

# **Vivado Design Suite**

## **Tutorial:**

### ***Implementation***

UG986 (v2014.3) October 23, 2014

This tutorial document has been validated for the following software versions: Vivado Design Suite 2014.3 and 2014.4.



---

## Revision History

The following table shows the revision history for this document.

<b>Date</b>	<b>Version</b>	<b>Revision</b>
10/23/2014	2014.3	Validated with release.
06/04/2014	2014.2	Validated with release.
04/02/2014	2014.1	Validated with release.

# Table of Contents

Revision History .....	2
Implementation Tutorial .....	5
Overview .....	5
Tutorial Design Description .....	5
Hardware and Software Requirements .....	6
Preparing the Tutorial Design Files .....	6
Lab #1: Using Implementation Strategies .....	7
Introduction .....	7
Step 1: Opening the Example Project .....	7
Step 2: Creating Additional Implementation Runs .....	9
Step 3: Analyzing Implementation Results .....	10
Step 4: Tightening Timing Requirements .....	13
Conclusion .....	14
Lab #2: Using Incremental Compile .....	15
Introduction .....	15
Step 1: Opening the Example Project .....	15
Step 2: Compiling the Reference Design .....	17
Step 3: Create New Runs .....	19
Step 4: Making Incremental Changes .....	21
Step 5: Running Incremental Compile .....	23
Conclusion .....	26
Lab #3: Manual and Directed Routing .....	27
Introduction .....	27
Step 1: Opening the Example Project .....	27
Step 2: Place and Route the Design .....	29
Step 3: Analyze Output Bus Timing .....	30

Step 4: Improve Bus Timing through Placement..... 36  
Step 5: Using Manual Routing to Reduce Clock Skew..... 40  
Step 6: Copy Routing to Other Nets..... 49  
Conclusion..... 51  
Notice of Disclaimer ..... 52  
Please Read: Important Legal Notices ..... 52



**IMPORTANT:** This tutorial requires the use of the Kintex®-7 family of devices. You will need to update your Vivado 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 three 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.

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 quick take video at <http://www.xilinx.com/training/vivado/implementing-the-design.htm>.

**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 Static Timing Analysis Design Constraints](#)
- [Vivado Advanced Tools and Techniques](#)

---

## 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.

Both the CPU Netlist and BFT Core designs target an XC7K70T device. Small designs are used to allow the tutorial to be run with minimal hardware requirements and to enable timely completion of the tutorial, as well as to minimize the data size.

---

## Hardware and Software Requirements

This tutorial requires that the 2014.2 Vivado Design Suite software release or later is installed. The following partial list describes the operating systems that the Vivado Design Suite supports on x86 and x86-64 processor architectures:

Microsoft Windows Support:

- Windows 8.1 Professional (32-bit and 64-bit), English/Japanese
- Windows 7 and 7 SP1 Professional (32-bit and 64-bit), English/Japanese

Linux Support:

- Red Hat Enterprise Workstation 6.4 and 6.5 (32-bit and 64-bit)
- SUSE Linux Enterprise 11 (32-bit and 64-bit)
- Cent OS 6.4 and 6.5 (64-bit)

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

You can find the files for 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 the `<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.*

# Lab #1: Using Implementation Strategies

## 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

Open the Vivado IDE:

- On Linux,
  1. Change to the directory where the lab materials are stored:
 

```
cd <Extract_Dir>/Vivado_Tutorial
```
  2. Launch the Vivado IDE: **vivado**
- On Windows,
  1. Launch the Vivado Design Suite IDE:

**Start > All Programs > Xilinx Design Tools > Vivado 2014.x > Vivado 2014.x**

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

**Note:** As an alternative, click the **Vivado 2014.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.

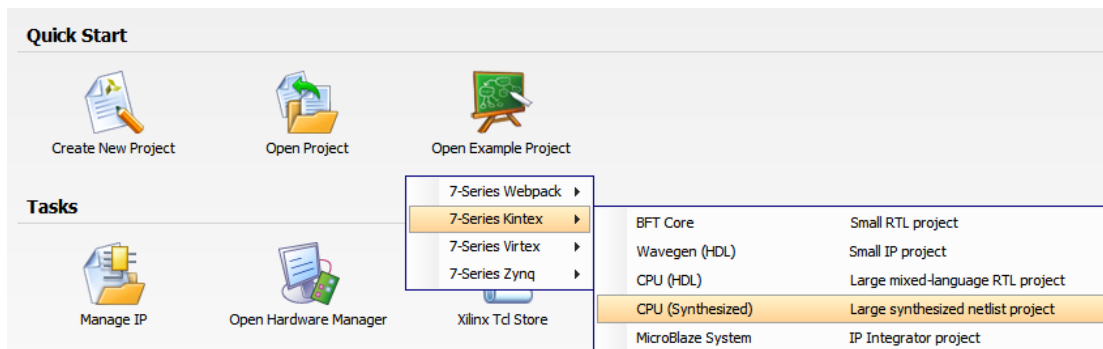
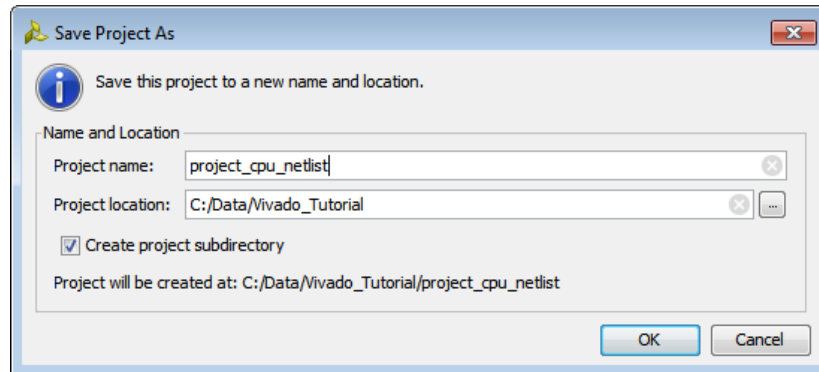


Figure 1: Open Example Project

- From the Getting Started page, click **Open Example Project** and select the **7 Series Kintex > CPU (Synthesized)** design.

A dialog box appears stating that the Project is Read-Only.

- Click **Save Project As** to specify a project name and location.



**Figure 2: Save Project As**

- Specify the following, and click **OK** :
  - Project name: **project\_cpu\_netlist**
  - Project location: <Extract\_Dir>

The Vivado IDE displays the default view of the opened project, as shown in [Figure 3](#).

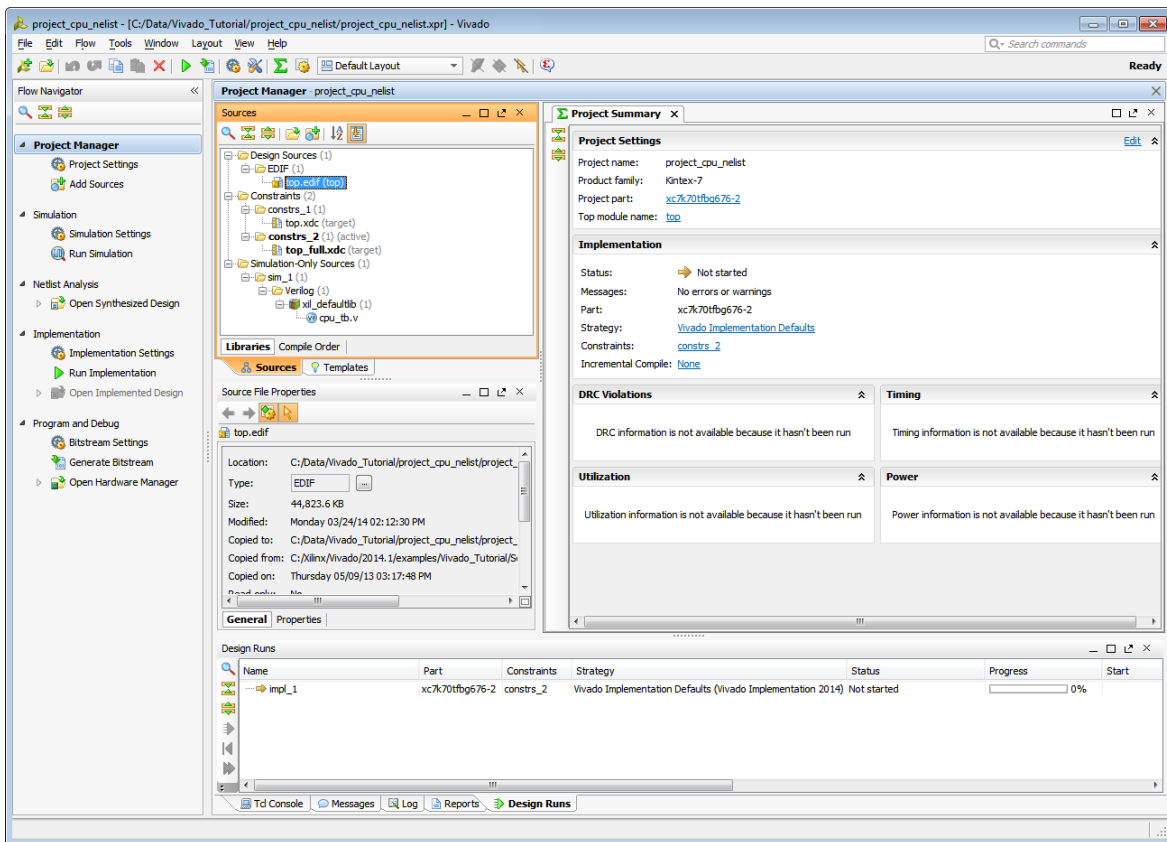


Figure 3: Project Summary Window

## 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**.

The Create New Runs dialog box opens.

2. Click **Next** to open the Configure Implementation dialog box.

The Configure Implementation Runs dialog box opens, with a new implementation run defined. You can configure this run as well as add other runs.

3. In the **Strategy** drop-down menu, select **Performance\_Explore** as the Strategy for the run.
4. In the lower-left of the dialog box, click **More** twice to create two additional runs.
5. Select **Flow\_RunPhysOpt** as the Strategy for the **impl\_3** run.
6. Select **Flow\_RuntimeOptimized** as the Strategy for the **impl\_4** run.

The Configure Implementation Runs dialog box now displays three new implementations along with the strategy you selected for each, as seen in [Figure 4](#).

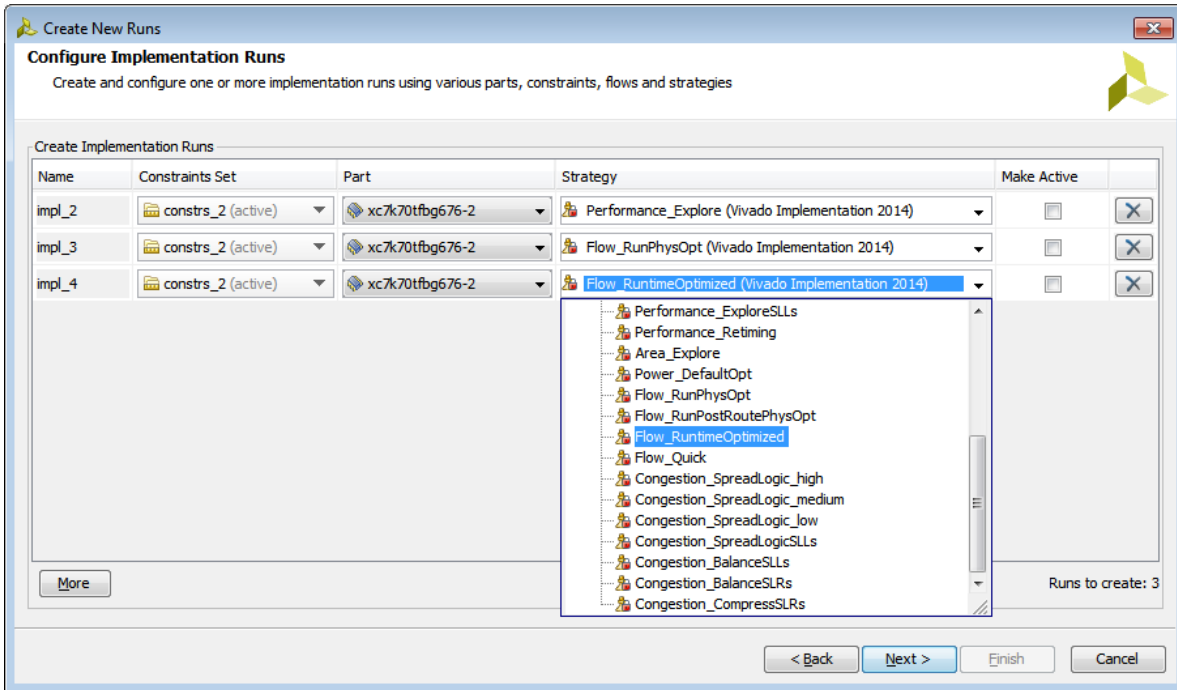


Figure 4: Configure Implementation Runs

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

## Step 3: Analyzing Implementation Results

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

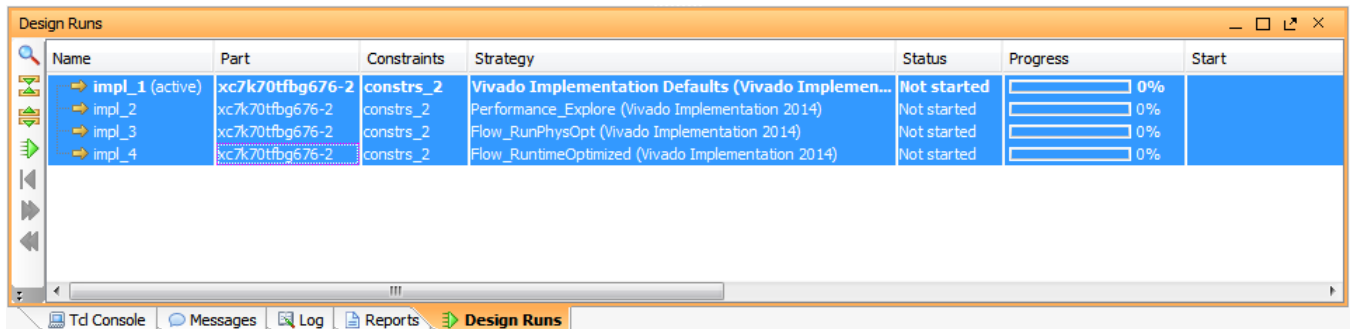

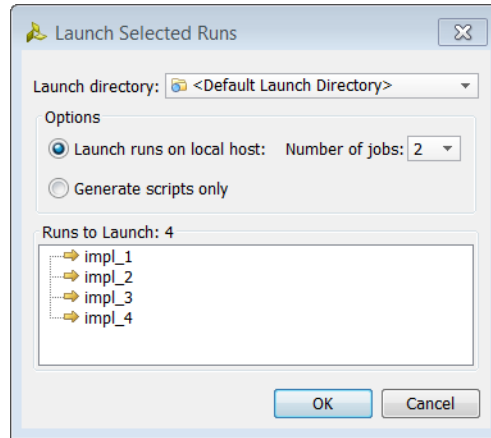


Figure 5: Design Runs Window

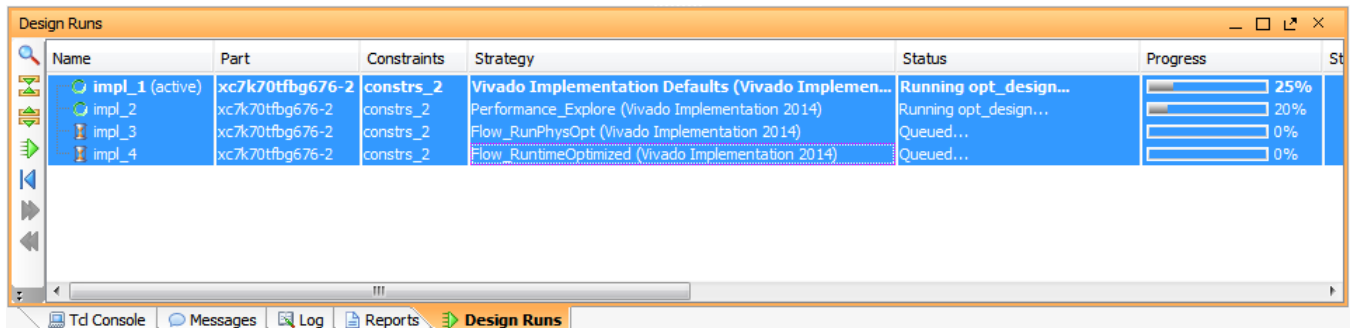
- On the sidebar menu, click **Launch Selected Runs**, .
- In the Launch Selected Runs dialog box, select **Launch runs on local host** and **Number of jobs: 2**, as shown below.



**Figure 6: Launch Selected Runs**

- Click **OK**.

Two runs launch simultaneously. The remaining runs are placed into a queue.

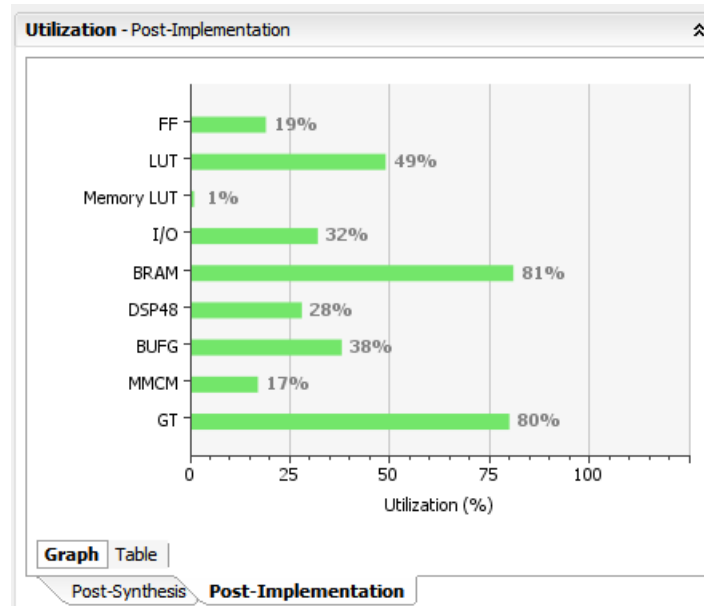


**Figure 7: Two Implementation Runs in Progress**

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.

- Click **Cancel** to close the dialog box.
- At the bottom of the Project Summary, click the **Post-Implementation** tab of the Utilization section.

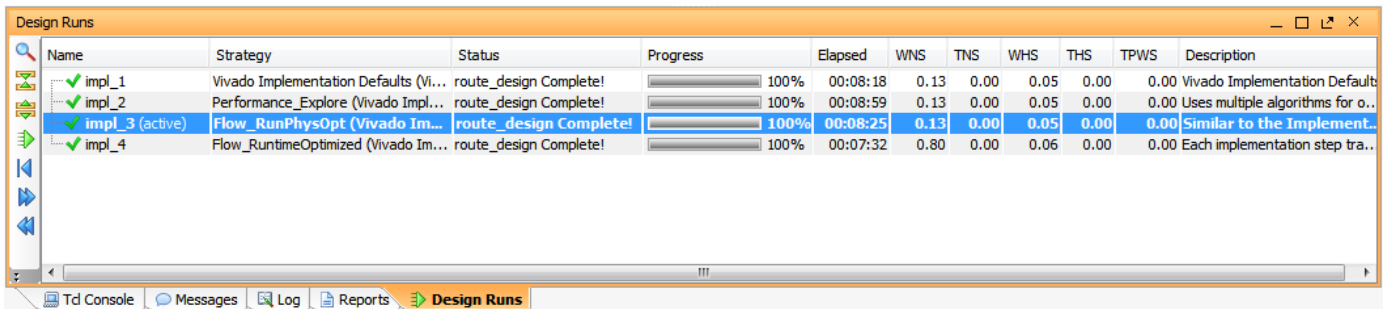
The implementation utilization results display as a bar graph, as shown in [Figure 8](#).



**Figure 8: Post-Implementation Utilization**

- When all the implementation runs are complete, 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 9: Compare Implementation Results**

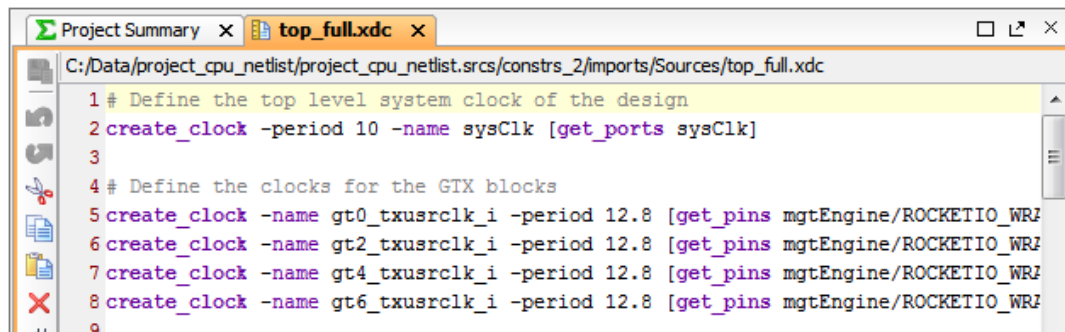
- Compare the results for the completed runs in the Design Runs window, as seen in [Figure 9](#).
  - The **Flow\_RuntimeOptimized** strategy in **impl\_4** completed in the least time, as seen in the **Elapsed** time column.
  - All runs met the timing requirements as seen in the **WNS** column.

## 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.



```

1 # Define the top level system clock of the design
2 create_clock -period 10 -name sysClk [get_ports sysClk]
3
4 # Define the clocks for the GTX blocks
5 create_clock -name gt0_txusrclk_i -period 12.8 [get_pins mgtEngine/ROCKETIO_WRF
6 create_clock -name gt2_txusrclk_i -period 12.8 [get_pins mgtEngine/ROCKETIO_WRF
7 create_clock -name gt4_txusrclk_i -period 12.8 [get_pins mgtEngine/ROCKETIO_WRF
8 create_clock -name gt6_txusrclk_i -period 12.8 [get_pins mgtEngine/ROCKETIO_WRF
9
  
```

Figure 10: `top_full.xdc` File

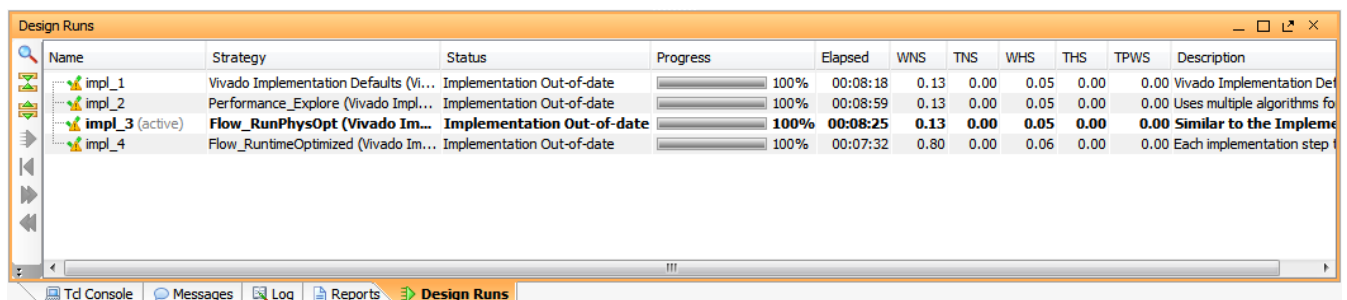
2. On line 2, change the period of the `create_clock` constraint from 10 ns to 7.45 ns.

The new constraint should read as follows:

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


3. Save the changes by clicking **Save File**, , 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.




Name	Strategy	Status	Progress	Elapsed	WNS	TNS	WHS	THS	TPWS	Description
impl_1	Vivado Implementation Defaults (Vi...	Implementation Out-of-date	100%	00:08:18	0.13	0.00	0.05	0.00	0.00	Vivado Implementation Def
impl_2	Performance_Explore (Vivado Impl...	Implementation Out-of-date	100%	00:08:59	0.13	0.00	0.05	0.00	0.00	Uses multiple algorithms fo
impl_3 (active)	Flow_RunPhysOpt (Vivado Im...	Implementation Out-of-date	100%	00:08:25	0.13	0.00	0.05	0.00	0.00	Similar to the Impleme
impl_4	Flow_RuntimeOptimized (Vivado Im...	Implementation Out-of-date	100%	00:07:32	0.80	0.00	0.06	0.00	0.00	Each implementation step t

Figure 11: Implementation Runs Out-of-Date

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

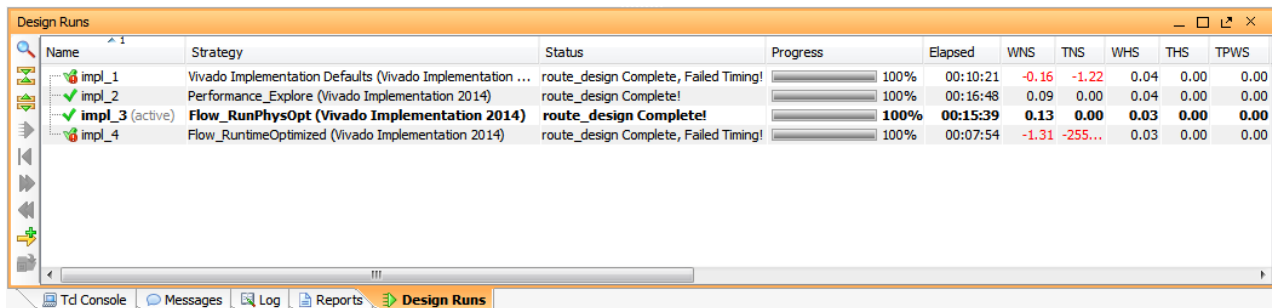
- With all runs selected in the Design Runs window, click **Launch Selected Runs**, .

The Launch Selected Runs window opens.

- 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.

- Click **Cancel** to close the dialog box.
- Compare the **Elapsed** time for each run in the Design Runs window, as seen in [Figure 12](#).
  - 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 greater negative slack.



Name	Strategy	Status	Progress	Elapsed	WNS	TNS	WHS	THS	TPWS
impl_1	Vivado Implementation Defaults (Vivado Implementation ...)	route_design Complete, Failed Timing!	100%	00:10:21	-0.16	-1.22	0.04	0.00	0.00
impl_2	Performance_Explore (Vivado Implementation 2014)	route_design Complete!	100%	00:16:48	0.09	0.00	0.04	0.00	0.00
impl_3 (active)	Flow_RunPhysOpt (Vivado Implementation 2014)	route_design Complete!	100%	00:15:39	0.13	0.00	0.03	0.00	0.00
impl_4	Flow_RuntimeOptimized (Vivado Implementation 2014)	route_design Complete, Failed Timing!	100%	00:07:54	-1.31	-255...	0.03	0.00	0.00

**Figure 12: Implementation Results**



**RECOMMENDED:** Reserve the Performance\_Explore strategy for designs that have challenging timing constraints. When timing is easily met, the Performance\_Explore strategy will increase implementation times while providing no timing benefit.

## Conclusion

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.

In this lab you have 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 that those strategies might be needed in a more challenging design.

## Lab #2: Using Incremental Compile

### Introduction

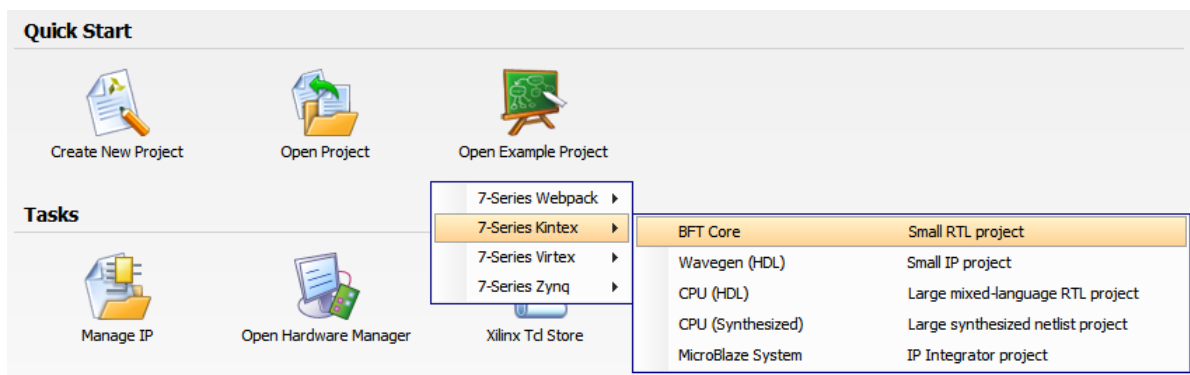
Incremental Compile is an advanced design flow to use on designs that are nearing completion, where small changes are required.

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 will 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

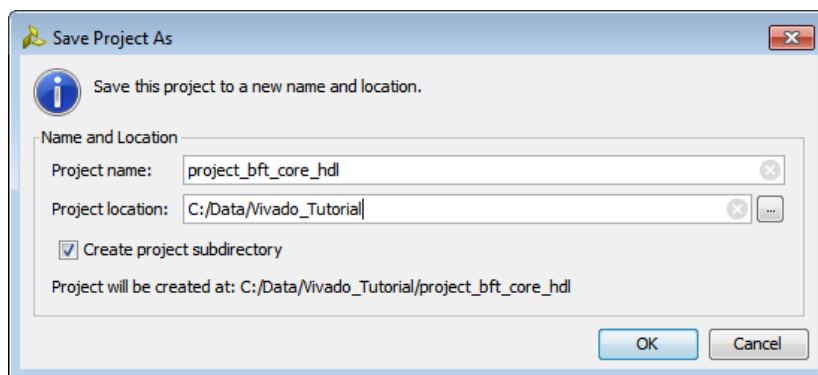
1. Start by loading Vivado IDE by doing one of the following:
  - Launch the Vivado IDE from the icon on the Windows desktop
  - Type `vivado` from a command terminal.
2. From the Getting Started page, click **Open Example Project** and select the **BFT Core** design.



**Figure 13: Open Example Design**

A dialog box appears stating that the Project is Read-Only.

3. Click **Save Project As** to specify a project name and location.



**Figure 14: Save Project As**

4. Specify the following, and click OK :
  - Project name: `project_bft_core_hdl`
  - Project location: `<Extract_Dir>`

The Vivado IDE opens with the default view.

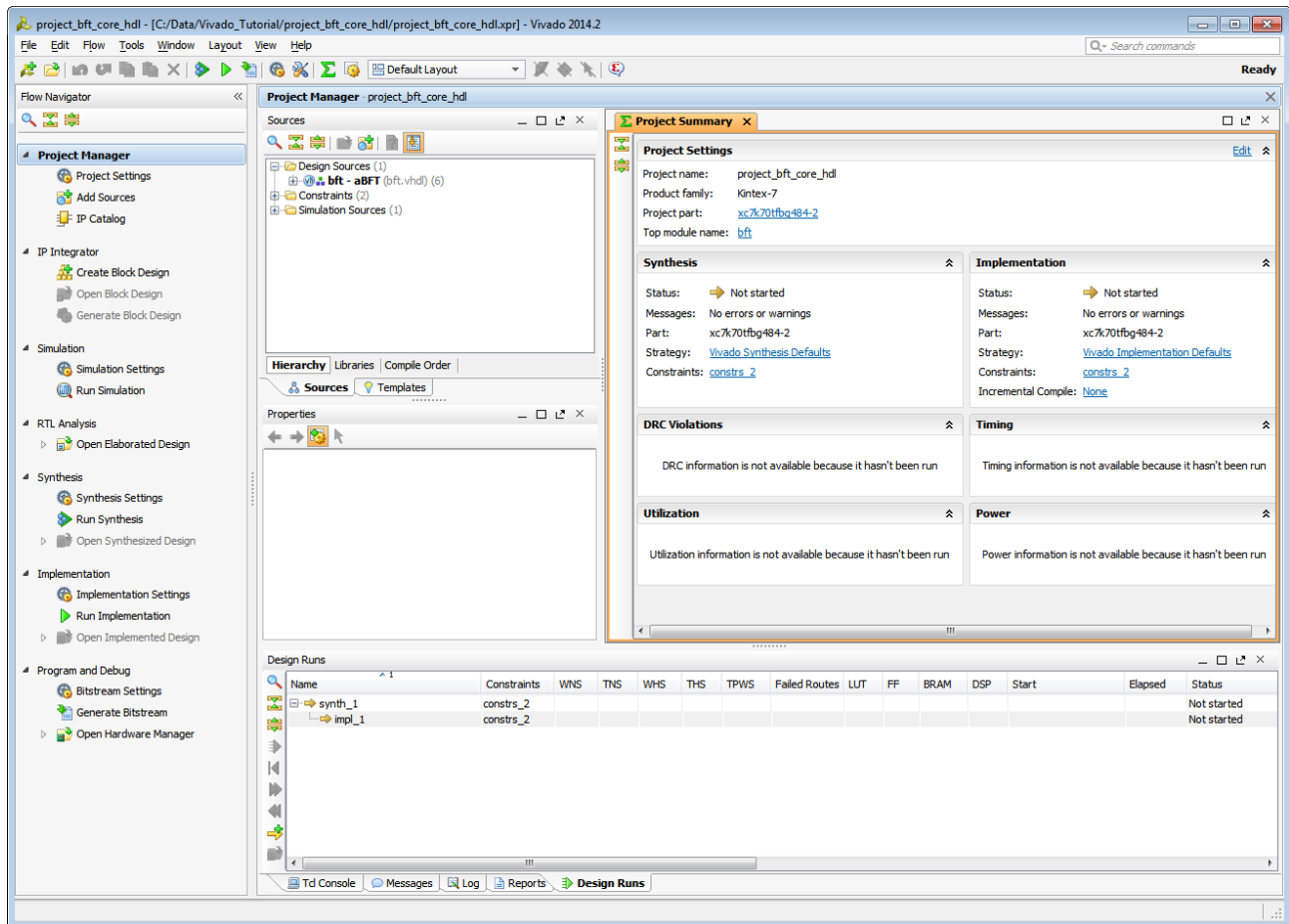


Figure 15: BFT Project Summary

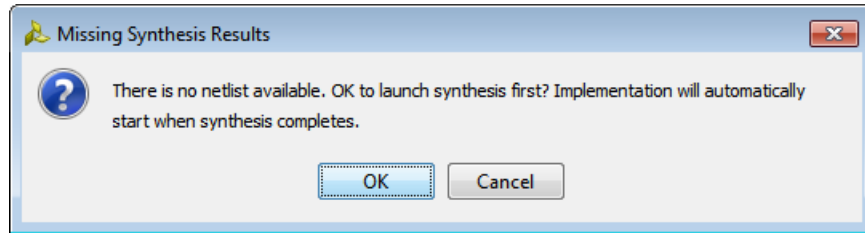
## Step 2: Compiling the Reference Design

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

The Missing Synthesis Results dialog box opens, as shown in Figure 16. 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 16: 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.

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 [Figure 17](#). 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

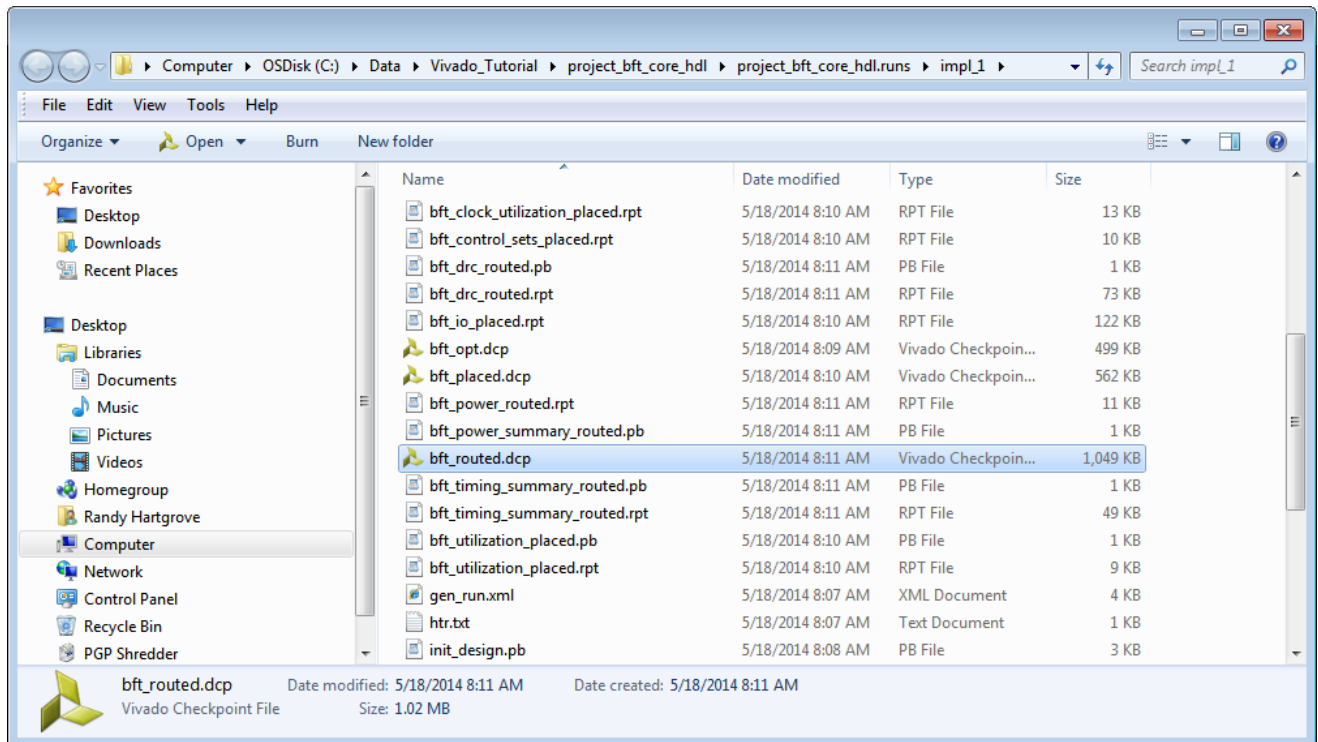


Figure 17: Implementation Run Directory

## Step 3: Create New Runs

In this step, you define new synthesis and implementation runs to preserve the results of the current runs. In the next step, you will be making changes to the design and rerunning synthesis and implementation. If you do not create new runs, the current results would be overwritten.

1. From the main menu, select **Flow > Create Runs**.

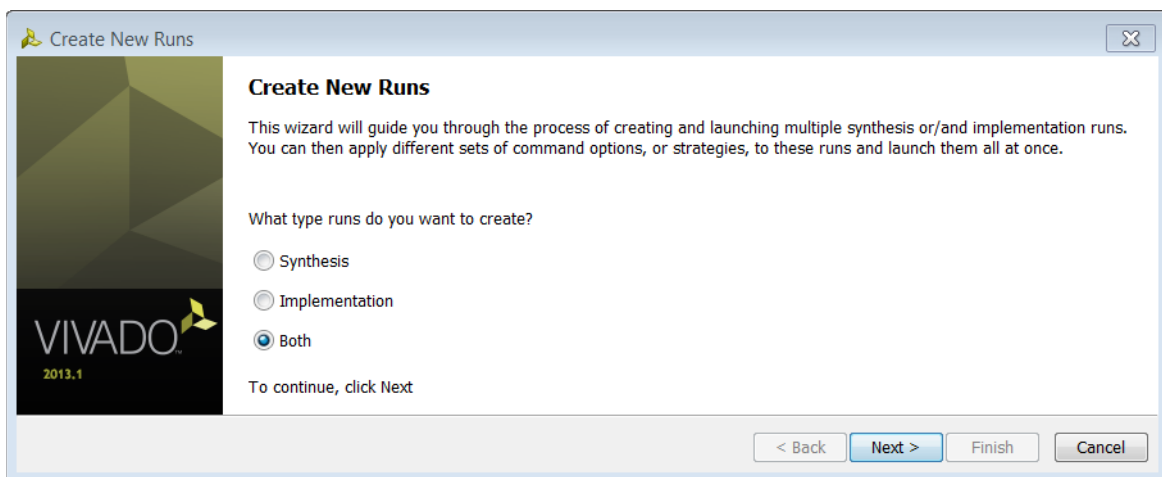


Figure 18: Create New Runs Wizard

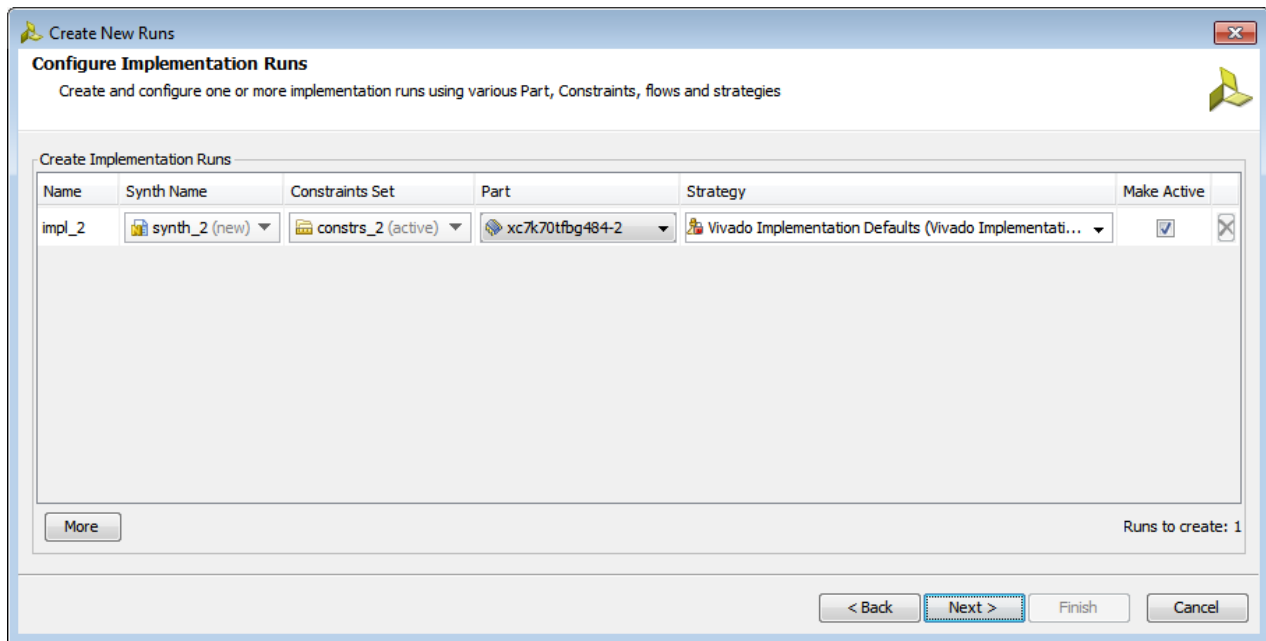
The Create New Runs dialog box opens, prompting you to create new Synthesis, or Implementation runs, or to create both.

2. Select **Both** and click **Next**.

The Configure Synthesis Runs window opens.

3. Accept the default run options by clicking **Next**.

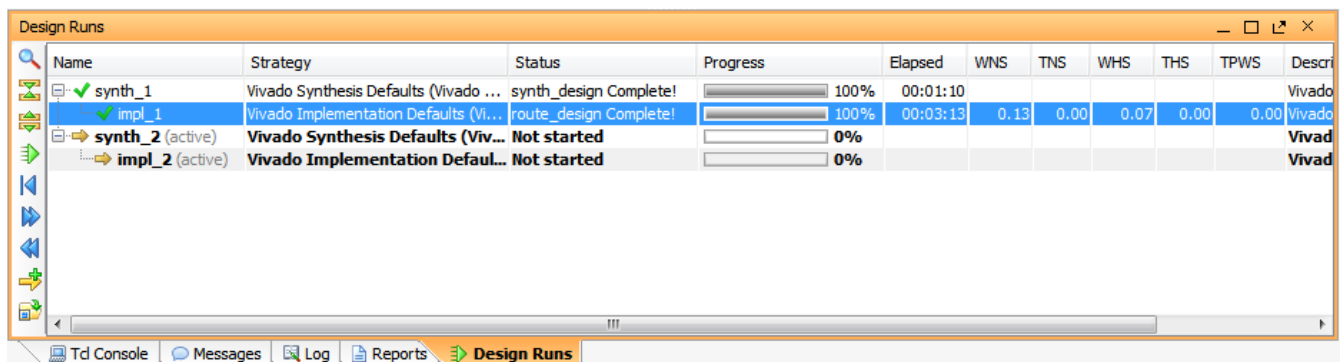
The Configure Implementation Runs window opens, as shown in [Figure 19](#).



**Figure 19: Configure Implementation Runs**

4. Enable the **Make Active** checkbox, and click **Next**.
5. From the Launch Options window, select **Do not launch now** and click **Next**.
6. In the Create New Runs Summary dialog box, click **Finish** to create the new runs.

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



**Figure 20: New Design Runs**

## Step 4: Making Incremental Changes

In this step, you will make minor changes to the RTL design sources. These changes will necessitate resynthesizing the netlist, and then 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.

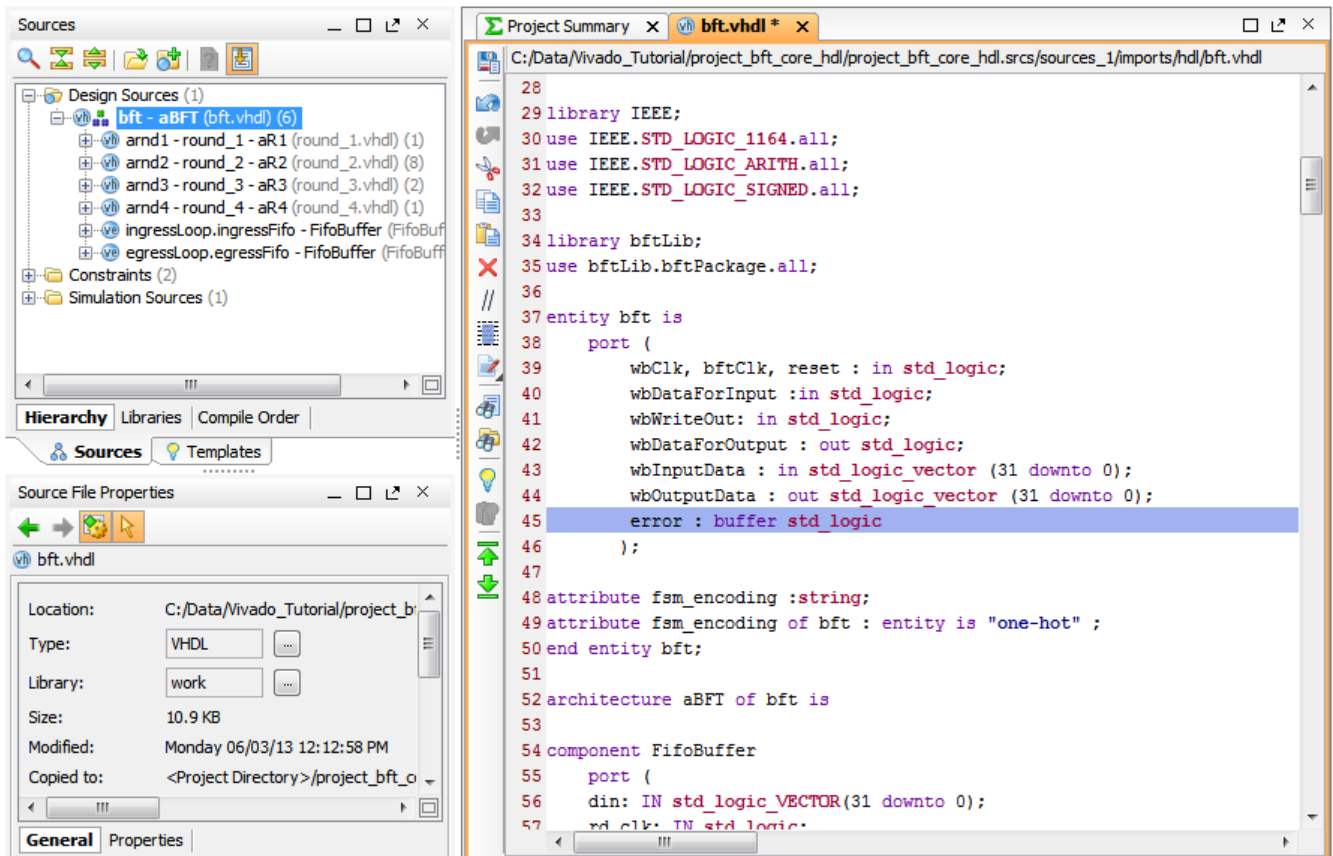


Figure 21: BFT VHDL File

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
<pre>-- enable the read fifo process (fifoSelect) begin     readEgressFifo &lt;= fifoSelect; end process;</pre>	<pre>-- enable the read fifo process (fifoSelect, error) begin     if (error = '0') then         readEgressFifo &lt;= fifoSelect;     else         readEgressFifo &lt;= (others =&gt; '0');     end if; end process;</pre>

The results should look like Figure 22.

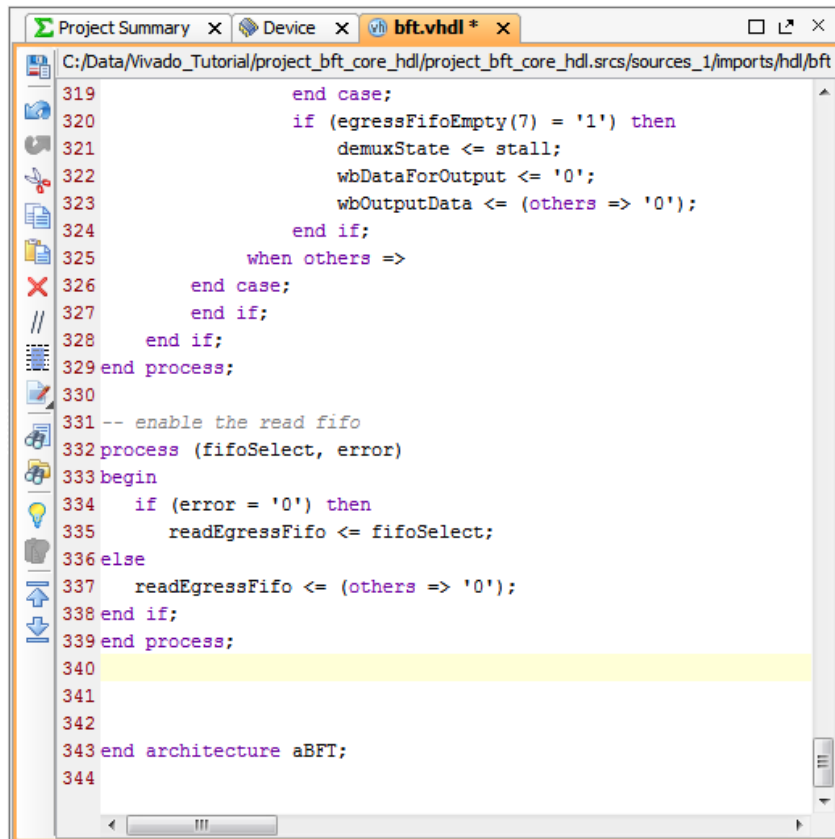


Figure 22: Modified VHDL Code

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

As you can see in Figure 23, changing the design source files also changes the run status for finished runs from **Complete** to **Out-of-date**.

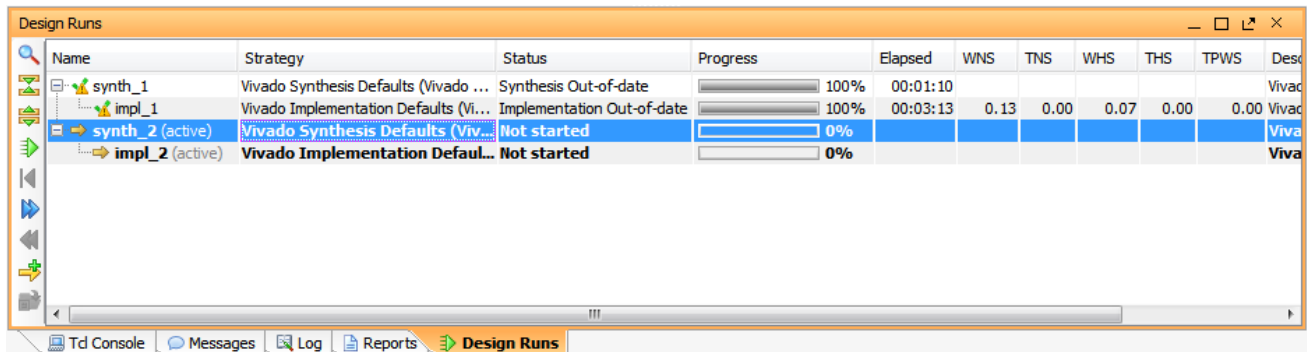


Figure 23: Design Runs Out-of-Date

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, select **Cancel** to close the dialog box.


## Step 5: Running Incremental Compile

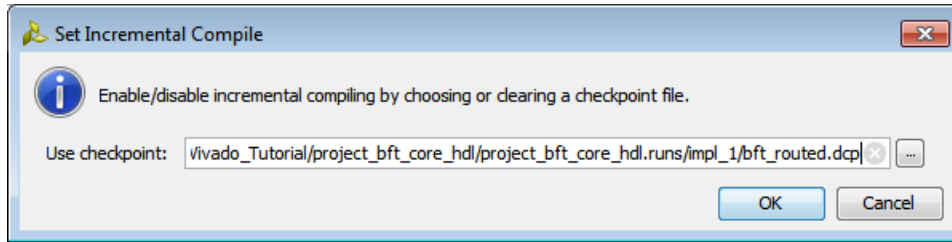
The design has now been updated with a few minor changes, necessitating re-synthesis. You could run implementation on the new netlist, to place and route the design and work to meet the timing requirements. However, with only minor changes between this iteration and the last, the incremental compile 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.

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 24](#).

2. Click **Browse**, , 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 24: Set Incremental Compile**

4. Click **OK**.

This is stored in the `INCREMENTAL_CHECKPOINT` property of the selected run. With this property set, the Vivado Design Suite will know 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 Figure 25, the Design Runs window shows that the elapsed time for implementation run `impl_2` is less than 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.



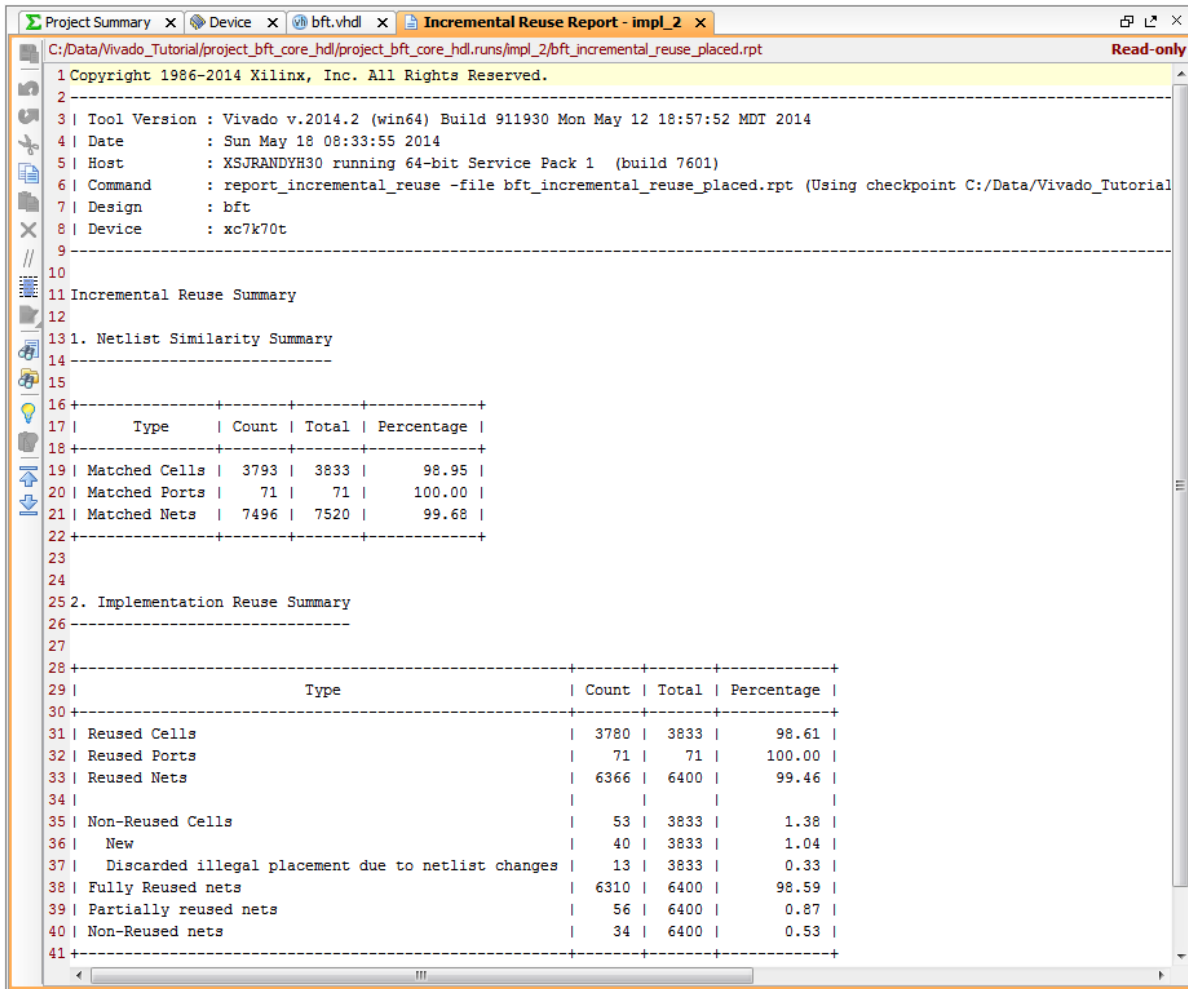


Figure 27: Incremental Reuse Report

In the report, fully reused nets indicate that the entire net’s routing is reused from the reference design. Partially reused nets indicate that some of the net’s routing from the reference design is reused. Some segments are re-routed 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 have learned how to run the Incremental Compile flow, using a checkpoint from a previously implemented design. You have also examined the similarity between a reference design checkpoint and the current design by examining the Incremental Reuse Report

# Lab #3: Manual and Directed Routing

## Introduction

In this lab, you will 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** and select the **7 Series Kintex > BFT Core** design.

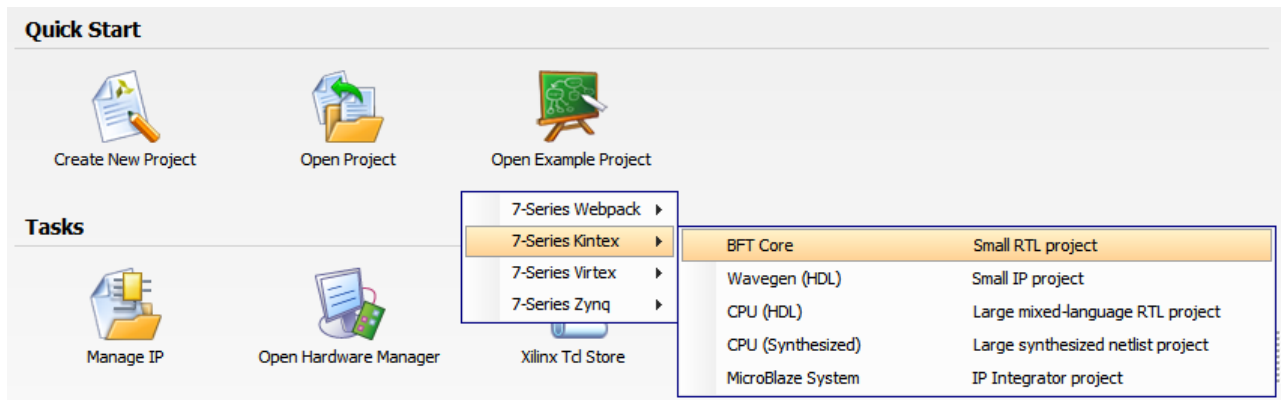


Figure 28: Open Example Design

A dialog box appears stating that the Project is Read-Only.

3. Click **Save Project As** to specify a project name and location.

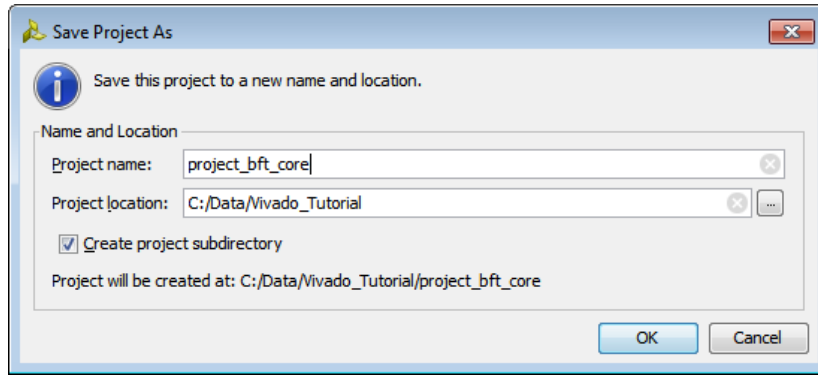


Figure 29: Save Project As

4. Specify the following, and click **OK** :

- Project name: **project\_bft\_core**
- Project location: <Extract\_Dir>

The Vivado IDE opens with the default view.

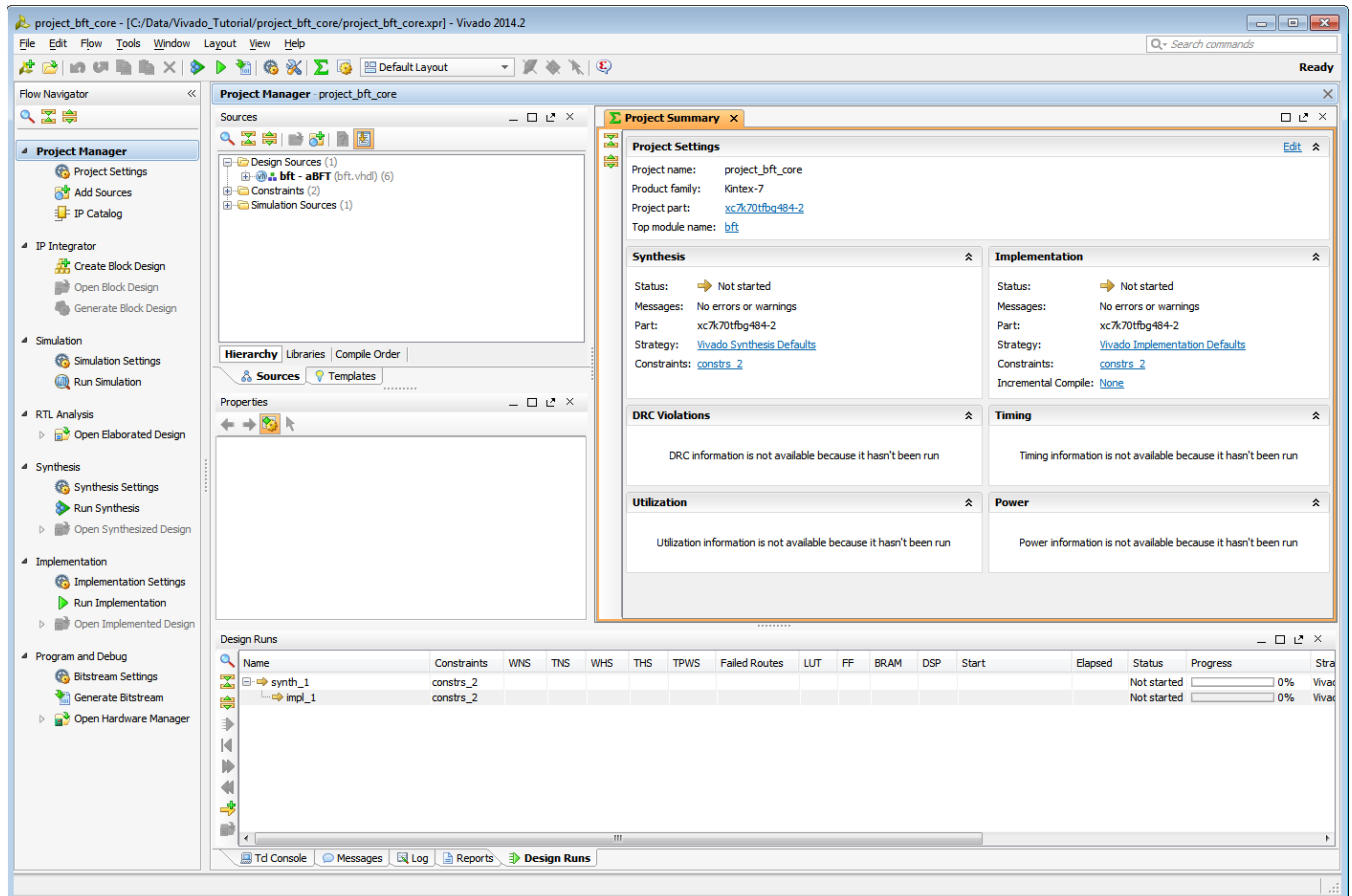
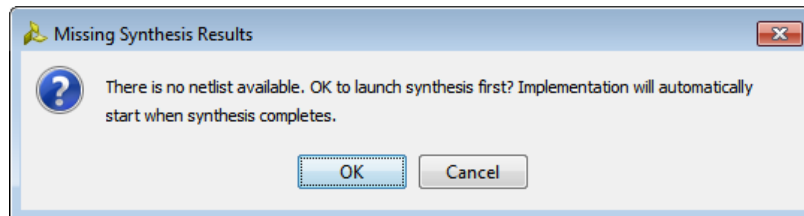


Figure 30: BFT Core Project Summary

## Step 2: Place and Route the Design

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

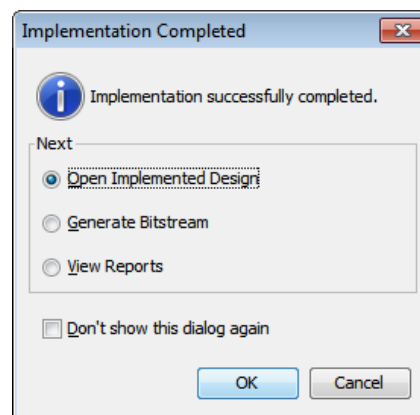
The Missing Synthesis Results dialog box opens to inform you that there is no synthesized netlist to implement. The Vivado Design Suite prompts you to start synthesis first.



**Figure 31: 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.



**Figure 32: Implementation Completed**

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

The Device window opens, and the results of placement are displayed.

4. Enable the **Routing Resources** command, , to view the detailed routing resources in the Device window.

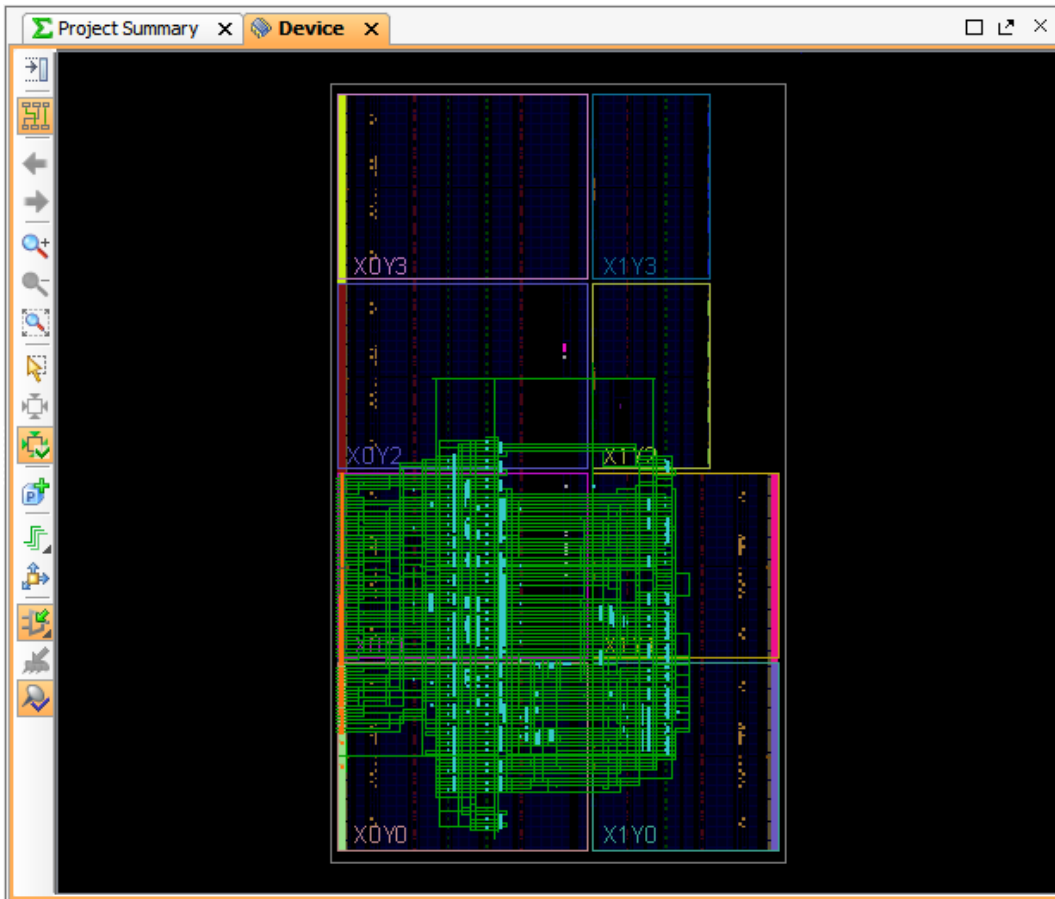


Figure 33: Device View

## Step 3: Analyze Output Bus Timing



**IMPORTANT:** The tutorial design has an output data bus, `wbOutputData`, that 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. Select Tools > Timing > Report Datasheet from the main menu.
2. Select the **Groups** tab in the Report Datasheet dialog box, as seen in Figure 34, and enter the following:
  - Reference: `[get_ports {wbOutputData[0]}]`
  - Ports: `[get_ports {wbOutputData[*]}]`

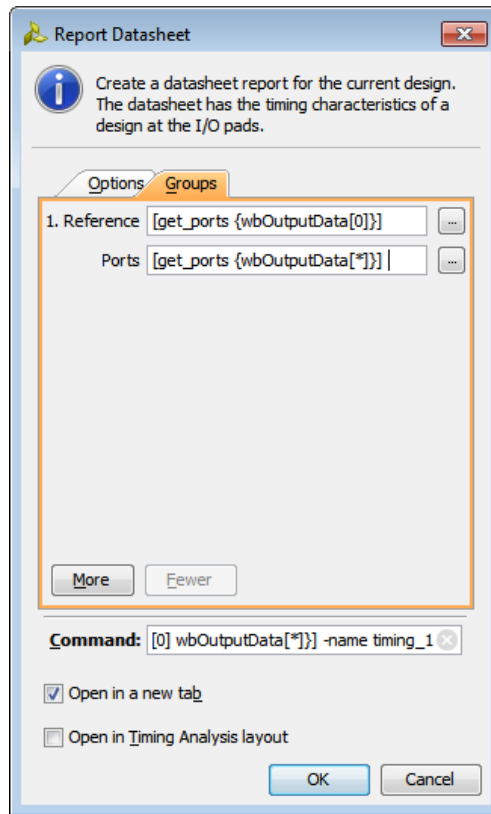



Figure 34: Report DataSheet

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. Maximize, , the **Timing - Datasheet** window to expand the results.
5. Select the **Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0]** section, as seen in Figure 35.

You can see from the report that the timing skew across the `wbOutputData` bus varies by almost 400 ps. The goal is to minimize the skew across the bus to less than 100 ps.

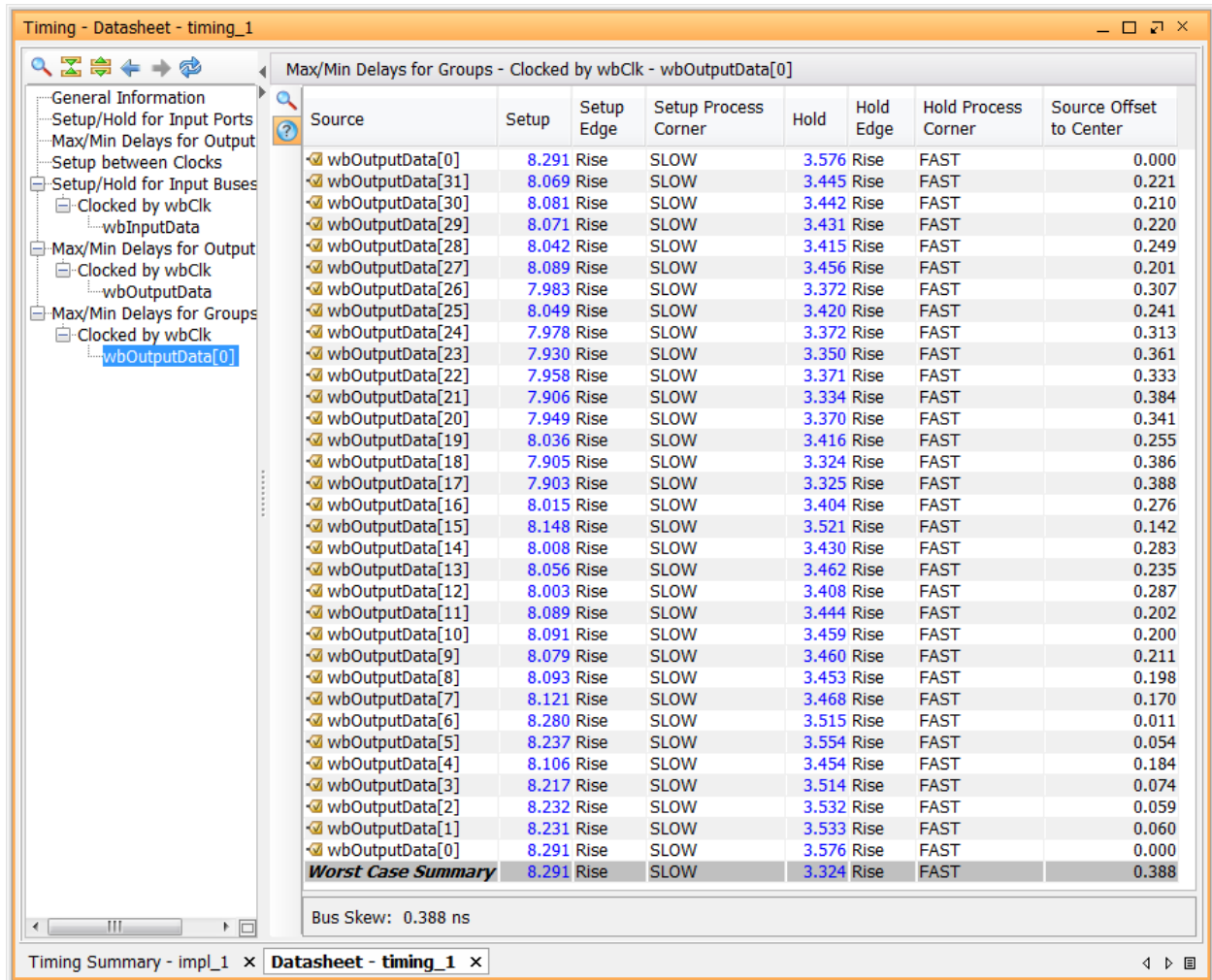




Figure 35: Report Datasheet Results

- Minimize the **Timing – Datasheet** window, , so you can simultaneously see the Device window and the Timing - Datasheet results.
- 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** command, , is highlighted in the Device window so you can see the entire path, as shown in Figure 36.

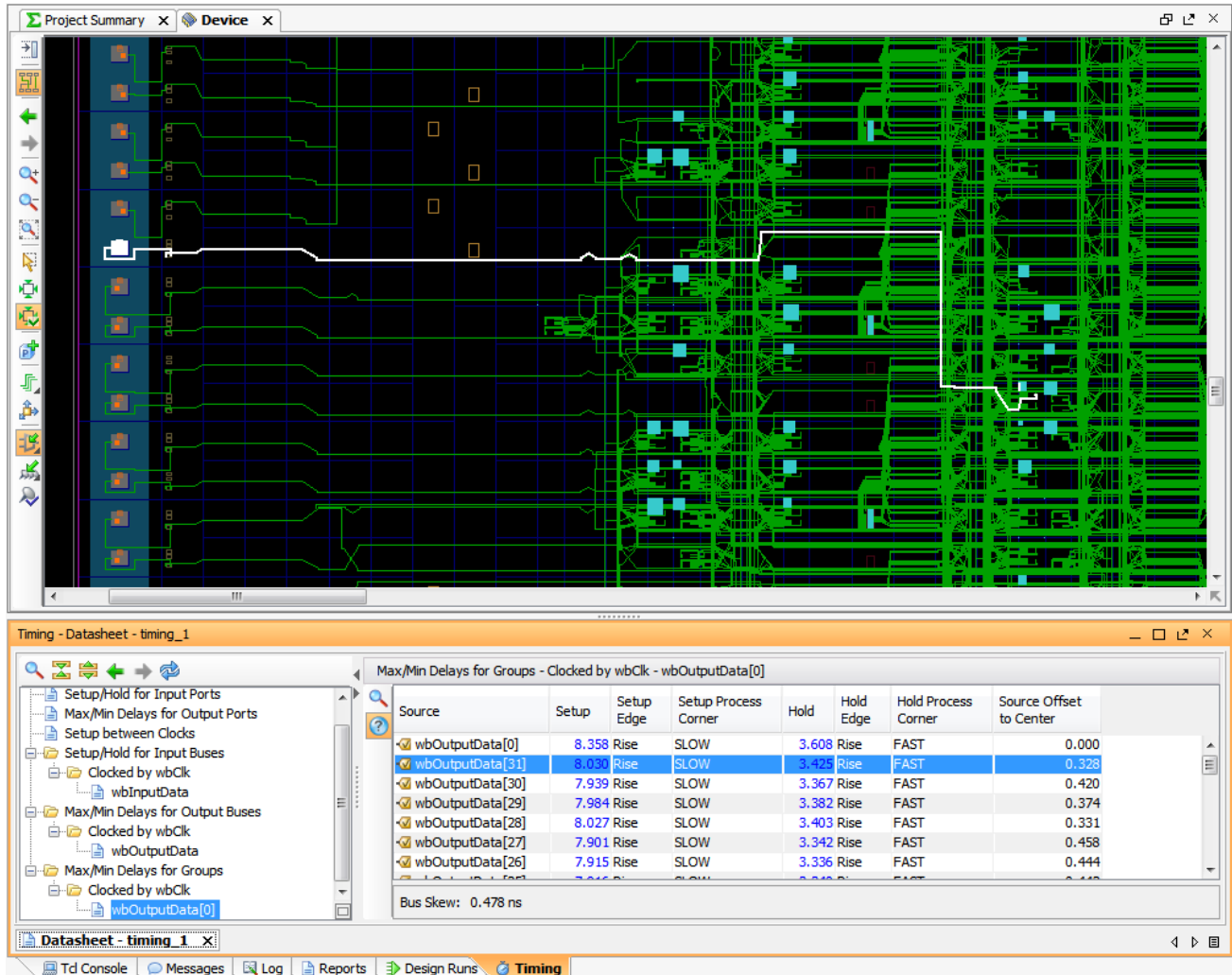


Figure 36: Detailed Routing View

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

The schematic for the selected output data bus is displayed as shown in [Figure 37](#). 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.

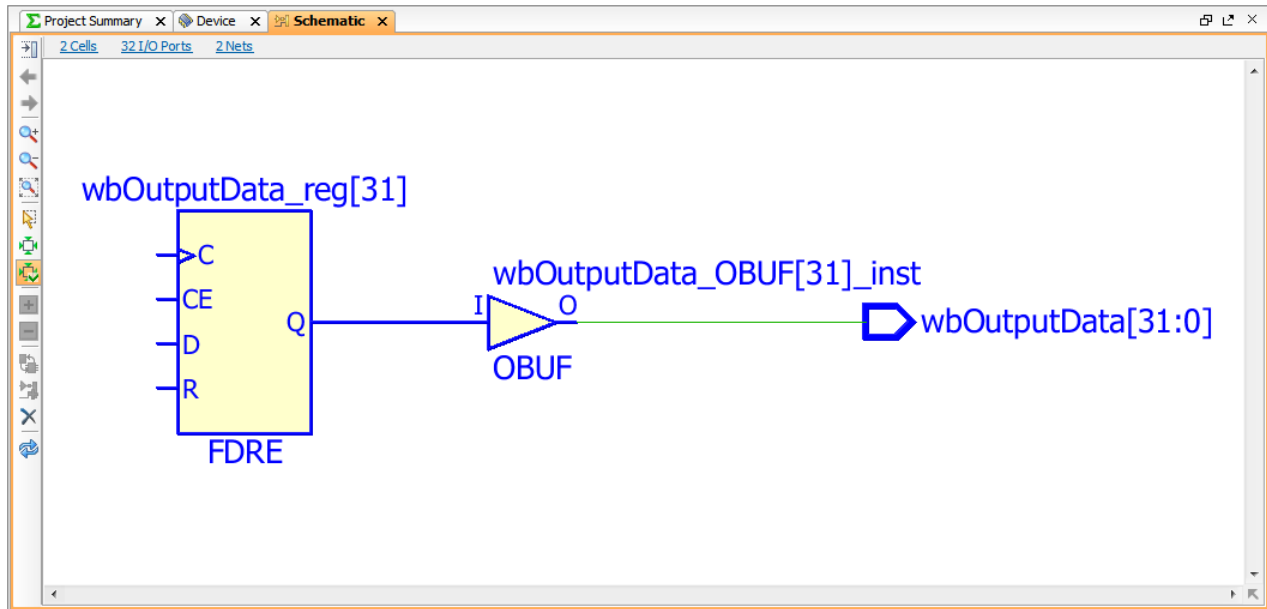


Figure 37: Schematic view of Output Path

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 appear on the output ports, and red diamond markers appear on the registers feeding the outputs.



Figure 38: Marked Routes

Zooming out on the Device window displays a picture similar to [Figure 38](#).

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 39](#).

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. If the output registers were more consistently placed in the slices next to each output port, you could eliminate a majority of the variability in the clock-to-out delay of the `wbOutputData` bus.

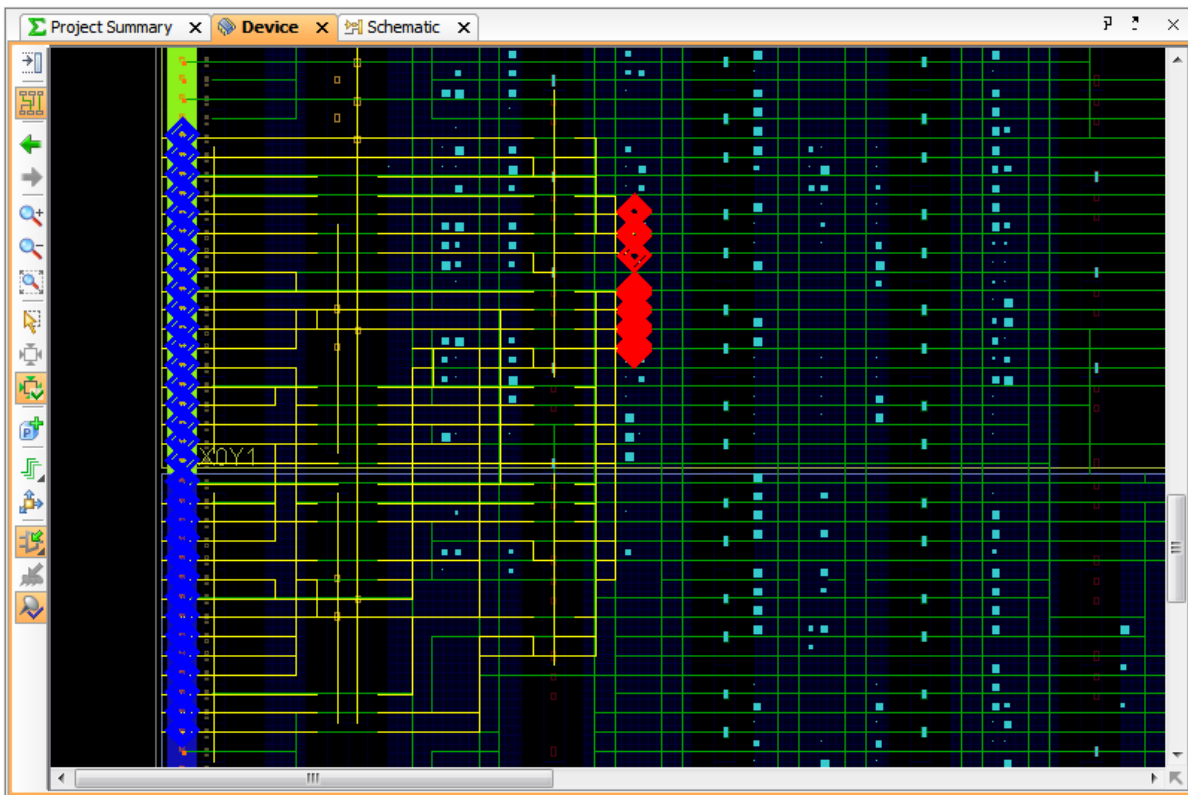




Figure 39: Highlighted Routes



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

## Step 4: Improve 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 40. By examining the device resources around the selected ports, you can identify a range of placement Sites for the output registers.

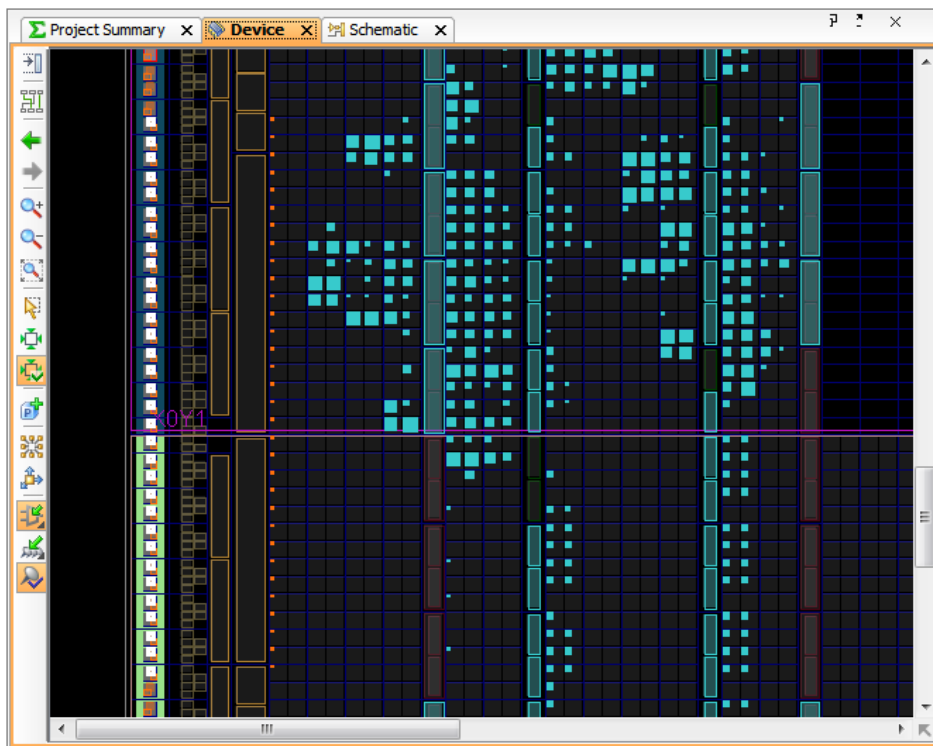
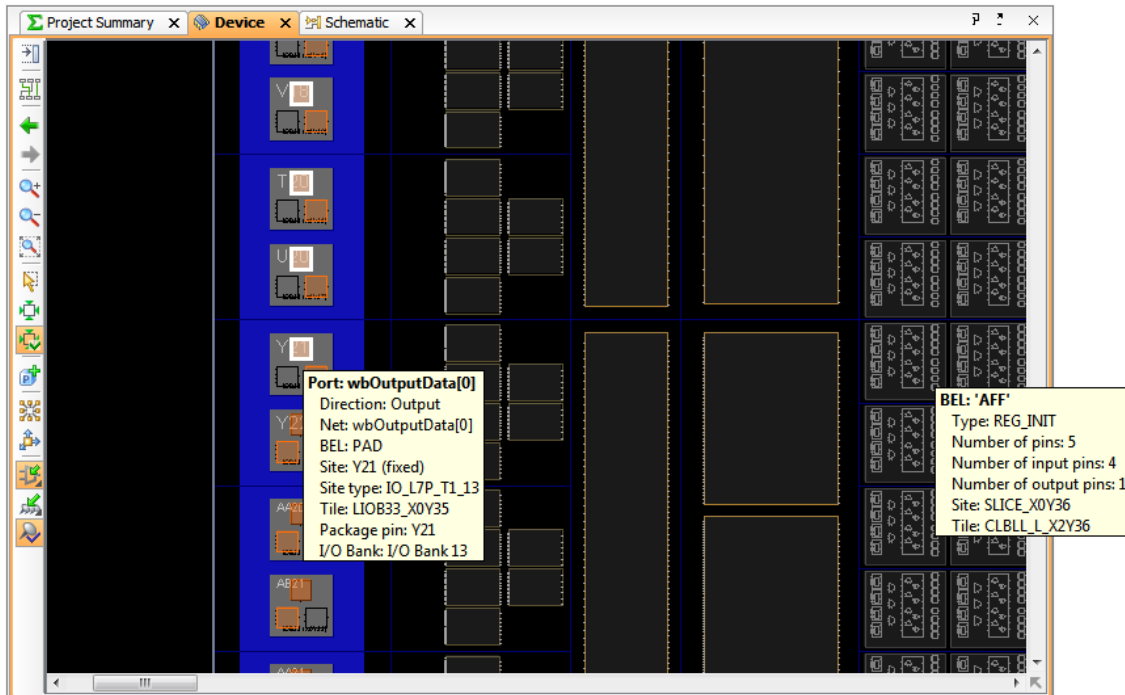


Figure 40: Selected `wbOutputData` Ports

3. Zoom into the Device window around the bottom selected output ports.



**Figure 41: wbOutputData[0] Placement Details**

The bottom ports are the lowest bits of the output bus, starting with `wbOutputData[0]`. As seen in [Figure 41](#), 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.

- 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.

- 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
}
```

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

```
place_design
```

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

- 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[*]]]
```


- Then route any currently unrouted nets in the design:

```
route_design
```

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

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

```
report_route_status
```

- Enable the **Routing Resources** command, , to view the detailed routing resources in the Device window.

- 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]]
```

- In the Device window, **zoom** into some of the **marked output ports**.

- 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 42](#), 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.

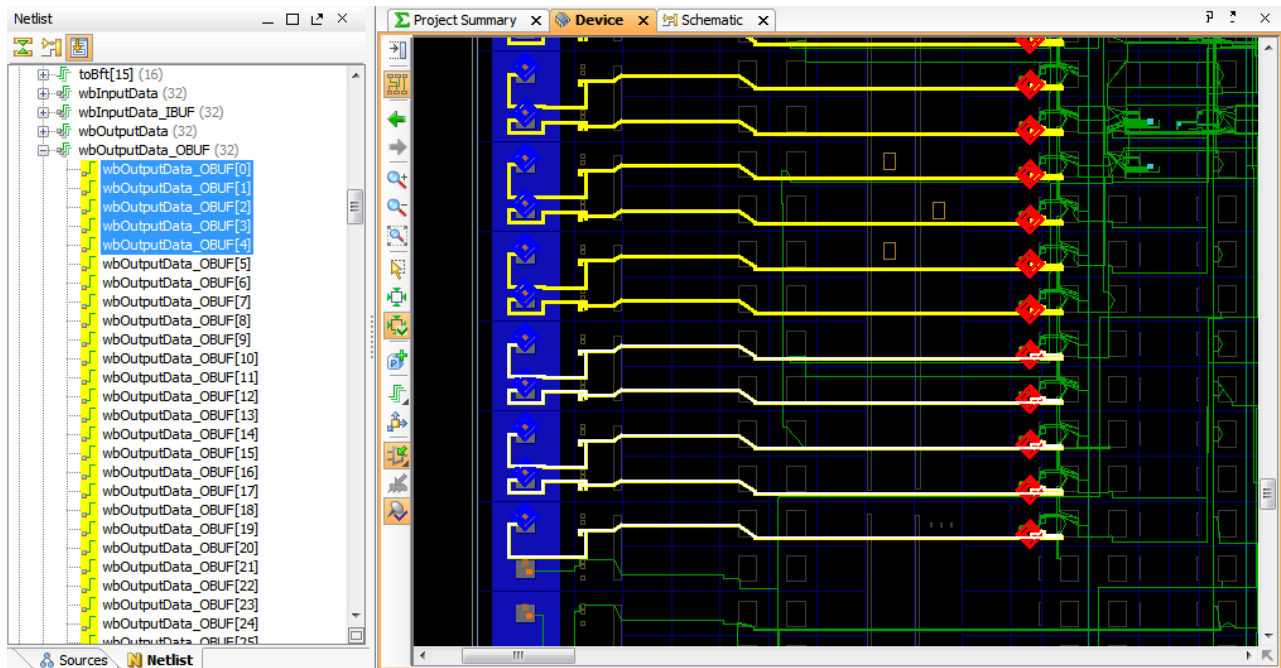


Figure 42: Improved Placement and Routing

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, as seen in [Figure 43](#), you can see that timing skew within the lower bits of the output bus, `wbOutputData[0-13]`, is closely matched. As is the skew within the upper bits, `wbOutputData[14-31]`. However, although the overall skew has been reduced, between the upper and lower bits the skew is still over 200 ps.

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, you will also notice that the Setup delay is greater on the lower bits than it is on the upper bits. To reduce the skew, you need to add delay to the upper bits.

You could 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.

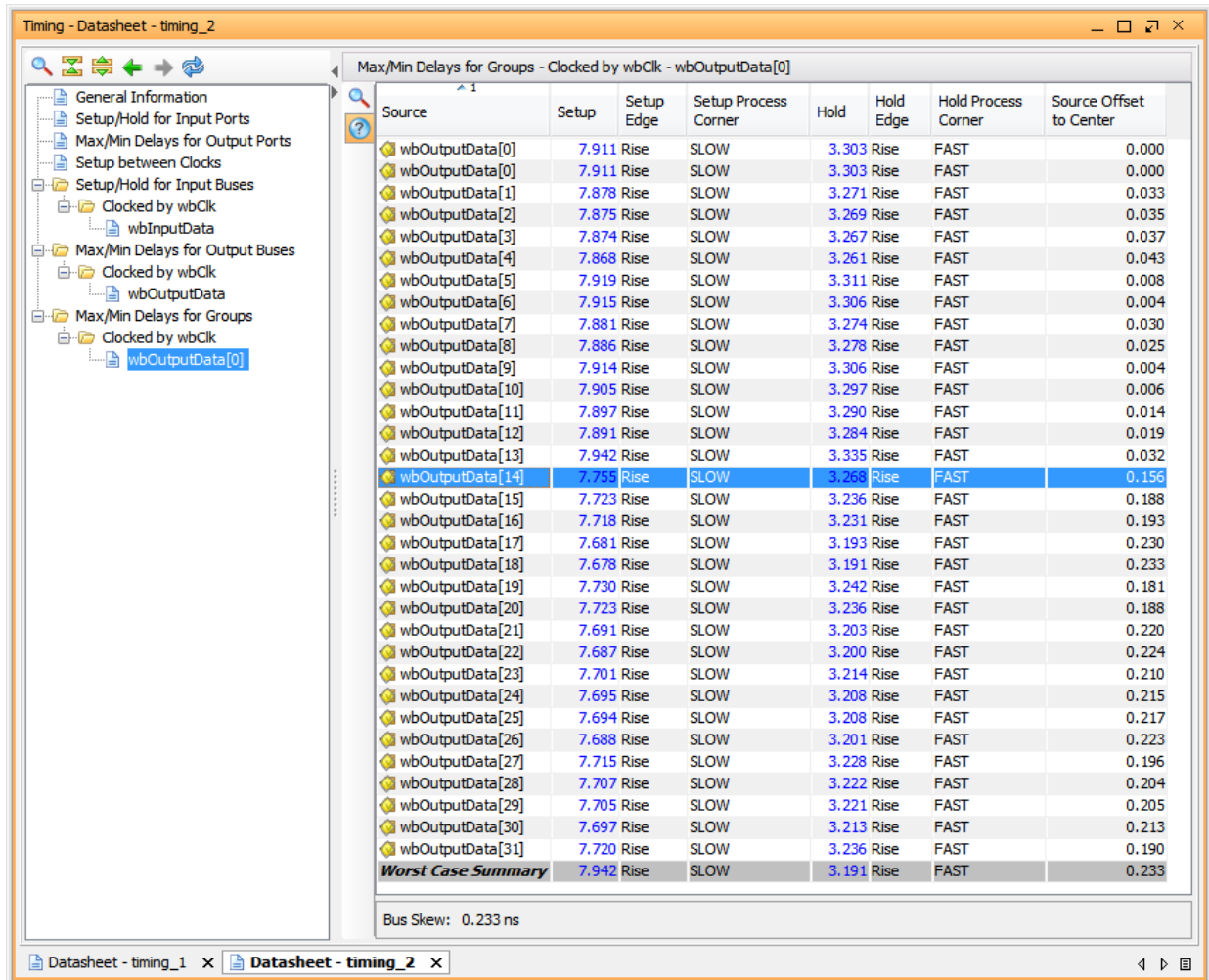


Figure 43: Report Datasheet – Improved Skew

## 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 you can see 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 should be able to 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.


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

```

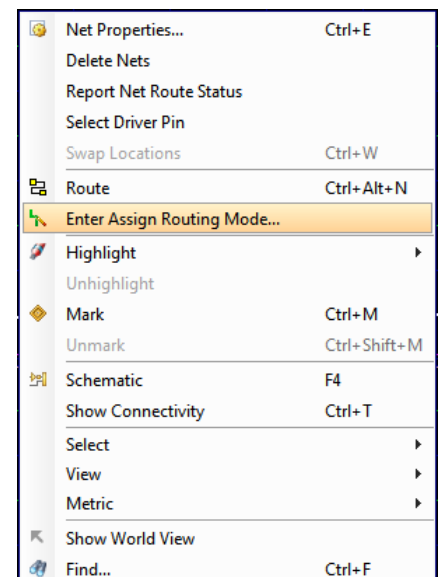
select_objects [get_nets -of [get_pins -of \
[get_cells wbOutputData_reg[14]] -filter DIRECTION==OUT]]
    
```

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

- Maximize**, , the Device window.
- Right-click the net, and select **Enter Assign Routing Mode**.

The Target Load Cell Pin dialog box opens, as seen in Figure 44, to let you select a load pin to route to or from. In this case, only one load pin is provided: `wbOutputData_OBUF[14]_inst`.



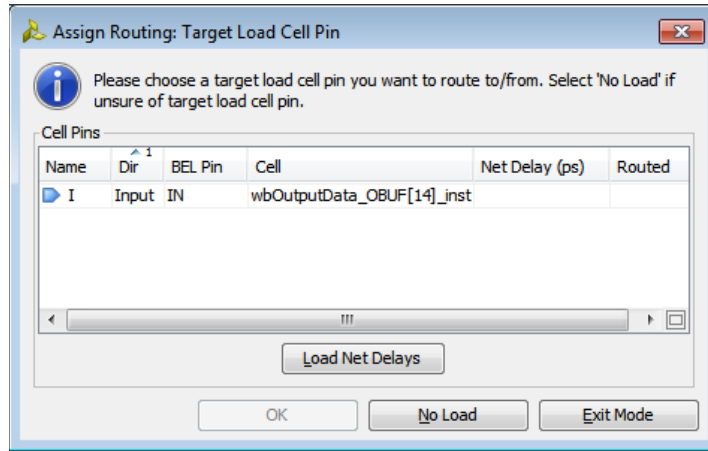


Figure 44: Target Load Cell Pin Dialog Box

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

The Vivado IDE enters into Assign Routing mode, and a new Routing Assignment window is displayed on the right side of the Device window.

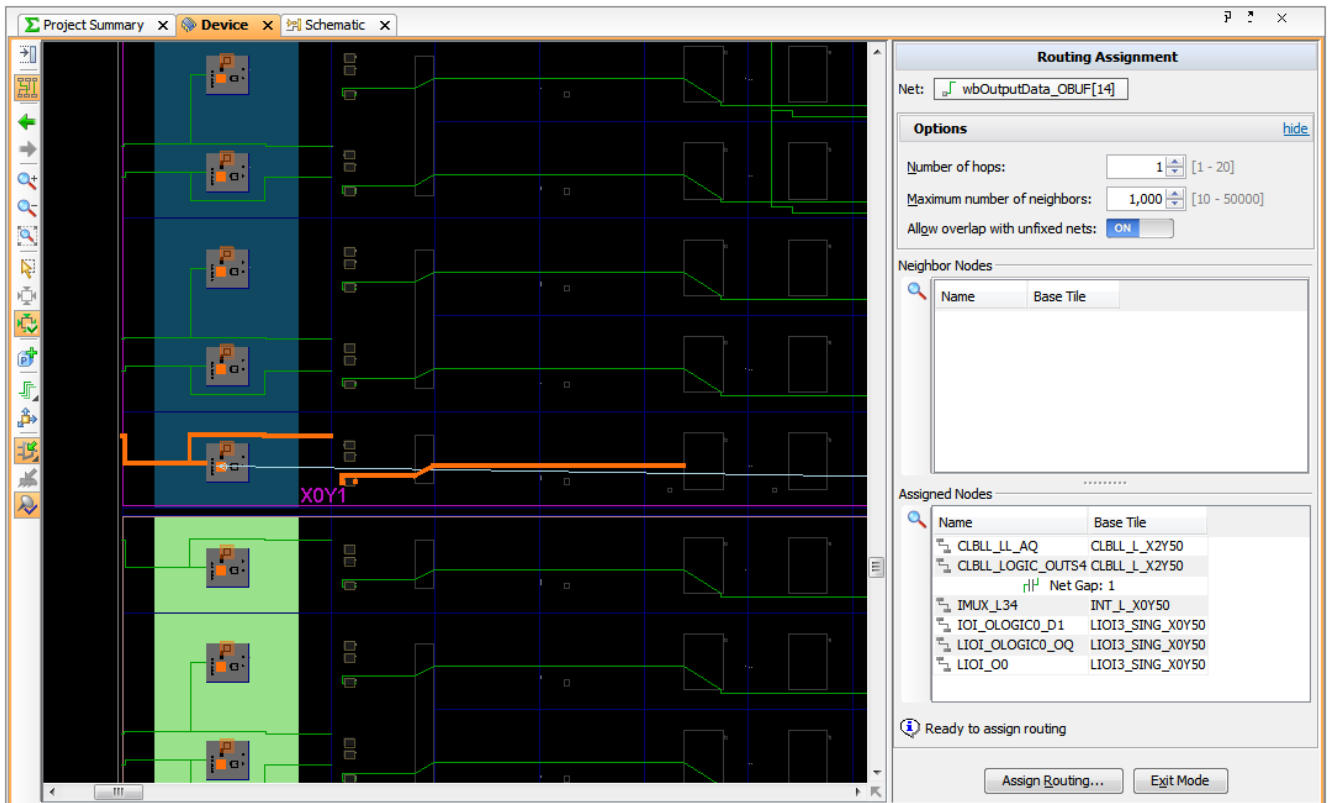


Figure 45: Assign Routing Mode

The Routing Assignment window includes the following sections, as seen in Figure 45:

- **Net** – Displays the current net being routed.
- **Options** – Are hidden by default, and can be displayed by clicking [show](#).
  - **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 45](#), 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 are displayed 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. When the node is selected, possible neighbor nodes are displayed in the Neighbor Nodes section of the Routing Assignment window, and are also displayed 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 are displayed as shown in Figure 46. To add delay to compensate for the clock skew, you must select a neighbor node that provides a slight detour over the more direct route previously chosen by the router.

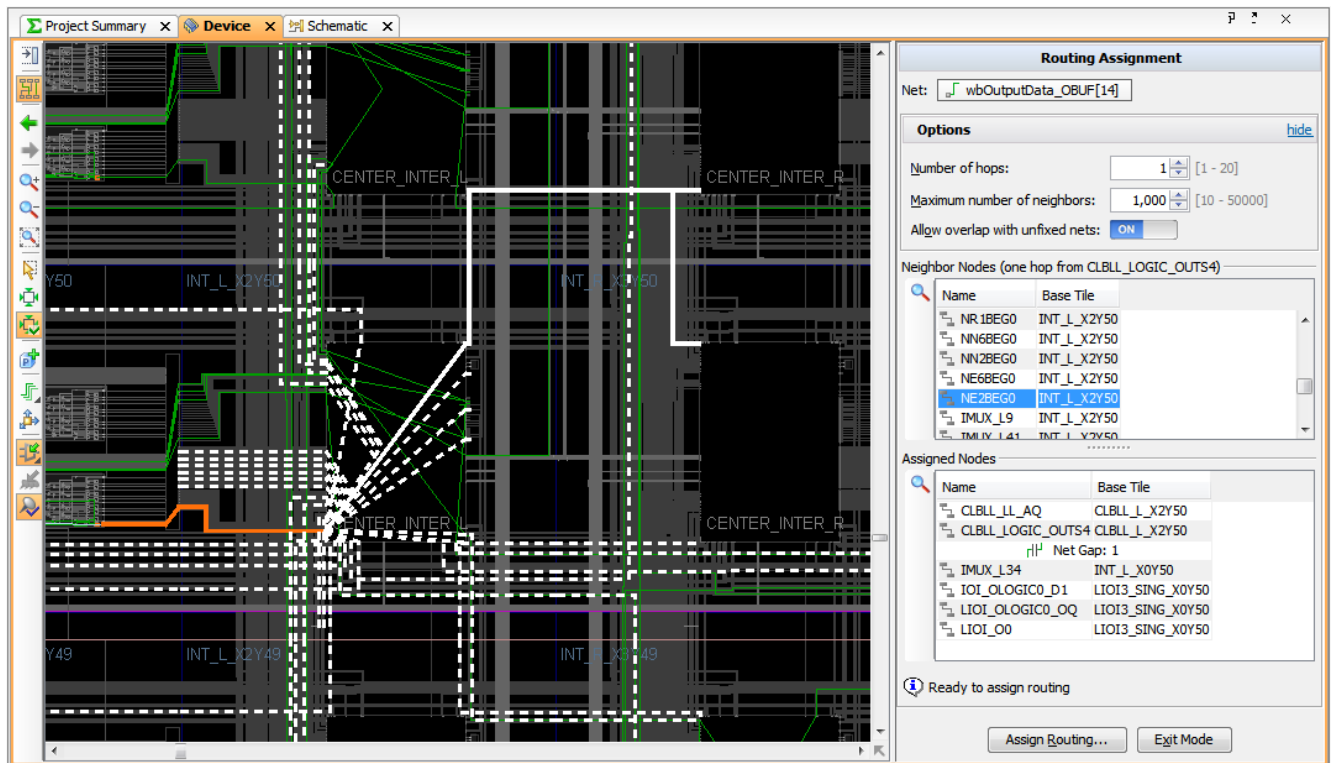


Figure 46: Routing the Gap from CLBL\_LOGIC\_OUTS4

- 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.

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

The node NE2BEG0 is added to the Assigned Nodes section of the Routing Assignment window, and the Neighbor Nodes are updated based on this selection.

- In Neighbor Nodes, select and assign nodes WR1BEG1, and then WR1BEG2.



#### TIPS


- 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.

Figure 47 shows the partially routed path using the selected nodes shown in orange. You will fill the remaining gap using the automatic routing feature.

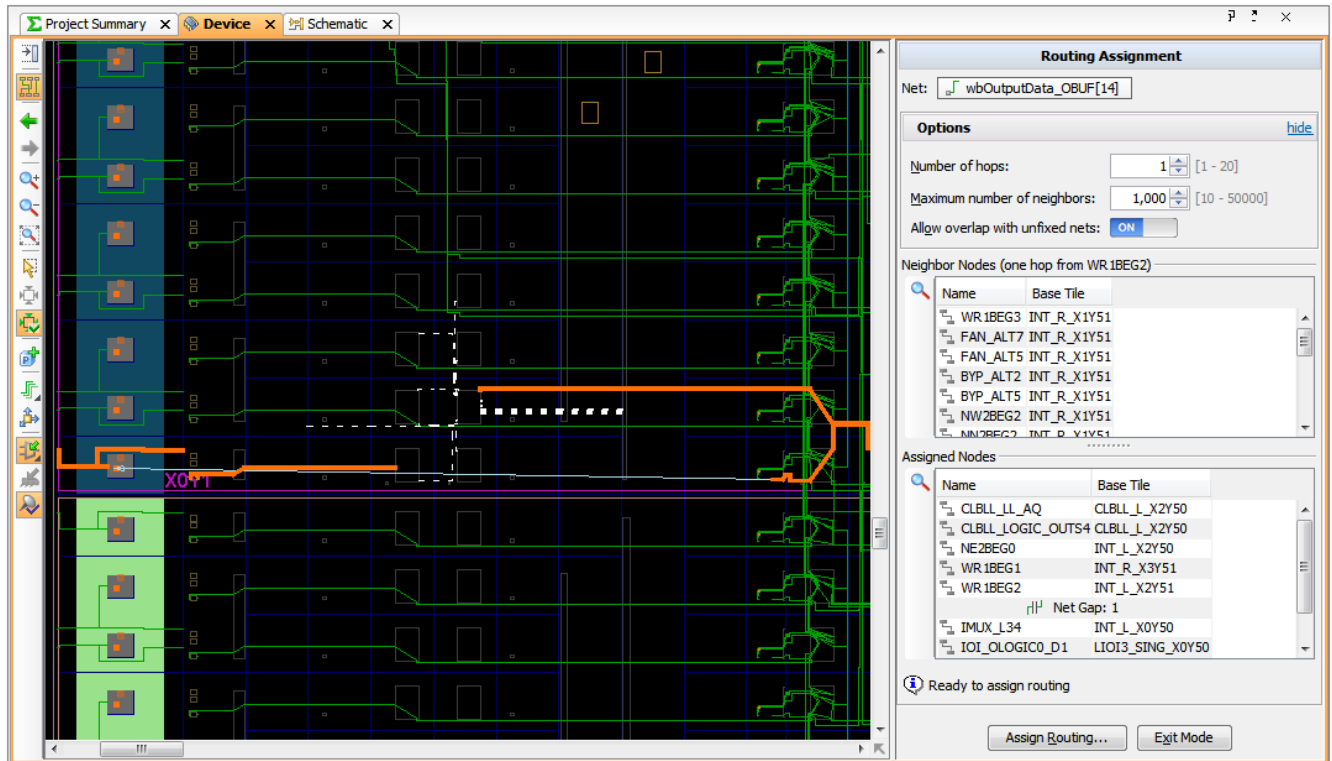


Figure 47: Closing the Gap

- Under the Assigned Nodes section of the Routing Assignment window, right-click the **Net Gap** to open the popup menu, and select **Auto-Route**.

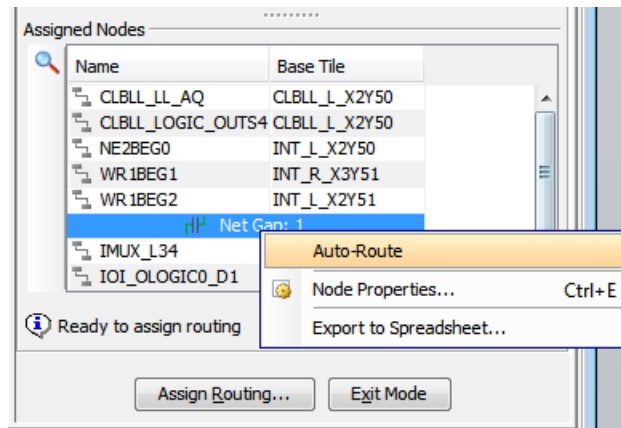
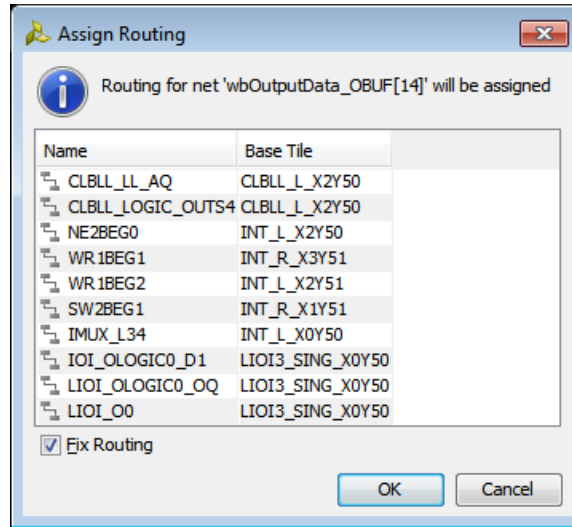


Figure 48: Auto-Route the Gap

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.

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

The Assign Routing dialog box opens, as seen in Figure 49. This displays the list of currently assigned nodes that define the route path. You can select any of the listed nodes, and the node is highlighted in the Device window. This lets you quickly review the route path prior to committing it to the design.



**Figure 49: 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, since you are routing the net manually to add delay to match clock skew.

15. Examine the Tcl commands in the Tcl Console.

The Tcl commands that assigned the routing for the current net are reported in the Tcl console. 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.

Having just completed the manual route definition for the even index nets, you must now define the route path for the odd index net, `wbOutputData_OBUF[15]`. You will apply the same steps you just completed to manually route the net.

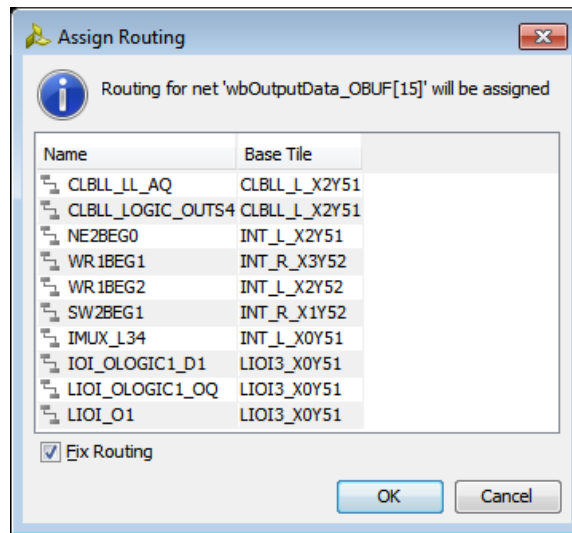
16. In the Tcl Console type the following to select the net:

```
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. Autoroute the gap.
- f. Assign the routing.

The Assign Routing dialog box in Figure 50 shows the nodes selected to complete the route path for the odd index nets.

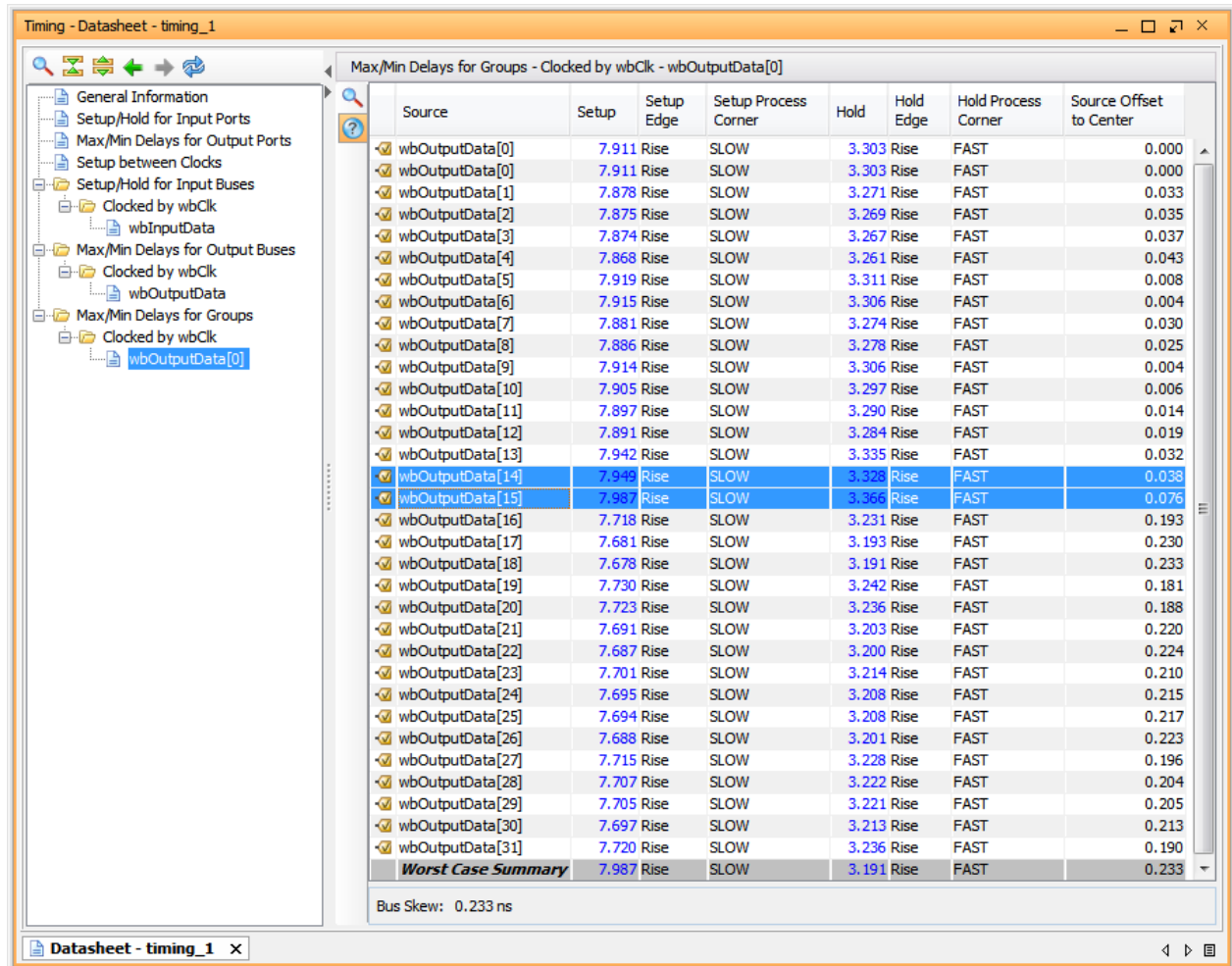


**Figure 50: Assign Routing – Odd Nets**

You have 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.

Switch to the Timing Datasheet report window. Notice the information message in the banner of the window indicates that the report is out of date because the design has been modified.

18. In the Timing Datasheet report, click **Rerun** to update the report with the latest timing information.
19. Select **Max/Min Delays for Groups > Clocked by wbClk > wbOutputData[0]** to display the timing info for the wbOutputData bus, as seen in [Figure 51](#).



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
<b>Worst Case Summary</b>	<b>7.987</b>	<b>Rise</b>	<b>SLOW</b>	<b>3.191</b>	<b>Rise</b>	<b>FAST</b>	<b>0.233</b>

Bus Skew: 0.233 ns

**Figure 51: 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].

You will now copy the same route path to the remaining nets, wbOutputData\_OBUF[31:16], to tighten the timing of the whole wbOutputData bus.

## Step 6: Copy 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 below.

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 same routing paths have been applied to all of the even and odd nets of the output data bus as needed to add 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 may 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 will take precedence over the auto-routed nets.

9. After `route_design`, repeat the `report_route_status` command to see the clean report.

Examine the output data bus in the Device view, as seen in Figure 52. 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 because that was done by the `place_cell` command in an earlier step.

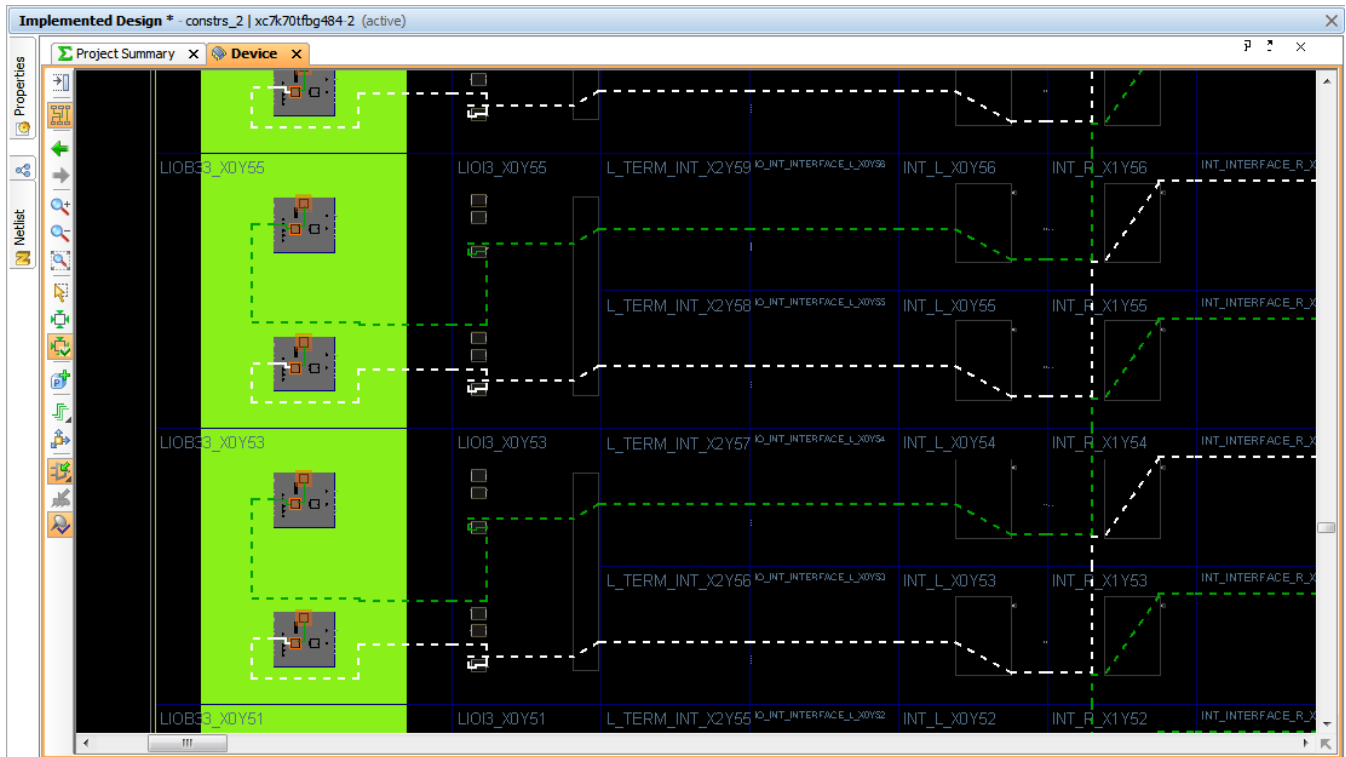


Figure 52: Final Routed Design

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.

10. 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.

11. Click **rerun** to update the report with the latest timing information.

12. 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 53, the clock-to-out timing within all bits of output bus `wbOutputData` is now closely matched to within 83 ps.

13. Save the constraints to write them to the target XDC, so that they are applied every time you compile the design.

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

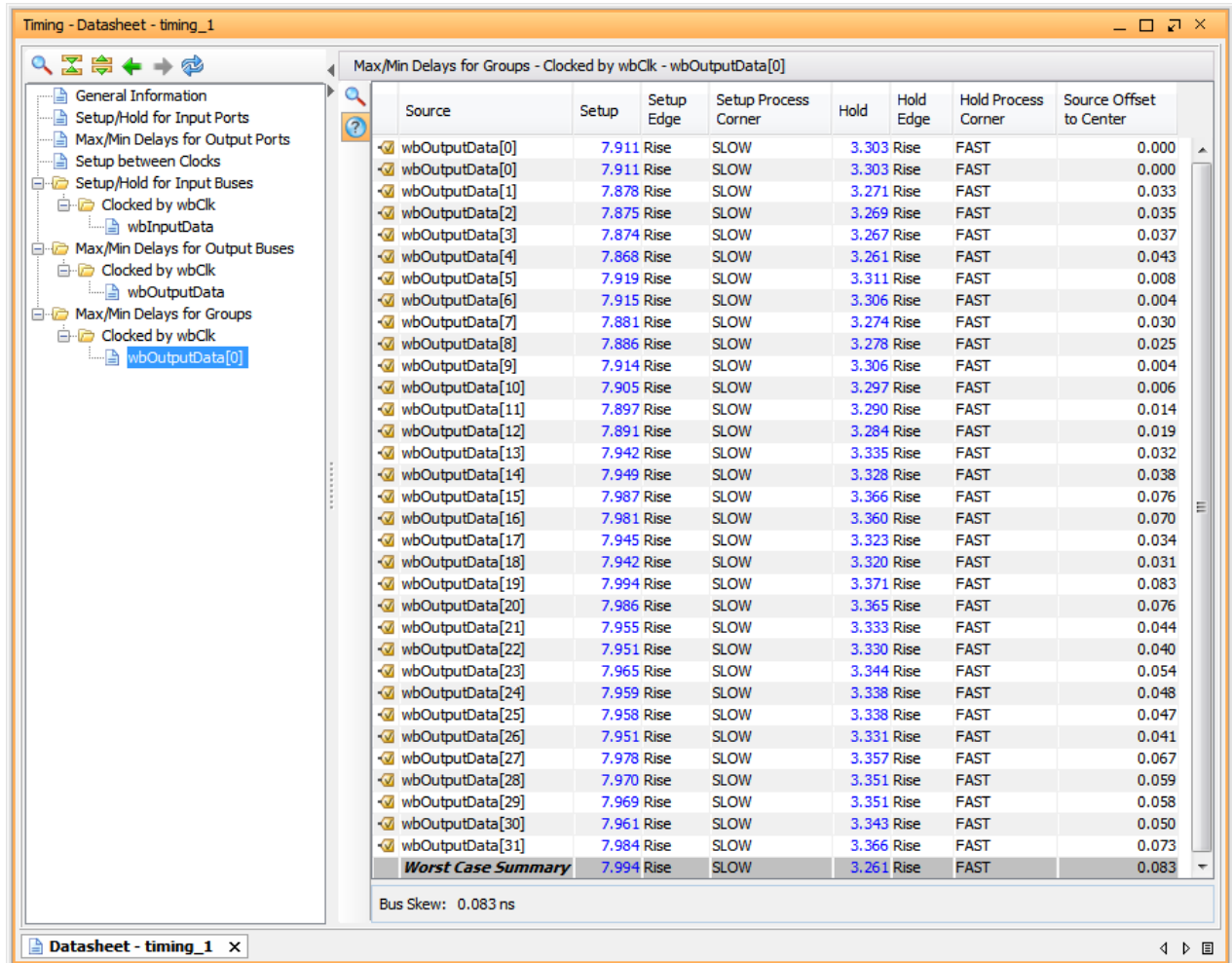


Figure 53: Report Datasheet – Final

## Conclusion

In this lab you:

- 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.

## *Notice of Disclaimer*

---

### **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>.

© Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, UltraScale, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.