Xilinx Customer Training

UltraFast Design Methodology
Lab Workbook

fpga-vdm-2022.2-wkb-lab-rev1

AMD
XILINX

UltraFast Design Methodology Lab Workbook 2022.2

Lab FAQ

- Where can I get the files for the labs?
  - www.xilinx.com/training/downloads.html
  - These are original files and do not contain any work that you may have performed.
  - Labs were developed using version 2022.2 of the tools. Later versions may work and will likely require you to update various pieces of IP. See the "Updating IP" topic in the *Lab Reference Guide* for instructions on how to update IP (Vivado Design Suite Operations > Vivado IP Integrator Operations > Updating IP).
  - These labs were validated using a virtual machine that hosts Ubuntu Linux. Many labs can be performed using the Windows OS; however, not all AMD Xilinx tools are supported under Windows (such as QEMU and PetaLinux).
  - Please use the latest release lab and ensure that the lab instruction version matches the lab file set version!
- What if I cannot answer a question in the lab?
  - Do your best! The questions are meant to stimulate thought, not to test your knowledge. After you have considered a question for a bit, you can find the answer at the end of each lab.
- Where can I get more detailed information on a topic?
  - The *Lab Reference Guide* is a collection of "how to" topics for commonly performed tasks categorized by the tool (Vivado Design Suite, Vivado analyzer, Vitis platform, etc.) and subdivided into major areas within the tool.
  - The *Lab Reference Guide* is available from the lab files download as well as from www.xilinx.com/training/downloads.htm.
- Where can I find more information on lab software requirements and hardware setup?
  - The lab setup guide details this information and is available from the lab files download.
- How do the instructions work?
  - The instructions are provided in three layers:
    - Steps – these are the major/broadest aspects of solving the problem that the lab poses (X).
    - Instructions – these represent the significant instructions towards solving the issue outlined by the step (X-X).
    - Tasks – these are the finest granularity items that (when combined with the other tasks) solve the instruction to which they are subordinate (X-X-X).

The *Step* covers the process of achieving a major milestone toward completing the lab

Launching the Vivado Design Suite — Step 1

There are a number of ways to launch the Vivado Design Suite. The two most popular mechanisms are shown here.

The *Step* explanation describes what will be accomplished in this *Step*

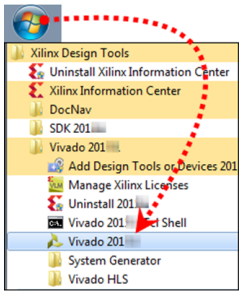
1-1. Launch the Vivado Design Suite.

This can be done in two standard ways, use your preferred method.

Each *Instruction* describes a major component of the *Step*

1-1-1. Select **Start** > **All Programs** > **Xilinx Design Tools** > **Vivado 2014.3** > **Vivado 2014.3**.

Each *Task* represents a clickable action - most tasks refer to a screenshot for additional clarity



Additional explanations may be present for instructions and tasks

Figure 4-2: Launching the Vivado Design Suite from the Start Menu

-- OR --

Table of Contents

Lab 1: Vivado Design Suite I/O Pin Planning

2022.2

Abstract

This lab introduces the I/O planning capabilities of the Vivado® Design Suite for FPGA devices.

The lab supports both Verilog and VHDL design files. However, the lab instructions and figures show only Verilog usage.

This lab should take approximately 30 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Use the basic design analysis features of the Vivado IDE to explore your design
- Step through the I/O pin planning process using the GUI
- Assign configured I/O ports to the physical package pins

Introduction

The Vivado IDE provides an I/O planning environment that enables I/O ports assignment to package pins. In this environment, you can assign I/O locations, specify I/O banks and I/O standards, or create legal pin assignments by reporting Design Rule Checks (DRC).

I/O planning can be performed at any stage of the design flow with any type of project. Some of the most common methods are:

- Pre-RTL I/O planning
- RTL I/O planning
- Netlist I/O planning
- I/O validation with an Implemented design

You can also use the I/O planning layout view to see the relationship of the physical package pins and banks with the corresponding I/O pads.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

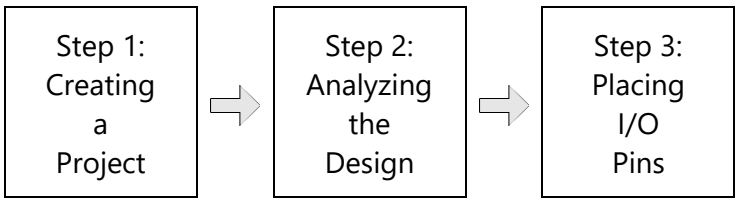
If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Creating a New Project Step 1

You will begin the lab by launching the Vivado Design Suite to create a new project using the New Project Wizard in the Vivado IDE.

The New Project Wizard in the Vivado IDE will create an XPR project file. The Vivado IDE project file (*.xpr*) organizes your design files and saves the design status whenever the processes are run from design entry through implementation to programming the targeted device.

Here are two ways to open the Vivado Design Suite.

1-1. Open the Vivado Design Suite.

1-1-1. Click the **Vivado** icon ( ) from the taskbar.

Figure 1-1: Launching the Vivado Design Suite from the Taskbar

Note: It takes a few moments to open. The order of the icons in your environment may be different.

Alternatively, from the Linux terminal window (<**Ctrl** + **Alt** + **T**>), enter the following:

```
source /opt/amd/Vivado/2022.2/settings64.sh; vivado
```

Note: This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

The tool opens to the Welcome window. From here you can create a new project, open an existing project, enter Tcl commands, and access documentation and examples.

1-1-2. [Optional] Maximize the window as there is a lot of information to see.

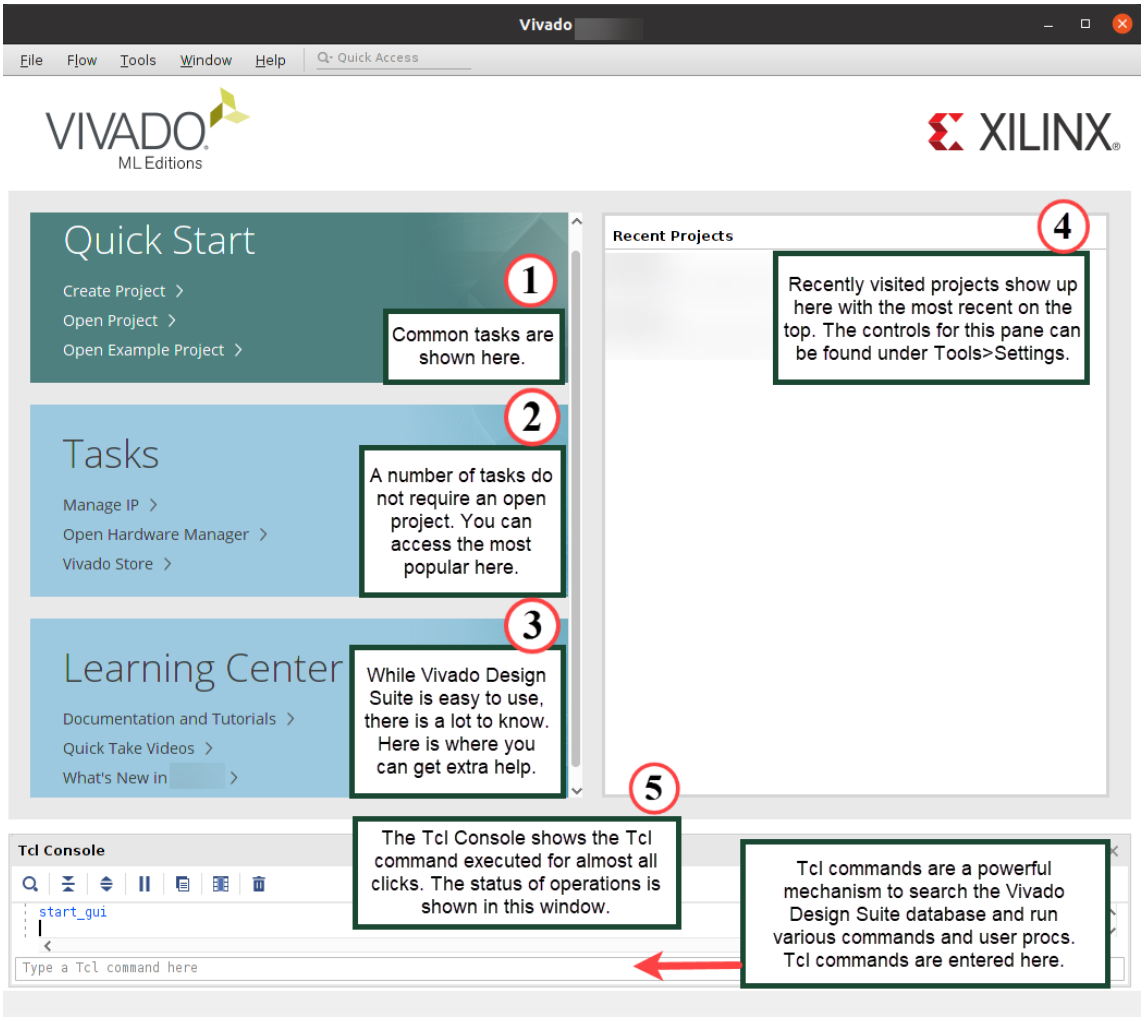


Figure 1-2: Vivado Design Suite Welcome Screen

Hint: If the Tcl Console is not visible, double-click the **Tcl** tab to make it visible.

Projects begin with the creation of a new project. The project contains sources, settings, graphics, IP, and other elements that are used to build a final bitstream.

1-2. Create a new Vivado Design Suite project.

1-2-1. Click **Create Project** (1).

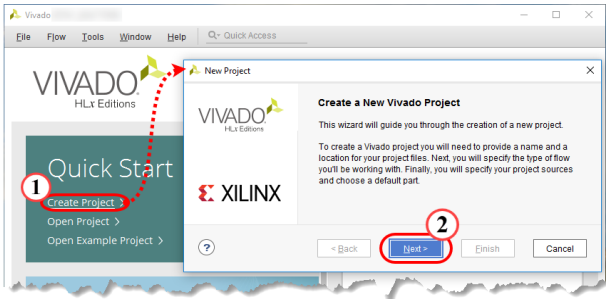


Figure 1-3: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

1-2-2. Click **Next** to begin entering the specifics for this project (2).

1-3. You will now encounter a series of dialog boxes asking you to enter different pieces of information describing the project.

1-3-1. Enter **uart_led** in the Project name field(1).

1-3-2. Enter **$TRAINING_PATH/IO_PinPlanning/lab/KCU105/[verilog | vhdl]** in the Project location field (2).

Select your preferred language to be either Verilog or VHDL.

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

1-3-3. Deselect the **Create Project Subdirectory** option, as leaving this checked will create an additional level of hierarchy in the lab (3).

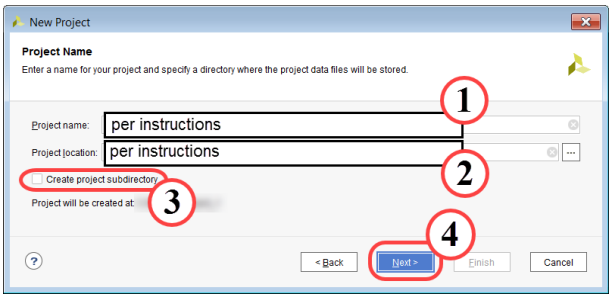


Figure 1-4: Entering the Project Name and Location

1-3-4. Click **Next** to accept the selections and advance to selecting a type of project (4).

The Project Type dialog box invites you to choose between an RTL project or a post-synthesis project. Simply put, an RTL project enables you to add or create new HDL files and synthesize them, whereas a post-synthesis project requires pre-synthesized files.

1-3-5. Select **RTL Project** since this project will be based on source code (rather than a netlist) (1).

1-3-6. Deselect the **Do not specify sources at this time** option (2), since you will be adding RTL files in the next instruction.

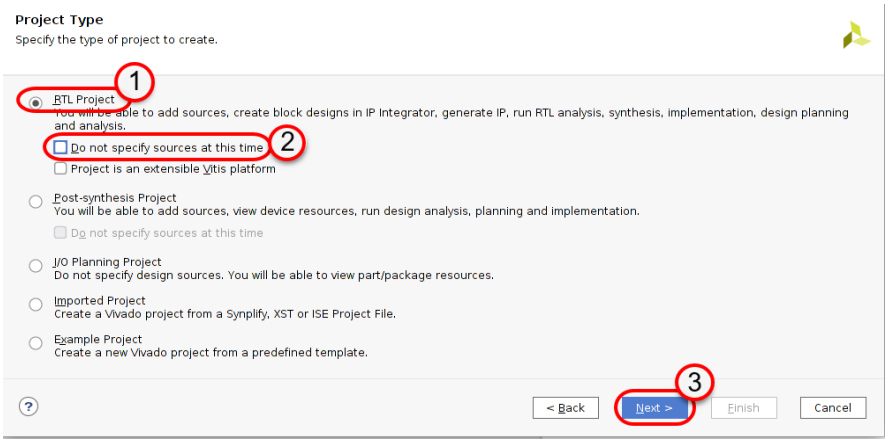


Figure 1-5: Setting the Project Type to RTL

1-3-7. Click **Next** to accept the selection and advance to the adding sources stage (3).

1-4. Now that the project name, type, and location have been entered, the wizard invites you to add source files to the project. You can add entire directories where the sources are located, add specific files, or create new sources.

Begin by adding the sources to your project.

1-4-1. Click the **Plus** icon (+) to begin adding objects to the project.

1-4-2. Select **Add Files** from the pop-up menu to begin adding your source files to the project.

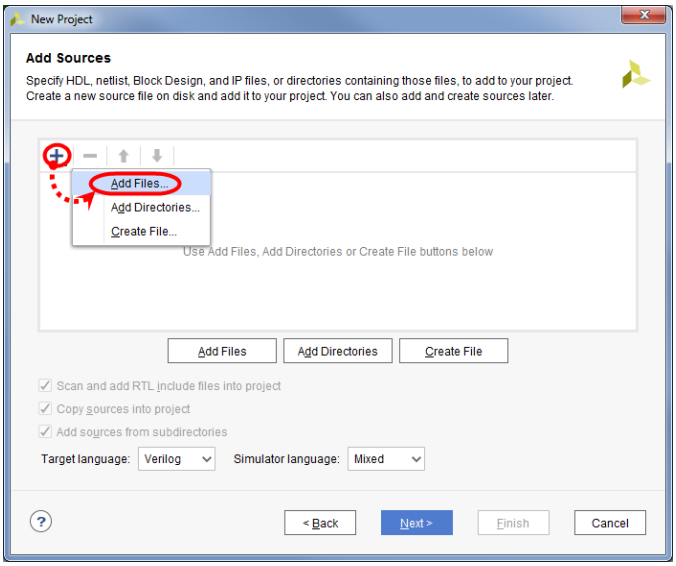


Figure 1-6: Adding Sources to the Project

After you add all the necessary files, the remainder of this dialog box will be addressed.

The Add Source Files dialog box opens.

1-4-3. Browse to the `$TRAINING_PATH/IO_PinPlanning/support` directory (1).

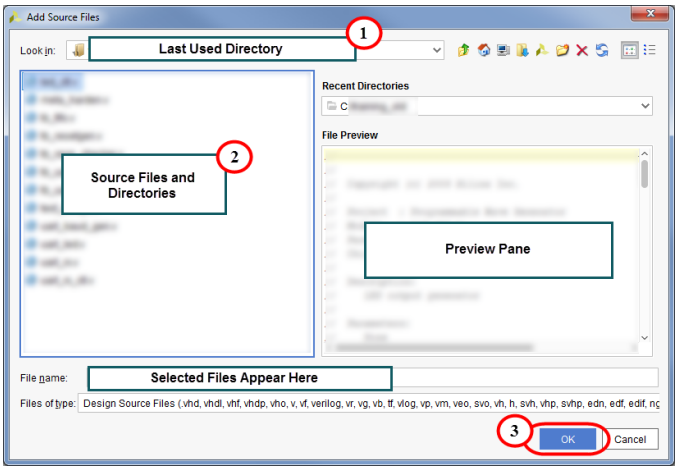


Figure 1-7: Adding Source Files

1-4-4. **Verilog users:** Select or multi-select **led_ctl_IO_PinPlanning.v, meta_harden_IO_PinPlanning.v, uart_baud_gen_IO_PinPlanning.v, uart_led_IO_PinPlanning.v, uart_rx_IO_PinPlanning.v, uart_rx_ctl_IO_PinPlanning.v** (2).

VHDL users: Select or multi-select **led_ctl_IO_PinPlanning.vhd, meta_harden_IO_PinPlanning.vhd, uart_baud_gen_IO_PinPlanning.vhd, uart_led_IO_PinPlanning.vhd, uart_led_pkg_IO_PinPlanning.vhd, uart_rx_IO_PinPlanning.vhd, uart_rx_ctl_IO_PinPlanning.vhd**.

1-4-5. Click **OK** to accept the selected files and add them as sources to the project (3).

The Add Source Files dialog box closes and returns you to the Add Sources dialog box.

1-4-6. Make sure that **Copy sources into project** is selected and that **Scan and add RTL include files into project** is not selected (1).

1-4-7. **Verilog users:** Make sure the **Verilog** is selected as the target language (2).

VHDL users: Make sure the **VHDL** is selected as the target language.

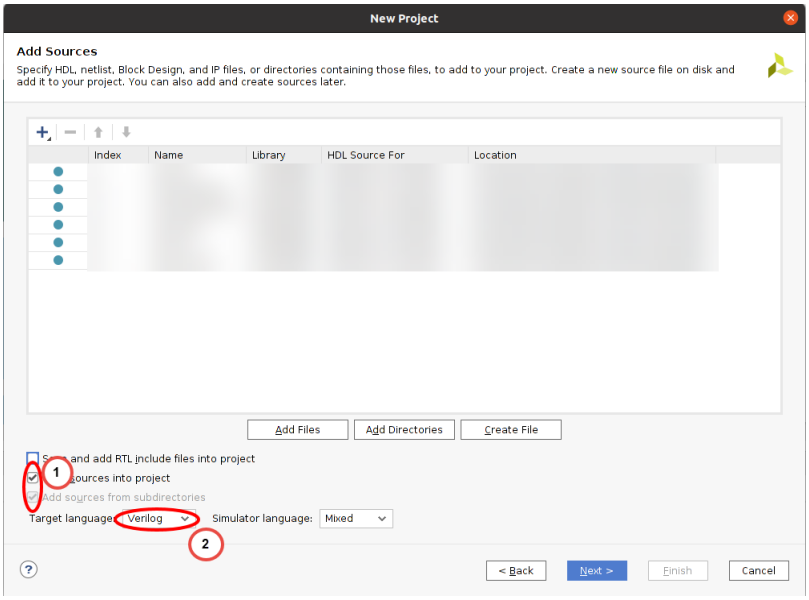


Figure 1-8: Setting Options in the Add Sources Window

1-4-8. Click **Next**.

1-5. Add any existing constraint files to the Vivado Design Suite project.

You will add uart_led_IO_PinPlanning.xdc to the project in the following instructions.

1-5-1. Click the **Plus** icon (+) to select the type of object you want to import (1).

1-5-2. Select **Add Files** to open the Add Constraints Files dialog box.

1-5-3. Browse to the `$TRAINING_PATH/IO_PinPlanning/support` directory.

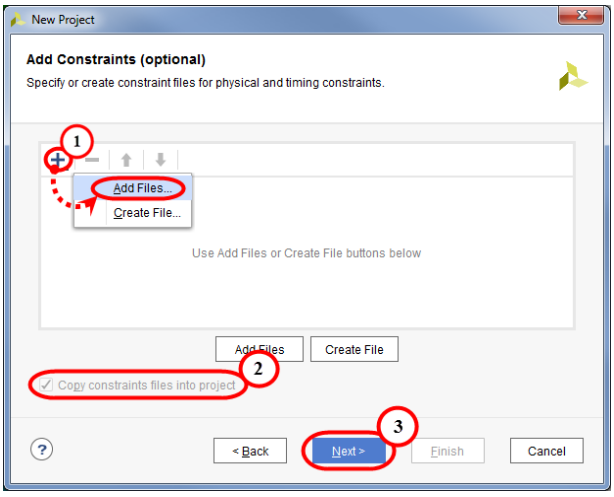


Figure 1-9: Adding Constraints

1-5-4. Select **uart_led_IO_PinPlanning.xdc**.

1-5-5. Click **OK** to accept the selected files and add them as sources to the project.

If you have additional files located in other directories you can repeat this instruction for each directory.

1-5-6. Confirm that the **Copy constraints files into Project** option is selected (2).

This will make a local copy of the source in the project space. Selecting this option enables you to make changes to the local copy without affecting the original source file.

1-5-7. Click **Next** to advance to selecting a target device (3).

1-6. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.

1-6-1. Select **Boards** from the Select area (1).

1-6-2. Select **All** from the Vendor drop-down list in the Filter area (2).

This filters the available boards to those that are populated with any member of the selected library.

1-6-3. Select **Kintex UltraScale KCU105 Evaluation Platform** from the board list.

Alternatively, you can select the board directly from the list at any time while in this dialog box.

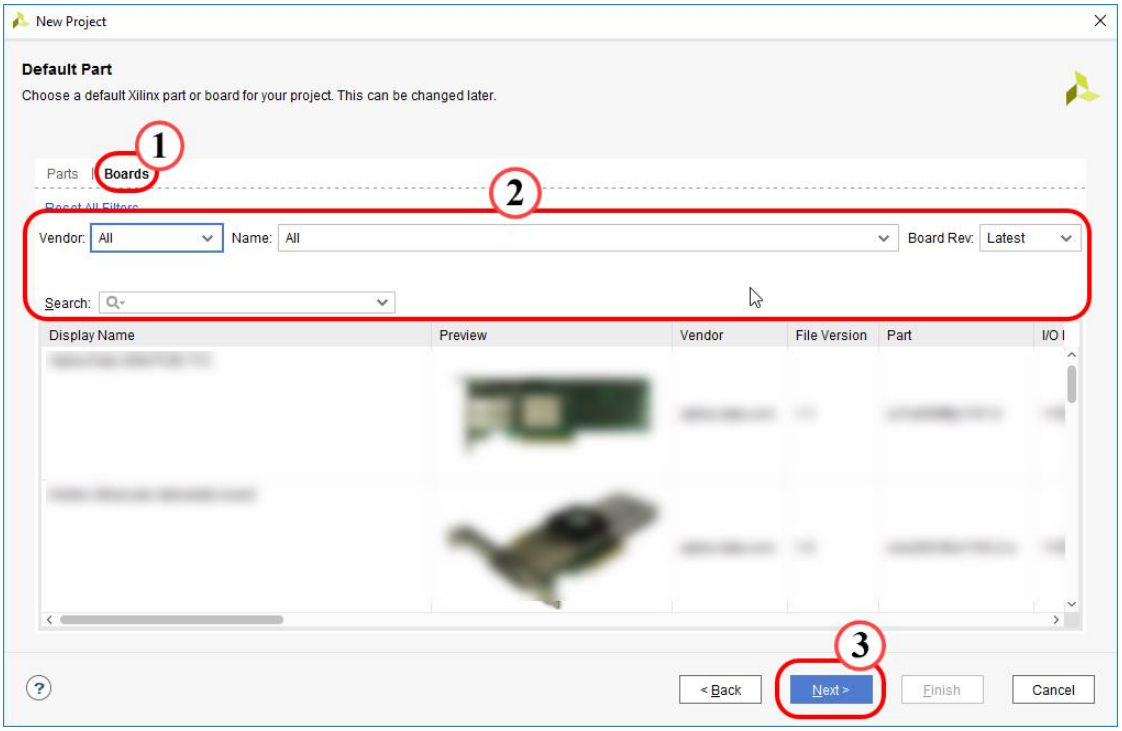


Figure 1-10: Selecting the Board for the Project

1-6-4. Click **Next** to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box and making the correction, or you can create the project now and edit the project properties, add or remove files, etc. later.

1-6-5. Click **Finish**.

Your project is constructed and you are presented with the Vivado Design Suite main workspace environment.

Analyzing the Design Using the Schematic and Hierarchy Views Step 2

You will open the elaborated design and briefly explore the design with the RTL Netlist, Schematic, and Hierarchy viewers.

2-1. Open the elaborated design.

2-1-1. Click **Open Elaborated Design** under RTL Analysis in the Flow Navigator to elaborate the design.

The elaborated RTL design enables various analysis views, including RTL Netlist, Schematic, and Graphical Hierarchy. These views have a cross-select feature, which allows you to debug and optimize the RTL.

The Elaborate Design dialog box opens.

2-1-2. Click **OK** in the dialog box to open the elaborated design.

2-1-3. If the schematic is not already open, click **Schematic** under RTL Analysis > Elaborated Design in the Flow Navigator to open the RTL schematic of the design in the main window.

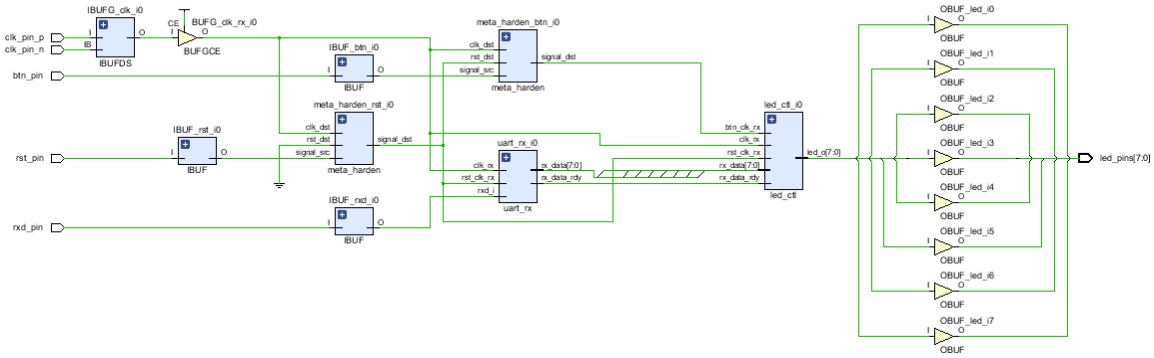


Figure 1-11: RTL Schematic of the uart_led Design (KCU105)

2-2. Explore the design with the Schematic view.

2-2-1. Select any logic instance in the Schematic view.

2-2-2. Right-click the instance and select **Go to Source**.

Notice that the RTL source file is opened in the text editor with the logic instance highlighted.

Also notice that the logic instance is also highlighted in the RTL Netlist window.

2-2-3. Double-click the **uart_rx_i0** logic instance in the Schematic view.

The subcomponents of the uart_rx_i0 are displayed.

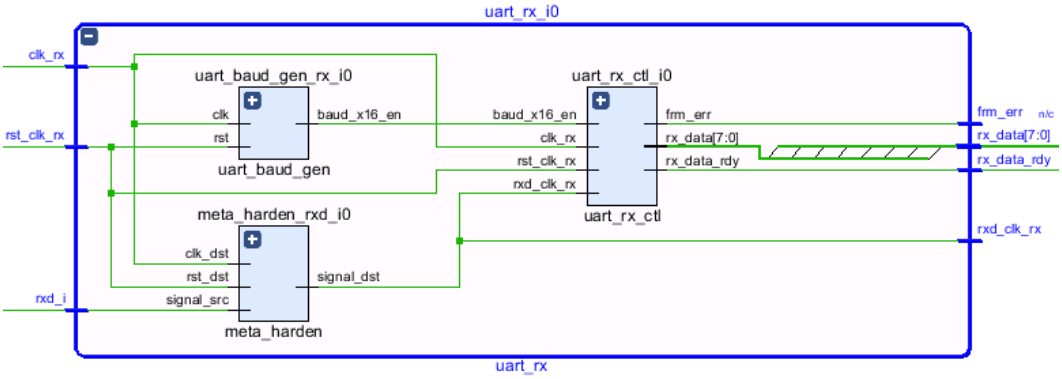


Figure 1-12: Viewing the Submodules of the uart_rx_i0 Component

2-2-4. Explore the RTL Schematic view by expanding each of the submodules and viewing the lowest level of the RTL schematic.

2-3. View the design hierarchy by using the Hierarchy view.

2-3-1. Select the **uart_baud_gen_rx_i0** logic instance in the Schematic view.

2-3-2. Select **Tools** > **Show Hierarchy**.

The RTL hierarchy window opens, displaying the hierarchical block structure of the selected *uart_baud_gen_rx_i0* instance.

2-3-3. Enable the **Show Tree View** icon from the Hierarchy window horizontal toolbar.

This displays and highlights the structure of the *uart_baud_gen_rx_i0* logic instance until the bottom leaf cells are present in the hierarchy.

2-3-4. Click the **Go Up One Level** icon until you are at the top level of the *uart_baud_gen_rx_i0* instance.

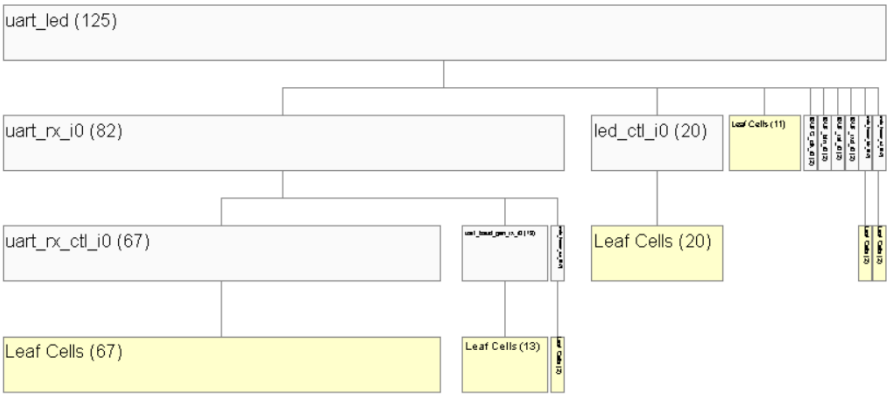


Figure 9-13: RTL Hierarchy View with uart_baud_gen_rx_i0 Highlighted (KCU105) (Numbers May Vary)

2-3-5. Explore the RTL Hierarchy view and see how the component selection in this view automatically highlight the same module in other views (Schematic view or Netlist window).

Question 1

From the top level, how many levels of hierarchy did you go through before reaching the lowest level? What can you tell from the design tree?

2-4. Close both the RTL Hierarchy and Schematic views.

2-5. Run DRC checks on the elaborated design.

2-5-1. Click **Report DRC** under RTL Analysis > Open Elaborated Design in the Flow Navigator.

The Report DRC dialog box opens.

2-5-2. Select **default** in the Vivado Rule Decks section.

2-5-3. Observe the categories in the **Rules** section.

2-5-4. Click **OK** to generate the DRC report.

2-5-5. Review the DRC violations in the DRC report.

Question 2

What are all the critical warnings reported by DRC? If so, what could be the reason(s)?

2-5-6. Right-click the **DRC** tab after reviewing the violations and select **Close**.

Placing Pins Using the I/O Planning View — Step 3

It is clear from the DRC report that there are three ports that do not have user-assigned LOC constraints and have default IOSTANDARD values. Here you will use the I/O Planning view to place the unplaced pins in the design.

3-1. Use the I/O Planning view to identify pins that do not have an assigned location.

3-1-1. Select **Layout** > **I/O Planning** from the layout selector, if it is in any other layout view.

3-1-2. Select the **Package** view, if it is not already opened.

The main window of the I/O Planning view displays the Package view of the previous Kintex® UltraScale™ device.

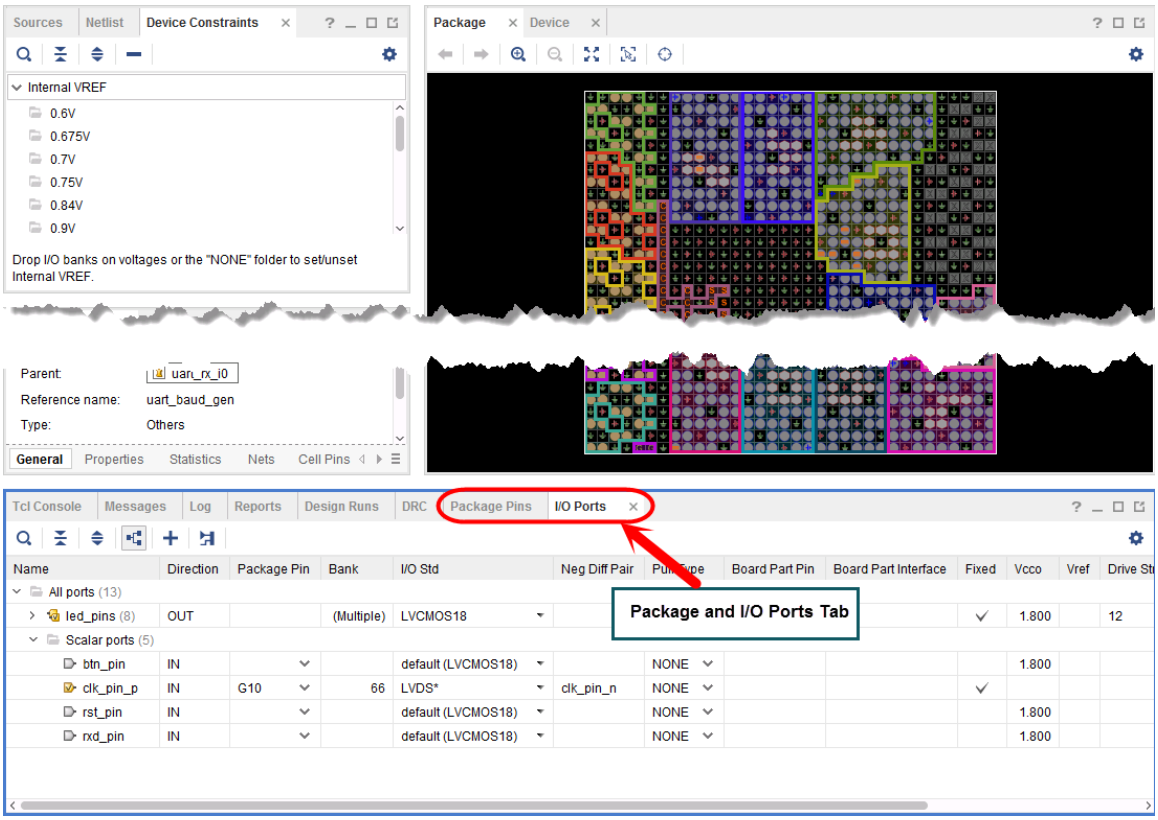


Figure 1-14: I/O Planning View

3-1-3. Observe the **I/O Ports** tab, which displays the list of I/O ports of the design, and the **Package Pins** tab, which displays the list of package pins available on the device package.

3-2. Assign the location and I/O standard for *btn_pin*.

3-2-1. Expand **Scalar ports** in the I/O Ports tab.

3-2-2. Select the **btn_pin** signal.

3-2-3. Select **AE10** (KCU105) under the Package Pin column.

3-2-4. Explicitly select **LVCMOS18** (KCU105) from the I/O Standard drop-down list.

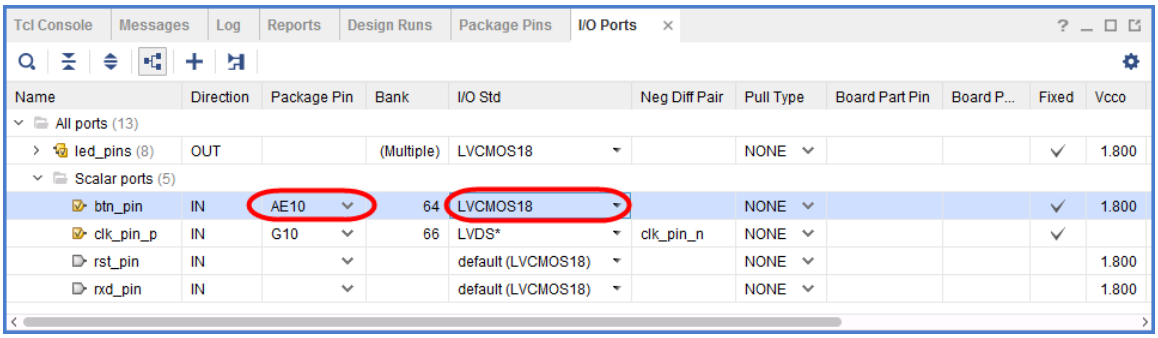


Figure 1-15: I/O Port Properties for btn_pin (KCU105)

3-3. Assign the location and I/O standard for *rxd_pin*.

3-3-1. Select **rxd_pin** in the I/O Ports view.

3-3-2. Select **G25** from under the Package Pin column.

3-3-3. Enter the following in the I/O Port Properties view.

- I/O Standard: **LVCMOS18**
- Leave the Drive Strength at its default value; drive strength does not apply to input pins.

3-4. Assign the location (using the Package view) and I/O standard for *rst_pin*.

3-4-1. Select **rst_pin** in the I/O Ports view.

3-4-2. Select **LVCMOS18** from the I/O Standard drop-down list and press <**Enter**>.

Rather than entering the package pin in the I/O Ports view, you can use the Package view to assign the package pin for this pin.

3-4-3. Drag the **rst_pin** from the I/O ports view onto location **AN8** in the Package view.

3-5. Save the I/O pin assignments to the targeted XDC file.

3-5-1. Select **File** > **Constraints** > **Save** to save the constraints.

The Save Constraint dialog box opens.

3-5-2. Select the **Select an existing file** option.

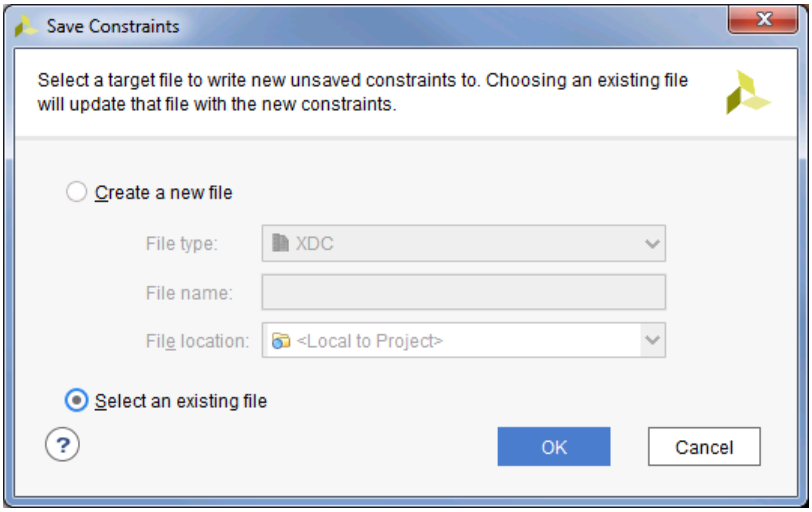


Figure 1-16: Selecting uart_led_IO_PinPlanning.xdc as the Target Constraints File

3-5-3. Click **OK**.

All these placement constraints will be saved to the existing **uart_led_IO_PinPlanning.xdc** file.

3-6. Open the *uart_led_IO_PinPlanning.xdc* file to confirm that your new pin assignments have been saved in the XDC file.

3-6-1. Select the **Hierarchy** tab of the Sources window.

3-6-2. Double-click the **uart_led_IO_PinPlanning.xdc** under **Constraints** > **constrs_1** to open the file in the text editor.

3-6-3. Scroll to the bottom of the file and validate the changes that you made in the Vivado IDE.

3-7. Rerun DRC to check for any violations.

This time, no IOB violations (Unconstrained Logical Ports) should be reported.

3-8. Close the elaborated design.

3-8-1. Select **File** > **Close Elaborated Design** to close the elaborated design.

The Confirm Close dialog box opens.

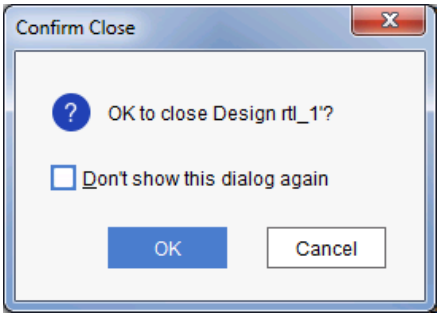


Figure 1-17: Confirm Close Dialog Box

3-8-2. Click **OK**.

3-9. Close the project.

3-10. Close the Vivado Design Suite.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/IO_PinPlanning` directory.

3-11. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

3-11-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/IO_PinPlanning`.

3-11-2. Select **`IO_PinPlanning`**.

3-11-3. Press <**Delete**>.

-- OR --

Using the command line:

3-11-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

3-11-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/IO_PinPlanning`

[Linux users]: `rm -rf $TRAINING_PATH/IO_PinPlanning`

Summary

In this lab, you checked DRC for violations and fixed the violations by using the I/O planning capability of the Vivado IDE to assign ports to package pins.

Answers

1. From the top level, how many levels of hierarchy did you go through before reaching the lowest level? What can you tell from the design tree?

   Three levels. Compare the design tree view hierarchy with the Sources view hierarchy. The complete design hierarchy is preserved and the lowest level of hierarchy displays the RTL schematic of the submodule.

2. What are all the critical warnings reported by DRC? If so, what could be the reason(s)?

   The DRC has displayed the following critical warnings:

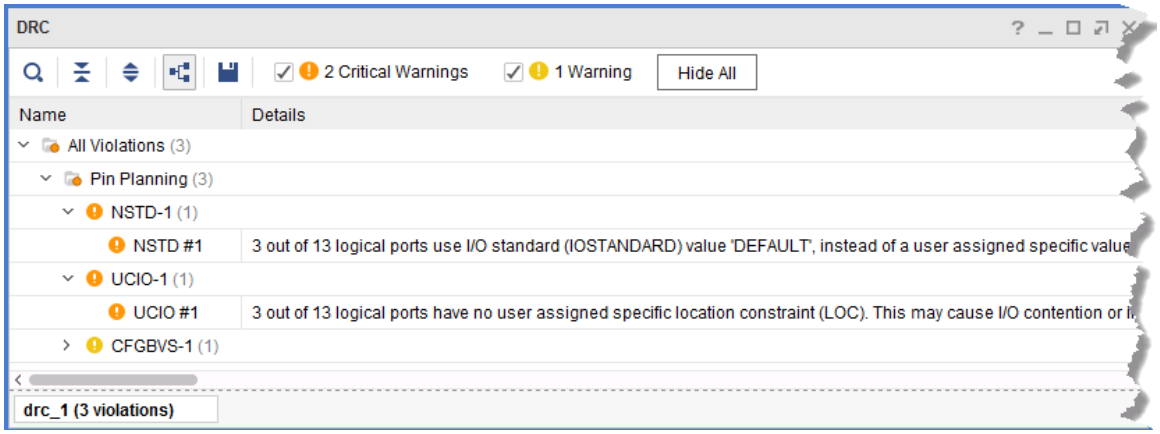


Figure 1-18: DRC Violations Found

   The violations are due to the absence of any I/O standard and package pin constraints for some of the input ports of the design.

Lab 2: Xilinx Power Estimator Spreadsheet

2022.2

Abstract

This lab introduces the Xilinx Power Estimator (XPE) spreadsheet, which is power estimation tool typically used in the pre-design and pre-implementation phases of the project.

Note: This XPE lab is only supported on the Windows platform with Excel 2007 or higher and not on Linux. CloudShare users can take the provided starting point XPE spreadsheet from the CloudShare environment and use it to work from their local Windows machine.

This lab should take approximately 40 minutes.

Objectives

After completing this lab, you will be able to:

- Estimate the resources required for a design, based on the high-level design description
- Enter the amount of resources required for the design into the XPE
- Enter the default activity rates for the components used
- Evaluate the estimated power calculated by the XPE

Introduction

The Xilinx Power Estimator (XPE) assists with architecture evaluation, device selection, appropriate power supply components, and thermal management components specific to your application.

The XPE considers your design resource usage, toggle rates, I/O loading, and many other factors which it combines with the device models to calculate the estimated power distribution. The device models are extracted from measurements, simulation, and/or extrapolation.

The accuracy of the XPE is dependent on two primary sets of inputs:

- Device utilization, component configuration, clock, enable, and toggle rates, and other information you enter into the tool
- Device data models integrated into the tool

For accurate estimates of your application, enter realistic information that is as complete as possible. Modeling a certain aspect of the design conservatively or without sufficient knowledge of the design can result in unrealistic estimates.

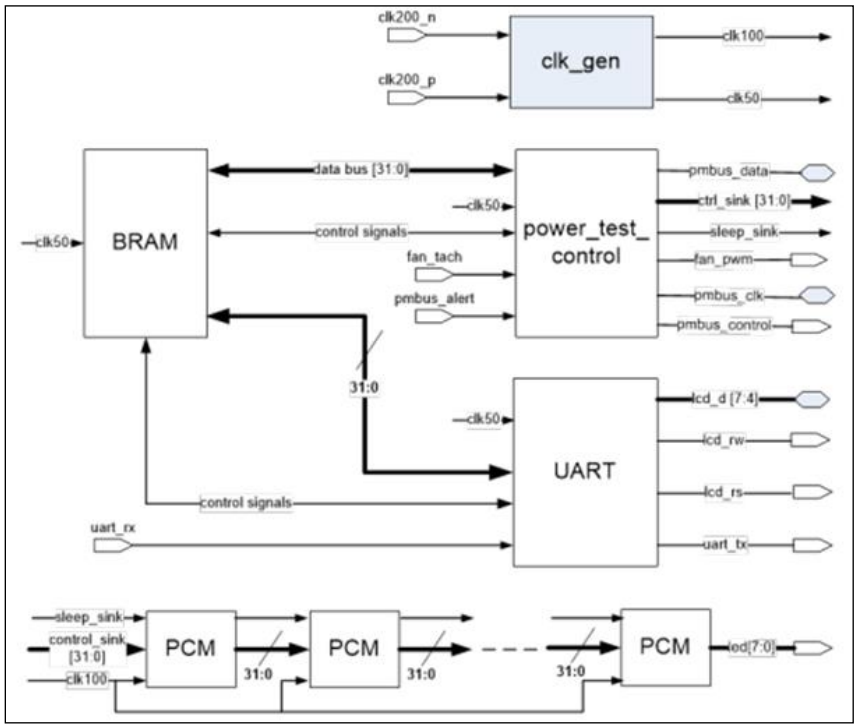


Figure 2-1: Example Block Diagram

The design also has power consuming modules instantiated 128 times and a UART module that sends/receive commands with the host PC at a 115200 baud rate.

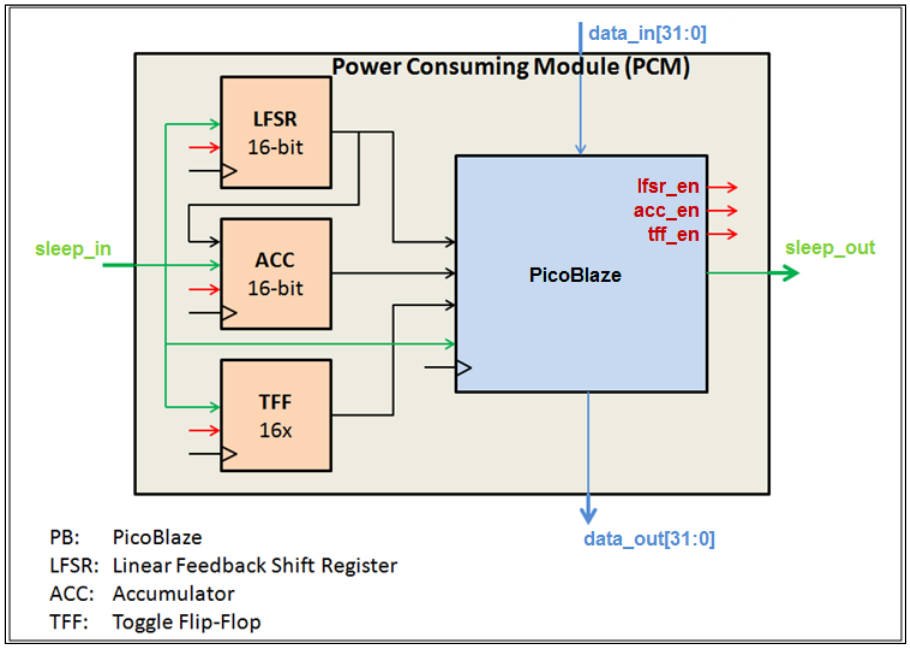


Figure 2-2: Power Consuming Module

clk_gen module: This module generates two clocks: 100 MHz and 50 MHz. The input clock to this module is from the clk200_p and clk200_n differential clock pair on the KCU105 board. The differential clock inputs are supplied to the MMCM, which generates clk100 and clk50 using the MMCM. The clk100 is used by the Power Consuming Module (PCM) while the remaining modules in design are driven by clk50.

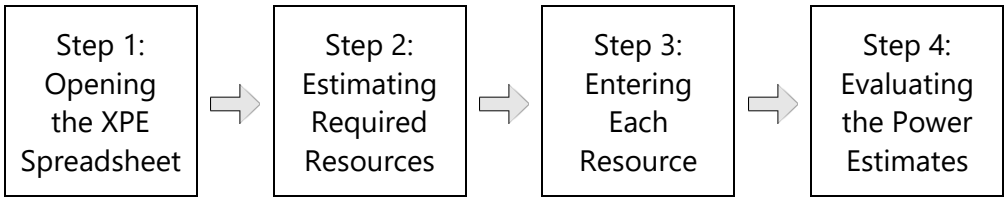
uart_control module: This module consists of UART_RX, UART_TX, and the PicoBlaze processor. UART_RX is used to receive the command from the host. UART_TX will send the information corresponding to the host command decoded by the processor.

Power Consuming Module (power_sink): This module consists of a pseudo-random number generator, 16-bit accumulator, 16-bit toggle shift register and another PicoBlaze processor. The program executed by the PicoBlaze processor (KCPSM6) can therefore be written to emphasize the use of certain features at different times either to adjust power consumption in general or to evaluate the relative power consumption of different resources.

power_test_control module: The PicoBlaze processor (KCPSM6) implements a PMBus protocol to communicate with the UCD9248 power supply controller (Texas Instruments) on the KCU105 board. This will be used to read the die temperature, power consumed by the device, and supply voltages.

Note: This XPE lab is only supported on the Windows platform and not on native Linux nor the VirtualBox environment.

General Flow



Opening the XPE Spreadsheet — Step 1

1-1. Open the spreadsheet targeted for the Kintex® UltraScale™ device and make sure that your Microsoft Excel settings allow macro extensions.

1-1-1. Open the **UltraScale_XPE.xlsm** XPE spreadsheet located in the `C:\training\XPE\lab\KCU105` directory.

The latest updated spreadsheets for various FPGAs will be available at www.xilinx.com/products/technology/power/xpe.html.

This lab uses the 2020.1 version of XPE, targeting the UltraScale boards. This XPE version is compatible with the latest version of the Vivado tools.

In Excel, the macro security level is set to High by default, which disables macros.

1-1-2. Change the macro security level if needed:

Microsoft Excel 2007 users:

- Click **Options** from the Security Warning.
- Select **Enable this content** and then click **OK**.

Figure 2-3: Enabling the Macro Settings in Excel

Microsoft Excel 2010 or 2013 users:

- Select **File** > **Options** > **Trust Center** > **Trust Center Settings**.
- Select the **Macro Settings** category. Under Macro Settings, select **Enable all macros** and click **OK**.
- Click **OK**.

1-1-3. Review all the available worksheets in the Xilinx Power Estimator (XPE) spreadsheet.

- Summary
- Snapshot
- Graphs
- IP_Manager
- Clock
- Logic
- IO
- BRAM
- DSP
- CLKMGR
- GTX
- Other
- User
- Release

Question 1

What information is provided in the Clock, Logic, IO, BRAM, DSP, CLKMGR, and GTX worksheets?

Estimating the Resources Required from the High-Level Design Step 2

Power estimation for programmable devices like FPGAs is a complex process, since it is highly dependent on the amount of logic in the design and the configuration of that logic. To supply the minimum input that will allow XPE to estimate power with reasonable accuracy, you need the following:

- A target device-package-grade combination
- A good estimate of resources you expect to use in the design (for example, flip-flops, look-up tables, I/Os, block RAMs, MMCMs, and PLLs)
- A clock frequency or frequencies for the design
- An estimate of the data toggle rates for the design
- The external memory and transceiver-based interfaces with their data rates for the design
- The thermal environment in which the design will be operating
- As a general rule, input as much information about your design as available, then leave the remaining settings to default values. This strategy will allow you to determine the device power supply and heat dissipation requirements.

Based on the lab design description, you will be estimating the resources required for this design.

2-1. Configure the Device and Environment settings.

2-1-1. Enter **early_stage_estimate** as the Project in the Summary worksheet.

2-1-2. Note down the default On-Chip Power and Junction Temperature in the Summary section.

2-1-3. Update the device and environment settings in the Summary worksheet with the following values:

- Device
  - Family: **Kintex UltraScale**
  - Device: **XCKU040**
  - Package: **FBVA900**
  - Speed Grade: **-2**
  - Temp Grade: **Industrial**
  - Process: **Typical**

- Environment
  - Ambient Temp: **30° C**
  - Airflow: **250 LFM**
  - Heat Sink: **Medium Profile**
  - # of Board Layers: **16 or more**

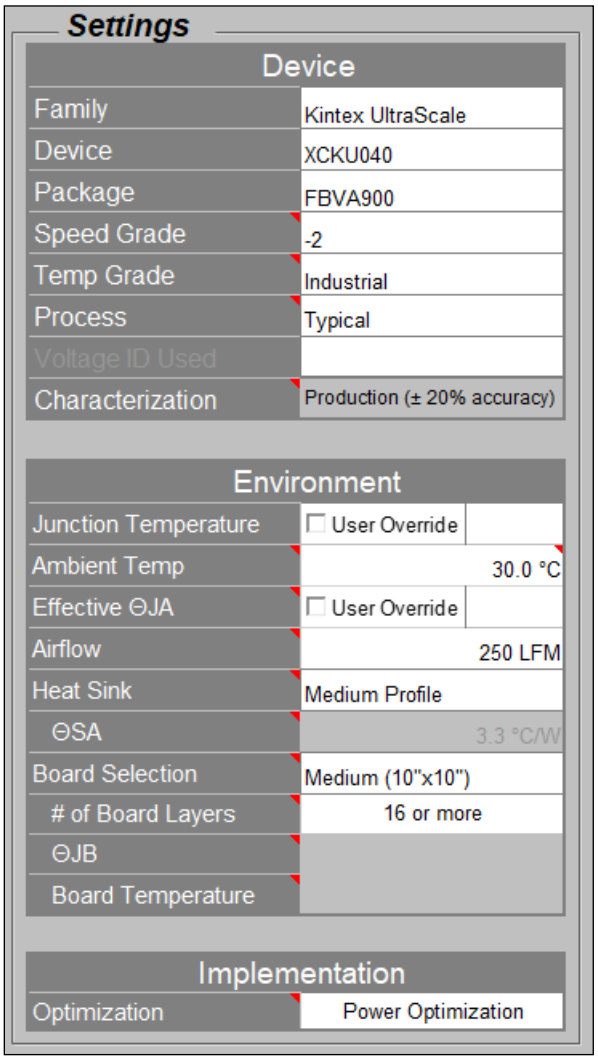


Figure 2-4: Device and Environment Settings for the Board

Note: FPGA quiescent power is highly affected by environment settings, so it is important to match your board's environment with the board data as precisely as possible. The power tools can be very useful to perform the analysis and define the best thermal strategy for the board.

2-2. Configure the implementation setting.

2-2-1. Set the implementation setting to **Power Optimization**.

The power optimization option in the Implementation setting minimizes the core dynamic power.

2-2-2. Save the file.

Question 2

Based on the Device characterization under settings (Production, Advance, and Preliminary), what are your expectations about the quality of power estimation?

2-3. Estimate the resources required for this design.

Estimates in the early stages of design may change because the specifications and algorithms may change frequently. Generally, prior knowledge about device architecture, design modules, and IPs used in the design is helpful when estimating the resources.

From the block diagram of the Power Consuming Module, you may observe that the PCM is composed of one PicoBlaze processor and flip-flops. To obtain the resource estimation for PicoBlaze processor, you can refer to the PicoBlaze user guide available at www.xilinx.com/picoblaze.

PicoBlaze processor: 26 slices (104 LUTs and 82 FFs)

LFSR/Toggle FFs/ACC: 48 FFs

Processor memory: 16 distributed RAM and 16 FFs

Approximate additional/glue logic (control signals to handles LFSR/Toggle FFs/ACC, reading/writing data of processor): 140 LUTs, 80 FFs

Total number of resources for one PCM: 274 FFs, 244 LUTs, and 16 distributed RAMs

In this design, the total number of resources for 128 PCMs (approximately) is 35072 FFs, 31232 LUTs (includes LUTs and distributed ROM), and 2048 distributed RAMs

In addition to PCMs, the design has uart_control and power_test_control modules, which use approximately 1000 FFs, 1000 LUTs, and 64 distributed RAMs.

2-3-1. Select the **Logic** worksheet tab.

2-3-2. Enter **100 MHz** in E12 and **50 MHz** in E13.

2-3-3. Enter the calculated data as shown in the figure below.

Name	Clock (MHz)	LUTs as Logic	LUTs as Shift Registers	LUTs as Distributed RAMs	Registers	Toggle Rate	Routing Complexity	Signal Rate (Mtr/s)	Power (W)
	100.0	31232		2048	35072	12.5%	8.00	12.5	**0.145**
	50.0	1000		64	1000	12.5%	8.00	6.3	**0.002**

Figure 2-5: Entering Estimated Logic Resources

2-3-4. Save the file.

2-3-5. Switch to Summary worksheet and review the increase in On-chip Power and Junction Temperature after adding the resources.

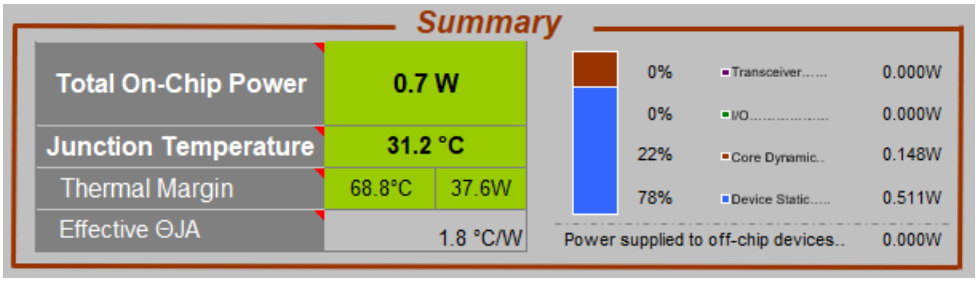


Figure 2-6: Summary Section After Adding Resources

Entering the Amount of Each Resource — Step 3

Enter the number of resources required, such as clock, block RAM, and MMCM. For an accurate XPE estimation, you will have to estimate the amount of device resources that your design will use. Providing accurate toggle rates in the various XPE sheets is essential to obtain quality power estimates. This information, however, might not be readily available at the stage in the design cycle where you enter data in XPE. Activity might be refined as the design gets more defined.

The following guidelines can help you enter design toggle activity:

- For synchronous paths, the toggle rate reflects how often an output changes relative to a given clock input and can be modeled as a percentage between 0–100%. The max data toggle rate of 100% means that the output toggles every active clock edge. When data changes twice per clock cycle, enter 200% for the toggle rate.
  - For example, consider a free running binary counter with a 100-MHz clock. For the least significant bit you would enter 100% in the Toggle Rate column because this bit toggles every rising edge of the clock. For the second bit you would enter 50% because this bit toggles every other rising edge of the clock.

- For non-periodic or event-driven portions of designs, toggle rates cannot be easily predicted. An effective method of estimating average toggle rates for a given design is to segregate the different sections of the design based on their functionality or hierarchy and estimate the toggle rates for each of the sub-blocks. An average toggle rate can then be arrived at by calculating the average for the entire design or hierarchy. Most logic-intensive designs work at around 12.5% average toggle rate, which is the default toggle rate setting in XPE.
- It has been observed that designs with random data patterns as input generally have toggle rates between 10%–30%. However, designs with a lot of glitch logic can have toggle rates as high as or even higher than 50%. Glitch logic is generally classified as combinatorial functions which have a high probability of the output changing when any one input changes, such as XOR gates or unregistered arithmetic logic (i.e., adders). Functions that use large amounts of such logic, such as error detection/correction circuitry, might exhibit higher toggle rates due to this. Designs with large amounts of control path logic, such as embedded designs, on average have lower toggle rates due to large sections of logic being inactive at any given time during operation.

3-1. Enter the activity rates and clock frequency.

3-1-1. Select the **Summary** worksheet and double-click the **Set Default Rates** button.

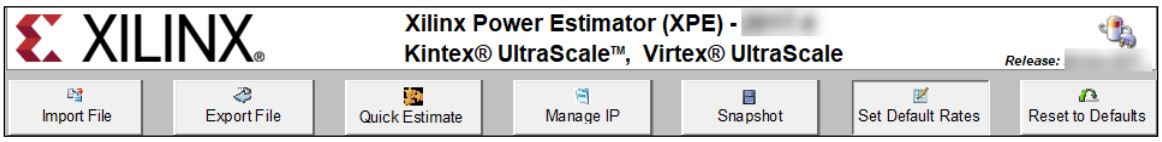


Figure 2-7: Set Default Rates Button

The details entered here are the default values for the LOGIC, I/O, BRAM, DSP, and other worksheets.

3-1-2. Set the Toggle rate to **20%** in the **Logic** field (for worst-case analysis). Set the rate to **100 MHz** in the All Clock Nets field.

3-1-3. Enter the default toggle rates for others as shown in the figure below and click **OK**.

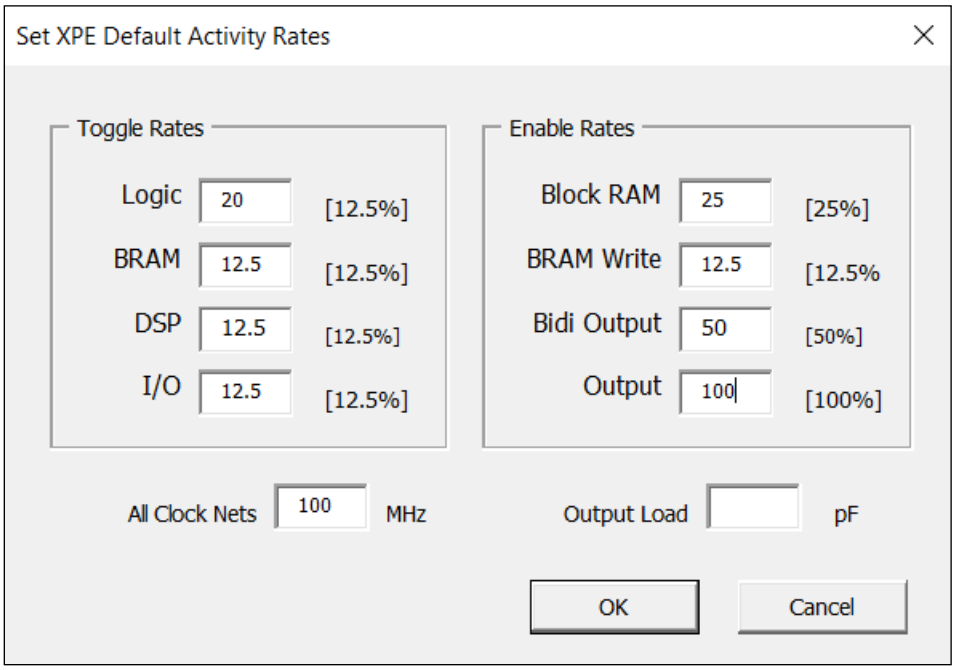


Figure 2-8: Setting the Toggle Rates and Clock Frequency

3-1-4. Review the Logic page after changing these parameters and verify that the toggle rate has changed from 12.5% to 20%. Update the clock in the E13 cell to 50 Mhz.

3-1-5. Record the changes in On-Chip Power and Junction Temperature in the Summary worksheet.

The Total On-Chip Power has been increased slightly.

3-1-6. Alter the toggle rate value to see the impact on power consumption.

Note: Typically, logic-intensive designs work at around 12.5% of the synchronizing clock (12.5% is the default value used in XPE). For a worst-case estimate, a toggle rate of 20% can be used. Average toggle rates greater than 20% are not very common.

Arithmetic-intensive modules of a design seem to take toggle rates of up to 50%, which is representative of the absolute worst case. An example of this would be a multiply-accumulate operation.

It is also common to model toggle rate for random input data at 50%. To appreciate what 100% toggle rate means, think of a constantly enabled toggle flip-flop (TFF) whose data input is tied High. The T-output of this flip-flop toggles every clock edge. Very few designs could have an average rate that high (100%).

3-2. Enter the clock information.

3-2-1. Select the **Clock** sheet.

3-2-2. Enter **clk100** in the B10 cell (Name column) and enter the corresponding frequency and fanout in the respective cells.

3-2-3. Enter **clk50** in the B11 cell (Name column) and enter the corresponding frequency and fanout in the respective cells as shown below.

Name	Frequency (MHz)	Type	Fanout	Clock Buffer Enable	Slice Clock Enable	Power (W)
clk100	100.0	Global	36000	100%	50%	**0.097**
clk50	50.0	Global	1000	100%	50%	**0.003**
		Global		100%	50%	0.000

Figure 2-9: Entering Clock Information

3-2-4. Review the Total On-Chip Power and Junction Temperature in the Summary worksheet after adding clock information.

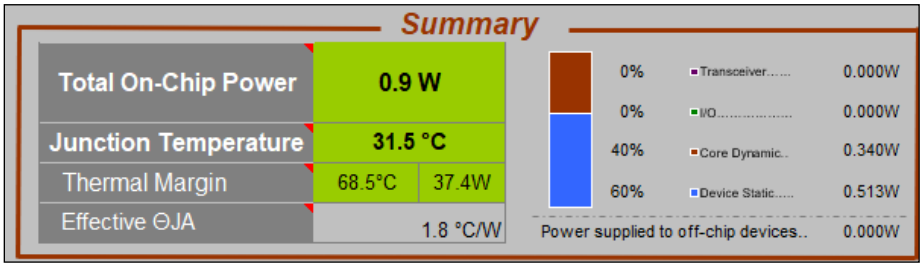


Figure 2-10: Summary Section After Adding Clock Information

3-3. Enter the input, output, and bidirectional pin details.

Signal Name	Direction	Size	I/O Standard
clk200_p	Input	1	LVDS_18
clk200_n	Input	1	LVDS_18
cpu_rst	Input	1	LVCMOS18
led	Output	7:0	LVCMOS18
lcd_rs	Output	1	LVCMOS18
lcd_rw	Output	1	LVCMOS18
lcd_e	Output	1	LVCMOS18
lcd_d	Inout	7:4	LVCMOS18

Signal Name	Direction	Size	I/O Standard
uart_tx	Output	1	LVCMOS18
uart_rx	Input	1	LVCMOS18
pmbus_clk	Inout	1	LVCMOS18
pmbus_data	Inout	1	LVCMOS18
pmbus_control	Output	1	LVCMOS18
pmbus_alert	Input	1	LVCMOS18
fan_tach	In	1	LVCMOS18
fan_pwm	Out	1	LVCMOS18

Top-Level Input/Output Signals

The table does not include all the information like I/O Bank, I/O Settings, and Clock Frequency. Fill the data in the spreadsheet as shown in the figure below.

3-3-1. Select the **IO** worksheet.

3-3-2. Enter the number of input pins, output pins, and I/O standards from the Top-Level Input/Output ports table.

3-3-3. Enter the clock as **200 MHz** for clk200.

3-3-4. Save the file.

The top-level input/output ports are shown in the following figure.

Name	Bank	I/O Settings								Activity				
	I/O Type	I/O Standard	Input Pins	Output Pins	Bidir Pins	BITSLICE	IBUF	Input Term	Output Impedance	Pre-Emphasis	Clock (MHz)	Toggle Rate	Data Rate	Output Enable	Term Disable
clk200_p	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				200.0	12.5%	SDR		
clk200_n	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				200.0	12.5%	SDR		
cpu_rst	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				100.0	12.5%	SDR		
led	HP	LVCMOS 1.8V 12mA (Slow)		8		No	Low Power				100.0	12.5%	SDR	100.0%	
lcd_rs	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
lcd_rw	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
lcd_e	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
lcd_d	HP	LVCMOS 1.8V 12mA (Slow)			4	No	Low Power				100.0	12.5%	SDR	100.0%	
uart_tx	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
uart_rx	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				100.0	12.5%	SDR		
pmbus_clk	HP	LVCMOS 1.8V 12mA (Slow)			1	No	Low Power				100.0	12.5%	SDR	100.0%	
pmbus_data	HP	LVCMOS 1.8V 12mA (Slow)			1	No	Low Power				100.0	12.5%	SDR	100.0%	
pmbus_control	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
pmbus_alert	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				100.0	12.5%	SDR		
fan_tach	HP	LVCMOS 1.8V 12mA (Slow)	1			No	Low Power				100.0	12.5%	SDR		
fan_pwm	HP	LVCMOS 1.8V 12mA (Slow)		1		No	Low Power				100.0	12.5%	SDR	100.0%	
	HP	LVCMOS 1.8V 12mA (Slow)				No	Low Power				100.0	12.5%	SDR		

Figure 2-11: Entering LVDS Pair Clock Frequency

The On-Chip power and Junction Temperature increases again by applying the I/O Pin details.

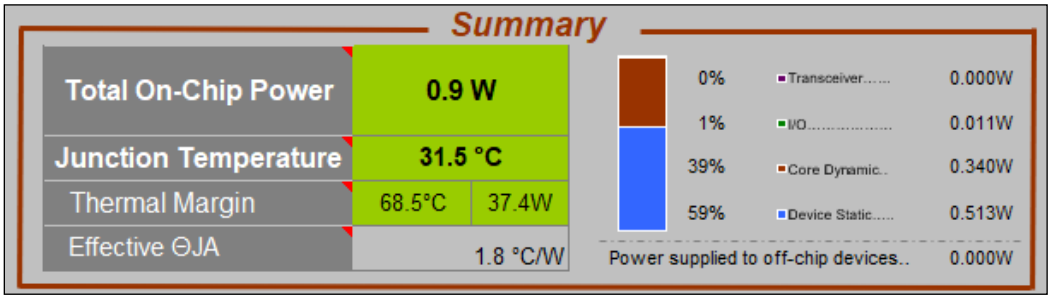


Figure 2-12: Summary Section After Adding I/O Pin Details

3-4. Provide the block RAM usage.

3-4-1. Select the **BRAM** worksheet.

3-4-2. Enter **1** in the Block RAMs column, select **RAMB36** from the Mode column, and **18** from the Bit Width column.

3-4-3. Select **WRITE_FIRST** from the drop-down list of **Write Mode** column as shown below for both Port A and Port B.

Name	Block RAMs	Cascade Group Size	Mode	Toggle Rate	Port A Clock (MHz)	Port A Enable Rate	Port A Bit Width	Port A Write Mode	Port A Write Enable	Port B Clock (MHz)	Port B Enable Rate	Port B Bit Width	Port B Write Mode	Port B Write Enable	Signal Rate (Mtr/s)	Power (W) V_{CCINT} 0.950V	Power (W) V_{CCBRAM} 0.950V
	1	4	RAMB36	50.0%	100.0	25.0%	18	WRITE_FIRST	50.0%	100.0	25.0%	18	WRITE_FIRST	50.0%	50.000	0.001	0.000
		4	RAMB18	50.0%	100.0	25.0%	1	NO_CHANGE	50.0%	100.0	25.0%	1	NO_CHANGE	50.0%	0.000	0.000	0.000

Figure 2-13: Entering Block RAM Information

3-4-4. Save the file.

Note: Since there are no DSP resources in the design, skip the DSP sheet.

3-5. Provide the MMCM usage and parameter information.

3-5-1. Select the **CLKMGR** worksheet.

3-5-2. Enter the information as shown in the following figure.

Name	MMCM or PLL	Clock (MHz)	Phase Shift	Divide Counter	Multiply Counter	Clock 0 Divide	Clock 1 Divide	Clock 2 Divide	Clock 3 Divide	Clock 4 Divide	Clock 5 Divide	Clock 6 Divide	Power Down	V_{CCINT} (W)	V_{CCAUX} (W)
MMCM_1	MMCM	200.0	None	1	5	10	20	off	off	off	off	off		0.000	0.114
			None	1	1	off	off	off	off	off	off	off		0.000	0.000

Figure 2-14: Entering MMCM Information

3-6. Review the IP Manager worksheet.

3-6-1. Select the **IP_Manager** worksheet.

3-6-2. Click the **Manage_IP** button and go through the Create IP tabs.

There is no need to add any IPs now. Just review the worksheet to create IP and close the **XPE IP Manager** tab.

Evaluating the Power Estimates — Step 4

4-1. Review the estimated power calculated by the tool.

4-1-1. Select the **Summary** sheet.

4-1-2. Review the On-Chip Power, Power Supply, and Summary sections.

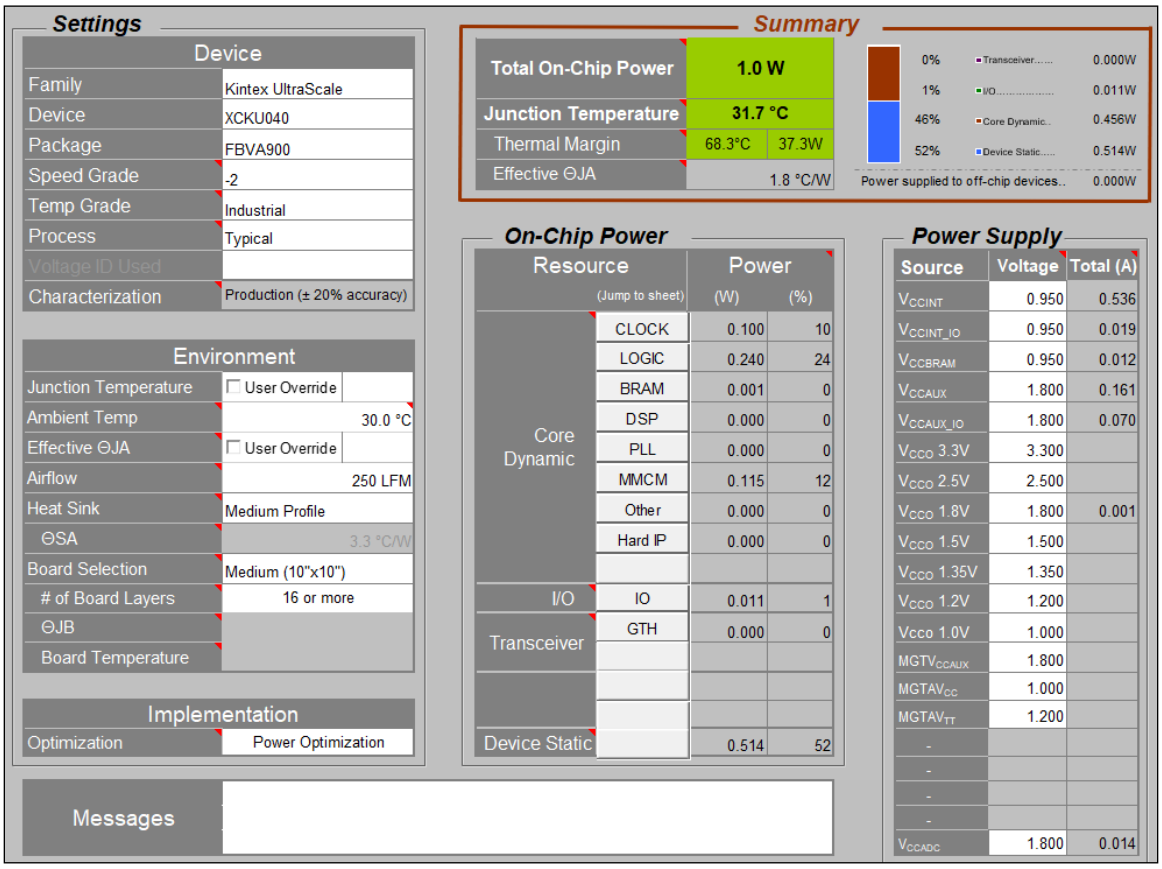


Figure 2-15: Summary of XPE Power Estimation

Question 3

Enter the data in the following tables:

Summary	XPE
Junction Temperature (C)	
Effective θJA (C/W)	
Total On-Chip Power (W)	

Summary of XPE

On-Chip Power (W)	XPE
CLOCK	
LOGIC	
BRAM	
DSP	
MMCM	
IO	
Device Static	

On-Chip Power of XPE

You should note that the logic (CLB) resources use the highest percentage of total power. Remember that their power consumption can be altered by redesigning so that more resources map to dedicated hardware (which uses less power) instead of logic.

You should also remember that this example provides only a rough estimate. The estimation improves with every step, as shown below.

- As the actual system is built and the amount of logic resources is actually known, the power estimate will get more accurate.
- Likewise, after the design has been routed, the power estimate will be more accurate.
- Finally, after implementation has been done and simulation data has been loaded (to estimate dynamic power consumption with an SAIF file) the power estimate will be the best.

You should also be able to verify (from the Summary) that dynamic power consumes a lot more power than static power. So if the clock frequencies used were slower, power consumption would decrease.

Power Supply Total Current (A)	XPE
Vccint	
Vccbram	
Vccaux	
Vcco1.8v	
Vccadc	

Power Supply of XPE

4-1-3. Select the **Graphs** worksheet.

4-1-4. Review the graphs.

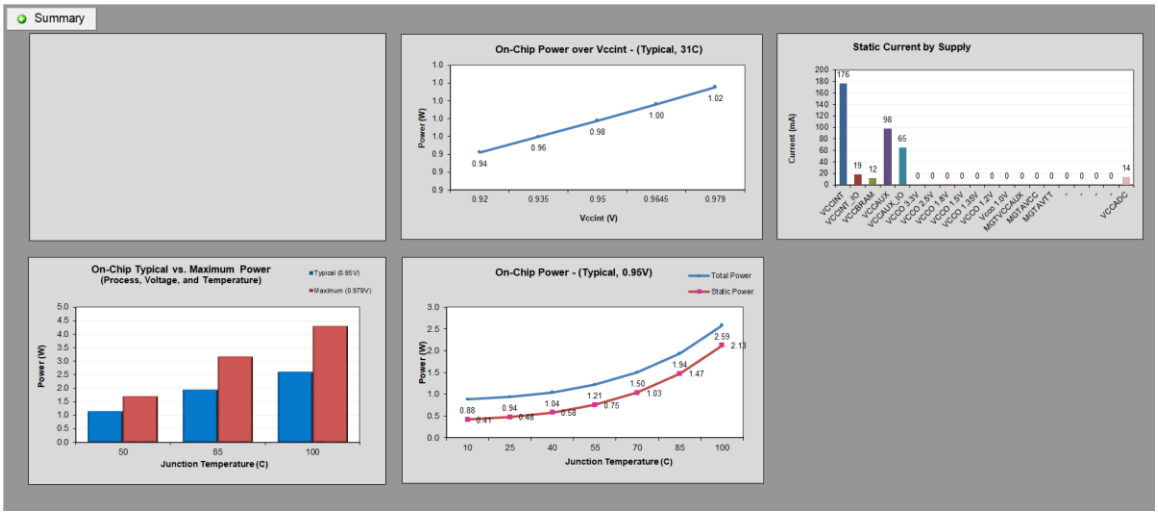


Figure 2-16: Graphical Summary of Estimated Power

4-2. Set up a power comparison using the Snapshots sheet.

4-2-1. Select the **Snapshot** worksheet.

4-2-2. Double click the **Snapshot** button.

4-2-3. Change Snapshot_1 to **Typical** in cell C3.

4-2-4. Save the file.

4-2-5. Select the **Summary** worksheet.

4-2-6. Change the Process to **Maximum** in the Settings section.

4-2-7. Select the **Snapshot** worksheet and double click the **Snapshot** button.

4-2-8. Change Snapshot_2 to **Maximum** in cell E3.

Settings	XPE: [redacted]	XPE: [redacted]
	4 Dec [redacted] @ 14:26	4 Dec [redacted] @ 14:27
	UltraScale_XPE_[redacted]	UltraScale_XPE_[redacted]
Part	XCKU040FBVA900-2I	XCKU040FBVA900-2I
Ambient Temperature	30.0 °C	30.0 °C
Process	Typical	Maximum
Implementation	Power Optimization	Power Optimization
Summary		
Total On-Chip Power	0.982 W	1.323 W
Junction Temperature	31.7 °C	32.3 °C
Effective ΘJA	1.8 °C/W	1.8 °C/W
On-Chip Power		
Clocking	0.215 W	0.215 W
Logic	0.240 W	0.240 W
I/O	0.011 W	0.011 W
BRAM	0.001 W	0.001 W
DSP	0.000 W	0.000 W
Transceiver	0.000 W	0.000 W
Device Static	0.514 W	0.856 W

Supply Summary	Voltage	Current	Voltage	Current
V_{CCINT}	0.950 V	0.536 A	0.950 V	2.650 A
V_{CCINT_IO}	0.950 V	0.019 A	0.950 V	0.061 A
V_{CCBRAM}	0.950 V	0.012 A	0.950 V	0.059 A
V_{CCAUX}	1.800 V	0.161 A	1.800 V	0.469 A
V_{CCAUX_IO}	1.800 V	0.070 A	1.800 V	0.193 A
V_{CCO} 3.3V				
V_{CCO} 2.5V				
V_{CCO} 1.8V	1.800 V	0.001 A	1.800 V	0.103 A
[illegible]				
V_{CCADC}	1.800 V	0.014 A	1.800 V	0.020 A

Figure 2-17: Viewing the Snapshot Worksheet

4-2-9. Save the file.

This helps with more easily comparing the different configuration results.

Question 4

The project manager decides to add additional logic and make other changes to the design. How will you make the changes outlined below? Determine if there is sufficient thermal margin.

- The design needs to be estimated as more DSP-like and the effective clock rates have to double.
- More block RAMs will be needed (30) and they will be clocked at 200 MHz.
- DSP slice resources will need to be added as well (40) and they will be clocked at 200 MHz.
- Increase the Logic and Registers to 150000 and fanout to 160000 for the clock with 200MHz.
- If all options are assumed to run in the worst-case scenario (except the ambient temperature can be 50 degrees Celsius).

Question 5

Modify the environment (heat sink use and airflow) so that the FPGA can operate as the project manager needs.

Summary

You entered the estimated resources for the design by using the outline of the design in the Xilinx Power Estimator (XPE) spreadsheet and reviewed the estimated power of the design.

Answers

1. What information is provided in the Clock, Logic, IO, BRAM, DSP, CLKMGR, and GTX worksheets?

   In the Clock sheet, you enter each clock, the expected frequency, and the expected clocking resource it will use. If you are not certain which clocking resource will be used, keep the default selection for Type as Global clock.

   In the Logic sheet, you enter an estimate for the number of slice resources. The LUTs column should represent the number of LUTs used for arithmetic or logic, Shift Registers are the number of LUTs configured as SRLs (shift register LUTs), and SelectRAMs are the number of LUTs configured as memory. Registers are the number of registers or latches configured in the design. Use the different rows to separate different logic functions and characteristics (for example, clock speed and toggle rate).

   It is important to fill out the I/O sheet of XPE properly to obtain an accurate overall estimation of all rails of the chip. Depending on the selected I/O standard and I/O circuitry, a significant amount of power may be consumed not only in the VCCO rail but also in the VCCINT and VCCAUX rails.

   Many times it is simplest to enter each device interface separately and also to break out the interface signals to the data, control, and clock signals. This makes it easier to specify different I/O standards as well as other I/O characteristics, such as load and toggle rates.

   To ease data entry for more complicated standards, such as the DDR standards, you can use the Memory Interface Configuration Wizard. You can enter the relevant inputs in the Memory Interface Configuration Wizard and the tool will automatically populate the relevant I/O rows in the I/O sheet.

   In the Block RAM sheet, you enter the number and configurations of the block RAM intended to be used for the design. Make sure to adjust the Enable Rate to the percentage of time the ENA or ENB port will be enabled. The amount of time the RAM is enabled is directly proportional to the dynamic power it consumes, so entering the proper value for this parameter is important for an accurate block RAM power estimation.

   In the DSP sheet, note that DSP blocks can be used for purposes other than multipliers, such as counters, barrel shifters, multiplexers, and other common functions.

   In the CLKMGR sheet, if an MMCM and/or PLL is used in the design, specify the use and configuration of each in the Clock Manager sheet.

   In the GTX sheet, if GTs (serial transceivers) are used in the design, specify the use and configuration of each in the GT sheet.

2. Based on the Device characterization under settings (Production, Advance, and Preliminary), what are your expectations about the quality of power estimation?

   The accuracy of the characterization data existing in the XPE is determined by the entry in the Characterization field (under Device section) in the Summary sheet of XPE. The entries in the Characterization field can be Advance, Preliminary, and Production. For the chosen device it is Production. This implies that the characterization data is fully correlated with the production silicon.

   Advance: The data integrated into XPE with this designation is based primarily on measurements and characterization data made on early production devices. A set of widely used device resources are included in the characterization. Characterization data is limited to these few blocks. This data is typically available within a year of product launch. Although the data with this designation is considered relatively stable and conservative, some under-reporting or over-reporting might occur. Advance data accuracy is considered lower than the Preliminary and Production data.

   Preliminary: The data integrated into XPE with this designation is based on complete early production silicon. Almost all the blocks in the device fabric are characterized. Data for most of the dedicated blocks like TEMAC and PCIe® block are also characterized and integrated into XPE. The accuracy of power reporting is improved compared to Advance data.

   Production: The data integrated into XPE with this designation is released after enough production silicon of a particular device family member has been characterized to provide full power correlation over numerous production lots. Characterization data for all blocks in the device fabric is included.

3. Enter the data in the following tables:

Summary	XPE
Junction Temperature (C)	31.7
Effective θJA (C/W)	1.8
Total On-Chip Power (W)	1.0

Summary of XPE

On-Chip Power (W)	XPE
CLOCK	0.100
LOGIC	0.240
BRAM	0.001
DSP	0.000
MMCM	0.115
IO	0.011
Device Static	0.514

On-Chip Power of XPE

Power Supply Total Current (A)	XPE
Vccint	0.536
Vccbram	0.012
Vccaux	0.161
Vcco1.8v	0.001
VccADC	0.014

Power Supply of XPE

4. The project manager decides to add additional logic and make other changes to the design. How will you make the changes outlined below? Determine if there is sufficient thermal margin.

   - The design needs to be estimated as more DSP-like and the clock rates have to double.

   This will require the default activity rate to be more DSP-like and logic to toggle at 50% (this is a big difference). Likewise, you could also have the block RAM, DSP slice, and I/O toggle at 50%. Note that you should not change the clock nets default activity rate (this will require you to reset the clock rates).

   Since the clock rates are increasing, this will mean that the clock rates will have to be changed for the clocks, logic, and block RAM (the I/O and CLKMGR pages will not need changes).

   - More block RAMs will be needed (30) and they will be clocked at 200 MHz.

   This will require a simple modification to the BRAM page.

   Toggle rate to be at 50% and clock at 200 MHZ for both Port A and Port B.

   - DSP slice resources will need to be added as well (40) and they will be clocked at 200 MHz.

   This will require a simple addition to the DSP page.

   Toggle rate to be set as 50%.

   - Increase the Logic and Registers to 150000 and fanout to 160000 for the clock with 200MHz.

   Increase the Logic and Registers in Logic sheet and increase the fanout in Clock Sheet.

   Double the clock frequency from 50 to 100 MHz and 100 to 200 MHz with the Toggle rate 50%.

   - If all options are assumed to run in the worst-case scenario (except the ambient temperature can be 50 degrees Celsius).

   This will require that the ambient temperature be modified to 50 degrees Celsius, the heat sink be turned off, and the airflow be set to 0 LFM. This will also require that the implementation tool settings be set to default (instead of power optimization). The Process setting should still be set to Maximum (worst case timing numbers used).

   After making these changes, you should get the junction temperature of 125+. The most significant changes were the ambient temperature, airflow, and default toggle rates.

5. Modify the environment (heat sink use and airflow) so that the FPGA can operate as the project manager needs.

   After modifying the airflow (turn the heat sink to medium and the airflow to 250 LFM) you should obtain a junction temperature of 65 degrees C. This is within the specification of 85 degrees C. However, this is still much higher. If you lower the ambient temperature (from 50 to 30 degrees C) you quickly realize that a lot of thermal margin is gained by reducing the temperature.

Lab 3: Resets

2022.2

Abstract

This lab introduces the use of resets and their impact on the design.

This lab should take approximately 60 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Remove resets thus producing smaller designs resulting in improved performance and run time
- Design using synchronous resets when absolutely necessary (such as local resets)
- Use a reset bridge to synchronize global resets (only if needed) for each clock domain
- Infer INIT attributes on sequential elements from your HDL
- Report high fanout nets using `report_high_fanout_nets`
- Analyze constrained and unconstrained reset timing paths
- Map high-fanout nets to the global routing resources (BUFG)

Introduction

The following is recommended:

- Remove global resets whenever possible (no resets is best).
- Smaller designs consume less power.
  - Less routing is consumed.
  - Performance and software run time is improved due to fewer timing paths in the design.
  - SRL inferencing is facilitated.
  - The number of control sets is reduced.
  - Initializing registers when removing resets improves functionality and simulation behavior.

- Change needed resets to synchronous (if reset is necessary).
- Register placement becomes more flexible.
  - Enhanced utilization and more robust timing coverage is possible.
  - If global, a reset bridge provides a safe mechanism to assert reset asynchronously and de-assert reset synchronously.
  - Every clock domain requires its own synchronous reset.
    - Ensure that the MMCM-generated clock is stable and locked before de-assertion of the synchronous global reset.
- Consider a hybrid approach.
- Rely on the built-in initialization that the GSR provides, along with explicit synchronous resets for local resets and/or portions of your design that can start autonomously.
- Make all control signals (including reset) active-high.
- Active-low signals require an inversion.
  - Active-high signals enable better device utilization and improve performance.
- MMCM reset is not necessary unless you expect the clock to be unstable at start up, or you plan to change the clock frequency at a later time.

In this lab, you will investigate the proper design and use of resets by examining several projects: one with asynchronous resets, the next project with resets removed entirely, the following project with synchronous resets used *where necessary,* and, optionally, a project with a high-fanout net (such as a reset) moved to a BUFG.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

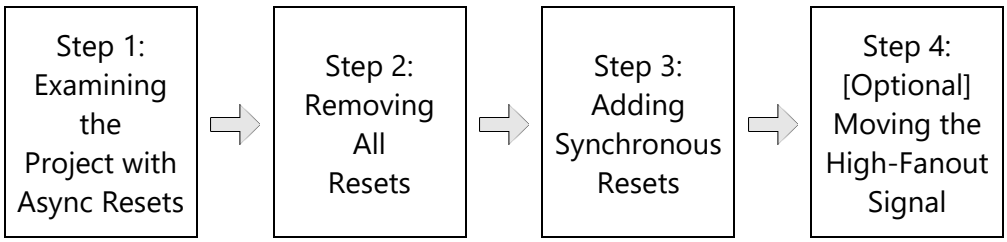
If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Examining the Project with Asynchronous Resets Step 1

You will begin by launching the Vivado® Design Suite and opening the provided project.

Here are two ways to open the Vivado Design Suite.

1-1. Open the Vivado Design Suite.

1-1-1. Click the **Vivado** icon ( ) from the taskbar.

Figure 3-1: Launching the Vivado Design Suite from the Taskbar

Note: It takes a few moments to open. The order of the icons in your environment may be different.

Alternatively, from the Linux terminal window (<**Ctrl** + **Alt** + **T**>), enter the following:

```
source /opt/amd/Vivado/2022.2/settings64.sh; vivado
```

Note: This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

The tool opens to the Welcome window. From here you can create a new project, open an existing project, enter Tcl commands, and access documentation and examples.

1-1-2. [Optional] Maximize the window as there is a lot of information to see.

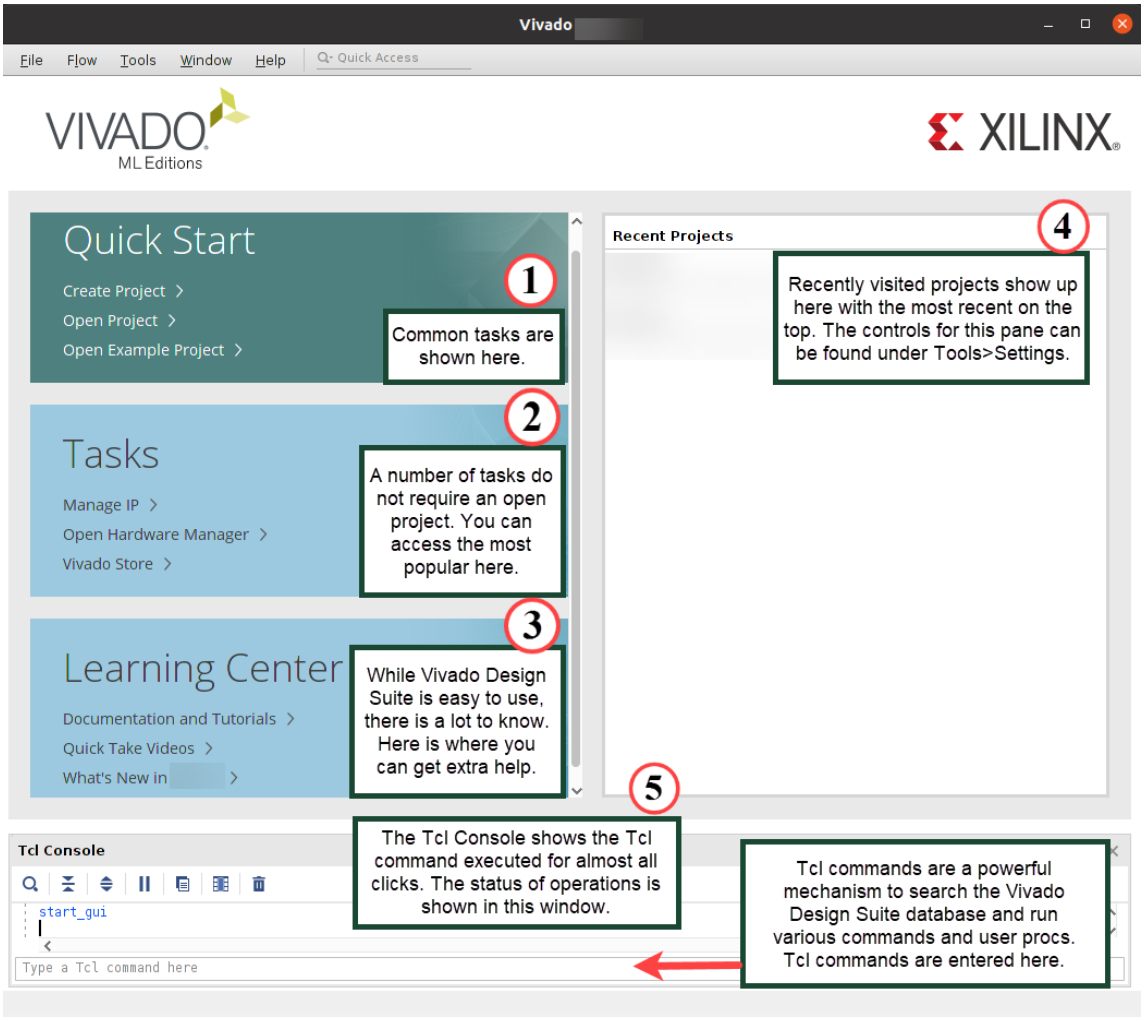


Figure 3-2: Vivado Design Suite Welcome Screen

Hint: If the Tcl Console is not visible, double-click the **Tcl** tab to make it visible.

1-2. Open the existing Vivado Design Suite project *async_reset.xpr*.

1-2-1. Click **Open Project** from the Quick Start section (1).

The Open Project dialog box opens (2).

1-2-2. Browse to the `$TRAINING_PATH/Reset/lab/KCU105/async_reset` directory in the Look in field (3).

Note: The drop-down arrow shows the directory hierarchy.

1-2-3. Select **async_reset.xpr** from the files displayed (4).

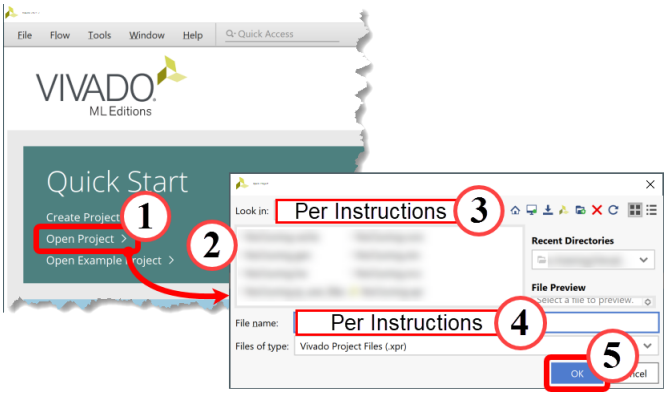


Figure 3-3: Opening an Existing Project

1-2-4. Click **OK** to open the selected project (5).

The project now opens in the Vivado Design Suite.

1-3. Open the synthesized design.

1-3-1. Click **Open Synthesized Design** under Synthesis in the Flow Navigator.

Alternatively, you can select **Flow** > **Open Synthesized Design**.

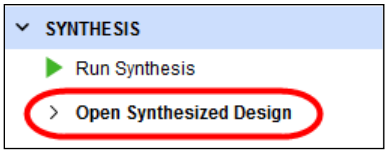


Figure 3-4: Opening the Synthesized Design

1-3-2. Enter the following command in the Tcl Console to select the cells connected to the SysRst_L input port in the design:

```
select_objects [get_cells SysRst_L*]
```

1-3-3. Press <**F4**> to view the schematic of the IBUF cell.

1-3-4. Double-click the output pin (i.e., the O pin) of **IBUF**.

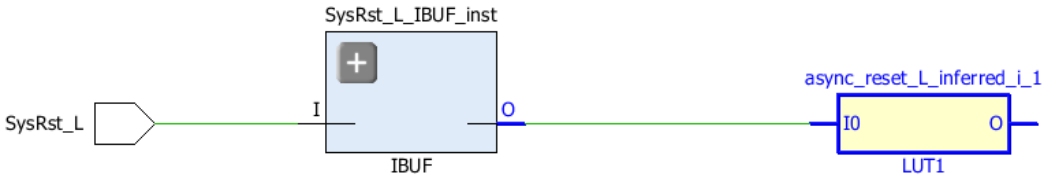


Figure 3-5: Schematic View of SysRst_L (KCU105)

1-3-5. Select the **LUT1** instance in the schematic to see the properties of the LUT cell in Cell Properties.

Question 1

What is the purpose of the LUT1?

Hint: Select the **LUT1** cell and observe the LUT equation in the Truth Table tab of the Cell Properties window.

1-3-6. Double-click the output of **LUT1** to see the connectivity of the output net of the LUT.

The output of LUT1 is connected to all the flip-flops in the design. Since the design uses active-low reset, the reset signal is connected to the flip-flops via the LUT, which performs the task of inverting the reset.

1-3-7. Zoom out as necessary to see all the flip-flops connected to this net.

1-3-8. Click **X** to close the Schematic tab in the main workspace area.

You have observed the impact of having asynchronous resets in the design in terms of inverting resets using a LUT. Next you will see how having asynchronous resets can be a bottleneck in the timing analysis perspective in the design.

You will generate the Timing Summary report to note the number of endpoints analyzed for an asynchronous reset design and you will compare this with the number of endpoints analyzed for a no-reset design and synchronous reset design.

1-4. Generate a Timing Summary report.

1-4-1. Click **Report Timing Summary** under **Synthesis** > **Open Synthesized Design** in the Flow Navigator.

The Report Timing Summary dialog box opens.

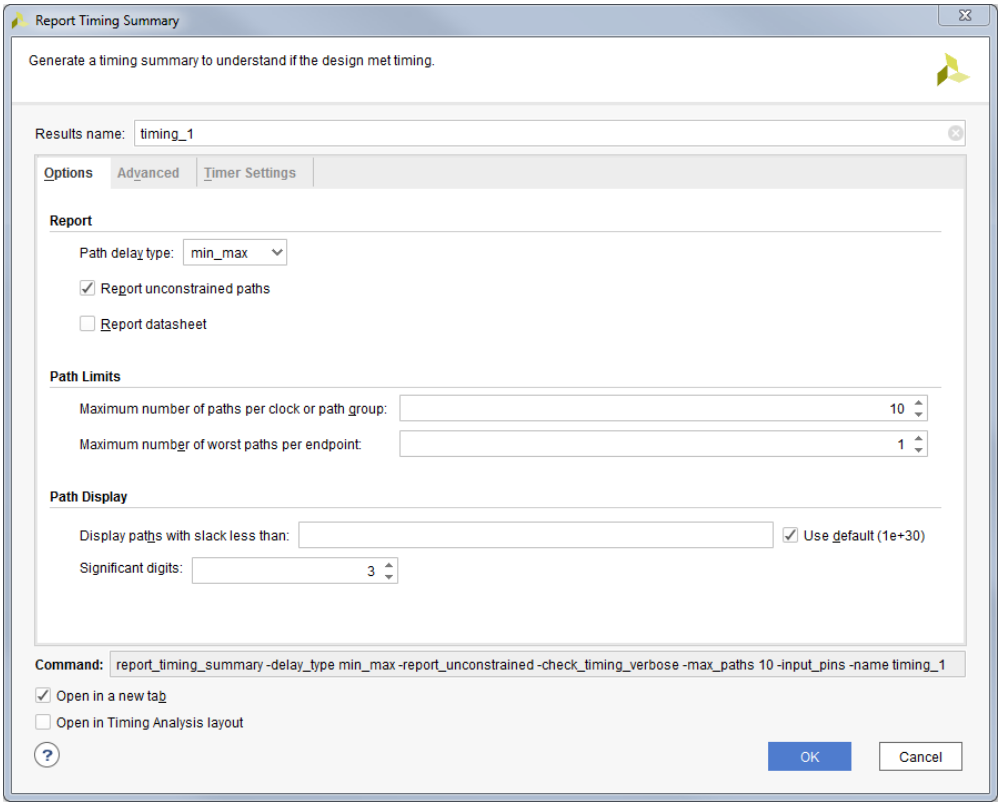


Figure 3-6: Report Timing Summary Dialog Box

1-4-2. Click **OK** to generate the Timing Summary report using the default settings.

The Design Timing Summary window opens at the bottom in the Timing tab.

Note down the **WNS**, **WHS**, and especially the **Total Number of Endpoints** values in the Design Timing Summary window as you will need these values for comparison later in the lab.

Question 2

What is the Total Number of Endpoints for Setup?

1-4-3. Click **Expand All** (⇳) in the Timing tab to see all the sections in the Timing Summary report.

1-4-4. Select the **Unconstrained Paths** section in the Timing Summary report to examine the unconstrained paths in the design.

This section helps you observe what timing paths, if any, are unconstrained and why they are unconstrained.

1-4-5. Select the **Setup** section in the **clk_out1_mmcm_clocks to NONE** group.

1-4-6. Observe the paths.

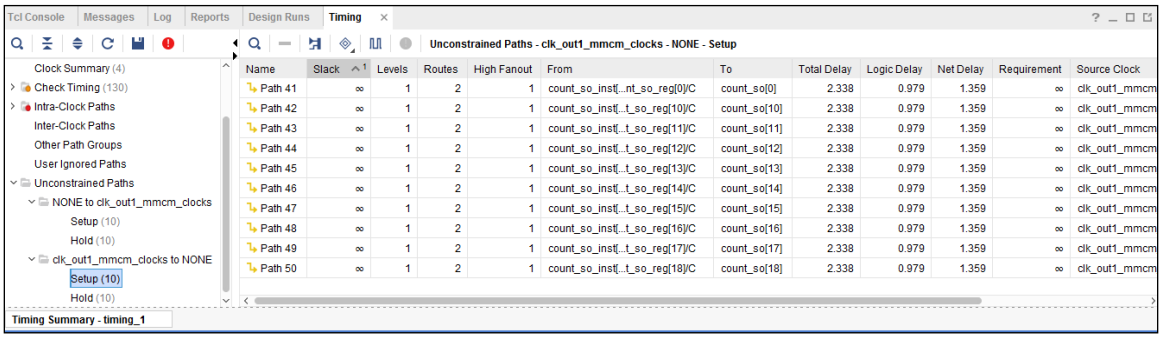


Figure 3-7: Timing Summary – Unconstrained Paths

Question 3

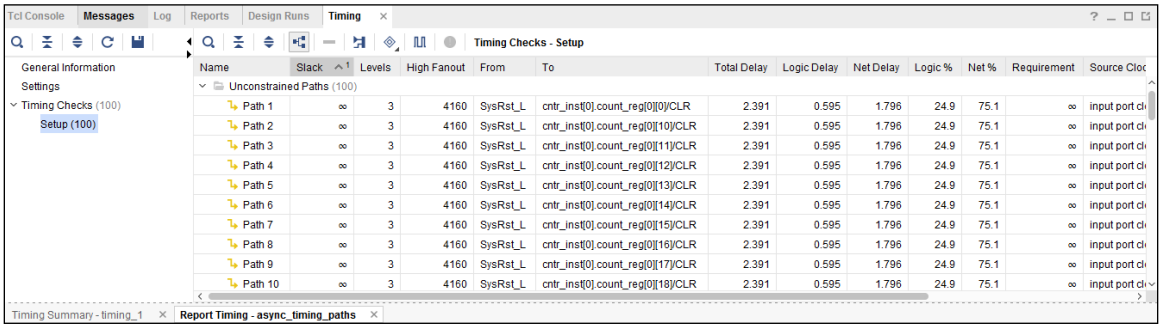
What is the slack on these paths and why?

1-4-7. Enter the following command in the Tcl Console to report the timing paths that originated from the SysRst_L input port, which is an asynchronous reset signal.

```
report_timing -from [get_ports {SysRst_L}] -max_paths 100 -name
async_timing_paths
```

Notice the `-name` option, which displays the timing report in the GUI and not in the Tcl Console.

You should see a new tab at the bottom (async_timing_paths) that resembles the following figure.



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Cloc
Path 1	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][0]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 2	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][10]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 3	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][11]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 4	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][12]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 5	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][13]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 6	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][14]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 7	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][15]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 8	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][16]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 9	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][17]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl
Path 10	∞	3	4160	SysRst_L	cntr_inst[0].count_reg[0][18]/CLR	2.391	0.595	1.796	24.9	75.1	∞	input port cl

Figure 3-8: Asynchronous Timing Paths (KCU105)

This report helps you to determine that all the timing paths that are originating from the SysRst_L port are not being constrained (i.e., unconstrained).

1-5. Examine the INIT attribute of one of the counter registers.

1-5-1. Enter the following command in the Tcl Console to return the INIT property of the count_reg flip-flop which the LUT1 output is connected to:

```
get_property INIT [get_cells *count_reg[0][0]]
```

Question 4

What value is returned and where is this INIT value set?

1-6. Examine the INIT attribute of one of the sload registers.

1-6-1. Enter the following command in the Tcl Console to return the INIT property of the sload flip-flops:

```
get_property INIT [get_cells sload_in_reg]
```

Question 5

What value is returned and where is this INIT value set?

1-7. Generate and examine the High Fanout Nets report.

1-7-1. Enter the following command in the Tcl Console to see the highest fanout net in the design:

```
report_high_fanout_nets
```

1-7-2. Review the generated report.

Question 6

What is the highest fanout net? What is the fanout? And what is the driver type?

Remember, if possible, you want to remove all of the global resets and initialize the registers instead.

1-8. Having found the highest fanout net, view the schematic by first selecting the net. (Hint: Choose the net from the report.)

1-8-1. Enter the following command in the Tcl Console to select the highest fanout net:

```
select_objects [get_nets async_reset_L]
```

1-8-2. Press <**F4**> to view the schematic.

By zooming in at the top, you can see that highest fanout net is nothing but the output of LUT1, which performs the conversion of the reset signal.

1-8-3. Observe that the net is connected to the CLR port of the following FDCEs:

cntr_inst[0].count_reg[0][0] to cntr_inst[0].count_reg[0][63]

cntr_inst[1].count_reg[1][0] to cntr_inst[1].count_reg[1][63]

cntr_inst[2].count_reg[2][0] to cntr_inst[2].count_reg[2][63]

cntr_inst[3].count_reg[3][0] to cntr_inst[3].count_reg[3][59]

1-8-4. Click **X** in Schematic tab to close the schematic.

1-8-5. Close the Timing tab to close the Timing Summary report.

1-9. Close the synthesized design.

You have analyzed the design at the post-synthesis stage. Now you will open the implemented design to analyze the design at the post-implementation stage.

1-10. Open the implemented design.

1-10-1. Click **Open Implemented Design** under Implementation in the Flow Navigator.

Alternatively, you can select **Flow** > **Open Implemented Design**.

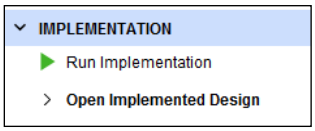


Figure 3-9: Opening the Implemented Design

Note that the Timing Summary report is automatically generated in read only mode when the implemented design is opened.

1-10-2. Click **Show Routing Resources** ( ) in the horizontal toolbar of the Device view, if it is not already enabled, to show the routing resources.

1-10-3. Enter the following command in the Tcl Console to determine the highest fanout net post-implementation:

```
report_high_fanout_nets
```

1-10-4. Select the highest fanout net again in the Tcl Console and look in the Device view at the routing that is used for the LUT-inverted reset signal.

Hint: You can press the up arrow key in the Tcl Console until you get back to the `select_objects [get_nets async_reset_L]` command.

Note the highlighted nets in the Device view.

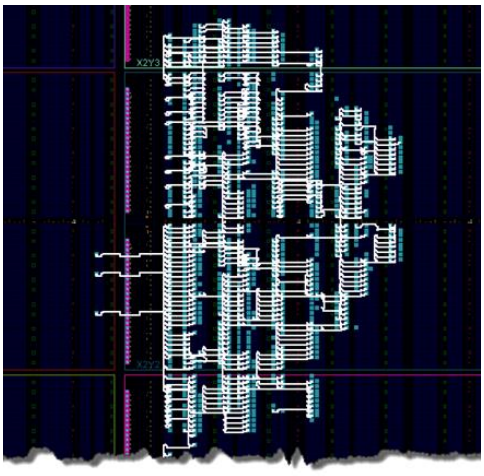


Figure 3-10: Device View Highlighting High-Fanout Net (KCU105)

You can observe that the reset signal is spread all over the design, which could lead to routing congestion if the design is big enough.

1-10-5. Enter the following command if you want to view all the pins that this net connects to in the Tcl Console:

```
join [get_pins -of [get_nets async_reset_L]] \n
```

1-10-6. View the number of pins (the Tcl list length) in the Tcl Console by pressing the up arrow key again to select the previous command and changing `join` to **`llength`** and removing the `\n`:

```
llength [get_pins -of [get_nets async_reset_L]]
```

Verify the value for the number of pins in Tcl console as 4161.

Key Point: Routing can be considered one of the most valuable resources. Resets compete for the same resources as the rest of the active signals in the design. More available routing gives the tools a better chance to meet your timing objectives.

1-11. Close the implemented design.

1-12. Close the project.

1-12-1. Select **File** > **Close Project** to close the project.

The Close Project dialog box opens.

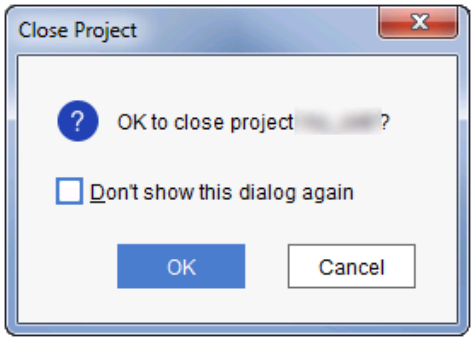


Figure 3-11: Close Project Dialog Box

1-12-2. Click **OK**.

You have analyzed a design that has asynchronous resets and observed the impact of resets on the design in terms of timing analysis, device utilization, routing congestion, etc. Next you will analyze a design that has NO resets.

Removing All Resets Using the no_reset Project — Step 2

Here you will open and analyze a project that does not have any asynchronous resets.

2-1. Open the Vivado Design Suite project named `no_reset.xpr` located in the directory below.

2-1-1. Browse to the `$TRAINING_PATH/Reset/lab/KCU105/no_reset` directory and open the **`no_reset.xpr`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-2. Examine the RTL to become familiar with the base design.

2-2-1. Select the **Hierarchy** tab in the Sources window.

2-2-2. Click the **Expand All** icon (⇳) to view all the files.

2-2-3. Double-click **reset_example_top** and examine the top-level RTL.

This example is parameterized in order to benchmark different ‘size’ designs with the device utilization varying based on two parameter settings. The example builds N binary counters (where N=NUM_CNTRS) of counter width C (where C=CNTR_WIDTH).

These parameters can be easily modified in the RTL or by entering a Tcl command (such as set_property generic CNTR_WIDTH=128 [current_fileset]) in the Tcl Console (also in the Vivado Design Suite GUI via Settings > General > Language Options > Generics/Parameters) and overriding the default parameters.

By reviewing the hierarchy (**Sources/Hierarchy** tab) you can see an mmcm_clocks module that was created through the IP Catalog (**IP Sources** tab to see the clock wizard files that were created).

You can also see that there is a reset_bridge module but this is shown at the same level as the top. The reset_bridge module is conditionally instantiated for some of the project types. It is used only when SYNC_RESET or SYNC_RESET_ON_BUFG are defined. Because it is not instantiated for the no_reset project, the reset_bridge is shown at the same level as top, not hierarchically below top.

2-2-4. Click **X** in the reset_example_top.v file tab to close the file.

2-3. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-4. Generate a Timing Summary report.

If you do not recall how to perform this task, refer to the "Generating a Timing Summary Report on the Synthesized Design" or "Generating a Timing Summary Report on the Implemented Design" topics under Vivado Design Suite Operations in the *Lab Reference Guide*.

Question 7

What is the Total Number of Endpoints for Setup?

2-5. Examine the flip-flop's reset behavior.

Remember that the control set of a flip-flop is composed of the clock input (CLK), the active-high chip enable (CE) and the active-high SR port (SR). The SR port can serve as a synchronous set/reset or an asynchronous preset/clear port. The synthesis tool will infer a FDCE/FDPE if it is asynchronous and a FDRE/FDSE if synchronous. The cell definition defines the reset nature.

The flip-flop output can be initialized to the value that the INIT 'attribute' specifies. The INIT value is loaded into the flip-flop during configuration and when the global set reset (GSR) signal is asserted.

2-5-1. Enter the following command in the Tcl Console to return the INIT property of the sload_in_reg flip-flops in the design:

```
get_property INIT [get_cells sload_in_reg*]
```

Question 8

What values are returned and where is this INIT value set?

2-6. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-7. Open the implemented design.

If you do not recall how to perform this task, refer to the "Opening the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-7-1. Enter the following Tcl command to view the INIT values of the sload_in_reg register:

```
select_objects [get_cells sload_in_reg]
```

2-7-2. Zoom in and see the selected flip-flop in the slice in the Device view.

2-7-3. Select the **Properties** tab to locate the **INIT** property for the selected cell in the Cell Properties window.

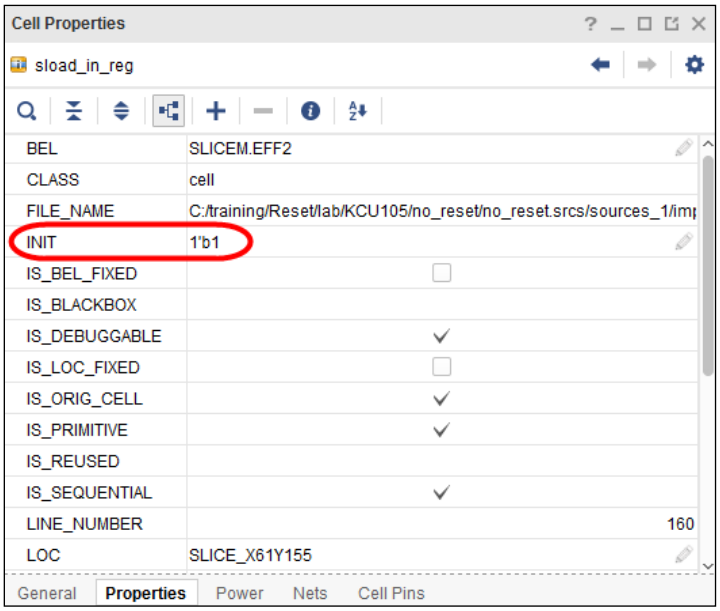


Figure 3-12: INIT Attribute

Key Point: For a design with no resets, you still have the ability to initialize the synchronous elements in the design to a known state at power up. You can initialize the flip-flop output to the value that the INIT attribute specifies. The INIT value is loaded into the flip-flop after configuration and when the global set reset (GSR) signal is asserted. For the Vivado and Synplify synthesis tools, the INIT value is extracted from the RTL code.

For this design example using the Vivado synthesis feature, the INIT values for sload_in_reg have been extracted from the RTL on line 87 (*reset_example_top_noreset.v*):

```
reg sload_in = `ACTIVE;
```

2-7-4. Enter the following command in the Tcl Console to view the high fanout nets, if any, in the design:

```
report_high_fanout_nets
```

2-7-5. Review the generated report.

Question 9

Are there any high fanouts in this no_reset project?

2-8. Close the implemented design.

If you do not recall how to perform this task, refer to the "Closing the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-9. Close the Vivado Design Suite project.

If you do not recall how to perform this task, refer to the "Closing the Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

You have analyzed a design that had NO resets. Next, you will analyze a design that has synchronous resets.

Adding Synchronous Resets to the sync_reset Project Step 3

Here you will open and analyze a design that has synchronous resets.

Remember that having no resets is the best approach. You might also consider a hybrid approach that relies on the built-in initialization that the GSR provides, along with explicit synchronous resets for local resets and portions of your design that can start autonomously.

If a global reset is needed for those portions of your design that can start autonomously, a reset bridge provides a safe mechanism to assert reset asynchronously and de-assert reset synchronously.

While not needed for this simple counter example, a global synchronous reset is used in this project in order to explore the use of a reset bridge. Again, we recommend removing resets from your design except where explicitly required for proper operation.

3-1. Open the Vivado Design Suite project named `sync_reset.xpr` located in the directory below.

3-1-1. Browse to the `$TRAINING_PATH/Reset/lab/KCU105/sync_reset` directory and open the **`sync_reset.xpr`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

You will now open the elaborated the design to view the RESET BRIDGE schematic.

3-2. Open the elaborated design.

3-2-1. Click **Open Elaborated Design** under RTL Analysis in the Flow Navigator to elaborate the design.

The elaborated RTL design enables various analysis views, including RTL Netlist, Schematic, and Graphical Hierarchy. These views have a cross-select feature, which allows you to debug and optimize the RTL.

The Elaborate Design dialog box opens.

3-2-2. Click **OK** in the dialog box to open the elaborated design.

3-3. View the RESET_BRIDGE_100 block.

3-3-1. Click **Schematic** under **RTL Analysis** > **Elaborated Design** in the Flow Navigator.

3-3-2. Enter the following Tcl command in the Tcl Console to select the RESET BRIDGE cell in the design:

```
select_objects [get_cells RESET_BRIDGE_100]
```

The RESET_BRIDGE_100 is selected in the RTL Schematic. You may either enable the **Auto-fit Selection** icon or scroll down and zoom in to see the selected reset_bridge block.

3-3-3. Double-click inside the selected (RESET_BRIDGE_100) symbol.

You should see the next level of hierarchy as shown in the figure below.

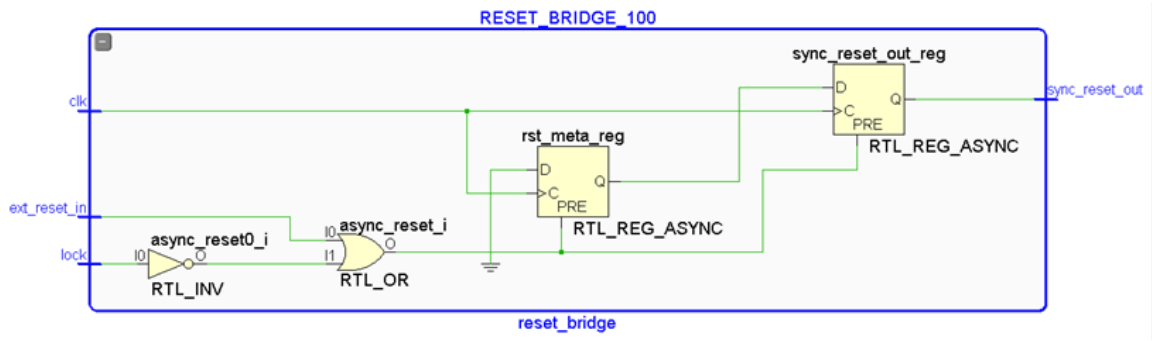


Figure 3-13: Schematic View of reset_bridge

Question 10

How does the reset bridge assert 'sync_reset_out'?

Question 11

How does the reset bridge de-assert 'sync_reset_out'?

3-4. Close the elaborated design.

3-4-1. Select **File** > **Close Elaborated Design** to close the elaborated design.

The Confirm Close dialog box opens.

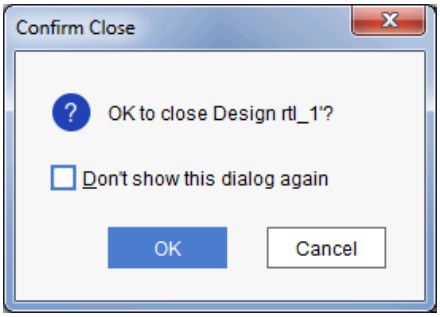


Figure 3-14: Confirm Close Dialog Box

3-4-2. Click **OK**.

3-5. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-6. Generate a Timing Summary report.

If you do not recall how to perform this task, refer to the "Generating a Timing Summary Report on the Synthesized Design" or "Generating a Timing Summary Report on the Implemented Design" topics under Vivado Design Suite Operations in the *Lab Reference Guide*.

Question 12

What is the total number of Setup Endpoints analyzed for the sync_reset project?

3-7. Generate the high fanout nets report.

3-7-1. Enter the following command in the Tcl Console to see the high fanout nets in the design:

```
report_high_fanout_nets
```

3-7-2. Review the report.

Question 13

What is the highest fanout net? What is the fanout? And what is the driver type?

3-8. Having found the highest fanout net RESET_BRIDGE_100/sync_reset_out, retrieve a list of all the endpoints it fans out to.

3-8-1. Enter the following command in the Tcl Console:

```
all_fanout -endpoints_only -flat [get_nets RESET_BRIDGE_100/
sync_reset_out]
```

A fairly long list is returned, and from the previous report, you would expect this to have a fanout of 4225.

3-9. Confirm the list length returned.

3-9-1. Press the up arrow key in the Tcl Console to find the previously executed `all_fanout` command.

3-9-2. Press the **HOME** key to get to the beginning of the line and add to the front of the previous command: `llength` and add `[`.

3-9-3. Press the **END** key to get to the end of the line and add `]`.

You should now have formed the following command:

```
llength [all_fanout -endpoints_only -flat [get_nets
RESET_BRIDGE_100/sync_reset_out]]
```

Question 14

What is the list length that is returned for the endpoint fanout from the RESET_BRIDGE_100/sync_reset_out net?

3-10. Now look at the worst-case timing path analysis through the reset_clk100 net.

3-10-1. Enter following Tcl command in the Tcl Console to generate a timing report that lists the timing paths that pass through the RESET_BRIDGE_100/sync_reset_out net:

```
report_timing -through [get_nets {RESET_BRIDGE_100/
sync_reset_out}] -name reset_timing_paths -max_paths 10
```

3-10-2. Select **Path 1** in the Report Timing window.

3-10-3. Right-click and select **View Path Report**.

For the next two questions, refer to the path report.

Question 15

What source clock pin is the start of this path analysis?

Question 16

What is the destination pin for this path analysis?

Key Point: This synchronous reset path analysis was timed from a /C pin to a /R pin.

While this sync_reset project has purposely used synchronous resets throughout the design, you have previously seen in the no_reset project that there is no need to code in the use of a reset if the purpose was to guarantee the post-configuration initial state. You have the INIT attribute for this as was previously shown.

3-11. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Keep the project open as you will modify it in the next step.

The next step is optional, illustrating how to move the high fanout synchronous reset signal to a BUFG. If you have an additional 15 minutes, you will be able to view the synchronous reset in a BUFG.

[Optional] Moving the High-Fanout Synchronous Reset Signal to a BUFG Step 4

Here you will move the reset signal to BUFG.

When there is a very large fanout of a global reset (such as a synchronous reset signal), you could choose to put this on a BUFG. This saves general routing resources at the expense of one of the global routing resources.

4-1. Modify the design by setting a 'define on the project.

4-1-1. Select **Settings** under Project Manager in the Flow Navigator.

The Settings dialog box opens with the General tab, which allows you to set project settings such as target language, top module, project device, etc.

4-1-2. Click the **Browse** icon in the **Language Options** > **Verilog Options** section of the General tab.

The Verilog Options dialog box opens and enables you add or remove the -verilog_define values for the design.

4-1-3. Click the **+** symbol in the Defines section of the Verilog Options dialog box.

The Add Value dialog box opens in which you can enter the define name and value.

4-1-4. Enter **SYNC_RESET_ON_BUFG** in the Name field.

4-1-5. Enter **1** in the Value field.

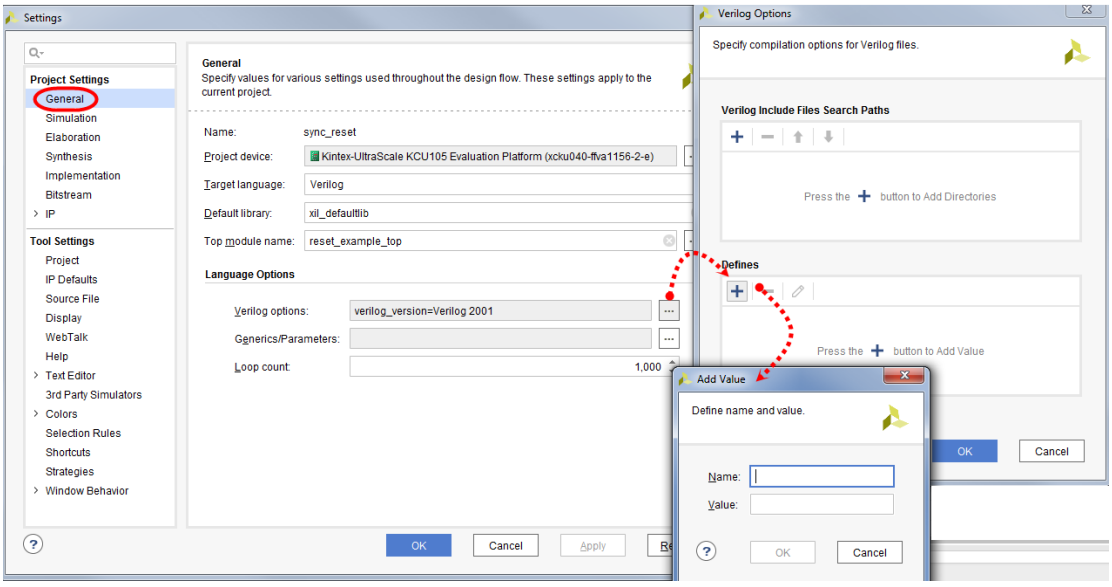


Figure 3-15: Adding Define Name and Its Value for the Design

4-1-6. Click **OK** in all dialog boxes.

4-1-7. Click **No** in the Create New Run dialog box that opens.

4-2. Examine the RTL to determine what the -verilog_define will do.

4-2-1. Double-click the **reset_example_top_sync.v** file in the Hierarchy tab of Sources window to open the file in the text editor.

4-2-2. Examine the reset_example_top_sync.v RTL module to find **SYNC_RESET_ON_BUFG**.

4-2-3. Observe the **'define** usage.

4-2-4. Uncomment **line 52** in the *reset_example_top_sync.v* RTL module.

4-2-5. Save and close the **reset_example_top_sync** file.

4-3. Run synthesis.

4-3-1. Click **Run Synthesis** in the Flow Navigator under Synthesis and click **OK** to launch runs.

Alternatively, you can also select **Flow** > **Run Synthesis** or press <**F11**>.

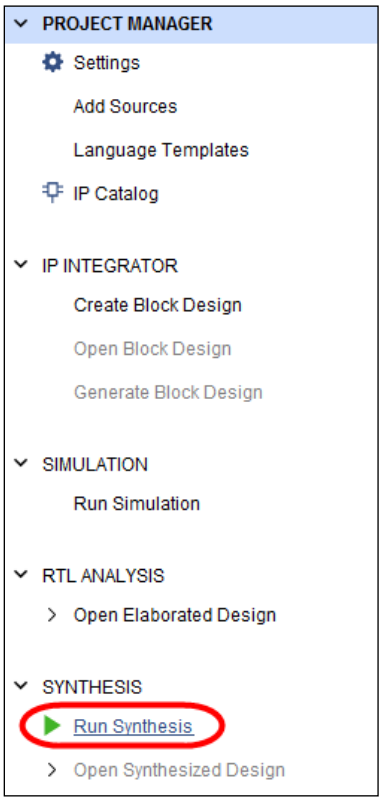


Figure 3-16: Selecting Run Synthesis

4-3-2. Click **Save** if you are asked to save your files.

After the synthesis process completes, the Synthesis Completed dialog box opens. The dialog box prompts you to run implementation, open the synthesized design, or view reports.

4-3-3. Select **Open Synthesized Design**.

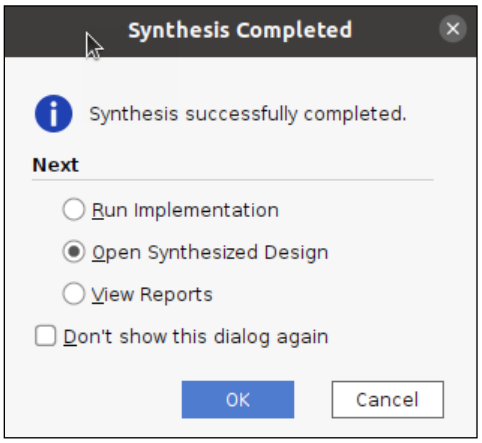


Figure 3-17: Synthesis Completed Dialog Box

4-3-4. Click **OK**.

4-3-5. Enter the following Tcl command in the Tcl Console to generate a high fanout nets report in the Tcl Console:

```
report_high_fanout_nets
```

Question 17

What is the reported driver type now for the reset_clk100 signal?

4-4. View the BUFG in the schematic.

4-4-1. Enter the following Tcl command in the Tcl Console:

```
select_objects [get_cells RESET_BRIDGE_100]
```

4-4-2. Press <**F4**> to view the schematic.

4-4-3. Go to the Settings icon on the top right corner and enable the **Fanout For Scalar Pin** option.

4-4-4. Double-click the **reset_clk100_i** net.

You should see the BUFGCE cell for the KCU105 board.

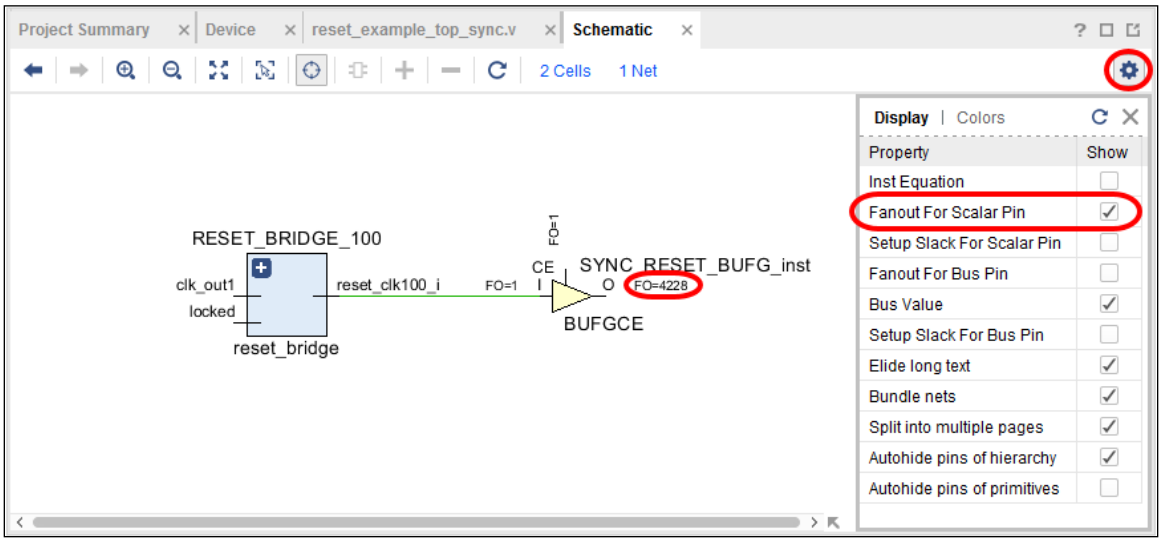


Figure 3-18: Schematic View of the BUFGCE Connection (KCU105)

4-4-5. Double-click the **O pin** of the BUFGCE to show the signal fanout to all the endpoints.

4-5. Rerun the worst-case timing path analysis through the reset_clk100 net.

Note that the reset_clk100 net is now the net connected to the output of the BUFGCE.

4-5-1. Enter the following Tcl command in the Tcl Console to generate a timing report that lists the timing paths that are passed through the reset_clk100 net:

```
report_timing -through [get_nets {reset_clk100}] -name
reset_timing_paths -max_paths 10
```

4-5-2. Right-click **Path 1** and select **Path Properties**.

4-5-3. Maximize the Path Properties window for a better view.

Question 18

Where is the SYNC_RESET_BUFG shown in the path report?

The Data Path Delay Section of your path report should resemble the following figure.

Delay Type	Incr (ns)	Path (...	Loca...	Netlist Resource(s)
FDPE (Prop_FDPE_C_Q)				RESET_BRIDGE_100/sync_reset_out_reg/Q
net (fo=1, unplaced)				reset_clk100_i
BUFGCE (Prop_BUFGCE_I_O)				SYNC_RESET_BUFG_inst/O
net (fo=4254, unplaced)				reset_clk100
FDRE				cntr_inst[0].count_reg[0][0]/R
Arrival Time				

Figure 3-19: Data Path Analysis with BUFGCE (KCU105)

Question 19

Complete the following table based on the previous questions and answers as well as your analysis.

	Async Resets	**No Resets**	**Sync Resets**	**Sync Resets (BUFG)**
Total Reset Fanout				
Reset Driver Type				
Total Number (Setup) of Endpoints Analyzed				

4-6. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

4-7. Close the Vivado Design Suite.

4-7-1. Select **File** > **Exit**.

The Exit Vivado dialog box opens.

Figure 3-20: Exit Vivado Dialog Box

4-7-2. If you are asked to save the project or a portion of the project, select whichever elements of the project you want to save, then click **Save** to save the selected elements; otherwise, click **Don't Save**.

4-7-3. Click **OK** when you are asked to exit the Vivado Design Suite.

4-7-4. **Note:** You can choose to select the *Don't show this dialog again* option to avoid being asked for confirmation when exiting the Vivado Design Suite.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/Reset` directory.

4-8. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

4-8-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/Reset`.

4-8-2. Select **`Reset`**.

4-8-3. Press <**Delete**>.

-- OR --

Using the command line:

4-8-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

4-8-5. Enter the following command to delete the contents of the workspace:

[Windows users]: **`rd /s /q $TRAINING_PATH/Reset`**

[Linux users]: **`rm -rf $TRAINING_PATH/Reset`**

Summary

Removing global resets and only using resets when absolutely necessary is recommended. Consider a hybrid approach that relies on the built-in initialization that the GSR provides, coupled with explicitly targeted resets when necessary for local resets or portions of your design that can start autonomously. Always use synchronous resets when resets are required. If a global reset is required for part of your design, use a reset bridge that asynchronously asserts and synchronously de-asserts reset for each clock domain.

Resets compete for the same routing resources as the rest of the signals in your design. Fewer resets means smaller designs, lower power consumption, fewer timing paths in your design, improved performance and software run time, reduced number of control sets, and better SRL inferencing. All control signals, including resets should be active-high, enabling better device utilization and improving performance.

When an FPGA is configured, synchronous elements are loaded with an initialization value determined by the INIT attribute. The INIT value is extracted from the RTL code for the synthesis tools.

Asynchronous resets will generate DRC warnings and should be either removed or selectively changed to synchronous resets. Asynchronous resets will be reported in the timing summary under Unconstrained Paths with infinite slack. Asynchronous resets that are not synchronized to a clock domain can also cause your system to fail to come out of reset properly. Asynchronous resets can cause unwanted clearing due to any glitch, and runt pulses can also cause metastability. Use of asynchronous resets is NOT recommended.

Answers

1. What is the purpose of the LUT1?

   This is used to invert the active-low reset signal to a native active-high signal used by the FDCEs. Avoid using active-low control signals because an inverter is required to convert these to native active-high.

2. What is the Total Number of Endpoints for Setup?

   From the Design Timing Summary: 12416.

3. What is the slack on these paths and why?

   The slack is infinity (∞) because these are all asynchronous paths.

4. What value is returned and where is this INIT value set?

   1'b0 is the value returned. The initial state (INIT values) of the count_reg registers was 0 by default because it was not initialized in the RTL. The INIT value is usually set in RTL.

5. What value is returned and where is this INIT value set?

   1'b1 is the value returned. The initial state (INIT values) of the sload_in register was initialized in the RTL at line 86 (*reset_example_top_async.v*).

6. What is the highest fanout net? What is the fanout? And what is the driver type?

   The highest fanout net is async_reset_L: Fanout = 4160 | Driver Type = LUT1.

7. What is the Total Number of Endpoints for Setup?

   12416.

8. What values are returned and where is this INIT value set?

   1'b1. This includes all of the replicated registers. The INIT value is initialized in the RTL.

9. Are there any high fanout nets in this no_reset project?

   Yes, but they are all sload_in registers. There are no resets in this project.

10. How does the reset bridge assert 'sync_reset_out'?

    It is asynchronously asserted when 'ext_reset_in' is asserted or MMCM 'lock' is deasserted.

11. How does the reset bridge de-assert 'sync_reset_out'?

    It is synchronously deasserted for the 'clk' domain. A separate reset bridge would be used for each and every clock domain.

12. What is the total number of Setup Endpoints analyzed for the sync_reset project?

    The total number of endpoints analyzed is 16645. Because the reset is synchronous, the reset signal endpoints are also included in the timing analysis.

13. What is the highest fanout net? What is the fanout? And what is the driver type?

    The highest fanout net is RESET_BRIDGE_100/sync_reset_out: Fanout = 4225 | Driver Type =FDPE

14. What is the list length that is returned for the endpoint fanout from the reset_clk100 net?

    4225, the same as that reported by report_high_fanout_nets.

15. What source clock pin is the start of this path analysis?

    In the Summary section, in Source field, and also the last line in Source Clock Path section, the indicated source clock pin is RESET_BRIDGE_100/sync_reset_out_reg/C.

16. What is the destination pin for this path analysis?

    In the Summary section, in Destination field, the pin is cntr_inst[0].count_reg[0][0]/R.

17. What is the reported driver type now for the reset_clk100 signal?

    BUFGCE

18. Where is the SYNC_RESET_BUFG shown in the path report?

    The Data Path shows the Clk-to-Q from the synch reset FDPE, the reset_clk100_i net to the BUFGCE, the BUFGCE prop delay, and the reset_clk100 net to a destination flip-flop—in this case an FDRE.

19. Complete the following table based on the previous questions and answers as well as on your analysis.

	Async Resets	**No Resets**	**Sync Resets**	**Sync Resets (BUFG)**
Total Reset Fanout	4161	0	4225	4225
Reset Driver Type	LUT1	None	FDPE	BUFGCE
Total Number (Setup) of Endpoints Analyzed	12416 (async resets not analyzed)	12416	16645	16645

Lab 4: Pipelining

2022.2

Abstract

This lab describes how pipelining can improve performance (increase clock rate and throughput) and facilitate timing closure.

This lab should take approximately 45 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Add pipeline register stages to improve performance
- Analyze the performance, latency, and timing closure impacts of pipelined designs
- List the properties associated with the `get_timing_paths` Tcl command
- Add the property to a custom Tcl script in order to obtain the Worst Negative Slack (WNS) value at each pipeline stage

Introduction

Pipelining is a design technique that splits a logic function into multiple cycles of combinatorial logic between register stages. The first stage can be accepting new input data while the last stage is finishing its processing of the previous stage. Pipelining is used to improve performance (increase clock rate and throughput) and facilitate timing closure.

Partitioning the combinatorial logic into smaller blocks between register stages and increasing the number of register stages will increase the performance of the design at the cost of increased utilization and latency. Too little pipelining will result in under-performing designs while too much pipelining might result in significant additional latency and diminishing performance gains.

The number and placement of pipelined registers need to be considered during initial coding. Adding pipeline register stages requires balancing pipeline trees and verifying operation with functional verification. Reducing the number of logic levels among registers along with their

respective wire-route delays can significantly improve overall timing performance as long as the additional register stages/latency can be tolerated.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

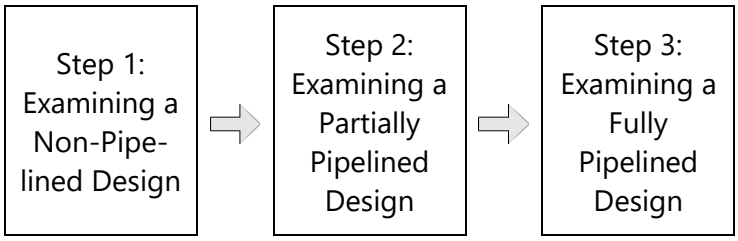
Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.

Symbol	Description	Example	Explanation
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Examining a Non-Pipelined Design Step 1

Here you will analyze a non-pipelined Vivado® Design Suite project.

1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-2. Open the Vivado Design Suite project named `cksum.xpr` located in the directory below.

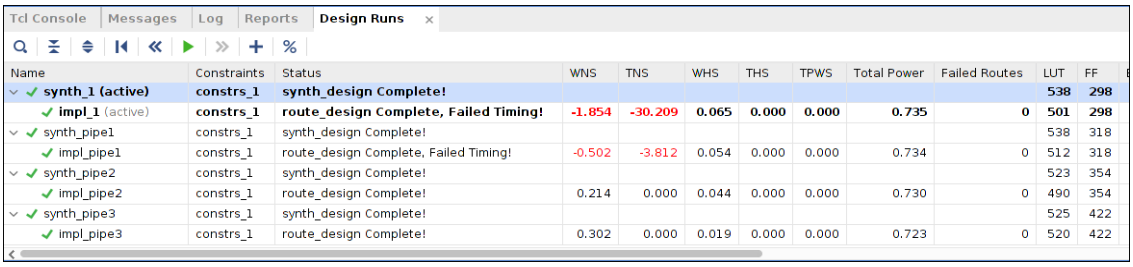
1-2-1. Browse to the `$TRAINING_PATH/Pipelining/lab/KCU105/cksum` directory and open the **`cksum.xpr`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-3. Examine the design run properties in the Design Runs tab at the bottom.

1-3-1. Select the **Design Runs** tab at the bottom, if it is not already selected.

Notice that there are four synthesis runs: *synth_1*, *synth_pipe1*, *synth_pipe2*, and *synth_pipe3*. Each of these runs corresponds to different pipeline stages.



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1 (active)	**constrs_1**	**synth_design Complete!**								**538**	**298**
impl_1 (active)	**constrs_1**	**route_design Complete, Failed Timing!**	**-1.854**	**-30.209**	**0.065**	**0.000**	**0.000**	**0.735**	**0**	**501**	**298**
synth_pipe1	constrs_1	synth_design Complete!								538	318
impl_pipe1	constrs_1	route_design Complete, Failed Timing!	-0.502	-3.812	0.054	0.000	0.000	0.734	0	512	318
synth_pipe2	constrs_1	synth_design Complete!								523	354
impl_pipe2	constrs_1	route_design Complete!	0.214	0.000	0.044	0.000	0.000	0.730	0	490	354
synth_pipe3	constrs_1	synth_design Complete!								525	422
impl_pipe3	constrs_1	route_design Complete!	0.302	0.000	0.019	0.000	0.000	0.723	0	520	422

Figure 4-1: Design Runs

1-3-2. Select **synth_pipe1** in the Design Runs tab to examine the properties of the synth_pipe1 design run.

1-3-3. Select **Window** > **Properties** to open the Properties window of the selected object.

You should see the Synthesis Run Properties window highlighted.

1-3-4. Select the **Properties** tab in the Synthesis Run Properties window.

1-3-5. Expand **STEPS** > **SYNTH_DESIGN** > **ARGS** in the Properties tab.

Question 1

What is entered under **SYNTH_DESIGN** > **ARGS** > **MORE OPTIONS**?

Each of the design runs uses a different conditional compile based on a `-verilog_define` attribute for the synthesis run. The Verilog `define used for each synthesis run is defined under **Synthesis Run Properties** > **Properties**.

1-3-6. Select the **Libraries** tab in the Sources window.

1-3-7. Expand **Design Sources** > **Verilog** > **xil_defaultlib**.

1-3-8. Double-click **cksum.v** to open it.

1-3-9. View the RTL, noting the `ifdef/`elsif/`else for the *PIPELINED1*, *PIPELNED2*, and *PIPELINED3* Verilog defines.

The four design runs correspond to different conditional compiles in cksum.v:

- No pipelining: Without the `-verilog_define` conditional compile option
- One pipe stage: `-verilog_define PIPELINED1`
- Two pipe stages: `-verilog_define PIPELINED2`
- Three pipe stages: `-verilog_define PIPELINED3`

1-4. Make sure that the synth_1 and impl_1 design runs are active.

1-4-1. Ensure that the **synth_1** synthesis design run is active in the Design Runs tab.

1-4-2. Right-click **synth_ 1** and select **Make Active**, if it is not already.

1-5. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-6. Generate a Timing Summary report.

If you do not recall how to perform this task, refer to the "Generating a Timing Summary Report on the Synthesized Design" or "Generating a Timing Summary Report on the Implemented Design" topics under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-6-1. Click the **Worst Negative Slack (WNS)** value in the Setup section of the Design Timing Summary window to view the most critical timing path details, such as Slack, Levels of logic, High Fanout, etc. Note down the WNS values.

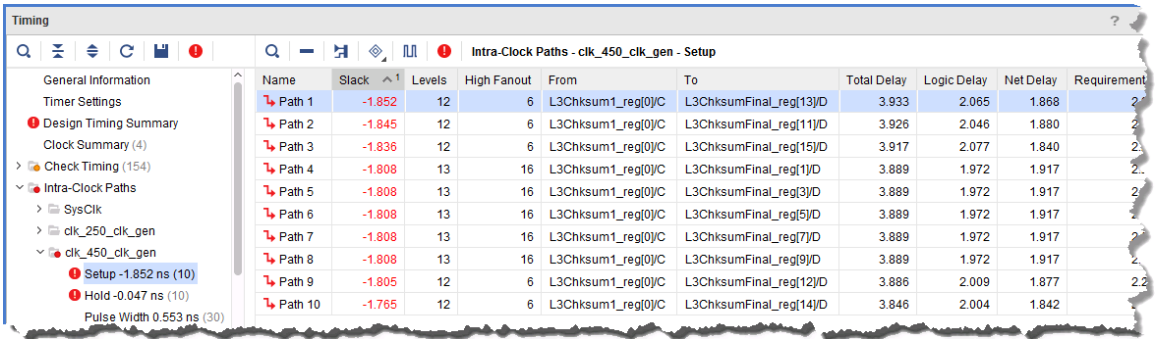


Figure 4-2: WNS Clock Paths without Pipelining (KCU105)

Question 2

Looking at Path 1, how many ‘stages’ or logic levels are indicated?

1-6-2. Select **Window** > **Properties** to examine the timing path properties in the Path Properties window.

Question 3

From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

It is clear that the design is far off from meeting the 450-MHz timing requirement for this no pipelining version (synth_1 project).

1-6-3. Right-click **Path 1** and select **Schematic** (or press <**F4**>).

You should see the schematic in the main window.

Question 4

Looking at the schematic for this WNS path, what are the names of the starting FDRE and ending FDRE registers?

1-6-4. Refer to the Path Properties window to answer the following questions.

Question 5

From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to the logic and what percentage delay is due to route?

Question 6

From **Path Properties** > **Summary** for Path 1, under Logic Levels, how many are LUTs? How many are CARRY primitives?

1-6-5. Examine the **cksum.v** RTL source at lines 240-250.

```
`else
    assign L3ChksumPartial =  L3Chksum7 + L3Chksum6 + L3Chksum5 + L3Chksum4 +
                              L3Chksum3 + L3Chksum2 + L3Chksum1 + L3Chksum0;
`endif

    // add the carries to get the 16-bit 1s complement sum
    always @ (posedge SysClk)
    begin
        L3ChksumFinal   <= #1 ~( Add1Comp(L3ChksumPartial[15:0], {13'h0000,L3ChksumPartial[18:16]}) );
        cksum_valid     <= #1 cksum_valid_1;
    end
```

Figure 4-3: RTL Snippet of synth_1 (No Pipelining)

Question 7

Having noted the WNS path and referring to the source RTL file (i.e., *cksum.v*) lines 240-250, (for this compiled version with no `-verilog_define`), what can you do in the RTL to increase performance at the expense of latency?

1-7. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Keep the project open for the next step, where you will examine the same project after the pipeline stages have been added.

Examining a Partially Pipelined Design — Step 2

2-1. The design runs *synth_pipe1* and *synth_pipe2* have added pipelined stages.

The following code snippet taken from cksum.v shows three pipelining stages.

```
`elsif PIPELINED2
    // Partial and Final sums
    always @ (posedge SysClk)
    begin
        L3Chksum7654_Partial <= #1 L3Chksum7 + L3Chksum6 + L3Chksum5 + L3Chksum4;
        L3Chksum3210_Partial <= #1 L3Chksum3 + L3Chksum2 + L3Chksum1 + L3Chksum0;
        cksum_valid_2        <= #1 cksum_valid_3;

        // Generate 2s complement sum
        L3ChksumPartial <= #1 L3Chksum7654_Partial + L3Chksum3210_Partial;
        cksum_valid_1   <= #1 cksum_valid_2;
    end
`elsif PIPELINED1
    // Partial and Final sums
    always @ (posedge SysClk)
    begin
        // Generate 2s complement sum
        L3ChksumPartial <= #1 L3Chksum7 + L3Chksum6 + L3Chksum5 + L3Chksum4 +
                              L3Chksum3 + L3Chksum2 + L3Chksum1 + L3Chksum0;
        cksum_valid_1   <= #1 cksum_valid_2;
    end
`else
    assign L3ChksumPartial =  L3Chksum7 + L3Chksum6 + L3Chksum5 + L3Chksum4 +
                              L3Chksum3 + L3Chksum2 + L3Chksum1 + L3Chksum0;
`endif

    // add the carries to get the 16-bit 1s complement sum
    always @ (posedge SysClk)
    begin
        L3ChksumFinal   <= #1 ~( Add1Comp(L3ChksumPartial[15:0], {13'h0000,L3ChksumPartial[18:16]}) );
        cksum_valid     <= #1 cksum_valid_1;
    end
```

PIPELINE 2

PIPELINE 1

PIPELINE 1

combinatorial

Figure 4-4: synth_pipe1 and synth_pipe2 RTL Code

When no pipeline is inserted (see the conditional compile for `else), *L3ChksumPartial* is combinatorial.

The *synth_pipe1* (`define PIPELINED1) run has added a single additional pipe register stage, registering *L3ChksumPartial* as shown in the figure above.

The *synth_pipe2* (`define PIPELINED2) run adds yet another register stage, breaking the partial sums into two pipeline registers: *{L3Chksum7654_Partial, L3Chksum3210_Partial}*, and *L3Chksum_partial*.

2-2. Make the *synth_pipe1* design run the active run in order to analyze the design with a single additional pipeline register stage.

2-2-1. Select **synth_pipe1** in the Design Runs tab.

2-2-2. Right-click and select **Make Active**.

2-3. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-4. Generate a Timing Summary report.

If you do not recall how to perform this task, refer to the "Generating a Timing Summary Report on the Synthesized Design" or "Generating a Timing Summary Report on the Implemented Design" topics under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-4-1. Examine the **WNS**, **TNS**, **WHS**, **THS**, and **Pulse Width** details for the most timing critical path in the Design Timing Summary tab.

2-4-2. Click the **Worst Negative Slack (WNS)** value in the Setup section of Timing Summary window. Note down the WNS values.

Question 8

Looking at Path 1, how many ‘stages’ or logic levels are indicated?

2-4-3. Right-click **Path 1** and select **Schematic** (or press <**F4**>).

You should see the schematic in the main window and the Path Properties to the left.

Question 9

Looking at the schematic for this WNS path, what are the names of the starting FDRE and ending FDRE registers?

Question 10

From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

Although a pipeline stage for calculating the partial L3Checksum has been added, the design is still far off from meeting the 450-MHz timing requirement for this pipe1 version.

Question 11

From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to the logic and what percentage delay is due to route?

Question 12

From **Path Properties** > **Summary** for Path 1, under Logic Levels, how many are LUTs? How many are CARRY primitives?

2-5. Another way to obtain this information is by running a Tcl script that lists the properties associated with WNS timing paths.

2-5-1. Enter the following command in the Tcl Console to create a new Tcl command that lists timing properties such as logic levels and WNS:

```
source {$TRAINING_PATH/Pipelining/support/
get_logic_levels_plus.tcl}
```

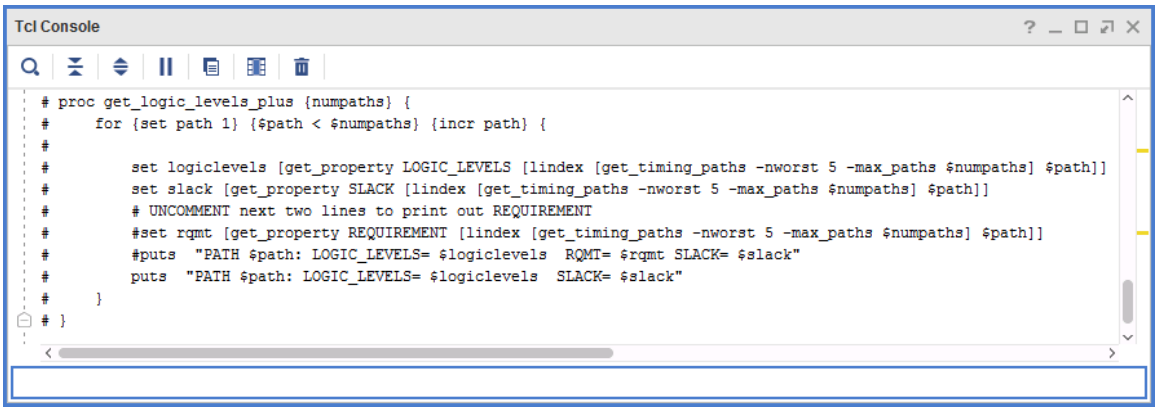


Figure 4-5: Sourcing a Tcl Script from the Console

This creates a new Tcl command: `get_logic_levels_plus`.

2-5-2. Enter the following command in the Tcl Console to list the timing properties in the Tcl Console:

```
get_logic_levels_plus 10
```

This will print the 10 WNS paths, number of logic levels, and slack in the Tcl Console.

The Tcl script that you previously executed to create this new command uses `get_timing_paths`.

2-5-3. Enter the following command in the Tcl Console to see the description, syntax, and usage of this command:

```
get_timing_paths -help
```

2-5-4. Enter the following command in the Tcl Console:

```
list_property [get_timing_paths]
```

This displays a list of the properties associated with the `get_timing_paths` Tcl command.

Question 13

How many properties do you see using the `list_property[get_timing_paths]` Tcl command?

Question 14

What Tcl command could you use to directly provide the returned Tcl list length?

2-5-5. Open the Tcl script `get_logic_levels_plus.tcl` from the following location with a text editor:

`$TRAINING_PATH/Pipelining/support`

Question 15

If you wanted this Tcl script to also print out a new property, such as REQUIREMENT, how would you modify the Tcl script?

2-5-6. Uncomment the lines in the script that add the REQUIREMENT property to be printed.

2-5-7. Select **File** > **Text Editor** > **Save File** to save the file.

2-5-8. Source the Tcl script again as you did previously and test your changes.

2-6. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-7. Open *synth_pipe2* from the design runs.

2-7-1. Make the **synth_pipe2** run active.

2-7-2. Enter the following command in the Tcl Console to open the synth_pipe2 design run:

```
open_run synth_pipe2
```

2-7-3. Generate a Timing Summary report.

Note down the WNS values.

2-7-4. Enter the following command in the Tcl Console:

```
get_logic_levels_plus 10
```

Question 16

For the worst-case path, how many logic levels are there? What is the estimated slack?

2-8. Close the synthesized design for the synth_pipe2 design run.

2-8-1. Enter the following command in the Tcl Console to close the synthesized design for the synth_pipe2 design run:

```
close_design
```

Examining a Fully Pipelined Design — Step 3

3-1. The design run *synth_pipe3* has added an additional pipeline stage.

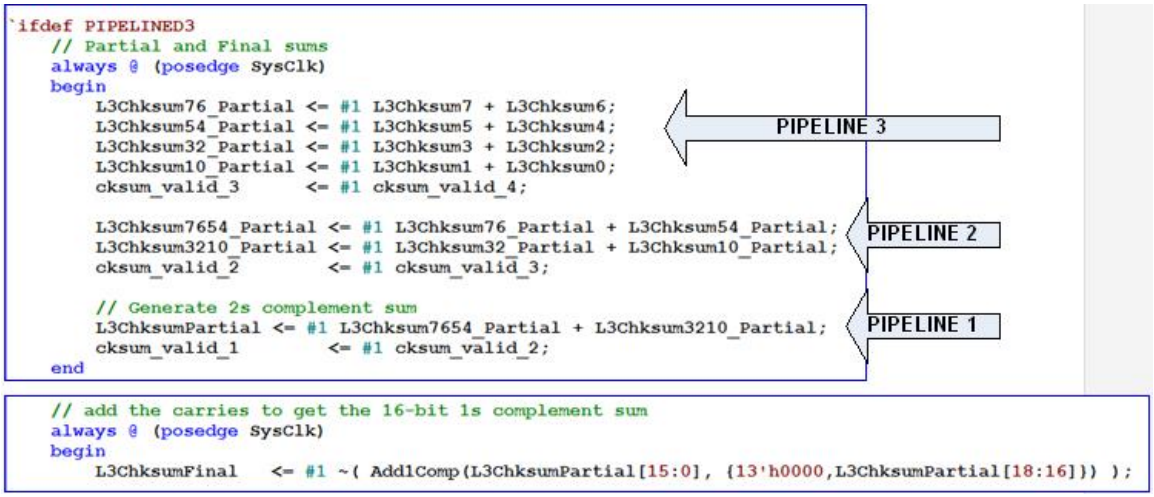


Figure 4-6: synth_pipe3 RTL Code

The *synth_pipe3* (`define PIPELINED3) run adds yet a third register stage, breaking the partial sums into three pipeline stages:

- ***{L3Chksum76_Partial, L3Chksum54_Partial, L3Chksum32_Partial* and *L3Chksum10_Partial}***
- ***{L3Chksum7654_Partial, L3Chksum3210_Partial}***
- ***L3Chksum_Partial***

3-2. Make the *synth_pipe3* design run the active run in order to analyze the design with three pipeline register stages.

3-2-1. Select **synth_pipe3** in the Design Runs tab.

3-2-2. Right-click and select **Make Active**.

3-3. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-4. Generate a Timing Summary report.

If you do not recall how to perform this task, refer to the "Generating a Timing Summary Report on the Synthesized Design" or "Generating a Timing Summary Report on the Implemented Design" topics under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-4-1. Click the **Worst Negative Slack (WNS)** value in Setup section of the Timing Summary window. Note down the WNS values.

Question 17

Looking at Path 1, how many ‘stages’ or logic levels are indicated?

Question 18

From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

The design is now meeting the 450-MHz timing requirement for the pipe3 version with this final stage of pipelining.

Question 19

From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to logic and what percentage is due to route?

Question 20

From **Path Properties** > **Summary** for Path 1, under Logic levels, how many are LUTs? How many are CARRY primitives?

3-4-2. Enter the following command in the Tcl Console to print the 10 WNS paths, number of logic levels, and slack in the Tcl Console:

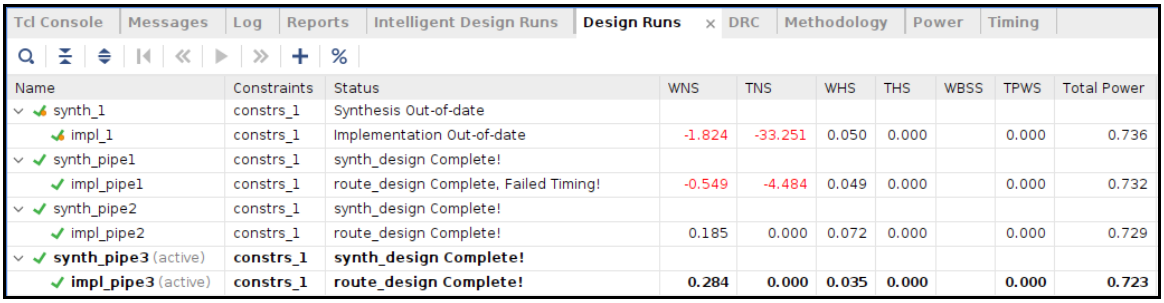
```
get_logic_levels_plus 10
```

3-5. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-6. View the actual WNS values from the implemented design runs.

3-6-1. Examine the WNS for all four implemented designs in the Design Runs tab.



Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power
synth_1	constrs_1	Synthesis Out-of-date							
impl_1	constrs_1	Implementation Out-of-date	-1.824	-33.251	0.050	0.000		0.000	0.736
synth_pipe1	constrs_1	synth_design Complete!							
impl_pipe1	constrs_1	route_design Complete, Failed Timing!	-0.549	-4.484	0.049	0.000		0.000	0.732
synth_pipe2	constrs_1	synth_design Complete!							
impl_pipe2	constrs_1	route_design Complete!	0.185	0.000	0.072	0.000		0.000	0.729
synth_pipe3 (active)	**constrs_1**	**synth_design Complete!**							
impl_pipe3 (active)	**constrs_1**	**route_design Complete!**	**0.284**	**0.000**	**0.035**	**0.000**		**0.000**	**0.723**

Figure 4-7: Viewing the WNS for All Implementations (KCU105)

Note that your timing values may vary slightly.

Question 21

Complete the following table using your previous answers for synth_design WNS and the implemented WNS values in the Design Runs tab.

Design Runs	impl_design WNS (ns)
-1 (no pipelining)	
_pipe1	
_pipe2	
_pipe3	

Pipelining has clearly improved performance and facilitated timing closure. Each additional stage of pipelining has resulted in a performance improvement.

3-7. Close the Vivado Design Suite project.

If you do not recall how to perform this task, refer to the "Closing the Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/Pipelining` directory.

3-8. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

3-8-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/Pipelining`.

3-8-2. Select **`Pipelining`**.

3-8-3. Press <**Delete**>.

-- OR --

Using the command line:

3-8-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

3-8-5. Enter the following command to delete the contents of the workspace:

[Windows users]: **`rd /s /q $TRAINING_PATH/Pipelining`**

[Linux users]: **`rm -rf $TRAINING_PATH/Pipelining`**

Summary

Pipelining is an effective design technique that splits a logic function into multiple stages of combinatorial logic between register stages. The first stage can be accepting new input data while the last stage is finishing its processing of the previous stage. Pipelining can be used to effectively increase performance and facilitate timing closure.

Increasing the number of pipeline stages can significantly improve the performance of the design at the cost of increased utilization and increased latency. Too little pipelining will result in under-performing designs while too much pipelining might result in significant additional latency and diminishing performance gains.

The number and placement of pipelined registers need to be considered during initial coding. Functional verification should always be used to verify expected operation and ensure balancing of pipeline logic while noting the additional latency.

Answers

1. What is entered under **SYNTH_DESIGN** > **ARGS** > **MORE OPTIONS**?

   `-verilog_define PIPELINED1`

2. Looking at Path 1, how many ‘stages’ or logic levels are indicated?

   11 logic levels.

3. From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

   The requirement is 2.2 ns and slack is -1.779 ns.

4. Looking at the schematic for this WNS path, what are the names of the starting FDRE and ending FDRE registers?

   They are *L3Chksum1_reg[1]* and *L3ChksumFinal_reg[11]*, respectively.

5. From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to the logic and what percentage delay is due to route?

   ~ 50% logic delay and 50% route delay.

6. From **Path Properties** > **Summary** for Path 1, under Logic Levels, how many are LUTs? How many are CARRY primitives?

   LUT2 =1, LUT3 = 3, LUT5 =1 and CARRY8 = 6.

7. Having noted the WNS path and referring to the source RTL file (i.e., *cksum.v*) lines 240-250, (for this compiled version with no `-verilog_defines`), what can you do in the RTL to increase performance at the expense of latency?

   Add a pipeline register stage, making *L3Chksum* Partial registered instead of combinatorial.

8. Looking at Path 1, how many ‘stages’ or logic levels are indicated?

   7 logic levels.

9. Looking at the schematic for this WNS path, what are the names of the starting FDRE and ending FDRE registers?

   The starting FDRE and ending FDRE registers are *L3Chksum1_reg[1]* and *L3ChksumPartial_reg[17]*, respectively.

10. From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

    2.2 ns requirement; slack = -0.647 ns.

11. From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to the logic and what percentage delay is due to route?

    ~ 47% logic, 53% route.

12. From **Path Properties** > **Summary** for Path 1, under Logic Levels, how many are LUTs? How many are CARRY primitives?

    carry8 = 4, LUT3 =2, LUT5 =1.

13. How many properties do you see using the `list_property[get_timing_paths]` Tcl command?

    36 properties

14. What Tcl command could you use to directly provide the returned Tcl list length?

    ```
    llength [list_property [get_timing_paths]]
    ```

15. If you wanted this Tcl script to also print out a new property, such as REQUIREMENT, how would you modify the Tcl script?

    Add the commented out (#) lines to use `get_property` REQUIREMENT and use `put string` (`puts`) to print it to the console.

16. For the worst-case path, how many logic levels are there? What is the estimated slack?

    The logic level and estimated slack are 4 and 0.392, respectively.

17. Looking at Path 1, how many ‘stages’ or logic levels are indicated?

    3 logic levels.

18. From **Path Properties** > **Summary** for Path 1, what is the timing requirement and estimated slack?

    2.2 ns requirement; slack = 0.780 ns.

19. From **Path Properties** > **Summary** for Path 1, under Data Path Delay, what percentage delay is due to logic and what percentage is due to route?

    ~ 57% logic and 43% route.

20. From **Path Properties** > **Summary** for Path 1, under Logic levels, how many are LUTs? How many are CARRY primitives?

    carry8 =2 and LUT3 =1.

21. Complete the following table using your previous answers for synth_design WNS and the implemented WNS values in the Design Runs tab.

Design Runs	impl_design WNS (ns)
-1 (no pipelining)	-1.824
_pipe1	-0.549
_pipe2	0.185
_pipe3	0.284

Your values may vary slightly, depending on your system configuration.

Lab 5: Designing with the IP Integrator

2022.2

Abstract

This lab shows you how to create the IPI subsystem design by using the Vivado® IP integrator (IPI).

This lab should take approximately 60 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Use the Vivado IP integrator to create a subsystem for your design
- Generate the output products of the subsystem

Introduction

The Vivado IP integrator lets you create complex system designs by instantiating and interconnecting IP from the Vivado IP catalog on a design canvas. You can create designs interactively through the IP integrator canvas GUI or programmatically through a Tcl programming interface. Designs are typically constructed at the interface level (for enhanced productivity) but can also be manipulated at the port level (for precision design manipulation).

The *uart_led* design will be created in this lab using the Vivado IP integrator (IPI).

The *uart_led* design receives data on a serial RX port and displays its binary equivalent value on LEDs.

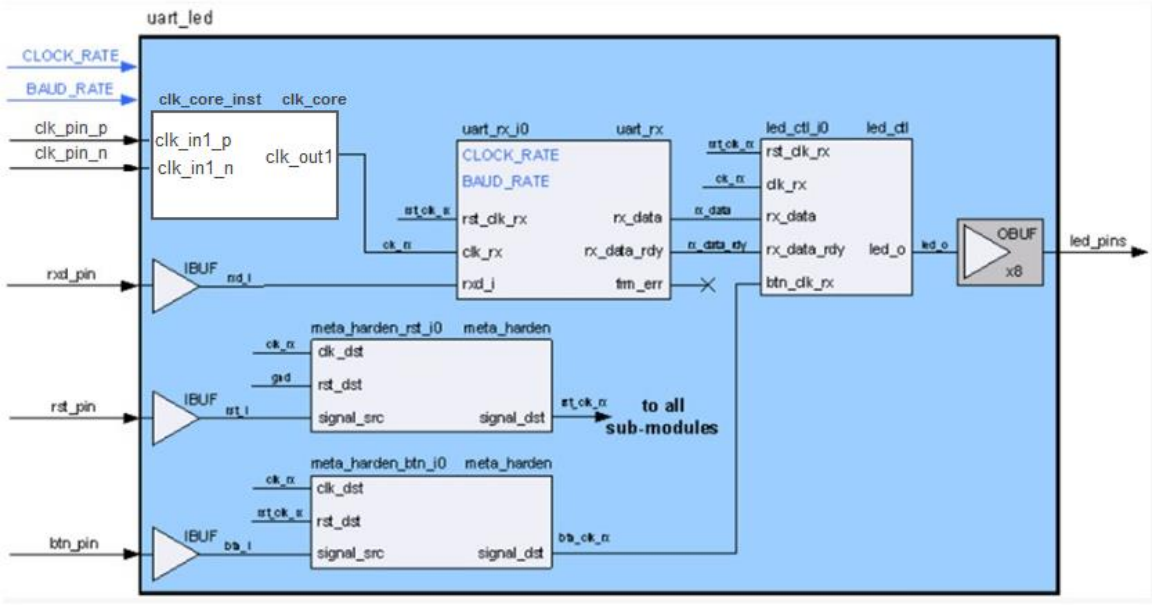


Figure 5-1: Block Diagram of the uart_led Design

This design implements an RS-232 protocol that receives serial data at 115200 baud rate (no parity, 8 data bits, no handshaking). When a character is successfully received, its binary equivalent displays on the LEDs. The eight significant bits are shown by default. Pressing a button (btn_pin) on the board shows the swapping of the four most and least significant bits.

Each module in the *uart_led* project has already been converted to IP and you will be adding these IPs to the Vivado IP catalog. Using the Vivado IPI, you will be including/adding all these IPs to create the *uart_led* design.

This lab also demonstrates using block designs as containers by using a simple logic block. This feature of the Vivado IP integrator enables you to instance one block design inside of another block design.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

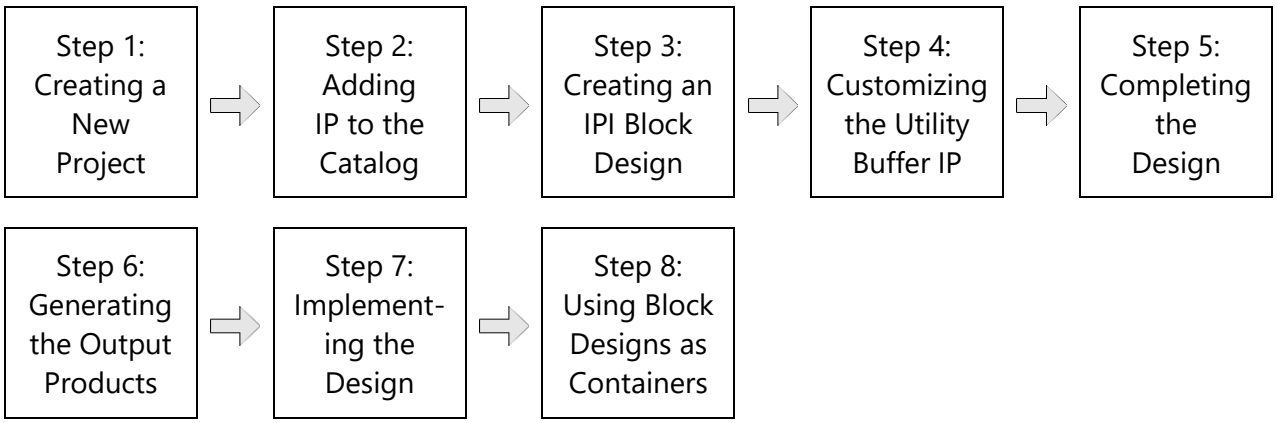
If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Creating a New Vivado Design Suite Project Step 1

You will begin the lab by creating a new Vivado Design Suite project via the **New Project** link in the Getting Started Welcome page.

1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-2. Create a new project.

Projects begin with the creation of a new project. The project contains sources, settings, graphics, IP, and other elements that are used to build a final bitstream.

1-2-1. Click **Create Project** (1).

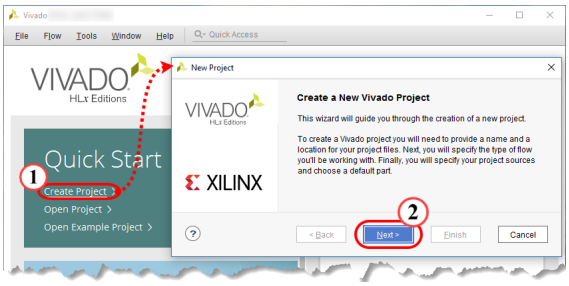


Figure 5-2: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

1-2-2. Click **Next** to begin entering the specifics for this project (2).

1-3. You will now encounter a series of dialog boxes asking you to enter different pieces of information describing the project.

1-3-1. Enter **UART_LED_Subsystem** in the Project name field.

1-3-2. Enter **`$TRAINING_PATH/IP_Integrator/lab/KCU105/<language>`** in the Project location field, where `<language>` can be either `verilog` or `VHDL`.

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

1-3-3. Deselect the **Create Project Subdirectory** option (if selected) as leaving this checked will create an unnecessary level of hierarchy for this lab.

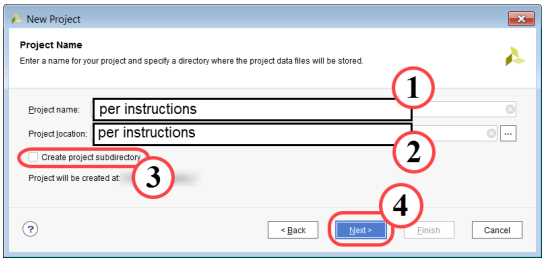


Figure 5-3: Entering the Project Name and Location

1-3-4. Click **Next** to accept the selections and advance to selecting a type of project.

The Project Type dialog box invites you to choose between an RTL project or a post-synthesis project. Simply put, an RTL project enables you to add or create new HDL files and synthesize them, whereas a post-synthesis project requires pre-synthesized files.

1-3-5. Select **RTL Project** (1).

1-3-6. Select **Do not specify sources at this time** to create a blank project (2).

While existing sources could be entered at this time, you will enter them later just to move through this portion of the project creation process more quickly.

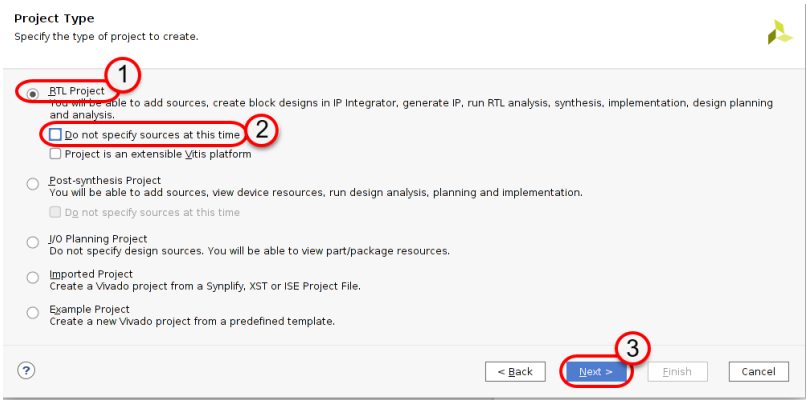


Figure 5-4: Setting the Project Type to RTL

1-3-7. Click **Next** to advance to the target device/platform selection (3).

1-4. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.

1-4-1. Select **Boards** from the Select area (1).

1-4-2. Select **All** from the Vendor drop-down list in the Filter area (2).

This filters the available boards to those that are populated with any member of the selected library.

1-4-3. Select **Kintex-UltraScale KCU105 Evaluation Platform** from the board list.

Alternatively, you can select the board directly from the list at any time while in this dialog box.

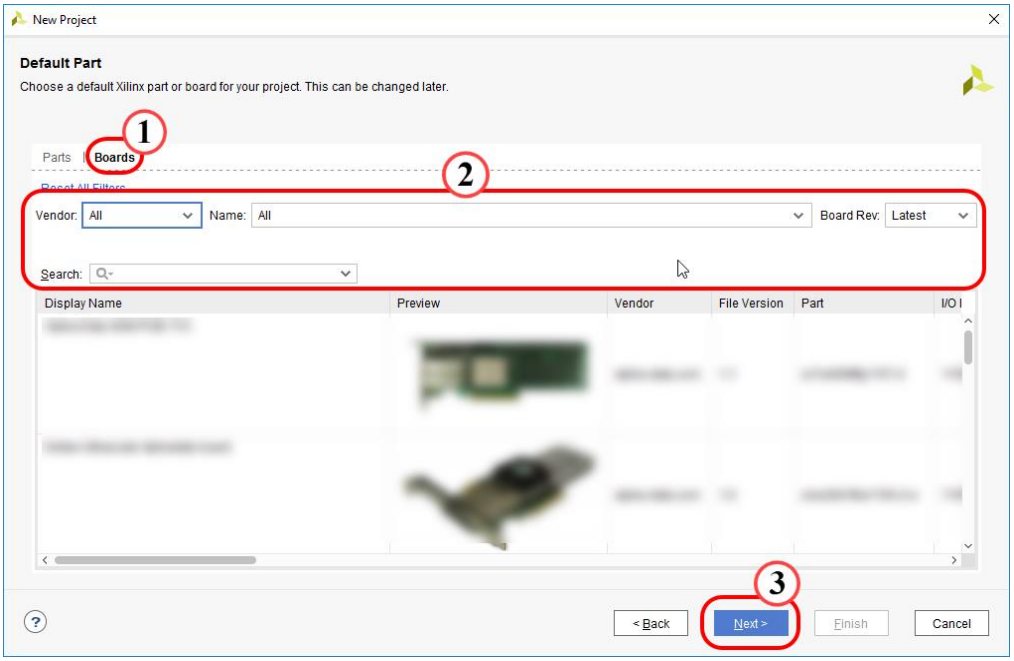


Figure 5-5: Selecting the Board for the Project

1-4-4. Click **Next** to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box and making the correction, or you can create the project now and edit the project properties, add or remove files, etc. later.

1-4-5. Click **Finish** to accept these settings and build the project.

1-5. Set the target language.

1-5-1. Click **Settings** under Project Manager in the Flow Navigator.

1-5-2. Select your preferred language from the Target Language drop-down list.

1-5-3. Click **OK** in the Project Settings dialog box.

Adding IP to the IP Catalog — Step 2

Before creating the UART_LED_Subsystem block design/subsystem using the IP integrator, you need to add the IPs used in the *uart_led* design to the IP catalog repository because these are custom IPs.

As per the *uart_led* block diagram, there are four IP: *led_ctl*, *meta_harden*, *uart_baud_gen*, and *uart_rx_ctl*. These IP have already been created for you. You will be adding these IP to the Vivado IP catalog.

2-1. Open the IP catalog and add the *uart_led* design IP to the IP catalog repository.

2-1-1. Click **IP Catalog** under Project Manager in the Flow Navigator**.**

Alternatively, you can select **Window** > **IP Catalog**.

2-1-2. Click **Settings** under **Flow Navigator** and go to IP.

The Settings window opens with the IP section selected.

2-1-3. Click **IP > Repository** (1).

2-1-4. Click the + icon to add the user IP repository to the IP catalog (2).

2-1-5. Browse to the `$TRAINING_PATH/IP_Integrator/lab/KCU105/<language>` directory (3), where `<language>` can be either `verilog` or `VHDL`.

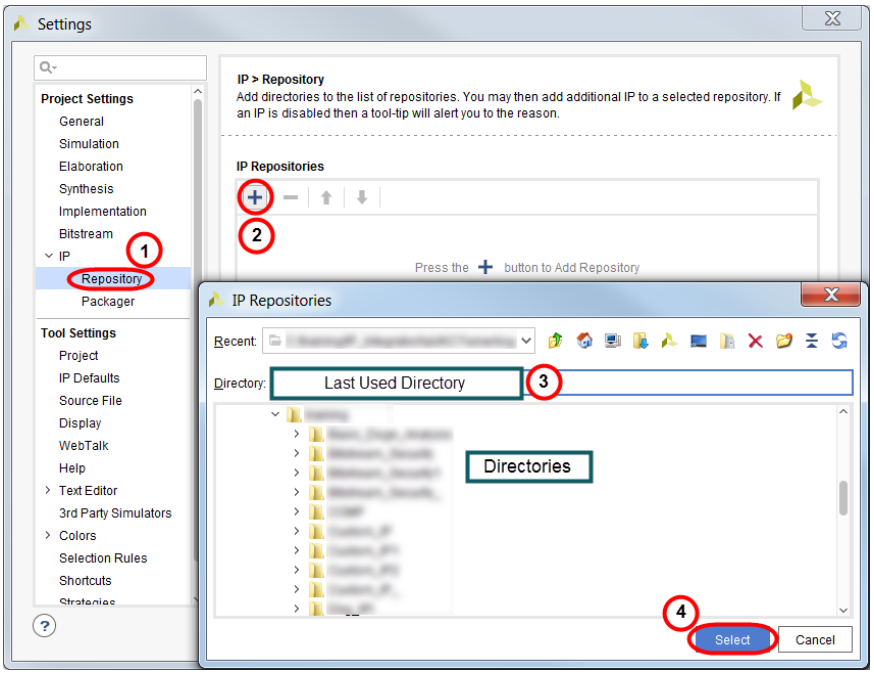


Figure 5-6: Adding IP to the IP Catalog

2-1-6. Click **Select** to add the IP to the IP catalog repository (4).

2-1-7. Click **OK** in the Add Repository dialog box.

2-1-8. Click **OK** in the Settings dialog box.

2-1-9. Expand **User Repository** > **UserIP** in the IP Catalog window to view the recently added IPs.

You should find the four IP in the selected repository; i.e, *led_ctl_v1_0*, *meta_harden_v1_0*, *uart_baud_gen_v1_0*, and *uart_rx_ctl_v1_0*.

Creating an IP Integrator Block Design — Step 3

Designs are typically constructed at the interface level (for enhanced productivity) but can also be manipulated at the port level (for precision design manipulation).

The Vivado IP integrator is a graphical tool that assists you in "stitching" together various pieces of IP. This tool can be used for both embedded and non-embedded designs.

3-1. Create a Vivado IP integrator block diagram.

3-1-1. Expand **IP INTEGRATOR** in the Flow Navigator if necessary (1).

3-1-2. Click **Create Block Design** to start creating a new IP subsystem (2).

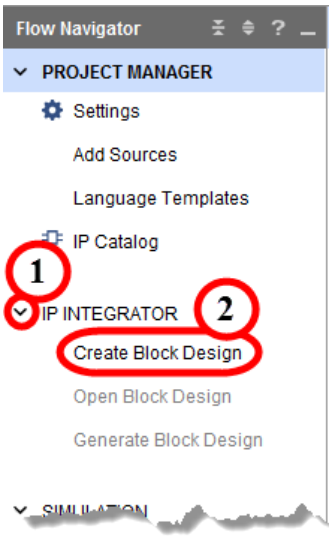


Figure 5-7: Launching the IP Integrator

3-1-3. Name the design **UART_LED_Subsystem** when the Create Block Design dialog box opens (1).

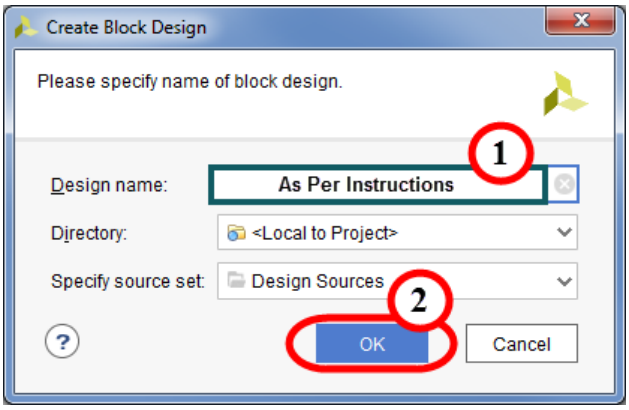


Figure 5-8: Creating an IP Integrator Block Design

3-1-4. Click **OK** to open a new, blank IP integrator canvas (2).

The IP integrator workspace opens with a note in the canvas area inviting you to begin adding IP.

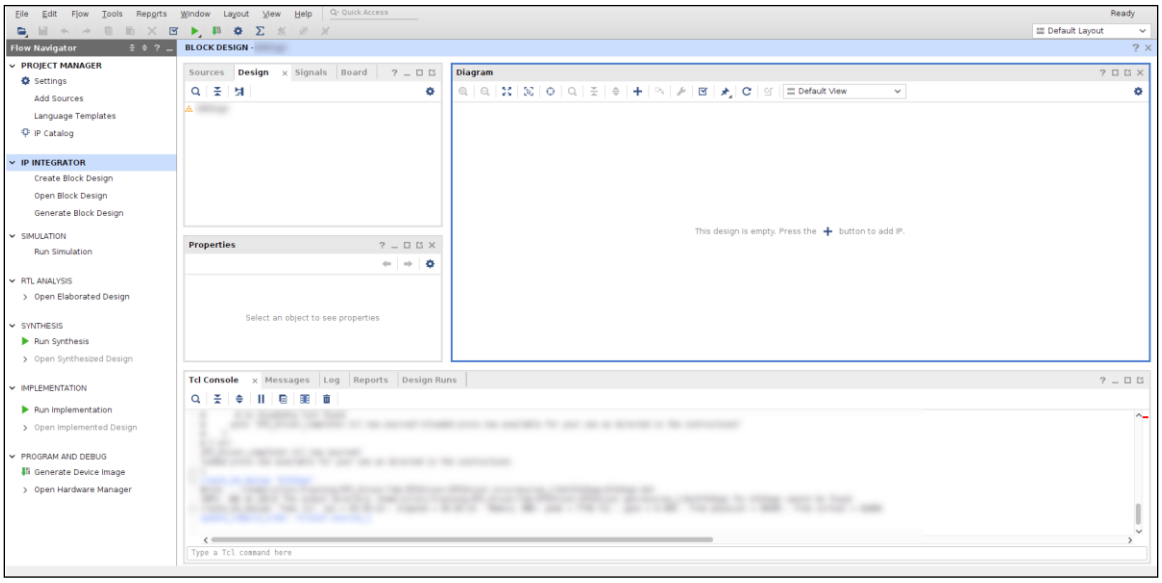


Figure 5-9: Initial View of the IP Integrator Tool

Tip: All of the features of the IP integrator are available through Tcl commands. The equivalent Tcl command for creating the new block diagram is `create_bd_design UART_LED_Subsystem`.

In the previous step, you added all the IPs required to create the UART_LED subsystem into the Vivado IP catalog. Now, it is time to include those IP into the UART_LED subsystem.

The following IP will be added to the UART_LED subsystem:

- *uart_baud_gen_v1_0*
- *uart_rx_ctl_v1_0*
- *led_ctrl_v1_0*
- *meta_harden_v1_0* (thrice)
- Utility Buffer

3-2. Add the *uart_baud_gen* and *uart_rx_ctl* IPs to subsystem.

3-2-1. Right-click in the IP integrator design canvas and select **Add IP**.

The IP integrator IP catalog opens, displaying a list of IP available in the IP integrator.

3-2-2. Type **uart** in the search box at the top of the IP integrator catalog.

3-2-3. Select both the **uart_baud_gen_v1_0** and **uart_rx_ctl_v1_0** IP by using the <**Ctrl**> key.

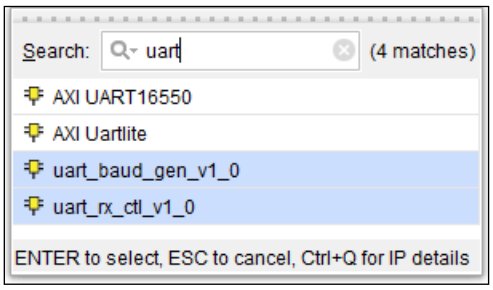


Figure 5-10: uart IP in the IP Catalog

3-2-4. Press <**Enter**>.

The IP are added in the integrator design canvas.

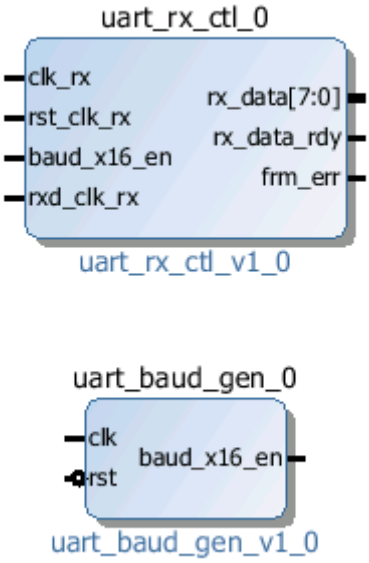


Figure 5-11: uart IP Added to the Design Canvas

3-3. Add the *led_ctrl* IP to the subsystem.

3-3-1. Right-click in the IP integrator design canvas and select **Add IP**.

3-3-2. Type **led** in the search box at the top of the IP integrator catalog.

3-3-3. Select the **led_ctrl_v1_0** IP from the results.

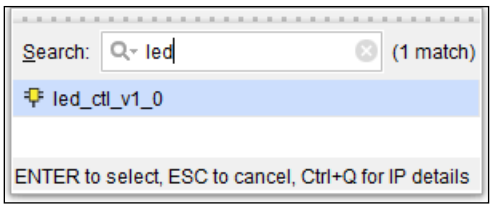


Figure 5-12: led IP in the IP Catalog

3-3-4. Double-click the IP to add it to the design canvas.

3-4. Add the *meta_harden* IP three times to the subsystem.

3-4-1. Right-click in the IP integrator design canvas and select **Add IP**.

3-4-2. Type **meta_harden** in the search box at the top of the IP integrator catalog.

3-4-3. Select the **meta_harden_v1 _0** IP from the results.

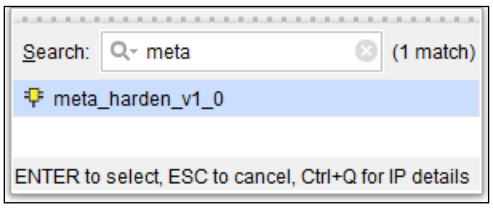


Figure 5-13: meta_harden IP in the IP Catalog

3-4-4. Double-click the IP to add it to the design canvas.

3-4-5. Repeat adding the *meta_harden* IP two more times because you need to add the *meta_harden* IP three times to the subsystem.

There will be three *meta_harden* modules in the design with each making sure that the input port does not face the meta stability condition.

3-5. Add the Utility Buffer block.

3-5-1. Right-click in the Diagram tab window and select **Add IP**.

3-5-2. Enter **buffer** in the Search field.

3-5-3. Double-click **Utility Buffer** to add it to the design canvas.

Customizing the Utility Buffer IP Step 4

The Utility Buffer core generates corresponding buffers to bring off-chip signals into or out from internal circuits.

4-1. Customize the Utility Buffer IP.

4-1-1. Double-click the **Utility Buffer** IP.

4-1-2. Make sure that the **DIFF_CLK_IN** IP interface is associated with board interface as **custom**.

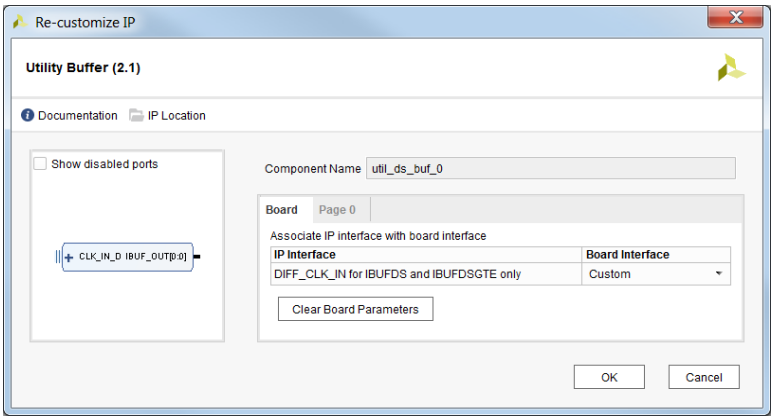


Figure 5-14: Utility Buffer IP Customization – Board Tab (KCU105)

4-1-3. Select the **Page 0** tab.

4-1-4. Make sure that the **IBUFDS** is selected as C_BUF type.

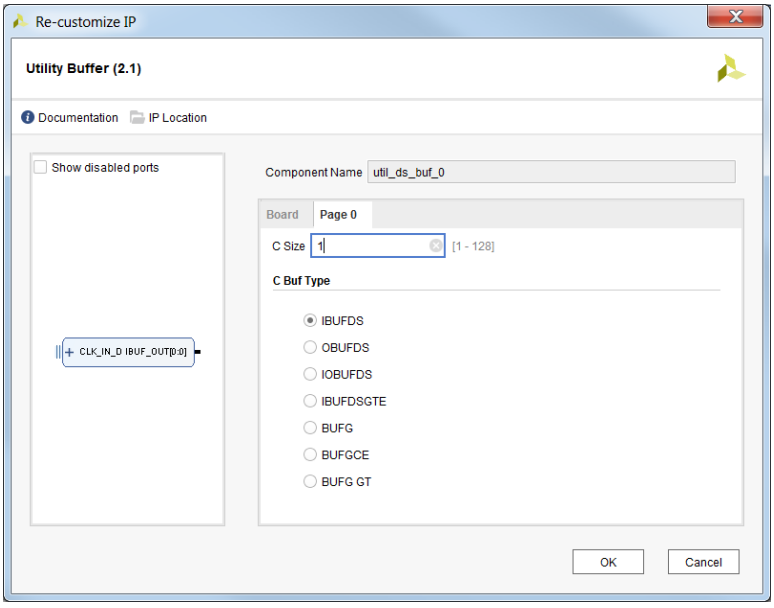


Figure 5-15: Utility Buffer IP Customization – Page 0 Tab (KCU105)

4-1-5. Click **OK**.

Completing the Subsystem Design — Step 5

Here you will complete the subsystem by making connections between the IP and creating external ports (such as input and output ports) for the top-level design.

5-1. Create external ports for the IPs.

5-1-1. Select the **Utility Buffer** IP.

5-1-2. Click the **+** symbol on **CLK_IN_D** to expand the signals.

5-1-3. Select the **IBUF_DS_P[0:0]** pin and then press and hold the <**Ctrl**> key and select the **IBUF_DS_N[0:0]** pin of the Utility Buffer IP.

You can select multiple ports and add external connections to all of them at the same time.

5-1-4. Highlight the ports, right-click, and select **Make External**.

5-1-5. Select the external connection port connected to **IBUF_DS_P**.

5-1-6. Right-click and select **External Port Properties**.

5-1-7. Type the name **clk_pin_p** and press <**Enter**> in the Name field of the General tab.

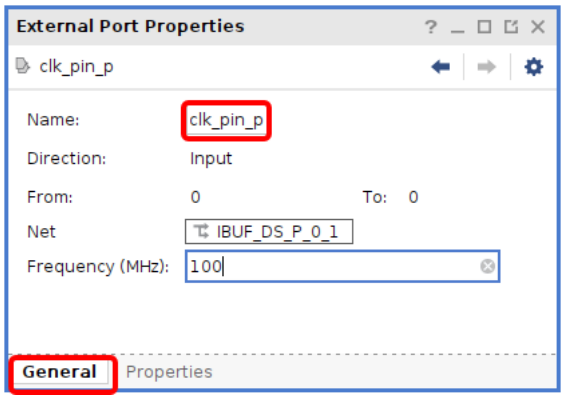


Figure 5-16: External Port Properties

5-1-8. Select the external connection port connected to **IBUF_DS_N**.

5-1-9. Select the **External Port Properties** window.

5-1-10. Type the name **clk_pin_n** and press <**Enter**> in the Name field of the General tab.

5-1-11. Click the **Regenerate Layout** icon ( ) if you would like the tool to arrange the blocks optimally.

Note: You should see the external connections made with the names *clk_pin_p[0:0]* and *clk_pin_n[0:0]*.

5-2. Create external ports for the *led_ctl_0* IP.

5-2-1. Select the **led_o[7:0]** port of the *led_ctl_0* IP.

5-2-2. Right-click and select **Make External**.

5-3. Rename the led_o[7:0] external port.

5-3-1. Select the external connection port connected to **led_o[7:0]**.

5-3-2. Select the **External Port Properties** window.

5-3-3. Type the name as **led_pins** and press <**Enter**> in the Name field of the General tab.

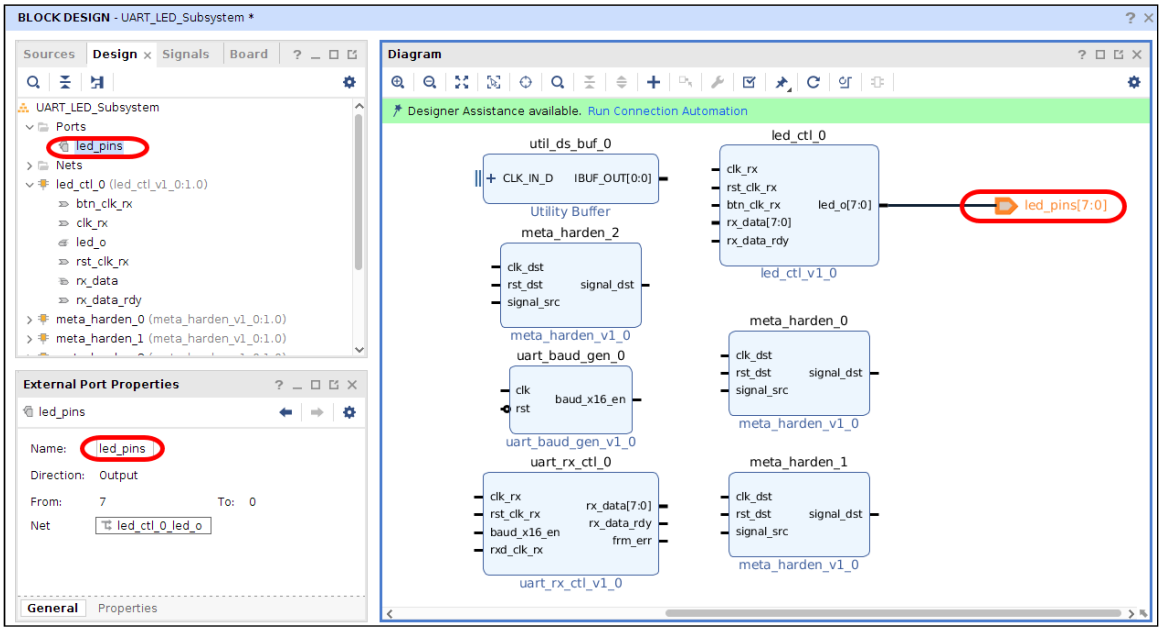


Figure 5-17: Making Ports External for the led_ctl IP - Renaming Connection (KCU105)

5-4. Rename the *meta_harden_1* IP to *meta_harden_btn*.

5-4-1. Select the **meta_harden_1** IP.

5-4-2. Select the **Block Properties** window.

5-4-3. Type the name **meta_harden_btn** and press <**Enter**> in the Name field of the General tab.

5-5. Make the input port of the *meta_harden_btn* IP external.

5-5-1. Select the **signal_src** input port of the *meta_harden_btn* IP in the design canvas.

5-5-2. Right-click and select **Make External**.

5-6. Rename this external pin to btn_pin.

5-6-1. Select the external connection port connected to **signal_src**.

5-6-2. Select the **External Port Properties** window.

5-6-3. Type the name **btn_pin** and press <**Enter**> in the Name field of the General tab.

5-7. Rename the *meta_harden_2* IP to *meta_harden_rst*.

5-7-1. Select the **meta_harden_2** IP.

5-7-2. Select the **Block Properties** window.

5-7-3. Type the name **meta_harden_rst** and press <**Enter**> in the Name field of the General tab.

5-8. Make the input port of the *meta_harden_rst* IP external.

5-8-1. Select the **signal_src** input port of the *meta_harden_rst* IP in the design canvas.

5-8-2. Right-click and select **Make External**.

5-9. Rename this external pin to rst_pin.

5-9-1. Select the external connection port connected to **signal_src**.

5-9-2. Select the **External Port Properties** window.

5-9-3. Type the name **rst_pin** and press <**Enter**> in the Name field of the General tab.

5-10. Make the input port of the *meta_harden_0* IP external.

5-10-1. Select the **signal_src** input port of the *meta_harden_0* IP in the design canvas.

5-10-2. Right-click and select **Make External**.

5-11. Rename this external pin to rxd_pin.

5-11-1. Select the external connection port connected to **signal_src**.

5-11-2. Select the **External Port Properties** window.

5-11-3. Type the name **rxd_pin** and press <**Enter**> in the Name field of the General tab.

5-12. Make the clock connections from the Utility Buffer block to the *meta_harden* IP in the subsystem.

5-12-1. Move the cursor to the **IBUF_out[0:0]** port of the **util_ds_buf_0** IP.

5-12-2. Click and drag the **IBUF_out** signal and connect it to the **clk_dst** signal of meta_harden_0.

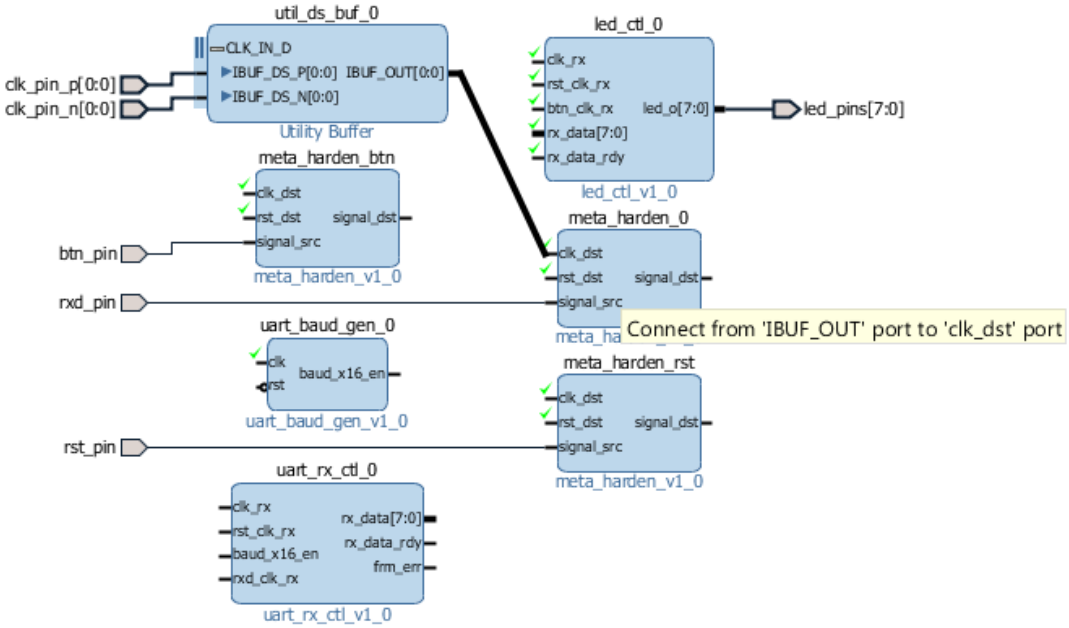


Figure 5-18: Clock Connection Between the Utility Buffer and meta_harden_0 IP (KCU105)

5-13. Similarly, make the connections to the meta_harden_btn, meta_harden_rst, led_ctl, uart_rx_ctl, and uart_baud_gen IP.

IBUF_out → clk_rx

IBUF_out → clk

IBUF_out → clk_rx

IBUF_out → clk_dst

IBUF_out → clk_dst

After you finish all the connections from the Utility Buffer block, your design should look like the figure below.

You can click the Regenerate Layout icon ( ) if you would like the tool to arrange the blocks.

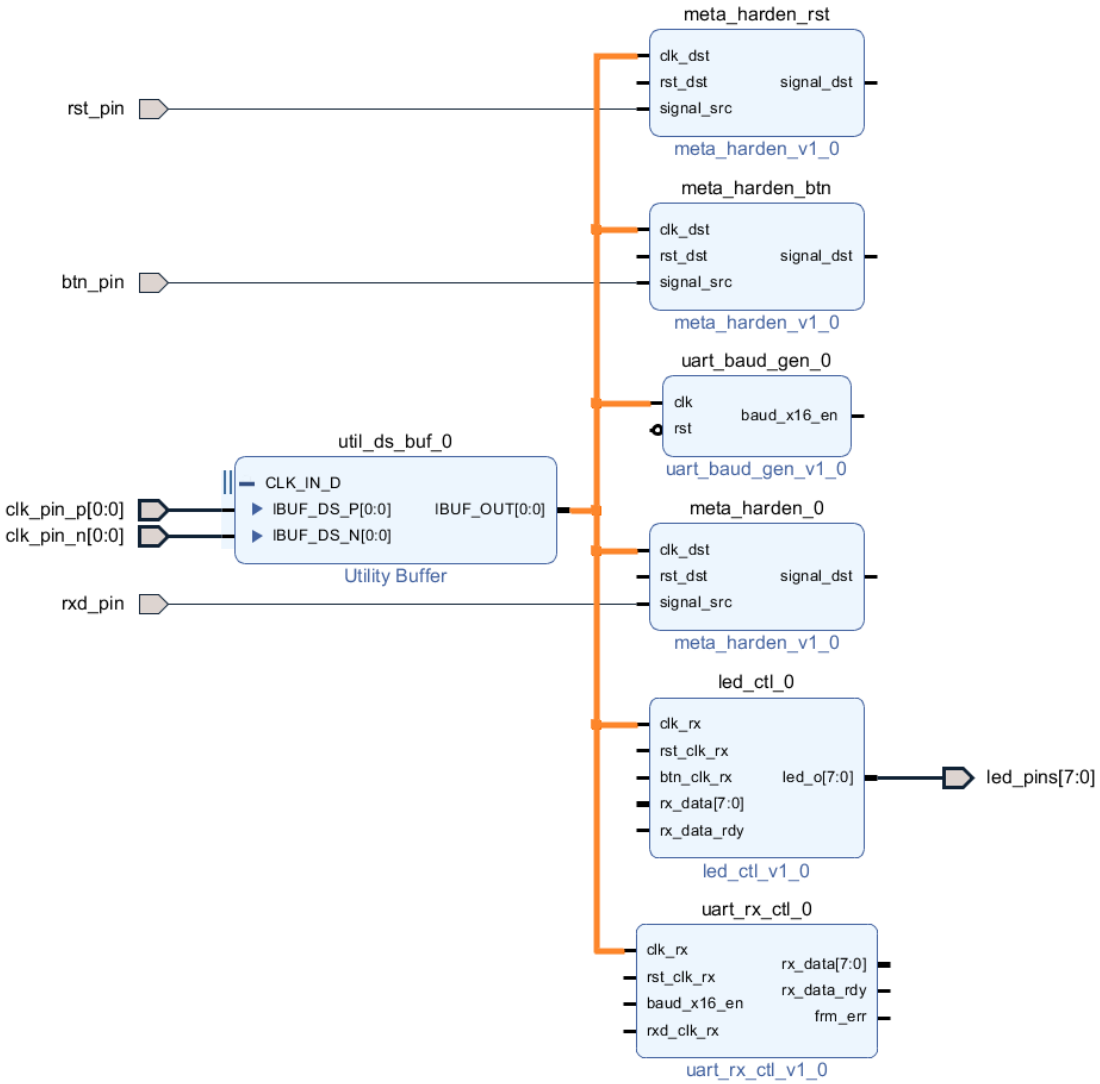


Figure 5-19: Completed Clock Connections (KCU105)

5-14. Make the reset connections from the *meta_harden_rst* IP.

5-14-1. Move the cursor to the **signal_dst** port of the *meta_harden_rst* IP.

The cursor pointer arrow changes to a pencil ( ).

5-14-2. Click and drag the **signal_dst** signal and connect to the **rst_dst** signal of *meta_harden_btn*.

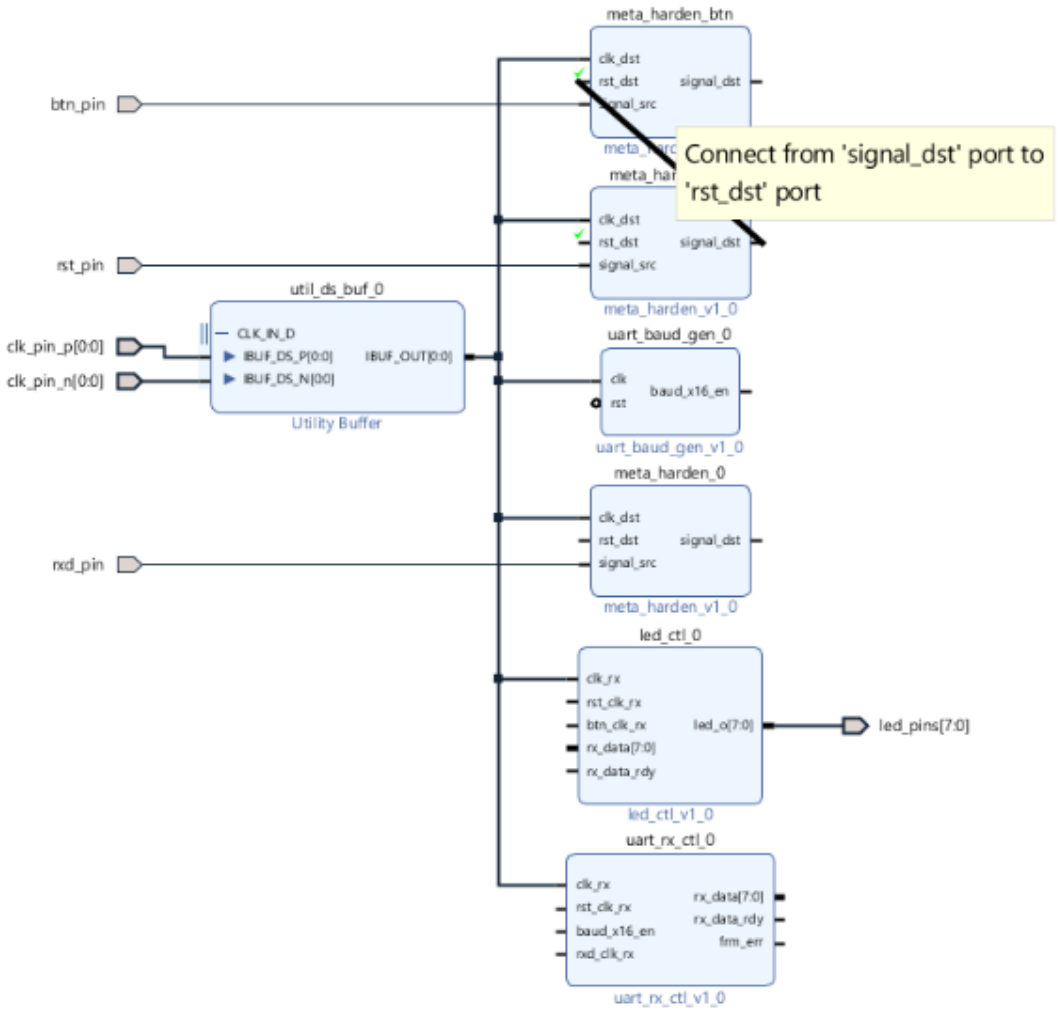


Figure 5-20: Reset Connection from meta_harden_rst to meta_harden_btn IP (KCU105)

5-15. Similarly, make the reset connections to the *meta_harden_0*, *led_ctl*, *uart_rx_ctl_0*, and *uart_baud_gen* IP.

5-15-1. From the *meta_harden_rst* IP to the *meta_harden_0* IP:

signal_dst → rst_dst

5-15-2. From the *meta_harden_rst* IP to the *uart_rx_ctl_0* IP:

signal_dst → rst_clk_rx

5-15-3. From the *meta_harden_rst* IP to the *uart_baud_gen_0* IP:

signal_dst → rst

5-15-4. From the *meta_harden_rst* IP to the *led_ctl_0* IP.

signal_dst → rst_clk_rx

After you finish all the connections from the meta_harden_rst IP, your design should look like the figure below.

You can click the **Regenerate Layout** icon ( ) if you would like the tool to arrange the blocks optimally.

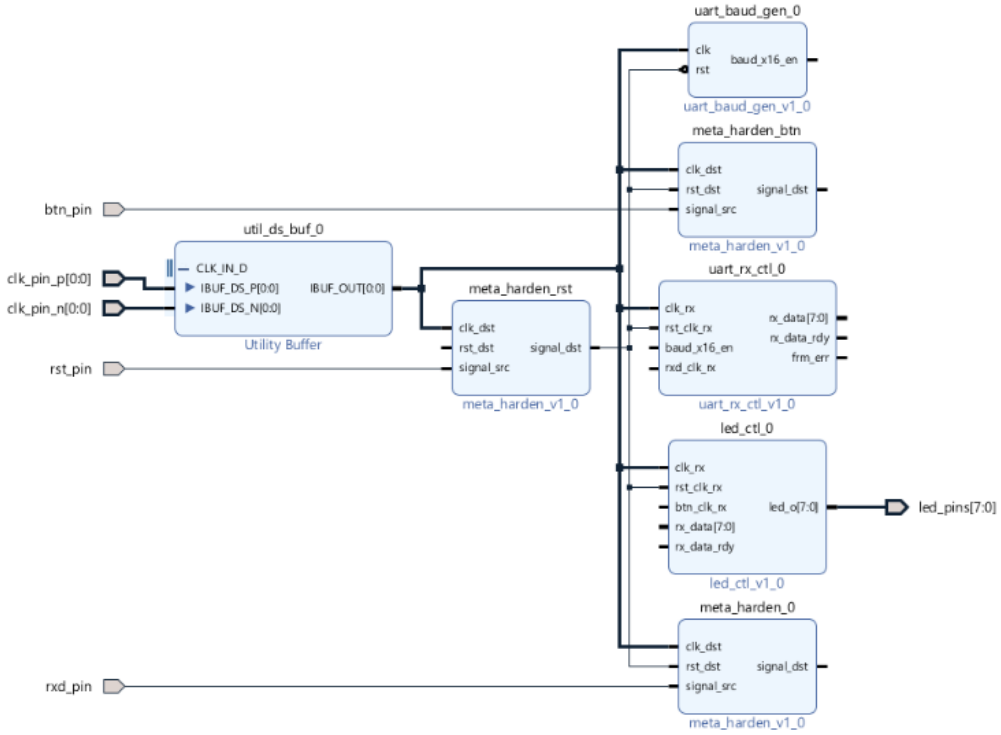


Figure 5-21: Reset Connections from the meta_harden_rst IP (KCU105)

5-16. Make the connections from the *meta_harden_0* IP to the *uart_rx_ctl_0* IP.

5-16-1. Move the cursor to the **signal_dst** port of the *meta_harden_0* IP.

The cursor pointer arrow changes to a pencil ().

5-16-2. Click and drag the **signal_dst** signal and connect to the **rxd_clk_rx** signal of *uart_rx_ctl_0*.

5-17. Make the connections from the *meta_harden_btn* IP to the *led_ctl_0* IP.

5-17-1. Move the cursor to the **signal_dst** port of the *meta_harden_btn* IP.

5-17-2. Click and drag the **signal_dst** signal and connect to the **btn_clk_rx** signal of *led_ctl_0*.

5-18. Make the connections from the *uart_rx_ctl_0* IP to the *led_ctl_0* IP.

5-18-1. Move the cursor to the **rx_data[7:0]** port of the *uart_rx_ctl_0* IP.

5-18-2. Click and drag the **rx_data[7:0]** signal and connect to the **rx_data[7:0]** signal of *led_ctl_0*.

5-18-3. Repeat the same for the **rx_data_rdy** signal.

5-19. Make the connections from the *uart_baud_gen_0* IP to the *uart_rx_ctl_0* IP.

5-19-1. Move the cursor to the **baud_x16_en** port of the *uart_baud_gen_0* IP

5-19-2. Click and drag the **baud_x16_en** port and connect to the **baud_x16_en** port of the *uart_rx_ctl_0* IP.

5-20. Create the *uart_rx_i0* hierarchy for the design.

5-20-1. Using the <**Ctrl**> key, select the **meta_harden_0**, **uart_baud_gen_0**, and **uart_rx_ctl_0** IPs**.**

5-20-2. Right-click and select **Create Hierarchy**.

5-20-3. Enter **uart_rx_i0** in the Cell name field of the Create Hierarchy dialog box.

5-20-4. Click **OK**.

5-21. Complete the reset connections by adding the Constant IP to the subsystem.

5-21-1. Right-click in the IP integrator design canvas and select **Add IP**.

5-21-2. Type **constant** in the search box at the top of the IP integrator catalog.

5-21-3. Select and double click the **Constant** IP from the results.

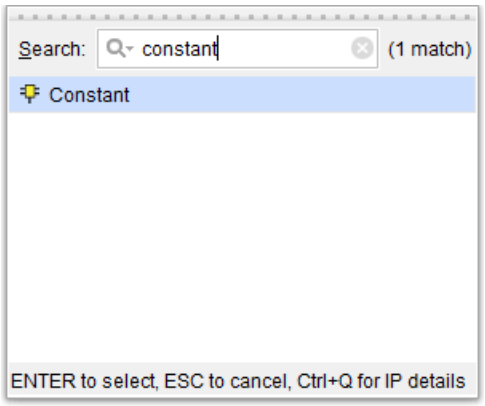


Figure 5-22: Searching for the Constant IP

5-21-4. Double-click the **xlconstant_0** IP.

5-21-5. Enter **0** in the Const Val field and click **OK**.

5-21-6. Move the cursor to the **rst_dst** port of the *meta_harden_rst* IP.

5-21-7. Click and drag the **rst_dst** signal and connect it to the external pin **dout[0:0]** of the *xlconstant_0* IP.

5-22. Create a comment in the canvas.

5-22-1. Right-click and select **Create Comment**.

5-22-2. Enter the following sentence in the comment field:

This is an IPI example project. This is to demonstrate that the IP integrator flow (IPI flow) can also be used for non-embedded designs.

5-23. Regenerate the layout.

5-23-1. Click the **Regenerate** (↻) icon from the toolbar.

When you have finished, your subsystem design should look like the figure below.

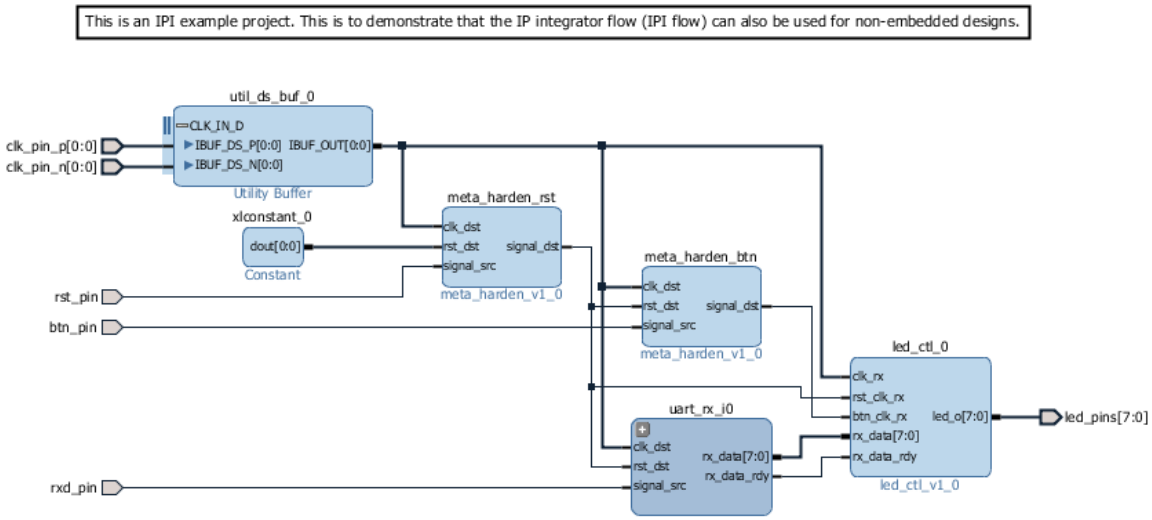


Figure 5-23: Completed UART_LED_Subsystem (KCU105)

5-24. Run a design check to make sure that the design does not contain any errors.

5-24-1. Select **Tools** > **Validate Design**.

You should see a message that validation was successful.

5-25. Save the block design.

5-25-1. Select **File** > **Save Block Design** to save the block design.

Generating the IP Output Products — Step 6

You will now generate the output products for the subsystem to synthesize and implement the design.

6-1. Generate the IP output products.

6-1-1. Select the **IP Sources** tab of the Sources view.

6-1-2. Select **UART_LED_Subsystem** under Block Designs.

6-1-3. Right-click and select **Generate Output Products**.

The Generate Output Products dialog box displays the various output products that are available for the subsystem design.

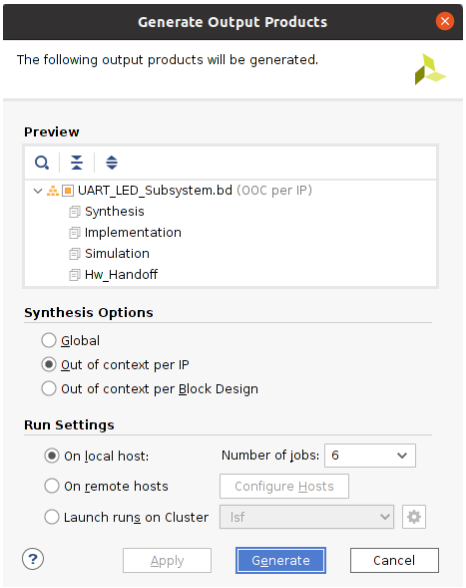


Figure 5-24: Generating Output Products

6-1-4. Click **Generate** to generate the output products for the subsystem with the default synthesis option.

6-1-5. Click **OK** in the Generate Output Products dialog box.

6-2. Generate the HDL wrapper to synthesize and implement the design.

6-2-1. Select **UART_LED_Subsystem** under Block Designs in the IP Sources tab.

6-2-2. Right-click and select **Create HDL Wrapper**.

The Create HDL Wrapper dialog box opens.

6-2-3. Select **Let Vivado manage wrapper and auto-update**.

6-2-4. Click **OK**.

After creating the HDL wrapper, the Vivado IDE automatically sets the subsystem as the top-level module for the design so that synthesis and implementation can be performed.

Hint: Select the **Hierarchy** tab of the Sources window to see UART_LED_Subsystem made as the top-level module.

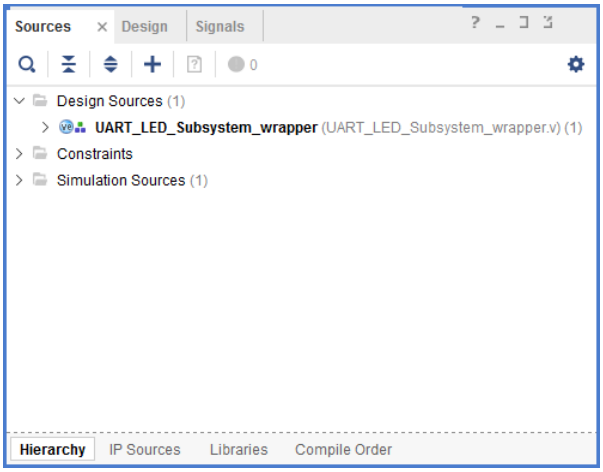


Figure 5-25: UART_LED_Subsystem as the Top-Level Module (Verilog)

6-3. Generate the Tcl script to save the block design and IP customization.

6-3-1. Select the **Tcl Console** tab at the bottom of the Vivado IDE or select **Window** > **Tcl Console**.

6-3-2. Enter the following Tcl command in the Tcl Console:

```
write_bd_tcl $::env(TRAINING_PATH)/IP_Integrator/lab/KCU105/
<language>/design.tcl
```

This creates the `design.tcl` script in the project directory.

Note that `<language>` can be either `verilog` or `VHDL`.

Implementing the Block Design — Step 7

You will now implement the design in this step.

7-1. Add the uart_led.xdc constraints file to the project.

7-1-1. Expand **Constraints** in the Sources tab to show constrs_1.

7-1-2. Right-click **constrs_1** and select **Add Sources**.

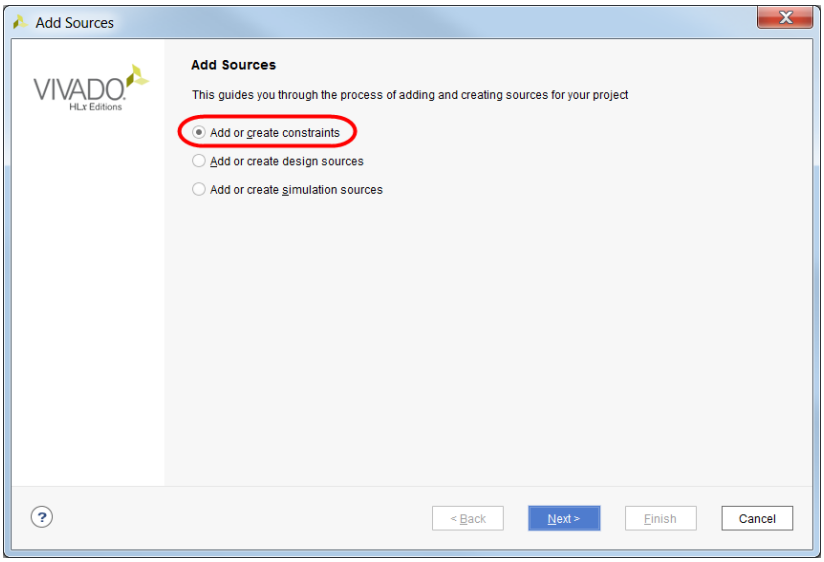


Figure 5-26: Add a Constraints File

7-1-3. Select the **Add or create constraints** option and click **Next**.

7-1-4. Select **Add Files** to choose the constraints file to add.

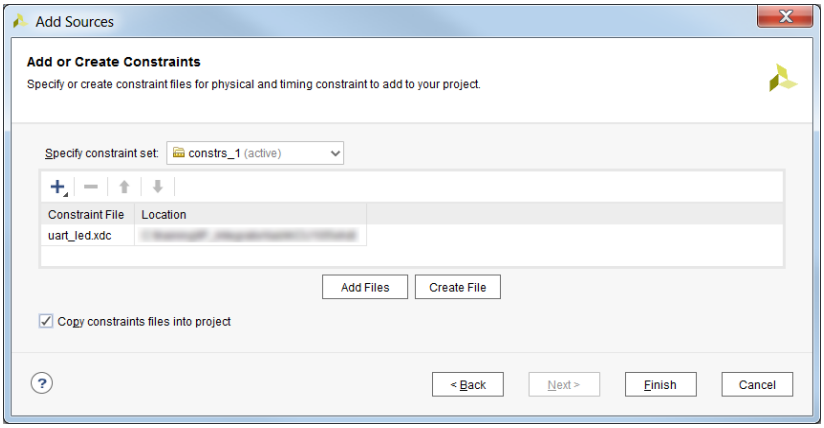


Figure 5-27: Adding the uart_led.xdc File

7-1-5. Go to `$TRAINING_PATH/IP_Integrator/support` and add the **`uart_led_IP_Integrator.xdc`** file.

7-1-6. Click **Finish** to add the file.

7-2. Implement the design.

7-2-1. Click **Run Implementation** in the Flow Navigator (under Implementation).

Alternatively, you can select **Flow** > **Run Implementation**.

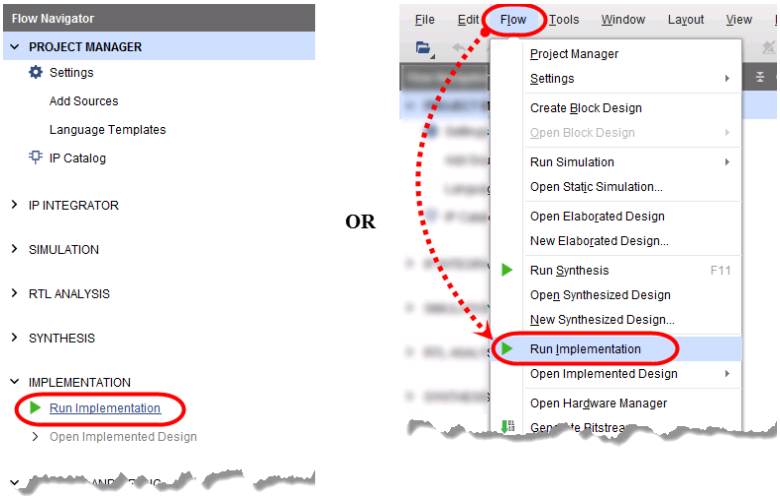


Figure 5-28: Selecting Run Implementation

Note: If needed, click **OK** to launch synthesis first if prompted.

Notice that the tools run all of the processes required to implement the design. This means that if the synthesized netlist is not available already, the Vivado Design Suite will run synthesis before running implementation.

After implementation completes, the Implementation Completed dialog box opens. The dialog box prompts you to open the implemented design, generate the bitstream, or view reports.

7-2-2. Click **OK** again to launch the selected runs.

7-2-3. Select **Open Implemented Design**.

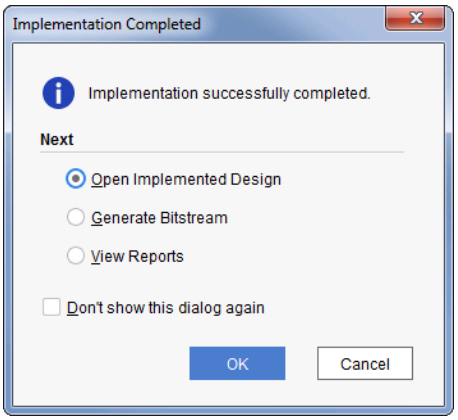


Figure 5-29: Implementation Completed Dialog Box

7-2-4. Click **OK**.

7-3. Close the block design.

7-3-1. Select **File** > **Close Block Design.**

Alternatively, you can click the "X" in the upper right-hand corner of the Block Design view.

7-3-2. Click **OK** when the Confirm Close dialog box appears.

Note: Select **Don't show this dialog again** if you do not want to confirm closure in the future.

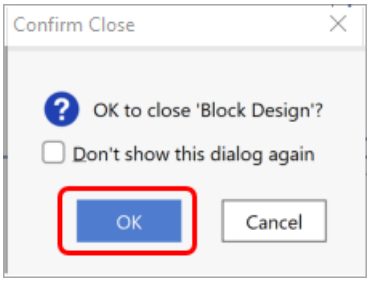


Figure 5-30: Confirming Block Design Closure

Using Block Designs as Containers (Optional) Step 8

One of the cornerstones of the IP integrator tool is reuse. The entire canvas becomes a container and can be used as IP. You will explore this capability now.

You will use a provided Tcl script to quickly add a registered vector XOR (built from discrete logic blocks rather than the the XOR capability built into the utility vector logic). This script creates a new block diagram containing some basic logic for the purposes of this example.

8-1. Run the `simpleLogic` Tcl script.

8-1-1. Enter the following command into the Tcl Console to source the `simpleLogic.tcl` script.

```
source $::env(TRAINING_PATH)/IP_Integrator/support/
SimpleLogic.tcl
```

This will create a new block design and populate it with an example design.

8-2. Create a new block design into which you will add multiple copies of the newly created *simpleLogic* block design.

8-2-1. From the Flow Navigator, click **IP Integrator** > **Create Block Design**.

8-2-2. Enter **topCanvas** in the Design name field.

8-2-3. Leave the other options at their default settings.

8-2-4. Click **OK**.

8-3. Add two copies of the *simpleLogic* block design to the open canvas (*topCanvas*).

8-3-1. Ensure that the **Sources** tab is selected (1).

8-3-2. Expand **Design Sources** (2).

8-3-3. Click-drag the **simpleLogic** block design into the *topCanvas* open canvas (3).

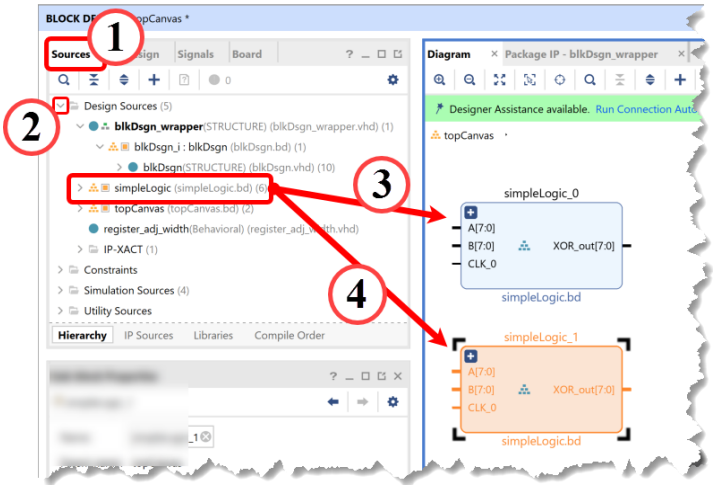


Figure 5-31: Adding One Canvas to Another

8-3-4. Repeat the click-drag to add a second copy of the *simpleLogic* block design into the *topCanvas* workspace (4).

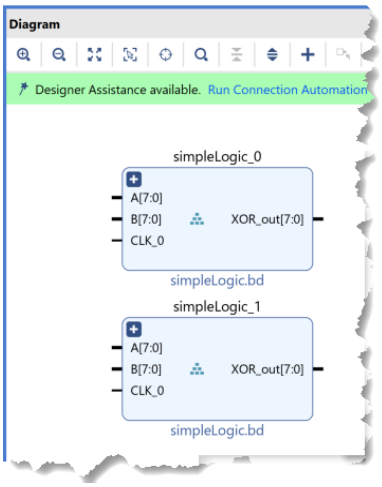


Figure 5-32: topCanvas Block Design Populated with Two Copies of the simpleLogic Block Design

At this point, the blocks can be wired together just like any other IP.

This capability is the logical extension of the hierarchy creation process you performed early; however, there is an important distinction: this block design can be independently tested and verified, whereas a hierarchal block is integrally connected with the overall block design and can only be tested as part of the whole.

The block design effectively becomes an IP block, although it will not appear in the IP catalog.

Both the block design container and the hierarchical structure can be expanded for the contents to be viewed.

8-3-5. Click the "+" symbol in the upper left-hand corner of the blocks to expand the block design containers to see their contents.

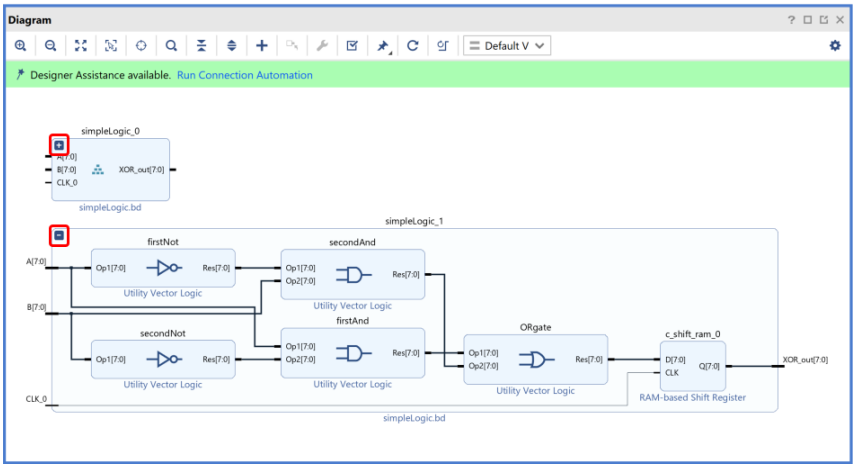


Figure 5-33: Expanding the Block Design Containers to View Their Contents

8-4. Save the block design and close the project.

8-5. Close the Vivado Design Suite.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/IP_Integrator` directory.

8-6. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

8-6-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/IP_Integrator`.

8-6-2. Select **`IP_Integrator`**.

8-6-3. Press <**Delete**>.

-- OR --

Using the command line:

8-6-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

8-6-5. Enter the following command to delete the contents of the workspace:

[Windows users]: **`rd /s /q $TRAINING_PATH/IP_Integrator`**

[Linux users]: **`rm -rf $TRAINING_PATH/IP_Integrator`**

Summary

In this lab, you learned how to create a subsystem/block design using the Vivado IP integrator GUI.

Lab 6: Baselining

2022.2

Abstract

This lab demonstrates the recommended Performance Baselining procedure for gaining timing closure progressively.

This lab supports both Verilog and VHDL design files. However, the lab instructions and figures show only Verilog usage.

This lab should take approximately 45 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Describe the Performance Baselining procedure
- Describe each stage in baselining

Introduction

Performance baselining is an iterative approach of adding timing constraints to your design, iterating through synthesis and implementation, then performing static timing analysis to isolate your timing-critical paths. It has three stages as shown in the figure below.

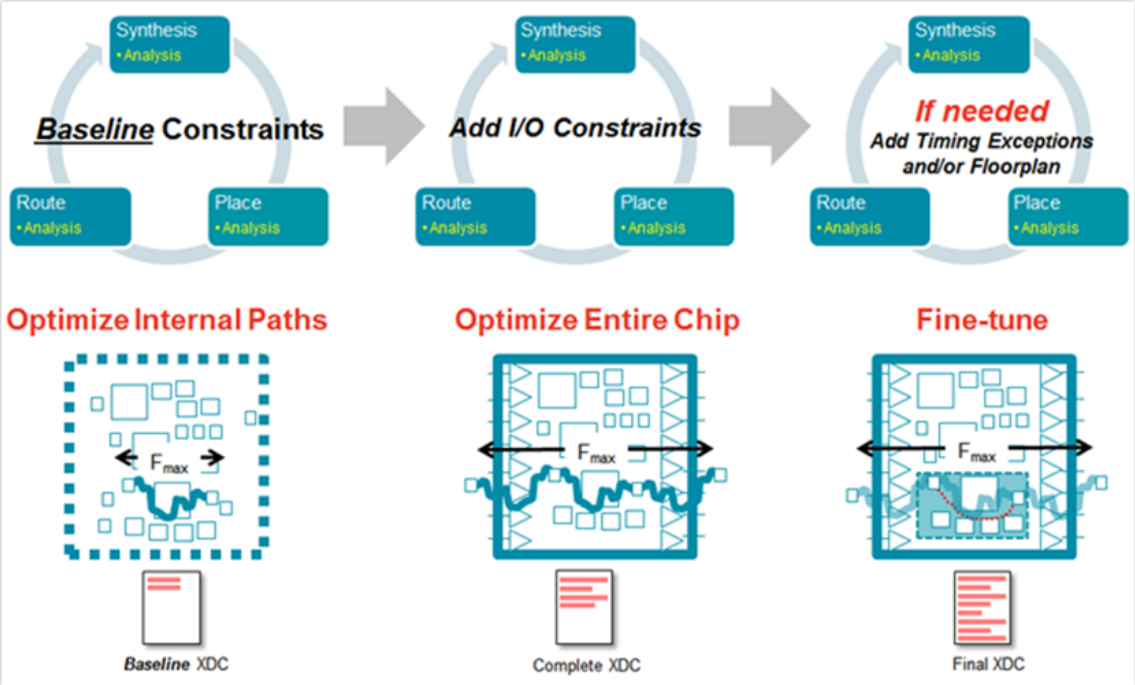


Figure 6-1: Progressive Approach for Design Timing Closure

The primary focus is to identify internal device timing challenges early in the process during the first stage. Then (still in the first stage) all the clock constraints and clock interactions are applied. I/O constraints can be added and verified during the second stage. Path-specific constraints such as multicycle paths, false paths, and max delay are added during the third stage.

This lab uses the *uart_led* design.

The *uart_led* design receives data on a serial RX port and displays its binary equivalent value on LEDs.

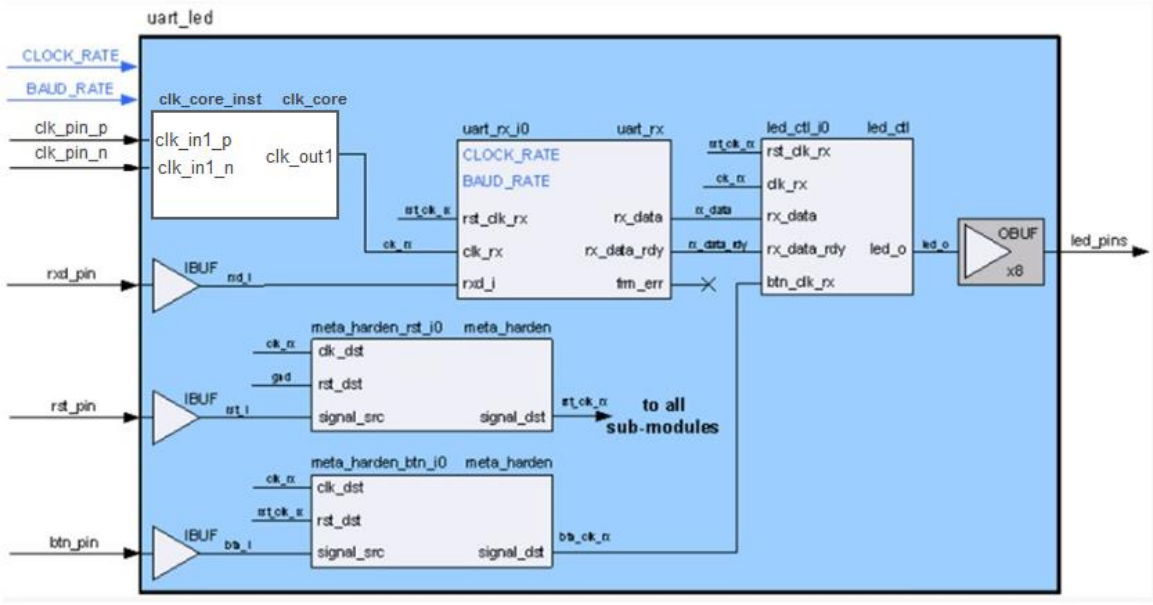


Figure 6-2: Block Diagram of the uart_led Design

This design implements an RS-232 protocol that receives serial data at 115200 baud rate (no parity, 8 data bits, no handshaking). When a character is successfully received, its binary equivalent displays on the LEDs. The eight significant bits are shown by default. Pressing a button (btn_pin) on the board shows the swapping of the four most and least significant bits.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

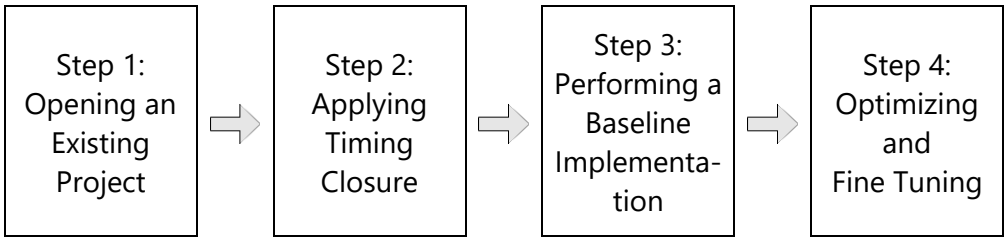
If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Opening an Existing Project — Step 1

You will begin by launching the Vivado® Design Suite and opening the *uart_led* design in the Vivado IDE.

Here are two ways to open the Vivado Design Suite.

1-1. Open the Vivado Design Suite.

1-1-1. Click the **Vivado** icon ( ) from the taskbar.

Figure 6-3: Launching the Vivado Design Suite from the Taskbar

Note: It takes a few moments to open. The order of the icons in your environment may be different.

Alternatively, from the Linux terminal window (<**Ctrl** + **Alt** + **T**>), enter the following:

```
source /opt/amd/Vivado/2022.2/settings64.sh; vivado
```

Note: This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

The tool opens to the Welcome window. From here you can create a new project, open an existing project, enter Tcl commands, and access documentation and examples.

1-1-2. [Optional] Maximize the window as there is a lot of information to see.

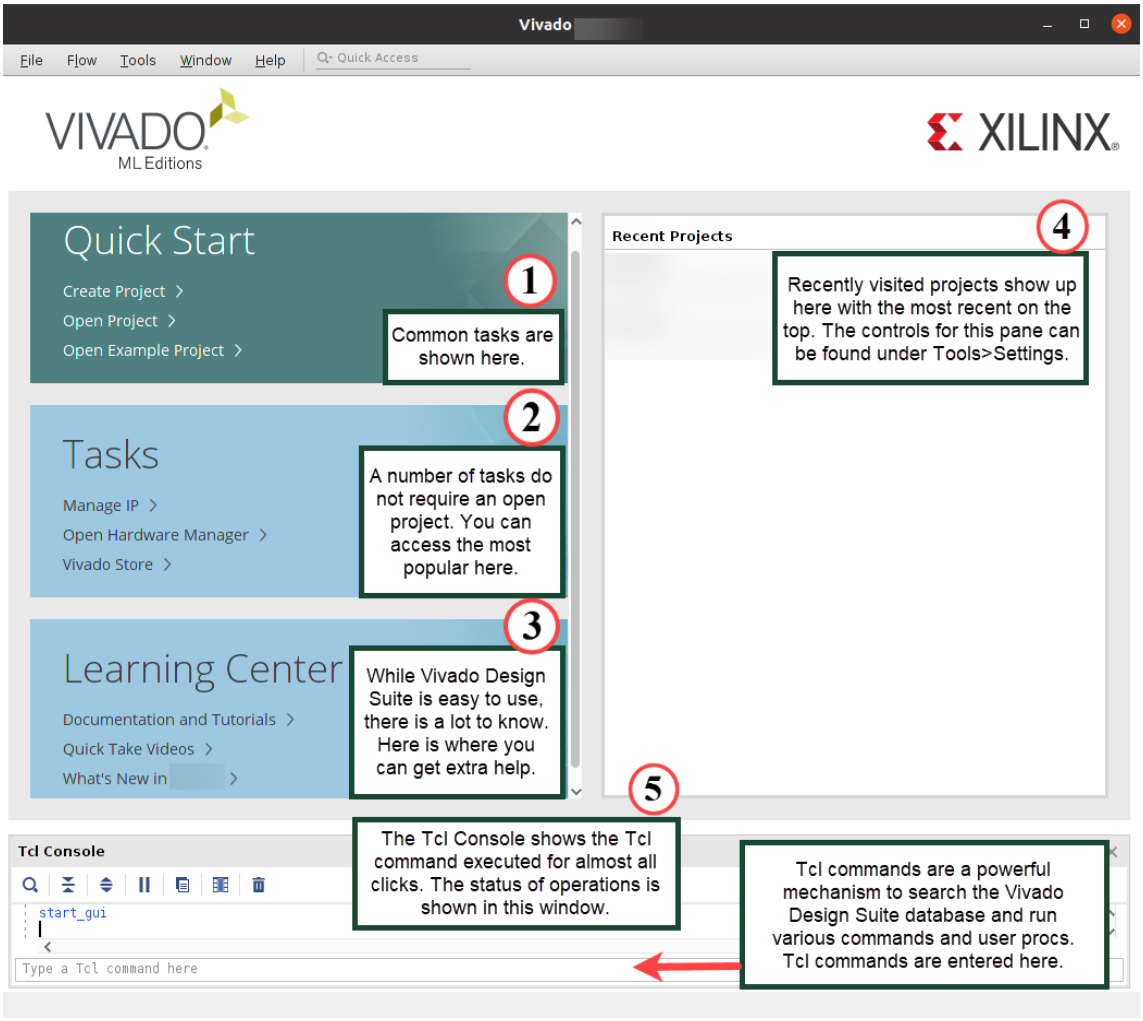


Figure 6-4: Vivado Design Suite Welcome Screen

Hint: If the Tcl Console is not visible, double-click the **Tcl** tab to make it visible.

1-2. Open the Vivado Design Suite project named `uart_led.xpr` located in the directory below.

1-2-1. Browse to the `$TRAINING_PATH/Baselining/lab/KCU105/[verilog | vhdl]` directory and open the **`uart_led.xpr`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

You can select either the KCU105 target board to perform the lab. Similarly, you can select either Verilog or VHDL to perform the lab.

Applying Timing Closure Methodology (Optimizing Internal Paths – Baseline) Step 2

You will perform most of the baselining steps in this lab. Your goal is to identify and specify all the clocks in the design and assess the feasibility of timing closure on the FPGA's internal paths. There are three broad stages in baselining.

- Optimize internal paths (baseline):
  - Define all primary clocks and generated clocks (period, edge relationships).
  - Specify asynchronous (unrelated) clocks: All clocks are assumed to be related to each other in the XDC unless specified. That is, the phases between any two clocks are derived from their individual clock definitions; timing paths between such clock domains are analyzed using these derived requirements. To avoid this redundant timing analysis and reporting of timing failure, asynchronous clocks need to be specified. When there are such asynchronous inter-clock paths specified, the design must use appropriate synchronization techniques to capture data reliably at the target clock domain.
  - With a complete clock definition as above, all FPGA internal paths (single-cycle paths) can be analyzed for timing. The feasibility of closing timing for the single-cycle internal paths can be assessed with reasonable confidence.
- Optimize the entire chip.
- Fine tune as necessary.

2-1. Open the synthesized design.

If you do not recall how to perform this task, refer to the "Opening the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

2-2. Determine if there are any unconstrained clocks in the design.

2-2-1. Select **Report Clock Networks** from the Flow Navigator under **Synthesis** > **Open Synthesized Design**.

2-2-2. Click **OK** to use the default settings.

Figure 6-5: Report Clock Networks (KCU105)

Note that there is an unconstrained primary clock in the design.

After having found that there is an unconstrained primary clock in the design, you now need to constrain the *clk_pin_p* primary clock to a specific clock frequency.

2-3. Constrain the primary clock to 125 MHz.

2-3-1. Select the **Hierarchy** tab in the Sources window.

2-3-2. Expand **Constraints** > **constrs_1** to see the *uart_led_Baselining.xdc* file.

2-3-3. Double-click the **uart_led_Baselining.xdc** file to open it in the text editor.

2-3-4. Uncomment the line beginning with "create_clock".

With this command, you will be constraining the clk_pin_p primary clock to the clock frequency of 125MHz.

2-3-5. Select **File** > **Text Editor** > **Save File** to save the changes to the file.

2-3-6. Click **Reload** in the status bar to reload the synthesized design with the new constraints applied.

2-4. Verify that *clk_pin_p* is constrained and reported in the Clock Networks report.

2-4-1. Enter the following command in the Tcl Console to report the design's clock networks in the GUI:

```
report_clock_networks -name {network_1}
```

Alternatively, you can also use the `report_clocks` Tcl command to verify.

2-4-2. Confirm that the clock frequency is defined for *clk_pin_p*.

2-5. After making sure that the primary clock is constrained, you will need to identify and specify the asynchronous clock interactions, if any.

For a design with multiple clocks, all the clock domain crossings (CDC) need to be specified for correct timing. This is particularly important when the inter-clock path is asynchronous as it needs to be specified to avoid unnecessary timing analysis.

The *uart_led* design for this lab has only one clock domain and is not a candidate for clock interaction analysis. For completeness, run clock interaction analysis anyway.

2-5-1. Select **Synthesis** > **Open Synthesized Design** > **Report Clock Interaction** from the Flow Navigator.

2-5-2. Click **OK** to use the default settings.

clk_pin_p is the only source and destination clock. All the paths sourced from *clk_pin_p* are timed (constrained) as indicated by the green color.

2-5-3. Close the Clock Interaction report.

2-6. Verify the constraint coverage in the design using the report generated by the `check_timing` command.

2-6-1. Enter the following command in the Tcl Console:

```
check_timing -help
```

Some of the supported timing checks are the following:

- `constant_clock`: Checks for clock signals connected to a constant signal(gnd/vss/data).
- `generated_clocks`: Number of missing generated clock definitions.
- `latch_loops`: Number of combinational latch loops in the design
- `loops`: Number of timing loops found in the design.
- `multiple_clock`: Number of clock pins reached by more than one timing clock.
- `no_clock`: Number of clock pins reached by a zero timing clock.

- `no_input_delay`: Number of input ports without at least one input delay constraint.
- `no_output_delay`: Number of output ports without at least one output delay constraint.
- `partial_input_delay`: Number of input ports with partially defined input delay constraints.
- `partial_output_delay`: Number of output ports with partially defined output delay constraints.
- `unconstrained_internal_endpoints`: Number of path end points without a timing requirement.
- `unexpandable_clocks`: Number of clock sets in which the period is not expandable with respect to each other

The `check_timing` command, as with all the reporting commands, has an option to write to a file with `-file <arg>`. Because you will be asked to generate reports during this lab, consider writing the results to a file and looking at what is produced.

2-6-2. Enter the following Tcl command in the Tcl Console to verify the timing constraint coverage on the design:

```
check_timing -verbose
```

You will see a summary of any missing timing constraints according to the timing engine. To view the report, you should expand the Tcl Console.

2-6-3. Confirm that there are no violations reported under no-clocks.

If there are any clocks nets available in the design without a clock definition (`created_clock` or `generated_clock`), it is reported here.

Notice that no I/O ports have been constrained with set_input_delay and set_output_delay constraints. You will constrain them in second stage of baselining procedure.

```
check_timing report

Table of Contents
-----------------
1. checking no_clock
2. checking constant_clock
3. checking pulse_width_clock
4. checking unconstrained_internal_endpoints
5. checking no_input_delay
6. checking no_output_delay
7. checking multiple_clock
8. checking generated_clocks
9. checking loops
10. checking partial_input_delay
11. checking partial_output_delay
12. checking latch_loops

1. checking no_clock
--------------------
 There are 0 register/latch pins with no clock.


2. checking constant_clock
--------------------------
 There are 0 register/latch pins with constant_clock.


3. checking pulse_width_clock
-----------------------------
 There are 0 register/latch pins which need pulse_width check


4. checking unconstrained_internal_endpoints
--------------------------------------------
 There are 0 pins that are not constrained for maximum delay.

 There are 0 pins that are not constrained for maximum delay due to constant clock.


5. checking no_input_delay
--------------------------
 There are 3 input ports with no input delay specified. (HIGH)

btn_pin
rst_pin
rxd_pin

 There are 0 input ports with no input delay but user has a false path constraint.
```

Figure 6-6: Check Timing

After constraining the input clock with a frequency of 125 MHz, you will use the Timing Summary report to see if the design meets timing at the synthesized design stage.

2-7. Generate the timing summary report on the synthesized design.

Question 1

What do you think each of the `report_timing_summary` command options do?

After the report has been generated, a new tab will be created at the bottom of the Vivado IDE called **Timing**. This tab makes it easy to view the timing summary information as well as find your design's timing-critical paths.

The Timing Summary window (in this case, it is labeled **Timing Summary - timing_1**) shows basic timing summary information. This includes worst negative slack for setup and component switching times.

This report also categorizes paths as either intra-clock (datapaths between synchronous elements clocked by the same clock) or inter-clock (between different clock domains). Each of these groups of datapaths are grouped by clock domain and each shows the worst-case setup and hold datapaths.

2-7-1. Expand **Intra-Clock Paths** > **clk_pin_p** in the Timing Summary window.

2-7-2. Select **Hold** to view the ten longest paths for the clock domain.

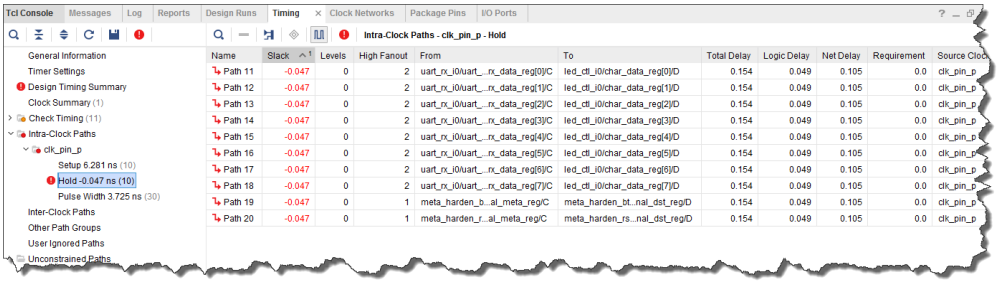


Figure 6-7: Longest Path (KCU105)

Note that the results will be different depending upon the target board for Verilog and VHDL designs. The above results correspond to the Verilog design targeted to the KCU105 board.

Note: The timing numbers may vary depending on the version of the Vivado Design Suite and the OS.

2-7-3. [Optional] Review the input and output ports not constrained by expanding the Unconstrained Paths section.

For outputs, expand the clk_pin_p to the None section. For inputs, expand the None to clk_pin_p section inside Unconstrained Paths.

2-7-4. Close the Timing Summary window.

2-8. Close the synthesized design.

If you do not recall how to perform this task, refer to the "Closing the Synthesized Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Performing a Baseline Implementation and Review the Timing Report Step 3

Here you will perform a baseline implementation of the design and review the Timing Summary report and analyze the report for critical paths.

3-1. Run implementation.

If you do not recall how to perform this task, refer to the "Running Implementation" section in the Vivado Design Suite Operations chapter of the Lab Reference Guide.

3-1-1. Click **Yes** in the dialog box to run synthesis before running implementation and click **OK** to open the runs dialog box.

The Timing Summary report is generated by default when the implemented design is opened.

3-1-2. Select **Open Implemented Design** in the Implementation Completed dialog box and click **OK**.

3-2. View the Timing Summary report.

3-2-1. View the Timing Summary report that was generated when the design was implemented.

Notice that the current design shows no timing violations.

Question 2

What difference did you see with the timing report performed on the implemented design compared to that of the netlist design?

3-3. Analyze the design’s timing-critical paths with the help of the options linked to delay path reporting.

The Timing Results window allows you to quickly display the longest (most critical) timing paths for each clock domain.

3-3-1. Expand **Intra-Clock Paths** > **clk_pin_p** in the Timing Summary window.

3-3-2. Select **Setup** to view the longest paths for the clock domain.

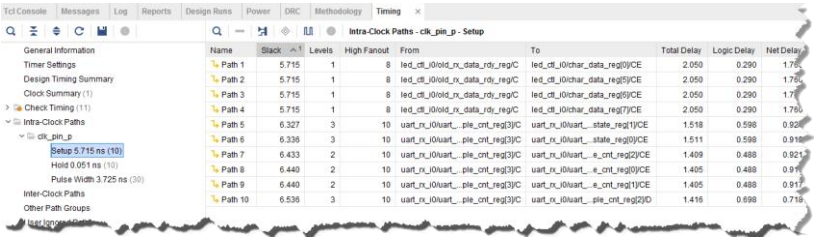


Figure 6-8: Longest Setup Paths for the clk_pin_p Clock Domain (KCU105)

Note that the results will be different for Verilog and VHDL designs. The above results correspond to the Verilog design and your timing values may vary slightly.

Note: The timing numbers may vary depending on the version of the Vivado Design Suite and the OS.

3-3-3. Select the longest path, right-click, and review some of the options that are available.

Note that as each path is selected, it automatically becomes highlighted in the Device view.

- **View Path Report:** View a timing report for the entire delay path selected.
- **Select Leaf Cells:** Find all of these specific components in the Netlist view.
- **Select Leaf Cell Parents:** Group all of the higher level components that contain this logic. This is useful for evaluating HDL coding style and grouping all the parent components into the same Pblock.
- **Schematic:** Generates a schematic of the datapath and the components that make up the path. This allows you to quickly traverse the design to determine how this critical path was made and even open the source code for each component along the datapath.

3-3-4. Double-click the longest path to view its path report.

3-3-5. Maximize or float the Path Properties window to look at the path details.

The Timing Path Properties report provides a detailed summary of the timing path covered by the specific clock path group. When you click the links in the report, the logic objects are selected in other views.

The Timing Path report provides detailed information of the logic objects in the path and their associated delays for the source clock path, datapath, and the destination clock path. The details of the timing path report are as follows:

- **Path Summary:** Provides a brief summary about the timing path and reports slack for the timing path endpoints. The slack is the difference between the data required time and the data arrival timing at the path endpoint.
- **Source Clock Path:** Provides detailed information on the logic objects in the path and their associated delays for the source clock path. This source clock path is the path followed by the source clock from its source point to the clock pin of the launching flip-flop.
- **Data Path:** Provides detailed information on the logic objects in the path and their associated delays for the internal circuitry between the launching and capturing flip-flops. The active clock pin of the launching flip-flop is called the path start point. The data input pin of the capturing flip-flop is called the path endpoint.
- **Destination Clock Path:** Provides detailed information on the logic objects in the path and their associated delays for the destination clock path. The destination clock path is the path followed by the destination clock from its source point (typically an input port) to the clock pin of the capturing flip-flop.

From the above analysis, it is clear that the internal timing paths inside the FPGA (i.e., timing paths between sequential elements) meet the timing comfortably and have plenty of slack.

Applying Timing Closure Methodology (Optimizing Entire FPGA and Fine Tuning) Step 4

In the first step for timing closure (baselining), timing constraints to cover the FPGA internal paths were applied. Using these constraints, timing analysis was performed at the synthesis netlist level to assess the feasibility of post-implementation timing closure. Based on the findings, implementation was run in the last step.

The next steps towards complete timing closure are:

- Checking that the internal timings are met as assessed during the baselining stage.
- Adding I/O delays to specify timing for the FPGA at the system level; also running timing analysis to check that I/O timings are met.
- Adding timing exceptions (multi-cycle paths, false paths, and path-specific max delays); also running timing analysis to confirm complete timing closure.

4-1. Check that the internal timings are met as assessed during the baselining stage.

4-1-1. Verify that the Design Timing Summary in the Timing Summary report confirms that all internal timing is now met.

Though there were failing hold paths during baselining in the synthesis stage, they have been improved to closure during implementation.

You will now specify the input delay requirements on *rst_pin*, *rxd_pin*, and *btn_pin* with respect to the primary clock *clk_pin_p* in the Input Delays section of the Constraints Wizard. This enables you to specify timing for the FPGA at a system level.

4-2. Add I/O delays to specify timing for the FPGA at the system level.

4-2-1. Double-click the **uart_led_Baselining.xdc** file under the constrs_1 constraint set in the Sources window to open it for editing.

4-2-2. Uncomment the lines (6-9) for set_input_delay and set_output_delay to enable the I/O timing specifications.

This is a simple design and only a few I/O constraints are required.

4-2-3. Select **File** > **Text Editor** > **Save File** to save the modified XDC file.

Synthesis and implementation status will become out-of-date because the XDC changes are not yet implemented.

4-3. Re-synthesize and re-implement.

4-3-1. Select **Implementation** > **Run Implementation** from the Flow Navigator.

4-3-2. Click **Yes** when prompted, since it is okay to launch synthesis first.

4-3-3. Select **Open Implemented Design** in the Implementation Completed dialog box.

Note: These new constraints could also have been added interactively through the Tcl Console, or with the Constraints Editor by updating the design database directly. After entering new constraints this way, you could have performed timing analysis with the original implementation netlist (without re-implementing).

This is a very effective way to explore a design for timing while avoiding potentially long run times. However, new constraints may benefit from the timing driven nature of re-implementation, especially for fixing hold violations (as chosen in this lab).

4-4. Verify timing for the entire FPGA.

4-4-1. View the Timing Summary report that was generated when the implemented design was opened.

As seen from the report, all timings have been met with the newly added I/O timing constraints.

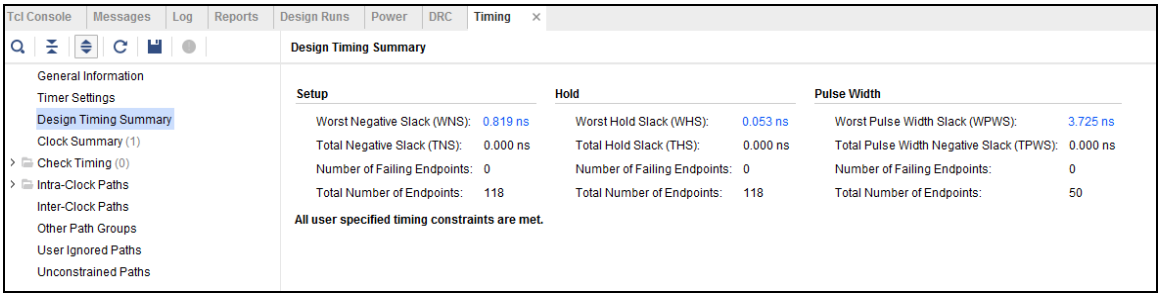


Figure 6-9: Timing Summary Report (KCU105)

Note: The timings of the paths may vary slightly.

4-5. Verify the constraint coverage in the design by using the report generated by the `check_timing` command.

4-5-1. Enter the following Tcl command in the Tcl Console to verify the timing constraint coverage on the design:

```
check_timing -verbose
```

There should be no warnings flagged in any of the categories. In particular there should be no missing input or output delays, confirming that the newly added I/O constraints cover all I/Os.

4-6. Add timing exceptions and fine tune the design.

At this stage, the FPGA is fully specified for timing. All paths are being constrained for the default timing relationship:

Intra-clock path Setup: Single cycle.

Intra-clock path Hold: Zero cycle (same edge).

Inter-clock path Setup: Worst-case launch to capture timing.

Inter-clock path Hold: Worst-case capture to launch timing.

Between Asynchronous clocks: Ignored for timing.

When the design contains paths with edge relationships different from the default, timing exceptions need to be specified. These include multi-cycle paths, false paths, and path-specific delays.

In this *uart_led* design, there are no such paths that need timing exceptions. This step is provided here for completeness.

4-7. Close the implemented design.

4-8. Close the Vivado Design Suite Project.

4-9. Close the Vivado Design Suite.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/Baselining` directory.

4-10. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

4-10-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/Baselining`.

4-10-2. Select **`Baselining`**.

4-10-3. Press <**Delete**>.

-- OR --

Using the command line:

4-10-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

4-10-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/Baselining`

[Linux users]: `rm -rf $TRAINING_PATH/Baselining`

Summary

In this lab, you learned how the baselining procedure in the Vivado Design Suite assists in gaining timing closure progressively.

Answers

1. What do you think each of the `report_timing_summary` command options do?

   The command to generate timing summary report is

   ```
   report_timing_summary -delay_type min_max -report_unconstrained
   -check_timing_verbose -max_paths 10 -input_pins -name timing_1
   ```

   Each of the options in this command have different usage.

   - `-delay_type`: Report both setup and hold time paths for each clock domain reported.
   - `-report_unconstrained`: Report timing on unconstrained paths.
   - `-check_timing_verbose`: Temporarily override any message limits and return all messages from this command.
   - `-max_paths 10`: The maximum number of paths to report per clock or per path group.
   - `-input_pins`: Show input pins in the timing path report.
   - `-name timing_1`: Specify a name for the results shown in the Vivado IDE.

2. What difference did you see with the timing report performed on the implemented design compared to that of the netlist design?

   The timing report during the netlist design has provided you timing path delays with the estimated net delays. The timing report in the implemented design phase has provided you timing path delays with the actual net delays. This is why you are seeing a difference between the timing values.

   The default timing report in the implementation shows the worst negative slack for setup, hold, and component switching times.

Lab 7: Increasing Design Performance Using Report QoR

2022.2

Abstract

This lab introduces the Report QoR Assessment and Report QoR Suggestions reports and their use in timing analysis. This lab also discusses how the `report_qor_suggestions` command can generate an implementation design strategy that is predicted to be optimal for the design using machine learning algorithms.

This lab should take approximately 60 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Analyze the QoR assessment report
- Analyze the QoR suggestions report
- Accumulate suggestions in a `report_qor_suggestions` (RQS) file by generating suggestions at different implementation stages or on different runs
- Add an RQS file to synthesis and implementation runs
- Generate ML strategy suggestions
- Set up an implementation run to use ML strategy suggestions

Introduction

When you are unsure as to whether the timing goals of your design can be met or not, the concept of quality of results (QoR) is important. The QoR of a design can be improved by making changes to the design or the constraints.

The `report_qor_suggestions` and `report_qor_assessment` commands are combined when selecting **Reports** > **Report QoR Suggestion/Assessment** in the Vivado® IDE. When running these commands outside of the Vivado IDE, each command is issued separately.

The `report_qor_suggestions` (RQS) command enables the Vivado Design Suite tools to analyze a design and provide automated solutions for enhancing QoR. The command can be run on an open design after synthesis or after any stage in the implementation flow.

The `report_qor_assessment` (RQA) command assesses the likelihood that a design will meet its timing requirements and provides a quick summary of design metrics.

The RQS command evaluates the design in five key areas and suggests fixes or improvements in these areas. The five areas are utilization, clocking, constraints, congestion, and timing.

Recommendations from RQS can take the following forms:

- RQS objects that can add:
  - Switches to a given command
  - Properties to a given design object
  - Implementation strategies customized for the design using machine learning algorithms
- Text recommendations that require user intervention

The `report_qor_suggestions` command can generate an implementation design strategy that is predicted to be optimal for the design using machine learning algorithms.

Understanding the Design

This lab uses an architected design to demonstrate some of the features of RQS. Suggestions are triggered by the design of the RTL and the placement of blocks using floorplanning.

The design contains the following modules:

Clocking Module

The main clocking circuit for the design resides in *clocking_module.vhd*. For simplicity, RST is tied to GND. LOCKED is registered and tied to an output port.

The structure of this block is shown in the following figure.

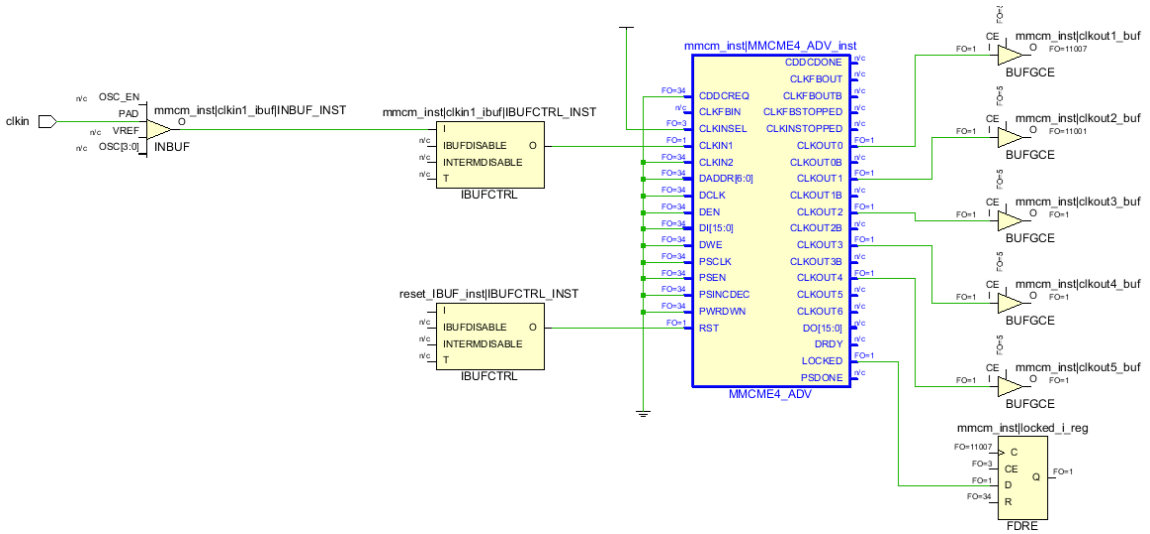


Figure 7-1: Clocking Module Overview

Reg CLKA to CLKB Module

This module contains a synchronous CDC for a large bus. It registers input data using CLKA and then passes it to a register on the CLKB domain to be passed to the output.

Registering large buses on different, related clock domains can impact hold slack (WHS/THS) and setup slack (WNS/TNS).

Bit Expander and Bit Reducer Modules

These modules enable the expansion and contraction of internal data widths so that the design does not run out of I/Os.

The modules take an arbitrary data width and expand or contract it to or from a desired size. The contraction logic creates many logic levels.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

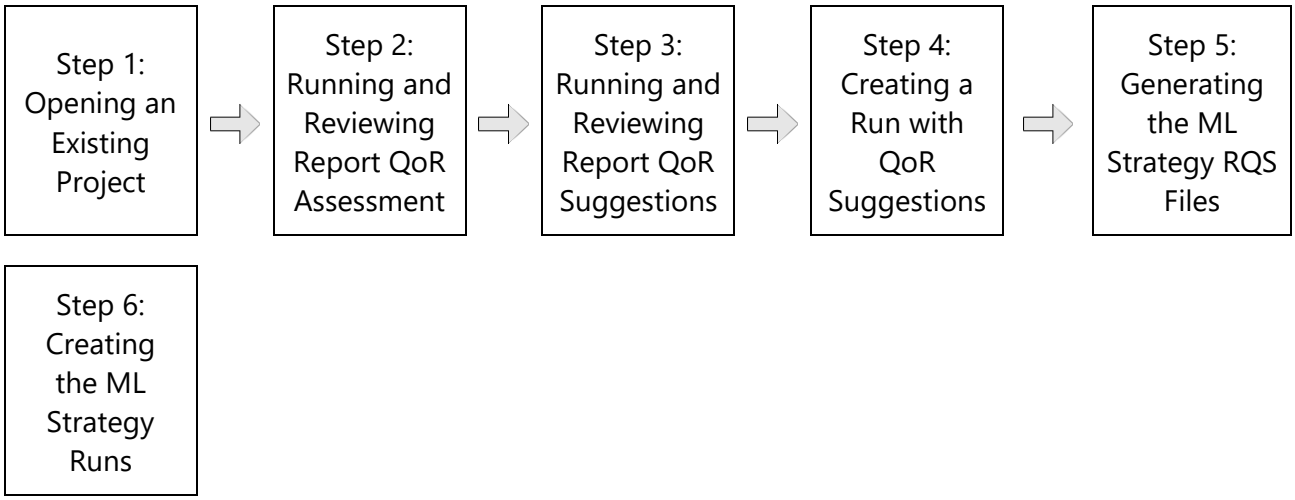
Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.

Symbol	Description	Example	Explanation
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Opening an Existing Project Step 1

In this step, you will open an existing project and open the synthesized design.

1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-2. Open the Vivado Design Suite project named `report_qor` located in the directory below.

1-2-1. Browse to the `$TRAINING_PATH/Report_QoR/lab/vhdl` directory and open the **`report_qor`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-3. Open the synthesized design.

1-3-1. Click **Open Synthesized Design** under Synthesis in the Flow Navigator.

The Vivado IDE opens the synthesized netlist and the target device into the Synthesized Design environment in which you can perform I/O pin planning, design analysis, and floorplanning.

1-3-2. Select the **Netlist** window in the Sources tab.

The Netlist window provides a hierarchical view of the elaborated or synthesized logic design, including the nets, logic primitives, and hierarchical modules of the design, starting with the currently defined top module.

1-3-3. Observe the hierarchy in the Netlist window.

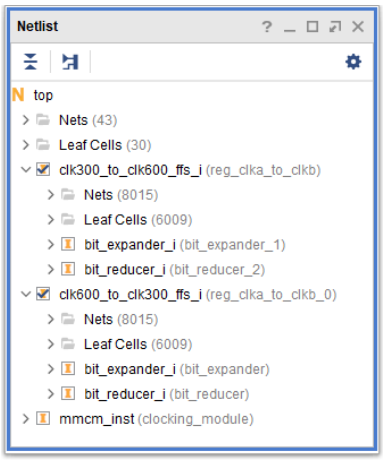


Figure 7-2: Netlist View

1-4. Observe the Pblock in the Device view.

1-4-1. Select the **Device** tab in the main window for the Device view.

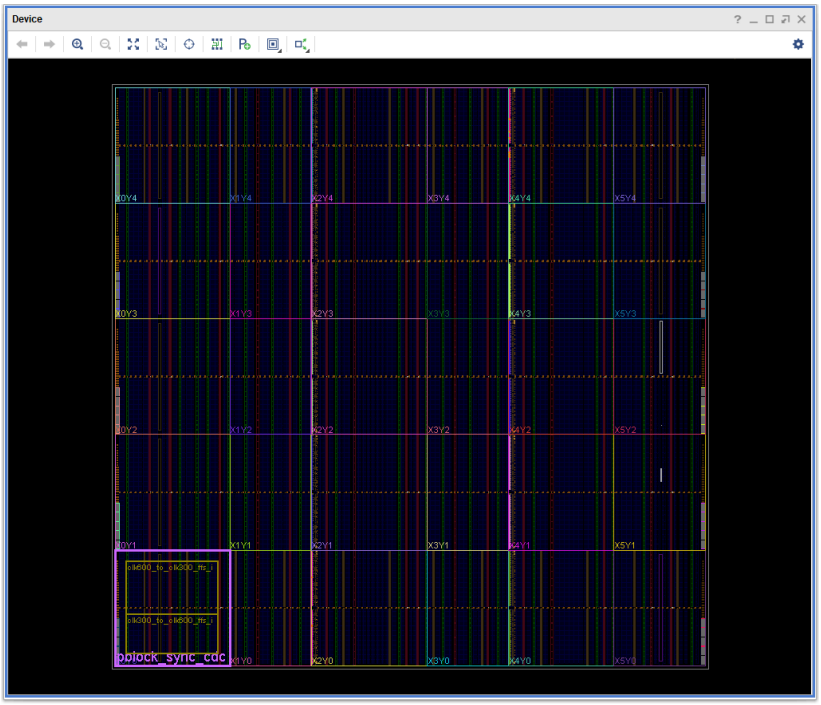


Figure 7-3: Pblock in the Device View

Note: The Pblock here has been added to control placement of the reg_clka_to_clkb modules and force a poor clock skew.

Running and Reviewing Report QoR Assessment — Step 2

You will now generate a QoR assessment report by using the `report_qor_assessment` command. This command can be run on an open design at any stage of the implementation flow after synthesis.

In project mode, this is typically after synthesis or implementation. In non-project mode, this can be after `synth_design`, `link_design`, `opt_design`, `place_design`, `phys_opt_design`, or `route_design`. It provides an assessment score that details the likelihood of the design closing timing.

This step also reviews different sections of the generated QoR assessment report.

To the left of the report window, you can navigate to the different sections of the report. On the right, more information is provided.

2-1. Generate the QoR assessment report.

2-1-1. Select **Reports** > **Report QoR Assessment**.

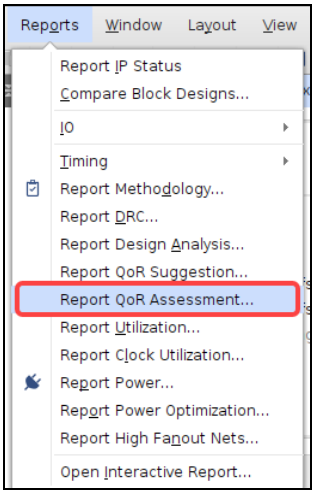


Figure 7-4: Selecting Report QoR Assessment

2-1-2. Select **Report passing metrics** in the Report QoR Assessment dialog box.

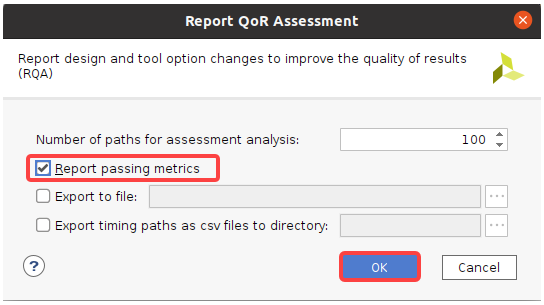


Figure 7-5: Report QoR Assessment Dialog Box

2-1-3. Click **OK**.

2-1-4. Observe that the generated QoR report opens in the QoR Assessments tab.

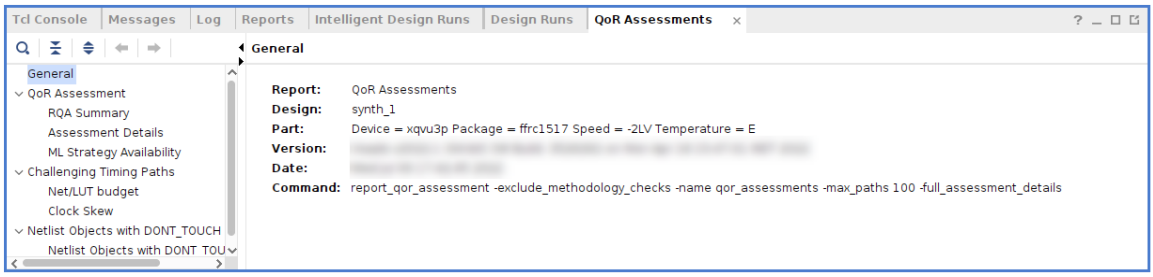


Figure 7-6: QoR Assessments Tab

Question 1

What is the equivalent Tcl command for generating the QoR assessment report using **Reports** > **Report QoR Assessment**?

Hint: Check the Tcl Console tab.

2-2. Review the QoR assessment report.

2-2-1. Click **RQA Summary** under the QoR Assessment section of the report.

2-2-2. Observe that the QoR assessment score is 2.

This means that the timing will most likely not meet.

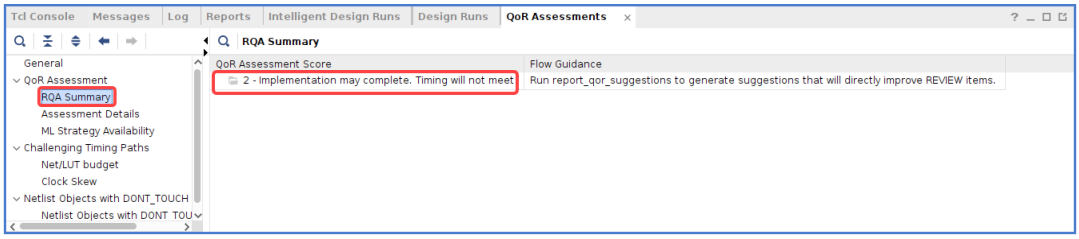


Figure 7-7: RQA Summary - QoR Assessments Tab

The flow guidance recommends referring to the QoR suggestions report, which may help with improving the design.

Note: The `report_qor_assessment` command runs using the latest data available to it. For example, clock skew is only accurate after logic is placed, so less emphasis is given to the clock skew score at the pre-place stage in the implementation flow.

Question 2

Describe what the QoR assessment score is. What is the range of the QoR assessment score?

2-2-3. Click the **Assessment Details** section of the report.

2-2-4. Observe that the detailed table shows the categories that have passed the assessment and individual failed metrics.

The score is an aggregate of the failed metrics.

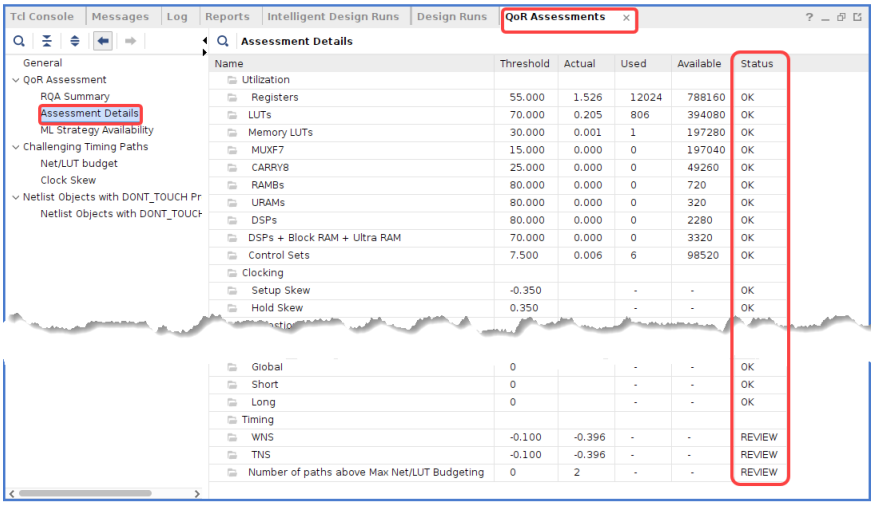


Figure 7-8: Assessment Details - QoR Assessments Tab

This section lists the items that have led to the assessment score of 2. Because the option to report passing metrics was selected, items marked as OK are shown.

2-2-5. Select **Net/LUT budget** under Challenging Timing Paths.

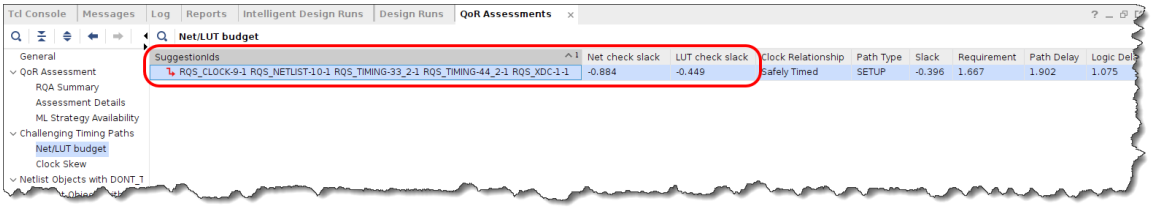


Figure 7-9: Net/LUT Budget - QoR Assessments Tab

2-2-6. Scroll across to see all the path characteristics reported.

The following items are of particular interest:

- **SuggestionIds:** These IDs correspond to suggestions that, if triggered, impact this path.
- **Net Check Slack:** This is the slack when the path is substituted with higher net delay values.
- **LUT Check Slack:** This is the slack when LUTs are substituted with higher LUT delay values.

Running and Reviewing Report QoR Suggestions Step 3

You will now generate a QoR suggestions report by using the `report_qor_suggestions` command. This command can be run on an open design at any stage of the implementation flow after synthesis.

In project mode, this is typically after synthesis or implementation. In non-project mode, this can be after `synth_design`, `link_design`, `opt_design`, `place_design`, `phys_opt_design`, or `route_design`.

This step also reviews different sections of the generated QoR suggestions report.

3-1. Generate the QoR suggestions report.

3-1-1. Select **Reports** > **Report QoR Suggestion**.

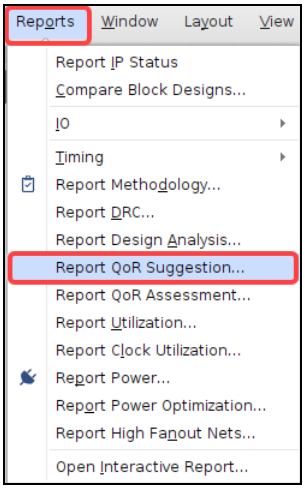


Figure 7-10: Selecting Report QoR Suggestions

3-1-2. Use the default settings and click **OK** in the Report QoR Suggestion dialog box.

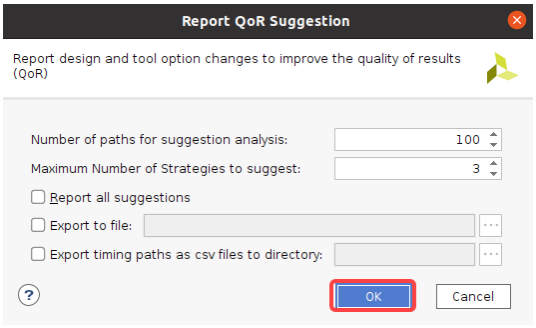


Figure 7-11: Report QoR Suggestion Dialog Box

3-1-3. Observe that the generated QoR report opens in the QoR Suggestions tab.

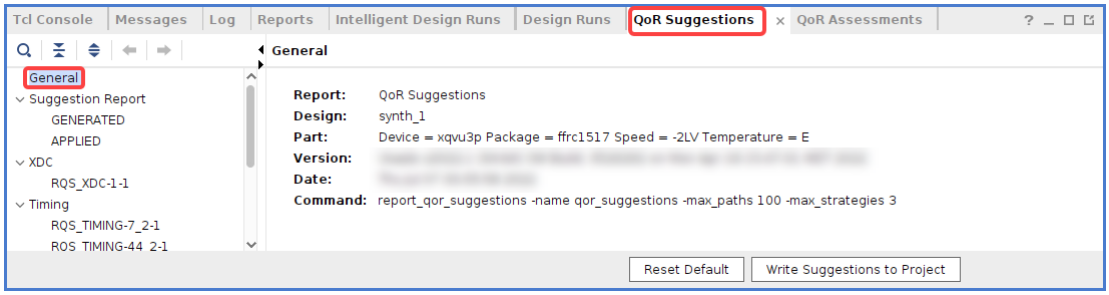


Figure 7-12: QoR Suggestions Tab

Question 3

What is the equivalent Tcl command for generating the QoR suggestions report using **Reports** > **Report QoR Suggestion**?

Hint: Check the Tcl Console tab.

3-2. Review the QoR suggestions report.

3-2-1. Under Suggestion Report, click **GENERATED**.

This brings up the report section as shown in the figure below.

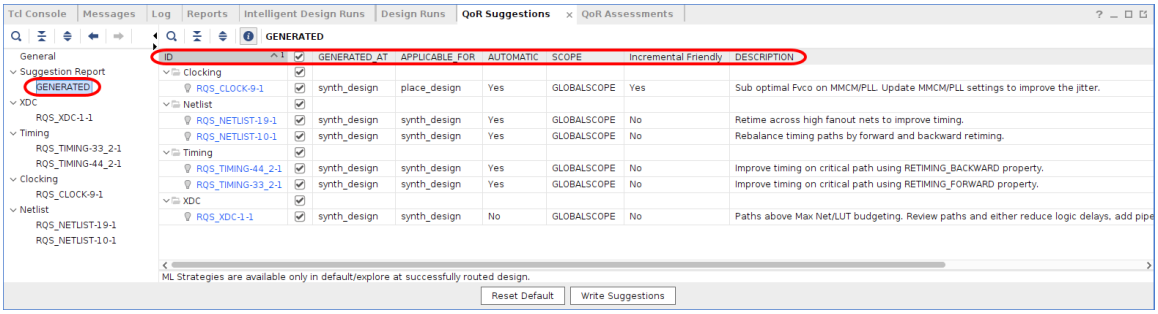


Figure 7-13: Suggestion Report - QoR Suggestions Tab

3-2-2. Observe that the GENERATED section provides a list of all the suggestions that have been generated at this stage of the current run.

Each suggestion has a description that details the reason for the suggestion as shown in the table below.

Item	Description	Comment
GENERATED_AT	This shows what stage of the design the suggestion was generated at. Typical values are `place_design` or `route_design`.	As you progress through the design stages, the decisions that the tool makes are based on the information available at the time. Information accuracy increases after placement and again after routing.
APPLICABLE_FOR	This stage must be rerun for the suggestion to take effect.	Most suggestions are executed at either `synth_design` or `place_design`.
AUTOMATIC	Indicates if the suggestion is executed automatically or if manual intervention is required.	Automatic suggestions either recommend a switch to the tool or a property to be added to a cell or net.
INCREMENTAL FRIENDLY	Indicates if the suggestion is optimized for the incremental flow.	Non-incremental friendly suggestions must be already present in the reference checkpoint. If you want to add non-incremental friendly suggestions, an updated reference checkpoint must be used.
SCOPE	Indicates the target synthesis run level. GLOBALSCOPE is top level. Otherwise, a sub-module is targeted..	Allows a single RQS file to be used on top-level and sub-module out of context (OOC) synthesis runs. Only applicable to synthesis suggestions.

Figure 7-14: RQS Suggestion Descriptions

3-2-3. Click the **RQS_XDC-1-1** hyperlink.

This takes you to the details section for this suggestion.

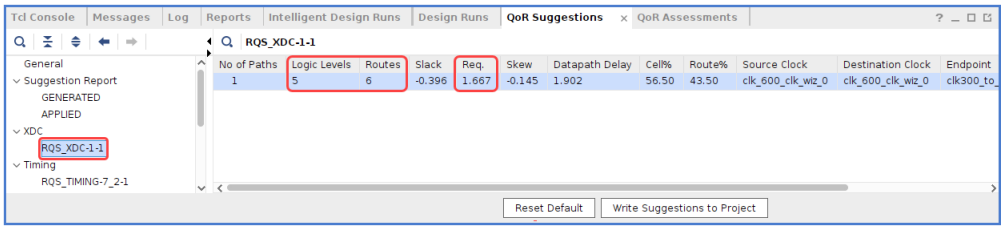


Figure 7-15: XDC Suggestion - QoR Suggestions Tab

Here, the suggestion description says that the timing constraint is too tight for the given path. The path has a large negative slack, which would stand out in a timing report. Close analysis shows that this is a 600 MHz path with high logic levels. This path needs to be fixed.

3-2-4. Click the back arrow icon ( ) to go back to the GENERATED view.

3-2-5. Click the **RQS_TIMING-33_2-1** row in the GENERATED view.

You can see that this is an automatic suggestion that is applicable for *synth_design* in the APPLICABLE_FOR column. This indicates that you must rerun the `synth_design` command to make use of this suggestion.

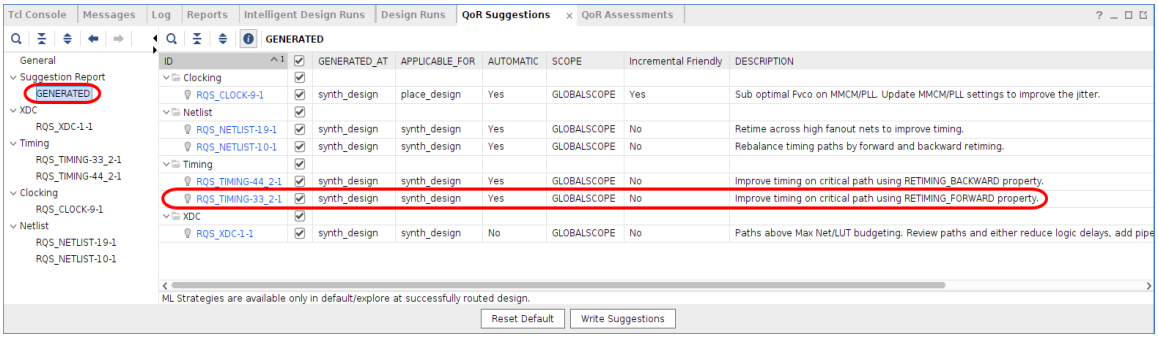


Figure 7-16: RQS_timing-33_2-1 - QoR Suggestions Tab

3-2-6. Observe that on selecting the row in the Suggestion Report > GENERATED view, the suggestion object is selected, and the QoR Suggestion Properties window appears.

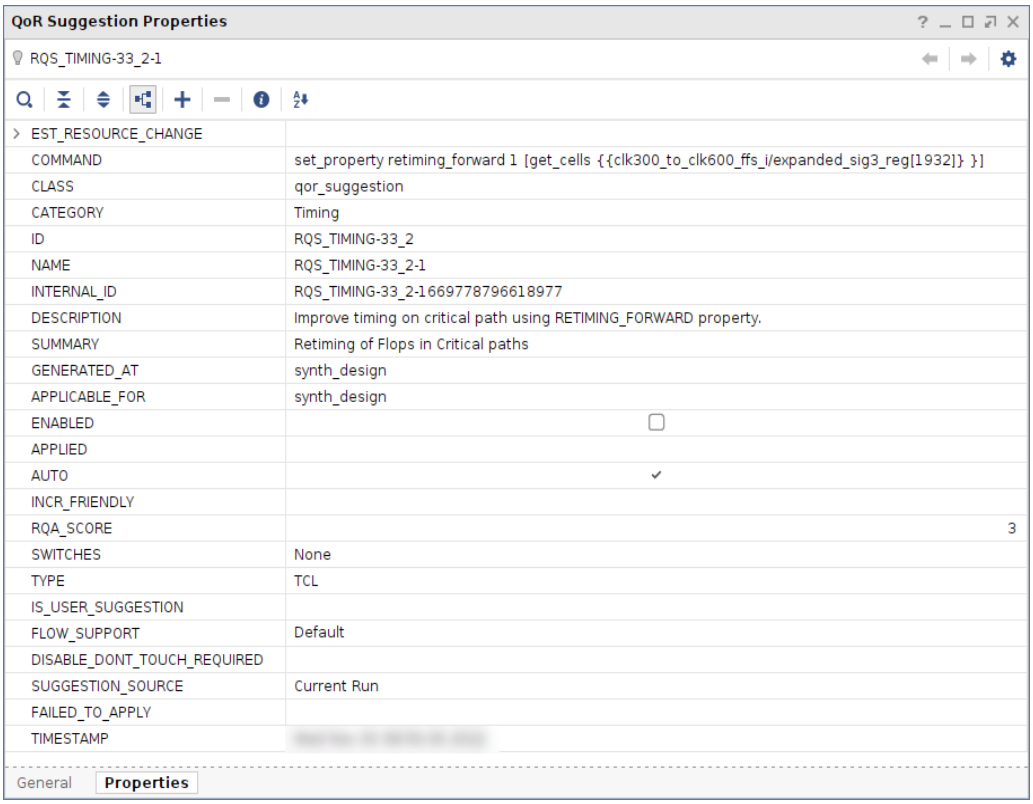


Figure 7-17: QoR Suggestion Properties Window

In the QoR Suggestions window, you can see the remaining suggestions. The RQS_CLOCK-9 suggestion is applicable for `place_design`.

3-2-7. Deselect all the suggestions and select only the following suggestions:

- **RQS_CLOCK-9**
- **RQS_NETLIST-19**
- **RQS_TIMING-44**
- **RQS_TIMING-33**

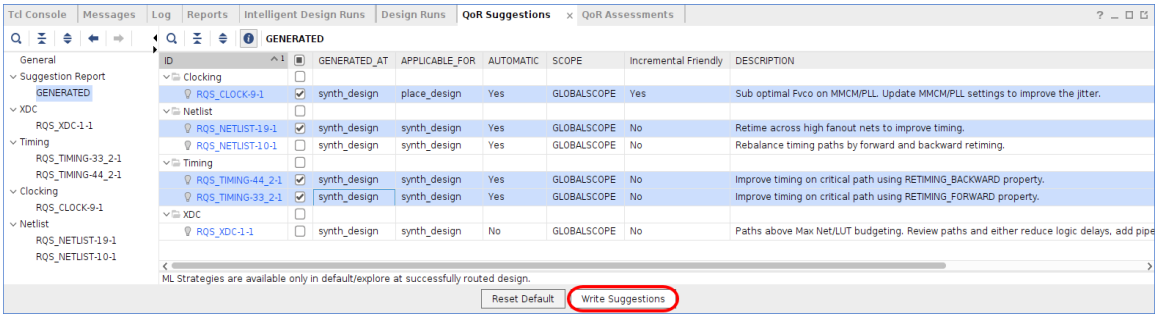


Figure 7-18: Writing the Selected Suggestions

3-2-8. Click **Write Suggestions** to write these suggestions to the project.

These suggestions apply to the entire module and overlap the other suggestions. For clarity, at this stage, the suggestions are only written but are not applied. You will need to create a new run and add these suggestions to the run to apply them.

3-2-9. Click **Browse** and go to the following location in the Write Suggestion to Project dialog box:

```
$TRAINING_PATH/Report_QoR/lab/vhdl/report_qor.srcs
```

3-2-10. Click **Save**.

3-2-11. Change the file name to **rqs_report.rqs** and select **Copy sources to project**.

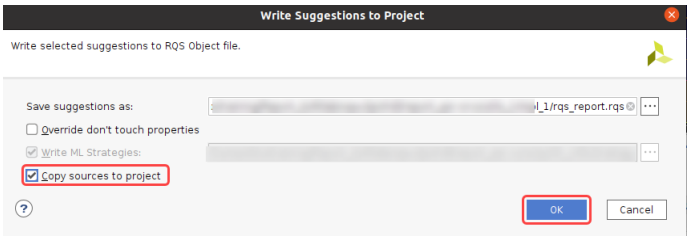


Figure 7-19: Write Suggestions to Project Dialog Box

3-2-12. Click **OK**.

3-2-13. Select the **Hierarchy** tab in the Sources view.

3-2-14. Observe that the RQS file has been added to the utils_1 fileset.

Creating a Run with QoR Suggestions — Step 4

You will now create a new run and add suggestions to the run and examine what happens when suggestions are applied and how they are reported.

4-1. Create a run with the QoR suggestions.

4-1-1. Go to the Design Runs window.

4-1-2. Right-click the **synth_1** synthesis run and select **Copy Run**.

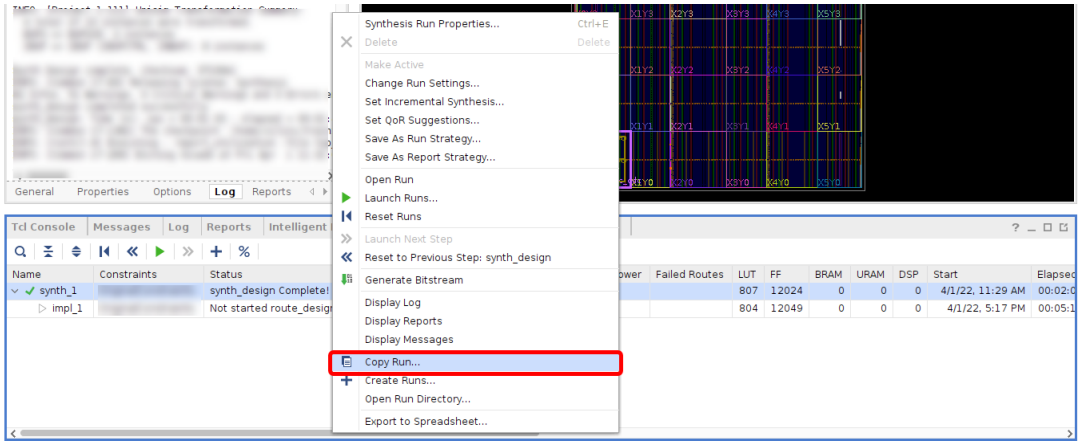


Figure 7-20: Selecting Copy Run

4-1-3. Click **OK** to use the default settings.

4-1-4. Right-click the **impl_1** implementation run and select **Copy Run**.

4-1-5. Select **synth_1_copy_1** from the Synth_Name column drop-down list.

Figure 7-21: Changing the Synthesis Run

4-1-6. Click **OK**.

4-1-7. Right-click the new **synth_1_copy_1** run and select **Set QoR Suggestions**.

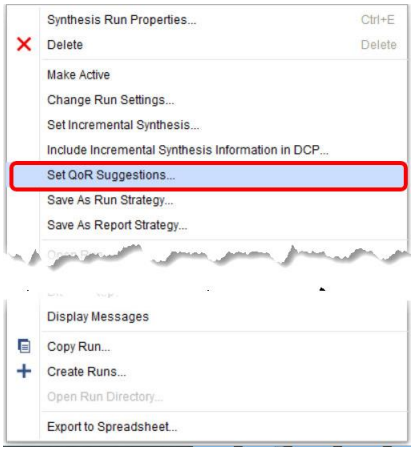


Figure 7-22: Selecting Set QoR Suggestions

4-1-8. Select **Enable suggestions** in the Set QoR Suggestions dialog box.

4-1-9. Specify **rqs_report.rqs** as the suggestion file, which will be added to the project.

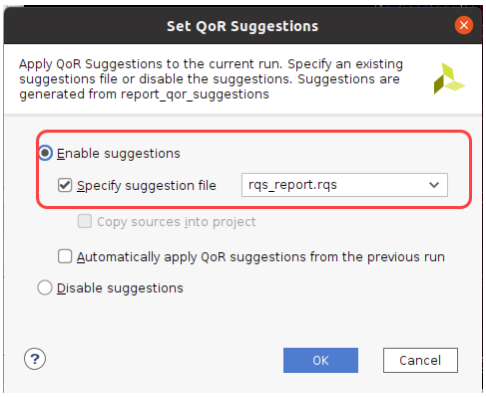


Figure 7-23: Enabling Suggestions and Selecting rqs_report.rqs

4-1-10. Click **OK**.

4-1-11. Repeat the previous steps for the implementation run and specify the same RQS file for the run in the Set QoR Suggestions dialog box.

4-1-12. Right-click **synth_1_copy_1** and select **Make Active**.

4-2. Run synthesis and generate the QoR assessment report on the synthesized design.

4-2-1. Click **Run Synthesis** under Synthesis in the Flow Navigator.

After the synthesis process completes, the Synthesis Completed dialog box opens. The dialog box prompts you to run implementation, open the synthesized design, or view reports.

4-2-2. Click **Open Synthesized Design** in the Synthesis Completed dialog box.

4-2-3. Select **Reports** > **Report QoR Assessment**.

4-2-4. Click **OK** to use the default settings.

4-2-5. Click **RQA Summary** and observe that the QoR assessment score has improved to 4 from 2.

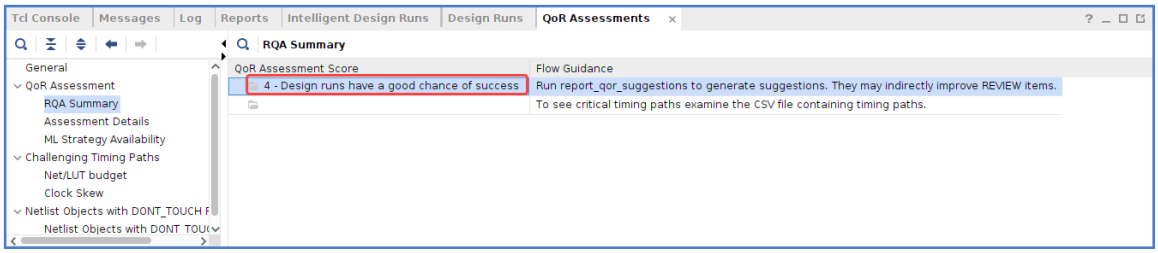


Figure 7-24: Improved QoR Assessment Score

4-2-6. Run the **Report QoR Suggestion** report.

4-2-7. Observe that there are now more sections under Suggestion Report.

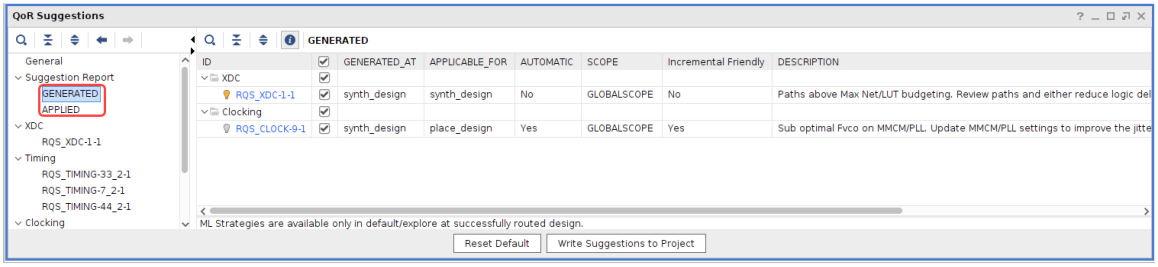


Figure 7-25: Options Under Suggestion Report - QoR Suggestions Tab

- GENERATED: New suggestions are listed in this section.
- EXISTING: Suggestions that existed previously but have not been applied are listed in this section (not shown).
- APPLIED: Suggestions that have been applied are listed in this section.
- FAILED TO APPLY: Suggestions that apply to design objects that no longer exist are listed in this section (not shown).

Note: For APPLIED suggestions, the timing path summary might still be available, but it is not possible to cross-probe to other views in the Vivado IDE because some items might have changed.

4-3. Close the project.

4-3-1. Select **File** > **Close Project** to close the project.

The Close Project dialog box opens.

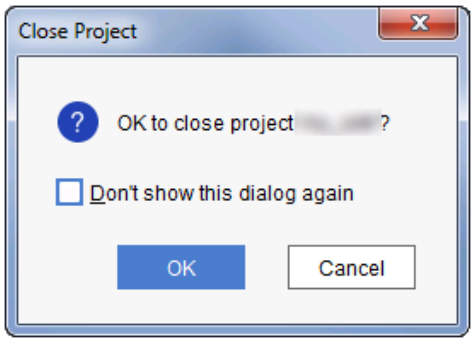


Figure 7-26: Close Project Dialog Box

4-3-2. Click **OK**.

Generating the ML Strategy RQS Files Step 5

The `report_qor_suggestions` command can generate an implementation design strategy that is predicted to be optimal for the design using machine learning algorithms.

The next two steps shows the process of:

- Generating ML strategy suggestions
- Setting up the implementation run to use the ML strategy suggestions

5-1. Open the provided implemented project in the Vivado IDE.

5-1-1. Click **Open Project** from the Welcome screen in the Vivado IDE.

5-1-2. Browse to the `$TRAINING_PATH/Report_QoR/lab/Implemented_design` directory and open the **`report_qor`** project.

5-2. Open the implemented design.

5-2-1. Click **Open Implemented Design** under Implementation in the Flow Navigator.

5-2-2. Ignore the critical warnings and click **OK**.

5-2-3. Select **Reports** > **Report QoR Assessment**.

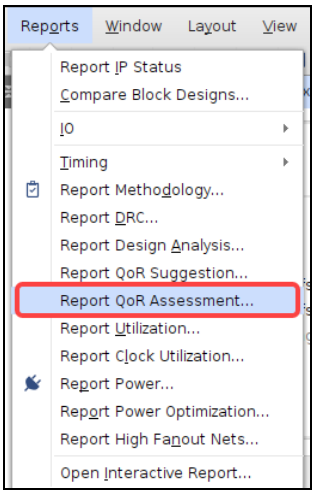


Figure 7-27: Selecting Report QoR Assessment

5-2-4. Use the default settings and click **OK** in the Report QoR Assessment dialog box.

5-3. Review the ML strategies on the design.

5-3-1. Click **ML Strategy Availability** in the QoR Suggestions tab.

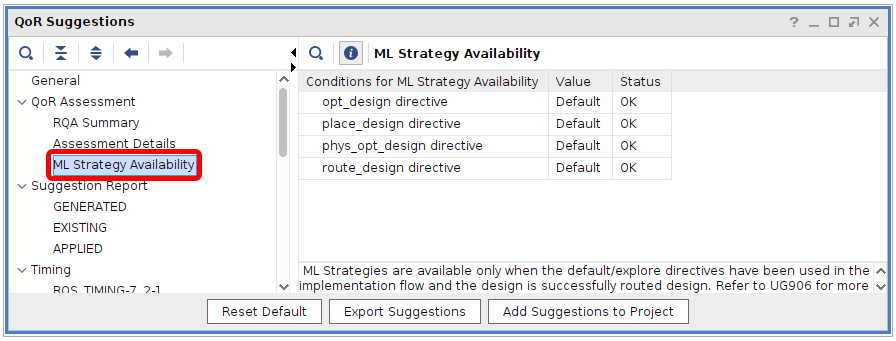


Figure 7-28: ML Strategy Availability - QoR Suggestions Tab

Requirements are as follows:

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

5-3-2. Make sure that the implemented design strategy is **Vivado Implementation Defaults** in the Design Runs window.

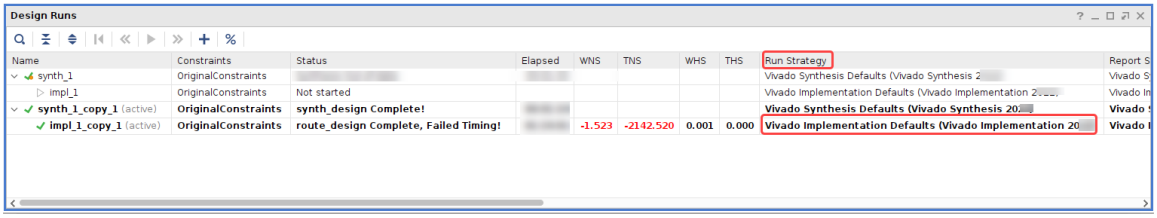


Figure 7-29: Run Strategy Defaults Settings - Design Runs Window

5-3-3. Select **Reports** > **Report QoR Suggestion**.

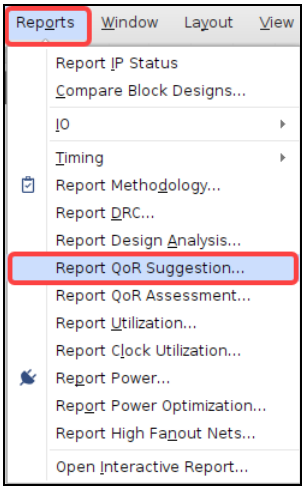


Figure 7-30: Selecting Report QoR Suggestions

5-3-4. Use the default settings and click **OK** in the Report QoR Suggestion dialog box.

5-3-5. Click **GENERATED** under Suggestion Report in the QoR Suggestions tab.

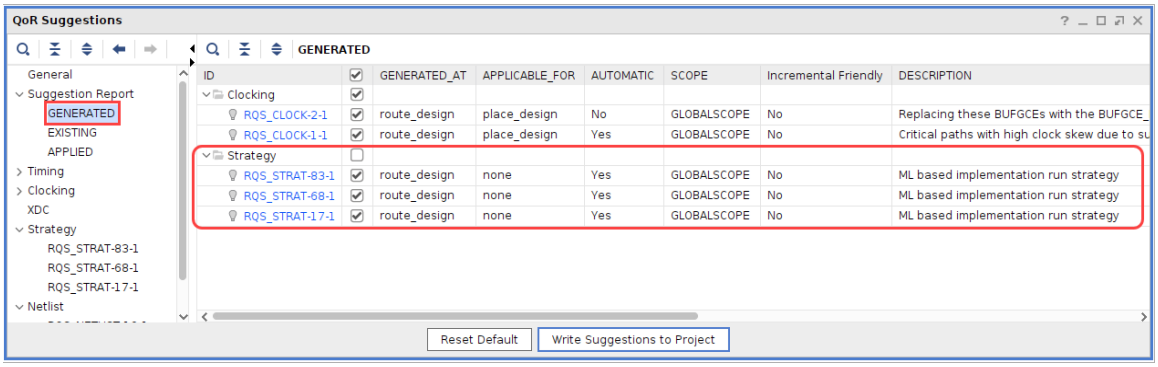


Figure 7-31: Generated Strategies - QoR Suggestions Tab

5-3-6. Click **RQS_STRAT-83-1** and observe the details of the strategy.

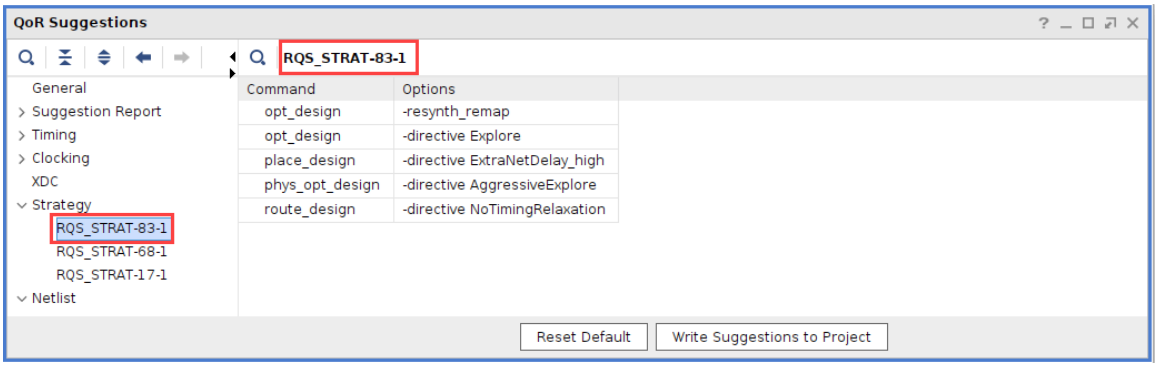


Figure 7-32: Strategy Suggestion Details

Each RQS file also contains all of the other suggestions that are not strategy suggestions.

5-4. Generate the ML strategies.

5-4-1. Right-click the **Impl_1_copy_1** run in the Design Runs window and select **Generate ML strategies**.

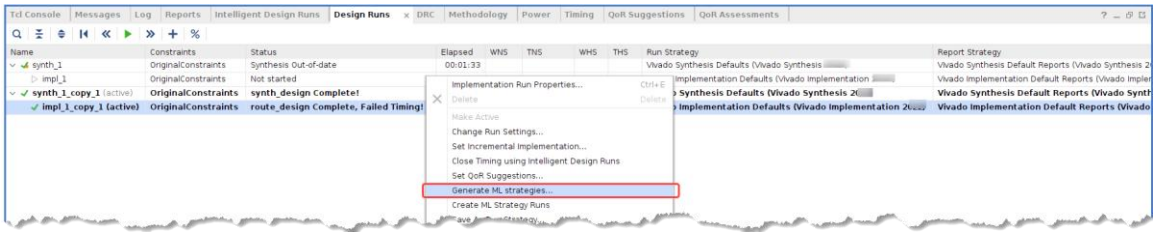


Figure 7-33: Generating the ML Strategies

5-4-2. Select **All GENERATED suggestions** in the Generate ML Strategies dialog box.

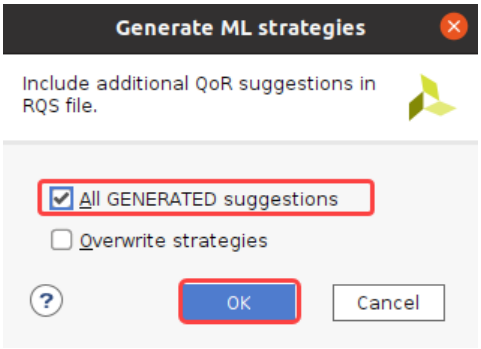


Figure 7-34: Generate ML Strategies Dialog Box

5-4-3. Click **OK**.

5-4-4. Right-click the **impl_1_copy_1** run and select **Open Run Directory**.

5-4-5. Observe the folder named *MLStrategy*.

5-4-6. Double-click the **MLStrategy** folder to open it.

You should see the following files:

- checkpoint_top_routedSuggestionFile1.rqs
- checkpoint_top_routedSuggestionFile2.rqs
- checkpoint_top_routedSuggestionFile3.rqs
- NonProject_MLStrategyCreateRun1.tcl
- NonProject_MLStrategyCreateRun2.tcl
- NonProject_MLStrategyCreateRun3.tcl

These RQS files are common for both project and non-project flows.

Creating the ML Strategy Runs — Step 6

You will now use the generated files from the previous step to create ML strategy project-based runs and observe the changes in the implementation run properties associated with the ML strategies.

6-1. Create the ML strategy runs.

6-1-1. Select the Vivado IDE.

6-1-2. Select the **impl_1_copy_1** run the Design Runs window.

6-1-3. Right-click and select **Create ML Strategy Runs**.

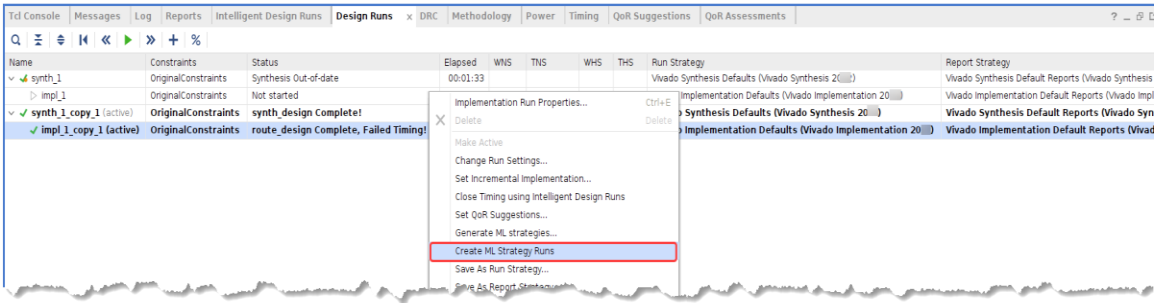


Figure 7-35: Creating ML Strategy Runs

6-1-4. Observe that the three strategies are added under the implementation run.

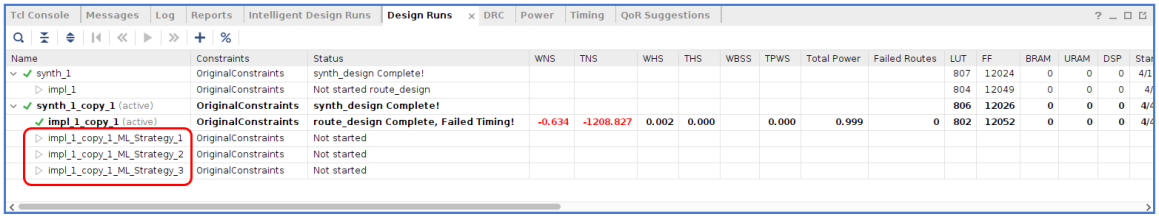


Figure 7-36: Sourcing the ML Strategies Under the Implementation Run

6-1-5. Select any ML strategy in the Design Runs window.

6-1-6. Select the **Options** tab in the Implementation Run Properties window.

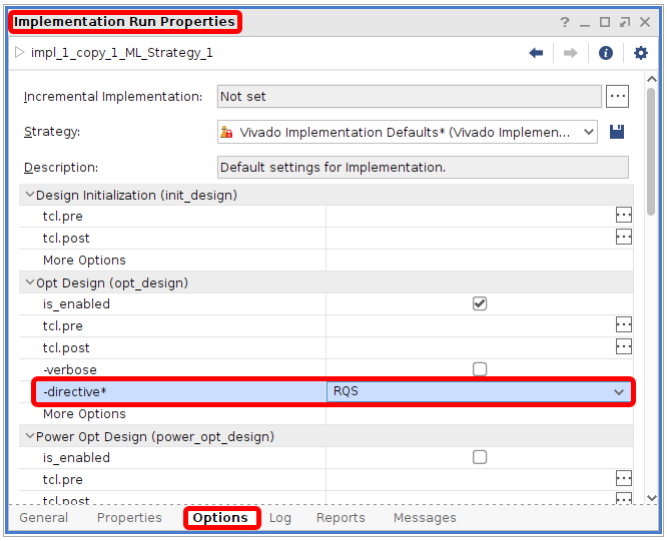


Figure 7-37: Implementation Run Properties

6-1-7. Observe that the directive is set to RQS for `opt_design`, `place_design`, `phys_opt_design`, and `route_design`.

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

These runs can proceed in parallel and complete like a standard run.

6-2. Close the implemented design.

6-3. Close the project.

6-3-1. Select **File** > **Close Project** to close the project.

The Close Project dialog box opens.

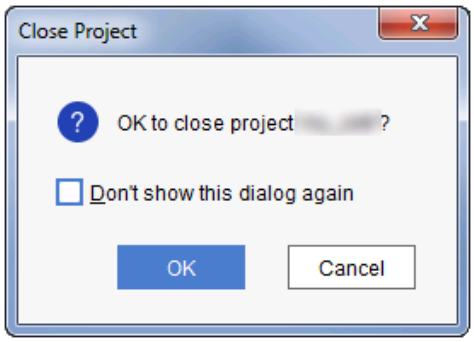


Figure 7-38: Close Project Dialog Box

6-3-2. Click **OK**.

6-4. Close the Vivado Design Suite.

6-4-1. Select **File** > **Exit**.

The Exit Vivado dialog box opens.

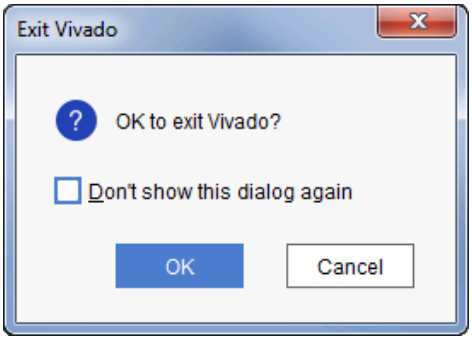


Figure 7-39: Exit Vivado Dialog Box

6-4-2. If you are asked to save the project or a portion of the project, select whichever elements of the project you want to save, then click **Save** to save the selected elements; otherwise, click **Don't Save**.

6-4-3. Click **OK** when you are asked to exit the Vivado Design Suite.

Note: You can choose to select the *Don't show this dialog again* option to avoid being asked for confirmation when exiting the Vivado Design Suite.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/Report_QoR` directory.

6-5. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

6-5-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/Report_QoR`.

6-5-2. Select **`Report_QoR`**.

6-5-3. Press <**Delete**>.

-- OR --

Using the command line:

6-5-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

6-5-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/Report_QoR`

[Linux users]: `rm -rf $TRAINING_PATH/Report_QoR`

Summary

In this lab, you learned how to:

- Use Report QoR Assessment to conduct an assessment before and after improvements were implemented—examining the improvement in the score.
- Use Report QoR Suggestions to conduct a complex analysis of the design.
- Generate and add an RQS file to a project implementation run.
- Use `report_qor_suggestions` to generate ML strategies.
- Create the RQS and Tcl files using `write_qor_suggestions`.
- Use the `MLStrategyCreateRun` Tcl script to set up ML strategy runs and confirm the key aspects of setting up ML strategy runs.

Answers

1. What is the equivalent Tcl command for generating the QoR assessment report using **Reports** > **Report QoR Assessment**?

   You will find the following equivalent Tcl command from the Tcl Console:
   ```
   report_qor_assessment -exclude_methodology_checks -name
   qor_assessments -max_paths 100 -full_assessment_details
   ```

   This command is in the Vivado IDE only. Outside of the Vivado IDE, you will need to run separate commands for `report_qor_assessment`.

   For example:

   ```
   report_qor_assessment -max_paths <N> ; # Tcl
   ```

   The above command uses the `-max_paths <N>` switch, where `N` is an integer.

2. Describe what the QoR assessment score is. What is the range of the QoR assessment score?

   The QoR assessment score is indicative of how likely the design is to meet performance targets generated by the QoR assessment report.

   This report analyzes multiple performance metrics and provides a design score in the range of 1–5, where:

   - 1 stands for "Design is unlikely to complete the implementation flow"
   - 2 stands for "Design will complete the implementation flow but is very unlikely to meet timing constraints"
   - 3 stands for "Design is unlikely to meet timing constraints"
   - 4 stands for "Design may meet timing"
   - 5 stands for "Design will easily meet the timing"

3. What is the equivalent Tcl command for generating the QoR suggestions report using **Reports** > **Report QoR Suggestion**?

   You will find the following equivalent Tcl command from the Tcl Console:
   ```
   report_qor_suggestions -name qor_suggestions -max_paths 100
   -max_strategies 3
   ```

   This command is in the Vivado IDE only. Outside of the Vivado IDE, you will need to run separate commands for `report_qor_suggestions`.

   For example:

   ```
   report_qor_suggestions -max_paths <N> ; # Tcl
   ```

   The above command uses the `-max_paths <N>` switch, where `N` is an integer.

Lab 8: Timing Closure Using Physical Optimization Techniques

2022.2

Abstract

This lab introduces physical optimization techniques for timing closure. You will learn to run *phys_opt* in different modes; that is, post_place and post_route. You will also review optimization messages to understand *phys_opt* techniques.

Note: In the latest release of the tool, the *post_place_phy_opt* feature is enabled by default. For learning purposes, *post_place_phy_opt* has been disabled.

This lab should take approximately 30 minutes.

CloudShare Users Only

You are provided with three attempts to access a lab, and the time allotted to complete each lab is twice the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Analyze a design without optimization enabled
- Create an implementation run with different strategies
- Implement a design with physical optimization enabled
- Describe the physical optimization technique used for timing closure
- Review timing reports to understand timing improvement

Introduction

Optimization in the Vivado® IDE happens at three levels: logic optimization, power optimization, and physical optimization.

Physical optimization performs timing-driven optimization on the negative-slack paths of a design. Physical optimization has two modes of operation: post-place and post-route.

In post-place mode, optimization occurs based on timing estimates based on cell placement. Physical optimization automatically incorporates netlist changes due to logic optimizations and places cells as needed.

In post-route mode, optimization occurs based on actual routing delays. In addition to automatically updating the netlist on logic changes and placing cells, physical optimization also automatically updates routing as needed.

The Vivado tools perform the physical optimizations on the in-memory design. Following table shows the available physical optimizations options.

Option Name	post-place		post-route	
	valid	default	valid	default
High-Fanout Optimization	Y	Y	N	n/a
Placement Optimization	Y	Y	Y	Y
Routing Optimization	N	n/a	Y	Y
Rewiring	Y	Y	Y	Y
Critical-Cell Optimization	Y	Y	Y	N
DSP Register Optimization	Y	Y	N	n/a
Block RAM Register Optimization	Y	Y	N	n/a
URAM Register Optimization	Y	N	N	n/a
Shift Register Optimization	Y	Y	N	n/a
Critical Pin Optimization	Y	Y	Y	Y
Block RAM Enable Optimization	Y	Y	N	n/a
Hold-Fixing	Y	N	Y	N
Negative-Edge FF Insertion	Y	N	N	n/a
Retiming	Y	N	Y	N
Forced Net Replication	Y	N	N	n/a
SLR-Crossing Optimization	Y	N	Y	Y
Clock Optimization	N	n/a	Y	Y

Figure 8-1: Post-Place and Post-Route Physical Optimizations

The wave_gen design used in this lab is a programmable waveform generator.

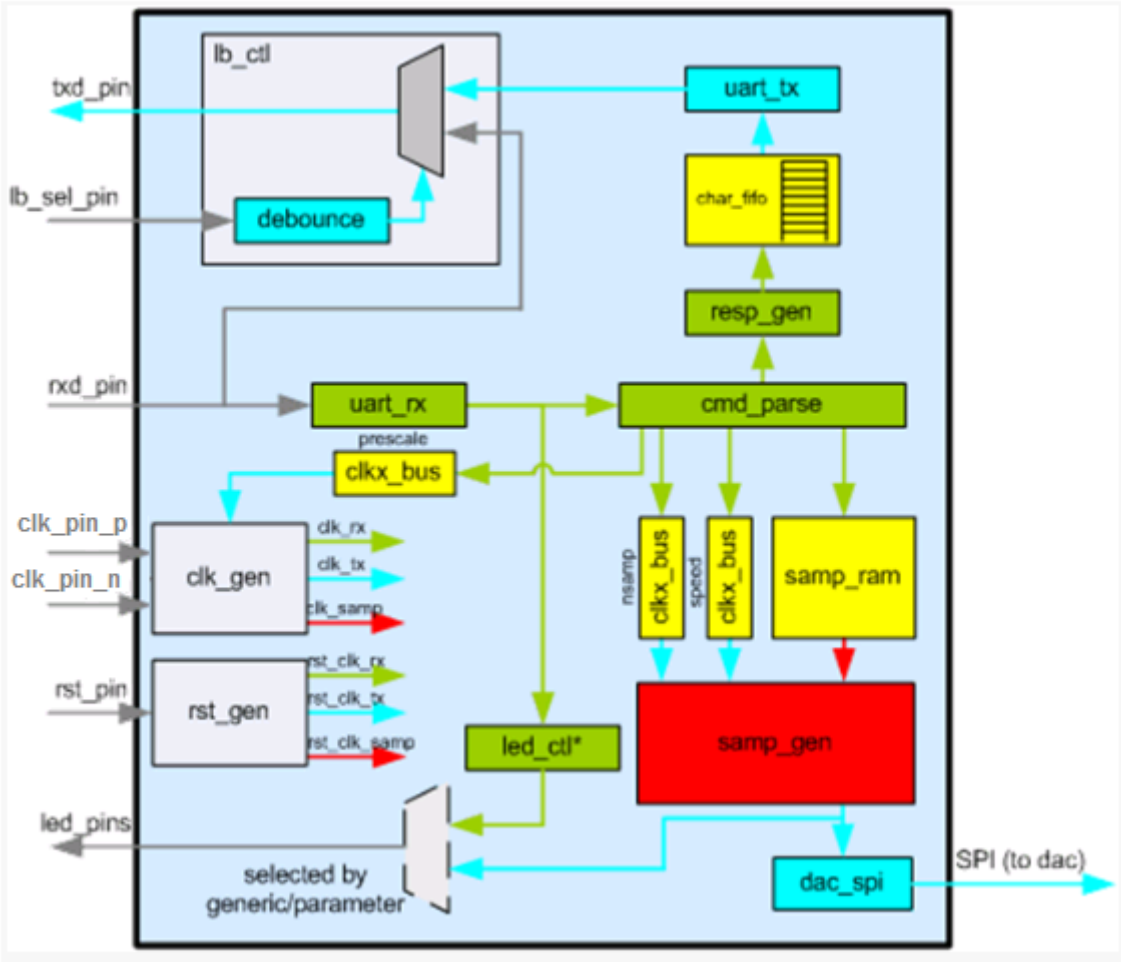


Figure 8-2: wave_gen Design Block Diagram

This design records specific information via RS-232 serial communication and stores this data in memory. After data has been stored, it can be retrieved via the RS-232 communications channel, or played out via a bank of LEDs or a DAC. The wave_gen design implements the RS-232 communication channel, the waveform generator and connection to the external DAC, and a simple parser to implement a small number of "commands" to control the waveform generation.

In the latest release of the tool, the `post_place_phy_opt` feature is enabled by default. For learning purposes in this lab, `post_place_phy_opt` has been disabled.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is expanded), and some tools require manual expansion (`/home/amd/training` for the

CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used in many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

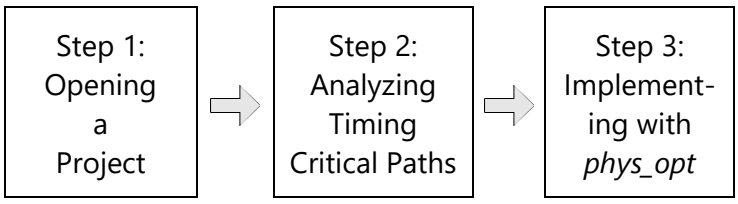
If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

Symbol	Description	Example	Explanation
<`text`**>**	Indicates a field	`cd <dir>`	`<dir>` represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is `XYZ`, then you would enter **`cd XYZ`** into the environment.
[`text`**]**	Indicates an optional argument	`ls [ | more]`	This could be interpreted as `ls <Enter>` or `ls | more <Enter>`. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the `more` tool, which paginates the output. Here, the pipe symbol (`|`) is a Linux operator.
|	Indicates choices	`cmd <ZCU104 | VCK190>`	The `cmd` command takes a single argument, which could be `ZCU104` OR `VCK190`. You would enter either `cmd ZCU104` or `cmd VCK190`.

General Flow



Opening a Project Step 1

This first step will show you how to open an RTL-based project and then open the implemented design.

1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-2. Open the Vivado Design Suite project named `wave_gen` located in the directory below.

1-2-1. Browse to the `$TRAINING_PATH/phys_opt/lab/KCU105/verilog` directory and open the **`wave_gen`** project.

If you do not recall how to perform this task, refer to the "Opening a Vivado Design Suite Project" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-3. Open the implemented design.

If you do not recall how to perform this task, refer to the "Opening the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Note: Click **OK** if the Critical Messages window appears.

Analyzing Timing Critical Paths Step 2

You will now analyze the timing failure path if any using timing reports and schematics.

Note: For learning purposes, the `post_place_phy_opt` feature in this lab has been disabled in the implementation settings.

2-1. Review the Timing Summary report to analyze timing critical path.

2-1-1. Select the **Timing** tab at the bottom of the Vivado IDE.

2-1-2. Review the **Design Timing Summary** report and answer the questions below.

Note: Click **OK** in the Critical Messages window if appears any.

Question 1

What are the Worst Negative Slack (WNS) and Total Negative Slack (TNS) values? What does this denote?

2-1-3. Click the hyperlink available for the **WNS** value.

This shows the failing path in the design, if any.

2-2. Review the failing path report.

2-2-1. Double-click the path to see the path details**.**

This will open the path report.

2-2-2. Review the path report and answer the question below.

Note: The timings of the paths may vary slightly.

Question 2

Using the Summary section of the path report, complete the **Before phys_opt** column in the table below.

	Before phys_opt	After phys_opt
Slack		
Data Path Delay		
Logic Levels		

2-3. Generate the schematic of the failing path.

2-3-1. Right-click the path from the Timing Summary report.

2-3-2. Click **Schematic.**

Note: Minimize the *clk_gen* hierarchy from the left side of schematic if needed.

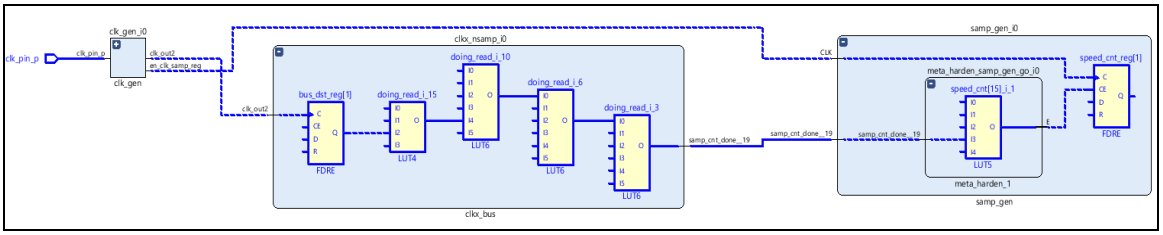


Figure 8-3: Schematic of Selected Path (KCU105)

You will compare this schematic with a *post-phys_opt* schematic in a later stage of this lab.

Question 3

What are the elements used in the data path?

2-4. Close the implemented design.

If you do not recall how to perform this task, refer to the "Closing the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

Implementing with phys_opt Step 3

Here you will create a new implementation run and use physical optimization techniques for timing closure. You will run implementation with `phys_opt` enabled.

3-1. Create a new implementation run with physical optimization option enabled.

3-1-1. Select **Flow** > **Create Runs** from the toolbar.

The Create New Runs wizard opens.

3-1-2. Select **Implementation** to create an implementation run.

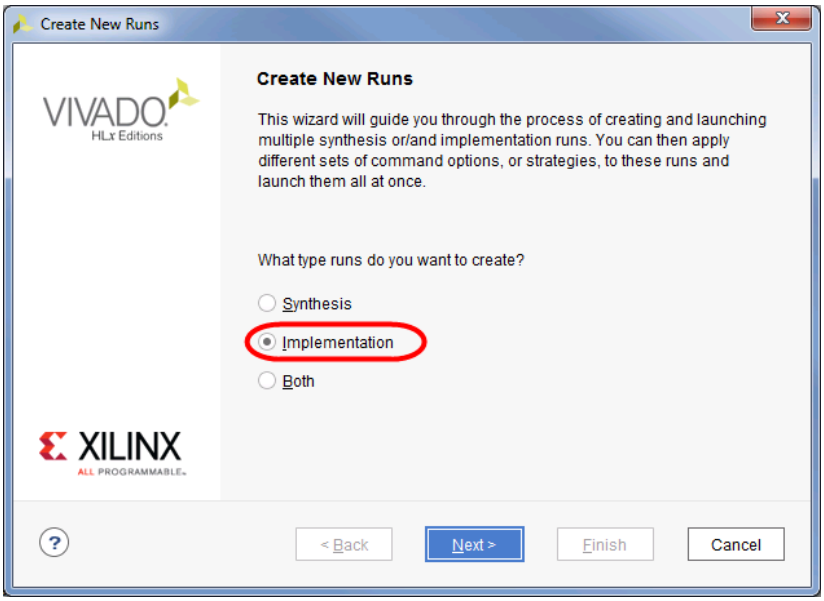


Figure 8-4: Creating a New Implementation Run

3-1-3. Click **Next** to configure an implementation run.

3-1-4. Select the **Make Active** check box in the Configure Implementation Runs dialog box to make the new run active.

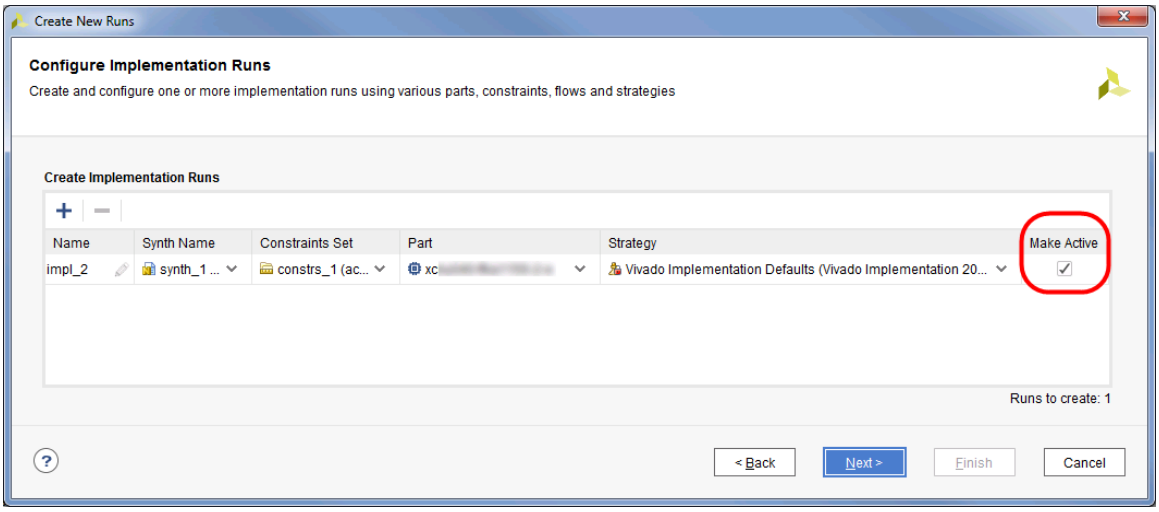


Figure 8-5: Making New Implementation Run Active

3-1-5. Click **Next** to set the launch option.

3-1-6. Select **Do not launch now** from the Launch Options dialog box.

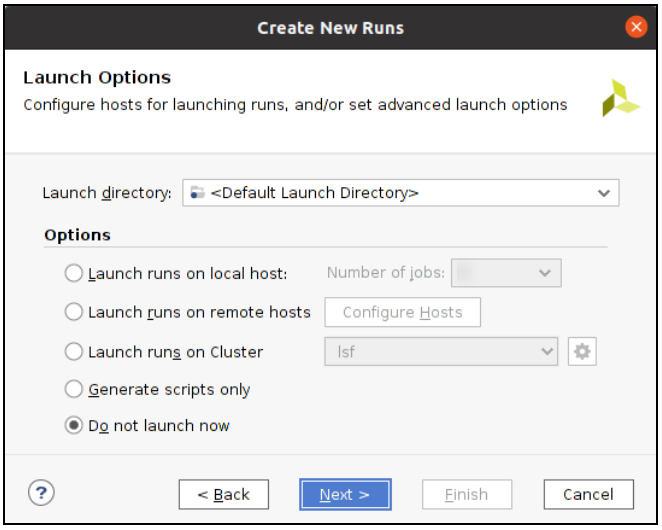


Figure 8-6: Launch Options Dialog Box

3-1-7. Click **Next**.

3-1-8. Review the summary and click **Finish**.

The Design Runs tab now shows the newly-created run.

3-2. Observe post-place and enable post-route `phys_opt`.

3-2-1. Select **Settings** under Project Manager using the Flow Navigator and select the **Implementation** tab**.**

3-2-2. Observe the check box beside **is_enabled** under Post-Place Phys Opt Design.

This is enabled by default for the **Vivado Implementation Default** run settings. This enables post-place physical optimization.

3-2-3. Select the check box beside **is_enabled** under Post-Route Phys Opt Design.

This enables post-route physical optimization.

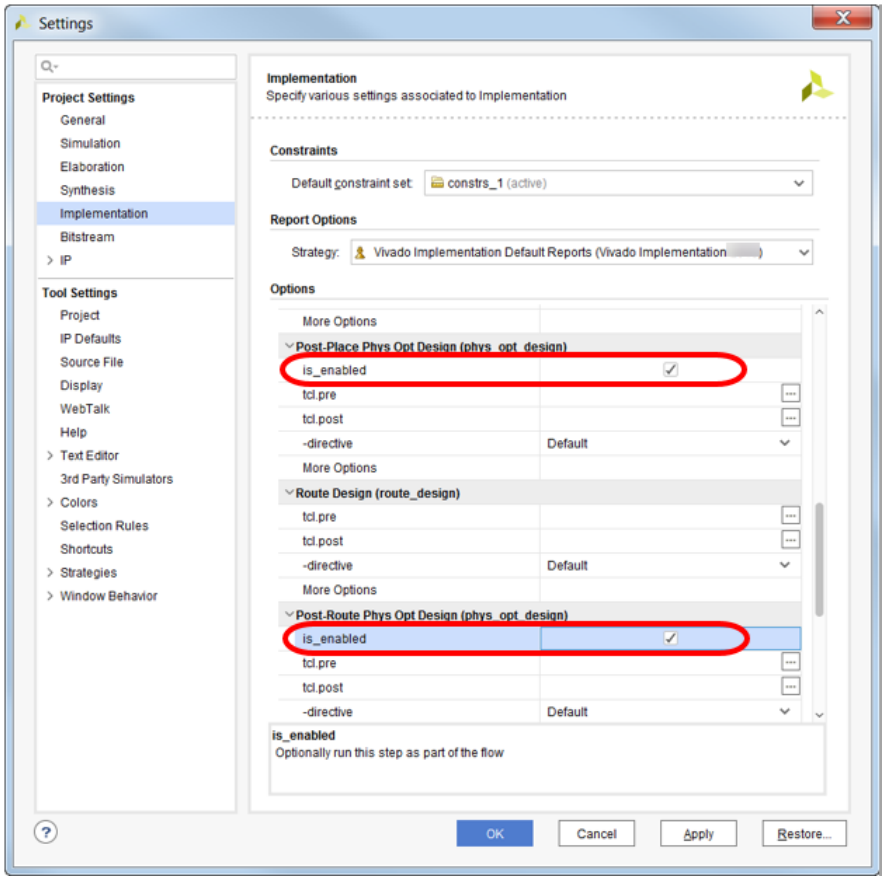


Figure 8-7: Enabling Post-Place and Post-Route phys_opt

3-2-4. Click **OK** to apply the changes.

3-3. Run implementation.

If you do not recall how to perform this task, refer to the "Running Implementation" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-4. Open the implemented design.

If you do not recall how to perform this task, refer to the "Opening the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-5. Review the Timing Summary report.

3-5-1. Select the **Timing** tab at the bottom of the IDE.

3-5-2. Review the **Timing Summary** report.

Question 4

What are the WNS and TNS value after physical implementation? Is the timing met?

3-6. Analyze the path that was failing previously.

3-6-1. Navigate to **Inter-Clock Paths > clk_out2_clk_core to clk_samp** > **Setup**.

3-6-2. Double-click the top-most path to open the path report.

3-6-3. Review the Summary section of the report.

Question 5

Using the Summary section of the path report, complete the **After phys_opt** column in the table below.

	Before phys_opt	After phys_opt
Slack		
Data Path Delay		
Logic Levels		

Question 6

What are the timing improvements that led to timing closure?

3-7. Generate the schematic of the timing path.

3-7-1. Right-click the path from the Timing Report.

3-7-2. Select **Schematic**.

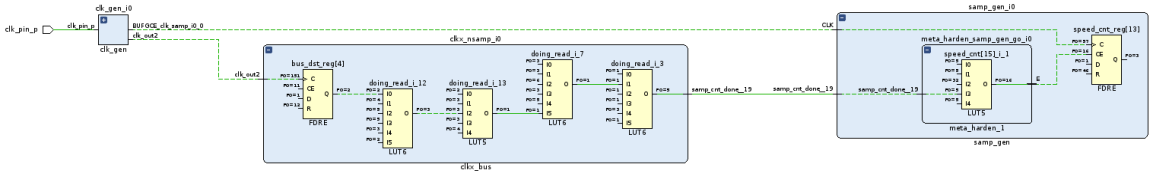


Figure 8-8: Schematic of the Timing Path after Phys_Opt (KCU105)

3-8. Review the implementation log to understand the physical optimization techniques performed.

3-8-1. Select the **Reports** tab at the bottom of the Vivado IDE.

3-8-2. Double-click **implementation_log** (**Vivado Implementation Log**) under Route Design.

This opens the log file in the main workspace area.

3-8-3. Use <**Ctrl** + **F**> to search for **phys_opt_design** after `place_design` is completed.

```
Time (s): cpu = 00:00:44 ; elapsed = 00:00:37 . Memory (MB): peak = 3046.676 ; gain = 1130.484
INFO: [Common 17-83] Releasing license: Implementation
104 Infos, 3 Warnings, 0 Critical Warnings and 0 Errors encountered.
place_design completed successfully
place_design: Time (s): cpu = 00:00:48 ; elapsed = 00:00:39 . Memory (MB): peak = 3046.676 ; gain = 1130.484
INFO: [Timing 38-480] Writing timing data to binary archive.
Writing placer database...
Writing XDEF routing.
Writing XDEF routing logical nets.
Writing XDEF routing special nets.
Write XDEF Complete: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.126 . Memory (MB): peak = 3046.676 ; gain = 0.000
INFO: [Common 17-1381] The checkpoint 'C:/training/phys_opt/lab/KCU105/wave_gen.runs/impl_2/wave_gen_placed.dcp' has been generated.
INFO: [runtcl-4] Executing : report_io -file wave_gen_io_placed.rpt
report_io: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.113 . Memory (MB): peak = 3046.676 ; gain = 0.000
INFO: [runtcl-4] Executing : report_utilization -file wave_gen_utilization_placed.rpt -pb wave_gen_utilization_placed.pb
INFO: [runtcl-4] Executing : report_control_sets -verbose -file wave_gen_control_sets_placed.rpt
report_control_sets: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.004 . Memory (MB): peak = 3046.676 ; gain = 0.000
Command: phys_opt_design
Attempting to get a license for feature 'Implementation' and/or device 'xcku040'
INFO: [Common 17-349] Got license for feature 'Implementation' and/or device 'xcku040'
INFO: [Vivado_Tcl 4-383] Design worst setup slack (WNS) is greater than or equal to 0.000 ns. Skipping all physical synthesis optimizations.
INFO: [Vivado_Tcl 4-232] No setup violation found. The netlist was not modified.
INFO: [Common 17-83] Releasing license: Implementation
113 Infos, 3 Warnings, 0 Critical Warnings and 0 Errors encountered.
phys_opt_design completed successfully
INFO: [Timing 38-480] Writing timing data to binary archive.
```

Figure 8-9: Post-Place Phys_Opt Messages in Impl Log (KCU105)

Note: Timing values may vary slightly.

If post-place `phys_opt` is performed, you will find a physical optimization summary table that lists all the `phys_opt` options that will be run by default and the corresponding timing improvement.

Question 7

Has the post-place physical optimization been performed?

3-8-4. Check if the design is meeting timing after routing is done.

Hint: Use <**Ctrl** + **F**> to search for **post router timing.**

3-8-5. Scroll down in the log file to see the post_route optimizations performed to improve timing.

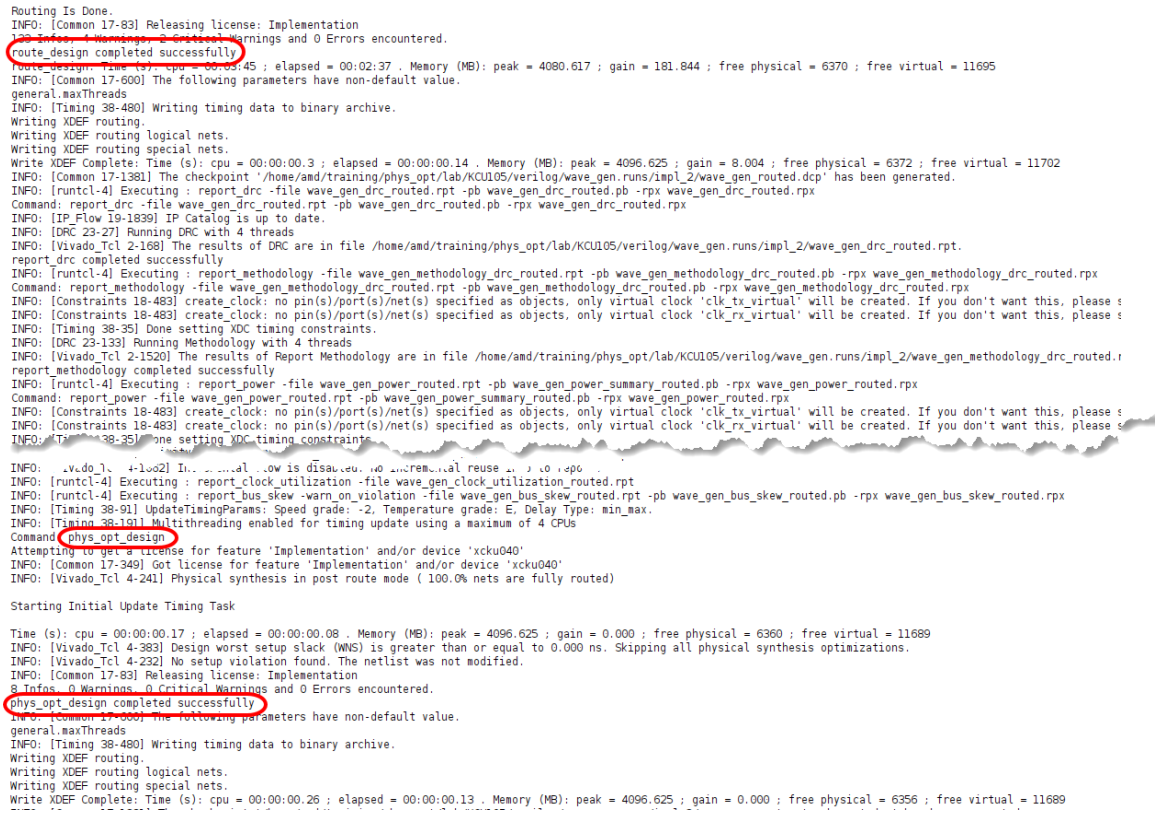


Figure 8-10: Post-Route Phys_Opt Messages in Impl Log (KCU105)

Note: Timing values may vary slightly.

The timing was passed when the post-route physical optimization was performed.

Question 8

What are the post-route optimizations performed to improve timing?

3-9. Close the implemented design.

If you do not recall how to perform this task, refer to the "Closing the Implemented Design" topic under Vivado Design Suite Operations in the *Lab Reference Guide*.

3-10. Close the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Closing the Vivado Design Suite" topic under the Vivado Design Suite Operations in the *Lab Reference Guide*.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/phys_opt` directory.

3-11. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

3-11-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/phys_opt`.

3-11-2. Select **`phys_opt`**.

3-11-3. Press <**Delete**>.

-- OR --

Using the command line:

3-11-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

3-11-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/phys_opt`

[Linux users]: `rm -rf $TRAINING_PATH/phys_opt`

Summary

In this lab, you have learned to implement a design with physical optimization techniques and observed the timing improvements brought about by *phys_opt*. You also viewed log files to understand the post-place and post-route phys_opt techniques performed.

Answers

1. What are the Worst Negative Slack (WNS) and Total Negative Slack (TNS) values?

   WNS = -0.607 ns and TNS = -5.784 ns

2. Using the Summary section of the path report, complete the **Before phys_opt** column in the table below.

	Before phys_opt	After phys_opt
Slack	-0.607	
Data Path Delay	7.863	
Logic Levels	5	

3. What are the elements used in the data path?

   One LUT3 and four LUT6 elements were used in the data path.

4. What are the WNS and TNS value after physical implementation? Is the timing met?

   Yes, timing met. WNS = 0.363 ns and TNS = 0.000 ns

   Note: Timing values may vary slightly.

5. Using the Summary section of the path report, complete the **After phys_opt** column in the table below.

	Before phys_opt	After phys_opt
Slack	-0.607	0.363
Data Path Delay	7.863	6.468
Logic Levels	5	5

   Note: Timing values may vary slightly.

6. What are the timing improvements that led to timing closure?

   The slack has improved from -0.607 to 0.363.

   The data path delay is reduced from 7.863 to 6.468.

7. Has the post-place physical optimization been performed?

   `phys_opt` is not performed if the WNS value after `place_design` is greater than or equal to 0.00 ns.

8. What are the post-route optimizations performed to improve timing?

   Physical optimization techniques like placement optimization and routing optimization are performed.

North America
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
U.S.A.
Tel: (408) 559-7778
www.xilinx.com

Europe
Xilinx Ireland Unlimited Company
2020 Bianconi Avenue
Citywest Business Campus
Saggart, County Dublin
Ireland
Tel: (353) 1-464-0311
www.xilinx.com

Asia Pacific
Xilinx Asia Pacific Pte. Ltd.
5 Changi Business Park Vista
Singapore 486040
Singapore
Tel: (65) 6407-3000
www.xilinx.com

Japan
Xilinx K.K.
Art Village Osaki Central Tower 4F
1-2-2, Osaki
Shinagawa-ku, Tokyo, 141-0032
Japan
Tel: (81) 3-6744-7777
japan.xilinx.com

Printed in the U.S.A.