Vivado Design Suite Tutorial

Design Analysis and Closure Techniques

Vivado Design Suite

UG938 (v2025.2) December 5, 2025

AMD

Table of Contents

Tutorial Overview

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. Topics in this document that apply to this design process include:
  - Chapter 2: Using Report QoR Suggestions and Report QoR Assessment for Timing Closure
  - Chapter 3: Running ML Strategies
  - Chapter 4: Intelligent Design Runs

Introduction

This tutorial uses the AMD Vivado™ design rules checker (`report_drc`), clock domain crossing checker (`report_cdc`), and quality of results enhancer (`report_qor_suggestions`) to analyze example designs for issues, and shows you how to take corrective actions. It also outlines how to run ML strategies and intelligent design runs (IDRs).

Tutorial Description

Lab 1 discusses creating waivers for (clock domain crossing) CDC, methodology, and DRC (design rule check) violations.

Lab 2 is a guide to using the `report_qor_suggestions` (RQS) command.

***Note*:** The designs used in this tutorial are intended to exhibit issues for demonstration purposes, and should not be used as a reference for designs outside this tutorial.

Software Requirements

This tutorial requires that the 2023.1 AMD Vivado™ Design Suite software release or later is installed.

Lab 1 to Lab 3 requires installing AMD UltraScale+™ and lab 4 requires installing AMD UltraScale™ family.

For a complete list of system and software requirements, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

Locating Tutorial Design Files

1. Download the reference design files from the AMD website.
2. Extract the ZIP file contents into any write-accessible location.

   This tutorial refers to the location of the extracted ZIP file contents as `<Extract_Dir>`.

Note: To use the commands provided in this tutorial, please access the web version. The PDF format may introduce line breaks that affect command execution.

AMD

Chapter 1

Setting Waivers with the Vivado IDE

Introduction

In the AMD Vivado™ Design Suite, you can use the waiver mechanism to waive clock domain crossing (CDC), design rule check (DRC), or methodology check violations. After a violation is waived, it is no longer reported by the `report_cdc`, `report_drc`, or `report_methodology` commands. Waived checks are also filtered out from the mandatory DRCs run at the start of the implementation commands, such as `opt_design`, `place_design`, and `route_design`. For more information, see *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906).

IMPORTANT! *The content of the waiver is built with the objects that exist when the waiver is created. However, if an instance referenced inside a waiver is replicated by Vivado, the replicated instance is automatically added to the waiver and saved in subsequent checkpoints and XDC.*

This lab shows how to set waivers with the Vivado integrated design environment (IDE) using both menu commands and the Tcl Console. The lab focuses on CDC waivers, but the methods for waiving DRC and methodology violations are similar.

***Note*:** Due to encryption limitations with Vivado 2025.1, it is recommended to run this waiver tutorial using Vivado 2024.1 or an earlier version.

Step 1: Starting the Vivado IDE

This lab uses a Vivado design checkpoint (`.dcp` file) that is a snapshot of a design. When you launch the Vivado IDE using a design checkpoint, a subset of the Vivado IDE functionality is available.

TIP: *To launch the Vivado Tcl Shell on Windows, select **Start → All Programs → Xilinx Design Tools → Vivado <version> → Vivado <version> Tcl Shell**.*

1. From the command line or the Vivado Tcl Shell, change to the directory where the lab materials are stored:

```
cd <Extract_Dir>/Lab1
```

2. To start the Vivado IDE with the design checkpoint loaded, enter the following:

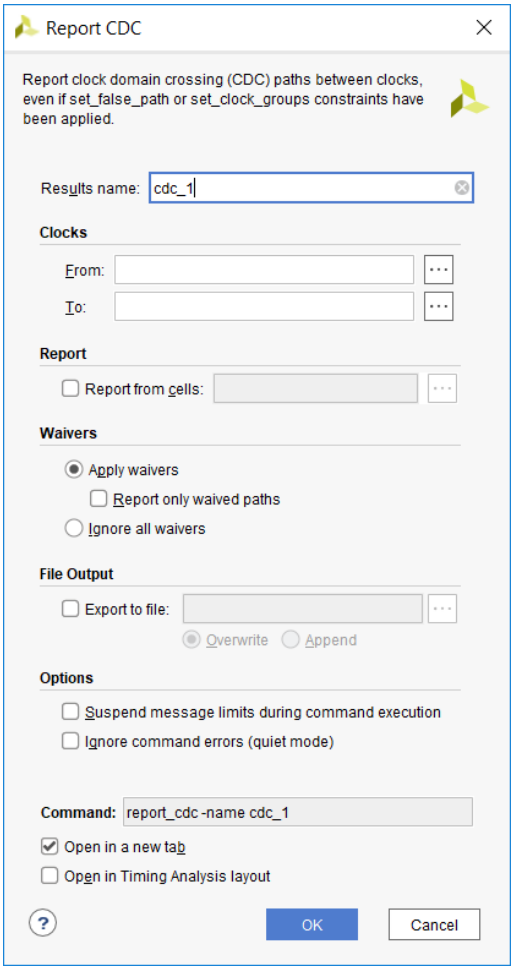
```
vivado my_ip_example_design_placed.dcp
```

TIP: *You can disregard the critical warnings about the unbounded GT locations.*

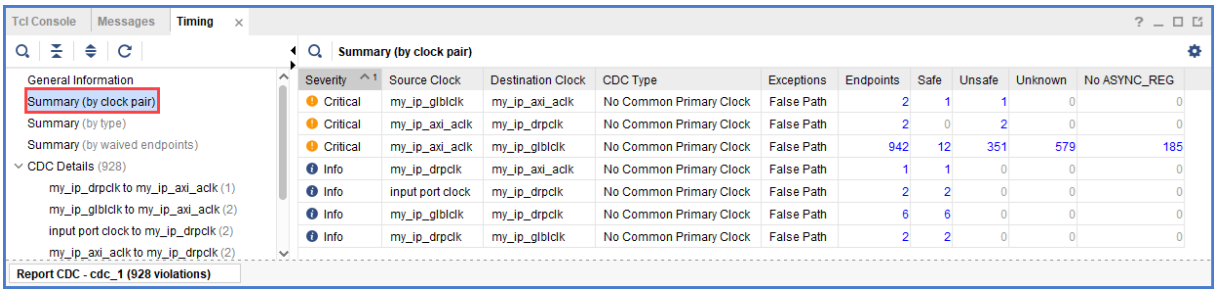
Step 2: Generating the CDC Report

In this step, you generate the CDC report to view the associated CDC violations.

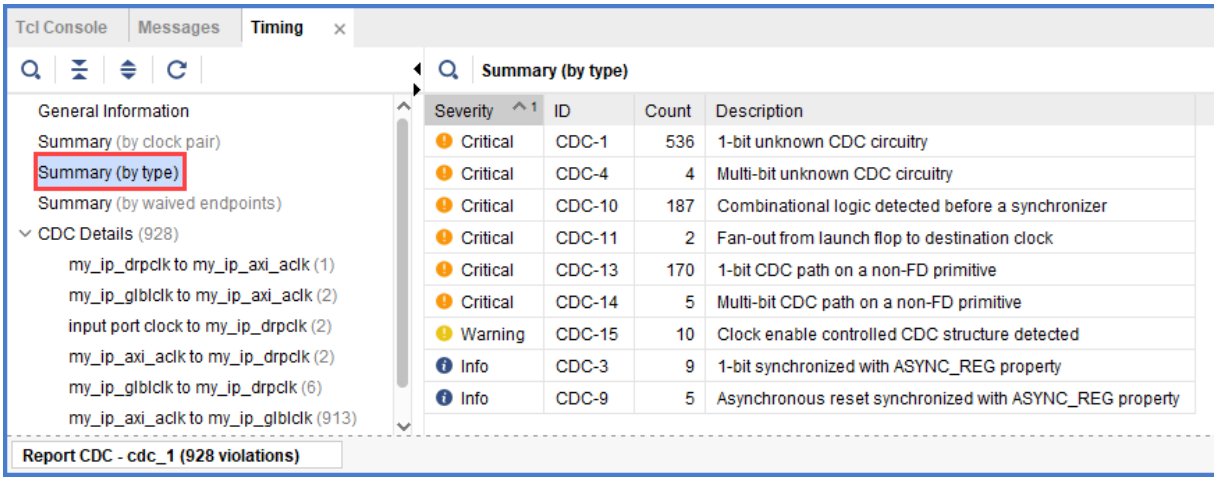
1. Select **Reports → Timing → Report CDC**.
2. In the Report CDC dialog box, leave the default settings as-is, and click **OK**.



The Summary (by clock pair) section of the CDC Report appears as follows.



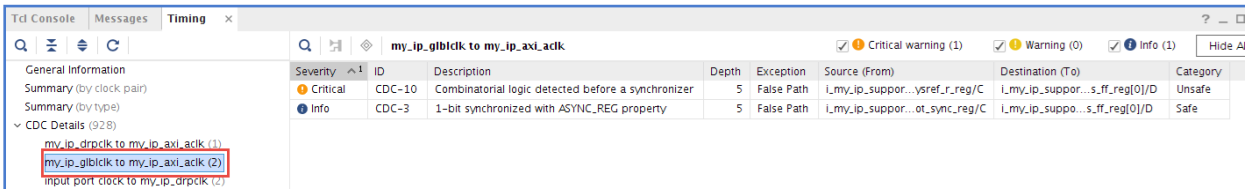
The Summary (by CDC type) section appears as follows.



Step 3: Waiving a Single CDC Violation

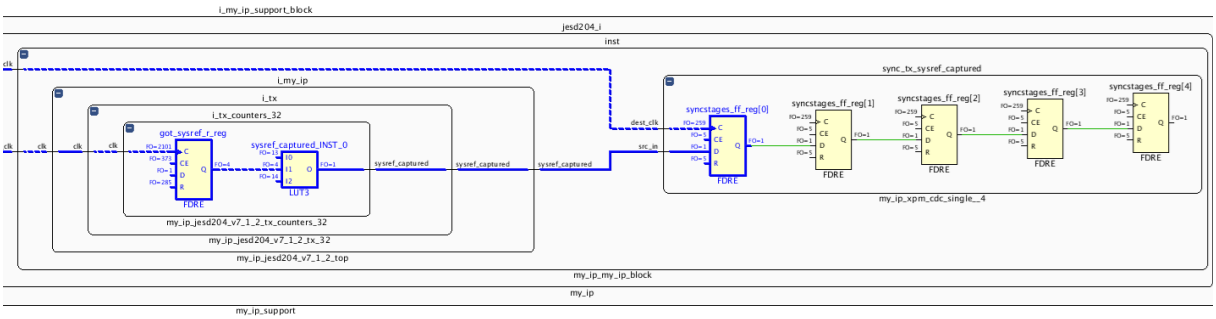
The `my_ip_glblclk` to `my_ip_axi_aclk` clock pair includes one Critical CDC-10 violation due to combinational logic on the CDC path. This step covers how to waive the CDC-10 violation.

Figure 1: **CDC-10 Violation**

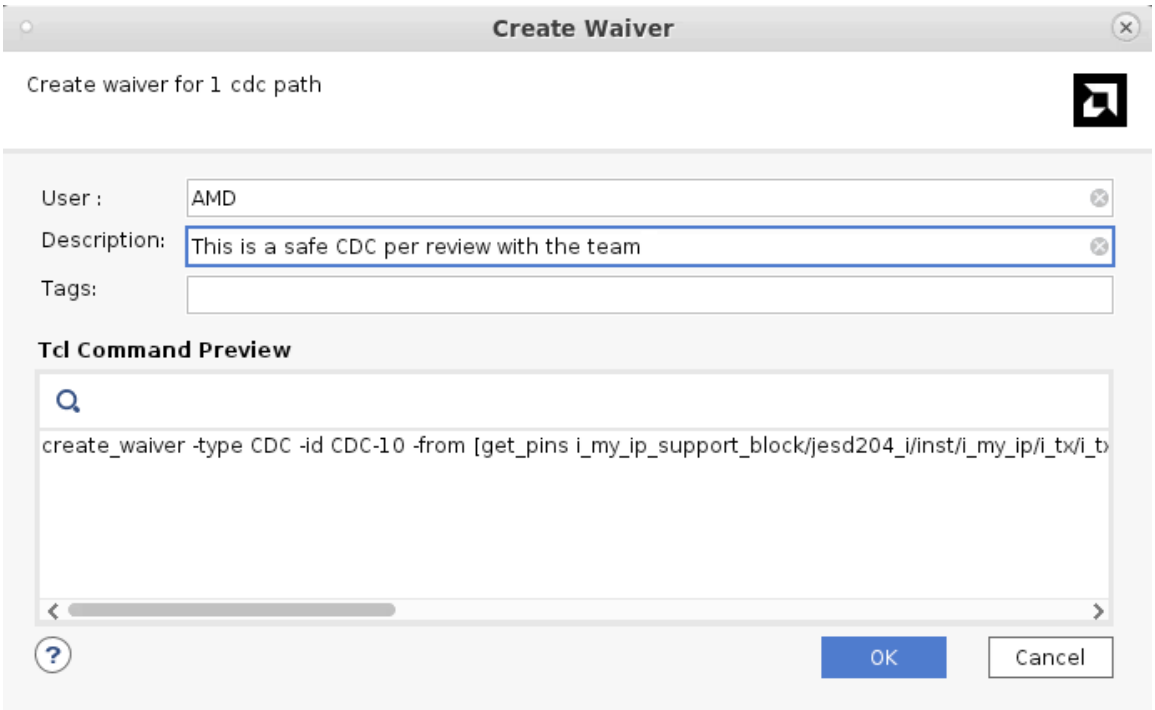


1. To view a schematic of the violation, select the CDC-10 row in the CDC Report, and click the Schematic toolbar button .

   Note: Alternatively, press **F4** to generate the schematic. However, using the toolbar button provides a more detailed schematic that includes all the levels of the downstream synchronizer.



2. To waive the violation, select the **CDC-10** row in the CDC Report, right-click, and select **Create Waiver**.
3. In the Create Waiver dialog box, enter a description, and click **OK**.

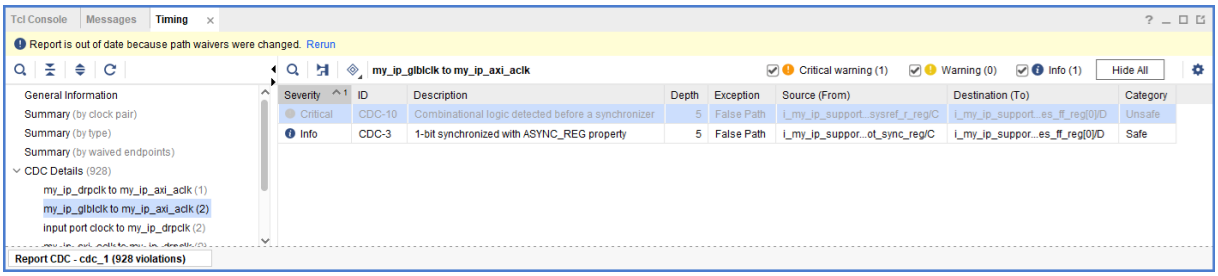


IMPORTANT! *A waiver tracks the date the waiver was added, the user that added the waiver, and a description of why the violation was waived. The date is automatically added by the system. The Tags field is an optional description or list of keywords that can be used for documentation purposes.*

4. After the waiver is created, check the CDC Report.

   To indicate that a waiver was created, the CDC-10 row is gray and disabled.

   Note: Rows are only disabled in the Report CDC result window from which the waivers were created.

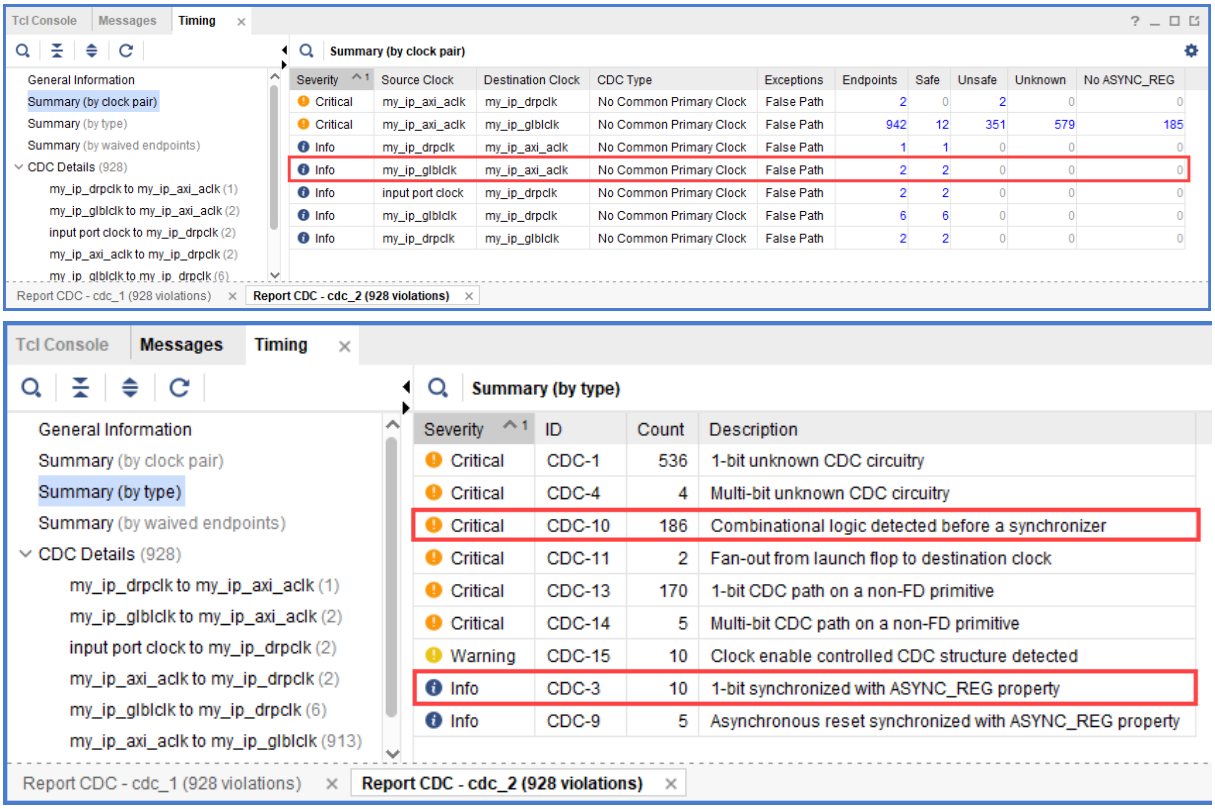


5. To see the impact of the CDC-10 waiver, select **Reports → Timing → Report CDC** to rerun Report CDC.

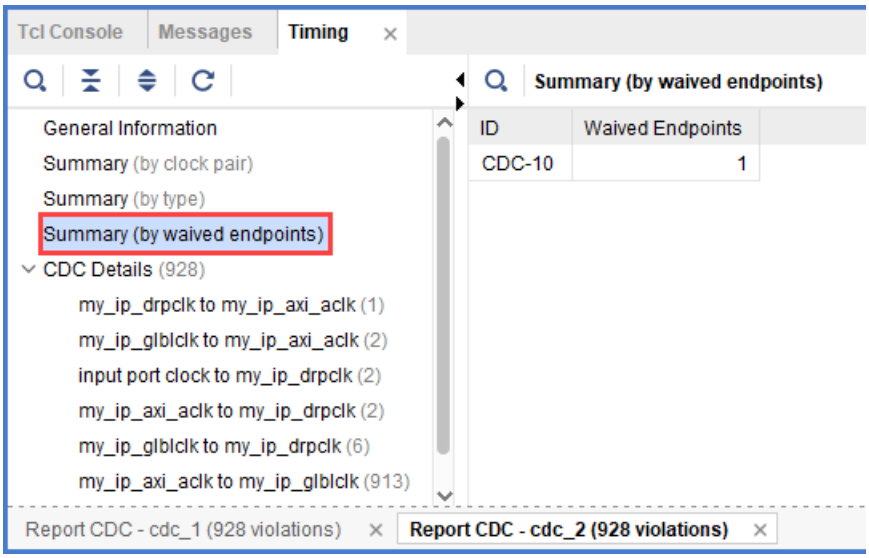
   ***Note*:** When a waiver is created or deleted, rerun Report CDC, Report DRC, or Report Methodology to see the updated results.

6. See the CDC Report to view the updated information.

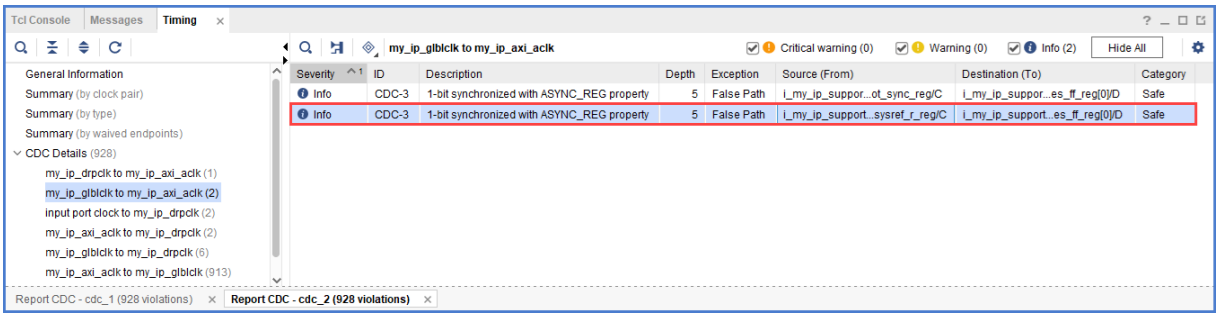
   The differences from the previous Summary by clock pair and Summary by type sections are highlighted in red in the following figures.



You can also view a summary with the list of waived endpoints.

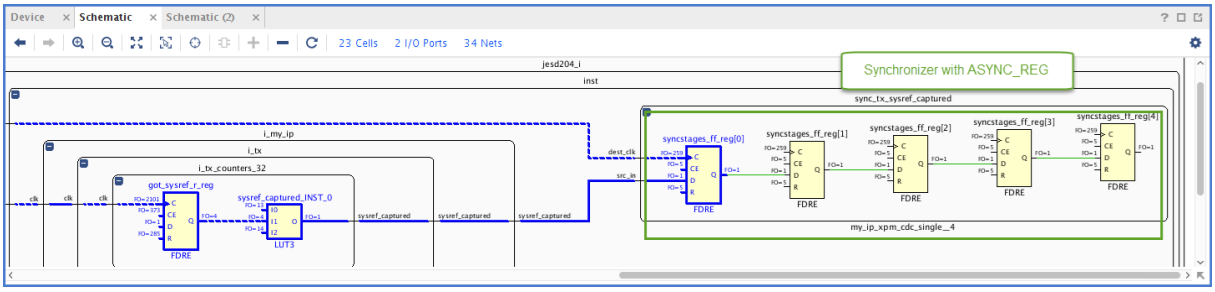


The detailed section for the `my_ip_glblclk` to `my_ip_axi_aclk` CDC shows that the Critical CDC-10 was replaced with an Info CDC-3.



7. Select the new CDC-3 row, and click the Schematic toolbar button  . If F4 is used to open the schematic, double-click the Q pin of the output register to expand the schematic to match what is shown in the following figure.

   The CDC path includes a 5-level synchronizer on the output of the selected destination register. This is the reason the CDC-10 was replaced with CDC-3 for this topology, as shown in the following figure.



IMPORTANT! *By default, Report CDC only reports a single violation per endpoint and per clock pair. When multiple violations apply to the same endpoint, only the violation with the highest precedence is reported. Because CDC-10 has a higher precedence than CDC-3, only CDC-10 is reported when both CDC-10 and CDC-3 apply to the same endpoint. For more information on CDC rules precedence, see CDC Rules Precedence in the Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906).*

TIP: *To report all of the CDC violations for each endpoint regardless of the precedence rules, use the command line option* `-all_checks_per_endpoint`. *However, unsafe rules are not reported on a register if at least one safe rule on the same register is detected.*

Step 4: Generating a Report for Waived Violations

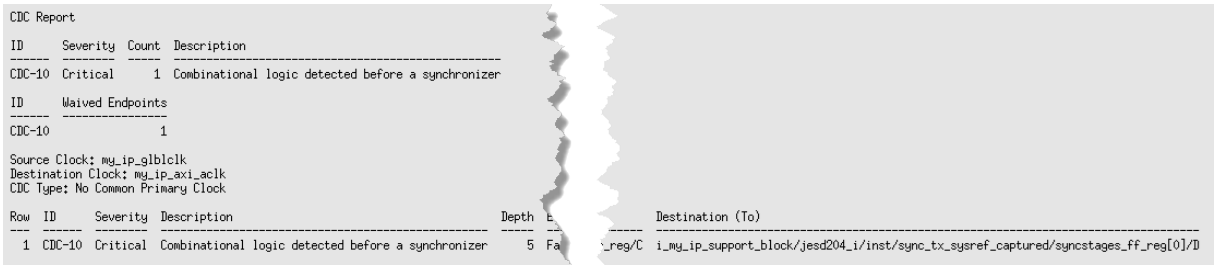
You can generate a report for the CDC, DRC, or methodology check violations that were waived. This step shows how to generate a report for waived CDC violations using the Tcl Console as well as the Vivado IDE menu commands.

Generating a Text Report for Waived Violations

1. In the Tcl Console, enter:

```
report_cdc -waived
```

2. In the CDC report, verify that a single CDC-10 violation is listed, because only one waiver was created.

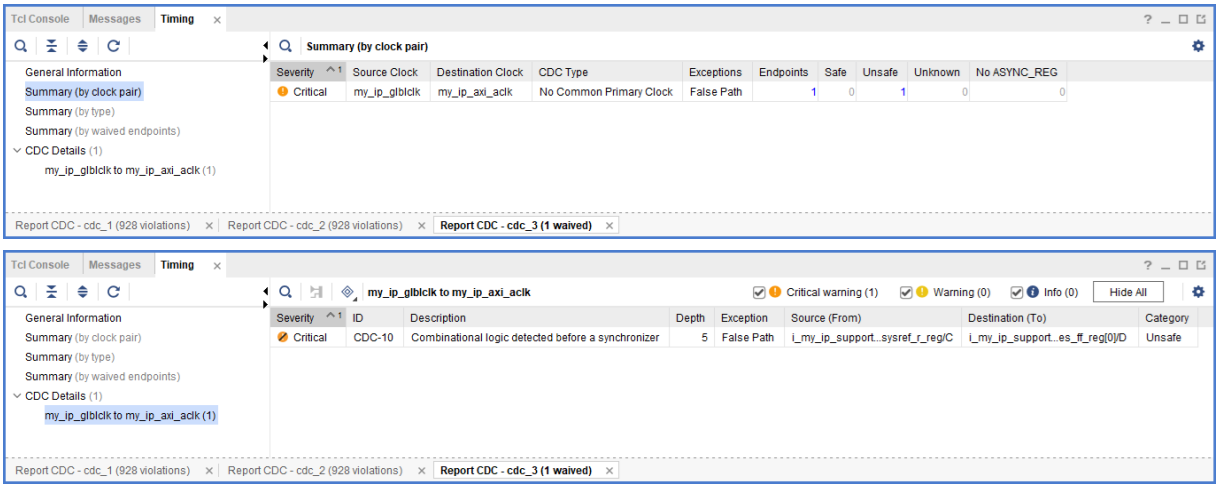


```
CDC Report

ID      Severity  Count  Description
------  --------  -----  ------------------------------------------------------------
CDC-10  Critical      1  Combinational logic detected before a synchronizer

ID      Waived Endpoints
------  ----------------
CDC-10                 1

Source Clock: my_ip_glblclk
Destination Clock: my_ip_axi_aclk
CDC Type: No Common Primary Clock

Row  ID      Severity  Description                                          Depth  E...  ...  Destination (To)
---  ------  --------  ---------------------------------------------------  -----  ---  -----  ----------------------------------------------------------------------------------------------------
  1  CDC-10  Critical  Combinational logic detected before a synchronizer      5  Fa...  ..._reg/C  i_my_ip_support_block/jesd204_i/inst/sync_tx_sysref_captured/syncstages_ff_reg[0]/D
```

Generating a Vivado IDE CDC Report for Waived Violations

1. Select **Reports → Timing → Report CDC**.
2. In the Report CDC dialog box, enable **Report only waived paths**, and click **OK**.
3. In the CDC Report, check the Summary (by clock pair) and CDC Details to verify that a single CDC-10 violation is listed.

Note: The icon next to the violation shows that the violation was waived .



Step 5: Generating a Text Report with Details for Waived Violations

In this step, you generate text reports with additional details, including a list of all of the rules and all of the violations regardless of the waivers.

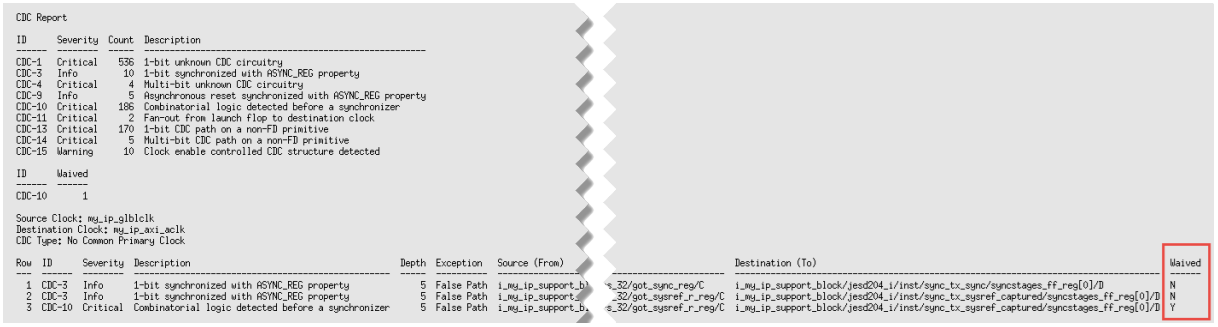
Generating a List of Rules with Waived Violations

1. In the Tcl Console, enter:

```
report_cdc -details -show_waiver
```

2. Verify that the `my_ip_glblclk` to `my_ip_axi_aclk` CDC-10 violation is waived and the two CDC-3 violations are not waived.

   Note: In the text report, all of the rules are reported, whether they were waived or not. The Waived column indicates the status of the rule.



Generating a List of All Violations Regardless of the Waivers

1. In the Tcl Console, enter:

```
report_cdc -no_waiver
```

2. In the text report, verify that the table matches the original report from Report CDC before the CDC-10 waiver was created.

```
CDC Report

Severity  Source Clock      Destination Clock  CDC Type                 Exceptions  Endpoints  Safe  Unsafe  Unknown  No ASYNC_REG
--------  ----------------  -----------------  -----------------------  ----------  ---------  ----  ------  -------  ------------
Critical  my_ip_glblclk     my_ip_axi_aclk     No Common Primary Clock  False Path          2     1       1        0              0
Critical  my_ip_axi_aclk    my_ip_drpclk       No Common Primary Clock  False Path          2     0       2        0              0
Critical  my_ip_axi_aclk    my_ip_glblclk      No Common Primary Clock  False Path        942    12     351      579            185
Info      my_ip_drpclk      my_ip_axi_aclk     No Common Primary Clock  False Path          1     1       0        0              0
Info      input port clock  my_ip_drpclk       No Common Primary Clock  False Path          2     2       0        0              0
Info      my_ip_glblclk     my_ip_drpclk       No Common Primary Clock  False Path          6     6       0        0              0
Info      my_ip_drpclk      my_ip_glblclk      No Common Primary Clock  False Path          2     2       0        0              0
```

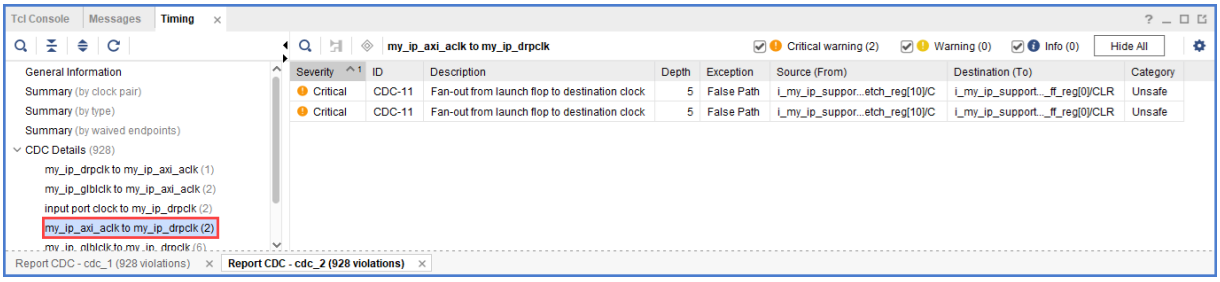
TIP: *You can also generate a list of all violations regardless of the waivers from the Vivado IDE. Select* ***Reports → Timing → Report CDC****. In the Report CDC dialog box, enable* ***Ignore all waivers****, and click* ***OK****.*

Step 6: Waiving Multiple CDC Violations

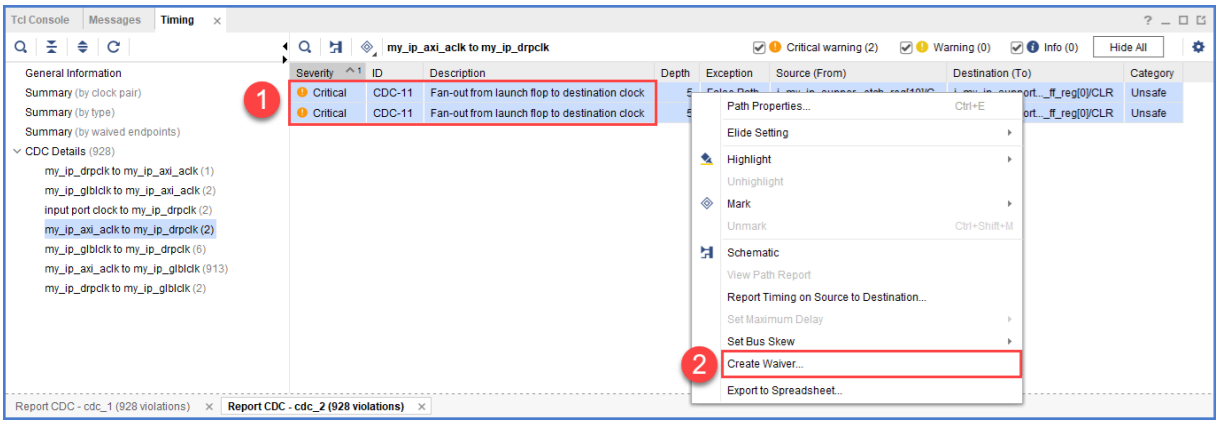
***Note*:** Before resuming step 6, go to `cdc_2` window.

The `my_ip_axi_aclk` to `my_ip_drpclk` CDC includes two critical CDC-11 violations. This step covers how to waive both CDC-11 violations simultaneously.

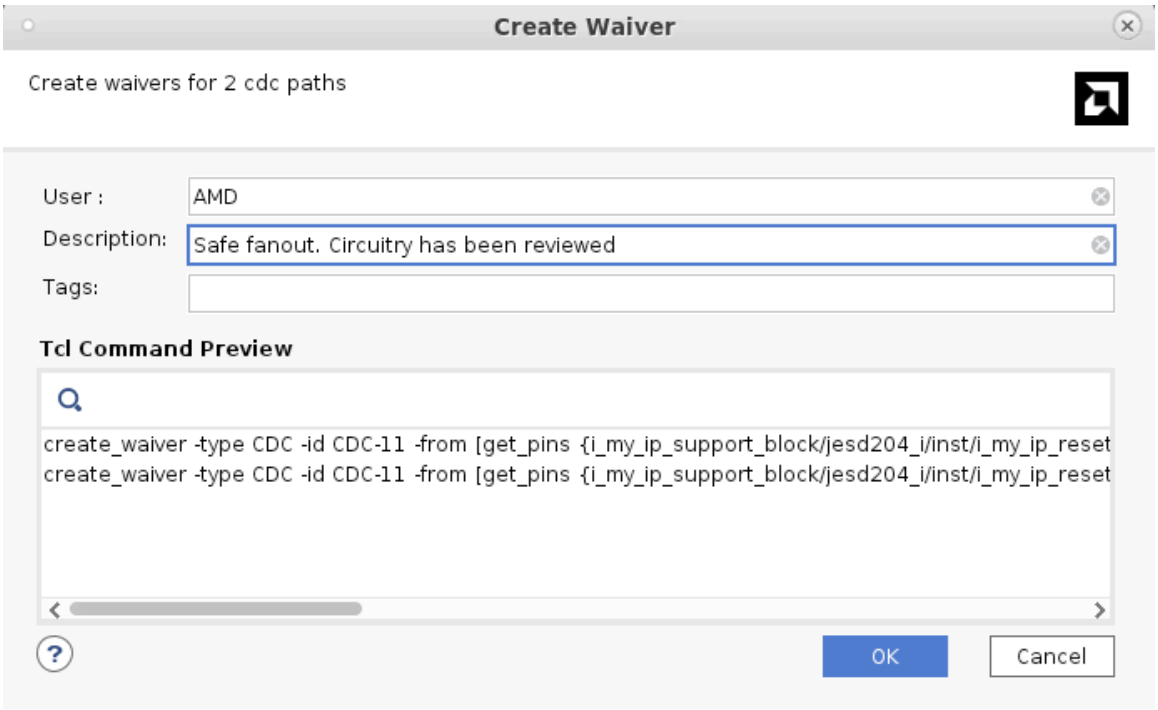
Figure 2: **CDC-11 Violations**



1. To waive the violations, select the **CDC-11** rows in the CDC Report, right-click, and select **Create Waiver**.

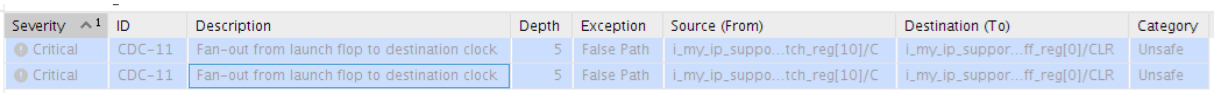


2. In the Create Waiver dialog box, enter a description, and click **OK**.



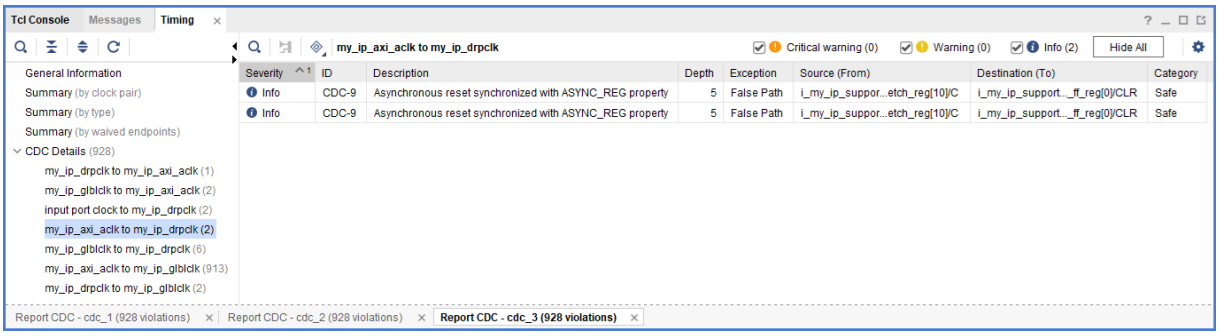
In the Timing Report, the two selected rows are disabled when the waivers are created.

***Note*:** One waiver is created for each selected row. In this example, two waivers are created.

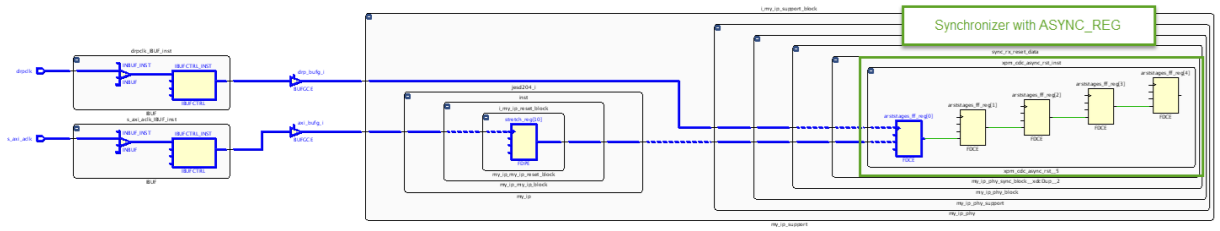


3. Select **Reports → Timing → Report CDC** to rerun Report CDC. In the Report CDC dialog box, make sure that *Report only waived paths* is unchecked, and click **OK**.
4. In the CDC Report, look at the `my_ip_axi_aclk` to `my_ip_drpclk` CDC.

   The two critical CDC-11 violations were replaced with two Info CDC-9 violations. Based on the CDC precedence rules, waiving CDC-11 unmasks CDC-9 for this circuit.

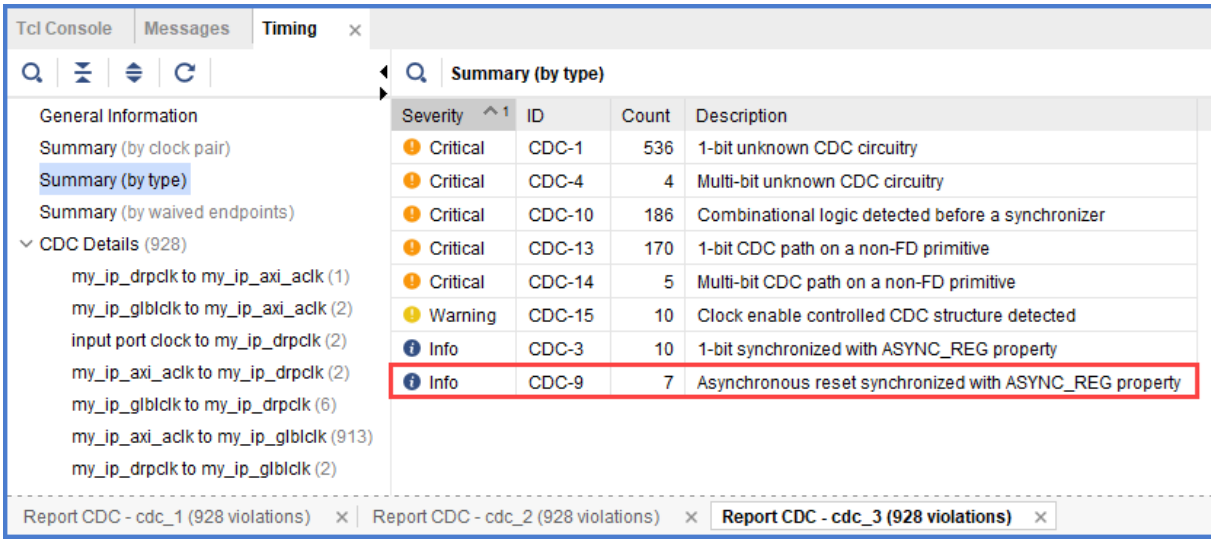


5. To view a schematic of the violation, select the CDC-9 row in the CDC Report, and click the Schematic toolbar button .
6. Verify that there is a 5-level synchronizer on the destination clock domain.



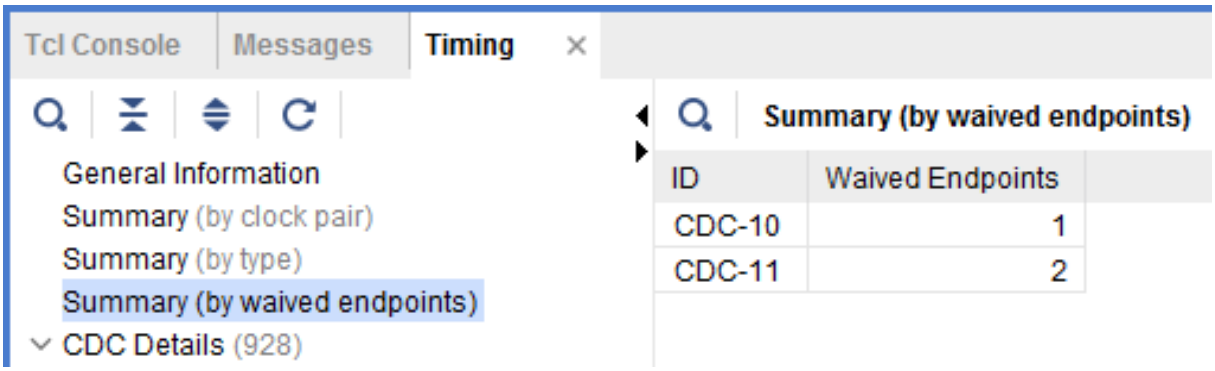
7. Compare the new Summary (by type) information with the information from the previous CDC Report.

   In the updated CDC Report, the two CDC-11 violations are no longer listed. Instead, there are two new CDC-9 violations.



8. Look at the Summary (by waived endpoints) information.

   In the updated CDC Report, there are three waived endpoints. This number is different from the number of waived violations (2), because CDC-11 is a multi-bit violation.

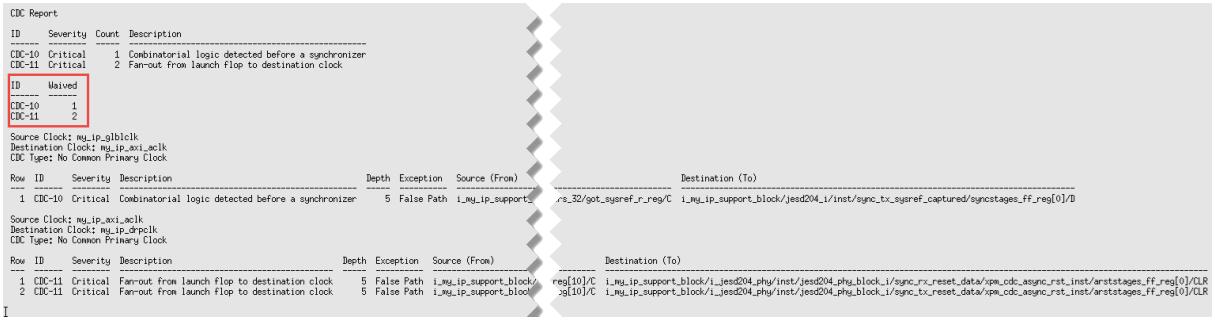


9. Generate different text reports and compare the results with previous reports.

   For example, you can run the following Tcl commands:

```
report_cdc -details
report_cdc -details -waived
report_cdc -details -show_waiver
report_cdc -details -no_waiver
```

   The following report was generated using the `report_cdc -details -waived` Tcl command and shows that three violations were waived.



Step 7: Exporting Waivers

In this step, you export waivers with the `write_waivers` Tcl command.

***Note*:** The XDC output file can be imported using the `read_xdc` or `source` Tcl commands.

1. To export the CDC waivers, enter: `write_waivers -type cdc waivers.xdc`.

   TIP: *Alternatively, because there are no DRC or methodology waivers, you can enter:*

   `write_waivers waivers.xdc` or `write_xdc -type waiver waivers.xdc`.

2. Open the `waivers.xdc` file to view the three waivers.

Note: The following example is reformatted to better show the different command line options.

```
create_waiver -type CDC -id {CDC-10} -user "AMD" \
  -desc "This is a safe CDC per review with the team" \
  -from [get_pins i_my_ip_support_block/jesd204_i/inst/i_my_ip/i_tx/
i_tx_counters_32/got_sysref_r_reg/C] \
  -to [get_pins {i_my_ip_support_block/jesd204_i/inst/
sync_tx_sysref_captured/syncstages_ff_reg[0]/D}] \
  -timestamp "<timestamp>" ;#1

create_waiver -type CDC -id {CDC-11} -user "AMD" \
  -desc "Safe fanout. Circuitry has been released" \
  -from [get_pins {i_my_ip_support_block/jesd204_i/inst/
i_my_ip_reset_block/stretch_reg[10]/C}] \
  -to
[get_pins {i_my_ip_support_block/i_jesd204_phy/inst/jesd204_phy_block_i/
sync_rx_reset_data/xpm_cdc_async_rst_inst/arststages_ff_reg[0]/CLR}] \
  -timestamp "<timestamp>" ;#1

create_waiver -type CDC -id {CDC-11} -user "AMD" \
  -desc "Safe fanout. Circuitry has been released" \
  -from [get_pins {i_my_ip_support_block/jesd204_i/inst/
i_my_ip_reset_block/stretch_reg[10]/C}] \
  -to
[get_pins {i_my_ip_support_block/i_jesd204_phy/inst/jesd204_phy_block_i/
sync_tx_reset_data/xpm_cdc_async_rst_inst/arststages_ff_reg[0]/CLR}] \
  -timestamp "<timestamp>" ;#2
```

Step 8: Using the create_waiver Command

Waivers added from the Report CDC dialog box are created using the `create_waiver` command. You can view these commands as follows.

Note: You can use the `create_waiver` command line command for CDC, DRC, and methodology waivers. The options differ slightly depending on whether you are creating a CDC, DRC, or methodology waiver. For more information, including information on the different options, see the *create_waiver* command in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

1. Open the Vivado journal file (`vivado.jou`) to see the three distinct `create_waiver` commands issued by the Vivado IDE.
2. Scroll through the history of the Tcl Console to see the same three `create_waiver` commands.

TIP: *The `-from` and `-to` options are used to specify the startpoints and endpoints. When a waiver is set from the Report CDC dialog box, both `-from` and `-to` are specified to match the exact violation. However, you can specify a CDC waiver using only the `-from` option or only the `-to` option, but more paths might be waived than expected.*

Step 9: Waiving Multiple CDC Violations

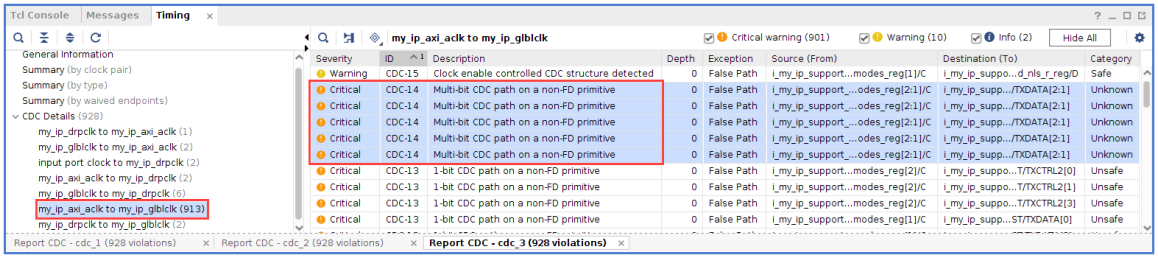
In this step, you waive multiple CDC violations simultaneously.

1. In the CDC Report, view the `my_ip_axi_aclk` to `my_ip_glblclk` CDC under CDC Details.

   This crossing has five CDC-14 violations that are multi-bit violations. The five CDC-14 violations all start from the same two register clock pins:

   ```
   i_my_ip_support_block/jesd204_i/inst/tx_cfg_test_modes_reg[2:1]/C
   ```

   TIP: *You can sort the table by the column ID to more easily see the five CDC-14 violations.*



2. Because `i_my_ip_support_block/jesd204_i/inst/tx_cfg_test_modes_reg[*]/C` matches five pins and you only need to target two of those five pins, construct the list of startpoints as follows:

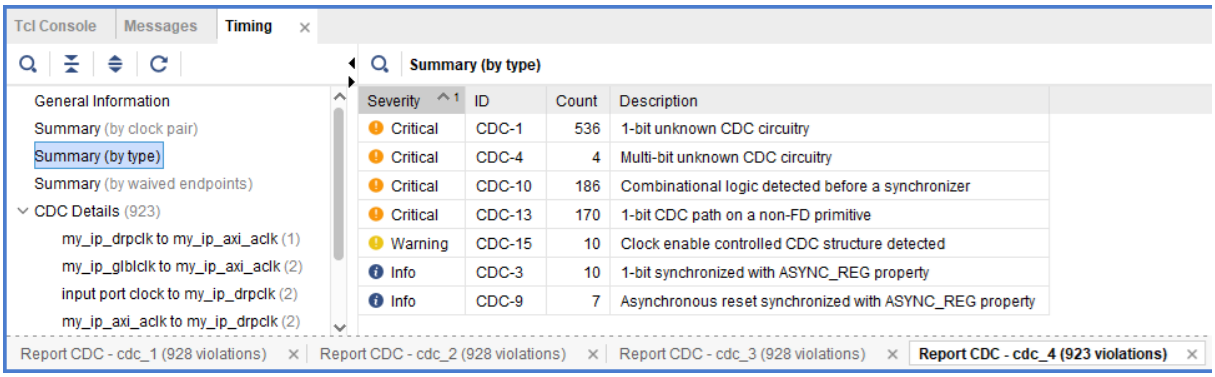
   ```
   set startpoints [list \
   [get_pins i_my_ip_support_block/jesd204_i/inst/
   tx_cfg_test_modes_reg[1]/C] \
   [get_pins i_my_ip_support_block/jesd204_i/inst/
   tx_cfg_test_modes_reg[2]/C] \
   ]
   ```

3. To waive the five CDC-14 violations, use the `create_waiver` Tcl command with the `-from` option:

   ```
   create_waiver -type {CDC} -id {CDC-14} -user {AMD} -desc {No more CDC
   14!} -from $startpoints
   ```

4. From the Vivado IDE, select **Reports → Timing → Report CDC** to rerun Report CDC.

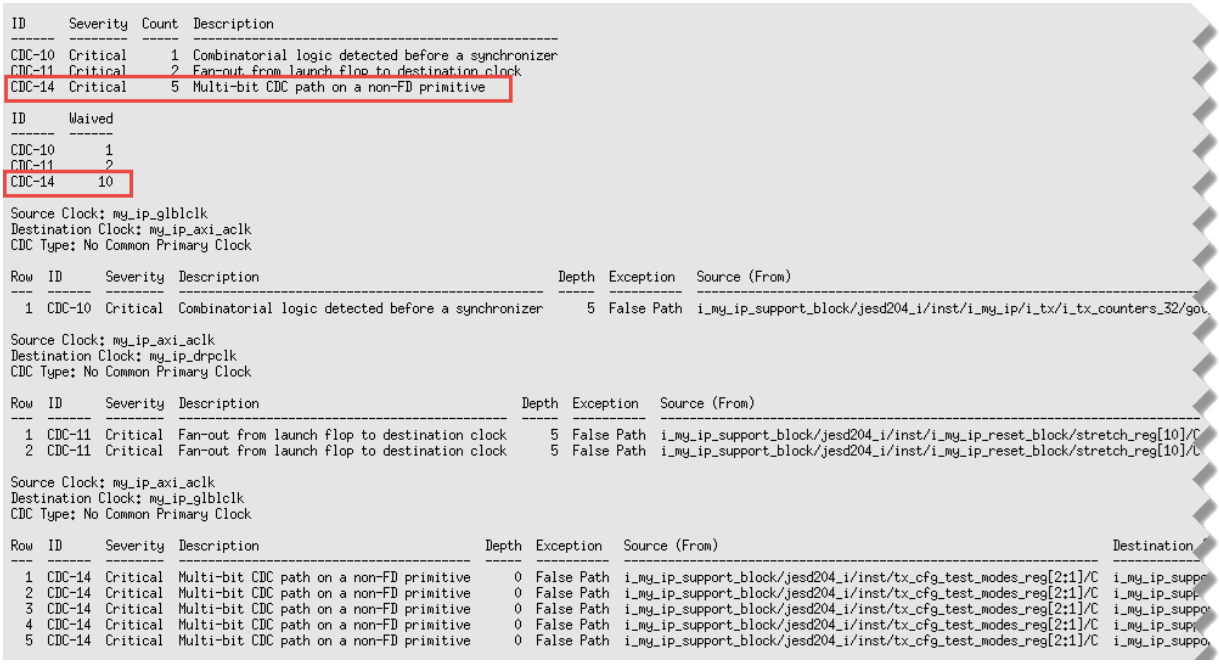
5. In the CDC Report, verify that the CDC-14 violations are no longer reported in the Summary section.



6. To report only the waived violations, enter:

```
report_cdc -details -waived
```

The following figure shows the waived CDC violations in two different tables. The first table shows the 5 CDC-14 violations waived as multi-bit violations. The second table shows the 10 single-bit violations, calculated by multiplying the 5 multi-bit violations by 2 bits per multi-bit violation.



7. To export all the waivers inside a script and verify that a total of four waivers were added, enter:

```
write_waivers -type cdc waivers.xdc -force
```

Note: Because the `waivers.xdc` file already exists, the `-force` option must be specified to override the file.

TIP: *Alternatively, because there are no DRC or methodology waivers, you can enter:*

```
write_waivers waivers.xdc -force
```

or

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

The list of waivers inside `waivers.xdc` appears as follows.

```
#
# WRITE CDC Waivers
# cmd: write_waivers -type cdc waivers.xdc -force
current_instance -quiet
create_waiver -type CDC -id {CDC-10} -user "AMD" -desc "This is a safe CDC peer review with the team" -from [get_pins i_my_ip_support_block/jesd204_i/inst/i_my_ip/i_tx/i_tx_counters_32/got_sysref_r_reg/C]
create_waiver -type CDC -id {CDC-11} -user "AMD" -desc "Safe fanout. Circuitry has been reviewed" -from [get_pins {i_my_ip_support_block/jesd204_i/inst/i_my_ip_reset_block/stretch_reg[10]/C}] -to [get_pins
create_waiver -type CDC -id {CDC-11} -user "AMD" -desc "Safe fanout. Circuitry has been reviewed" -from [get_pins {i_my_ip_support_block/jesd204_i/inst/i_my_ip_reset_block/stretch_reg[10]/C}] -to [get_pins
create_waiver -type CDC -id {CDC-14} -user "AMD" -desc "No more CDC14!" -from [list [get_pins {i_my_ip_support_block/jesd204_i/inst/tx_cfg_test_modes_reg[1]/C}] [get_pins {i_my_ip_support_block/jesd204_i/i
#
current_instance -quiet
```

8. To import the `waivers.xdc` file, enter:

```
read_xdc waivers.xdc
```

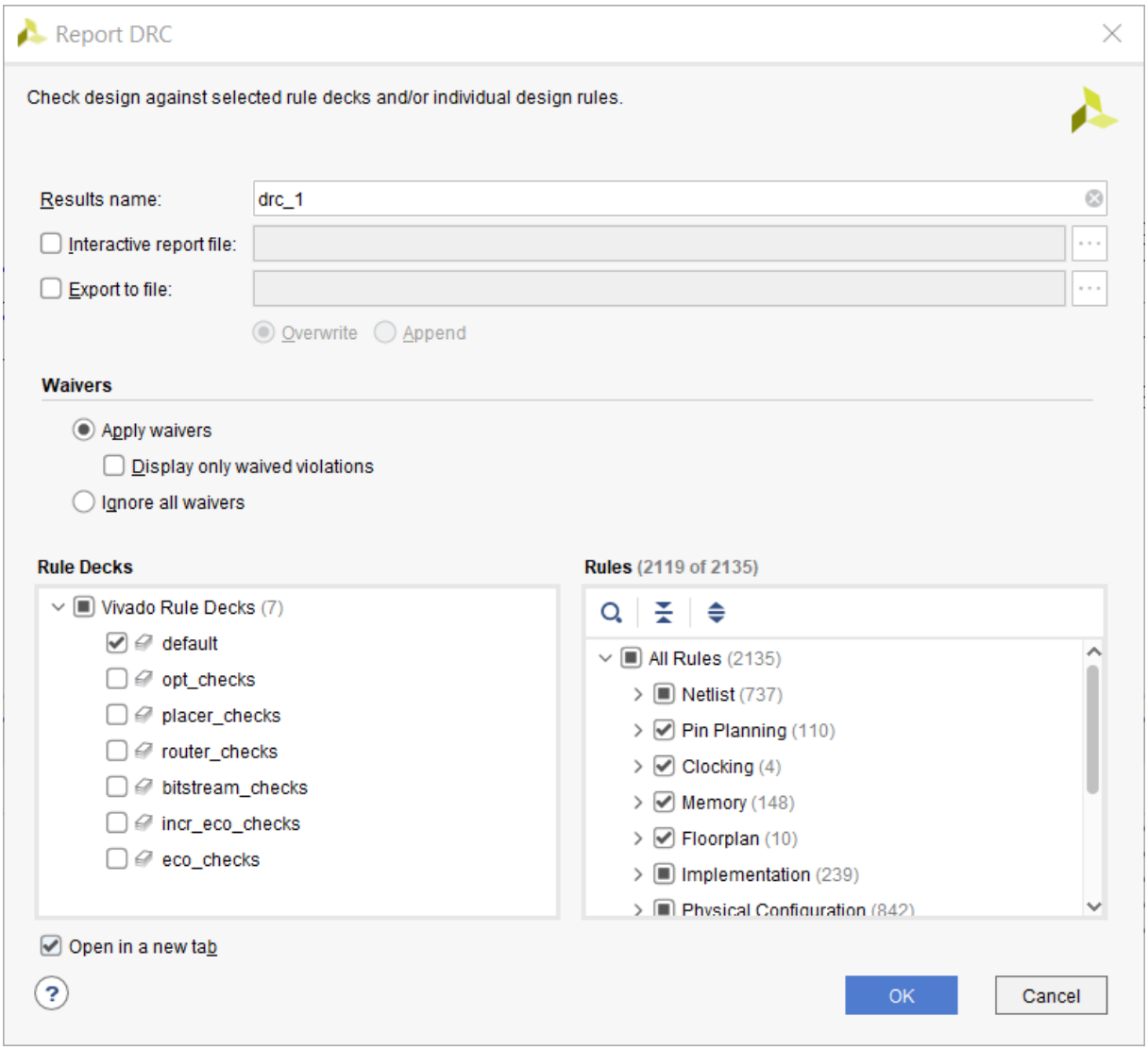
The following warnings show that the duplicate waivers were not added to the existing waivers. Only waivers that are exact duplicates of existing waivers are rejected.

```
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-10' is a duplicate and will
not be added again.
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-11' is a duplicate and will
not be added again.
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-11' is a duplicate and will
not be added again.
WARNING: [Vivado_Tcl 4-935] Waiver ID 'CDC-14' is a duplicate and will
not be added again.
```

Step 10: Waiving Multiple DRC Violations

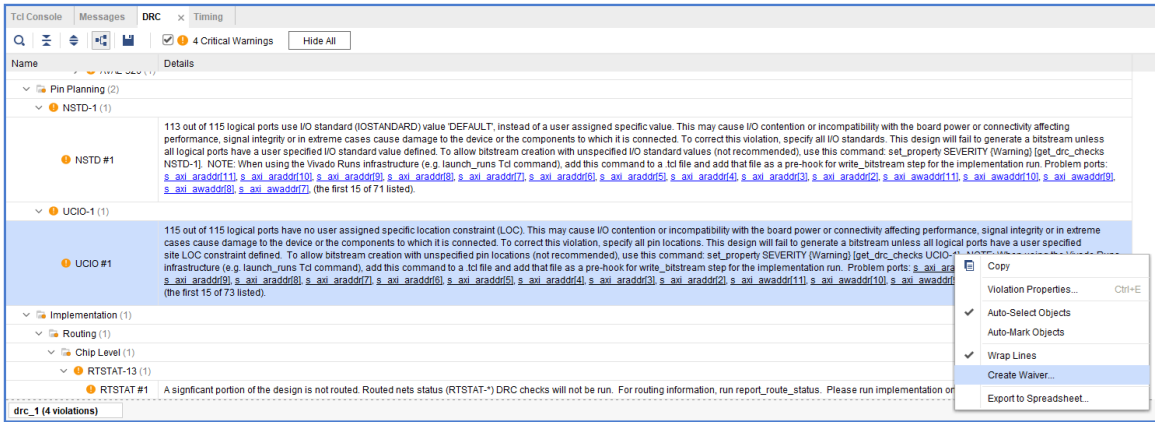
In this step, you waive multiple DRC violations simultaneously.

1. Select **Reports → Report DRC**.
2. In the Report DRC dialog box, leave all settings at their default, and click **OK**.

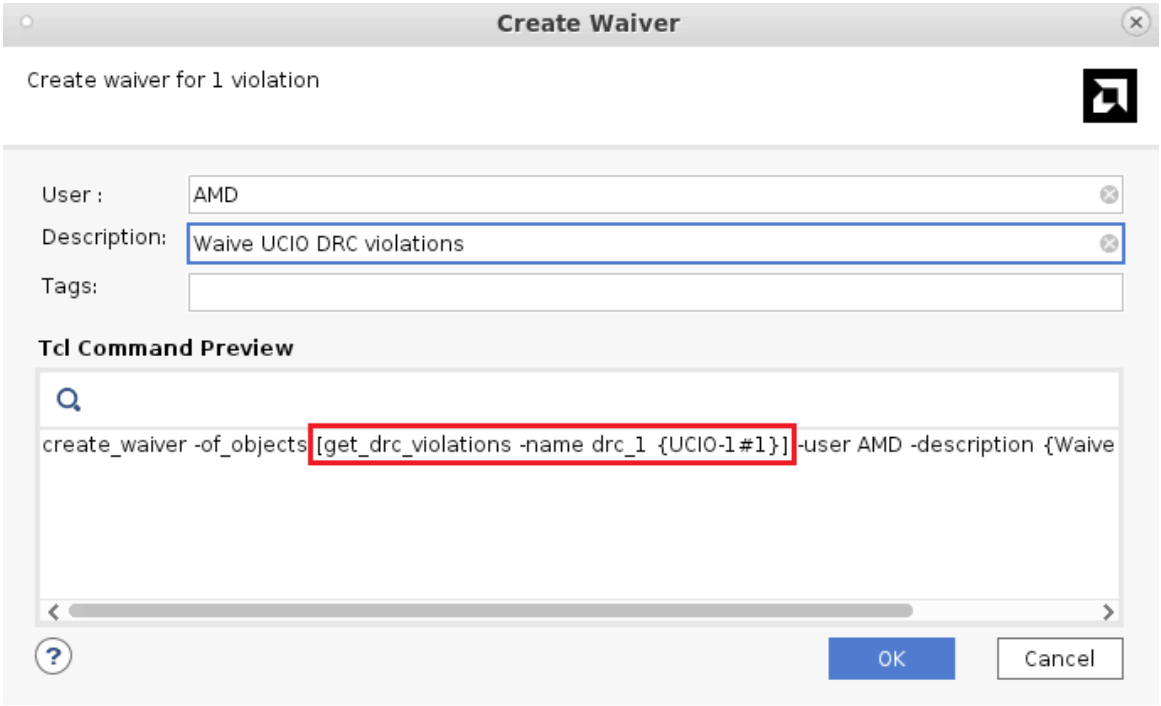


3. In the DRC Report, right-click **UCIO#1**, and select **Create Waiver** to create a waiver for the UCIO-1 violations.

   ***Note*:** The UCIO#1 violation combines 115 individual violations into a single violation. Similarly, the NSTD#1 violation covers 113 ports.



4. In the Create Waiver dialog box, look at the output in Tcl Command Preview, and click **OK**.



5. To generate the `drc_waivers.xdc` file and verify that the waiver is waiving all 115 objects, enter:

```
write_waivers -type DRC drc_waivers.xdc
```

6. In the `XDC` file, look at the expanded port list, and check that some of the strings from the violations message were converted to wildcards (*).

   Strings are automatically converted to wildcards for UCIO-1, NSTD-1, TIMING-15, and TIMING-16 type violations. For UCIO-1, the numbers of objects in the violations are replaced with wildcards, because the numbers of elements are not meaningful.

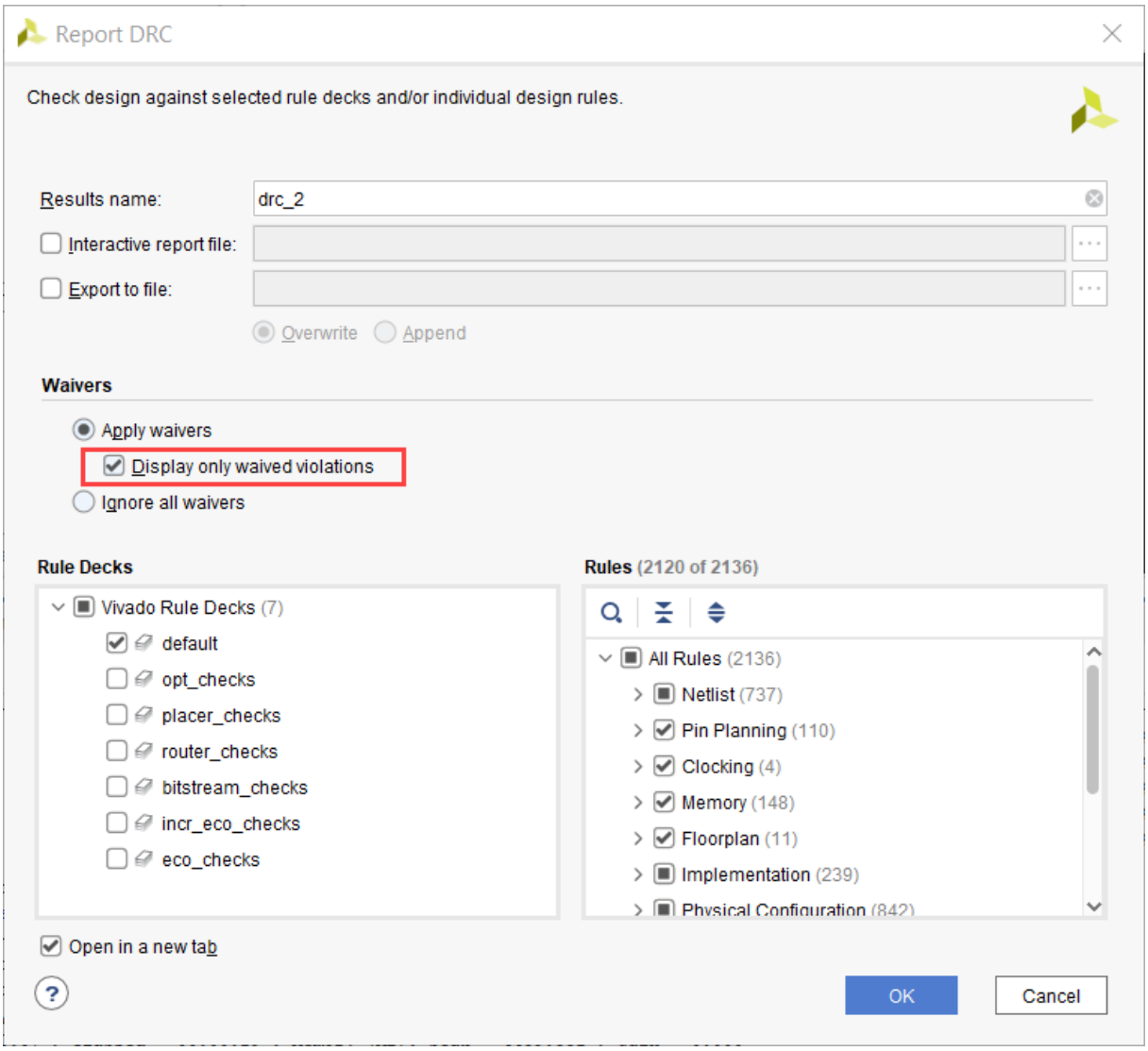
```
#
# WRITE DRC WAIVERS
# cmd: write_waivers -type DRC drc_waivers.xdc
current_instance -quiet
create_waiver -type DRC -id {UCIO-1} -user "AMD" -desc "Waive UCIO DRC violations" -objects [get_ports { tx_sync tx_sysref tx_aresetn s_axi_rready s_axi_rvalid s_axi_rresp[0] s_axi_rresp[1] s_axi_rdata[0] s_a
xi_rdata[1] s_axi_rdata[2] s_axi_rdata[3] s_axi_rdata[4] s_axi_rdata[5] s_axi_rdata[6] s_axi_rdata[7] s_axi_rdata[8] s_axi_rdata[9] s_axi_rdata[10] s_axi_rdata[11] s_axi_rdata[12] s_axi_rdata[13] s_axi_rdata[
14] s_axi_rdata[15] s_axi_rdata[16] s_axi_rdata[17] s_axi_rdata[18] s_axi_rdata[19] s_axi_rdata[20] s_axi_rdata[21] s_axi_rdata[22] s_axi_rdata[23] s_axi_rdata[24] s_axi_rdata[25] s_axi_rdata[26] s_axi_rdata[
27] s_axi_rdata[28] s_axi_rdata[29] s_axi_rdata[30] s_axi_rdata[31] s_axi_arready s_axi_arvalid s_axi_araddr[2] s_axi_araddr[3] s_axi_araddr[4] s_axi_araddr[5] s_axi_araddr[6] s_axi_araddr[7] s_axi_araddr[8]
s_axi_araddr[9] s_axi_araddr[10] s_axi_bready s_axi_bvalid s_axi_bresp[0] s_axi_bresp[1] s_axi_wready s_axi_wvalid s_axi_wstrb[0] s_axi_wstrb[1] s_axi_wstrb[2] s_axi_araddr[11] s_axi_wdata[0] s_axi_wdata[1] s
_axi_wdata[2] s_axi_wdata[3] s_axi_wdata[4] s_axi_wdata[5] s_axi_wdata[6] s_axi_wdata[7] s_axi_wdata[8] s_axi_wdata[9] s_axi_wdata[10] s_axi_wdata[11] s_axi_wdata[12] s_axi_wdata[13] s_axi_wdata[14] s_axi_wda
ta[15] s_axi_wdata[16] s_axi_wdata[17] s_axi_wdata[18] s_axi_wdata[19] s_axi_wdata[20] s_axi_wdata[24] s_axi_wdata[25] s_axi_wdata[26] s_axi_wdata[27] s_axi_wdata[28] s_axi_awready s_axi_awvalid s_axi_awaddr[
2] s_axi_awaddr[3] s_axi_awaddr[4] s_axi_awaddr[5] s_axi_awaddr[6] s_axi_awaddr[7] s_axi_awaddr[8] s_axi_awaddr[9] s_axi_awaddr[10] s_axi_awaddr[11] s_axi_aresetn s_axi_aclk tx_start_of_multiframe[0] s_axi_ws
trb[3] tx_start_of_multiframe[2] tx_start_of_frame[0] tx_start_of_multiframe[1] tx_start_of_frame[2] tx_start_of_frame[1] tx_start_of_frame[3] tx_reset tx_start_of_multiframe[3] glblclkn drpclk glblclkp refcl
k0n refclk0p }] -strings { "*" } -strings { "*" } -timestamp "Tue Oct 31 16:26:07 GMT 2023" ;#1
#
current_instance -quiet
```

7. To delete the DRC waiver and rewrite the waiver using wildcards to target a subset of the ports objects, enter:

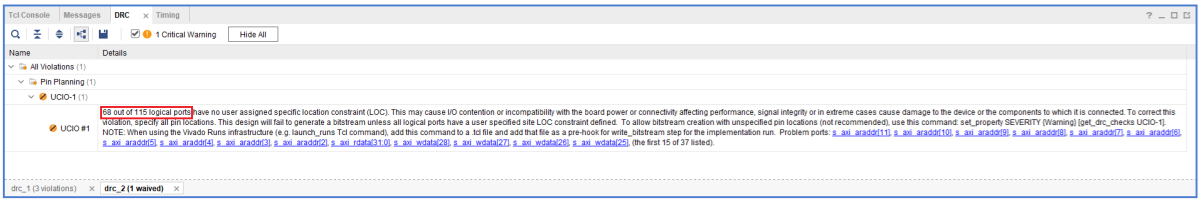
```
delete_waivers [get_waivers -type drc]
create_waiver -type DRC -id {UCIO-1} -user "AMD" -desc "Waive selected
UCIO violations" -objects [get_ports { s_axi_rdata[*] s_axi_wdata[*]
s_axi_araddr[*] } ] -strings { "*" } -strings { "*" }
```

   ***Note*:** This command only covers a subset of the original 115 objects.

8. Select **Reports → Report DRC** to rerun Report DRC.
9. In the Report DRC dialog box, select **Display only waived violations** to report only waived violations, and click **OK**.



In the DRC Report, verify that only 68 violations are waived out of 115.



IMPORTANT! *You cannot waive READONLY or NODISABLE violations. For example, if you enter:*

```
create_waiver -type DRC -id RTSTAT-1 -description "Waive RTSTAT-1"
```

The Vivado tools issue the following error:

```
 ERROR: [Vivado_Tcl 4-934] Waiver ID 'RTSTAT-1' is READONLY or
NODISABLE and cannot be waived.
These Factory designations specify that a check is required and may
not be overridden by user action.
```

Step 11: Generating a Summary Report for Waived Violations

This step covers how to use the `report_waivers` Tcl command to generate a summary report for CDC, DRC, and methodology waivers.

IMPORTANT! *Before running the `report_waivers` command, you must rerun Report CDC, Report DRC, or Report Methodology to ensure that added or removed waivers are included in the statistics reported by `report_waivers`.*

1. To rerun Report CDC, enter:

   ```
   report_cdc
   ```

2. To rerun Report DRC, enter:

   ```
   report_drc
   ```

   Note: You do not need to rerun Report Methodology, because no methodology waivers were set.

3. To create a summary report, enter:

   ```
   report_waivers
   ```

   By default, `report_waivers` reports only waived violations. The following figure shows the UCIO-1, CDC-10, CDC-11, and CDC-14 rules that have defined waivers.

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


-----------------
1. REPORT SUMMARY
-----------------
Waiver Type  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-----------  ----------  --------------  -----------  ------------  -----------
DRC          230         162             68           1             1
METHODOLOGY  0           0               0            0             0
CDC          957         944             13           4             4
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
-------  ----------------  -------------------------  ----------  --------------  -----------  ------------  -----------
UCIO-1*  Critical Warning  Unconstrained Logical Port  115         47              68           1             1


--------------------------------
4. REPORT DETAILS (CDC)
--------------------------------
Rule    Severity  Description                                          Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
------  --------  ---------------------------------------------------  ----------  --------------  -----------  ------------  -----------
CDC-10  Critical  Combinational logic detected before a synchronizer   187         186             1            1             1
CDC-11  Critical  Fan-out from launch flop to destination clock        2           0               2            2             2
CDC-14  Critical  Multi-bit CDC path on a non-FD primitive             10          0               10           1             1

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.
```

Note the number of waived objects and total violations:

- The aggregating DRCs are reported as 1 violation per object inside the violation. Because there are 113 objects in NSTD-1, 115 objects in UCIO-1, 1 in RTDAT-13, and 1 in AVAL-326, a total number of 240 DRC violations are reported in the summary table.
- The Report Summary table reports all of the violations.
- The Report Details tables only report the check IDs that have one or more waivers.

4. To generate detailed tables with all of the rules, including rules with no waivers, enter:

```
report_waivers -show_msgs_with_no_waivers
```

The following figure shows the report with all DRC and CDC rules reported in the Report Details.

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


-----------------
1. REPORT SUMMARY
-----------------
Waiver Type  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-----------  ----------  --------------  -----------  ------------  -----------
DRC          230         162             68           1             1
METHODOLOGY  0           0               0            0             0
CDC          957         944             13           4             4
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
---------  ----------------  -------------------------  ----------  --------------  -----------  ------------  -----------
UCIO-1*    Critical Warning  Unconstrained Logical Port  115         47              68           1             1
RTSTAT-13  Critical Warning  Insufficient Routing        1           1               0            0             0
AVAL-326   Critical Warning  Hard_block_must_have_LOC    1           1               0            0             0
NSTD-1*    Critical Warning  Unspecified I/O Standard    113         113             0            0             0


---------------------------------
4. REPORT DETAILS (CDC)
---------------------------------
Rule    Severity  Description                                          Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
------  --------  ---------------------------------------------------  ----------  --------------  -----------  ------------  -----------
CDC-10  Critical  Combinational logic detected before a synchronizer   187         186             1            1             1
CDC-11  Critical  Fan-out from launch flop to destination clock        2           0               2            2             2
CDC-14  Critical  Multi-bit CDC path on a non-FD primitive             10          0               10           1             1
CDC-1   Critical  1-bit unknown CDC circuitry                           536         536             0            0             0
CDC-3   Info      1-bit synchronized with ASYNC_REG property           9           9               0            0             0
CDC-4   Critical  Multi-bit unknown CDC circuitry                      28          28              0            0             0
CDC-9   Info      Asynchronous reset synchronized with ASYNC_REG property  5           5               0            0             0
CDC-13  Critical  1-bit CDC path on a non-FD primitive                 170         170             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.
.
```

5. To run Report Methodology, enter:

```
report_methodology
```

6. To generate detailed tables with all of the rules, including rules with no waivers, enter:

```
report_waivers -show_msgs_with_no_waivers
```

The exact statistics are reported, as shown in the following figure.

Note: This figure does not include the Report Details (CDC) section.

```
-----------------
1. REPORT SUMMARY
-----------------
Waiver Type  Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
-----------  ----------  --------------  -----------  ------------  -----------
DRC          230         162             68           1             1
METHODOLOGY  158         158             0            0             0
CDC          957         944             13           4             4
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
---------  ----------------  -------------------------  ----------  --------------  -----------  ------------  -----------
UCIO-1*    Critical Warning  Unconstrained Logical Port  115         47              68           1             1
RTSTAT-13  Critical Warning  Insufficient Routing        1           1               0            0             0
AVAL-326   Critical Warning  Hard_block_must_have_LOC    1           1               0            0             0
NSTD-1*    Critical Warning  Unspecified I/O Standard    113         113             0            0             0


---------------------------------
3. REPORT DETAILS (METHODOLOGY)
---------------------------------
Rule        Severity          Description                     Total Vios  Remaining Vios  Waived Vios  Used Waivers  Set Waivers
----------  ----------------  ------------------------------  ----------  --------------  -----------  ------------  -----------
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
AVAL-324    Critical Warning  Hard_block_needs_LOC            1           1               0            0             0
LUTAR-1*    Warning           LUT drives async reset alert    40          40              0            0             0
```

Step 12: Using Waiver Commands

In this step, you run additional commands related to the waivers.

1. To return a collection of CDC waiver objects, enter:

   ```
   get_waivers -type cdc
   ```

   The following CDC waivers are returned:

   ```
   CDC-10#1 CDC-11#1 CDC-11#2 CDC-14#1
   ```

2. To filter the list of waivers to only return CDC-14 waivers, enter:

   ```
   get_waivers -filter {ID == CDC-14}
   CDC-14#1
   ```

3. To report all of the properties on a CDC waiver object, enter:

   ```
   report_property [lindex [get_waivers -type cdc] end]
   ```

   The following properties are returned:

   ```
   Property        Type    Read-only  Value
   CLASS           string  true       cdc_waiver
   DESCRIPTION     string  false      No more CDC-14!
   ID              string  true       CDC-14
   INDEX           string  true       1
   IS_SCOPED       bool    true       0
   NAME            string  true       CDC-14#1
   OBJECT_COUNTS   string  true       pins:2
   ```

```
SCOPE           string  true
TAGS            string  false
TIME            string  true        <timestamp>
TYPE            string  true        CDC
USED_CNT        string  true        10
USER            string  true        AMD
```

Note: You cannot retrieve the design objects attached to a waiver object.

4. To delete all of the previously created CDC-14 waivers, enter:

```
delete_waivers [get_waivers -filter {ID == CDC-14}]
```

Note: After a waiver object is deleted, the waiver no longer applies and the violations that it waived are reported again.

5. To delete all of the remaining CDC waivers, enter:

```
delete_waivers [get_waivers -type cdc]
```

Summary

In this lab, you accomplished the following:

- Waived CDC and DRC violations
- Generated reports for waived violations
- Exported waivers
- Used waiver commands

Chapter 2

Using Report QoR Suggestions and Report QoR Assessment for Timing Closure

Introduction

The `report_qor_suggestions` (RQS) command enables the AMD Vivado™ Design Suite tools to analyze a design and provide automated solutions for enhancing quality of results (QoR). The command can be run on an open design after synthesis or after any stage in the implementation flow. The RQS command evaluates the design and suggests fixes or improvements. Recommendations from RQS can take the following forms:

- RQS objects. These objects can add switches to a given command, or properties to a given design object. They also allow you to add implementation strategies customized for the design using machine learning algorithms.
- Text recommendations that require user intervention.

Using the same analysis techniques, the `report_qor_assessment` (RQA) command assesses the likelihood that a design meets its timing requirements and provides a quick summary of design metrics.

This lab covers how to do the following:

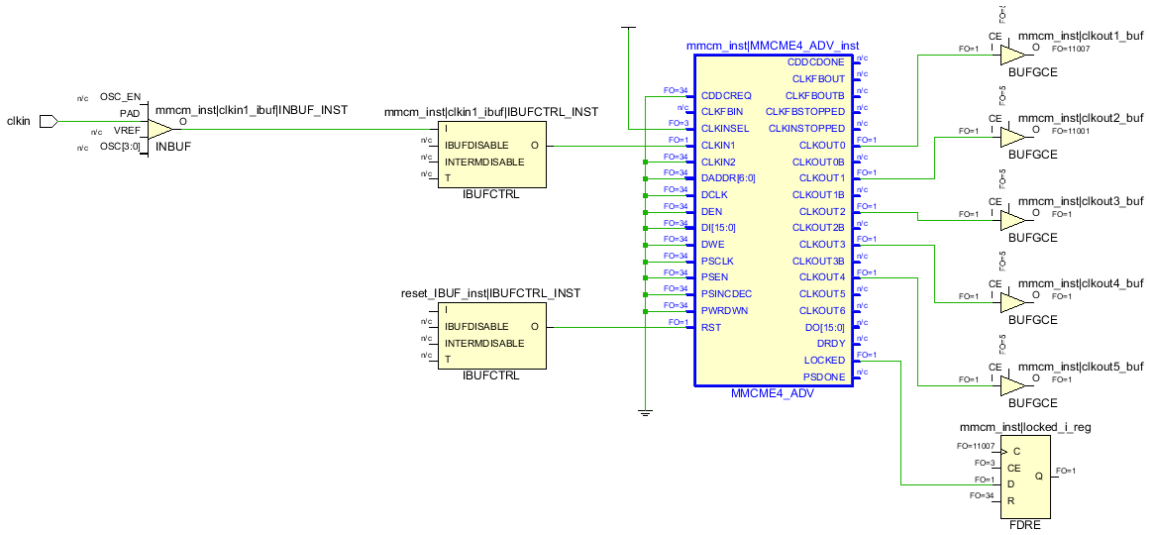
- Analyze the QoR assessment report.
- Analyze the QoR suggestion report.
- Export suggestions to an RQS file.
- Add an RQS file to synthesis and implementation runs.
- Accumulate suggestions in an RQS file by generating suggestions at different implementation stages or on different runs.
- Add the AUTO_RQS property to a project-based design run and automatically accrue suggestions.

Step 1: Understanding the Design

This lab uses an architected design to demonstrate some of the features of RQS. Suggestions are triggered by the design of the RTL and the placement of blocks using floor-planning. The design contains the following modules:

- **Clocking Module:** The main clocking circuit for the design resides in `clocking_module.vhd`. For simplicity, RST is tied to GND. LOCKED is registered and tied to an output port. The structure of this block is shown in the following figure.

Figure 3: **Block Structure**



- **Reg CLKA to CLKB Module:** This module contains a synchronous CDC for a large bus. It registers input data using CLKA and then passes it to a register on the CLKB domain to be passed to the output. Registering large buses on different related clock domains can impact hold slack (WHS/THS) and setup slack (WNS/TNS).
- **Bit Expander and Bit Reducer Modules:** These modules enable the expansion and contraction of internal data widths so that the design does not run out of I/Os. The modules take an arbitrary data width and expand or contract it to or from a desired size. The contraction logic creates many logic levels.

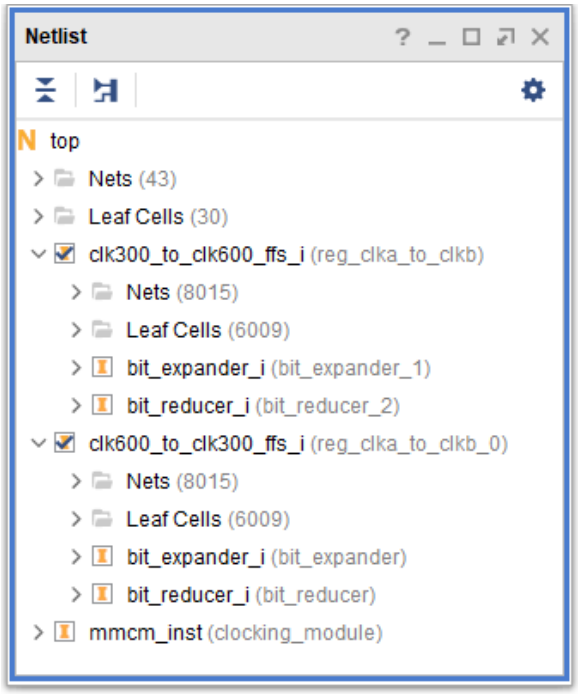
The following steps cover opening the project and examining the placement of the floor-planned modules.

1. In the Tcl console, change directory to the lab directory using the cd command as shown

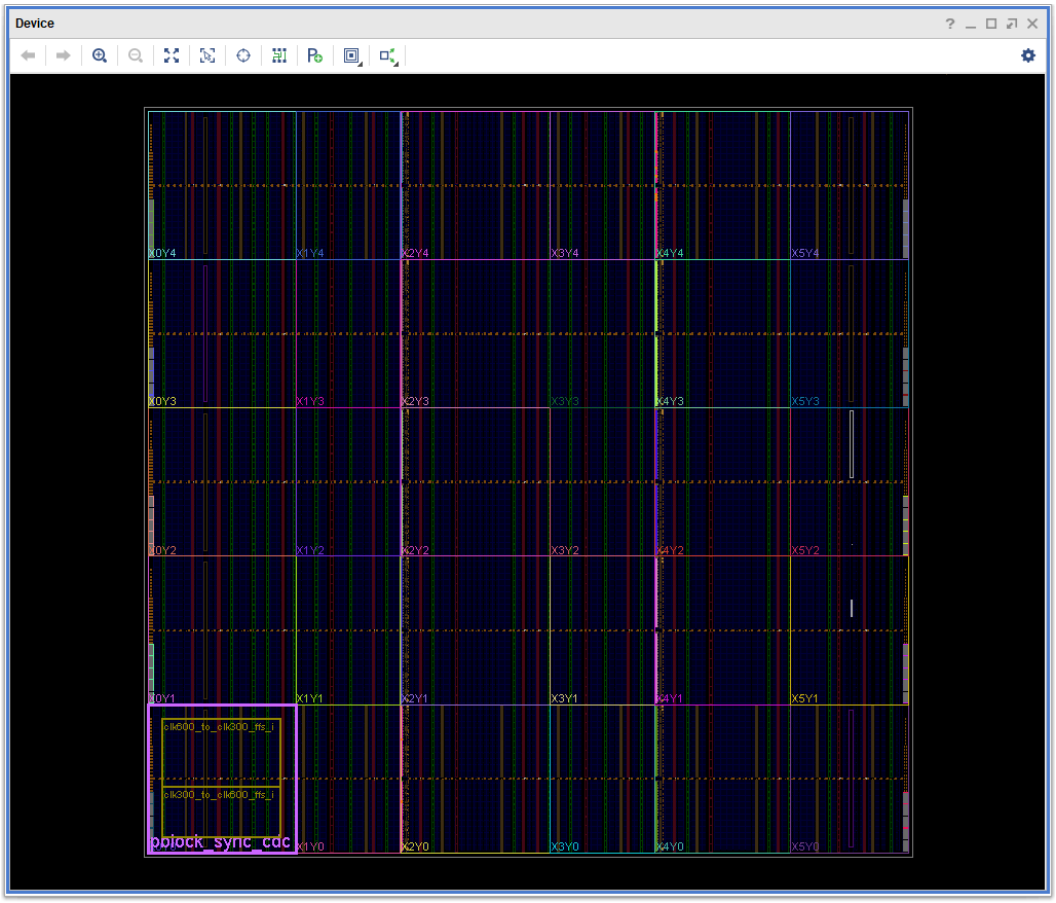
```
cd <extract_dir>/Lab2
```

2. In the Tcl console type `source ./tcl/create_project.tcl` file to create the project and run synthesis. You can explore the RTL files to familiarize yourself with them while synthesis is running.

3. In the Flow Navigator, click **Open Synthesized Design**.
4. In the Netlist view, look at the hierarchy.



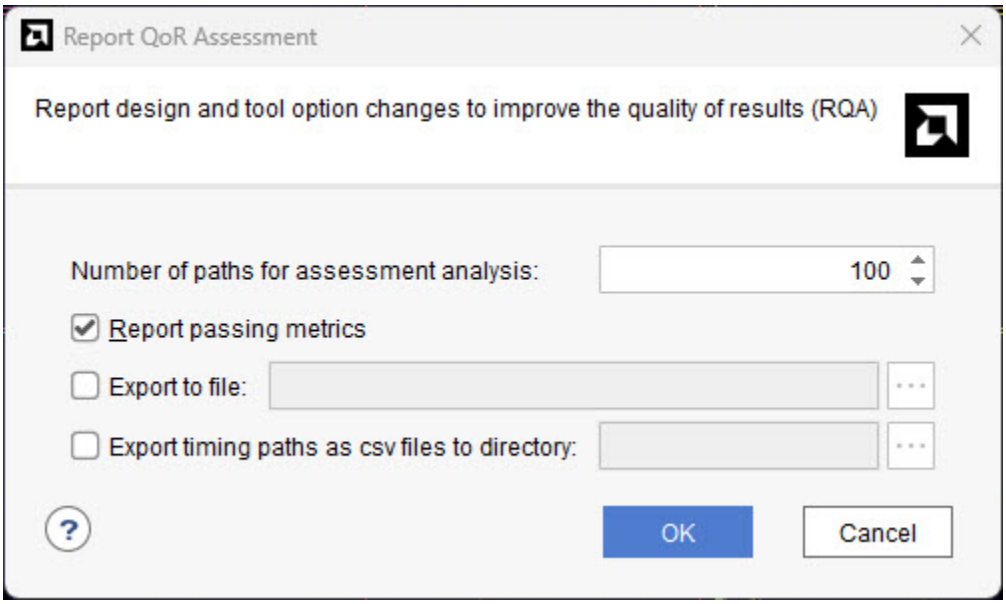
5. In Device view, look at the pblock. This has been added to control placement of the `reg_clka_to_clkb` modules and force a poor clock skew.



Step 2: Running Report QoR Assessment

In this step, you run the `report_qor_assessment` command on the open design. This step can be performed after any stage of the implementation process: `synth_design`, `opt_design`, `place_design`, `phys_opt_design`, and `route_design`. It provides an assessment score that indicates the likelihood of the design closing timing, as well as a quick first analysis returning items that need to be further investigated.

1. With the design open, click **Reports → Report QoR Assessment...**.
2. In the Report QoR Assessment dialog box, select **Report Passing Metrics**, then click **OK**.



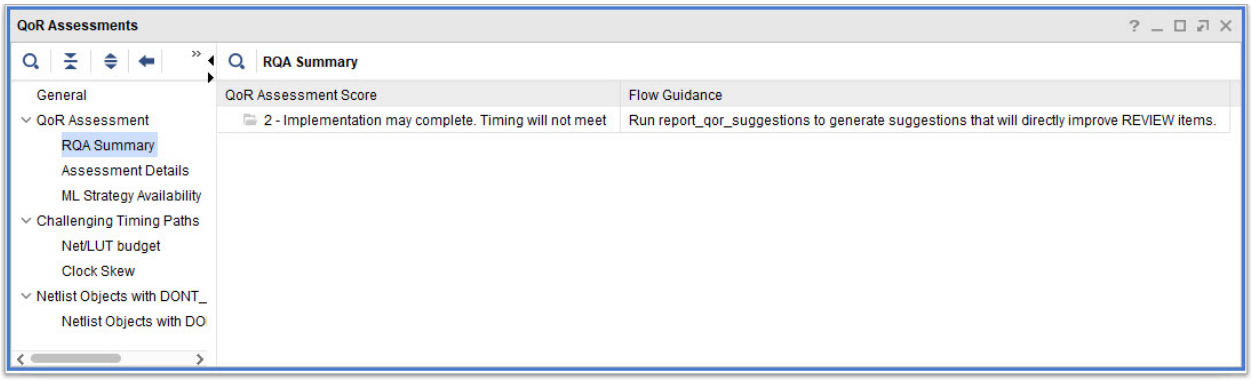
The equivalent Tcl command is:

```
report_qor_assessment -exclude_methodology_checks -name qor_assessments
-max_paths 100 -full_assessment_details
```

This opens the assessment report, which contains three main sections:

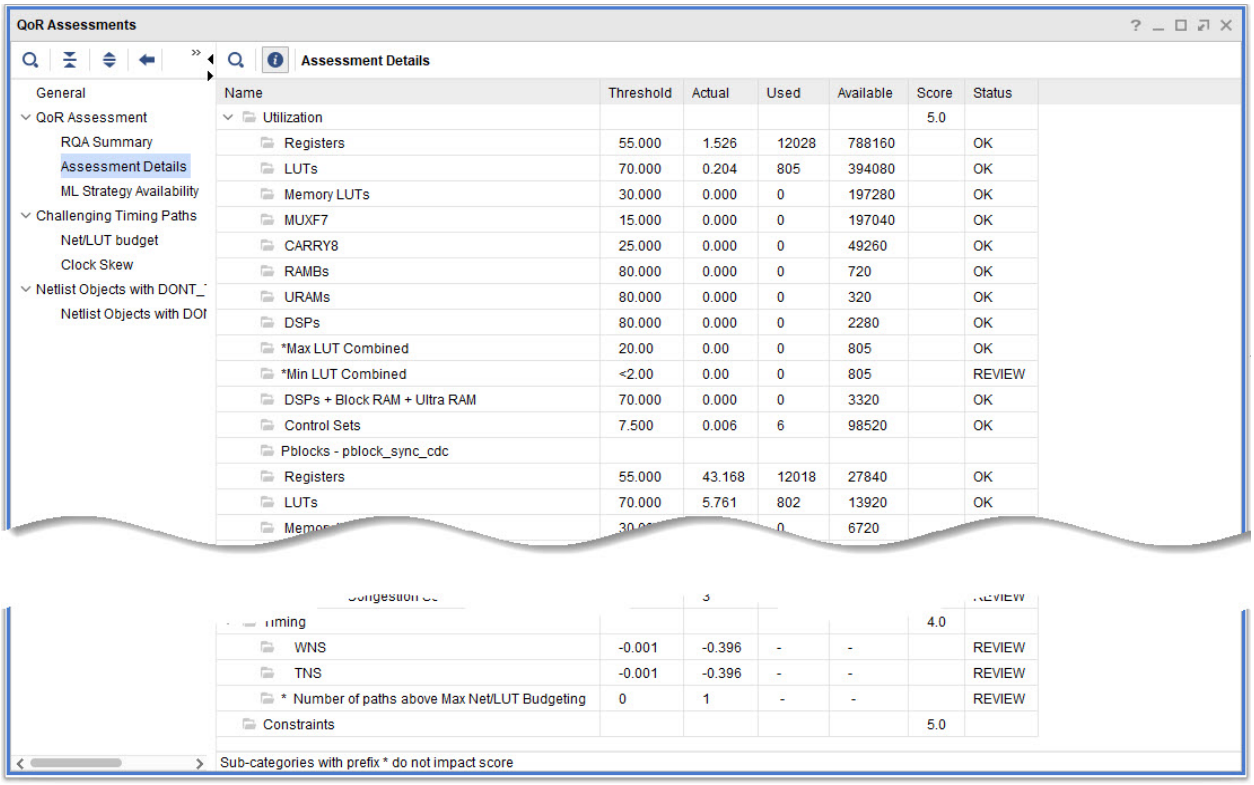
- Assessment summary
- Difficult timing paths requiring investigation
- Objects with DONT_TOUCH properties that prevent tool optimizations

3. In the first section of the report, select **RQA Summary**.



This section shows the score marked as "2 - Implementation may complete. Timing will not meet". It also recommends running `report_qor_suggestions`. The RQA command detects when suggestions are available and provides guidance to review them.

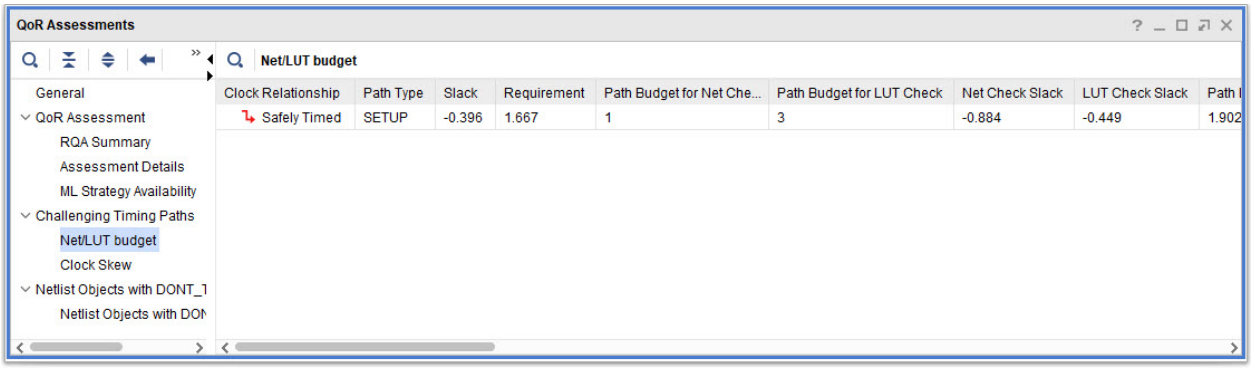
4. Select **Assessment Details**.



This section lists the items that have led to the assessment score of 2. Because the option to **Report passing metrics** was selected, items marked as OK are shown. Typically, only items marked REVIEW are displayed. In this example, you can see WNS and TNS metrics marked for review, as well as paths exceeding their net/LUT budget.

Timing paths usually reflect the values that are achievable in the best-case scenarios. Not all paths can meet these values. `Net` and `LUT budget checks` substitute net or LUT values with more typical values and penalize paths that are more challenging due to a netlist profile or a resource being less sparse. As a result, two checks performed under this item. Review all of these paths before continuing.

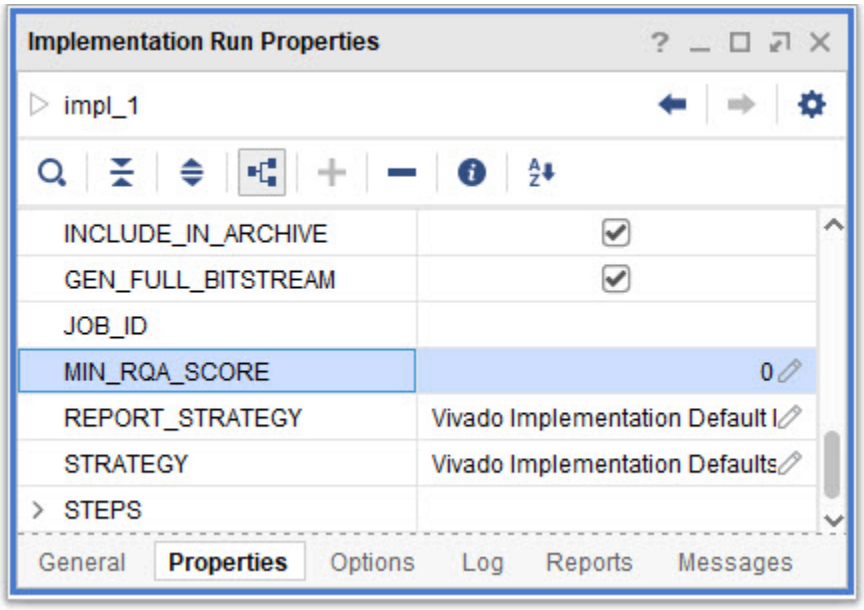
5. Select **Net/LUT budget** under Challenging Timing Paths.



Scroll across the screen to view all reported path characteristics. The following items are of particular interest:

- **Net Check Slack:** The slack when the path is substituted with higher net delay values.
- **LUT Check Slack:** The slack when LUTs are substituted with higher LUT delay values.

6. Double-click on the path to open the Timing Path report. You can also press `F4` to view a schematic. All of these items use standard Vivado cross-probing. At this stage, you can also explore other sections in the report.
7. In the Design Runs window, select **impl_1**, expand the **Implementation Run Properties** window, and click the **Properties** tab. Locate the MIN_RQA_SCORE property.



When the MIN_RQA_SCORE property is set, an implementation run is terminated if RQA is executed and the assessment score is less than the value specified. As the minimum RQA score is 1, MIN_RQA_SCORE must be set to 2 or greater to have an impact.

Edit the value of MIN_RQA_SCORE and set it to 3.

8. Next, the implementation run generates a QoR Assessment report.

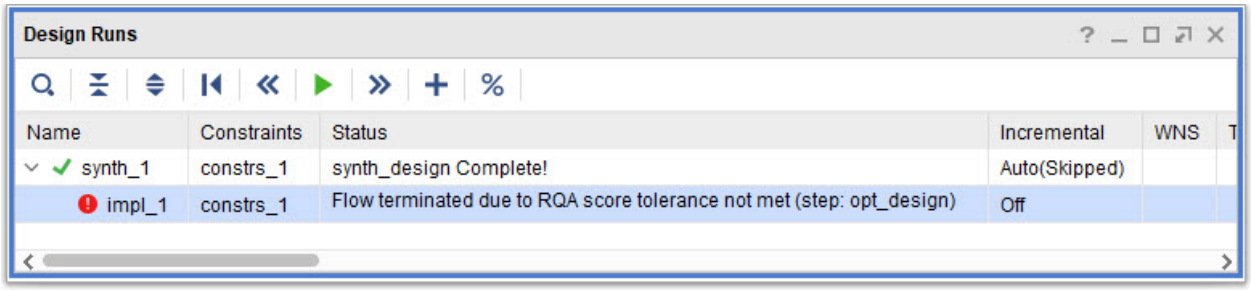
   In the implementation Run Properties window, go to the **Reports tab**. In the Report Strategy drop-down menu, select the **Timing Closure Reports Strategy**.

   ***Note*:** This is one method to configure the run to generate the report. Other methods are:

   1. Creating a custom report strategy.
   2. Adding `report_qor_assessment` to a Tcl hook. This must run the report in the run directory to be effective.

9. Launch the **impl_1** run.

After `opt_design` is complete, the `report_qor_assessment` command runs. If the score is below the threshold, the run should terminate. The following message appears in the Design Runs status column:



The log file contains the following message:

```
INFO: [runtcl 7-1] RQA Score (opt_design): 2.000  (RQA score tolerance:
3)
ERROR: [runtcl 8-1] Flow terminated - RQA score value 2.000, is below
RQA_MIN_SCORE 3 at opt_design step. Please evaluate the QoR Assessment
report for more information on the score and next steps to take.
```

The following message appears when the flow successfully passes the check:

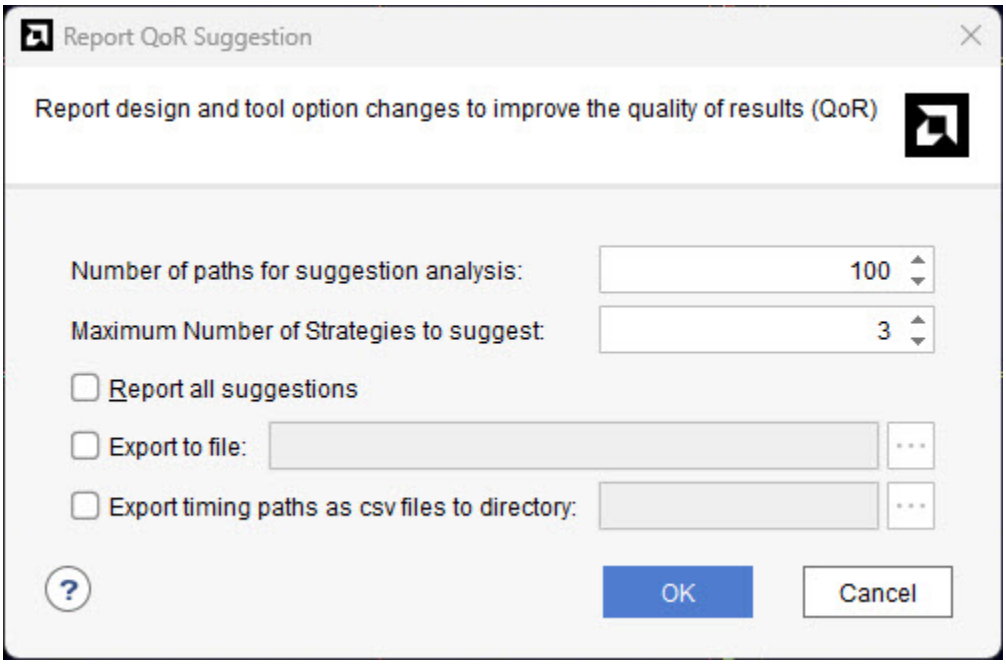
```
INFO: [runtcl 7-1] RQA Score (opt_design): 2 (RQA score tolerance: 2)
INFO: [runtcl 9-1] Flow continues - RQA score threshold met after
'opt_design' step
```

10. To revert the changes, update **MIN_RQA_VALUE** to **1**, set the Report Strategy back to **Vivado Implementation Default Reports**, and reset the run.

Step 3: Running Report QoR Suggestions

This step covers running the `report_qor_suggestions` command to generate a report. The command can be run on an open design at any stage of the implementation flow after synthesis. In project mode, this is typically after synthesis or implementation. In non-project mode, this can be after `synth_design`, `link_design`, `opt_design`, `place_design`, `phys_opt_design`, or `route_design`.

1. In the Vivado IDE, from the pull-down menus, click **Reports → Report QoR Suggestions...** to bring up the dialog box shown in the following figure.



2. Click **OK** to run the command. The report opens automatically in the integrated design environment (IDE). Due to the interactive nature of the report, only one instance of the report can be open at any time. The equivalent Tcl command is as follows:

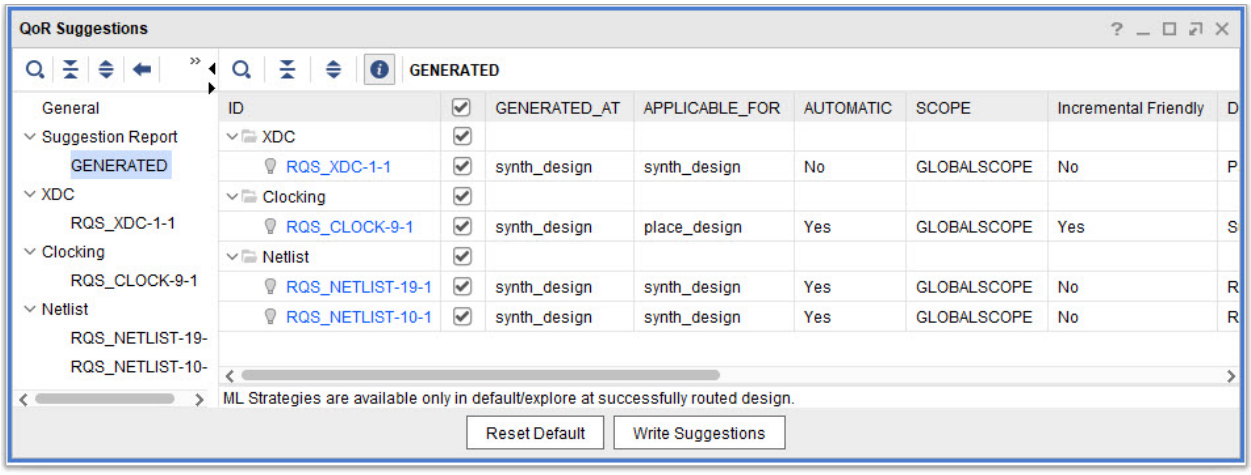
```
report_qor_suggestions -max_paths 100 -file rqs.rpt
```

***Note*:** By default, the RQS command reports on the 100 worst failing paths per clock group. You can change the number of paths that RQS uses for the analysis of timing-critical paths by modifying the `-max_paths` switch. Increasing this number generates more suggestions, but on paths that are reducing in criticality.

Step 4: Understanding the Report

This step explains the different sections of the generated QoR Suggestions report. On the left of the report window, you can navigate to the different sections of the report; on the right, more information is provided.

1. In the Suggestion Report, select **GENERATED**. This brings up the report section shown in the following figure.



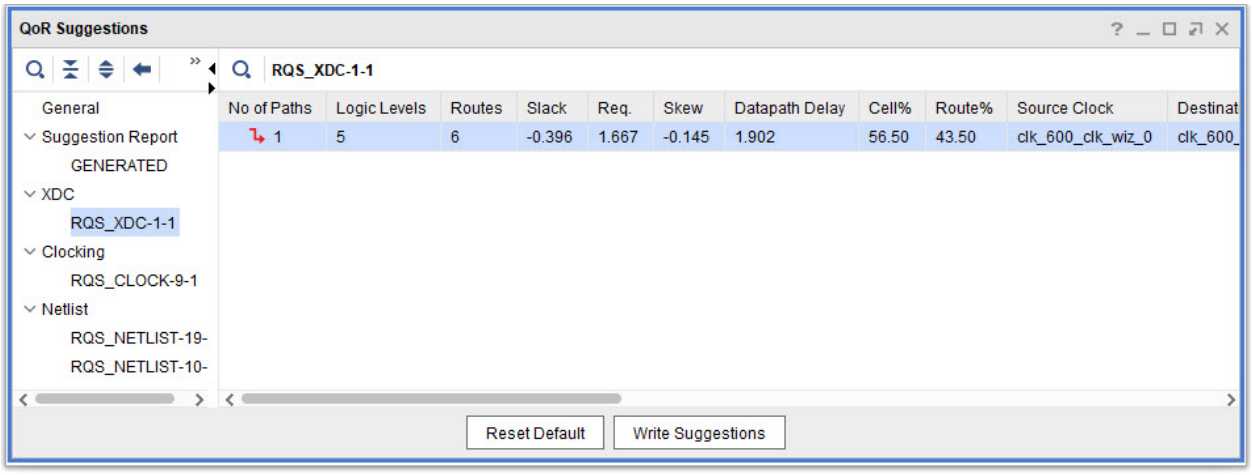
The GENERATED section provides a list of all the suggestions that have been generated at this stage of the current run. Each suggestion has a description that details the reason for the suggestion. For each suggestion, the following information is also provided.

Table 1: **RQS Summary Column Description**

Item	Description	Comment
GENERATED_AT	This shows what stage of the design the suggestion was generated at. Typical values are `place_design` or `route_design`.	As you progress through the design stages, the decisions that the tool makes are based on the information available at the time. Information accuracy increases after placement and again after routing.
APPLICABLE_FOR	This stage must be rerun for the suggestion to take effect.	Most suggestions are executed at either `synth_design` or `place_design`.
AUTOMATIC	Indicates if the suggestion is executed automatically or if manual intervention is required.	Automatic suggestions either recommend a switch to the tool or a property to be added to a cell or net.
INCREMENTAL FRIENDLY	Indicates if the suggestion is optimized for the incremental flow.	Non-incremental friendly suggestions must be already present in the reference checkpoint. If you want to add non-incremental friendly suggestions, an updated reference checkpoint must be used. This is not supported for Versal.
SCOPE	Indicates the target synthesis run level. GLOBALSCOPE is top level. Otherwise, a sub-module is targeted.	Allows a single RQS file to be used on top-level and sub-module out of context (OOC) synthesis runs. Only applicable to synthesis suggestions. Ignore this when not using synthesis suggestions or running non-OOC processes.

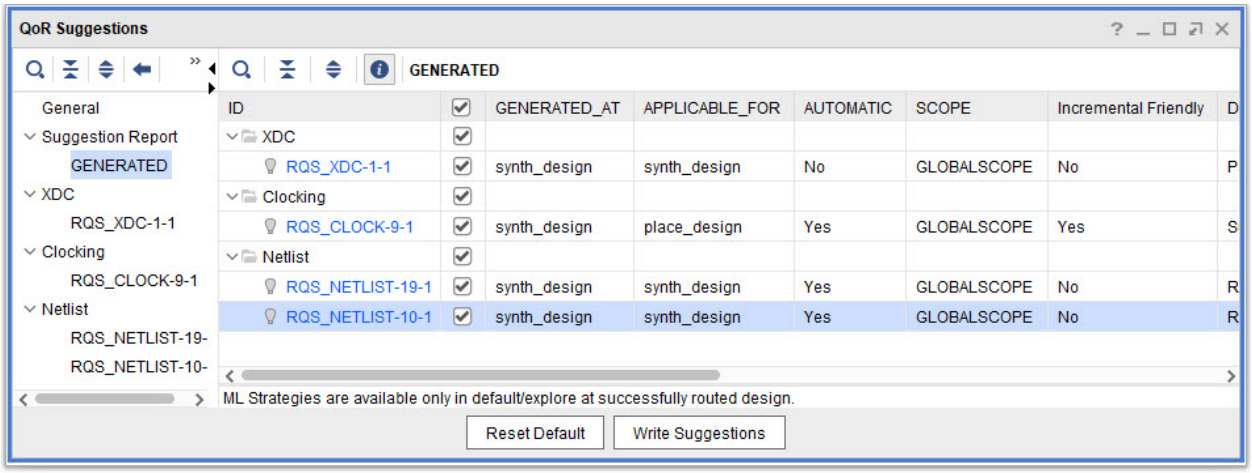
The other sections of the report usually provide details about the individual suggestions that have been generated.

2. Click the **RQS_XDC_1_1** hyperlink. This takes you to the detailed section for this suggestion.

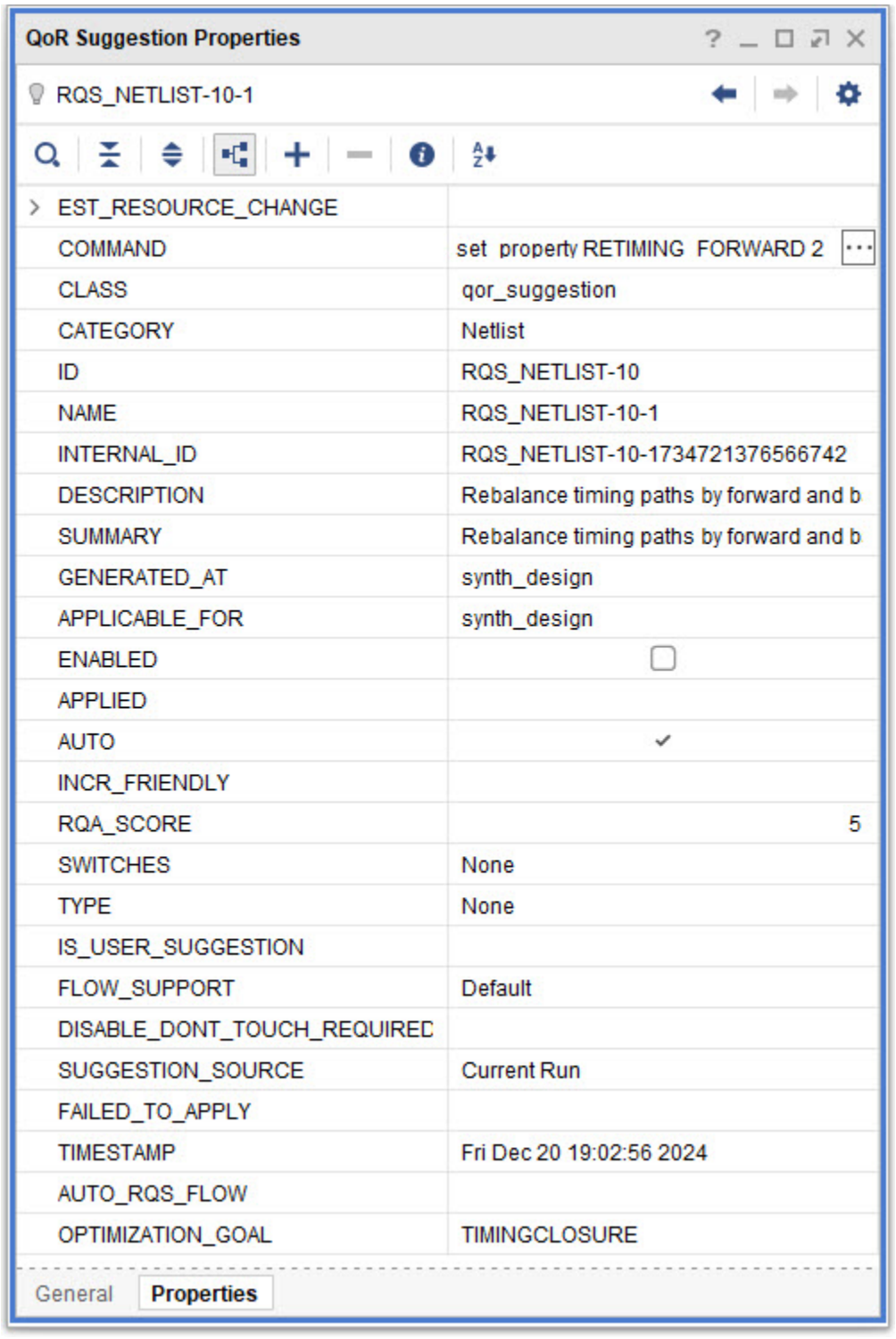


The suggestion description says that the timing constraint is too tight for the given path(s). The path has a large negative slack that stands out in a timing report. Timing paths use net delays that are optimal. This gives the tools the correct order to place and route them. Close analysis shows this is a 600 MHz path with high logic levels. This path needs to be fixed.

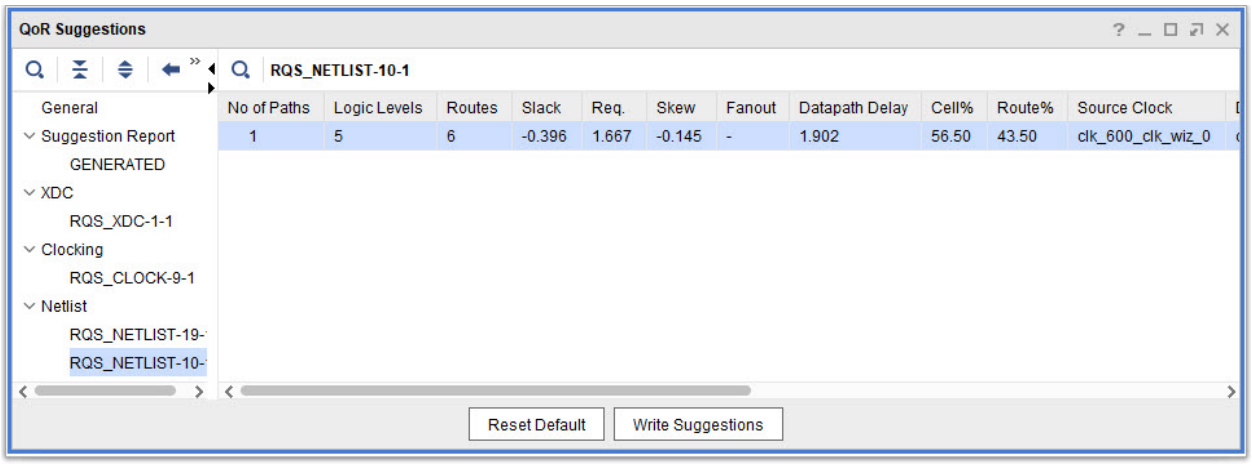
3. Click the back arrow button ( ) to go back to the GENERATED view.
4. In the GENERATED view, click the **RQS_NETLIST-10-1** row. You can see this is an AUTOMATIC suggestion that is applicable for `synth_design` (see the APPLICABLE_FOR column). This indicates that you must rerun the `synth_design` command to make use of this suggestion.



When you select the row in the GENERATED view, the suggestion object is selected and the properties can be seen in the QoR Suggestion Properties window. If you examine the COMMAND property, you can confirm that it generates a retiming forward property for `synth_design`.

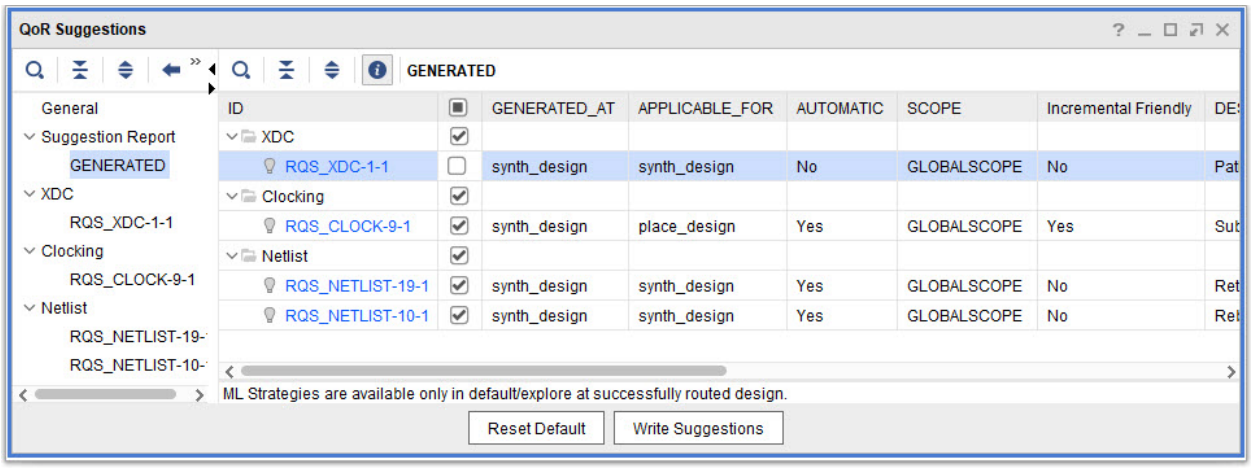


5. In the GENERATED view, click the **RQS_NETLIST-10-1** ID to navigate to the details table for that suggestion. Careful examination of the Endpoint column confirms that this is the same path that was mentioned for RQS_XDC-1-1.

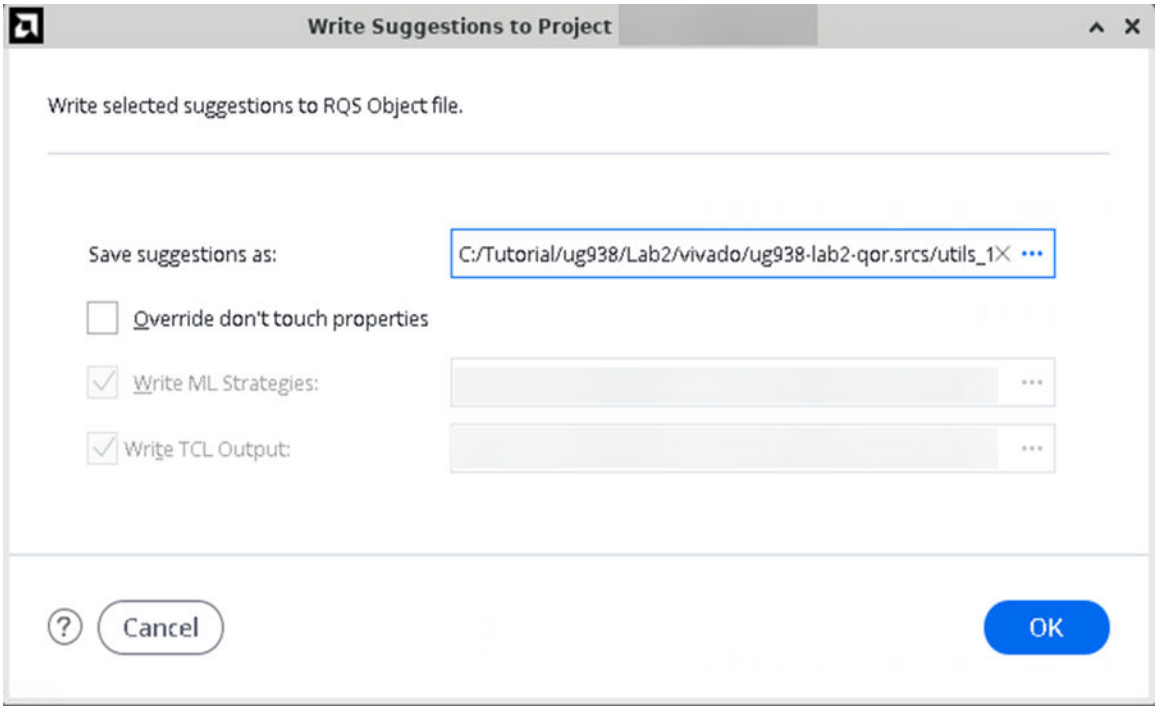


6. In the GENERATED view, you can see the remaining suggestions. The RQS_CLOCK-9 suggestion is applicable for `place_design`. The RQS_NETLIST-19-1 suggestion retimes the driver of a high fanout net to a flop primitive as it improves the timing on a path that could be critical and require driver replication.

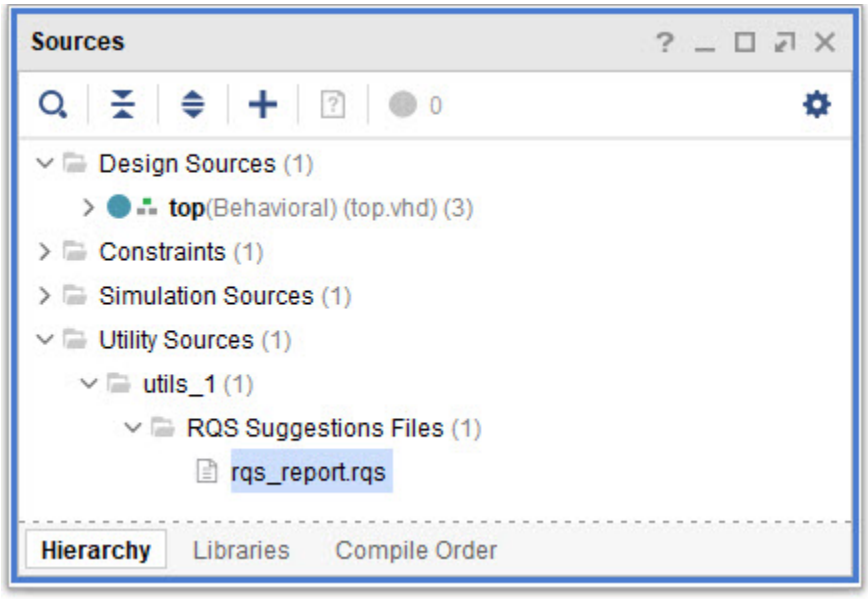
7. You can select the suggestions you write to the file by checking or unchecking the boxes associated with each suggestion. Here we have unchecked the RQS_XDC-1:



8. Click **Write Suggestions**. When the Write Suggestions to Project dialog box opens, ensure it is set to write to the `rqs_report.rqs` file in the `utils_1` directory, as shown in the following figure. Add the file to the project using `add_files -fileset utils_1 rqs_report.rqs`.



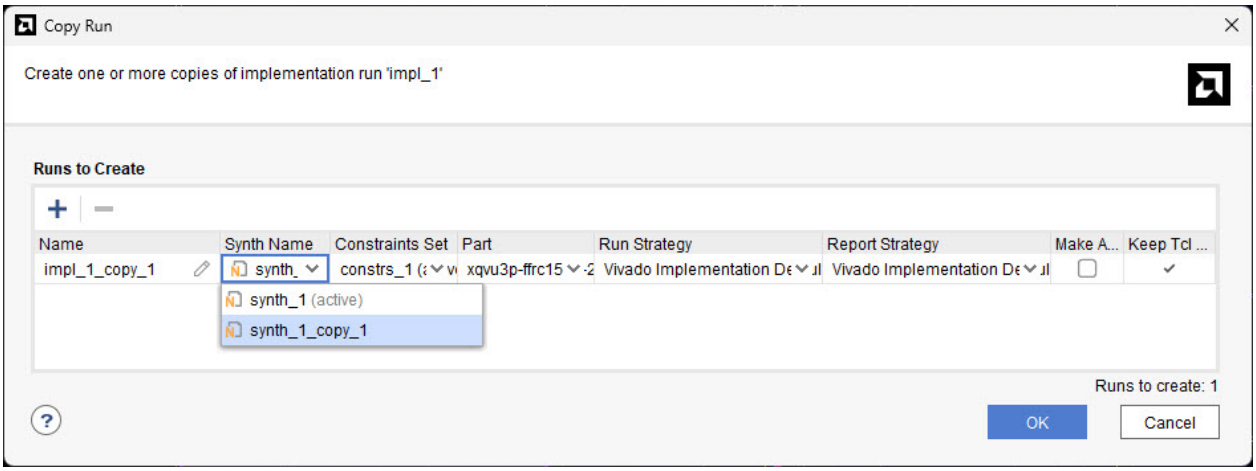
9. Examine the Sources window. The RQS file has been added to the `utils_1` fileset. This ensures that the file is captured using the `get_files` command and recognized in the next step when you add the file to a run.



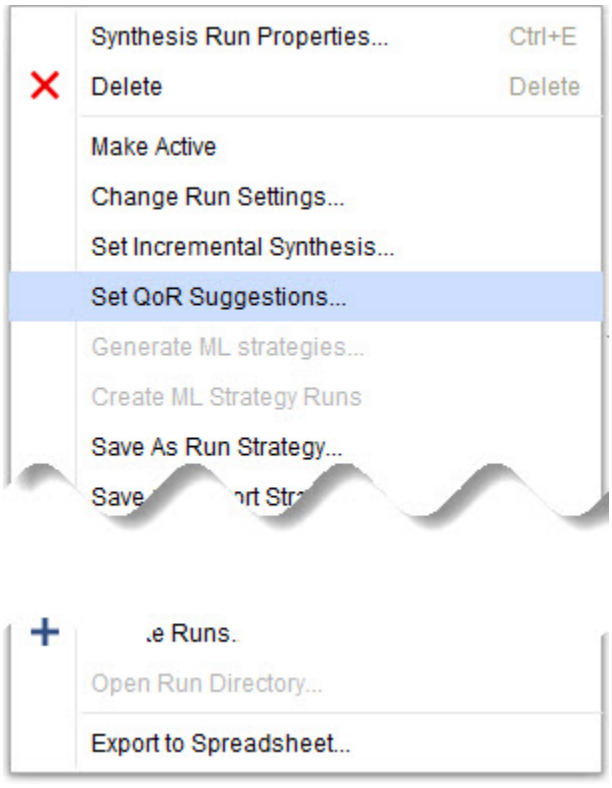
Step 5: Run with Suggestions

In this step, you add a suggestion to a run and examine what happens when a suggestion is applied and how it is reported.

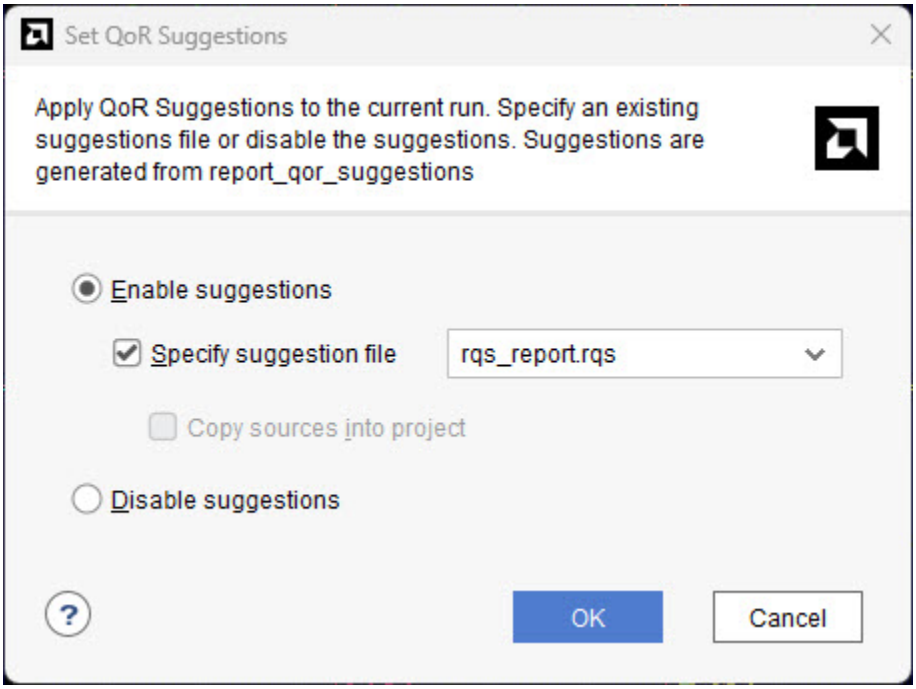
1. In the Design Runs window, right-click on the synthesis run, select **Copy Run**, and click **OK**. Do the same for the implementation run, but update the synthesis run name to the new one you have just created, and click **OK**.



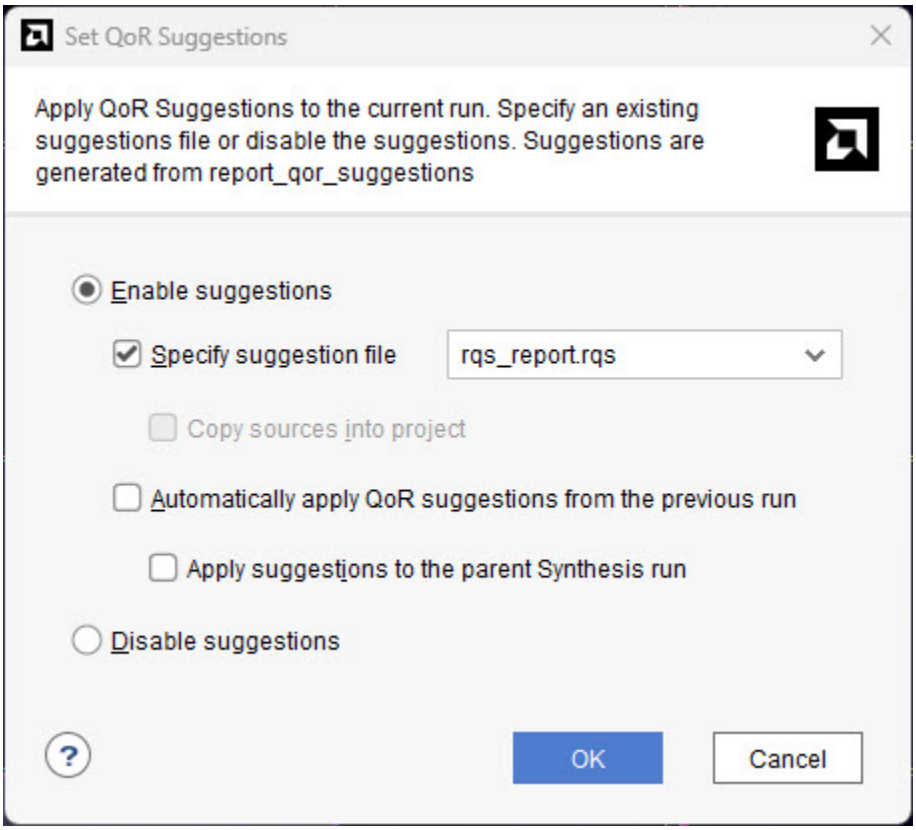
2. In the Design Runs window, right-click the new **synth_1_copy_1** run and select **Set QoR Suggestions**.



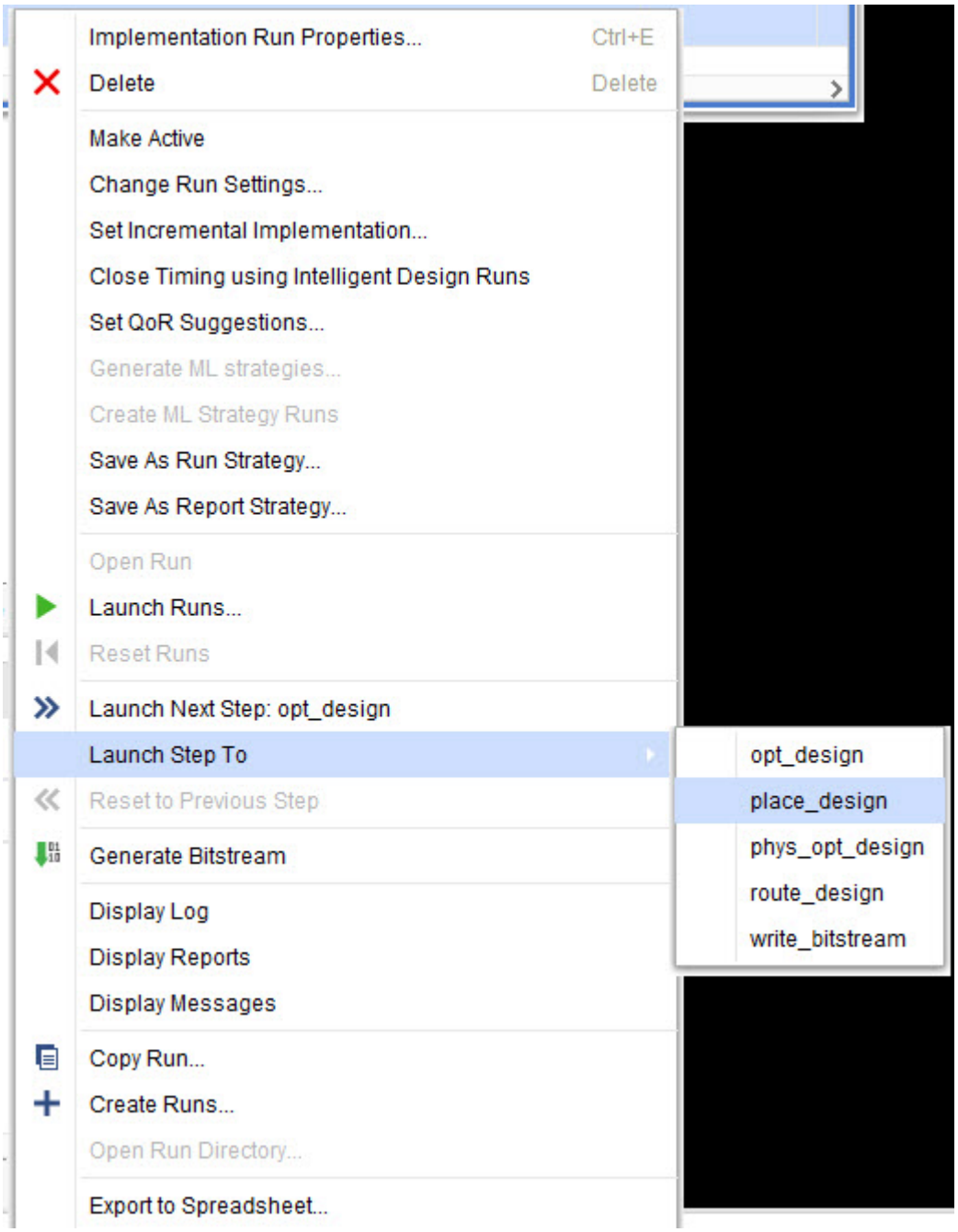
3. Specify the suggestion file as the RQS file added to the project from the previous step and click **OK**.



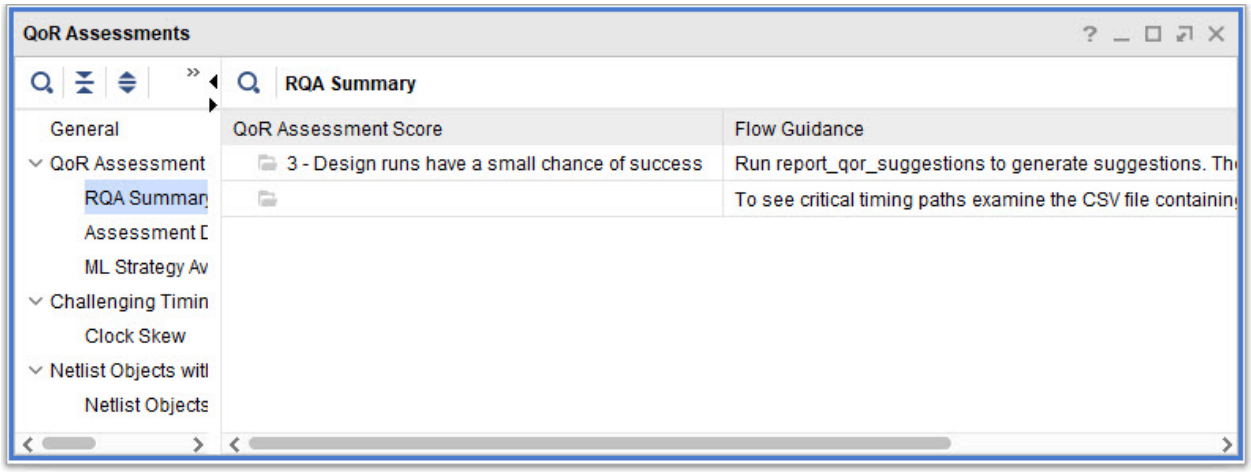
4. Repeat steps 2 and 3 for the implementation run, making sure to specify the new synthesis run as the parent run. Specify the same RQS file for each run. There is a slightly different dialog box for implementation.



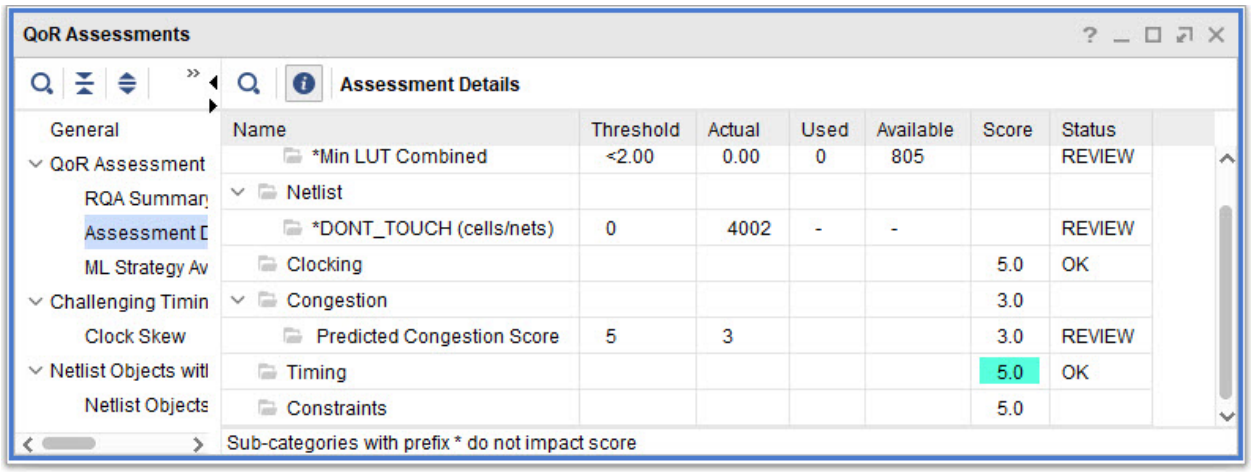
5. In the Design Runs window, right-click **synth_1_copy** and select **Make Active**.
6. In the **Flow Navigator**, click **Run Synthesis**.
7. Because this design takes a long time to route, you only run to `place_design` and analyze at this stage. When synthesis is complete, in the Design Runs window, right-click on the new implementation run and select **Launch Step To → place_design**.



8. With the implementation running, select the Design Runs window. Right-click the **synth_1_copy** synthesis run and click **Open Run**.
9. When the run has opened, select **Reports → QoR Assessment...** and click **OK**.
10. Click **RQA Summary**. The score has improved from 2 to 3.



11. Click **Assessment Details**. The Net and LUT budget score has been eliminated.

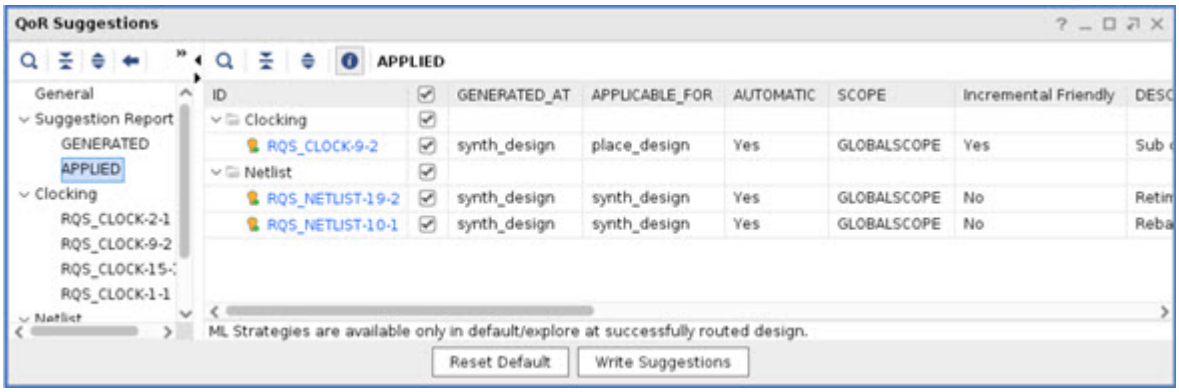


12. Close the synthesized design.
13. When `place_design` is finished, examine the very top of the implementation log file for the new implementation run. It provides a table summary of the suggestions that have been read in. This summary helps you confirm that what has been read in is what you expect.

```
1. Read QOR Suggestions Summary
-------------------------------


Read QOR Suggestions Summary
+-----------------------------------------------+----------------+--------+
|              Suggestion Summary               | Incr Friendly  | Total  |
+-----------------------------------------------+----------------+--------+
| Total Number of Enabled Suggestions           |              1 |      3 |
|   Automatic                                   |              1 |      3 |
|   Manual                                      |              0 |      0 |
|   APPLICABLE_FOR                              |                |        |
|    synth_design                               |              0 |      2 |
|    opt_design                                 |              0 |      0 |
|     That overlap with synthesis suggestions   |              0 |      0 |
|    place_design                               |              1 |      1 |
```

```
|    postplace_phys_opt_design                |              0 |      0 |
|    route_design                             |              0 |      0 |
|    postroute_phys_opt_design                |              0 |      0 |
|   ML Strategy                               |              0 |      0 |
| Total Number of Disabled Suggestions        |              0 |      0 |
+---------------------------------------------+----------------+--------+
```

14. Go to **File → Checkpoint → top_placed.dcp** in the `impl_1_copy_1` run directory. Do not close the current project.
15. In the new instance of the Vivado tools, select **Reports → Report QoR Suggestions ...** and click **OK**.
16. In the new report, there are more sections under Suggestion Report:
    - **GENERATED:** New suggestions are listed in this section.
    - **EXISTING:** Suggestions that existed previously but have not been applied are listed in this section (not shown).
    - **APPLIED:** Suggestions that have been applied are listed in this section.
    - **FAILED TO APPLY:** Suggestions that apply to design objects that no longer exist are listed in this section (not shown).



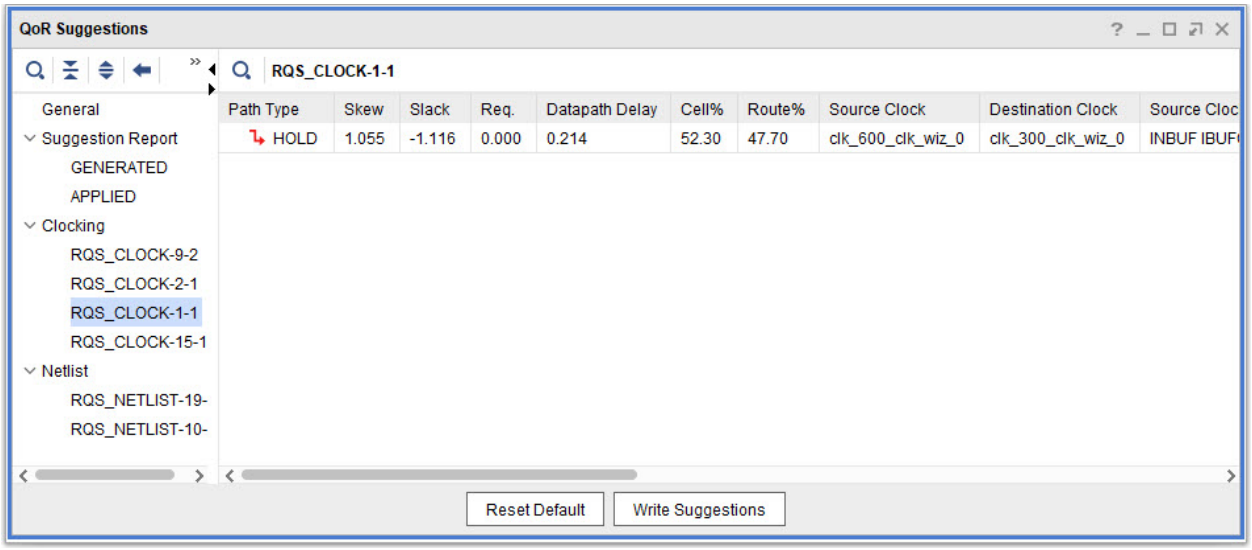
17. Click **APPLIED** and select the details table for one of the items. For APPLIED suggestions, the timing path summary is still available but it is not possible to cross probe to other views in Vivado because some items might have changed.

Step 6: Accumulating Suggestions

You can now review the newly generated suggestions and add them to the RQS file.

1. Click **GENERATED**. The RQS_CLOCK-15 message reports high THS paths but does not provide an automatic suggestion.

2. Examine RQS_CLOCK-2-1. This suggestion recommends changing the clock buffers to BUFGCE_DIV to improve the timing path uncertainty. This is a highly recommended implementation, but it cannot be automated because it requires an RTL edit. If you wish, you can make the recommendation and see the improvement, but this step can be skipped. The next steps focus on the automated suggestions.
3. Click on **RQS_CLOCK-1-1** to view the detailed report. This suggestion applies CLOCK_DELAY_GROUP to related clocks. In this report, you can see that there is a high clock skew and failing slack.



   RQS generates different suggestions at different stages of the flow. This is because the design state changes, and RQS uses the updated information to generate new suggestions. Whenever there is a change in information level, it is advisable to run `report_qor_suggestions`. The following summarizes the changes as you progress through the tool flow:

   - Clocking estimates are accurate after `place_design`.
   - Congestion is available after placement and improves further after routing.
   - Timing estimates improve throughout the flow and are impacted by the number of paths analyzed.

4. Click **Write Suggestions**. When suggestions are written, the APPLIED status is reset. All the previous suggestions and the new RQS_CLOCK-1-1 suggestion are combined into one file. You can overwrite the previous file and reuse the runs, or create a new file and new runs.
5. Select the file location to overwrite the existing file. You can find out the location of this by selecting it in the sources window. Alternatively, it should be at the following location if you have followed the steps carefully: `path <extract_dir>/Lab2/project_2/project_2.srcs/utils_1/imports/impl_1/rqs_report.rqs`.
6. Close the DCP.

7. In the Design Runs window, right-click the implementation run **impl_1_copy_1** and select **Launch runs**. When routing is complete, you can relaunch the run to see the new suggestions take effect.

Step 7: Automatically Running QoR Suggestions

In this section you run the AUTO_RQS flow, where suggestions are generated after `opt_design`, `place_design`, and at the end of the implementation run. In this flow, suggestions are automatically generated, applied, and accumulated without user intervention.

Flow Background: In the AUTO_RQS flow, an optimized list of suggestions is generated in three stages. These suggestions are targeted for maximum improvement in results. The suggestions are generated at the following points:

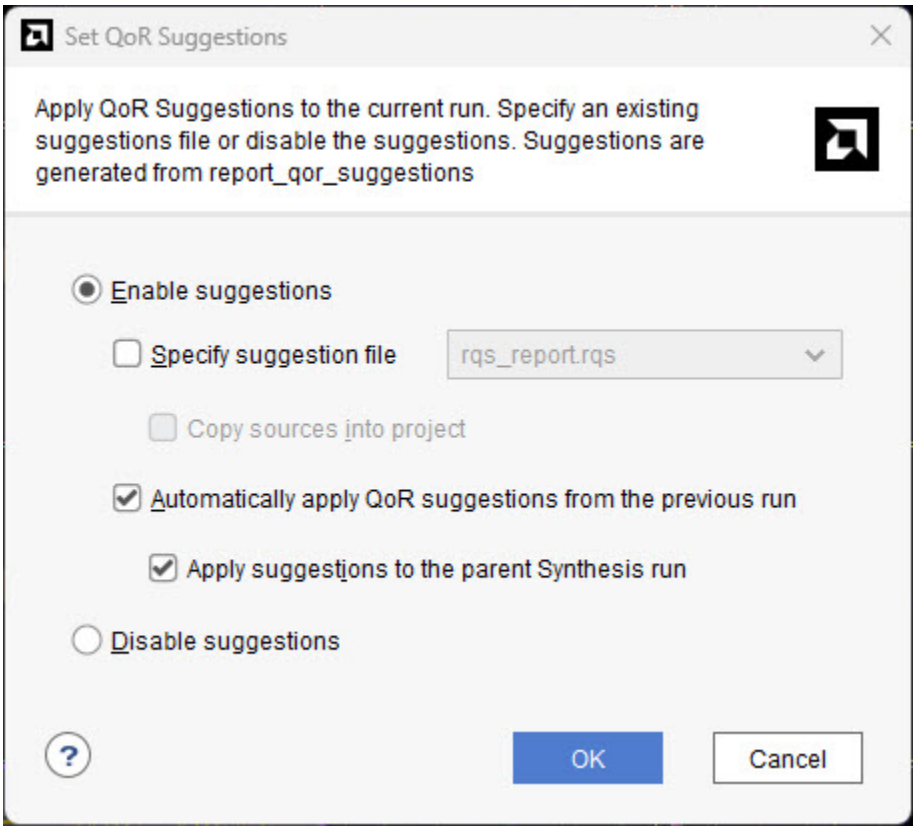
- After `opt_design`. Suggestions are enabled and applied in the current run if the APPLICABLE_FOR stage occurs later in the flow.
- After `place_design`.
- After the final step (`route_design` or post route `phys_opt_design`): These suggestions, along with the place suggestions, are added to the opt_design suggestions and written to an RQS file. When the run is reset, an RQS file will be written and added to the utils_1 source set. In the next run, all the suggestions are applied.

The suggestions written to the RQS file include one or more of the following properties:

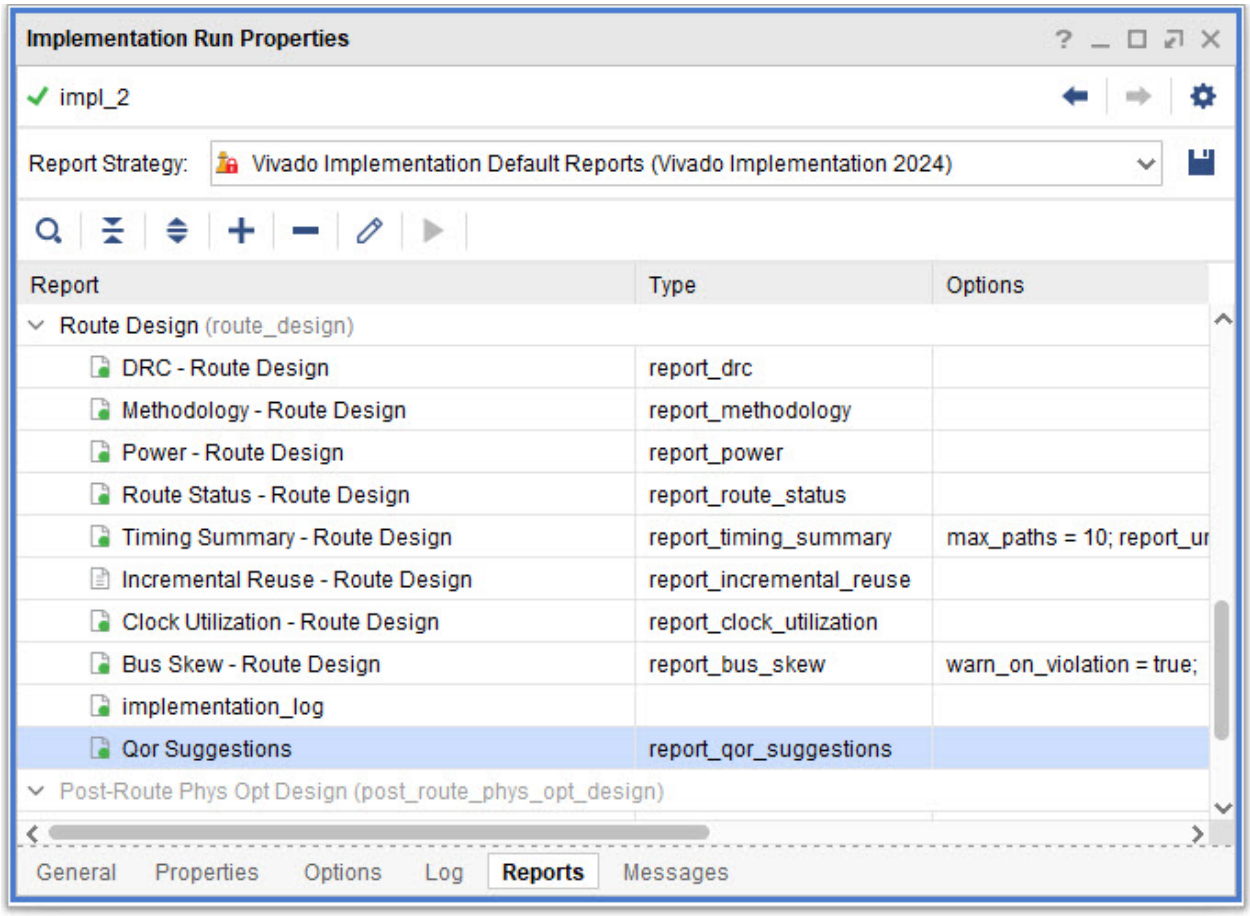
- AUTO_RQS_FLOW
- AUTOMATIC
- APPLIED

Suggestions that do not have these properties are not written to the RQS file and are dropped.

1. In the Design Runs window, right-click the original synthesis and implementation run, an select **Copy**.
2. Right-click **impl_1_copy_2 → Set QoR Suggestions**.
3. Select **Automatically apply QoR suggestions from the previous run** and **Apply suggestions to the parent Synthesis run** check-boxes.



4. Right-click **impl_1_copy_2 → Launch runs**.
5. Once the run completes, go to the Reports tab and select the **QoR Suggestions** report.

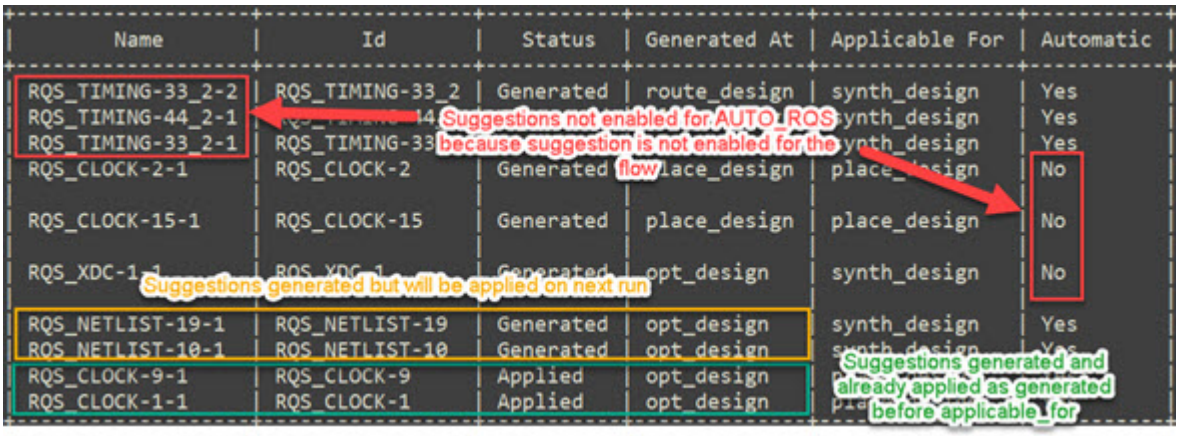


This report is generated automatically when the AUTO RQS flow is enabled. It is generated after `opt_design`, `place_design` and `route_design`. The Route Design report includes information from each step. The other reports are generated during the process of generating suggestions and useful only while the run is ongoing.

6. Examine the report in detail. You may notice:

   - Some suggestions are generated early in the flow but are not applied.
   - Others are generated and applied at the same stage.

   This happens when *applicable for* stage is after the *generated* stage.



Note the *applicable for* value of the suggestions you need to apply. If it is set to `synth_design` you must also reset the synthesis run.

7. In the Design Runs window, note the WNS number. Right click **impl_1_copy_2 → Reset runs**. This also gives the option to reset the synthesis run, make sure you also select this.

   Note: The runs go out-of-date because the RQS file has an updated timestamp or is newly added to the run. The tool does not determine whether synthesis needs to be re-run based on new suggestions. You must decide whether re-running synthesis is necessary to save time.

8. Right-click **impl_1_copy_2 → Launch runs** and launch the implementation run and select the option to restart the synthesis run.

9. Once the run launches, select **impl_1_copy_2** in the Design Runs window. Examine the RQS_FILES property on the Implementation Run Properties window. During the reset run process in step 7, the RQS file is copied from the run directory to the `utils_1` fileset. The RQS_FILES property is updated if required at the same time. RQS files in the implementation run directory are deleted when the run is reset.

   If the RQS_FILES property previously pointed to a different file, all the APPLIED suggestions from this file are copied to the new RQS file and subsequently, this RQS file is no longer required and dropped.

10. When the implementation run completes, verify that the suggestions generated at `place_design` and `route_design` are applied in the new run. There may also be new suggestions generated. As the timing picture changes, suggestions generated can be different at each stage of each run.

Summary

In this lab, you achieved the following:

- Using RQA, you conducted an assessment before and after improvements were implemented, examining an improvement in the score.

- Using RQS, you conducted a complex analysis of a demonstration design. You first examined the reports that showed RQS recommendations to solve implementation problems, then you generated an RQS file and added it to a project implementation run. The Vivado implementation tools executed the suggestions automatically for you. You subsequently performed further analysis and generated further suggestions, accumulating them in the RQS file.

Chapter 3

Running ML Strategies

Introduction

The `report_qor_suggestions` command can generate an implementation design strategy that is predicted to be optimal for the design using machine learning algorithms. In this tutorial, you look at:

- How to generate ML strategy suggestions.
- How to setup the implementation run to use ML strategy suggestions.
- Reporting specifics related to ML strategies.

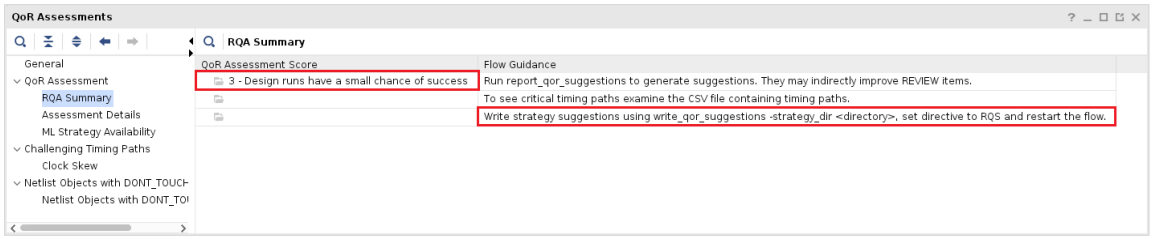
Step 1: Generating an ML Strategy RQS File

This step shows the process of opening a routed design with QoR suggestions and generating a new RQS file with strategies. For details on the design, refer to Step 1: Understanding the Design.

1. In the Vivado Design Suite, source the Tcl file using the command to create the project and run the implementation flow.

   ```
   source <extract_Dir>/Lab3/tcl/create_project.tcl
   ```

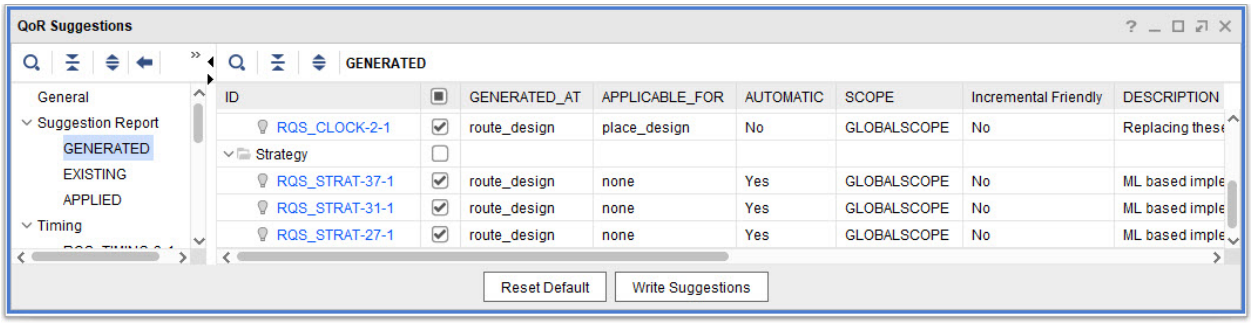
2. In the Flow Navigator, click **Open Implemented Design**.
3. From the pull-down menus, select **Reports → Report QoR Assessment**, and click **OK**.
4. In the RQA Summary table, you see the QoR Assessment Score and Flow Guidance. This table helps identify optimal candidate designs that can effectively leverage ML strategy suggestions. QoR assessment scores of three and above have a chance to meet timing. Designs with an RQA score of less than three are not prevented from generating ML strategies.
5. Click **ML Strategy Availability**. This table details the required directives for the reference run to generate strategies.



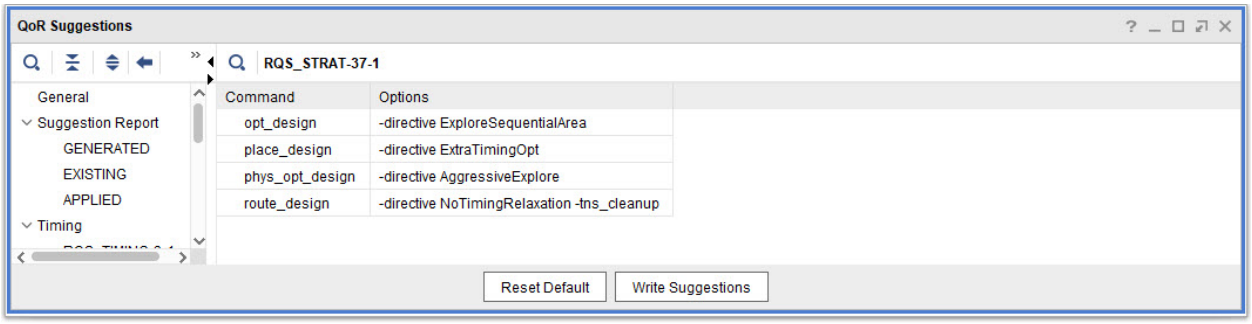
The status for all directives must be listed as OK to generate strategies. The requirements are as follows:

- The `opt_design` directive value must be either Default or Explore.
- The `place_design`, `phys_opt_design`, and `route_design` conditions must be the same as each other and must be set to either Default or Explore.

6. In the Design Runs window, confirm the strategy is Vivado Implementation Defaults. This requirement is met when a design has been run with either the Vivado implementation defaults or the `performance_explore` strategy.
7. From the pull-down menus, select **Reports → Report QoR Suggestions**, and click **OK**.
8. In the QoR suggestion report, select **GENERATED**. Three new strategies have been generated.



9. In the Strategy section, select the topmost strategy. Here, you can see the details of the strategy being suggested. It is possible to set these up manually, but to automate the process more easily, the recommended flow is to read an RQS file containing strategies and set the directive to RQS on the implementation commands.



10. Select **Write Suggestions** to write the following files:
    - A top-level RQS file that does not contain strategies (this file can be ignored)

- An RQS file for each strategy as well as any other QoR suggestions (written to the `MLStrategy` directory).

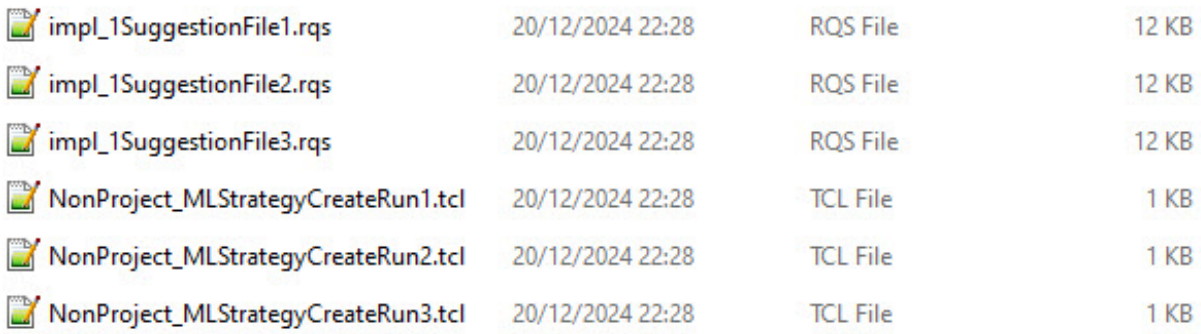
Generating the strategy RQS files is the first part of a two-step process. This way of generating the suggestions gives complete control over what other suggestions are in the RQS file. Other ways to generate these files are as follows:

- Automatic generation using AUTO_RQS flow (as seen in Chapter 2: Using Report QoR Suggestions and Report QoR Assessment for Timing Closure).
- Clicking **Generate ML Strategies** from the right-click menu in the Design Runs window.
- Automatic generation at the end of Stage 1: Design Optimization of Intelligent Design Runs.
- A post-route Tcl hook script that calls both `report_qor_suggestions` and `write_qor_suggestions`.

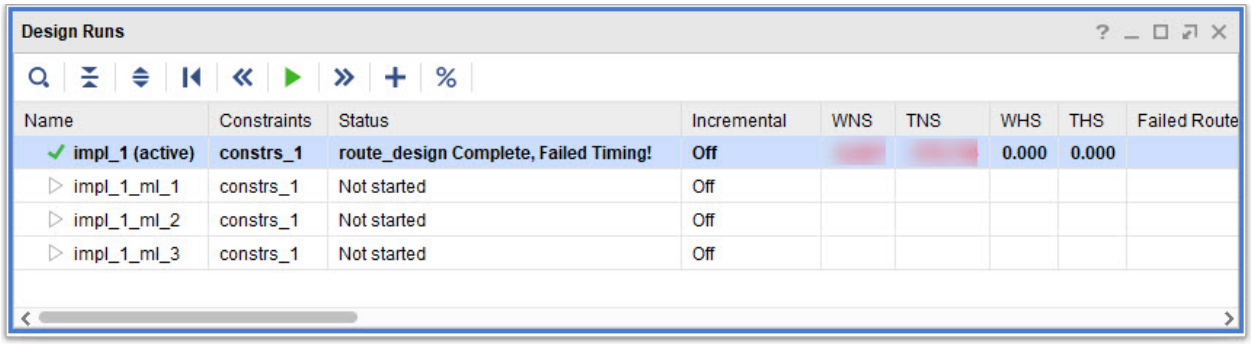
Step 2: Creating ML Strategy Runs

In this step, you create the ML strategy implementation runs using the files created in Step 1: Generating an ML Strategy RQS File.

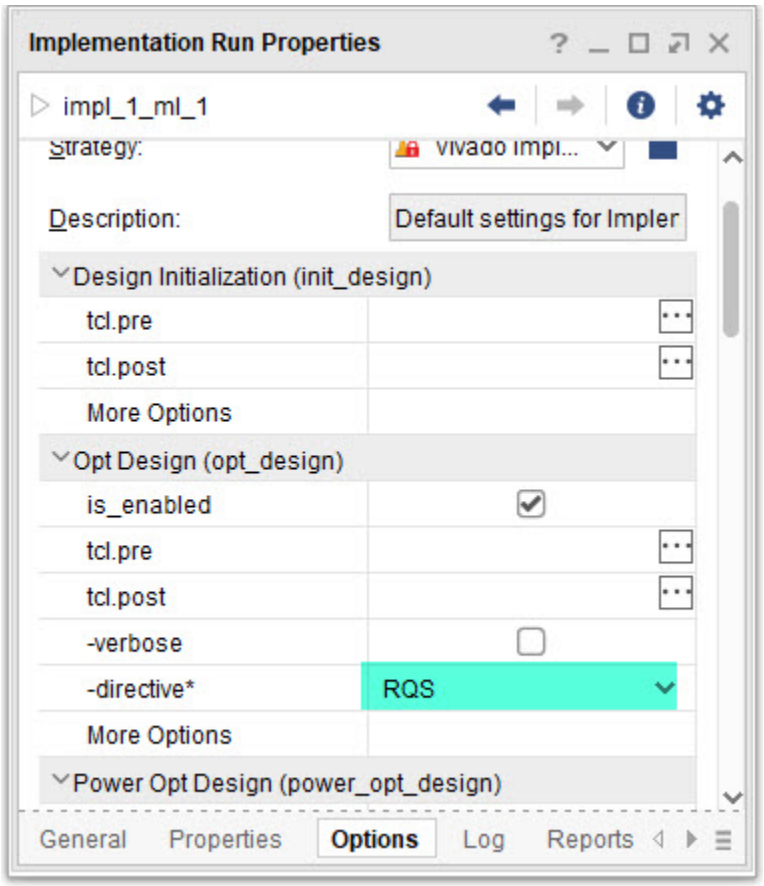
1. In the Design Runs window, select **impl_1**, then right-click and select **Open Run Directory**.
2. Search for the **MLStrategy** directory and examine the contents. You see three RQS files and three non-project-based Tcl scripts. The RQS files are common for both project and non-project flows. The non-project scripts are examples of how to use the RQS file.



3. The next step is to generate the ML strategy runs. As long as the files reside in the `<run_directory>/MLStrategy` directory, this process can be automated. In the Design Runs window, right-click **Impl_1 → Create ML Strategy runs** . Doing this creates three runs, one for each ML strategy file. Configure the runs to use the RQS directive and the RQS file to be read into each of them.
4. In the Design Runs window, select **impl_1_ML_1**.



5. In the Implementation Run Properties window, select the **Properties** tab and confirm that RQS_FILES is set.
6. In the Implementation Run Properties window, select the **Options** and confirm the directive is set to RQS for the `opt_design`, `place_design`, `phys_opt_design`, and `route_design` commands.



You are now set up to run with ML strategies. By the time you have an ML strategy file, you cannot generate new strategies after design changes, but you can add other suggestions.

7. You are now ready to launch the runs. Select all the ML strategy runs, right-click, and select **Launch Runs...**. The runs now proceed in parallel, and complete like a standard run.

Summary

In this lab, you achieved the following:

- Used `report_qor_suggestions` to generate ML strategies.
- Created the RQS and Tcl files using `write_qor_suggestions`.
- Sourced the Tcl to set up the ML strategy runs and confirmed the key aspects of setting up an ML strategy run.

Intelligent Design Runs

Introduction

In this tutorial, you work with intelligent design runs (IDRs). An IDR is a simple-to-use implementation run that extends the performance of your design by automating QoR suggestions and ML strategies. These features are applied in a way that maximizes the improvement in performance.

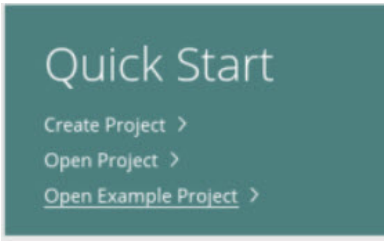
This lab covers how to:

- Create an IDR.
- Add Tcl hooks to the run.
- Examine the log file for an IDR.
- Examine reports for an IDR.
- Create a single pass run from an IDR.

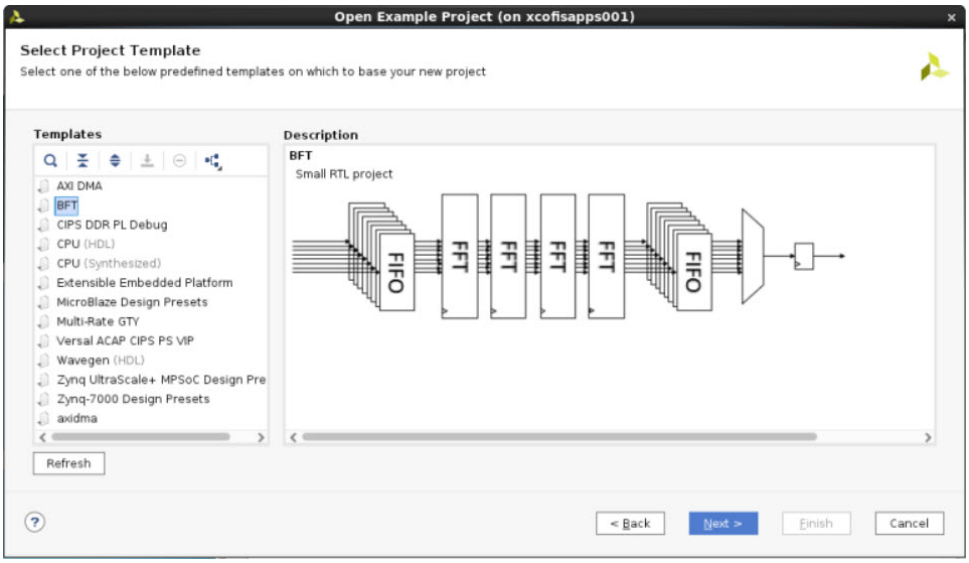
Step 1: Creating Intelligent Design Runs

In this section, you create an example design and an Intelligent Design Run (IDR).

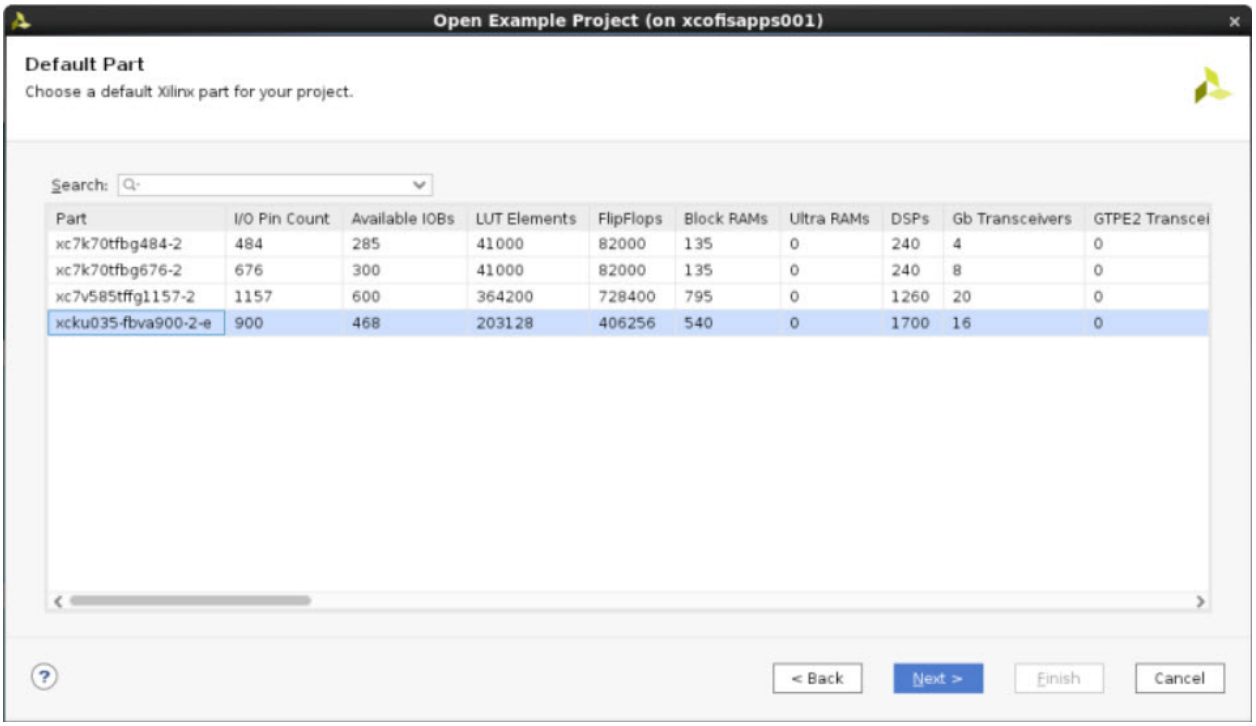
1. First, you need a project. Select **Open Example Project**.



2. Click **Next→ BFT** and then select **Next** again.



3. In the next screen, ensure you are working on in a suitable writable directory, and select **Next**.
4. For part selection, you *must* select **xcku035-fbva900-2-e** and then select **Next → Finish**.

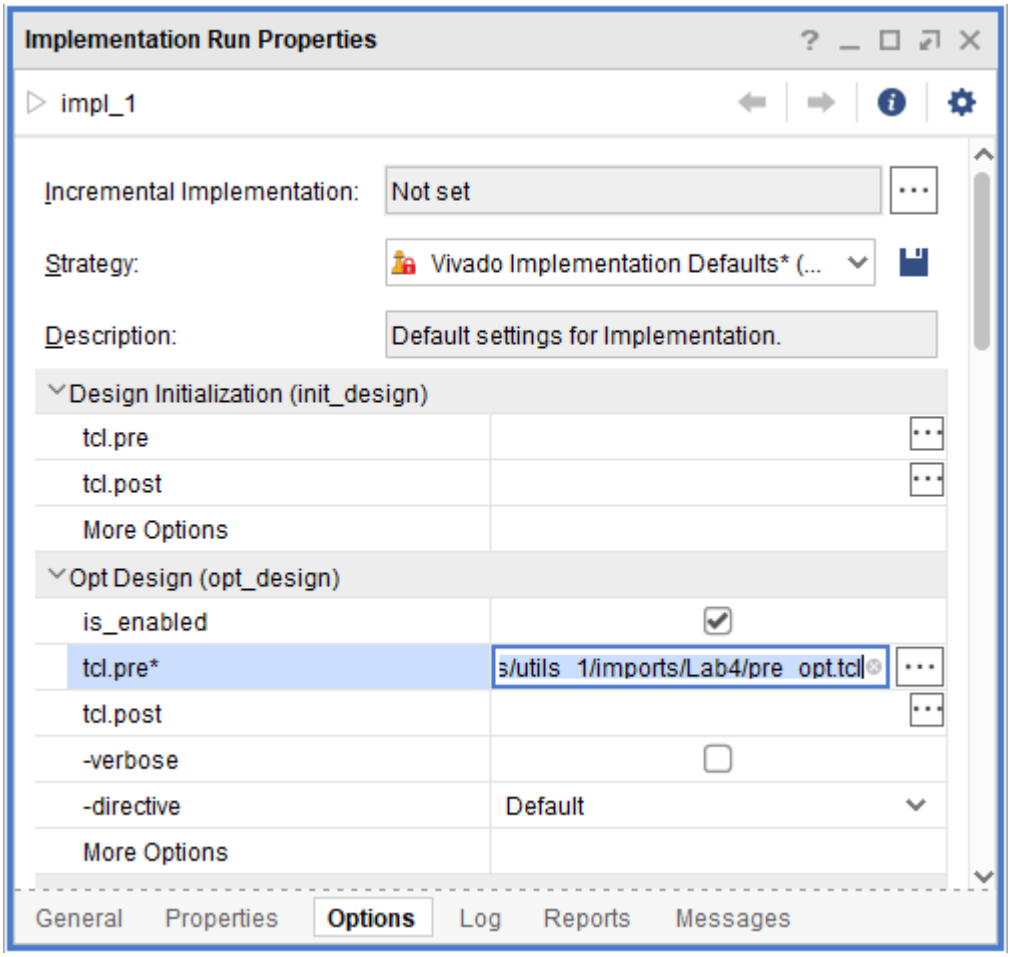


You should now have an open project that is waiting to be synthesized.

***Note*:** IDRs are not supported for 7 series and Versal devices. Selecting the incorrect family requires you to regenerate the project.

5. This lab aims to demonstrate how to add a Tcl file to an IDR. To do this, add a file to impl_1. In the Design Runs window, select **impl_1**.

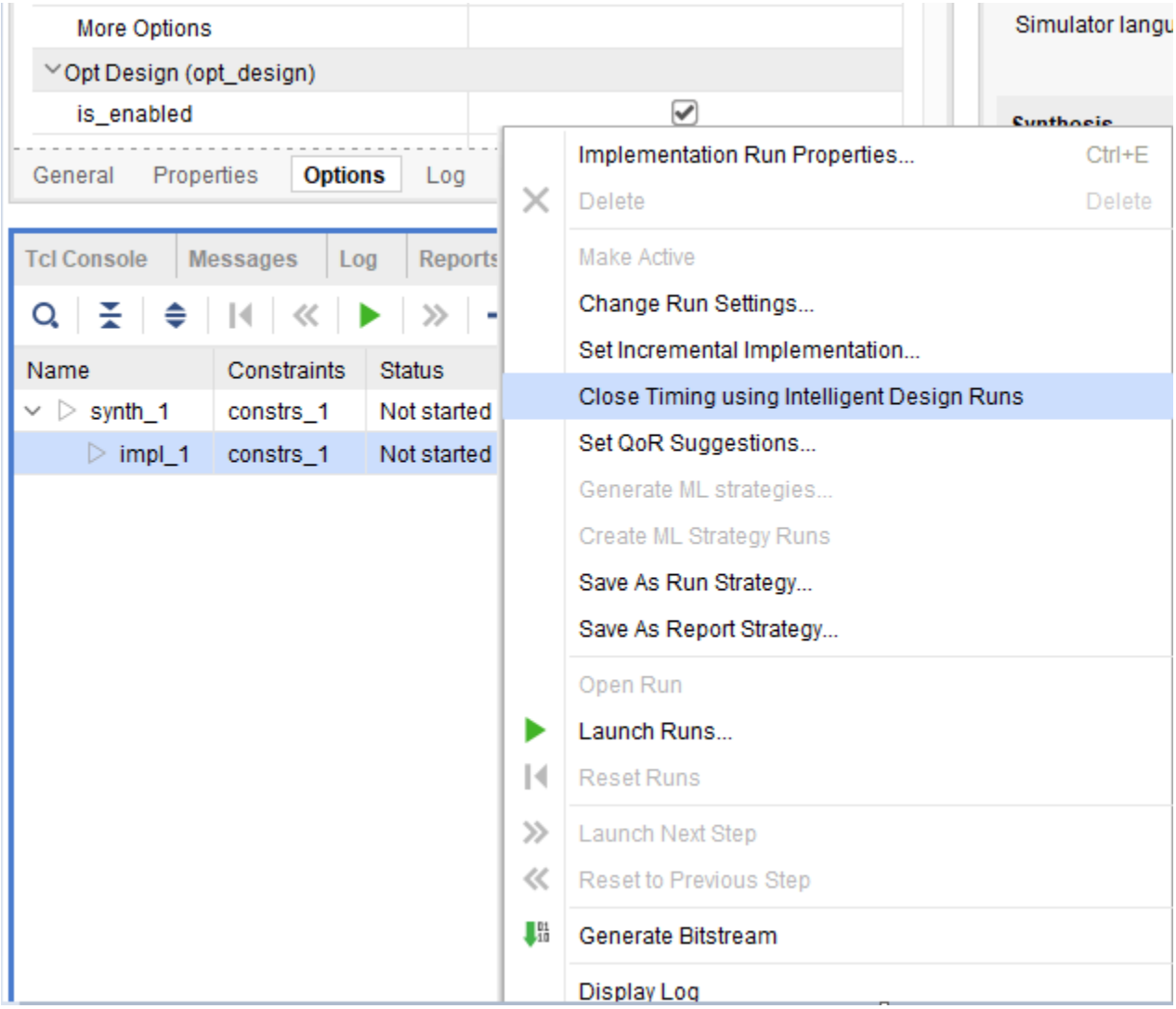
6. In the Implementation Run Properties window, select the **Options** tab.
7. Scroll to the `opt_design` **tcl.pre** option and add the Tcl script `<extract_dir>/Lab4/pre_opt.tcl` to the option.



Alternatively, use the following Tcl syntax:

```
add_files -fileset utils_1 -norecurse <extract_dir>/Lab4/pre_opt.tcl
set_property STEPS.OPT_DESIGN.TCL.PRE [ get_files <extract_dir>/Lab4/
pre_opt.tcl -of [get_fileset utils_1] ] [get_runs impl_1]
```

8. In the Flow Navigator, click **Run Synthesis**.
9. The next step is to create the Intelligent Design Run. In the **Design Runs** window, right-click on **impl_1 → Close Timing using Intelligent Design Runs**.



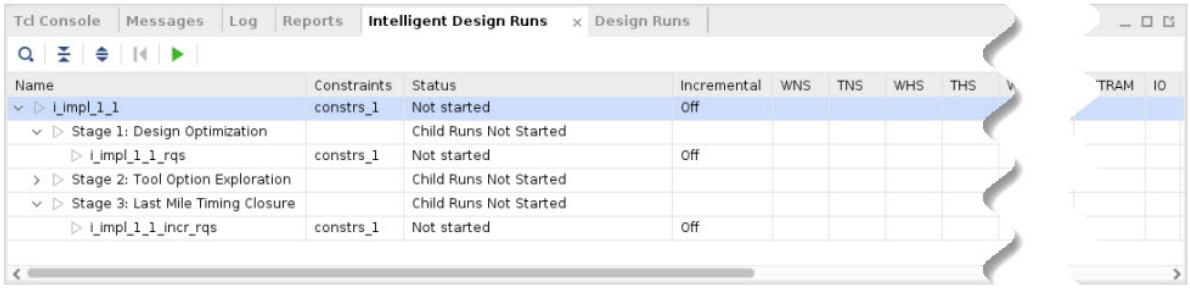
If you try this step again to create an additional run, you find that the option is disabled. This is to prevent the creation of runs that would produce the same results. Runs created from the same netlist with different directives produce the same results because the IDR controls the directive, meaning that they are effectively the same.

If you have a different floorplan or speed grade, create different implementation runs. You can create one IDR from each implementation run.

Step 2: Navigating Intelligent Design Runs

In this step, you learn how to control intelligent design runs and navigate options and reports.

1. Select the **Intelligent Design Runs** tab at the bottom of the AMD Vivado™ IDE.

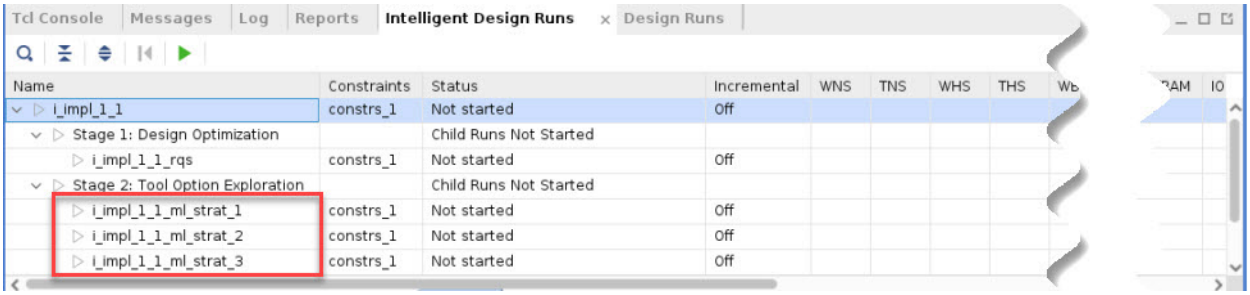


This window is opened when an IDR is created. If closed, it can be reopened by selecting **Window → Intelligent Design Runs**. From this window you can see that there is a hierarchical nature to Intelligent Design Runs. The top-level run is split into three stages:

- Stage 1: Design Optimization
- Stage 2: Tool Option Exploration
- Stage 3: Last Mile Timing Closure

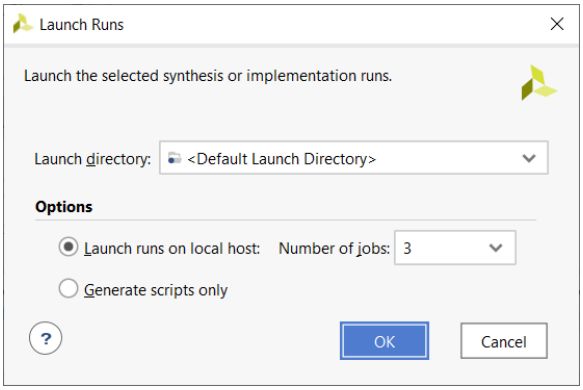
In this lab, only stage 1 is demonstrated. This is the most complex stage, but can be completed quickly because you are working with a simple design.

2. Expand **Stage 2: Tool Option Exploration**.

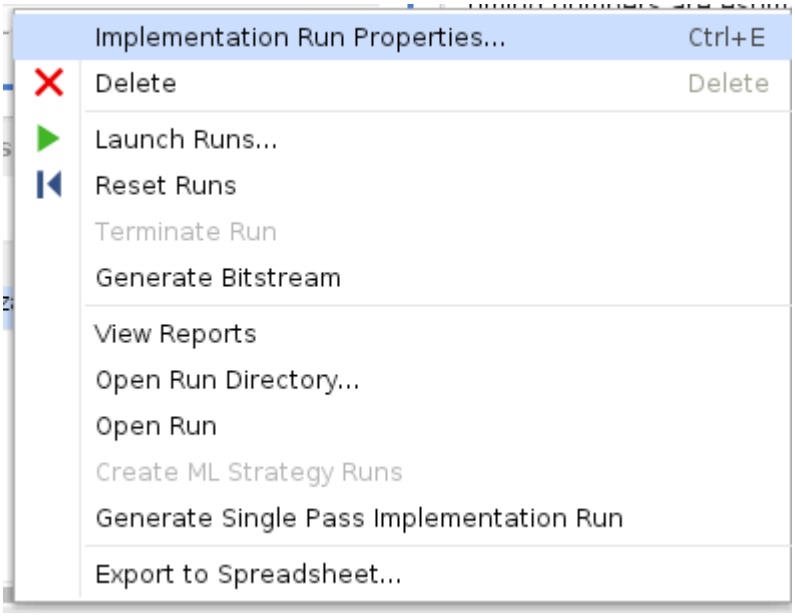


There are three runs underneath this stage. These runs can be run in parallel if your compute resources allow.

3. Right-click **i_impl_1_1 → Launch Runs** and select how you normally run jobs. Click **OK**.



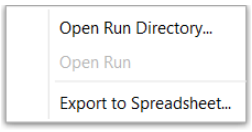
4. Select **impl_1_1** and view the Properties in the Implementation Run Properties window. A PARENT property is set to synth_1. As is the case with a normal implementation run, there is a dependency on the synthesis run to complete when this is set. If the RTL code is modified, the run goes out of date.
5. Click the **Design Runs** tab and confirm that the synth_1 run is running or finished.
6. Click back to **Intelligent Design Runs** and right-click on the **impl_1_1** top-level run.



The right-click menu from the top-level run is the main way to control the Intelligent Design Runs. Some options only become available at specific times:

- Reset Runs is only available after a run is launched.
- View Reports is only available after reports are generated.
- Open Run is only available after a run is successfully completed.
- Generate Single Pass Run is only available after a run is successfully completed.

7. Select the lower-level run, **i_impl_1_1_rqs**, and right-click on it.

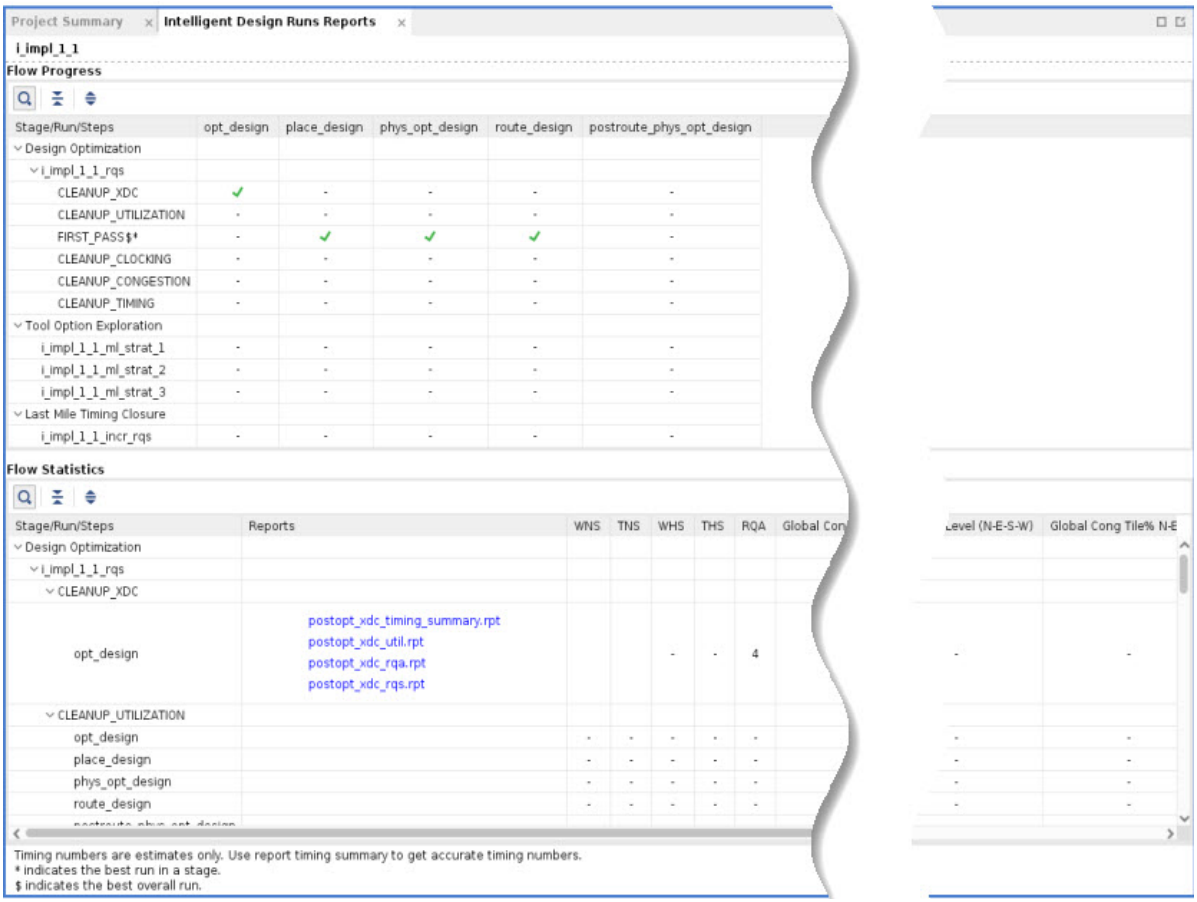


When the lower-level run is selected, you can see that there is a reduced set of options. The Open Run Directory and Open Run options are scoped to the lower-level run. If Open Run is selected, the best completed routed run is opened.

Step 3: Analyzing the Reports and Log File

In this task, you see where to find reports and analyze runs and intermediate checkpoints from an intelligent design run.

1. If the window is not already open, select **Reports → Intelligent Design Run Reports**. This window captures all the metrics that are captured throughout the IDR and provides links to any generated reports.



The report is broken into two windows. The Flow Progress section is in the top half. This details which steps have been run and which steps are running currently. This is more important for an IDR as opposed to a standard implementation run for the following reasons:

- The IDR is a dynamic run, and not all stages are run for every design.
- The steps might be run repeatedly as the run is reset to apply QoR suggestions.

The Flow Statistics section is in the bottom half, and provides the following:

- Hyperlinks to any available reports
- Timing metrics such as WNS/TNS/WHS/THS

- RQA score
- Congestion level and tile % metrics for post-place and post-route designs

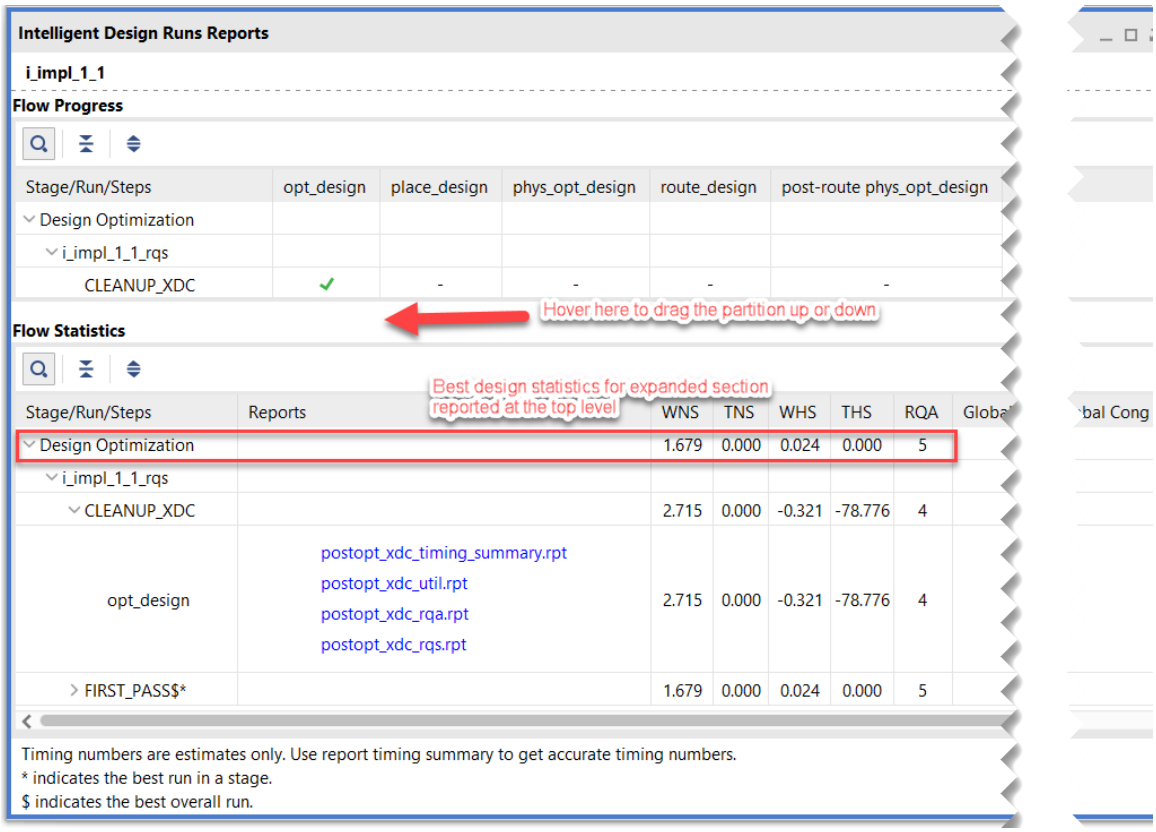
2. Focus in on the **Flow Progress** section of the report.

- Green ticks indicate a completed stage.
- Hyphens indicate the stage is not run.
- Circles indicate the stage is running.

Because there are no circle icons, you can infer that the run has completed successfully. Note also the FIRST_PASS step that is generated for reporting purposes. This special step occurs when there are no utilization or clock suggestions. Only designs without these suggestions have this step.

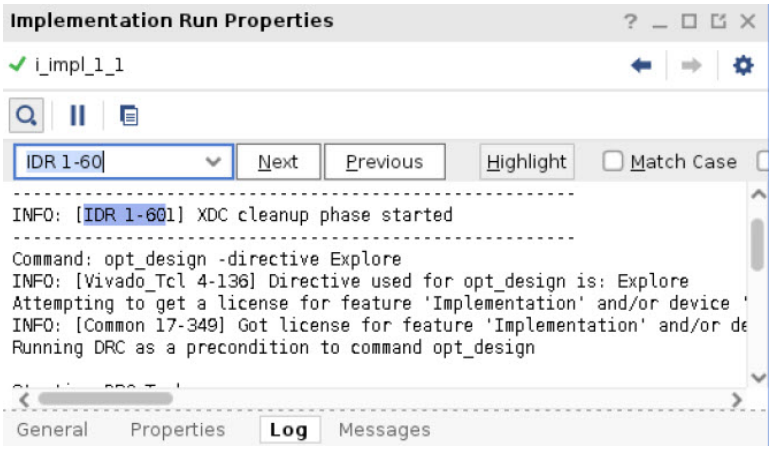
At the bottom, there is a table footer that describes the meaning of the $ and * notations. These notations help you identify which runs you are opening when you select **Open Run**.

3. Look at the **Flow Statistics** section of the report. This report can be larger, and you can make more space by reducing the divider between the section above and clicking on the arrows to reduce sections of the report.



On the left of this window are the reports, in the middle are the timing and RQA statistics, and on the right are the congestion metrics. Statistics are only available if the flow stage has been run, or if the flow stage generates the statistics. For example, during place, hold statistics are not generated.

4. Expand **FIRST_PASS** and click **post_place_physopt_def_util.rpt.**. Pay attention to the name of this report: postplace_phys_opt indicates the implementation step and first_pass indicates the flow step. When you are ready, close the report.
5. In the Intelligent Design Runs window, right-click **impl_1_1** and select **Open Run Directory**. In this directory, open the **idr_flow_summary.rpt** file. This is the text equivalent to the IDE report. It contains the same information except the links to the report. When you are ready, close the file.
6. In the directory explorer, go up one level to the project_1.runs folder. There is a directory for the main IDR and each of the sub-runs. Click into **impl_1_1_rqs**. Here, you can see all the reports and intermediate checkpoints. When you are ready, close the directory explorer.
7. In the Intelligent Design Runs window, select the top-level run **impl_1_1**.
8. In the Implementation Run Properties window, select the **Log** tab and maximize the window.
9. Select the search icon and enter "IDR 1-60". Click **Next** multiple times to navigate through the log file. This ID highlights the start and end banners that have been added when you enter and exit a step within Stage 1: Design Optimization.



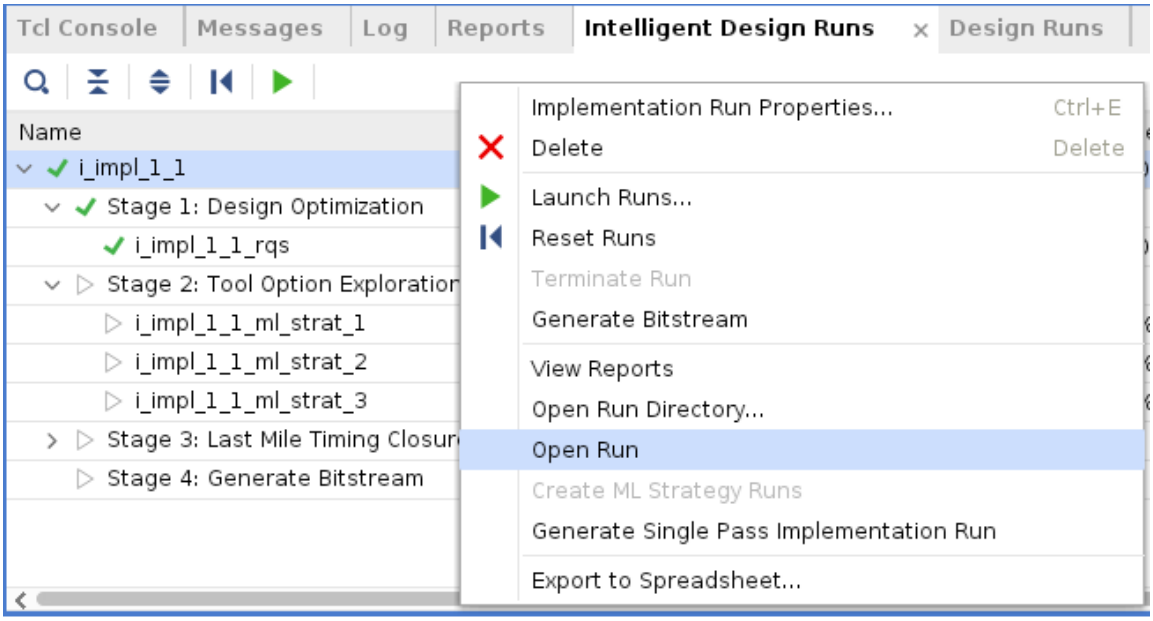
It can be difficult to identify which step you are in as the implementation process goes back and forth, . These messages can help clarify this.

10. Search for "INFO-TUTORIAL4." This has been inserted by the Tcl script you added in Step 1. This Tcl script is executed every time `opt_design` is called. In this case, `opt_design` is called only once, so it is similar to a normal flow.

Step 4: Final Analysis

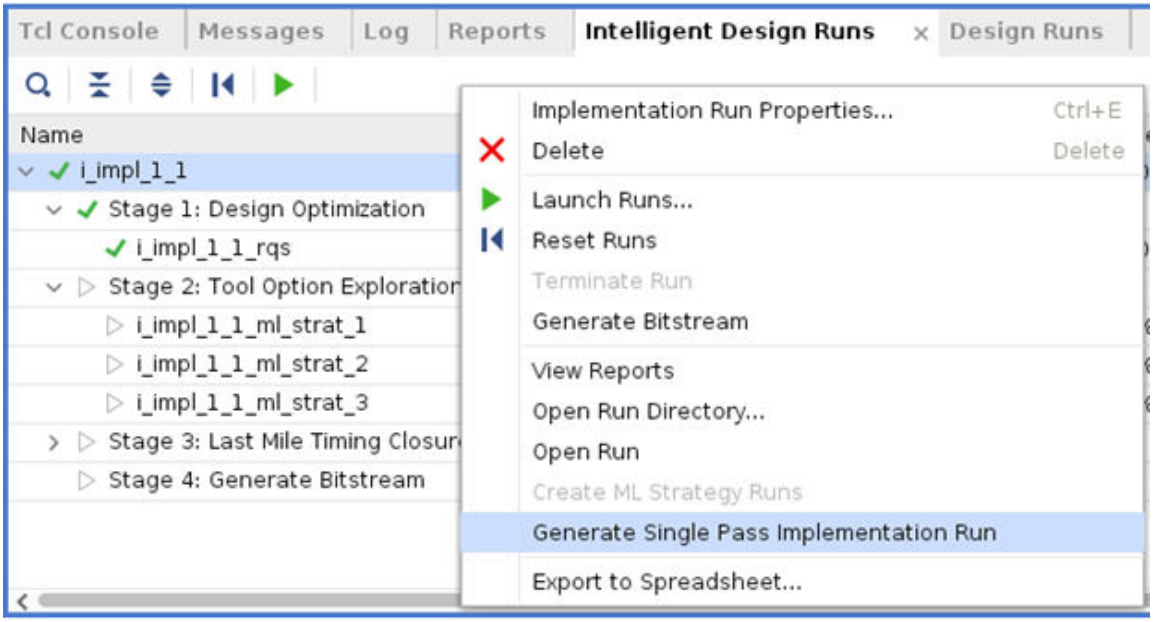
In this task, you see how to analyze the design and create a single pass run.

1. In the Intelligent Design Runs window, right-click on the top-level run, **impl_1_1**, and then click **Open Run**. Doing this opens the best overall run from the three stages. This is also the best run from stage 1. Stage 2 and stage 3 were not run with this design. The best run and best stage run are indicated by the $ and * notations.

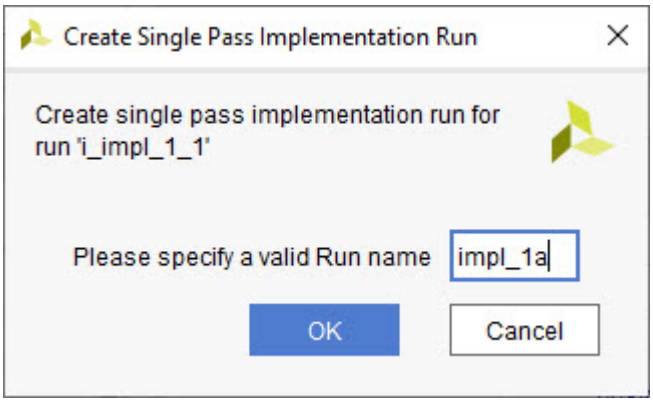


A device view opens where you can generate reports and analyze the design as you would with a normal implementation run. When you are ready, close the design.

2. In the Intelligent Design Runs window, right-click on the top-level **impl_1_1_rqs** and click **Open Run Directory**. This directory has all the intermediate checkpoints from the stage. You can open a new instance of Vivado and open the checkpoints for further analysis. If the DCP is associated with Vivado, you can open the checkpoints by double-clicking on them.
3. When you are finished with analyzing the design, because an IDR can take the equivalent of many implementation runs to complete, it is recommended to run the required commands and suggestions in a single implementation run. In the Intelligent Design Runs window, right-click on the top-level **impl_1_1** and select **Generate Single Pass Implementation Run**.



4. In the Create Single Pass Implementation dialog box, enter a run name, `impl1_a`, and click **OK**.



5. Switch to the **Design Runs** window. You can see that the new run has been made. Right-click on the run and select **Launch Runs**. Confirm that the results are the same as the IDR.

Note: For more difficult runs, you can examine the RQS_FILES property and directives, but because this is a design that meets timing easily, these are not set.

Summary

In this tutorial, you learned how to:

- Create and launch intelligent design runs.
- Analyze reports and checkpoints generated by an IDR.
- Create a single pass flow.

Using the Dataflow Viewer

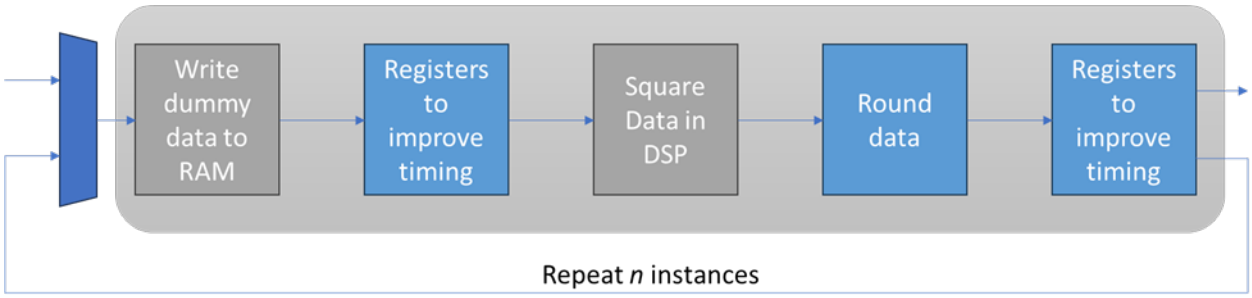
In this tutorial, you learn to use Dataflow Viewer (DFV) to analyze a design. The Dataflow Viewer allows you to better understand key parts of a design that impact overall place and route quality. This is achieved by:

1. Identifying wide, high bandwidth buses in the stripped-down dataflow optimized netlist.
2. Identifying the cells associated with these buses whose placement have a significant impact on the placement quality of the design.
3. Generating paths of multiple cells related to the target wide buses that facilitate a more holistic view of placement.
4. Generating a floorplan that joins associated cells on a bus together.

Step 1: Opening the Project and Understanding the Design

Open the following checkpoint within the folder: `<extract_dir>/Lab5/part_filler_dsp_ram_routed.dcp`. The design in the project is a module named `part_filler_dsp_ram`. This module is a chain of smaller modules that do the following:

Figure 4: **Repeat n instances**



The design works as follows:

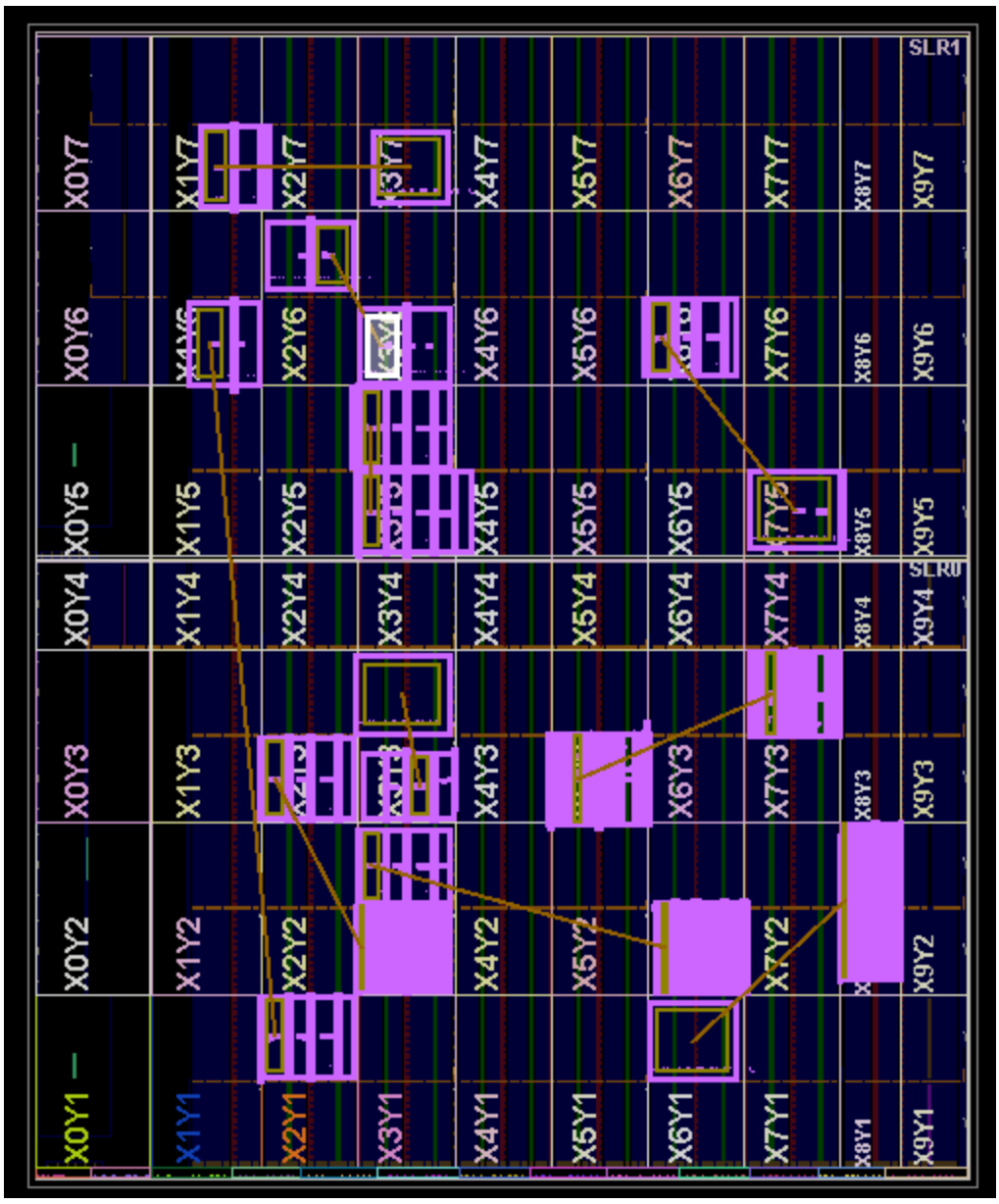
1. There is one memory in the design per DSP. Dummy data is written to and then read from this memory.

2. The read data is registered.
3. The registered data is squared in pipelined DSP.
4. The bit expansion is then controlled by doing some rounding.
5. The rounded data is registered.
6. The registered data is written to the next instance, RAM. The chain repeats this.

The blocks highlighted in blue contain cells that are lower level cells, whilst the grey boxes contain cells that are placement drivers.

The cells are floorplanned to force the datapath to zig-zags through the part. The following image shows the floorplanning:

Figure 5: **Floorplanning**



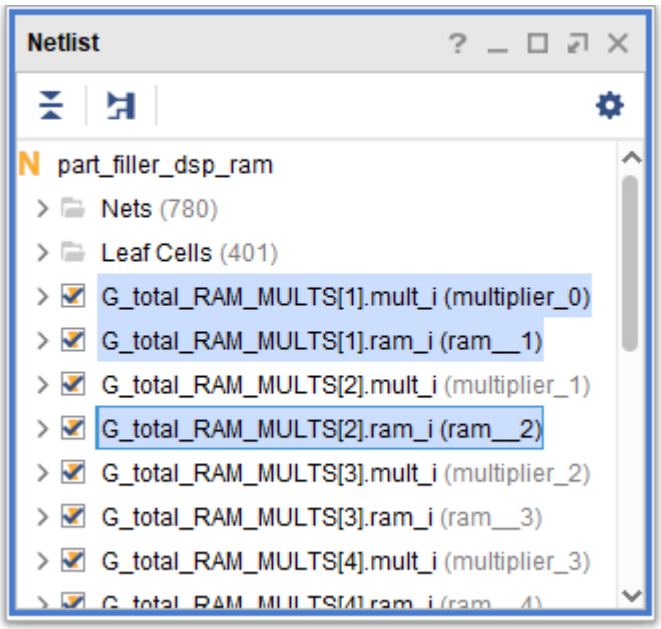
By design, the floorplan is disorganized with cell placement scattered and cells not in their ideal positions. The ideal placement for this design is to locate the cells next to each other in the following order:

1. G_total_RAM_MULTS[1].ram_i
2. G_total_RAM_MULTS[1].mult_i
3. G_total_RAM_MULTS[2].ram_i
4. G_total_RAM_MULTS[2].mult_i

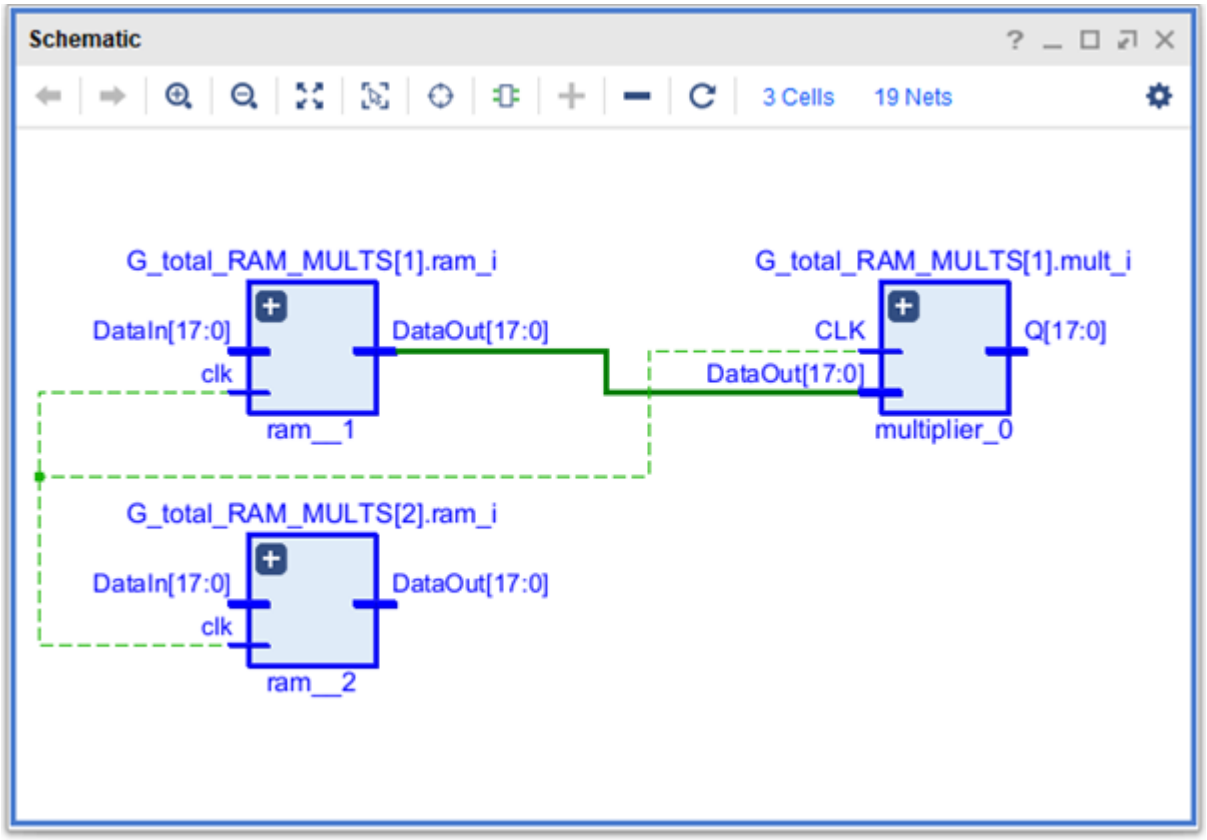
5. G_total_RAM_MULTS[3].ram_i
6. G_total_RAM_MULTS[3].mult_i

And so on upto G_total_RAM_MULTS[10].mult_i.

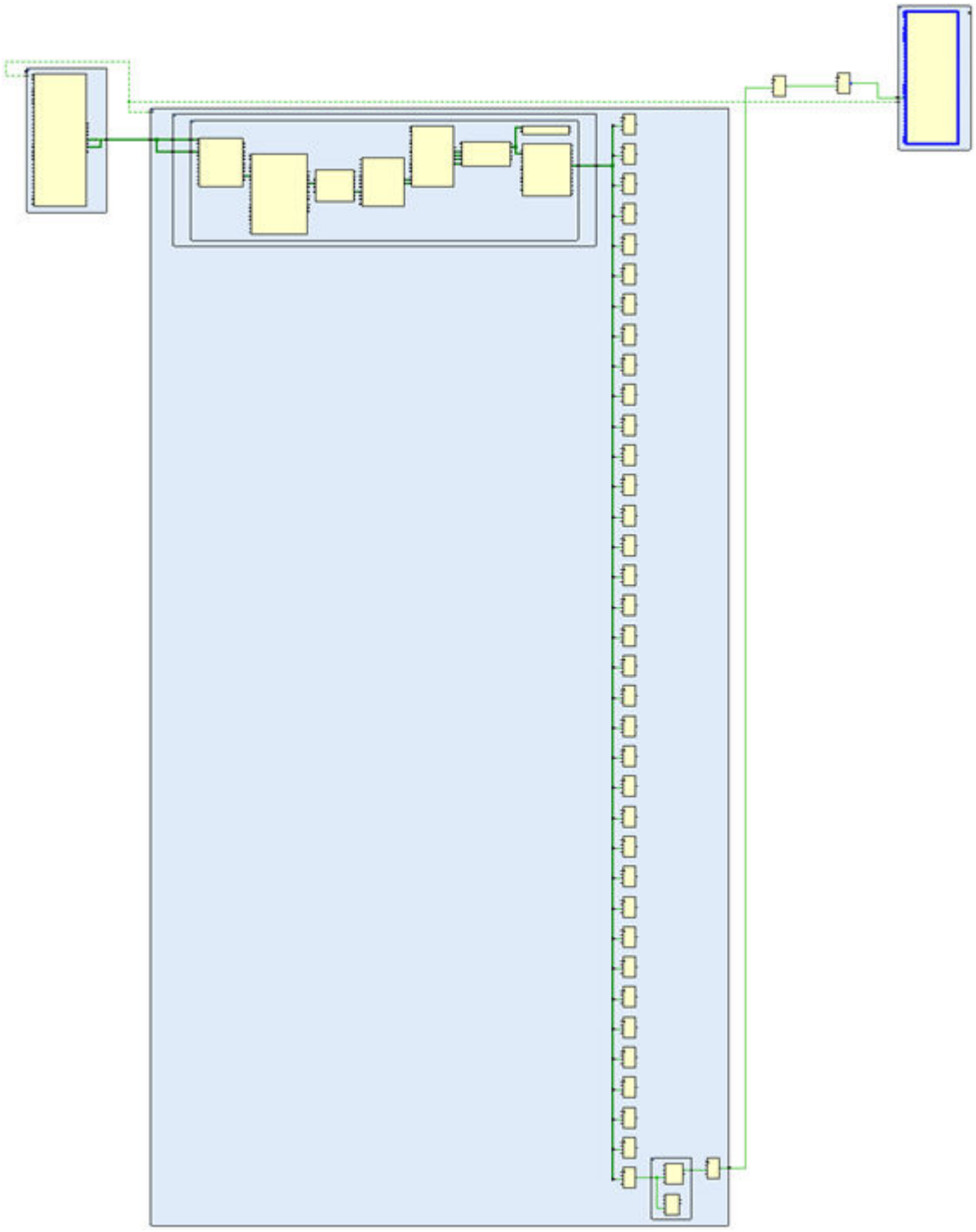
1. Select some of the modules in the netlist view and explore the connectivity. In particular look at the connectivity between RAM[n] -> MULT[n] blocks and MULT[n] -> RAM[n+1]. Pay particular attention to the longer net bundles between two Pblocks further away. Alternatively, you can also open the Physical Constraints window and select each Pblock.
2. Next, observe the connectivity between two RAMs. In the Netlist window, select the following 3 cells:
   - G_total_RAM_MULTS[1].ram_i
   - G_total_RAM_MULTS[1].mult_i
   - G_total_RAM_MULTS[2].ram_i



   - Next, right click and create a schematic



- Next, expand the schematic to have the RAM and DSP leaf cells in view in the schematic and at least one bit of a bus connecting between cells. Allocate a brief amount of time to this task. If you encounter challenges beyond this timeframe, it is acceptable to discontinue your efforts. If successful, the results are such:



To help you get to this point:

1. First expand back into G_total_RAM_MULTS[1].ram_i
2. In the DSP slice click on the following output pins
   a. A1DATA
   b. A2A1

c. V
d. V_DATA
e. X
f. ALU_OUT
g. P

3. Only click on D and Q pins of flops. Use the MSB of the bus when clicking through.
4. Click on the O pin of LUTCY1

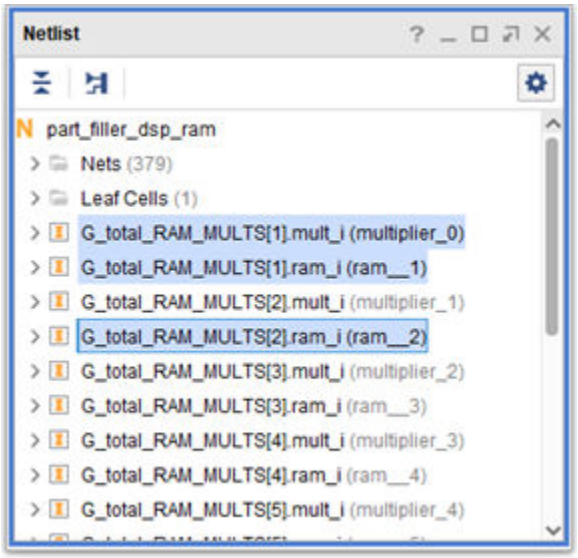
This takes around 15 clicks to get between the two cells. It is complicated and confusing as you go through the LOOKAHEAD and the DSP slice. If you find nets looping around, ask Vivado to redraw the schematic by clicking on the Regenerate icon .

This simple process demonstrates that tracing between cells can be complex in the standard view. Be informed, this is a simple design. Next step traces the same path in the Dataflow Viewer.

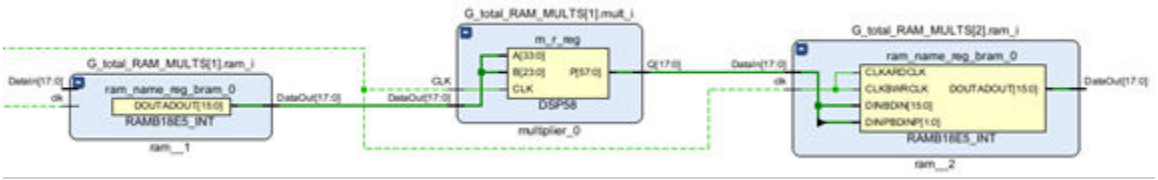
Step 2: Navigating the Dataflow Viewer

Following are the steps to create Dataflow Viewer.

1. In the Flow menu, select **Open Dataflow Design**.
2. This creates the Dataflow Netlist and it becomes the active window. The Netlist window shows the cells from the dataflow design.
3. Next, select the same three cells as previously selected in the Netlist window.



4. Right click and generate a schematic. Click on the input pin of the second RAM and to make the schematic more readable. click on the redraw symbol. The schematic expansion is significantly faster than tracing through a standard netlist:

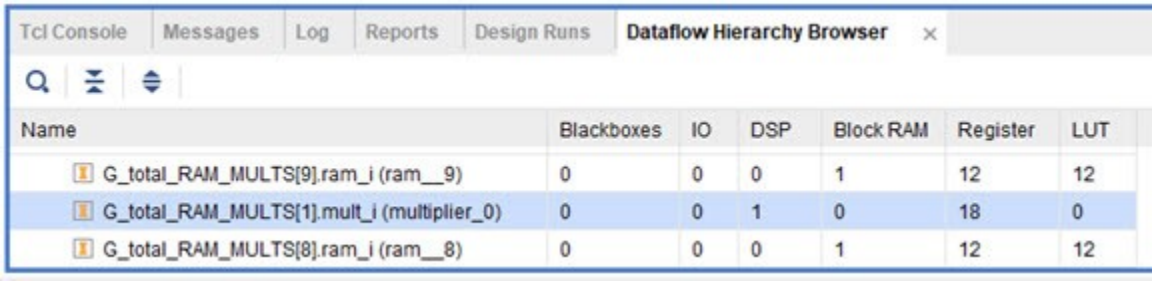


When comparing this to the schematic on the previous page, you understand how this process is quicker:

- Internal cells are not displayed from within the DSP
- Registers are removed
- LUTs are removed
- Only RAMs and DSPs remain

This netlist simplification translates into a faster navigation speed. This comes at the cost of low level elements within the design no longer being visible.

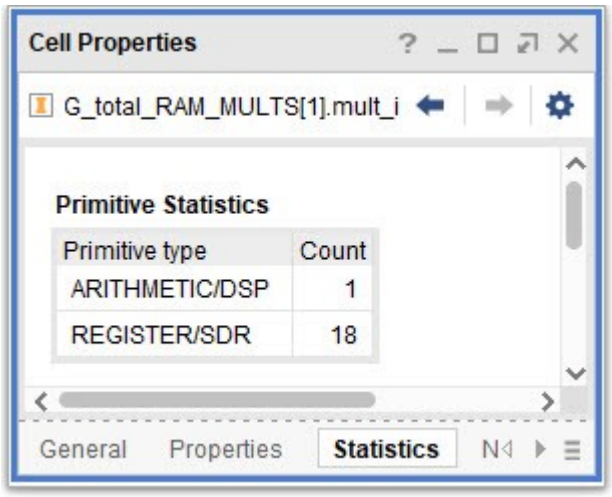
5. Click on the hierarchy around the DSP slice and observe the Dataflow Hierarchy Browser. The following should be shown:



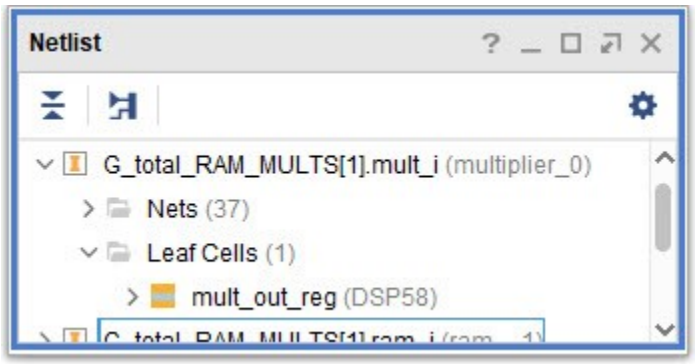
Name	Blackboxes	IO	DSP	Block RAM	Register	LUT
G_total_RAM_MULTS[9].ram_i (ram__9)	0	0	0	1	12	12
G_total_RAM_MULTS[1].mult_i (multiplier_0)	0	0	1	0	18	0
G_total_RAM_MULTS[8].ram_i (ram__8)	0	0	0	1	12	12

The cross probing within the dataflow viewer functions similarly to other windows in Vivado. Observe that the Dataflow Hierarchy Browser shows the resource utilization of the selected hierarchies.

6. Next, in the Cell Properties window, click on **Statistics**. This shows the following window:



7. Next, expand the Leaf Cells under the level of selected hierarchy. It only has one cell. The reason for this is that some statistics are taken from the full design.



8. Next step is to look at switching between designs (to distinguish from switching activity for power analysis). When you open a dataflow design, there are two netlists loaded into memory, the parent design and the dataflow design. Only one can be viewed at a time. To see the other, you need to switch designs. With the G_total_RAM_MULTS[1].mult_i hierarchy level still selected, click on the design switch icon at the top of the screen. This displays the parent design, and the selected hierarchy from the dataflow netlist is selected to facilitate cross probing between different designs. In the netlist window, expand the cells and you can see the registers.
9. Switch back to the dataflow design using the switch icon.

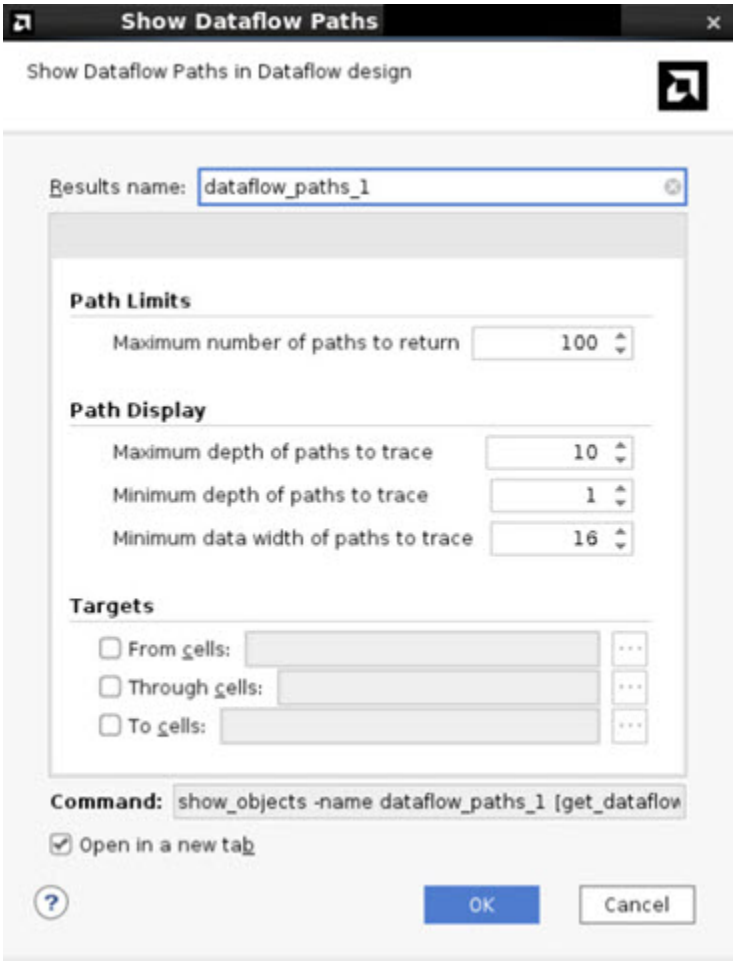
   ***Note*:** It slightly changes its appearance and looks like .

Step 3: Generating Dataflow Paths

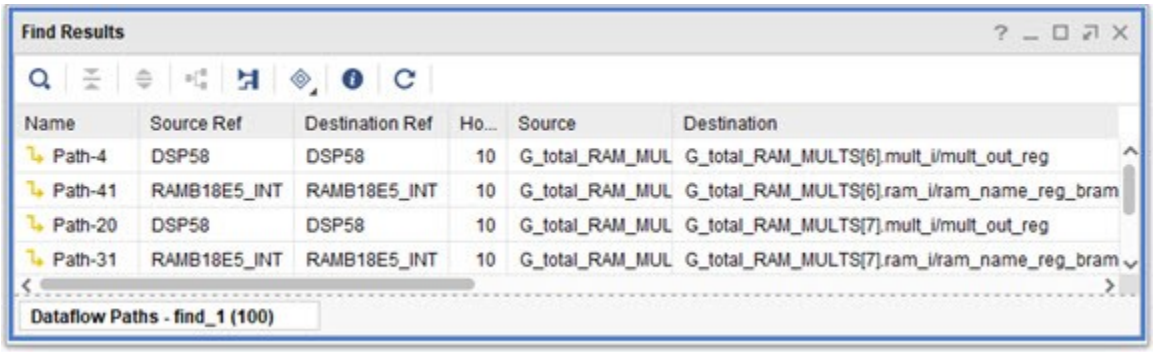
This step explains how to generate dataflow paths. These are objects that contain paths with multiple cells. These allow you to:

- Trace the connectivity based on the longer chain connections seen in the design
- Target analysis to specific paths in the design
- View connectivity between one object type and another

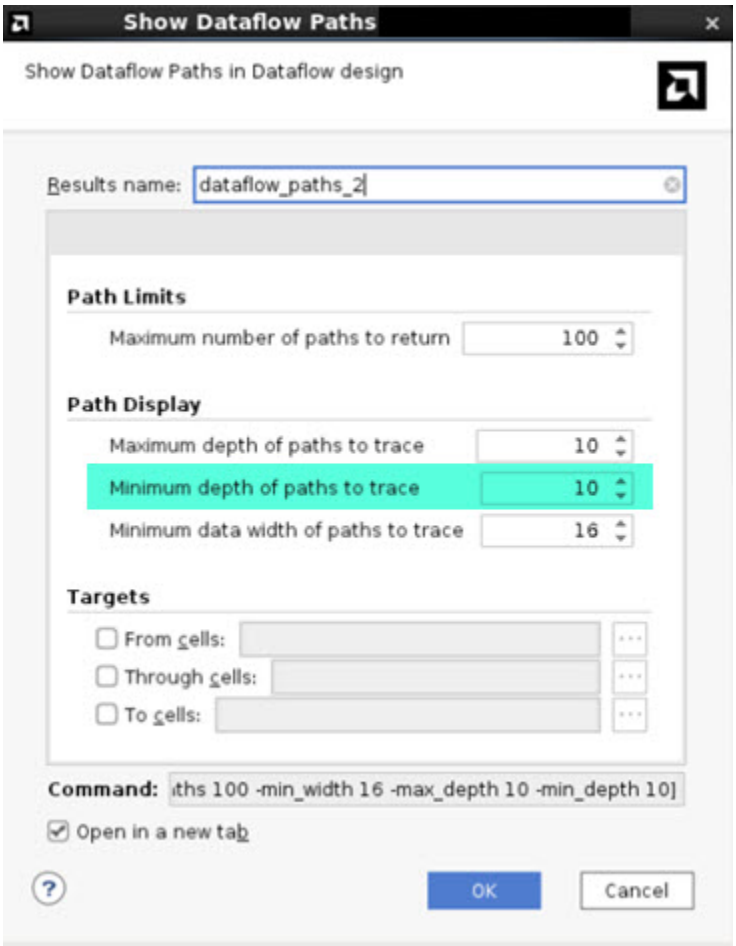
1. In the dataflow design, click **Tools → Show Dataflow Paths**.



2. Click **OK**. This generates the top 100 paths in the design. It starts by identifying paths of which the depths are at the value of the -max_depth value, which defaults to 10. Depth can also be referred to as hops as each cell within the dataflow path is considered a hop. It returns a dataflow paths window that looks like the following:



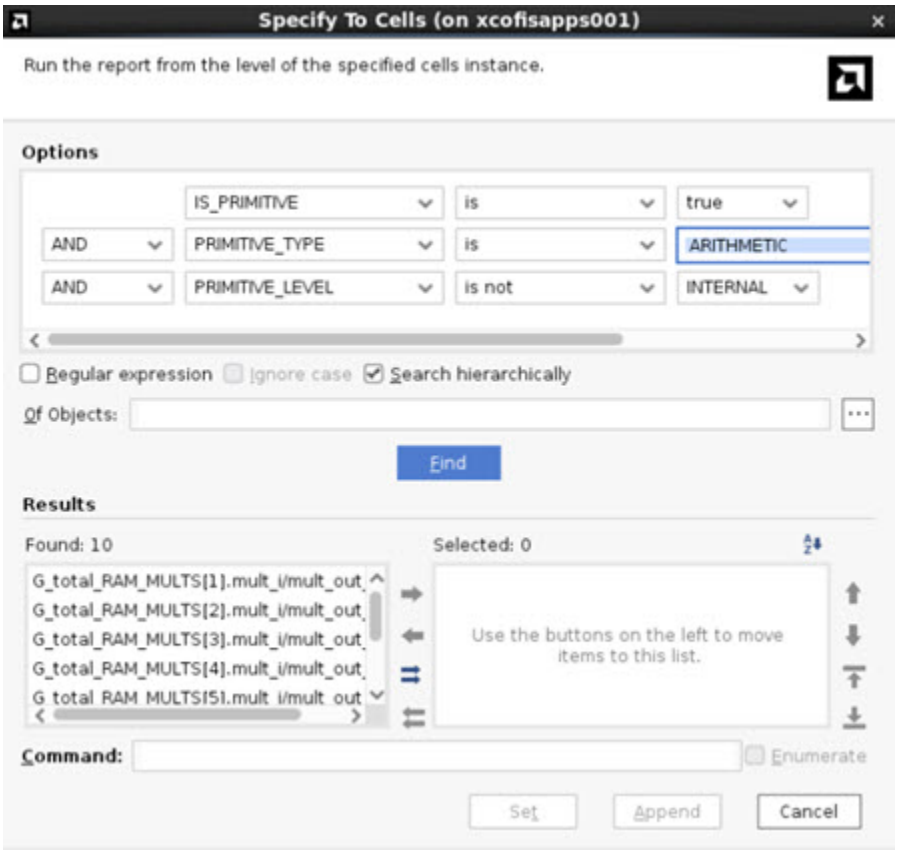
3. Examine the headers in the Find Results window. Try sorting by the following items:
   - Source Ref - This is the type of primitive of the first cell in the dataflow path
   - Hops - This is the number of jumps from one cell to another in the path
   - Source - This is the cell name
4. When you sort by Source, you will see that there are many paths from the same source. These have differing number of hops. In this design, the elements are all in a chain so the paths with fewer hops are a subset of the longest path. Tidy this up by increasing the `-min_depth` switch to 10. Rerun the `Show Dataflow Paths` command with this updated and you will have a result with fewer paths to work with.



5. In a lot of cases, to target paths you are interested in, you will have to create a path where specific cells are targeted. Reopen the Show Dataflow Paths dialog box and select "From cells". Then click on the ... box. Unlike timing paths, Dataflow Paths can only accept leaf cells as the target objects for the `-from`, `-to` and `-through` switches. Nets, pins and hierarchical pins are excluded. Generate a dataflow path with the following options:
   - Start cell: get_cells {G_total_RAM_MULTS[1].ram_i/ram_name_reg_bram_0} (BLOCKRAM)
   - End cell: get_cells {G_total_RAM_MULTS[10].mult_i/mult_out_reg} (DSP)
   - Maximum depth: 20

The following tips will help you find the cells more easily:

- IS_PRIMITIVE=TRUE
- PRIMITIVE_GROUP set to BLOCKRAM for RAM search
- PRIMITIVE_GROUP set to ARITHMETIC for DSP search
- PRIMITIVE_LEVEL is not equal to INTERNAL for DSP search

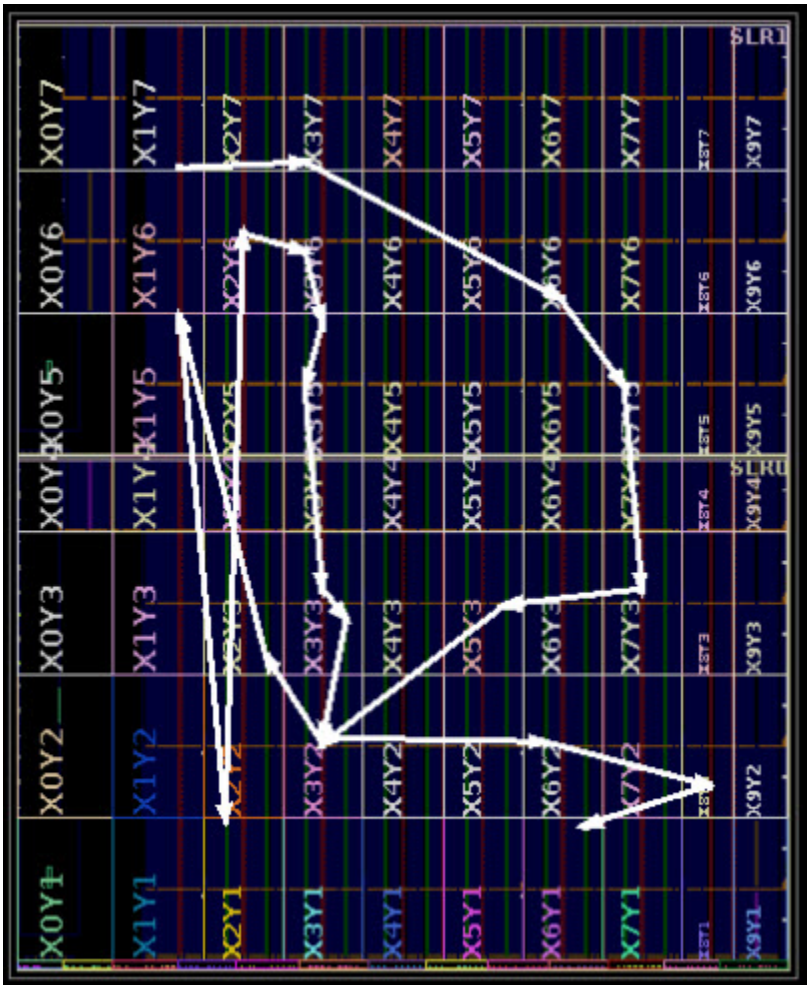


6. From the Tcl Console, you can find the syntax used to generate the dataflow paths.

```
show_objects -name dataflow_paths_3 [get_dataflow_paths -max_paths
100 -min_width 16 -max_depth 20  -from [get_cells
{G_total_RAM_MULTS[1].ram_i/ram_name_reg_bram_0}] -to [get_cells
{G_total_RAM_MULTS[10].mult_i/mult_out_reg}]]
```

The `get_dataflow_paths` command returns objects that you analyze like other objects in Vivado. These can be viewed in the Vivado IDE using the `show_objects` command. You can also use other commands such as `report_property`, `get_property` and `select_objects` as well. Experiment with this, if you need a single object you can use lindex to reference just one object. For example: `report_property [lindex [get_dataflow_paths] 0]`.

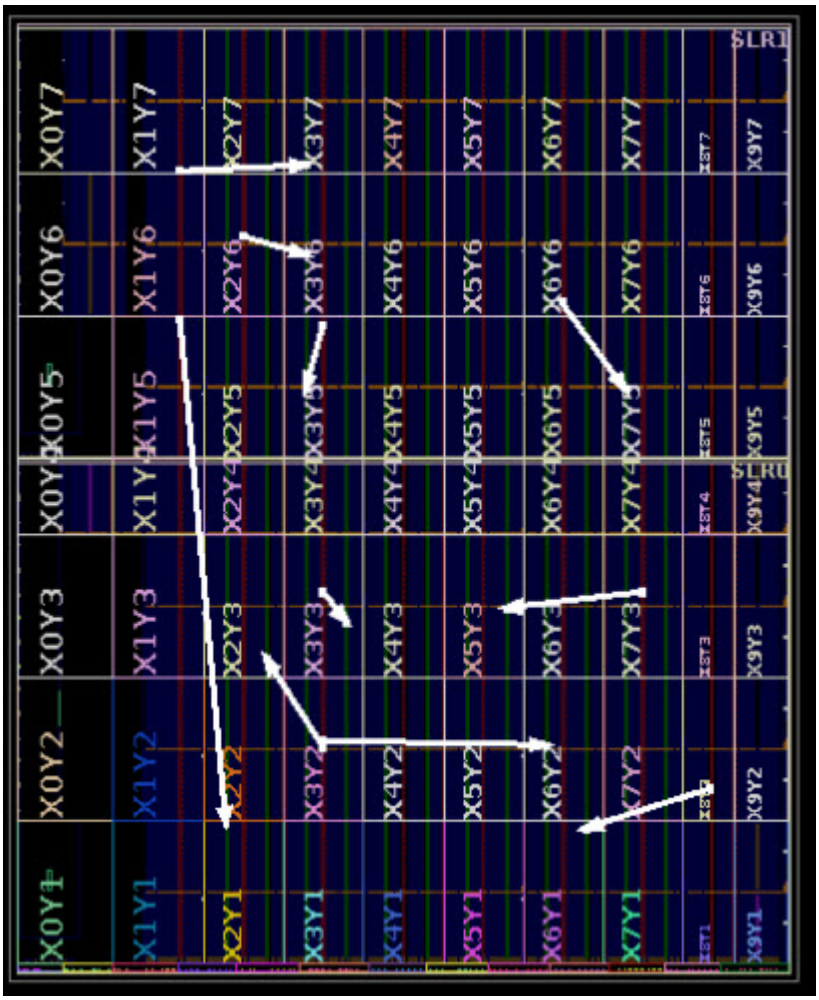
7. Select the returned path. Next, open the Device window if it is not already open. When the parent design is placed, placement of the cells in the dataflow netlist is imported and can be observed in the device view when a dataflow path is selected. In addition, you can also see the direction of the path between the source and the destination as shown in the following figure:



8. There are also some other useful ways to leverage the `get_dataflow_paths` command. For example you can look at paths between different macros such as RAMs – DSPs. We can use the `get_cells` command to return all the block rams and DSP slices and look at paths from RAMs to DSPs along with the max depth set to 1. Lets do this in Tcl:

```
set rams [get_cells -hier -filter {PRIMTIVE_GROUP==BLOCKRAM}];
set dsps [get_cells -hier -filter
{PRIMTIVE_GROUP==ARITHMETIC&&PRIMITIVE_LEVEL!=INTERNAL}];
show_objects [get_dataflow_paths -from $rams -to $dsps -max_depth 1]
```

9. Select all the returned paths. It should produce a device view that looks as the following:



In this design, we have floorplanned the poor placement, but in a more typical design, when you have stretched paths, it can be symptomatic of a type of resource being heavily used and being unavailable locally to the other resource type.

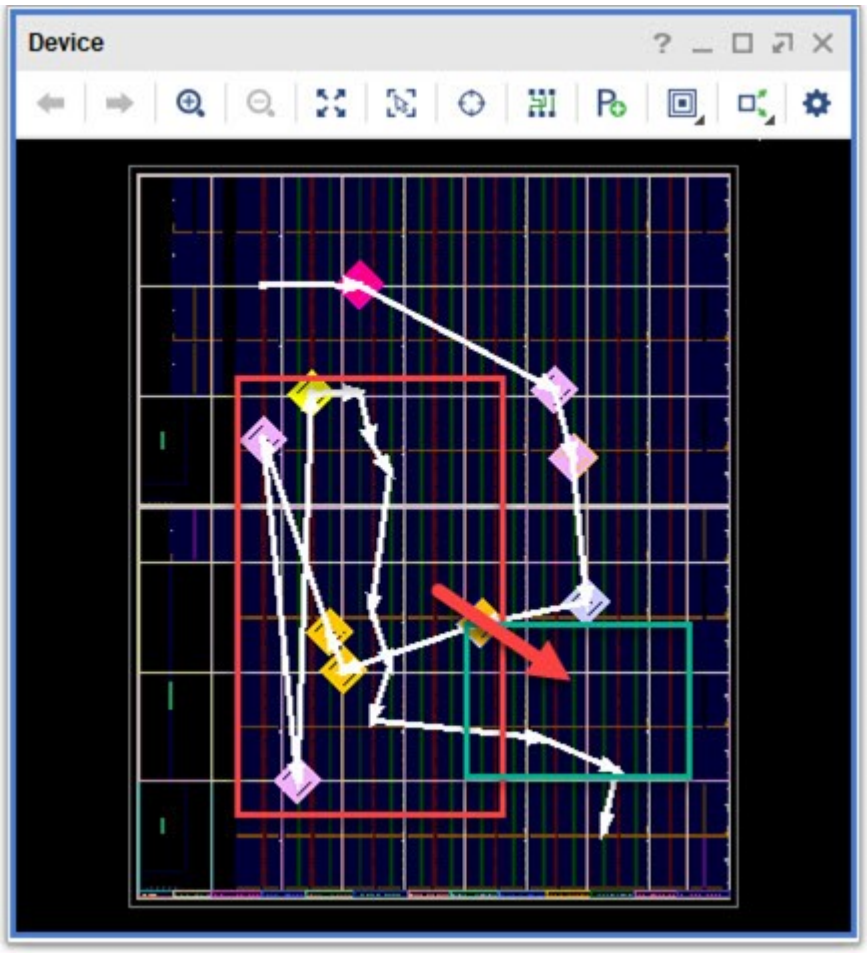
Step 4: Floorplanning

This step explains how floorplanning must be done when using the DFV. As this is a different design, with different objects, and no timing constraints, information must be gathered in DFV and floorplanning must be done in the parent design.

1. Run the following Tcl command:

```
show_objects [set paths [get_dataflow_paths -max_depth 1]]
select_objects $paths
```

In the device view, you again see the individual paths that you saw earlier. When you examine these more closely, you observe that some paths are crossing over each other. For this, you can alleviate the problem by updating the floorplan. The goal is to move all the items in the red box to where the green box is:



2. In this design, there is a chain and the indices of the cells increase. Identify the first cell name in the red box and the last cell name and fill in the blanks. Zoom in within the device view and select the first cell in the chain in the red box. You should find it is the cell with the following name `G_total_RAM_MULTS[3].mult_i/mult_out_reg`. Repeat the process and select the final cell in the chain:`G_total_RAM_MULTS[9].ram_i/ram_name_reg_bram_0`.
3. From the above, you can conclude that you need to floorplan the following cells:

```
DSPs:
G_total_RAM_MULTS[3].mult_i/mult_out_reg
..
G_total_RAM_MULTS[8].mult_i/mult_out_reg
RAMs:
G_total_RAM_MULTS[4].ram_i/ram_name_reg_bram_0
..
G_total_RAM_MULTS[9].ram_i/ram_name_reg_bram_0
```

4. You can create a single path using the following syntax, then select the path and filter it so that only cells are selected

```
set path [get_dataflow_paths -max_depth 11 \
-from [get_cells G_total_RAM_MULTS[3].mult_i/mult_out_reg] \
-to [get_cells G_total_RAM_MULTS[9].ram_i/ram_name_reg_bram_0]]
show_objects -name single $path
select_objects $path
select_objects [set cells [filter [get_selected_objects] {CLASS==cell}] ]
puts "[join $cells \n]"
```

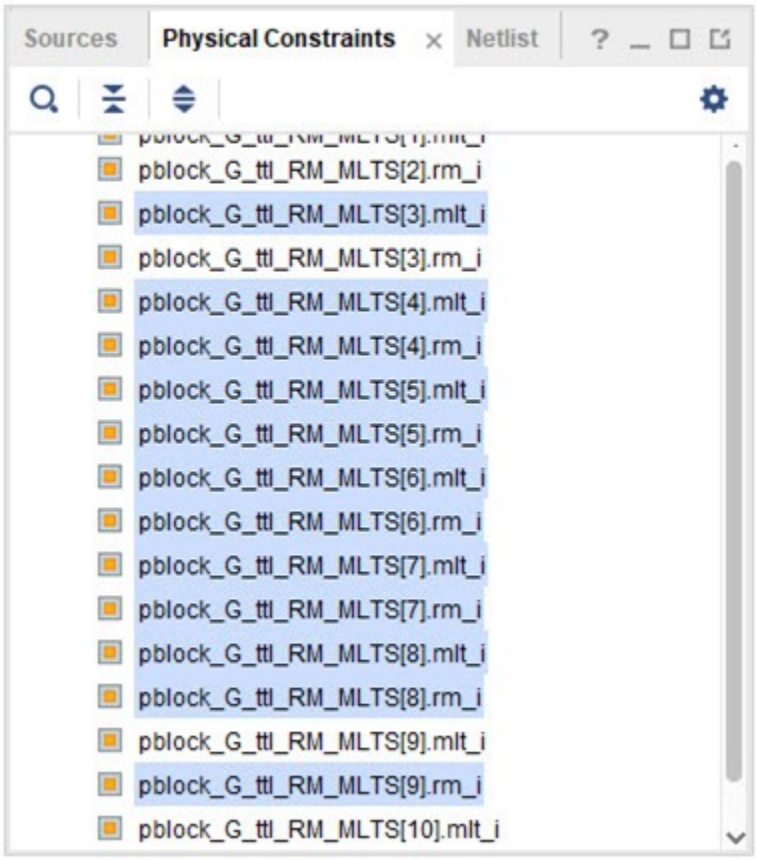
   The resulting output should print out the cell names that are in the path. These are also stored in the cells variable:

```
G_total_RAM_MULTS[3].mult_i/mult_out_reg
G_total_RAM_MULTS[4].ram_i/ram_name_reg_bram_0
G_total_RAM_MULTS[4].mult_i/mult_out_reg
G_total_RAM_MULTS[5].ram_i/ram_name_reg_bram_0
G_total_RAM_MULTS[5].mult_i/mult_out_reg
G_total_RAM_MULTS[6].ram_i/ram_name_reg_bram_0
G_total_RAM_MULTS[6].mult_i/mult_out_reg
G_total_RAM_MULTS[7].ram_i/ram_name_reg_bram_0
G_total_RAM_MULTS[7].mult_i/mult_out_reg
G_total_RAM_MULTS[8].ram_i/ram_name_reg_bram_0
G_total_RAM_MULTS[8].mult_i/mult_out_reg
G_total_RAM_MULTS[9].ram_i/ram_name_reg_bram_0
```

5. Once you have the cell names, switch designs using the switch icon at the top of the screen. At the Tcl Console, prove that the variable still exists after switching designs by typing the following command:

```
puts “$cells”
```

6. Next, you have to delete the existing Pblocks in the design. Open the physical constraints window and delete the Pblocks associated with the above cells. The name is not the same but similar enough to figure out. You should have a list like the following before deleting them:



Once you select these, right click on one and select delete.

7. Next, you need to create a new Pblock with your cells. Use the following syntax to select the cells: `select_objects $cells`. Next, with the objects selected, open the Netlist window and on a selected cell, right click **Draw Pblock**. Ensure that **Assign Selected Cells** is checked and draw a Pblock anywhere in the approximate area. Click through any boxes that pop up.
8. At this point the Pblock is created but not saved. You can either save the constraints or you can go into the Tcl console and copy the text output and manually add this to your design constraints.

Step 5: Correlating Long Congestion with Dataflow Paths

Long congestion is the congestion of only the longer routes in the device which are fewer in number to the shorter routes. It is typically caused by two reasons:

1. Transporting data buses from one point to another in a device. This is caused by placement of cells driving or receiving data buses being far away from each other.

2. Areas of imbalanced cell type resource forcing local modules to be spread. This can force smaller contained modules to spread and end up using long routes.

This section focuses on long congestion caused by data path placement. When you have long congestion, the tool switches to shorter routes and consequently it traverses through more switchboxes. It is a slow process, hence a process to avoid when you have the following scenarios:

- Timing failures going through long congested areas
- Short congestion overlapping with long congestion

On a design with congestion, when you run `report_design_analysis`, it displays congested areas and gives a profile of the cells within the congested area. For short congestion, analysis is typically contained within the congested area. However, for long congestion, relevant paths not only start and end within the congested area but might also begin outside it or end outside it.

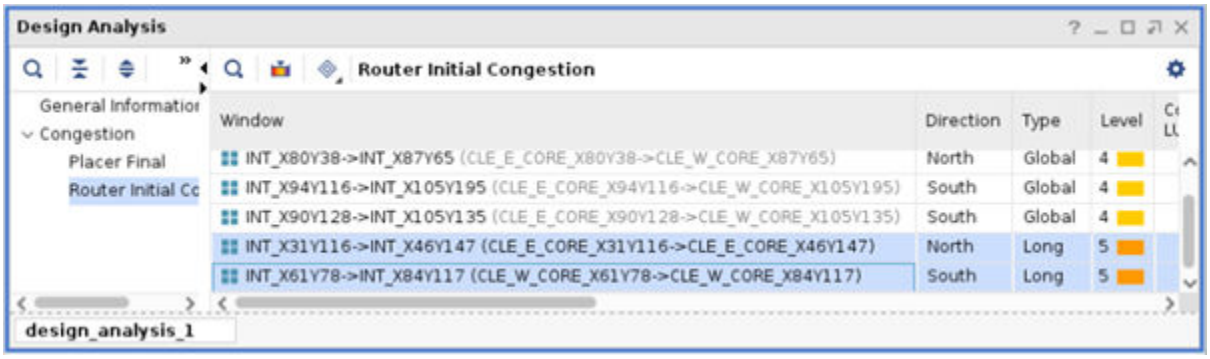
In this step, you must identify paths within an area, filter them based on direction and length and finally expand the paths to get a more holistic view of the path.

Note:

The following is to illustrate how to get the congested tiles in a normal design. The tutorial design does not exhibit congestion.

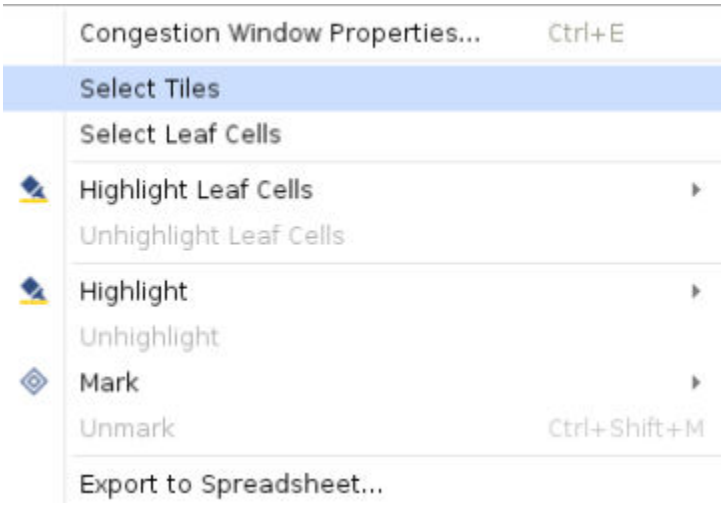
When you run report design analysis, the UI shows Router Initial Congestion analysis, which include specific nets, their routing directions, congestion type, and severity levels.

Figure 6: **Router Initial Congestion**



From this picture you can right click **Select tiles** and this selects all the tiles.

Figure 7: **Select Tiles**



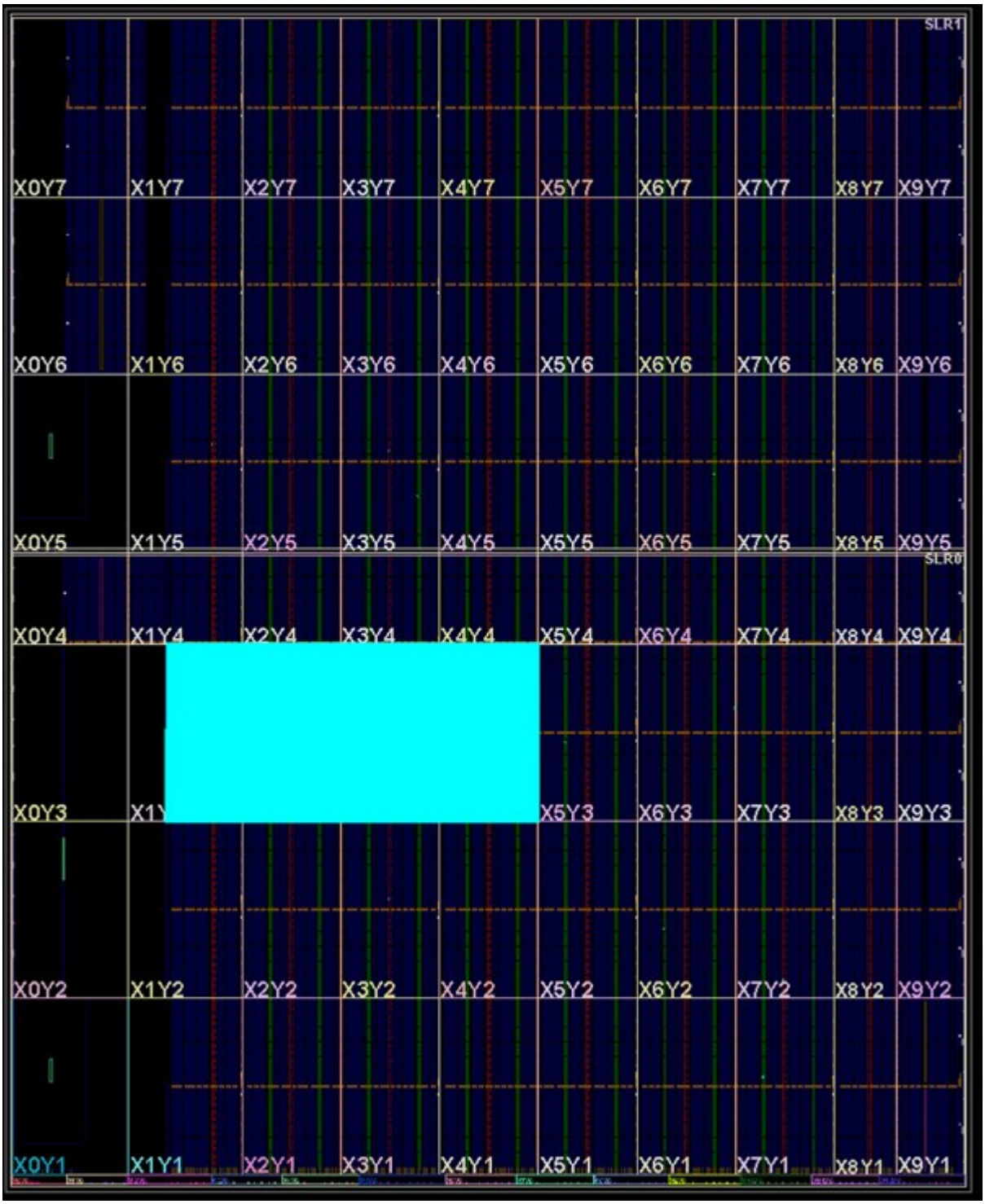
From here you can assign to a variable and switch to the dataflow design.

```
set congested_tiles [get_selected_objects]
```

1. It is at this point where you resume working on the tutorial design. Run the following command to set up a `congested_tiles` variable. This assumes that you have *long* congestion in the *south* direction within these clock regions:

   ```
   set clk_regions [get_clock_regions [list X1Y3 X2Y3 X3Y3 X4Y3]]
   set congested_region [get_tiles -of $clk_regions]
   highlight_objects -color cyan $congested_region
   ```

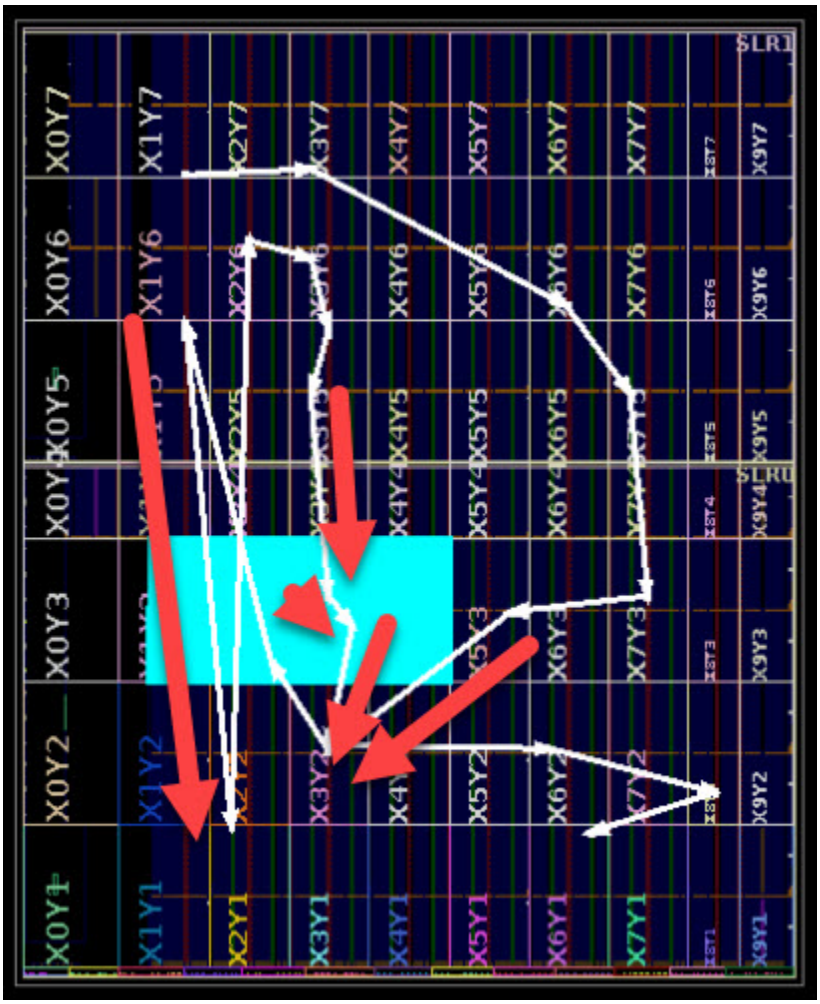
   This command assigns tiles to the `congested_tiles` variable and highlight all the tiles. You should see the following:



2. Next highlight the paths like you did previously using the following command:

```
set start_cell [get_cells G_total_RAM_MULTS[1].ram_i/
ram_name_reg_bram_0];
::dfv::trace_individual_paths $start_cell 20
```

You can identify the following paths. Then within the paths, you can see the marked the paths that you are interested in.



These paths have the following trait

- They can travel through the congested areas.
- They can travel southwards some distance that is at least one long line in length.

3. Next, identify the single hop dataflow paths that pass through the congested area that you highlighted previously. The -through, -from and -to switches of the `get_dataflow_paths` command only accepts cell objects. Hence, you need to identify the start and end cells.

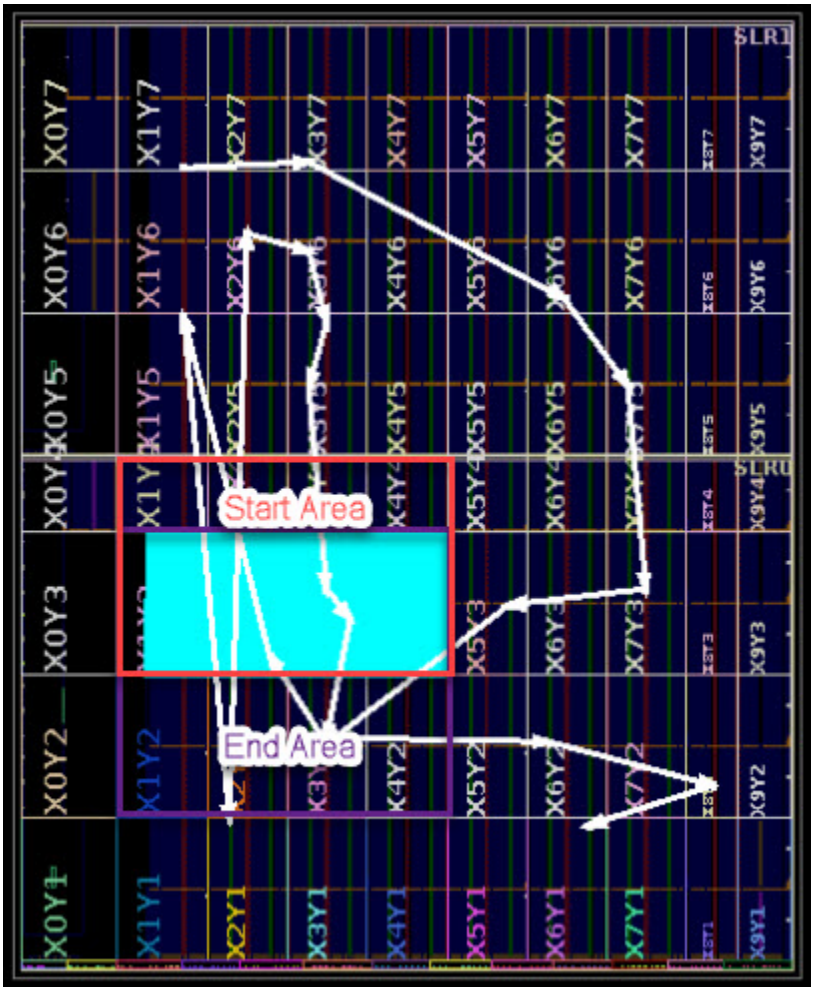
   To identify the cells, firstly, you must identify the areas around the congested region that could have a start cell and an end cell. Once you have the areas, you can use Tcl to extract the cell information and generate some paths.

   Working on a clock region basis is the most convenient approach. Clock regions are defined around the congested area, determining the potential locations of the start cells and end cells of your dataflow path.

Run the following Tcl command:

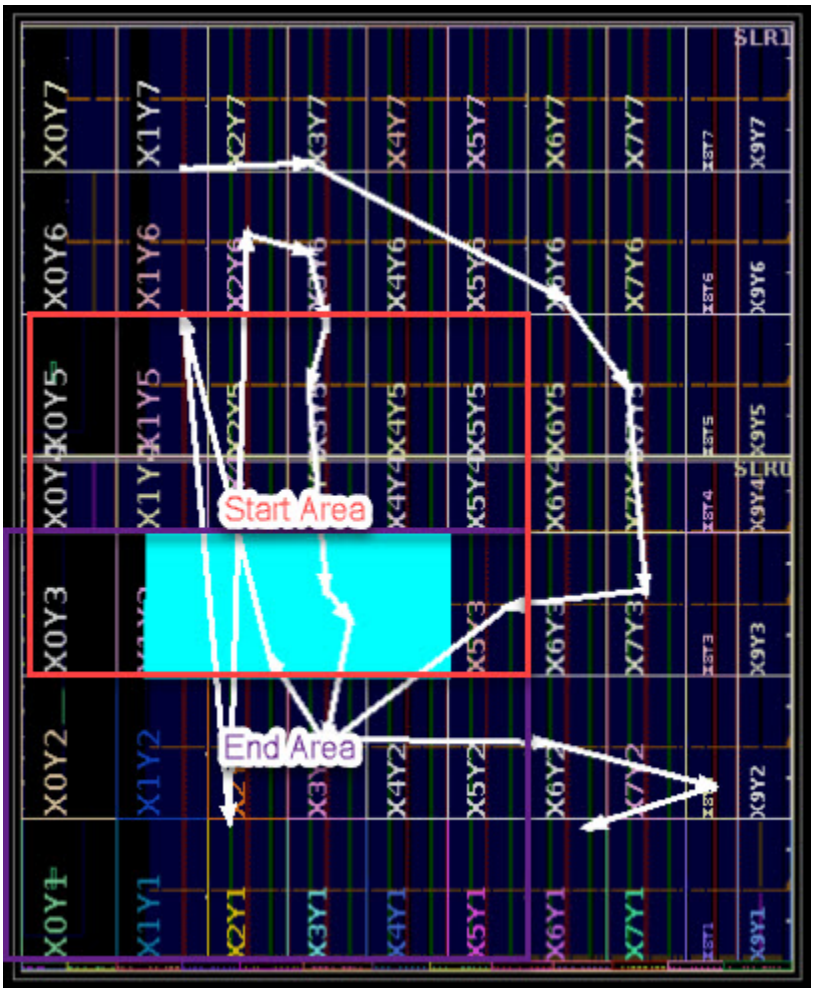
```
set direction south
set src_dest_clock_regions [::dfv::find_congested_clock_regions \
$direction $congested_region 0]
```

This command returns a list of two elements. The first element is a list of clock regions that represents where you look for the startpoint cell locations, and the second is a list of clock regions where you look for endpoint cell locations. In the above command, you have specified that the expansion is 0. This is observed at the following regions:



4. With this approach, some path's startpoints fall outside the Start Area and some endpoints fall outside the End Area. To capture these, you have to increase the expansion window. Run the following command, this expands the start and end areas by 1 clock region around the outside of each area:

```
set src_dest_clock_regions [::dfv::find_congested_clock_regions \
$direction $congested_region 1]
```



This larger area now captures everything you are interested in.

5. Next, run the following command.

```
 ::dfv::get_all_paths_in_region $src_dest_clock_regions $direction 2 \
$congested_region
```

It does the following:

a. Capture all the paths that start and end in the two areas

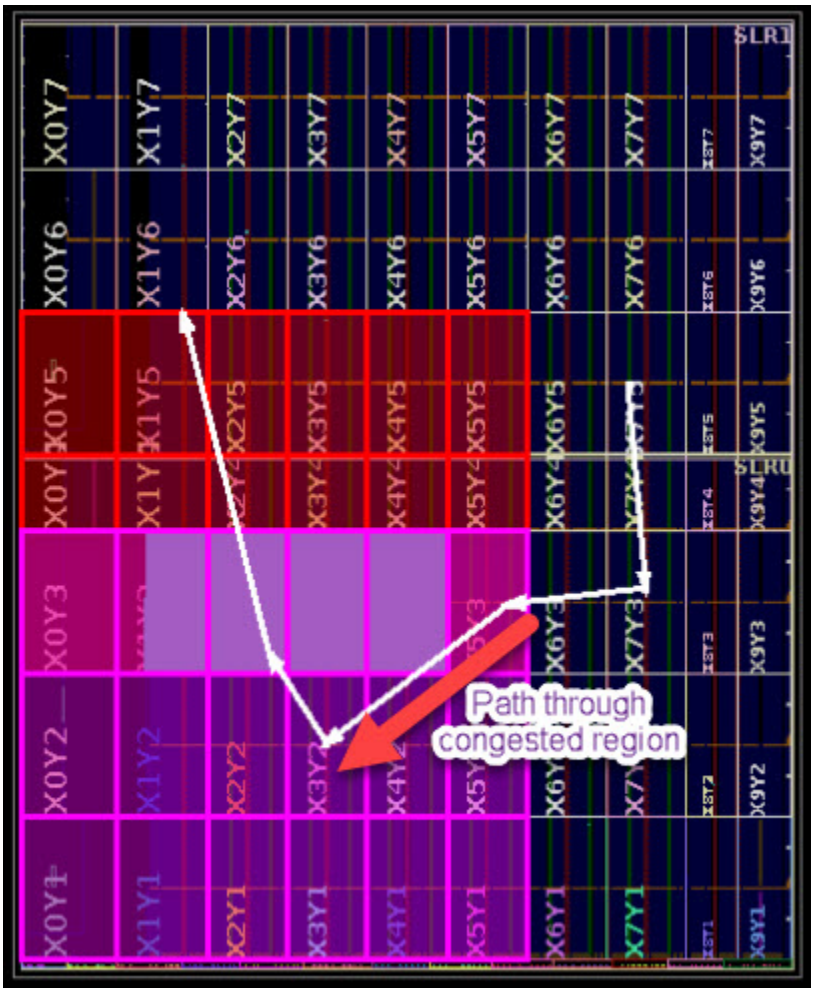
b. Filter out paths that are less than one long line in south direction

c. Filter out paths that are going in the wrong direction

d. Filter out paths that do not pass through the congestion window

e. Extends the start point by two hops and end point by two hops to make 5 hops in total

6. This will return a number of paths. Select the first path, it will look like the following:



The expansion of the paths is two hops at each end. You have:

```
hop1 → hop2 → congestion path → hop3 → hop4
```

This allows you to see the cells that are before and after the congested path. When trying to resolve longer congestion, it is possible to take a longer divergence when you can incorporate more of the paths before and after the congested path.

7. Try repeating this analysis for the other paths and confirm that they pass through the congested region.
8. The next point is to extract the cells that would make up something that could be put in to a floorplan. Select all the five paths that were returned from the original congestion path.

Then type the following Tcl:

```
set cells_to_floorplan [filter [get_sel] {CLASS==cell}]
```

At the end, you have a variable that contains all the cells that you are interested in. It is possible to follow some steps in the previous section to floorplan these cells to a new location.

Summary

In this tutorial, you have:

- Created a dataflow design
- Explored a dataflow design using GUI options, including switching to parent designs and back leveraging cross probing.
- Created dataflow paths to enable targeted analysis
- Floorplanned cells identified in that dataflow design
- Identified long congestion paths

AMD

Appendix A

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 Analysis and Closure Techniques* (UG906)
3. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/05/2025 Version 2025.2	
Chapter 2: Using Report QoR Suggestions and Report QoR Assessment for Timing Closure	Removed the unsupported feature: Copy sources to project.
Step 1: Creating Intelligent Design Runs	Added a note.
06/16/2025 Version 2025.1	
Chapter 1: Setting Waivers with the Vivado IDE	Added a note in the Introduction section.
Chapter 2: Using Report QoR Suggestions and Report QoR Assessment for Timing Closure	Updated the path in Step 6: Accumulating Suggestions
Step 7: Automatically Running QoR Suggestions	Updated the section.
Step 2: Running Report QoR Assessment	Updated the section.

Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software

changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright