Vivado Design Suite Tutorial

Design Analysis and Closure Techniques

UG938 (v2012.2) September 6, 2012
Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent.

Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

©Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2012</td>
<td>2012.2</td>
<td>First publication.</td>
</tr>
</tbody>
</table>
# Table of Contents

Design Analysis and Closure Techniques ............................................................................................................... 1

Revision History ...................................................................................................................................................... 2

Vivado Design Analysis and Closure Techniques Tutorial .......................................................................... 4

Overview .................................................................................................................................................................... 4

Software Requirements ....................................................................................................................................... 5

Hardware Requirements ...................................................................................................................................... 5

Tutorial Design Description ............................................................................................................................... 5

Preparing the Tutorial Design Files ................................................................................................................. 5

Lab #1: Design Analysis ............................................................................................................................................ 6

Step 1: Opening the Example Project ............................................................................................................. 6

Step 2: Viewing the Device Resources and Clock Regions .................................................................... 8

Step 3: Exploring the Logical Hierarchy ...................................................................................................... 15

Step 4: Displaying Design Resources Statistics ....................................................................................... 17

Step 5: Running Design Rule Checks (DRC) .............................................................................................. 19

Step 6: Performing Pre-Implementation Timing Analysis ................................................................... 22

Step 7: Implementing the Design ................................................................................................................. 28

Conclusion ............................................................................................................................................................. 33

Lab #2: Timing Closure Techniques .................................................................................................................. 35

Step 1: Analyzing Post-Implementation Timing ...................................................................................... 35

Step 2: Exploring the Hierarchy of the Timing Paths ............................................................................. 39

Step 3: Highlighting Module Placement and Connectivity .................................................................. 44

Step 4: Floorplanning the Design ................................................................................................................. 47

Step 5: Running Implementation with the Floorplan ............................................................................ 59

Conclusion ............................................................................................................................................................. 60
Vivado Design Analysis and Closure Techniques Tutorial

Overview

This tutorial introduces some of the capabilities and benefits of using the Xilinx® Vivado™ Design Suite for designing high-end FPGAs. The document contains a series of step-by-step exercises that highlight methods to achieve and maintain better performing designs in less time. The steps detail the following:

- Pre-implementation design and analysis capabilities
- Implementation exploration features
- Floorplanning implementation results

Note: This tutorial covers a subset of the features of the Vivado™ Design Suite. Additional features are covered in detail in other tutorials.

Tutorial Objectives

This tutorial takes you through the various analysis, floorplanning, and implementation features of the Vivado Design Suite. When using this tutorial, you should focus on the demonstrated techniques and how they relate to your own designs. The exercises cover the following topics:

- Analyze device utilization statistics to target alternate devices and choose the optimal device.
- Run Design Rule Checks (DRC) to quickly resolve constraint conflicts that would otherwise cause implementation errors.
- Use the Netlist, Logic Hierarchy, and Schematic windows to explore logic.
- Perform a quick estimation of timing performance to assess design feasibility and identify potential problem areas.
- View, modify, or create constraints in the design.
- Analyze the design hierarchical connectivity and data flow, as well as identify critical logic connectivity and clock domains.
- Floorplan timing-critical logic to improve timing and data flow.
- View Routing Resources in the detailed Device window.
Software Requirements

This tutorial requires that the 2012.2 Vivado Design Suite software release or later is installed. For installation instructions and information, see the Xilinx Design Tools: Installation and Licensing Guide (UG798).

Hardware Requirements

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

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

Tutorial Design Description

The sample design used throughout this tutorial consists of a small design called project_cpu_hdl, which targets an xc7k70t device. The small sample design used in this tutorial includes:

- A RISC processor CPU core
- A pseudo FFT
- Four gigabit transceivers (GTs)
- Two USB interfaces

A small design is used to enable timely completion of the synthesis and implementation runs with minimum hardware requirements.

Preparing the Tutorial Design Files

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

- <Xilinx_2012.2_install_area>/Vivado/<version>/examples/Vivado_Tutorial

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

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

- <Xilinx_2012.2_install_area>/Vivado/<version>/examples/Vivado_Tutorial.zip

The extracted Vivado_Tutorial directory is referred to as the <Extract_Dir> in this Tutorial.

Note: You will modify the tutorial design data while working through this tutorial. You should use a new copy of the original Vivado_Tutorial directory each time you start this tutorial.
Lab #1: Design Analysis

Step 1: Opening the Example Project

1. Start by loading the Vivado Integrated Design Environment (IDE).
   - Launch Vivado IDE from the icon on the Windows desktop, or
   - Type `vivado` from a command terminal.
   - The Vivado IDE opens.

   ![Figure 1: Vivado IDE Getting Started Page](image)

   The Vivado IDE Getting Started page contains links to open or create projects and to view documentation.
2. From the Getting Started page, click **Open Example Project** and select the **CPU (HDL)** design.

![Open Example Project](image1.png)

**Figure 2: Open Example Design**

3. The design project opens in the Vivado IDE, and a dialog box opens stating that the project is read-only and prompting you to save the project to a new location.

![Project is Read-Only](image2.png)

**Figure 3: Read-Only Project**

4. Select **Save Project As** and specify a project name and location.

![Save Project As...](image3.png)

**Figure 4: Save Project As...**

The project is saved to the specified location. If the specified directory does not exist, the Vivado Design Suite will create the directory as needed.

The Vivado IDE displays project information in the Project Summary window. The Project Summary window shows the project settings, and the run details for both the synthesis and implementation runs.

**Note:** This design will also be used in Lab #2 for the second portion of this tutorial. You will need to remember where you have saved the tutorial project in this step.
Step 2: Viewing the Device Resources and Clock Regions

1. In the Sources window, expand the Constraints folder to ensure constrs_2 is active, as shown in the following figure.

2. If needed, select constrs_2, right-click and select **Make active** from the popup menu.
Step 2: Viewing the Device Resources and Clock Regions

Click the Design Runs window and examine the runs information. The Design Runs window offers commands to manage, launch, and reset runs for both synthesis and implementation. You can manage multiple runs in the Design Runs window.

3. In the Flow Navigator, select Run Synthesis to synthesize the design.
   Watch the Log window for run status. It takes several minutes for the run to complete.

4. After synthesis completes, select Open Synthesized Design from the Synthesis Completed dialog box and click OK.
   The Synthesized Design opens with the netlist window as seen in Figure 8.

5. In the Layout Selector, in the main toolbar, change the window layout to the Default Layout if that is not the current window layout.

6. Click on the Device window to make it active.
Figure 8: Open the Synthesized Design

7. In the Device window, use **Zoom Area** by clicking and holding the left mouse button, while dragging down and to the right.

   This will draw a Zoom Area rectangle. Repeat as needed to zoom into the device close enough to see the different device resources.

   In the Device window, the Vivado IDE displays the FPGA device resources that can be used to implement the design, as a matrix of tiles. Sites are available within a tile for placement of netlist instances. The available sites include, for example, SLICEs, RAMs, MULTs, and DSPs, and vary in shape and color in the Device window to differentiate the object types contained in the site. Slices are fundamental building blocks of Xilinx FPGAs, containing LUTs and registers, though the specific contents of a slice may vary with the device family. For more information on these terms, refer to the [Xilinx Glossary](http://www.xilinx.com).

   “LOC” and “BEL” placement constraints are used to assign logic to either a site or a slice. Primitive logic can be assigned specifically to a site, or generally to a slice with a “LOC” placement constraint, or can be assigned to specific logic resources within a slice using both “LOC” and “BEL” constraints.
8. Move the cursor slowly over the various device resources to view the tooltips, and take note of the following:

- Hover the mouse over a site to show a tooltip.
- I/O port locations and I/O buffer assignments appear inside I/O banks.
- Tall dark red tiles indicate a RAMB36 site that can also hold two RAMB18s or a FIFO.¹
- Tall green tiles indicate sites for DSP48s.
- Square blue tiles are a CLB containing SLICEs.
- SLICE coordinates appear in the status bar at the bottom right of the Vivado IDE main window.

¹ The colors mentioned here are for the Vivado IDE default color scheme. Your colors may be different.
Examining Sites and Bels

Below the Netlist window, the Site Properties or BEL Properties window is displayed. These windows appear when a site or bel is selected within the Device window, or some other window.

![Figure 10: Bel Properties window](Image)

9. Select several different types of resources within the Device window, and examine the Site and BEL Property information.

10. Use the **Zoom Fit** command from the Device window toolbar menu, or click and drag the cursor up and to the left in the Device window, to see the whole device.

11. Click the **Auto Fit Selection** button on the Device window tool bar menu, to zoom and center any objects selected in the next section.

Examining Clock Regions

The Device window shows the clock regions on the device, which is useful to help guide floorplanning and placement. The Clock Region window lists all the clock regions in the device. The Clock Region Properties window displays clock region information to indicate potential clock contentions prior to implementation. Viewing clock domains is covered later in the tutorial.

12. Select **Window > Clock Regions** to display the Clock Regions window.

13. Select one of the clock regions listed in the Clock Regions window.

The clock region is highlighted in the Device window, and the Clock Regions Properties window is displayed.

**TIP:** The Clock Regions and the Clock Regions Properties windows are initially overlapping. To see them as shown in **Figure 11**, you must select one of the windows and move it up onto the Sources window. Refer to the Vivado Design Suite User Guide: Using the Vivado IDE (UG893) for more information on moving windows.
Step 2: Viewing the Device Resources and Clock Regions

14. Click the Properties tab to examine the Clock Region Properties window.

15. Click the Statistics tab of the Clock Region Properties, and scroll to examine the available device resources of the selected clock region.

16. Click the Resources tab of the Clock Region Properties, and select and examine the locations of the BUFR and IDELAYCTRL sites.

17. Select one of the BUFRs, and notice it is highlighted in the Device window.
   
   **Note:** If Auto Fit Selection is enabled, the Device window is fit to the BUFR on selection.

18. In either the Device window or the Properties window, right-click and select Mark to mark the selected BUFR site.

19. In the Device window, use Zoom Fit to fit the whole device.
   
   Notice the mark on the BUFR allows you to quickly locate the site in the zoomed out Device window.

20. Select the Unmark All button on the main tool bar menu to remove the mark.

21. In the Clock Region Properties window, click the I/O Banks tab to examine the I/O Banks in the selected clock region.

**TIP:** To locate the I/O Banks tab, you may need to scroll the Clock Region Properties window using the displayed arrows.
22. In the Clock Region Properties window, select one of the I/O banks.

The selected I/O bank is highlighted in the various open windows, such as the I/O Ports window, the Device window, and the Package and Package Pins windows. You can open and switch between these different windows to see the various representations of the selected I/O bank.

23. Select Window > Clock Resources to open the Clock Resources window.

In the Clock Resources window, the Vivado IDE displays a simplified view of the I/O and clocking resources in a spreadsheet-like interface. This window shows the relationship and interactions between regional and global clocking resources on the device: BUFRs, BUFIOs, BUFGs, MMCMs and GTs, while maintaining proper relative positioning between these resources. You can use the Clock Resources window to plan and place critical clocking and I/O logic prior to placing and routing the rest of the design.

In Figure 12, the Clock Resources window and the Device window are sharing space in the graphical windows area of the Vivado IDE. You can arrange windows in this manner using...
Step 3: Exploring the Logical Hierarchy

1. Click the Netlist window tab, and if needed, click the **Collapse All** button.

2. Click the plus sign next to the **usbEngine0** module to examine the hierarchy.

   The Netlist window should now look like **Figure 13**.

![Figure 13: Expanding the Netlist Window](image)

3. Click the plus sign next to the **Nets** folder of the **usbEngine0**.

   These are the nets inside the **usbEngine0** module. You can collapse the Nets folder by clicking the ‘-’ next to it.

4. Click the plus sign next to the **Primitives** folder of **usbEngine0** to expand and examine the contents.

   These are the primitive instances inside the top-level of the **usbEngine0** module, not inside any of the sub-modules.

   **Note**: There is also a Primitives folder at the top level of the Netlist window that contains the top-level instances of each module.

5. **Select** the **u4** module.

6. Right-click and select **Show Hierarchy**, or press **F6**.

   The Hierarchy window opens in the Workspace. The Hierarchy window is used primarily during design analysis and floorplanning. The Hierarchy window displays the hierarchical...
relationship of the modules. The width of the blocks in the Hierarchy window are relative to the device resources consumed by each block. It lets you see how a timing path traverses the logic hierarchy, or gauge how big a module is before you floorplan the module.

Figure 14: The Netlist and the Hierarchy Windows

You can select modules directly from the Hierarchy window for floorplanning. Logic selected in other windows is highlighted in the Hierarchy window.

7. Select any module in the Hierarchy window, and notice that the module is selected in the Netlist and other windows, as well. Selection works across windows.

8. Click the Unselect All main toolbar button, or press F12.

Cross probing RTL Source Files

9. Select the u4 block in the Hierarchy window, or in the Netlist window.

10. Right-click and select Go to Instantiation from the popup menu.

Notice the RTL source file is now open in the Text Editor, with the line number highlighted where the logic instance was derived. You can use this feature, and the Go to Definition command, to quickly locate a specific instantiation of a module, or the definition of a module in the RTL source files.

**TIP:** The Text Editor displays some file formats with context-sensitive syntax highlighting to make it easy to identify keywords and comment lines within the file.
11. Select several other logic instances in either the Netlist or Hierarchy window and view the RTL source files.

12. Close all open RTL source files in the Text Editor.

13. Click the Unselect All main toolbar button, or press F12.

---

**Step 4: Displaying Design Resources Statistics**

The Vivado Design Suite utilization analysis provides design statistics to help you determine the optimal device to fit the design. The tool helps you see how the logic resources are split between modules. It is easy to explore multiple device types to determine the best overall utilization and performance estimations.

**Viewing the Resource Estimates of the Entire Design**

   
   This opens the Report Utilization dialog box.

2. Click **OK** to accept the defaults.

3. Scroll down the **Utilization** window.
   
   The Utilization window shows device resource utilization sorted by logic type on the left side, and displayed according to the design hierarchy on the right side.

4. In the Utilization window, click **Memory > RAMB36/FIFO36**

   Notice that the `usbEngine0` and `usbEngine1` are the two largest consumers of block RAM, with each consuming approximately 27% of the available resources on the target device.
5. Click the plus sign next to the `usbEngine1` to expand and examine the hierarchy. Expand and examine some of the other resources as well.

6. Select `usbEngine0` from the Memory section of the Utilization window.

   Notice that `usbEngine0` is also selected in other open windows, such as the Netlist window, and the Hierarchy window if it is still open.

7. Close the Utilization window by clicking the X button in the upper right corner of the window.

Examining Resources in More Detail

The Pblock Properties window offers you another way to view the device resources, and the resource utilization by the design.

8. Select **Window > Physical Constraints** to open the Physical Constraints window.

9. In the Physical Constraints window, select the **ROOT** design.

   This opens the Pblock Properties window. The Vivado Design Suite provides the ability to hierarchically divide the design into smaller, more manageable physical blocks (Pblocks). Pblocks can include logic modules and primitive logic from anywhere in the design hierarchy.

   The Vivado tool also creates a single Pblock from the root design, or the top-level design, even when there are no other Pblocks defined. In this section you will examine the resources available in the root Pblock.

10. In the **Pblock Properties** window, select the **Statistics** tab.

11. View the **Physical Resource Estimates** displayed in the Pblock Properties window, as shown in **Figure 16**.

   ![Figure 16: Viewing Design Resource Statistics](image)
Step 5: Running Design Rule Checks (DRC)

- **Available**: Reflects the available sites covered by the Pblock, as drawn.
- **Required**: Reflects the number of sites required by the logic assigned to the Pblock.
- **% Util**: Displays the estimated utilization percentage for each logic type. If utilization for logic objects is over 100%, the text appears in red. In this case the Pblock may need to be redrawn.

**Note**: The calculated Utilization % assumes the maximum site utilization, or all resources used by the required logic. Realistically, due to placement and routing congestion, and timing constraints, the maximum site utilization is seldom achieved. You should target a utilization of approximately 80%, depending on the target device and the characteristics of your design and constraints.

Remember that the ROOT Pblock includes the whole design placed onto the entire Xilinx FPGA. In this case the Pblock Properties statistics reflect the logic requirements of the whole design, and reflect the resources available on the target part. If the utilization % is exceeded for the ROOT, you will need to select a different target part.

12. Scroll through the design statistics displayed in the Pblock Properties window.

   The Statistics tab displays the device resource utilization for each type of logic element, carry chain count and longest length, clock report, I/O utilization, and primitive and I/O statistics. If the design had any RPMs, the RPM count and maximum size information would also be shown.

13. Close the **Physical Constraints** window.

---

**Step 5: Running Design Rule Checks (DRC)**

Xilinx recommends running the Design Rule Checker (DRC) before implementation to check for some common design issues. The Vivado Design Suite implementation tool DRCs are used for design sign-off, and take precedence over the DRCs in the Vivado IDE.

**Running the Design Rule Checker (DRC)**

   
   The Report DRC dialog box opens.
2. Click **OK** to run all rule checks.

   The DRC window opens and indicates that there are several DSP48 and block RAM warnings and Errors in the GTs.

   Errors, warnings and information messages display in the DRC violations window.
   - Information messages are indicated by a blue icon.
   - Warnings are indicated by a yellow icon.
   - Critical Warnings are indicated by an orange icon.
   - Errors are indicated by a red icon.
3. In the DRC window, click the **REQP #1** error.

The Violation Properties window opens and describes the violation. The selected violation is a required pin error (REQP-198) associated with the connection requirements for a GTXE2 on a Kintex™-7 device.

![Figure 19: Violations Properties window](image)

Notice that the GTXE2_CHANNEL cell is also selected in the Netlist and Device windows.

4. Right click in the Violations Properties window.

By default the Vivado IDE will automatically select the objects reported in the DRC messages. You can also configure the tool to mark the objects reported in the DRC messages.

5. In the Violations Properties window, press **F4** to open the schematic for the object related to the selected violation.

![TIP: Since the object is also selected when you select the error, you can use the Schematic command, or press F4, from the Netlist or Device windows.](image)

6. **Zoom** into, and **scroll** down the GTXE2_CHANNEL symbol in the Schematic window, until you locate the RXOUTCLK and TXOUTCLK pins associated with the error, about half way down the symbol.

7. **Select** the outside portion of either the RXOUTCLK or TXOUTCLK pins, and **double click** on that pin.

This expands the schematic from the selected pin to the next connected primitive. Because there is no load on the output, which is the source of the error, only a stub of wire is displayed in the expanded schematic.

In this case, you can ignore the reported error, because the no load wire will be optimized out of the design during implementation, when opt_design is run prior to placement and routing. Refer to the **Vivado Design Suite User Guide: Implementation (UG904)** for more information.

8. Close the Schematic and DRC windows by clicking the **X** button in the tab.
Step 6: Performing Pre-Implementation Timing Analysis

In this step, you will investigate timing before implementing the design.

At this point, prior to implementation, the Vivado IDE timing engine provides an estimate of what the route delays will be, and does not report on whether the design has met timing. Before running implementation it can be helpful to perform an early Static Timing Analysis, including estimated route delays, to determine the achievability of the timing constraints in the design.

Analyzing Timing End Points with the Slack Histogram

The Slack Histogram shows the timing slack calculated at the endpoint. It provides a visual indication of the timing delays in the design, grouping endpoints into bins based on slack. The histogram lets you visualize how many endpoints have tight timing versus those that have a margin.

This can assist you in determining your next steps if the design is not meeting performance requirements; helping you focus on critical timing paths with negative or low slack values. Endpoints with negative slack are expected to have timing problems. Endpoints with low slack margins in preliminary timing analysis may not meet timing targets during implementation.

IMPORTANT: The Slack Histogram does not show the complete timing path. You must use the Report Timing command to see the timing of a complete path.

2. Change the Number of bins to 20.

The range of slack values found by the Vivado tool is divided into the number of bins specified, then the timing endpoints are grouped into the appropriate bin based on the slack value at the endpoint. A smaller number of bins can offer a quick summary view of the timing performance of the design. A larger number of bins is better used to provide details within a specific range of delays.

The Generate Slack Histogram for Endpoints dialog box should look like the following:
3. Click the **Timer Settings** tab.

Examine the Timer Settings tab. You can specify various delay parameters used by the timing engine, to determine how the Vivado tool accounts for route delay when calculating slack. The timer can estimate route delay, or assume there is no interconnect routing delay.

4. Click **OK**.

A histogram displays the slack at the timing endpoints, grouped into bins. A table under the histogram lists the timing endpoints, and the calculated slack value.
Figure 21: Slack Histogram

If there were endpoints with negative slack in this design, the failing endpoints would be displayed in a red bin, to the left of the 0 value in the X-axis. Endpoints with negative slack are expected to have timing problems.

5. Click on various bins.

The endpoints listed in the table, under the histogram, are filtered according to the selected bins. The slack range is also updated.

6. Select the left most bin, which is the bin with endpoints with the lowest slack margins, closest to failing timing.

Scroll down to inspect the endpoints and see the calculated slack values. Negative slack values would be displayed in red.

7. Select the first ten timing paths listed in the table below the slack histogram.

Use the Shift key to select multiple timing paths from the list.

8. Right click in the table, or in the histogram, and select the Report Timing command from the popup menu.
The Report Timing dialog box is opened, with the selected endpoints from the slack histogram listed as shown in Figure 22.

Analyzing the Pre-Placed Timing

The Report Timing dialog box opens displaying the Targets tab. This lets you define the timing paths for the Vivado timing engine to analyze by specifying start points, through points, and endpoints.

The Report Timing command is a static timing path analyzer. You can use Report Timing to view specific timing paths at any point of the flow after synthesis when you need to investigate timing problems, or when you want to report the validity and the coverage of particular timing constraints.

![Figure 22: Report Timing – Targets](image)

9. Select the **Options** tab.
10. In the **Number of paths per group** field, type 30.

11. Click **Advanced** and **Timer Settings**, and examine the dialog box to view the various options available.

   Do not change anything on these tabs.

   At this point in the design flow, prior to implementation, the Interconnect pull-down in the Timer Settings tab has two values: estimated, and none. Prior to running the Vivado implementation tools, the timer cannot give you actual interconnect delays. The routing delay can be estimated based on a routing delay model, or can be left out of the analysis.

12. Click **OK** to run timing analysis.

   The Report Timing window opens.
Step 6: Performing Pre-Implementation Timing Analysis

The tree view on the left displays the settings used to generate the timing report, and allows you to display either the timing results for setup or hold analysis.

The list of paths in the right side of the timing report are initially grouped by constraint. This can be disabled, to simply list all paths without grouping, by disabling the Group by Constraint command on the timing report tool bar menu.

The report includes columns for the calculated timing slack, start point, endpoint, delay values, the number of logic stages, and the source and destination clocks. Paths that fail timing, appear in red in the report.

Scroll down the list of reported timing paths. Notice that most of the timing paths have start points in the usbEngine0 module. The timing report can be sorted by clicking on the header of any of the columns in the report.

13. Click the To column header twice to sort the list by destination.

The report is now reverse sorted by To column values.

**TIP:** You can sort all of the table style windows in the Vivado IDE in the same way. Click once to sort in ascending order, click twice to sort in descending order. To add a secondary sort, press Ctrl and select an additional column header.

The timing report in Figure 24 is grouped by constraints, and sorted first by slack, in ascending order, and then by destination, in descending order.

14. Click HOLD on the left side of the Timing window to view the paths listed under the hold report.

Prior to implementation, you should focus on Setup violations, as Hold checks are dependent on routing.

15. Click Settings on the left side of the Timing window to examine the timer settings used when the report was generated.
16. **Close** the Timing Report and the Slack Histogram windows by clicking the X button in the window.

---

**Step 7: Implementing the Design**

Vivado implementation is a timing-driven flow with native support of industry standard Synopsys Design Constraints (SDC) commands, and Xilinx Design Constraints format (XDC), to specify design requirements and restrictions. Vivado implementation encompasses all of the design steps required to place and route the netlist onto the FPGA device resources while meeting the logical, physical, and timing constraints of the design.

**Important!** The Vivado Design Suite does not support the UCF format. For information on migrating UCF constraints to XDC commands, refer to the Vivado Design Suite: Migration Methodology Guide (UG912).

---

**Examining the Constraints File**

1. If the Sources window is hidden from view, display it by selecting **Window > Sources**, or selecting the Sources tab as shown in **Figure 25**.

2. In the Sources window, click the **Collapse All** button, then expand constrs_1 and constrs_2.

![Figure 25: Sources Window](image)

When running the implementation tools, the Vivado tool uses the active constraint set by default.

3. **Make sure** that constrs_2, with top_full.xdc, is the active constraint set.

4. Double-click **top_full.xdc** to open the file, and examine the contents of the constraint file.
5. Close the XDC file.
   Do not save any changes if prompted.

**Defining Implementation Runs**

The Vivado IDE allows you to configure and run multiple synthesis and implementation runs simultaneously. This tutorial will demonstrate how to do that, but you will only launch one implementation run in order to conserve time.

6. In the Flow Navigator, right-click on Implementation, and click **Create Implementation Runs** in the popup menu.

7. Click **Next** in the Create New Runs confirmation dialog.

8. In the Set-up Implementation Runs dialog, accept the default Synthesized netlist, Constraints set, and Part, and click **Next**.

   The Choose Implementation Strategies dialog box opens, letting you define and name one or more implementation runs, and select from standard Vivado strategies, or user-defined strategies to use during the run.

   The first implementation run listed should be `impl_2`. The strategy listed will depend on whether you have created user-defined implementation strategies. See the *Vivado Design Suite User Guide: Implementation (UG904)* for more information on defining strategies.

9. Click the **Strategy** pull-down menu to select the **HighEffort** strategy listed under Vivado Strategies.

10. Click **More** to add a new run, called `impl_3`.

![Figure 26: Choose Implementation Strategies](image)
11. Click the **Strategy** pull-down menu to select the **HighEffortPhySynth** strategy.
   
   This is the implementation run you will be using.

12. Click the optional **Make Active** radio button for the **impl_3** run.

   If you are creating multiple new runs, only one run can be made the active run. By default, the Vivado IDE will direct commands to the active run, and display reports for the active run.

13. Click **Next** to open the Launch Options dialog box.

   Review the available launch options. You can launch the runs locally, launch the runs remotely (on Linux only), compile run scripts to be used later, or save the runs without launching.

14. Select **Do not launch now**, and click **Next**.

15. Review the Create New Runs Summary, and click **Finish**.

   The Design Runs window should look as follows:

   ![Figure 27: Design Runs window](private_url)

### Launching Runs

The next step in the flow is implementing the design through the Vivado IDE implementation tools. The Vivado implementation process includes logical as well as physical transformations of the design, and consists of the following sub-processes:

- **Opt Design**: Optimize the logical design to make it easier to fit into the target Xilinx FPGA. The DRC issue examined earlier in this tutorial will be resolved during this optimization.

- **Power Opt Design**: Optionally optimize elements of the design to reduce power demands of the implemented FPGA. This optimization will not be used in the implementation run you just created.

- **Place Design**: Place the design onto the target Xilinx device.
• **Phys Opt Design**: Optionally optimize the timing of the design by replicating drivers of high-fanout nets to distribute the loads. This optimization is used as part of the strategy you selected for your implementation run.

• **Route Design**: Route the design onto the target Xilinx device.


   Implementation takes several minutes to complete. The status of the implementation run is displayed in the project status bar at the upper right of the Vivado IDE.

   While you are waiting for implementation to complete, you can experiment with some of the analysis features available for use with the synthesized design. Commands such as the Report Clock Networks and Report Clock Interaction can be run from the Flow Navigator on the open synthesized design while implementation is running.

   The Implementation Completed dialog box displays when the run is finished.

17. Select **Open Implemented Design** in the Implementation Completed dialog, and click **OK**.

18. Select **Yes** if prompted to close the synthesized design before opening the implemented design, and **Don’t Save**, if prompted.

---

**TIP:** This prompt might have been turned off if you elected to never show this dialog again. To restore this prompt, go to **Tools > Options > Window Behavior** and check the **Show dialog before switching to a different design** checkbox.

---

The results from the implemented design are imported into the Vivado IDE as shown in **Figure 27**. The Device window shows the placement of all logic instances onto the available device resources.
19. Click the **Messages** window tab at the bottom of the Vivado IDE, to display the Messages window.

   **Note:** The Messages window collects all information, warning, and error messages from the various tools of the Vivado Design Suite.

20. Click the **Collapse All** toolbar button to display the various message categories.

21. Expand the **Implementation** messages.
22. **Turn off** the Info messages by clicking the **checkbox** in the header of the Messages window, next to the blue circle with the exclamation mark, to toggle the display of informational messages.

23. Expand **Route_Design** folder to see the messages generated at this step.

24. Click the **Reports** window tab to display the various reports generated by different tools in the Vivado Design Suite.

   If the Reports window tab is not displayed, select **Window > Reports** to display the Reports window.

25. Double-click any reports listed in the Reports window to open those reports in the Vivado IDE text editor.

26. **Close** any open report files.

---

**Conclusion**

In this lab, you have opened the CPU HDL project, and examined various aspects of the design using tools available within the Vivado IDE. You have examined the hierarchy of the design,
analyzed the timing slack histogram, and the timing report prior to implementation, and synthesized and implemented the design. Finally, you examined the messages and reports generated by the Vivado implementation tools.

This concludes Lab #1. You can continue to Lab #2, or use the following steps to exit the tool:

1. Click **File > Close Project** to close the CPU HDL design.
2. Click **File > Exit** to exit the Vivado Design Suite.
Lab #2: Timing Closure Techniques

This lab continues from the end of Lab #1 in this tutorial. You must complete Lab #1 prior to beginning Lab #2. If you closed the tool, or closed the tutorial project at the end of Lab #1, you will need to open them again.

1. Start by loading the Vivado Integrated Design Environment (IDE).
   - Launch Vivado IDE from the icon on the Windows desktop, or
   - Type `vivado` from a command terminal.
   - The Vivado IDE opens.

2. From the main menu, click **File > Open Recent Project** and select the project you saved in Lab #1.

3. After the project has finished opening, from the Flow Navigator click **Open Implemented Design**.

---

**Step 1: Analyzing Post-Implementation Timing**

With the design implemented onto the target device, you can begin to explore the design and constraints to resolve any challenges with meeting the timing requirements of the design. The Vivado Design Suite offers an array of tools and techniques to help you achieve timing closure.

The Report Timing Summary command creates a set of timing reports intended to provide sign-off level timing analysis for the post-implemented design. You can analyze timing results from the implementation process to drive any floorplanning that might be required to meet timing requirements.

**IMPORTANT:** Only the Vivado timing analysis Report Timing Summary command provides a sign-off level timing report.

The Vivado tool runs timing analysis after implementation, and generates a report file you can open in the Text Editor for review. However, to view the report interactively in the Vivado IDE, and cross-select and analyze the design, you must regenerate the report.

**Viewing Timing Paths**

1. From the Flow Navigator select **Report Timing Summary** and click **OK** to run Vivado timing analysis with the default settings.

   The Timing Summary window opens, as shown in **Figure 31**. The left side of the window displays a tree view of different reports available in the Timing Summary. The right side of
the window displays the details of the selected report. In the figure below, the Design Timing Summary is selected.

![Figure 31: Report Timing Summary](image)

The Design Timing Summary reports timing slack values, such as Worst Negative Slack (WNS), showing whether the design satisfied the timing constraints. Any negative slack values, or a number of failing endpoints, is an indication of timing problems.

In this design, the Design Timing Summary reports a WNS of 0.551 ns, which is actually a positive slack value and so not a timing failure. However, in this case, you will investigate the timing paths with the most critical slack.

2. Use the **Show Search** command in the Timing Summary window, and enter a value of **.55** to search for paths associated with the worst slack value, .551 ns.

![Figure 26: Searching for the Critical Timing Path](image)

3. Select **Path 1**.

Notice that the worst timing paths in the report connect to the `usbEngine1` module. This information will guide your floorplanning strategy later in this tutorial.

4. Right-click and select **Mark** in the popup menu.

In an implemented design, selecting the timing path in the report also selects the path in the Device window. The Device window can make it easier to see timing challenges due to placement and routing, and help you visualize how floorplanning could improve timing.
Step 1: Analyzing Post-Implementation Timing

The selected path is marked with a green diamond showing the timing path start point, and a red diamond shows the timing path end point. Yellow diamonds show any through points in the path.

5. Right-click and, from the popup menu, select Unmark, or select the Unmark All button from the main toolbar to clear the marks.

Viewing the Routed Path

6. In the Device window, select the Routing Resources toolbar button to enable the display of routing resources.

When enabled, the Routing Resources command displays a graphical representation of the device components, and the FPGA interconnections between the components. This provides a detailed view of the available routing resources on the device, and shows the specific resources used by a selected path, as seen in Figure 34.

7. In the Device window, select the Auto Fit Selection toolbar button to enable automatic zoom for each newly selected object.

Select various paths in the Timing Summary window, and notice the detailed routing displayed for each timing path in the Device window. This routing resources view can
provide you a better understanding of the interconnect portion of the path delay, and help you resolve timing problems.

Figure 33: Routing Resources in the Device window

8. **Zoom** closely into a timing path displayed in the Device window to see the details of the routing resources.

   When looking at the routed path, it may seem to take an indirect route between connection points. This is because the Vivado router considers many different aspects of a design. If the design is meeting timing, other considerations such as routing density and power analysis, or timing requirements on other paths may affect the route solution.

9. Click the **Unselect All** button on the main toolbar menu, or press **F12**, to unselect all currently selected objects.

10. In the Device window toolbar menu, select the **Routing Resources** toolbar button again to disable the detailed routing view.

    Notice the placed cells are now displayed in the Device window again, without the added details of the routing resources.
Step 2: Exploring the Hierarchy of the Timing Paths

Viewing Path Properties

When you select paths in the Timing Summary window, the Path Properties window opens to show a summary and details of the timing path.

11. In the Timing Summary window, reselect Path 1, or the first path.

12. Select the Path Properties window, and Maximize, or Float, the window for the selected path, and examine the information provided as shown in Figure 34.

```
Figure 34: Path Properties window
```

13. Restore or dock the Path Properties window to its original size and location.

TIP: You can maximize or restore a window by double-clicking in the window banner.

Step 2: Exploring the Hierarchy of the Timing Paths

The Vivado IDE offers many tools to help you locate logic associated with the reported timing paths. In this step of the tutorial, you will use the Schematic and Hierarchy windows to visualize the logic associated with the timing paths with the worst delay.

1. Select all ten paths currently displayed under Inter-Clock Paths on the right side of the Timing Summary window.
You can hold down the SHIFT or CTRL keys to select multiple paths. All of the selected paths should start from `usbEngine1`.

2. Right-click select **Schematic** from the popup menu, or press the **F4** key.

   When you select a timing path in the Timing Summary window, the Vivado tool also selects the cells associated with the start points, end points, and through points on the timing paths. The Schematic window shows all of the logic cells on the selected timing paths.

3. In the Netlist window, click the **Collapse All** button.

   This simplifies the information displayed in the Netlist window, so that the items selected in the next task are more visible, and easier to see.

   **TIP:** Sometimes, because of the volume of information presented in the Netlist window, it is beneficial to collapse the tree view, so you can more clearly see newly selected objects.

4. In the Schematic window, right-click and select **Select Primitive Parents**.

---

**Figure 35: Timing Path Schematic – Select Primitive Parents**
Step 2: Exploring the Hierarchy of the Timing Paths

The Select Primitive Parents command selects the hierarchical parent of any currently selected primitives or modules, working up through the hierarchy until it gets to the top-level of the design. The parent modules are also selected in the Netlist window.

5. Right-click and select the **Show Hierarchy** command from the popup menu, or press **F6**.

The selected logic, which is associated with the timing paths with the worst delays, is all contained in the `usbEngine1` module. There is a duplicate module, `usbEngine0`, which with identical logic, will have many of the same problems. You will explore these two `usbEngine` blocks further for floorplanning.

![Figure 36: Locating Logic Associated with Timing Paths](image)

6. **Close** the Hierarchy and Schematic windows, by clicking the **X** button on the window tabs.

7. Select timing **Path 1** in the Timing Summary window, or the first path in the list.

8. Right-click and select **Schematic** from the popup menu.

   The timing path logic is shown in the Schematic window as seen in **Figure 38**.
Step 2: Exploring the Hierarchy of the Timing Paths

The nested rectangles shown in the Schematic window, represent the hierarchical structure of the design. With this view, it is easy to identify the various logic modules associated with the critical path logic. For the selected timing path, you can see the logic is contained within the `usbEngine1` block.

9. Click the Unselect All button on the main toolbar menu, or press F12.

10. Select the `RAMB36E1` in the `buffer_fifo` block.

11. Right-click and select Expand/Collapse > Collapse Outside from the popup menu.

   This will collapse all of the modules outside of the selected cell, so that the schematic is only displaying the hierarchy of the selected cell. In this case, the selected cell is the source of the critical timing path.

12. Double-click the outside of the `ADDRARDADDR[15:0]` pin, in the top left of the block RAM, and notice that the logic connected to it is expanded outward.

   **Tip:** Double-clicking outside the pin is the same as selecting the pin and using the Expand Cone > To Primitives command from the popup menu.

13. Select the `infer_fifo.wr_addr_reg[0]` cell, displayed at the top of the newly added logic.

14. Right-click and select Expand Cone > To Primitives from the popup menu, or double click the specified cell.
This expands the logic connected to the selected cell outward to the next connected primitives, as shown in Figure 39. The logic associated with this expansion includes a reset net, a global clock buffer, and logic in the usbEngine block. You can see the block ram is relatively well insulated from the other logic.

15. In the Schematic window, select the Previous schematic button.

You can use the Previous schematic and Next schematic toolbar buttons to switch between prior views displayed in the Schematic window.

16. In the Schematic window tab, click the X button to close the Schematic window.

The input to this block RAM is contained in the usbEngine block. You do not have to consider other hierarchy for floorplanning.

17. Click the Unselect All button, or press F12.

18. In the Netlist window, click the Collapse All button.
Step 3: Highlighting Module Placement and Connectivity

At this point, you have determined that the worst timing delays are found in the `usbEngine` blocks, and examined the logic in the schematic. You will now evaluate the current module placement to understand how the logic was implemented without floorplanning, to see how placement can be improved. This will lead you to a placement strategy you can implement through floorplanning.

1. In the Netlist window, select `usbEngine0` and `usbEngine1`.
2. Right-click and select Highlight Primitives > Cycle Colors from the popup menu.

The primitives in each module are highlighted in a different color as shown in Figure 39. Notice the placement of the primitives within each module, with `usbEngine0` highlighted in one color, and `usbEngine1` highlighted in a different color. Notice the logic is spread out across the device.

Figure 39: Highlight Module Placement
3. In the Device window, click the **Device View Options** button.

4. Uncheck the box next to **Instances** to disable the display of the placed instances in the Device window.

   Various elements of the Device window can be hidden, to make it easier to see important details of the design or device.

   Zoom into the Device window for a closer look, and pan around the window for a better view. Notice the logic from the two `usbEngine` blocks are intermingled. With logic spread out across the device, these two blocks could benefit from floorplanning to improve timing.

5. Enable Instances in the **Device View Options** menu.

6. Click the **Device View Layers** button to hide the menu.

7. From the main toolbar, click the **Unhighlight All** button.

**Visualizing the Connectivity**

You can alleviate routing congestion and timing issues by floorplanning to prevent logic from being placed into critical or congested areas during implementation. Evaluating the design connectivity will help you determine which modules are suitable for floorplanning. For example, logic modules that connect to logic throughout the device may not be suitable for floorplanning, while tightly grouped and self-contained modules are suitable.

8. In the Device window, select the **Show I/O Nets** toolbar button.

   The wires in the Device window in Figure 40 show the connectivity between the I/O ports and placed instances on the device.
9. Click to select one of the I/O Nets.

**TIP:** If you have difficulty selecting a net, check to see that the Select checkbox is enabled for I/O Nets in the Vivado Options dialog box. Refer to “Setting Selection Rules” in the Vivado Design Suite User Guide: Using the Vivado IDE (UG893) for more information.

10. Inspect the net name in the Netlist window.

11. In the Netlist window, click the **Collapse All** button.

12. From the Netlist window, select `usbEngine0` and `usbEngine1`.

13. Right-click and select the **Show Connectivity** command from the popup menu, or press **Ctrl-T**.

   The interface nets that connect `usbEngine0` and `usbEngine1` to the rest of the design are highlighted, as shown in the following figure. You can see that most of the connections go
to the I/O Blocks on the left side of the device. This will help guide your placement during floorplanning.

You will floorplan the `usbEngine` blocks to keep the I/O logic near the I/O blocks not, not specifically to improve timing. However, this could improve timing if the design was failing timing in some way.

![Figure 41: Show usbEngine Connectivity](image)

14. Right-click and select **Show Connectivity** again to select all of the logic objects to which the interface nets connect.

You can use the Show Connectivity command to highlight or select a cone of logic from any source net or logic object.

15. Click the **Unselect All** button or press **F12**.

---

**Step 4: Floorplanning the Design**

With an understanding of the design connectivity, and the problems of the prior placement, you can now begin floorplanning the design. The process of floorplanning begins by grouping some or all of the logic in the design into physical blocks, called Pblocks, and constraining that logic with common constraints. Grouping associated logic together into Pblocks helps you analyze the data flow of the design, limits interconnect lengths, prevents logic migration, and reduces timing delays.
Creating a Pblock results in a `create_pblock` Tcl command that is written to the target constraint file. The constraint defines the assigned logic, specified site ranges, and defined attributes of the Pblock.

**Automatically Creating Pblocks**

1. Select **Tools > Floorplanning > Auto-create Pblocks**.
   
   This opens the Create Pblocks dialog box.
   
   ![Figure 42: Auto-Create Pblocks](image)
   
   In the dialog box, review the options to define the maximum number of Pblocks to create, and specify the minimum Pblock size.

   **TIP:** *If the design has more modules than the maximum number of Pblocks to generate, the Vivado tool creates Pblocks starting with the largest modules.*

2. Click **OK** to create Pblocks for the listed modules.

3. Select the **Floorplanning** layout from the View Layout drop down menu to display the Physical Constraints window.

   The newly created Pblocks, as well as the top-level ROOT, are listed in the Physical Constraints window, as shown in **Figure 43**.

   In the Netlist view, the icon next to the six modules changed from ![icon](image) to ![icon](image), to show that the module has been defined as a Pblock.
Placing Pblocks

4. Select **Tools > Floorplanning > Place Pblocks**.

The Place Pblocks dialog box contains options to define which Pblocks to place, and what the target SLICE utilization for the Pblocks should be.

The size of the Pblock created by the Vivado IDE depends on the Utilization % specified in the Place Pblocks dialog box. Pblocks are sized based upon SLICE logic utilization only, which is useful for estimating the size of the Pblock. However, Pblocks created using the Place Pblocks command will need to be resized later, prior to implementation, in order to insure sufficient resources are available for all logic assigned to the Pblock.
5. Click **OK** to accept the default values, and place all the Pblocks.

The specific placement of the Pblocks is preliminary, and may be different each time the Place Pblocks command is run. You can arrange the Pblocks in the Device window, as shown in **Figure 45**, to untangle the connectivity by selecting and moving them.

6. Select the **Show/Hide I/O Nets** toolbar button to enable the display of connections between the Pblocks and the I/O ports.

7. Open the **Device View Options** menu from the toolbar, and enable the display of Bundle Nets under the Design heading.

The bundle nets and I/O connections help you visualize the connectivity of the design, as seen in **Figure 45**. The preliminary floorplan provides early indications of data flow through the design, and highlights where potential routing congestion could occur. Bundles are color coded to show the number of nets shared between two Pblocks.

To see or change the current definitions, use the **Tools > Options** command to open the Vivado Options dialog box. Select the Colors button on the left side of the dialog box, and select the Bundle Nets category on the right. You can also view the colors defined under the Device View Options menu in the Device window, but you cannot change the colors.
8. In the Device window, select a bundle net.

Examine the list of nets in the Bundle Net Properties window. These are the nets contained within the bundle, between the two Pblocks.

![Figure 46: Bundle Net Properties](image)

**Finding the Global Clock**

Effective floorplanning is often dependent on proper placement of the synchronous elements from different clock domains. You can highlight clock domains to visualize connectivity and to ensure Pblocks are properly placed relative to clock regions. In this step, you will use the Find command to locate and select the USB Global Clock (\texttt{usbClk}) to highlight the clocked Pblocks.

9. Select \textbf{Edit > Find}, or type \textbf{Control-F}.

This opens the Find dialog box to search through the design for specific types of objects.

10. In the Find dialog box, set the following options:

   - \textbf{Find: Nets}
   - \textbf{Criteria: Type, is, Global Clock}

11. Ensure the \textbf{Unique Nets Only} checkbox is enabled, and click \textbf{OK}.

The Find results window opens at the bottom of the Vivado IDE.

12. Click the \textbf{Flat Pins} column header \textit{twice} to sort the Find Results window.

13. In the Find Results window list, select \texttt{clkgen/usbClk_o}.

Notice the highlighted net connecting the two usbEngine Pblocks in the Device window. This clock is limited to these two blocks.
Moving Pblocks

With a preliminary Pblock placement, and the net bundles displaying the data flow between Pblocks, you can now begin to clean up the placement to meet some of the design requirements. In this case, you will be locating Pblocks to reduce the timing delays in the usbEngine blocks.

Since you are only concerned with the usbEngine blocks at this time, select and clear the other Pblocks in the Physical Constraints window.


15. Right click and select the Delete command.

This will delete the selected Pblocks, leaving only the usbEngine0 and usbEngine1 Pblocks. The other Pblocks were used to help you understand the data flow through the different modules of the design, and are not needed for floorplanning.

16. Select pblock_usbEngine1 in the Physical Constraints window, or select it directly in the Device window.

17. Move the Pblock by dragging it to a new location near the top of the Device window.

The placement you choose should seem appropriate when you are moving the Pblock, because the guide wires show connection to the upper I/O ports. To reduce delay, you should place the Pblock near the I/O ports it is connected to.

When you place the Pblock in its new location, the Move dialog box opens to prompt you to select which resources in the new location should be enabled for use by the Pblock.

18. Select OK to include all resource types.
19. In some cases, with elements of the Pblock previously placed, the Choose LOC Mode dialog box will prompt you to decide how to handle existing cell placement, or LOC constraints, as shown in Figure 49.

The choices are:

- Leave cells where they are currently placed.
- Unplace cells that fall outside the new Pblock area.
- Unplace only unfixed cells that fall outside the new Pblock area.
- Unplace all cells in the Pblock.

20. Select Delete all location constraints of the pblock and click OK.

This will unplace any currently placed cells assigned to the Pblock, clearing the LOC constraint on the cell, allowing it to be relocated to the new area covered by the Pblock.

![Figure 48: Choose LOC Mode](image)

**IMPORTANT:** Both fixed and unfixed cells are placed.

Fixed and Unfixed are terms that apply to placed cells, and describe the way the Vivado tool views placed cells in the design.

- **Fixed cells have been placed by you, or another designer, or have been imported from a constraints file. These placed cells are therefore considered fixed by the Vivado Design Suite, and will not be moved unless directed to do so.**

- **Unfixed cells have been placed by the Vivado tool in implementation, during the place_design command, or one of optimization commands. These placed cells are considered unfixed, or loosely placed, by the Vivado Design Suite, and can be moved as needed in design iterations.**

21. Select **pblock_usbEngine0** and move it toward the bottom of the Device window as guided by the displayed connections.

Repeat the choices for the Move dialog box and the Choose Loc Mode dialog box as needed.
Resizing Pblocks

With the `usbEngine` blocks placed approximately where you would like them, you must now resize the Pblocks to include the required resources.

22. Select a Pblock, and click the **Statistics** tab in the Pblock Properties window.

   The Statistics tab of the Pblock Properties window shows the different logic resources available on the device in the area covered by the Pblock, and shows the amount of resources required by the logic assigned to the Pblock, and shows the current utilization %.

   The red values, as shown in **Figure 49** indicate insufficient resources available in the Pblock area to meet the requirements of the design logic assigned to the Pblock. In this case, the Pblock will need to be resized to increase its resources.

   ![Figure 49: Pblock Properties - Statistics](image)

23. Select `pblock_usbEngine0` in the Device window.

24. Right click and select the **Set Pblock Size** command from the popup menu.

25. Draw a rectangle to cover Clock Region X0Y1, on the left side the Device window.

   If you size the Pblock correctly to cover the clock region borders, you'll be prompted to confirm the clock region as the Pblock range. If the Pblock does not encompass the whole clock region, you'll be prompted to accept the logic types contained in the Pblock area.
26. Click **OK** to accept either the clock region, or the included logic.

At this time you can review the Statistics of the Pblock Properties window, and see that `pblock_usbEngine0` is not large enough to include the required logic resources, as shown in **Figure 50**.

27. Scroll down to the bottom of the Pblock Properties window and examine the different types of information provided.

28. Stretch the bottom of the Pblock all way to the bottom edge of the device and click **OK** to accept the Clock Region as the Pblock range.

Expand the Pblock rectangle to cover both Clock Region X0Y0 and X0Y1, as shown in the figure below. You will be prompted to accept the Clock Region and not the individual logic site types.
29. Select \texttt{pblock\_usbEngine1} and resize the Pblock to cover both Clock Region X0Y2 and X0Y3.

The end result is shown in \textit{Figure 53}. 
With new constraints created for the `usbEngine0` and `usbEngine1` Pblock placement, you must save the design constraints. It is a good idea to save the constraints into a new constraints set so that you can keep track of different design strategies. This allows you to revert to an older design strategy if needed, or develop and implement different strategies for a single design.

30. Select the **File > Save Constraints As** command.

   This opens the `Save Design As` dialog box as shown in **Figure 54**.
Step 4: Floorplanning the Design

31. Save the new constraint set as `constrs_usb_fp`, for the usbEngine floorplan.

32. Click OK.

The `top_full.xdc` file is copied to the new constraint set, and the constraints are updated as needed to reflect the current state of the design.

33. In the Sources window, expand the `constrs_usb_fp` constraint set, and double-click the `top_full.xdc` file to open it in the Text Editor.

Scroll through the open constraint file to examine the constraints. Toward the bottom of the file you will see the new Pblock constraints.

When you save the constraint set, the Synthesized Netlist and Implemented Design may go out of date, as reflected in the project status bar. In some cases this may be the result of small changes that you know do not actually affect synthesis or implementation. When this happens, you can force the design into an up-to-date state, as shown in Figure 55.

The floorplanning of the usbEngine blocks does mean that the implemented design is no longer up-to-date. However, the changes you made do not affect the Synthesized Netlist. Force the design into an up-to-date state, and then rerun implementation.

![Figure 54: Saving a new Constraint Set](image)

![Figure 55: Force Up-to-date](image)
34. Click the more info link in the project status bar, and click the Force up-to-date link.

The project status bar is updated to reflect the up-to-date condition for the design.

---

**Step 5: Running Implementation with the Floorplan**

With the floorplan of the usbEngine blocks completed. You are now ready to rerun implementation.

1. Select the Design Runs window to make it active.
2. Right click and select the Create Runs command from the popup menu.
3. Select Implementation as the type of runs to create, and click Next.
4. Select synth_1 for the Synthesized netlist, constrs_usb_fp for the Constraint set, and xc7k70tfbg676-2 for the Part.
5. Click Next.

This opens the Choose Implementation Strategies dialog box.

![Create New Runs](image)

**Figure 56: Create Floorplan Implementation Run**

6. In the Name field, enter impl_usb_fp, for usbEngine Floorplan.
7. For Strategy select HighEffortPhySynth.
8. Select Make Active.
9. Click Next.
10. In the Launch Options dialog box, select Launch Runs on Local Host and click Next.
11. On the Create New Runs Summary, check that the settings are correct, and click Finish.
This launches the implementation run with the `usbEngine` blocks placed according to your floorplan.

The Implementation Completed dialog box displays when the run is finished.

12. Select **Open Implemented Design** in the Implementation Completed dialog, and click **OK**.

The design has been implemented with your floorplan for the `usbEngine` blocks. Which you can see by highlighting the block primitives as you did earlier in this tutorial.

13. In the Netlist window, select `usbEngine0` and `usbEngine1`.

14. Right click and select **Highlight Primitives > Cycle Colors** from the popup menu.

![Figure 57: Implemented Floorplan](image)

**Conclusion**

In this tutorial, you used the Vivado IDE to explore and analyze the synthesized design and targeted device prior to running the implementation tools. This enabled you to find potential design issues and errors early in the design cycle, rather than discovering issues during implementation. In addition, you used the graphical presentation of design resource estimates, design rule violations, timing estimation, constraints, and connectivity to help you understand your design and any areas with potential issues.
After running the design through the synthesis and implementation tools, you:

- Viewed implementation results and examined timing results.
- Analyzed critical path objects in the schematic, and selected the parent modules of those path objects.
- Highlighted module placement and displayed the connectivity of the modules using the Show Connectivity command.
- Analyzed placement and routing of critical timing paths.
- Floorplanned possible placements based on the hierarchy of the design.
- Implemented the floorplanned design.