:** Enabling the `-details` option can generate large reports or log files depending on the design size. Use it only when detailed analysis is needed.

Understanding the Clock Domain Crossings Report Rules

Vivado's `report_cdc` command identifies and classifies CDC paths by matching each path to a known CDC topology. Each topology is associated with one or more CDC rules. You cannot change the severity of these rules, unlike DRCs or Messages.

Simplified schematics and descriptions of the CDC topologies can be found in the Simplified Schematics of the CDC Topologies section ordered from the highest to the lowest precedence.

Rule Precedence

CDC rules are analyzed using a defined precedence. Only one violation is reported per endpoint by default. If multiple CDC issues apply to the same endpoint, only the highest precedence violation is shown.

For example, if both CDC-15 (higher precedence) and CDC-10 (lower precedence) apply to the same register, only CDC-15 is reported.

You can override this behavior using the `-all_checks_per_endpoint` option, which reports all Info, Warning, and Critical checks per endpoint. However, if one safe rule applies to a register, the unsafe rules on that register are still not reported.

CDC Rules and Descriptions

Table 11: **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.

Rule Precedence: Highest to Lowest

Table 12: **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
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

This rule set ensures that unsafe or unknown CDC structures are flagged appropriately, and known safe synchronizations are recognized and prioritized.

Reviewing the Clock Domain Crossings Report Sections

When you run the CDC report in the Vivado IDE, it generates the following three sections by default:

- Summary by Clock Pair
- Summary by Type
- Detailed Report

Use the summary sections to review potential CDC issues and identify areas that might need RTL or constraint changes. These summaries provide a high-level overview and help you navigate to the most severe violations, which are detailed in the Detailed Report section.

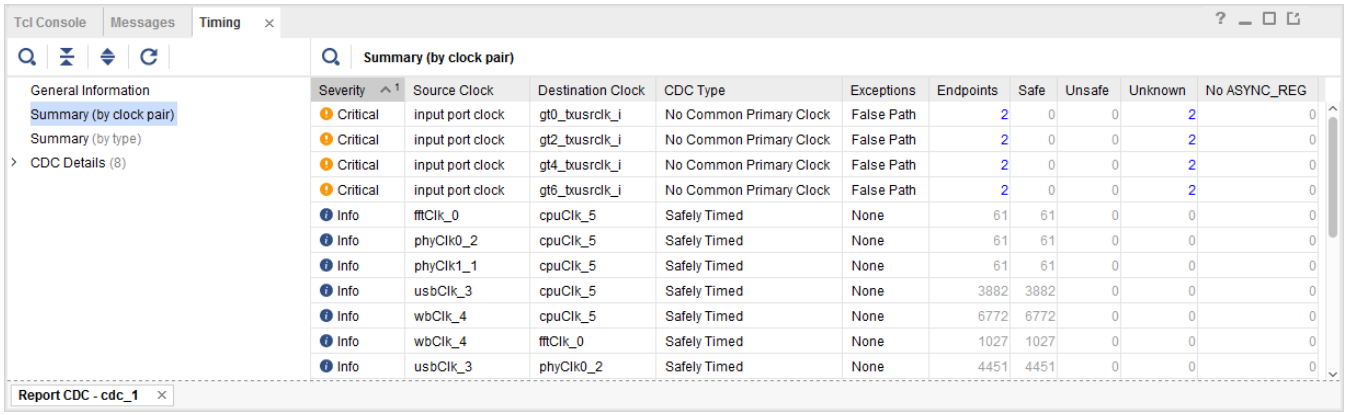
Note: When you run the report from the Tcl Console (text mode), only the Summary by Clock Pair section is reported by default.

Summary by Clock Pair

The Summary by Clock Pair section shows how many clock domain crossing (CDC) paths exist between two clocks and identifies the most critical CDC issue. This section helps you quickly assess the most severe CDC violations across clock pairs.

An example is shown in the following figure.

Figure 132: **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

The table includes the following columns:

- **Severity:** Shows the most critical severity level among all CDC paths between the listed clocks. Values include:
  - Info
  - Warning
  - Critical
- **Source Clock:** Displays the name of the source clock for the CDC path.
- **Destination Clock:** Displays the name of the destination clock for the CDC path.
- **CDC Type:** Describes the relationship between the two clocks and the dominant timing exception, if any. Types include:
  - **Safely Timed:** Clocks are synchronous and all CDC paths are accurately timed.
  - **User Ignored:** All CDC paths are covered by `set_false_path` or `set_clock_groups`.
  - **No Common Primary Clock:** Clocks are asynchronous, and at least one CDC path exists between clocks without a common primary clock.
  - **No Common Period:** Clocks are asynchronous and do not share a common period. For the definition of clocks with no common period, refer to Timing Analysis Key Concepts.
  - **No Common Phase:** Clocks are asynchronous with no known phase relationship.

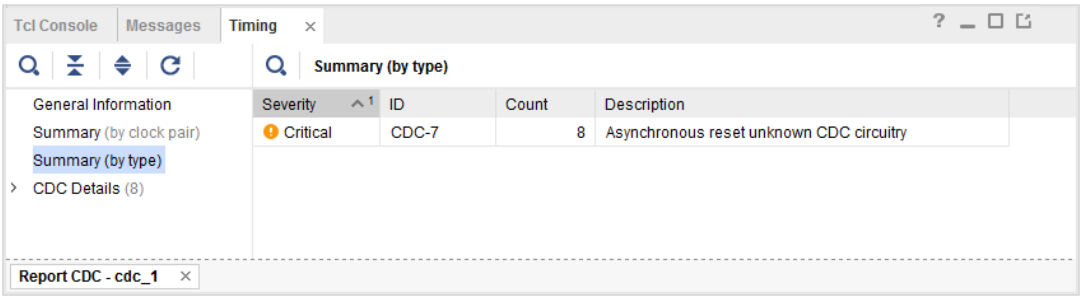
- **Exceptions:** The timing exceptions applied to the CDC (if any) are:
  - **None:** No `set_clock_groups`, `set_false_path`, or `set_max_delay -datapath_only` exceptions exist for the CDC.
  - **Asynch Clock Groups:** `set_clock_groups -asynchronous` is applied on the CDC.
  - **Exclusive Clock Groups:** `set_clock_groups -exclusive` is applied on the CDC.
  - **False Path:** `set_false_path` is applied between the clocks or on all CDC paths.
  - **Max Delay Datapath Only:** `set_max_delay -datapath_only` is applied to all paths.
  - **Partial Exceptions:** A combination of `set_false_path` and `set_max_delay -datapath_only` is applied, and at least one CDC path is still timed.
- **Endpoints:** Reports the total number of CDC path endpoints. Each endpoint is a sequential cell input pin. A flip-flop (FD) cell can count as multiple endpoints depending on connectivity to D, CE, or control pins (such as SET, RESET, CLEAR, or PRESET). Some topologies report only the first pin, even when there are several paths crossing the clock domain boundary to reach the CDC structure.
- **Safe:** Indicates the number of CDC path endpoints considered safe. Safe endpoints meet one of the following conditions:
  - Path uses asynchronous clocks and known safe CDC structures
  - Path uses synchronous clocks with safe structures and timing exceptions
  - Path uses synchronous clocks without exceptions and is safely timed
  - Path synchronized with `HARD_SYNC` macro
- **Unsafe:** Reports the number of CDC path endpoints identified as unsafe. Unsafe topologies include:
  - CDC-10: Combinatorial logic in fanin
  - CDC-11: Fanout before synchronizer
  - CDC-12: Multiple clocks in fanin
  - CDC-13: CDC on a non-FD primitive
- **Unknown:** Shows the number of endpoints that use unknown or unrecognized CDC structures. These include:
  - CDC-1: Single-bit CDC without known synchronization
  - CDC-4: Multi-bit CDC without known synchronization
  - CDC-7: Asynchronous reset without synchronization
- **No ASYNC_REG:** Indicates how many synchronizers are missing the `ASYNC_REG` property on one or both of the first two flip-flops in the synchronizer chain.

Summary by Type

The Summary by Type table helps you quickly assess the types of CDC structures identified in the report. Use this summary to prioritize your review based on the severity of the CDC rule violations.

An example is shown in the following figure.

Figure 133: **Summary by Type Table**



The table includes the following columns:

- **Severity:** Indicates the severity level of the CDC rule. Levels include Info, Warning, or Critical.
- **ID:** Displays the unique identifier for each CDC rule, as listed in the CDC Rules and Description table.
- **Count:** Shows how many times the CDC rule appears in the report.
- **Description:** Provides a brief explanation of the CDC rule type.

When reviewing the summary, start with the rules marked as Critical severity. The severity levels are defined as follows:

- **Critical:** Indicates a CDC path with unknown or unsafe CDC structures. Review each path in detail and either fix the issue in the RTL or waive it. You can see full path details in the Vivado IDE or by using `report_cdc -details` on the command line.
  - Combinatorial logic is present on the crossing net or multiple source clocks exist in the fanin, which can reduce mean time between failures (MTBF).
  - Fanout exists on the crossing net within the destination clock domain, which can cause data coherency issues.
- **Warning:** Indicates CDC paths with known synchronization structures that are considered safe but not ideal. Possible causes include:
  - One or more of the first two synchronizer flip-flops lack the `ASYNC_REG` property.

  - The CDC structure type requires functional correctness that the CDC engine cannot validate, such as:
    - Clock enable controlled CDCs
    - Multiplexer controlled CDCs
    - Multiplexer data-hold CDCs
- **Info:** Indicates that the CDC structure is both safe and correctly constrained.

Detailed Report

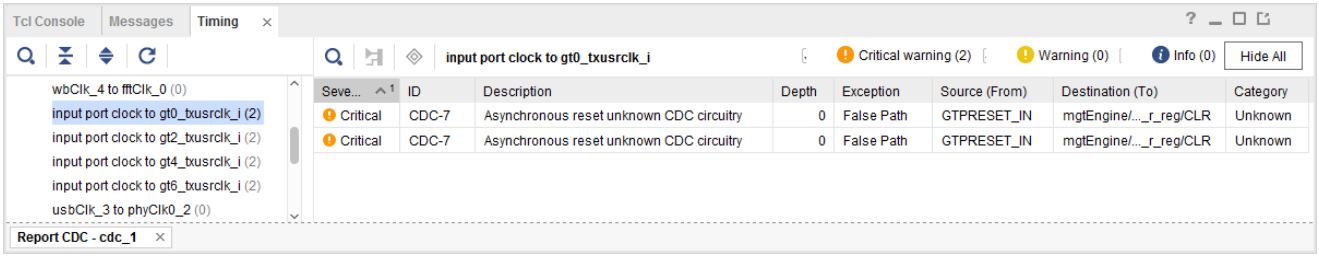
You can view the Report CDC details in the CDC Details section of the report. Use this section to:

- Open the schematic of a selected path by pressing **F4**.
- View an existing timing report.
- Generate a new timing report by right-clicking the entry.

Use the CDC Details to investigate unexpected CDC paths, find missing or incorrect timing exceptions, and locate synchronizers missing the `ASYNC_REG` property.

An example is shown in the following figure.

Figure 134: **CDC Detailed Report**



The CDC Detailed Report table includes the following columns:

- **Severity:** Reports the severity level of the CDC rule: `Info`, `Warning`, or `Critical`.
- **ID:** The unique identification number of the CDC rule, as listed in the CDC Rules and Description table.
- **Description:** A short explanation of the CDC rule.
- **Depth:** The number of synchronizer stages found. This column applies only to synchronizer topologies.
- **Exception:** The timing exception applied to the CDC path.
- **Source (From):** The startpoint of the CDC timing path.

- **Destination (To):** The endpoint of the CDC timing path.
- **Category:** Describes the path classification (for example safe, unsafe, unknown).
- **Source Clock (From):** The name of the source clock. This column appears only when you select **CDC Details** from the left pane of the Vivado Timing Report window.
- **Destination Clock (To):** The name of the destination clock. This column appears only when you select **CDC Details** from the left pane of the Vivado Timing Report window.

IMPORTANT! *The CDC report might flag issues in some AMD intellectual properties (IPs) because the CDC engine does not recognize all possible CDC topologies and does not offer a built-in waiver mechanism. For more information, refer to the corresponding AMD IP Product Guide.*

Simplified Schematics of the CDC Topologies

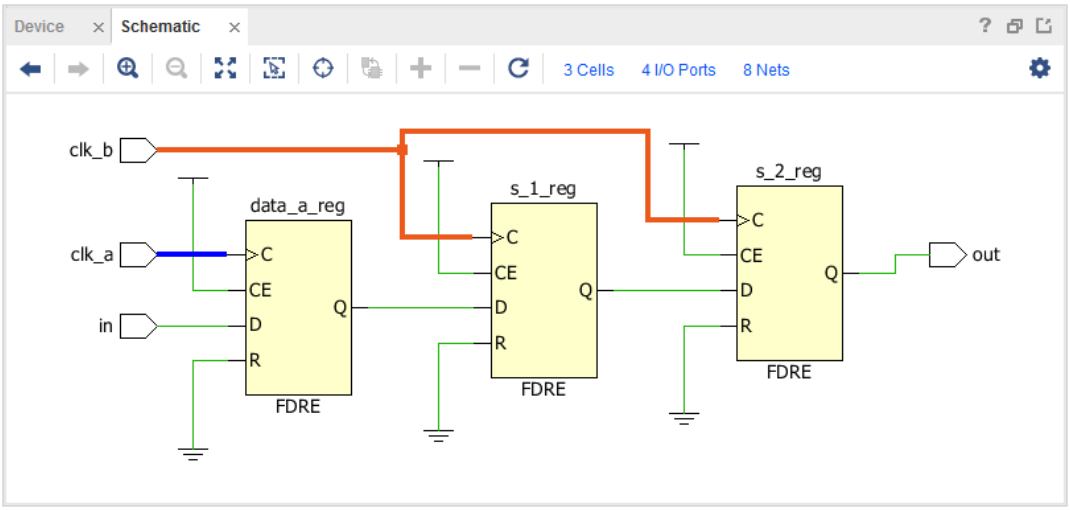
The following sections show simplified schematics and brief descriptions of the CDC topologies. In all diagrams:

- The source clock net (typically `clk_a`) appears in blue.
- The destination clock net (typically `clk_b`) appears in orange.

Single-Bit Synchronizer

The single-bit synchronizer consists of a chain of flip-flops used to safely transfer a signal from one clock domain to another.

Figure 135: **Simplified Topology of a Single-Bit Synchronizer**



Key characteristics:

- You must set the `ASYNC_REG` property on at least the first two flip-flops in the chain.

- Synchronizer depth is defined by the number of chained flip-flops that share the same control signals.
- If the `CLEAR` or `PRESET` pins of the flip-flops are connected to an asynchronous source, the CDC engine reports the circuit as a single-bit synchronizer, not as an asynchronous reset synchronizer.

Multi-Bit Synchronizer

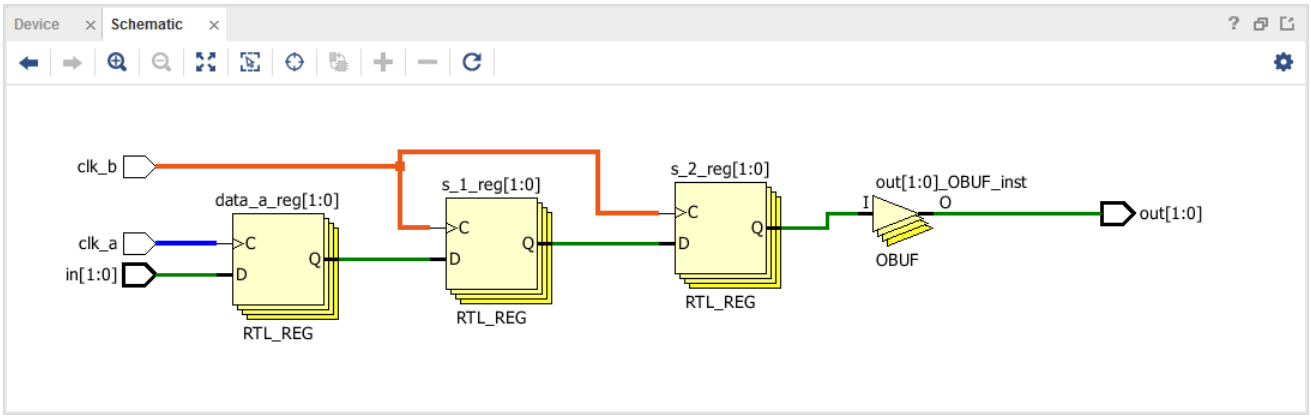
The multi-bit synchronizer groups multiple single-bit synchronizers based on startpoint and endpoint cell names, as well as matching CDC rules.

To be identified as a multi-bit bus:

- The startpoint and endpoint cell names must use the format `baseName[index]`.
- The corresponding index values must match between the startpoints and endpoints.

If some bits of the bus match different CDC rules, the report splits the bus into segments or single bits that share the same rule.

Figure 136: **Multi-Bit Synchronizer with 2-Bit Width**



Important Considerations

- A register-based synchronizer alone does not guarantee the safety of a multi-bit domain crossing.
- The CDC engine reports rule CDC-6 (multi-bit synchronizer) as a warning because it cannot confirm the safety of the bus structure and needs to be reviewed by the user.
- You must review and determine whether the synchronization topology is appropriate for your design.

Safe Usage

If the bus is gray coded, you can safely use a register-based synchronizer on all bits, provided you apply appropriate timing constraints. This ensures only one data transition occurs at a time in the destination clock domain.

Unsafe Usage

If the bus is not gray coded, consider using alternative synchronization techniques such as clock enable (CE) controlled CDC or multiplexer (MUX) controlled CDC.

Asynchronous Reset Synchronizer

The asynchronous reset synchronizer ensures that the reset signal is safely deasserted in the destination clock domain. The following figures show two types of synchronization:

- Clear-based synchronization
- Preset-based synchronization

In each case, the `FF1` cell connects to the synchronized clear or preset signal, and the deassertion is safely timed against `clk_a`.

Important considerations:

- Do not mix flip-flops with `CLEAR` and `PRESET` in the same asynchronous reset synchronizer.
- Avoid multiple synchronizations of the same reset signal inside the destination clock domain. Multiple synchronizations can cause parts of the destination logic to exit reset at different times, potentially placing the system in an unknown state. This issue is reported as a critical `CDC-11` violation (fanout from launch flop to destination clock).

Figure 137: **CLEAR-Based Asynchronous Reset Synchronizer**

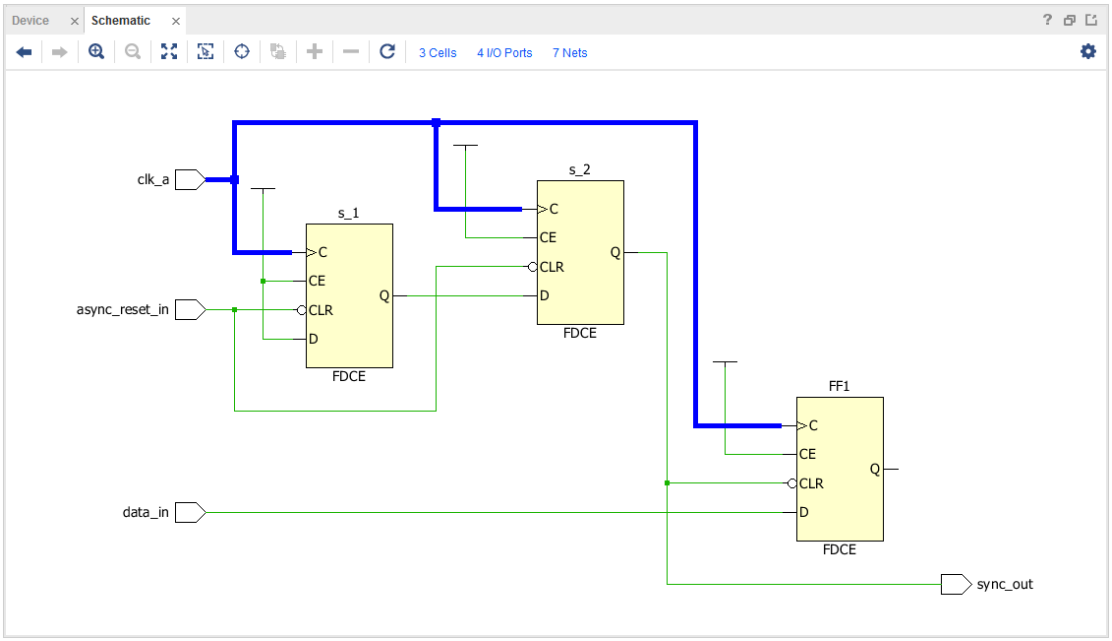
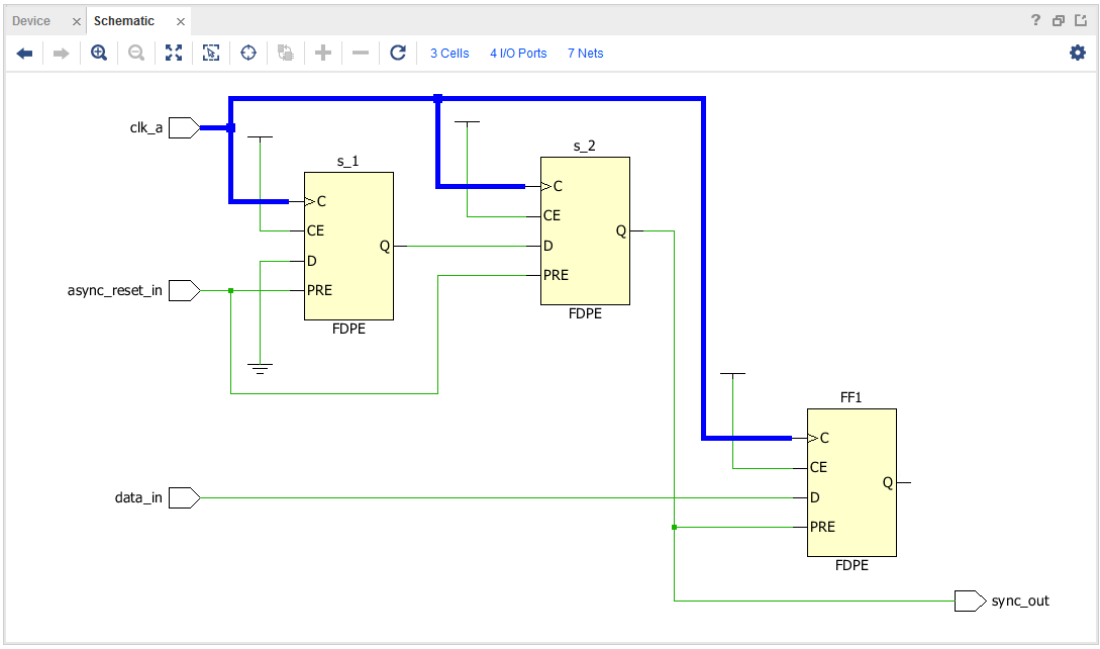


Figure 138: **PRESET-Based Asynchronous Reset Synchronizer**



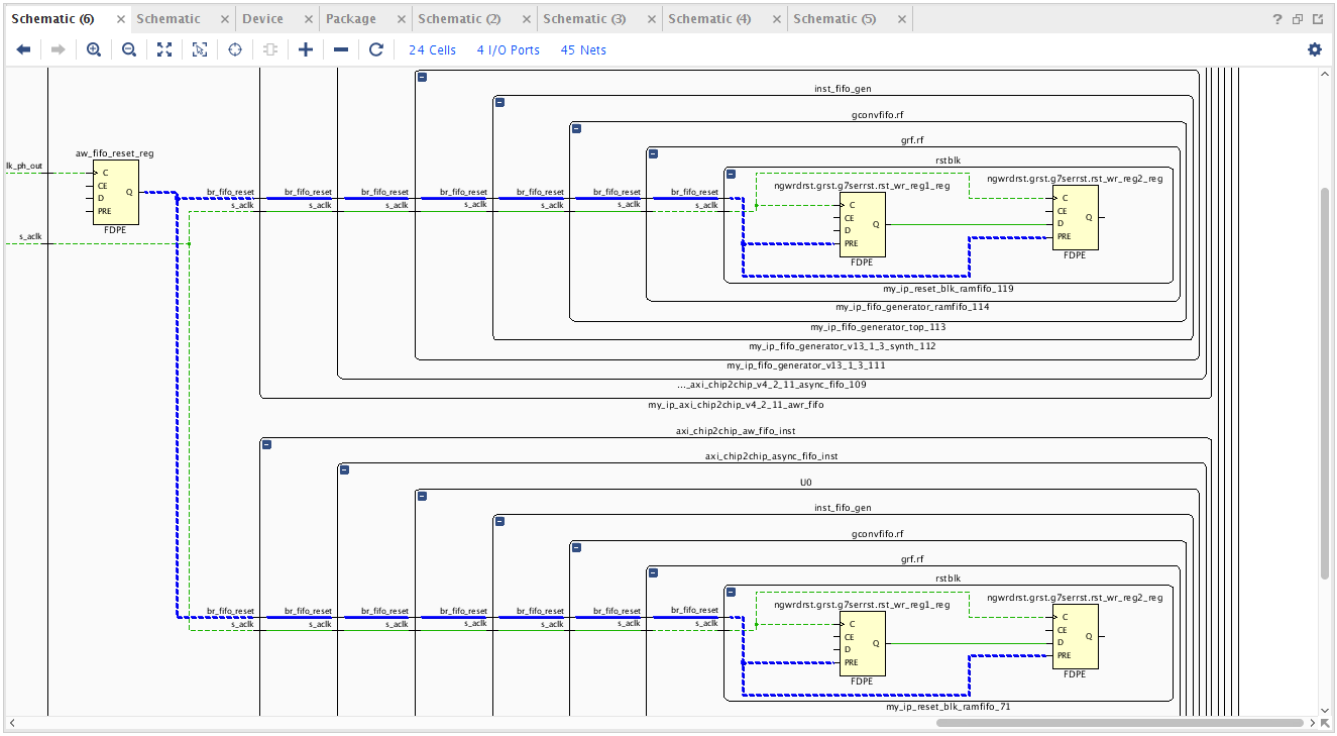
However, certain scenarios allow safe multiple synchronizations of the reset signal:

- The FIFO Generator IP safely handles reset synchronization by entering reset asynchronously and exiting reset synchronously. Although the block RAM receives a synchronous reset, the reset input to the FIFO Generator remains asynchronous. If your design uses the `wr_rst_busy` signal to control data flow, then reset release timing is safe across the logic.

- The AXI interface uses five FIFO Generator IPs to synchronize reset signals independently within each destination clock domain. This architecture safely manages multiple reset synchronizations by design.

In such cases using the FIFO Generator IP, you can ignore `CDC-11` violations reported for reset fanout.

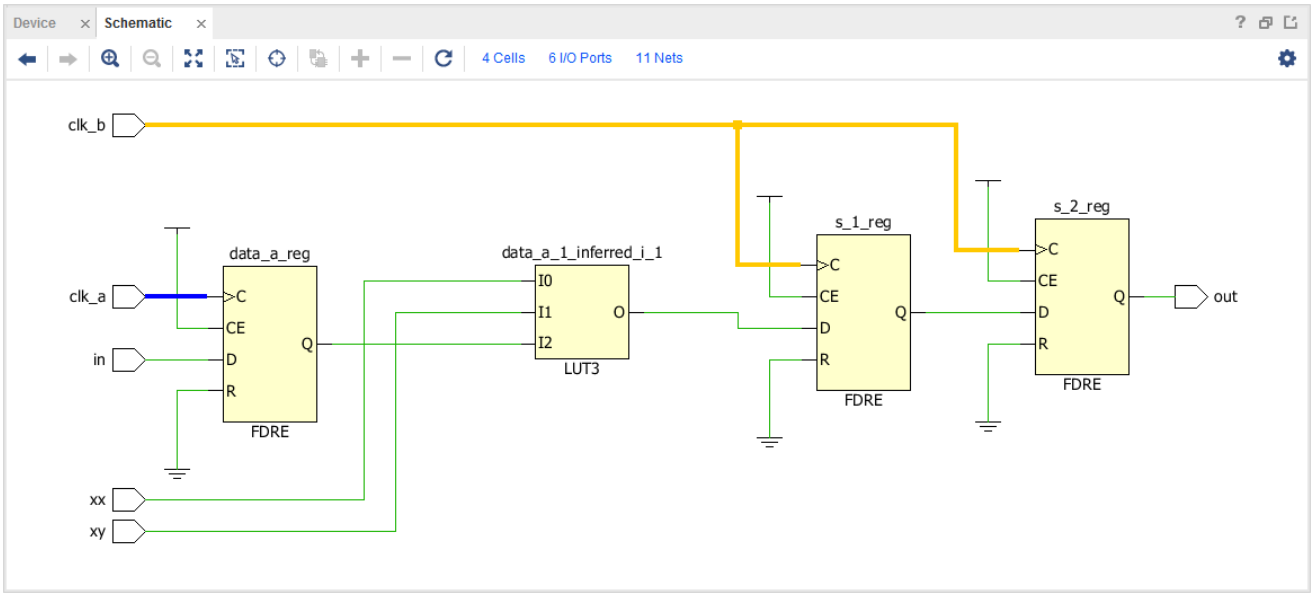
Figure 139: **Safe Reset Synchronization Example**



Combinatorial Logic

The following figure shows a simplified example of a CDC structure that includes combinatorial logic. In this topology, a logic function implemented by a `LUT3` sits between the synchronizers from the `clk_a` domain to the `clk_b` domain.

Figure 140: **Combinatorial Logic Simplified Example**



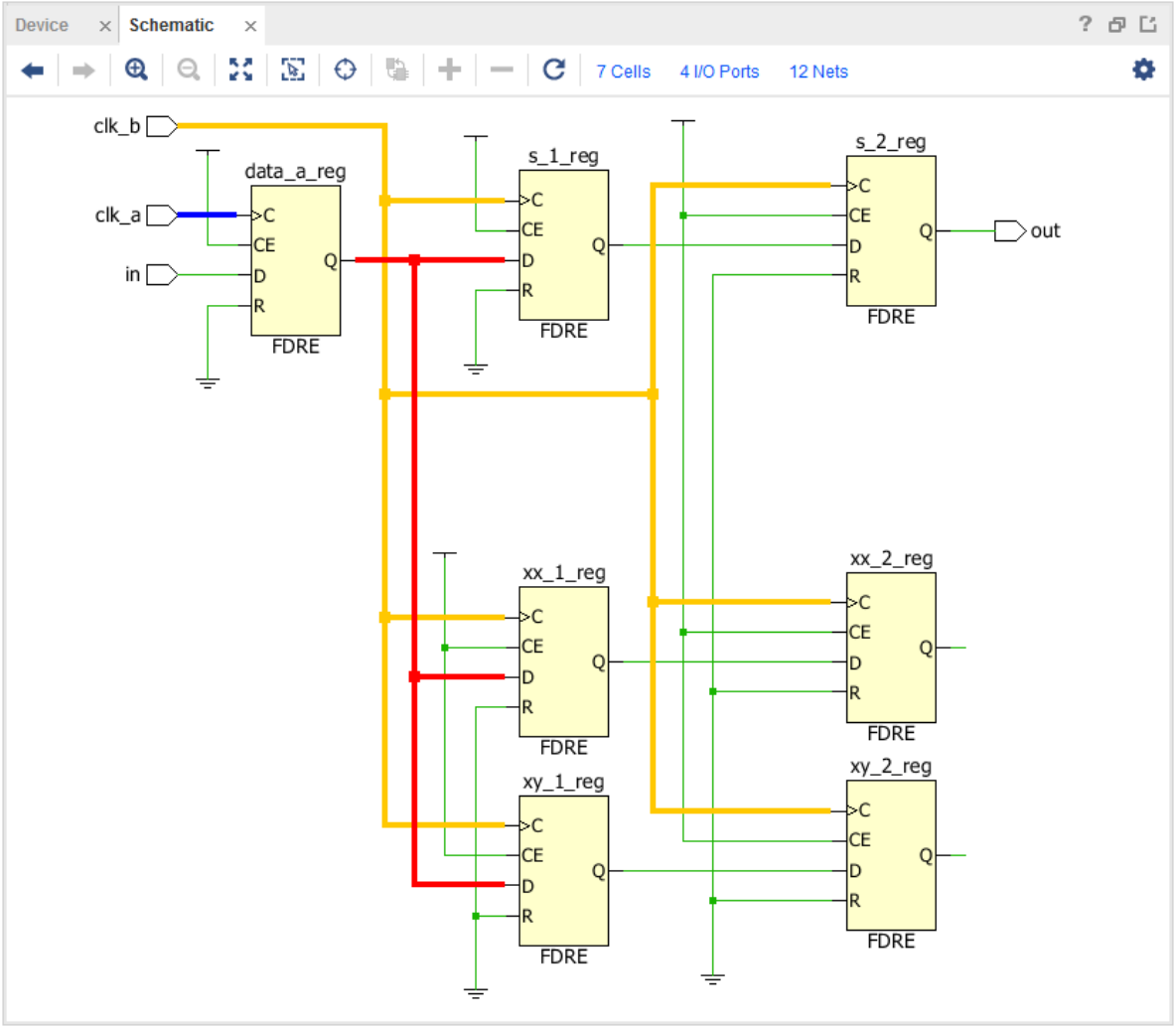
This structure is not recommended because glitches can occur at the output of the combinatorial logic. These glitches can be captured by the synchronizer and then propagate through the design, potentially causing unexpected behavior.

Fanout

The following figure illustrates a simplified fanout scenario. In this example, a single flip-flop in the

`clk_a` domain drives a net that is synchronized three separate times in the `clk_b` domain, highlighted in red.

Figure 141: **Simplified Fanout Example**



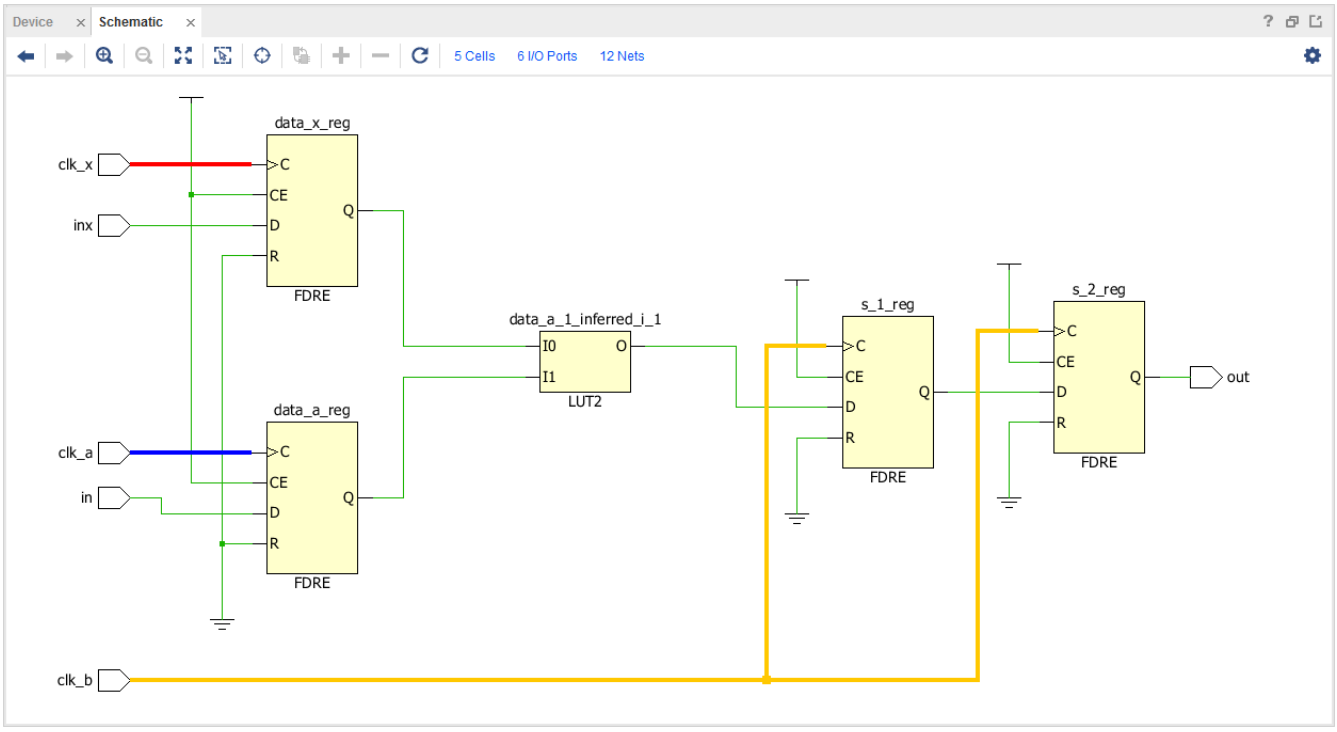
This structure is discouraged because the multiple synchronizers introduce latency that is bounded but not cycle-accurate. The misaligned timing can cause data coherency issues in the destination clock domain.

Note: A fanout of N signals to N different clock domains is not considered a CDC problem and does not trigger a `CDC-11` violation. See the Asynchronous Reset Synchronizer section for safe examples of reset signal fanout.

Multi-Clock Fanin

The following figure illustrates a multi-clock fanin example. In this structure, both `clk_a` and `clk_x` transfer data through combinatorial logic (`LUT2`) to a synchronizer circuit in the `clk_b` domain.

Figure 142: **Multi-Clock Fanin Example**

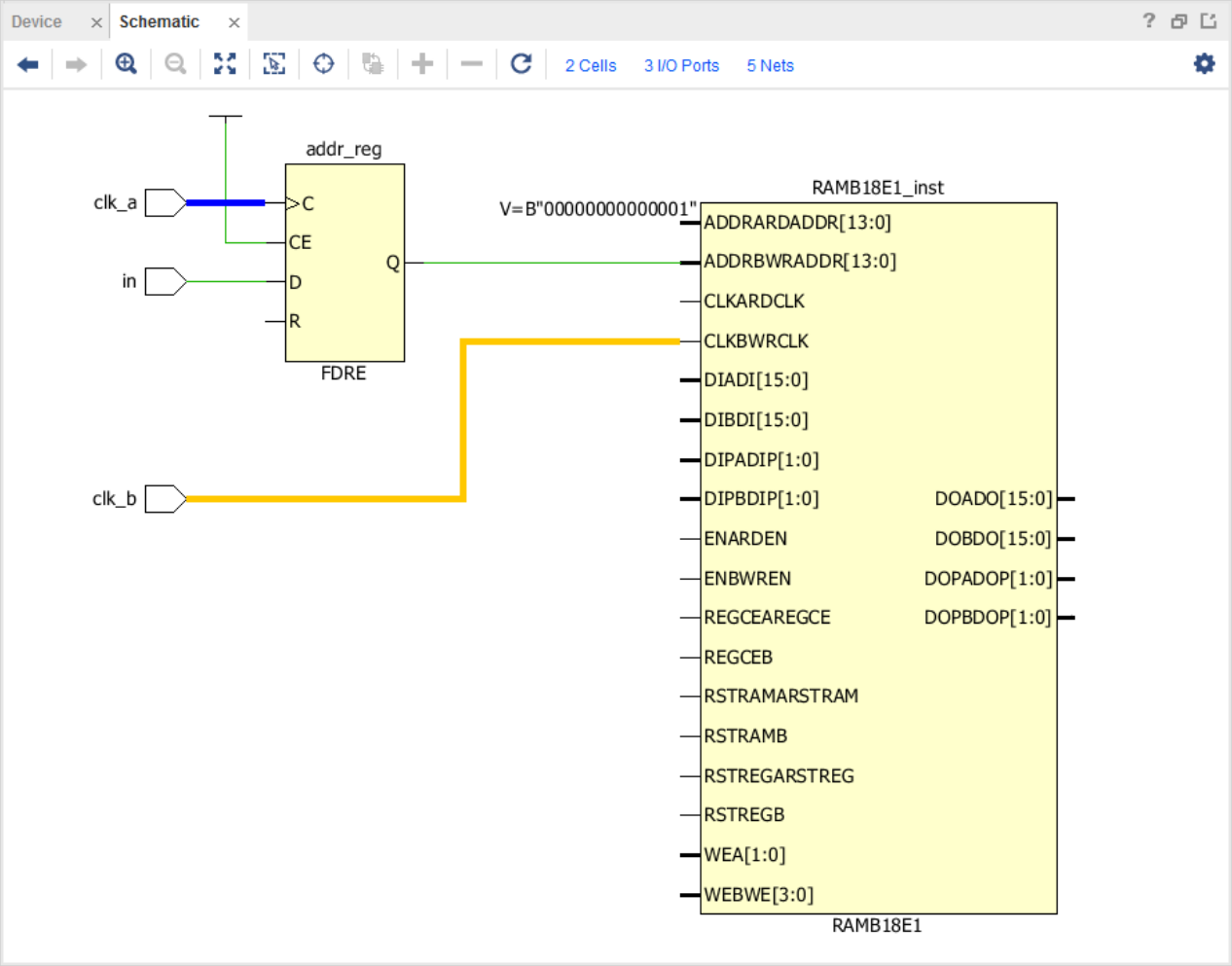


To improve the mean time between failures (MTBF), first synchronize the source data from `clk_a` and `clk_x` individually. Then combine the synchronized data using interconnect or FPGA logic. This approach also prevents glitches from propagating into the destination clock domain.

Non-FD Primitive

The following figure illustrates a non-FD primitive example. In this structure, a clock domain crossing (CDC) occurs between an `FDRE` and a `RAMB` primitive. No synchronization logic exists inside the `RAMB`.

Figure 143: **Non-FD Primitive Example**



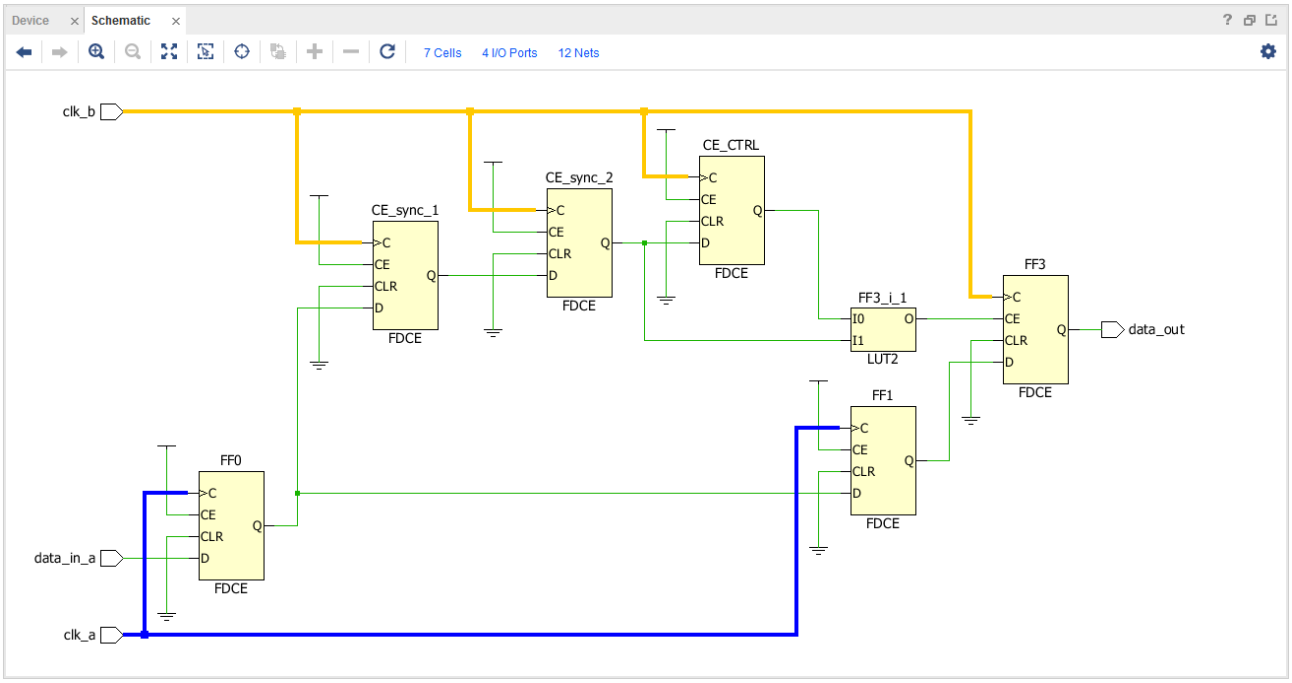
Even if you insert a single-stage flip-flop connected to `clk_b` in front of the `RAMB`, the tool considers the synchronizer inadequate. This is because the routing distance between the `FDRE` and `RAMB` cells remains too large to qualify as a safe synchronization structure.

***Note*:** This rule does not apply to the `HARD_SYNC` macro. The tool detects that structure separately and classifies it under CDC-18.

CE-Controlled CDC

The following figure shows a clock enable controlled CDC example. In this structure, the clock enable signal is synchronized in the `clk_b` domain before it controls the crossing flip-flops.

Figure 144: **CE-Controlled CDC Example**



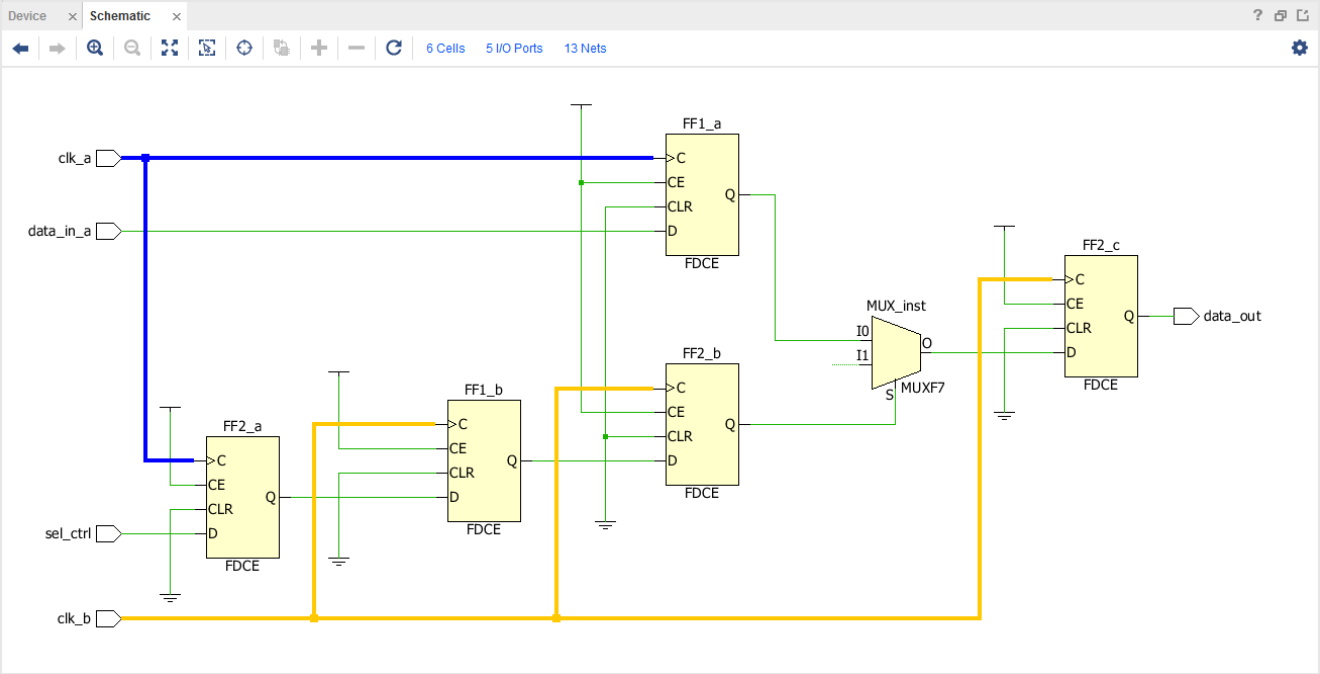
The CDC engine checks only that the signal connected to `FF3/CE` is launched by `clk_b`. It does not place restrictions on how the clock enable signal is synchronized or on the circuitry driving the `CE` pin, as long as that path is independently reported as a safe CDC.

You are responsible for constraining the latency from the `clk_a` domain to `FF3`. To do this, apply a `set_max_delay -datapath_only` constraint.

Mux-Controlled CDC

The following figure shows a multiplexer controlled CDC example. In this structure, the multiplexer select signal is synchronized to the destination clock domain, `clk_b`.

Figure 145: **Mux-Controlled CDC Example**

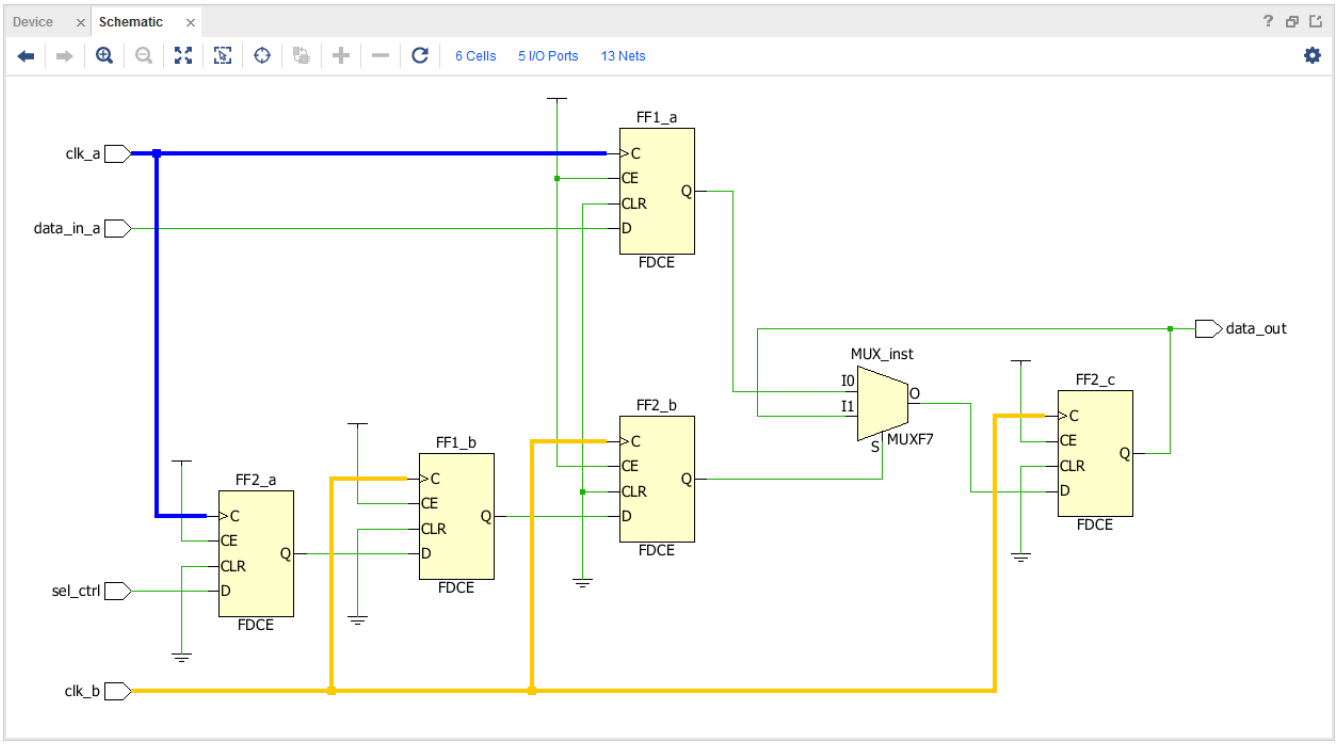


As with clock enable controlled CDC, the CDC engine does not restrict how the select signal is synchronized. The select signal must be reported as a safe CDC individually. You are responsible for constraining the crossing delay on `FF2_c`, typically with a `set_max_delay -datapath_only` constraint.

Mux Data Hold CDC

The following figure shows a multiplexer data-hold CDC example. In this structure, the multiplexer select signal is synchronized to the destination clock domain, `clk_b`, and `data_out` feeds back to the multiplexer input.

Figure 146: **Mux Data Hold CDC Example**

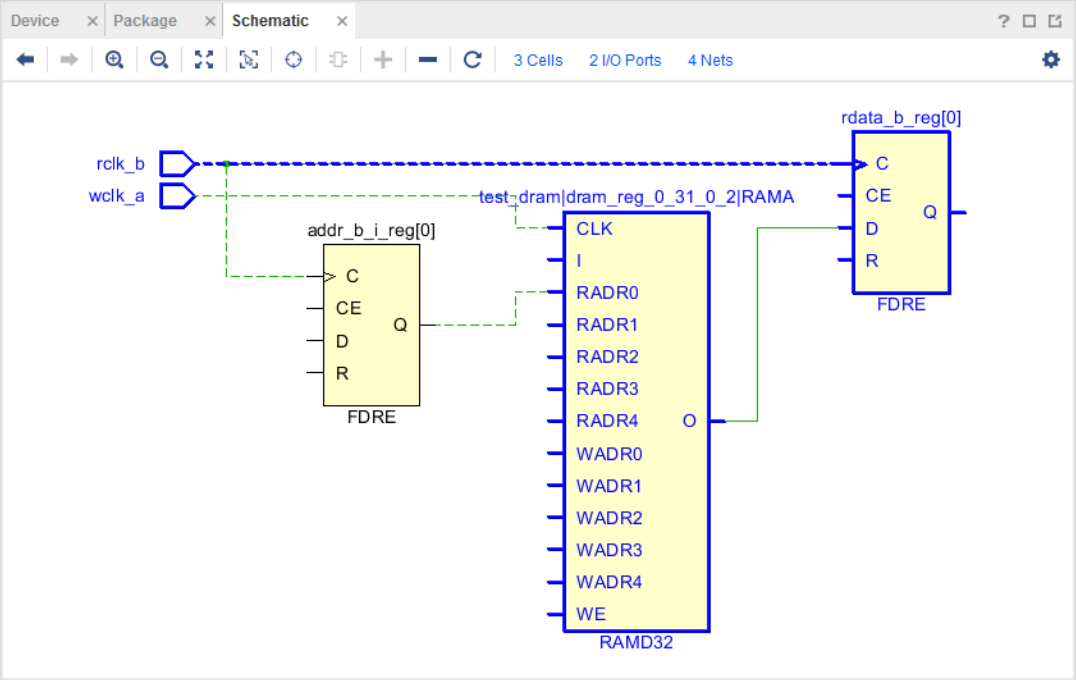


As with clock enable controlled CDC, there is no restriction on how the multiplexer select signal is synchronized. The signal must be reported as a safe CDC path individually. You are responsible for constraining the crossing delay on `FF2_c`, typically using a `set_max_delay -datapath_only` constraint.

LUTRAM Read/Write Potential Collision

The following figure shows a LUTRAM read/write potential collision. In this example, data is written to the LUTRAM using the write clock, and the output is captured using the read clock.

Figure 147: **LUTRAM Read/Write Potential Collision**



When the read and write addresses differ, no CDC path exists between the write and read clocks. However, when the addresses match, a CDC path forms between the write clock and the read clock.

To prevent this CDC condition, ensure that the logic surrounding the LUTRAM never generates the same read and write addresses during active read and write operations. When this condition is guaranteed, you can waive the associated CDC violation.

AMD's FIFO generator intellectual property (IP), for example, includes built-in logic to prevent read/write collisions by design.

Timing Closure Reports

Report QoR Assessment

The `report_qor_assessment` command generates a report which provides the following:

- **Assessment score:** Indicates how likely your design is to meet performance targets.
- **Flow guidance:** Recommends next steps based on the current state of your design.

- **Utilization and performance summary:** Summarizes key design metrics related to device usage and timing.
- **Methodology checks summary:** Reports critical methodology checks that affect quality of results (QoR). This section appears only in the text version of the report.
- **ML strategy availability:** Provides information on available machine learning (ML) strategies that can improve implementation results (UltraScale and UltraScale+ architectures only).

Overall Assessment Summary

The QoR Assessment Summary includes two parts:

- A score that predicts whether your design meets timing goals.
- Flow guidance based on the current implementation status.

The assessment score estimates your design's likelihood of meeting performance targets at the current point in the flow. Run the command early in the flow to maximize benefits, as earlier runs save more compile time. Although early results are slightly less accurate, the score typically differs by no more than one point from the final post-route score.

Vivado calculates the score by analyzing design metrics such as:

- AMD UltraFast™ methodology compliance
- Device utilization
- Control sets
- Clocking structure
- Setup and hold slack
- Device-specific factors depending on the implementation stage

For example:

- After `synth_design`, the tool evaluates clocking netlist structures.
- After `place_design`, clock skew contributes more heavily.
- After `route_design`, routing completion becomes a key factor.

Scores range from 1 to 5. If the score is less than 5, use the `report_qor_suggestions` command to improve it.

Table 13: **Report QoR Assessment Scoring**

Score	Meaning
1	Design will likely not complete implementation.
2	Design will complete implementation but will not meet timing.

Table 13: **Report QoR Assessment Scoring** *(cont'd)*

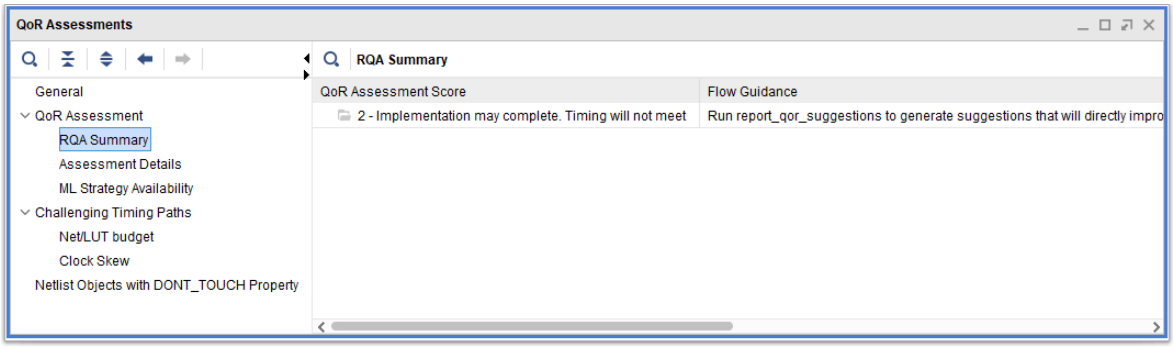
Score	Meaning
3	Design will likely not meet timing.
4	Design will likely meet timing.
5	Design will meet timing.

Flow guidance updates dynamically based on design status. It can include:

- Whether to address methodology issues
- Whether to apply suggestions from `report_qor_suggestions`
- Whether to use machine learning (ML) strategies or incremental compile

The following figure shows an example of a design with a QoR assessment score of 2.

Figure 148: **Overall Assessment Summary**

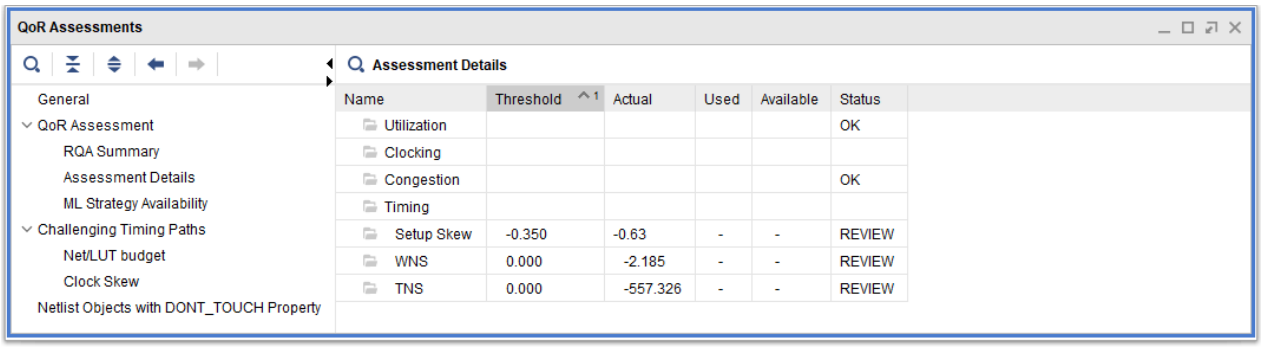


QoR Assessment Details

The QoR Assessment Details table summarizes design issues in five categories that form the basis of the `report_qor_assessment` score:

- Utilization
- Netlist
- Clocking
- Congestion
- Timing

Figure 149: **QoR Assessment Details**



Each category receives an OK status when no sub-items are marked REVIEW. If any sub-item requires attention, the failing item appears with its threshold and current value. These thresholds are not hard limits, but exceeding them can make timing closure more difficult. Focus on items that exceed thresholds significantly or appear in multiple categories.

Items marked with an asterisk (*) do not directly affect the QoR score but can still impact timing closure and require review.

Utilization

Vivado checks utilization across the entire device, including SLR and Pblock levels. Run `report_qor_suggestions` to reduce utilization if needed.

Netlist

Netlist checks evaluate the logical structure and non-timing constraints. Vivado flags issues such as:

- Use of the `DONT_TOUCH` property
- High fanout nets with poor driver profiles
- Other netlist features that can challenge implementation

Clocking

Vivado checks for high clock skew on setup and hold paths.

- If a skew issue is detected, Vivado automatically includes path details in the report.
- In text mode, use the `-csv_output_dir <directory>` option to generate timing paths in CSV format.
- Run `report_qor_suggestions` to receive automated fixes for skew problems.

Congestion

Vivado evaluates netlist structures that can lead to routing congestion. Congested region data becomes available after placement.

You can:

1. Run `place_design` and `route_design` to assess congestion.
2. Use `report_qor_suggestions` to generate cell-specific congestion fixes.

Timing

Vivado reviews the 100 worst-case paths per clock group. It checks:

- Worst negative slack (WNS)
- Total negative slack (TNS)
- Worst hold slack (WHS)
- Total hold slack (THS)

To help identify potential failures earlier in the flow, Vivado also performs:

- Net budget checks: Adds conservative delay values for nets instead of estimated values.
- LUT budget checks: Replaces estimated LUT delays with more conservative values.

Review and resolve paths with negative slack early to reduce late-stage timing issues. Use the Challenging Timing Paths section in the Vivado IDE or export the data to a CSV file for further review.

If the design is routed and is either the UltraScale or UltraScale+ architectures, Vivado also checks whether the last mile flow used by Intelligent Design Runs can close timing. This analysis includes:

- Worst-case paths
- Slack before and after optimization
- Involved logic primitives

These details help determine whether the design can meet final timing goals.

Methodology Checks

The `report_qor_assessment` command runs a limited set of methodology checks to verify that your design has a solid foundation for effective quality of results (QoR) suggestions. These checks are a subset of those performed by the `report_methodology` command.

Vivado follows this behavior:

1. If methodology check results already exist and the design has not changed, Vivado reuses the cached results.
2. If the design has changed or no cached results are available, Vivado reruns the necessary checks. This can increase runtime.

To skip methodology checks, use the `-exclude_methodology_checks` switch.

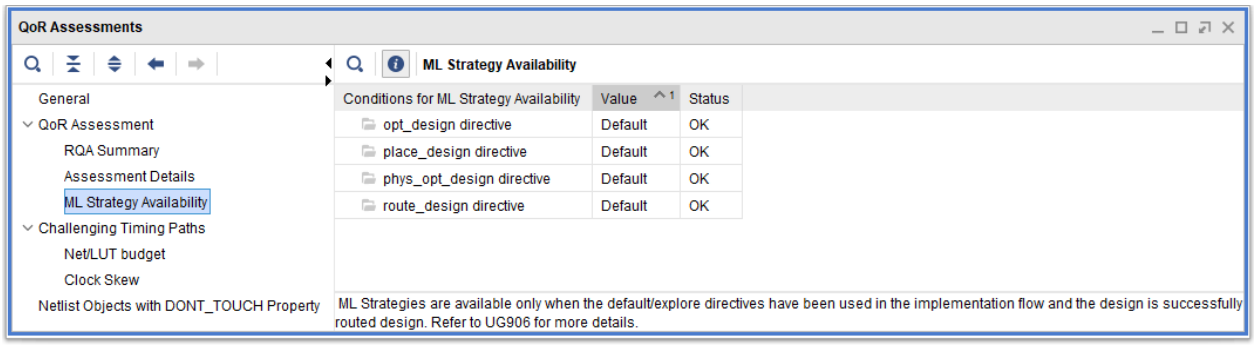
***Note*:** The Vivado IDE does not display this section. Instead, generate and review the Report Methodology separately. Most implementation reporting strategies include this report by default.

ML Strategy Availability

The ML Strategy Availability table, shown in the following figure, indicates whether your design meets the requirements for generating machine learning (ML) strategies using the `report_qor_suggestions` command.

ML strategies are not generated unless all required implementation steps in the reference run are completed. The tool evaluates the following conditions:

Figure 150: **ML Strategy Availability**



The flow requirements are as follows:

- The `opt_design` command must run with the directive set to either `Explore` or `Default`. You can run `opt_design` more than once, but the final call must meet this condition.
- All remaining implementation directives must be consistently set to either `Default` or `Explore`. Do not mix and match these settings across implementation steps.
- The `phys_opt_design` command must be enabled.
- The design must be fully routed.
- The target device must be from the UltraScale or UltraScale+ family.

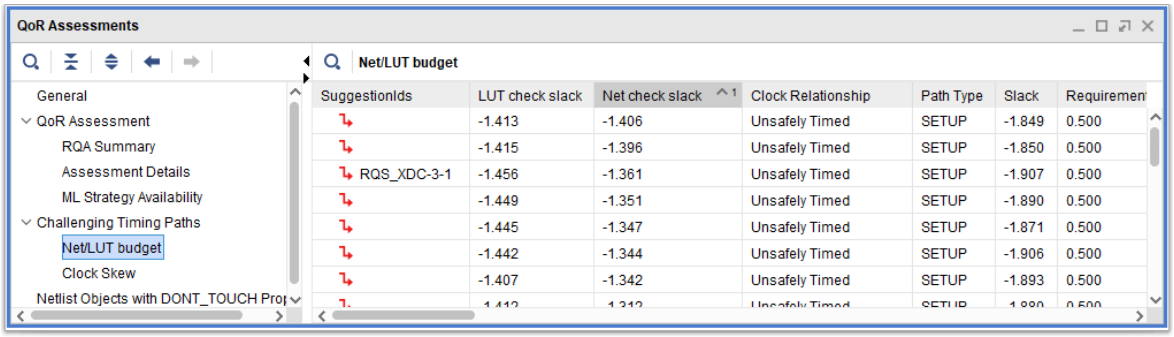
Challenging Timing Paths

The Challenging Timing Paths section highlights key properties of timing paths that failed the checks in the Assessment Details section. By default, the command evaluates 100 failing paths for each clock group. It analyzes the following factors:

- Net budget
- LUT budget
- Clock skew

The following figure shows an example of this analysis.

Figure 151: **Net/LUT Budget Report**



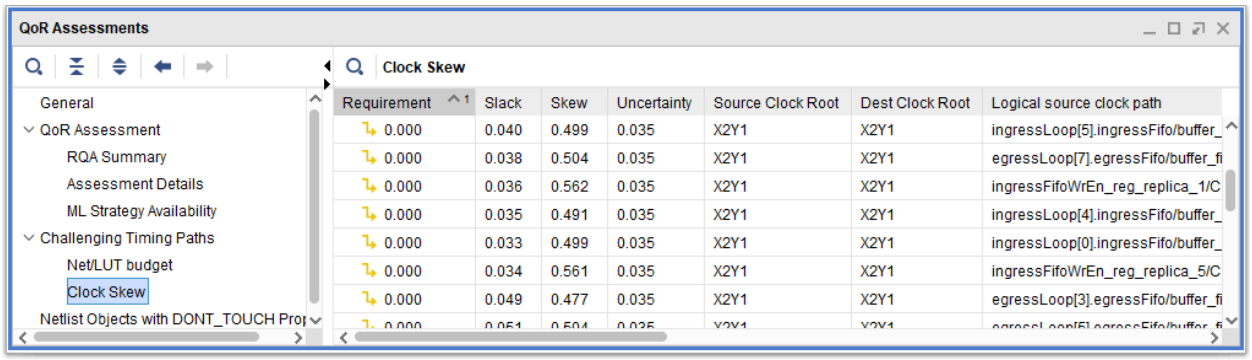
For these checks, the tool replaces estimated net or LUT delays with typical device-specific values, then recalculates the path budget. It also applies penalties to paths that start or end at block RAMs, DSPs, or other hard blocks. Additional penalties apply when the design cannot leverage clock tree skew to improve timing margin. The recalculated slack appears in the LUT Check Slack and Net Check Slack columns.

The SuggestionsID column shows any path-related QoR suggestions. If no suggestions are listed, investigate the paths and evaluate potential RTL modifications. If suggestions are present, applying them can resolve the issue without requiring code changes.

The Clock Skew section, shown in the figure labeled “Clock Skew Report,” displays the following information:

- Skew value
- Source and destination clock names
- Clock root for both clocks
- Clock uncertainty

Figure 152: **Clock Skew Report**



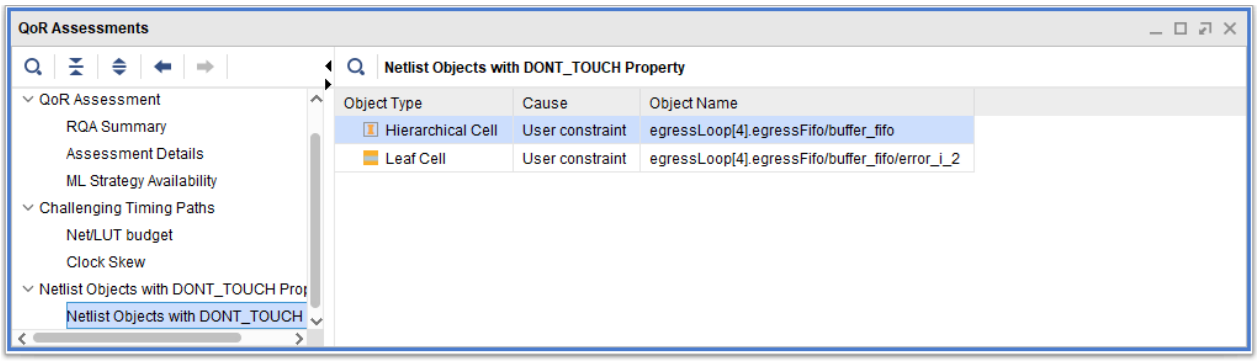
Netlist Objects with DONT_TOUCH Properties

Netlist objects with `DONT_TOUCH` properties can block optimizations that would otherwise improve design performance. This section of the QoR Assessment report identifies the objects that have the `DONT_TOUCH` property set.

The report includes:

- Hierarchical cells
- Leaf cells
- Nets

Figure 153: **Netlist Objects with DONT_TOUCH Property**



The cause for each setting is also shown.

Note: The `DONT_TOUCH` property prevents the tools from optimizing the associated paths. Vivado might automatically apply this property through other design attributes. Remove `DONT_TOUCH` properties only when appropriate. For example:

- The DFX flow uses `DONT_TOUCH` to prevent cross-boundary optimization between static and reconfigurable modules. In this case, do not remove the property.

- If DONT_TOUCH was added through MARK_DEBUG, it is not critical to the design flow. However, removing it could make the signal unavailable for hardware probing if the tool optimizes it away.

Non-FD High Fanout Nets > 10K

This section reports nets that meet both of the following conditions:

- The net has a fanout greater than 10,000.
- The net is not driven by a flip-flop.

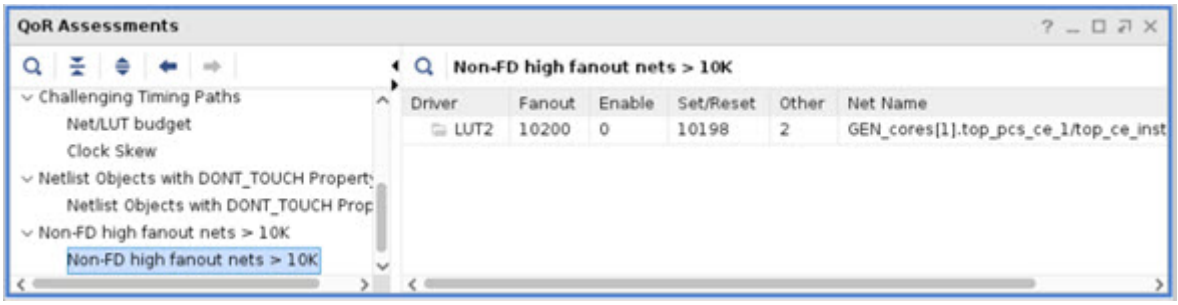
High fanout nets require a driver profile that supports replication without affecting pre-path timing. Replication enables the tool to place duplicate drivers near the loads, improving overall routing efficiency. However, if source constraints exist, the tool might cluster the replicated drivers together, which limits placement flexibility and reduces the effectiveness of replication.

The details section of the report includes the following:

- Load types
- Driver profile information
- Net name

Timing information is not included. If the net drives a false path or a path with a very low timing requirement, you can ignore the entry.

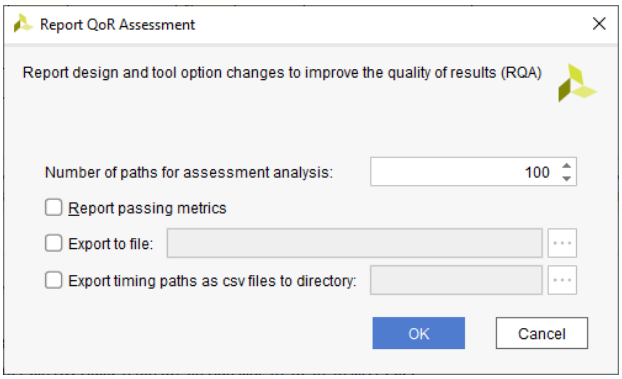
Figure 154: **Non-FD High Fanout Nets Example**



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 155: **Report QoR Assessment Dialog Box**



The equivalent command at the Tcl Console is as follows:

```
report_qor_assessment -name rqa_1
```

You can adjust the number of timing paths analyzed by modifying the *Number of paths for suggestion analysis* setting in the dialog box. The equivalent Tcl command uses the `-max_paths <N>` where `<N>` is the number of paths:

```
report_qor_assessment -max_paths <N>
```

To show all the metrics that passed during assessment, enable the Report passing metrics option in the dialog box. The equivalent Tcl command option is:

```
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 option uses the `-csv_output_dir` switch:

```
report_qor_assessment -csv_output_dir <directory>
```

Other Command Options

The `-exclude_methodology_checks` option omits methodology checks from the report. By default, the tool runs these checks or retrieves them from the internal cache if the report has been generated previously. Using this option can reduce compile time when running the command for the first time. Because methodology checks remain mostly static throughout the implementation flow, this option is recommended when calling `report_qor_assessment` at multiple stages of the run.

Auto Termination of Runs

When you run the QoR Assessment feature during a project-based implementation run, the tool generates a score between 1 and 5 to indicate how likely the design is to close timing. A higher score suggests a greater likelihood of success. You can enable early termination of low-quality runs based on this score to free up server resources sooner. Early termination happens under two conditions:

- When the `MIN_RQA_SCORE` run property is set. For example, if it is set to 3, then any run with an RQA score of 1 or 2 terminates early.
- When the `report_qor_assessment` command is included in the Timing Closure Report Strategy after `opt_design`. For additional calls, use a custom report strategy.

To set the `MIN_RQA_SCORE` property to 1 on a run named `impl_1`, use the following command:

```
set_property MIN_RQA_SCORE 1 [get_runs impl_1]
```

Note: This feature does not directly apply to non-project mode. To generate the RQA score in script-based flows, call `set rqa_score [get_qor_assessment]`.

Report QoR Suggestions

Use the `report_qor_suggestions` command to work with quality of results (QoR) suggestion objects. These objects create commands and properties that help improve the timing performance of your design. For more information, see QoR Suggestions

The `report_qor_suggestions` command performs two tasks:

- Reports existing QoR suggestion objects
- Generates new QoR suggestion objects

You can run this command at any stage after synthesis.

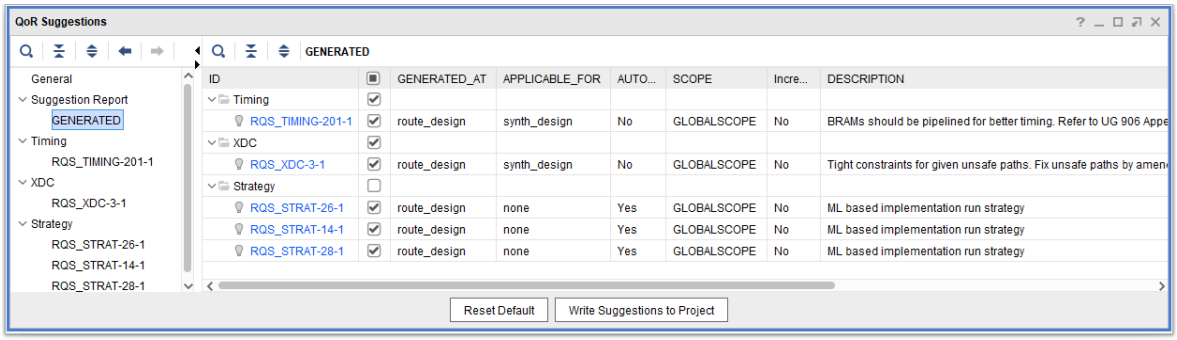
To create an `.rqs` file from the report, use the `write_qor_suggestions` command. This section includes details on how to generate that file.

QoR Suggestion Report

The QoR Suggestions report includes two main sections:

- Summary Section: Displays a list of all suggestions, grouped by category.
- Details Section: Provides additional information about each suggestion, organized by analysis type.

Figure 156: **Example report_qor_suggestions Report**



The top section, labeled Suggestion Report, categorizes suggestions into four groups, arranged in two pairs:

- GENERATED and EXISTING
  - **GENERATED:** Suggestions created during the current stage of the flow.
  - **EXISTING:** Suggestions from earlier stages or imported from an `.rqs` file.
- APPLIED and FAILED TO APPLY
  - **APPLIED:** Suggestions that are enabled, valid for the current stage, and successfully applied.
  - **FAILED TO APPLY:** Suggestions that are enabled and valid for the stage but could not be applied. Check the log file to understand the failure reason.

The Details Section at the bottom of the report provides category-specific analysis. Vivado uses `report_qor_suggestions` to evaluate your design across the following areas:

- Clocking
- Congestion
- Utilization
- Timing
- Netlist
- XDC
- Strategy

Working with Suggestions

When reviewing GENERATED suggestions, the details section explains why each suggestion appears. You can use the following cross-probing methods to explore the design:

1. Select objects to highlight them in other windows such as Device view
2. Press **F4** to open a schematic of the selected objects
3. Right-click objects to generate a timing report

For EXISTING suggestions, cross-probing might not be available if the objects were removed by commands such as `opt_design`.

Additional Information Columns

Each suggestion includes extra attributes to help you understand its context and application:

Table 14: **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

You can generate a QoR Suggestions report in the Vivado IDE or through the Tcl Console.

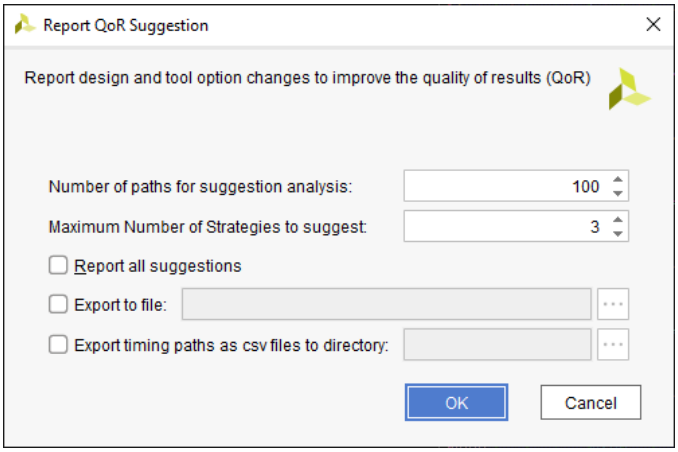
1. Generate the report:
   - In the Vivado IDE: Select **Reports → Report QoR Suggestions**.
   - In Tcl: `report_qor_suggestions -name qor_suggestions_1`
2. Adjust the report parameters:
   - **Number of timing paths to analyze:** This expands the number of suggestions but might include paths that are not critical.
     - In the GUI: Change Number of paths for suggestion analysis. The default is 100.
     - In Tcl: `-max_paths <N>`

3. • **Number of ML strategies to suggest:**
   - In the GUI: Change Maximum Number of Strategies to suggest.
   - In Tcl: `-max_strategies <N>`
   - **Report all suggestions:** This option includes suggestions that do not violate threshold criteria:
     - Timing suggestions include paths even if timing is met.
     - Utilization suggestions include non-critical resource suggestions.
     - Congestion suggestions include post-route suggestions even if timing is met.
     - In the GUI: Select the Report all suggestions checkbox.
     - In Tcl: `-report_all_suggestions`
   - **Generate CSV files for failing paths:** This creates:
     - A CSV file listing failing timing paths and their associated suggestions
     - A second file containing a DONT_TOUCH report
     - In the GUI: Check the box to generate CSV files and specify a directory.
     - In Tcl: `-csv_output_dir <directory>`

     Note: Use caution when removing DONT_TOUCH properties. Vivado can apply these automatically:

     - Do not remove DONT_TOUCH from static or reconfigurable modules in a DFX flow. These prevent cross-boundary optimizations.
     - Optional to remove DONT_TOUCH from signals marked with `MARK_DEBUG`. These do not affect the flow but prevent the signal from being optimized or available for hardware probing.

Figure 157: **Report QoR Suggestions Dialog Box**



Other Command Options

You can refine the behavior of the `report_qor_suggestions` command using the following options:

- **`-of_objects <suggestion objects>`:** Use this option to report specific QoR suggestions without generating new ones.
  - This mode runs quickly and is helpful for reviewing suggestions from an `.rqs` file that has already been loaded.
  - Example:

```
report_qor_suggestions -of_objects [get_qor_suggestions <objectNames>]
```

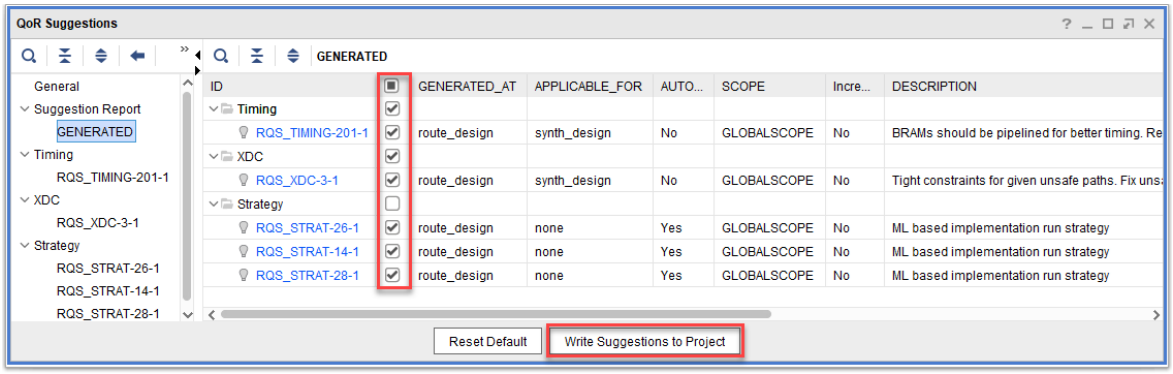
- **`-cells <cellName>`:** Use this option to change the top-level cell for the analysis.
  - By default, the command analyzes the design's top-level cell.
  - To analyze a different module, specify it with this option.

Writing the Suggestion Object File

After generating the QoR Suggestions report, you need to write an `.rqs` file that contains the selected suggestions for use in a future suggestion run.

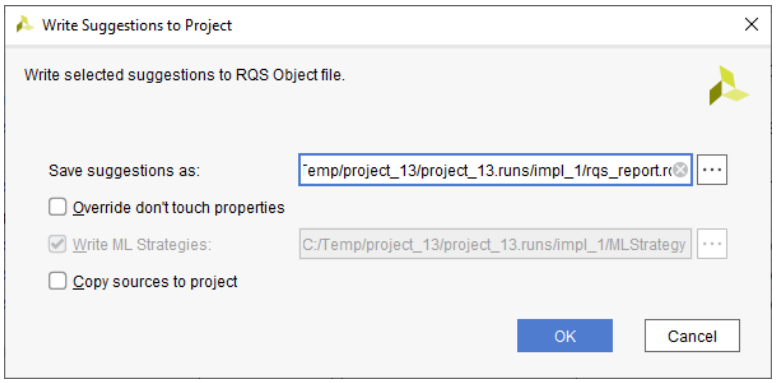
In the Vivado IDE, choose the suggestions you want to include in the `.rqs` file.

Figure 158: **Select/Write Suggestions**



Use the Write Suggestions to Project dialog box to specify the file name and any required options.

Figure 159: **Write Suggestions to Project Dialog Box**



The equivalent Tcl option is as follows:

```
write_qor_suggestions filename.rqs
```

Some suggestions require you to override DONT_TOUCH properties. If needed, in Tcl, add this option:

```
write_qor_suggestions -disable_dont_touch
```

Machine learning (ML) strategy suggestions are handled separately from standard suggestions. You can generate multiple strategy `.rqs` files from a single run. The equivalent Tcl option is as follows:

```
write_qor_suggestions -strategy_dir <directory>
```

For more details, refer to the Strategy Suggestions section.

Report Design Analysis

The Design Analysis report helps you evaluate timing path characteristics, interconnect complexity, and routing congestion. Use this information to adjust your design or constraints to improve QoR and reduce routing congestion.

Running Report Design Analysis

You can run Report Design Analysis from the Tcl Console or the Vivado IDE. The report generates three categories:

- **Timing:** Shows timing and physical characteristics of timing paths.
- **Complexity:** Analyzes routing complexity and LUT distribution.
- **Congestion:** Identifies routing congestion in the design.

Follow these steps to run the report:

1. In the Vivado IDE, go to **Reports → Report Design Analysis**.

   The equivalent Tcl command option is:

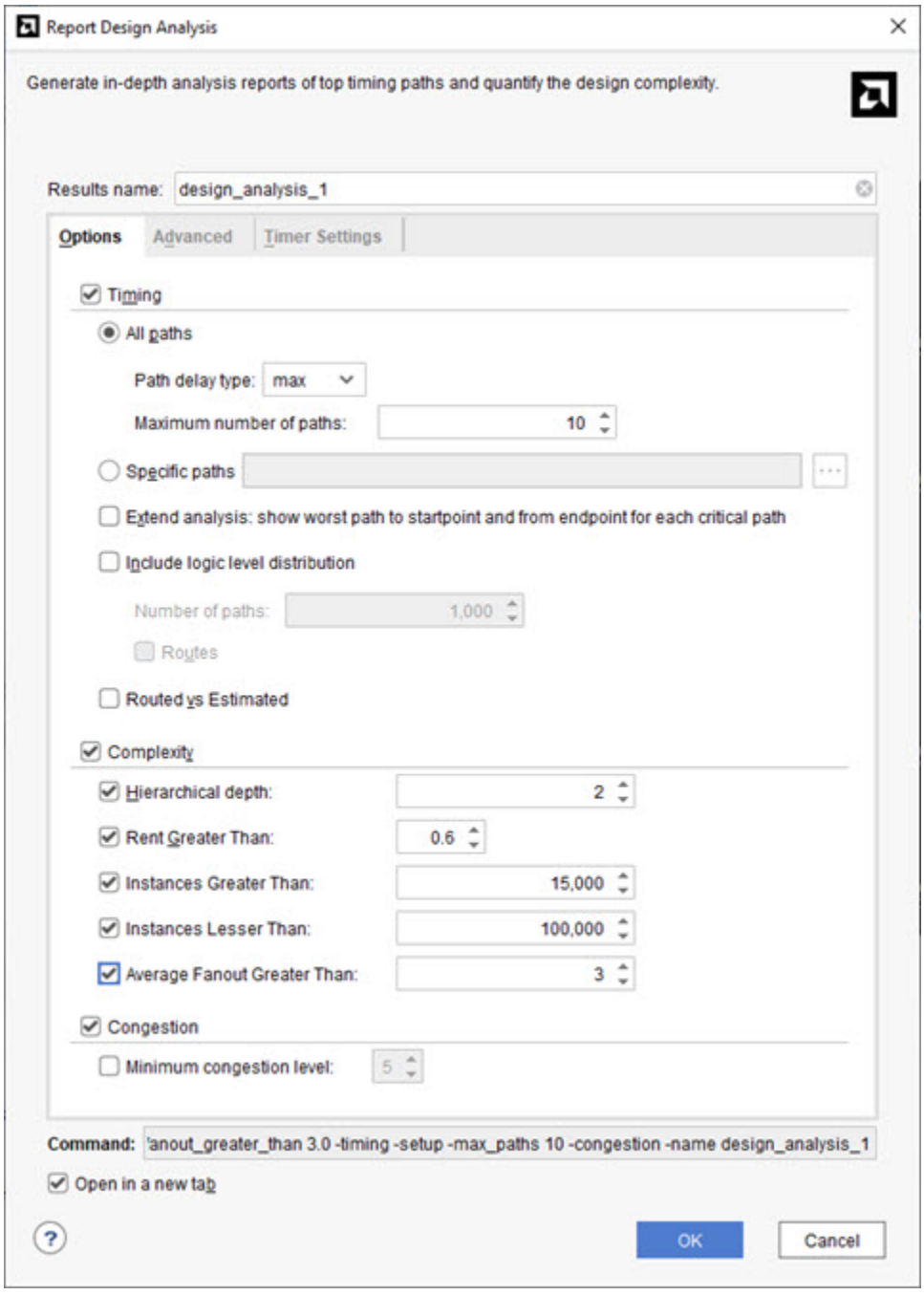
   ```
   report_design_analysis -name design_analysis_1
   ```

   ***Note*:** Some `report_design_analysis` options are only available through Tcl. Use the `-name` option to generate the report and view the results in the Vivado IDE.

2. Configure report settings in the dialog box. The dialog box includes the following tabs:

   - **Results Name:** Enter a name for the report results.Equivalent Tcl option: `-name <windowName>`
   - **Options:** Select the categories to include (Timing, Complexity, Congestion).See the Options Tab section for more information.
   - **Advanced:** Adjust advanced controls if needed.
   - **Timer Settings:** Modify timer settings used for analysis.

Figure 160: **Report Design Analysis Dialog Box**



Options Tab

Timing Field

Follow these steps to configure the Timing field in the Report Design Analysis dialog box:

1. Enable timing analysis by ticking the Timing checkbox to report timing and physical characteristics of timing paths.

   Equivalent Tcl option: `-timing`

2. Choose between All Paths or Specific Paths analysis:
   - If analyzing all paths:
     1. Select **All Paths**.
     2. Choose a delay type:
        . `max` for setup
        . `min` for hold
        . `min_max` for both setup and hold

        Equivalent Tcl options: `-setup`, `-hold`
     3. Set the Maximum Number of Paths per Clock Group (default is 10).

        Equivalent Tcl option: `-max_paths <arg>`
   - If analyzing a specific path:
     1. Select **Specific Paths**.
     2. Click **Browse** to open the search dialog box and select timing path objects.
     3. For more on selecting paths, see the `get_timing_paths` command in UG835.

     Equivalent Tcl option: `-of_timing_paths <args>`

     For more information about `get_timing_paths`, refer to get_timing_paths in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).
3. (Optional) Enable Extend Analysis by selecting **Extend Analysis**. This analyzed each path of interest along with:
   - The worst path to the startpoint
   - The worst path from the endpoint

   This helps determine whether hold fixes are affecting setup timing.

   Equivalent Tcl option: `-extend`
4. (Optional) Include logic-level distribution by selecting the Logic-Level Distribution option and specify how many paths to include.
   - If you selected All Paths, this number overrides the max paths per clock group.
   - If you selected Specific Paths, logic-level distribution applies only to those paths.

   Equivalent Tcl options:
   - `-logic_level_distribution`

- `-logic_level_dist_paths <arg>`

Complexity Field

Follow these steps to configure the Complexity field in the Report Design Analysis dialog box:

1. Enable complexity analysis:
   - Select the **Complexity** checkbox to report design netlist complexity.
   - This helps identify potential congestion based on connectivity density.
   - Equivalent Tcl option: `-complexity`

   ***Note*:** This option is compile-time sensitive, especially for large designs.
2. Set the hierarchical depth:
   - Select **Hierarchical Depth** to specify how many levels below the top cell to analyze.
   - To limit analysis and speed up processing, use the `-cells` option in the Advanced tab.
   - Equivalent Tcl option: `-hierarchical_depth <arg>`
3. Limit results using Rent coefficient threshold:
   - Use the **Rent Greater Than** option to filter the report to only include modules with Rent values above a threshold (default is 0.6).
   - Equivalent Tcl option: `-rent_greater_than <arg>`
4. Limit analysis based on instance count:
   - Use **Instances Greater Than** to ignore small modules unlikely to affect congestion. The default is 15,000.
   - Equivalent Tcl option: `-instances_greater_than <arg>`
   - Use **Instances Lesser Than** to exclude very large modules where congestion fixes might overly increase area. The default is 100,000.
   - Equivalent Tcl option: `-instances_lesser_than <arg>`
5. Filter based on average fanout:
   - Use `Average Fanout Greater Than` to skip modules with low average signal fanout. The default threshold is 3.0.
   - High Rent values combined with high average fanouts are more predictive of congestion.
   - Equivalent Tcl option: `-av_fanout_greater_than <arg>`

Congestion Field

Follow these steps to configure the Congestion field in the Report Design Analysis dialog box:

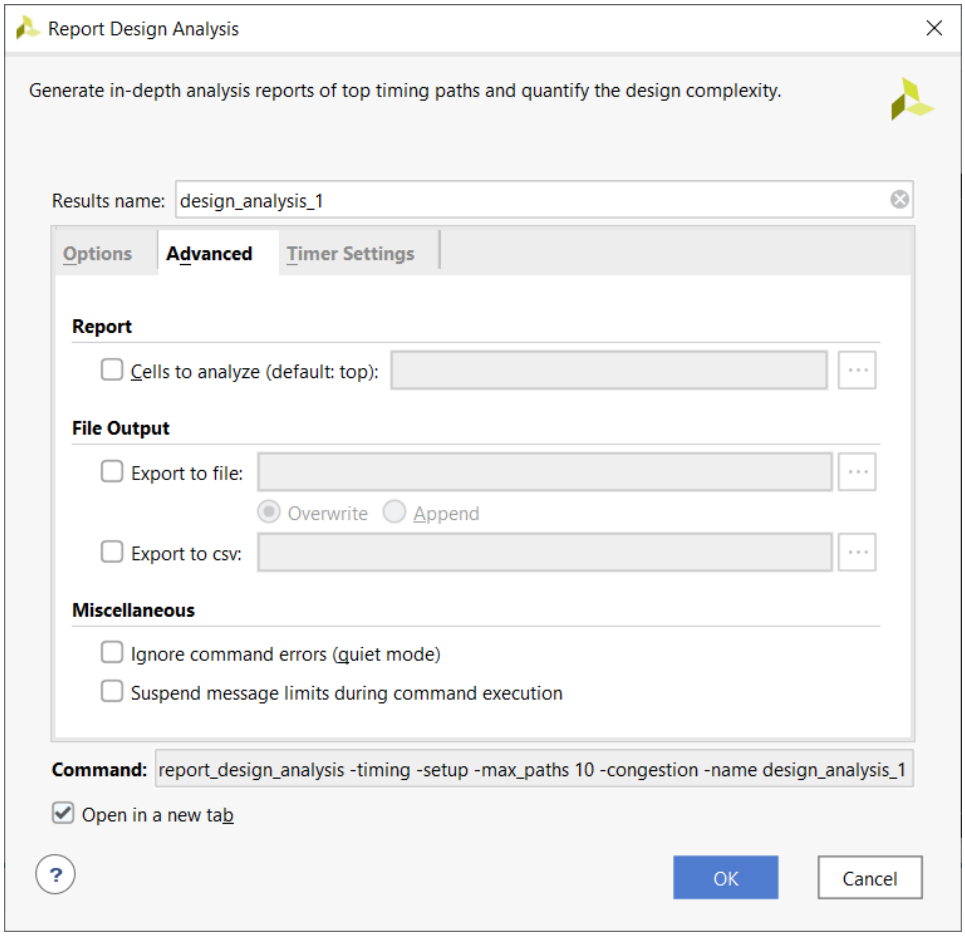
1. Enable congestion analysis:
   - Select the **Congestion** checkbox to toggle the `-congestion` Tcl switch.

- Equivalent Tcl option: `-congestion` (implied when field is enabled)

2. Set the minimum congestion level:
   - Select the **Minimum congestion level** option to set the threshold for reported router congestion.
   - Enter a value between 3 and 8. The default is 5.
   - Equivalent Tcl option: `-min_congestion_level <arg>`
3. Check results and adjust if needed:
   - If the report shows no congested regions, reduce the threshold value and rerun the report.
   - Congestion data appears only when regions meet or exceed the specified threshold.

Advanced Tab

Figure 161: **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.

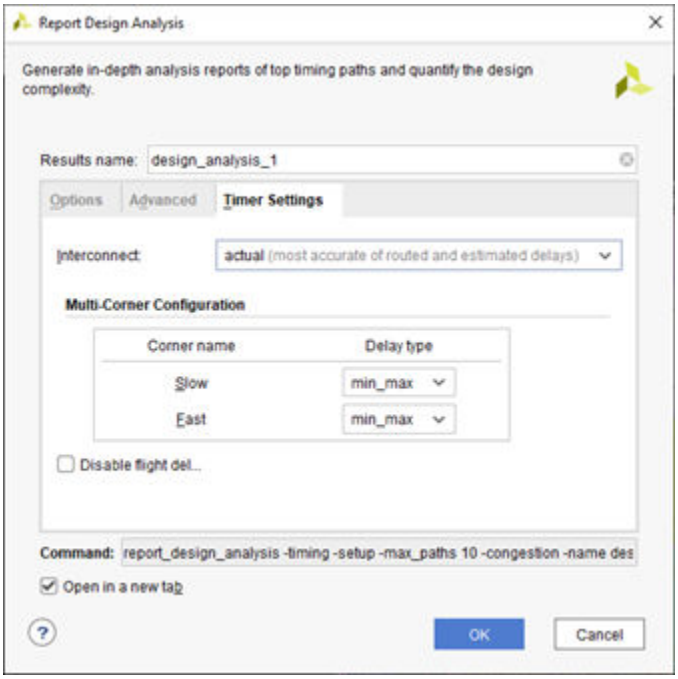
  Select **Append** to append the new results.

  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

Figure 162: **Time Settings Tab**



Follow these steps to configure the timer settings in the Report Design Analysis dialog box:

1. Select the **Interconnect**:
   - **actual:** Applies accurate interconnect delays for a fully routed design.

- **estimated:** Uses estimated interconnect delays based on placement and connectivity, even for routed designs.
- **none:** Ignores interconnect delays and uses only logic delays, which can help identify paths dominated by logic delay

Equivalent Tcl command option: `set_delay_model -interconnect <arg>`

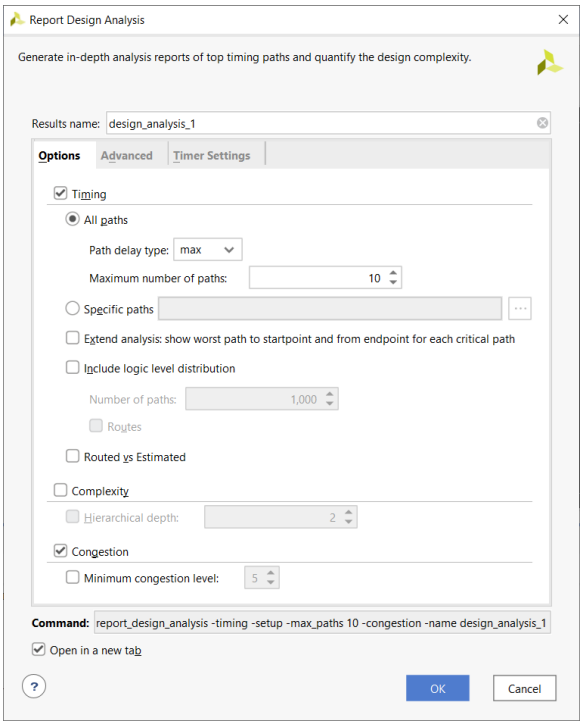
For more information about `set_delay_model`, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

2. Select the **Multi-Corner Configuration** to limit the default four-corner analysis performed by the timing engine. Adjust the corners and delay types based on your analysis needs.

   Equivalent Tcl command option: `config_timing_corners -corner <arg> -delay_type <arg>`

3. Select **Disable Flight Delays** if you want to exclude package delays from I/O timing calculations.

   Equivalent Tcl command option: `config_timing_analysis -disable_flight_delays <arg>`



For more information about these Tcl command options, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Command Line Only Options

The following options are only available through the Tcl Console. You can use them with the `-name` option to generate a GUI report.

Timing Options

- **`csv <filename>.csv`:** Generates a CSV file containing timing path information. This is useful when sorting or reviewing a large number of paths.
- **`-routed_vs_estimated`:** Reports estimated versus routed delays side by side for the same timing path. Fields within the Timing category are labeled Estimated or Routed for direct comparison.
- **`-max_level <arg>`:** Groups all timing paths with logic levels (or routing depth) greater than the specified value into a single bin. This value must be greater than that specified for `-min_level`.
- **`-min_level <arg>`:** Groups all timing paths with logic levels (or routing depth) less than the specified value into a single bin. The value must be at least 1.
- **`-return_timing_paths`:** Returns timing path objects for further analysis based on logic level and clock domain. This option must be used with both `-end_point_clock` and `-logic_levels`.
- **`-end_point_clock <arg>`:** Limits logic-level distribution reporting to timing paths with the specified endpoint clock.
- **`-logic_levels <arg>`:** Filters timing paths used in the logic-level histogram by logic depth. Only one logic level value can be specified.

Complexity Options

`-bounding_boxes <arg>` performs complexity analysis on specific device regions. Provide bounding boxes in the following format:

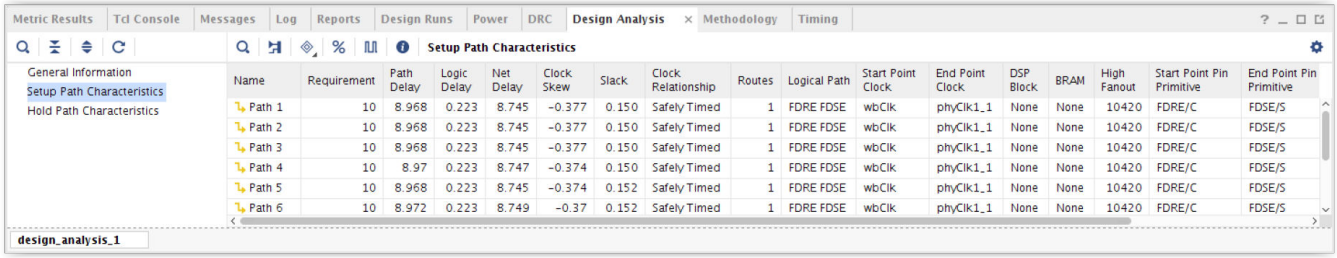
```
-bounding_boxes { "CLE_M_X21Y239:CLEL_R_X28Y254"
"CLEL_R_X18Y171:CLE_M_X26Y186" }
```

Note: A space must appear between the opening `{` and the first bounding box.

Timing Path Characteristics Report

After you run Report Design Analysis in Timing mode, you can view a report showing characteristics of the ten worst setup paths.

Figure 163: **Example Setup Path Characteristics**



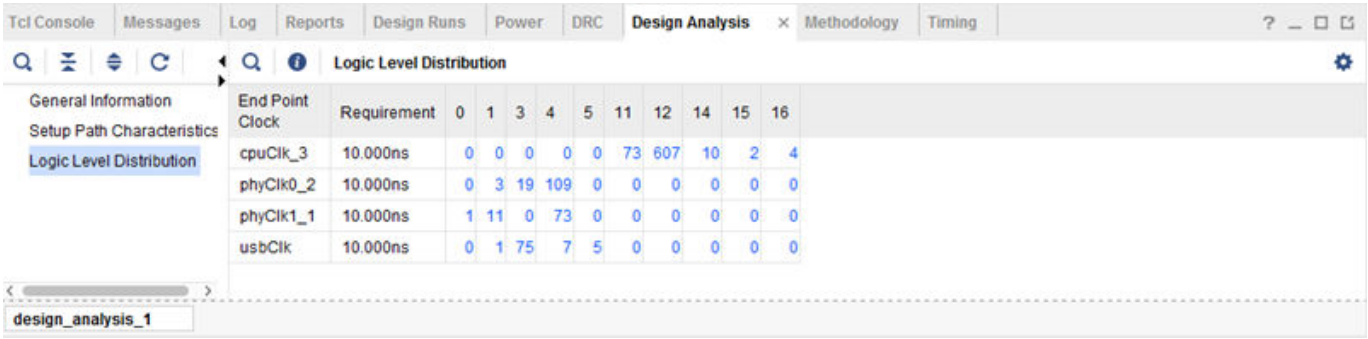
You can generate this report in either of the following ways:

- In the Vivado IDE, select **Reports → Report Design Analysis**.
- In Tcl, enter the 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 option. For more information on Tcl command option syntax, see the Vivado Design Suite Tcl Command Reference Guide (UG835).*

The Logic Level Distribution table shows the distribution of logic levels for the worst timing paths.

Figure 164: **Example of Logic Level Distribution Report**

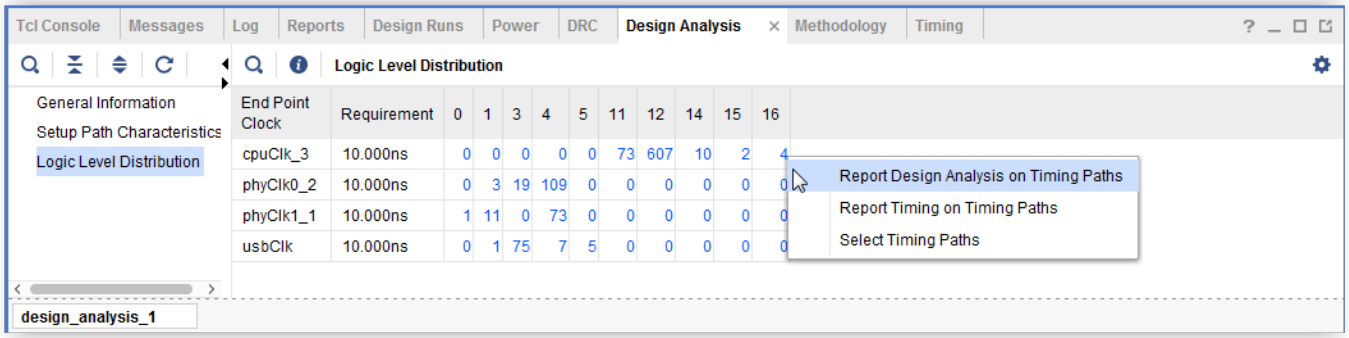


To generate this table:

1. Select **Options → Include logic level distribution**.
2. By default, the tool analyzes 1,000 paths. You can change this value in the same tab.

In the GUI, logic level bins now include hyperlinks.

Figure 165: **Report Design Analysis on a Selected Path**



Click a bin to do one of the following:

- Run `report_design_analysis` or `report_timing` on paths in that bin.
- Select specific timing path objects.

Command Line Enhancements for Logic Level Analysis

- Use `-routes` with `-logic_level_distribution` to generate bins based on routing segments instead of logic levels.
- Use `-min_level` and `-max_level` to control which logic levels are grouped into bins.
  - Paths with logic levels below `-min_level` are grouped in one bin.
  - Paths above `-max_level` are grouped in another.
  - All other paths are binned individually by logic level if at least one path exists at that level.

Figure 166: **Logic Level Analysis with `-routes` Enhancements**

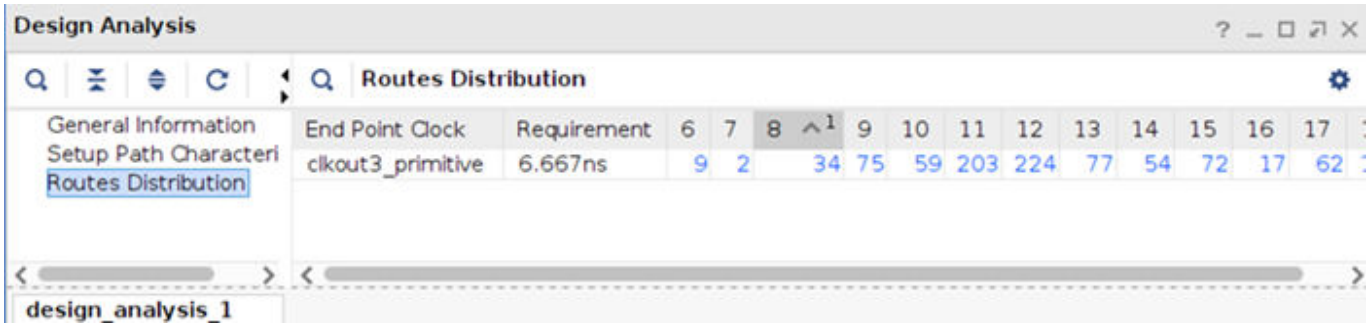
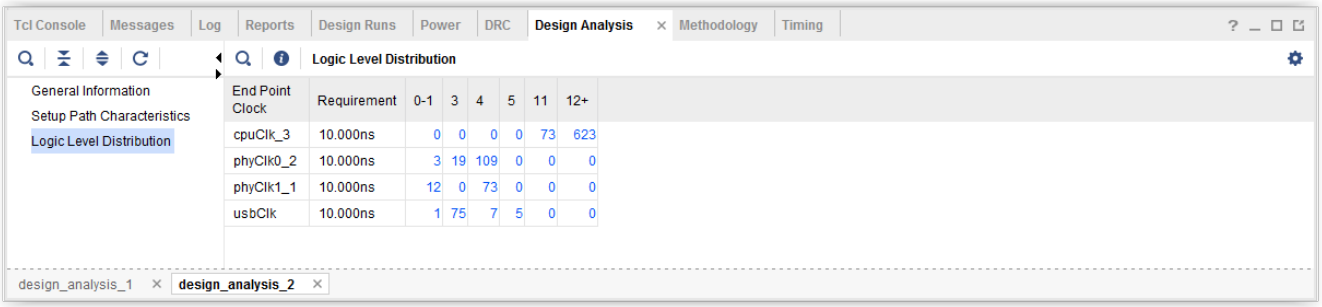


Figure 167: **Logic Level Analysis with `-min_level` and `-max_level` Enhancements**



For example, if your design has logic levels of 0, 1, 3, 4, 5, 11, 12, 14, 15, and 16, and you specify:

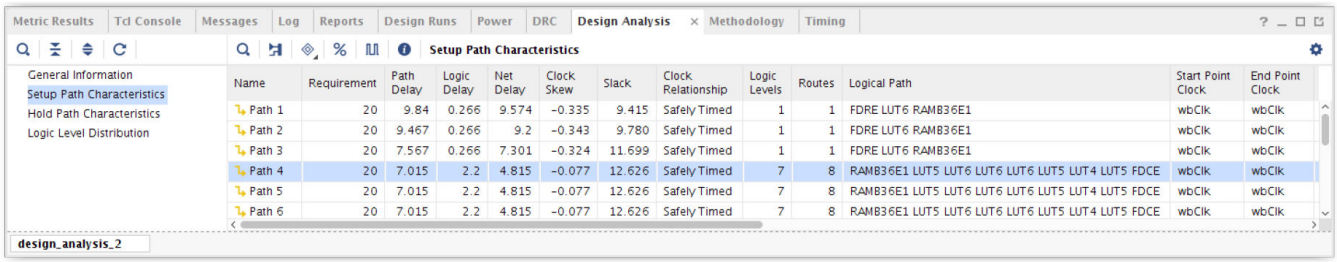
```
-min_level 3 -max_level 11
```

The report creates these bins: 0–2, 3, 4, 5, 11, and 12+. See Timing Path Characteristics Report for reference.

Analyzing Specific Paths

You can analyze specific timing paths using Report Design Analysis in Timing mode.

Figure 168: **Example of Specific Timing Path Characteristics**



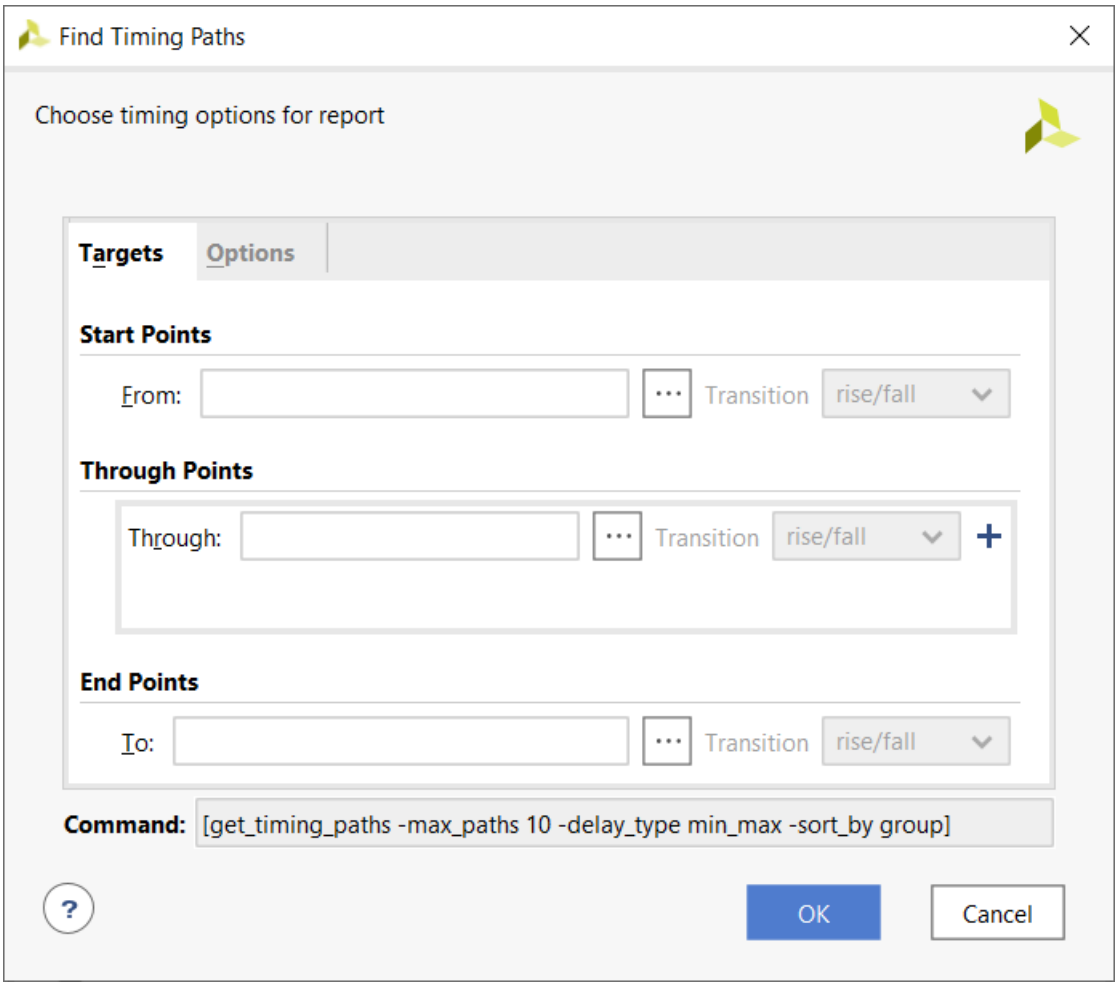
The previous figure is an example report from Report Design Analysis in Timing Mode with specific paths selected.

When you select specific paths, the Path Characteristics and optional Logic Level Distribution tables display data only for the selected paths.

To specify which paths to analyze:

1. Select **Specific paths** in the Report Design Analysis dialog box.
2. Click the **Browse** button next to that option.
3. In the Find Timing Paths dialog box that opens, choose the paths you want to analyze.

Figure 169: **Find Timing Paths Dialog Box**

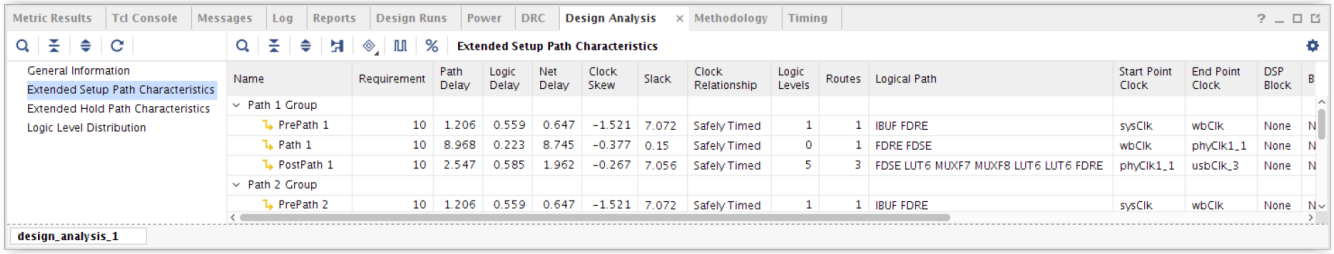


Analyzing the Worst Path along with Preceding and Following Worst Paths

You can enable Extend analysis in Report Design Analysis (Timing mode) to analyze not only the worst setup path, but also:

- The worst setup path to the startpoint cell (`PrePath`)
- The worst setup path from the endpoint cell (`PostPath`)

Figure 170: **Extended Path Characteristics of the Worst Setup Path**



***Note*:** The Extend Analysis for All Paths option is currently available only for setup analysis.

To run this analysis:

- In the Vivado IDE, select the **Extend analysis** checkbox in the dialog box.
- In Tcl, use the following command: `report_design_analysis -extend`

This option increases runtime because it performs multiple timing analyses to collect characteristics for each reported path. Use it when you need full context around the worst setup path.

Reading and Interpreting Timing Path Characteristics Reports

The timing path characteristics in Report Design Analysis fall into five main categories: timing, logic, physical, property, and DFX-specific information for dynamic function exchange designs. To view detailed descriptions of each characteristic, run the following Tcl command option: `report_design_analysis -help`

You can also find the same information in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Category 1: Timing

Timing characteristics include Path Type, Requirement, Slack, and Timing Exception. These fields help you understand the type of analysis (SETUP or HOLD), whether the timing is met, and if any exceptions apply (such as `multicycle` or `max_delay`).

When debugging missing or incorrect timing constraints, check the path requirement first:

- Review setup paths with requirements under 4 ns, especially for clock domain crossing paths.
- Investigate paths with setup requirements under 2 ns. These are difficult to meet and usually indicate a problem.
- For paths with small setup requirements, look for missing timing exceptions or incorrect clock relationships.
- Check for positive hold requirements. These are uncommon and usually signal that multicycle hold constraints are missing or applied only to setup.

- Examine source and destination clocks to confirm expected edge alignment.

For the datapath, review the Path Delay, Logic Delay, and Net Delay values:

- If Logic Delay accounts for more than 50% of the total path delay, examine logic depth and cell types. You might need to change the RTL or synthesis settings.
- If Net Delay dominates a path with a reasonable requirement, look at physical and property characteristics. Check for high fanout nets or hold fix detours.
- In 7 series devices, LUT delay is included in net delay. In UltraScale devices, it appears under logic delay. Expect higher Net Delay to Logic Delay ratios in 7 series architectures.

Category 2: Logic

Logic characteristics describe the basic path structure:

- Start Point Pin Primitive and End Point Pin Primitive show the types of the starting and ending cells. Confirm these are expected for your design.
- Start Point Pin and End Point Pin provide the exact pin locations.
- Logic Levels and Logical Path summarize path depth and primitive usage.
- Routes indicate the number of routable nets in the path.

Check for control signals like CLR, PRE, RST, and CE that often appear on high-fanout nets. Paths with DSPs or block RAMs tend to have tighter timing budgets due to longer Clock-to-Q delays.

If you see mostly LUTs in the path, investigate:

- Why smaller LUTs are used instead of larger LUT6s.
- Whether constraints like KEEP, DONT_TOUCH, or MARK_DEBUG are limiting synthesis optimization.
- Whether changing synthesis options or using `opt_design -remap` can reduce logic depth.

Look at the presence of DSPs or RAMs. Timing is harder to meet when these cells are not registered and are followed by deep logic.

Category 3: Physical

Architectural Boundary Crossings

Architectures are organized into super logic regions (SLRs), and each SLR contains vertical columns of fabric resources such as logic, BRAMs, DSPs, and other features. When a datapath net crosses from one SLR to another, the design incurs a timing penalty. The SLR crossing value in the report indicates how many SLR boundaries the net crosses.

In addition to SLRs, a datapath net might cross non-slice columns. These crossings also add delay to the path and can contribute to routing detours, especially in congested regions. You can identify these crossings using property fields such as:

- I/O Crossings
- RAM Crossings
- DSP Crossings
- NOC Crossings
- CONFIG Crossings
- OTHER Crossings

Each value shows how many columns of that type the path crosses. While crossing architectural columns does not always cause timing issues, check for high net delay or large skew in combination with these crossings. If a particular module repeatedly shows timing violations across multiple runs and includes many column crossings, consider using minimal floorplanning with Pblocks to reduce the number of crossings or to localize the logic within one SLR.

Path Placement Restrictions: Pblocks

Excessive floorplanning with Pblocks can also create timing problems. When a timing path crosses multiple Pblocks, review their locations and assess whether the floorplan is limiting placement flexibility. If the Pblocks are adjacent, try combining them into one larger Pblock to reduce placement restrictions. If physical separation is required between the Pblocks, consider pipelining between them to improve timing.

Placement Box: Bounding Box Size, Clock Region Distance, Combined LUT Pairs

The bounding box size and clock region distance of a timing path also affect timing. If either metric is too large, try applying additional directives in the `place_design` step. In UltraScale devices, clock region distance has a greater impact and can contribute to increased clock skew.

Net Fanout and Detour

The report shows both High Fanout and Cumulative Fanout values. High fanout typically leads to routing delays and can explain setup failures. If physical optimization does not reduce fanout, check whether MARK_DEBUG or DONT_TOUCH constraints are blocking replication.

If you want to apply replication before implementation, you can use the `MAX_FANOUT` constraint in synthesis. Add this constraint in your RTL or in the XDC file. However, synthesis replication is not always effective for high-fanout nets because the placement has a strong influence on timing. Instead, rely on physical optimization after placement. To increase effort during implementation, use directives such as Explore, AggressiveExplore, or AggressiveFanoutOpt.

If you need to reduce fanout for a specific net, force replication with this command:

```
phys_opt_design -force_replication_on_nets <netName>
```

The report might also show that Hold Fix Detour is asserted. This indicates the router intentionally added extra delay to the path to meet the hold time requirement. If the same path is now failing setup timing, check for excessive skew between the clocks and verify the constraints between the source and destination clocks. Hold requirements are generally zero or negative. A positive hold requirement often indicates a missing or incomplete multicycle constraint for hold analysis.

***Note*:** The HOLD_DETOUR property is set only during certain hold-fixing phases in the router. Other routing stages might also introduce detours for hold paths, but the property is not set in those cases. For example, if the detour occurs due to congestion rather than explicit hold fixing.

Category 4: Property

Review properties that might block optimization:

- Combined LUT Pairs can restrict placement and increase congestion. If this causes issues, disable LUT combining in synthesis using the `-no_lc` option.
- MARK_DEBUG and DONT_TOUCH identify objects that cannot be optimized. By default, setting MARK_DEBUG also sets DONT_TOUCH.

If optimization is blocked:

- Set DONT_TOUCH to FALSE where appropriate.
- Avoid using DONT_TOUCH unless needed for logic preservation.
- If a net crosses into a hierarchical cell marked with DONT_TOUCH, only the external portion can be optimized.

If the path has fixed placement or routing:

- Fixed Loc and Fixed Route show constraints that might restrict timing optimization.
- Fixed cell locations can help stabilize timing but might prevent recovery after design changes.
- Fixed routes prevent delay optimization. Use them only when necessary, and ensure they don't negatively affect connected paths.
- If you modify physical constraints like Pblocks, update fixed placement and routing constraints as needed.

Category 5: Dynamic Function eXchange Designs

For Dynamic Function eXchange (DFX) designs, cell names in the logical path are prefixed to show whether they belong to a reconfigurable partition (RP), or to the static region of the design (S:).

- **DFX Path Type:** Specifies whether the path is entirely in the static region, entirely in an RP, or crosses between regions.

- **DFX Boundary Nets:** Shows how many boundary crossings occur between static logic and RMs or between two RMs.
- **Boundary Fanout:** Shows the fanout of the net at the reconfigurable partition boundary (PPLOC) to downstream loads.

Complexity Report

The complexity report provides insight into the Rent Exponent, average fanout, and the distribution of leaf cell types for the top-level design or specified hierarchical cells. These metrics help you estimate routing complexity and anticipate placement challenges.

The Rent exponent describes the relationship between the number of ports and the number of cells in a partitioned section of the netlist. It is calculated using a recursive min-cut algorithm similar to the one used by the placer during global placement. If your design hierarchy closely aligns with the physical partitions determined during placement, this value can give you an early indication of how difficult the design might be to place and route.

Rent's rule is defined as:

Equation 1: **Rent's Rule**

$$ports = constant \times cells^{Rent}$$
$$log(ports) = Rent \times log(cells) + constant$$

A higher Rent exponent indicates that groups of highly connected logic also maintain strong interconnectivity with other logic groups. This often results in increased routing complexity and heavier use of routing resources. The Rent exponent reported here is based on the unplaced and unrouted netlist.

After placement, the same design can produce a different Rent exponent because it is then measured against physical partitions instead of logical hierarchy. Post-placement Rent is not included in this report. For placed designs, analyze congestion reports instead.

Vivado runs Report Design Analysis in Complexity Mode when you either:

- Select the Complexity checkbox in the Report Design Analysis dialog box (Options tab).
- Use the Tcl command `report_design_analysis` with any of the following options:

Table 15: **Options that Run Report Design Analysis in Complexity Mode**

Tcl Option	Description
`-complexity`	Forces the report to run in Complexity Mode.
`-cells <arg>`	Limits the hierarchical cells considered to cells under the specified levels of hierarchy.

Table 15: **Options that Run Report Design Analysis in Complexity Mode** *(cont'd)*

Tcl Option	Description
`-hierarchical_depth <arg>`	The levels of hierarchy under the qualifying cells to be considered for reporting.
`-av_fanout_greater_than`	Minimum average fanout of nets within a hierarchical cell.
`-instances_greater_than`	Minimum number of instances a hierarchical cell must contain.
`-instances_lesser_than`	Maximum number of instances a hierarchical cell must contain.
`-rent_greater_than`	Only report on hierarchical cells that exceed the rent value specified.

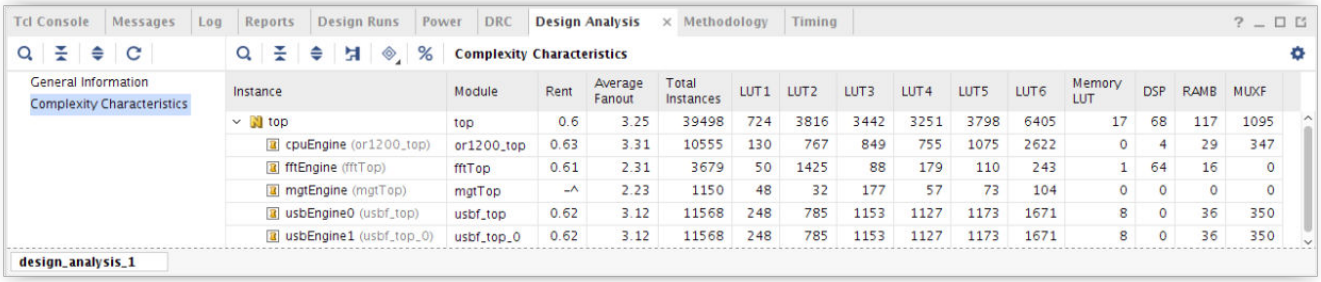
Analyzing the Design Complexity at the Top Level

You can use Report Design Analysis in Complexity Mode to evaluate connectivity and fanout at the top level of the design and its immediate hierarchy.

To generate a complexity report that includes the top module and one level below it, run the following Tcl command option:

```
report_design_analysis -complexity -hierarchical_depth 1
```

Figure 171: **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

***Note*:** To simplify report navigation, right-click any column header to enable column filtering. You can apply multiple filters to reduce the report size, especially when a large number of hierarchies are present.

Reading and Interpreting a Complexity Report

Use the Complexity Characteristics table to evaluate Rent exponent and average fanout values for each level of hierarchy. These metrics help identify potential placement and routing issues early in the design process. By comparing the measured values against typical ranges and considering other contributing factors, you can determine whether corrective action is needed to improve design performance.

1. Review the Complexity Characteristics table to identify the Rent exponent and average fanout for each hierarchy level below the top level.

2. Compare each metric to these ranges:
   - **Rent exponent:**
     - 0.0–0.65: Low to normal complexity; no potential problems.
     - 0.65–0.85: High complexity, especially with more than 25,000 total instances.
     - Above 0.85: Very high complexity; with a high number of instances, implementation can fail.
   - **Average fanout:**
     - Below 4: Normal.
     - 4–5: Can cause placement difficulty and congestion. For SSI devices with more than 100,000 instances, placement in one SLR or across two SLRs can be problematic.
     - Above 5: Implementation can fail.
3. Always check the Total Instances column along with the Rent exponent and average fanout. Larger modules with high values have higher priority for review. Smaller modules (under 10,000 instances) can sometimes have high values without placement or routing issues.
4. Keep in mind that complexity metrics alone might not predict routing congestion. Review congestion reports after placement to assess the effects of I/O constraints, floorplanning, and macro primitive locations.
5. Consider these additional factors when interpreting results:
   - High LUT6 percentage increases average fanout and possibly the Rent exponent.
   - Many RAMB or DSP primitives can raise the Rent exponent due to high connectivity.
   - High metric values for some hierarchical instances might not be problematic because the placer can split them into easier groups of logic.
   - Low Rent exponent modules can still cause congestion if large dataflows pass through them.
6. If a large module with high metrics causes congestion or timing issues, take these actions:
   a. Reduce module connectivity by preserving hierarchy to limit cross-boundary optimization, which reduces LUT6 usage and netlist density.
   b. Reduce LUT inputs during synthesis.
   c. Disable LUT combining in synthesis.
   d. Use a Congestion Strategy or SpreadLogic placement directive. For SSI devices, try multiple SSI placement directives.
   e. Apply simple floorplanning at the SLR level (SSI devices) or clock region level (general) to separate congested logic or guide placement toward a previously successful configuration.
   f. Use QoR suggestions and Intelligent Design Runs to address congestion automatically.

Congestion Report

The Congestion report identifies congested areas in the device and lists the design modules located in those regions. Congestion can restrict placement options and create timing closure challenges, particularly when critical paths are located in or near congested zones.

Analyzing the Design Congestion

To analyze design congestion in Report Design Analysis:

1. In the Report Design Analysis dialog box, open the Options tab and select **Congestion**.
2. Ensure the design is placed and/or routed before running the report. Running Congestion Mode on an unplaced design produces no results.

Report Design Analysis in Congestion Mode generates three congestion tables:

- Placer Final Congestion Reporting
- Router Initial Congestion Reporting

Maximum Congestion Reports

The Maximum Congestion reports list all windows with the same maximum congestion level in a specific routing direction. The following columns appear in the report:

- **Direction:** The direction of the congested routing resources (North, South, West, or East).
- **Congestion Level:** The maximum congestion level in CLB tiles.
- **Congestion:** The estimated routing resource utilization in the defined window. This value can exceed 100%.
- **Congestion Window:** The bounding CLB tiles where the congestion for the identified direction is present. The coordinates correspond to the lower-left and upper-right corners of the window. This column is available only in the text report. In the GUI report, selecting the congestion window highlights the congested area in the Device window.
- **Cell Names:** The parent instance containing the hierarchical cells in the congestion window, listing up to the three largest contributors with their contribution percentage. In the GUI report, these names are hyperlinked to highlight the respective leaf cells in the congestion window.
- **Avg LUT Input:** The average number of inputs for the LUTs in the window.
- **COMBINED LUTs %:** 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 UltraRAM 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.

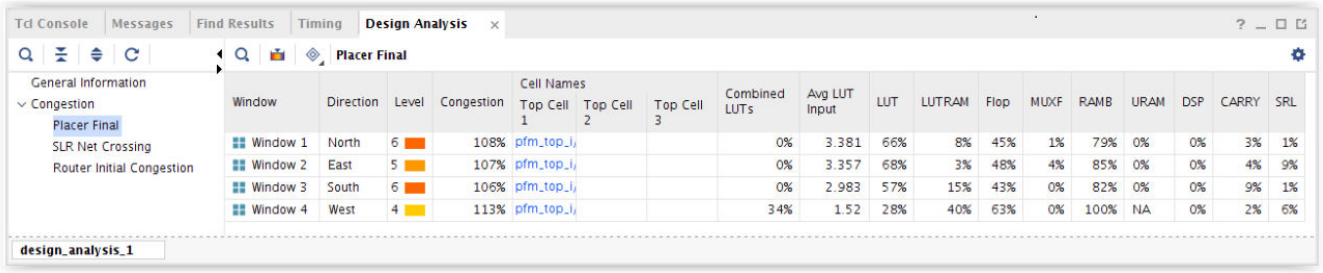
Placer Final Congestion Reporting

The Placer Final Congestion Reporting table identifies congested areas in the device, defined by the congestion window, and lists the modules located in those areas. The resource usage percentages indicate the types of resources present in the congested region.

1. Open the Placer Final Congestion Reporting table for your design.
2. Check for high LUT usage.
3. If high LUT usage exists, examine the instances that have a high percentage of LUT6s in the Complexity report.
4. If the congested area has high RAMB or DSP utilization, check for Pblock constraints that limit the available placement area of the reported modules.
5. Use targeted placement directives to relieve congestion, such as BlockPlacement or SpreadLogic.
6. In some cases, reuse the RAMB or DSP placement from a previous run that showed low congestion and good Timing QoR.
7. Use the report to examine areas of the device defined by the Congestion window along with the modules residing in that window.
8. Use the resource usage percentages to understand the types of resources located in the congested area.

The following figure shows an example of the Placer Final Congestion Reporting table. This report highlights areas of the device defined by the congestion window, lists the modules in those areas, and shows resource usage percentages to indicate the types of resources in the congested region.

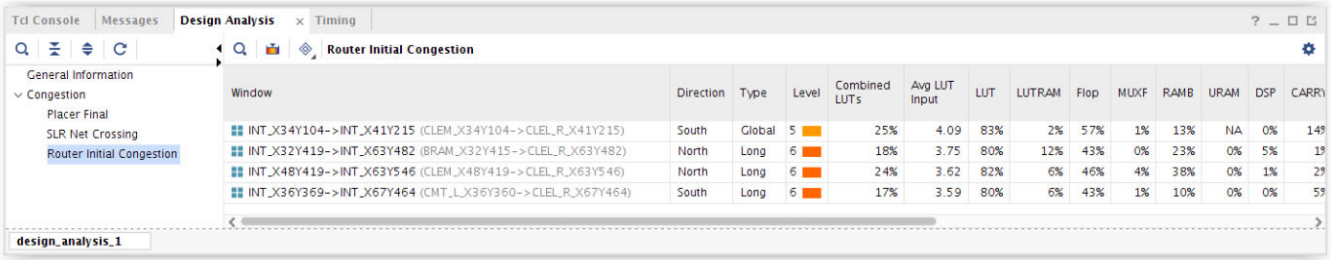
Figure 172: **Example Placer Final Congestion Reporting Table**



Router Initial Congestion Reporting

The Router Initial Congestion report (called Initial Estimated Router Congestion for 7 series FPGAs) is available only after the router has been run. It shows the routing congestion encountered by the router during the early stages of routing.

Figure 173: **Example of Router Initial Congestion Reporting Table**



When the congestion level is 5 or higher, the `report_design_analysis` command generates a congestion table with details about the nature of the congestion and the region(s) associated with the highest congestion in a particular direction and type:

- **Global congestion:** Estimated similar to placer congestion and based on all types of interconnects.
- **Long congestion:** Considers only long interconnect utilization for a given direction.
- **Short congestion:** Considers all other interconnect utilization for a given direction.

Any congestion area greater than 32×32 (level 5) can impact QoR and routability. Congestion on long interconnects increases short interconnect usage, leading to longer routed delays. Congestion on short interconnects can increase runtimes and, when widespread, degrade QoR. To analyze, follow these steps:

1. Open the Router Initial Congestion table.
2. If the congestion level is greater than 6, the design is unlikely to meet timing and might fail during routing.
3. If the congestion level is 4 or 5, identify the module(s) located in the congested area(s).

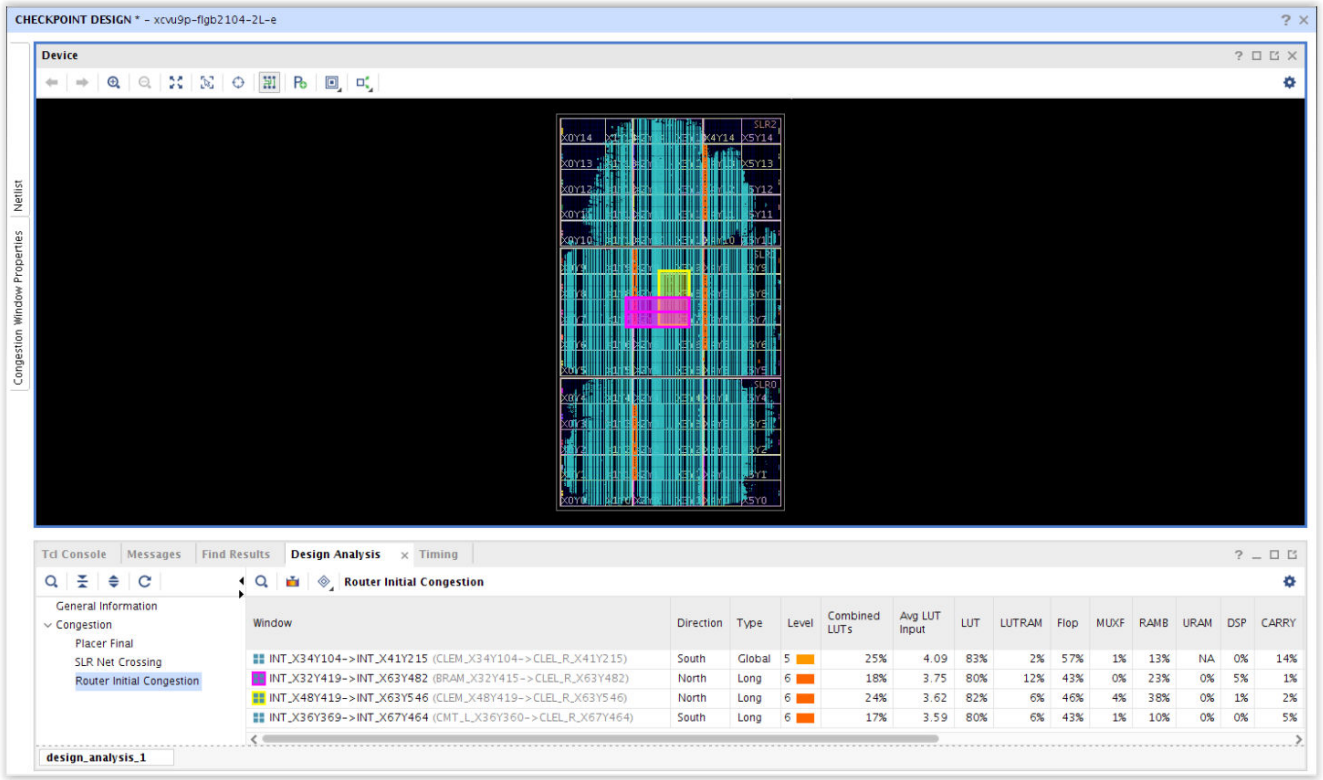
4. Apply a congestion alleviation technique to these modules or rerun placement with different directives, such as SpreadLogic.
5. If the congestion level is 3 or less, congestion is probably not a cause for concern unless the design has a very tight timing budget.

The previous figure shows an example where regions with congestion level 5 or higher are reported. To generate a congestion report with a lower congestion threshold, use the `-min_congestion_level` switch (default is 5, valid range 3–8).

In addition to the region with the maximum congestion in a given direction and type, the report can also contain other regions with the same maximum congestion level in that direction and type. These regions might overlap or appear in different areas of the device.

The following figure shows an example where the design has a congestion level 6 for North (Direction) Long (Type) in more than one region.

Figure 174: **Example of Router Initial Congestion Reporting Table**



Design QoR Summary

The `-qor_summary` command-line option generates a quality of results (QoR) summary for each step of the flow. This option is available only from the Tcl Console and can produce either a text-based report or a JSON-formatted report.

To generate the text-based format, run:

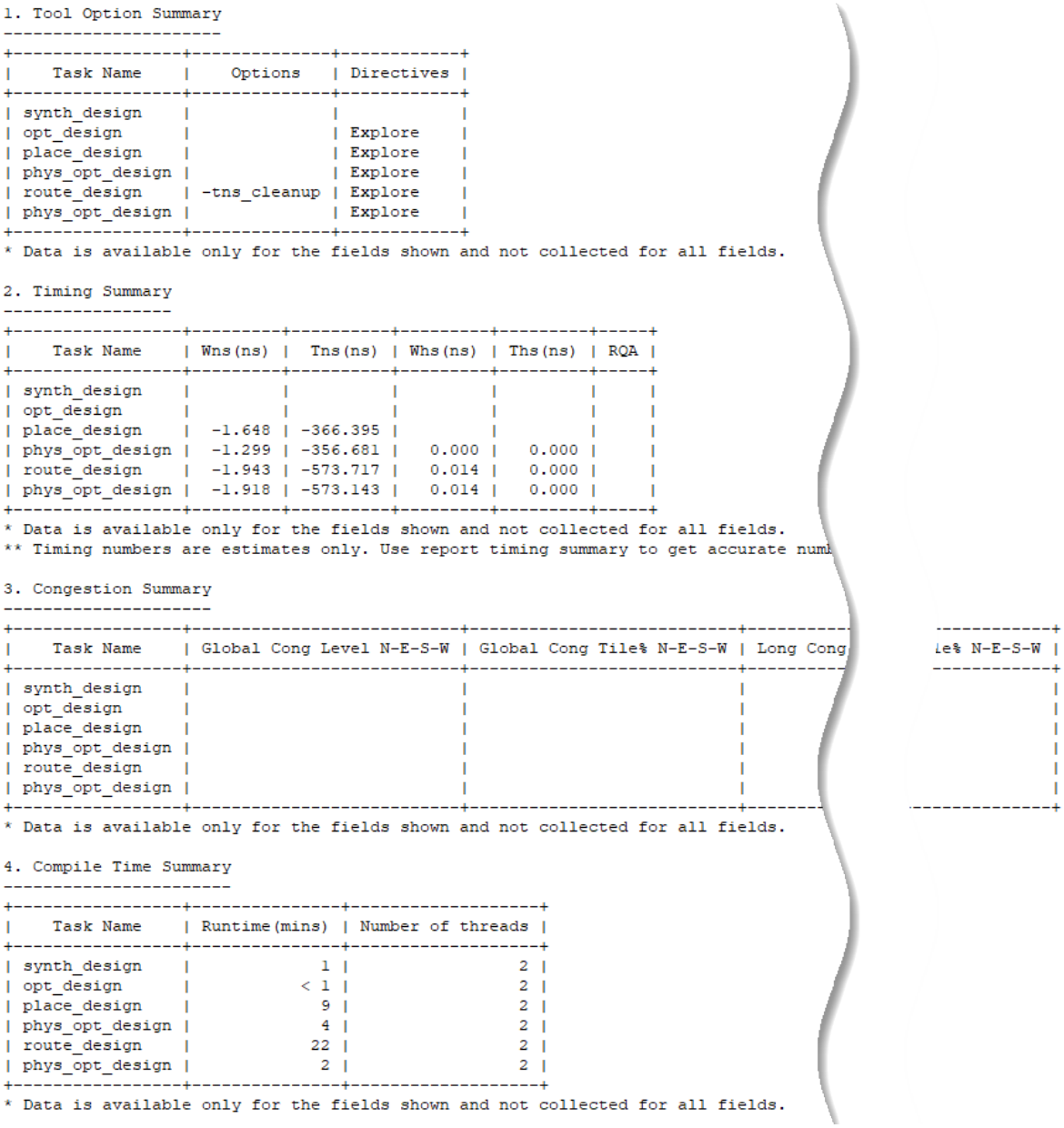
```
report_design_analysis -qor_summary
```

To generate the JSON format, run:

```
report_design_analysis -qor_summary -json <json filename>
```

The JSON format presents the table as a flat, single table, making it easier to parse. This section of the report is the only one that can be output in JSON format, but it provides an efficient way to extract key design metrics.

Figure 175: **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.

Using Report Design Analysis

Two main categories of QoR problems encountered during design analysis:

- Timing Violations
- Congestion

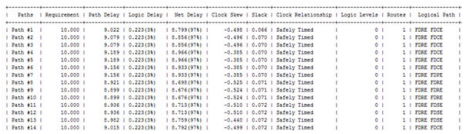
Timing Violations

While fixing the worst timing violation often improves overall QoR, you must also review other critical paths, as they frequently contribute to timing closure challenges.

Follow these steps to identify and address timing violations using Report Design Analysis:

1. Generate the Setup Path Characteristics table by running the following. In this example, the command reports the 50 worst setup timing paths.

```
report_design_analysis -max_paths 50 -setup
```



2. Review the table to isolate which characteristics are introducing timing violations for each path.
3. If the violation is caused by a high logic delay percentage (Logic Delay):
   - Check if the path has many levels of logic (Logic Levels).
   - Determine whether constraints or attributes prevent logic optimization (Don't Touch, Mark Debug).
   - Identify cells with high logic delay such as RAMB or DSP.
   - Evaluate whether the path requirement is too tight for the current path topology (Requirement).
4. If the violation is caused by a high net delay percentage (Net Delay):
   - Look for high fanout nets (High Fanout, Cumulative Fanout).
   - Check whether cells are assigned to several Pblocks that might be placed far apart (Pblocks).
   - Check whether cells are placed far apart (Bounding Box Size, Clock Region Distance).
   - For SSI devices, check whether nets cross SLR boundaries (SLR Crossings).
   - Investigate unusually high net delay values when placement appears correct; see the Congestion section for reference.
5. If the violation is due to a missing pipeline register in a RAMB or DSP cell:
   - Verify that pipeline registers are enabled for RAMBs or DSP cells.
6. If the violation is due to high skew (<-0.5 ns for setup and >0.5 ns for hold) (Clock Skew):
   - Check whether the path is a clock domain crossing path (Start Point Clock, End Point Clock).
   - Determine whether the clocks are synchronous or asynchronous (Clock Relationship).

- Check whether the path crosses I/O columns (I/O Crossings).

7. Review the Logic Level Distribution table generated for the worst 1000 paths to identify long paths in the design.

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

8. Eliminate longer paths where possible to improve overall QoR. Based on your findings, improve the netlist by modifying the RTL, changing synthesis options, or adjusting timing and physical constraints.

Congestion

Follow these steps to generate and analyze congestion reports using Report Design Analysis:

1. In the same Vivado session where the placer and router ran, generate the congestion tables by running:

```
report_design_analysis -congestion
```

2. Review the generated congestion tables, which show congested areas as seen by the placer and router.

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%	3.288	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%	3.468	93%	0%	8%	0%	0%	NA	35%	0%	top(100%)
South	Short	5	(CLE_E_CORE_X64Y160,CLE_E_CORE_X79Y191)	13%	3.509	94%	0%	7%	0%	NA	NA	36%	0%	top(100%)

3. The Module Names in the report correspond to the hierarchical cells present in each reported tile.

4. Retrieve the complete name of a module by running:

```
get_cells -hier <moduleName>
```

5. After identifying the hierarchical cells present in the congested area, apply congestion-alleviating techniques to try reducing overall design congestion.

Chapter 6

Synthesis Analysis and Closure Techniques

This chapter describes methods for analyzing synthesized designs and making targeted RTL and synthesis-level changes to improve quality of results (QoR). The techniques focus on identifying and addressing timing bottlenecks, optimizing memory structures, and restructuring critical logic to make better use of FPGA resources.

Using the Elaborated View to Optimize the RTL

When analyzing timing results after any implementation step using `report_timing`, `report_timing_summary`, or `report_design_analysis`, review the structure of critical paths to determine whether they can be mapped to logic primitives more efficiently by modifying the RTL, using synthesis attributes, or applying different synthesis options. This is especially important for paths with a high number of logic levels, which can strain the implementation tools and limit overall design performance. For more information about the Linter tool, see the *Vivado Design Suite User Guide: Synthesis* (UG901)

If you find a critical path with many logic levels, assess whether the functionality truly requires that logic depth. Determining the optimal number of logic levels depends on both your knowledge of the design and RTL optimization in general. Analyzing the post-synthesis optimized netlist to pinpoint RTL inefficiencies can be complex.

In project mode, the AMD Vivado™ Integrated Design Environment (IDE) provides a cross-probing mechanism between the synthesized or implemented design and the elaborated design to simplify this analysis.

1. Open both the synthesized or implemented design and the elaborated design in memory.
2. In the synthesized or implemented design view, select the timing path and show its schematics by pressing **F4** key.
3. In the Flow Navigator pane, select **Elaborated Design**. The RTL cells corresponding to the timing path are selected automatically. You can then:
   - Press **F4** to open the RTL schematics and view the same path in the elaborated view.

- Trace from the endpoint pin back to the startpoint cell.

Example

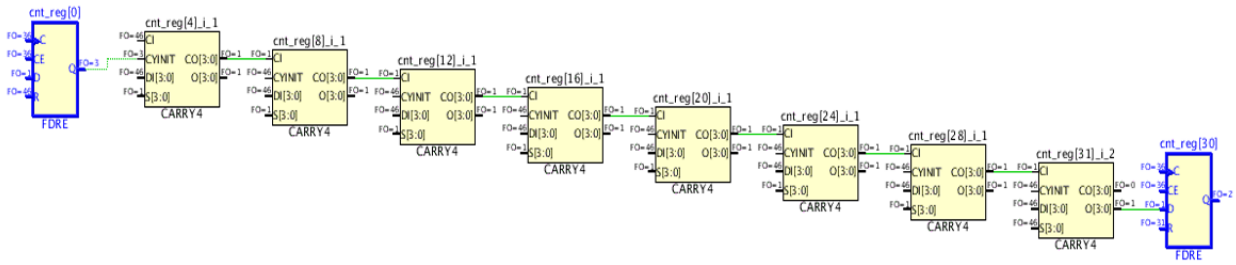
The following example uses a counter design.

Figure 176: **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;
```

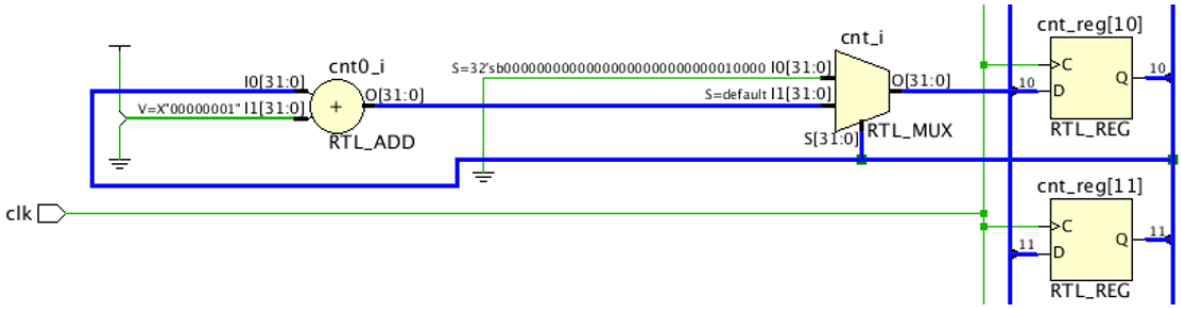
The signal `cnt` counts from 0 to 16, which requires a 5-bit vector to encode.

Figure 177: **cnt Counter Post-Route Critical Path Schematic**



In this example, the post-route critical path schematics shows the endpoint as bit 30 of the `cnt` signal. After selecting the startpoint and endpoint cells of the critical path, you can visualize the equivalent path in the elaborated view by opening a schematic of the selected cells and expanding the logic from the endpoint pin back to the startpoint.

Figure 178: **cnt Counter in the Elaborated View**



The elaborated view shows that the adder input has been sized to 32 bits because the signal `cnt` is declared as an integer. In this case, the 32-bit operator is retained throughout synthesis optimizations. The elaborated view reveals this behavior, which indicates a possible RTL improvement.

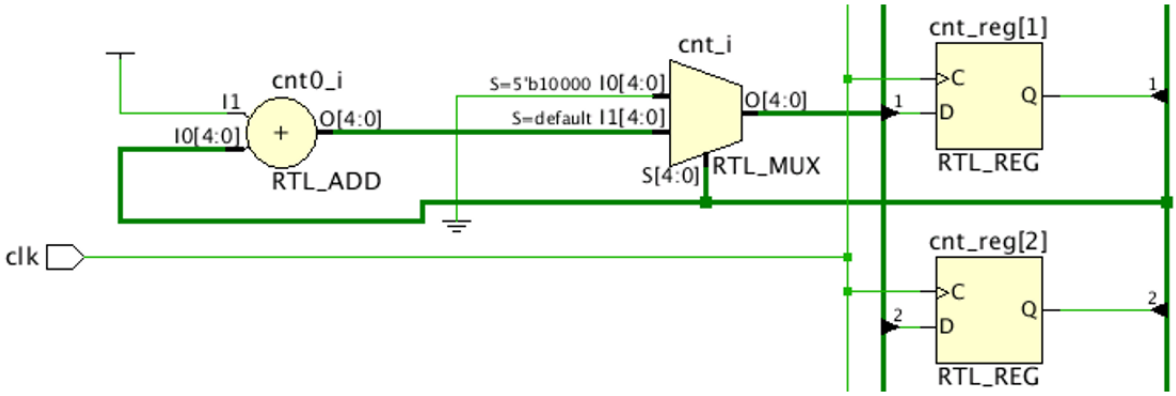
Because the counter increments from 0 to 16, you can define a range for the `cnt` signal. This change forces the adder inputs to be 5 bits wide instead of 32 bits wide.

Figure 179: **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;
```

This RTL change impacts synthesis optimization, which you can verify in the elaborated view without going through the entire compilation flow.

Figure 180: **cnt Counter in the Elaborated View after RTL Improvement**

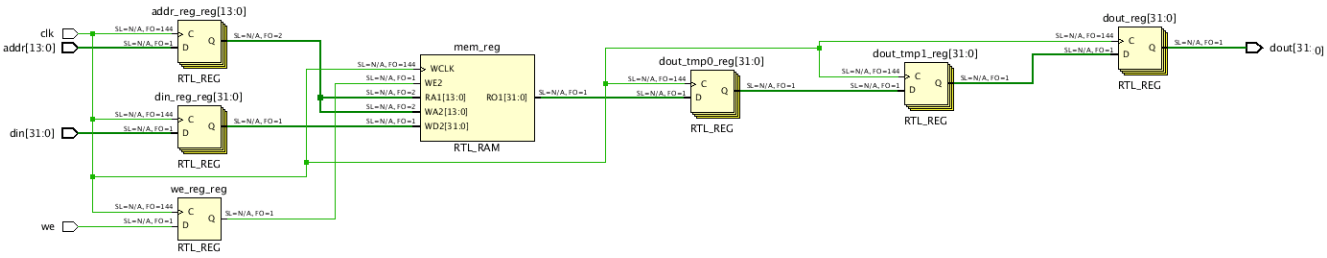


Decomposing Deep Memory Configurations for Balanced Power and Performance

In deep memory configurations, the synthesis attribute `RAM_DECOMP` can be applied in the RTL to improve memory decomposition and reduce power consumption. When applied, the memory is configured in a wider arrangement of primitives instead of a deep and narrow configuration.

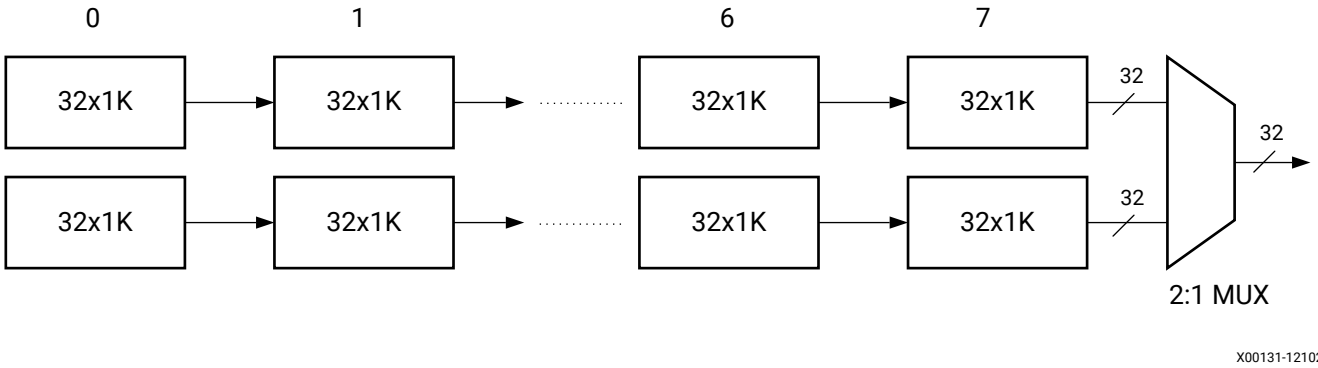
When the `CASCADE_HEIGHT` attribute is used together with `RAM_DECOMP`, synthesis inference gains more granular control over cascading. This provides balanced power and performance. While this approach requires additional address decoding logic, it reduces the number of block RAMs accessed at a time, helping to lower power consumption.

Figure 181: **32 × 16K Memory Configuration**



For example, applying `RAM_DECOMP` = power and `CASCADE_HEIGHT` = 4 infers 16 RAMB36E2 blocks and decomposes the memory as shown below.

Figure 182: **Generated Structure for 32 × 16K Memory Configuration Using RAM_DECOMP and CASCADE_HEIGHT Attributes**



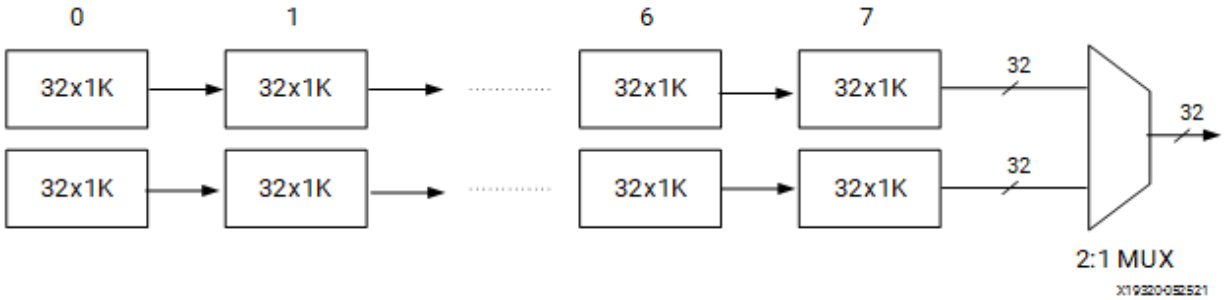
The base primitive in this configuration is 32 × 1K. Four block RAMs are cascaded to form a 32 × 4K configuration. Four such parallel structures create a 16K-deep memory, with outputs multiplexed to generate the output data.

Figure 183: **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, 16 RAMB36E2 blocks are still inferred, but the decomposition changes as shown in the following figure.

Figure 184: **Generated Structure for 32 × 16K Memory Configuration using `RAM_DECOMP` Attribute**



In this case, the base primitive is 32 × 1K, with eight block RAMs cascaded to form a 32 × 8K configuration. Two such parallel structures create a 16K-deep memory, with outputs multiplexed through a 2:1 MUX.

Figure 185: **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
```

Power savings are similar in both configurations (Figure 182 and Figure 184), because only one block RAM is active at a time. However, performance differs: a four-level deep cascaded block RAM chain (Figure 182) provides better performance than an eight-level deep cascaded block RAM chain (Figure 184).

Optimizing RAMB Utilization when Memory Depth is not a Power of 2

The following test case can be used to review the synthesis log file and determine whether the RTL can be improved to guide the tool toward a more optimal implementation.

Figure 186: **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;
```

This example describes a 40K-deep, 36-bit-wide memory in VHDL, requiring a 16-bit address bus.

Using the `report_utilization` command post-synthesis shows that the synthesis tool generates 72 block RAMs.

Figure 187: **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

Manual calculation reveals that fewer block RAMs are actually needed:

- Break the memory into two parts: 32K × 36 and 8K × 36.
- Use an address decoder based on the MSB address bits to enable one memory or the other for read/write operations and select the proper output data.
- The 32K × 36 memory can be implemented with 32 RAMBs: 4 × 8 × (4K × 9).
- The 8K × 36 memory can be implemented with 8 RAMBs: 8 × (1K × 36).
- The total optimal number of RAMBs required is 40.

The synthesis log file includes details on how each memory is configured and mapped to FPGA primitives. In this case, the memory depth is treated as 64K, indicating that non–power-of-two depths are not handled optimally.

Figure 188: **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 uses 64K × 1 (two block RAMs with cascade feature), repeated 36 times for the 36-bit data width. This results in 36 × 2 = 72 block RAMs.

Applying the revised RTL forces synthesis to infer the optimal number of RAMBs.

Figure 189: **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

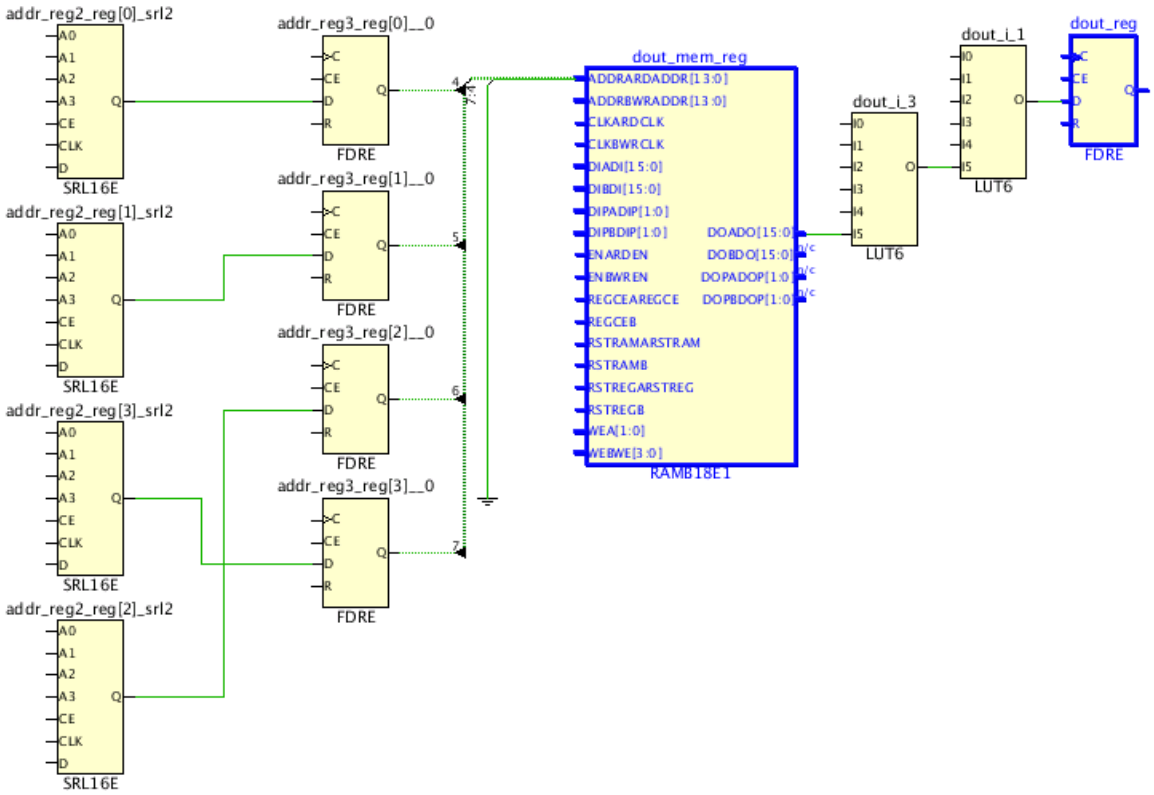
The following RTL code snippet generates a critical path from a block RAM (in this case, a ROM) with multiple logic levels ending at a flip-flop (FF). The RAMB cell is inferred without the optional output registers (DOA-0), adding more than 1 ns of extra delay to the RAMB output path.

Figure 190: **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 this RTL code is shown in the next figure.

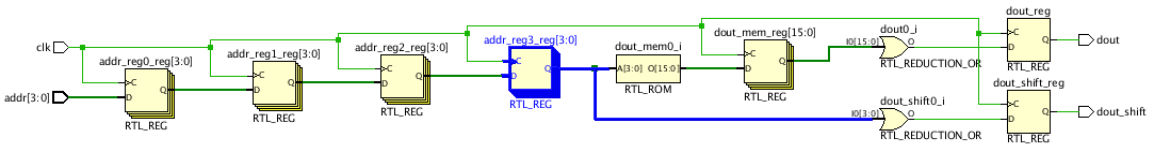
Figure 191: **Critical Path from RAMB Without Output Register Enabled**



It is good practice to review critical paths after synthesis and after each implementation step to identify which groups of logic require improvement. For long paths, or paths that do not take optimal advantage of FPGA hardware features, examine the RTL description to understand why the synthesized logic is not optimal and modify the code to help the synthesis tool improve the netlist.

Vivado provides an embedded debugging mechanism through the elaborated view, which helps identify where inefficiencies occur without manually searching through the RTL code.

Figure 192: **Elaborated View of RTL Code Snippet**



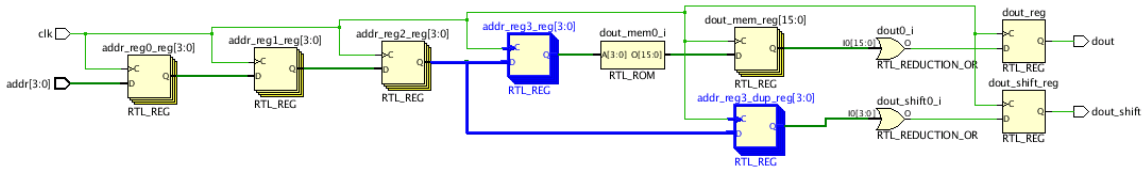
In this case, the problem arises from the address register fanout (`addr_reg3_reg`), which drives both the memory address and some interconnect logic (highlighted in blue). RAMB inference by the synthesis tool requires a dedicated address register in the RTL code, which is incompatible with the current address register fanout. As a result, the synthesis tool re-times the output register to allow 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 interconnect logic are driven by separate registers, the RAMB can be inferred with the output registers enabled.

Figure 193: **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 194: **Elaborated View of the Replicated Address Register**

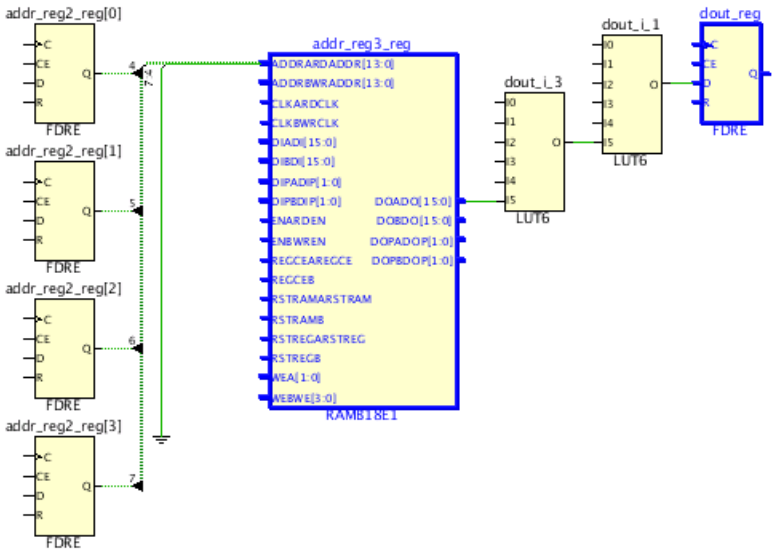


The critical path for the modified RTL code shows the improvement:

- 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, significantly reducing the datapath delay on the RAMB outputs.

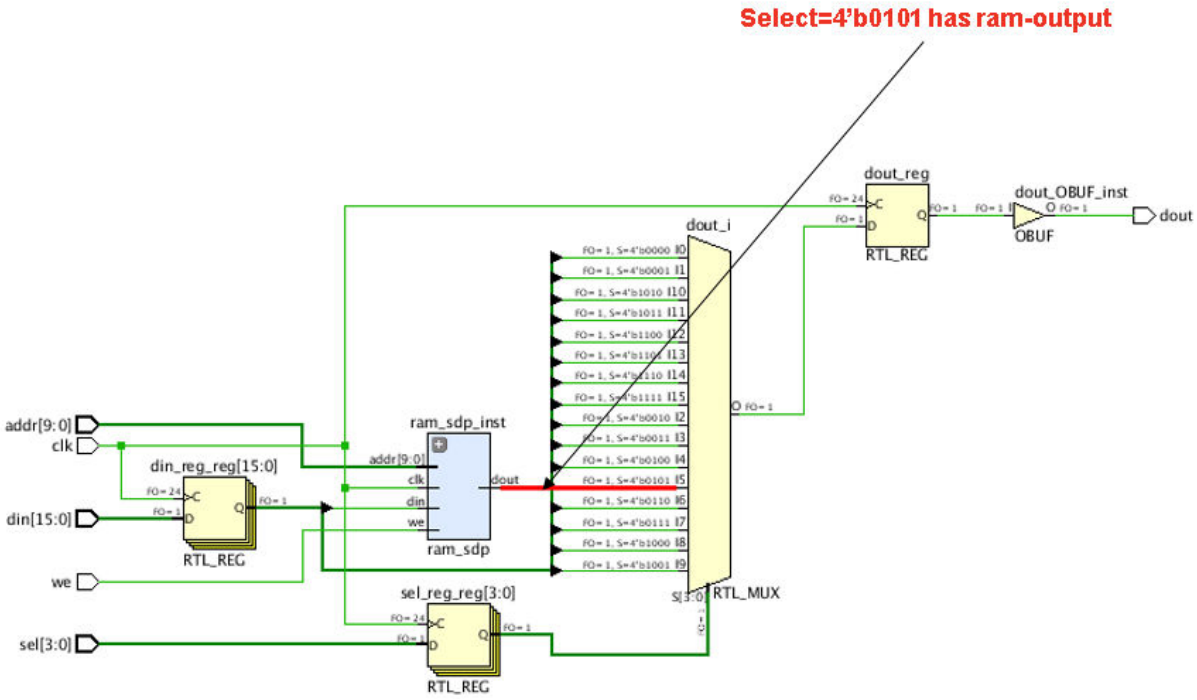
Figure 195: **Critical Path for the Modified RTL Code**



Improving Critical Logic on RAMB Outputs

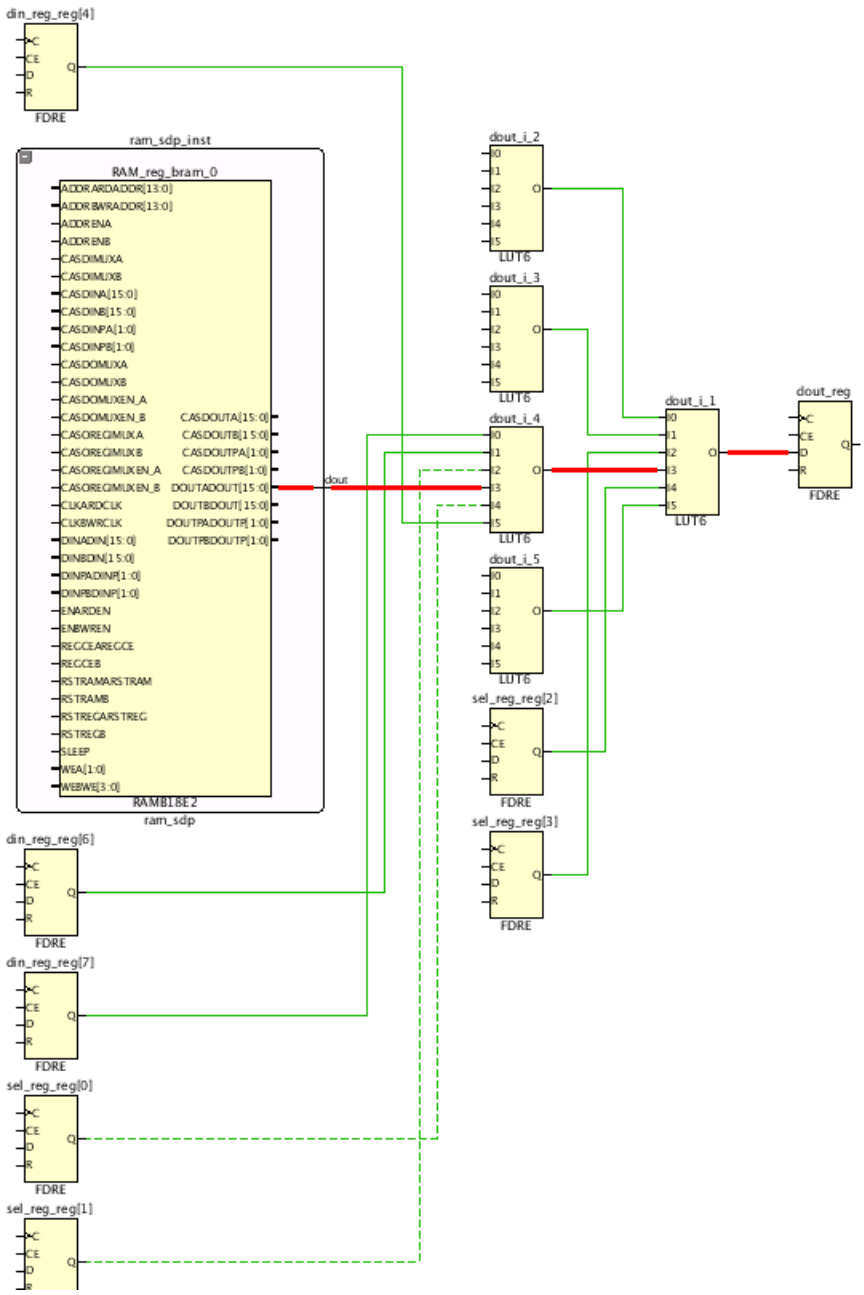
This test case demonstrates how to improve critical paths through logic restructuring, such as pushing a macro (block RAM) closer to the destination register.

Figure 196: **16x1 Multiplexer Connected to Block RAM Outputs**



In this example, a 16×1 multiplexer has only one input from a block RAM; the remaining inputs are fed by registers. The critical path is block RAM → 2 logic levels → flip-flop (FF).

Figure 197: **Critical RAMB-LUT-FF Path**



The figure shows the critical path in red, from block RAM to FF. The path has two logic levels from block RAM to FF, as well as FF to FF. Because the block RAM CLK→Q delay is higher than for registers, the block RAM to FF path is critical.

Figure 198: **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 logic can be restructured for better results. In this example, the original code is rewritten by breaking the 16×1 multiplexer into two multiplexers. The condition for select value `4'd5` is exempted and used as an enabling condition for a 2×1 multiplexer.

Figure 199: **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
```

This cascade multiplexer structure results in FF→FF with three logic levels, but reduces the block RAM to FF path to a single logic level. This improves the block RAM to FF path and helps downstream tools achieve better placement, because RAMB placement is more challenging than LUT or FF placement. In general, reducing the number of long paths around macro primitives such as RAMB, UltraRAM, and DSP yields better QoR.

Chapter 7

Implementation Analysis and Closure Techniques

This chapter explains timing closure techniques that you can use in addition to those described in the *UltraFast Design Methodology Timing Closure Quick Reference Guide* (UG1292) and *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

- Use intelligent design runs to automate the timing closure flow. This approach solves complex timing closure issues and requires minimal user knowledge.
- Apply the QoR suggestion object flow to automatically enhance QoR by setting properties.
- Use the ML strategy flow to help you select optimal tool options for your design.
- Apply floorplanning to guide the placer. This advanced technique can improve timing paths and reduce congestion.
- Check if your design has issues meeting hold time requirements, as this can be a key factor in choosing your timing closure strategy.

Intelligent Design Runs

An intelligent design run (IDR) is a specialized implementation run that uses a complex flow to attempt timing closure. Because an IDR can be aggressive, expect a compile time about 3.5 times longer than a standard run.

The IDR provides a simple interface for complex timing closure features and can achieve results comparable to those of FPGA experts for a high percentage of designs.

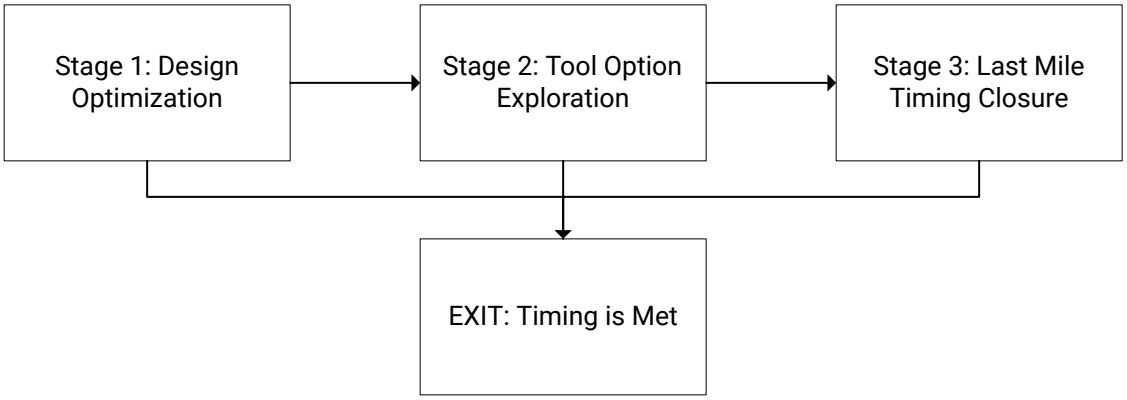
Note: Intelligent Design Runs are not available for AMD Versal™ architectures. As an alternative, use the instructions found in the Automatically Generating and Applying Suggestions section to automatically generate and apply suggestions.

Overview

The IDR is an aggressive timing closure implementation run focused solely on meeting timing. Power and compile time are not priorities, although some power optimizations can occur if utilization savings are achieved.

The IDR process is divided into three stages, as shown in the simplified diagram:

Figure 200: **IDR Overview**



The flow is fully automated, and you cannot control which stages run. Before using IDR, make sure your design is free of methodology issues. Run `report_methodology` and fix or waive all critical warnings and warnings.

Stage Details

- **Stage 1: Design Optimization:** In this stage, the tool generates and applies QoR suggestions. Compile time is typically up to two and a half times longer than a standard implementation run because:
  - The implementation tools must run to post-place or post-route to produce accurate analysis data.
  - To apply the suggestions, the design run must be reset and the implementation tools rerun.
  - By applying suggestions before re-analysis, you avoid overestimating the impact of design issues, which maximizes the QoR improvement.
- **Stage 2: Tool Option Exploration:** This stage uses ML strategies to predict the best tool options for the design.
- **Stage 3: Last Mile Timing Closure:** This stage applies post-route `phys_opt_design`, incremental implementation with a Last Mile directive, and incremental QoR suggestions to close timing.

  To enter this stage, the design must meet both of the following:

  - An RQA score of 3 or higher.
  - WNS between -0.250 and 0.000.

  If these criteria are not met, the stage is skipped and the flow exits.

Exit Conditions

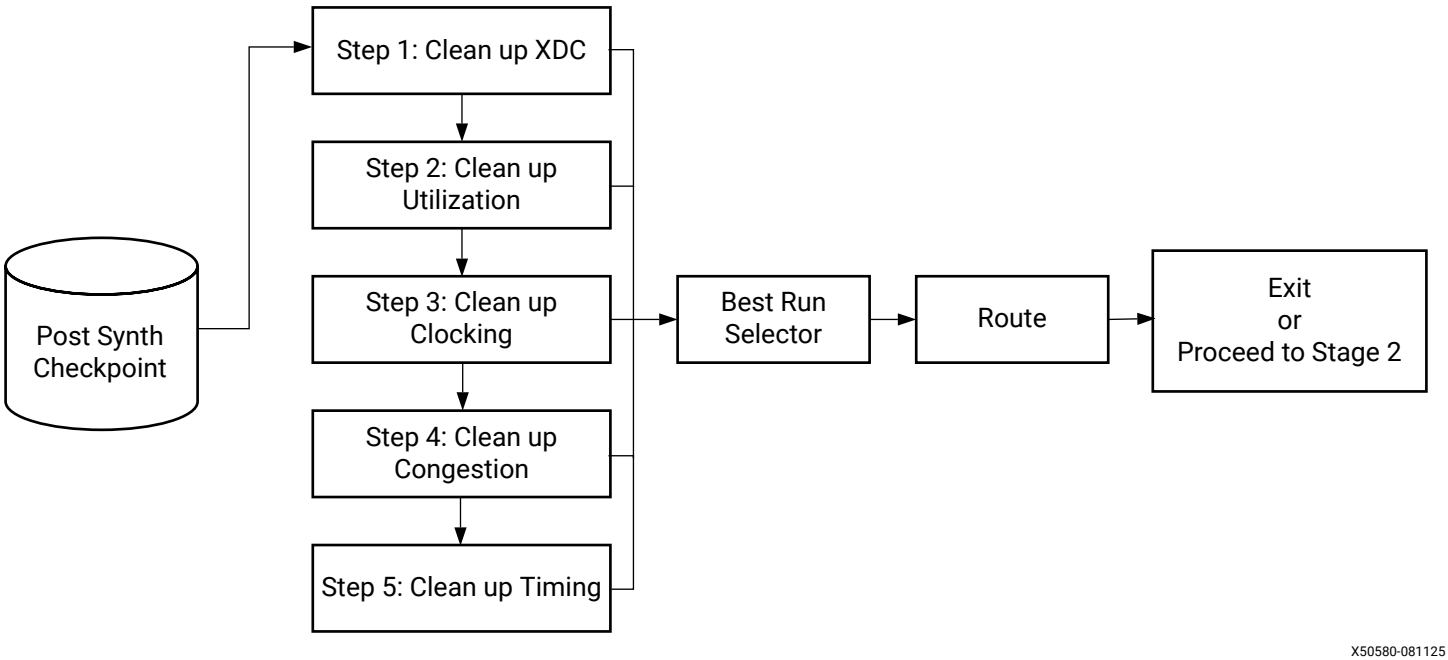
The flow can exit under the following conditions:

- 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 exit-on-failed-methodology-checks option is enabled and there are failed methodology checks with IDs TIMING-6, TIMING-7, TIMING-8, or TIMING-13.
  - The design fails to route.
  - No ML strategies are predicted.
- At stage 2, if the Last Mile entry criteria are not met.
- At the end of stage 3, if the Last Mile algorithms were used and no further improvements are possible.

Stage 1: Design Optimization

The Design Optimization stage runs in sequential steps, as shown in the following diagram.

Figure 201: **Design Optimization Steps**



In each step, multiple implementation commands such as `opt_design`, `place_design`, and `route_design` can run, and QoR suggestions can be generated. Each step has a target suggestion list. If any generated suggestions appear on this list, the design is reset to the required design stage so the suggestions can be successfully applied. If there are no suggestions on the target list for a step, that step is skipped.

The details of design optimization steps are as follows.

- **Clean Up XDC:** Check the design for issues that cause implementation errors or timing failures that cannot be fixed. If an error is found, the flow exits. No suggestions are generated or applied in this step.
- **Clean Up Utilization:** Look for suggestions that reduce utilization without impacting timing. Other non-utilization suggestions might also be applied if they can be detected and fixed early.
- **Clean Up Clocking:** Run the design to `place_design` to generate accurate clock skew timing values. If clocking suggestions exist, reset the flow.

  Note: If there are no suggestions in the Clean Up Utilization and Clean Up Clocking steps, the flow reports a special `First Pass` stage. This serves as a baseline reference for later stages. To save compile time, this stage is skipped if suggestions are found.

- **Clean Up Congestion:** Run a limited portion of the router to identify congestion in the design, then generate congestion-related suggestions. Apply these suggestions if they exist.

  Note: The log file might show that `route_design` failed during congestion analysis. This is expected because a full route is not intended at this point.

- **Clean Up Timing:** Generate QoR suggestions based on failing timing paths from the preceding phase's checkpoint, then rerun the placer.

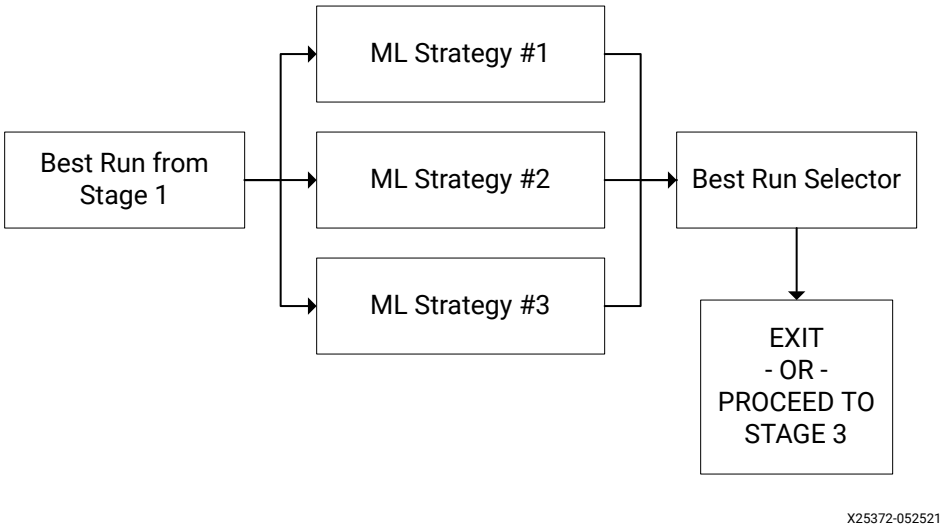
At the end of the Design Optimization stage, the tool decides whether to exit or to advance the best run to Stage 2: Tool Option Exploration.

To review the modifications made during the IDR, use the QoR suggestion report to see which suggestions were `GENERATED` and `APPLIED` at each step. The design run directory also contains multiple checkpoints that you can access for further analysis.

Stage 2: Tool Option Exploration

The goal of the Tool Option Exploration stage is to achieve the maximum QoR improvement from tool options. To do this, the flow runs three implementation jobs using tool options predicted by ML strategies, as shown in the following diagram:

Figure 202: **Tool Exploration Implementation Runs**



QoR suggestions are carried forward from the best run in Stage 1. If later steps in Stage 1 caused QoR degradation, their suggestions are dropped. Running three ML strategy runs (when three or more strategies are available) helps smooth out QoR fluctuations from any single run.

This phase uses standard implementation runs that you can run in parallel if your compute resources allow. When run in parallel, the compile time is approximately equal to one implementation run.

When the implementation runs finish, the best run selector chooses which run to take forward. The flow behaves 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.

***Note*:** To run Stage 2 in parallel using multiple remote hosts refer to section Appendix A of the *Vivado Design Suite User Guide: Implementation* (UG904).

Stage 3: Last Mile Timing Closure

The last mile timing closure stage takes the best implementation run from either of the previous two stages and attempts to close timing. Although QoR gains in this phase can be small compared to compile time, this step is still important for meeting timing. The stage only runs if the design meets the last mile timing closure entry requirements.

The last mile directive continues from an existing routed checkpoint and focuses on fixing failing paths. Timing closure is achieved in about 20 percent of designs with a WNS below –0.100 ns.

The goal of this stage is to close timing, which differs from the default tool flow that focuses on achieving the best WNS possible along with timing-closed WHS. The algorithms in this stage aim to improve timing without significantly altering the place and route results.

To achieve this, the flow applies the last mile incremental directive and QoR suggestions:

- Suggestions with the APPLIED property are reused from the reference run.
- Suggestions with the INCREMENTAL_FRIENDLY property are applied.

After routing completes, `phys_opt_design` can run to further improve timing.

To enter the last mile timing closure stage from stage 1 or 2, the design must have the following:

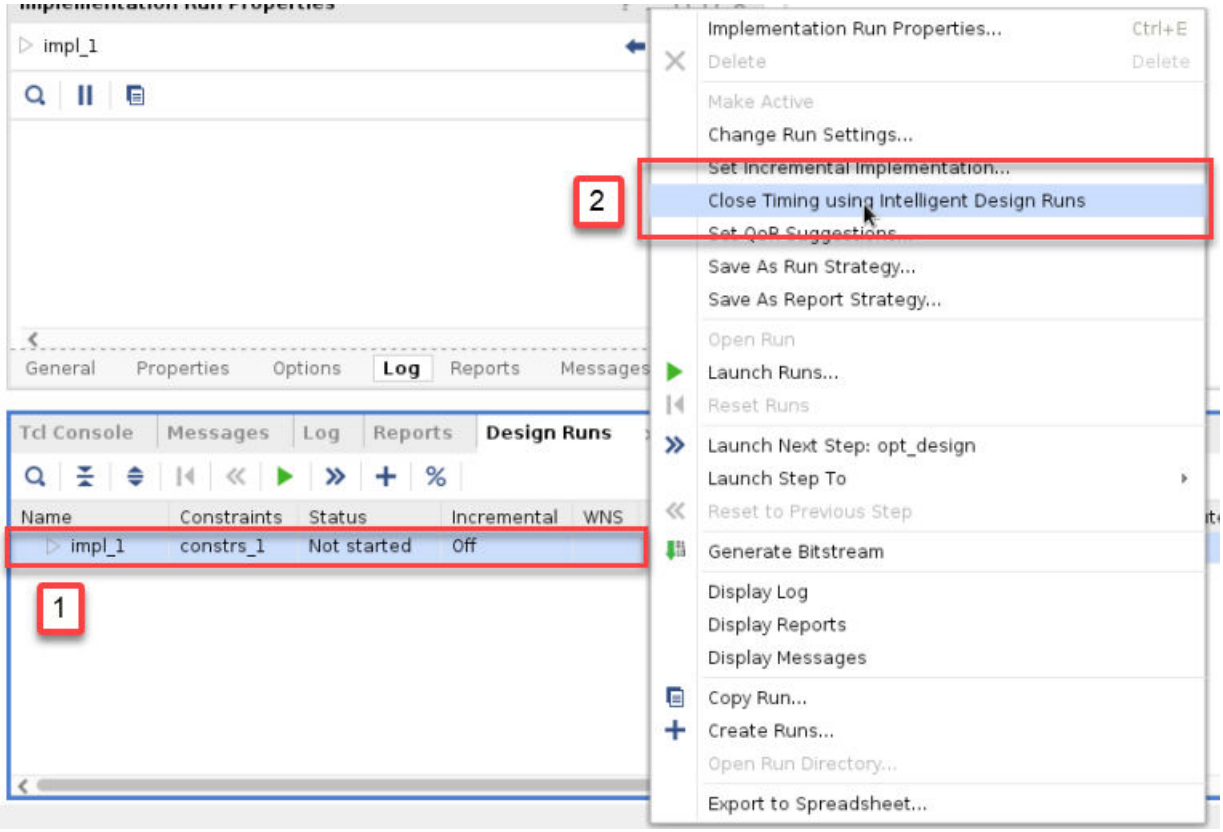
- A fully routed run
- A WNS greater than –0.250
- An RQA score of 3 or greater

Creating an Intelligent Design Run

The intelligent design run (IDR) is created from a standard implementation run.

1. In the Design Runs window, right-click the implementation run.
2. Select **Close Timing using Intelligent Design Runs**.

Figure 203: **Close Timing Using Intelligent Design Runs Command**



The equivalent Tcl command options 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>]
```

You can create only one IDR run from a given implementation run. To create another, delete the first or create a second implementation run and generate the IDR from that run.

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 executes 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.

Use the following steps to add a Tcl hook to an IDR:

1. Create an implementation run.
2. Add the Tcl hook.
3. 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.

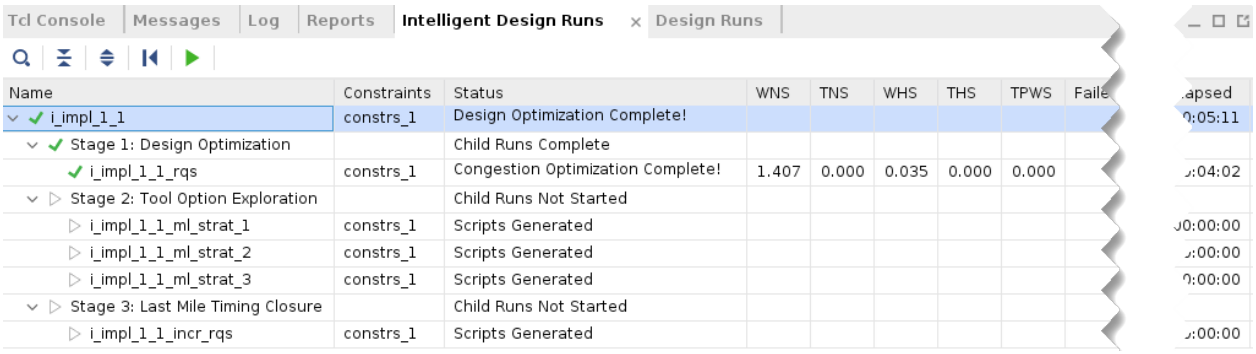
***Note*:** It is not possible to create an IDR run directly from a synthesis run.

Flow Control within Intelligent Design Runs Window

The Intelligent Design Runs window, shown in the following figure, provides two functions:

- A context-sensitive menu for accessing 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

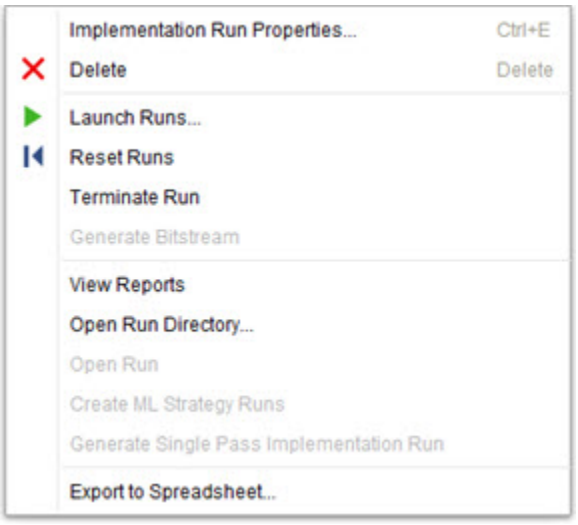
Figure 204: **Intelligent Design Runs Tab**



The metrics display the best run results for that IDR stage. They update regularly during the flow.

The right-click menu is context-sensitive and targets the selected flow stage. Right-clicking at the top level provides a superset of the options available at lower stages.

Figure 205: **Flow Control and Design Analysis Options Menu**



The menu options include:

- **Implementation Run Properties:** Opens the run properties for the IDR run. These properties are reduced compared to a normal run.
- **Delete:** Removes the run.
- **Launch Runs:** Starts the runs.
- **Reset Runs:** Resets the IDR and deletes all files.
- **Terminate Run:** Stops the selected run without deleting 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 no bitstream is generated, this option creates one from the best 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:** Accesses the run directory to view intermediate checkpoints and text reports.
- **Open Run:** Opens the selected run or the best run from the selected stage for design analysis. Available only for routed checkpoints.
- **Create ML Strategy Runs:** Available after stage 1 completes when the design still fails timing. This option automatically creates ML strategies with any applied QoR suggestions and generates three runs equivalent to stage 2. For changing designs, this works better than a single pass run with Last Mile because it can accommodate larger changes.

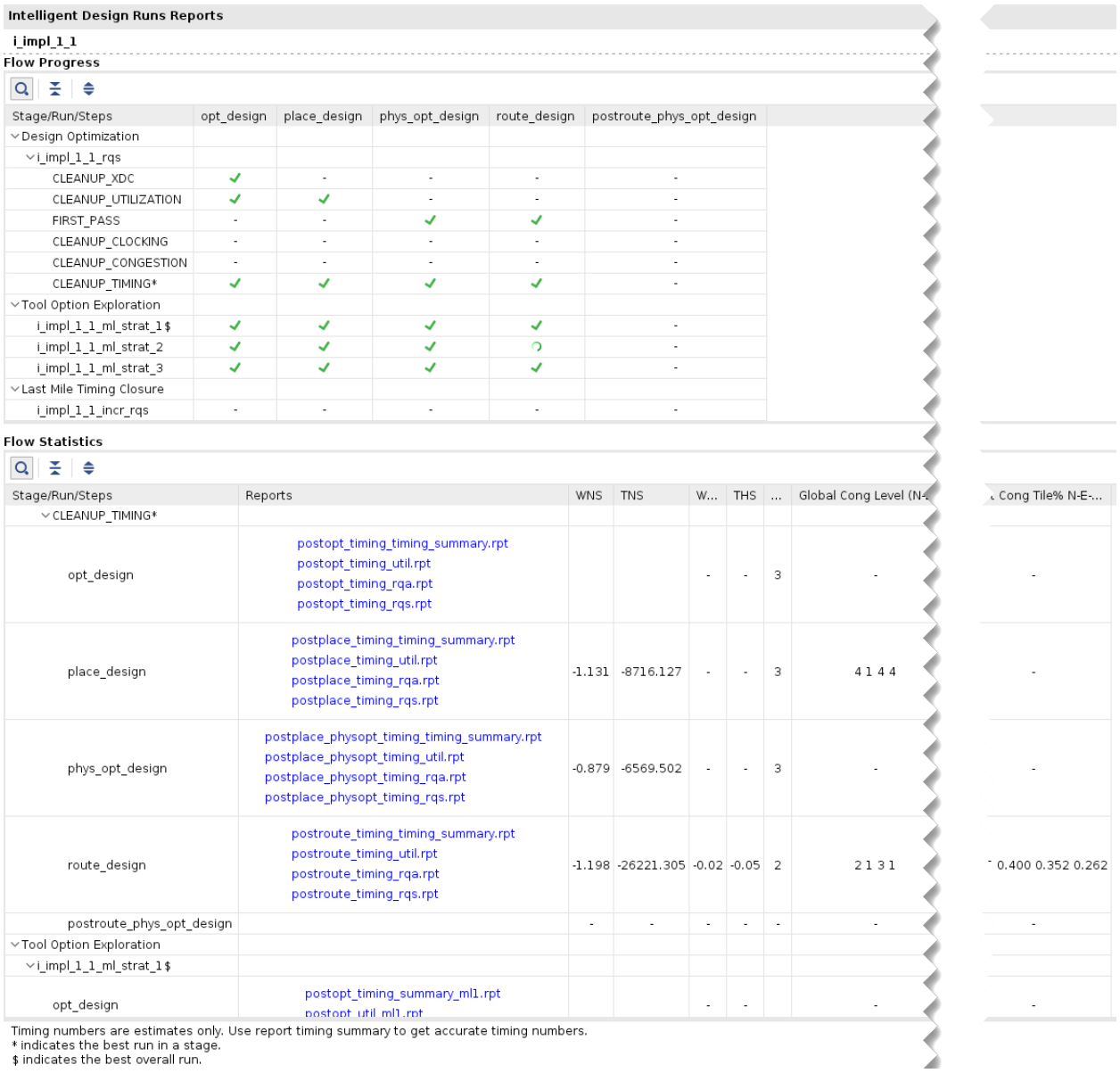
- **Generate Single Pass Implementation Run:** Creates a standard implementation run that sets up the RQS file and incremental checkpoints (if required) to match the IDR results.Available only after a successful IDR completion.

Intelligent Design Run Reports

The Intelligent Design Run Reports window, shown in the following figure, is divided into two sections:

- **Flow Progress:**
  - Shows which stages have completed and which stages are currently running.
  - Marks the best overall run with a dollar sign ($) and the best run within a stage with an asterisk (*).
- **Flow Statistics:**
  - Displays collected design statistics for timing and congestion.
  - Provides hyperlinks to the reports generated in the IDR.

Figure 206: **Intelligent Design Run Reports Window**



Intelligent Design Runs Reports

i_impl_1_1

Flow Progress

Stage/Run/Steps	opt_design	place_design	phys_opt_design	route_design	postroute_phys_opt_design
Design Optimization					
i_impl_1_1_rqs					
CLEANUP_XDC	✓	-	-	-	-
CLEANUP_UTILIZATION	✓	✓	-	-	-
FIRST_PASS	-	-	✓	✓	-
CLEANUP_CLOCKING	-	-	-	-	-
CLEANUP_CONGESTION	-	-	-	-	-
CLEANUP_TIMING*	✓	✓	✓	✓	-
Tool Option Exploration					
i_impl_1_1_ml_strat_1$	✓	✓	✓	✓	-
i_impl_1_1_ml_strat_2	✓	✓	✓	◌	-
i_impl_1_1_ml_strat_3	✓	✓	✓	✓	-
Last Mile Timing Closure					
i_impl_1_1_incr_rqs	-	-	-	-	-

Flow Statistics

Stage/Run/Steps	Reports	WNS	TNS	W...	THS	...	Global Cong Level (N-	. Cong Tile% N-E-...
CLEANUP_TIMING*								
opt_design	postopt_timing_timing_summary.rpt postopt_timing_util.rpt postopt_timing_rqa.rpt postopt_timing_rqs.rpt			-	-	3	-	-
place_design	postplace_timing_timing_summary.rpt postplace_timing_util.rpt postplace_timing_rqa.rpt postplace_timing_rqs.rpt	-1.131	-8716.127	-	-	3	4 1 4 4	-
phys_opt_design	postplace_physopt_timing_timing_summary.rpt postplace_physopt_timing_util.rpt postplace_physopt_timing_rqa.rpt postplace_physopt_timing_rqs.rpt	-0.879	-6569.502	-	-	3	-	-
route_design	postroute_timing_timing_summary.rpt postroute_timing_util.rpt postroute_timing_rqa.rpt postroute_timing_rqs.rpt	-1.198	-26221.305	-0.02	-0.05	2	2 1 3 1	0.400 0.352 0.262
postroute_phys_opt_design		-	-	-	-	-	-	-
Tool Option Exploration								
i_impl_1_1_ml_strat_1$								
opt_design	postopt_timing_summary_ml1.rpt postopt_util_ml1.rpt			-	-		-	-

Timing numbers are estimates only. Use report timing summary to get accurate timing numbers.
* indicates the best run in a stage.
$ indicates the best overall run.

The IDR captures the following data:

- RQA score, collected throughout the run
- 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 exists for a metric, the value is shown as a hyphen (-).

Note: Timing numbers collected in the IDR are typically estimates and might differ slightly from results generated by `report_timing_summary`.

The generated reports are fixed and cannot be directly controlled by the user. To add extra reports, use Tcl hooks or generate them manually after opening checkpoints in the run directory. The equivalent text report is automatically created in the top-level run directory with the fixed name `idr_flow_summary.rpt`.

Intelligent Design Runs Next Steps

Because intelligent design runs take a long time to complete, it is not practical to run them continuously after every design change. When an IDR completes, you need to incorporate its results into a main project run.

Incorporating IDR Results into a Main Project Run

1. Right-click the top-level IDR.
2. Choose one of the following:
   - **Generate Single Pass Implementation Run** to create an equivalent run to the IDR that replicates the best run results with greater compile-time efficiency. The structure of this run is dynamic and depends on:
     - QoR suggestions that improved the design
     - Implementation command directives from either the Stage 1 implementation or the Stage 2 ML-predicted strategies, depending on which produced the best run
     - An additional Tcl procedure added after `route_design` if Stage 3 (Last Mile) produced the best results

     This run is most useful for reproducing results with the same input netlist, such as closing out a timing-closed build. It can perform poorly with design changes, and including Last Mile can increase compile time.
   - **Generate ML Strategy Runs** to create three implementation runs using the top three ML-predicted implementation strategies, including any QoR suggestions that improved the design. This approach is generally better for design iterations because it excludes Last Mile, which adds extra compile time.

If you later need to add Last Mile to an ML strategy run, you can use the Last Mile script on a fully routed DCP file.

Running the Last Mile Script

The Last Mile script is supported in AMD Vivado™ 2023.1 and later for Versal, AMD UltraScale+™, and AMD UltraScale™ devices. It runs intensive algorithms to close timing on a fully routed design and can be called manually or added as a `TCL_POST` hook after `phys_opt_design -directive Explore`.

1. Open a fully routed design in Vivado.

2. Confirm that the design meets the entry criteria:
   - QoR Assessment score of 3 or greater
   - WNS of at least –0.250
3. Generate QoR suggestions for the Last Mile flow.
4. Enable any incremental-friendly QoR suggestions found.
5. Run Last Mile optimization commands:
   - `phys_opt_design` with clock, retiming, and LUT optimizations
   - `place_design -directive LastMile`
   - Additional `phys_opt_design` and `route_design` commands with the Explore directive
6. Optionally write intermediate or final checkpoints and timing reports.

Tcl Implementation

```
package require Vivado 1.2022.1

namespace eval ::xilinx::designutils {
    namespace export last_mile
}

proc ::xilinx::designutils::last_mile {{write_intermediate_files 0}
{write_final_files 0}} {
# Vivado   : Supported 2023.1 and later
#
# Families : Versal, Ultrascale+, Ultrascale
#
# Summary  : Run last mile flow. This is equivalent to
#            running stage 3 in IDR.
#            The last mile flow will run intensive
#            algorithms trying to close out timing
#            on a fully routed design.
#            It is recommended to run post route
#            phys_opt_design -directive Explore before
#            running last mile.
#            A call to this script can be added as a
#            TCL_POST hook to post route
#            phys_opt_design -directive Explore.
#
#            The requirements to run this are:
#              i)   A fully routed open design
#              ii)  When called a check will be made to
#                   see if the QoR Assessment Score of 3
#                   or 4 is achieved, if not it will exit.
#              iii) A WNS >= -0.250
#
# Arguments:
#
# write_intermediate_files -
# When set to 1, writes out intermediate DCP and timing
# reports after:
#  i)   preplace phys_opt_design,
#  ii)  place_design
```

```
#  iii) postplace phys_opt_design
#
# write_final_files -
# Writes reports and DCPs after the final stage
# route_design
# when set to 1.
#  Project users:     It is recommended to leverage
#                     project infrastructure to write
#                     reports
#  Non-Project users: Recommended to add custom
#                     write_checkpoint and reporting
#                     commands after this script is
#                     executed.
#
# Return Value:
# 1 - last mile completed successfully
#
# Categories: xilinxtclstore, designutils
set top [get_property TOP [current_design]]

# Entry Checks
puts "-I: Checking last mile entry criteria is met..."
if {[llength [current_design -quiet]] == 0} {
    puts "-E: Last mile requires an open design that is\
    fully routed"
    return -code 2 "Error: Last mile requires an open\
    design that is fully routed"
}
if {[report_route_status -boolean_check ROUTED_FULLY] == 0} {
    puts "-E: Last mile requires a design that is fully\
    routed"
    return -code 2 "Error: Last mile requires a design\
    that is fully routed"
}
# actual value is 3
if {[set score [get_assessment_score]] < 3} {
    puts "-E: Last mile requires a design has an QoR\
    Assessment Score of 3 or greater.\
    Design Score: $score"
    return -code 2 "Error: Last mile requires a design\
    has an QoR Assessment Score of 3 or greater.\
    Design Score: $score"
}
# actual value is -0.250
if {[set slack [get_property SLACK [get_timing_paths -setup]]] < -0.250} {
    puts "-E: Last mile requires a design a WNS >= -0.250\
    Design Slack: $slack"
    return -code 2 "Error: Last mile requires a design a\
    WNS >= -0.250. Design Slack: $slack"
}
puts "-I: Passed last mile entry criteria checks"

# QoR Suggestion Generation
puts "-I: Generating QoR Suggestions for Last Mile"
report_qor_suggestions -file ${top}_qor_suggestions_lastmile.rpt
if {[llength [get_qor_suggestions -filter {INCR_FRIENDLY} -quiet ] ] > 0} {
    puts "-I: Enabling generated incremental friendly\
    QoR Suggestions..."
    write_qor_suggestions -of_objects \
    [get_qor_suggestions -filter {INCR_FRIENDLY}] \
    -force -file ${top}_qor_suggestions_lastmile.rqs
    set_property ENABLED 1 [get_qor_suggestions \
    -filter {INCR_FRIENDLY}]
```

```
} else {
    puts "-I: No QoR Suggestions found for last mile\
    flow. Flow will continue.."
}

# Tool Flow Commands
phys_opt_design -clock_opt -retime -lut_opt
if {$write_intermediate_files != 0} {
    report_timing_summary -delay_type min_max \
    -report_unconstrained -check_timing_verbose \
    -max_paths 10 \
    -input_pins -routable_nets \
    -rpx ${top}_timing_lastmile_preplace_physopt.rpx \
    -file ${top}_timing_lastmile_preplace_physopt.rpt
    write_checkpoint -force \
    ${top}_lastmile_preplace_physopt.dcp
}
place_design -directive LastMile
if {$write_intermediate_files != 0} {
    report_timing_summary -delay_type min_max \
    -report_unconstrained \
    -check_timing_verbose \
    -max_paths 10 \
    -input_pins \
    -routable_nets \
    -rpx ${top}_timing_lastmile_placed.rpx \
    -file ${top}_timing_lastmile_placed.rpt
    write_checkpoint -force \
    ${top}_lastmile_placed.dcp
}
phys_opt_design -directive Explore
if {$write_intermediate_files != 0} {
    report_timing_summary -delay_type min_max \
    -report_unconstrained \
    -check_timing_verbose \
    -max_paths 10 \
    -input_pins \
    -routable_nets \
    -rpx ${top}_timing_lastmile_postplace_physopt.rpx \
    -file ${top}_timing_lastmile_postplace_physopt.rpt
    write_checkpoint -force \
    ${top}_lastmile_postplace_physopt.dcp
}
route_design -directive Explore
if {$write_final_files != 0} {
    report_timing_summary -delay_type min_max \
    -report_unconstrained -check_timing_verbose \
    -max_paths 10 -input_pins -routable_nets \
    -rpx ${top}_timing_lastmile_routed.rpx \
    -file ${top}_timing_lastmile_routed.rpt
    write_checkpoint -force \
    ${top}_lastmile_routed.dcp
}
phys_opt_design -directive Explore
phys_opt_design -directive Explore
if {$write_final_files != 0} {
    report_timing_summary -delay_type min_max \
    -report_unconstrained -check_timing_verbose \
    -max_paths 10 -input_pins -routable_nets \
    -rpx ${top}_timing_lastmile_postroute_physopt.rpx \
    -file ${top}_timing_lastmile_postroute_physopt.rpt
    write_checkpoint -force \
```

```
    ${top}_lastmile_postroute_physopt.dcp
}

return 1
}
```

Intelligent Design Runs Recommendations for Non-Project Users

A Vivado project is required to use the IDR feature because of the run management it requires. The following instructions describe the easiest way to create a post-synthesis project. They apply to:

- Non-project implementation runs
- Synthesis using earlier versions of Vivado
- Synthesis using third-party tools

The simplest way to access the IDR feature is to add a single full-design DCP source to a post-synthesis project. This provides the complete netlist and all design constraints. After creating the project, you can launch the IDR by following the steps in the Creating an Intelligent Design Run section.

1. Generate a single checkpoint from your existing implementation run:
   a. Locate the `opt_design` call in the implementation run Tcl script
   b. Write a checkpoint before this stage, for example:

```
write_checkpoint -force <PreOptDesign>.dcp
opt_design -directive Explore ; ## FOR EXAMPLE ONLY ##
```

2. Create a post-synthesis project by using the New Project wizard or the following Tcl commands:

```
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 Using Project Mode in the *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 16: **IDR Support Summary**

Item	Support Status
Supported Families	UltraScale, UltraScale+
Flows	Project Mode
DFX	Yes
Vitis	No

QoR Suggestions

Quality of results (QoR) suggestions help improve the ability of a design to meet timing through tactics such as:

- Adding switches to commands such as `opt_design`
- Adding properties to design objects such as cells and nets
- Applying full implementation strategies

The `report_qor_suggestions` command generates a report in either the Vivado Integrated Design Environment (IDE) or as a text file. You can use it to:

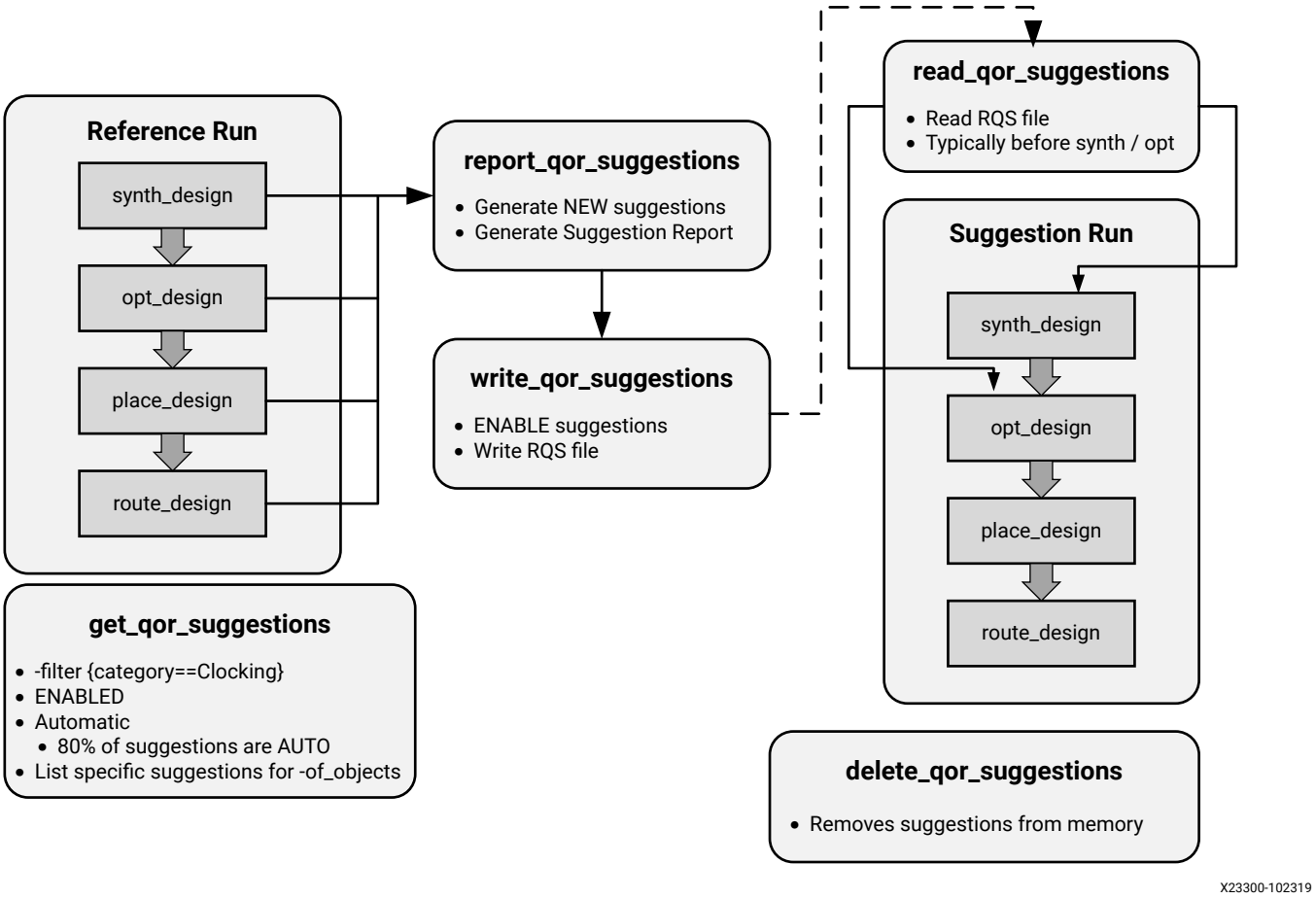
- Generate and view new suggestions for the current design in memory
- View existing suggestions that were read using the `read_qor_suggestions` command

You can run `report_qor_suggestions` on a design loaded in memory at any point after synthesis. The generated suggestion objects consider many design characteristics and can be grouped into the following categories:

- Clocking
- XDC
- Netlist
- Utilization
- Congestion
- Timing
- Strategies

The generated suggestions must be fed back into the flow to take effect, which usually requires rerunning specific design stages, as shown in the following figure:

Figure 207: **Suggestion Flow**



Before generating new suggestions, load a design into memory. `report_qor_suggestions` can be run at any stage of the flow after synthesis. The report lists suggestions in order of importance, with the most important at the top. Only suggestions that are necessary to improve QoR are included.

Some suggestions require placement or routing information before they can be issued. Additional rules control when suggestions are generated:

- Netlist suggestions are based on netlist analysis. They identify structures that often lead to timing failures later in the flow, without directly examining timing paths. These can be generated for designs that already meet timing.
- Clocking suggestions are usually generated after placement, though exceptions exist when accurate information is available earlier. Most require a failing timing path.
- Timing suggestions are based on analysis of the top 100 failing timing paths per clock group.
- Utilization suggestions are generated when a resource is overused and the change does not increase the use of a critical resource. These can be reported at any design stage.

- Congestion suggestions are reported only after placement. They are omitted from routed designs that already meet timing, as they are not affecting timing closure.
- Strategies are implementation strategies generated using machine learning algorithms that analyze multiple design characteristics. Their usage flow differs from the other categories and is explained in the Strategy Suggestions section.

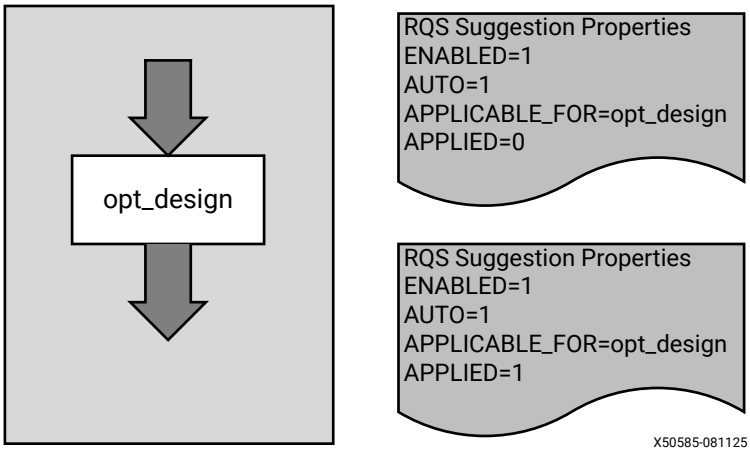
Executing Suggestions

Suggestions execute in the suggestion run when all of the following criteria are met::

- The suggestion is ENABLED.
- The APPLICABLE_FOR stage is run.
- The suggestion is set to AUTO.

When a suggestion executes, the APPLIED settings update, as shown in the following figure:

Figure 208: **Suggestion Execution**



In the implementation flow, if a property in a suggestion is not applied correctly to the associated cell or net, the FAILED_TO_APPLY value is set to 1. If a suggestion is partially applied, a new suggestion is generated, splitting it into two: one for the applied portion and one for the failed portion.

***Note*:**

- `FAILED_TO_APPLY` is not set if the implementation tools reject the property later in the flow.
- It is set only when the property cannot be applied to the object.
- The flow does not track the actual application of the property.

Suggestions can be executed in the same run in which they are generated if the APPLICABLE_FOR stage occurs after the stage where they were generated. To do this:

```
set_property ENABLED 1 [get_qor_suggestions <SuggID>]
```

When using this method, remember to write the suggestion to the RQS file when the run is complete so it can be used in future runs.

All Related Commands

The following commands are available when working with QoR suggestion objects:

Table 17: **All Related Commands**

Command	Function
`report_qor_suggestions`	Generates new suggestions and reports on existing suggestions.
`write_qor_suggestions`	Writes suggestion objects to a file and automatically enables them.
`read_qor_suggestions`	Reads suggestion objects from a file.
`get_qor_suggestions`	Returns QoR suggestion objects.
`delete_qor_suggestions`	Removes QoR suggestions from memory.
`get_qor_checks`	Returns a list of all the suggestions that can be generated by Vivado.

Managing the Suggestions

In project mode, several features simplify managing suggestion files and improve performance in common workflows. You can also apply these methods in a non-project flow.

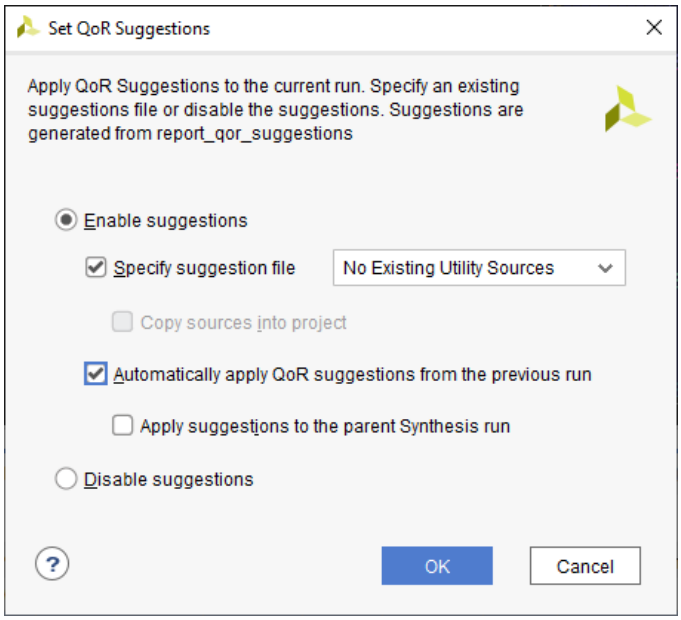
Project Mode

In project mode, after suggestions are written to an RQS file, the file is automatically added to the project's utility sources fileset (`utils_1`). Store the file outside the run directory, because it is deleted when the run is reset. Use a unique directory per run so multiple runs can each have their own suggestion file. The recommended location is: `<project_dir>/<project>.srcs/utils_1/<run_name>`.

To set QoR suggestions in the Vivado IDE:

1. In the Design Runs window, right-click a run and select **Set QoR Suggestions**.
2. Select **Enable Suggestions** to choose a suggestion file, the automatic run, or both.

Figure 209: **Set QoR Suggestions Window**



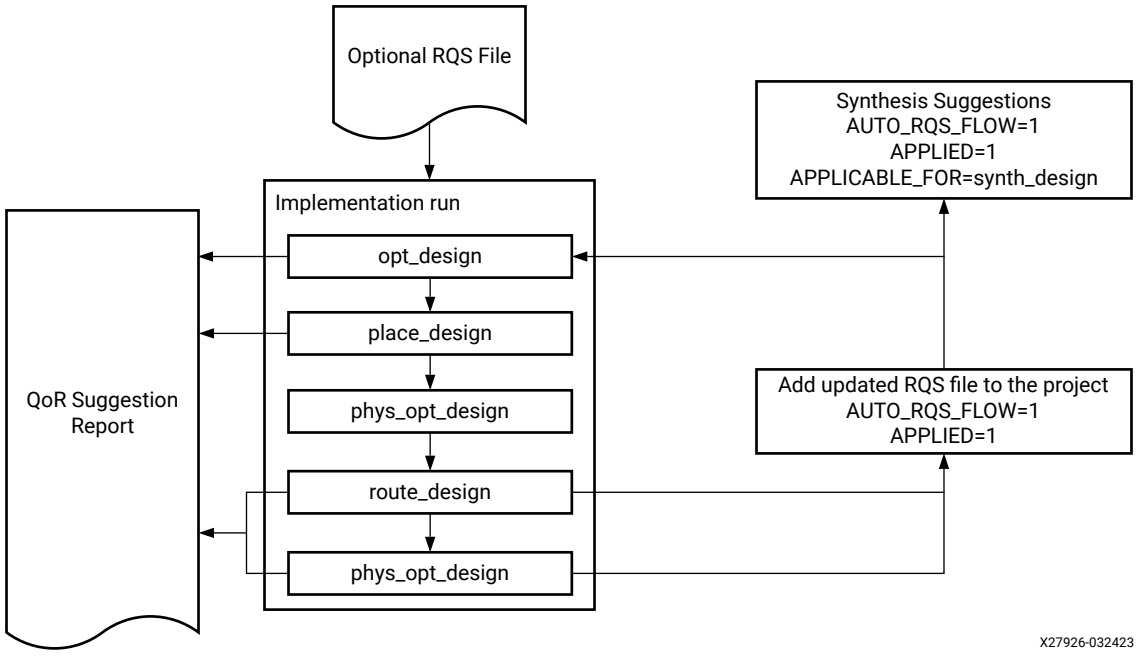
Equivalent Tcl Commands

```
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 simplest method for designs that change frequently. When enabled, the standard implementation flow is adjusted as shown in the following figure.

Figure 210: **Implementation Flow with Automatic QoR Suggestions**



You can start with either:

- A user-selected RQS file (remains in place until replaced by newly generated suggestions)
  - Any suggestions in the file are written into a `<top>_routed.rqs` file, removing the requirement to keep the original file.
- No RQS file, allowing the tool to generate one automatically

The `report_qor_suggestions` command runs after:

- `opt_design`
- `place_design`
- Either `route_design` or post-route `phys_opt_design` (if enabled)

To examine the suggestions, open the `report_qor_suggestions` report in the Reports tab at each point where the command runs.

Most suggestions are written to the RQS file at the end of the run so they can be applied in the next run. Suggestions included are:

- New suggestions with `AUTO_RQS_FLOW==1`
- Older suggestions with `APPLIED==1`

`RQS_CLOCK-1` and `RQS_CLOCK-26` suggestions can be generated at `opt_design` and applied in the same 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 changed.

Suggestions can also be applied to a parent synthesis run if one exists. Only one child implementation run can supply synthesis suggestions; if more than one is selected, the most recent is used.

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.

The following properties are used on the first class `run` object:

Table 18: **Auto QoR Suggestion Flow Run Properties**

Property	Values	Description
RQS_FILES	RQS file name	Allows input of a custom RQS file.
AUTO_RQS	• **1:** On • **0:** Off	Enables the automatic RQS flow.
AUTO_RQS.DIRECTORY	<directory>	Specifies an alternative location to copy the RQS file.
AUTO_RQS.SUGGESTION_RUN	Implementation run name that generates the RQS file	Generates the RQS file for the parent synthesis 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 starts, point to the RQS file using the `RQS_FILES` property. Strategy suggestions are added to the new RQS file at the end of the flow.

If ML strategies do not exist, they are generated automatically in the `<run_name>/MLStrategy` directory during suggestion generation.

***Note*:** The ML Strategy flow is supported only for UltraScale and UltraScale+ devices.

Automatic QoR Suggestions and the Incremental Flow

When the incremental flow is enabled, only new incremental-friendly suggestions can be applied in later steps. This limits the size of timing improvements but helps maintain stability.

Applying suggestions at `opt_design` before starting the incremental flow can allow larger-impact changes. However, some suggestions might need to be disabled if they cause timing degradation.

If the incremental flow switches to the default flow because of a poor-quality reference checkpoint, all suggestions can be applied.

***Note*:** The QoR Suggestion and Incremental flow is supported only for UltraScale and UltraScale+ devices.

Disabling Unwanted Suggestions

If you find any suggestion degrades results it is possible to disable it by adding the following to a Tcl script at the pre `opt_design` step:

```
set_property ENABLED 0 [get_qor_suggestions RQS_TIMING-1*]
```

Non-Project Mode

In non project mode, you can modify your run scripts to achieve the same results as in the project flow. The first `read_qor_suggestions` command can be before `synth_design` or `opt_design`. The following example uses `read_qor_suggestions` before `synth_design`. Also note the usage of the `AUTO_RQS` property on suggestions to filter out suggestions excluded from this flow.

The equivalent Tcl command options for this flow are as follows:

```
read_vhdl <some_file>.vhd
set rqs_file <rqs_file>.rqs
if {[file exists $rqs_file]} {read_qor_suggestions $rqs_file}
synth_design -top <top> -part <part>
opt_design
report_qor_suggestions -file opt_report_qor_suggestions.rpt
set_property ENABLED 1 [get_qor_suggestions -filter
{GENERATED_AT==opt_design && SUGGESTION_SOURCE=="Current Run" && ENABLED &&
APPLICABLE_FOR==place_design && AUTO_RQS_FLOW}]
place_design
route_design
report_qor_suggestions -file route_report_qor_suggestions.rpt
write_qor_suggestions -of_objects [get_qor_suggestion -filter {APPLIED ||
(AUTO_RQS_FLOW && APPLIED==0)}] -file postroute.rqs
```

***Note*:** The `report_qor_suggestions` command requires a loaded design.

RQS in the Incremental Flow

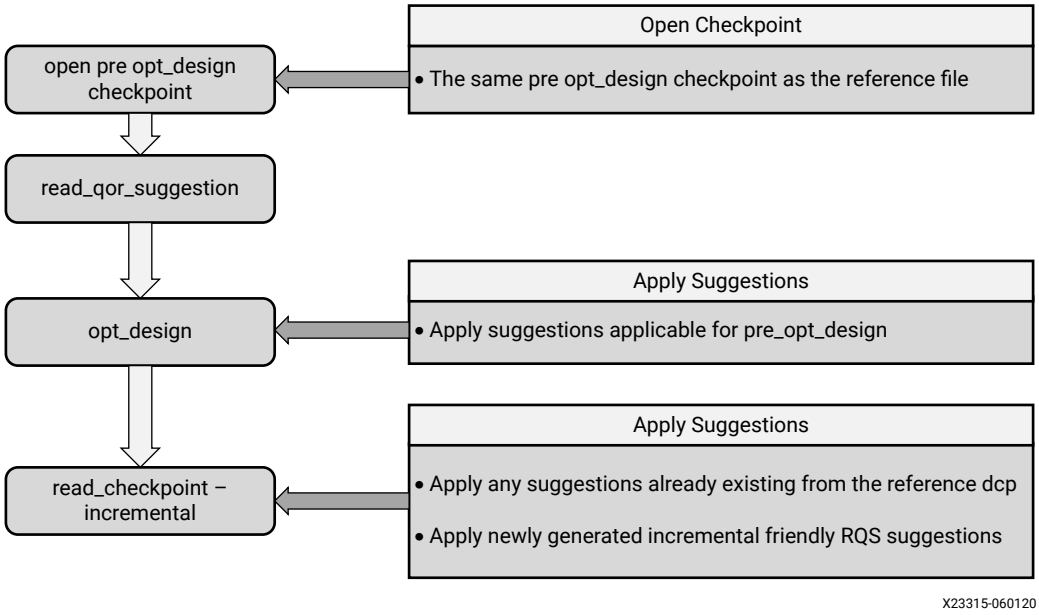
When your design is very close to timing closure, typically with a worst negative slack (WNS) less than –250 ps, enable the incremental flow with RQS suggestions. This approach helps achieve timing closure faster by combining the benefits of the incremental flow and RQS suggestions. The `report_qor_assessment` command indicates when to use this flow in the Flow Guidance section.

In this flow, the suggestions generated from the reference routed DCP are read in before running the incremental flow commands.

***Note*:** This flow is supported only for UltraScale and UltraScale+ devices.

Vivado automatically applies the rest of the flow. It determines which suggestions to apply at each stage by differentiating between suggestions that were newly GENERATED and those APPLIED in the reference run, as shown in the following figure:

Figure 211: **Incremental Flow**



When the incremental flow runs:

- Suggestions applied in the reference are read from the reference DCP and applied regardless of whether they are enabled. The `ENABLED` property is ignored to ensure the reference checkpoint is replicated as closely as possible.
- Incremental-friendly suggestions from the RQS file are applied. These must be enabled, which is done automatically during `read_qor_suggestions`.
- Suggestions are applied during `read_checkpoint -incremental` rather than at the `APPLICABLE_FOR` stages. Do not read or enable suggestions after this stage, as they are ignored.
- Any new non-incremental-friendly suggestions in the RQS file are ignored unless they already exist in the reference, in which case they are applied.

Take special care with suggestions applicable for `opt_design`. Because this stage occurs before the flow switches to incremental mode, it cannot be managed automatically. You must ensure that the same suggestions applied in the reference are also applied in the incremental run, and no new suggestions are introduced. If you want to use new `opt_design` suggestions, update the reference accordingly.

If the incremental flow reverts to the default flow, usually because of a negative quality change, all suggestions from the RQS file are executed. Before launching the next incremental run, export all suggestions to the RQS file, not only the incremental-friendly ones.

An example of the commands required to run the flow is shown here:

Prerequisites for Using This Flow

- The device part must match between the reference run and the incremental run.
- The reference checkpoint must be a post-route checkpoint.
- The same directive must be used for `opt_design` in both the reference and incremental runs.
- The design must not have major issues such as high congestion, unbalanced clocking, or an RQA score lower than 4.
- Regenerate suggestions from the reference checkpoint.
- Newly generated suggestions are applied only if they are incremental-friendly. Non-incremental-friendly suggestions are executed only if the flow reverts to default.
- All newly generated suggestions must be created from the reference checkpoint to ensure they do not affect resolved timing paths, such as those after `post-route phys_opt_design`.

Example Commands

Reference Run:

```
# 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 Run:

```
# 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 excessive accumulation of suggestions, the Vivado Design Suite automatically manages them. When new suggestions are generated, any suggestions that are identical to previously generated ones are deleted.

Viewing Suggestions in Tcl or Text Format

Suggestion objects are stored in binary form. The only way to read a suggestion is to load the design, read the suggestions, and run `report_qor_suggestions`.

If you prefer not to use the object flow, Vivado supports viewing and executing suggestions in Tcl.

To write out suggestions in Tcl, use:

```
write_qor_suggestions -tcl_output_dir <outputDir>
```

Running this command outputs one or more Tcl files to the specified directory. This option is not available in the Vivado IDE.

For Tcl scripts generated for implementation commands, manage their integration into the design constraints manually. For synthesis Tcl scripts:

- In project mode, add them to the constraints set (update the file filter to Tcl to see them).
- In non-project mode, access them using: `read_xdc -unmanaged <Tcl file>`

When suggestions are in Tcl format, you are responsible for maintaining the scripts by removing suggestions that are no longer required and applying only those you want. Append newly generated Tcl scripts to the existing ones.

***Note*:** Suggestions entered using Tcl are no longer reported by `report_qor_suggestions`.

Debugging QoR Suggestions

In some cases, it is necessary to examine a suggestion in detail to understand why it was generated. Common areas of investigation include:

- **Utilization and Netlist Suggestions:** Typically require examining the cells in the netlist targeted by the suggestion. This can be done by selecting the objects and opening a schematic.
- **Clocking and Timing Suggestions:** Typically require examining the timing path associated with the suggestion. The relevant timing reports are included in the QoR Suggestion report.
- **Congestion Suggestions:** Typically require examining the cells and their locations.

To obtain a list of targeted objects for a suggestion, write the suggestions to a Tcl file. Each suggestion in the file is labeled with its ID. From this, copy the `[get_cells <cells>]` or `[get_nets <nets>]` portion of the suggestion for further analysis.

Example Tcl command options to explore the objects:

```
set cells <cells from Tcl file>
report_timing -from $cells
report_design_analysis -of [get_timing_paths -from $cells -max_paths 100]
-name -RQS
show_objects -name RQS_objs $cells
select_objects [get_sites -of $cells]
```

Supported Families and Flows

The following table summarizes the supported device families and design flows for QoR suggestions in this release.

Table 19: **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

Vivado can generate more than 100 QoR suggestions, with over 80 capable of executing automatically without edits to constraints or RTL.

To view all possible suggestions, run the `get_qor_checks` command. The properties attached to the returned objects can be inspected. The objects returned by this command are not generated suggestions, and they cannot be written.

Use the `-family` switch to display only suggestions applicable to a specific family. For example, to list suggestions available for the family of the currently opened design:

```
get_qor_checks -family [get_property FAMILY [get_parts [get_property PART
[current_design]]]]
```

Use the `-filter` switch to filter based on object properties. Common properties to filter on include:

- `CATEGORY`
- `AUTO`

- INCR_FRIENDLY

Example to view the IDs and descriptions of all AUTO checks:

```
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

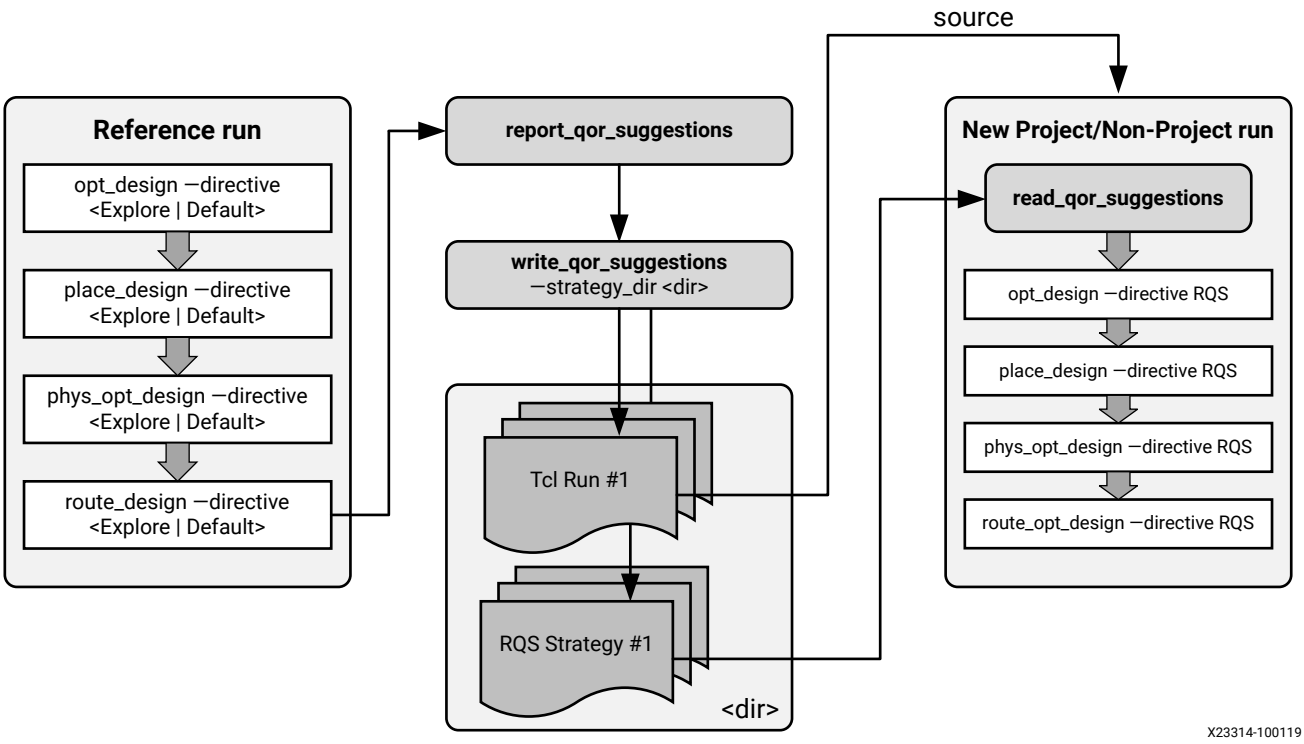
***Note*:** Strategy Suggestions are supported only for UltraScale and UltraScale+ architectures.

Typical timing closure strategies involve running multiple implementation strategies and selecting the best one for lab testing. Machine learning (ML) strategies provide an alternative by requiring only three strategies to achieve a similar QoR benefit. ML strategies analyze features from a post-route design to predict the performance of different strategies on the same design.

The best three strategies are stored in RQS files generated by report_qor_suggestions (and write_qor_suggestions) and can be applied in subsequent runs. This approach significantly reduces the server resources required.

When the directive is set to RQS on implementation commands, the tool references the RQS file for both the directive and other tool command options. The flow is shown in the following figure:

Figure 212: **Strategy Suggestion Flow**



Strategy Suggestions Flow

1. Generate strategy suggestions by running `report_qor_suggestions` on a fully routed design generated using either `Default` or `Explore` directives. For detailed requirements, see ML Strategy Availability.
2. Write the strategy RQS files by using the following command to generate the required RQS files in the specified directory: `write_qor_suggestions -strategy_dir <dir>`. By default, three strategies are generated. Each strategy RQS file contains all suggestion objects and the strategy suggestions object. The RQS file created using `write_qor_suggestions -file <fn>.rqs` can be discarded because the information is duplicated in the strategy RQS files.

   ***Note*:** To generate more strategies, increase the number with:

   ```
   report_qor_suggestions -max_strategies <n>
   ```

3. Load the strategy RQS file by reading the generated RQS file into the new implementation run.
4. Run the RQS strategy flow by setting the directive to RQS. Include calls to `opt_design`, `place_design`, `phys_opt_design` and `route_design` in the script.

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 that contain strategies and, optionally, other QoR suggestions.
2. Create the new implementation runs, read the QoR suggestion file, and set the directives to RQS.

Several methods in the Vivado IDE can accomplish step 1. Each method differs based on the flow used and the other QoR suggestions included in the RQS file.

Table 20: **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 process.	Individually selected suggestions are included.
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. 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, containing three RQS files. When these files are detected, the `create_rqs_runs` option becomes available in the Design Runs right-click menu.

Selecting this option automatically creates three new implementation runs and links them with the reference run. Only three ML strategies are allowed per reference. To recreate them, delete the existing strategies and rerun the process. Minor design changes do not require regeneration of the strategies.

The equivalent Tcl command option to create new ML strategy runs from an implementation run `impl_1` is as follows:

```
create_rqs_runs -reference_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 demonstrates how to read the RQS file and set the directives for the implementation commands to RQS. These scripts are designed to be run on a design loaded into memory at the pre-`opt_design` stage. They do not include any reporting commands or checkpoint-writing steps.

Floorplanning

Floorplanning is the process of guiding placement and routing to improve performance, consistency, and timing closure in your design. While the Vivado tools automatically place and route most designs successfully, some designs require additional direction to meet timing goals or achieve predictable results between implementation runs.

You can apply floorplanning at different levels of details, from broad hierarchical guidance to precise cell-level placement. In each case, the goal is to influence how the placer organizes logic on the device so that related elements are physically close together, route delays are reduced, and specialized resources are used efficiently.

Common reasons to use floorplanning include:

- A design has never met timing, or does not meet timing consistently
- Timing is sensitive to placement of high-speed I/O interfaces, clock logic, or macros
- Critical paths span multiple regions of the device, especially in SSI devices with multiple SLRs
- Specialized resources such as block RAM, DSPs, and transceivers must be organized to reduce congestion and route length

The topics covered in this section include:

- 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 improve setup slack (TNS, WNS) by reducing the average route delay. During implementation, the timing engine works to resolve the worst setup violations and all hold violations. Floorplanning improves only setup slack.

Manual floorplanning works best when the netlist has hierarchy. Design analysis slows significantly when synthesis flattens the entire netlist. Configure 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 work with designs where separate logical structures are in lower sub-hierarchies. Register all outputs of a hierarchical module to simplify analysis. Paths that pass through multiple hierarchical blocks are more difficult to analyze and place.

Understanding Floorplanning Basics

Not every design meets timing on its own. You might need to guide the tools to a solution. Floorplanning lets you guide the tools through high-level hierarchy layout or detailed gate placement.

You can achieve the greatest improvements by fixing the worst or most common problems first. For example, if there are outlier paths with significantly worse slack or high logic levels, fix those paths first. Use **Reports → Timing → Create Slack Histogram** to view outlier paths. If the same timing endpoint appears in several negative slack paths, improving one of these paths can also improve the others on that endpoint.

Use floorplanning to increase performance by reducing route delay or increasing logic density on a non-critical block. Logic density measures how tightly the logic is packed on the chip. Floorplanning can help you reach a higher clock frequency and improve consistency.

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

Detailed Gate-Level Floorplanning

Detailed gate-level floorplanning places individual leaf cells in specific device sites.

Advantages of Detailed Gate-Level Floorplanning

- Works with hand routing nets
- Can extract maximum performance from the device

Disadvantages of Detailed Gate-Level Floorplanning

- Time consuming
- Requires extensive device and design knowledge
- Might need to be redone if the netlist changes

RECOMMENDED: *Use detailed gate-level floorplanning as a last resort.*

Information Reuse

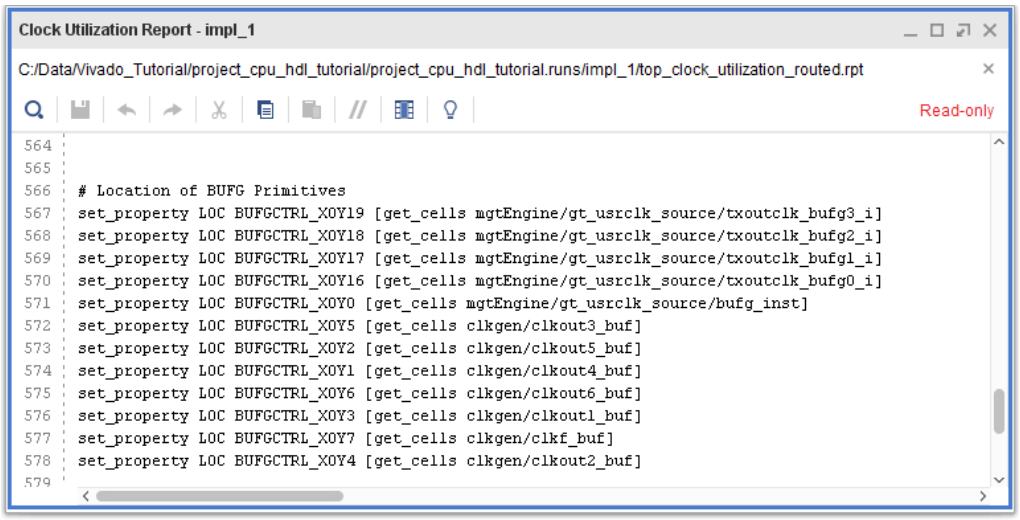
Reuse placement and timing information from a design that met timing if your current design does not consistently meet timing.

1. Open two implementation runs:
   - One that meets timing
   - One that does not meet timing

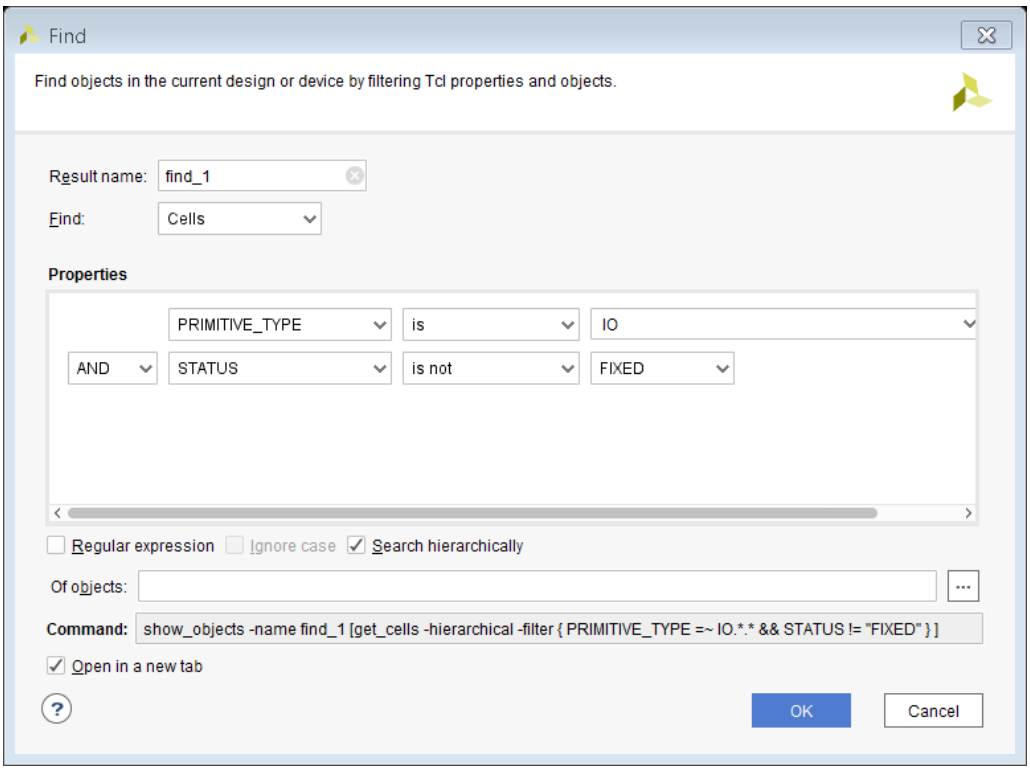
   TIP: *On a system with multiple monitors, click* ***Open Implementation in New Window*** *to view designs side by side.*
2. Identify failing timing paths in the run that does not meet timing using `report_timing_summary`.
3. On the run that meets timing, run `report_timing` in `min_max` mode for those same paths.
4. Compare the timing results:
   a. Clock skew
   b. Datapath delay
   c. Placement
   d. Route delays
5. If logic delay differs between path endpoints, review the synthesis runs.

Review I/O and Cell Placement

1. Compare I/O reports between the two runs. Review I/O placement and I/O standards. Confirm all I/Os are placed.
2. Search for I/Os without fixed placement.
3. If clock skew differs between runs, reuse clock primitive placement from the timing-closed run. Use the Clock Utilization Report to locate clock tree drivers. Copy LOC constraints into your XDC file.



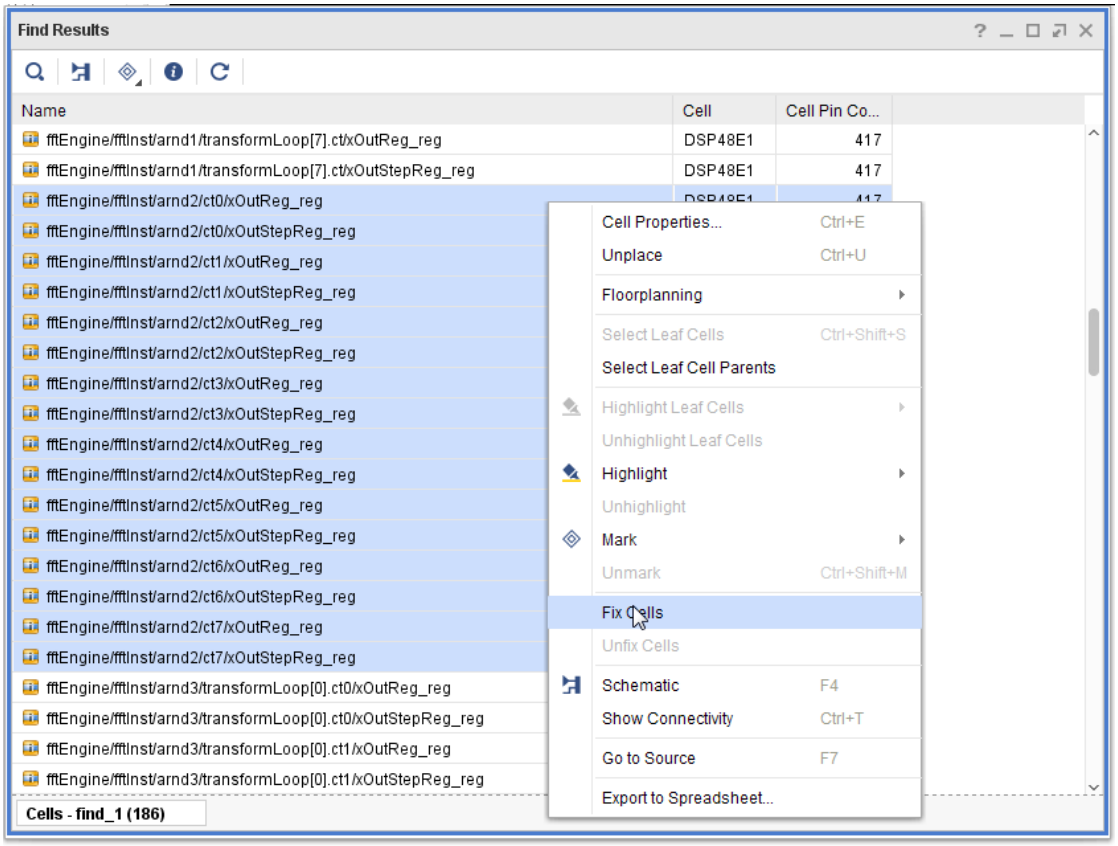
4. Consider reusing placement of Block RAMs and DSPs by using **Edit → Find** to list the instances.



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**



RECOMMENDED: *Highlight and analyze placement by hierarchy name before fixing placement.*

Reusing Placement

You can reuse placement for I/Os, global clock resources, Block RAM macros, and DSP macros to reduce variability between netlist revisions. These primitives generally have stable names and maintainable placement.

TIP: *Do not reuse placement for general slice logic or sections of the design likely to change.*

Reusing Placement with Incremental Compile

1. Reference an existing placed or routed DCP before running `place_design`.
2. Choose to reuse:
   - The full design
   - A specific hierarchy level
   - A cell type such as DSPs or Block RAMs
3. Let Incremental Compile handle changes automatically for modified sections.

For more details, see the *Vivado Design Suite User Guide: Implementation* (UG904).

Floorplanning Techniques

Consider gate-level floorplanning for a design that has never met timing, and when changing the netlist or constraints is not a good option.

RECOMMENDED: *Try hierarchical floorplanning before using gate level floorplanning.*

Hierarchical Floorplanning

Hierarchical floorplanning places one or more hierarchy levels into a region on the chip. This region guides the placer at a global level, and the placer handles detailed placement.

Advantages Over Gate-Level Floorplanning

- Creation is faster than gate-level floorplanning
- Can improve timing
- Less sensitive to design changes
- The hierarchy level acts as a container for all gates and generally works even if the netlist changes

Using Hierarchical Floorplanning

1. Identify the lower-level hierarchies containing the critical path.
2. Use the top-level floorplan to choose where to place them.
3. Let implementation place individual cells.
4. Leverage the tool's knowledge of cell connectivity and timing paths for fine-grain placement.

Manual Cell Placement

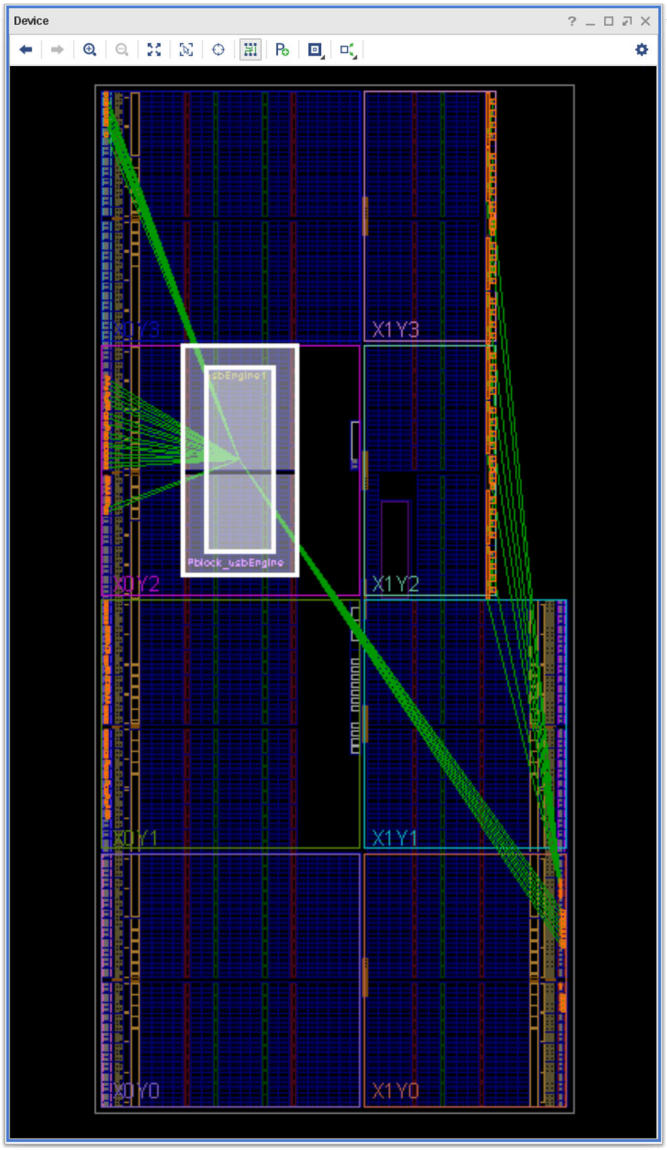
Manual cell placement achieves the best performance on a device. Designers often use it on a small portion of the design, such as logic around a high-speed I/O interface, or for block RAMs and DSPs. Manual placement can be slow.

All floorplanning techniques can require significant engineering time and multiple iterations. If cell names change, update the floorplan constraints.

When floorplanning:

- Know the final pinout
- Fix I/Os to provide anchor points for the floorplan
- Place blocks that communicate with I/Os near their I/Os

Figure 213: **I/O Components Pulling Design Apart**



TIP: *If the pinout is pulling a block apart, consider:*

- Modifying the pinout
- Modifying the RTL
- Constraining only block RAMs and DSPs
- Unplacing I/O registers if external timing requirements allow

Using Pblock-Based Floorplanning

The highlighted block in the previous figure is a Pblock. Pblocks constrain cells to a specific area of a device. The Pblock in the previous figure constrains some fabric logic, DSPs, and RAMs to the highlighted area. The Pblock was created with the following commands:

```
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}
```

The first line creates the Pblock. The second line (`add_cells_to_pblock`) assigns a hierarchical cell to the Pblock. Four resource types are assigned to the Pblock: SLICE, DSP48, RAMB18, and RAMB36. The resources assigned to a Pblock are called the Pblock's range.

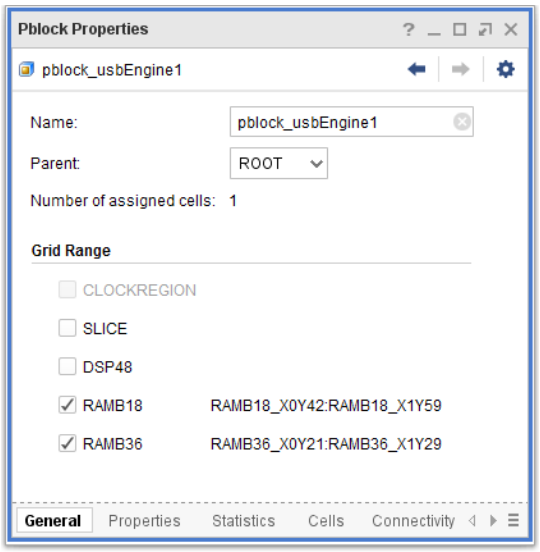
***Note*:** By default, Pblocks are soft constraints, which guide cell placement within the Pblock range while permitting placement outside the range when required for optimal quality of results (QoR). To make a Pblock hard and force all assigned cells to be placed within the Pblock range, set the `IS_SOFT` property to 0:

```
set_property IS_SOFT 0 [get_pblocks Pblock_usbEngine]
```

***Note*:** Although Pblocks are soft by default, they cannot exceed the utilization capacity of the assigned resources.

Logic assigned to a Pblock that has no corresponding site types in the Pblock range is not constrained by the Pblock and can be placed anywhere on the device. This means you can assign an entire hierarchy to a Pblock while only constraining specific cell types by including only those resource types in the Pblock range. For example, to update the previous Pblock example to only constrain the RAMs in the assigned hierarchy, assign only the RAM sites to the Pblock range.

Figure 214: **Pblock Grids**



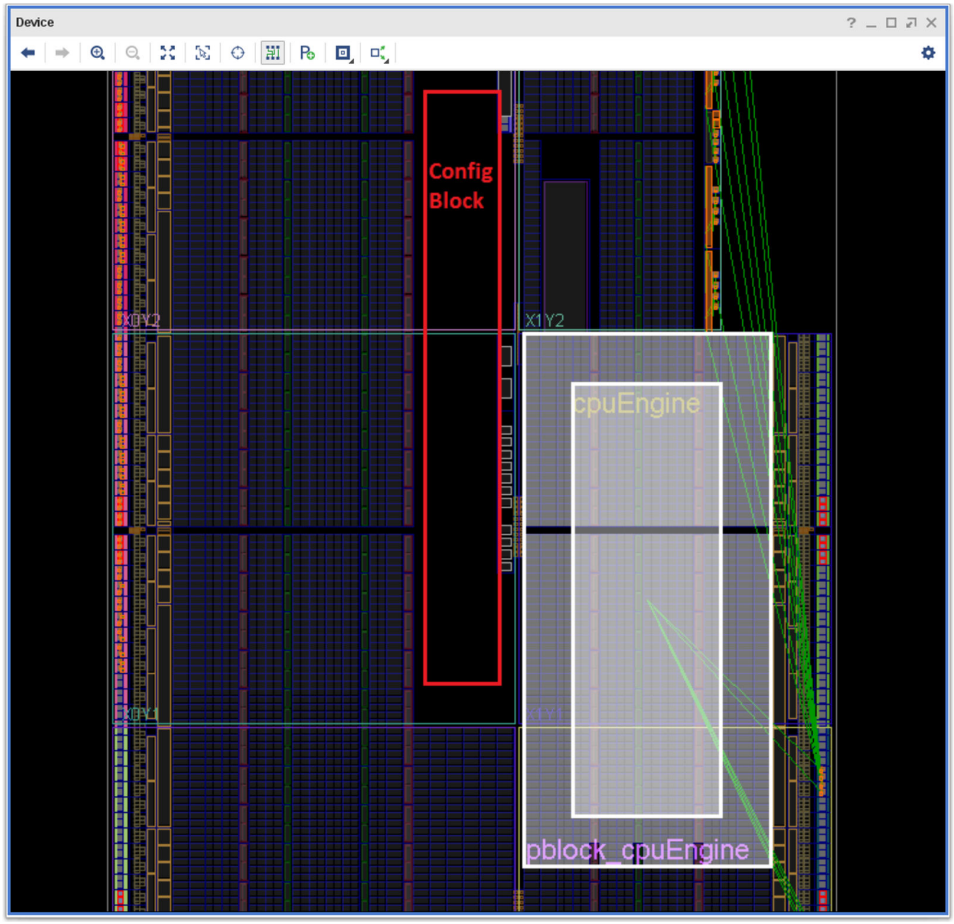
The equivalent XDC commands to change the Pblock to only constrain the RAM cells are:

```
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 to the Pblock boundaries, 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 215: **Avoiding the Config Block**



Using Pblocks to View Design Interconnect

When you integrate RTL into a design, visualizing the design inside the device can help you understand how it is structured. Viewing how blocks interconnect with each other and with the I/O pinout after synthesis provides valuable insight.

To view the interconnect, generate a top-level floorplan using Pblocks on upper levels of hierarchy.

Pblocks can be more than 100 percent full during analysis, but not during implementation. Overfilling a Pblock during analysis makes it smaller on the device. This is a useful technique for seeing the relative size of your top-level blocks and how they occupy the device.

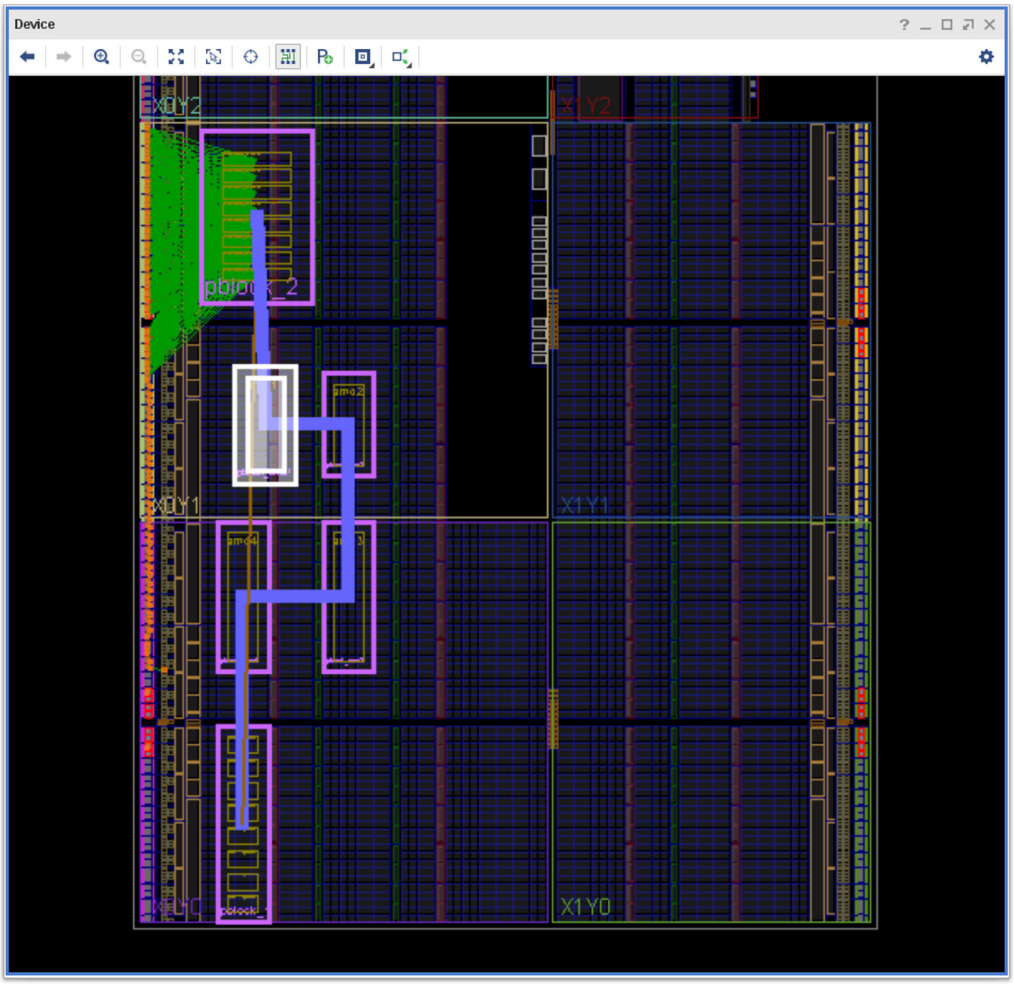
Top-Level Floorplan

A top-level floorplan shows which blocks communicate with I/Os. Green lines represent I/O connections. Nets connecting two Pblocks are bundled, and the bundles change size and color based on the number of shared nets.

Data Path Top-Level Floorplan

This view shows data flow between top-level blocks. Each block communicates only with two neighbors. Green lines indicate well-placed I/Os that communicate with a single block.

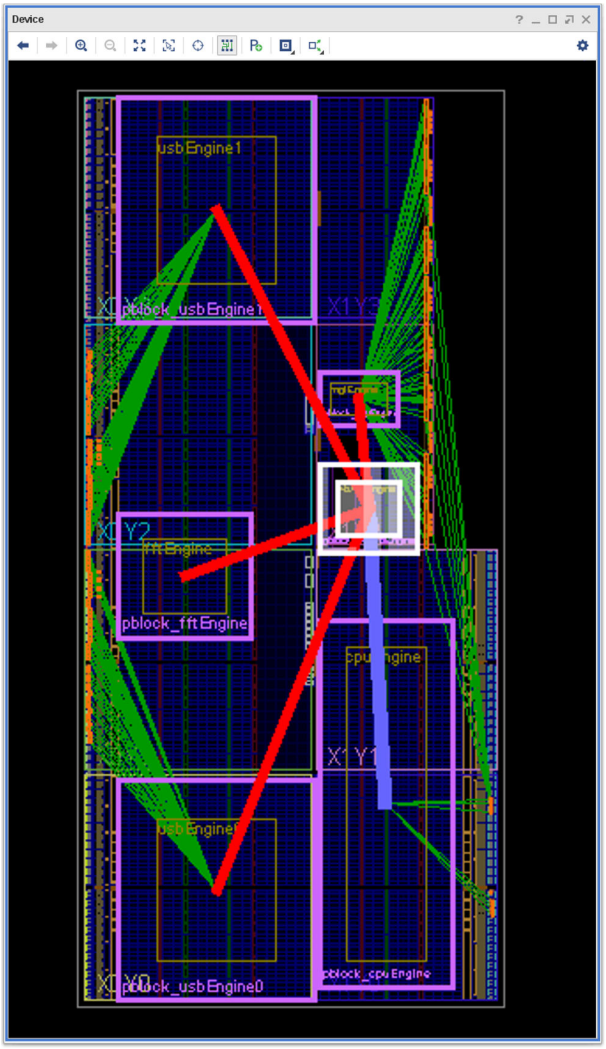
Figure 216: **Data Path Top Level Floorplan**



Control Path Floorplan

This view shows a design where all 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 reach all other loads.

Figure 217: **Control Path Floorplan**



Analyzing Utilization Statistics

A common cause of implementation issues is failing to account for explicit and implicit physical constraints:

- **Explicit physical constraints:** Examples include the pinout, which directly constrains logic placement.
- **Implicit physical constraints:** Certain specialized resources are available only in specific device locations, influencing placement. Examples include:
  - I/Os
  - Gigabit transceivers
  - DSP slices

- Block RAM
- Clock management blocks such as MMCM
- Clock buffers such as BUFG

Blocks that consume large amounts of these specialized resources might need to be spread across the device, which affects placement and routing for the rest of the design.

Pblocks are explicit physical constraints that define allowable placement areas for specified logic. To analyze block resource usage on the device, use one or more of these methods:

- Report utilization
- Netlist properties
- Pblock properties

Locking Specific Logic to Device Sites

You can place cells on specific locations on the FPGA, such as assigning all I/O ports in an AMD 7 series FPGA design. Place the I/Os before attempting to close timing.

I/O placement can influence cell placement in the FPGA fabric. Manually placing other cells in the fabric can help create consistent clock logic and macro placement, improving the consistency of implementation runs.

Table 21: **Constraints Used to Place Logic**

Constraint	Use	Notes
LOC	Places a gate or macro at a specific site.	SLICE sites contain subsites called BEL sites.
BEL	Specifies the subsite in the slice to use for a basic element.	

Fixed and Unfixed Cells

These terms apply to placed cells and describe how Vivado tools treat them in the design. For more details, see *Modifying Placement* in the *Vivado Design Suite User Guide: Implementation* (UG904).

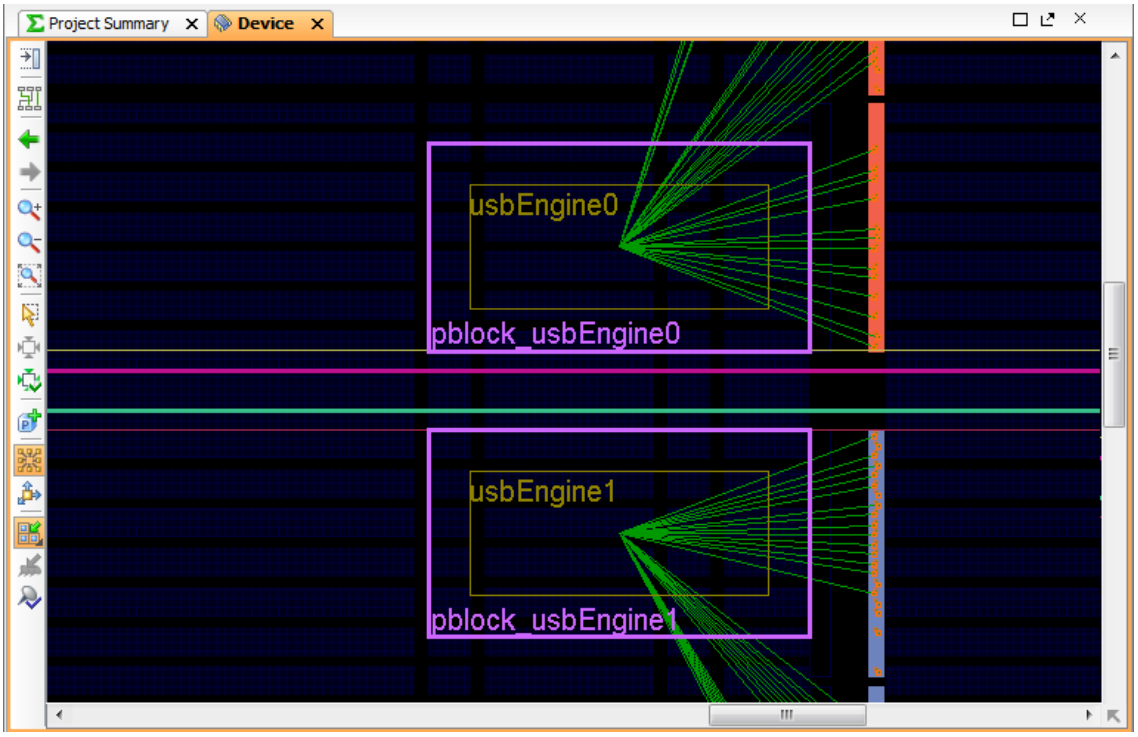
RECOMMENDED: *After placing the I/Os, use a hierarchical Pblock floorplan as a starting point for controlled placement. Use manual cell placement when Pblocks are not effective.*

Floorplanning With Stacked Silicon Interconnect (SSI) Devices

SSI devices consist of multiple super logic regions (SLRs) joined by an interposer. The interposer connections, called super long lines (SLLs), introduce additional delay when crossing from one SLR to another.

Follow these guidelines:

- Keep SLR boundaries in mind when structuring the design, generating a pinout, and floorplanning.
- Minimize SLL crossings by keeping critical timing path logic inside a single SLR.



- Place I/Os in the same SLR as the associated I/O interface circuitry.
- Carefully plan clock placement when laying out logic for SSI devices.
- Let the placer perform an automatic placement before attempting extensive manual partitioning. Review the results for floorplanning opportunities you might not have considered.

Determining if Hold-Fixing is Negatively Impacting the Design

The Vivado router prioritizes fixing hold timing before setup timing. A design might still function in the lab if setup is failing by a small amount because you can lower the clock frequency. However, if hold timing fails, the design most likely does not function.

In most cases, the router can meet hold timing without affecting setup timing. In some cases, often caused by design or constraint errors, setup timing can be significantly degraded.

Improper hold checks are commonly caused by incorrect `set_multicycle_path` constraints that omit the `-hold` option. Large hold requirements can also result from excessive clock skew. In this case, review the clocking architecture for the affected circuit. See Identifying Timing Violations Root Cause in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

This issue can occur when the design meets setup timing after placement but fails setup timing after routing. To investigate, use the `report_design_analysis` command with the `-show_all` option to view path delay caused by routing detours that the router adds to fix hold violations.

Figure 218: **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 estimated hold timing after placement and look for unusually large hold violations greater than 500 ps.*

If you suspect hold fixing is affecting timing closure, use one of these methods:

- Method 1: Routing without Hold Fixing
- Method 2: Checking Hold Timing on the Worst Failing Setup Path

Method 1: Routing without Hold Fixing

1. Read the post-placed checkpoint into Vivado.

2. Add a constraint to disable all hold checks:

```
set_false_path -hold -from [get_clocks]set_false_path -hold -to
[get_clocks]
```

> **CAUTION!** *Use this constraint only for testing. Do not use it in production or deliver it to another designer. Remove it before releasing the production design.*

3. Run the routing process:

```
route_design
```

4. Generate a timing summary:

```
report_timing_summary
```

5. Compare the WNS values with and without hold checks. A significant difference indicates the hold violations might be large enough to affect setup paths.

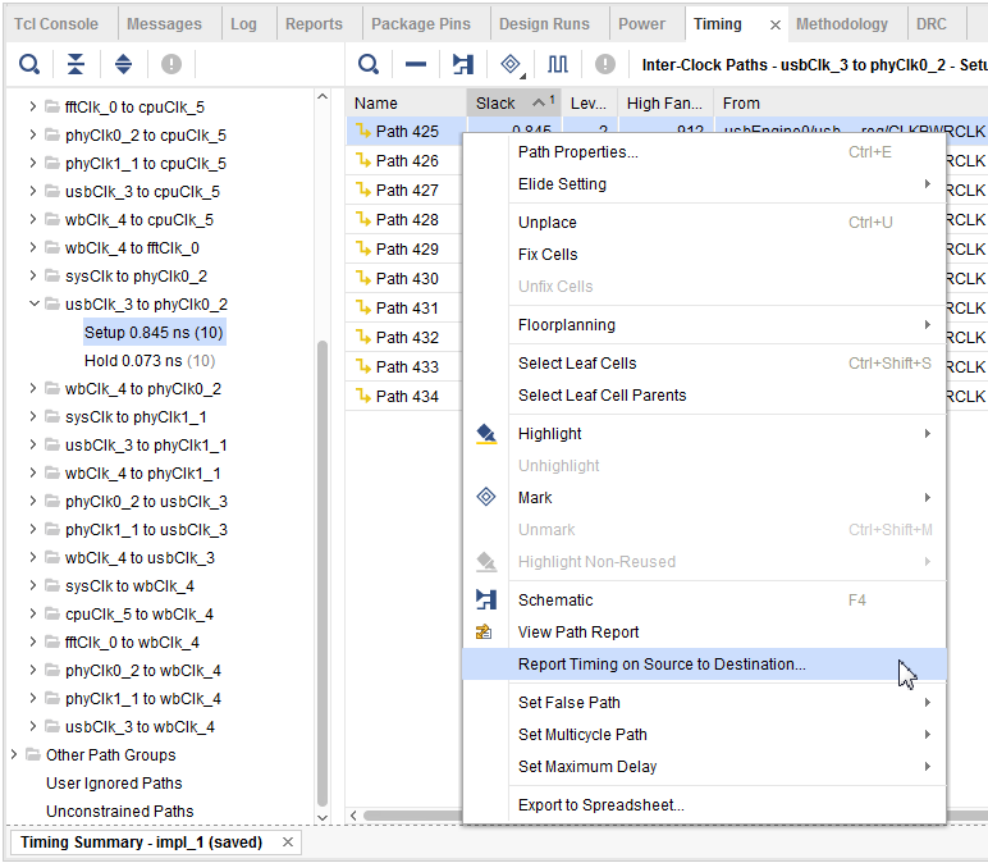
Method 2: Checking Hold Timing on the Worst Failing Setup Path

1. Identify the worst failing setup path.
2. In the Vivado IDE, right-click the path and run a timing report from source to destination.
3. Run the following command to view hold timing:

```
report_timing -hold
```

4. Verify the hold requirement and check whether additional delay was added to meet hold timing.

Figure 219: **Running a Timing Report on Specific Paths**

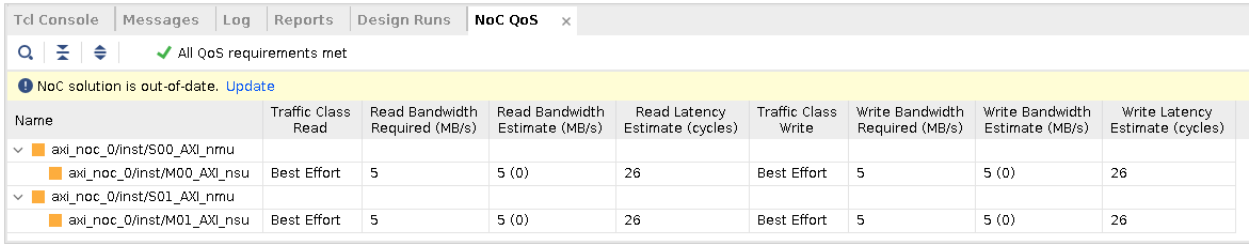


Performing NoC Quality of Service Analysis in Versal Devices

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 AXI4-Stream NoC IP.

If the NoC solution becomes out-of-date, run the NoC compiler to regenerate the solution and update the QoS report. An out-of-date report displays a banner message indicating the status.

Figure 220: **Out-of-Date Vivado NoC QoS Report**



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

Selecting **Update** regenerates the NoC solution.

***Note*:** In IP integrator, use the `validate_bd_design` Tcl command to update the NoC solution. Use the `update_noc_qos` Tcl command in the implementation tools or flows.

Actions That Can Cause the NoC Solution to Become Out-of-Date

- In the IP integrator:
  - Updating the AXI NoC IP QoS requirements in the IP integrator
  - Changing the IP integrator block design
  - Modifying the NoC master unit (NMU) or NoC slave unit (NSU) assignments in the NoC view
- In implementation tools or flow:
  - Unplacing NMU or NSU instances
  - Adding XDC physical constraints that impact NMU or NSU placement
  - Changing logical net connections to NMU or NSU

Information Provided in the NoC QoS Report

For each NoC connection, the report shows read and write transaction data:

- **Traffic Class:** The traffic class specified in IP integrator
- **Bandwidth Required:** The required bandwidth in MB/sec, specified in the NoC IP integrator
- **Bandwidth Estimate:** The estimated bandwidth in MB/sec from the NoC Compiler
- **Latency Estimate:** The structural latency, in NoC clock cycles, from the NoC Compiler

IMPORTANT! *The latency estimate is structural, not dynamic. It reports the minimum round-trip time for a transaction. The report measures latency in NoC clock cycles, whereas the AMD performance traffic generator IP reports latency in AXI clock cycles during simulation. Validate your design with actual traffic stimulus before design signoff*

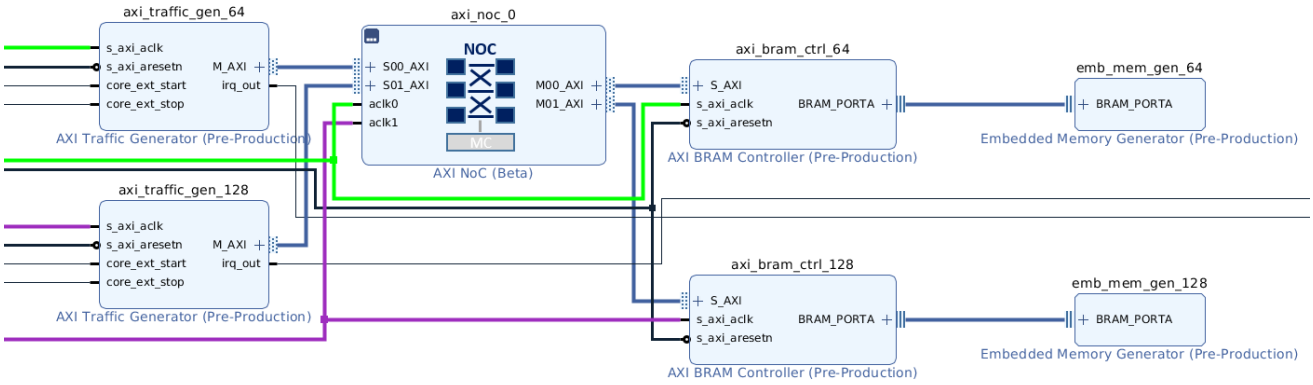
Example NoC QoS Analysis

The following example shows an IP integrator design with two AXI Traffic Generators and two AXI Block RAM Controllers, each connected to embedded memory. This example demonstrates adjusting data width and PL clock frequency to achieve the same bandwidth.

- `axi_traffic_gen_64`, `axi_bram_ctrl_64`, and `emb_mem_gen_64` use 64-bit data widths and connect to a 200 MHz clock (green).
- `axi_traffic_gen_128`, `axi_bram_ctrl_128`, and `emb_mem_gen_128` use 128-bit data widths and connect to a 100 MHz clock (purple).

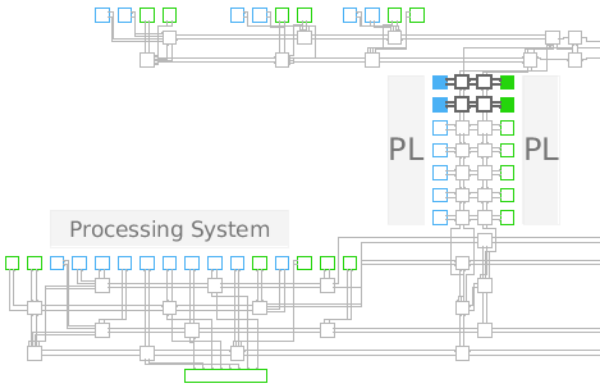
Both traffic generators connect to their respective Block RAM controllers through the NoC. The required read/write bandwidth for each connection is 1000 MB/sec.

Figure 221: **Example IP Integrator Block Design**



After validating the design in IP integrator, the initial NoC solution routes each connection through the horizontal NoC.

Figure 222: **Initial NoC Solution**



The QoS report for this solution shows that the bandwidth requirements are met and that each connection has a structural latency of 26 NoC clock cycles.

Figure 223: **Initial NoC QoS Report**

Tcl Console | Messages | Log | Reports | Design Runs | NoC QoS

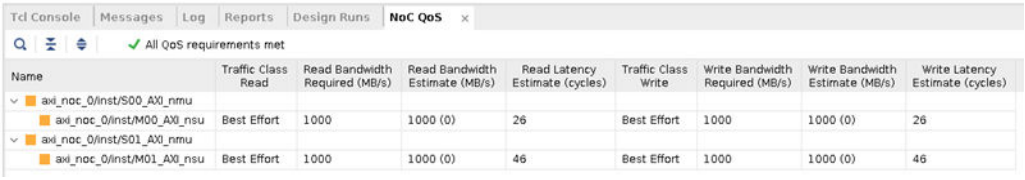
All QoS requirements met

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

Modifying NMU and NSU Assignments

In the NoC view, you can reassign NMUs and NSUs to explore latency and routing effects. In this example, the `axi_noc_0/inst/M01_AXI_nsu` is moved away from `axi_noc_0/inst/S01_AXI_nmu` to create a longer path through additional NoC switches.

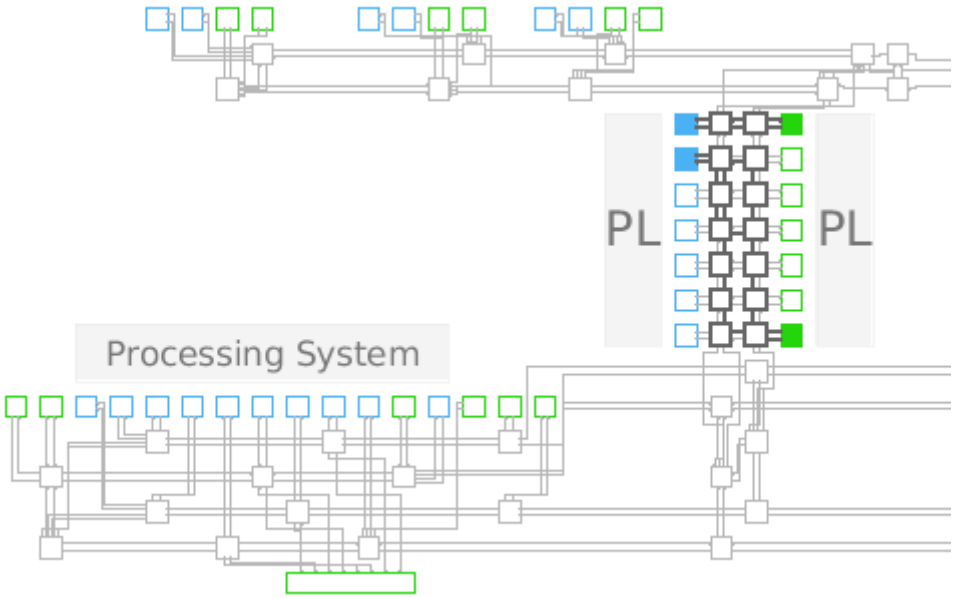
Figure 224: **NoC Solution with Manual NSU Assignment**



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

The updated QoS report shows that bandwidth is maintained but the structural latency increases from 26 to 46 NoC clock cycles.

Figure 225: **NoC QoS with Manual NSU Assignment**



Appendix A

Timing Methodology Checks

The timing methodology checks identify potential issues in design constraints, clock definitions, clock relationships, and timing exceptions that can lead to inaccurate timing analysis or failures in hardware. These checks run automatically as part of the design rule check (DRC) process in the AMD Vivado™ Design Suite.

Each timing methodology check is assigned a unique identifier (for example, TIMING-1) and includes the following information:

- Description of the condition that triggers the check.
- Resolution steps to address or prevent the issue.
- Example (if applicable) showing diagrams, constraint code, or other illustrations.

The checks cover a wide range of scenarios, including:

- Invalid clock definitions and waveform mismatches.
- Improperly related clocks without a common source, node, or period.
- Unsafe or incomplete clock domain crossing (CDC) synchronizations.
- Missing or incorrect input/output delay constraints.
- Large setup or hold violations caused by skew or design topology.
- Improper use of timing exceptions such as `set_max_delay` with `-datapath_only`.

Not every timing methodology check includes an example, but all provide a clear description of the problem and recommended steps to resolve it. Following these recommendations improves the accuracy of timing analysis, reduces the risk of hardware failures, and increases the likelihood of successful timing closure.

TIMING-1: Invalid Clock Waveform on Clock Modifying Block

Message

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

Vivado automatically derives clocks on the output of a clock modifying block (CMB) based on the CMB settings and the characteristics of the incoming master clock. If you define 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 the behavior in hardware.

Resolution

1. If the user-defined generated clock is not necessary, remove the constraint and use the auto-derived clock instead.
2. If the constraint is necessary, verify that the generated clock constraint matches the auto-derived clock waveform or modify the CMB properties to match the expected waveform.
3. If you only want to rename the auto-derived clock, use the `create_generated_clock` constraint with only the `-name` option defined, along with the name of the object where the clock is defined (typically the output pin of the 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.

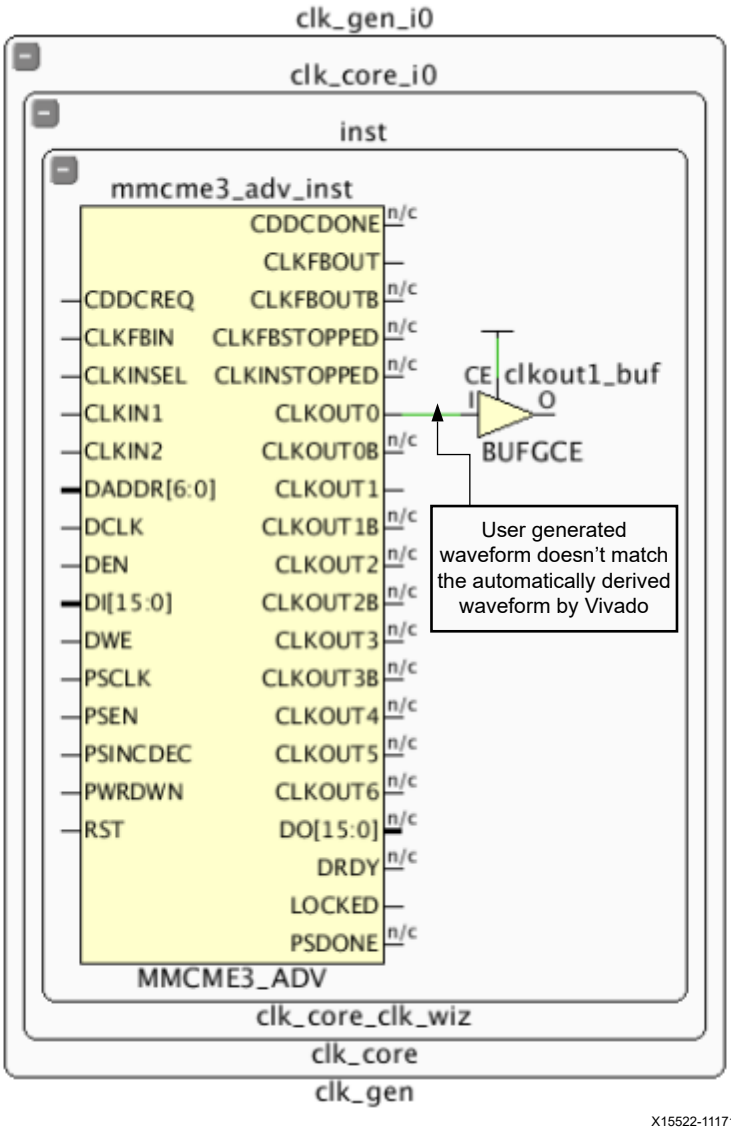
Example

In the following figure, a `create_generated_clock` constraint was defined on the mixed-mode clock manager (MMCM) instance pin `CLKOUT0`, but it does not match the auto-derived waveform generated by Vivado from the MMCM attribute settings.

To rename the auto-derived clock, add this constraint immediately after defining the master clock in your constraint files:

```
create_generated_clock -name clkName [get_pins clk_gen_i0/clk_core_i0/inst/
mmcme3_adv_inst/CLKOUT0]
```

Figure 226: **Invalid Clock Waveform on Clock Modifying Block**



TIMING-2: Invalid Primary Clock Source Pin

Message

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 at 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 becomes inaccurate because it ignores the insertion delay before the primary clock source point. This prevents correct skew computation and can cause failures in hardware.

Creating a primary clock on an internal driver pin is discouraged. This practice can cause inaccurate timing analysis results and improper clock skew reporting.

Resolution

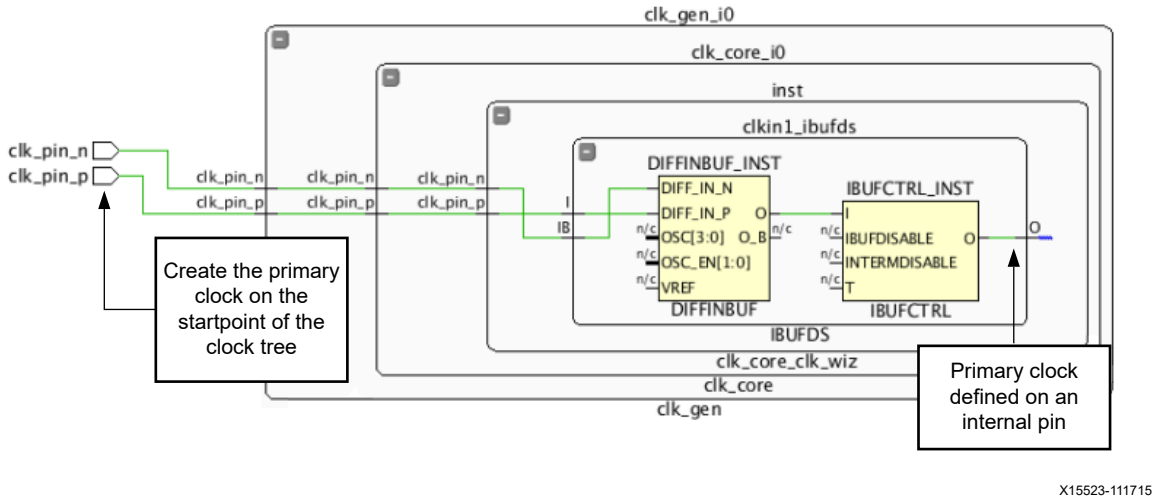
Modify the `create_clock` constraint to use the actual clock tree source.

Example

In the following figure, the primary clock definition using the `create_clock` constraint is placed on the output pin of the input buffer control (IBUFCTRL) instance. If the clock `clk_pin_p` is used to time an input or output port path, the slack is inaccurate because part of the clock tree insertion delay is missing.

Place the primary clock definition for the differential input buffer on the top-level port `clk_pin_p`.

Figure 227: **Invalid Primary Clock on Internal Pin**



TIMING-3: Invalid Primary Clock on Clock Modifying Block

Message

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 you define a primary clock on the output of a CMB, Vivado does not auto-derive a clock on the same output.

This check reports that a primary clock has been created on the output of a CMB. This breaks the relationship with the incoming clock and prevents accurate clock insertion delay computation. Defining a primary clock here is not recommended because it can cause inaccurate timing analysis hardware failure.

Resolution

1. If the violation refers to a user-defined clock, remove the `create_clock` constraint on the output of the CMB.

2. If you want to force the name of the auto-generated clock, use the `create_generated_clock` constraint with only the `-name` option and the CMB output pin.
3. If the clock is generated by the Clocking Wizard, verify the setting in the **Input Clock Information → Primary Input Clock** menu:
   - Use **Single ended clock enabled pin** when the input clock is directly connected to a top-level I/O port.
   - Use **Global Buffer** when the input clock comes from another Clocking Wizard instance.

See the *Vivado Design Suite User Guide: Using Constraints* (UG903) for additional information about creating generated clocks.

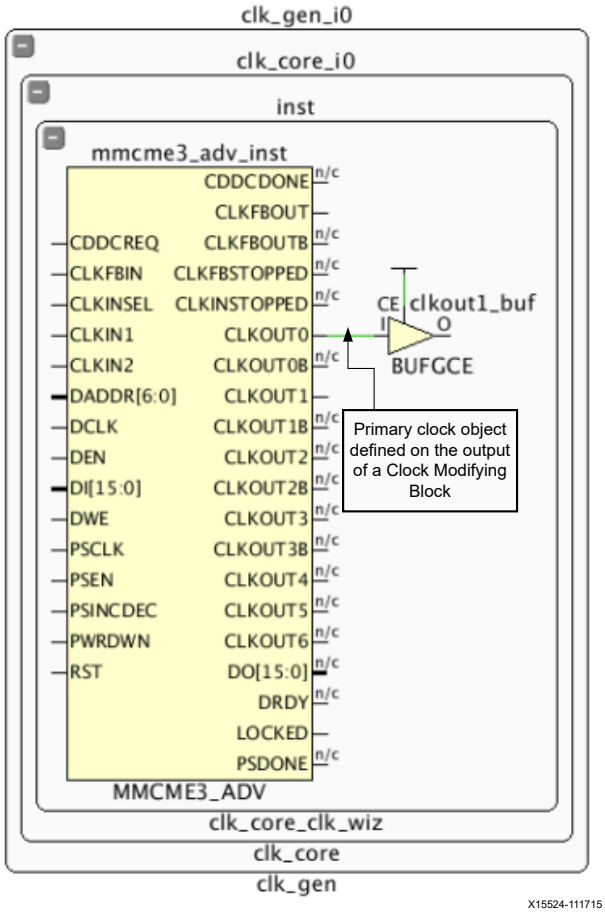
Example

In the following figure, a `create_clock` constraint is defined on the MMCM instance pin `CLKOUT0`. This overrides the automatically derived clock created by Vivado and removes the relationship with the incoming clock.

To rename the auto-derived clock, add this constraint immediately after defining the master clock in your constraint files:

```
create_generated_clock -name clkName [get_pins clk_gen_i0/clk_core_i0/inst/
mmcme3_adv_inst/CLKOUT0]
```

Figure 228: **Invalid Primary Clock on Clock Modifying Block**



TIMING-4: Invalid Primary Clock Redefinition on a Clock Tree

Message

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 at the source of the clock tree. For example, this is typically the input port of the design. When a primary clock is defined downstream and overrides the incoming clock definition, timing analysis becomes inaccurate because it ignores the insertion delay before the redefined primary clock source point. This prevents correct skew computation and can result in incorrect timing analysis, which can lead to hardware failures.

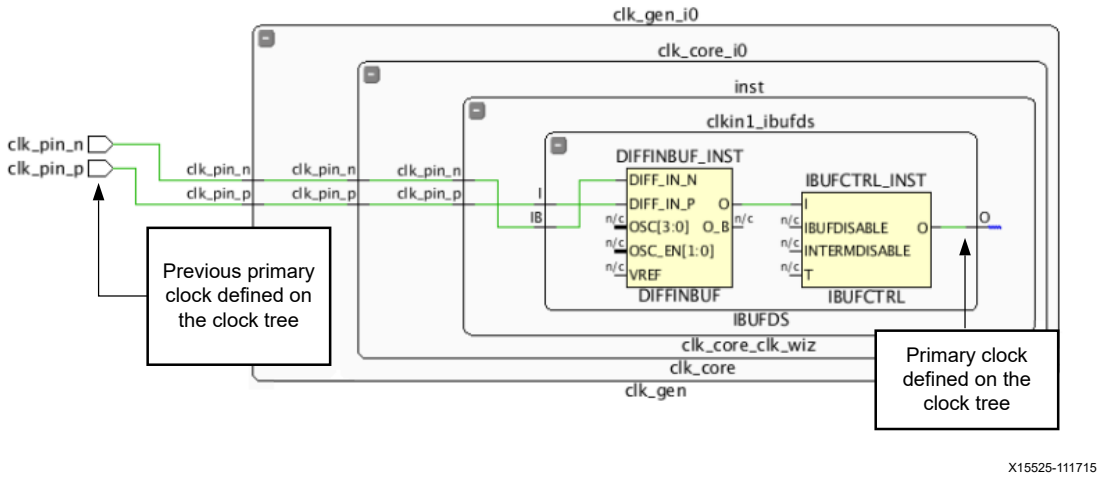
Resolution

Remove the `create_clock` constraint on the downstream object and allow the propagation of the upstream clock, or create a generated clock that references the upstream primary clock.

Example

In the following figure, the primary clock is correctly defined on the top-level port `clk_pin_p`. However, a `create_clock` constraint is used to redefine the primary clock on the output of the input buffer control (IBUFCTRL) output. This new clock ignores all delays prior to the IBUFCTRL.

Figure 229: **Invalid Primary Clock Redefinition on a Clock Tree**



TIMING-5: Invalid Waveform Redefinition on a Clock Tree

Message

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

Define a generated clock in relation to the incoming clock. This check reports an invalid definition, such as a different period, phase shift, or inversion compared to the incoming clock.

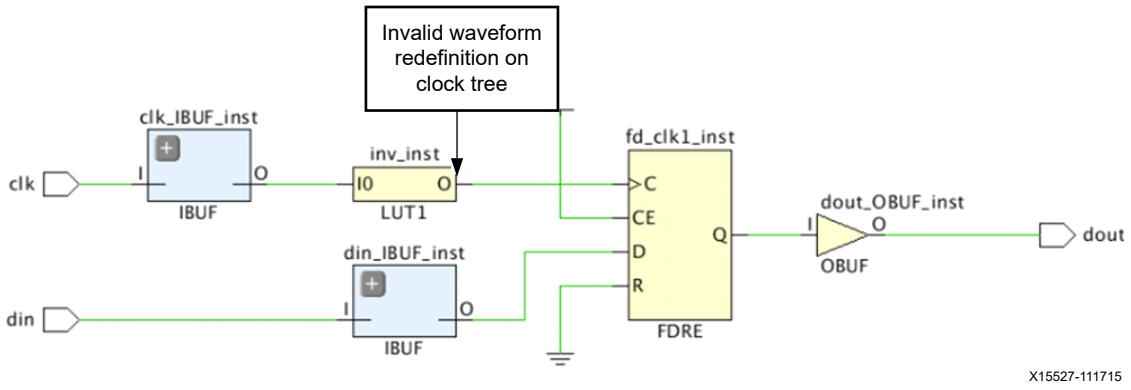
Resolution

Modify the `create_generated_clock` constraint to define a waveform that matches the incoming clock definition. For more details about creating a proper generated clock constraint, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Example

In the following figure, a `create_generated_clock` constraint is created on the output of the LUT1 inverter, but the `-invert` switch is not applied.

Figure 230: **Invalid Waveform Redefinition on a Clock Tree**



TIMING-6: No Common Primary Clock Between Related Clocks

Message

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. This check reports that the timing engine cannot guarantee these clocks are synchronous.

Resolution

The solution depends on whether the two clock domains are asynchronous or synchronous.

- If asynchronous, cover the paths between the two domains with a timing exception such as:

  ```
  set_max_delay -datapath_only ...
  set_clock_groups ...
  set_false_path ...
  ```

- If synchronous, adjust the clock constraints to correctly define the relationship between the two clocks.

Example

1. If both clocks have the same waveform, define one timing clock on both source objects:

   ```
   create_clock -period 10 -name clk1 [get_ports <clock-1-source> <clock-2-
   source>]
   ```

2. If the clocks have different waveforms, define the first clock as a primary clock and the second as a generated clock, with the first clock as the master:

   ```
   create_clock -period 10 -name clk1 [get_ports <clock-1-source>]
   create_generated_clock -source [get_ports <clock-1-source>] -name
   clk2 ... [get_ports <clock-2-source>]
   ```

3. If the clocks are related with a period ratio of 2, create a primary clock on one source and a generated clock on the other:

```
create_clock -period 10 -name clk1 [get_ports <clock-1-source>]
create_generated_clock -source [get_ports <clock-1-source>] -name clk2
-divide_by 2 [get_ports <clock-2-source>]
```

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

Message

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. This check reports that the timing engine cannot guarantee these clocks are synchronous in hardware because it cannot determine a common node between the two clock trees.

Resolution

The resolution depends on whether the clock domains are asynchronous or synchronous:

- If asynchronous clocks, cover all paths between the two domains with a timing exception such as:

```
set_max_delay -datapath_only ...
set_clock_groups ...
set_false_path ...
```

- If synchronous clocks, you can waive this check.

If the violation occurs during out-of-context (OOC) synthesis of a module, and the two clocks have a common node at the top level, you can prevent the violation during OOC synthesis by:

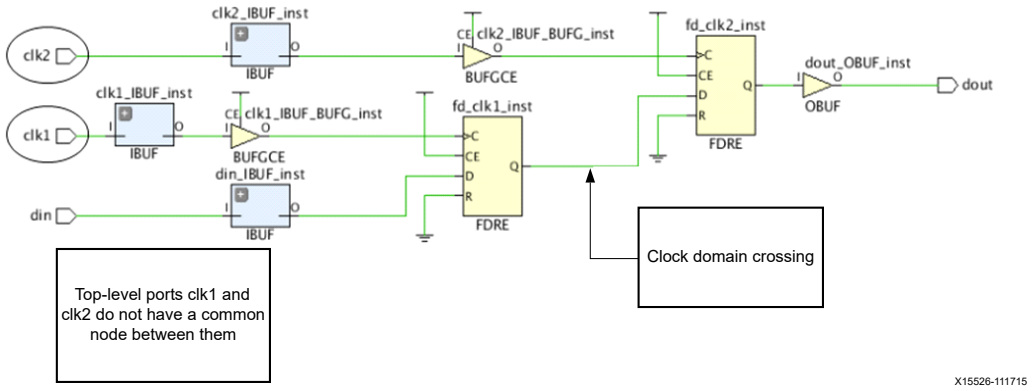
1. Defining one of the clocks as a primary clock on the first input clock port.
2. Defining the second clock as a generated clock on the second input clock port, referencing the primary clock from step 1.
3. Defining the `HD.CLK_SRC` property on both input clock ports.

Example

In the following figure, a synchronous clock domain crossing exists between `clk1` and `clk2`. Both clocks are considered synchronous by default, but no common node relationship is detected. If synthesized out-of-context and the clocks have a common node at the top level, you can suppress the violation with constraints such as:

```
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 231: **No Common Node Between Related Clocks**



TIMING-8: No Common Period Between Related Clocks

Message

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 cannot determine a common period after expanding the waveform of both clocks over 1,000 cycles. In this case, the worst setup relationship over these 1,000 cycles is used for timing analysis, but the timing engine cannot ensure this is the most pessimistic case.

This condition typically occurs with clocks that have an odd fractional period ratio.

Resolution

Because the waveforms do not allow safe timing analysis between the two clocks, treat these clocks as asynchronous. Cover the paths between the two clock domains with a timing exception such as:

```
set_max_delay -datapath_only ...
set_false_path ...
set_clock_groups ...
```

TIMING-9: Unknown CDC Logic

Message

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

This check ensures that inter-clock domain paths constrained with timing exceptions use safe asynchronous CDC circuitry. For more information on recognized safe topologies, see Report Clock Domain Crossings.

Resolution

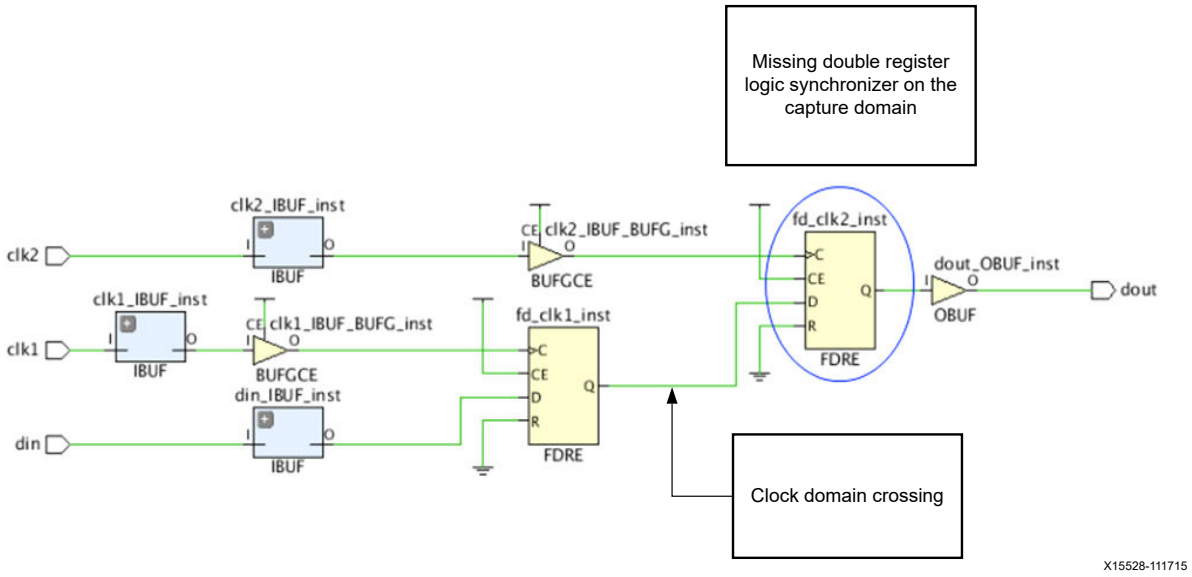
Design the crossing path with proper synchronization. At minimum, add a double-register logic synchronizer on the capture clock side.

If a FIFO or higher-level protocol already exists on the path, this check can be safely ignored. For a detailed list of CDC violations, run `report_cdc`.

Example

In the following figure, an asynchronous clock domain crossing exists between `clk1` and `clk2`. The `clk2` capture domain does not contain a double-register logic synchronizer to synchronize the data.

Figure 232: **Missing Synchronizer**



TIMING-10: Missing Property on Synchronizer

Message

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 the `ASYNC_REG` property set to `TRUE` to preserve the cells through logic optimization during synthesis and implementation, and to optimize placement for improved mean time between failure (MTBF) statistics.

The TIMING-10 violation is triggered when at least one of the first two synchronizer registers is missing the `ASYNC_REG` property.

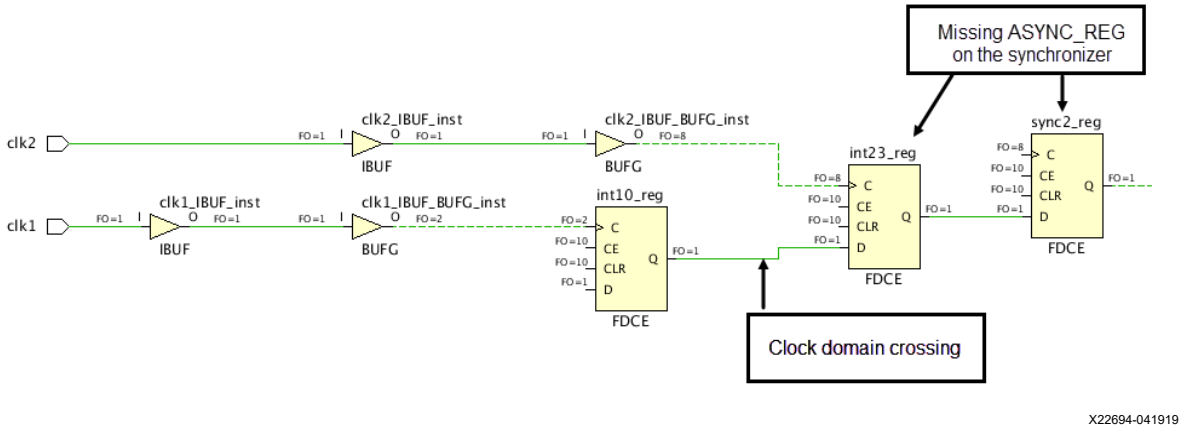
Resolution

Add the `ASYNC_REG` constraint to each stage of the logic synchronizer. For a detailed list of CDC violations, run `report_cdc`. For more information on the `ASYNC_REG` constraint, see 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 must have the `ASYNC_REG` property applied to increase timing slack and improve MTBF.

Figure 233: **Missing Property on Synchronizer**



TIMING-11: Inappropriate Max Delay with Datapath Only Option

Message

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` command with the `-datapath_only` option removes clock skew from the setup slack computation and ignores hold timing. This option is intended for asynchronous signal timing paths that:

- Do not have a clock relationship
- Require maximum delay constraints

It is not recommended to use this constraint on synchronous paths.

Resolution

Modify the `set_max_delay -datapath_only` constraint so that it does not cover synchronous timing paths. Review the startpoint and endpoint cells listed in the violation message to identify the associated `set_max_delay` constraint.

TIMING-12: Clock Reconvergence Pessimism Removal Disabled

Message

Clock reconvergence pessimism removal (CRPR) mode has been disabled. This is not recommended because over-pessimistic clock tree delays could make timing closure impossible.

Description

The CRPR feature removes artificially induced pessimism caused by using the maximum and minimum delays along the common portion of a clock network. If CRPR is disabled, timing analysis can become overly pessimistic, making closure more difficult.

Resolution

Enable CRPR analysis to ensure accurate timing information. Use the following Tcl command:

```
config_timing_pessimism -enable
```

TIMING-13: Timing Paths Ignored Due to Path Segmentation

Message

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 maximum and minimum 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).

Path segmentation is not recommended because it can cause incorrect timing analysis and lead to hardware failures.

Resolution

Avoid path segmentation by choosing valid startpoints and endpoints in the `set_max_delay` and `set_min_delay` constraints. For more information on path segmentation and proper usage of minimum and maximum delay constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-14: LUT on the Clock Tree

Message

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 a clock path can cause excess skew because the clock must be routed through general routing resources in the fabric rather than dedicated clock routing. In addition to increased skew, these paths are more susceptible to process, voltage, and temperature (PVT) variations.

This situation can be created during synthesis in cases such as clock gating or inversion. For example, an inversion LUT1 cell might be absorbed into the downstream SLICE after `opt_design`.

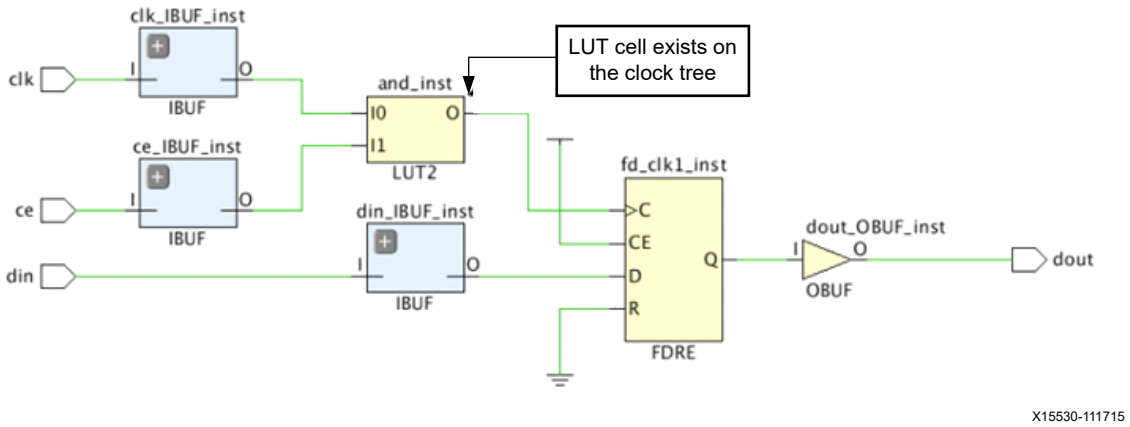
Resolution

Modify the design to remove the LUT located on the clock tree. If the LUT was created by synthesis for gating or inversion, review whether it is absorbed after `opt_design`.

Example

In the following figure, a LUT is used to gate the clock with a clock enable signal. The LUT on the clock path can cause excess skew, which is undesirable.

Figure 234: **LUT on the Clock Tree**



TIMING-15: Large Hold Violation on Inter-Clock Path

Message

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 reports that a large hold violation caused by inter-clock skew is likely to make timing closure difficult during implementation. Investigate any inter-clock skew greater than 1.0 ns to ensure proper constraints or design topology.

Resolution

Determine whether the large inter-clock skew is on a valid timing path or is related to non-optimal timing constraints. If the skew occurs on an unconstrained CDC path, add the appropriate timing exception. If the violation is due to clock tree-related logic, review the path topology for improvements that make timing closure easier.

TIMING-16: Large Setup Violation

Message

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 reports setup violations that are likely to make timing closure difficult during implementation. Review any setup violation greater than 1.0 ns to ensure constraints and design topology are optimal.

Resolution

Determine whether the setup violation occurs on a valid timing path or is due to non-optimal constraints. If the violation is on an unconstrained CDC path, add the necessary timing exception. If the violation results from a large amount of combinational logic, investigate changes to the path topology to improve timing closure.

TIMING-17: Non-Clocked Sequential Cell

Message

The clock pin `<pin_name>` is not reached by a timing clock.

Description

The DRC lists sequential cells unconstrained by a timing clock. This impacts timing analysis accuracy and coverage. Missing timing analysis can lead to hardware failures.

Resolution

Create the missing primary or generated clock on the clock tree driving the unconstrained sequential cells.

TIMING-18: Missing Input or Output Delay

Message

An `<input/output>` delay is missing on `<port_name>` relative to clock(s) `<clock_name>`.

Description

Input/output (I/O) timing is based on paths that involve external devices. The input and output delays specify the path delay of the ports relative to a clock edge at the interface. Missing these constraints can result in an FPGA interface that fails to meet the external device timing requirements.

***Note*:** A TIMING-18 violation is reported for any missing clock edge. If a clock is inverted during clock network propagation and a required delay is missing for the falling edge, a violation is still generated.

Resolution

Add the required input and output delay constraints according to the board application.

TIMING-19: Inverted Generated Clock Waveform on ODDR

Message

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 must be defined in relation to the incoming clock. An inverted waveform can cause hardware failures because timing analysis for ports associated with the forwarded clock does not match actual device behavior.

Resolution

Modify the `create_generated_clock` constraint to match the incoming clock waveform. For more details, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-20: Non-Clocked Latch

Message

The latch `<cell_name>` cannot be properly analyzed because its control pin `<pin_name>` is not reached by a timing clock.

Description

The DRC lists latch cells that are not constrained by a timing clock, which impacts timing analysis coverage and accuracy. Incomplete timing analysis can lead to hardware failures.

Resolution

Create the primary or generated clock on the clock tree source driving the unconstrained latch control pins.

TIMING-21: Invalid COMPENSATION Property on MMCM

Message

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 AMD Vivado™ IDE automatically selects 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

Message

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

The MMCM can be configured for external deskew when the feedback board trace matches the trace to the external components. The external delay value is used in calculating the MMCM compensation delay. This configuration can lead to hardware failures, especially on I/O paths, when the timing analysis of the MMCM compensation does not match actual device behavior.

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

Message

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

A combinational timing loop occurs when the output of combinational logic feeds back to its input, creating a loop that infinitely cycles through the same path. This loop cannot be timed and unnecessarily increases the number of cycles. To address this, the Vivado Integrated Design Environment (IDE) disables the timing arc on the cell in the loop.

Resolution

If the combinational feedback loop is unintentional, modify the design source files (RTL) to remove it. If the loop is intentional, use the `set_disable_timing` command to break the loop at the most appropriate point, typically the feedback path, rather than allowing Vivado timing to break it at a random location.

TIMING-24: Overridden Max Delay Datapath Only

Message

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 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 is not reported.

Resolution

Replace the `set_clock_groups` or `set_false_path` constraints between clocks with point-to-point false path constraints. This approach prevents incorrect overriding of a `set_max_delay -datapath_only` constraint.

TIMING-25: Invalid Clock Waveform on Gigabit Transceiver (GT)

Message

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 GT clocks, so a primary clock must be created on the GT output pins. This DRC warning indicates that the defined clock does not match the expected auto-derived clock that Vivado would generate. This mismatch can result in hardware failures because the timing constraints do not reflect actual device behavior.

Resolution

If the user-generated clock constraint is unnecessary, remove it and use the auto-derived clock instead. If the constraint is required, verify that it matches the auto-derived clock waveform or adjust the GT properties to match the expected clock waveform. To only rename the auto-derived clock, use the `create_generated_clock` constraint with the `-name` option and the GT output pin. For more information about creating generated clocks and restrictions on renaming auto-derived clocks, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-26: Missing Clock on Gigabit Transceiver (GT)

Message

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 GT outputs based on the GT settings and the characteristics of the incoming master clock. This DRC warning indicates that Vivado could not perform this auto-derivation because the required primary clock on the GT input port is not defined. Without this clock, timing analysis does not include downstream logic driven by GT-related clocks.

Resolution

Create a primary clock on the recommended GT input port to enable proper clock derivation and timing analysis.

TIMING-27: Invalid Primary Clock on Hierarchical Pin

Message

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 hierarchical pin are analyzed differently than the cells on the fanout of the driver pin. When synchronous paths exist between the driver clock and the hierarchical pin clock, skew becomes inaccurate and timing signoff is invalid. This situation can cause hardware failure.

Resolution

Remove the primary clock definition on the hierarchical pin. If a downstream clock is required, use a `create_generated_clock` constraint with the driver clock specified as the master clock instead.

TIMING-28: Auto-Derived Clock Referenced by a Timing Constraint

Message

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 must be referenced by its source pin object. The auto-derived clock name can change during development due to modifications to the netlist or constraints. Unless renamed intentionally, referencing an auto-derived clock by name is discouraged because it can result in invalid constraints in subsequent runs after design modifications.

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 other timing constraints. For more information about using a generated clock constraint to force a clock name, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-29: Inconsistent Pair of Multicycle Paths

Message

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 modifies the path requirement multipliers with respect to the source clock for hold or the destination clock for setup. In certain cases, the path requirement must be multiplied with respect to a specific clock edge.

Resolution

For both setup and hold, update the `set_multicycle_path` constraints so that they reference the destination clock (`-end`) for SLOW-to-FAST synchronous clocks and the source clock (`-start`) for FAST-to-SLOW synchronous clocks. For more information about properly setting multicycle paths between clocks, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-30: Sub-Optimal Master Source Pin Selection for Generated Clock

Message

The generated clock <clock_name> has a suboptimal master source pin selection, timing can be pessimistic.

Description

A generated clock must reference the clock propagating through its direct fan-in, although the `create_generated_clock` command allows specifying any reference clock. This DRC warning reports that the generated clock is associated with a master clock defined farther upstream than the actual incoming master clock. In this case, timing analysis can become more pessimistic and apply additional clock uncertainty to paths between the master clock and the generated clock. This can make timing closure slightly more difficult.

Resolution

Update 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

Message

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 devices, a phase-shifted output of an MMCM or PLL is modeled by default as a change in the clock waveform (`PHASESHIFT_MODE=WAVEFORM`). In this mode, the edges of the clock waveform shift to reflect the pin phase shift.

In UltraScale+ devices, the pin phase shift is modeled by default as a latency propagation delay through the clock modifying block (`PHASESHIFT_MODE=LATENCY`), without changing the clock waveform.

In waveform mode, multicycle constraints might be required to adjust the timing path requirement for the phase shift between the source and destination clock waveforms. In latency mode, these constraints are typically unnecessary.

For more information on phase shift modes, see MMCM/PLL Phase Shift Modes.

Resolution

Determine whether the multicycle constraint is needed. If the constraint was added to adjust the path requirement in waveform mode, remove it when using latency mode. To revert to waveform behavior, set `PHASESHIFT_MODE=WAVEFORM` on the clock modifying block.

TIMING-32: Bus Skew Constraint Applied on Too Many Signals

Message

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 meets the bus skew constraint by adding extra delay to individual signals through routing detours. When the number of signals in the same bus skew constraint is very high, the tool might need to insert a large number of detours. This can cause routing congestion, increase `route_design` runtime, or result in failure to meet the bus skew requirement.

Resolution

Review the bus skew constraint to confirm that all included signals require it. If the constraint is valid, consider redesigning the CDC interface to reduce the number of signals under the bus skew constraint.

TIMING-33: Invalid Bus Skew Constraint on Safely Timed Paths

Message

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 on a synchronous path can be unnecessary and might not reflect the intended timing strategy. When applied to safely timed paths, it could create misleading constraints or redundant analysis.

Resolution

Review the bus skew constraint to confirm it is applied only to asynchronous CDC paths. If the paths are expected to be asynchronous but are reported as synchronous by Vivado, check for missing timing exceptions such as `set_clock_groups`, `set_false_path`, or `set_max_delay -datapath_only`.

TIMING-34: Bus Skew Constraint with Unrealistic Value

Message

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

A bus skew value lower than half of the source or destination clock period, or less than 1 ns, is overly aggressive. While bus skew timing can be met by adding routing delay during `route_design`, meeting such a target can significantly increase runtime, lead to unnecessary routing congestion, and in some cases be impossible to achieve.

Resolution

Adjust the bus skew value so it is greater than or equal to half of the smallest clock period between the source and destination clocks, or at least 1 ns. This helps ensure reasonable routing effort and avoids excessive runtime.

TIMING-35: No Common Node in Paths with the Same Clock

Message

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 lack a common node between the source and destination clock pins when the clock is defined on more than one object (port, pin, or net). In this situation, the path is unsafe to time and is 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 intentionally defined on multiple source objects, apply asynchronous timing exceptions—such as `set_false_path` or `set_max_delay -datapath_only`—to cover paths clocked by different clock source objects.

TIMING-36: Invalid Generated Clock due to No Edge Propagation

Message

There is no rising or falling edge propagation between master clock <clock_name> to generated clock <clock_name>.

Description

No rising or falling edge from the master clock source pin propagates 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 timing signoff cannot be accurate.

Resolution

Review the master clock used to define the generated clock. There must be a physical path between the master clock and the generated clock. If such a path exists, verify that all timing arcs along it are enabled and that rising and falling clock edges can propagate.

TIMING-37: Bus Skew Constraint Applied on Signals with Fanout

Message

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 prevent potential logic reconvergence inside the destination clock domain, which could lead to hardware failures, avoid fanout on clock domain crossing paths. A signal can only fan out within the destination clock domain after it passes through synchronization logic.

Resolution

Review the design and clock domain crossing paths to remove fanout within asynchronous clock domain crossing paths.

TIMING-38: Bus Skew Constraint Applied on Multiple Clocks

Message

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 accurate bus skew timing analysis, ensure that the sequential elements specified in the `set_bus_skew` constraint are driven by only one source clock and one destination clock. Involving multiple clocks can produce unreliable timing results.

Resolution

Review the clock domain crossing to ensure that all source elements and all destination elements are reached by only one clock. If multiple clocks reach the source or destination registers, define them as logically or physically exclusive.

TIMING-39: Invalid Bus Skew Constraint on Paths that have Too Many Levels of Logic

Message

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 prevent hardware failures from logic reconvergence inside the destination clock domain, there should be no combinational logic on clock domain crossing paths. Each register in the source clock domain should drive only one register in the destination clock domain, with no logic in between.

Resolution

Review the design and remove any combinational logic from the asynchronous clock domain crossing path.

TIMING-40: Max Skew Violation between OSERDESE3 CLK and CLKDIV Pins Crossing SLRs

Message

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 strict skew requirement that must not be exceeded. Placing the clock sources for CLK and CLKDIV in a different SLR than the OSERDESE3 can increase clock path delay and cause max skew violations.

Resolution

Review the clocking topology for the OSERDESE3 and ensure that both clock sources for the CLK and CLKDIV pins are placed inside the same SLR as the OSERDESE3 instance.

TIMING-41: Invalid Forwarded Clock Defined on an Internal Pin

Message

The forwarded clock <clock_group> is defined on the pin <netlist_element> instead of the port <netlist_element>.

Description

A forwarded clock is defined on a leaf pin connected to the output port instead of the output port itself. For proper I/O timing computation, the forwarded clock must be defined on the output port.

Resolution

Review the generated clock constraint and move the forwarded clock definition from the internal leaf pin to the output port.

TIMING-42: Path Segmentation Detected in the Clock Tree

Message

The <message_string> delay constraint position <message_string> is blocking the propagation of clock <clock_group> on pin <netlist_element>.

Description

Path segmentation is detected on the clock tree due to an invalid startpoint or endpoint in a min or max delay timing constraint. When this occurs, the tool must disable the timing arc to the invalid pin, which blocks clock propagation. This results in inaccurate timing signoff and can cause hardware failure.

Resolution

Review the min or max delay constraint applied to the clock tree and specify only valid startpoints and endpoints. For more information, see Path Segmentation in *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-43: Min Period or Min Pulse Width Violation on Gigabit Transceiver (GT)

Message

The GT pin `<instance/pin>` has MIN_PERIOD or MIN_PULSE_WIDTH violations. The Power Analysis Report is inaccurate for the GT instance.

Description

Minimum period checks on a GT clock pin ensure that the clock driving the GT instance does not exceed the maximum frequency tolerated by the hardware inside the primitive. Violations can cause hardware failure and inaccurate power analysis.

Resolution

Check the AC and DC characteristics datasheet for the relevant device family to determine the maximum permitted frequency for this primitive pin. Because this is a silicon-level limitation, reduce the frequency to eliminate the violation. Refer to this blog for more information.

TIMING-44: Unreasonable User Intra-Clock Uncertainty

Message

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

Defining an excessively high user intra-clock uncertainty impacts timing closure, compile time, and QoR. It can also increase power consumption and prevent timing closure.

Resolution

Review the user intra-clock uncertainty and reduce it to the minimum value needed. Avoid overconstraining beyond 0.5 ns. Overconstraining can increase power consumption and runtime. For more information, see Overconstraining the Design in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-45: Unreasonable User Inter-Clock Uncertainty

Message

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

Defining an excessively high user inter-clock uncertainty between two clocks impacts timing closure, compile time, and QoR. It can also increase power consumption and prevent timing closure.

Resolution

Review the user inter-clock uncertainty and reduce it to the minimum value needed. Avoid overconstraining beyond 0.5 ns. For more information, see *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-46: Multicycle Path with Tied CE Pins

Message

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, and the CE pin is tied to VCC. Because there is a direct datapath connection between the registers, the data on this path is captured using a single-cycle path requirement. The multicycle path defined for the path does not match hardware behavior and can cause hardware failure.

Resolution

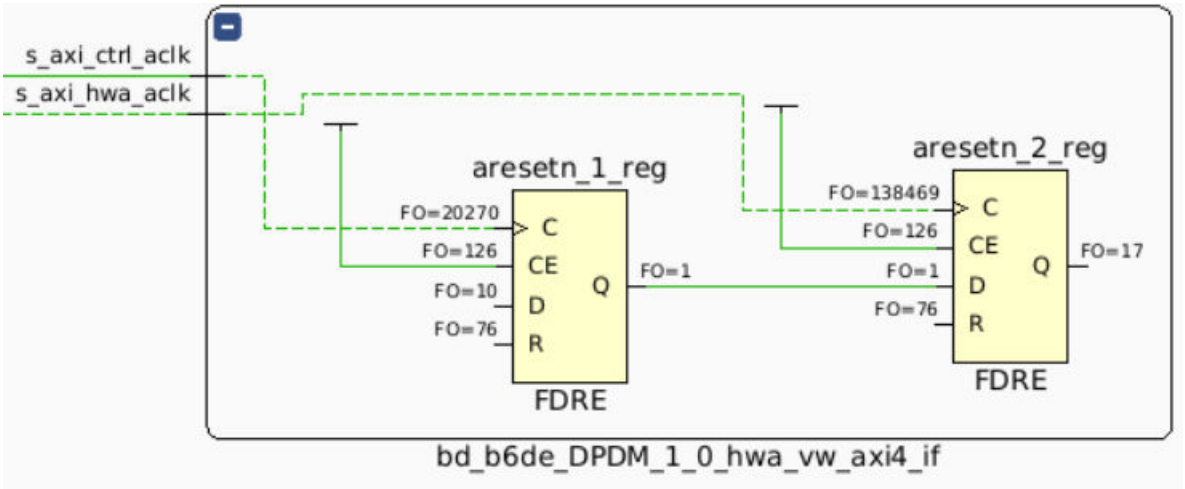
Review the path and associated timing constraint. If the CE pins are intended to be tied, remove the multicycle path. If the path is intended to be multicycle, drive the CE pin of the source and/or destination register with a dynamic signal that toggles according to the defined multicycle constraint.

Example

The following constraint is applied to the datapath between the two registers shown in the schematic, where the CE pin of the destination register is tied 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 235: **Multicycle Path with Tied CE Pins Example**



Verify whether the data changes at every clock edge outside the scope of the multicycle path. If it does, add the necessary logic connected to the CE pin.

The following example shows a flop-to-flop path with proper clock enable logic tied to the CE pin. This configuration enables the flop on alternating clock cycles:

```
set_multicycle_path 2 -setup
-from [get_pins data0_reg/C]
-to [get_pins data1_reg/D]
```

Figure 236: **Flop-to-Flop Path**

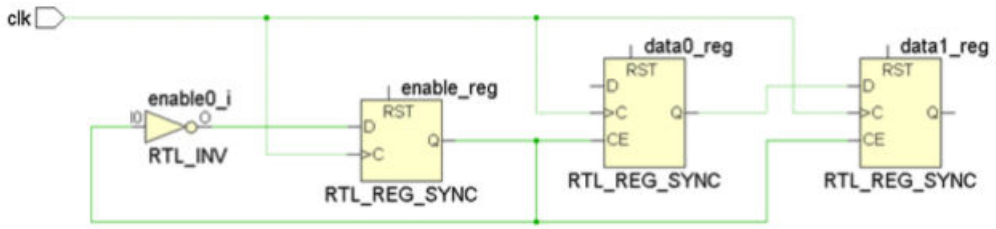
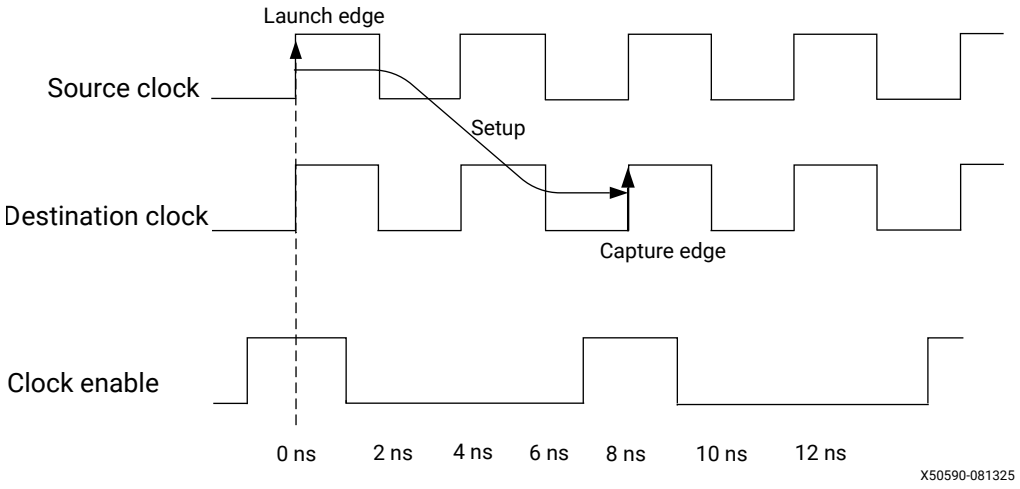


Figure 237: **Timing Diagram**



TIMING-47: False Path, Asynchronous Clock Group or Max Delay Datapath Only Constraint between Synchronous Clocks

Message

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

Do not apply a `set_false_path`, `set_clock_group -asynchronous`, or `set_max_delay -datapath_only` constraint to a synchronous clock domain crossing. Such constraints prevent the paths from being properly timed, resulting in inaccurate sign-off timing and potential hardware failure.

Resolution

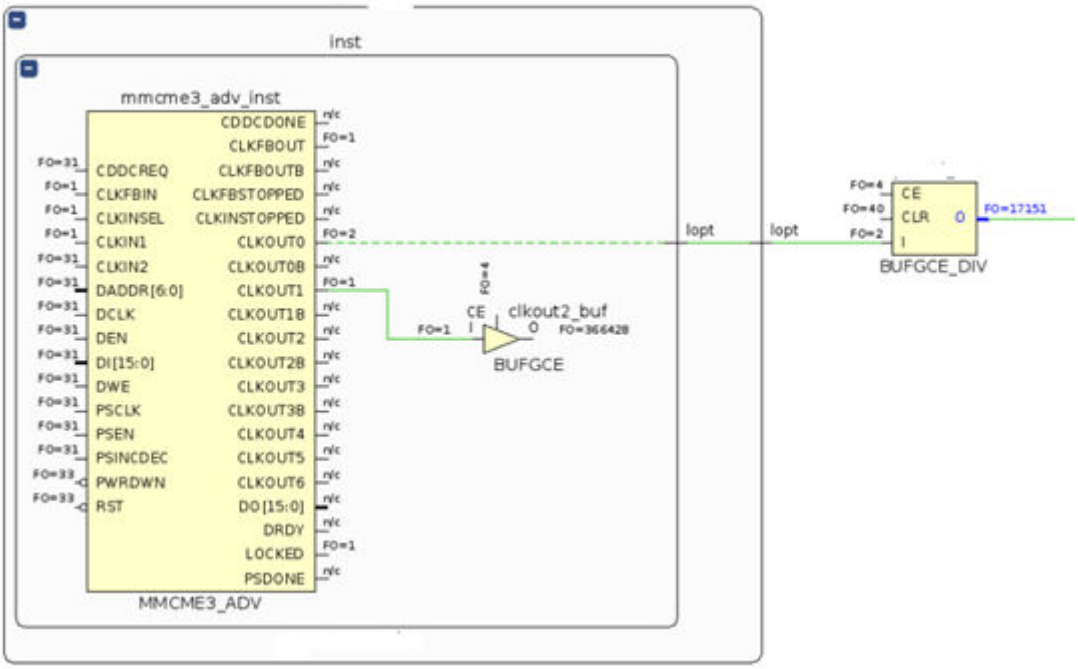
Remove the `set_false_path`, `set_clock_group-asynchronous`, or `set_max_delay -datapath_only` constraints between the synchronous clocks. If the clocks are truly asynchronous, add an asynchronous clock constraint and implement proper synchronization circuitry for the asynchronous clock domain crossing. To determine whether a clock domain crossing is synchronous or asynchronous, use the clock interaction report.

Example

In this example, two MMCM outputs generate the clocks. The BUFGCE_DIV output produces a divisional clock, `clk1`, whose master clock comes from `CLKOUT0`. `CLKOUT1` produces a generated clock, `clk2`. If you add the following constraint, Vivado issues a TIMING-47 warning:

```
set_clock_groups -asynchronous
-group [get_clocks clk1]
-group [get_clocks clk2]
```

Figure 238: **False Path, Asynchronous Clock Group, or Max Delay Datapath Only Constraint between Synchronous Clocks**



Because both `clk1` and `clk2` originate from the same MMCM, they are synchronous. Data crossing between these domains is under synchronous CDC and does not require `set_clock_groups -asynchronous`.

TIMING-48: Max Delay Datapath Only Constraint on Latch Input

Message

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 must not be applied to a latch input because it alters the path requirement and can cause unrealistic latch time borrowing that does not reflect actual hardware behavior. This mismatch can lead to hardware failure.

Resolution

Review the timing constraint and compare it against the intended hardware behavior. Remove the `set_max_delay -datapath_only` constraint from the latch input if it does not align with the expected hardware operation.

TIMING-49: Unsafe Enable or Reset Topology from Parallel BUFGCE_DIV

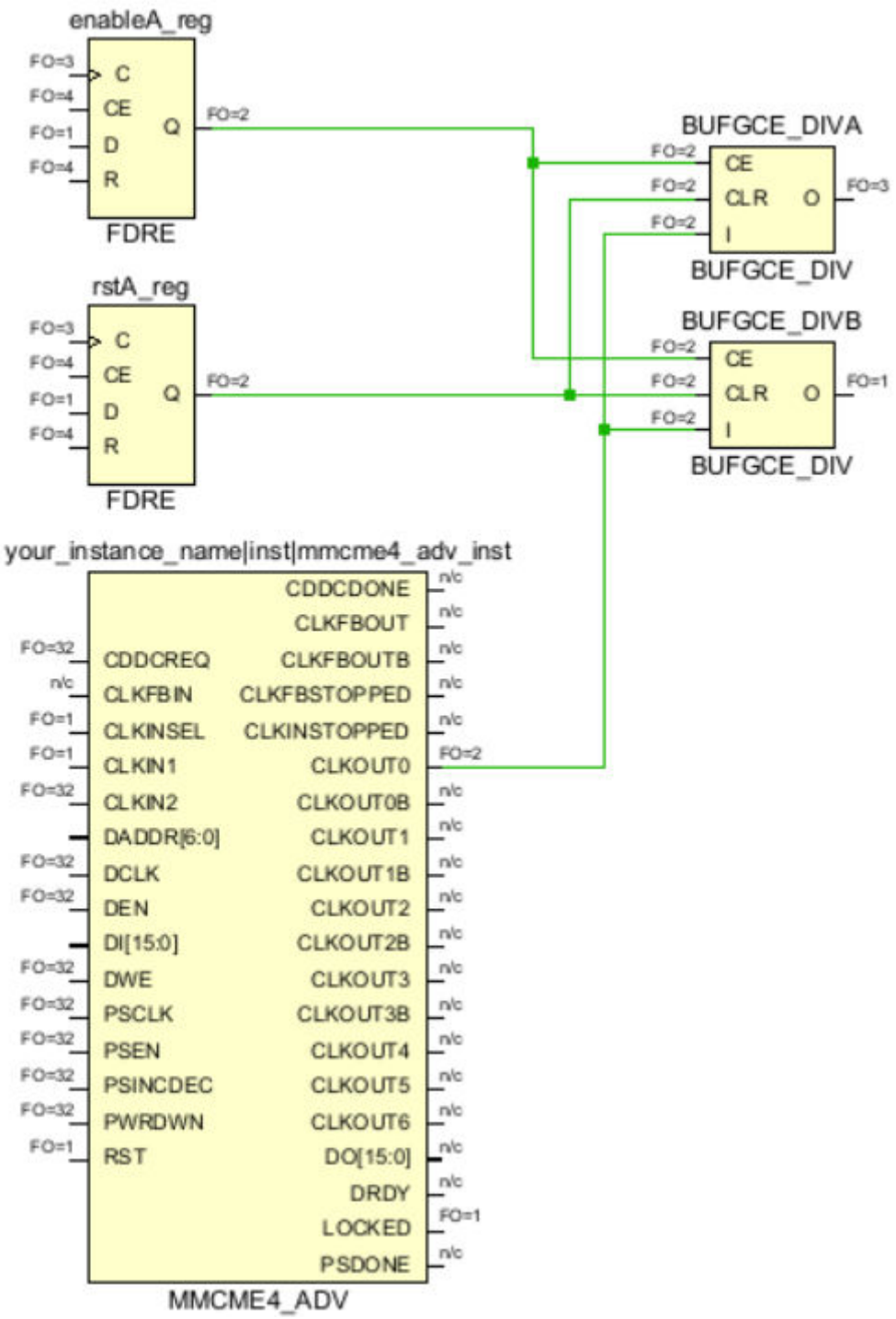
Message

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

When BUFGCE_DIV buffers have `BUFGCE_DIVIDE` set to a value greater than 1, they must share common control signals to avoid phase ambiguity caused by differences in the internal state of each buffer. Without common CE and CLR signals, the buffers can reset on different clock cycles, creating an unknown clock relationship between their outputs. This situation can lead to hardware failure. TIMING-49 is not limited to parallel BUFGCE_DIV buffers driven by the same clock modifying block (CMB) and applies to all synchronous topologies that must be safely timed.

Figure 239: **Correct Implementation of BUFGCE_DIV Buffers**



Resolution

Tie the CLR pins of both parallel BUFGCE_DIV buffers to the same signal or ensure they are driven by the same logic. Use safe clock startup reset circuitry for consistent operation in hardware. This can be enabled in the Clocking Wizard IP. For more details, refer to Synchronous CDC in the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-50: Unrealistic Path Requirement between Same-Level Latches

Message

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 a conservative 0 ns requirement between latches that share the same sensitivity level. Because the second latch in the path can borrow time, this topology increases the complexity and runtime of timing analysis and implementation. This configuration is uncommon, as cascaded latches are typically designed with opposite sensitivity levels to prevent data from crossing multiple latches in the same clock cycle.

Resolution

Review the latch topology. If this configuration is intentional, verify the expected hardware behavior and apply a multicycle path constraint between the cascaded latches to increase the path requirement to a realistic value.

TIMING-51: No Common Phase between Related Clocks from Parallel CMBs

Message

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 a clocking topology uses clocks derived from parallel MMCMs or PLLs, the clock domain crossing is only safe to time if the clock divider (`DIVCLK_DIVIDE`) of each CMB is set to 1. Any other value removes the phase relationship between the parallel CMBs, making the clock domain crossing asynchronous.

Resolution

If the clock domain crossing between the parallel PLLs is intended to be synchronous, set DIVCLK_DIVIDE to 1 on each CMB. If the crossing is intended to be asynchronous, update the timing constraints to reflect an asynchronous clock domain crossing and add the appropriate synchronization circuitry in the destination clock domain. For more details, see *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-52: No Common Phase between Related Clocks from Spread Spectrum MMCM

Message

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

When spread spectrum mode is enabled, the VCO frequency of the MMCM changes over time. These variations make it unsafe to perform timing analysis between two different outputs of the same MMCM, as the phase relationship between them is not constant.

Resolution

Avoid timing between different outputs of the MMCM when spread spectrum mode is enabled. If interaction between these clock domains is required, use asynchronous clock domain crossing techniques and add the appropriate synchronization logic.

TIMING-53: No Common Phase between Related Clocks from DPLL

Message

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 these devices, it is safe to time between the DPLL input clock and one of its output clocks only 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. The clock domain crossing between them must be considered asynchronous, and any path in the crossing must include synchronization logic 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 correct deskew configuration. If the clocks are asynchronous, ensure that the crossing has proper synchronization circuitry in the destination clock domain.

TIMING-54: Scoped False Path, Clock Group, or Max Delay Datapath Only Constraint between Clocks

Message

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. Defining a scoped timing constraint such as `set_clock_groups`, `set_false_path`, or `set_max_delay -datapath_only` between clocks impacts sign-off timing globally for the entire design. This can cause confusion and result in unexpected timing coverage beyond the hierarchical scope. Define these constraints at the top level to ensure the intended behavior.

Resolution

Review the timing constraint and move it to the top level if it is meant to apply to the entire design. If the constraint is only intended to apply under a specific instance, adjust the design topology or the constraint itself to achieve the desired coverage. For more details, see Synchronous CDC in *Vivado Design Suite User Guide: Using Constraints* (UG903).

TIMING-55: Multiple Clocks Reaching a CMB Deskew Pin

Message

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 propagate to the deskew pin of the MMCM, XPLL, or DPLL. If multiple clocks reach the deskew pin, sign-off timing becomes inaccurate and the design is at risk of hardware failure.

Resolution

Review the clocking topology and timing constraints to ensure that only one clock propagates to the MMCM, XPLL, or DPLL deskew pin.

TIMING-56: Missing Logically or Physically Excluded Clock Groups Constraint

Message

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, these clocks cannot be active on hardware at the same time. To align static timing analysis with hardware behavior, provide physically or logically exclusive clock group constraints between these clocks. Without these constraints, the tool times some clock pairs that cannot exist simultaneously in hardware.

Resolution

Provide the appropriate physically or logically exclusive clock group constraint between the clocks generated on the source pin based on the clock tree topology. For more information, see Constraining Exclusive Clock Groups in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).

TIMING-57: Unsupported Configuration with PHASESHIFT_MODE and Digital Deskew

Message

The clock modifying block `<netlist_element>` is configured for digital deskew and has `PHASESHIFT_MODE=WAVEFORM`. This combination is unsupported, and timing analysis is 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, or 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. If the element is set to PHASESHIFT_MODE=WAVEFORM, the timer cannot honor waveform modeling and instead enforces latency modeling. This does not affect sign-off timing accuracy because the latency model is applied automatically.

Resolution

Change the pin phase shift modeling to PHASESHIFT_MODE=LATENCY. Review any existing timing exceptions to ensure they still match the original design intent when switching from WAVEFORM to LATENCY 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 gives the place and route tools more flexibility to place the RAM optimally and allows the register to be placed in the fabric instead of the RAM to optimize the critical path.

The synthesis tool can easily infer output registers. These registers must either have a synchronous reset or no reset.

Verilog Code Example

Figure 240: **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 241: **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 242: **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 243: **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 exceeds the maximum width supported by the DSP slice in the target architecture, require additional pipeline stages to achieve the maximum operating frequency of the DSP slice. The required number of stages depends on the multiplier width.

Adding extra stages to the output of wide multipliers in the RTL allows the synthesis tool to move them to optimal positions, making recoding straightforward.

Verilog Code Example

Figure 244: **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 245: **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 246: **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;
```

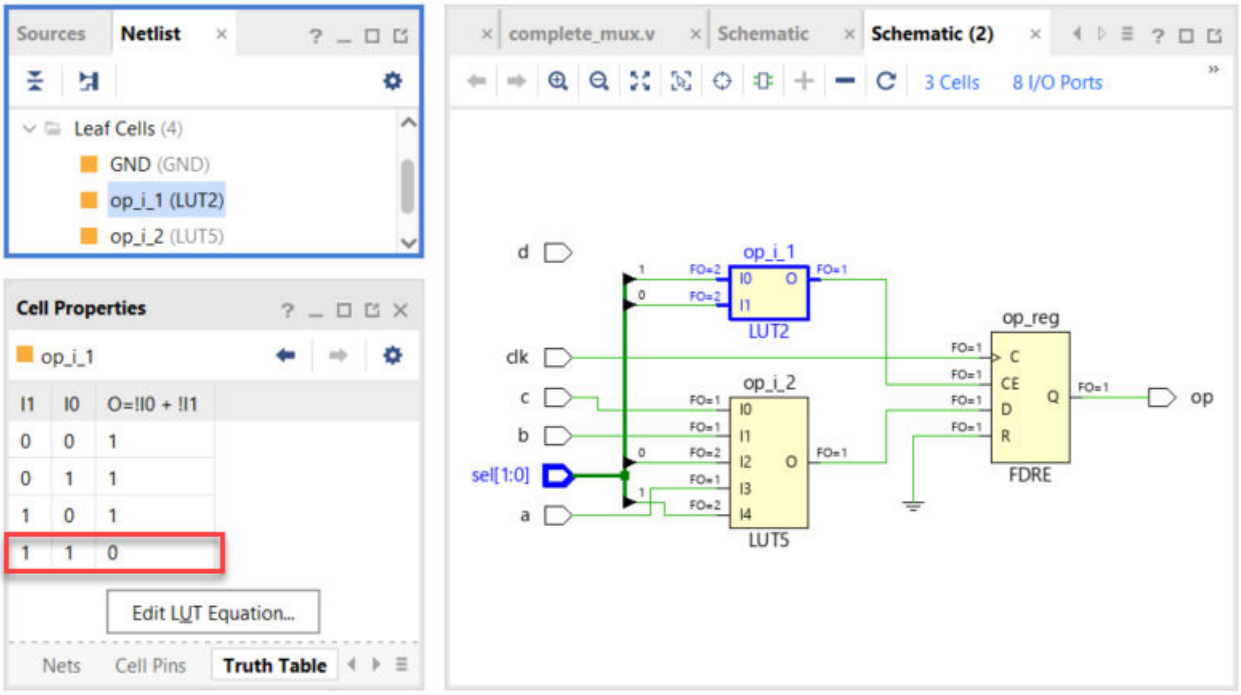
Figure 247: **After**

```
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 retains its previous state, which infers a clock enable. Define all states in the case statement to avoid increasing control sets. For any additional states, assign the output to a previously used signal, a constant 1, or a constant 0 to prevent adding unnecessary logic.

Figure 248: **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 with an array depth significantly over 64 bits are better inferred into block RAM instead of distributed RAM. The synthesis tool attempts this by default but might fail due to coding style or constraint restrictions.

The most common reason block RAM is not inferred is a missing output register, because block RAM supports only synchronous reads. Distributed RAM does not have this limitation. Another reason is use of an array read method or a `ROM_STYLE` attribute that forces distributed RAM inference.

By making minor coding adjustments, you can improve LUT utilization, timing, and, where applicable, congestion.

Verilog Code Example

Figure 249: **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 250: **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 251: **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 252: **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.

Custom QoR Suggestions

You can create Custom QoR Suggestions as part of the QoR suggestion flow. Using the same method as custom DRCs, reference a Tcl proc as a new suggestion. After you register the suggestion, you can reference it in the `report_qor_suggestions` command. After the suggestion is generated, the flow works exactly the same as for other suggestions.

New QoR checks you develop for this flow are not generated by default. You need to call them using the following command:

```
report_qor_suggestions -checks [get_qor_checks <check ID>]
```

RQS_AMD_NELTIST-1: Extract Registers from SRLs Driven by LUTs

This check generates the `SRL_STAGES_TO_REG_INPUT` properties on SRL cells that are driven by LUTs (for example, LUT/O -> SRL/D). After you extract registers from the SRL, the register site next to the LUT can be used, which can make timing paths easier to meet.

The check considers the following:

- Whether an SRL is static or variable. You only extract registers from static SRLs.
- The net driving the SRL D pin must have a fanout of 1.
- When the driving LUT is LUT-combined, and one of the LUTs drives an FDC* primitive, register extraction is not performed due to packing limitations.

Use this suggestion in fast designs to help prepare the netlist for higher performance. However, performance drops if too many registers are introduced and register utilization becomes a bottleneck, restricting optimizations later in the tool flow.

Property	Value
APPLICABLE_FOR	`place_design`
AUTO	`1`

RQS_AMD_NETLIST-11: GT Floorplan

This check creates Pblock constraints for GT-related logic. It checks for advanced primitive usage, identifies related GT connections, and creates a Pblock sized appropriately for those connections. This helps guide the placer. It is recommended to use this suggestion with `RQS_AMD_NETLIST-12`.

The check can create three sizes of Pblocks:

- A single clock region
- A three clock region horizontal row
- A three clock region side of an SLR

The Pblocks are based on the hierarchical cell connected to the GT output clocks.

Property	Value
APPLICABLE_FOR	`place_design`
AUTO	`1`

RQS_AMD_NETLIST-12: GT Floorplan Synthesis Constraints

When you create a hierarchical floorplan, preserve hierarchy at synthesis to prevent logic optimizations that can cause timing issues later in the flow. For example, if you optimize GT control logic that was coded to be replicated on each side of the device and in different SLRs, the optimization can result in that logic being shared between all GTs. This sharing can create timing problems later in the flow. Using `KEEP_HIERARCHY` during synthesis prevents this optimization.

This suggestion selects the same hierarchies chosen for floorplanning when you use `RQS_AMD_NETLIST-11`. It is strongly recommended to apply `RQS_AMD_NETLIST-12` before using `RQS_AMD_NELTIST-11`.

Property	Value
APPLICABLE_FOR	`synth_design`
AUTO	`1`

Appendix D

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.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, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

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, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

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 Tcl Command Reference Guide* (UG835)
2. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
3. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
4. *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894)
5. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)
6. *Vivado Design Suite User Guide: Synthesis* (UG901)
7. *Vivado Design Suite User Guide: Using Constraints* (UG903)
8. *Vivado Design Suite User Guide: Implementation* (UG904)
9. *Vivado Design Suite User Guide: Power Analysis and Optimization* (UG907)
10. *Vivado Design Suite Properties Reference Guide* (UG912)
11. *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)
12. *UltraFast Design Methodology Timing Closure Quick Reference Guide* (UG1292)
13. *Versal Adaptive SoC System Integration and Validation Methodology Guide* (UG1388)
14. 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 the trainings in this link to explore related training resources:

1. Designing FPGAs Using the Vivado Design Suite 1 (FPGA-VDES1)
2. Designing FPGAs Using the Vivado Design Suite 2 (FPGA-VDES2)
3. Designing FPGAs Using the Vivado Design Suite 3 (FPGA-VDES3)

4. Designing FPGAs Using the Vivado Design Suite 4 (FPGA-VDES4)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/10/2025 Version 2025.2	
N/A	General updates.
Report RAM Utilization	Removed a bullet.
Running the RAM Utilization Report	Removed the last command.
Report SLR Crossing	Added new section and subsections.
Appendix C: Custom QoR Suggestions	Added new section and subsections.
05/29/2025 Version 2025.1	
Clock Uncertainty	Added an important note.
Results Name Field	Updated the paragraph after the Report Utilization figure.
Show SLR Utilization	Updated the section.
Show Hierarchical Information with Customized Options	Updated the code.
Auto Termination of Runs	Changed MIN_RQA_SCORE property to 3 to MIN_RQA_SCORE property to 1.
Category 3: Physical	Updated the section.

Please Read: Important Legal Notices

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