Vivado Design Suite Tutorial:

Implementation

UG986 (v2016.4) November 30, 2016

This tutorial was validated with 2016.3. Minor procedural differences might be required when using later releases.

Revision History

11/30/2016: Released with Vivado® Design Suite 2016.4 without changes from 2016.3.

Date	Version	Changes
10/05/2016	2016.3	Validated for Vivado® Design Suite 2016.3. Changes include: • Updated screencaptures • Added information about new option for routing ECO changes on page 93.
06/20/2016	2016.2	Validated for Vivado Design Suite 2016.2.
05/09/2016	2016.1	Validated for Vivado Design Suite 2016.1. Added Lab 4: Vivado ECO Flow.

Table of Contents

> ***IMPORTANT:*** *This tutorial requires the use of the Kintex®-7 and Kintex UltraScale™ family of devices. You will need to update your Vivado® Design Suite tools installation if you do not have this device family installed. Refer to the Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973) for more information on Adding Design Tools or Devices.*

Overview

This tutorial includes four labs that demonstrate different features of the Xilinx® Vivado® Design Suite implementation tool:

- Lab 1 demonstrates using implementation strategies to meet different design objectives.
- Lab 2 demonstrates the use of the incremental compile feature to quickly make small design changes to a placed and routed design.
- Lab 3 demonstrates the use of manual placement and routing, and duplicated routing, to fine-tune the timing on the design.
- Lab 4 demonstrates the use of the Vivado ECO to make quick changes to your design post implementation.

Vivado implementation includes all steps necessary to place and route the netlist onto the FPGA device resources, while meeting the logical, physical, and timing constraints of a design.

> ***VIDEO:*** *You can also learn more about implementing the design by viewing the following Quick Take videos:*
>
> - Vivado Quick Take Video: Implementing the Design
> - Vivado Quick Take Video: Using Incremental Implementation in Vivado

> ***TRAINING:*** *Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use these links to explore related courses:*
>
> - Essentials of FPGA Design
> - Vivado Design Suite Static Timing Analysis and Xilinx Design Constraints
> - Advanced Tools and Techniques of the Vivado Design Suite

Tutorial Design Description

The design used for Lab #1 is the CPU Netlist example design, `project_cpu_netlist_kintex7`, provided with the Vivado Design Suite installation. This design uses a top-level EDIF netlist source file, and an XDC constraints file.

The design used for Lab #2 and Lab #3 is the BFT Core example design, `project_bft_kintex7`. This design includes both Verilog and VHDL RTL files, as well as an XDC constraints file.

The design used for Lab #4 is available as a Reference Design from the Xilinx website. See information in Locating Design Files for Lab 4.

The CPU Netlist and BFT Core designs target an XC7K70T device, and the design for Lab #4 targets an XCKU040 device. Running the tutorial with small designs allows for minimal hardware requirements and enables timely completion of the tutorial, as well as minimizing data size.

Hardware and Software Requirements

This tutorial requires that the 2016.3 Vivado Design Suite software release or later is installed.

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

Preparing the Tutorial Design Files

Locating Design Files for Labs 1-3

You can find the files for Labs 1-3 in this tutorial in the Vivado Design Suite examples directory at the following location:

```
<Vivado_install_area>/Vivado/<version>/examples/Vivado_Tutorial
```

You can also extract the provided zip file, at any time, to write the tutorial files to your local directory, or to restore the files to their starting condition.

Extract the zip file contents from the software installation into any write-accessible location.

```
<Vivado_install_area>/Vivado/<version>/examples/Vivado_Tutorial.zip
```

The extracted `Vivado_Tutorial` directory is referred to as `<Extract_Dir>` in this tutorial.

> ***Note:*** *You will modify the tutorial design data while working through this tutorial. You should use a new copy of the original* `Vivado_Tutorial` *directory each time you start this tutorial.*

Locating Design Files for Lab 4

To access the reference design for Lab #4, do the following:

1. In your C: drive, create a folder called `/Vivado_Tutorial`.
2. Download the Reference Design Files from this link on the Xilinx website.
3. Unzip the tutorial source file to the `/Vivado_Tutorial` folder.

Introduction

In this lab, you will learn how to use implementation strategies with design runs by creating multiple implementation runs employing different strategies, and comparing the results. You will use the CPU Netlist example design that is included in the Vivado® IDE.

Step 1: Opening the Example Project

1. Open the Xilinx® Vivado IDE.

   On Linux:

   a. Change to the directory where the lab materials are stored:

   ```
   cd <Extract_Dir>/Vivado_Tutorial
   ```

   b. Launch the Vivado IDE: **`vivado`**

   On Windows:

   a. Launch the Vivado Design Suite IDE:

   `Start > All Programs > Xilinx Design Tools > Vivado 2016.x > Vivado 2016.x`

   Note: *Your Vivado Design Suite installation might be called something other than* ***Xilinx Design Tools*** *on the* ***Start*** *menu.*

   Note: *As an alternative, click the* ***Vivado 2016.x*** *Desktop icon to start the Vivado IDE.*

   The Vivado IDE Getting Started page contains links to open or create projects and to view documentation.

   b. From the Getting Started page, click **Open Example Project**.

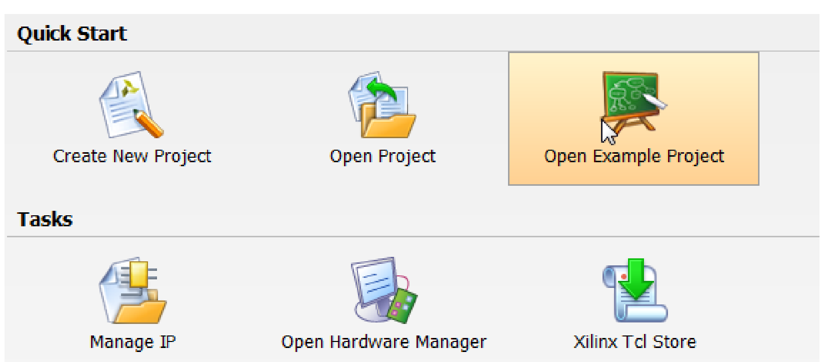


Figure 1: Open Example Design

2. Click **Next** in the Open Example Project screen.

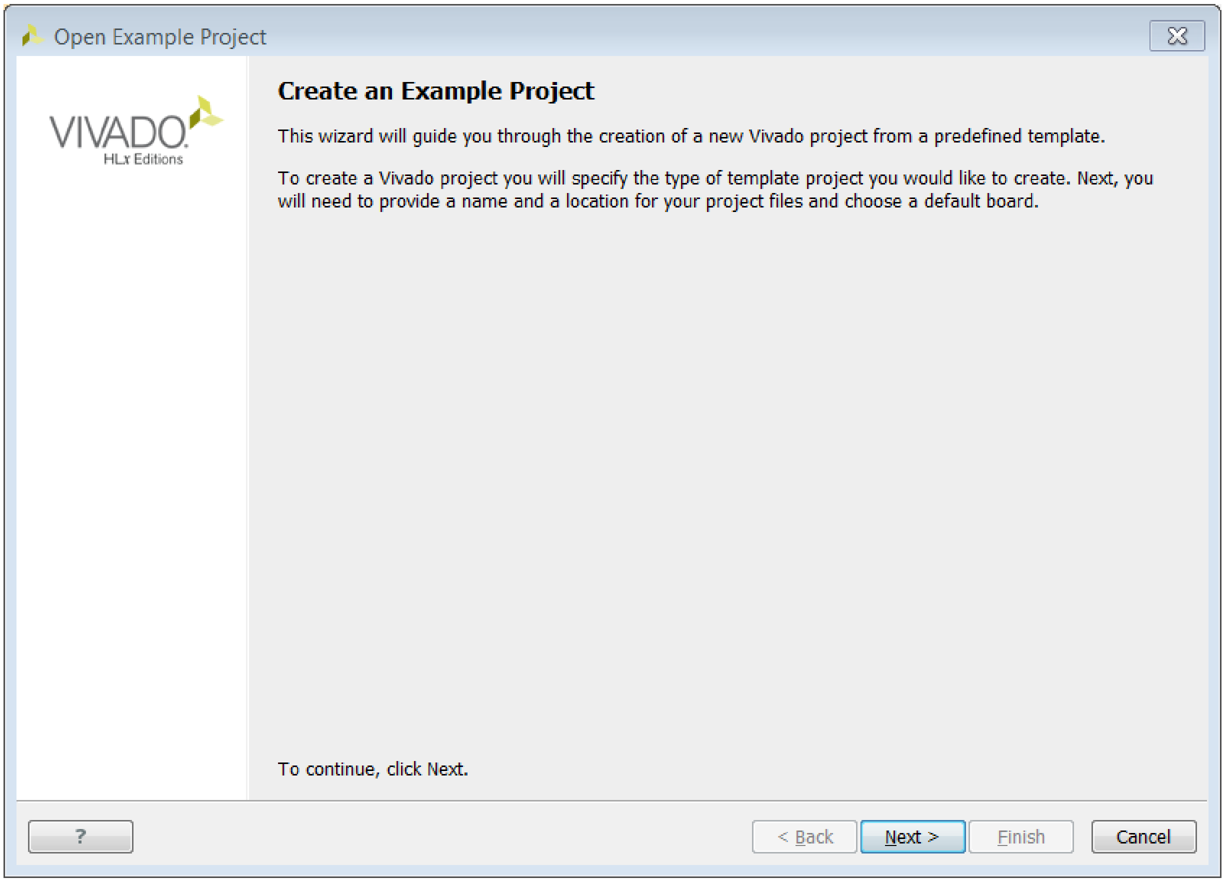


Figure 2: Open Example Project, Create an Example Project Screen

3. In the Select Example Project screen, choose the **CPU (Synthesized)** project and click **Next**.

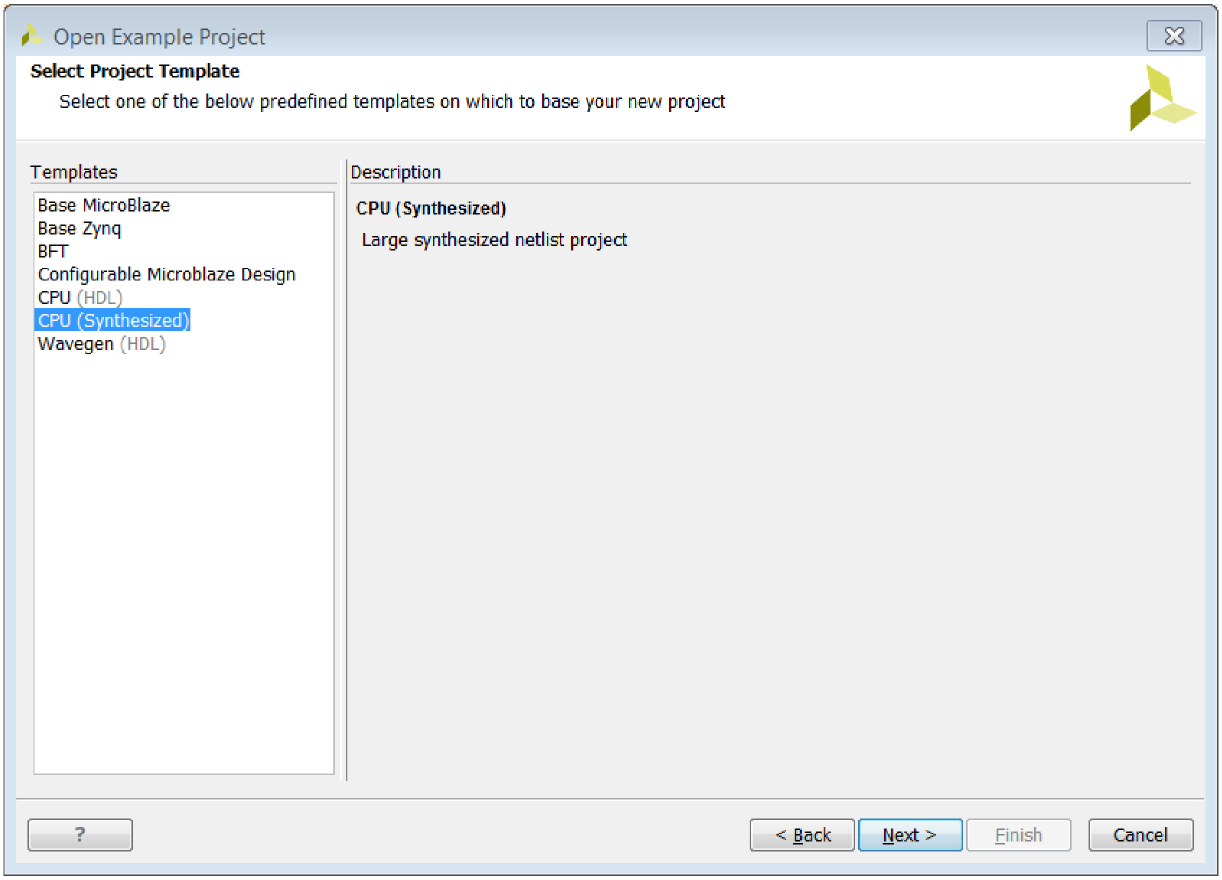


Figure 3: Open Example Project, Select Example Project Screen

4. In the Project Name screen, specify the following:
    - Project name: **project_cpu_netlist**
    - Project location: <Project_Dir>

5. Click **Next**.

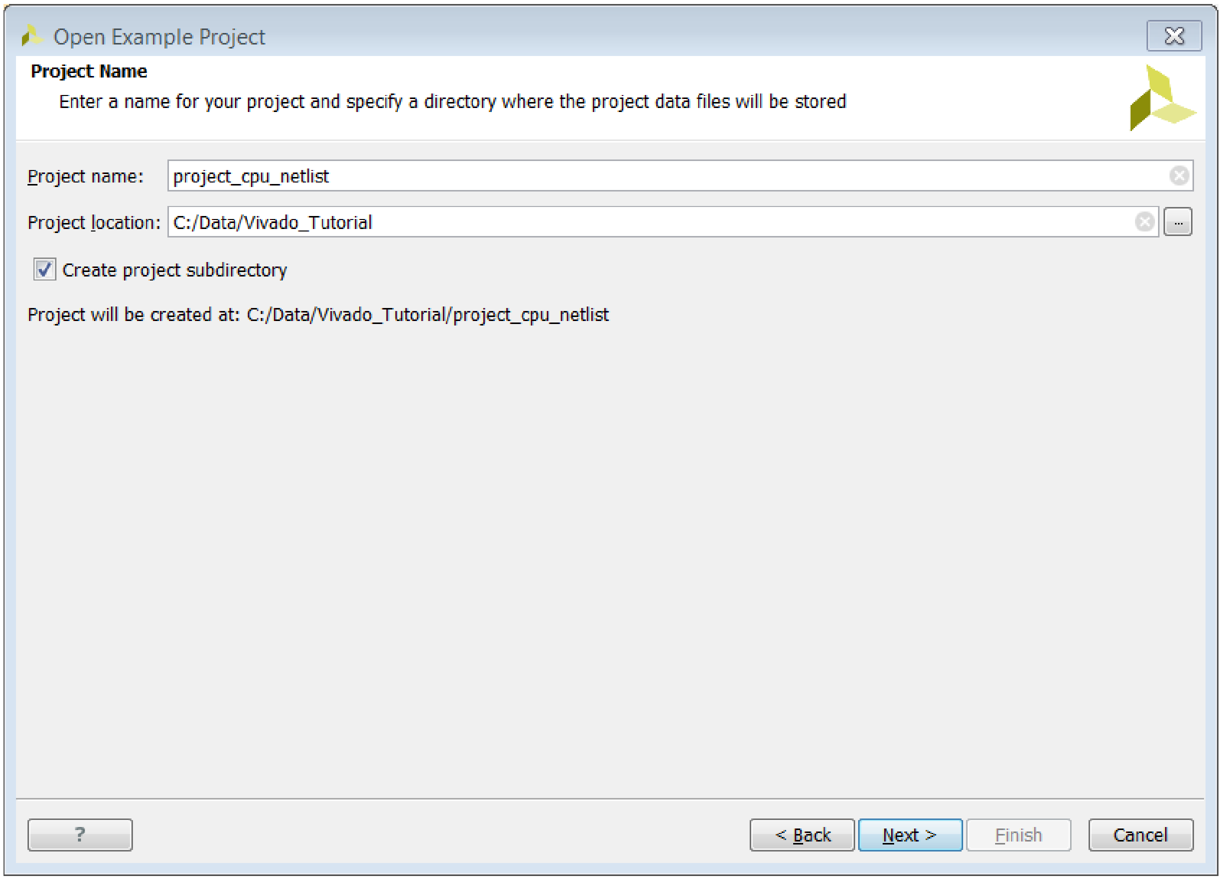


Figure 4: Project Name Screen

6. In the Default Part screen, select the **xc7k70tfbg676-2** part and click **Next**.

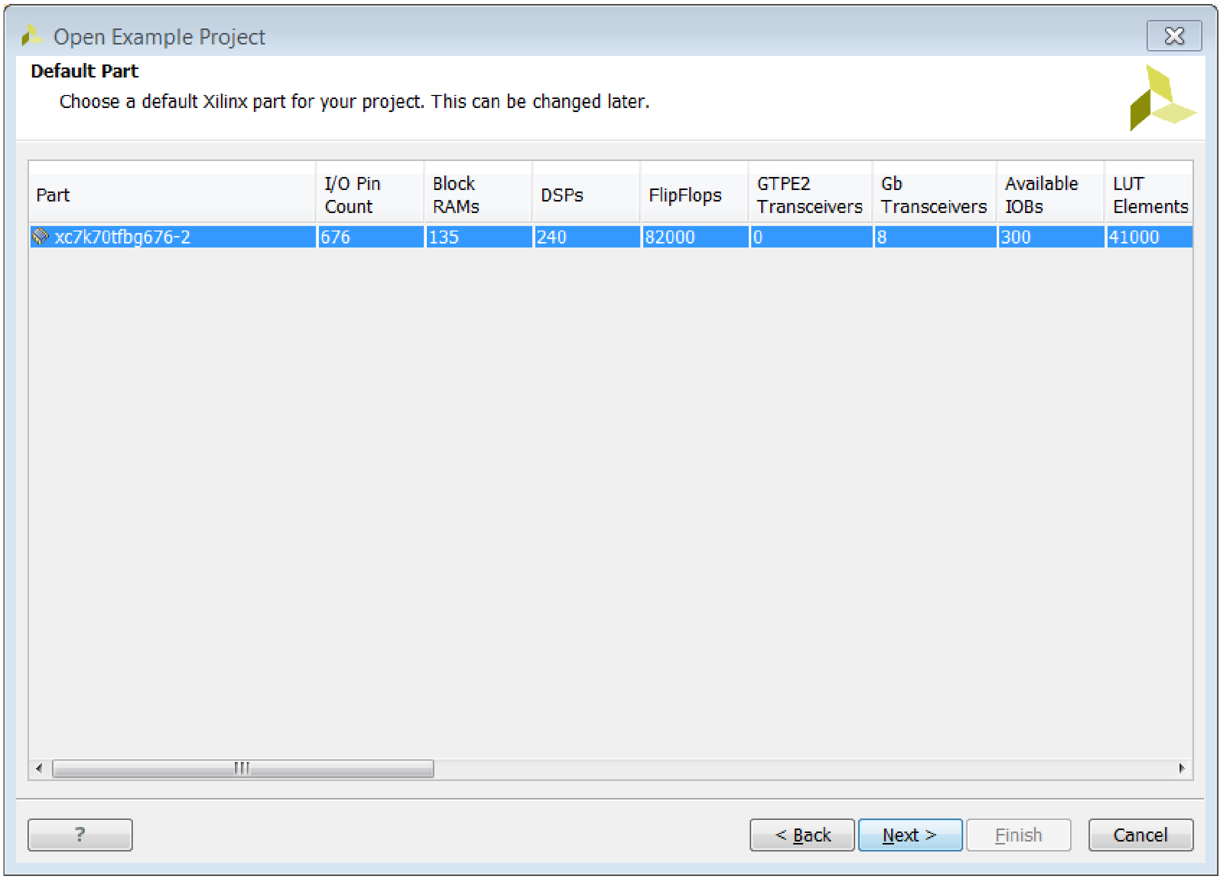


Figure 5: Default Part Screen

7. A summary page appears, allowing you to review project details. Click **Finish** to return to the Vivado IDE.

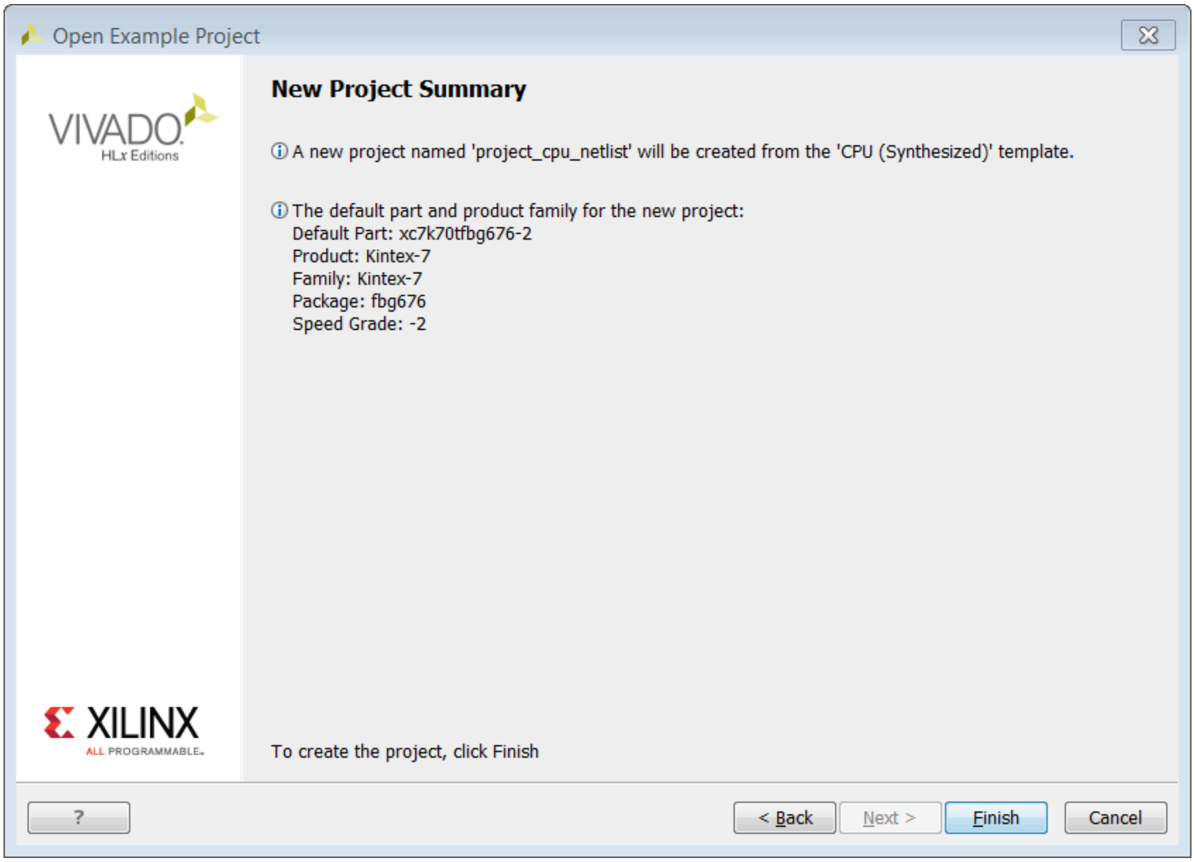


Figure 6: New Project Summary Screen

The Vivado IDE displays the default view of the opened project, as shown in the following figure:

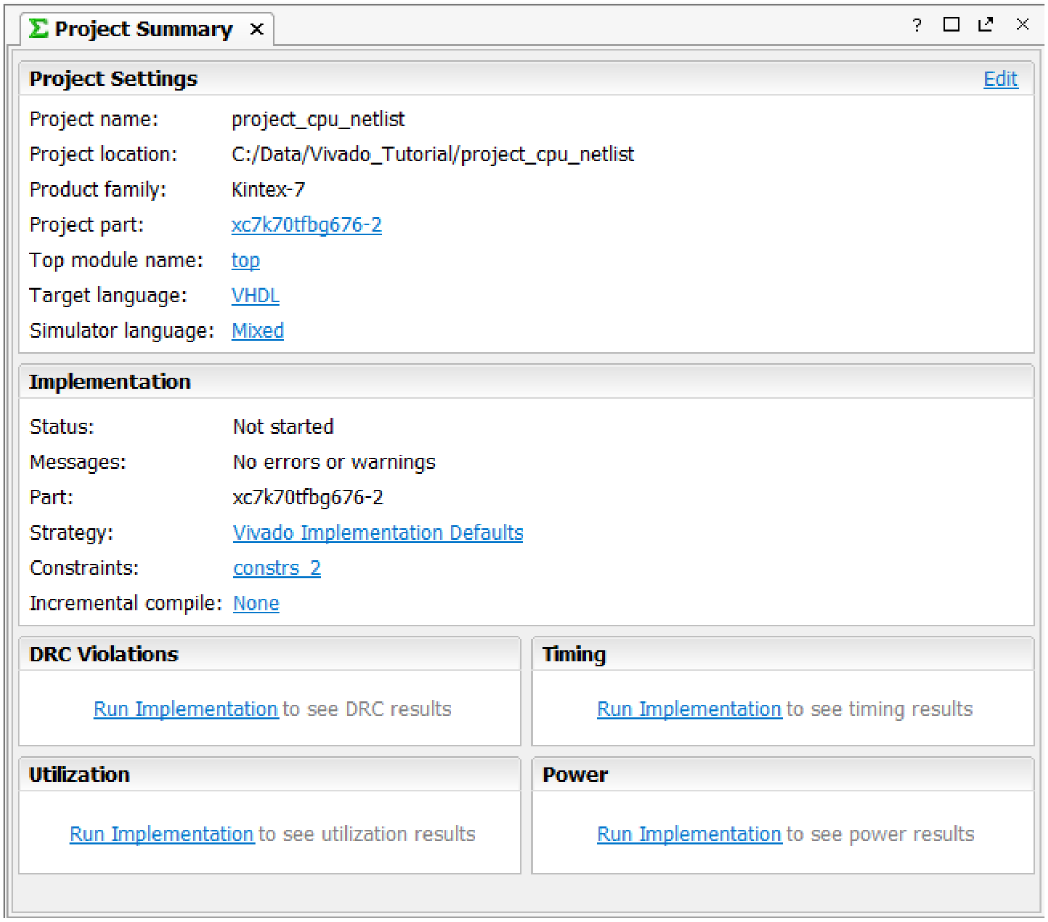


Figure 7: Project Summary Window

Step 2: Creating Additional Implementation Runs

The project contains previously defined implementation runs as seen in the Design Runs window of the Vivado IDE. You will create new implementation runs and change the implementation strategies used by these new runs.

1. From the main menu, select **Flow > Create Runs**.
2. The Create New Runs wizard opens.
3. Click **Next** to open the Configure Implementation Runs screen.

   The screen appears with a new implementation run defined. You can configure this run and add other runs as well.
4. In the **Strategy** drop-down menu, select **Performance_Explore** as the strategy for the run.
5. Click the **Add** button twice to create two additional runs. 

6. Select **Flow_RunPhysOpt** as the Strategy for the **impl_3** run.
7. Select **Flow_RuntimeOptimized** as the Strategy for the **impl_4** run.

   The Configure Implementation Runs screen now displays three new implementations along with the strategy you selected for each, as shown in the below figure.

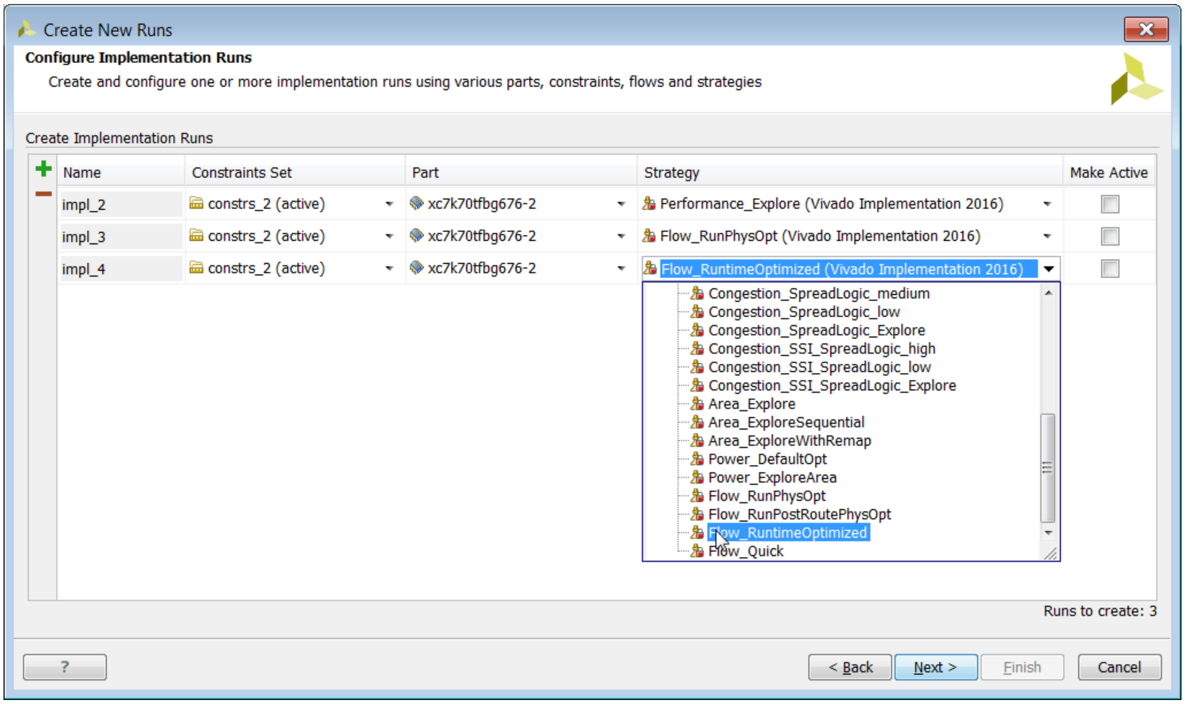


Figure 8: Configure Implementation Runs

8. Click **Next** to open the Launch Options screen.
9. Select **Do not launch now**, and click **Next** to view the Create New Runs Summary.
10. In the Create New Runs Summary page, click **Finish.**

Step 3: Analyzing Implementation Results

1. In the Design Runs window, select all the implementation runs.

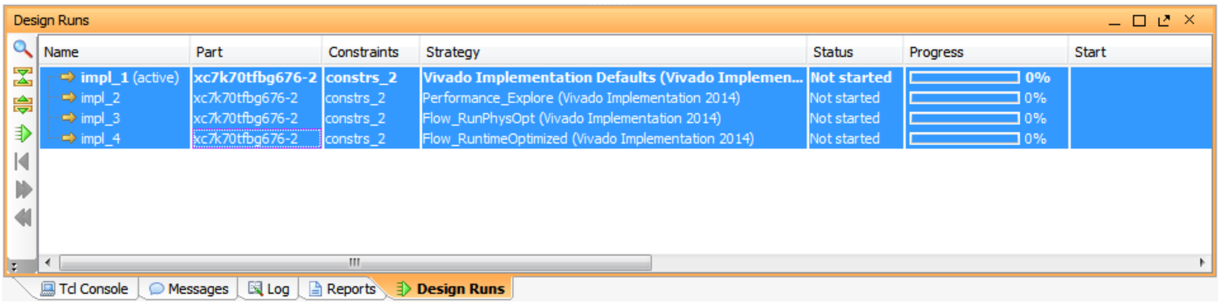


Figure 9: Design Runs Window

2. On the sidebar menu, click the **Launch Selected Runs** button .
3. In the Launch Selected Runs dialog box, select **Launch runs on local host** and **Number of jobs: 2**, as shown below.

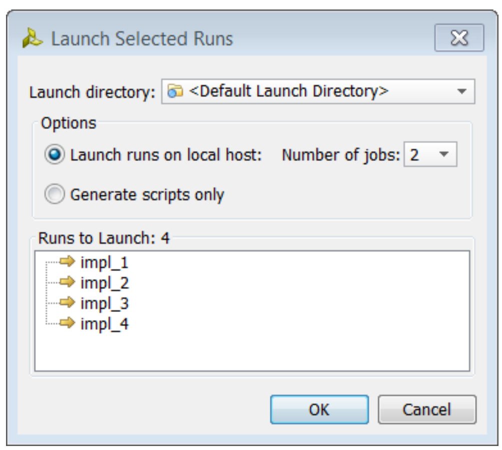


Figure 10: Launch Selected Runs

4. Click **OK**.

   Two runs launch simultaneously, with the remaining runs going into a queue.

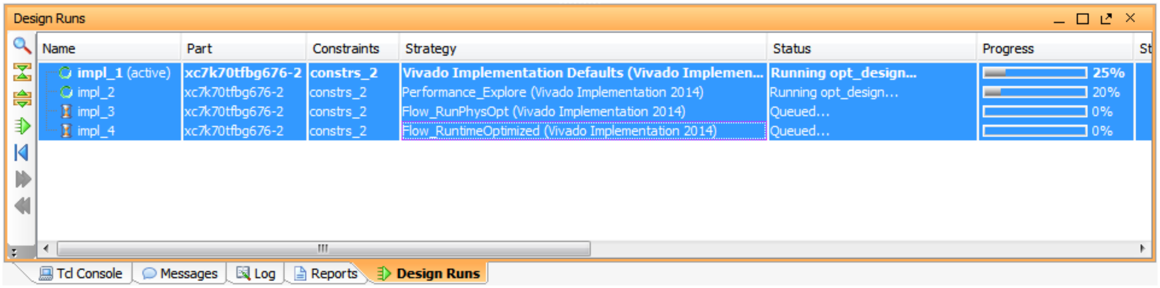


Figure 11: Two Implementation Runs in Progress

When the active run, `impl_1`, completes, examine the Project Summary. The Project Summary reflects the status and the results of the active run. When the active run (`impl_1`) is complete, the Implementation Completed dialog box opens.

5. Click **Cancel** to close the dialog box.

   Note: *The implementation utilization results display as a bar graph at the bottom of the summary page (you might need to scroll down), as shown in the following figure.*

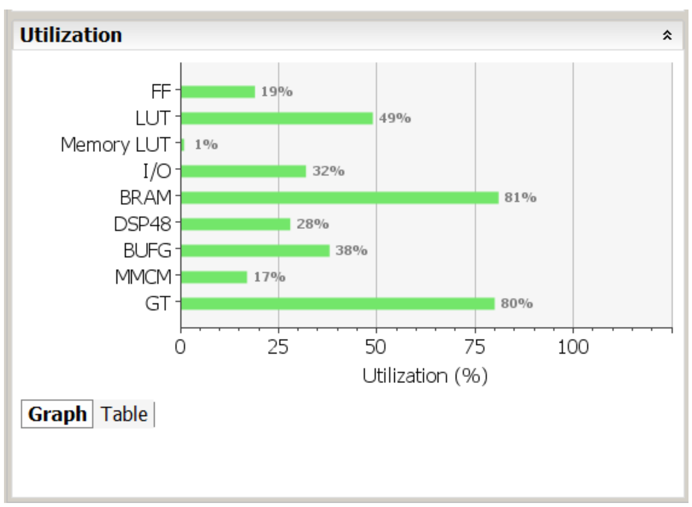


Figure 12: Post-Implementation Utilization

When you open an implementation run, the report_power and the report_timing_summary results are automatically opened for the run in a new tab in the Results Window.

6. When all the implementation runs are complete, select the **Design Runs** tab.
7. Right-click the **impl_3** run in the Design Runs window, and select **Make Active** from the popup menu.

   The Project Summary now displays the status and results of the new active run, **impl_3**.

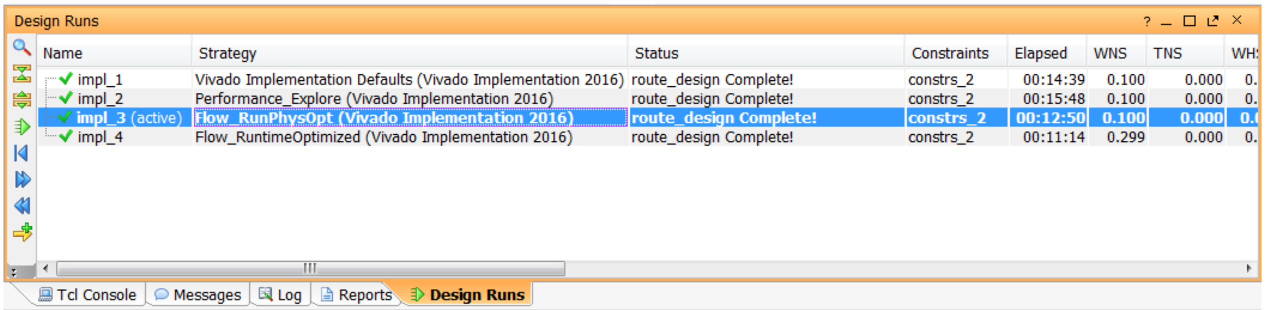


Figure 13: Compare Implementation Results

8. Compare the results for the completed runs in the Design Runs window, as shown in Figure 13.
   - The **Flow_RuntimeOptimized** strategy in **impl_1** completed in the least amount of time, as you can see in the **Elapsed** time column.
   - The WNS column shows that all runs met the timing requirements.

Step 4: Tightening Timing Requirements

To examine the impact of the Performance_Explore strategy on meeting timing, you will change the timing constraints to make timing closure more challenging.

1. In the Sources window, double-click the `top_full.xdc` file in the `constrs_2` constraint set.

   The constraints file opens in the Vivado IDE text editor.

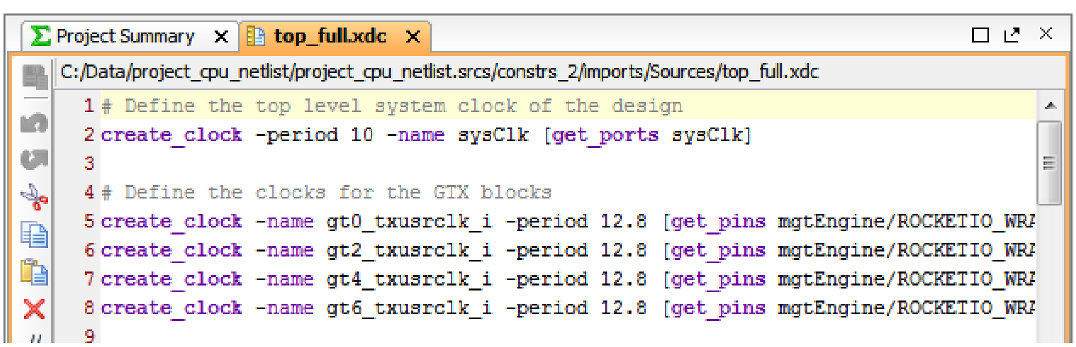


Figure 14: top_full.xdc File

2. On line 2, change the period of the `create_clock` constraint from `10` ns to `7.45` ns.

   The new constraint should read as follows:

   ```
   create_clock -period 7.45 -name sysClk [get_ports sysClk]
   ```

3. Save the changes by clicking the **Save File** button in the sidebar menu of the text editor.

   Note: *Saving the constraints file changes the status of all runs using that constraints file from "Complete" to "Out-of-date," as seen in the Design Runs window.*

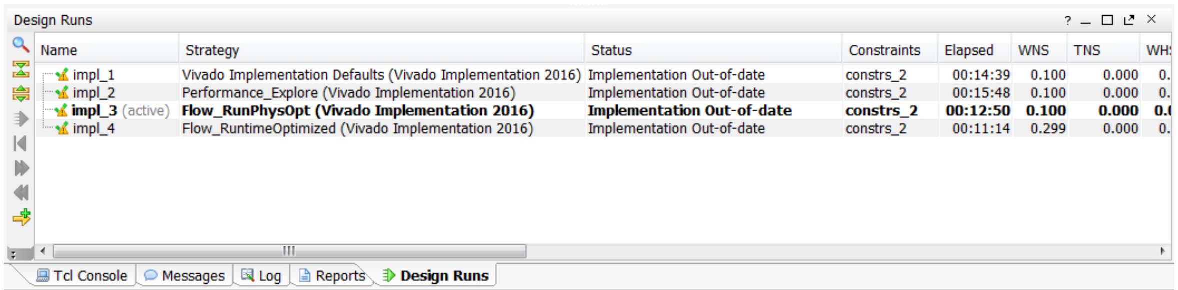


Figure 15: Implementation Runs Out-of-Date

4. In the Design Runs window, select all runs and click the **Reset Runs** button.

5. In the Reset Runs dialog box, enable the **Delete the generated files in the working directory** checkbox and click **Reset**.

   This checkbox directs the Vivado Design Suite to remove all files associated with the selected runs from the project directory. The status of all runs changes from "Out-of-date" to "Not started."

6. With all runs selected in the Design Runs window, click the **Launch Selected Runs** button.

   The Launch Selected Runs window opens.

> ***TIP****: You can also launch runs without resetting them first. If the runs are out of date, the Reset Runs dialog box displays. In this dialog box, you can reset the runs before they are launched.*

7. Select Launch runs on local host and Number of jobs: 2 and click **OK**.

   When the active run (`impl_3`) completes, the Implementation Completed dialog box opens.

8. Click **Cancel** to close the dialog box.

9. Compare the **Elapsed** time for each run in the Design Runs window, as seen in the following figure.

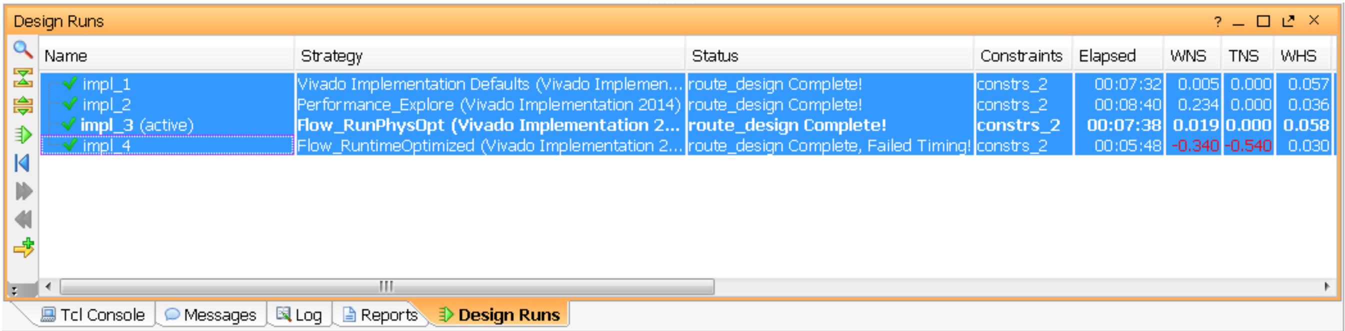


Figure 16: Implementation Results

- Notice that the impl_2 run, using the Performance_Explore strategy has passed timing with positive slack (WNS), but also took the most time to complete.
- Only one other run passed timing, but with less positive slack.

RECOMMENDED: *Reserve the Performance_Explore strategy for designs that have challenging timing constraints. When timing is easily met, the Performance_Explore strategy increases implementation times while providing no timing benefit.*

Conclusion

In this lab, you learned how to define multiple implementation runs to employ different strategies to resolve timing. You have seen how some strategies trade performance for results, and learned how to use those strategies in a more challenging design.

This concludes Lab #1. If you plan to continue directly to Lab #2, keep the Vivado IDE open and close the current project. If you do not plan to continue, you can exit the Vivado Design Suite.

Introduction

Incremental Compile is an advanced design flow to use on designs that are nearing completion, where small changes are required.

After resynthesizing a design with minor changes, the incremental compile flow can speed up placement and routing by reusing results from a prior design iteration. This can help you preserve timing closure while allowing you to quickly implement incremental changes to the design.

In this lab, you use the BFT example design that is included in the Vivado® Design Suite, to learn how to use the incremental compile flow. Refer to the *Vivado Design Suite User Guide: Implementation* (UG904) to learn more about Incremental Compile.

Step 1: Opening the Example Project

1. Start by loading Vivado IDE by doing one of the following:
   - Launch the Vivado IDE from the icon on the Windows desktop.
   - Type **`vivado`** from a command terminal.
2. From the Getting Started page, click **Open Example Project**.

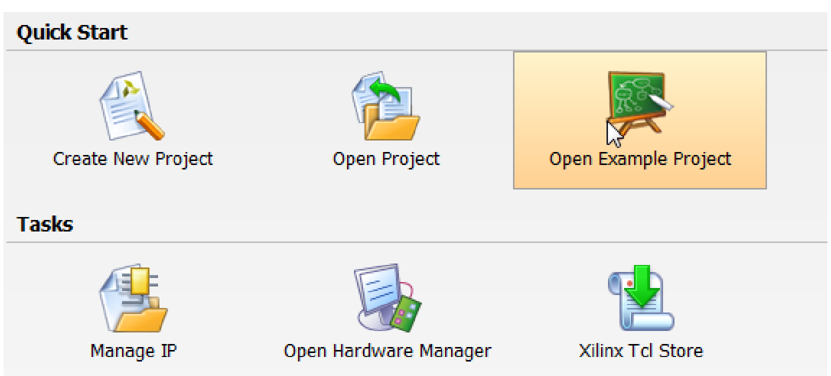


Figure 17: Open Example Design

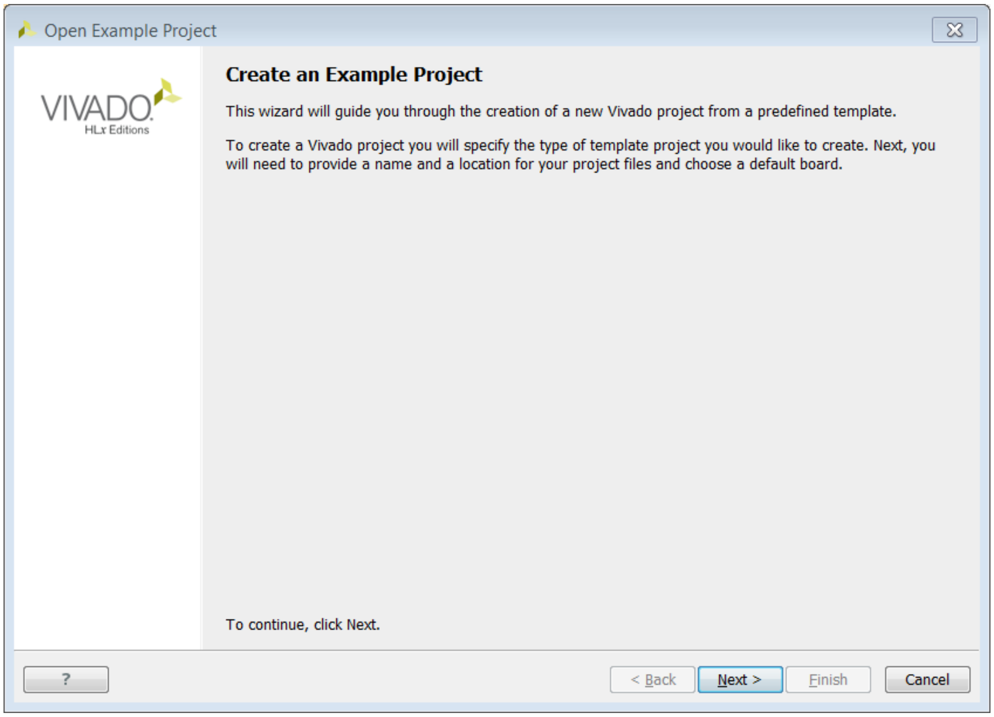


Figure 18: Open Example Project

3. Click **Next** in the Open Example Project screen.
4. Select the BFT (Small RTL project) design.

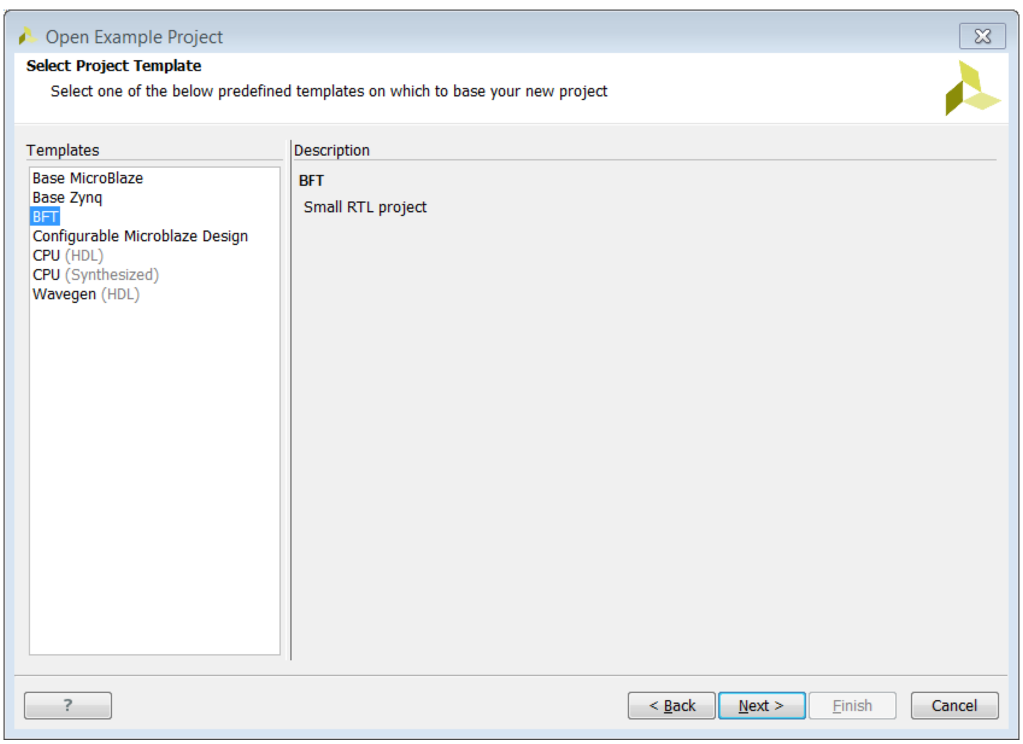


Figure 19: Select Example Project Screen

5. Click **Next**.
6. In the Project Name screen, specify the following :
   - Project name: **`project_bft_core_hdl`**
   - Project location: **`<Project_Dir>`**
7. Click **Next**.

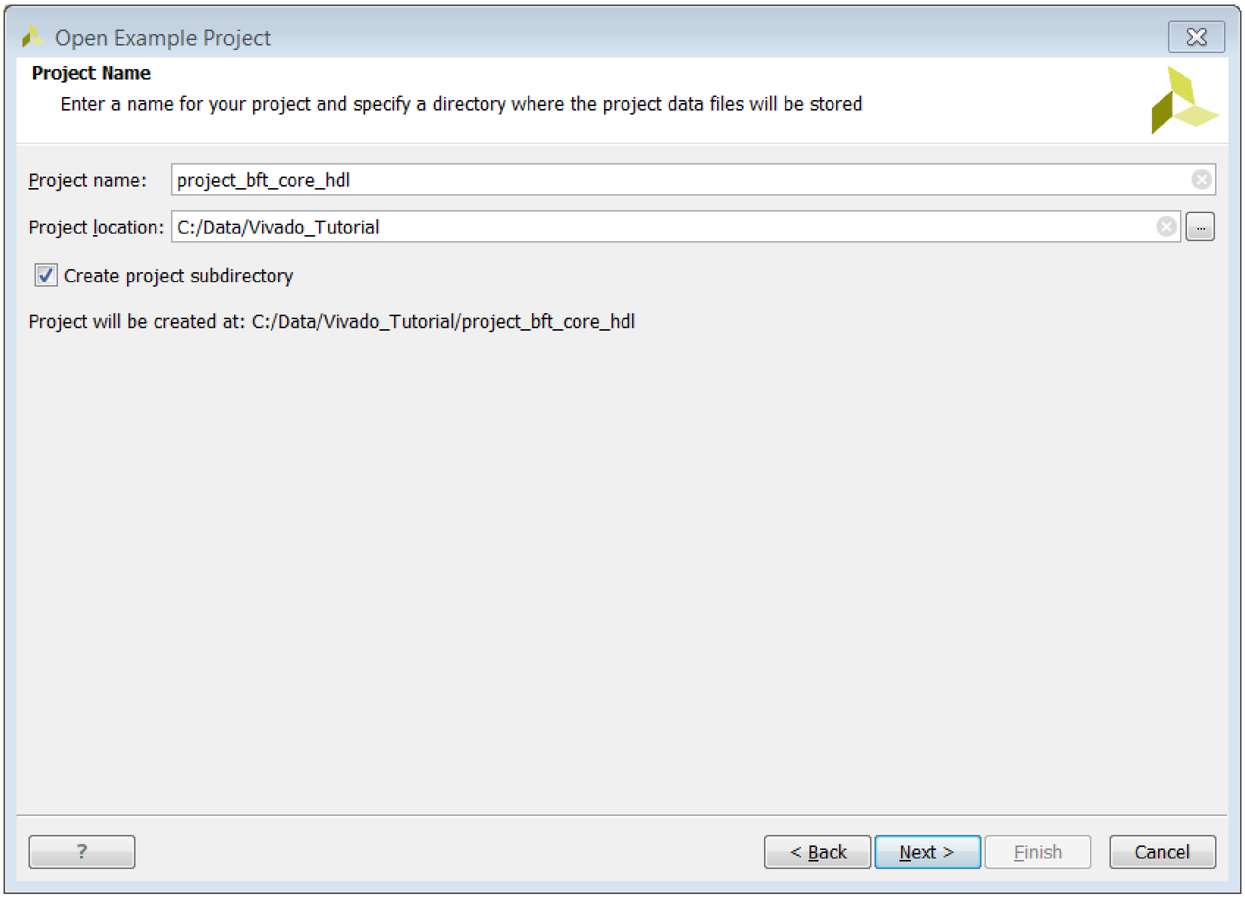


Figure 20: Project Name Screen

8. In the Default Part screen, select the **xc7k70tfbg484-2** part and click **Next**.

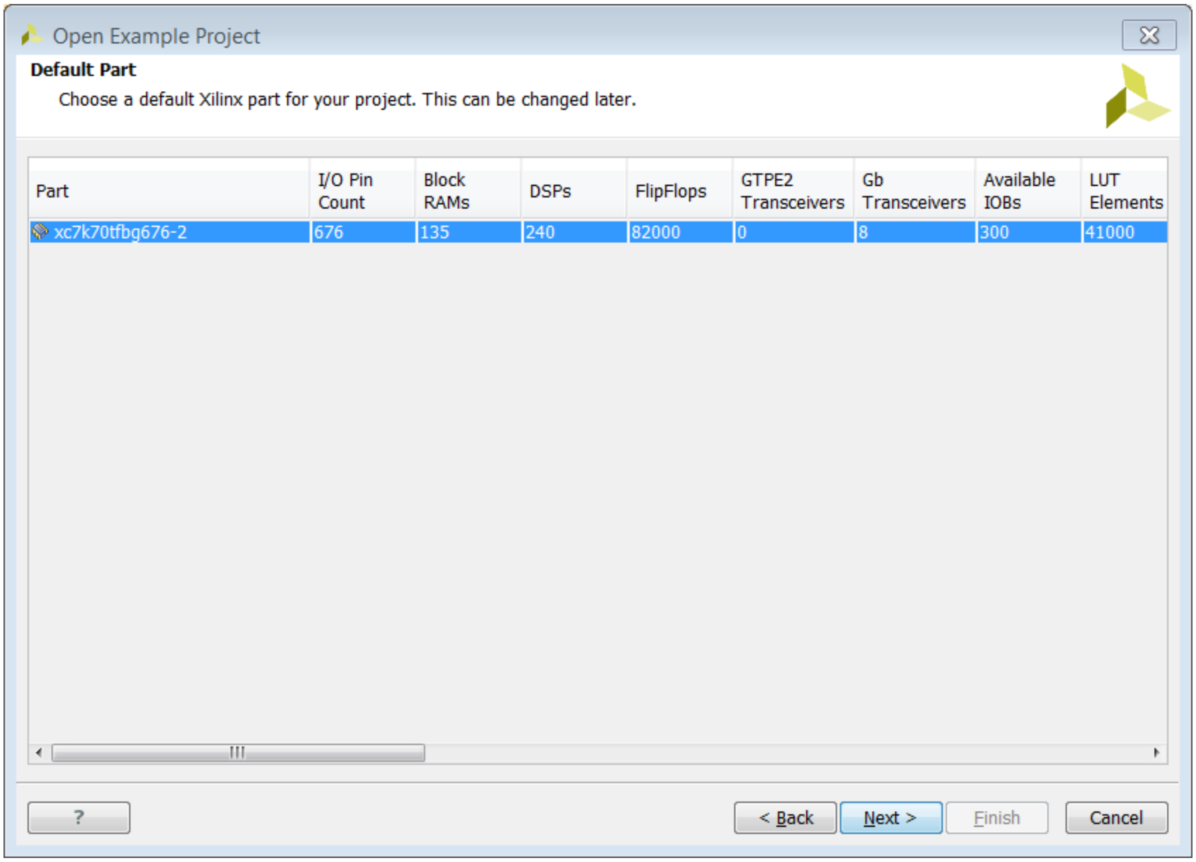


Figure 21: Default Part Screen

9. The New Project Summary screen appears, displaying project details. When you have reviewed these, click **Finish**.

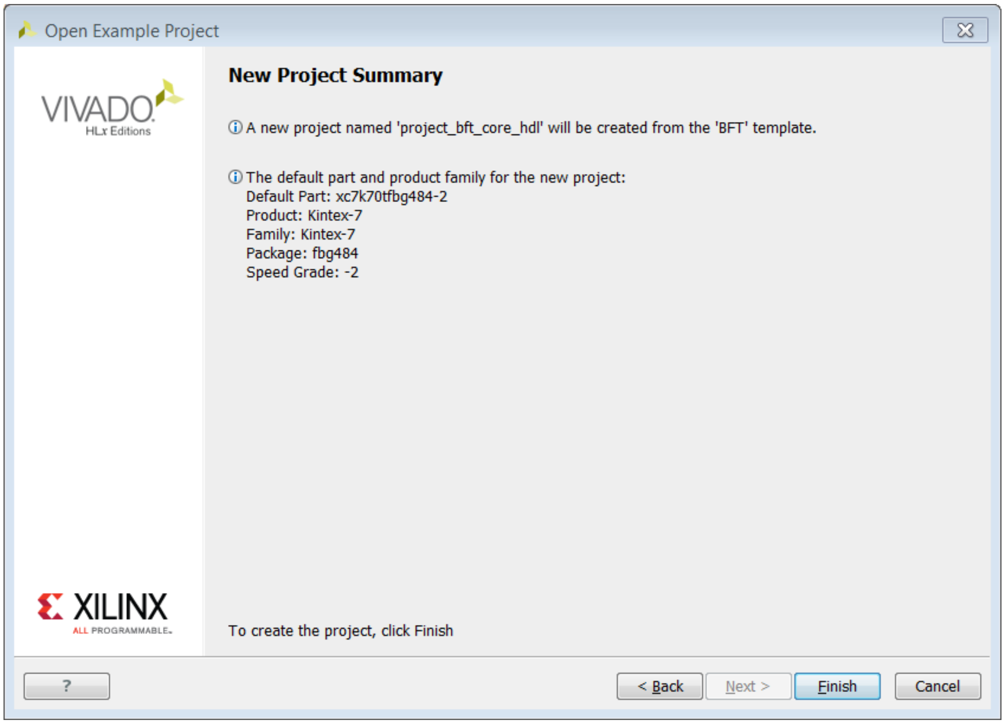


Figure 22: New Project Summary Screen

The Vivado IDE opens with the default view.

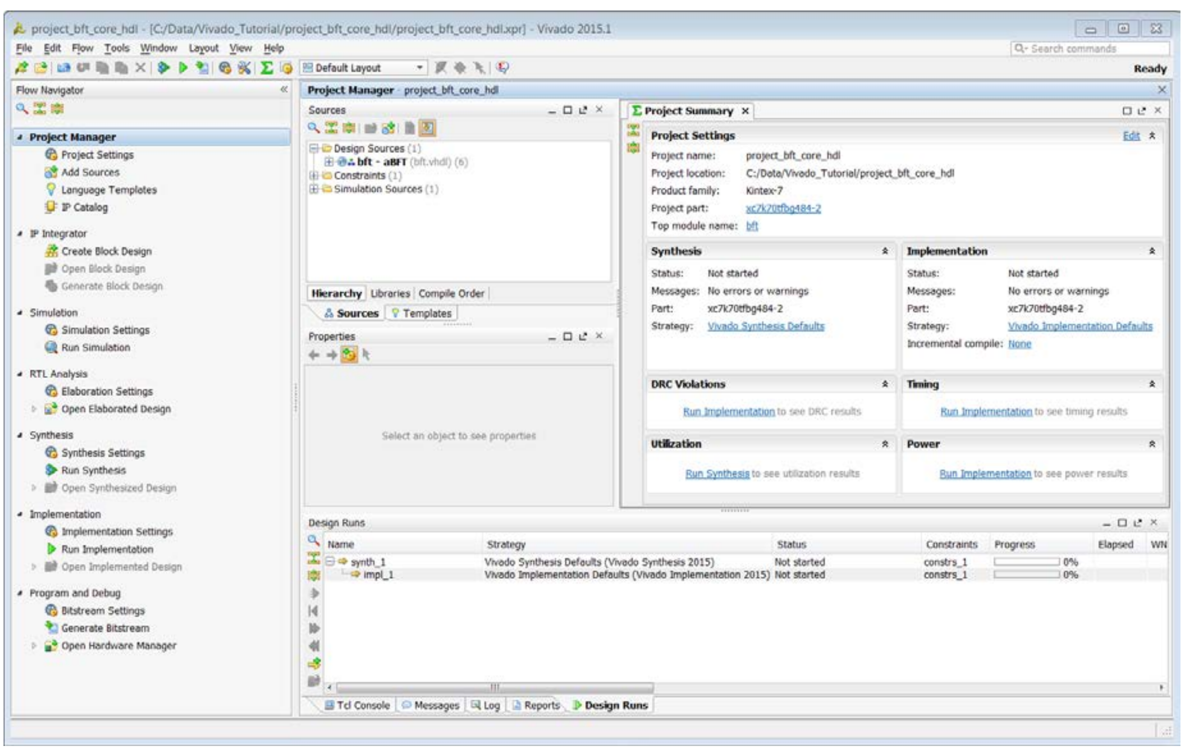


Figure 23: Vivado IDE Showing project_bft_core_hdl Project Details

Step 2: Compiling the Reference Design

1. From the Flow Navigator, select **Run Implementation**.

   The Missing Synthesis Results dialog box opens, as shown in the figure below. The dialog box appears because you are running implementation without first running synthesis. This lets you choose to run synthesis at this time.

2. Click **OK** to launch synthesis first.

   Synthesis runs, and implementation starts automatically when synthesis completes.

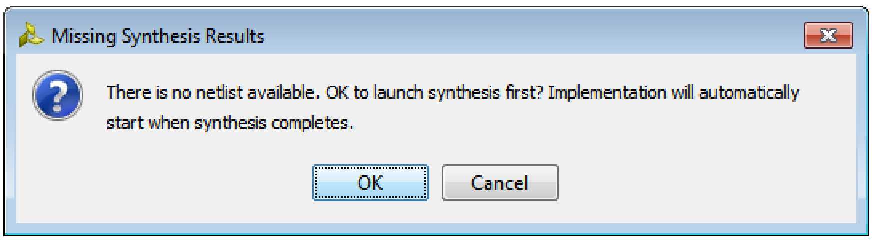


Figure 24: Missing Synthesis Results

   After implementation finishes, the Implementation Complete dialog box opens.

3. Click **Cancel** to dismiss the dialog box.

   In a project-based design, the Vivado Design Suite saves intermediate implementation results as design checkpoints in the implementation runs directory. You will use one of the saved design checkpoints from the implementation in the incremental compile flow.

> **TIP:** *When you re-run implementation the previous results are deleted. Save the intermediate implementation results to a new directory or create a new implementation run for your incremental compile to preserve the reference implementation run directory.*

4. In the Design Runs window, right click on **impl_1** and select **Open Run Directory** from the popup menu.

   This opens the run directory in a file browser as seen in the figure below. The run directory contains the routed checkpoint (`bft_routed.dcp`) to be used later for the incremental compile flow.

   The location of the implementation run directory is a property of the run.

5. Get the location of the current run directory in the Tcl Console by typing:

   ```
   get_property DIRECTORY [current_run]
   ```

   This returns the path to the current run directory that contains the design checkpoint. You can use this Tcl command, and the `DIRECTORY` property, to locate the DCP files needed for the incremental compile flow.

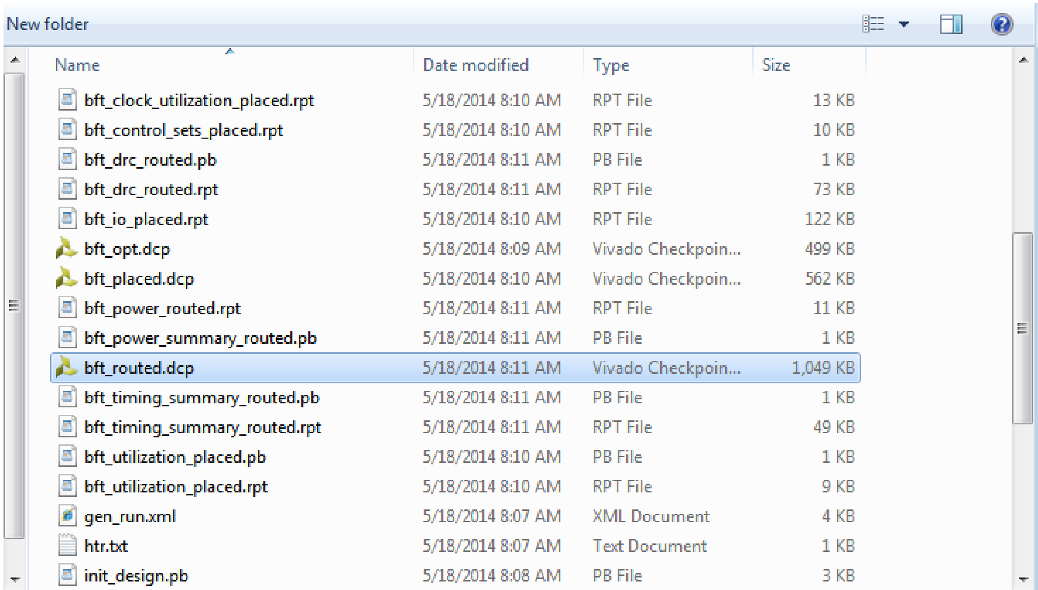


Figure 25: Implementation Run Directory

Step 3: Creating New Runs

In this step, you define new synthesis and implementation runs to preserve the results of the current runs. Then you make changes to the design and rerun synthesis and implementation. If you do not create new runs, Vivado overwrites the current results.

1. From the main menu, select **Flow > Create Runs.**
2. The Create New Runs wizard screen, shown in the following figure, prompts you to create new Synthesis and/or Implementation runs. Select **Both** and click **Next**.

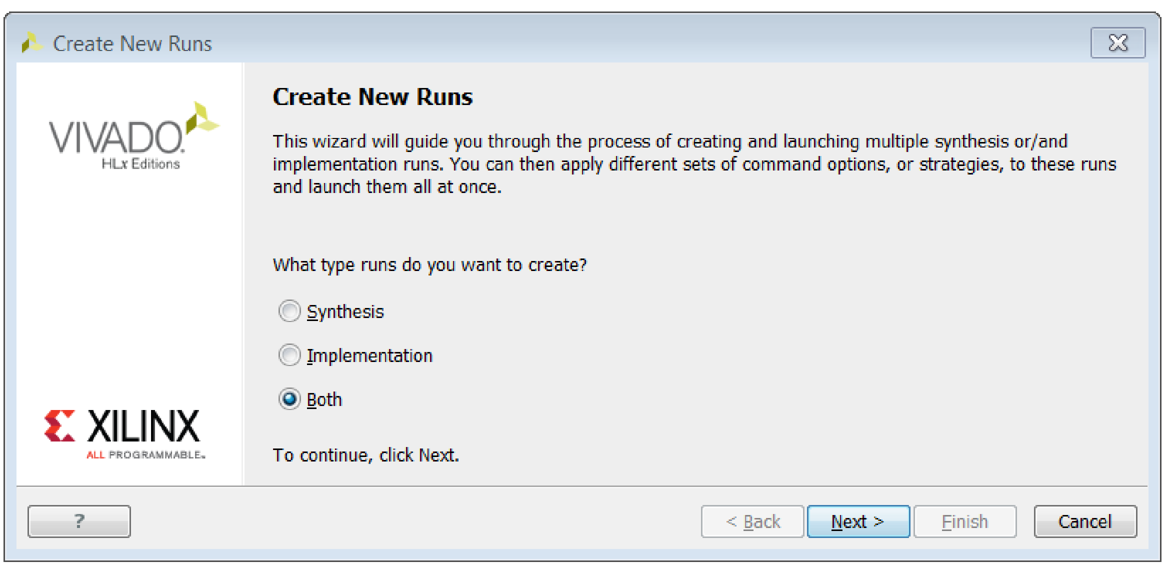


Figure 26: Create New Runs Wizard

3. The Configure Synthesis Runs screen opens. Accept the default run options by clicking **Next**.
4. The Configure Implementation Runs screen opens, as shown in the figure below. Enable the **Make Active** check box, and click **Next**.

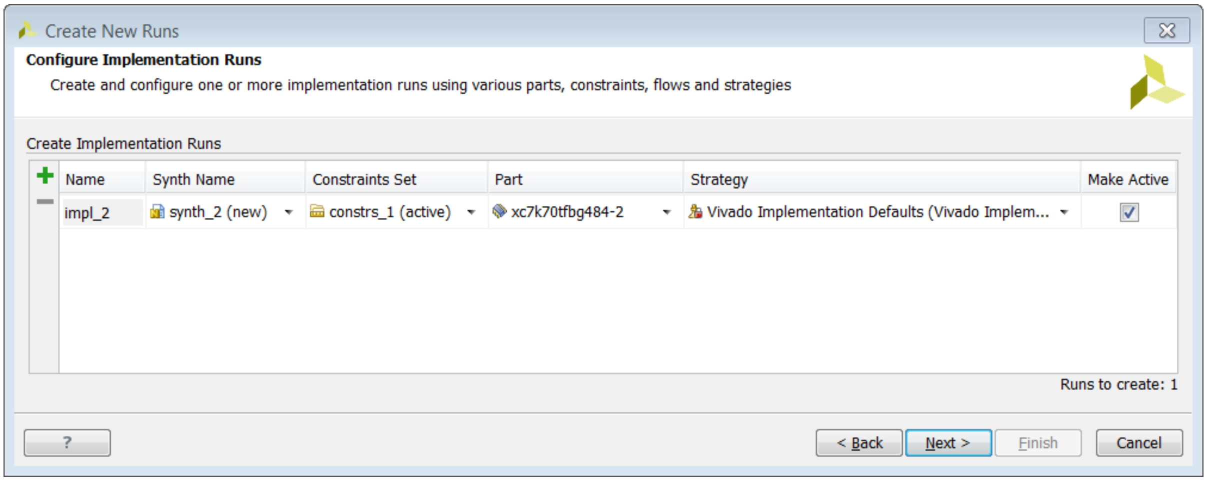


Figure 27: Configure Implementation Runs

5. From the Launch Options window, select **Do not launch now** and click **Next**.
6. In the Create New Runs Summary screen, click **Finish** to create the new runs.

   The Design Runs window displays the new active runs in bold.

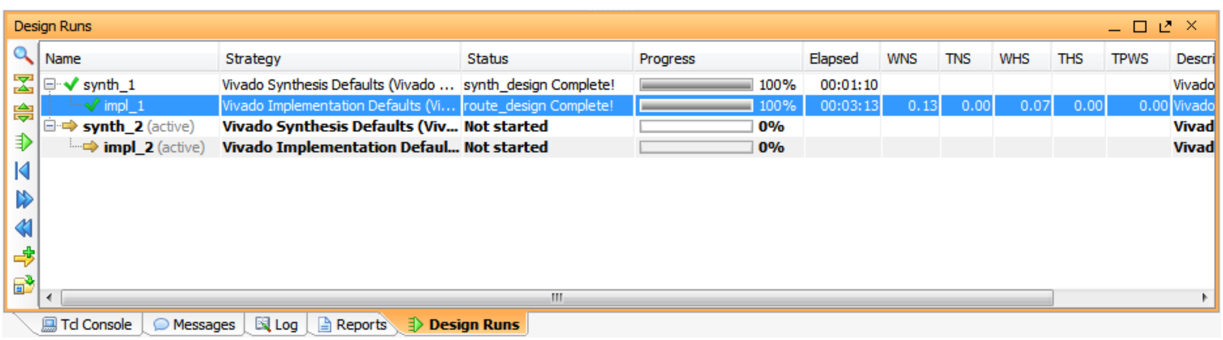


Figure 28: New Design Runs

Step 4: Making Incremental Changes

In this step, you make minor changes to the RTL design sources. These changes necessitate resynthesizing the netlist and re-implementing the design.

1. In the Hierarchy tab of the Sources window, double-click the top-level VHDL file, `bft.vhdl`, to open the file in the Vivado IDE text editor, as shown in the following figure.

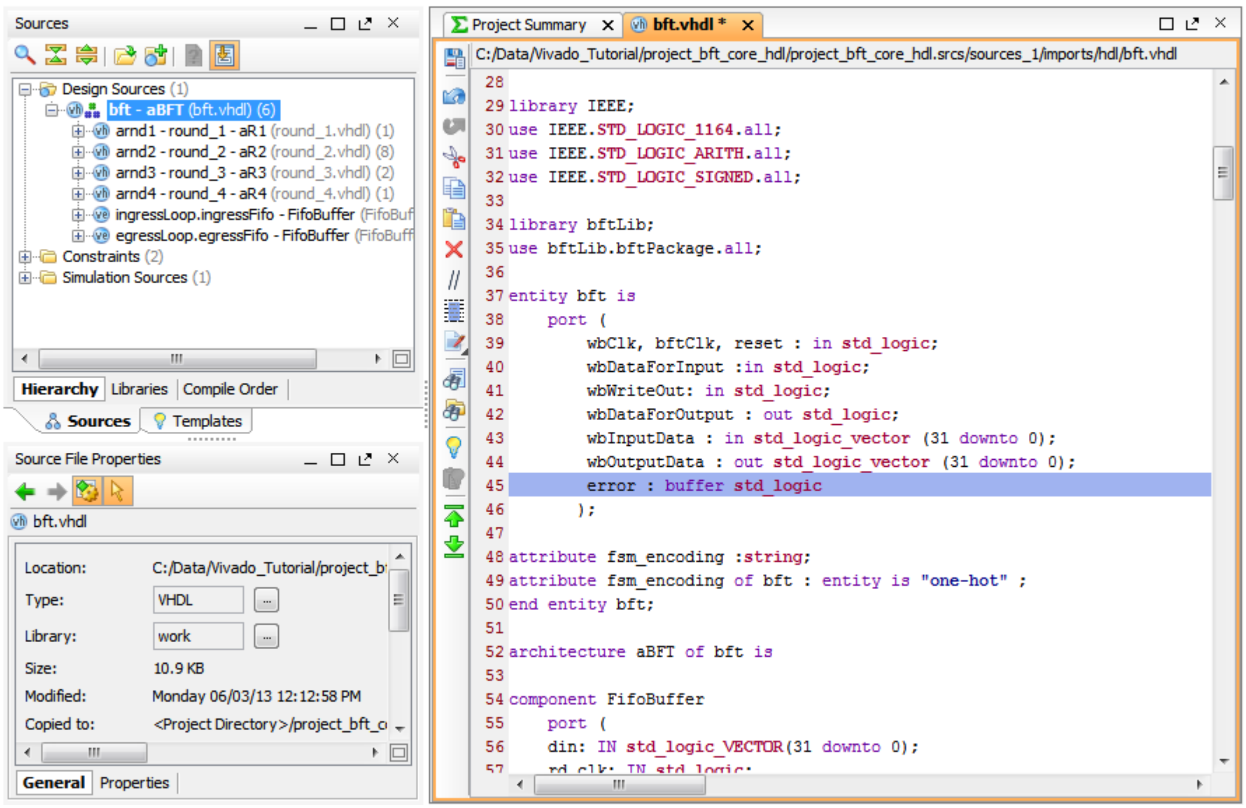


Figure 29: BFT VHDL File

2. Go to **line 45** of the `bft.vhdl` file and change the error port from an output to a buffer, as follows:

```
error : buffer std_logic
```

3. Go to **line 331** of the `bft.vhdl` file and modify the VHDL code, as follows:

From	To

`-- enable the read fifo`
`process (fifoSelect)`
`begin`
`readEgressFifo <= fifoSelect;`
`end process;`
	`-- enable the read fifo`
	`process (fifoSelect, error)`
	`begin`
	`if (error = ‘0’) then`
	`readEgressFifo <=`
	`fifoSelect;`
	`else`
	`readEgressFifo <= (others =>`
	`'0');`
	`end if;`
	`end process;`

The results should look like the following figure.

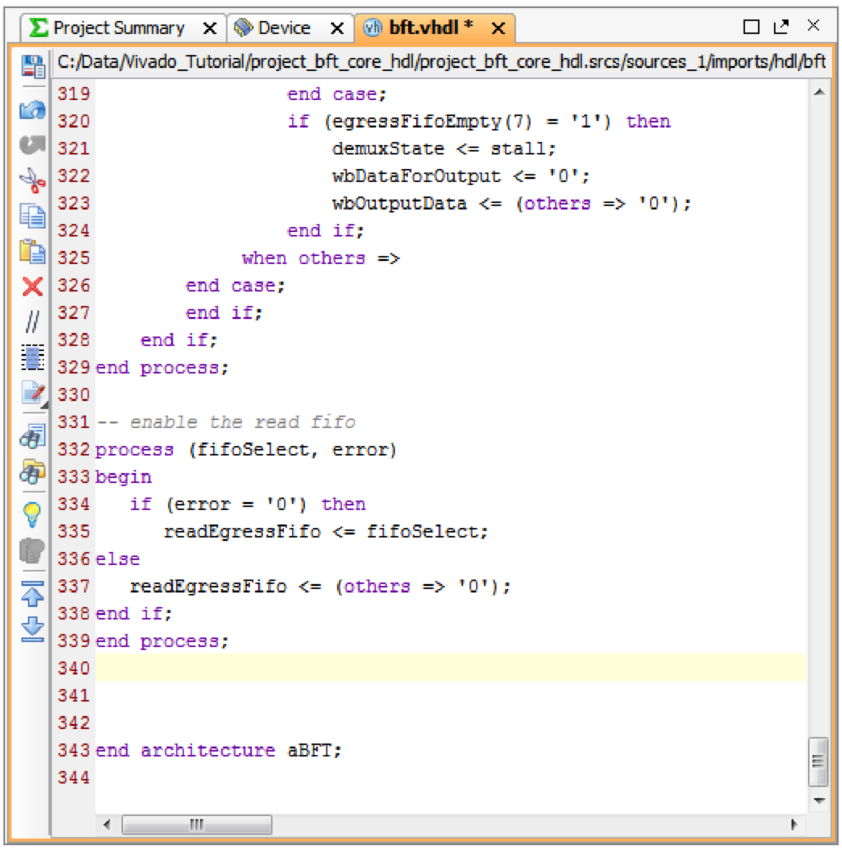


Figure 30: Modified VHDL Code

4. Save the changes by clicking the **Save File** button in the sidebar menu of the text editor.

   As you can see in the following figure, changing the design source files also changes the run status for finished runs from **Complete** to **Out-of-date**.

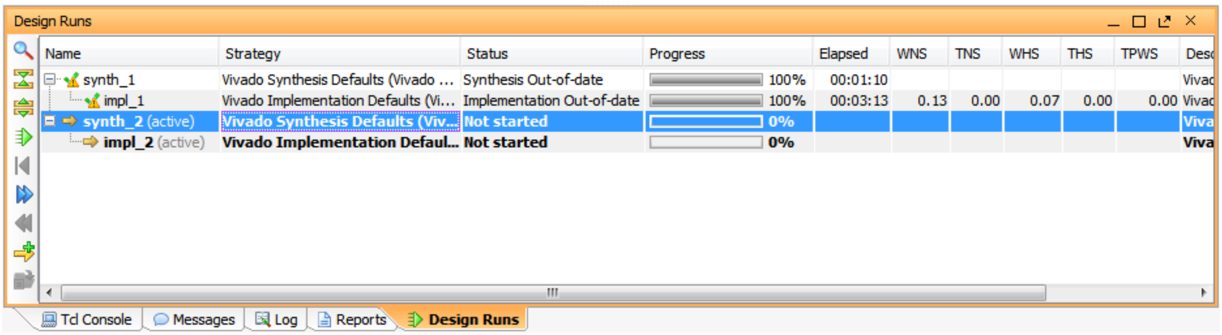


Figure 31: Design Runs Out-of-Date

5. In the Flow Navigator, click **Run Synthesis** to synthesize the updated netlist for the design using the active run, **synth_2**.
6. When the Synthesis Completed dialog box opens, click **Cancel** to close the dialog box.

Step 5: Running Incremental Compile

You updated the design with a few minor changes, necessitating re-synthesis. You could run implementation on the new netlist, to place and route the design and work to meet the timing requirements. However, with only minor changes between this iteration and the last, the incremental compile flow lets you reuse the bulk of your prior placement and routing efforts. This can greatly reduce the time it takes to meet timing on design iterations.

For more information, refer to this link in the *Vivado Design Suite User Guide: Implementation* (UG904).

Start by defining the design checkpoint (DCP) file to use as the reference design for the incremental compile flow. This is the design from which the Vivado Design Suite will draw placement and routing data.

1. In the Design Runs window, right-click the **impl_2** run and select **Set Incremental Compile** from the popup menu.

   The **Set Incremental Compile** dialog box opens, as shown in Figure 32.

2. Click the **Browse** button in the Set Incremental Compile dialog box, and browse to the `./project_bft_core_hdl.runs/impl_1` directory.

3. Select `bft_routed.dcp` as the incremental compile checkpoint.

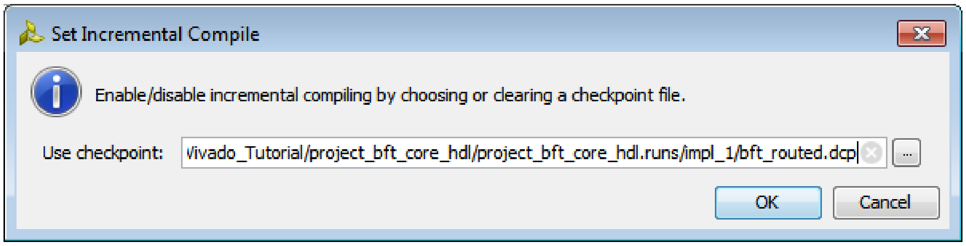


Figure 32: Set Incremental Compile

4. Click **OK**.

   This is stored in the `INCREMENTAL_CHECKPOINT` property of the selected run. Setting this property tells the Vivado Design Suite to run the incremental compile flow during implementation.

5. You can check this property on the current run using the following Tcl command:

   ```
   get_property INCREMENTAL_CHECKPOINT [current_run]
   ```

   This returns the full path to the `bft_routed.dcp` checkpoint.

TIP: *To disable incremental compile for the current run, clear the INCREMENTAL_CHECKPOINT property. This can be done using the Set Incremental Compile dialog box, or by editing the property directly through the Properties window of the design run, or through the* `reset_property` *command.*

6. From the Flow Navigator, select **Run Implementation**.

   This runs implementation on the current run, using the `bft_routed.dcp` file as the reference design for the incremental compile flow. When the run is finished, the Implementation Completed dialog box opens.

7. Select Open Implemented Design and click **OK**.

As shown in the following figure, the Design Runs window shows that the elapsed time for implementation run `impl_2` is the same as for `impl_1` because of the incremental compile flow.

***Note:** This is an extremely small design. The advantages of the incremental compile flow are greater with larger, more complex designs.*

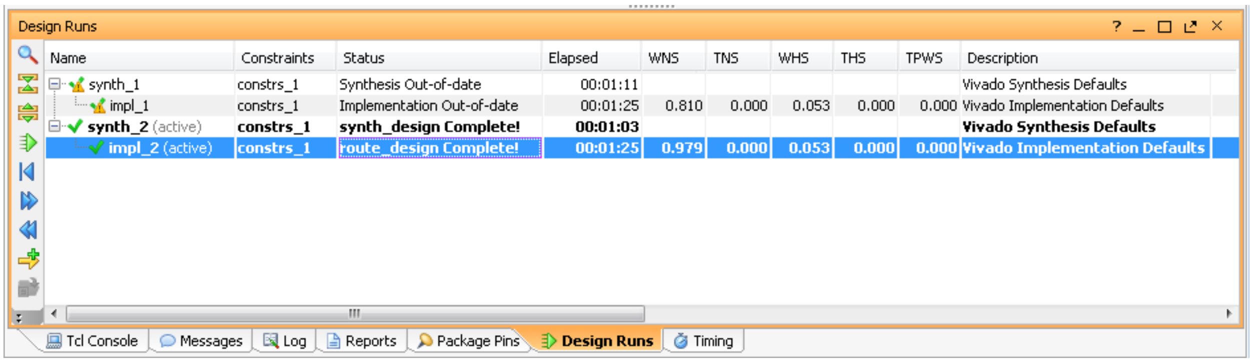


Figure 33: Elapsed Time for Incremental Compile

8. Select the **Reports** tab in the Results Window area and double-click the **Incremental Reuse Report** in the Place Design section, as shown in the following figure.

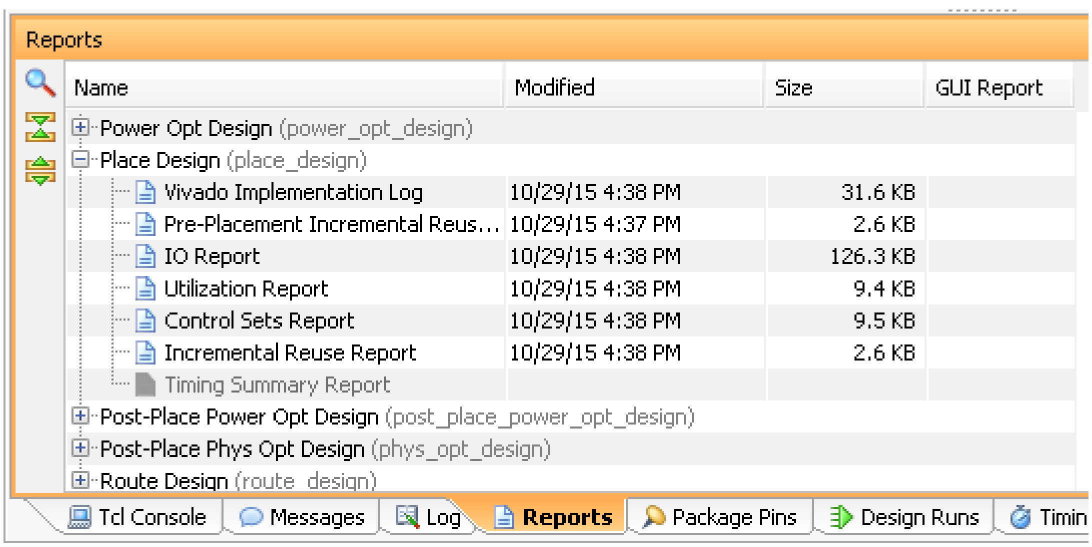
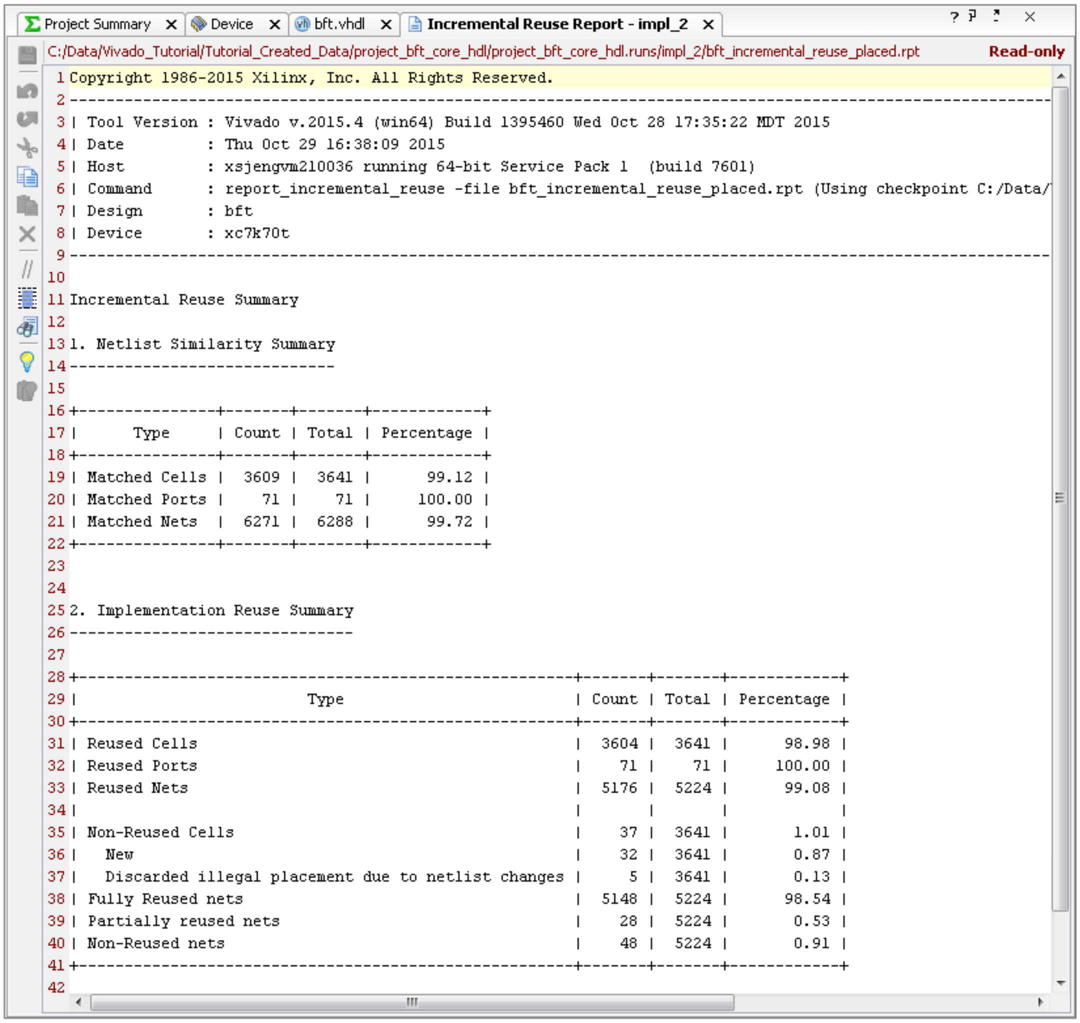


Figure 34: Reports Window

The Incremental Reuse Report opens in the Vivado IDE text editor, as displayed in Figure 35. This report shows the percentage of reused Cells, Ports, and Nets. A higher percentage indicates more effective reuse of placement and routing from the incremental checkpoint.




```
Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.
---------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2015.4 (win64) Build 1395460 Wed Oct 28 17:35:22 MDT 2015
| Date         : Thu Oct 29 16:38:09 2015
| Host         : xsjengvm210036 running 64-bit Service Pack 1  (build 7601)
| Command      : report_incremental_reuse -file bft_incremental_reuse_placed.rpt (Using checkpoint C:/Data/
| Design       : bft
| Device       : xc7k70t
---------------------------------------------------------------------------------------------------------------

Incremental Reuse Summary

1. Netlist Similarity Summary
-----------------------------


+---------------+-------+-------+------------+
|      Type     | Count | Total | Percentage |
+---------------+-------+-------+------------+
| Matched Cells |  3609 |  3641 |      99.12 |
| Matched Ports |    71 |    71 |     100.00 |
| Matched Nets  |  6271 |  6288 |      99.72 |
+---------------+-------+-------+------------+


2. Implementation Reuse Summary
-------------------------------

+--------------------------------------------------------+-------+-------+------------+
|                          Type                          | Count | Total | Percentage |
+--------------------------------------------------------+-------+-------+------------+
| Reused Cells                                           |  3604 |  3641 |      98.98 |
| Reused Ports                                           |    71 |    71 |     100.00 |
| Reused Nets                                            |  5176 |  5224 |      99.08 |
|                                                        |       |       |            |
| Non-Reused Cells                                       |    37 |  3641 |       1.01 |
|   New                                                  |    32 |  3641 |       0.87 |
|   Discarded illegal placement due to netlist changes   |     5 |  3641 |       0.13 |
| Fully Reused nets                                      |  5148 |  5224 |      98.54 |
| Partially reused nets                                  |    28 |  5224 |       0.53 |
| Non-Reused nets                                        |    48 |  5224 |       0.91 |
+--------------------------------------------------------+-------+-------+------------+

```

Figure 35: Incremental Reuse Report

In the report, fully reused nets indicate that the entire routing of the nets is reused from the reference design. Partially reused nets indicate that some of the routing of the nets reuses routing from the reference design. Some segments re-route due to changed cells, changed cell placements, or both. Non-reused nets indicate that the net in the current design was not matched in the reference design.

Conclusion

This concludes Lab #2. You can close the current project and exit the Vivado IDE.

In this lab, you learned how to run the Incremental Compile flow, using a checkpoint from a previously implemented design. You also examined the similarity between a reference design checkpoint and the current design by examining the Incremental Reuse Report.

Introduction

In this lab, you learn how to use the Vivado® IDE to assign routing to nets for precisely controlling the timing of a critical portion of the design.

- You will use the BFT HDL example design that is included in the Vivado Design Suite.
- To illustrate the manual routing feature, you will precisely control the skew within the output bus of the design, `wbOutputData`.

Step 1: Opening the Example Project

1. Start by loading Vivado IDE by doing one of the following:
   - Launch the Vivado IDE from the icon on the Windows desktop.
   - Type vivado from a command terminal.
2. From the Getting Started page, click **Open Example Project**.

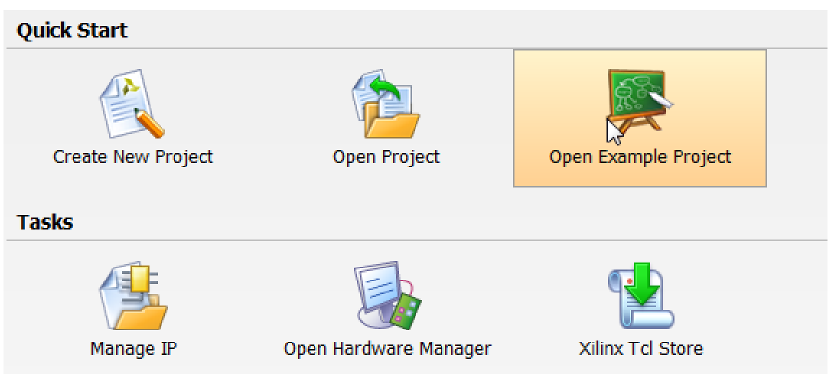


Figure 36: Open Example Design

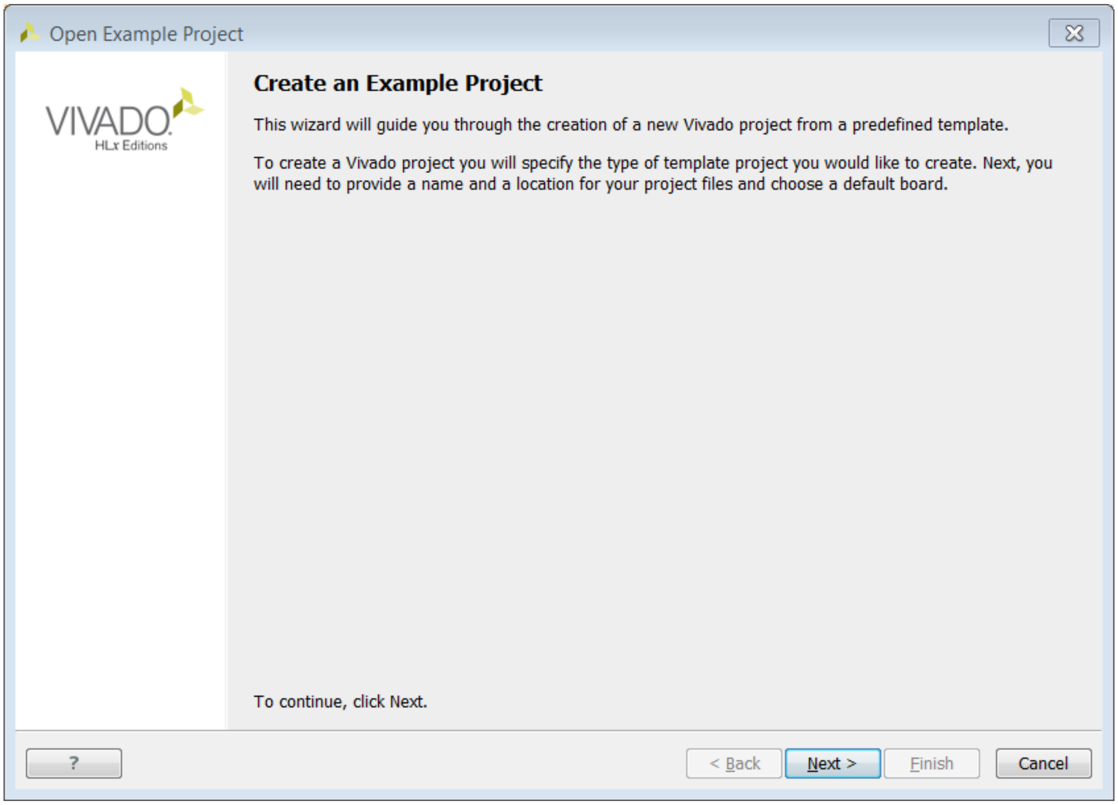


Figure 37: Open Example Project

3. Click **Next** in the Open Example Project window**.**

4. Select the BFT (Small RTL project) design.

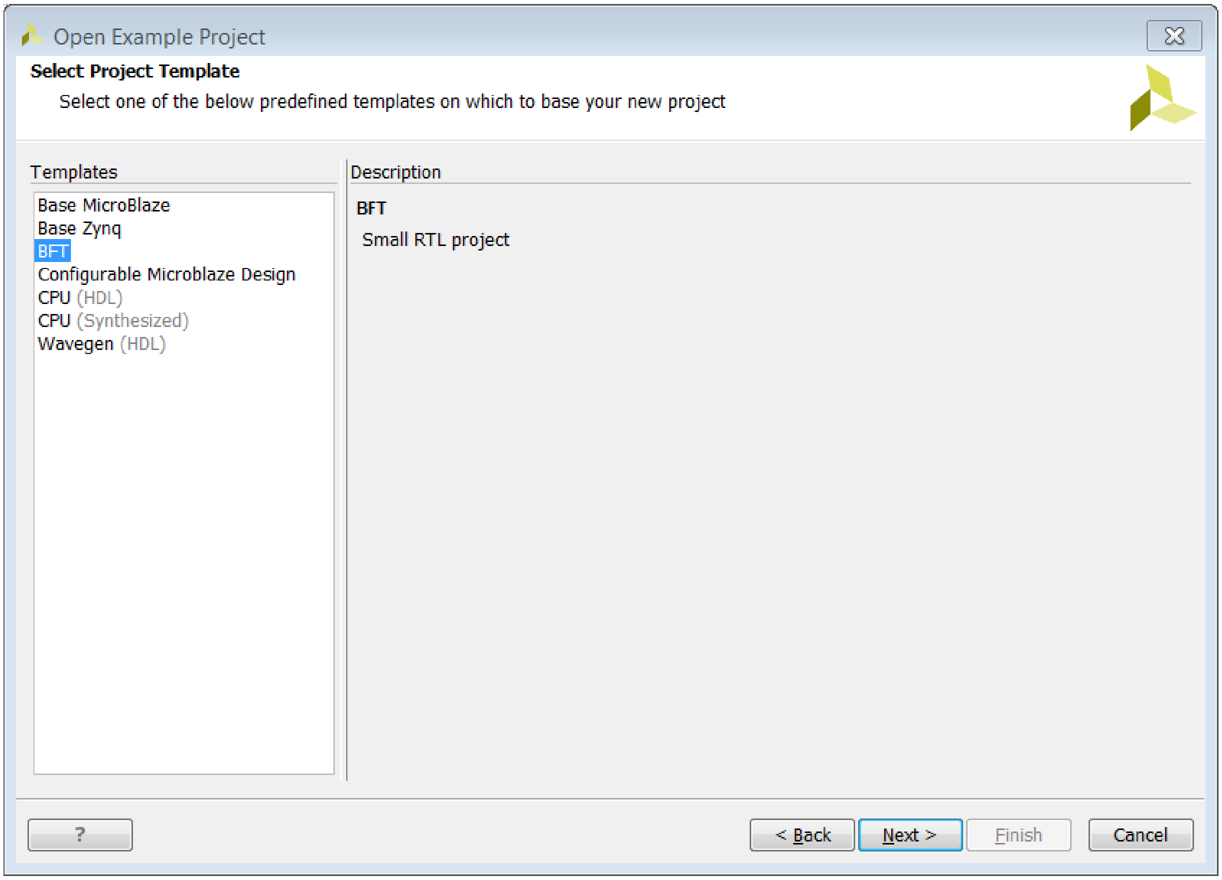


Figure 38: Open Example Project, Select Example Project Screen

5. Specify the following:
   - Project name: **project_bft_core**
   - Project location: <Project_Dir>

6. Click **Next**.

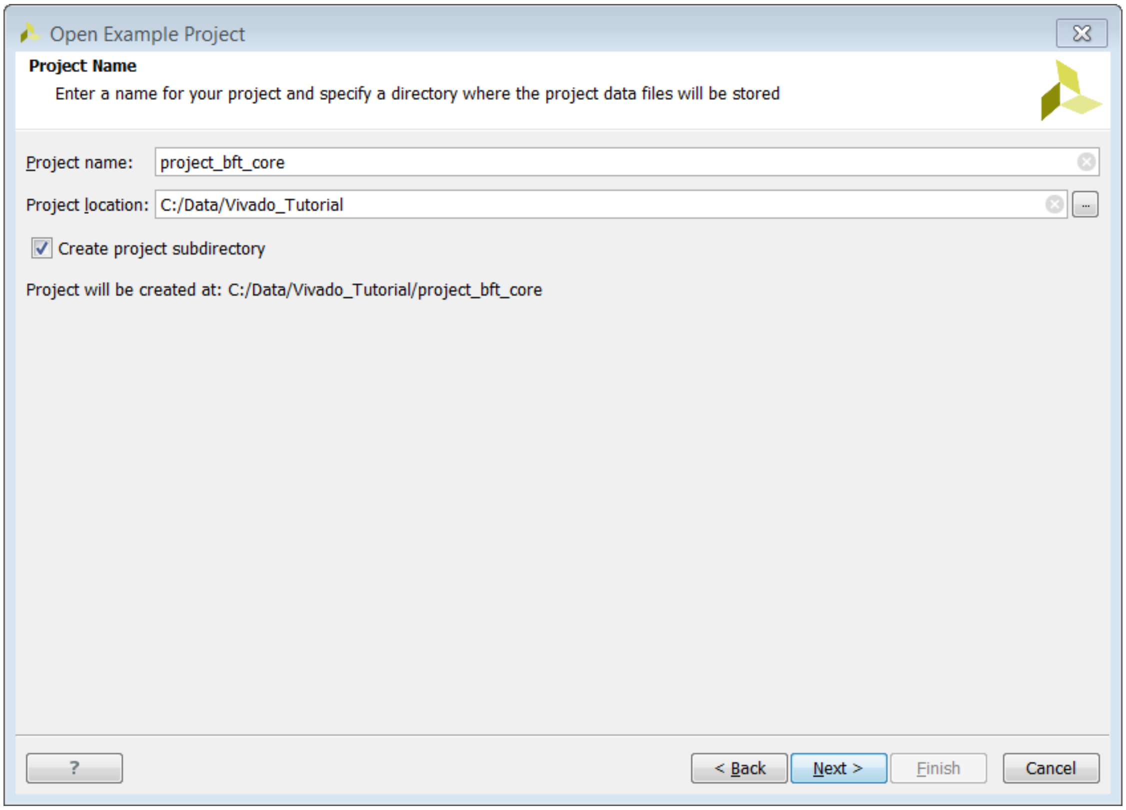


Figure 39: Project Name Screen

7. Select the **xc7k70tfbg484-2** as your Default Part.

8. Click **Next**.

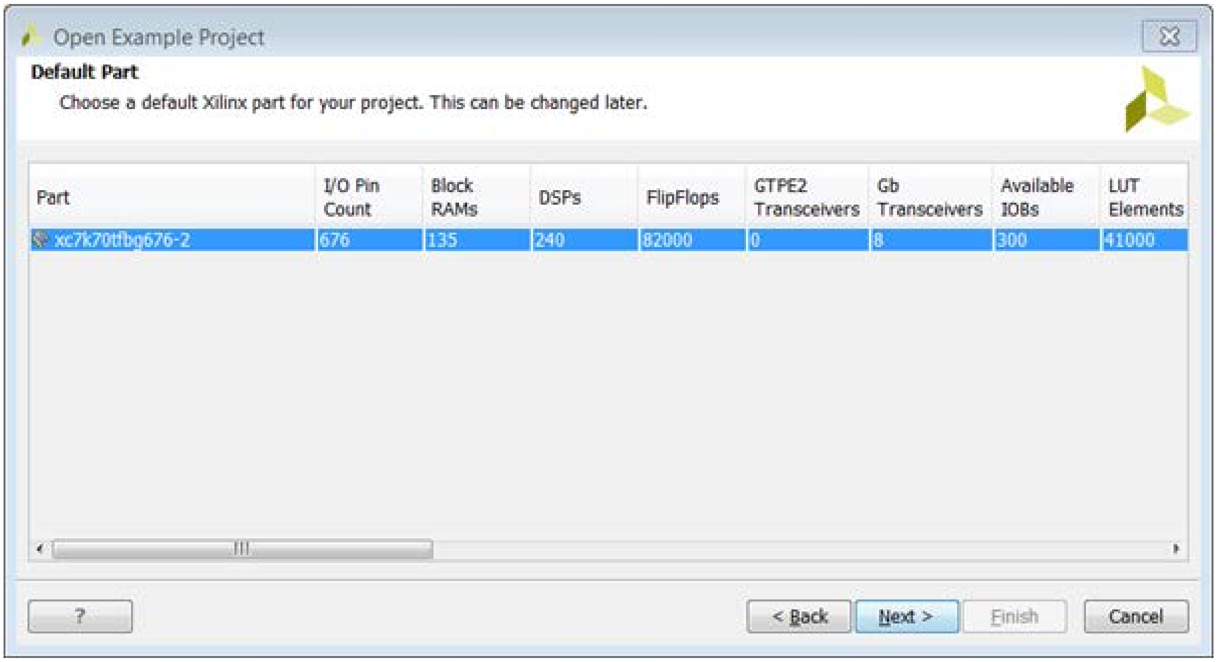


Figure 40: Default Part Screen

9. Click **Finish**.

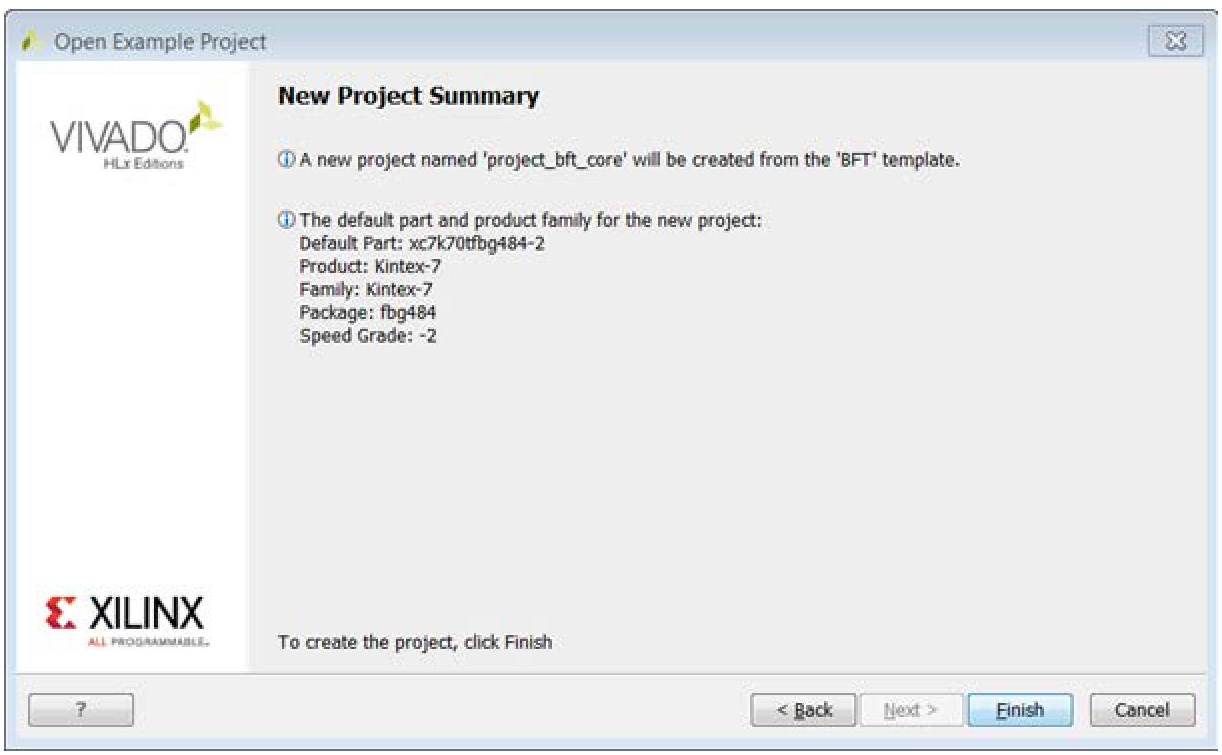


Figure 41: New Project Summary Screen

The Vivado IDE opens with the default view, as shown in the following figure.

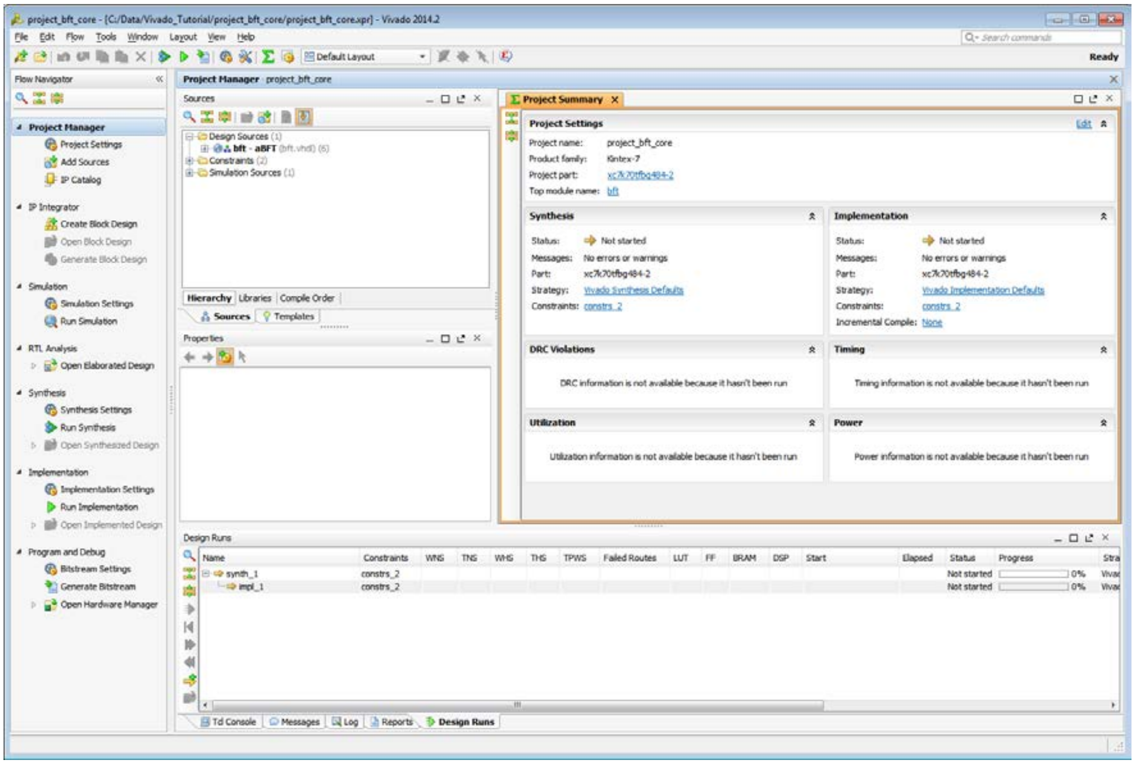


Figure 42: BFT Core Project Summary

Step 2: Performing Place and Route on the Design

1. In the Flow Navigator, click **Run Implementation**.

   The Missing Synthesis Results dialog box opens to inform you that there is no synthesized netlist to implement, and prompts you to start synthesis first.

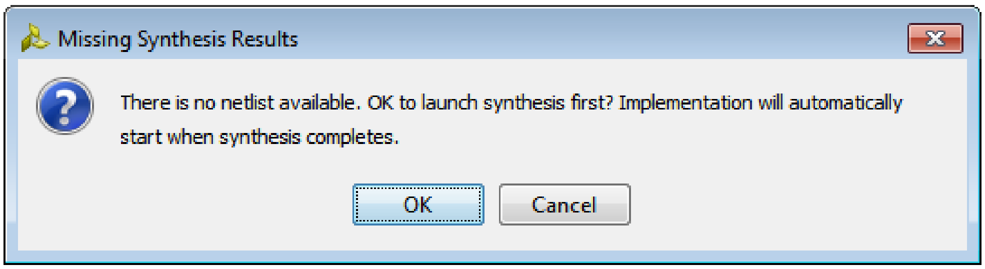


Figure 43: Missing Synthesis Results

2. Click **OK** to launch synthesis first.

   Implementation automatically starts after synthesis completes, and the Implementation Completed dialog box opens when complete, as shown in the following figure.

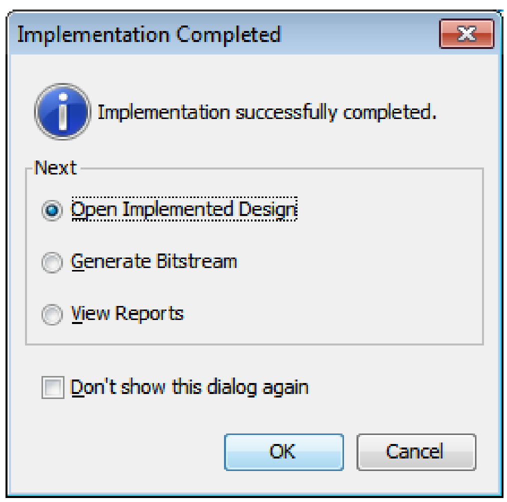


Figure 44: Implementation Completed

3. In the Implementation Completed dialog box, select **Open Implemented Design** and click **OK**.

   The Device window opens, displaying the placement results.

4. Click the **Routing Resources** button to view the detailed routing resources in the Device window.

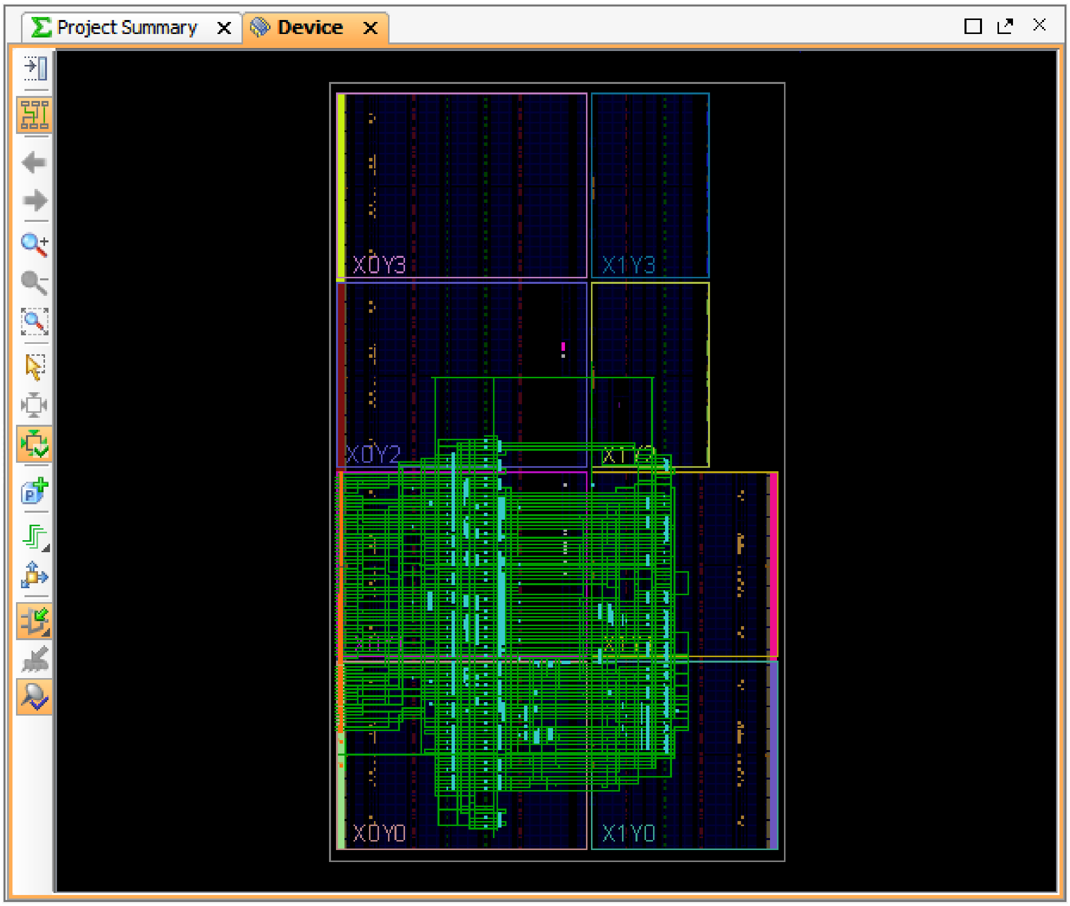


Figure 45: Device View

Step 3: Analyzing Output Bus Timing

IMPORTANT: *The tutorial design has an output data bus,* `wbOutputData`*, which feeds external logic. Your objective is to precisely control timing skew by manually routing the nets of this bus.*

You can use the Report Datasheet command to analyze the current timing of members of the output bus, `wbOutputData`. The Report Datasheet command lets you analyze the timing of a group of ports with respect to a specific reference port.

1. From the main menu, select **Tools > Timing > Report Datasheet**.
2. Select the **Groups** tab in the Report Datasheet dialog box, as seen in the following figure, and enter the following:
   - o Reference: `[get_ports {wbOutputData[0]}]`
   - o Ports: `[get_ports {wbOutputData[*]}]`

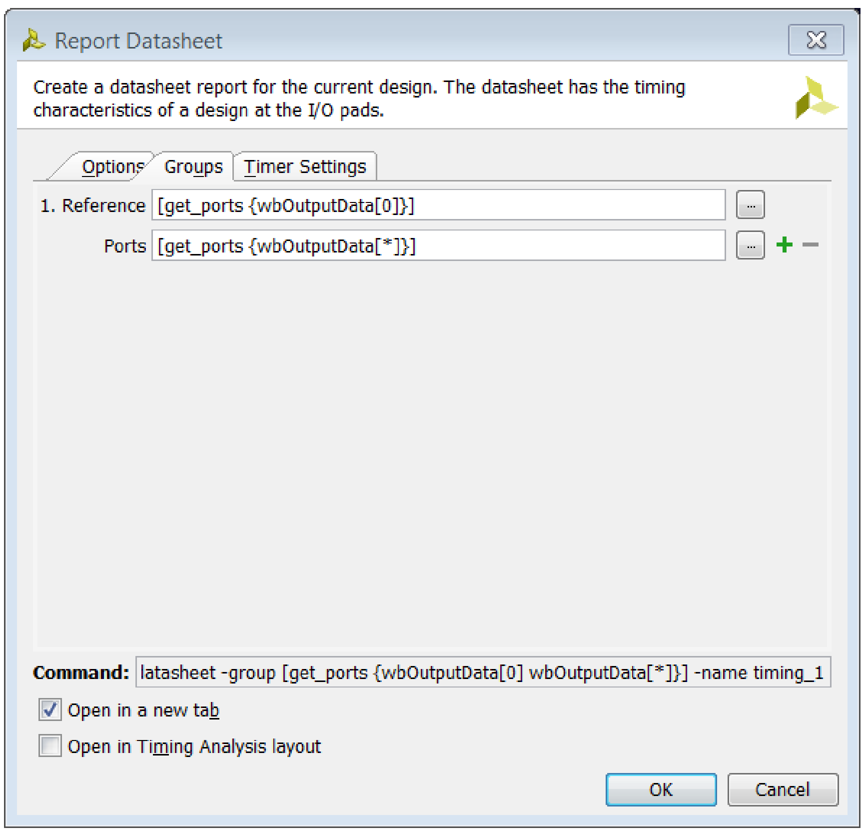


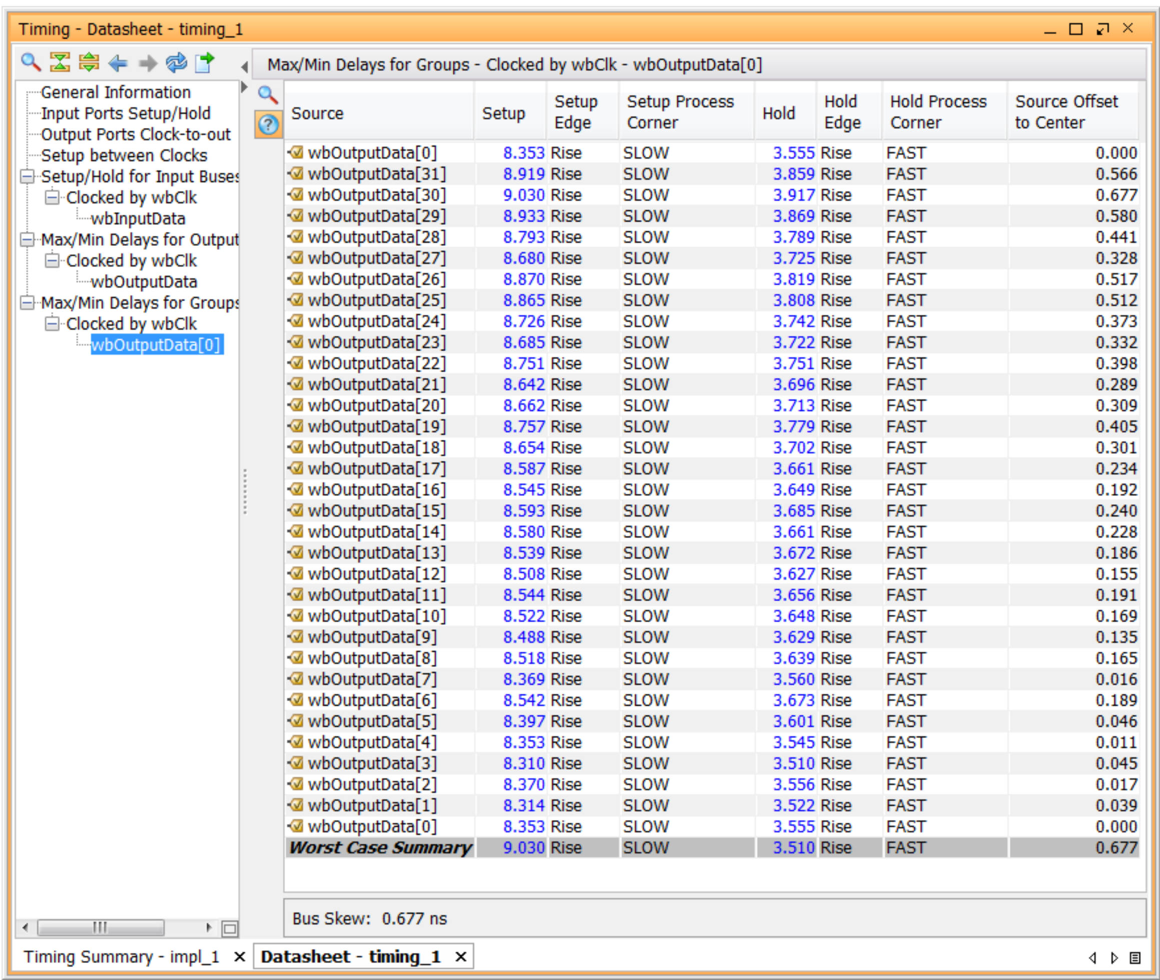
Figure 46: Report Datasheet

3. Click **OK**.

In this case, you are examining the timing at the ports carrying the `wbOutputData` bus, relative to the first bit of the bus, `wbOutputData[0]`. This allows you to quickly determine the relative timing differences between the different bits of the bus.

4. Click the **Maximize** button to maximize the **Timing - Datasheet** window and expand the results.
5. Select the Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0] section, as seen in the following figure.

   You can see from the report that the timing skew across the `wbOutputData` bus varies by almost 700 ps. The goal is to minimize the skew across the bus to less than 100 ps.



Source	Setup	Setup Edge	Setup Process Corner	Hold	Hold Edge	Hold Process Corner	Source Offset to Center
wbOutputData[0]	8.353	Rise	SLOW	3.555	Rise	FAST	0.000
wbOutputData[31]	8.919	Rise	SLOW	3.859	Rise	FAST	0.566
wbOutputData[30]	9.030	Rise	SLOW	3.917	Rise	FAST	0.677
wbOutputData[29]	8.933	Rise	SLOW	3.869	Rise	FAST	0.580
wbOutputData[28]	8.793	Rise	SLOW	3.789	Rise	FAST	0.441
wbOutputData[27]	8.680	Rise	SLOW	3.725	Rise	FAST	0.328
wbOutputData[26]	8.870	Rise	SLOW	3.819	Rise	FAST	0.517
wbOutputData[25]	8.865	Rise	SLOW	3.808	Rise	FAST	0.512
wbOutputData[24]	8.726	Rise	SLOW	3.742	Rise	FAST	0.373
wbOutputData[23]	8.685	Rise	SLOW	3.722	Rise	FAST	0.332
wbOutputData[22]	8.751	Rise	SLOW	3.751	Rise	FAST	0.398
wbOutputData[21]	8.642	Rise	SLOW	3.696	Rise	FAST	0.289
wbOutputData[20]	8.662	Rise	SLOW	3.713	Rise	FAST	0.309
wbOutputData[19]	8.757	Rise	SLOW	3.779	Rise	FAST	0.405
wbOutputData[18]	8.654	Rise	SLOW	3.702	Rise	FAST	0.301
wbOutputData[17]	8.587	Rise	SLOW	3.661	Rise	FAST	0.234
wbOutputData[16]	8.545	Rise	SLOW	3.649	Rise	FAST	0.192
wbOutputData[15]	8.593	Rise	SLOW	3.685	Rise	FAST	0.240
wbOutputData[14]	8.580	Rise	SLOW	3.661	Rise	FAST	0.228
wbOutputData[13]	8.539	Rise	SLOW	3.672	Rise	FAST	0.186
wbOutputData[12]	8.508	Rise	SLOW	3.627	Rise	FAST	0.155
wbOutputData[11]	8.544	Rise	SLOW	3.656	Rise	FAST	0.191
wbOutputData[10]	8.522	Rise	SLOW	3.648	Rise	FAST	0.169
wbOutputData[9]	8.488	Rise	SLOW	3.629	Rise	FAST	0.135
wbOutputData[8]	8.518	Rise	SLOW	3.639	Rise	FAST	0.165
wbOutputData[7]	8.369	Rise	SLOW	3.560	Rise	FAST	0.016
wbOutputData[6]	8.542	Rise	SLOW	3.673	Rise	FAST	0.189
wbOutputData[5]	8.397	Rise	SLOW	3.601	Rise	FAST	0.046
wbOutputData[4]	8.353	Rise	SLOW	3.545	Rise	FAST	0.011
wbOutputData[3]	8.310	Rise	SLOW	3.510	Rise	FAST	0.045
wbOutputData[2]	8.370	Rise	SLOW	3.556	Rise	FAST	0.017
wbOutputData[1]	8.314	Rise	SLOW	3.522	Rise	FAST	0.039
wbOutputData[0]	8.353	Rise	SLOW	3.555	Rise	FAST	0.000
Worst Case Summary	9.030	Rise	SLOW	3.510	Rise	FAST	0.677

Bus Skew: 0.677 ns

Figure 47: Report Datasheet Results

6. Click the **Restore** button so you can simultaneously see the Device window and the Timing - Datasheet results.

7. Click the hyperlink for the Setup time of the source `wbOutputData[31]`.

   This highlights the path in the Device view window that is currently open.

   Note: *Make sure that the **Autofit Selection** is highlighted in the Device window so you can see the entire path, as shown in the following figure.*

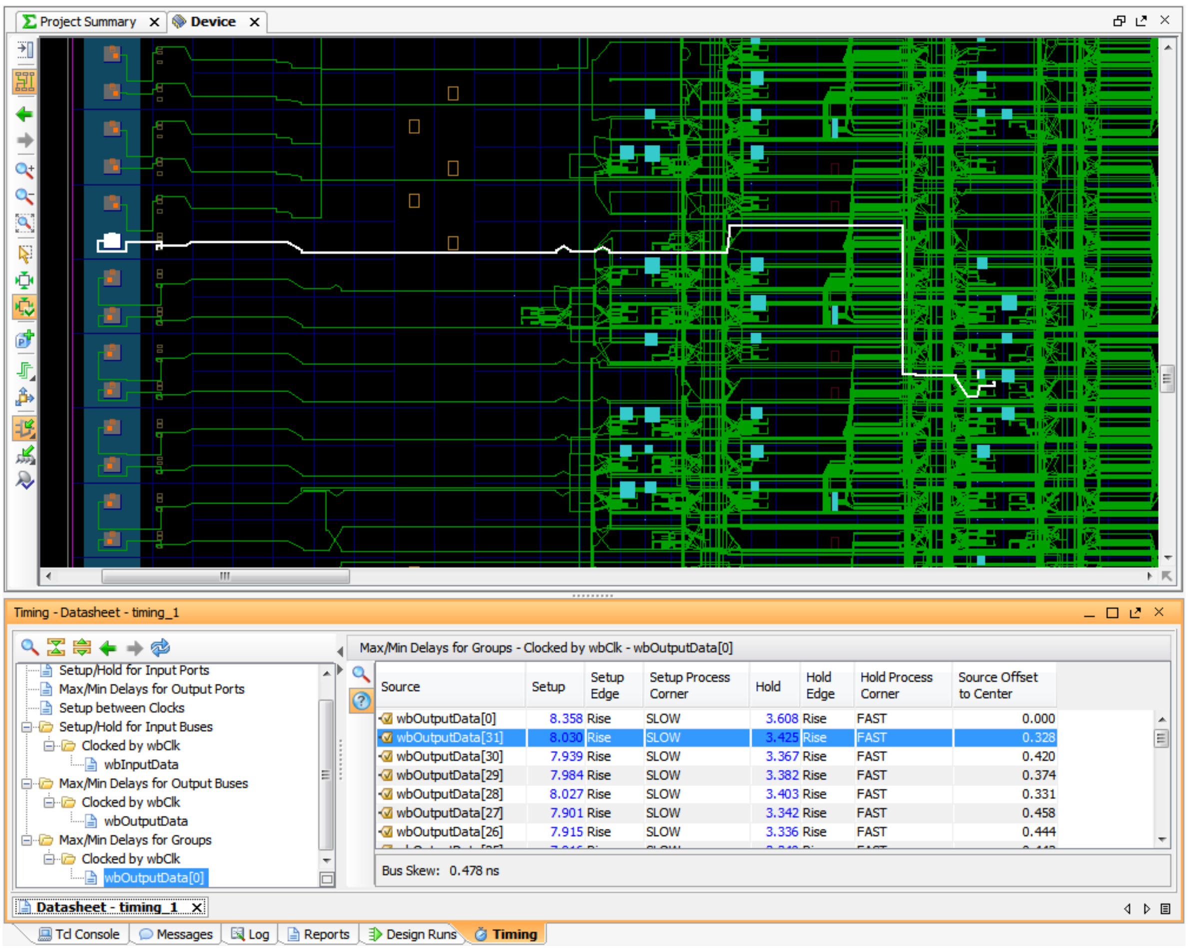


Figure 48: Detailed Routing View

8. In the Device window, right click on the **highlighted path** and select **Schematic** from the popup menu.

   This displays the schematic for the selected output data bus, as shown in Figure 49. From the schematic, you can see that the output port is directly driven from a register through an output buffer (OBUF).

   If you can consistently control the placement of the register with respect to the output pins on the bus and control the routing between registers and the outputs, you can minimize skew between the members of the output bus.

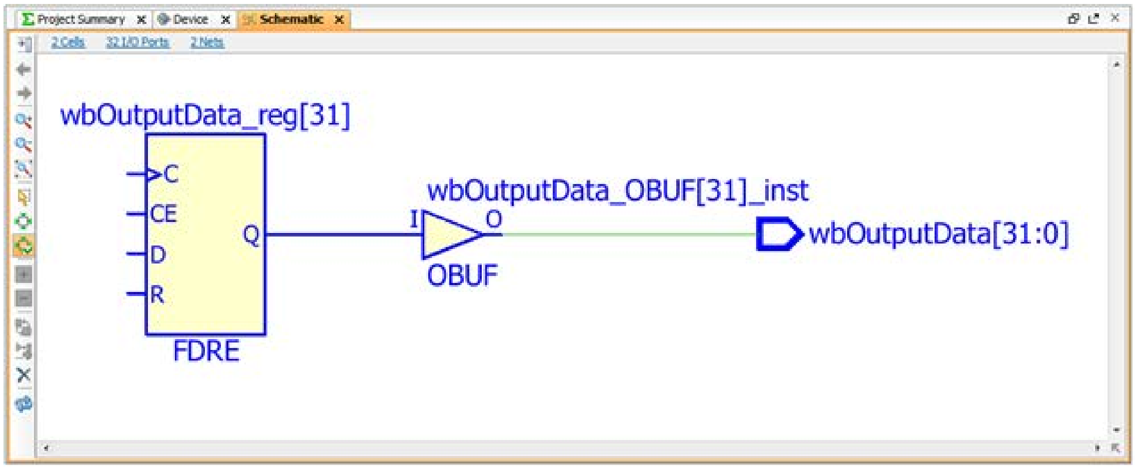


Figure 49: Schematic View of Output Path

9. Change to the Device window.

   To better visualize the placement of the registers and outputs, you can use the `mark_objects` command to mark them in the Device view.

10. From the Tcl Console, type the following commands:

```
mark_objects -color blue [get_ports wbOutputData[*]]
mark_objects -color red [get_cells wbOutputData_reg[*]]
```

   Blue diamond markers show on the output ports, and red diamond markers show on the registers feeding the outputs, as seen in the following figure.

Figure 50: Marked Routes

Zooming out on the Device window displays a picture similar to Figure 50.

The outputs marked in blue are spread out along two banks on the left side starting with `wbOutputData[0]` (on the bottom) and ending with `wbOutputData[31]` (at the top), while the output registers marked in red are clustered close together on the right.

To look at all of the routing from the registers to the outputs, you can use the `highlight_objects` Tcl command to highlight the nets.

11. Type the following command at the Tcl prompt:

```
highlight_objects -color yellow [get_nets -of [get_pins -of [get_cells\
wbOutputData_reg[*]] -filter DIRECTION==OUT]]
```

This highlights all the nets connected to the output pins of the `wbOutputData_reg[*]` registers as shown in Figure 51.

In the Device window, you can see that there are various routing distances between the clustered output registers and the distributed outputs pads of the bus. Consistently placing the output registers in the slices next to each output port eliminates a majority of the variability in the clock-to-out delay of the `wbOutputData` bus.

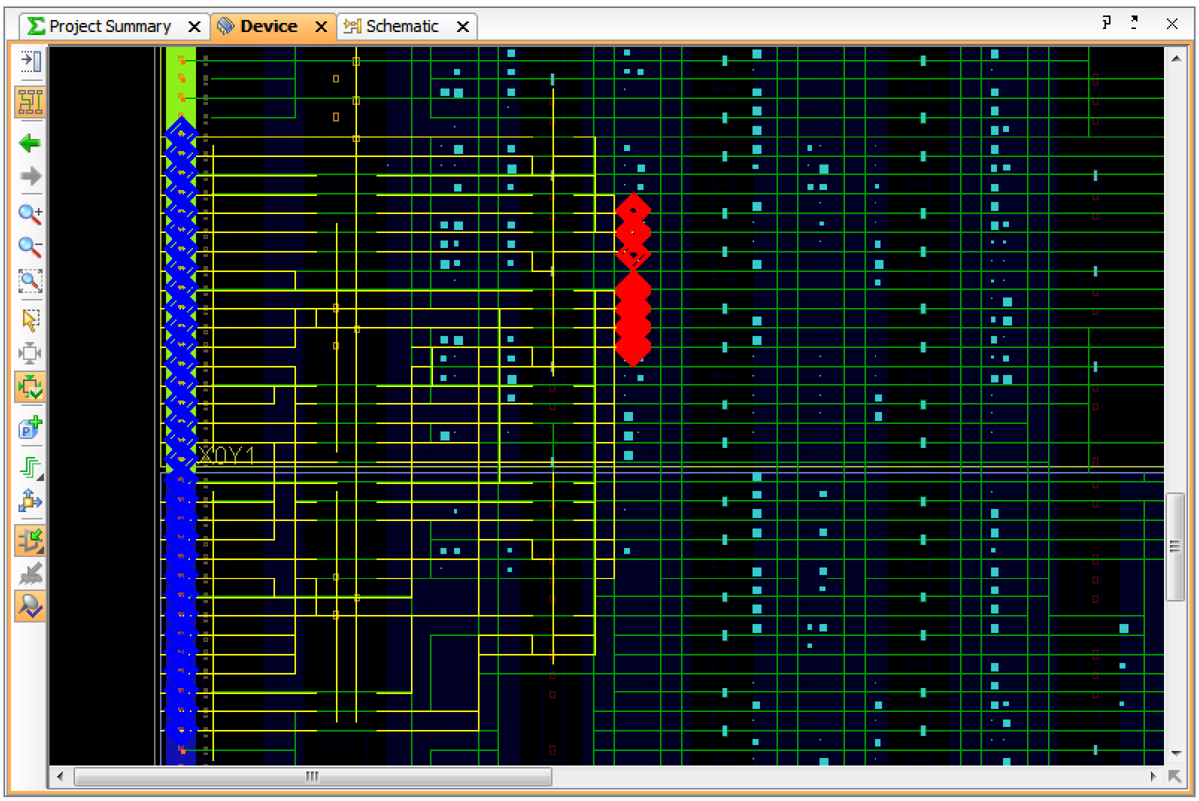


Figure 51: Highlighted Routes

12. Click the **Unhighlight All** button  and the **Unmark All** button  in the main toolbar menu.

Step 4: Improving Bus Timing through Placement

To improve the timing of the `wbOutputData` bus you will place the output registers closer to their respective output pads, then rerun timing to look for any improvement. To place the output registers, you will identify potential placement sites, and then use a sequence of Tcl commands, or a Tcl script, to place the cells and reroute the connections.

RECOMMENDED: *Use a series of Tcl commands to place the output registers in the slices next to the wbOutPutData bus output pads.*

1. In the Device window click to **disable Routing Resources** and make sure **AutoFit Selection** is still enabled on the sidebar menu.

   This lets you see placed objects more clearly in the Device window, without the added details of the routing.

2. Select the wbOutputData ports placed on the I/O blocks with the following Tcl command:

   ```
   select_objects [get_ports wbOutputData*]
   ```

   The Device window will show the selected ports highlighted in white, and zoom to fit the selection. The view should be similar to Figure 52. By examining the device resources around the selected ports, you can identify a range of placement Sites for the output registers.

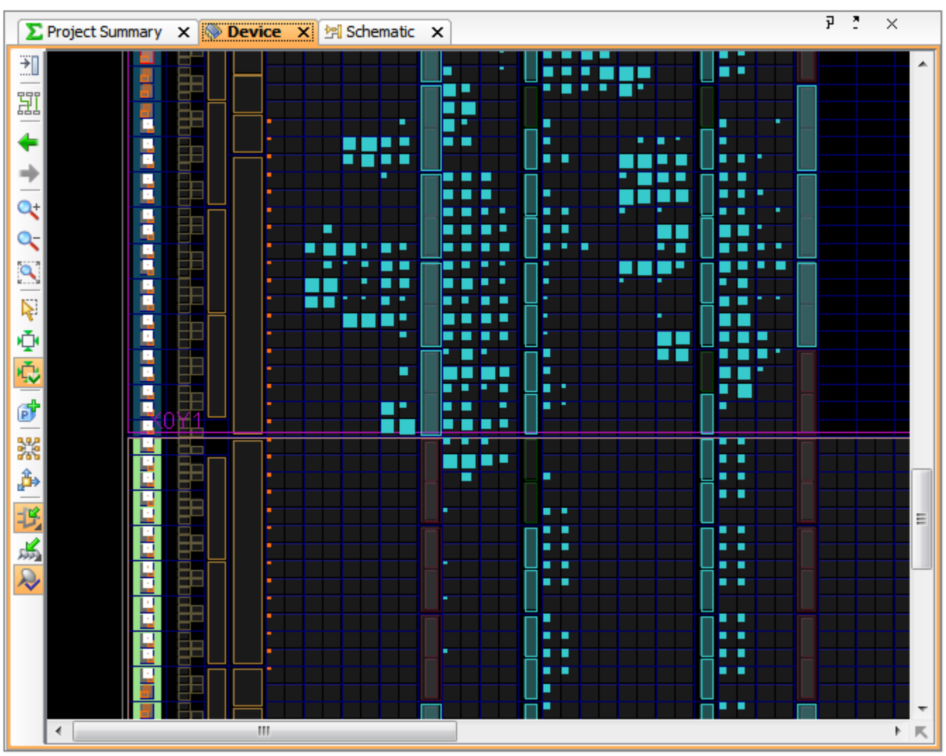


Figure 52: Selected wbOutputData Ports

3. Zoom into the Device window around the bottom selected output ports. The following figure shows the results.

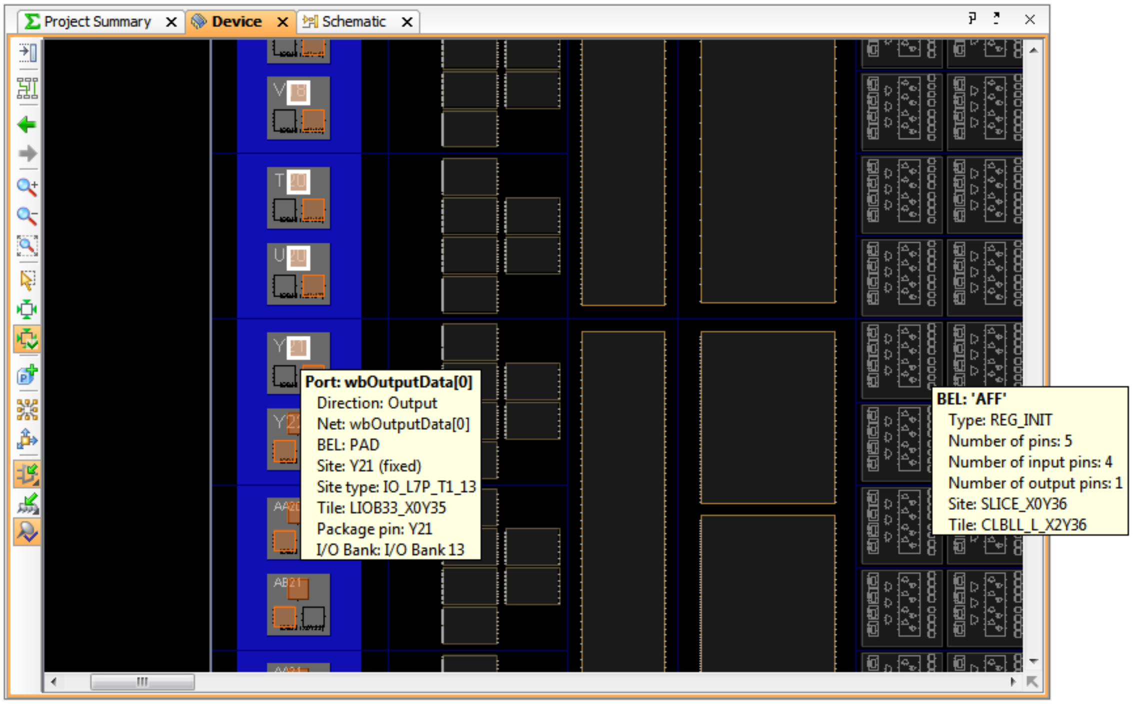


Figure 53: wbOutputData[0] Placement Details

The bottom ports are the lowest bits of the output bus, starting with `wbOutputData[0]`.

As seen in Figure 53, this port is placed on Package Pin Y21. Over to the right, where the Slice logic contains the device resources needed to place the output registers, the Slice coordinates are X0Y36. You will use that location as the starting placement for the 32 output registers, `wbOutputData_reg[31:0]`.

By scrolling or panning in the Device window, you can visually confirm that the highest output data port, `wbOutputData[31]`, is placed on Package Pin K22, and the registers to the right are in Slice X0Y67.

Now that you have identified the placement resources needed for the output registers, you must make sure they are available for placing the cells. You will do this by quickly unplacing the Slices to clear any currently placed logic.

4. Unplace any cells currently assigned to the range of slices needed for the output registers, `SLICE_X0Y36` to `SLICE_X0Y67`, with the following Tcl command:

```
for {set i 0} {$i<32} {incr i}  {
   unplace_cell [get_cells -of [get_sites SLICE_X0Y[expr 36 + $i]]]
}
```

This command uses a FOR loop with an index counter (i) and a Tcl expression (36 + $i) to get and unplace any cells found in the specified range of Slices. For more information on FOR loops and other scripting suggestions, refer to the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894).

TIP: *If there are no cells placed within the specified slices, you will see warning messages that nothing has been unplaced. You can safely ignore these messages.*

With the Slices cleared of any current logic cells, the needed resources are available for placing the output registers. After placing those, you will also need to replace any logic that was unplaced in the last step.

5. Place the output registers, `wbOutputData_reg[31:0]`, in the specified Slice range with the following command:

```
for {set i 0} {$i<32} {incr i}  {
   place_cell wbOutputData_reg[$i] SLICE_X0Y[expr 36 + $i]/AFF
}
```

6. Now, place any remaining unplaced cells with the following command:

```
place_design
```

Note: *The Vivado placer works incrementally on a partially placed design.*

7. As a precaution, unroute any nets connected to the output register cells, `wbOutputData_reg[31:0]`, using the following Tcl command:

```
route_design -unroute -nets [get_nets -of [get_cells wbOutputData_reg[*]]]
```

8. Then route any currently unrouted nets in the design:

```
route_design
```

Note: *The Vivado router works incrementally on a partially routed design.*

9. Analyze the route status of the current design to ensure that there are no routing conflicts:

```
report_route_status
```

10. Click the **Routing Resources** button to view the detailed routing resources in the Device window.

11. Mark the output ports and registers again, and re-highlight the routing between them using the following Tcl commands:

```
mark_objects -color blue [get_ports wbOutputData[*]]
mark_objects -color red [get_cells wbOutputData_reg[*]]
highlight_objects -color yellow [get_nets -of [get_pins -of [get_cells\
wbOutputData_reg[*]] -filter DIRECTION==OUT]]
```

TIP: *Because you have entered these commands before, you can copy them from the journal file (*`vivado.jou`*) to avoid typing them again.*

12. In the Device window, zoom into some of the marked output ports.
13. Select the nets connecting to them.

TIP: *You can also select the nets in the Netlist window, and they will be cross-selected in the Device window.*

In the Device window, as seen in Figure 54, you can see that all output registers are now placed equidistant from their associated outputs, and the routing path is very similar for all the nets from output register to output. This results in clock-to-out times that are closely matched between the outputs.

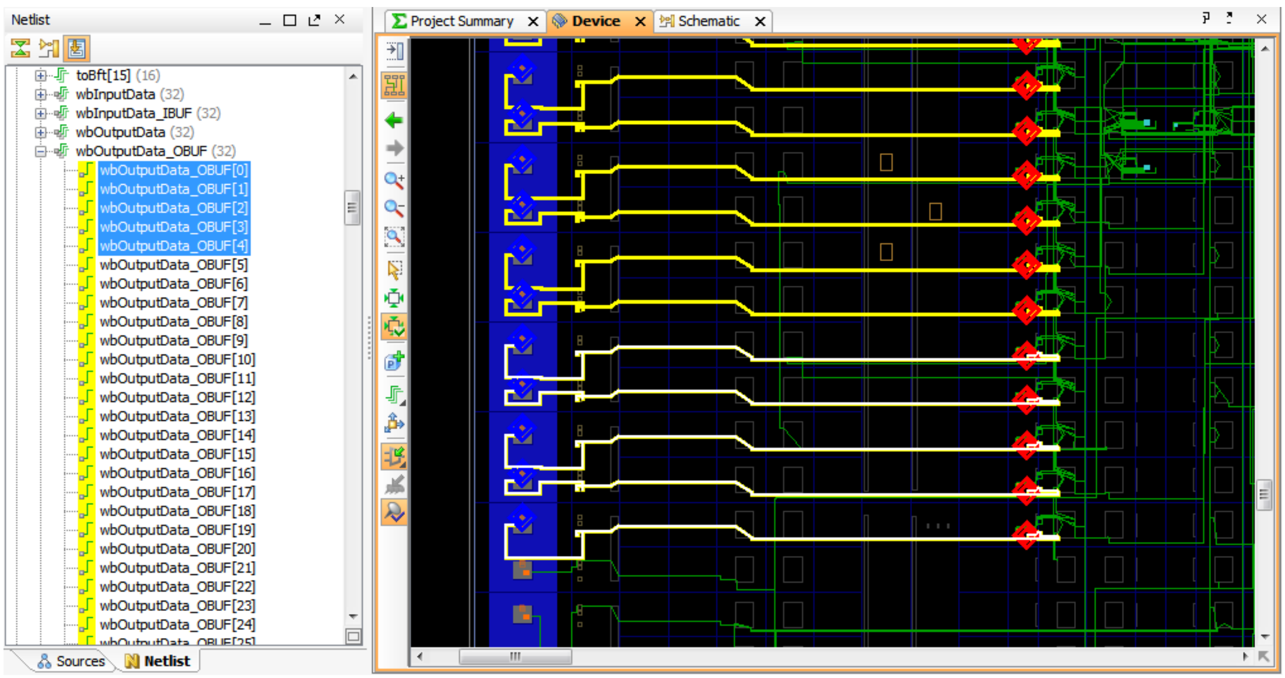


Figure 54: Improved Placement and Routing

14. Run the Tools > Timing > Report Datasheet command again.

The Report Datasheet dialog box is populated with settings from the last time you ran it:

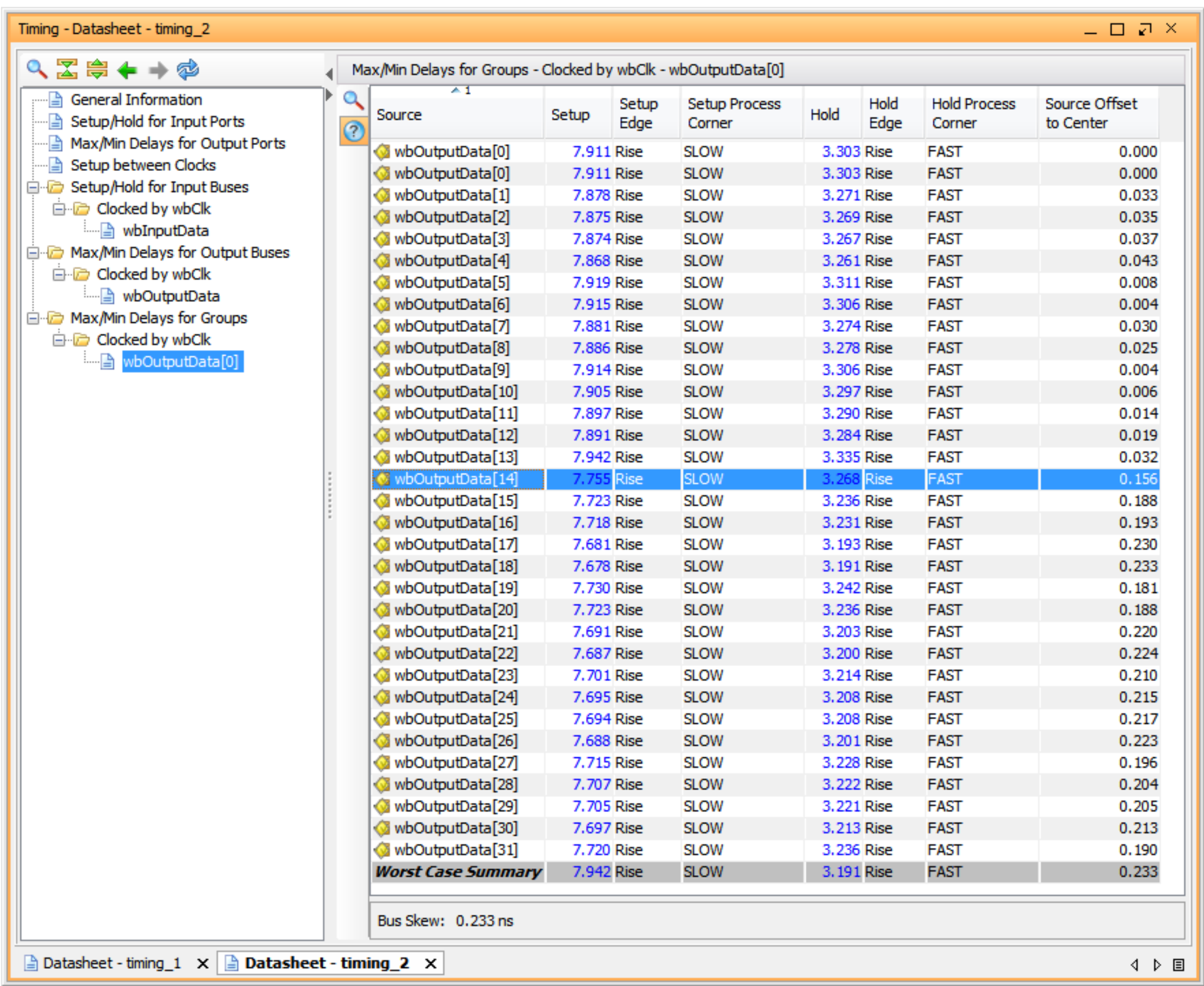
- Reference: `[get_ports {wbOutputData[0]}]`
- Ports: `[get_ports {wbOutputData[*]}]`

15. In the Report Datasheet results, select the Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0] section.

    Examining the results shown in Figure 55, the timing skew is closely matched within both the lower bits, `wbOutputData[0-13]`, and the upper bits, `wbOutputData[14-31]`, of the output bus. While the overall skew is reduced, it is still over 200 ps between the upper and lower bits.

    With the improved placement, the skew is now a result of the output ports and registers spanning two clock regions, X0Y0 and X0Y1, which introduces clock network skew. Looking at the `wbOutputData` bus, notice that the Setup delay is greater on the lower bits than it is on the upper bits. To reduce the skew, add delay to the upper bits.

    You can eliminate some of the skew using a BUFMR/BUFR combination instead of a BUFG, to clock the output registers. However, for this tutorial, you will use manual routing to add delay from the output registers clocked by the BUFG to the output pins for the upper bits, `wbOutputData[14-31]`, to further reduce the clock-to-out variability within the bus.



Source	Setup	Setup Edge	Setup Process Corner	Hold	Hold Edge	Hold Process Corner	Source Offset to Center
wbOutputData[0]	7.911	Rise	SLOW	3.303	Rise	FAST	0.000
wbOutputData[0]	7.911	Rise	SLOW	3.303	Rise	FAST	0.000
wbOutputData[1]	7.878	Rise	SLOW	3.271	Rise	FAST	0.033
wbOutputData[2]	7.875	Rise	SLOW	3.269	Rise	FAST	0.035
wbOutputData[3]	7.874	Rise	SLOW	3.267	Rise	FAST	0.037
wbOutputData[4]	7.868	Rise	SLOW	3.261	Rise	FAST	0.043
wbOutputData[5]	7.919	Rise	SLOW	3.311	Rise	FAST	0.008
wbOutputData[6]	7.915	Rise	SLOW	3.306	Rise	FAST	0.004
wbOutputData[7]	7.881	Rise	SLOW	3.274	Rise	FAST	0.030
wbOutputData[8]	7.886	Rise	SLOW	3.278	Rise	FAST	0.025
wbOutputData[9]	7.914	Rise	SLOW	3.306	Rise	FAST	0.004
wbOutputData[10]	7.905	Rise	SLOW	3.297	Rise	FAST	0.006
wbOutputData[11]	7.897	Rise	SLOW	3.290	Rise	FAST	0.014
wbOutputData[12]	7.891	Rise	SLOW	3.284	Rise	FAST	0.019
wbOutputData[13]	7.942	Rise	SLOW	3.335	Rise	FAST	0.032
wbOutputData[14]	7.755	Rise	SLOW	3.268	Rise	FAST	0.156
wbOutputData[15]	7.723	Rise	SLOW	3.236	Rise	FAST	0.188
wbOutputData[16]	7.718	Rise	SLOW	3.231	Rise	FAST	0.193
wbOutputData[17]	7.681	Rise	SLOW	3.193	Rise	FAST	0.230
wbOutputData[18]	7.678	Rise	SLOW	3.191	Rise	FAST	0.233
wbOutputData[19]	7.730	Rise	SLOW	3.242	Rise	FAST	0.181
wbOutputData[20]	7.723	Rise	SLOW	3.236	Rise	FAST	0.188
wbOutputData[21]	7.691	Rise	SLOW	3.203	Rise	FAST	0.220
wbOutputData[22]	7.687	Rise	SLOW	3.200	Rise	FAST	0.224
wbOutputData[23]	7.701	Rise	SLOW	3.214	Rise	FAST	0.210
wbOutputData[24]	7.695	Rise	SLOW	3.208	Rise	FAST	0.215
wbOutputData[25]	7.694	Rise	SLOW	3.208	Rise	FAST	0.217
wbOutputData[26]	7.688	Rise	SLOW	3.201	Rise	FAST	0.223
wbOutputData[27]	7.715	Rise	SLOW	3.228	Rise	FAST	0.196
wbOutputData[28]	7.707	Rise	SLOW	3.222	Rise	FAST	0.204
wbOutputData[29]	7.705	Rise	SLOW	3.221	Rise	FAST	0.205
wbOutputData[30]	7.697	Rise	SLOW	3.213	Rise	FAST	0.213
wbOutputData[31]	7.720	Rise	SLOW	3.236	Rise	FAST	0.190
Worst Case Summary	7.942	Rise	SLOW	3.191	Rise	FAST	0.233

Figure 55: Report Datasheet – Improved Skew

Step 5: Using Manual Routing to Reduce Clock Skew

To adjust the skew, begin by examining the current routing of the nets, `wbOutputData_OBUF[14:31]`, to see where changes might be made to consistently add delay. You can use a Tcl FOR loop to report the existing routing on those nets, to let you examine them more closely.

1. In the Tcl Console, type the following command:

```
for {set i 14} {$i<32} {incr i} {
   puts "$i [get_property ROUTE [get_nets -of [get_pins -of \
   [get_cells wbOutputData_reg[$i]] -filter DIRECTION==OUT]]]"
}
```

   This For loop initializes the index to 14 (`set i 14`), and gets the `ROUTE` property to return the details of the route on each selected net.

   The Tcl console returns the net index followed by relative route information for each net:

```
14  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
15  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
16  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
17  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
18  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
19  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
20  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
21  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
22  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
23  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
24  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
25  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
26  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
27  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
28  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
29  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
30  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }
31  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 WW2BEG0 IMUX_L34 IOI_OLOGIC1_D1 LIOI_OLOGIC1_OQ LIOI_O1 }
```

   From the returned `ROUTE` properties, note that the nets are routed from the output registers using identical resources, up to node `IMUX_L34`. Beyond that, the Vivado router uses different nodes for odd and even index nets to complete the connection to the die pad.

   By reusing routing paths, you can manually route one net with an even index, like `wbOutputData_OBUF[14]`, and one net with an odd index, such as `wbOutputData_OBUF[15]`, and copy the routing to all other even and odd index nets in the group.

2. In the Tcl Console, select the first net with the following command:

```
select_objects [get_nets -of [get_pins -of \
[get_cells wbOutputData_reg[14]] -filter DIRECTION==OUT]]
```

3. In the Device window, right-click to open the popup menu and select **Unroute**.

4. Click **Yes** in the Confirm Unroute dialog box.

   The Device view displays the unrouted net as a fly-line between the register and the output pad.

5. Click the **Maximize** button to maximize the Device window.

6. Right-click the net and select **Enter Assign Routing Mode**.

   The Target Load Cell Pin dialog box opens, as seen in Figure 56, to let you select a load pin to route to or from. In this case, only one load pin populates: `wbOutputData_OBUF[14]_inst`.

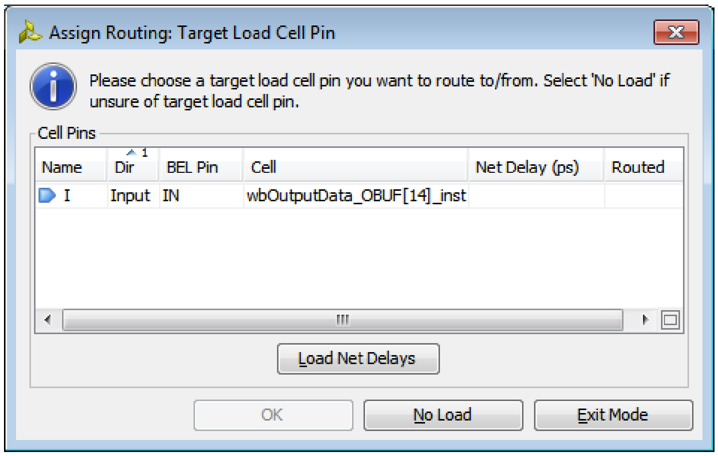


Figure 56: Target Load Cell Pin Dialog Box

7. Select the **load cell pin**, and click **OK**.

   The Vivado IDE enters into Assign Routing mode, displaying a new Routing Assignment window on the right side of the Device window, as shown in the following figure.

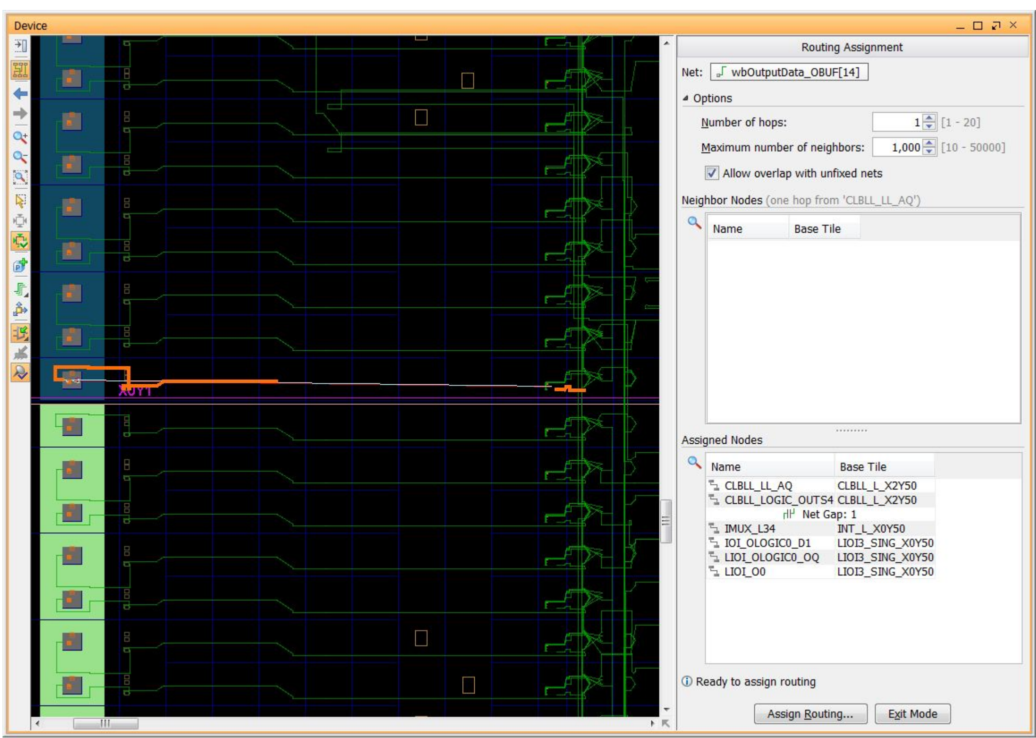


Figure 57: Assign Routing Mode

The Routing Assignment window includes the following sections, as seen in Figure 57:

- **Net**: Displays the current net being routed.
- **Options**: Are hidden by default, and can be displayed by clicking **Options**.
  - **Number of Hops**: Defines how many programmable interconnect points, or PIPs, to look at when reporting the available neighbors. The default is 1.
  - **Number of Neighbors**: Limits the number of neighbors displayed for selection.
  - **Allow Overlap with Unfixed Nets**: Enables or disables a loose style of routing which can create conflicts that must be later resolved. The default is ON.
- **Neighbor Nodes**: Lists the available neighbor PIPs/nodes to choose from when defining the path of the route.
- **Assigned Nodes**: Shows the currently assigned nodes in the route path of the selected net.
- **Assign Routing**: Assigns the currently defined path in the Routing Assignment window as the route path for the selected net.
- **Exit Mode**: Closes the Routing Assignment window.

As you can see in Figure 57, the Assigned Nodes section displays six currently assigned nodes. The Vivado router automatically assigns a node if it is the only neighbor of a selected node and there are no alternatives to the assigned nodes for the route. In the Device window, the assigned nodes appear as a partial route in orange.

In the currently selected net, `wbOutputData_OBUF[14]`, nodes `CLBLL_LL_AQ` and `CLBLL_LOGIC_OUTS4` are already assigned because they are the only neighbor nodes available to the output register, `wbOutputData_reg[14]`. The nodes `IMUX_L34`, `IOI_OLOGIC0_D1`, `LIOI_OLOGIC0_OQ`, and `LIOI_O0` are also already assigned because they are the only neighbor nodes available to the destination, the output buffer (OBUF).

A gap exists between the two routed portions of the path where there are multiple neighbors to choose from when defining a route. This gap is where you will use manual routing to complete the path and add the needed delay to balance the clock skew.

You can route the gap by selecting a node on either side of the gap and then choosing the neighbor node to assign the route to. Selecting the node displays possible neighbor nodes in the Neighbor Nodes section of the Routing Assignment window and appear as dashed white lines in the Device window.

TIP: *The number of reachable neighbor nodes displayed depends on the number of hops defined in the Options.*

---

8. Under the **Assigned Nodes** section, select the `CLBLL_LOGIC_OUTS4` node before the gap.

   The available neighbors appear as shown in the following figure.

   To add delay to compensate for the clock skew, select a neighbor node that provides a slight detour over the more direct route previously chosen by the router.

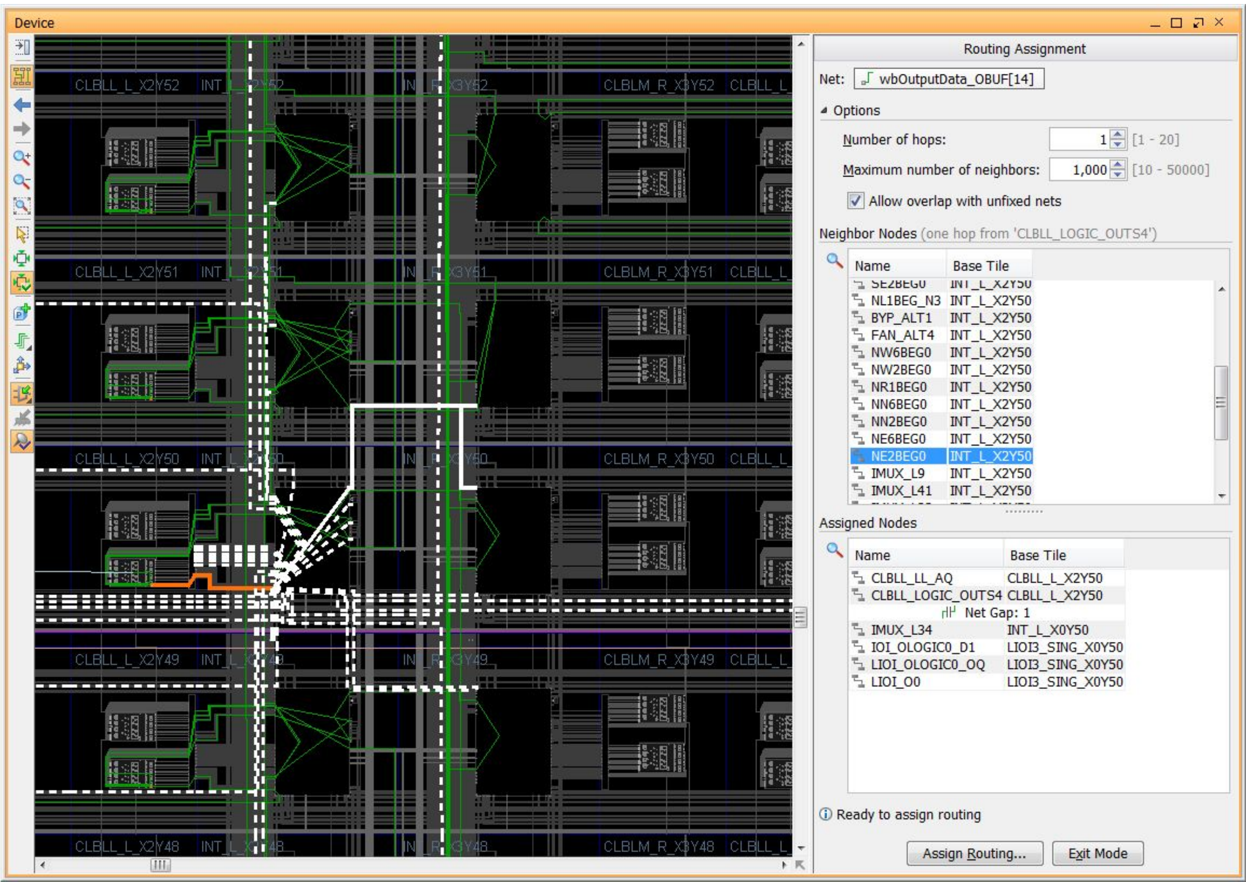


Figure 58: Routing the Gap from CLBLL_LOGIC_OUTS4

9. Under **Neighbor Nodes**, select node `NE2BEG0`.

   This node provides a routing detour to add delay, as compared to some other nodes such as `WW2BEG0`, which provide a more direct route toward the output buffer. Clicking a neighbor node once selects it so you can explore routing alternatives. Double-clicking the node temporarily assigns it to the net, so that you can then select the next neighbor from that node.

10. In **Neighbor Nodes**, assign node `NE2BEG0` by double-clicking it.

    This adds the node to the Assigned Nodes section of the Routing Assignment window, which updates the Neighbor Nodes.

11. In **Neighbor Nodes**, select and assign nodes `WR1BEG1`, and then `WR1BEG2`.

TIP:

In case you assigned the wrong node, you can select the node from the Assigned Nodes list, right click, and select **Remove** on the context menu.

You can turn off the Auto Fit Selection in the Device view if you would like to stay at the same zoom level.

The following figure shows the partially routed path using the selected nodes shown in orange. You can use the automatic routing feature to fill the remaining gap.

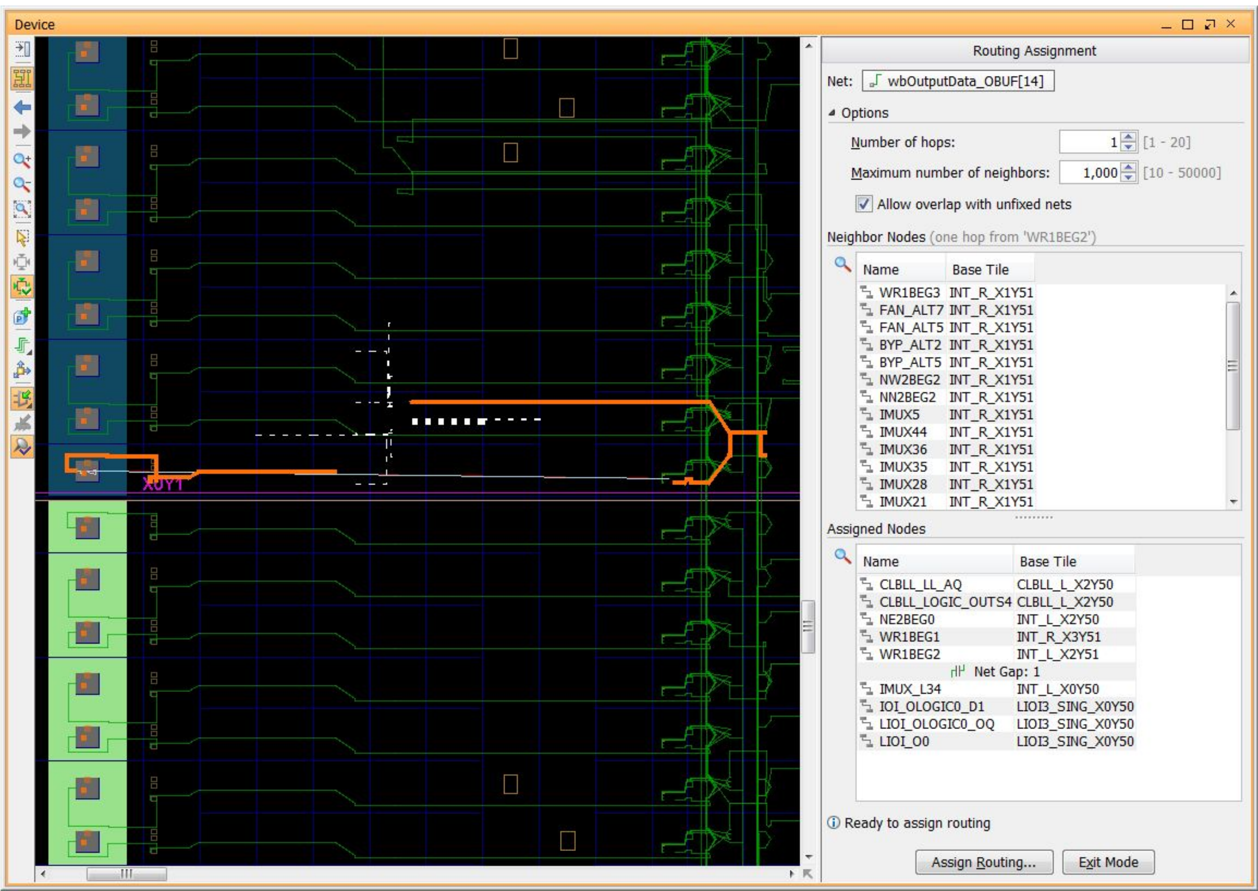


Figure 59: Closing the Gap

12. Under the **Assigned Nodes** section of the Routing Assignment window, right-click the **Net Gap**, and select **Auto-Route**, as shown in the following figure.

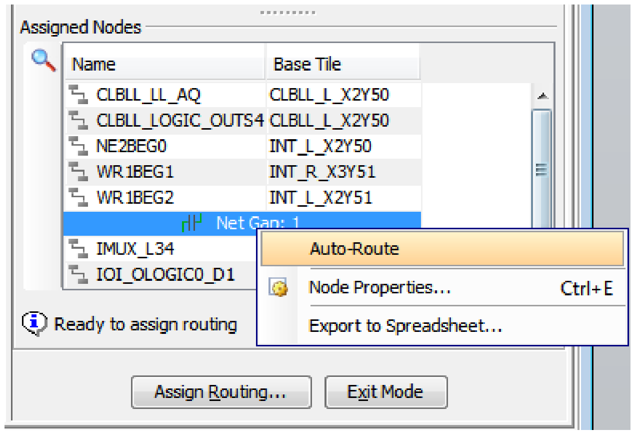


Figure 60: Auto-Route the Gap

The Vivado router fills in the last small bit of the gap. With the route path fully defined, you can assign the routing to commit the changes to the design.

13. Click **Assign Routing** at the bottom of the Routing Assignment window.

The Assign Routing dialog box opens, as seen in the following figure. This displays the list of currently assigned nodes that define the route path. You can select any of the listed nodes, highlighting it in the Device window. This lets you quickly review the route path prior to committing it to the design.

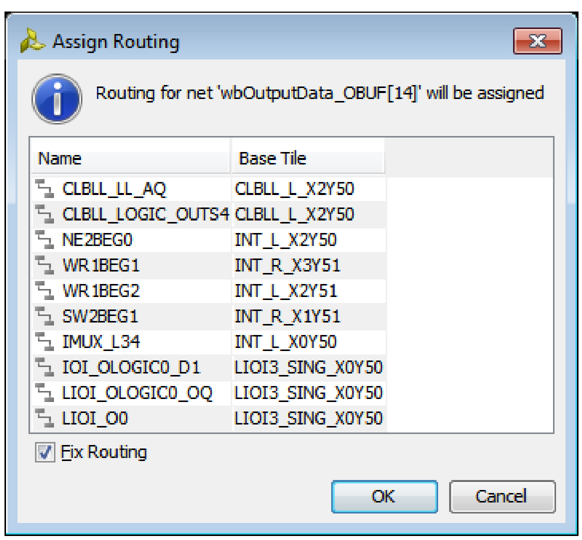


Figure 61: Assign Routing – Even Nets

14. Make sure **Fix Routing** is checked, and click **OK**.

    The Fix Routing checkbox marks the defined route as fixed to prevent the Vivado router from ripping it up or modifying it during subsequent routing steps. This is important in this case, because you are routing the net manually to add delay to match clock skew.

15. Examine the Tcl commands in the Tcl Console.

    The Tcl console reports any Tcl commands that assigned the routing for the current net. Those commands are:

```
set_property is_bel_fixed 1 [get_cells {wbOutputData_reg[14]
wbOutputData_OBUF[14]_inst }]
set_property is_loc_fixed 1 [get_cells {wbOutputData_reg[14]
wbOutputData_OBUF[14]_inst }]
set_property fixed_route {  { CLBLL_LL_AQ CLBLL_LOGIC_OUTS4 NE2BEG0 WR1BEG1
WR1BEG2 SW2BEG1 IMUX_L34 IOI_OLOGIC0_D1 LIOI_OLOGIC0_OQ LIOI_O0 }  } [get_nets
{wbOutputData_OBUF[14]}]
```

IMPORTANT: *The FIXED_ROUTE property assigned to the net, wbOutputData_OBUF[14], uses a directed routing string with a relative format, based on the placement of the net driver. This lets you reuse defined routing by copying the FIXED_ROUTE property onto other nets that use the same relative route.*

After defining the manual route for the even index nets, the next step is to define the route path for the odd index net, `wbOutputData_OBUF[15]`, applying the same steps you just completed.

16. In the Tcl Console type the following to select the net:

```
select_objects [get_nets wbOutputData_OBUF[15]]
```

17. With the net selected:

    a. Unroute the net.

    b. Enter Routing Assignment mode.

    c. Select the Load Cell Pin.

    d. Route the net using the specified neighbor nodes (`NE2BEG0`, `WR1BEG1`, and `WR1BEG2`).

    e. Auto-Route the gap.

    f. Assign the routing.

The Assign Routing dialog box, shown in the following figure, shows the nodes selected to complete the route path for the odd index nets.

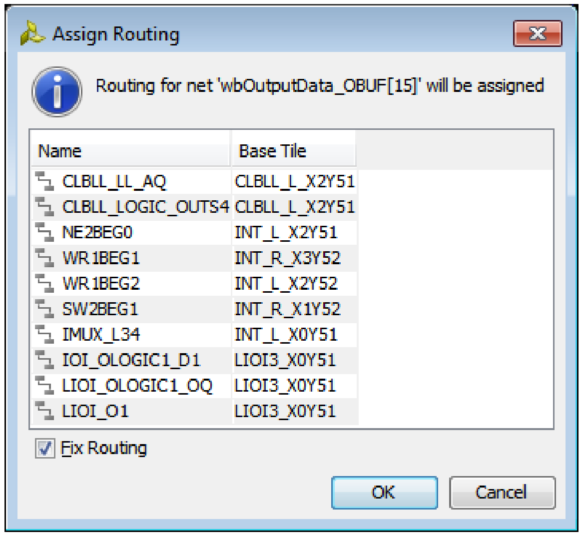
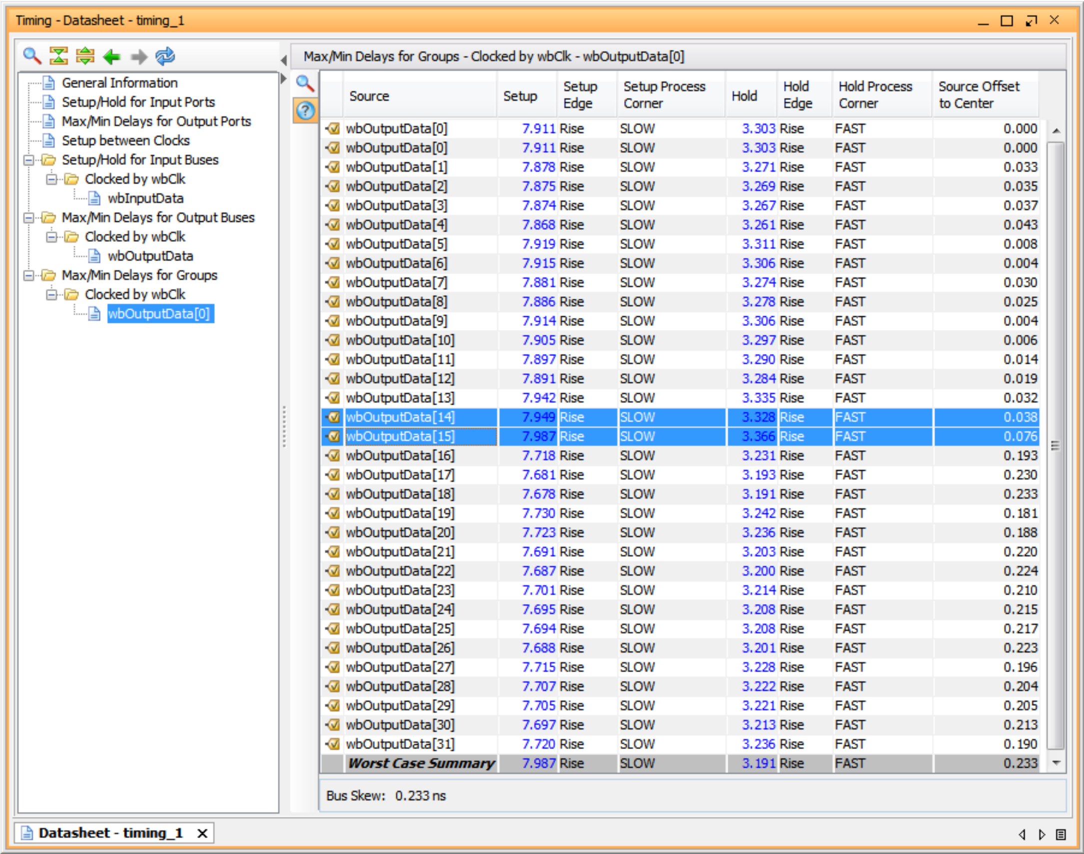


Figure 62: Assign Routing – Odd Nets

You routed the `wbOutputData_OBUF[14]` and `wbOutputData_OBUF[15]` nets with the detour to add the needed delay. You can now run the **Report Datasheet** command again to examine the timing for these nets with respect to the lower order bits of the bus.

18. Switch to the Timing Datasheet report window. Notice the information message in the banner of the window indicating that the report is out of date because the design was modified.
19. In the Timing Datasheet report, click **Rerun** to update the report with the latest timing information.
20. Select **Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0]** to display the timing info for the `wbOutputData` bus, as seen in Figure 63.



Source	Setup	Setup Edge	Setup Process Corner	Hold	Hold Edge	Hold Process Corner	Source Offset to Center
wbOutputData[0]	7.911	Rise	SLOW	3.303	Rise	FAST	0.000
wbOutputData[0]	7.911	Rise	SLOW	3.303	Rise	FAST	0.000
wbOutputData[1]	7.878	Rise	SLOW	3.271	Rise	FAST	0.033
wbOutputData[2]	7.875	Rise	SLOW	3.269	Rise	FAST	0.035
wbOutputData[3]	7.874	Rise	SLOW	3.267	Rise	FAST	0.037
wbOutputData[4]	7.868	Rise	SLOW	3.261	Rise	FAST	0.043
wbOutputData[5]	7.919	Rise	SLOW	3.311	Rise	FAST	0.008
wbOutputData[6]	7.915	Rise	SLOW	3.306	Rise	FAST	0.004
wbOutputData[7]	7.881	Rise	SLOW	3.274	Rise	FAST	0.030
wbOutputData[8]	7.886	Rise	SLOW	3.278	Rise	FAST	0.025
wbOutputData[9]	7.914	Rise	SLOW	3.306	Rise	FAST	0.004
wbOutputData[10]	7.905	Rise	SLOW	3.297	Rise	FAST	0.006
wbOutputData[11]	7.897	Rise	SLOW	3.290	Rise	FAST	0.014
wbOutputData[12]	7.891	Rise	SLOW	3.284	Rise	FAST	0.019
wbOutputData[13]	7.942	Rise	SLOW	3.335	Rise	FAST	0.032
wbOutputData[14]	7.949	Rise	SLOW	3.328	Rise	FAST	0.038
wbOutputData[15]	7.987	Rise	SLOW	3.366	Rise	FAST	0.076
wbOutputData[16]	7.718	Rise	SLOW	3.231	Rise	FAST	0.193
wbOutputData[17]	7.681	Rise	SLOW	3.193	Rise	FAST	0.230
wbOutputData[18]	7.678	Rise	SLOW	3.191	Rise	FAST	0.233
wbOutputData[19]	7.730	Rise	SLOW	3.242	Rise	FAST	0.181
wbOutputData[20]	7.723	Rise	SLOW	3.236	Rise	FAST	0.188
wbOutputData[21]	7.691	Rise	SLOW	3.203	Rise	FAST	0.220
wbOutputData[22]	7.687	Rise	SLOW	3.200	Rise	FAST	0.224
wbOutputData[23]	7.701	Rise	SLOW	3.214	Rise	FAST	0.210
wbOutputData[24]	7.695	Rise	SLOW	3.208	Rise	FAST	0.215
wbOutputData[25]	7.694	Rise	SLOW	3.208	Rise	FAST	0.217
wbOutputData[26]	7.688	Rise	SLOW	3.201	Rise	FAST	0.223
wbOutputData[27]	7.715	Rise	SLOW	3.228	Rise	FAST	0.196
wbOutputData[28]	7.707	Rise	SLOW	3.222	Rise	FAST	0.204
wbOutputData[29]	7.705	Rise	SLOW	3.221	Rise	FAST	0.205
wbOutputData[30]	7.697	Rise	SLOW	3.213	Rise	FAST	0.213
wbOutputData[31]	7.720	Rise	SLOW	3.236	Rise	FAST	0.190
Worst Case Summary	7.987	Rise	SLOW	3.191	Rise	FAST	0.233

Figure 63: Report Datasheet – Improved Routing

You can see from the report that the skew within the rerouted nets, `wbOutputData[14]` and `wbOutputData[15]`, more closely matches the timing of the lower bits of the output bus, `wbOutputData[13:0]`. The skew is within the target of 100 ps of the reference pin `wbOutputData[0]`.

In Step 6, you copy the same route path to the remaining nets, `wbOutputData_OBUF[31:16]`, to tighten the timing of the whole `wbOutputData` bus.

Step 6: Copying Routing to Other Nets

To apply the same fixed route used for net `wbOutputData_OBUF[14]` to the even index nets, and the fixed route for `wbOutputData_OBUF[15]` to the odd index nets, you can use Tcl For loops as described in the following steps.

1. Select the Tcl Console tab.
2. Set a Tcl variable to store the route path for the even nets and the odd nets:

```
set even [get_property FIXED_ROUTE [get_nets wbOutputData_OBUF[14]]]
set odd [get_property FIXED_ROUTE [get_nets wbOutputData_OBUF[15]]]
```

3. Set a Tcl variable to store the list of nets to be routed, containing all high bit nets of the output data bus, `wbOutputData_OBUF[16:31]`:

```
for {set i 16} {$i<32} {incr i}  {
   lappend routeNets [get_nets wbOutputData_OBUF[$i]]
}
```

4. Unroute the specified nets:

```
route_design -unroute -nets $routeNets
```

5. Apply the `FIXED_ROUTE` property of net `wbOutputData_OBUF[14]` to the even nets:

```
for {set i 16} {$i<32} {incr i 2}  {
   set_property FIXED_ROUTE $even [get_nets wbOutputData_OBUF[$i]]
}
```

6. Apply the `FIXED_ROUTE` property of net `wbOutputData_OBUF[15]` to the odd nets:

```
for {set i 17} {$i<32} {incr i 2}  {
   set_property FIXED_ROUTE $odd [get_nets wbOutputData_OBUF[$i]]
}
```

   The even and odd nets of the output data bus, as needed, now have the same routing paths, adding delay to the high order bits. Run the route status report and the datasheet report to validate that the design is as expected.

7. In the Tcl Console, type the following command:

```
report_route_status
```

TIP: *Some routing errors might be reported if the routed design included nets that use some of the nodes you have assigned to the FIXED_ROUTE properties of the manually routed nets. Remember you enabled* ***Allow Overlap with Unfixed Nets*** *in the Routing Assignment window.*

8. If any routing errors are reported, type the `route_design` command from the Tcl console.

   The nets with the `FIXED_ROUTE` property takes precedence over the auto-routed nets.

9. After `route_design`, repeat the `report_route_status` command to see the clean report.
10. Examine the output data bus in the Device view, as seen in the following figure:
    - All nets from the output registers to the output pins for the upper bits 14-31 of the output bus `wbOutputData` have identical fixed routing sections (shown as dashed lines).
    - You do not need to fix the `LOC` and the `BEL` for the output registers. It was done by the `place_cell` command in an earlier step.

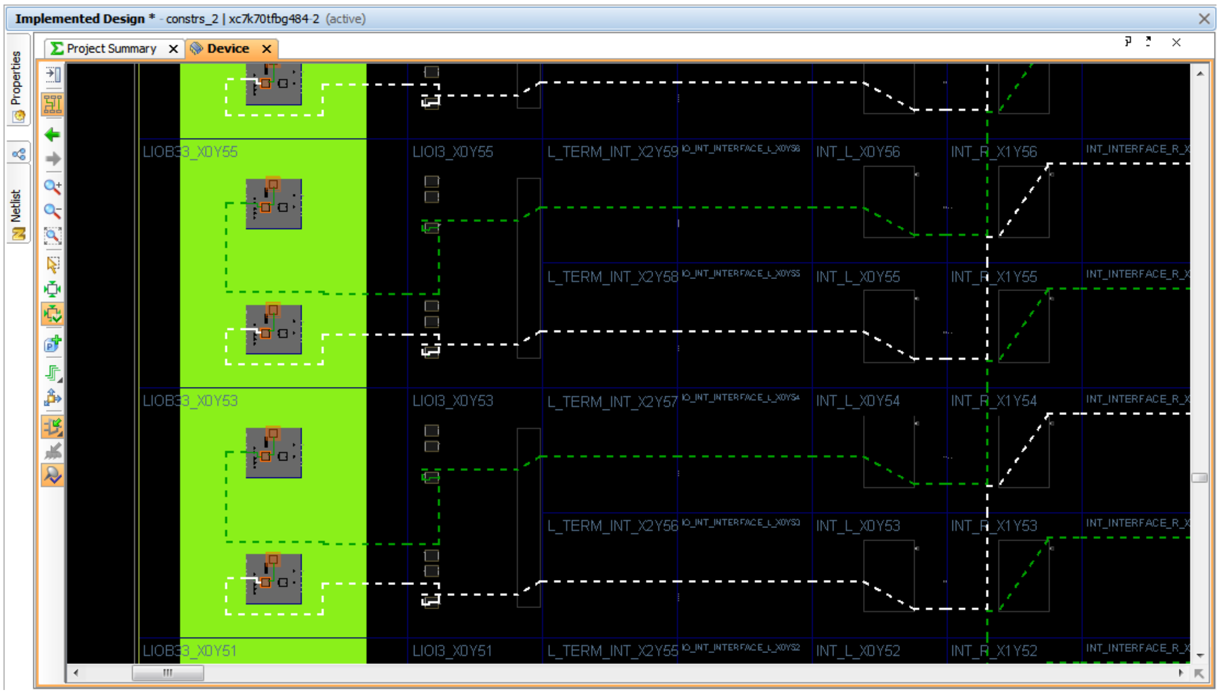


Figure 64: Final Routed Design

Having routed all the upper bit nets, `wbOutputData_OBUF[31:14]`, with the detour needed for added delay, you can now re-examine the timing of output bus.

11. Select the Timing tab in the Results window area.

    Notice the information message in the banner of the window indicating that the report is out of date because timing data has been modified.

12. Click **rerun** to update the report with the latest timing information.
13. Select the **Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0]** section to display the timing info for the `wbOutputData` bus.

    As shown in Figure 65, the clock-to-out timing within all bits of output bus `wbOutputData` is now closely matched to within 83 ps.

14. Save the constraints to write them to the target XDC, so that they apply every time you compile the design.

15. Select **File > Save Constraints** to save the placement constraints to the target constraint file, `bft_full.xdc`, in the active constraint set, `constrs_2`.

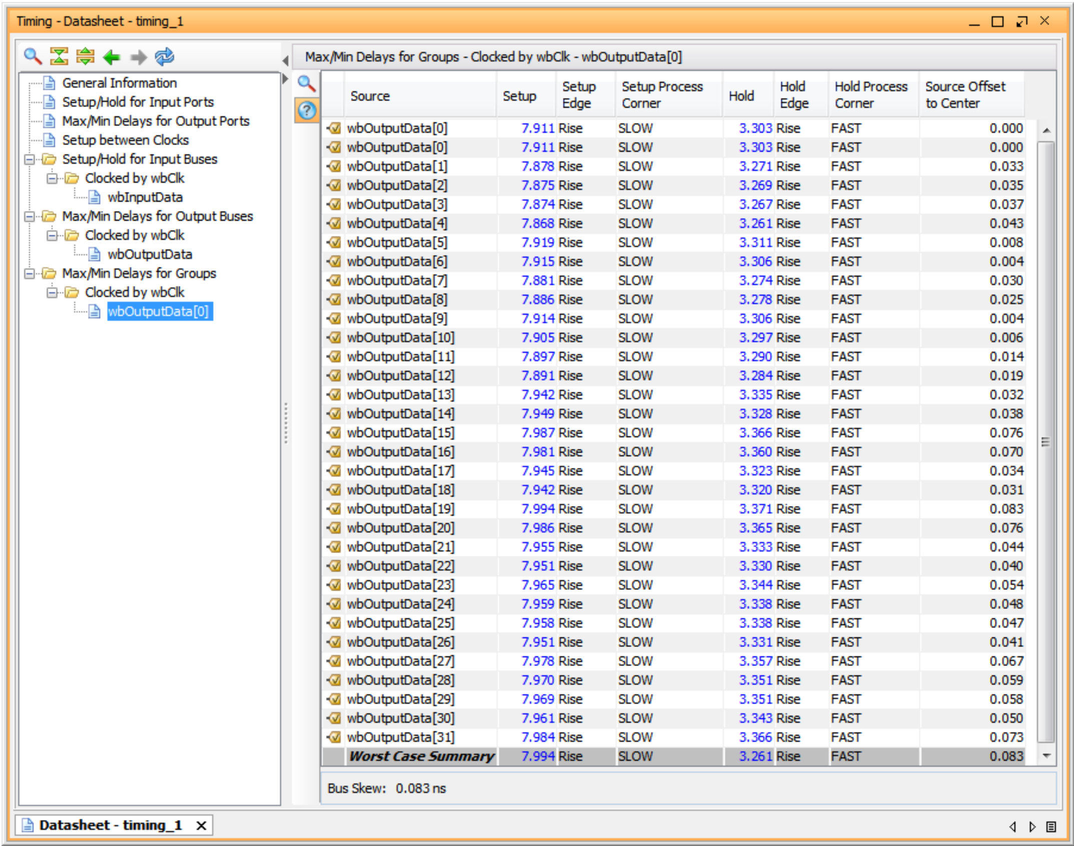


Figure 65: Report Datasheet – Final

Conclusion

In this lab, you did the following:

- Analyzed the clock skew on the output data bus using the Report Datasheet command.
- Used manual placement techniques to improve the timing of selected nets.
- Used the Assign Manual Routing Mode in the Vivado IDE to precisely control the routing of a net.
- Used the `FIXED_ROUTE` property to copy the relative fixed routing among similar nets to control the routing of the critical portion of the nets.

Introduction

In this lab, you will learn how to use the Vivado® Engineering Change Order (ECO) flow to modify your design post implementation, implement the changes, run reports on the changed netlist, and generate programming files.

For this lab, you will use the design file that is included with this guide and is targeted at the Kintex® UltraScale® KCU105 Evaluation Platform. For instructions on locating the design files, see Locating Design Files for Lab 4.

A block diagram of the design is shown in the following figure.

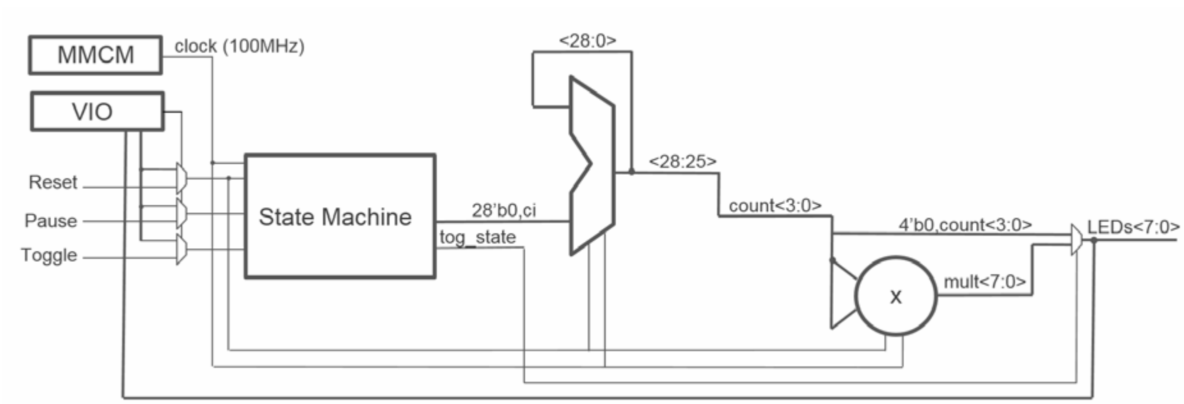


Figure 66: Block Diagram of the Design

In this design, a mixed-mode clock manager (MMCM) is used to synthesize a 100 MHz clock from the 300 MHz clock provided by the board.

A 29-bit counter is used to divide the clock down further. The 4 most significant bits of the counter form the `count<3:0>` signal that is 0-extended to 8 bits and drives the 8 on-board LEDs through an 8-bit 2-1 mux.

The `count<3:0>` signal is also squared using a multiplier, and the product drives the other 8 inputs of the mux. A `Toggle` signal controls the mux select and either drives the LEDs (shown in the following figure) with the counter value or the multiplier output.

A `Pause` signal allows you to stop the counter, and a `Reset` signal allows you to reset the design. The `Toggle`, `Pause`, and `Reset` signals can either be controlled from on-board buttons shown in Figure 67 or a VIO in the Hardware Manager as shown in Figure 68. The VIO also allows you to observe the status of the LEDs. The following figures show the location of the push-buttons and the LEDs on the KCU105 board and a Hardware Manager dashboard. These allow you to control the push button and observe the LEDs through the VIO.

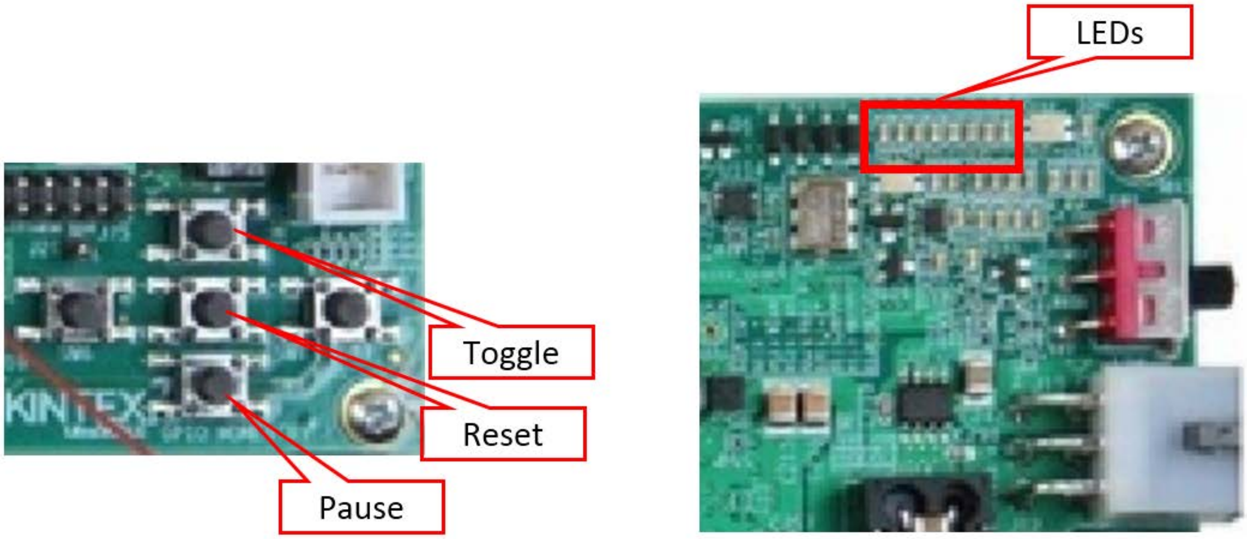


Figure 67: KCU105 On-Board Push Buttons and LEDs

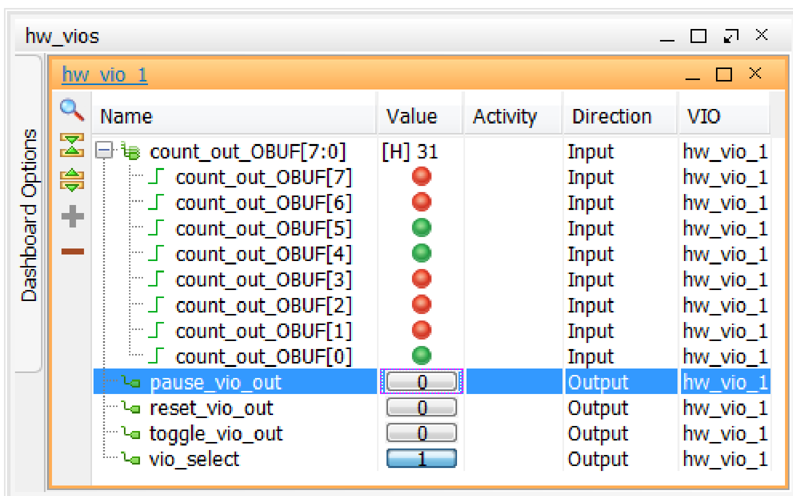


Figure 68: VIO Dashboard

Step 1: Creating a Project Using the Vivado New Project Wizard

To create a project, use the New Project wizard to name the project, to add RTL source files and constraints, and to specify the target device.

1. Open the Vivado Design Suite integrated development environment (IDE).
2. In the Getting Started page, click **Create New Project** to open the New Project wizard.
3. Click **Next**.
4. In the Project Name page, do the following:
   a. Name the new project `project_ECO_lab`.
   b. Provide the project location `C:/Vivado_Tutorial`.
   c. Ensure that Create project subdirectory is selected.
   d. Click **Next**.
5. In the Project Type page, do the following:
   a. Specify the **Type of Project** to create as **RTL Project**.
   b. Leave the **Do not specify sources at this time** check box unchecked.
   c. Click **Next**.
6. In the Add Sources page, do the following:
   a. Set the Target Language to Verilog.
   b. Click **Add Files**.
   c. In the Add Source Files dialog box, navigate to the `/src/lab4` directory.
   d. Select all Verilog source files.
   e. Click **OK**.
   f. Verify that the files are added, and that Copy Sources into Project is selected.
   g. Click **Next**.
7. In the Add Existing IP page, do the following:
   a. Click the **Add** button , and then select **Add Files**.
   b. In the Add Configurable IP dialog box, navigate to the `/src/lab4/IP` directory.
   c. Select all of the XCI source files and click **OK**.
   d. Verify that the files are added and **Copy sources into project** is selected.
   e. Click **Next**.

8. In the Add Constraints dialog box, do the following:
   a. Click the **Add** button, and then select **Add Files**.
   b. Navigate to the `/src/lab4` directory and select `ECO_kcu105.xdc`.
   c. Click **Next**.
9. In the Default Part page, do the following:
   a. Specify the **xcku040-ffva1156-2-e** part for the KC705 platform.

   ***Note**: You can also select **Boards** and then select **Kintex-UltraScale KCU105 Evaluation Platform**.*

   b. Click **Next**.
10. Review the New Project Summary page. Verify that the data appears as expected, per the steps above.
11. Click **Finish**.

    ***Note**: It might take a moment for the project to initialize.*
12. In the Sources window in the Vivado IDE, expand `top` to see the source files for this lab.

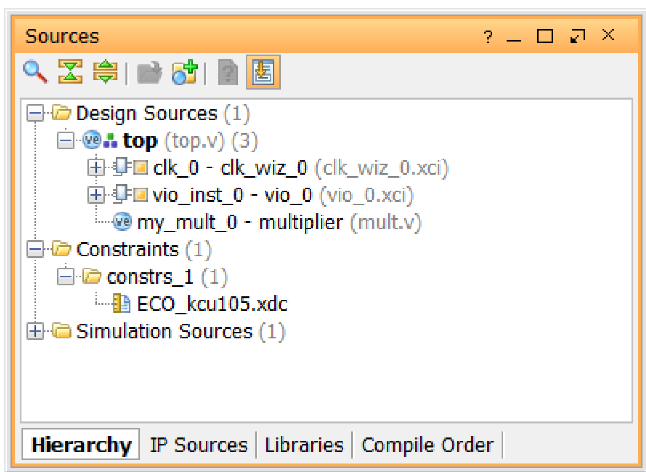


Figure 69: Sources Window

Step 2: Synthesizing, Implementing, and Generating the Bitstream

1. In the Flow Navigator, under **Program and Debug**, click **Generate Bitstream**.

   This synthesizes, implements, and generates a bitstream for the design.

   The No Implementation Results Available dialog box appears.

2. Click **Yes**.

   After bitstream generation completes, the **Bitstream Generation Completed** dialog box appears. **Open Implemented Design** is selected by default.

3. Click **OK**.

4. Inspect the Timing Summary report and make sure that all timing constraints have been met.

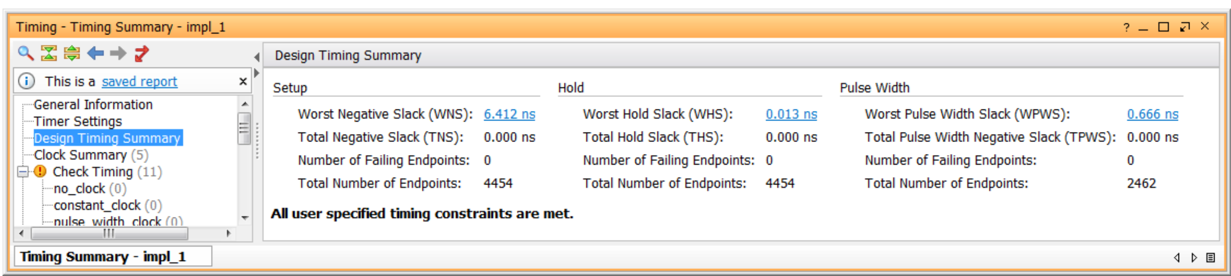


Figure 70: Timing Summary Report

You can use the generated bitstream programming file to download your design into the target FPGA device using the Hardware Manager. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Step 3: Validating the Design on the Board

This step is optional, but will help you understand the ECO modifications that you will make in Step 4: Making the ECO Modifications.

1. From the main menu, select **Flow > Open Hardware Manager**.

   The Hardware Manager window opens.

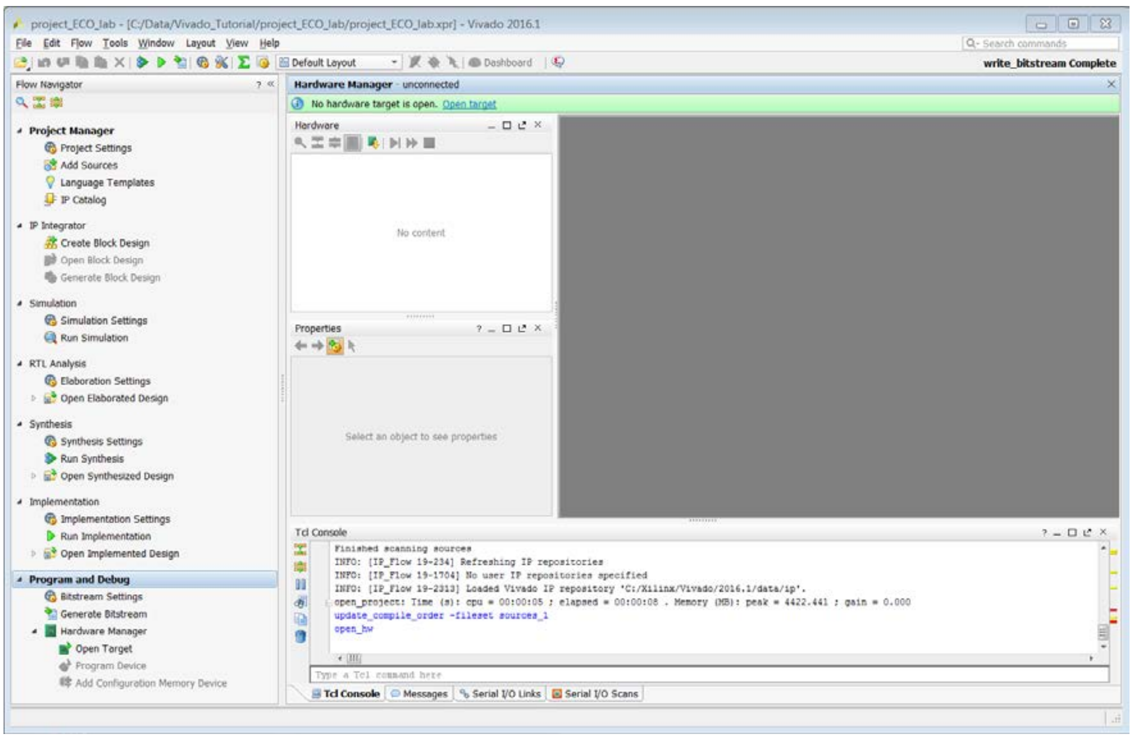


Figure 71: Hardware Manager Window

2. Connect to a hardware target Using hw_server.

TIP: *For more information about different ways to connect to a hardware target, refer to the* ***Vivado Design Suite User Guide: Programming and Debugging*** (UG908).

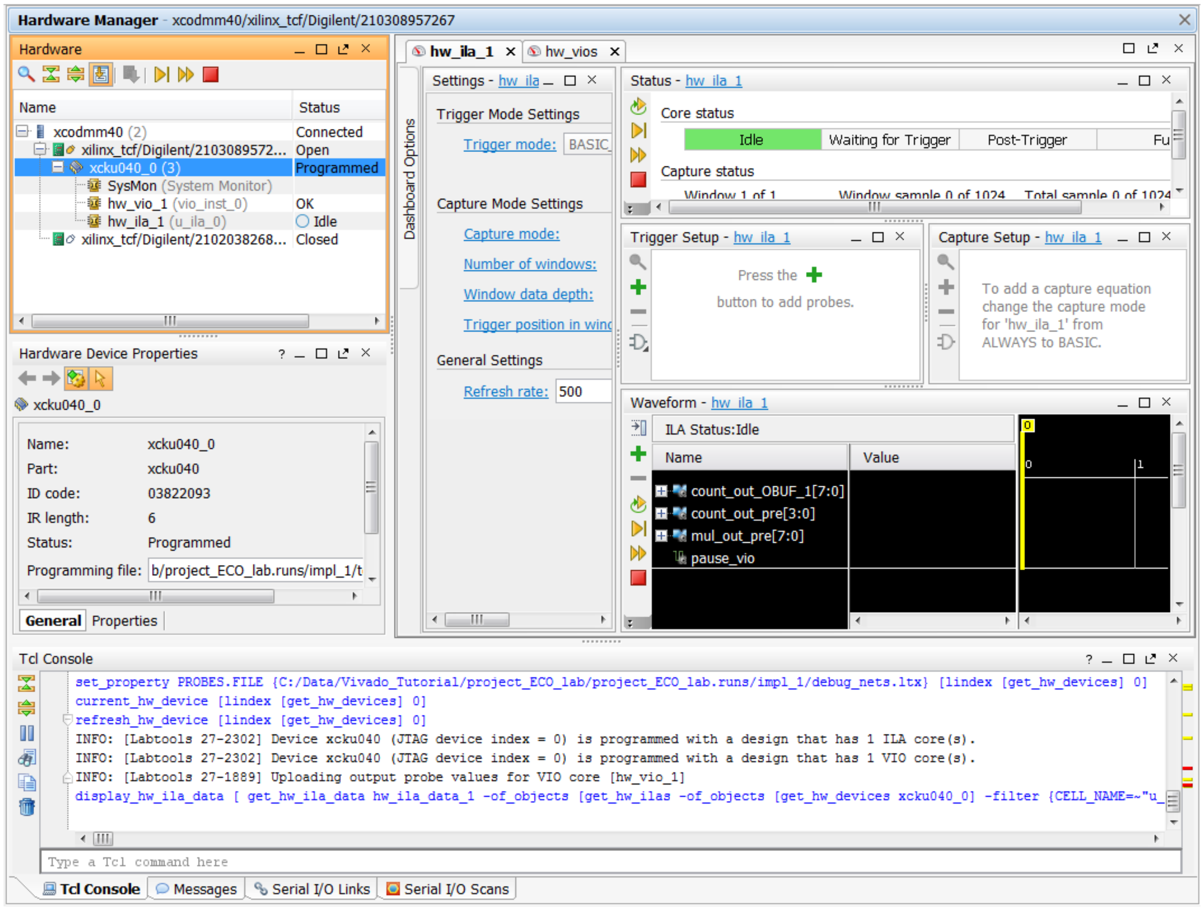


Figure 72: Use hw_server to Connect to a Hardware Target

3. In the Vivado Flow Navigator, under **Program and Debug**, click **Program Device**.

   The Program Device dialog box opens.

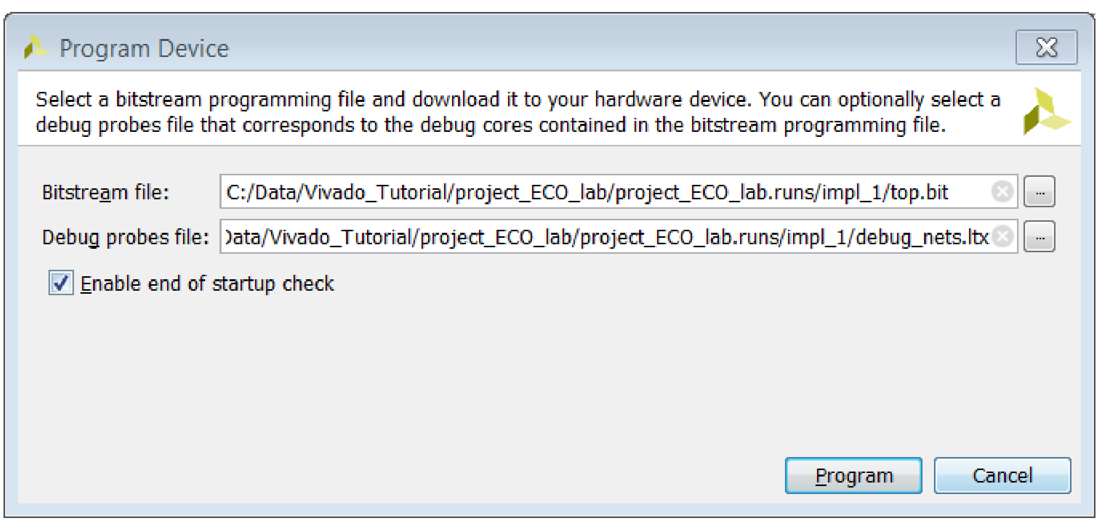


Figure 73: Program Device Dialog Box

4. Navigate to the Bitstream file and Debug Probes file.
5. Click **Program**.

   Now that the FPGA is configured, you can use the on-board buttons and the on-board LEDs to control and observe the hardware. Press the **Pause** button to pause the counter. Press the **Toggle** button to select between the count and the multiplier result. Press the **Reset** button to reset the counter.

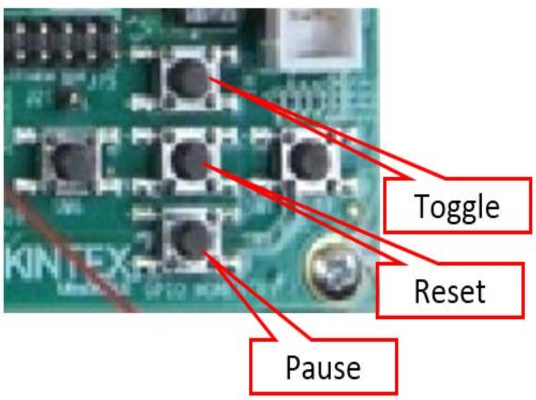


Figure 74: On-Board Push Buttons and LEDs

Alternatively, you can use the VIO to control and observe the hardware.

If the following warning message appears, select one of the alternatives suggested in the message.

```
WARNING: [Labtools 27-1952] VIO hw_probe OUTPUT_VALUE properties for hw_vio(s)
[hw_vio_1] differ from output values in the VIO core(s).
Resolution:
To synchronize the hw_probes properties and the VIO core outputs choose one of
the following alternatives:
  1) Execute the command 'Commit Output Values to VIO Core', to write down the
hw_probe values to the core.
  2) Execute the command 'Refresh Input and Output Values from VIO Core', to
update the hw_probe properties with the core values.
  3) First restore initial values in the core with the command 'Reset VIO Core
Outputs', and then execute the command 'Refresh Input and Output Values from VIO
Core'.
```

6. Select the **hw_vios** tab in the dashboard and click the **Add** button to add probes.

   The Add Probes dialog box opens.

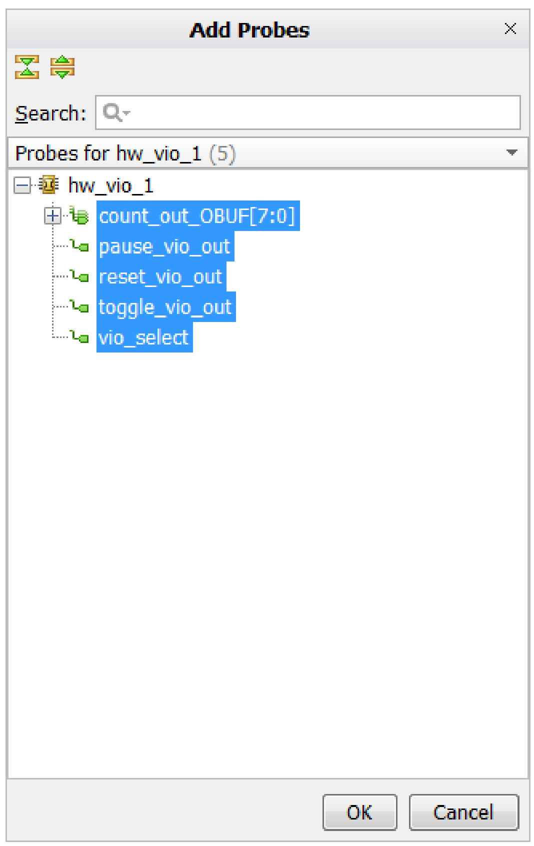


Figure 75: Add Probes Dialog Box

7. Select all of the probes for `hw_vio_1` and click **OK**.

8. In the hw_vios dashboard, select `count_out_OBUF[7:0]`, then right-click and select **Radix Unsigned Decimal**.

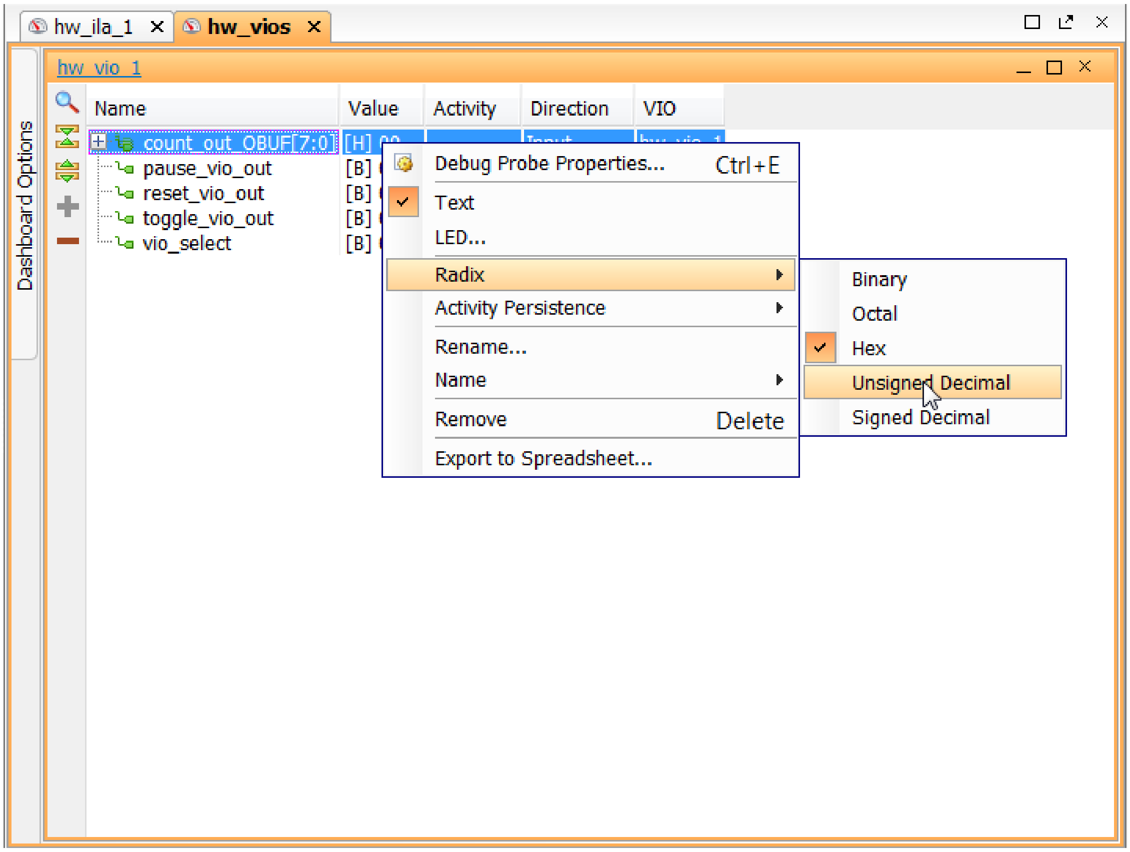


Figure 76: Selecting Radix > Unsigned Decimal

9. In the hw_vios dashboard, select `count_out_OBUF[7:0]`, then right-click and select **LED**.

   The Select LED Colors dialog box opens.

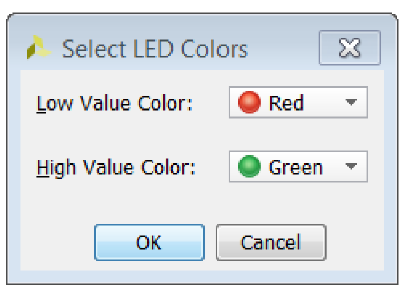


Figure 77: Select LED Colors Dialog Box

10. Select **Red** for the Low Value Color and **Green** for the High Value Color.
11. Click **OK**.

12. In the hw_vios dashboard, select pause_vio_out, reset_vio_out, and toggle_vio_out, then right-click and select **Active-High Button**.

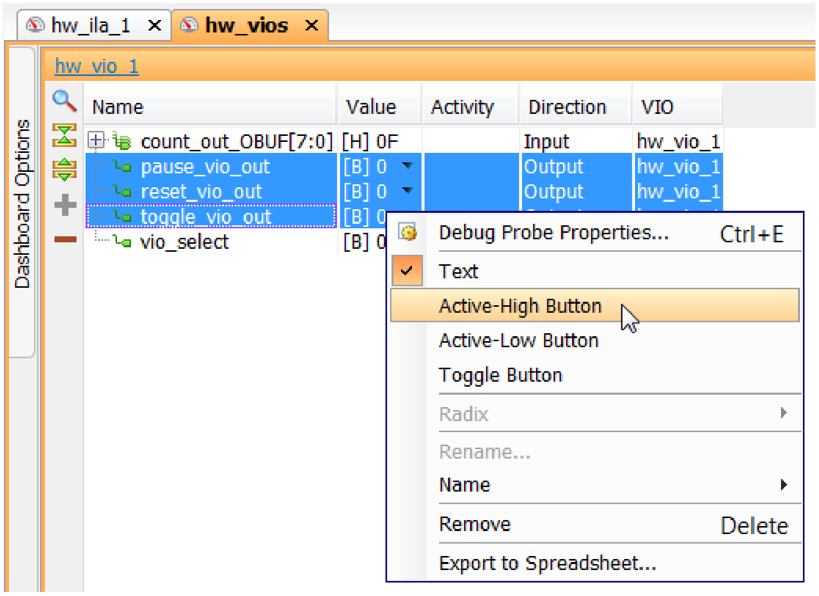


Figure 78: Selecting Active-High Button

13. In the hw_vios dashboard, select `vio_select` and select **Toggle Button**.

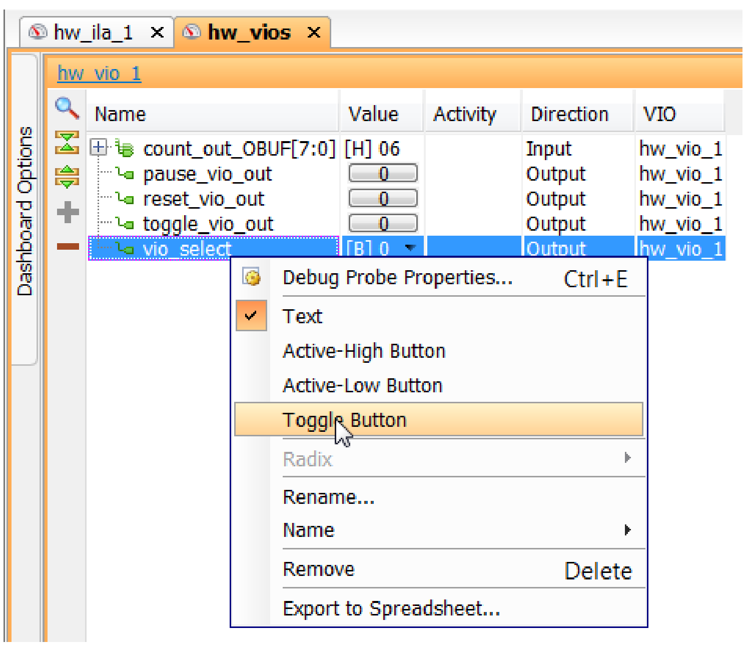


Figure 79: Selecting Toggle Button

14. Expand `count_out_OBUF[7:0]` to view the VIO LEDs.

Now that the VIO is set up, you are ready to analyze the design.

1. Toggle the **vio_select** button to control the hardware from the VIO.
2. Press the **pause_vio_out** button to pause the counter.
3. Press the **toggle_vio_out** button to select between the count and the multiplier result.
4. Press the **reset_vio_out** button to reset the counter.

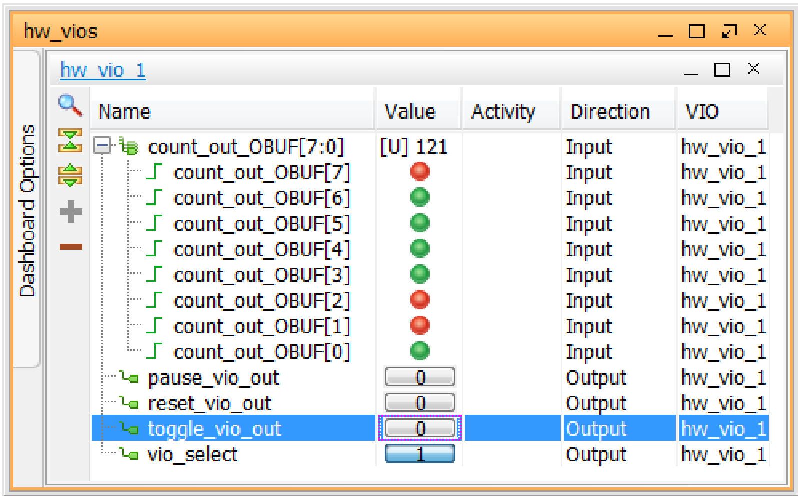


Figure 80: hw_vios Toggle Button Window

Step 4: Making the ECO Modifications

1. In the Flow Navigator, select the **Project Manager**.
2. In the Design Runs window, right-click on **impl_1** and select **Open Run Directory**.
3. The run directory opens in a file browser, as seen in the following figure. The run directory contains the routed checkpoint (`top_routed.dcp`) to be used for the ECO flow.

> **TIP:** *In a project-based design, the Vivado Design Suite saves intermediate implementation results as design checkpoints in the implementation runs directory. When you re-run implementation, the previous results are deleted. Save the router checkpoint to a new directory to preserve the modified checkpoint.*

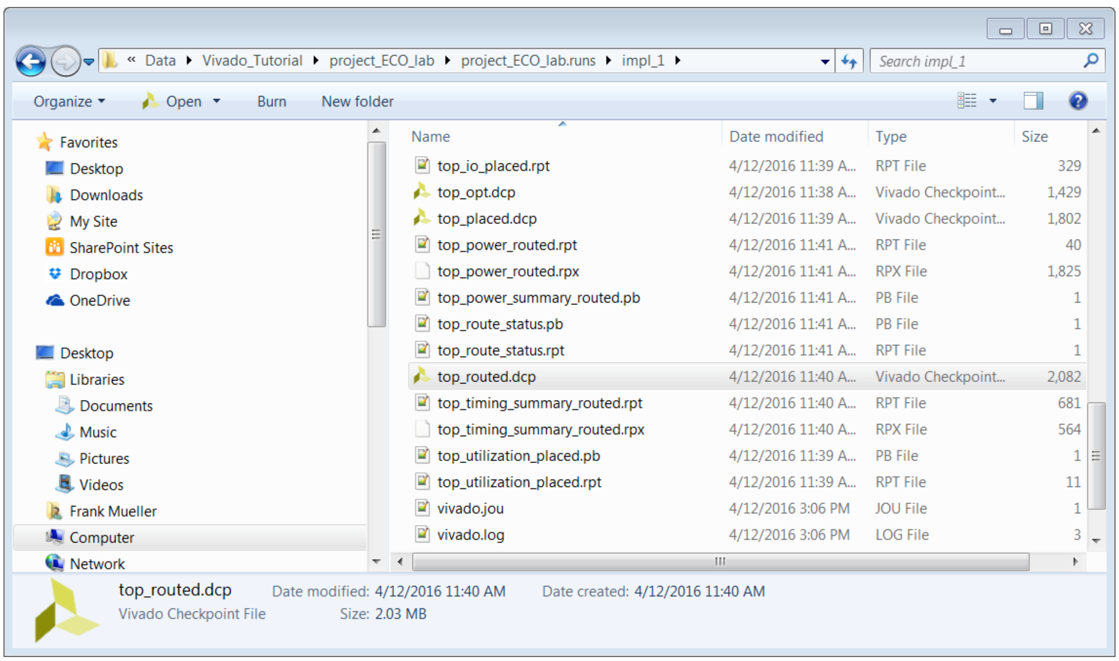


Figure 81: Implementation Run Directory

4. Create a new directory named `ECO` in the original `C:/Vivado_Tutorial/project_ECO_lab` project directory, and copy the `top_routed.dcp` file from the implementation runs directory to that newly created directory.
5. From the main menu, select **File > Open Checkpoint**.

   The Open Checkpoint dialog box opens.
6. Navigate to `C:/Vivado_Tutorial/project_ECO_lab/ECO` and select the `top_routed.dcp` checkpoint.

   A dialog box opens, asking whether to close the current project.
7. Click **Yes.**

8. From the main menu, select **Layout > ECO**.

   The ECO Layout is selected. The ECO Navigator is displayed on the left of the layout (highlighted in red in the following figure). It provides access to netlist commands, run steps, report and analysis tools, and commands to save changes and generate programming files.

   The Scratch Pad in the center of the layout (highlighted in red in the following figure) tracks netlist changes, as well as place and route status for cells, pins, ports, and nets.

   Note: *ECOs only work on design checkpoints. The ECO layout is only available after you have opened a design checkpoint in the Vivado IDE.*

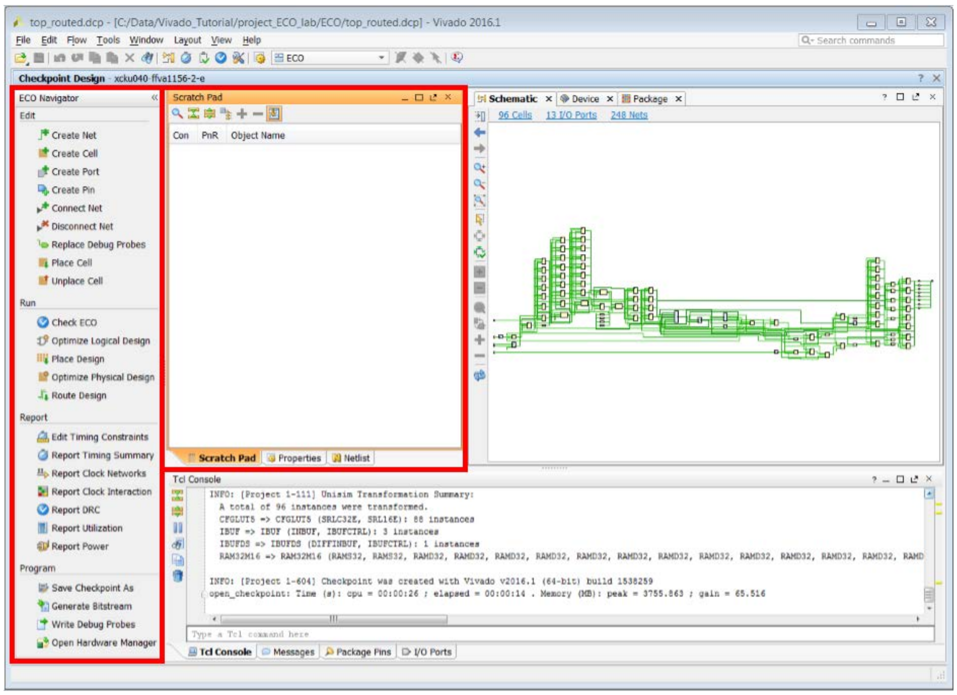


Figure 82: ECO Layout

To illustrate the capabilities of the ECO flow, you next change the functionality of the multiplier from a square of `count[3:0]` to a multiply by two.

9. From the Tcl Console, type the following command:

```
mark_objects -color blue [get_cells my_mult_0]
```

TIP: To make it easier to locate objects that are included in the ECO modifications, it helps to mark or highlight the objects with different colors.

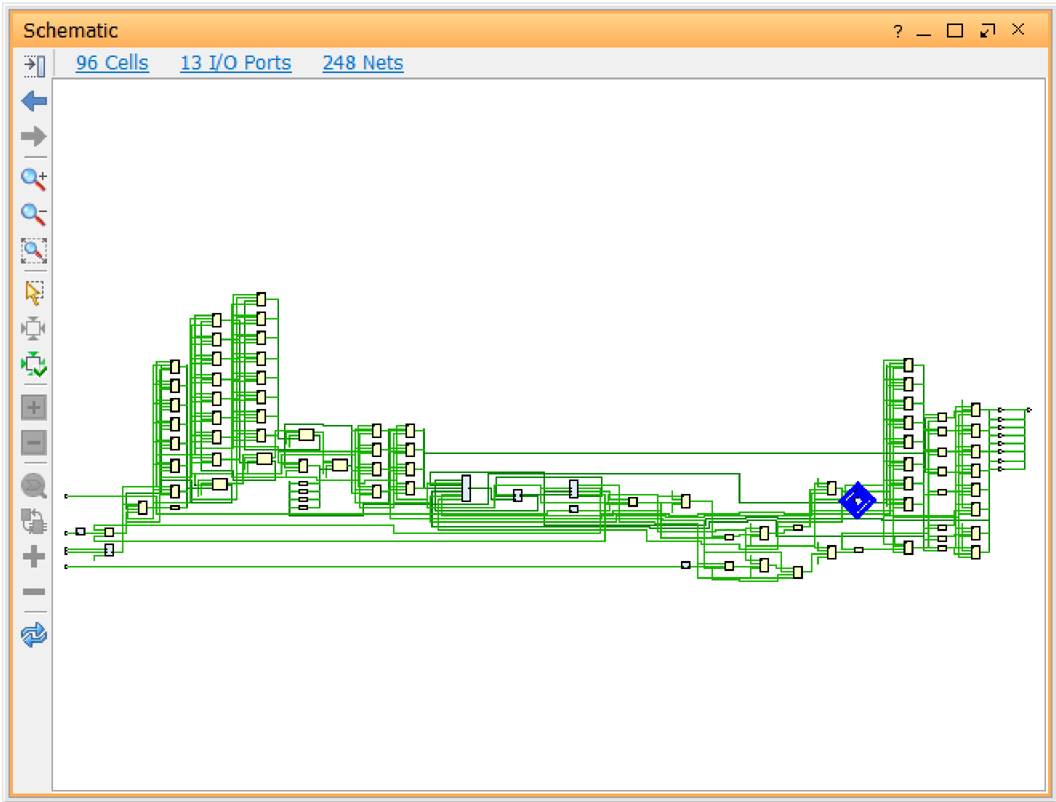


Figure 83: Schematic with Multiplier Marked

10. Zoom into the multiplier in the schematic window and select the `in2[3:0]` pins.

Alternatively, you can type the following command in the TCL console:

```
select_objects [get_pins my_mult_0/in2[*]]
```

11. Click the **Disconnect Net** button in the **Edit** section of the Vivado ECO Navigator. The net is disconnected from the pins in the schematic.

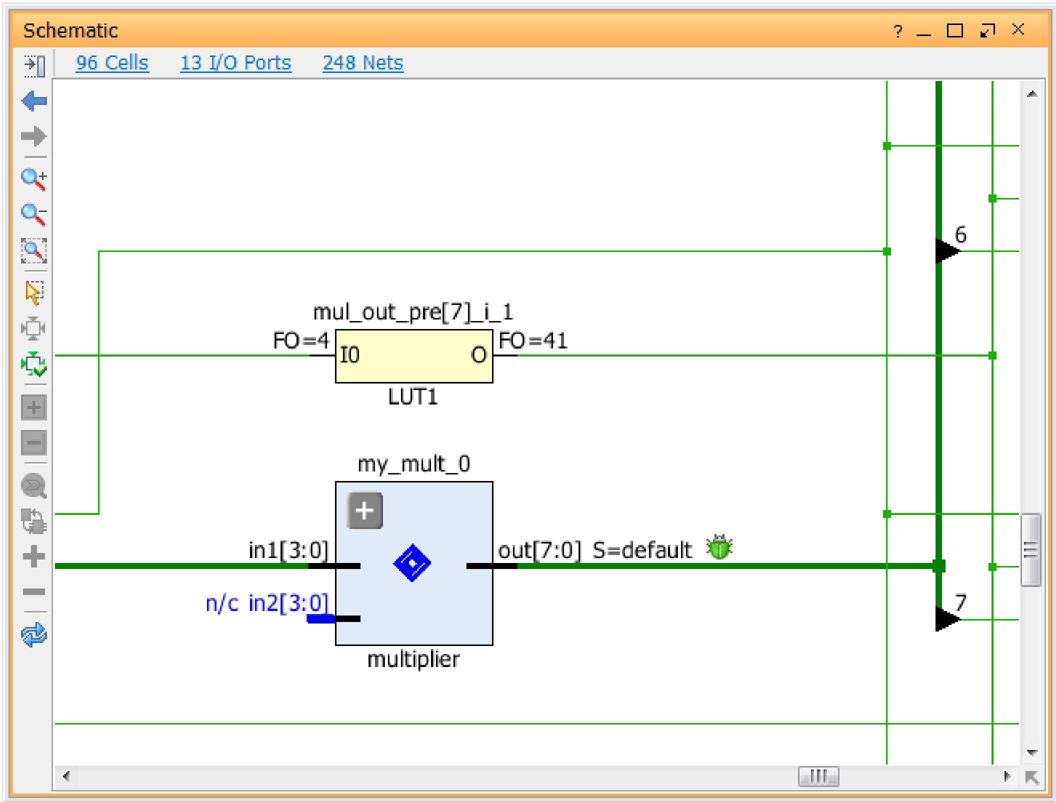


Figure 84: Schematic with Net Disconnected from Pins in2[3:0]

The Tcl Console reproduces the `disconnect_net` command that you just executed in the ECO Navigator. This is useful if you want to replay your ECO changes later by opening the original checkpoint and sourcing a TCL script with the ECO commands.

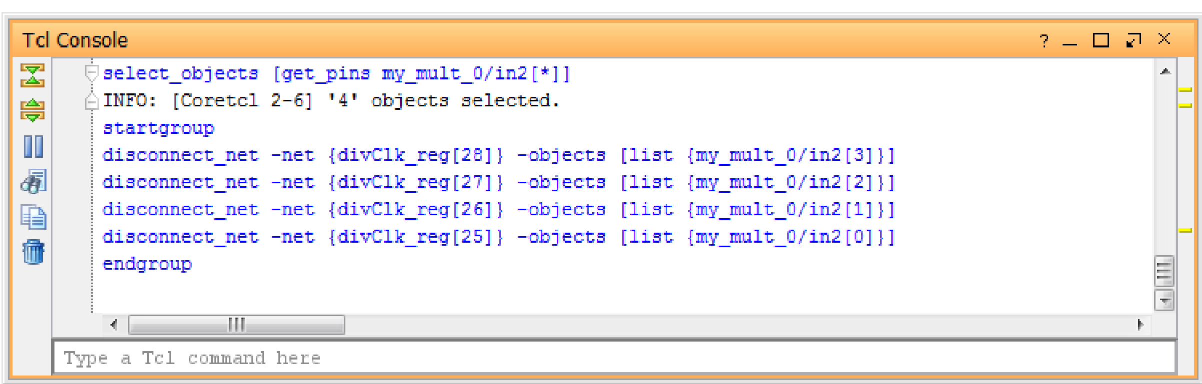


Figure 85: TCL Console Showing Executed ECO Commands

The Scratch Pad is populated with the 4 nets `divClk_reg[28:25]` that you disconnected and the multiplier input pins `my_mult_0/in2[3:0]`. Note the following in the Scratch Pad:

- The Scratch Pad connectivity column (`Con`) shows a check mark next to the `divClk_reg[28:25]` nets, indicating that they are still connected to the other multiplier inputs.
- The `my_mult_0/in2[3:0]` pins do not show a check mark next to them because they no longer have nets connected.
- The Place and Route (`PnR`) column is unchecked for everything, indicating that the changes have not yet been implemented on the device.

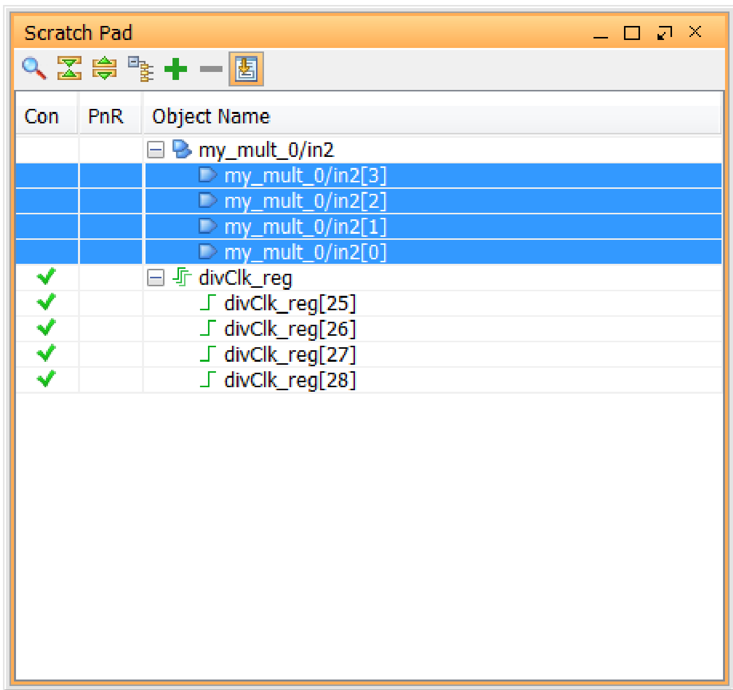


Figure 86: Scratch Pad Showing Status of ECO

12. In the Scratch Pad, select the {my_mult_0/in2[3], my_mult_0/in2[2], and my_mult_0/in2[0]} pins.
13. In the **Edit** section of the Vivado ECO Navigator, click **Connect Net**.

    The Connect Net dialog box opens.

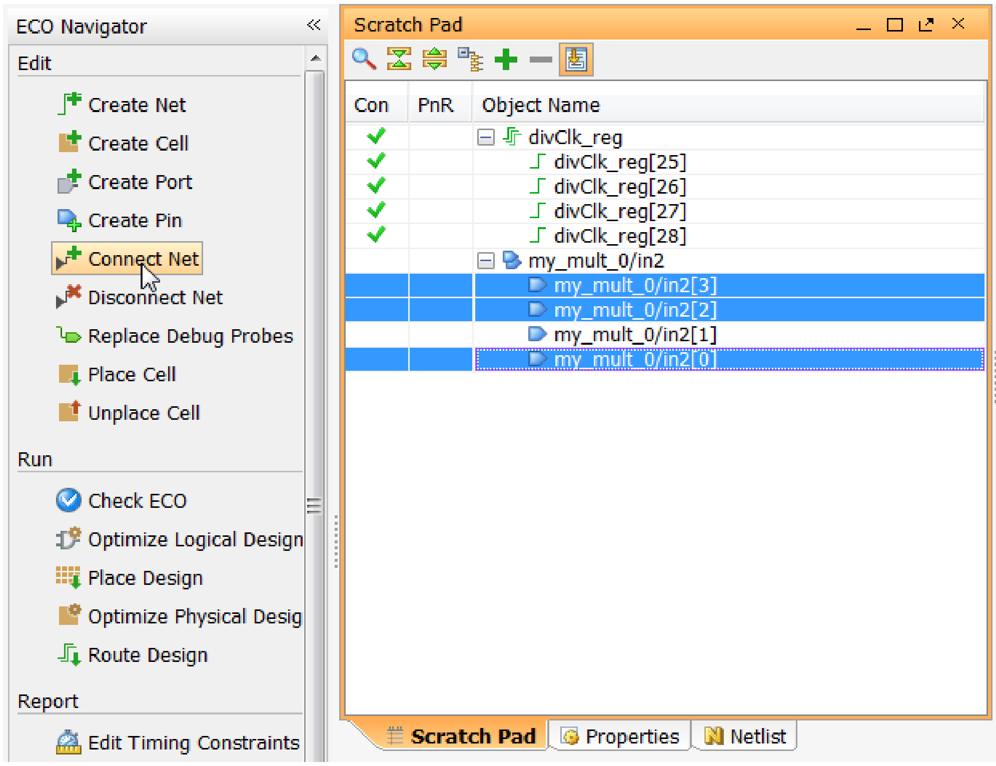


Figure 87: Connect Selected Pins

14. In the Connect Net dialog box, select <const0> from the GROUND section.

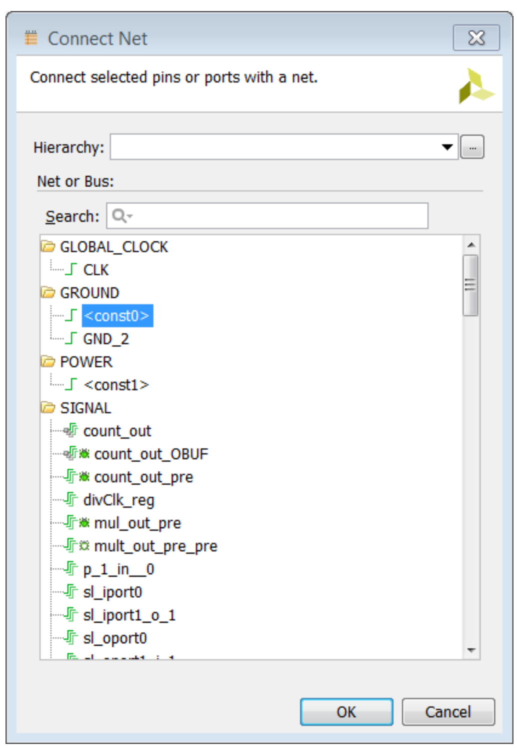


Figure 88: Connect Net Dialog Box

15. Click **OK**.

    `<const0>` is added to the Scratch Pad.

16. Collapse the `<const0>` signal.

    The three pins that you connected now show check marks in the Connectivity column of the Scratch Pad.

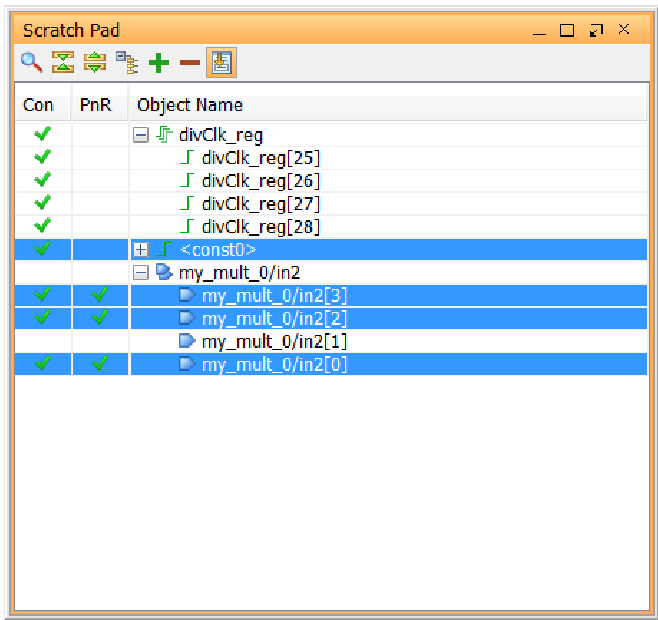


Figure 89: Scratch Pad Showing <const0> Connected to Pins

17. In the Scratch Pad, select the `my_mult_0/in2[1]` pin.

18. Click Connect Net.

    The Connect Net dialog box opens.

19. In the Connect Net dialog box, select `<const1>` from the POWER section.

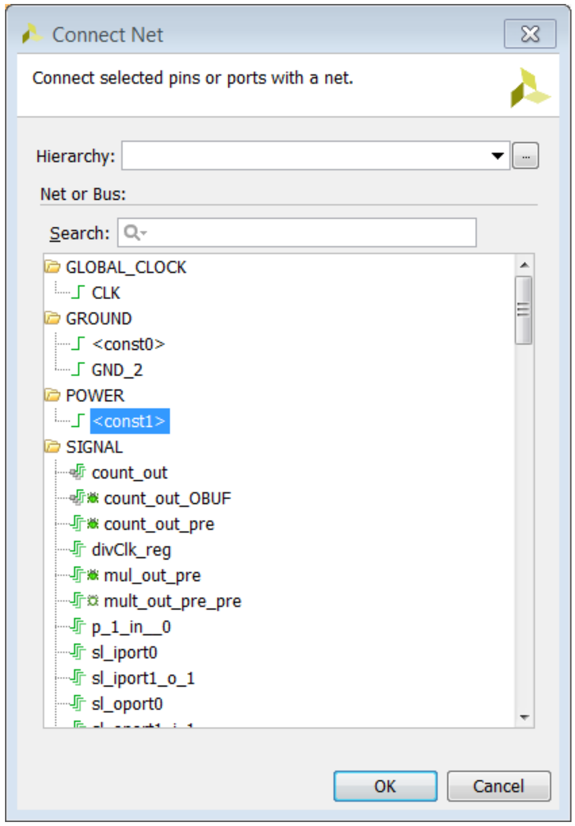


Figure 90: Connect Net Dialog Box

20. Click **OK**.

    `<const1>` is added to the Scratch Pad.

21. Collapse the `<const1>` signal.

The pin that you connected now shows check marks in the Connectivity column of the Scratch Pad.

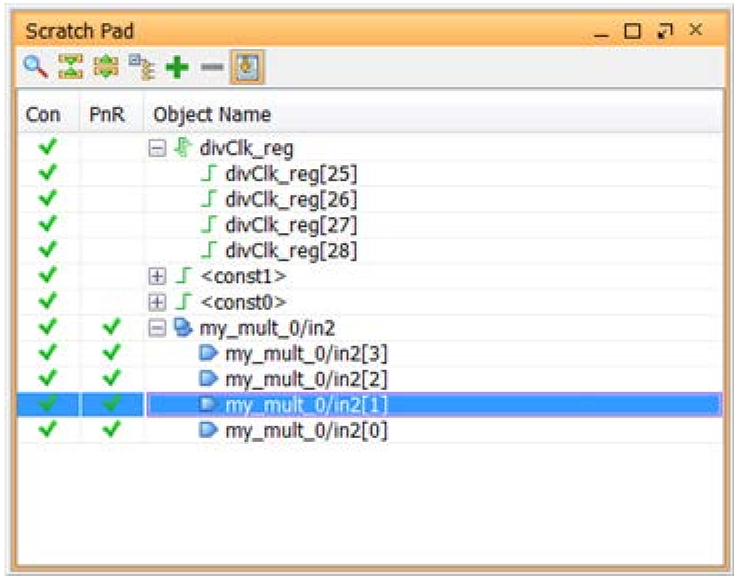


Figure 91: Scratch Pad Showing <const1> Connected to Pin

22. Select the `my_mult_0/in2` pin in the Scratch Pad.

    This command highlights the pins in the currently open Schematic view window, and shows the updated connections.

    Note: *Make sure that the* ***Autofit Selection***  *toggle button is highlighted in the Schematic window so you can see the entire path, as shown in the following figure.*

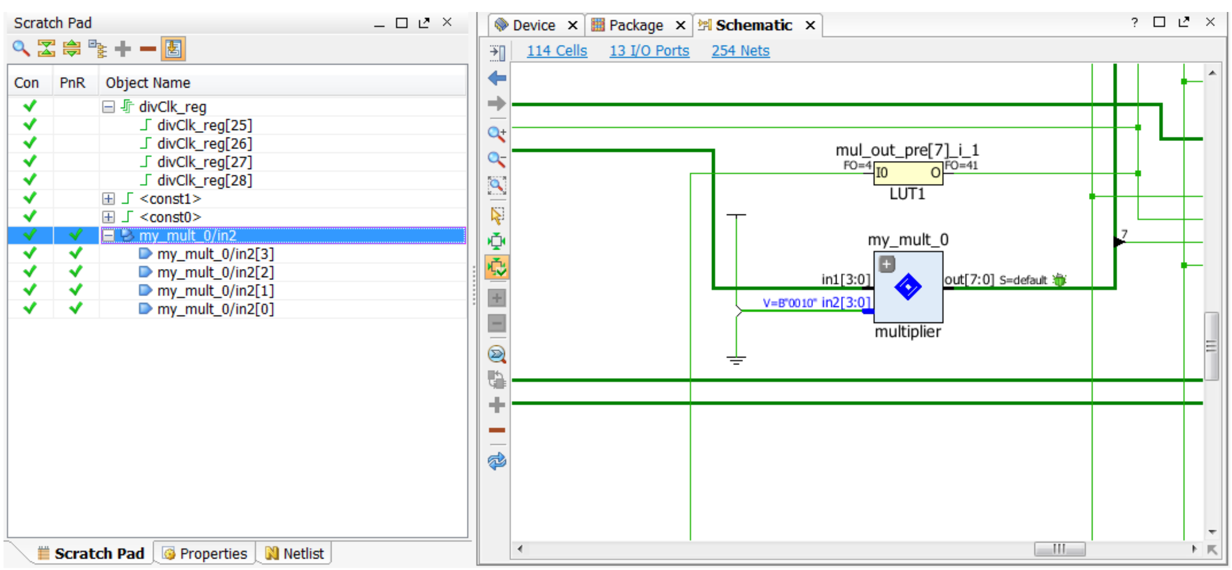


Figure 92: Resulting Schematic After Applying ECO Flow

When you observe the count signal on the LEDs, you only use 4 bits. The upper 4 bits are padded with zeroes.

Now, you will use the ECO flow to observe counter bit 24 on LED 7. The first step is to analyze the logic that drives `count_out_reg[3]`.

23. From the Tcl Console, type the following command:

```
select_objects [get_cells count_out[3]_i_1]
```

This lets you quickly identify the `LUT3` that drives the `count_out_reg[3]` register, which drives LED 3. The inputs are:

- mul_out_pre_reg[7] for pin I0
- count_out_pre_reg[3] for pin I1
- tog_state_reg for pin I2

24. Click the **Properties** tab to view the cell properties and select the **Truth Table** tab.

25. Click **Edit LUT Equation** to view the equation for the `LUT3`. Note the LUT equation:

```
O= I1 & !I2 + I0 & I2
```

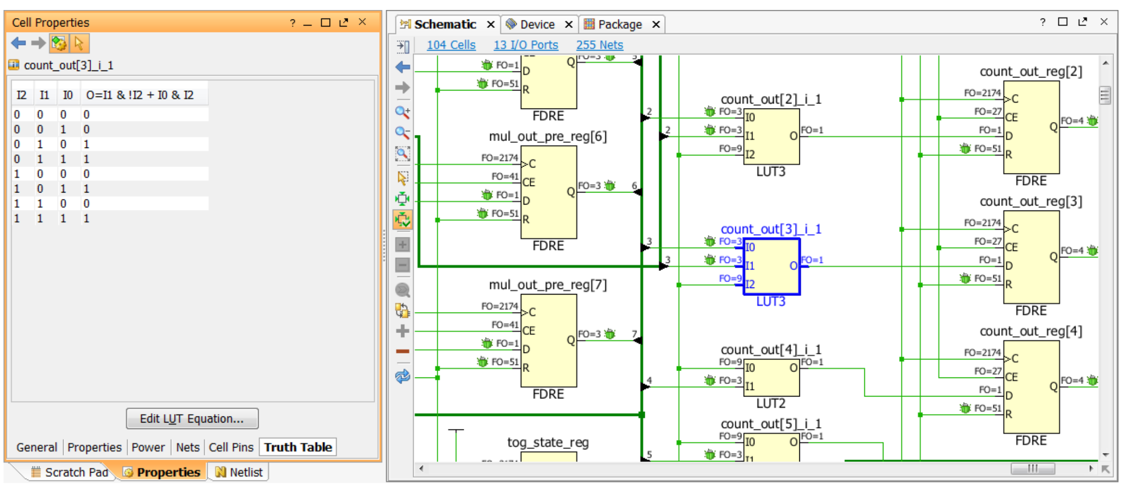


Figure 93: LUT3 Driving count_out_reg[3]

26. From the Tcl Console, type the following command:

```
select_objects [get_cells count_out[7]_i_1]
```

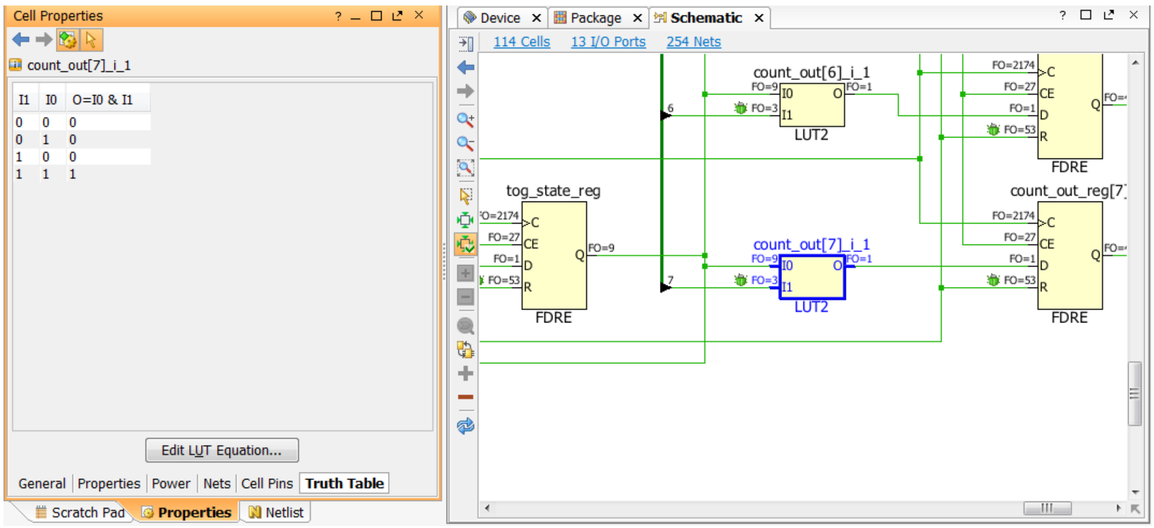


Figure 94: LUT2 Driving count_out_reg[7]

This command selects the LUT2 that drives the `count_out_reg[7]` register, which drives LED 7 on the KCU105 board. The only inputs are `tog_state_reg` for pin `I0` and `mul_out_pre_reg[7]` for pin `I1`. You need to replace the LUT2 with a 3-input LUT and connect the output of counter register `divClk_reg[24]` to the additional input pin.

27. In the Vivado ECO Navigator, under **Edit**, click **Create Cell**.

    The Create Cell dialog box opens.

    a. In the **Cell name** field, enter `ECO_LUT3`.

    b. In the **Search** field, enter `LUT3`.

    c. Select LUT3 as the cell type and copy the LUT equation `O=I1 & !I2 + I0 & I2` from cell `count_out[3]_i_1`.

    d. Click **OK**.

       ECO_LUT3 is added to the Scratch Pad and the schematic.

    e. Right-click the newly added `ECO_LUT3` cell in the Scratch Pad, then select **Mark** and the color red.

Note*: Marking the ECO_LUT3 cell makes it easier to locate.*

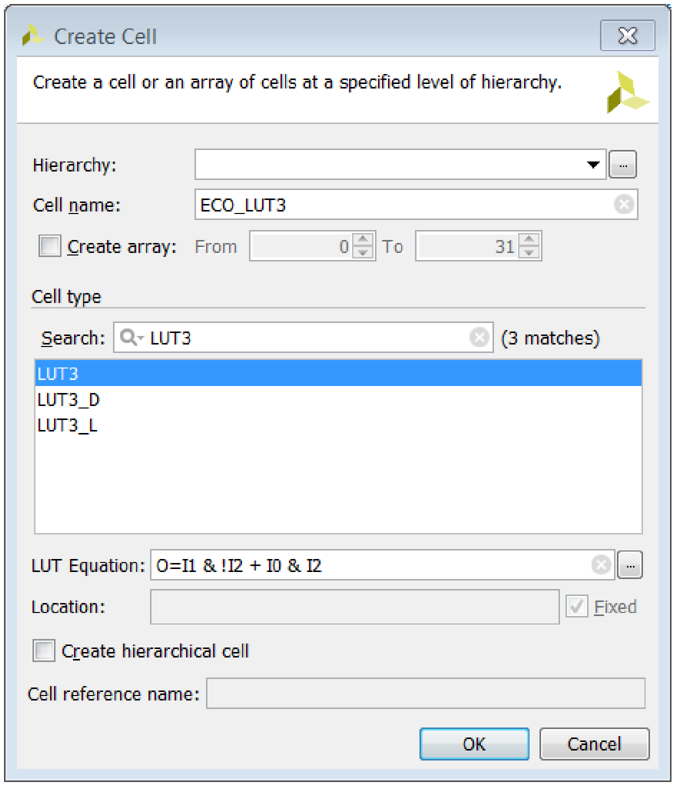


Figure 95: Create Cell Dialog Box

Because you copied the LUT equation from cell `count_out[7]_i_1`, the nets must be hooked up in the same order, with the following connections:

- Net `mul_out_pre[7]` connected to pin `I0`
- Net `divClk_reg_n_0_[24]` connected to pin `I1`
- Net `tog_state` connected to pin `I2` of `ECO_LUT3`

28. Locate the `tog_state` net driven by the `tog_state_reg` register in the schematic and select it. Alternatively you can select the net from the Tcl console by running the following command:

```
select_objects [get_nets tog_state]
```

29. Connect the `I2` pin of the newly added `ECO_LUT3` cell by doing the following:

    a. Hold down the **CTRL** key and select pin `I2` in the Scratch Pad. This selects pin `I2` in addition to the already selected `tog_state` net.

    b. Click **Connect Net**.

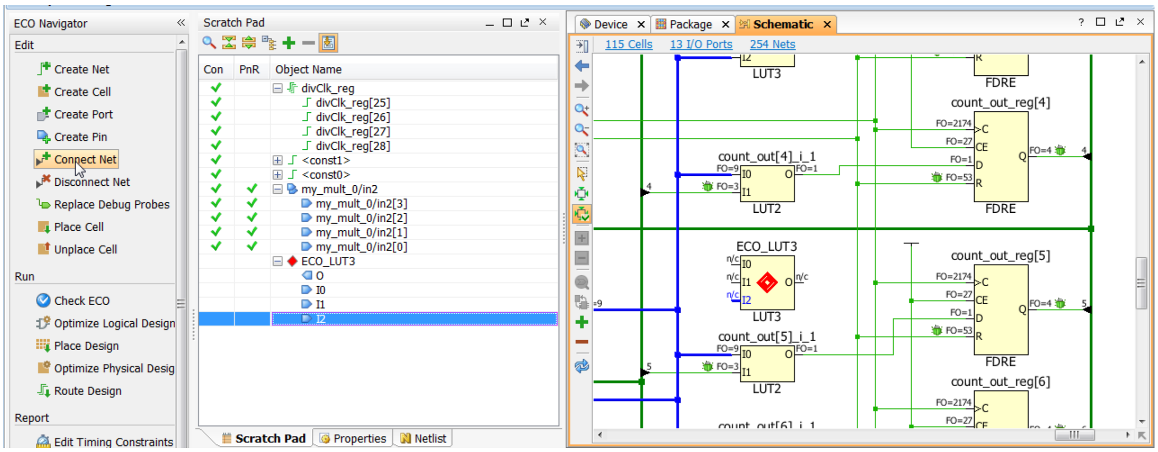


Figure 96: Scratch Pad and Schematic Showing ECO_LUT3 Added

30. Locate the `mul_out_pre[7]` net in the schematic and select it.

    Alternatively, you can select the net from the Tcl Console by executing the following command:

    ```
    select_objects [get_nets mul_out_pre[7]]
    ```

31. Connect the `I0` pin of the newly added `ECO_LUT3` cell by doing the following:

    a. Hold down the **CTRL** key and select pin `I0` in the Scratch Pad. This selects pin `I0` in addition to the already selected `mul_out_pre[7]` net.

    b. Click **Connect Net**.

       A dialog box opens, asking if you want to unset the `DONT_TOUCH` property on the net `mul_out_pre[7]` net to continue.

c. Click **Unset Property and Continue**.

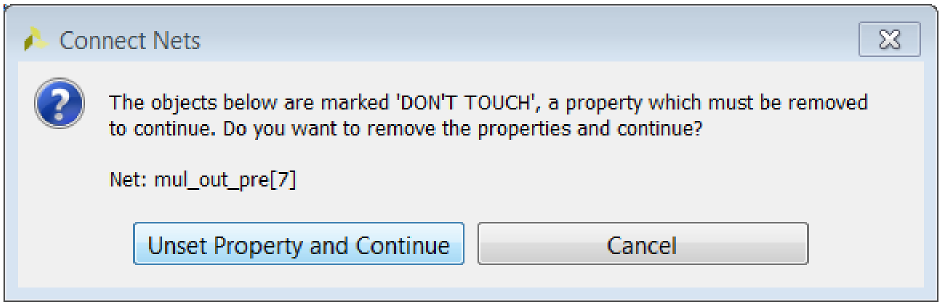


Figure 97: Connect Nets Dialog Box for Nets with DONT_TOUCH Property

Note: *Because the `mul_out_pre[7]` net previously had been connected to an ILA debug probe, the MARK_DEBUG and DONT_TOUCH properties have been applied to the net to preserve it for debug. In order to change the connectivity of the net, the DONT_TOUCH property first has to be removed.*

32. Locate the `divClk_reg_n_0_[24]` net in the schematic and select it.

    Alternatively, you can select the net from the TCL console by executing the following command:

    ```
    select_objects [get_nets divClk_reg_n_0_[24]]
    ```

33. Connect the `I1` pin of the newly added `ECO_LUT3` cell by doing the following:

    a. Hold down the **CTRL** key and select pin `I1` from the Scratch Pad. This selects pin `I1` in addition to the already selected `divClk_reg_n_0_[24]` net.

    b. Click **Connect Net**.

    Next, you need to connect the updated logic function implemented in the newly created `LUT3` to the `D` input of `count_out_reg[7]`. The first step is to delete the `LUT2` that was previously connected to the `D` input.

34. Select the `LUT2` `count_out[7]_i_1` in the schematic window.

    Alternately, you can select it by executing the following command in the Tcl console:

    ```
    select_objects [get_cell count_out[7]_i_1]
    ```

35. From the toolbar, click the **Delete** button  to delete the selected cell.

36. Select the net connected to the `D` input of the `count_out_reg[7]` register in the schematic window.

    Alternatively you can select the net from the TCL console by executing the following command:

    ```
    select_objects [get_nets count_out[7]_i_1_n_0]
    ```

37. Connect the `O` pin of the newly added `ECO_LUT3` cell by doing the following:

    a. Hold down the **CTRL** key and select pin `O` from the Scratch Pad.

    b. Click **Connect Net**.

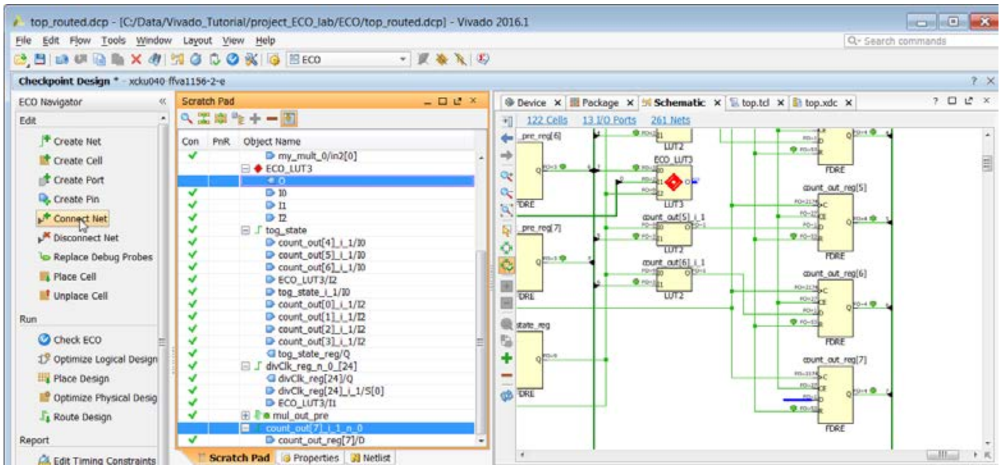


Figure 98: Connecting ECO_LUT3 Output

The ECO modifications are completed.

Step 5: Implementing the ECO Changes

Before you place and route the updates, you need to check for any illegal logical connections or other logical issues introduced during the ECO that would prevent a successful implementation of your changes.

1. In the Vivado ECO Navigator, under **Run**, click **Check ECO**.

   The following figure shows the messages generated by the ECO DRC.

   - The two Critical Warnings are due to the partially routed signals that are a result of the ECO and will be cleaned up during incremental place and route.
   - No other warnings have been issued and you are ready to implement the changes.

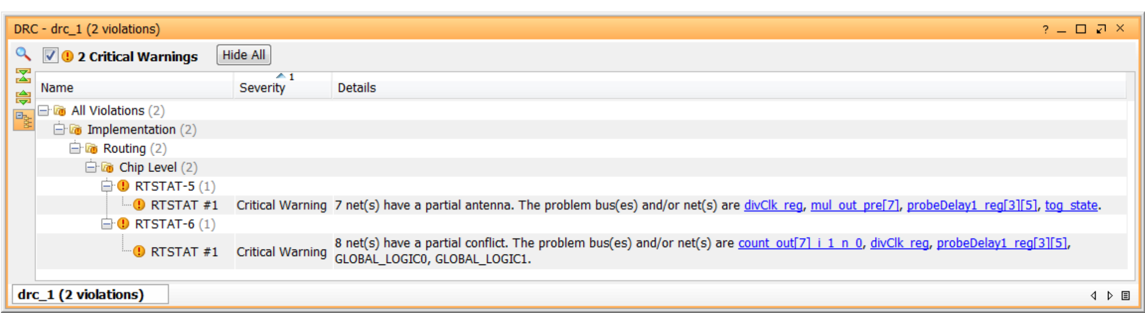


Figure 99: Check ECO DRC Messages

Because you added additional logic, you need place the logic using the incremental place, and then route the updated net connections using incremental route.

2. In the Vivado ECO Navigator, under **Run**, click **Place Design**.

   The Place Design dialog box opens, allowing you to specify additional options for the `place_design` command. For this exercise, do not specify additional options.

3. Click **OK**.

4. Vivado runs the incremental placer.

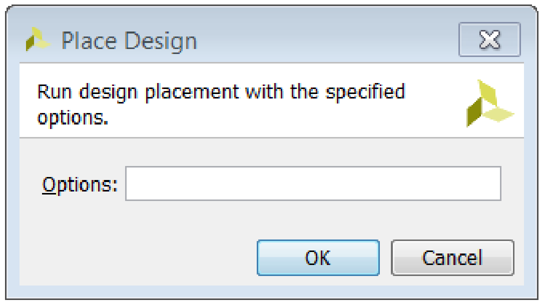


Figure 100: Place Design Dialog Box

At the end of the place_design step, the incremental Placement Summary is displayed in the TCL Console.

```
+--------------------------------------------------------------------------------+
|Incremental Placement Summary                                                   |
+--------------------------------------------------------------------------------+
|                        Type                            | Count  | Percentage |
+--------------------------------------------------------------------------------+
|  Total instances                                       |   4207 |     100.00 |
|  Reused instances                                      |   4205 |      99.95 |
|  Non-reused instances                                  |      2 |       0.05 |
|    New                                                 |      1 |       0.02 |
|    Discarded illegal placement due to netlist changes  |      1 |       0.02 |
+--------------------------------------------------------------------------------+
|Incremental Placement Runtime Summary                                           |
+--------------------------------------------------------------------------------+
|  Initialization time(elapsed secs)                                  |    6.50 |
|  Incremental Placer time(elapsed secs)                              |    6.66 |
+--------------------------------------------------------------------------------+
```

Figure 101: Post-Place Incremental Reuse Summary

The incremental placement summary shows that the following two cells did not have their previous placement reused:

- o The new ECO_LUT3 cell, which had to be placed from scratch
- o The count_out_reg[7] cell, which had to get updated placement due to the placement of the ECO_LUT3 driving it

5. In the Vivado ECO Navigator, under **Run**, click **Route Design**.

The Route Design dialog box opens.

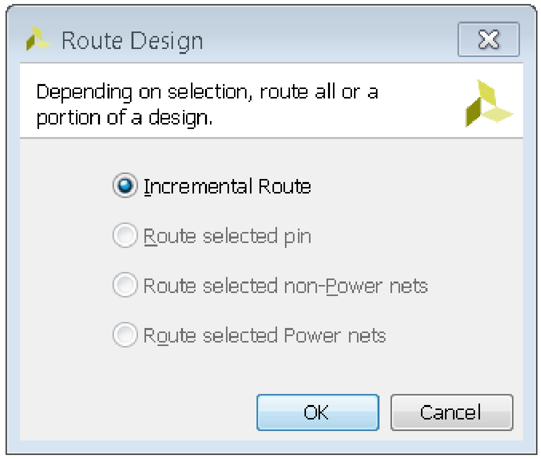


Figure 102: Route Design Dialog Box

Depending on your selection, you have four options to route the ECO changes:

- **Incremental Route**: This is the default option.
- **Route selected pin**: This option limits the route operation to the selected pin.
- **Route selected non-Power nets**: This option routes only the selected signal nets.
- **Route selected Power nets**: This option routes only the selected VCC/GND nets.

In this case, the best choice is to route the changes you made incrementally.

6. Select **Incremental Route**.
7. Click **OK**.

At the end of the `route_design` step, the incremental Routing Reuse Summary displays in the Tcl Console.

```
-----------------------------------------------------------
|Incremental Routing Reuse Summary                        |
-----------------------------------------------------------
|Type                     | Count     | Percentage        |
-----------------------------------------------------------
|Fully reused nets        |      3200|            99.88 |
|Partially reused nets    |         3|             0.09 |
|Non-reused nets          |         1|             0.03 |
-----------------------------------------------------------
```

Figure 103: Post-Route Incremental Reuse Summary

Most of the nets did not require any routing and have been fully reused.

> ***TIP:*** *It is a good idea to run* `report_route_status` *after the route operation to make sure all the nets have been routed and none have any routing issues. This is especially true if you only routed selected pins or selected nets and want to make sure you have not missed any routes.*

8. In the Tcl Console, run the `report_route_status` command.

   The Design Route Status looks similar to the following status.

```
Design Route Status
                                               :      # nets :
   ------------------------------------------- : ----------- :
   # of logical nets.......................... :        4444 :
       # of nets not needing routing.......... :        1235 :
           # of internally routed nets........ :        1177 :
           # of nets with no loads............ :          58 :
       # of routable nets..................... :        3209 :
           # of fully routed nets............. :        3209 :
       # of nets with routing errors.......... :           0 :
   ------------------------------------------- : ----------- :
```

Figure 104: Design Route Status Post Implementation

Before you generate a bitstream, run the ECO DRCs on the design.

9. In the ECO Navigator, click **Check ECO**. Make sure no violations are reported.

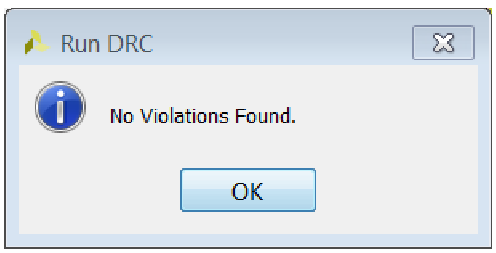


Figure 105: Post Implementation Check ECO Results

10. In the Vivado ECO Navigator, under **Program**, click **Save Checkpoint As**.

    The Save Checkpoint As dialog box opens and you can specify a name for the checkpoint file to write to disk.

11. Click **OK** to save a checkpoint file with your changes.

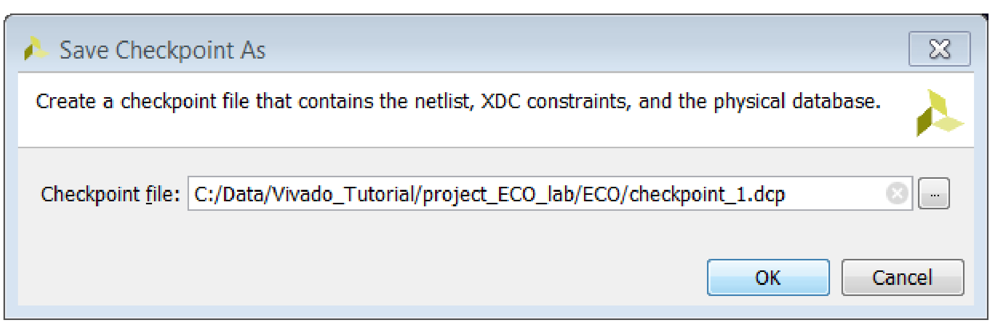


Figure 106: Saving ECO Modifications to a New Checkpoint

12. In the Vivado ECO Navigator, under **Program**, click **Generate Bitstream**.

    The Generate Bitstream dialog box opens.

You can specify a name for a Bit file and select the desired options for the write_bitstream operation.

13. Click **OK** to generate a bitstream with your changes.

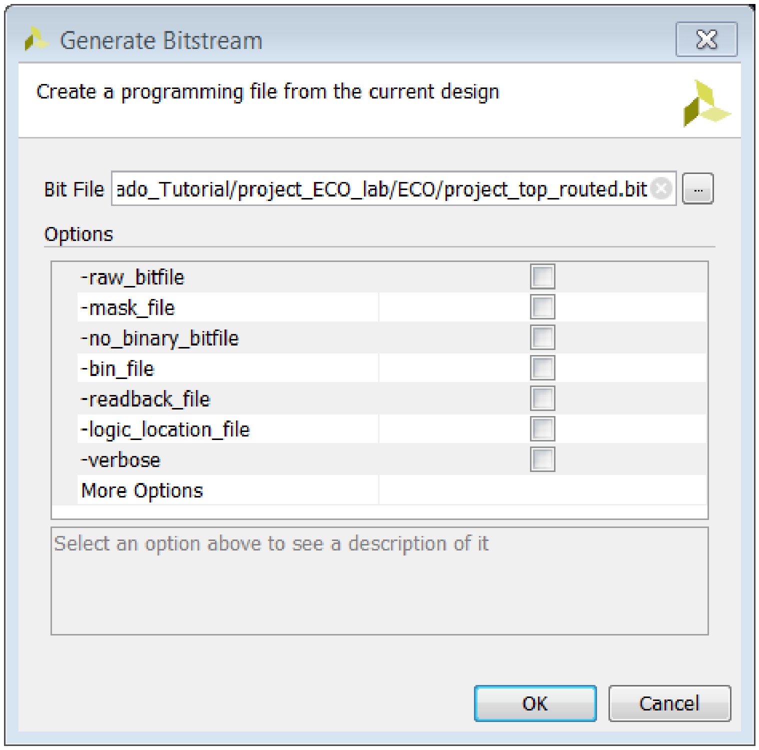


Figure 107: Generate Bitstream Dialog Box

14. In the Vivado ECO Navigator, under **Program**, click **Write Debug Probes**.

The Write Debug Probes dialog box opens.

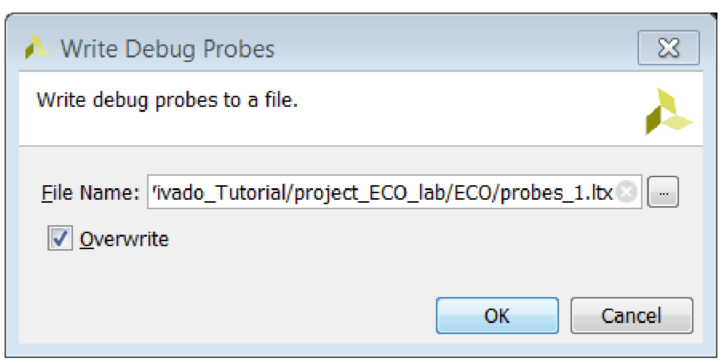


Figure 108: Write Debug Probes Dialog Box

You can specify a name for a .ltx file for your debug probes.

15. Click **OK** to generate debug probes file (LTX).

    This command allows you to generate a new .ltx file for your debug probes. If you made changes to your debug probes using the **Replace Debug Probes** command, you need to save the updated information to a new debug probes file to reflect the changes in the Vivado Hardware Manager

16. Follow the instructions in Step 4: Making the ECO Modifications to download the generated bitstream programming file and debug probes file into the target FPGA device using the Hardware Manager to check your ECO modifications.

Step 6: Replacing Debug Probes

Another powerful feature of the Vivado ECO flow is the ability to replace debug probes on a previously inserted Debug Hub. After the debug probes have been replaced, a new LTX file can be generated that contains the updated debug probe information.

To replace a debug probe in your previously modified design, do the following:

1. From the main menu, select **File > Open Checkpoint**.

   The Open Checkpoint dialog box opens.

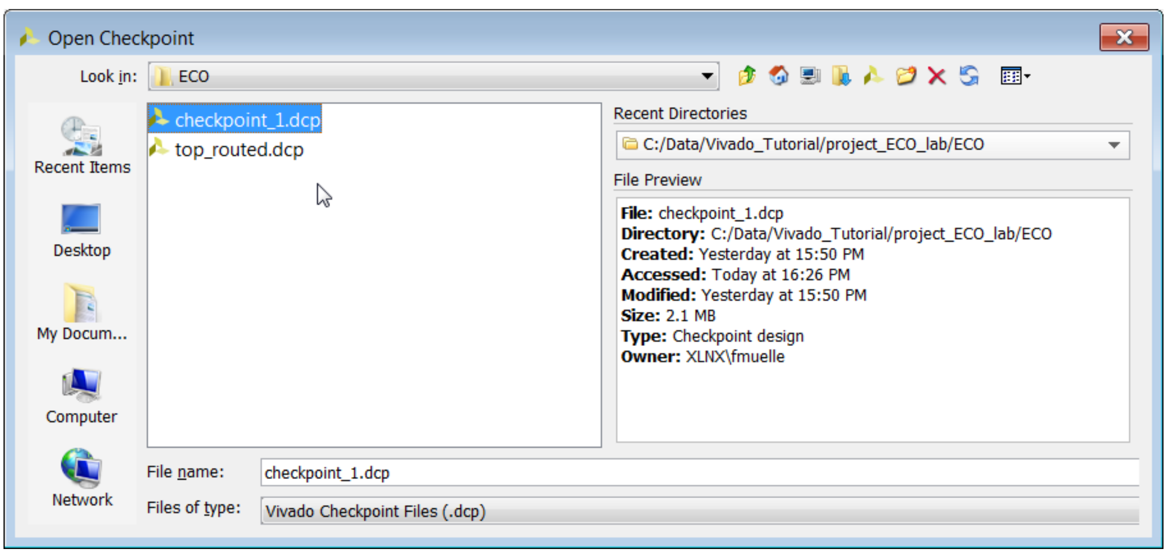


Figure 109: Open Checkpoint Dialog Box

2. Browse to the `C:/Data/Vivado_Tutorial/project_ECO_lab/ECO` directory and select the previously saved `checkpoint_1.dcp` file.

3. Close any previously open checkpoints.

4. From the main menu, select **Layout > ECO**.

5. In the Vivado ECO Navigator, under **Edit**, click **Replace Debug Probes**.

   The Replace Debug Probes dialog box opens.

In this example, you will replace the net `reset_vio` that is connected to `probe4` of `u_ila_0` with the net `toggle_vio`.

6. Scroll to the bottom of the probes for `u_ila_0` in the Replace Debug Probes dialog and click the icon next to the `reset_vio` net name in the Probe column.

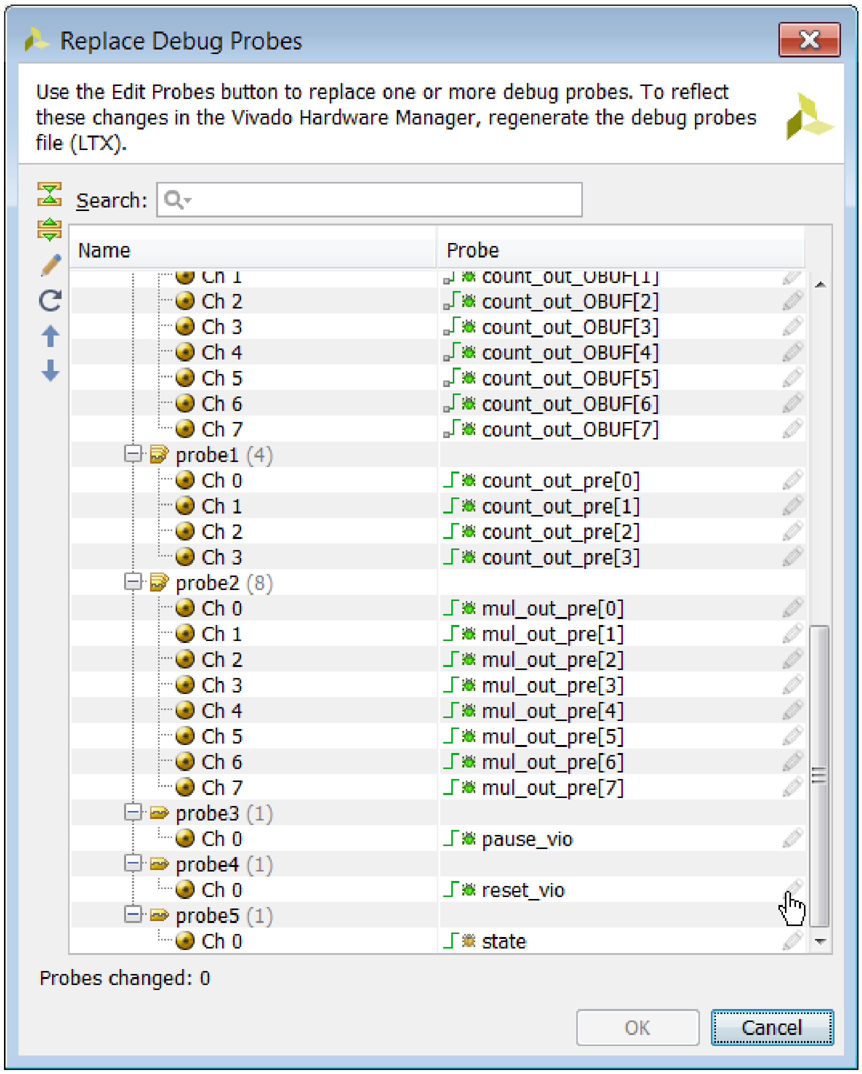


Figure 110: Replace Debug Probes Dialog Box

The Choose Nets window opens.

7. Choose a new net to connect to the debug probe `probe4` by doing the following:
   a. Type **toggle_vio** in the search field of the Choose Nets dialog box.
   b. Click **Find.**
   c. Select the `toggle_vio` net, and move it to the **Selected names** section.

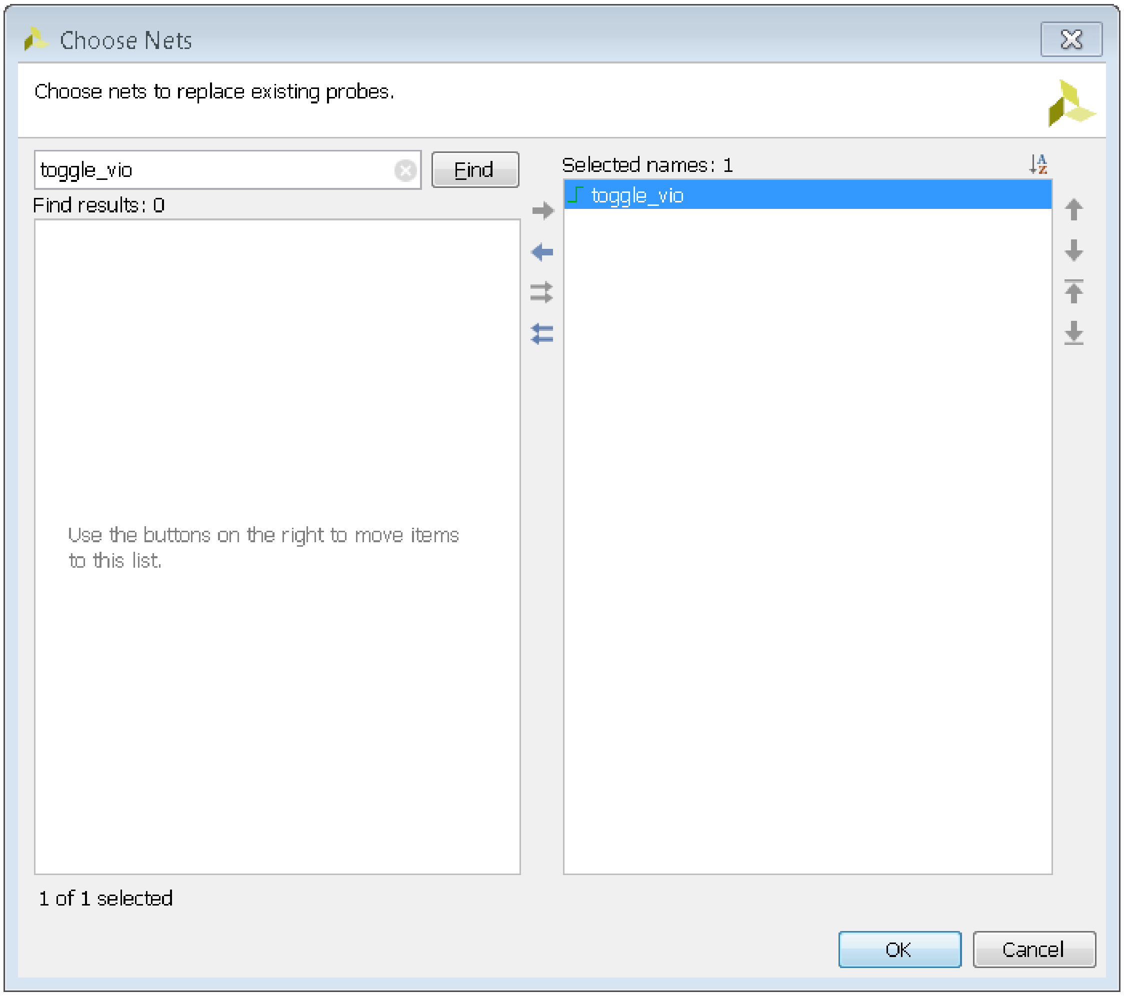


Figure 111: Choose Nets Dialog Box

8. In the Replace Debug Probes dialog box, click **OK**.
9. In the message window that opens, click **Unset Property and Continue**.

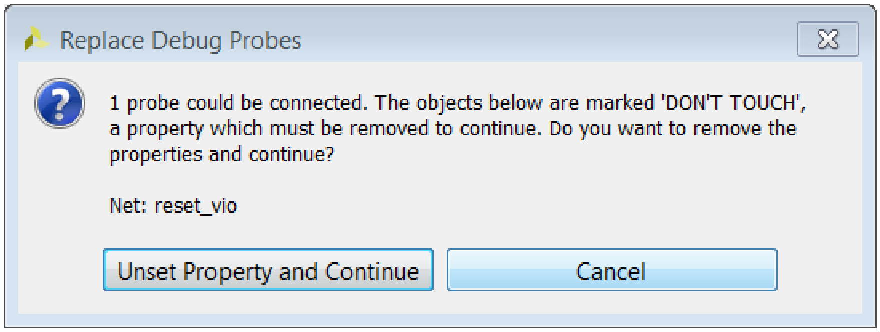


Figure 112: Unset DONT_TOUCH Property Dialog Box

10. Repeat steps 5 through 14 of Step 5: Implementing the ECO Changes to generate an updated design checkpoint, bitstream file, and probes file (LTX).

    The updated debug probes file has the `reset_vio` net for `probe4` replaced with net `toggle_vio`, which you can verify when you program the device with the updated bit file and debug probes file.

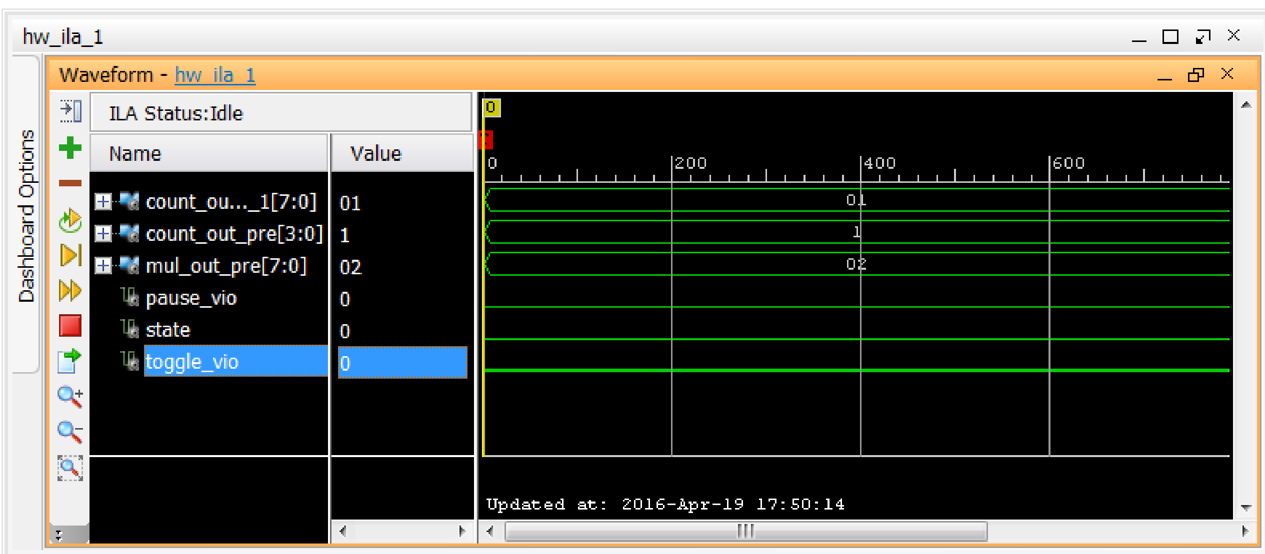


Figure 113: ILA Waveform with Update Debug Probes

Conclusion

In this lab, you did the following:

- Made changes to the previously implemented design using the Vivado ECO flow.
- Implemented the changes using incremental place and route.
- Generated a bitstream and probes file with your changes to configure the FPGA.
- Used the Replace Debug Probes command to switch the sources for debug probes in the design.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

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.