Vivado Design Suite Tutorial

Using Constraints

UG945 (v2014.3) November 7, 2014

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

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/07/2014	2014.3	Fixed document to correctly show Lab 2. Made editorial and terminology corrections. Updated several screenshots.
10/01/2014	2014.3	Validated with release.
06/04/2014	2014.2	Validated with release.
04/02/2014	2014.1	Validated with release.

Table of Contents

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

Overview

This tutorial is comprised of two labs that demonstrate aspects of constraining a design in the Vivado® Design Suite. The constraints format supported by the Vivado Design Suite is called Xilinx® Design Constraints (XDC), which is a combination of the industry standard Synopsys® Design Constraints and proprietary Xilinx constraints.

VIDEO: *You can also learn more about defining constraints in the Vivado Design Suite by viewing the quick take video at* Vivado Design Constraints Overview

TRAINING: *Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use these links to explore related courses:*

- Essentials of FPGA Design
- Vivado Design Suite Static Timing Analysis and Xilinx Design Constraints
- Vivado Advanced Tools and Techniques

XDCs are not just simple strings; they are Tcl commands that the Vivado Tcl interpreter sequentially reads and parses. You can enter design constraints in several ways at different points in the design flow. You can store XDCs in one or more files that can be added to a constraint set in Vivado Project Mode, or read the same files directly into memory using the `read_xdc` command in Non-Project mode. For more information on Project and Non-Project modes, refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892). With a design open in Vivado tools, you can also type constraints as commands directly in the Tcl console when working in the Vivado IDE or at the Tcl command prompt when working outside of the IDE. This is particularly powerful for defining, validating, and debugging new constraints interactively in the design.

The Vivado Design Suite synthesis and implementation tools are timing driven. Having accurate and correct timing constraints is vital for meeting design goals and ensuring correct operation. Because the Vivado tools are timing driven, it is important to fully constrain a design, but not over-constrain, or under-constrain it. Over-constraining a design can lead to long run-times and sub-optimal results because the tool can struggle with unrealistic design objectives. Under-constraining a design can cause

the Vivado tools to perform unnecessary optimizations, such as examining paths with multicycle delays or false paths, and prevent focus on the real critical paths.

This tutorial discusses different methods for defining and applying design constraints.

Tutorial Design Description

The sample design used throughout this tutorial consists of a small design called `project_cpu_netlist`. There is a top-level EDIF netlist source file, as well as an XDC constraints file.

The design targets an XC7K70T device. A small design is used to allow the tutorial to be run with minimal hardware requirements and to enable timely completion of the tutorial, as well as to minimize the data size.

Hardware and Software Requirements

This tutorial requires that the 2014.3 Vivado Design Suite software release or later is installed. The following partial list describes the operating systems that the Vivado Design Suite supports on x86 and x86-64 processor architectures:

Microsoft Windows Support:

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

Linux Support:

- Red Hat Enterprise Workstation 5.8 - 5.10 (32-bit and 64-bit)
- Red Hat Enterprise Workstation 6.4 and 6.5 (32-bit and 64-bit)
- SUSE Linux Enterprise 11.1 - 11.2 (32-bit and 64-bit)
- Cent OS 6.4 and 6.5 (64-bit)
- Ubuntu Linux 14.04 LTS (64-bit)

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

Preparing the Tutorial Design Files

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

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

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

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

You can also extract the provided ZIP file at any time to restore the files to their starting condition.

> ***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.*

Introduction

In this lab, you will learn two methods of creating constraints for a design. You will be using the Kintex-7 CPU Netlist example design that is included in the Vivado IDE.

Step 1: Opening the Example Project

Open Vivado IDE:

On Linux:

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

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

2. Launch the Vivado IDE: **vivado**

On Windows,

1. Launch the Vivado Design Suite IDE:

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

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

   Note: *As an alternative, click the* ***Vivado 2014.x*** *Desktop icon to start the Vivado IDE.*

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

2. From the Getting Started page, click **Open Example Project** and select the **7-Series Kintex > CPU (Synthesized)** design, as shown in Figure 1.

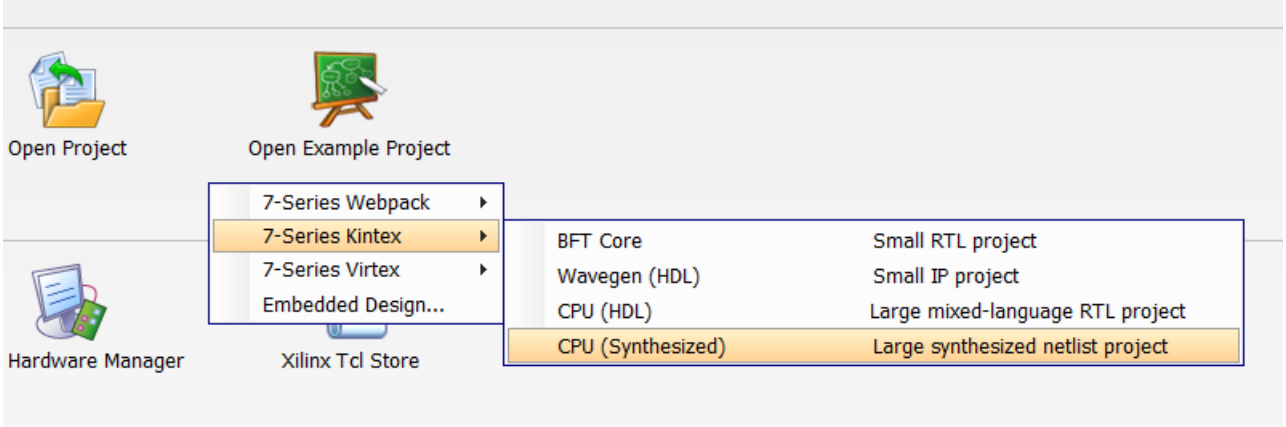


Figure 1: Open Example Project

A dialog box appears stating that the project is read-only**.**

3. Click **Save Project As** to specify a project name and location, as shown in Figure 2.

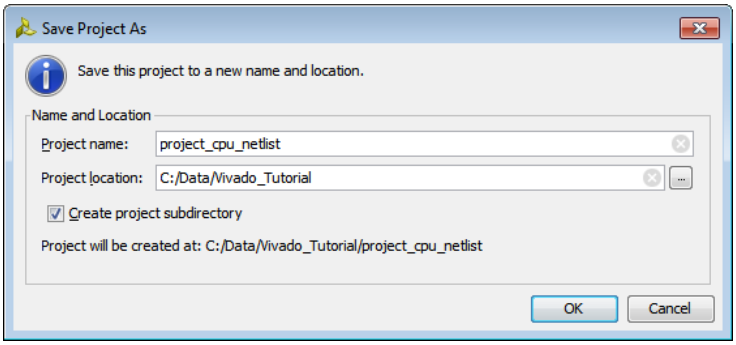


Figure 2: Save Project As

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

The Vivado IDE displays the default view of the opened project, as shown in Figure 3.

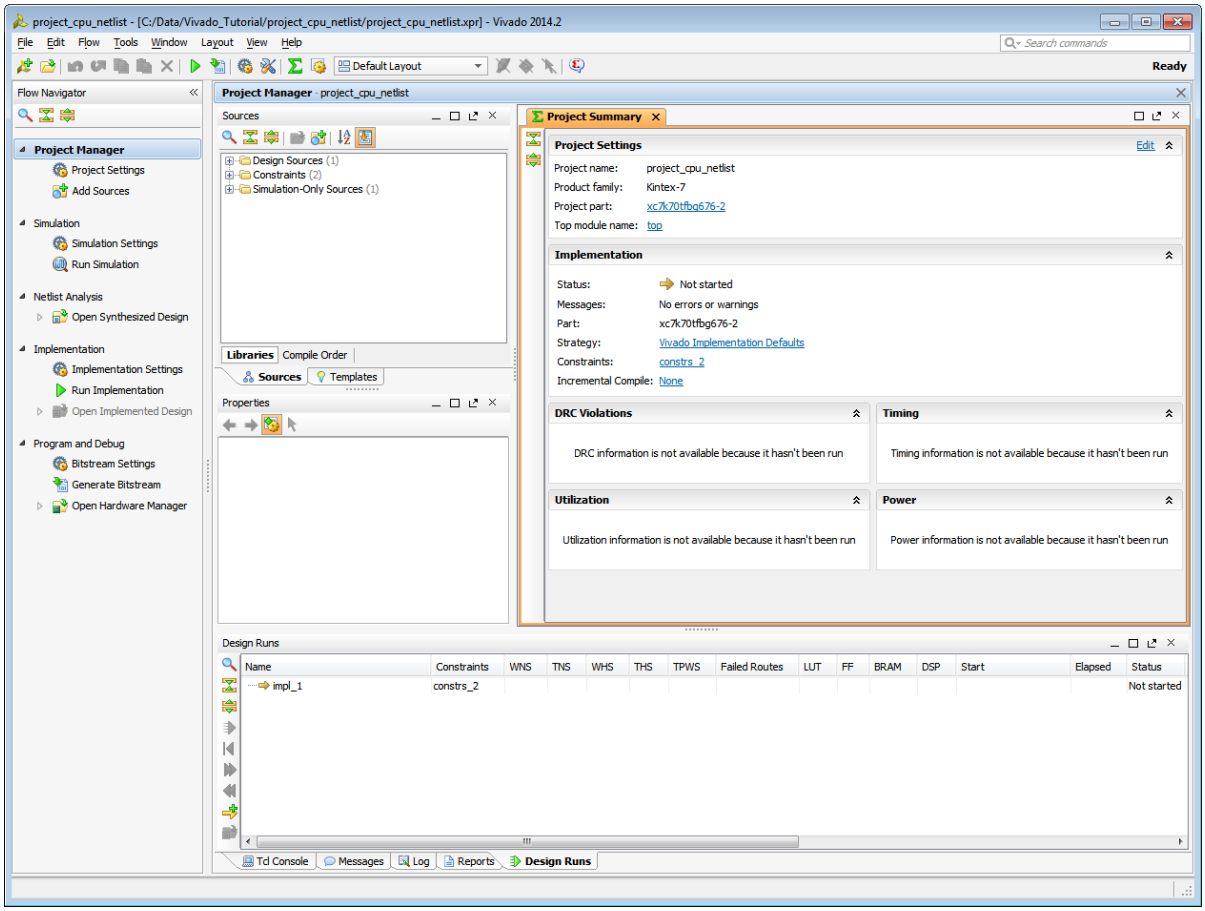


Figure 3: Project Summary Window

Step 2: Defining Constraint Sets and Files

Start by creating a new constraint set and adding an empty XDC constraints file to it. The sample design already contains two constraint sets, but you do not use them for this lab.

1. From the Flow Navigator, select **Add Sources** in the Project Manager section.
2. From the list displayed in the Add Sources dialog box, select **Add or Create Constraints** and click **Next**.
3. From the Add or Create Constraints dialog box, use the Specify Constraint Set: drop down menu to select **Create Constraint Set** as shown in Figure 4.

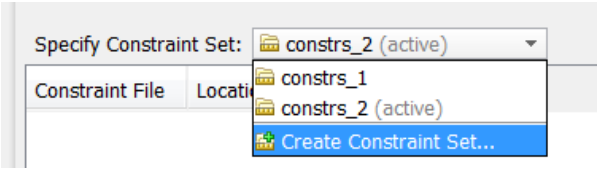


Figure 4: Create Constraint Set

4. In the Create Constraint Set Name dialog box, specify the constraint set name as **`lab1`** and click **OK**.
5. Enable the **Make active** checkbox.
6. Select **Create File** to add a new XDC file to the project. Enter **`timing`** as the file name. Leave the file as **<Local to Project>**, and click **OK**. See Figure 5.

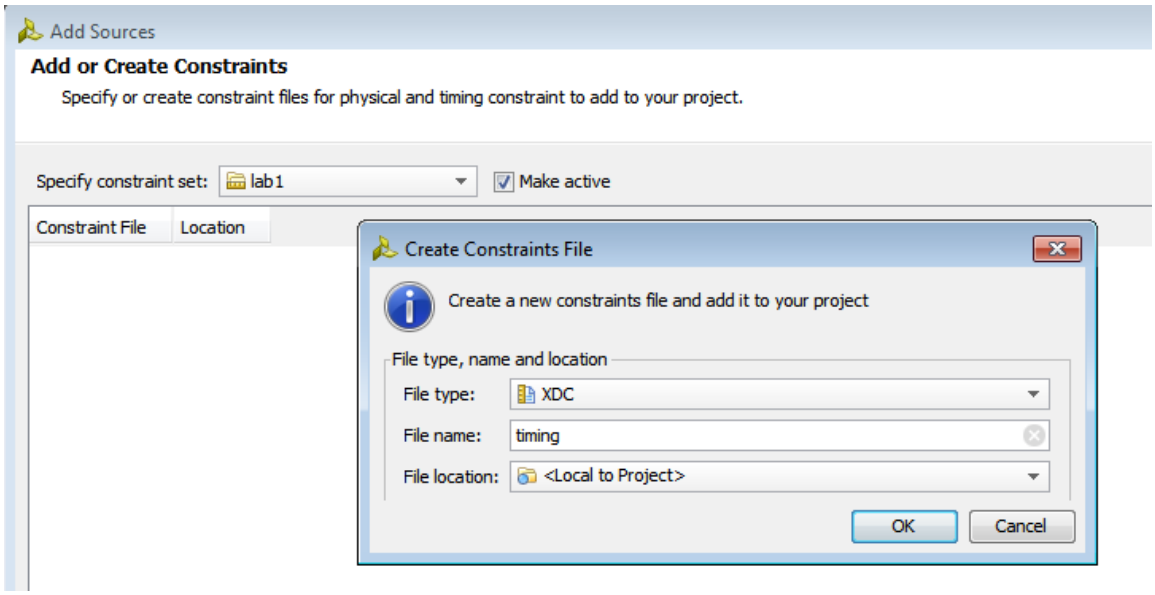


Figure 5: Constraints File Name

The `timing.xdc` file is added to the `lab1` constraint set.

7. Select **Finish** to complete the creation of the new constraint set and XDC file.

   You should see the new constraint set and XDC file in the Sources window as shown in Figure 6. The constraint set is made active as you directed when you created it.

8. Set the `timing.xdc` file as the target XDC file. All constraints added to the design will be saved in the target XDC file. In the Sources window, right click the `target.xdc` and choose **Set as Target Constraint File** from the context menu. See Figure 6.

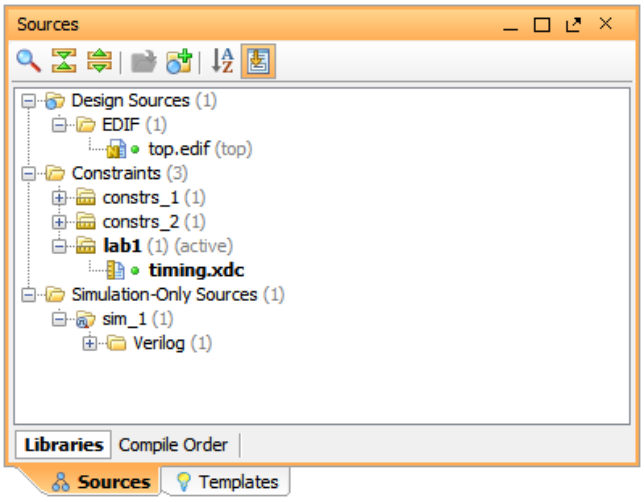


Figure 6: Sources window

Step 3: Creating Timing Constraints

Open the synthesized design and use the timing constraints wizard. The timing constraints wizard analyzes the gate level netlist and finds missing constraints. Use the timing constraints wizard to generate constraints for this design.

1. From the Flow Navigator, select **Open Synthesized Design**.
2. Select **Constraints Wizard** from the Flow Navigator under the Synthesized Design section. The introduction page of the Wizard will appear. This page describes the types of constraints that the wizard will create.
3. After reading the page, click **Next** to continue.
4. The primary clocks page of the Timing Constraints wizard displays all the clock roots with a missing clock definition. The wizard detected five missing clock constraints that are needed to time logical paths, and four missing clock constraints that are only needed to verify pulse width and minimum or maximum period requirements. The second category of clocks is optional and is shown in the Constraints for Pulse Width Check Only table. For this example, do not add these last constraints. Use Table 1 to fill in the periods of the five missing primary clocks in the design.

Table 1: Values to use on the Primary Input Page of the Wizard

Primary Clock	Period (ns)
mgtEngine/ROCKETIO_WRAPPER_i/gt0_ROCKETIO_WRAPPER_TILE_i/GT0_TXOUTCLK_OUT	12.8
mgtEngine/ROCKETIO_WRAPPER_i/gt0_ROCKETIO_WRAPPER_TILE_i/GT2_TXOUTCLK_OUT	12.8
mgtEngine/ROCKETIO_WRAPPER_i/gt0_ROCKETIO_WRAPPER_TILE_i/GT4_TXOUTCLK_OUT	12.8
mgtEngine/ROCKETIO_WRAPPER_i/gt0_ROCKETIO_WRAPPER_TILE_i/GT6_TXOUTCLK_OUT	12.8
sysClk	10.0

Each row of the wizard is a missing constraint. If you would prefer not to enter the constraint, you can uncheck the box next to the constraint. If you would like more information about how the wizard is finding these missing constraints, there is a reference button in the lower left-hand corner of the wizard. The reference pages are context specific and contain more information about the topologies the wizard is looking for and an explanation as to why the constraint is being suggested.

5. The completed page should look like Figure 7. Click **Next** to continue.

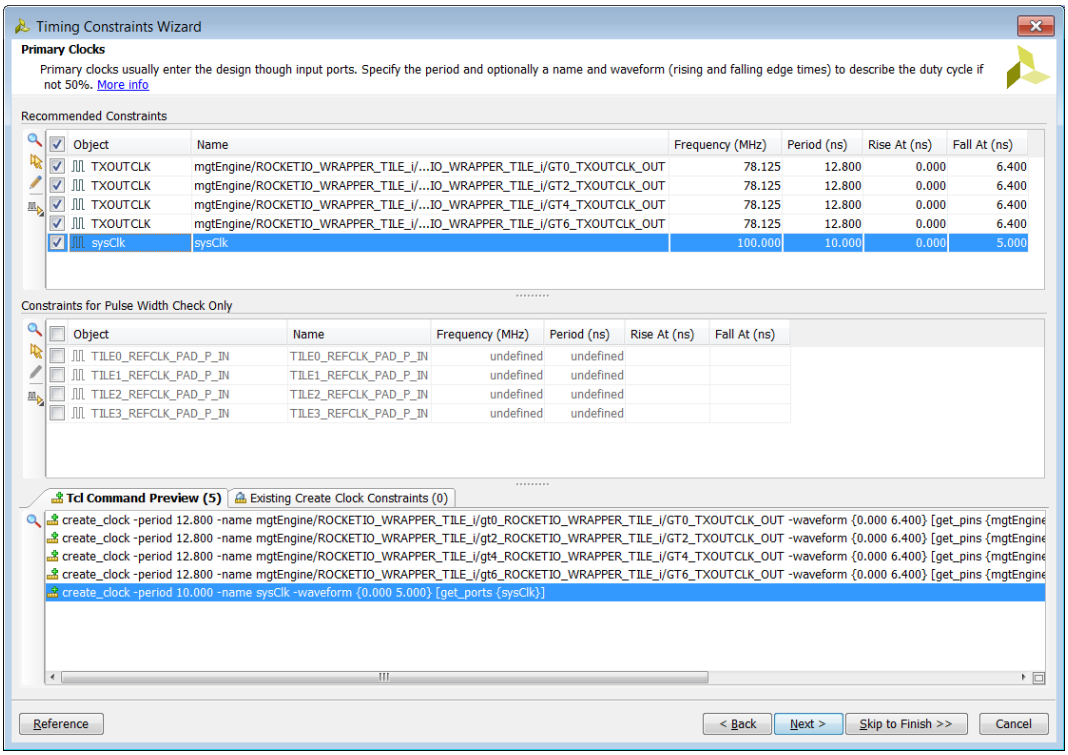


Figure 7: Completed primary clocks page of the timing constraints wizard

6. The primary clock constraints have been added to the design. Next, the wizard looks for generated clocks. Generated clocks are derived from primary clocks in the FPGA fabric. A good example would be a binary counter used to create a divided clock. In this design, the wizard determined that there are no unconstrained generated clocks.

7. Click **Next** to continue. A forwarded clock is a generated clock on a primary output port of the FPGA. These are commonly used for source synchronous buses when the capture clock travels with the data. The wizard has also determined that there are no unconstrained forwarded clocks in the design.
8. Click **Next** to continue. MMCM or PLL feedback delay outside the FPGA is used to compute the clock delay compensation in the timing reports. The wizard did not find any unconstrained MMCM external feedback delays in the design.
9. Click **Next** to continue.

   Figure 8 shows the inputs page of the timing constraints wizard. There are three sections to the page. In section A, you can see all the input ports that are missing input delay constraints in the design. In this table, you select the timing template you would like to use to constrain the input. In section B, you provide the delay values for the template. This section will change depending on the template chosen in section A. Finally, in section C there are three tabs. One previews the Tcl commands that will be used to constrain the design. A second tab shows input delay constraints that already exist in the design. The third tab displays the waveform associated with the template.

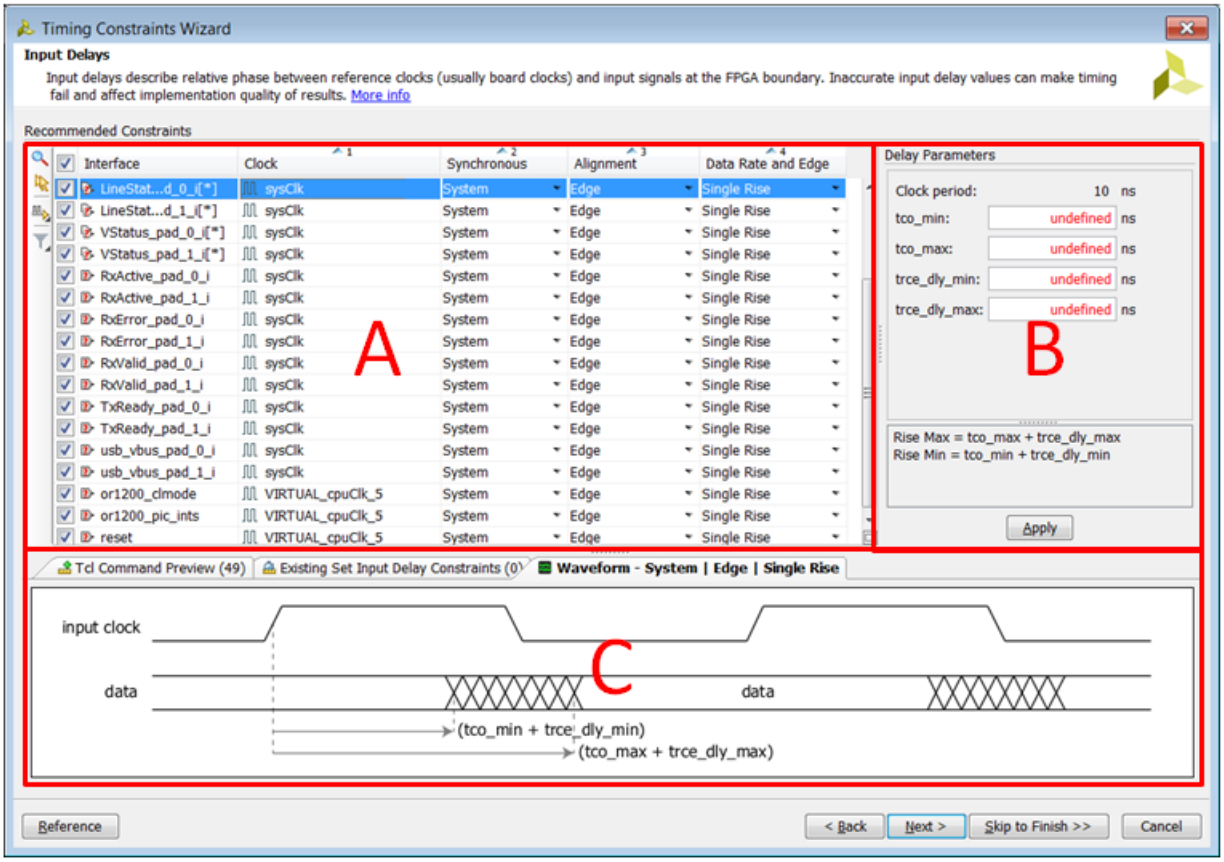


Figure 8: Inputs page of timing constraints wizard showing the major sections

10. Fill out the form based on Table 2.
11. Skip entering a few constraints by unchecking the box to the left of the constraint.

    In this particular case, you false path these constraints later.

    The blocks of colored rows in Table 2 can all be entered at the same time by selecting multiple rows in the wizard and then entering the values once. Some inputs are constrained relative to virtual clocks, since they are captured by an internal generated clock with a waveform different than the board clock. In this case, the wizard creates a virtual clock with the same frequency and waveform as the internal clock, and recommends a constraint relative to the virtual clock.

Table 2: Input Constraint Values

Interface	Clock	Synchronous	Alignment	Data Rate and Edge	tco_min	tco_max	trce_dly_min	trce_dly_max
GTPRESET_IN	mgtEngine/...	System	Edge	Single Rise	Uncheck constraint – will false path later			
GTPRESET_IN	mgtEngine/...	System	Edge	Single Rise	Uncheck constraint – will false path later			
GTPRESET_IN	mgtEngine/...	System	Edge	Single Rise	Uncheck constraint – will false path later			
GTPRESET_IN	mgtEngine/...	System	Edge	Single Rise	Uncheck constraint – will false path later			
DataIn_pad_0_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
DataIn_pad_1_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
LineState_pad_0_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
LineState_pad_1_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
VStatus_pad_0_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
VStatus_pad_1_i[*]	sysClk	System	Edge	Single Rise	1	2	1	1
RxActive_pad_0_i	sysClk	System	Edge	Single Rise	1	2	1	1
RxActive_pad_1_i	sysClk	System	Edge	Single Rise	1	2	1	1
Rx_Error_pad_0_i	sysClk	System	Edge	Single Rise	1	2	1	1
Rx_Error_pad_1_i	sysClk	System	Edge	Single Rise	1	2	1	1
Rx_Valid_pad_0_i	sysClk	System	Edge	Single Rise	1	2	1	1
Rx_Valid_pad_1_i	sysClk	System	Edge	Single Rise	1	2	1	1
TxReady_pad_0_i	sysClk	System	Edge	Single Rise	1	2	1	1
TxReady_pad_1_i	sysClk	System	Edge	Single Rise	1	2	1	1
usb_vbus_paf_0_i	sysClk	System	Edge	Single Rise	1	2	1	1
usb_vbus_paf_1_i	sysClk	System	Edge	Single Rise	1	2	1	1
or1200_clmode	VIRTUAL_cpuClk_5	System	Edge	Single Rise	0.1	2.5	0.1	0.2
or1200_pic_ints	VIRTUAL_cpuClk_5	System	Edge	Single Rise	0.1	2.5	0.1	0.2
reset	VIRTUAL_cpuCLK_5	System	Edge	Single Rise	0.1	2.5	0.1	0.2

12. Figure 9 shows the completed input delay page. Note the four constraints being skipped.

13. When you have successfully entered all the input constraint values, click **Next**.

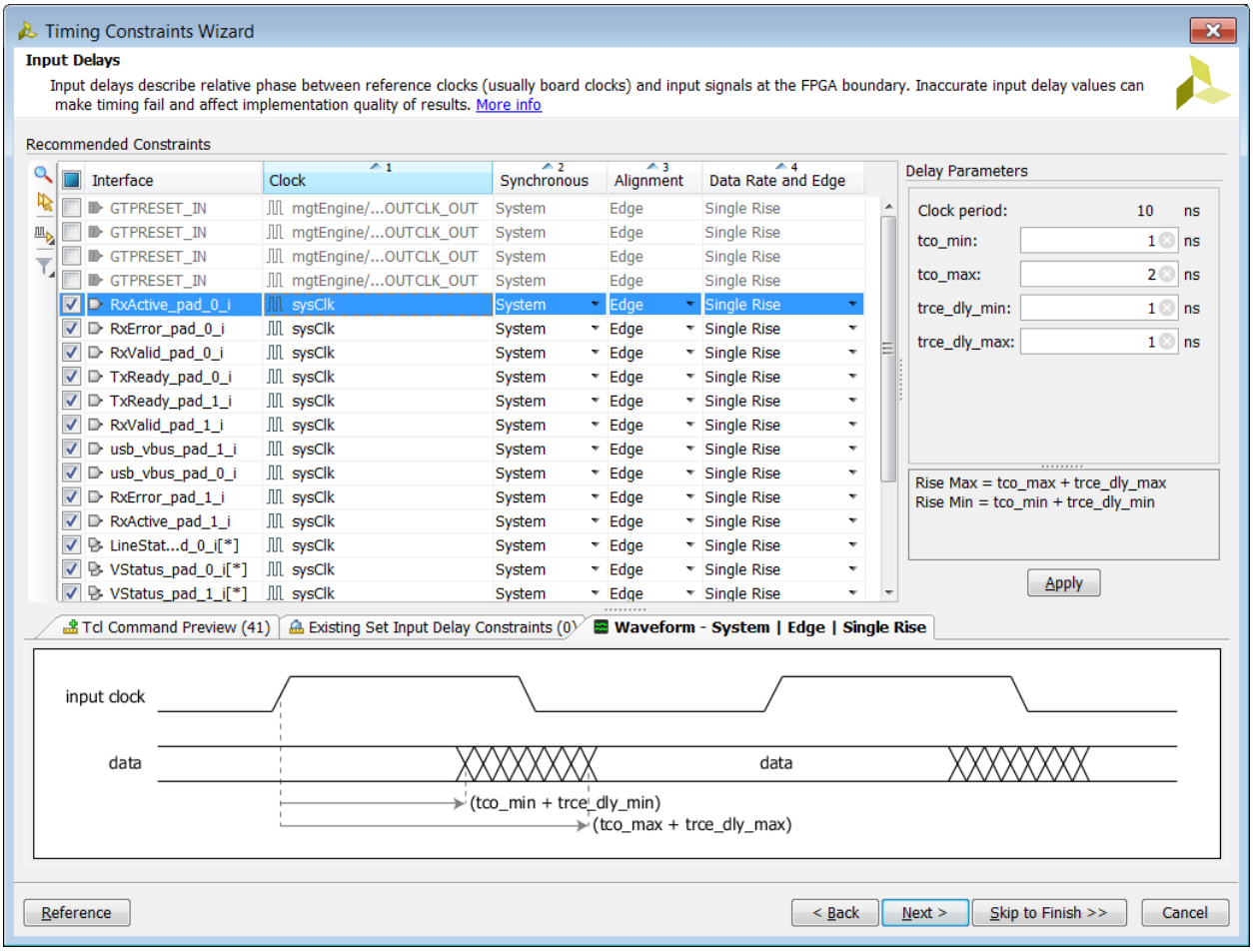


Figure 9: Completed Inputs Page of the Timing Constraints Wizard

The output page of the wizard displays all the outputs that are unconstrained in the design. The page layout is very similar to the inputs page.

14. Select the template in section A, enter the values in section B, and observe the Tcl commands, existing output delays, and the template specific waveform in section C. Use Table 3 below to constrain all the outputs. You can select multiple lines in the wizard at once and edit several entries at the same time.

    Table 3 is shaded to indicate which groups of signals can be entered in this manner.

Table 3 : Output Constraint Values

Interface	Clock	Synchronous	Alignment	Data Rate and Edge	tsu	thd	trce_dly_max	trce_dly_min
OpMode_pad_0_o[*]	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
OpMode_pad_1_o[*]	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
VControl_pad_0_o[*]	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
VControl_pad_0_o[*]	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
SuspendM_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
SuspendM_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
TermSel_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
TermSel_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
TxValid_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
TxValid_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
VControl_Load_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
VControl_Load_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
XcvSelect_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
XcvSelect_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
phy_rst_pad_0_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
phy_rst_pad_1_o	sysClk	System	Setup/Hold	Single Rise	1.0	0.1	0.1	0.1
DataOut_pad_0_o[*]	VIRTUAL_wbClk_4	System	Setup/Hold	Single Rise	2.1	0.6	0.1	0.0
DataOut_pad_1_o[*]	VIRTUAL_wbClk_4	System	Setup/Hold	Single Rise	2.1	0.6	0.1	0.0
Or1200_pm_out[*]	VIRTUAL_wbClk_4	System	Setup/Hold	Single Rise	2.1	0.6	0.1	0.0

15. Click **Next** to continue.
16. The wizard then looks for any unconstrained combinational paths through the design. A combinational path is a path that traverses the FPGA without being captured by any sequential elements. This design does not contain any combinational paths. Click **Next** to continue.
17. Physically exclusive clock groups are clocks that do not exist in the design at the same time. There are no unconstrained physically exclusive clock groups in this design. Click **Next** to continue.
18. Logically exclusive clocks with no interaction are clocks that are active at the same time except on shared clock tree sections. Then these clocks do not have logical paths between each other and outside the shared sections, they are logically exclusive. There are no unconstrained logically exclusive clock groups with no interaction in the design. Click **Next** to continue.
19. Logically exclusive clocks with interaction are clocks that are active at the same time except on shared clock tree sections. When these clocks have logical paths between each other, only the clocks limited to the shared clock tree sections are logically exclusive and are therefore constrained differently than the logically exclusive clock with no interaction. There are no unconstrained logically exclusive clock groups with interaction in the design. Click **Next** to continue.

20. The Asynchronous Clock Domain Crossings page recommends constraints for safe clock domain crossings. This design does not contain any unconstrained clock domain crossings. Click **Next** to continue.
21. Figure 10 shows the final page of the timing constraints wizard. All the constraints that were generated by the wizard can be viewed by clicking the links. If you would like to run any reports once the wizard is finished, you can select them.

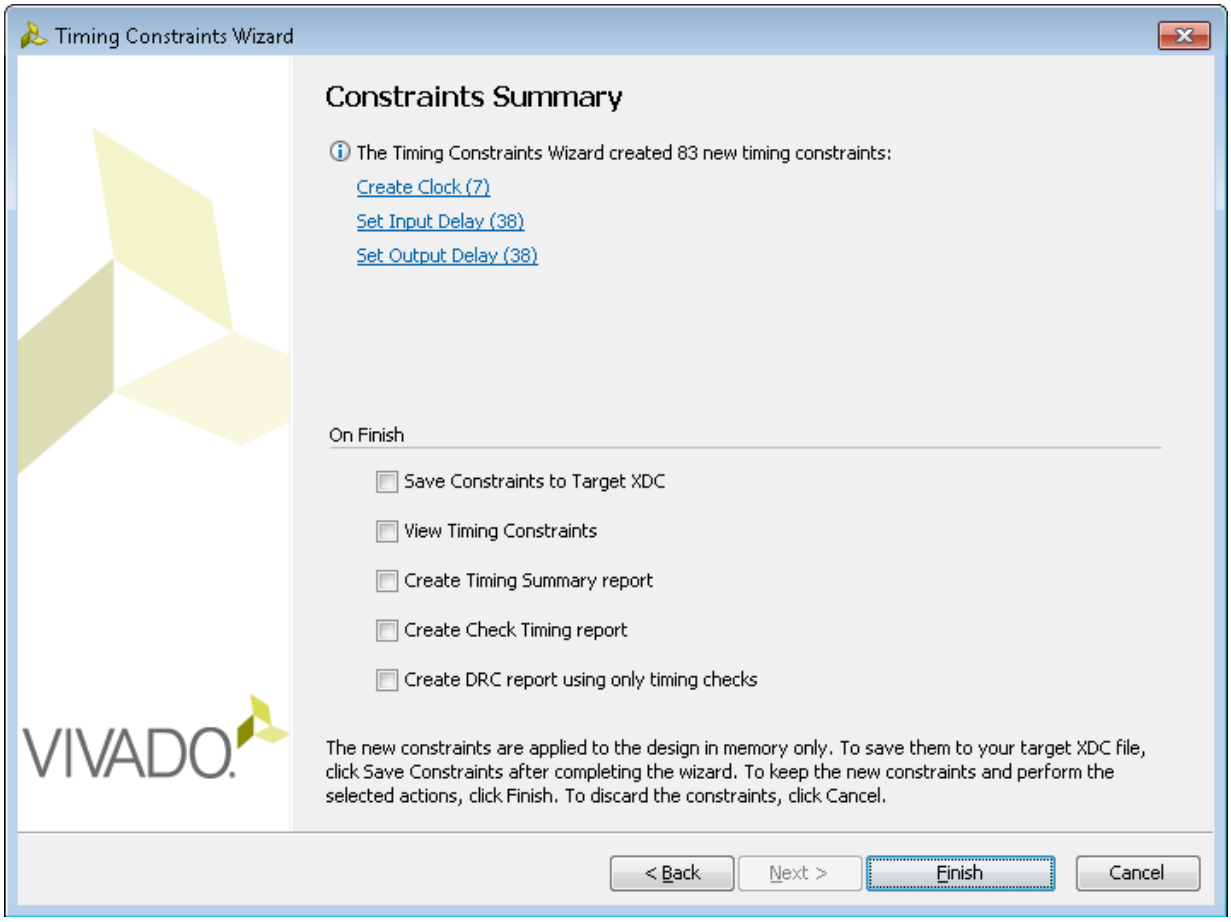


Figure 10: Timing Constraints Wizard Summary Page

22. Click **Finish** to complete the timing constraints wizard.

23. The timing constraints wizard creates all timing constraints in-memory. To save the constraints to disk, you must click the save icon from the main Vivado toolbar as shown in Figure 11.

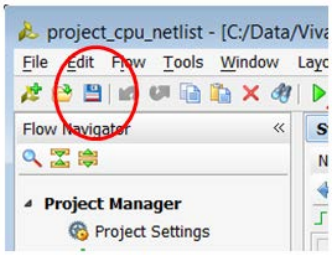


Figure 11: Save Constraints to Disk

Important! *You must use the* ***Save Constraints*** *command to save any constraint changes to the* `timing.xdc` *file.*

Step 4: Using the Constraints Editor

1. Select **Edit Timing Constraints** from the Flow Navigator under the Synthesized Design section. The Vivado IDE displays the Timing Constraints window.

   There are three sections to the Timing Constraints window:

   - **Constraints tree view** is labeled section "A" in Figure 12. This section displays standard timing constraints, grouped by category. Double-clicking a constraint in this section opens a form to help you define the selected constraint.
   - **Constraints Spreadsheet** is labeled as section "B" in Figure 12. This section displays timing constraints of the type currently selected in the Constraints tree view. If you prefer, you can use this to directly define or edit constraints instead of using the Constraints wizard.
   - **All Constraints** is labeled as section "C" in Figure 12, this section displays all the timing constraints that currently exist in the design.

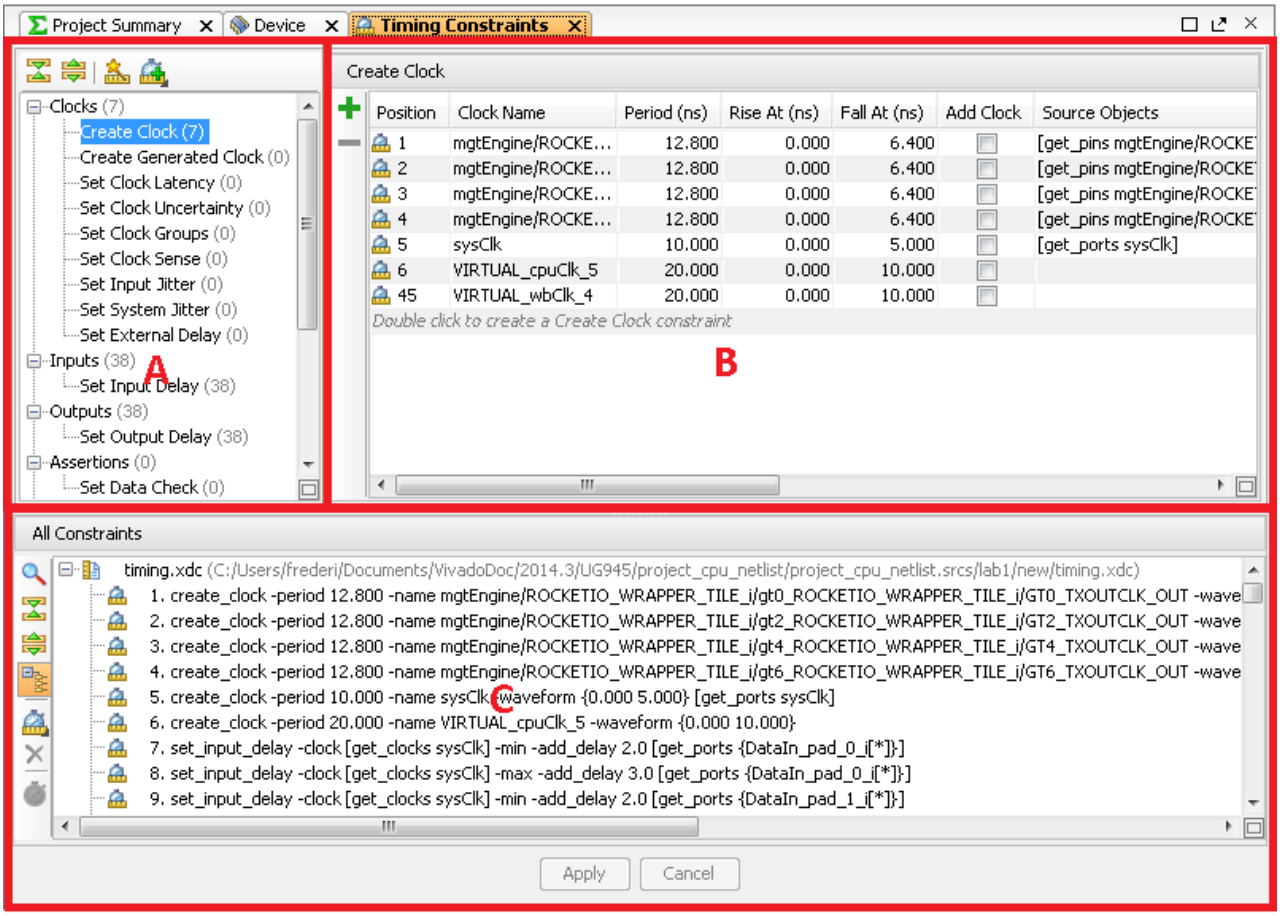


Figure 12: Timing Constraints window

2. The timing constraints wizard identifies missing clocks, I/O delays, and clock domain crossings exceptions, but it does not handle general timing exceptions. You are going to use the timing constraints editor to create the exceptions that exist in this design.
3. First, you are going to set a false path on the `GTPRESET_IN` input. This is the input that you unchecked on the input delay page of the timing constraints wizard.

4. In the Constraints Tree view, scroll down to the Exceptions section.
5. Double-click **Set False Path** under the Exceptions category of the tree as shown in Figure 13.

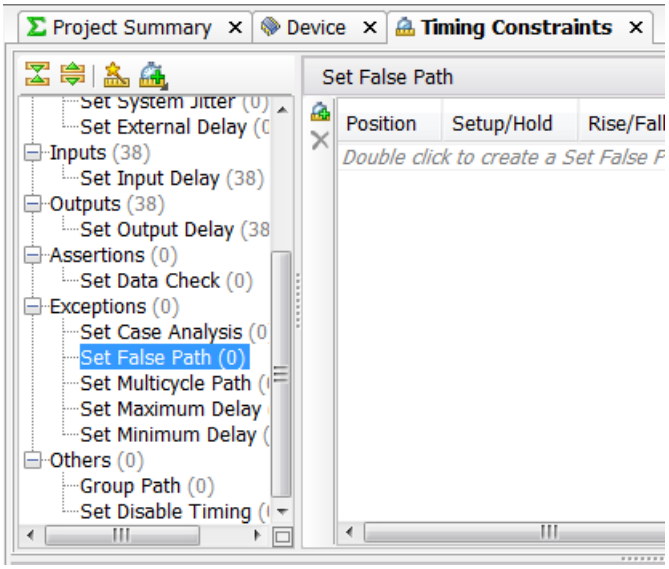


Figure 13: Set False Path Section in the Constraints Editor

6. In the Set False Path window, to the right of the **From** text box, click the  button as shown in Figure 14.

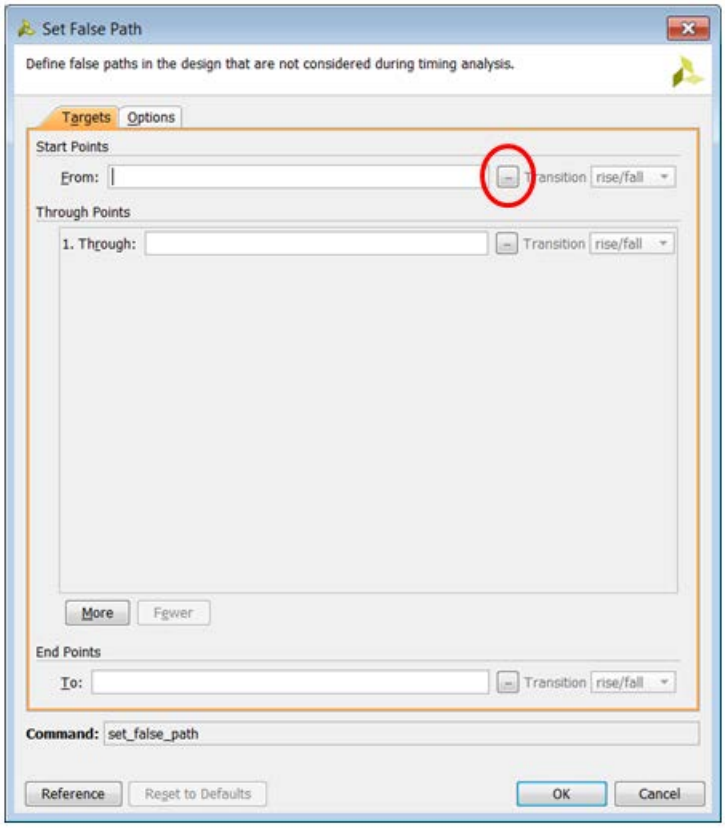


Figure 14: From Box

7. A Choose Start Points window will appear. In this window change the pull-down **Find names of type** to **ports** as shown in Figure 15.
8. In the with pattern text box, enter **`GTPRESET_IN*.`**
9. Click the **Find** button.
10. Select **GTPRESET_IN** in the find results text box and press the right arrow to move it to the selected names text box.

    Notice that the Command field displayed at the bottom of the dialog box changes as you perform these different actions. The `get_ports` command changes to:

    ```
    get_ports GTPRESET_IN
    ```

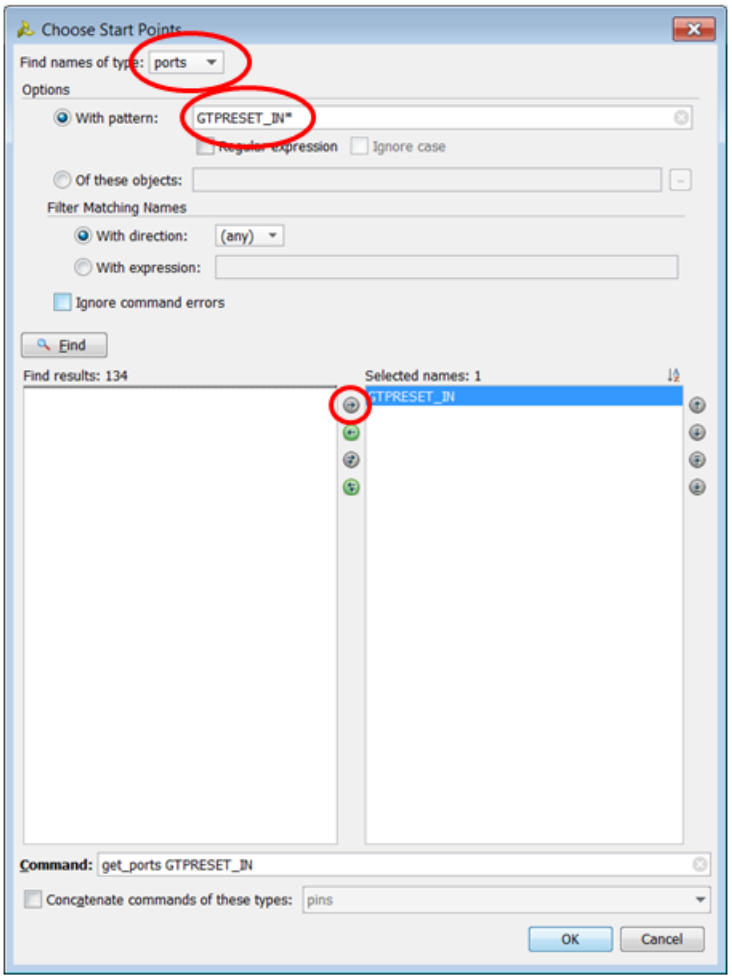


Figure 15: Choose Start Points Window for set_false_path Dialog Box

TIP: *You can also double-click GTPRESET_IN to move it from **Find results** to **Selected names**.*

11. Click **OK** in the choose start points window

    The completed set false path form is shown in Figure 16. Notice the **Command:** field at the bottom of the window:

    ```
    set_false_path -from [get_ports GTPRESET_IN]
    ```

    The Vivado IDE displays the Tcl command form of all constraints created via the dialogs for your review. This is useful for learning the Tcl command syntax, and for verifying the final constraint before adding it.

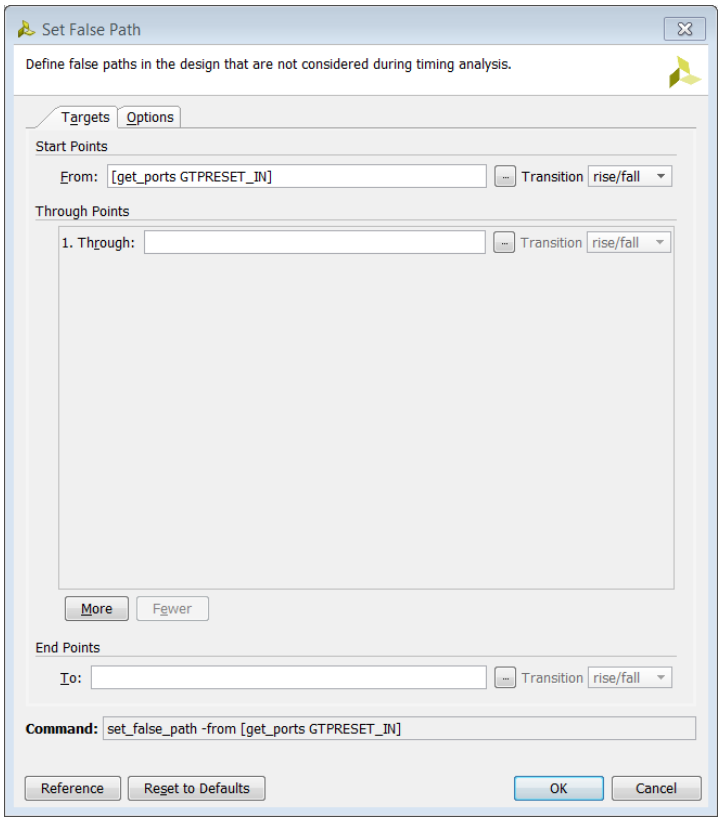


Figure 16: Completed set_false_path constraint window

12. Click **OK** to close the Create Clock wizard, and create the false path constraint as shown in Figure 17.

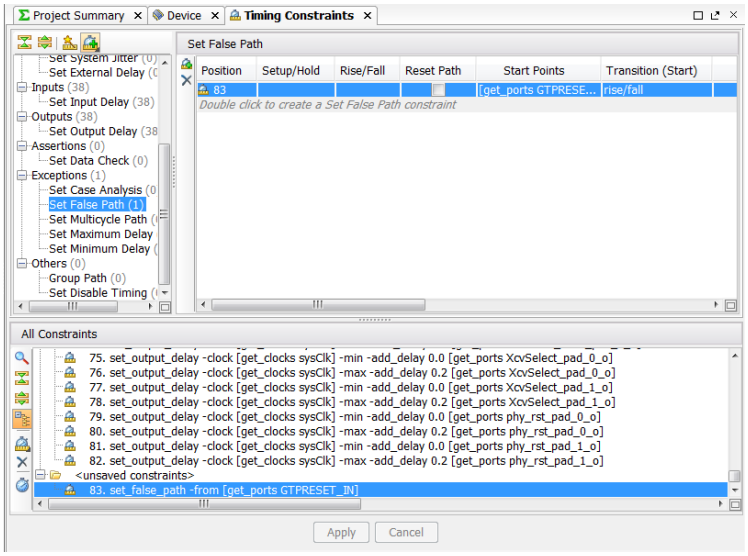


Figure 17: New set_false_path constraint in the constraint editor

13. Next we are going to add a multicycle path using the constraints editor

    a. Double click on **Set Multicycle Path** under the Exceptions category of the tree

    b. In the Set Multicycle Path window, change the **Path Multiplier** to **2**.

    c. In the **Through Points and Transitions** entry box, type the following string:
    `[get_pins cpuEngine/or1200_cpu/or1200_alu/*]`

    d. Click **OK**. A new multicycle path will be added to the constraints editor in the **<unsaved_constraints>** section as shown in Figure 18.

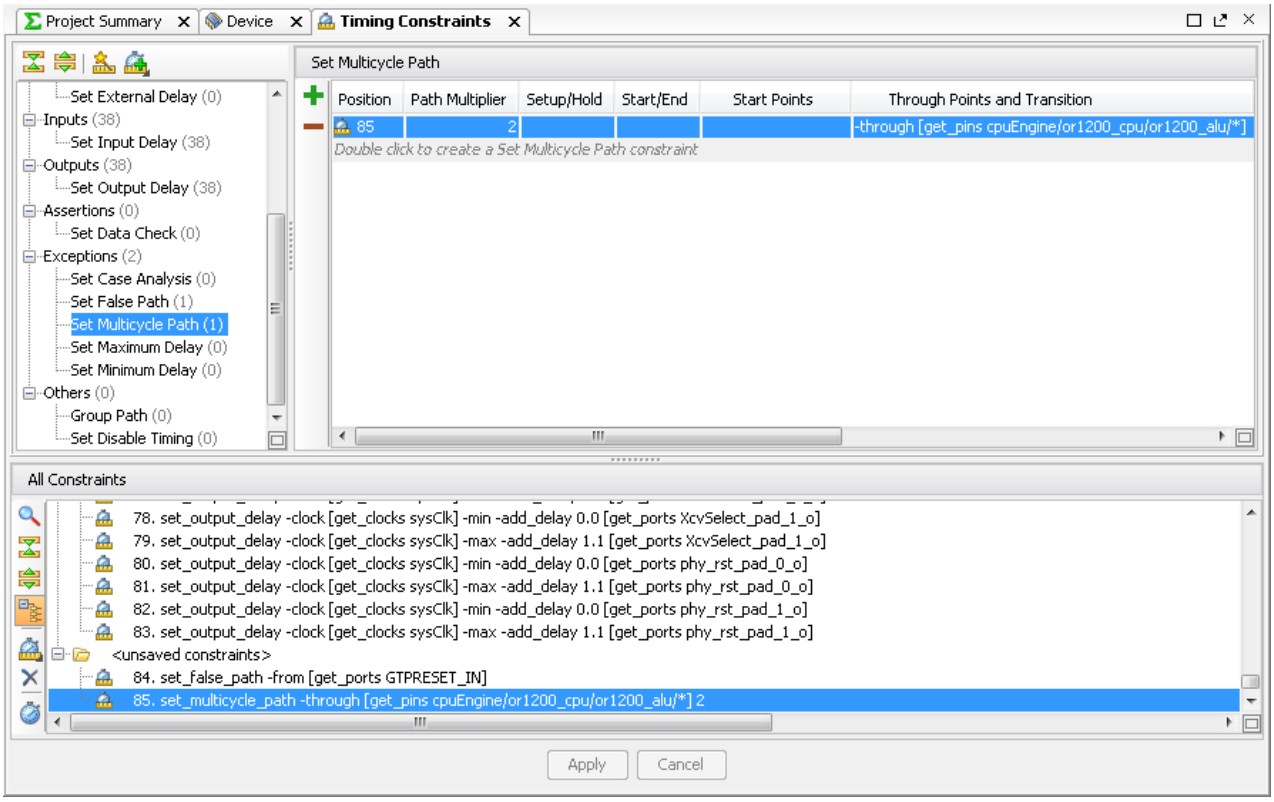


Figure 18: New Multicycle Path Added to the Constraints Editor

14. Adding a multicycle path by default pushes the setup timing to the specified number of cycles (N), *but it also pushes the hold timing to N – 1 cycles*. This is usually not what is intended and could cause Vivado tools to spend a lot of time fixing large hold violations. In this case we want the setup path to be timed to two cycles of the clock, but we want the hold path to be timed to zero cycles of the clock. To achieve this, we need to define another multicycle path on the hold edge to 1, such that
N – 1 is zero. For more information about this situation, please refer to *Vivado Design Suite User Guide: Using Constraints* (UG903).

    a. Double click on **Set Multicycle Path** under the Exceptions category of the tree for a second time.

b. Note that all the fields you entered previously are still filled in

c. In the Set Multicycle Path window, change the **Path Multiplier** to **1**

d. Select the **Options** tab

e. Check the box that says **Use path multiplier for hold (minimum delay) calculation**.

f. Click **OK**

We now have a fully constrained design in memory. Saving the constraints to disk is covered in Step 5.

Step 5: Saving Constraints

Constraint management is an important step of the design flow, and the Vivado Design Suite provides you the flexibility of adding new constraints into an existing constraint file, overwriting existing constraints, or creating a new constraints file to track design changes or complete missing constraints.

You have created a few timing exceptions for the design, but the constraints exist only in memory and not on disk yet. You need to save the constraints to the `timing.xdc` file.

1. From the Sources view in Vivado, double-click `timing.xdc` under **Constraints > lab1**
2. Scroll to the bottom of the file and notice that the `set_false_path` and `set_multicycle_path` constraints do not exist in the file. This is also reflected in the Timing Constraints Editor as `unsaved_constraints`, as shown in Figure 19.

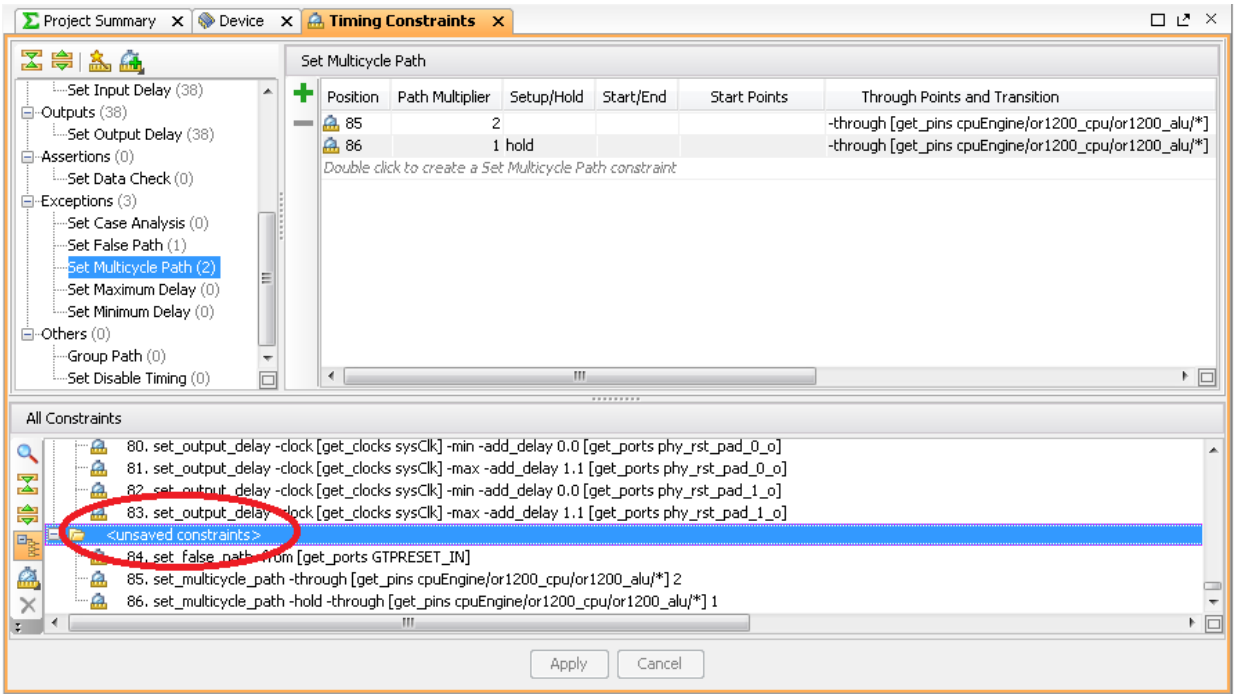
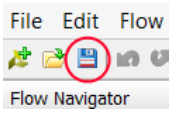


Figure 19: Unsaved Constraints Displayed in the Constraints Editor.

3. Click the **Save Constraints** button, or use the **File > Save Constraints** command from the main menu.



4. Click the **Reload** banner in the **timing.xdc** tab to reload the constraints file from disk. Notice that the false path and multi-cycle paths are now visible in the text file, as shown in Figure 20.

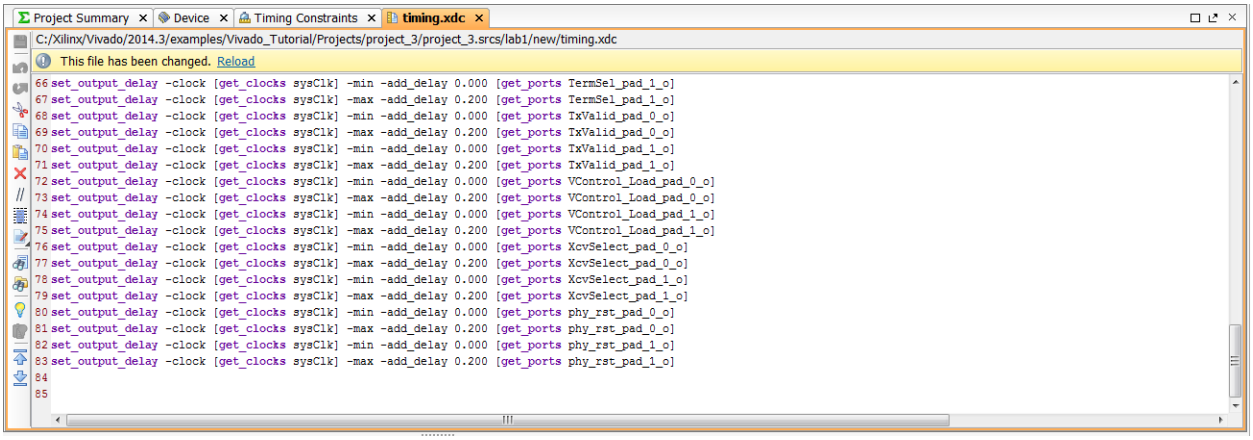


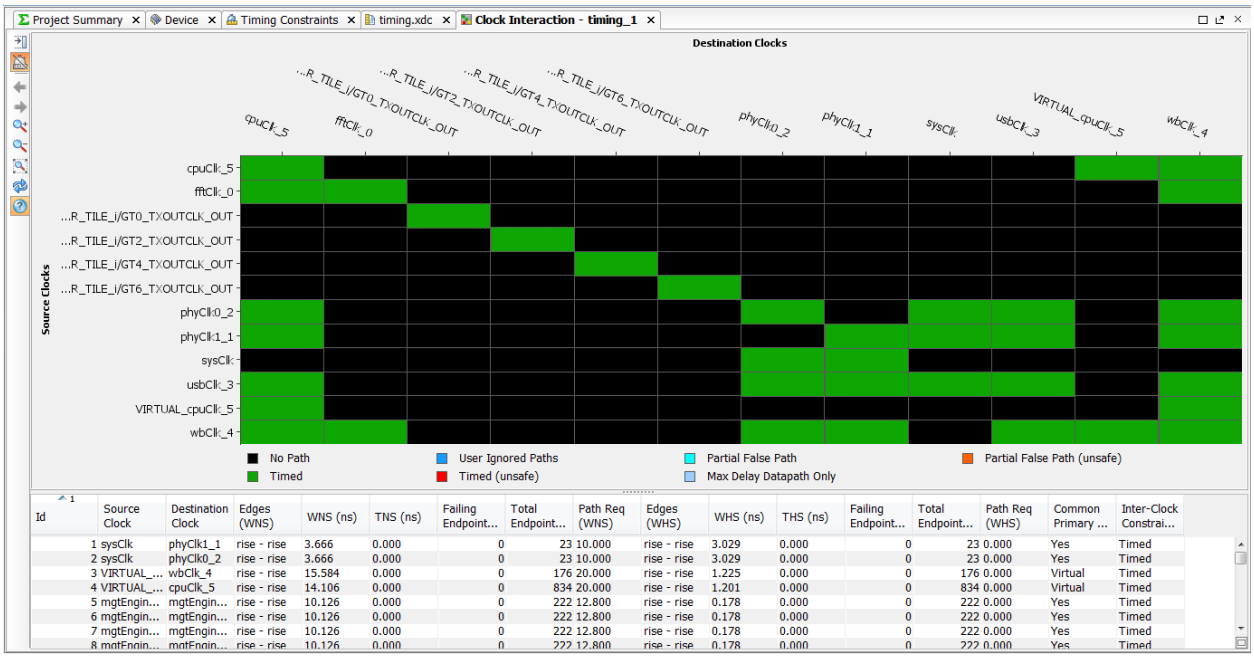
Figure 20: timing.xdc Tab

Step 6: Clock Interaction Report

After or during constraints creation, you must verify that the constraints are complete and safe. Vivado Design Suite times all clocks together by default unless you specify otherwise by defining clock groups or other timing exceptions. The `set_clock_groups` command specifies asynchronous or exclusive clock domains and disables timing analysis between them. You can also use `set_false_path` between two clocks to disable timing on all paths between them, or use it on specific netlist objects to only disable some paths. The `set_multicycle_path` constraint modifies the clock edges used during timing analysis instead of the default single cycle assumption. For more information on using these constraints refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Vivado automatically infers timing path requirements for paths that cross between two different clock domains, called inter-clock paths, making assumptions regarding phase and offset. The Report Clock Interaction command reports inter-clock paths, to help identify potential problems such as unrealistic setup or hold requirements between two clocks, or unsafe timing between asynchronous clocks (no known phase relationship) which can lead to unstable hardware behavior.

1. In the Netlist Analysis section of the Flow Navigator, under Synthesized Design, select **Report Clock Interaction** and click **OK** in the Report Clock Interaction dialog box to accept the default settings.



Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoint...	Total Endpoint...	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoint...	Total Endpoint...	Path Req (WHS)	Common Primary ...	Inter-Clock Constrai...
1	sysClk	phyClk1_1	rise - rise	3.666	0.000	0	23	10.000	rise - rise	3.029	0.000	0	23	0.000	Yes	Timed
2	sysClk	phyClk0_2	rise - rise	3.666	0.000	0	23	10.000	rise - rise	3.029	0.000	0	23	0.000	Yes	Timed
3	VIRTUAL_...	wbClk_4	rise - rise	15.584	0.000	0	176	20.000	rise - rise	1.225	0.000	0	176	0.000	Virtual	Timed
4	VIRTUAL_...	cpuClk_5	rise - rise	14.106	0.000	0	834	20.000	rise - rise	1.201	0.000	0	834	0.000	Virtual	Timed
5	mgtEngin...	mgtEngin...	rise - rise	10.126	0.000	0	222	12.800	rise - rise	0.178	0.000	0	222	0.000	Yes	Timed
6	mgtEngin...	mgtEngin...	rise - rise	10.126	0.000	0	222	12.800	rise - rise	0.178	0.000	0	222	0.000	Yes	Timed
7	mgtEngin...	mgtEngin...	rise - rise	10.126	0.000	0	222	12.800	rise - rise	0.178	0.000	0	222	0.000	Yes	Timed
8	mgtEngin...	mgtEngin...	rise - rise	10.126	0.000	0	222	12.800	rise - rise	0.178	0.000	0	222	0.000	Yes	Timed

Figure 21: Clock Interaction Report

The Vivado IDE generates a graphical matrix illustrating the relationship of the various clocks in the design, as shown in Figure 21. For this design, the primary clock (sysClk) connects to an MMCM, which generates six additional clocks. The clock interactions shown are between these generated clocks. In addition, the timing constraints wizard created a few more generated clocks and virtual clocks to fully constrain the design.

The Clock Interaction report shows clock pairs with no path between them (black), with paths safely timed (green and light blue), with paths not safely timed (reg and orange) and with paths covered by Max Delay Datapath Only constraints.

Important! *Green in the matrix does not mean that timing is met, it simply means that the timing constraints and the clock tree topologies allows safe timing analysis and accurate slack computation.*

In the Clock Interaction report, unsafe means there is no common primary clock (no known phase relationship) or no common clock period within the first 1000 clock cycles of the source and destination clocks . The Vivado timing engine selects edges on the launch and capture clocks based on the first 1000 cycles, but these edges may not reflect the most pessimistic analysis between the clocks.

TIP: *The colors described here are the default colors. Your colors may be configured differently from those shown in Figure 21.*

2. Close the Clock Interaction window by clicking the ✕ in the window tab.

Step 7: Timing Summary Report

Timing paths start and end at clocked elements. Input and Output ports are not sequential elements, and by default Vivado timing analysis does not time paths to or from I/O ports in the design, unless input/output delay constraints are specified.

In this step you will generate and observe timing reports in Vivado.

1. Select **Tools > Timing > Report Timing Summary**; leave the default options.
2. Press **OK** to generate the report.

   The timing summary tab will open as seen in Figure 22.

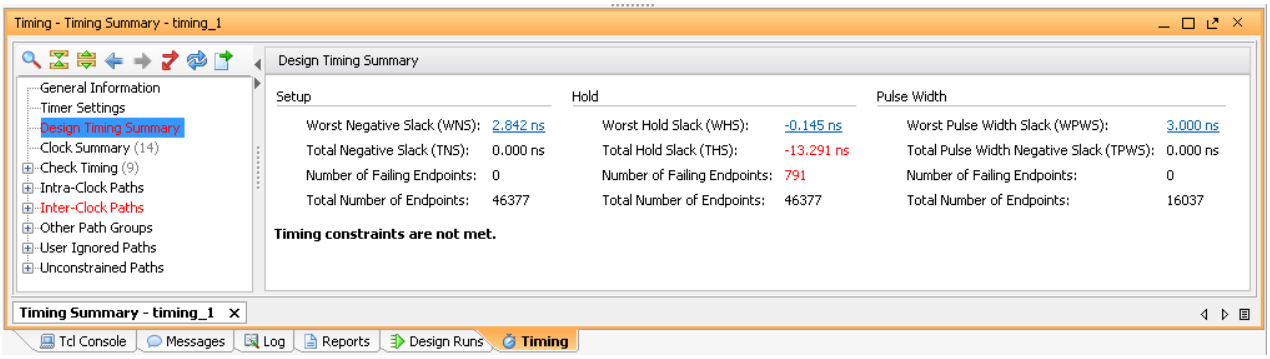


Figure 22: Report Timing Summary Results

3. This design passes setup timing but fails hold analysis. Before implementing the design, timing analysis uses estimated net delays that represent ideal placement. Small hold violations are common at this point of the flow and will be fixed during the routing step. For now, let us review the content of the report. Click on the **Worst Negative Slack** link in the design timing summary section to see the worst timing path in the design.

   Figure 23 shows the worst path in the design.

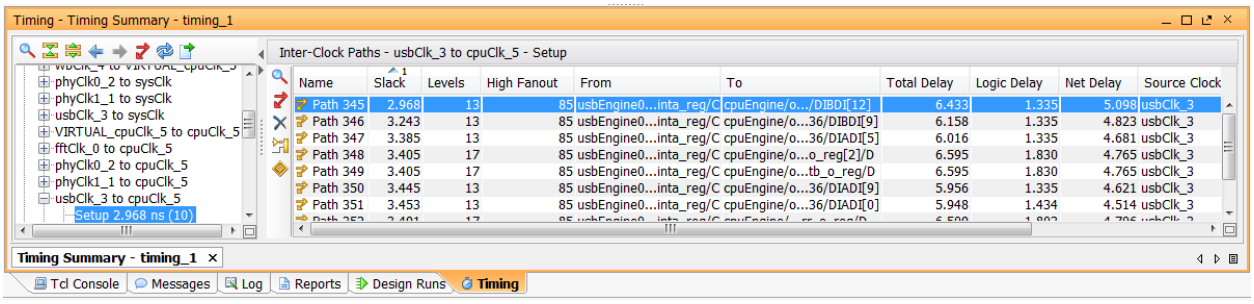


Figure 23: Worst Timing Path in the Design

4. When the worst path is selected, press **F4** to bring up its schematic. Figure 24 shows the worst setup path in the design.

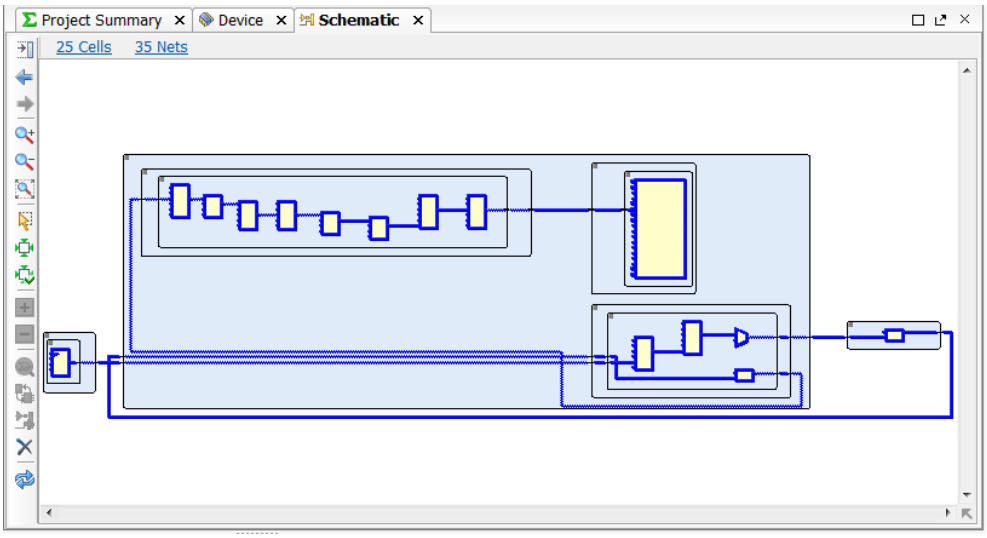


Figure 24: Schematic of Worst Setup Path in the Design

5. In the timing summary tree, select **Check Timing**.
   - There are nine issues flagged by Check Timing shown in Figure 25.
   - Eight of these are `pulse_width_clock` checks which were also flagged by the timing constraints wizard, but were not constrained. These violations have low severity because the corresponding missing clocks are not needed for timing logic paths.
   - The remaining issue flagged by Check Timing is a missing input constraint on the reset signal that was set to `false_path`. This can also be ignored.

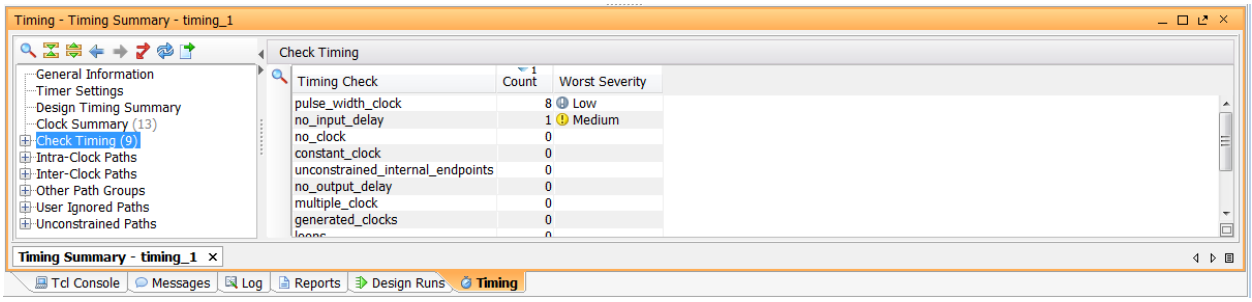


Figure 25: Check Timing Results in the Timing Summary Window

6. In the timing summary tree, select the **Clock Summary**, as shown in Figure 26. The clock summary section of the timing summary report lists all the clocks in the design and shows the resulting frequencies and waveforms of each clock. The hierarchy shows the relationship between the generated clocks and the primary clock (for example, cpuClk_5 vs. sysClk).

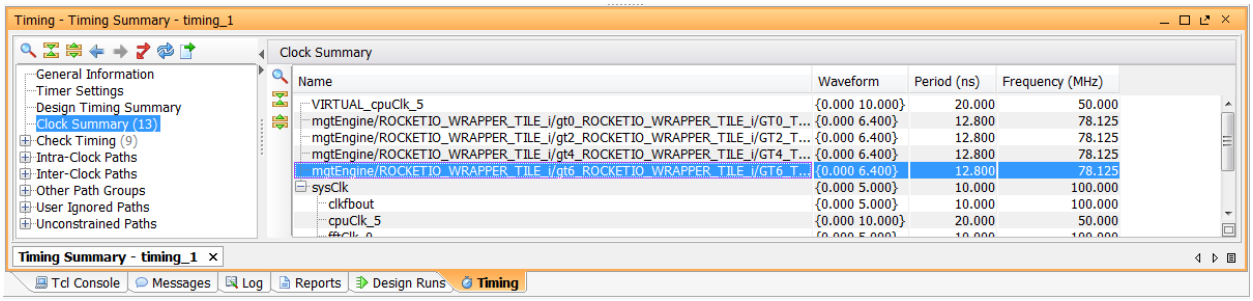


Figure 26: Clock Summary

7. The remaining sections of the timing summary report group paths by their type. Each section lists the top ten paths (specified when the report was generated) in that group. These include inter-clock paths, intra-clock paths, other path groups, user ignored paths, and unconstrained paths. Clicking the roots will show a summary of the paths beneath. Expanding the tree further will ultimately display the top timing paths for each group.

Conclusion

At this point you may either continue to **Lab #2**: Setting Physical Constraints, or exit the Vivado Design Suite and continue later.

You have learned:

- Creating a constraint set and setting a target constraint file
- How to add timing constraints to a design using the timing constraints wizard
- How to add timing constraints using the timing constraints editor
- The importance of saving constraints to disk versus in-memory constraints
- How to generate the clock interaction report and properly interpret the resulting matrix
- How to generate the timing summary report and properly interpret the results

In this lab, you will create physical constraints for the CPU Netlist design, observing how actions in the GUI call Tcl commands. Using Tcl commands, complex operations are easily scripted for repeated use, at various stages of the flow.

> ***Note:*** *If you are continuing from Lab1, and your design is open, skip ahead to* *Step 2: Adding Placement Constraints*.

Step 1: Opening the Project

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.

   From the Getting Started screen you can open recent projects as listed on the right hand side of the window.
2. Under **Recent Projects**, click **project_cpu_netlist** as shown below.

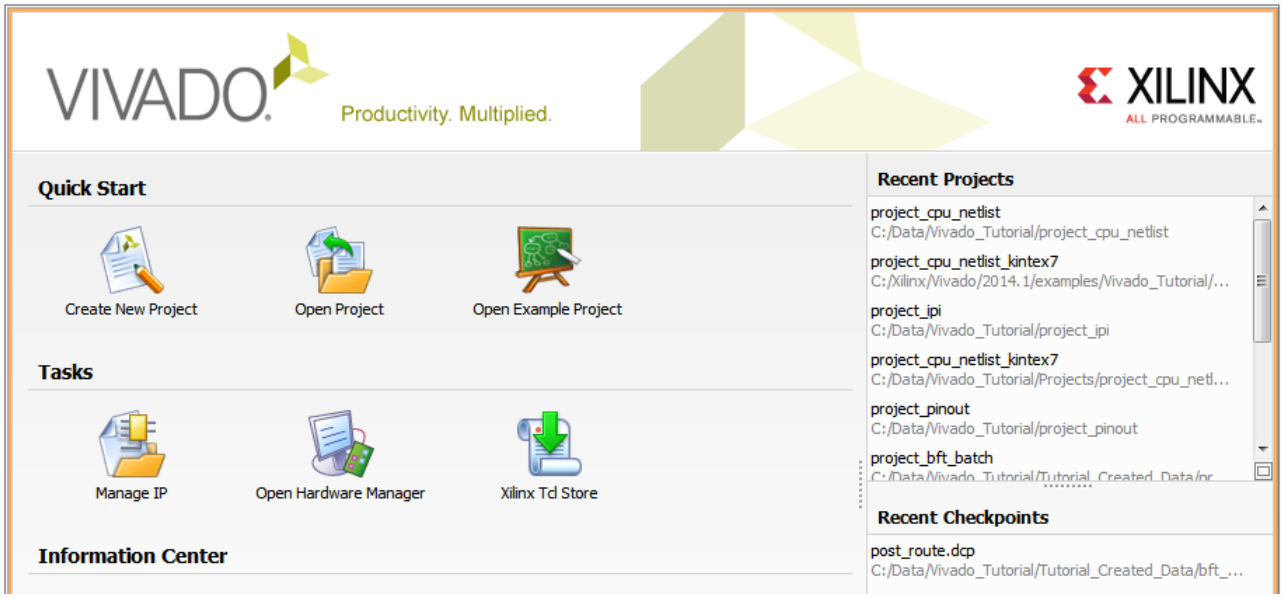


Figure 27: Open Recent Projects from Getting Started

Step 2: Adding Placement Constraints

Explore some of the design hierarchy, and begin placing logic elements to create physical constraints.

1. From the Flow Navigator select **Open Synthesized Design**.

   The synthesized netlist opens with the Device window displayed.

2. Select the **Netlist** window and expand the **clkgen** hierarchy.

3. Expand the **Leaf Cells** folder and select the **mmcm_adv_inst** (MMCME2_ADV).

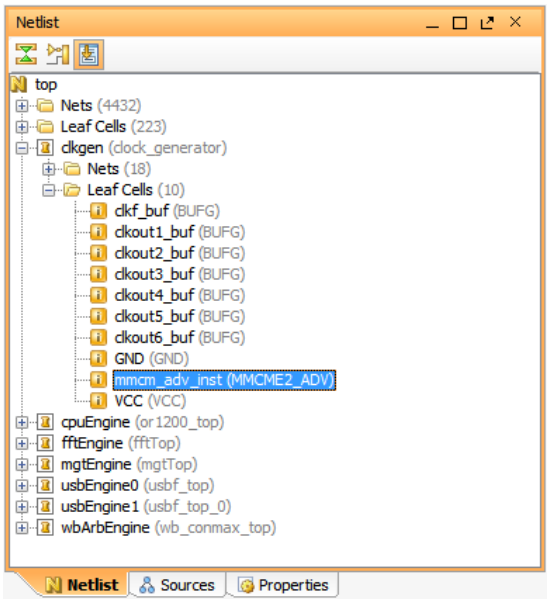


Figure 28: Netlist window

4. Look in the Properties view, under the Properties tab, and notice that the STATUS is UNPLACED, and there are no `IS_LOC_FIXED` or `IS_BEL_FIXED` properties shown.

5. Check this in the Tcl Console by typing:

   ```
   get_property IS_LOC_FIXED [get_cells clkgen/mmcm_adv_inst]
   ```

   This returns a zero, indicating the object is not fixed to a location.

6. Zoom into the bottom right of the Device view, to display the lower half of **Clock Region X1Y0**, to prepare for placing the selected object. See Figure 29.

Tip: *It is easier to place logic in the Device window if Routing Resources are not displayed. If necessary, select the* **Routing Resources** *toolbar button*  *to disable the display of routing resources.*

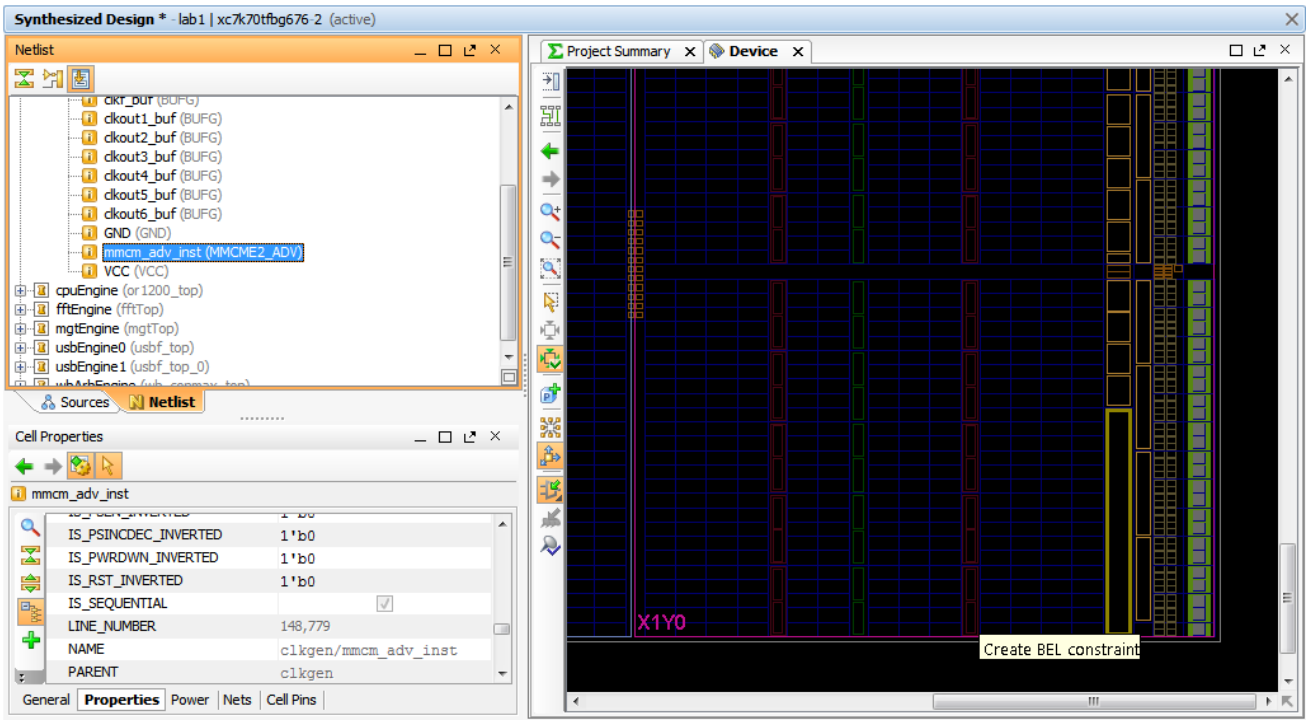


Figure 29:Placing the MMCM

7. In the Netlist window, click on the **mmcm_adv_inst** and drag it into the Device window to place it into the bottom right MMCME2_ADV.

   Look in the Tcl Console. You should see something like these three commands:

   ```
   startgroup
   place_cell clkgen/mmcm_adv_inst MMCME2_ADV_X1Y0/MMCME2_ADV
   endgroup
   ```

   The `startgroup` and `endgroup` Tcl commands bracket sequences of commands to support the undo function in the Vivado tools. If you make a mistake, you can use the `undo` command in the Tcl Console, or the **Edit > Undo** command. This will undo the placement and allow you to redo it. For more information on `startgroup`, `endgroup`, and `undo`, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

8. Look at the Properties tab of the Cell Properties window for the MMCM cell you placed.

   Notice the `IS_BEL_FIXED` and `IS_LOC_FIXED` properties now reflect that the object has been placed, as shown in Figure 30. The `STATUS` is `FIXED` as well.

Tip: *The Cell Drag and Drop mode in the Device window determines whether `IS_LOC_FIXED` is set, or `IS_BEL_FIXED` is also set, when placing objects. Refer to the Vivado Design Suite User Guide: Using the Integrated Design Environment (UG893) for more information on using the Device window.*

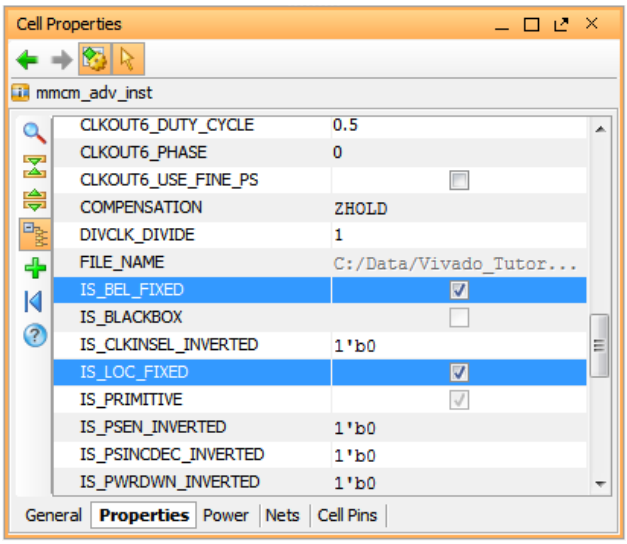
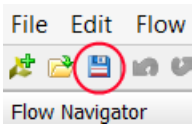


Figure 30: BEL and LOC Placement Constraints

The `IS_BEL_FIXED` and `IS_LOC_FIXED` properties on the object are physical constraints reflecting the placement of the object. These constraint are used by Vivado implementation, and will not be changed by the tool. However, if the properties are invalid, they will cause errors downstream in the design flow.

Notice that when you place the `mmcm_adv_inst` in the Device window, the Save Constraints icon is enabled. The physical constraints are added to the Vivado tool in-memory design, but are not yet saved to the target constraint file.



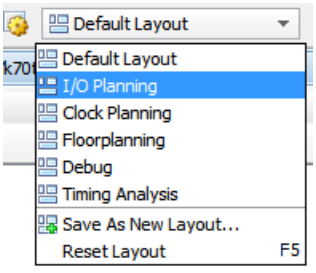
Step 3: Defining Additional Physical Constraints

In this step you will define additional physical constraints to the design, such as the `PACKAGE_PIN`, and `PROHIBIT` constraints.

1. Select the **I/O Planning** view layout from the Layout Selector in the tool bar menu.

   The I/O Planning view layout displays the Package window, as well as the I/O Ports and Package Pins windows, to facilitate planning the I/O port assignment for the design.

   For the purposes of this tutorial, assume the PCB layout has been completed, and therefore certain pins are not accessible on the FPGA package. You can prohibit the Vivado tool from using these pins during placement and routing (assuming you have not already specified all of your I/O assignments).



2. Select the **AA8** pin in the Package window.

Tip: *Use the X and Y-axis values on the edge of the Package window to help you locate this pin on the package.*

3. With the pin selected, right-click and select **Set Prohibit**.

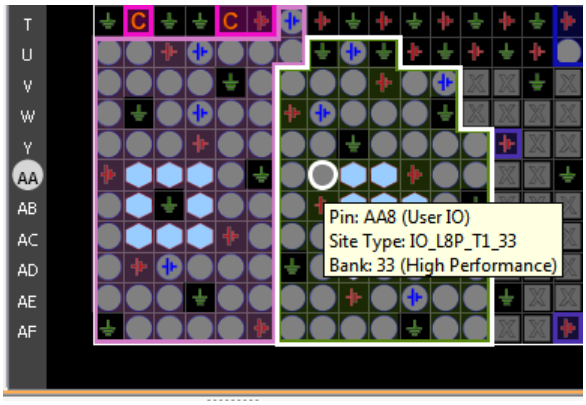


Figure 31: Package Pin A8

When you unselect the pin, you will notice the site now has a red circle with a diagonal line through it, indicating it is unusable.

4. Look in the Tcl Console and review the TCL command produced by the Vivado IDE:

```
set_property prohibit 1 [get_sites AA8]
```

Step 4: Defining Constraints with Object Properties

You can create timing and placement constraints as you have seen in this tutorial. You can also change the properties of cells to control how they are handled by Vivado implementation. Many physical constraints are defined as properties on a cell object.

For example, if you discover a timing issue with a RAM in the design, to avoid resynthesis, you can change a property of the RAM cell to add in pipeline registers. After confirming with the designer and validation teams that this is an acceptable approach, you can change the design.

Setting Cell Properties

Because it can be too time consuming and costly to go back to the RTL after synthesis, you can make changes in the netlist as follows.

1. Select **Edit > Find** to open the Find dialog box, seen in Figure 32.
    a. Specify Find **Cells.**
    b. Under Properties, specify **PRIMITIVE_TYPE is BMEM.BRAM**
    c. Click **OK.**

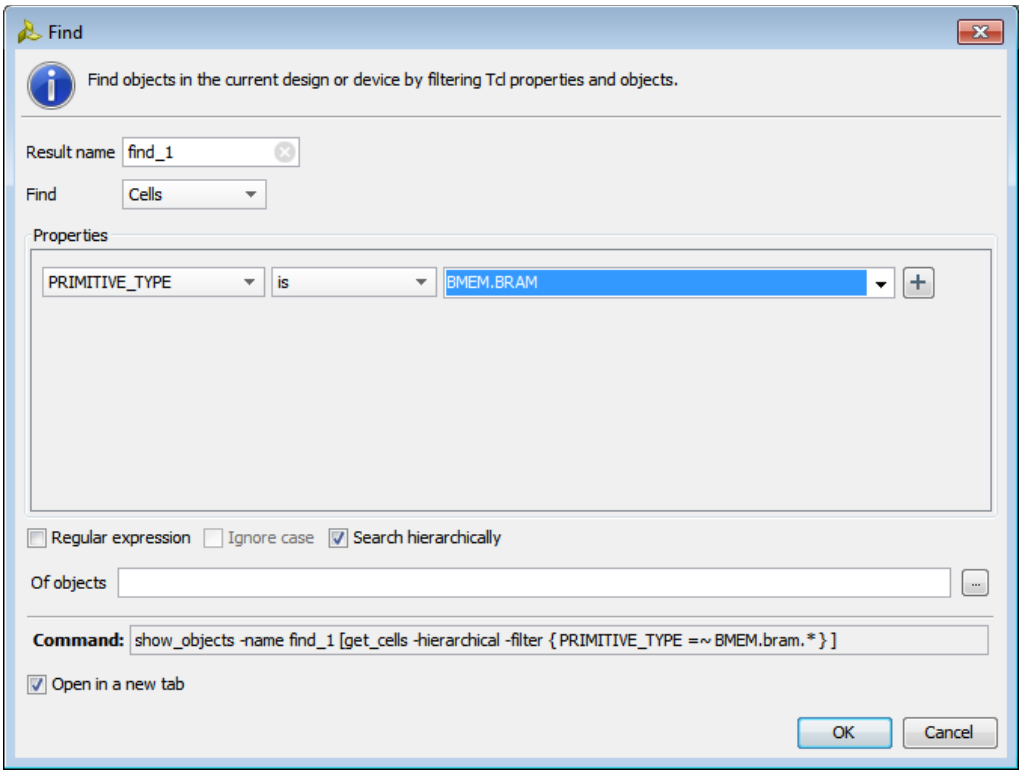


Figure 32: Find dialog box

The Find Results window displays.

2. **Select** the **Show Search** command, ,on the sidebar menu of the Find Results window.
3. **Search** for **`ingressLoop`**, and select the following cell:

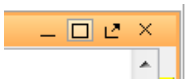
   `fftEngine/fftInst/ingressLoop[7].ingressFifo/...`

   In the Properties tab of the Cell Properties window, you can see the `DOA_REG` and `DOB_REG` are set to zero, indicating that the output registers are disabled.
4. Generate a custom timing report from this cell directly from the Tcl Console. The Tcl command to enter is:

```
report_timing -from [get_cells
fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block_
ram_performance.fifo_ram_reg]
```

Tip: *You can copy and paste the cell name from the General tab of the Cell Properties window into the Tcl console.*

5. In the upper right corner of the **Tcl console**, click the **Maximize** button to maximize the window and better view the timing report.



6. In the data path section of the report, 1.800ns is added for this RAM.

```
-------------------------------------------------    -------------------
  RAMB36E1 (Prop_ramb36e1_CLKBWRCLK_DOBDO[16])
                              1.800    -0.645 r  fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block_ram_performance.
  net (fo=2, unplaced)        0.466    -0.179    fftInst/toBft[15]_55[0]
  DSP48E1                                    r  transformLoop[7].ct/xOutReg_reg/A[0]
-------------------------------------------------    -------------------
```

7. Restore the **Tcl Console** to its normal size.
8. In the **Properties** tab of the Cell Properties window, select the **DOA_REG** and **DOB_REG** properties for this cell, and change their values from "0" to "**1**".

   You can see the two `set_property` commands run in the Tcl Console.

   ```
   set_property DOA_REG {1} [get_cells
   {fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block
   _ram_performance.fifo_ram_reg}]

   set_property DOB_REG {1} [get_cells
   {fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block
   _ram_performance.fifo_ram_reg}]
   ```

9. Rerun the **timing report** from the selected cell. The Tcl command to enter is:

   ```
   report_timing -from [get_cells
   fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block_
   ram_performance.fifo_ram_reg]
   ```

10. Notice that the **data path delay** for the RAM is now 0.622ns.

Setting Design Properties

Next, set the configuration mode on the design. This is another property that results in a physical constraint, in this case a property of the design rather than of a cell. To begin, list all of the properties of the current design.

1. List the properties of the design in the Tcl Console:

   ```
   list_property [current_design]
   ```

   This command returns the list of all defined properties on the current design. To make the list more readable, you can use the standard Tcl `join` command to combine the properties output with "\n" newline character, resulting in each property displaying on a separate line.

   ```
   join [list_property [current_design]] \n
   ```

2. The specific property of interest is CONFIG_MODE. To see what values this particular property can accept, use the list_property_value Tcl command:

   ```
   join [list_property_value CONFIG_MODE [current_design]] \n
   ```

3. Use the **Tools > Edit Device Properties** command to set the CONFIG_MODE property for the current design.

The Edit Device Properties dialog box opens as shown in Figure 33.

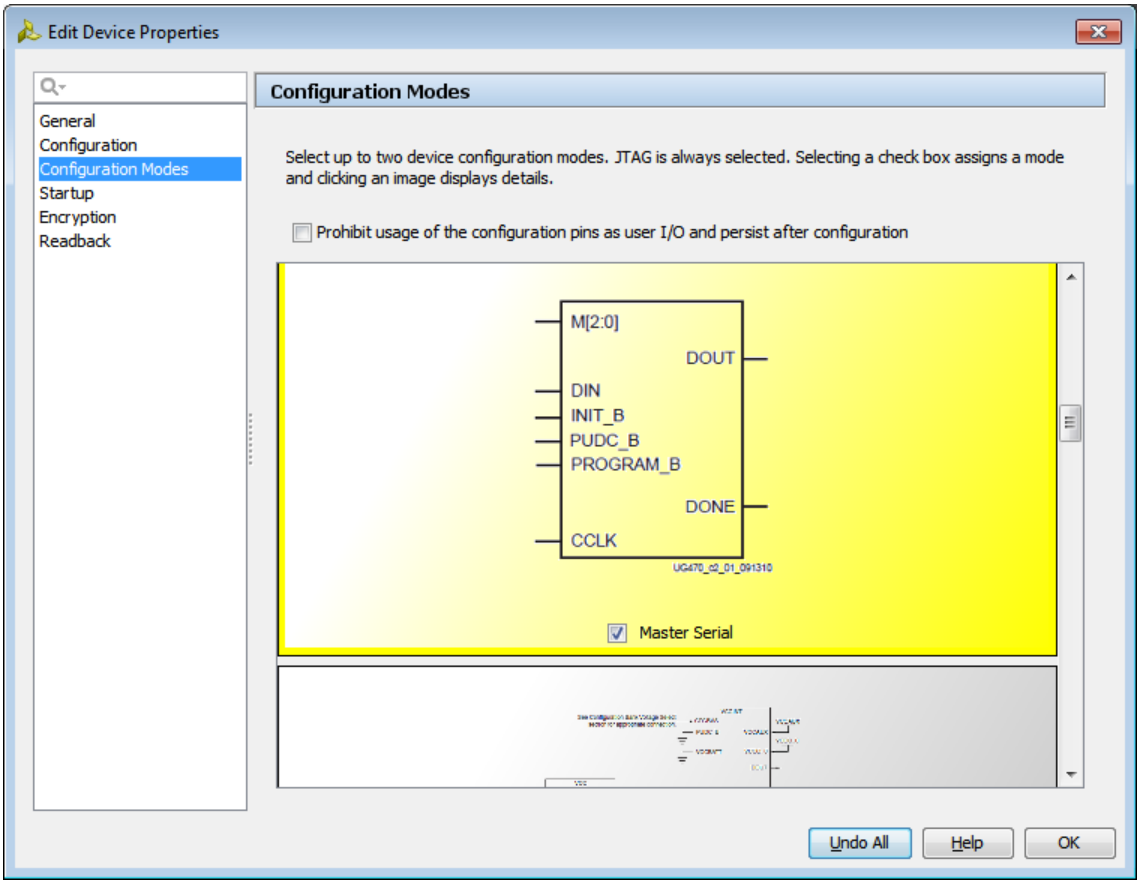


Figure 33: Edit Device Properties – CONFIG_MODE

4. **Select** the **Master Serial** configuration mode as shown, and click OK to close the dialog box.

   The Tcl console shows the `set_property` command that sets the CONFIG_MODE:

   ```
   set_property CONFIG_MODE M_SERIAL [current_design]
   ```

   The configuration mode has now been set.

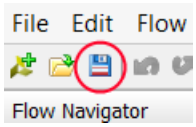
5. **Use** the **`get_property`** command to confirm that the CONFIG_MODE property was correctly set:

   ```
   get_property CONFIG_MODE [current_design]
   ```

   The property value `M_SERIAL` is returned by the Vivado tool.

Step 5: Saving Constraints

Notice that the Save Constraints icon is enabled because there are new design constraints. The cell and design properties you modified in Lab #2 have been added to the Vivado tool in-memory design, but are not yet saved to the target constraint file.



1. Click the **Save Constraints** button.

   The physical constraints you defined over the course of Lab #2 are saved to the target constraint file.

2. **Select** the **target XDC** from the active constraint set in the Sources window to **open** the **file** in the Vivado IDE text editor.

   Notice that the six `set_property` commands you used in Lab #2 are saved to the constraint file. Only design constraints are written to the XDC file, not the object queries or reporting commands that you also used in this lab.

3. Look for the following constraints in the open constraint file:

```
set_property BEL MMCME2_ADV [get_cells clkgen/mmcm_adv_inst]
set_property LOC MMCME2_ADV_X1Y0 [get_cells clkgen/mmcm_adv_inst]
set_property PROHIBIT true [get_sites AA8]
set_property DOA_REG 1 [get_cells
{fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block
_ram_performance.fifo_ram_reg}]
set_property DOB_REG 1 [get_cells
{fftEngine/fftInst/ingressLoop[7].ingressFifo/buffer_fifo/infer_fifo.block
_ram_performance.fifo_ram_reg}]
set_property CONFIG_MODE M_SERIAL [current_design]
```

4. **Exit** the **Vivado** IDE.

Summary

In this lab, you learned how to use both the Vivado IDE and the Tcl Console to create and verify physical constraints. Most actions performed in the IDE result in Tcl commands being run in the Tcl Console. The Vivado IDE provides powerful interactive capabilities for developing physical and timing constraints, which can then be saved to constraint files and reused as needed.

Please Read: Important Legal Notices