# Vivado Design Suite Tutorial:

## *Implementation*

**UG986 (v 2013.3) November 14, 2013**

XILINX®

# Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 11/14/2013 | 2013.3 | Removed Lab #3. |
| 06/19/2013 | 2013.2 | Added a third lab, "Manual Routing and Directed Routing," to the tutorial. |
| 04/26/2013 | 2013.1 | First publication |

# Table of Contents

# Implementation Tutorial

## Overview

This tutorial includes two labs which demonstrate an aspect of the Xilinx® Vivado® implementation flow:

- Lab #1: Using Implementation Strategies
- Lab #2: Using Incremental Compile

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

> **TIP:** *You can also learn more about implementing the design by viewing quick take video at [http://www.xilinx.com/csi/training/vivado/implementing-the-design.htm](http://www.xilinx.com/csi/training/vivado/implementing-the-design.htm).*

## Tutorial Design Description

The sample design used for Lab #1 consists of a small design called `project_cpu_netlist`. There is a top-level EDIF netlist source file, as well as an XDC constraints file.

The sample design used for Lab #2 consists of a small RTL design called `project_bft_core_hdl`. The design includes both Verilog and VHDL RTL files, as well as an XDC constraints file.

Both 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.

## Software Requirements

This tutorial requires that the 2013.3 Vivado Design Suite software release or later is installed. For installation instructions and information, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (*UG973*).

## Hardware Requirements

The supported Operating Systems include Redhat 5.6 Linux 64 and 32 bit, and Windows 7, 64 and 32 bit.

Xilinx recommends a minimum of 2 GB of RAM when using the Vivado tool.

# 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 2013.x > Vivado 2013.x
```
[1]

***Note:*** As an alternative, click the **Vivado 2013.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.

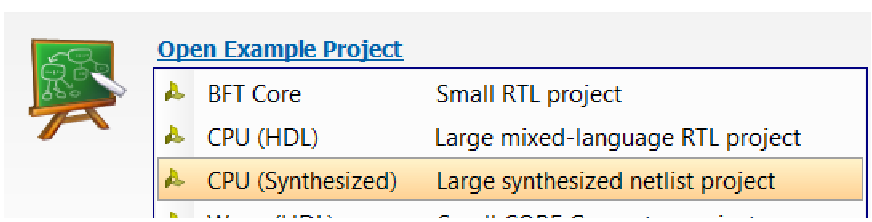



**Figure 1: Open Example Project**

2. From the Getting Started page, click **Open Example Project** and select the **CPU (Synthesized)** design.

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

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

---

[1] Your Vivado Design Suite installation may called something different than **Xilinx Design Tools** on the **Start** menu.

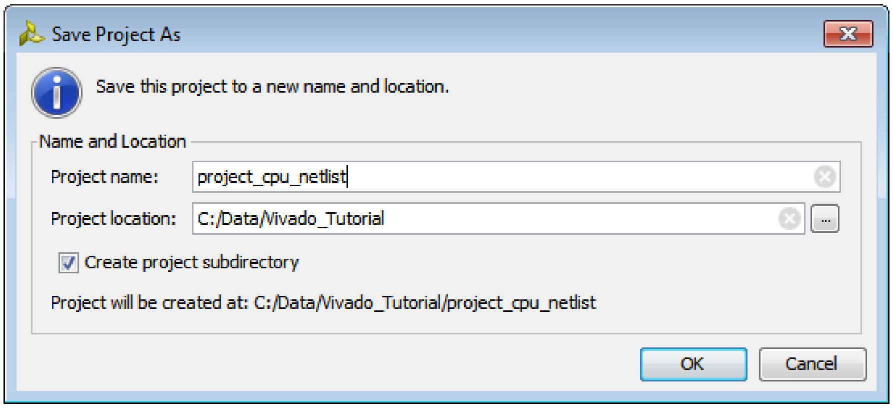



**Figure 2: Save Project As**

4. 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.

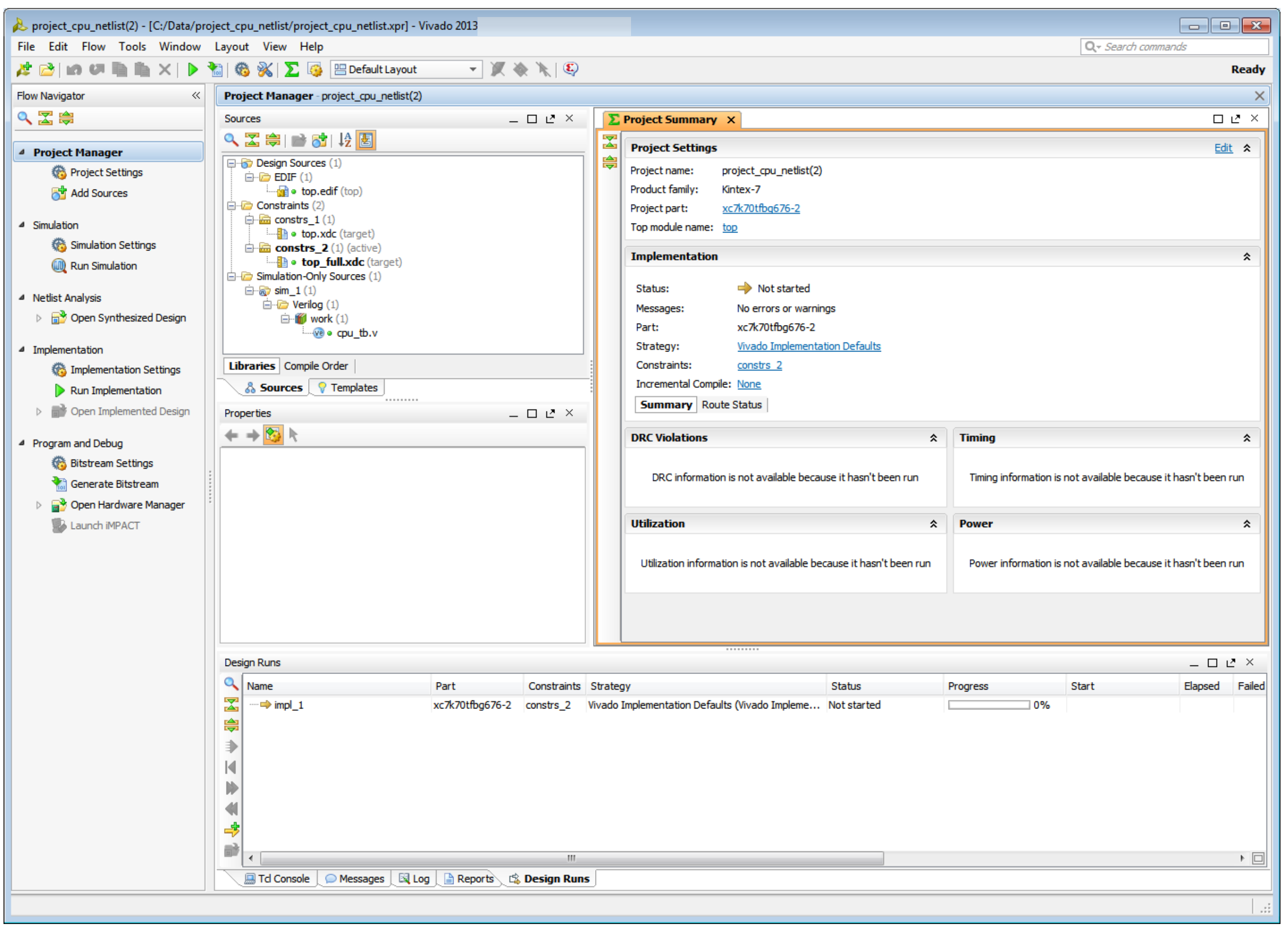



**Figure 3: Project Summary Window**

# 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.

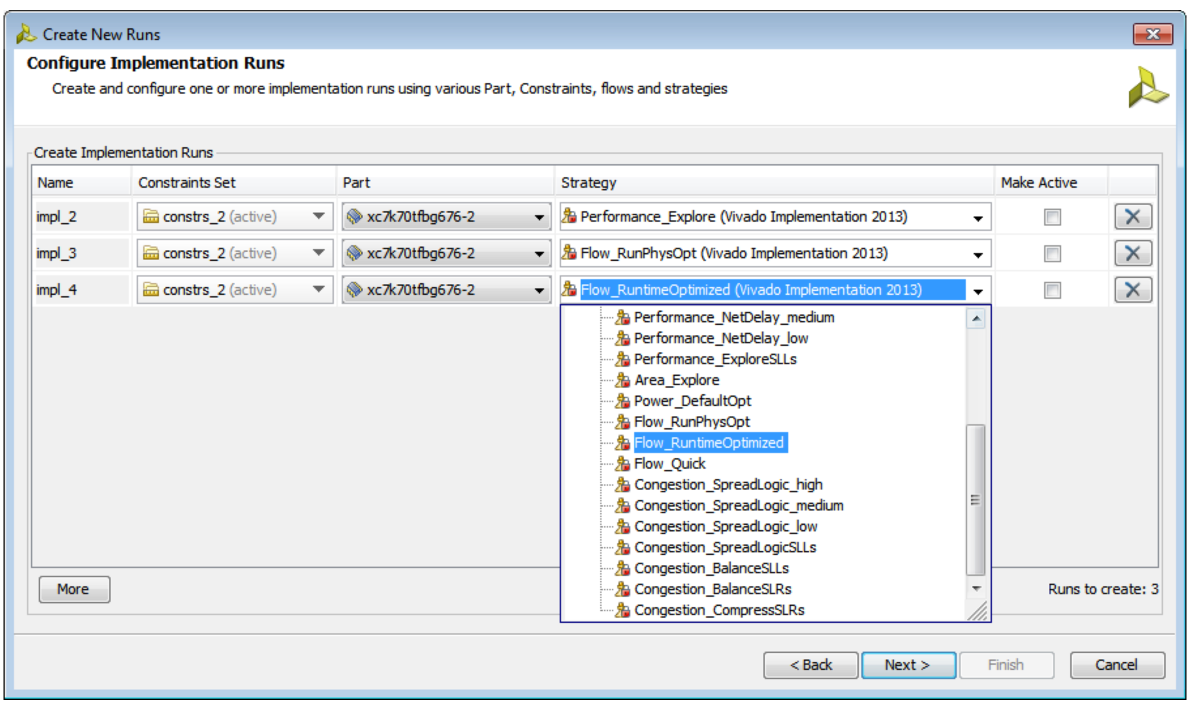



**Figure 4: Configure Implementation Runs**

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** the **implementation runs**.

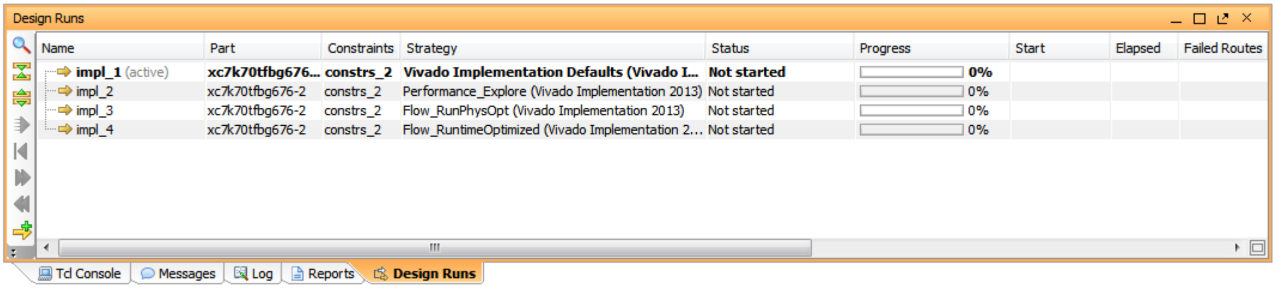



**Figure 5: Design Runs Window**

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

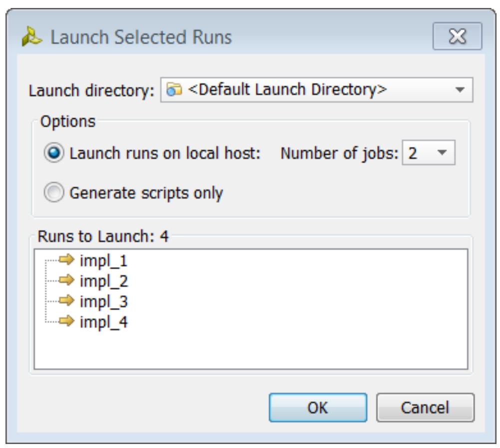



**Figure 6: Launch Selected Runs**

4. Click **OK**.

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

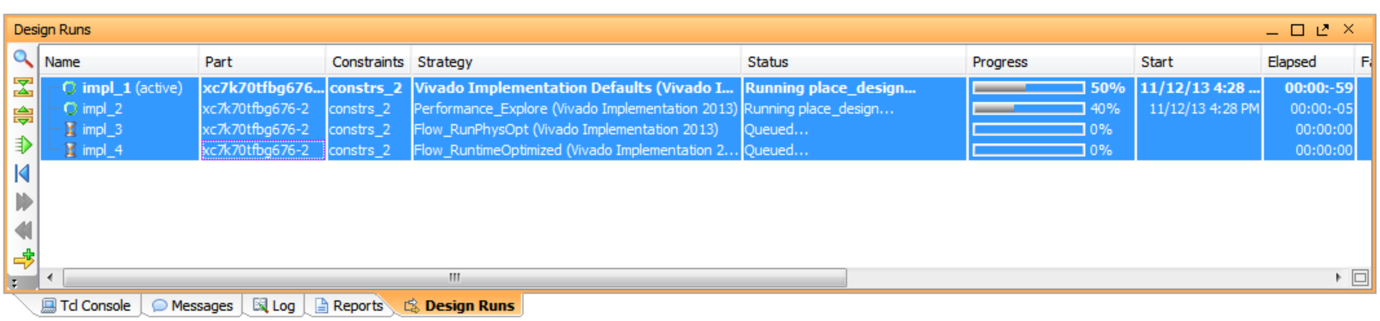



**Figure 7: Two Implementation Runs in Progress**

When the active run, `impl_1`, completes, examine the 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.
6. 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.

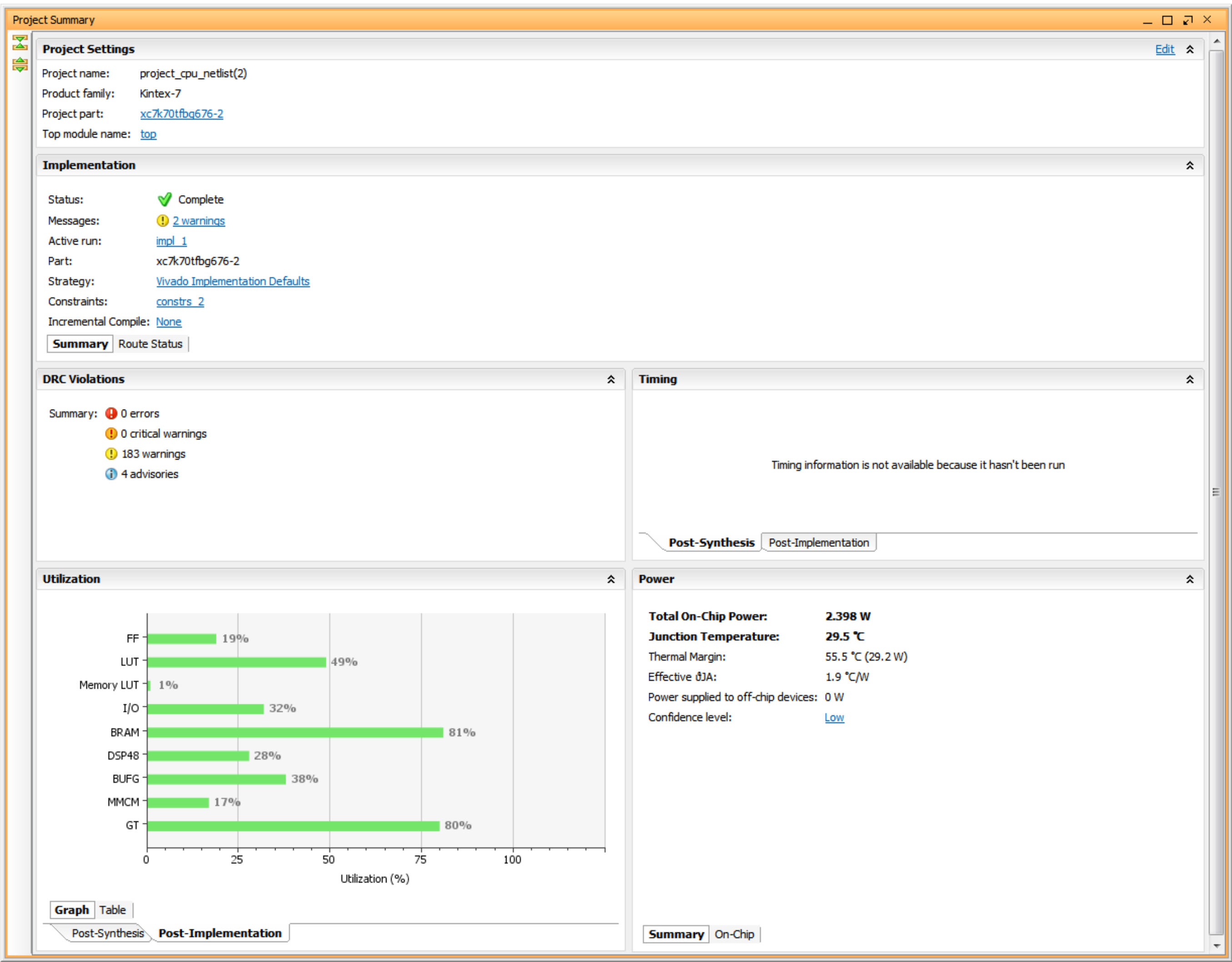



**Figure 8: Project Summary**

7. 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`.

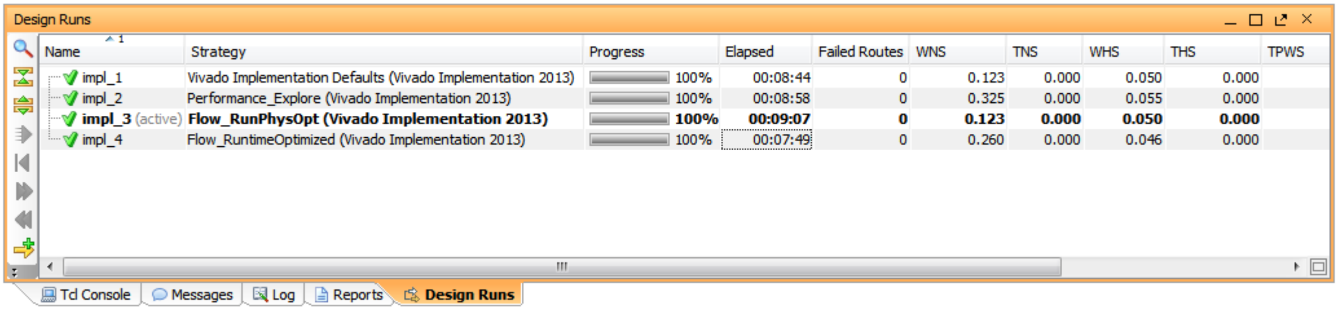



**Figure 9: Compare Implementation Results**

8. **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, with the **Performance_Explore** strategy offering the greatest positive slack.

# 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.

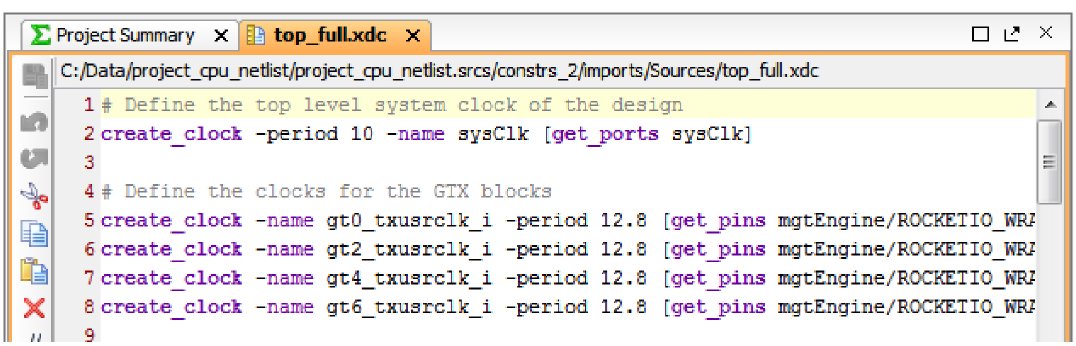



**Figure 10: top_full.xdc File**

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.

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

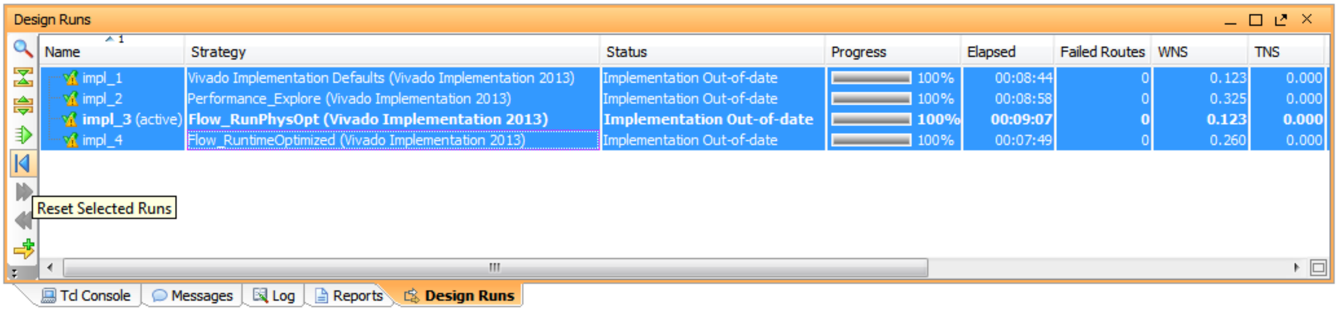



**Figure 11: Reset Runs**

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."

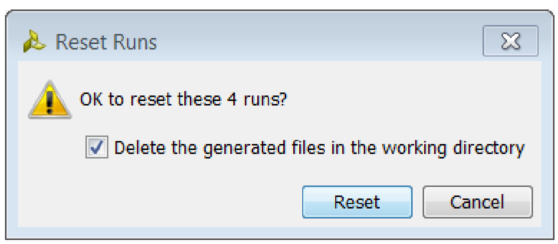



**Figure 12: Reset Runs**

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

   The Launch Selected Runs window opens, as seen in Figure 13.

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

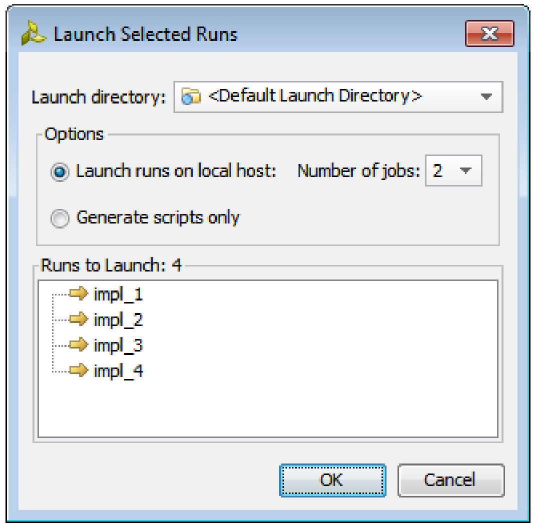



**Figure 13: Launch Selected Runs Window**

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 XXX.

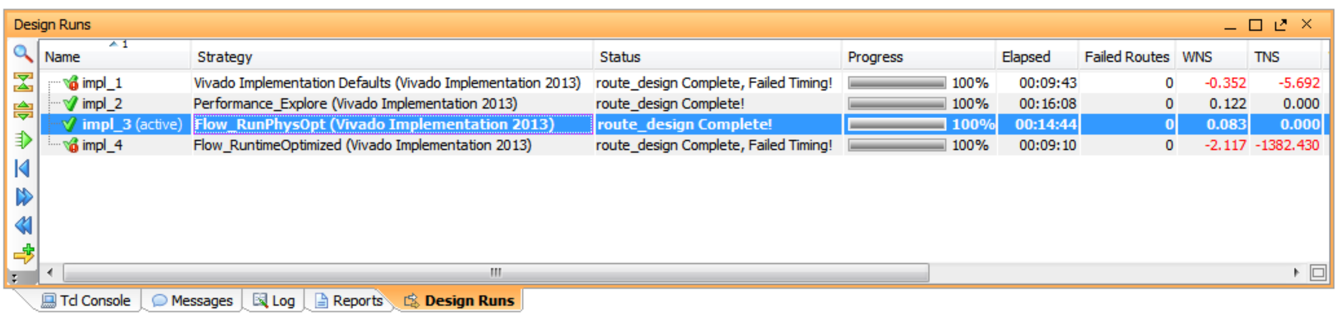



**Figure 14: Implementation Results**

- Notice that the `impl_2` run, using the **Performance_Explore** strategy has the greatest positive WNS.
- The **Flow_RunPhysOpt** strategy, in `impl_3`, also resulted in a positive slack.
- The implementation runs that met the timing requirements took longer than the runs that failed timing because they spent more time resolving timing.

***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 may be needed in 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.

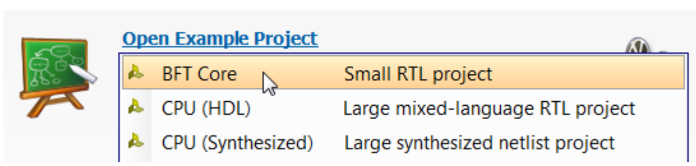



**Figure 15: Open Example Design**

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

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

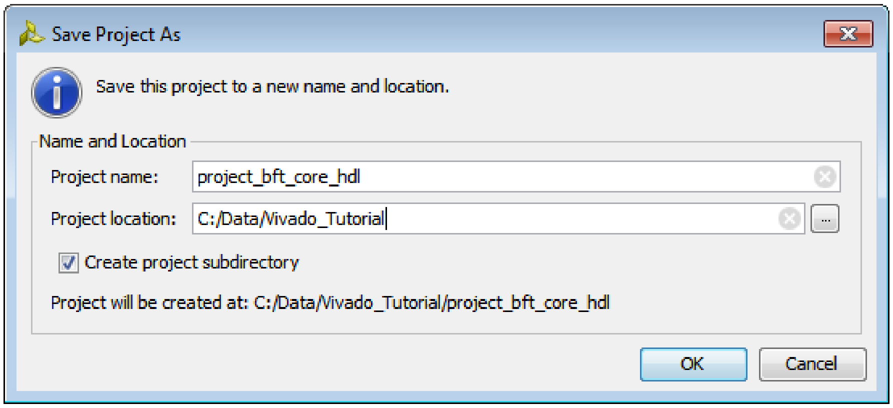



**Figure 16: Save Project As**

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.

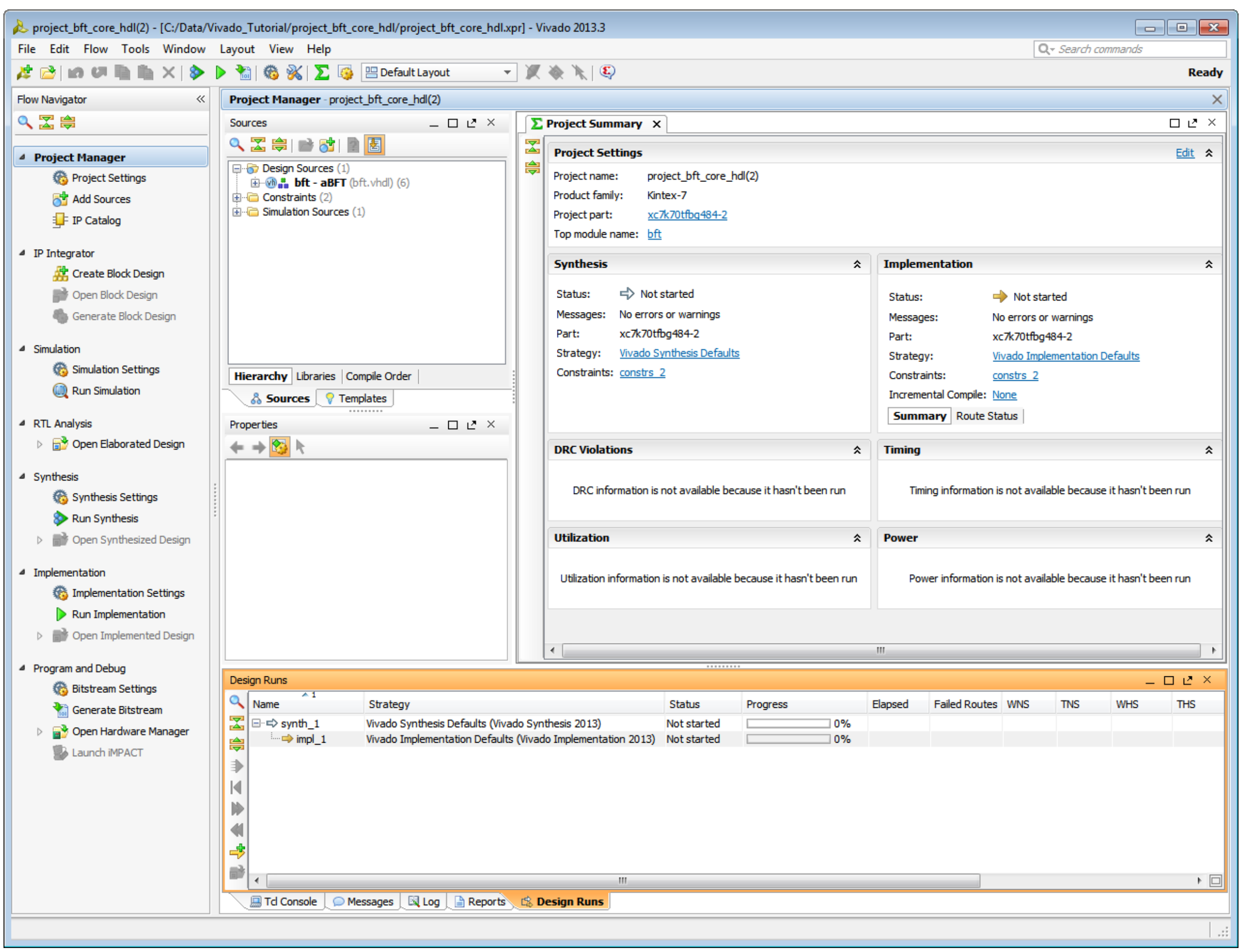


**Figure 17: BFT Project Summary**

# 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 18. 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.

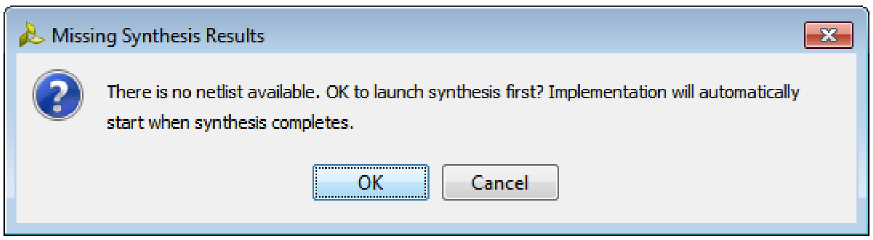



**Figure 18: Missing Synthesis Results**

   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 19. 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

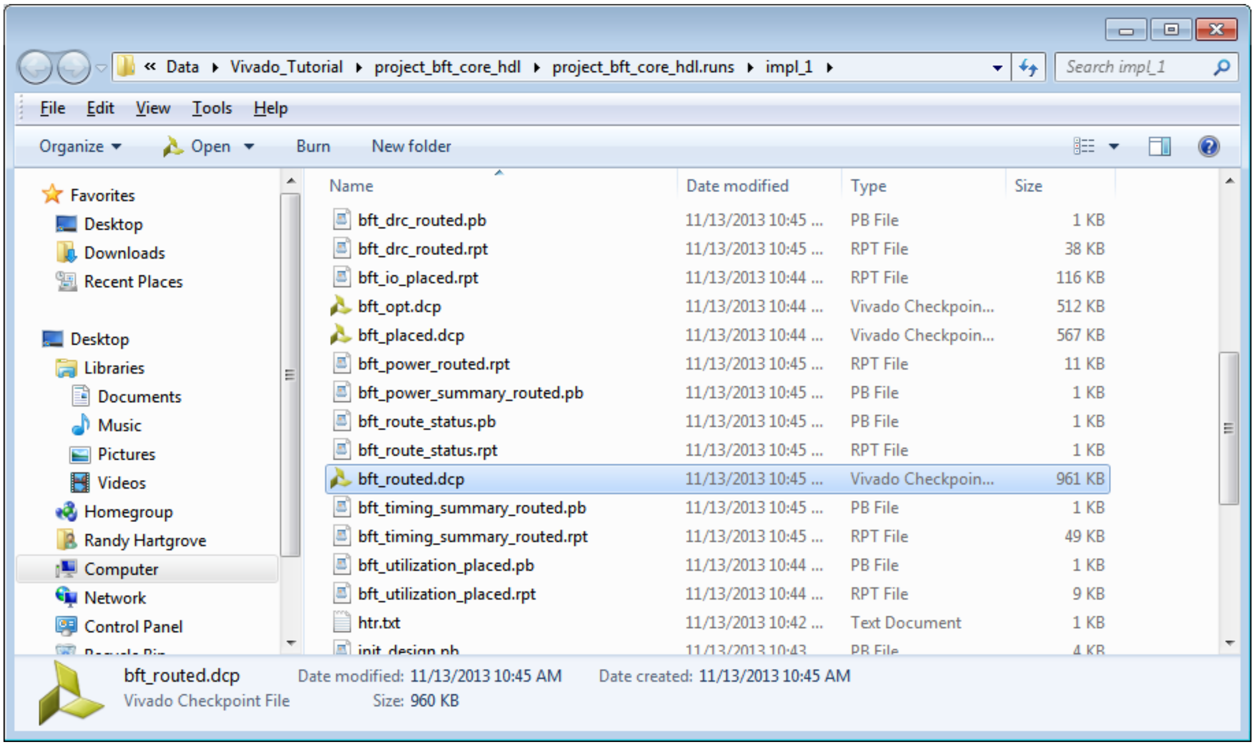



**Figure 19: Implementation Run Directory**

# 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 over written.

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

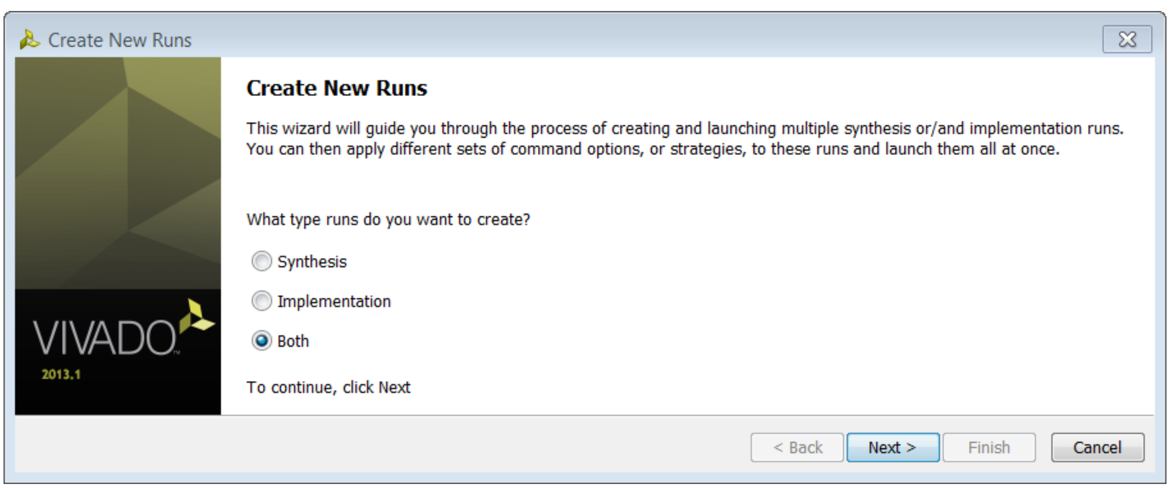



**Figure 20: Create New Runs Wizard**

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 21.

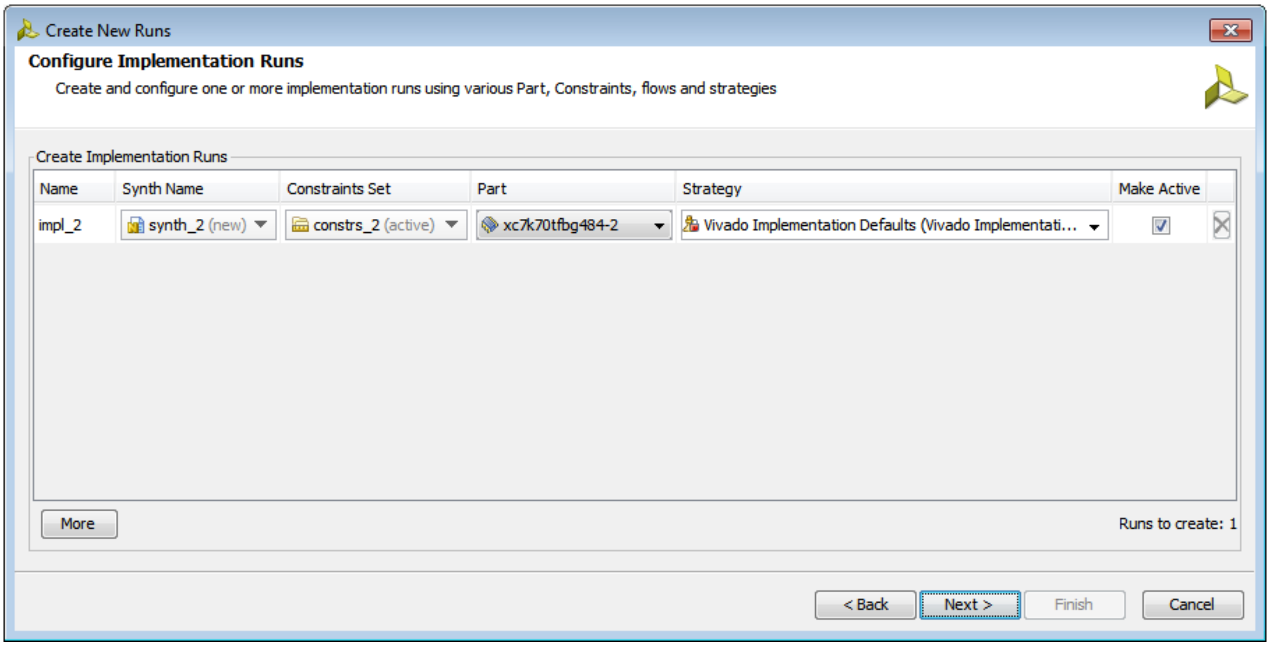



**Figure 21: Configure Implementation Runs**

4. Enable the **Make Active** checkbox, and click **Next**.

5. The Launch Options window opens, as shown in Figure 22.

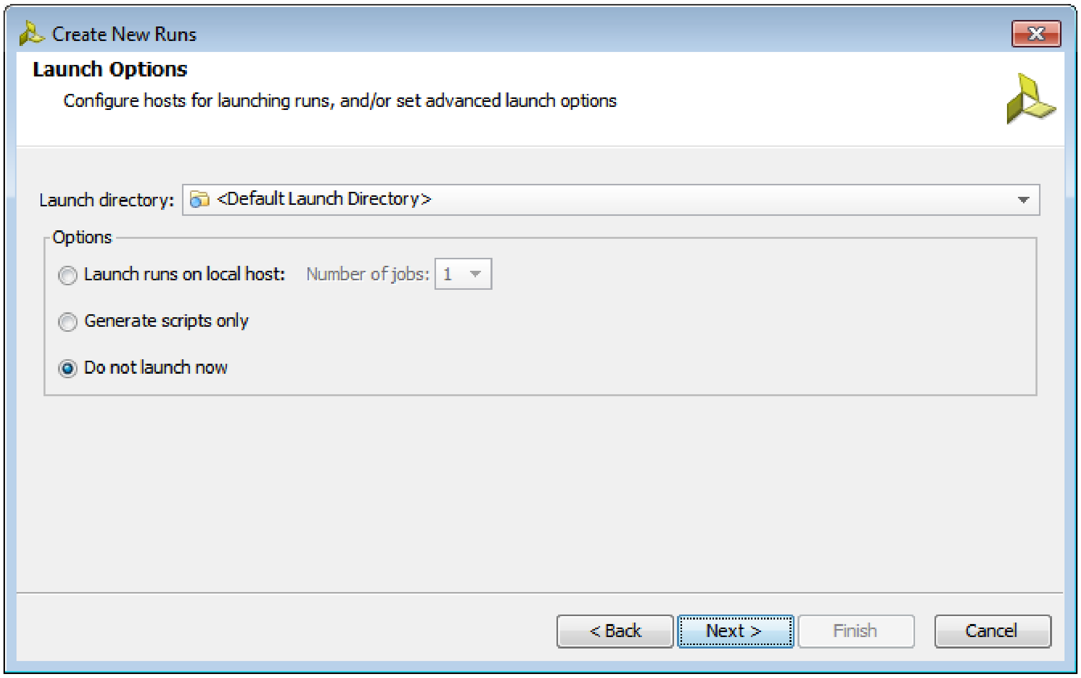



**Figure 22: Launch Options**

6. Select **Do not launch now** and click **Next**.
7. 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. .

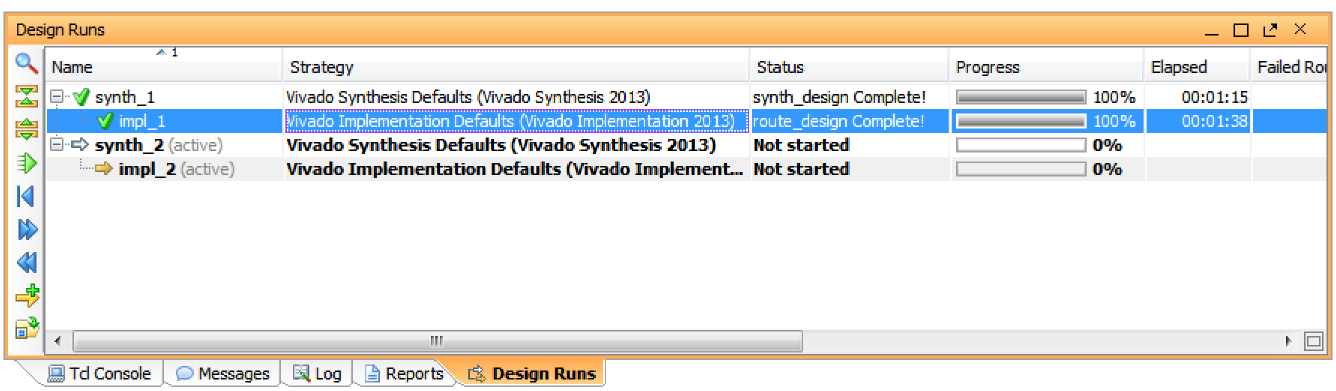



**Figure 23: New Design Runs**

# 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.

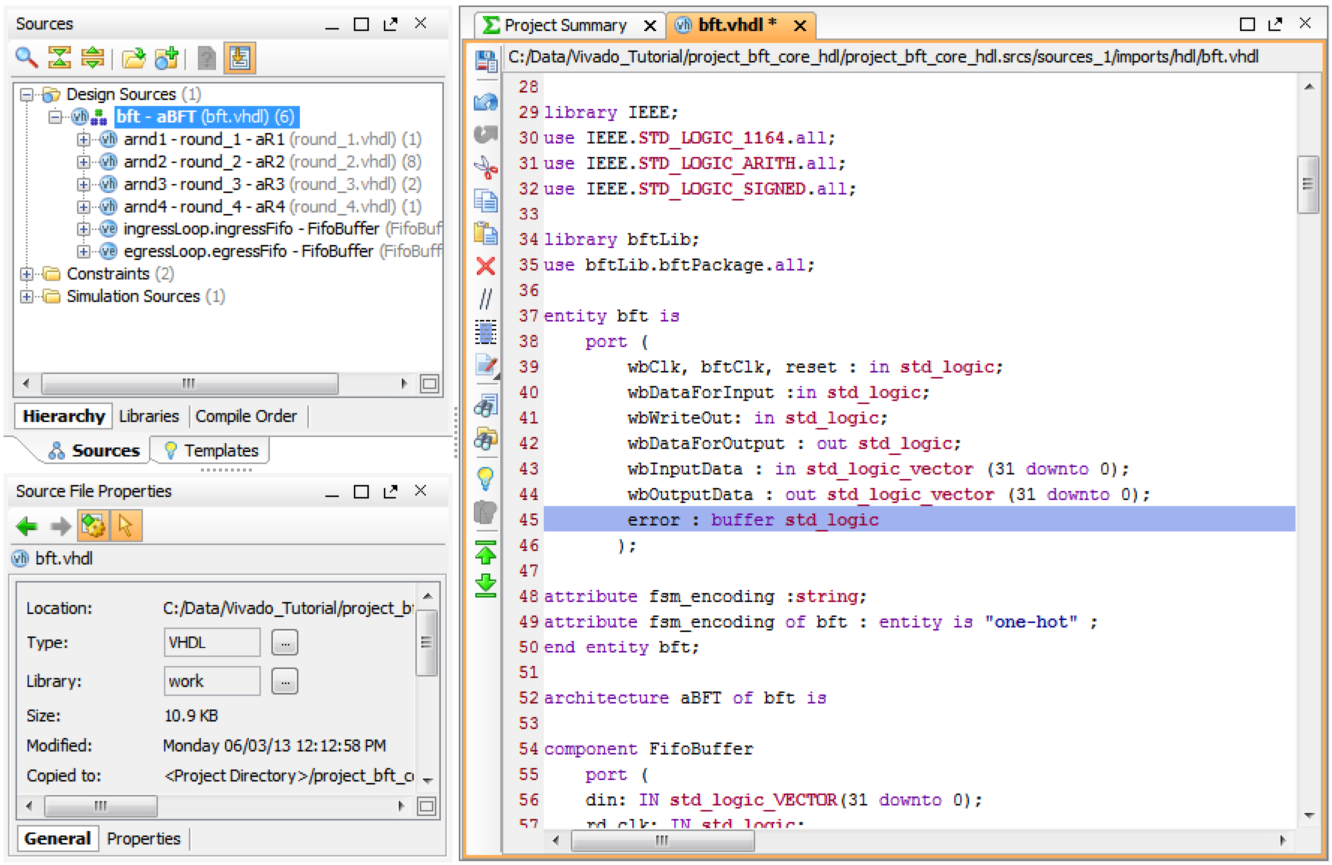



**Figure 24: BFT VHDL File**

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**

```
-- enable the read fifo
process (fifoSelect)
begin
    readEgressFifo <= fifoSelect;
end process;
```

**To**

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

The results should look like Figure 25.

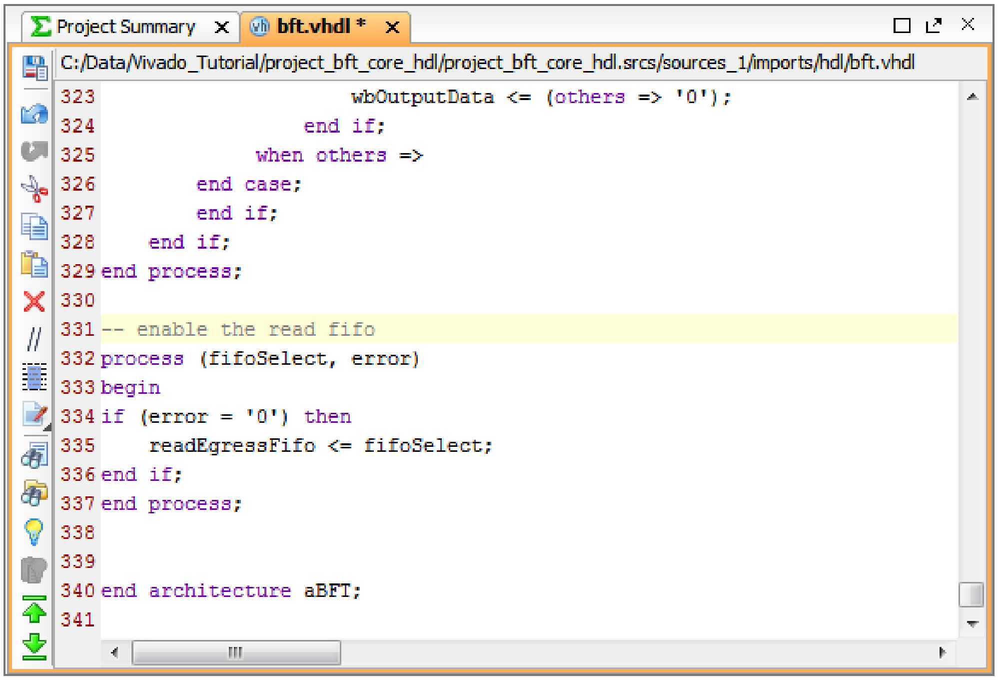



**Figure 25: Modified VHDL Code**

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

   As you can see in Figure 26, changing the design source files also changes the run status for finished runs from **Complete** to **Out-of-date**.

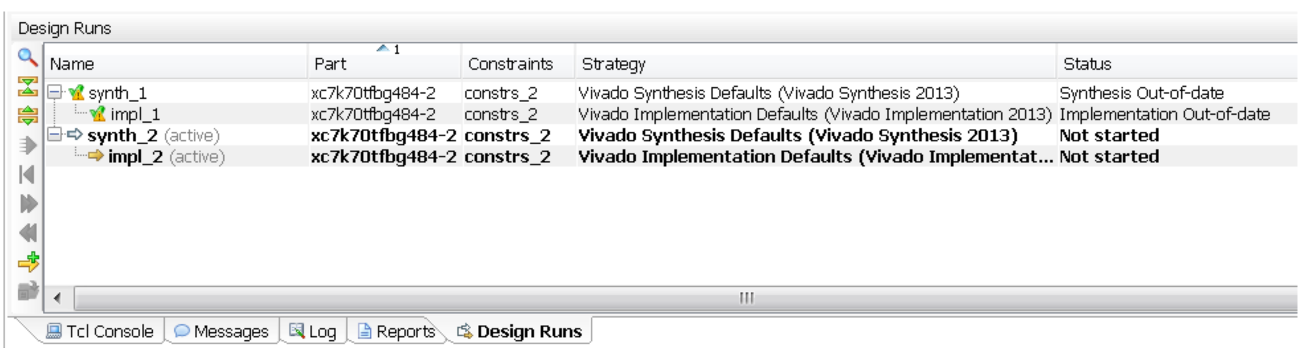



**Figure 26: Design Runs Out-of-Date**

5. In the Flow Navigator, click **Run Synthesis** to synthesize the updated netlist for the design.

# 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 that the Vivado Design Suite will draw placement and routing data from.

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 27.

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.

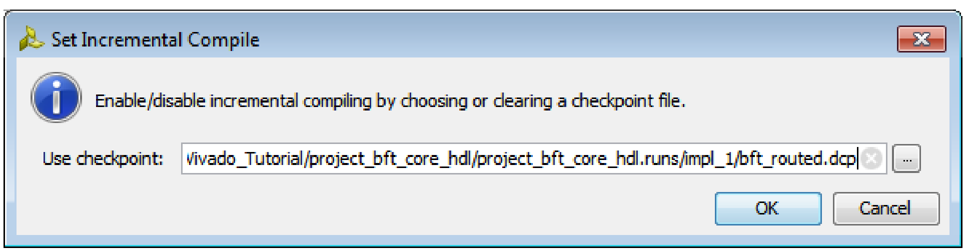



**Figure 27: Set Incremental Compile**

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 28, the Design Runs window shows that the elapsed time for implementation run `impl_2` is lower 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.

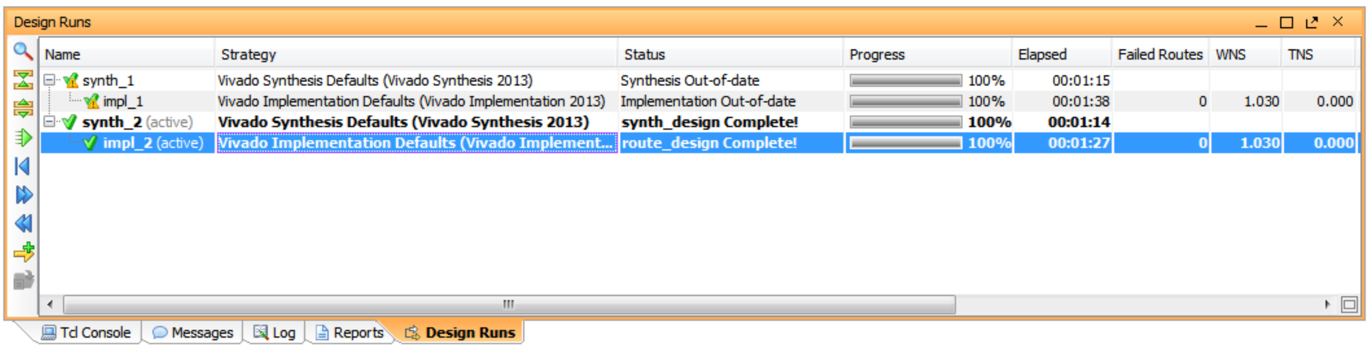



**Figure 28: Elapsed Time for Incremental Compile**

8. Select the **Reports** tab in the Results Window Area and double-click the **Incremental Reuse Report** in the Place Design section, as highlighted in Figure 29.

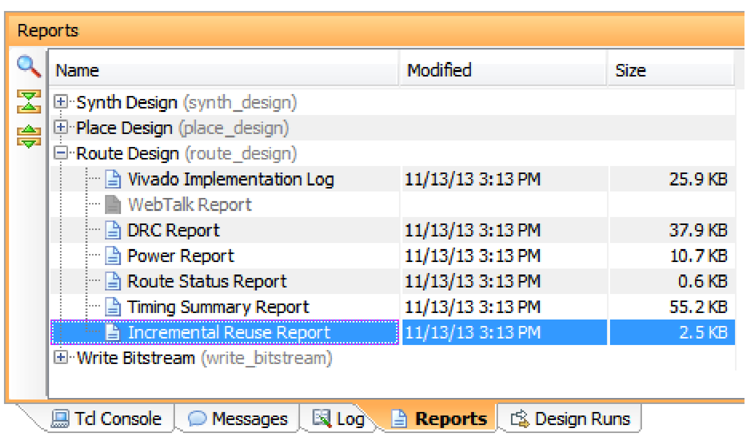
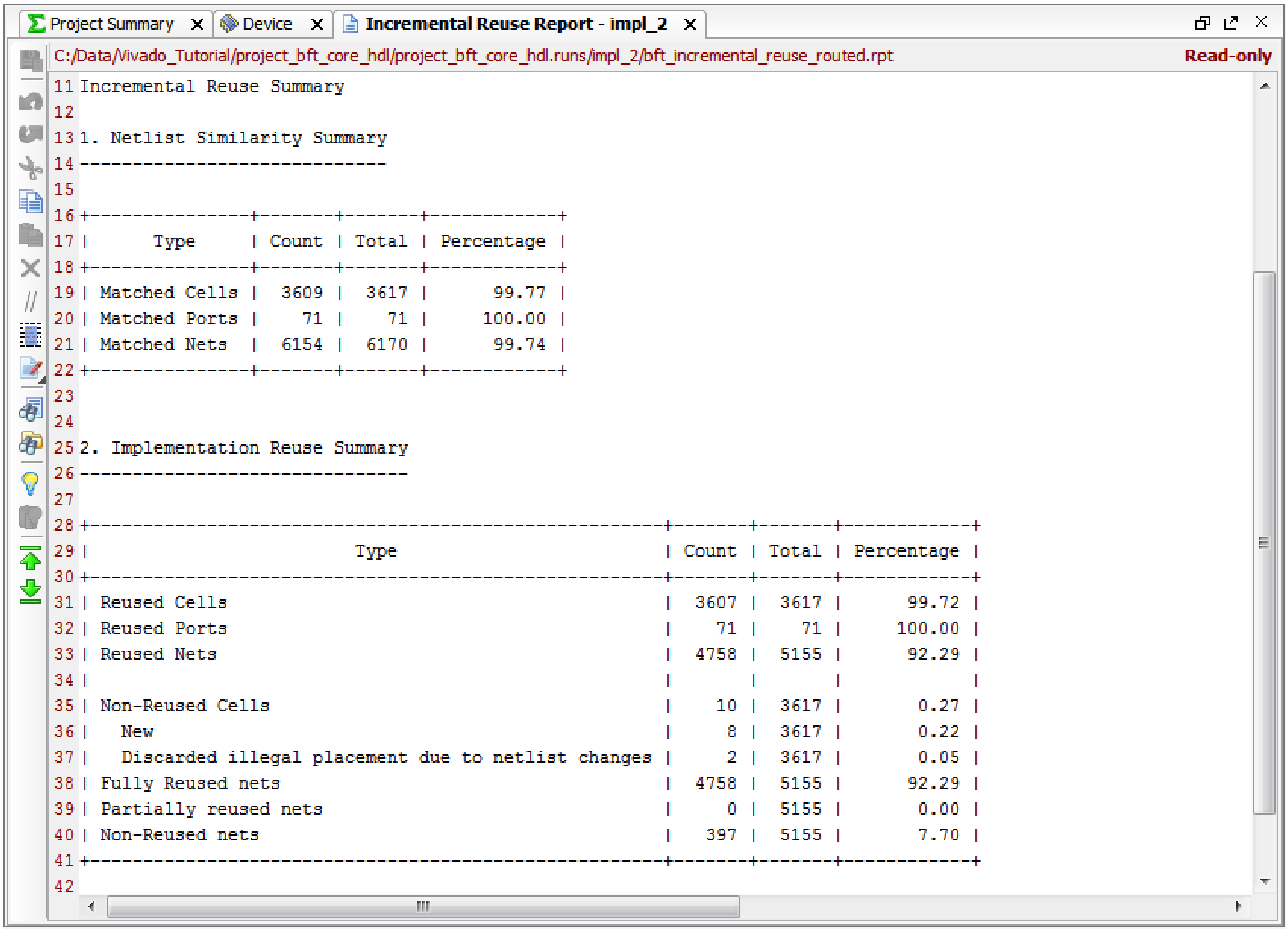



**Figure 29: Reports Window**

The Incremental Reuse Report opens in the Vivado IDE text editor, as seen in Figure 30. This report shows the percentage of reused Cells, Ports and Nets. The higher the percentage, the more effective the reuse of placement and routing from the incremental checkpoint.




```
Incremental Reuse Summary

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

+---------------+-------+-------+------------+
|      Type     | Count | Total | Percentage |
+---------------+-------+-------+------------+
| Matched Cells |  3609 |  3617 |      99.77 |
| Matched Ports |    71 |    71 |     100.00 |
| Matched Nets  |  6154 |  6170 |      99.74 |
+---------------+-------+-------+------------+


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

+-------------------------------------------------------+-------+-------+------------+
|                         Type                          | Count | Total | Percentage |
+-------------------------------------------------------+-------+-------+------------+
| Reused Cells                                          |  3607 |  3617 |      99.72 |
| Reused Ports                                          |    71 |    71 |     100.00 |
| Reused Nets                                           |  4758 |  5155 |      92.29 |
|                                                       |       |       |            |
| Non-Reused Cells                                      |    10 |  3617 |       0.27 |
|   New                                                 |     8 |  3617 |       0.22 |
|   Discarded illegal placement due to netlist changes  |     2 |  3617 |       0.05 |
| Fully Reused nets                                     |  4758 |  5155 |      92.29 |
| Partially reused nets                                 |     0 |  5155 |       0.00 |
| Non-Reused nets                                       |   397 |  5155 |       7.70 |
+-------------------------------------------------------+-------+-------+------------+
```

**Figure 30: 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