FPGA Tutorial: Implementation

Using VHDL and Vivado to design a pulse width modulator system

Material adapted from:	
Title	**Source**
Basic FPGA Tutorial	so-logic, https://www.so-logic.net/en/
Vivado Implementation Tutorial	Xilinx, https://www.xilinx.com/support.html#documentation

Contents

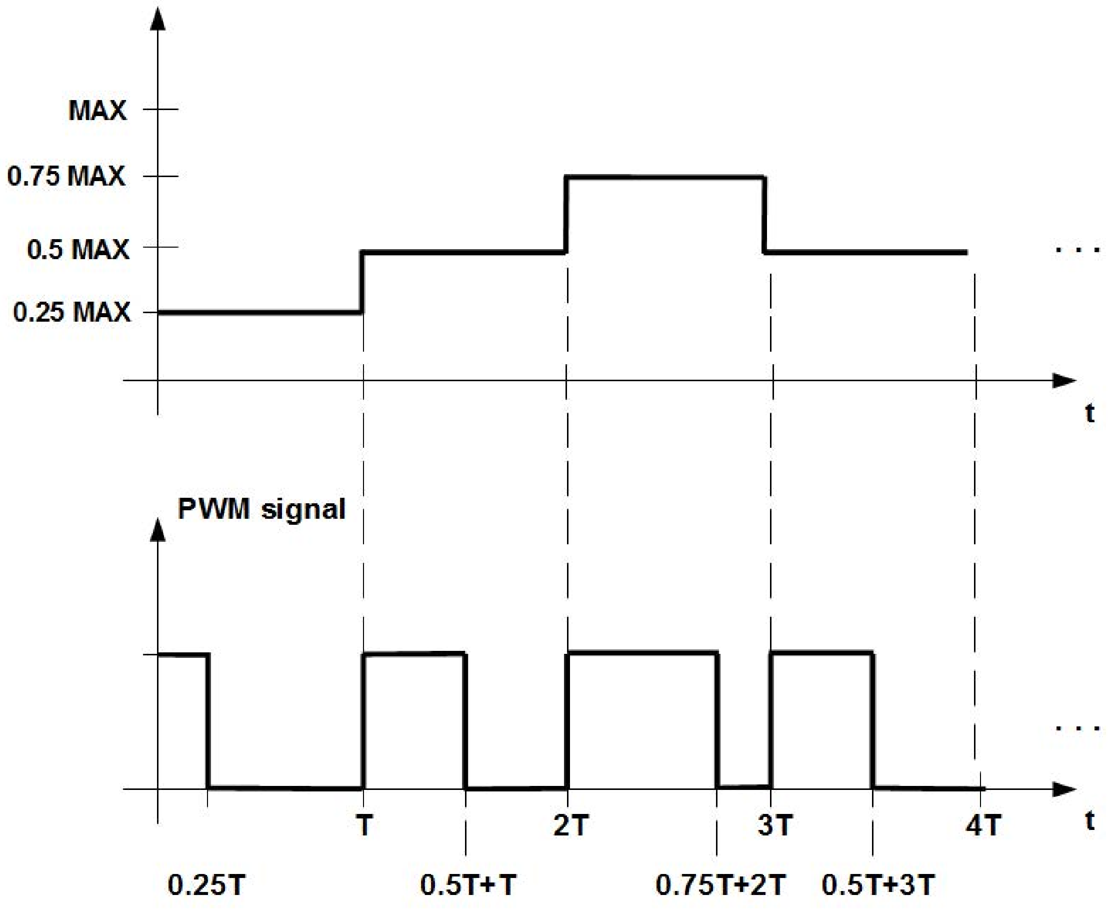
Design Description

In this tutorial a PWM signal modulated using the sine wave with two different frequencies (1 Hz and 3.5 Hz) will be created. Frequency that will be chosen depends on the position of the two-state on-board switch (sw0).

PWM Signal

Pulse-width modulation (PWM) uses a rectangular pulse wave whose pulse width is modulated by some other signal (in our case we will use a sine wave) resulting in the variation of the average value of the waveform. Typically, PWM signals are used to either convey information over a communications channel or control the amount of power sent to a load.

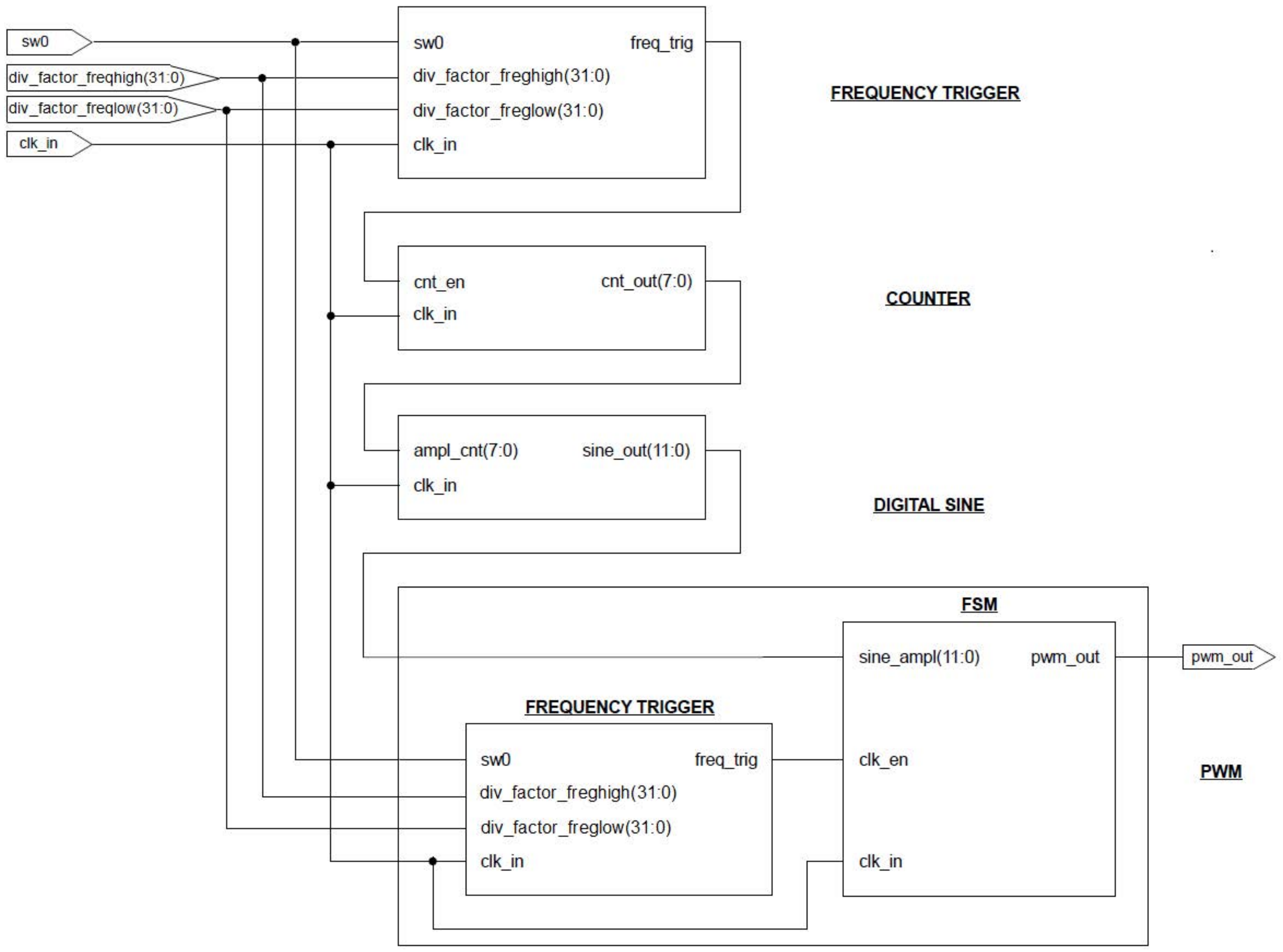
To learn more about PWM signals, please visit http://en.wikipedia.org/wiki/Pulse-width_modulation.



Example of the PWM signal

Modulator Design

The block diagram for the pulse width modulator used in this tutorial is shown below:



Block diagram for PWM system

Let us briefly explain each module shown on the diagram above:

Frequency Trigger
This module will generate one output signal with two possible frequencies
one with 256 Hz and the second one with 896 Hz. Which frequency will be chosen depends on the position of the two-state on-board switch (sw0).

Counter
This module will be an universal (generic) counter. Its task will be to generate read addresses for the ROM where samples of the sine wave are stored. The speed of the counting will be controlled by the Frequency Trigger module, via freg_trig port, and the output of the Counter module will be an input of the Digital Sine module.

Digital Sine

This module will generate an digital representation of an analog (sine) signal with desired frequency. It will use the counter values as addresses to fetch the next value of the sine wave from the ROM.

In our case we will make a VHDL package with a parametrized sine signal. 2^8=256 unsigned amplitude values during one sine-period that will be stored into an ROM array.

A VHDL package is a way of grouping related declarations that serve a common purpose. Each VHDL package contains package declaration and package body.

Note: Don't forget to include the Sine package in the code of the Digital Sine module!

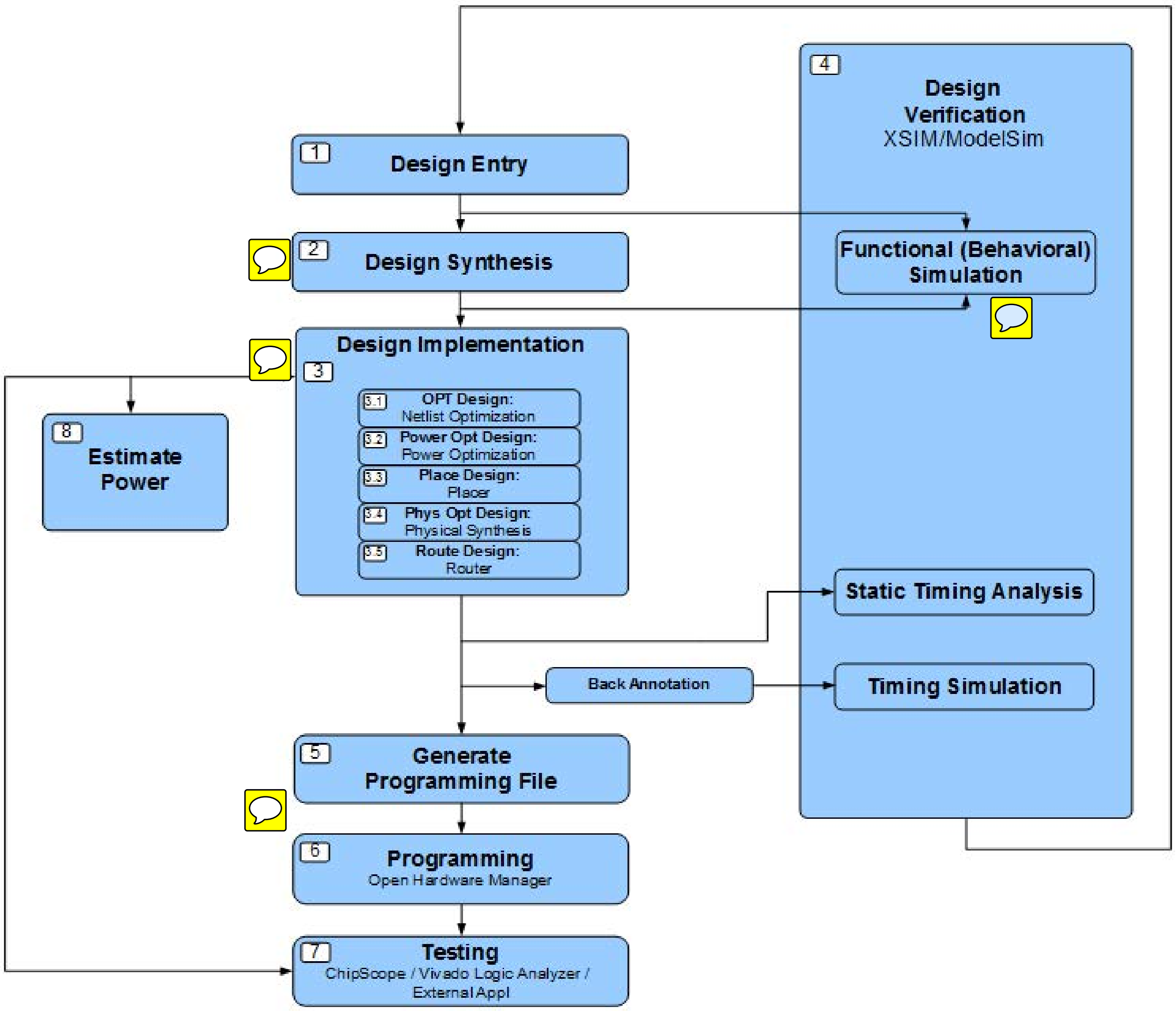
PWM

This module will generate an PWM signal modulated using the digital sine wave from the Digital Sine module. This module will be composed of two independent modules. One will be the Frequency Trigger, for generating two different frequencies and the second one will be the Finite State Machine (FSM), for generating the PWM signal.

Frequency Trigger - output from this module will be used to control the frequency at which FSM module works. As we have already said, in PWM signal information is represented as duty cycle value in each period of the signal. Since our digital sine signal can have 4096 possible values, there will also be 4096 different duty cycle values. This means that PWM's FSM must operate at frequency that is 4096 times higher than the one used by the Digital Sine module.

FSM - is necessary to generate the PWM signal. It will generate the PWM signal with correct duty cycle for each period based on the current amplitude value of digital sine signal, that is stored in the ROM.

Vivado FPGA Design Flow



Creating a New Project

To create a project, we will use the New Project Wizard. The source files are provided on Blackboard in the ___ folder.

Step 1. Open the Vivado Design Suite IDE.

Step 2. In the **Getting Started** page, click Create Project to open the **New Project wizard**. Click **Next**.

Step 3. In the **Project Name** dialog box, do the following:

- Name the new project "modulator".
- Specify the project location.
- Ensure that Create Project Subdirectory is selected.
- Click Next.

Step 4. In the **Project Type** dialog box, do the following:

- Specify the Type of Project to create as RTL project.
- Ensure that the **Do not specify sources at this time** box is selected.
- Click Next.

Step 5. In the **Add Sources** page, add all the source files provided in the ___ folder on Blackboard. Set the target language to Verilog and the simulator language to Mixed. Verify that "Copy sources into project" is selected. Click Next.

Step 6. Bypass the Add Constraints dialog box by clicking **Next**.

Step 7. In the **Default Part** page, select Boards and then select the Zedboard Zynq Evaluation and Development Kit. Since we will go no further than the Implementation step today, the board will not actually be used, so it doesn not matter that this is not our model. Click Next.

Step 8. Review the Project Summary page and click Finish.

When we have all the necessary design files for our design, we can implement targeting the FPGA design. First we should create an XDC constraints file where we will define placement and timing constraints for our design. Then, we should synthesize and implement our design.

Creating an XDC File

The Vivado IDE software allows you to specify different types of constraints to help improve your design performance. Each type of constraint serves a different purpose and is recommended under different circumstances. Below are some of the most commonly used types of constraints:

- **Timing Constraints** - are typically specified globally but can also be specified for individual paths. Global constraints include period constraints for each clock, setup times for each input, and clock-to-out constraints for each output. You can enter timing constraints using the option for the timing constraints creation in the Flow Navigator. This creates a text-based Xilinx Design Constraints (XDC) file.
- **Placement Constraints** - for FPGA designs, you can specify placement constraints for each type of logic element, such as BRAMs, DSPs, LUTs, FFs, I/Os, IOBs, and global buffers. Individual logic gates, such as AND and OR gates, are mapped into CLB function generators before the constraints are read and cannot be constrained.
- **Synthesis Constraints** - Synthesis constraints instruct the synthesis tool to perform specific operations. When using "Vivado Synthesis" for synthesis, synthesis constraints control how "Vivado Synthesis" processes and implements FPGA resources, such as state machines, multiplexers, and multipliers, during the HDL synthesis and low level optimization steps. Synthesis constraints also allow control of register duplication and fanout control during global timing optimization.

Important: The Vivado IDE doesn't support use of User Constraints File (UCF). UCF constraints are replaced with Xilinx Design Constraints (XDC). The tool supports XDC, which is based on the industry-standard Synopsys Design Constraints (SDC).

You can enter XDC constraints in several ways, at different points in the flow:

- Store the constraints in one or more XDC files
- Generate the constraints with Tcl script

There are two different ways of generating an XDC File:

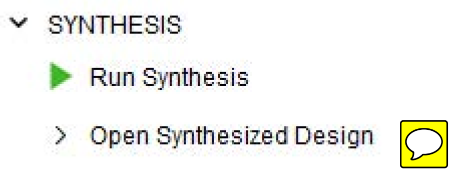
- Using the Vivado GUI (I/O Planning View)
- Using Text Editor

Creating a XDC File using the Vivado GUI (I/O Planning view):

In this step, you will be using the I/O Planning View to place the unplaced pins in the design. In order to assign pins to the FPGA, the proper pin assignments will be determined by using the ***"ZedBoard Hardware User's Guide"***. Every board's user guide contains the pin details and a reference master XDC file specifying the location and the I/O standards to be used while selecting a pin for the design.

In order to apply the constraints to the design, the design has to be synthesized at least once. Synthesis is the process of transforming an RTL-specified design into a gate-level representation. Therefore, you will start the constraints file creation by synthesizing the design and opening the synthesized design.

Step 0. To open the synthesized design, click the **Open Synthesized Design** option in the Flow Navigator.



Open Synthesized Design button

To create a XDC file using the Vivado IDE GUI, do the following:

Step 1. Change the layout from the **Default Layout** to **I/O Planning** view, in the layout pull-down menu in the main toolbar, to identify pins that don't have an assigned location, see Fig 1.1

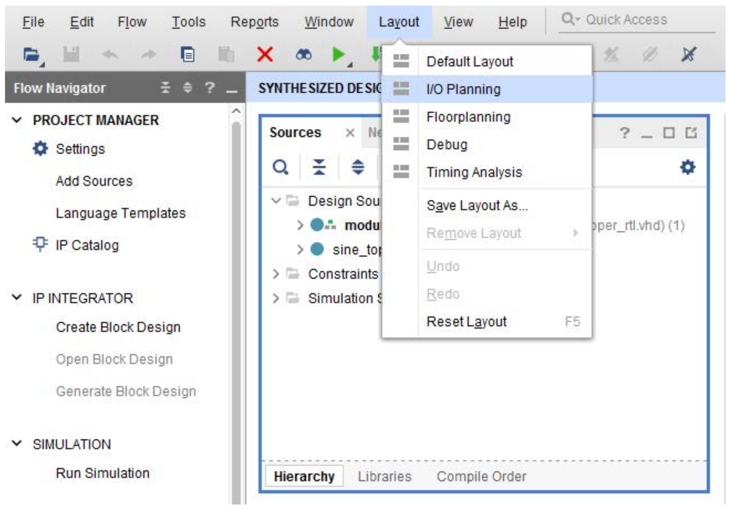


Fig 1.1: I/O Planning option

This will change the layout from the Default view to the I/O Planning view, see Fig 1.2.

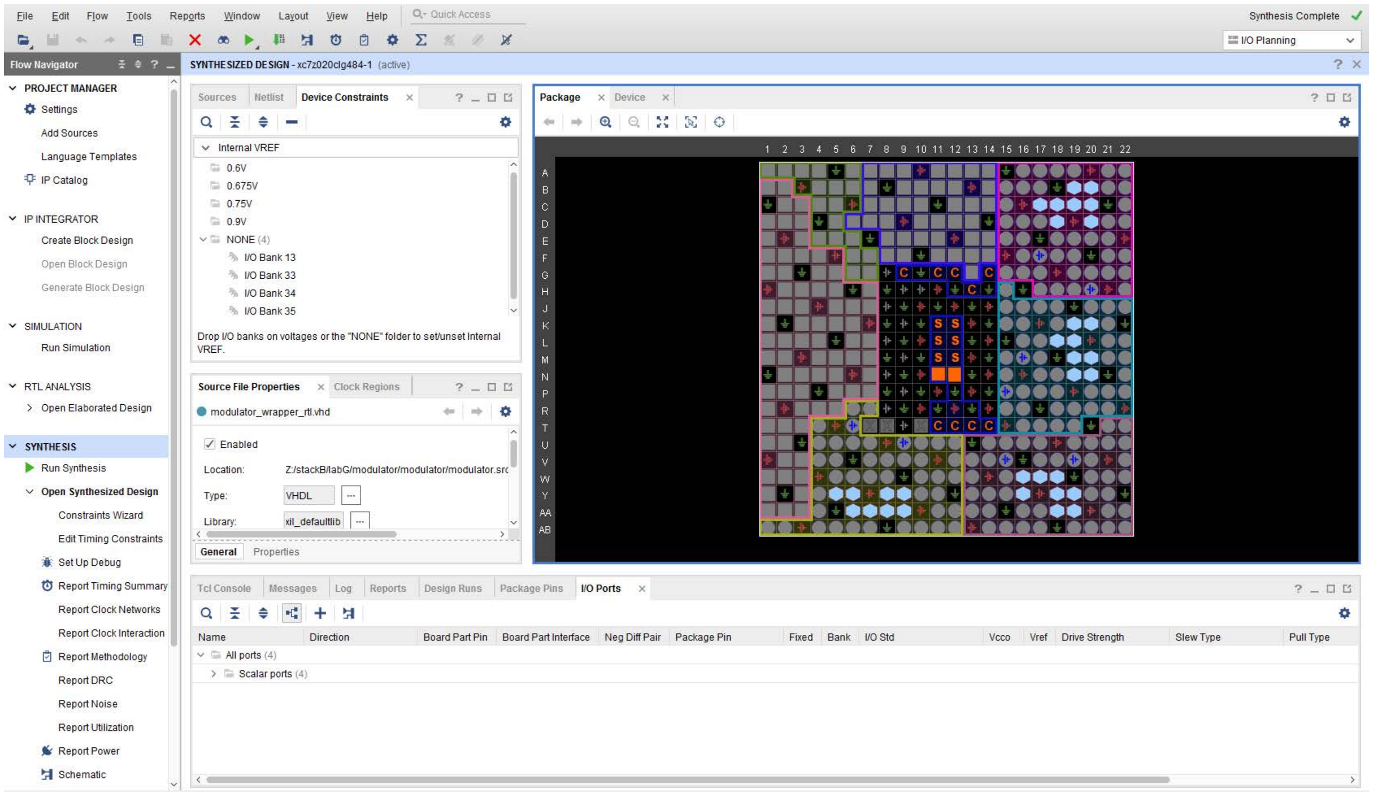
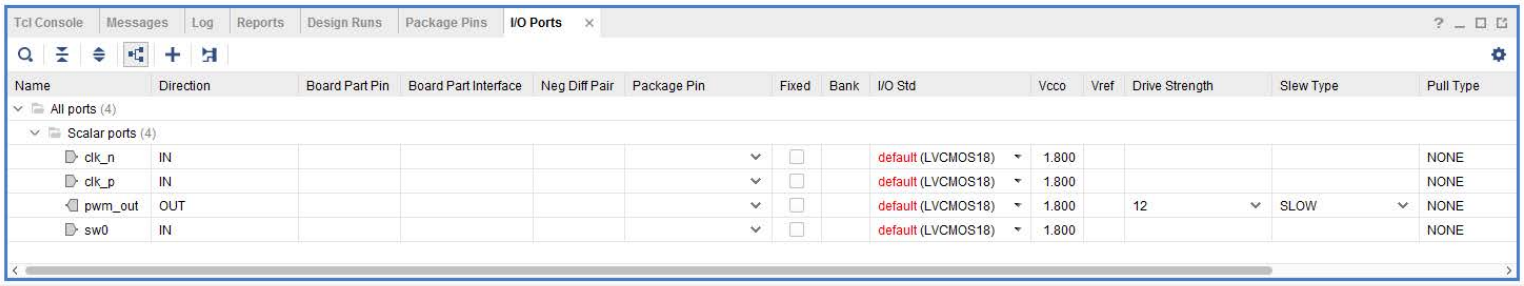


Fig 1.2: I/O Planning View

The main window of the I/O Planning view displays the package view of the ZedBoard device, which we will be using for this tutorial. Below the Package view, two additional tabs are populated. One tab displays the list of I/O ports of the design and the second tab displays the list of package pins on the device package.

Step 2. In the I/O Ports tab, click **Expand All** option, or just expand **Scalar ports**, which shows all I/O Ports of your design (see Fig 1.3).


Tcl Console Messages Log Reports Design Runs Package Pins I/O Ports

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (4)													
Scalar ports (4)													
clk_n	IN					☐		default (LVCMOS18)	1.800				NONE
clk_p	IN					☐		default (LVCMOS18)	1.800				NONE
pwm_out	OUT					☐		default (LVCMOS18)	1.800		12	SLOW	NONE
sw0	IN					☐		default (LVCMOS18)	1.800				NONE

Fig 1.3: I/O Ports tab

Note that none of the pins in this view have an assigned location.

Grey icons indicate unplaced ports, while yellow icons indicate placed ports. In Fig 1.3 we can see that all I/O ports are coloured grey, since none of them has been placed to a specific pin location. After we assign a pin location to each of the I/O ports they will be coloured yellow, as can be seen in Fig 1.5.

Step 3. To connect your logical with your physical ports, select one scalar port (for example **pwm_out**) and find in the user guide for your target board to which pin location you would like to connect your pwm_out port. For this tutorial, we should connect pwm_out port with one of the LED diodes that are physically present on the ZedBoard evaluation board. If you open ZedBoard user guide you can find that the FPGA pin location of the LD0 diode is **T22** and that the I/O standard that must be used is **LVCMOS33**.

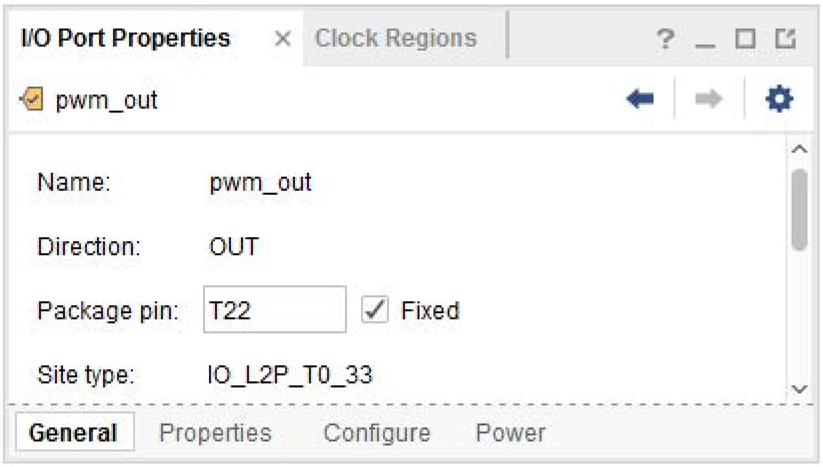
LVCMOS33 is a low voltage CMOS I/O standard using 3.3V power supply voltage.

Step 4. In the **I/O Ports** tab, click on the pwm_out's **Package Pin** column and choose **T22** as a pin location to connect the pwm_out port.

Step 5. Click on the pwm_out's **I/O Std** column and change the I/O standard from default LVCMOS18 to **LVCMOS33.**

Step 6. Leave all the other pwm_out's options unchanged, because they are default values.

Note: After assigning pin location and I/O standard for pwm_out port, we can notice that **I/O Port Properties** window popped up. This is the another way to change port properties, see Fig 1.4.



Name: pwm_out
Direction: OUT
Package pin: T22 ✓ Fixed
Site type: IO_L2P_T0_33

Fig 1.4: I/O Port Properties window

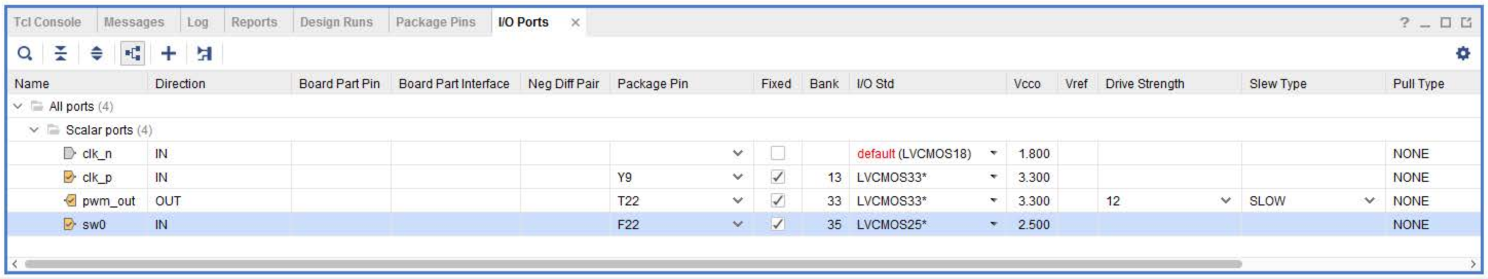
Step 7. Repeat these configuration steps for the remaining ports using the pin locations and necessary I/O standards information shown below:

- **clk_p** - pin location: Y9, I/O standard: LVCMOS33
- **sw0** - pin location: F22, I/O standard: LVCMOS25

Note: All this information has been extracted from the user guide for the ZedBoard evaluation board.

LVCMOS25 is a low voltage CMOS I/O standard using 2.5V power supply voltage.

Note: After all modifications, the **I/O Ports** tab should look like as it is shown in Fig 1.5.



Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (4)													
Scalar ports (4)													
clk_n	IN					☐		default (LVCMOS18)	1.800				NONE
clk_p	IN				Y9	✓	13	LVCMOS33*	3.300				NONE
pwm_out	OUT				T22	✓	33	LVCMOS33*	3.300		12	SLOW	NONE
sw0	IN				F22	✓	35	LVCMOS25*	2.500				NONE

Fig 1.5: I/O Ports tab with assigned pin locations and I/O standards

Note that **clk_n** port doesn't have an assigned pin location and I/O standard. This is because **clk_n** port is the differential input pair of **clk_p** port and our target ZedBoard evaluation board doesn't have a differential reference clock signal.

As pins or banks are selected, the corresponding pins or banks become highlighted in the other views. This makes it easier to see that the pins assigned in each bank meet the I/O banking rules and are grouped appropriately.

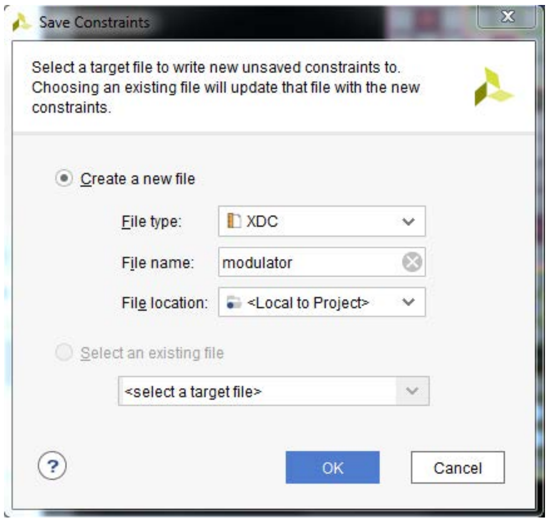
As you drag across the package view, yellow icons indicate assigned pins, grey icons indicate unassigned pins and both displayed indicates assigned I/O banks.

In the Package view you can also notice that:

- the coloured areas between the pins display the I/O banks
- the clock pins are shown as grey hexagons
- the clock-capable pins are shown as blue hexagons
- the power pins (VCC) are shown as red squares
- the ground pins (GND) are shown as green squares

Step 8. When you are finished with the placement constraints, click **the save button.**

Step 9. In the **Save Constraints** dialog box, type the name of the constraints file in the **File name** field. In our case, the name will be **modulator**, as seen below in Fig 1.6.



Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.

[● Create a new file]

File type: XDC

File name: modulator

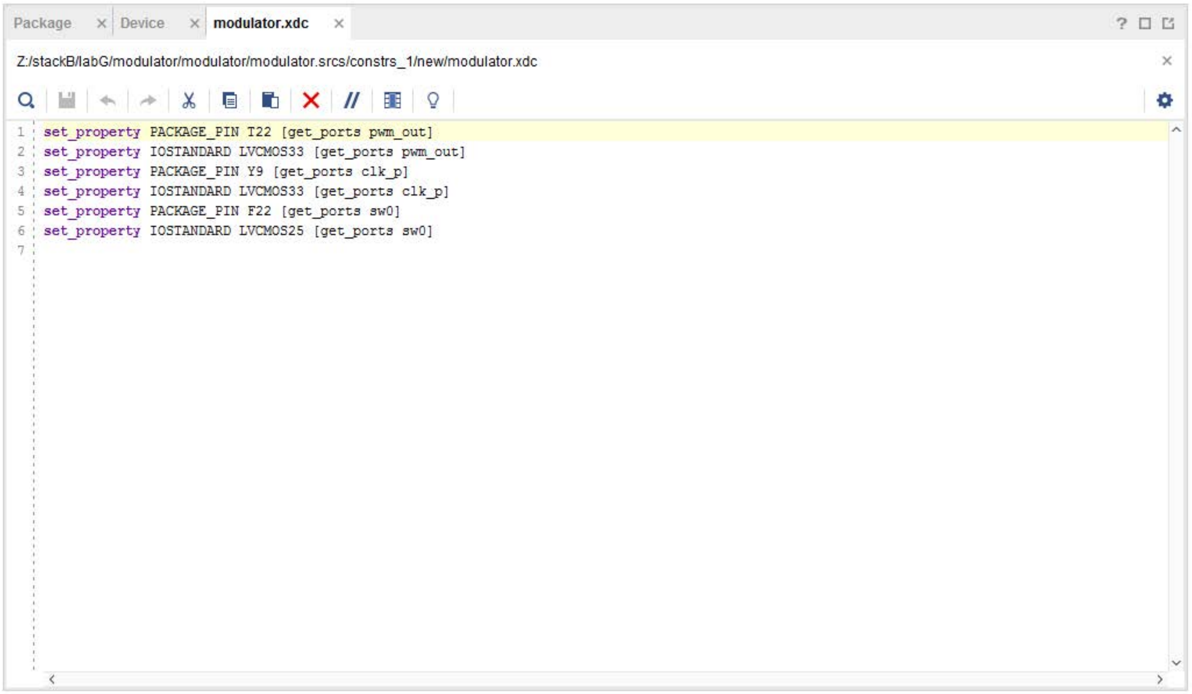
File location: <Local to Project>

○ Select an existing file

<select a target file>

Fig 1.6: Save Constraints dialog box

Step 10. Double-click on the **modulator.xdc** file to open it, see Fig 1.7.


Package
Device
modulator.xdc
Z:/stackB/labG/modulator/modulator/modulator.srcs/constrs_1/new/modulator.xdc

```
set_property PACKAGE_PIN T22 [get_ports pwm_out]
set_property IOSTANDARD LVCMOS33 [get_ports pwm_out]
set_property PACKAGE_PIN Y9 [get_ports clk_p]
set_property IOSTANDARD LVCMOS33 [get_ports clk_p]
set_property PACKAGE_PIN F22 [get_ports sw0]
set_property IOSTANDARD LVCMOS25 [get_ports sw0]
```

Fig 1.7: modulator.xdc file with physical constraints

In the **modulator.xdc** constraints file you can see assigned pin locations and I/O standards for each logical port of our design. For each logical port two constraints are necessary:

- First constraint connects selected logical port (by using ***get_ports*** Tcl command) with specified pin location (by setting the PACKAGE_PIN property, using ***set_property*** Tcl command).
- Second constraint sets the I/O standard that should be used for selected logical port by setting the IOSTANDARD property, using ***set_property*** Tcl command.

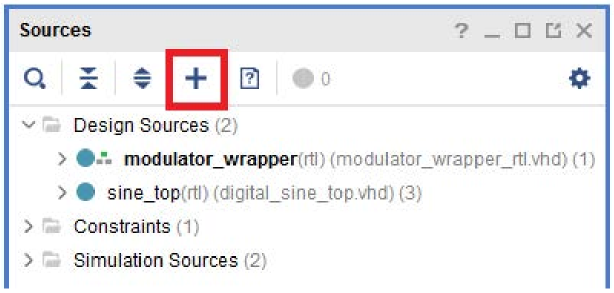
Creating a XDC File using Vivado Text Editor:

Here are the steps for creating XDC file using Vivado text editor:

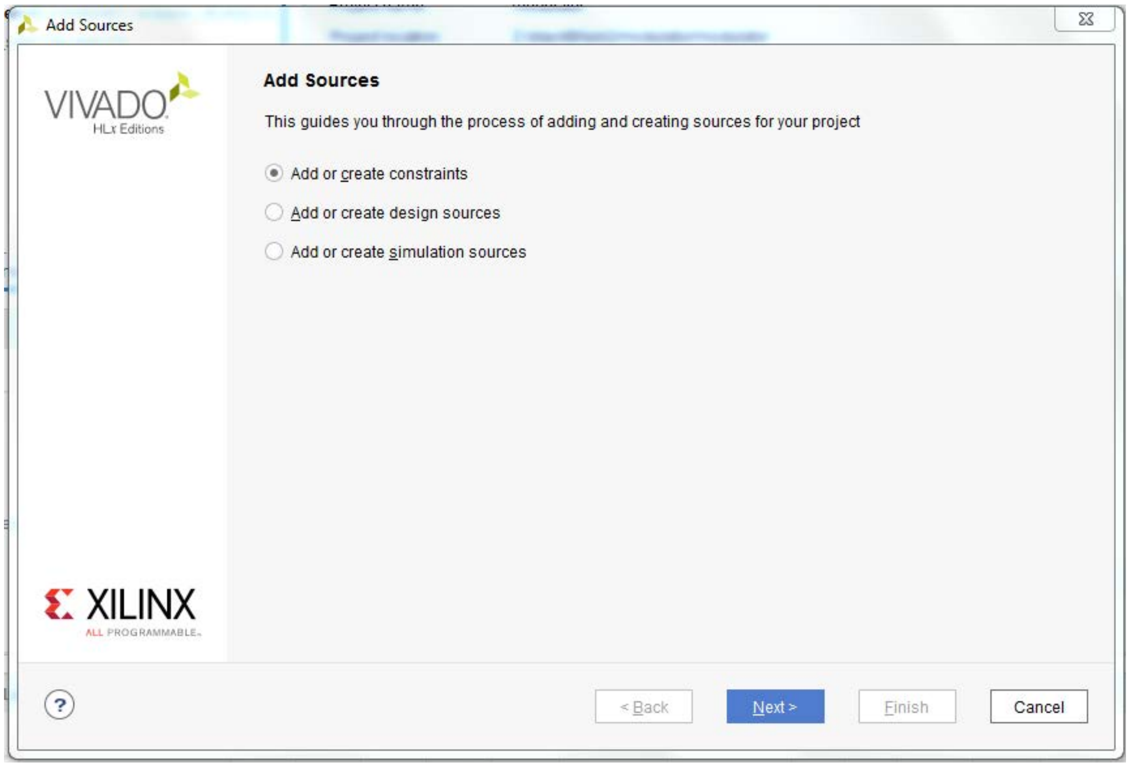
Step 1. **Optional:** Launch **Vivado IDE** (if it is not already launched)

Step 2. **Optional:** Open "Modulator" project (**modulator.xpr**) (if it is not already opened)

Step 3. In the Sources window, click the **Add Sources** button, shown below.


Sources
0
Design Sources (2)
modulator_wrapper(rtl) (modulator_wrapper_rtl.vhd) (1)
sine_top(rtl) (digital_sine_top.vhd) (3)
Constraints (1)
Simulation Sources (2)

Step 4. In the **Add Sources** dialog box, select **Add or create constraints** and click **Next.**



Add Sources

This guides you through the process of adding and creating sources for your project

(●) Add or create constraints
( ) Add or create design sources
( ) Add or create simulation sources

Fig 1.9: Save Constraints dialog box

Step 5. In the **Add or Create Constraints** dialog box, click **Create File**. Name the file "modulator" and click OK to add the constraint file to the project with default settings.

Step 6. The **modulator.xdc** file should appear in the Sources view, under Constraints. Double click on the file to open it in the Text Editor.

Step 7. Type or paste the constraints shown in Fig 1.7 into the file and click the **Save** button.

Defining Timing Constraints

Prior to implementation, there are physical and timing constraints that need to be defined. In the previous steps we have defined physical constraints. Now, it's time to define timing constraints also.

To define timing constraints you can choose between two approaches:

- using **Constraints Wizard** , or
- using **Constraints Editor**

Defining timing constraints using Constraints Wizard

As we already explained, the Vivado IDE provides Timing Constraints wizard to walk you through the process of creating and validating timing constraints for the design. The Timing Constraints wizard analyzes the gate level netlist and finds missing constraints. It is only available in the synthesized and implemented designs.

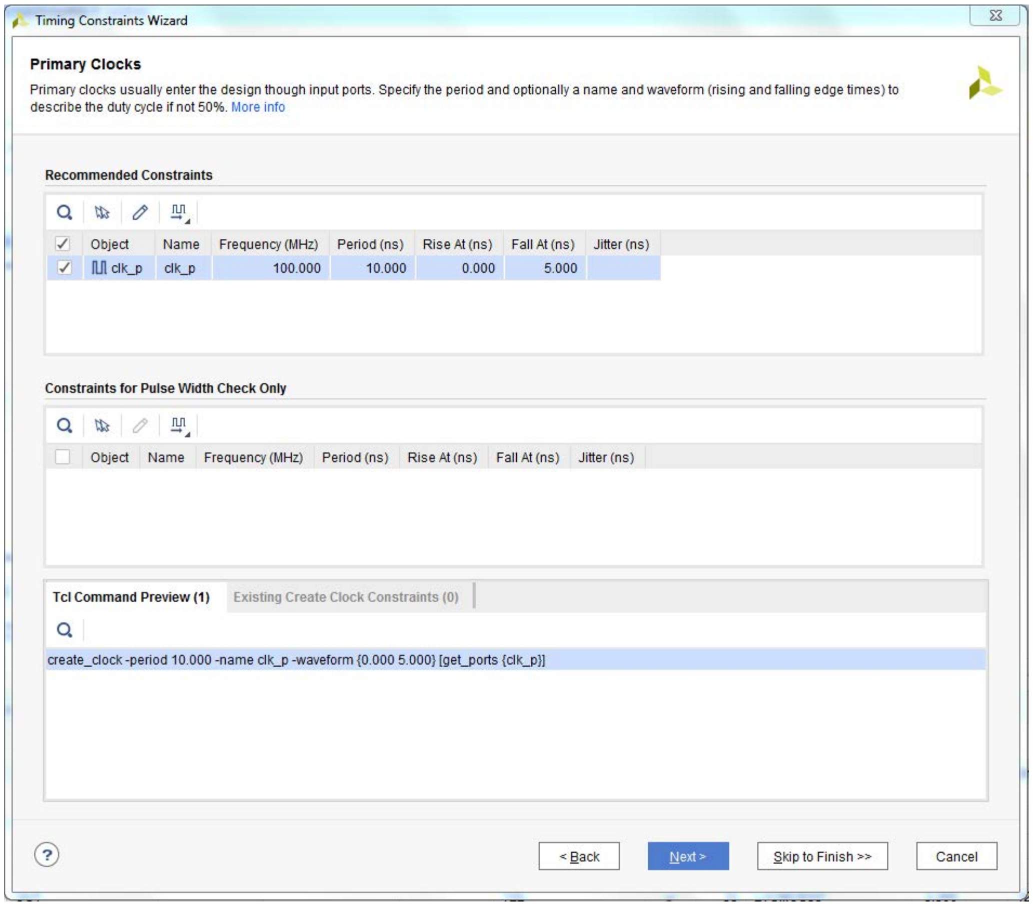
To define timing constraints using Constraints Wizard, follow the next steps:

Step 1. In the Flow Navigator, under the Synthesis Design section, select first offered **Constraints Wizard** command.

Step 2. The introduction page opens. This page describes the types of constraints that the wizard will create: Clocks, Input and Output Ports, and Clock Domain Crossings. After reading the page, click **Next** to continue.

Step 3. In the **Primary Clocks** dialog box, Timing Constraints Wizard will display all the clock sources with a missing clock definition. Specify **100 MHz** frequency for the **clk_p** clock and wizard will automatically calculate values for Period (ns), Rise At (ns), Fall At (ns) and Jitter (ns) fields, see Fig 1.10. Click **Next** to continue.

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 finds 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.


Timing Constraints Wizard
Primary Clocks
Primary clocks usually enter the design though input ports. Specify the period and optionally a name and waveform (rising and falling edge times) to describe the duty cycle if not 50%. More info
Recommended Constraints
Constraints for Pulse Width Check Only
Tcl Command Preview (1)
Existing Create Clock Constraints (0)
< Back
Next >
Skip to Finish >>
Cancel

✓	Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)
✓	clk_p	clk_p	100.000	10.000	0.000	5.000	

	Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)

```
create_clock -period 10.000 -name clk_p -waveform {0.000 5.000} [get_ports {clk_p}]
```

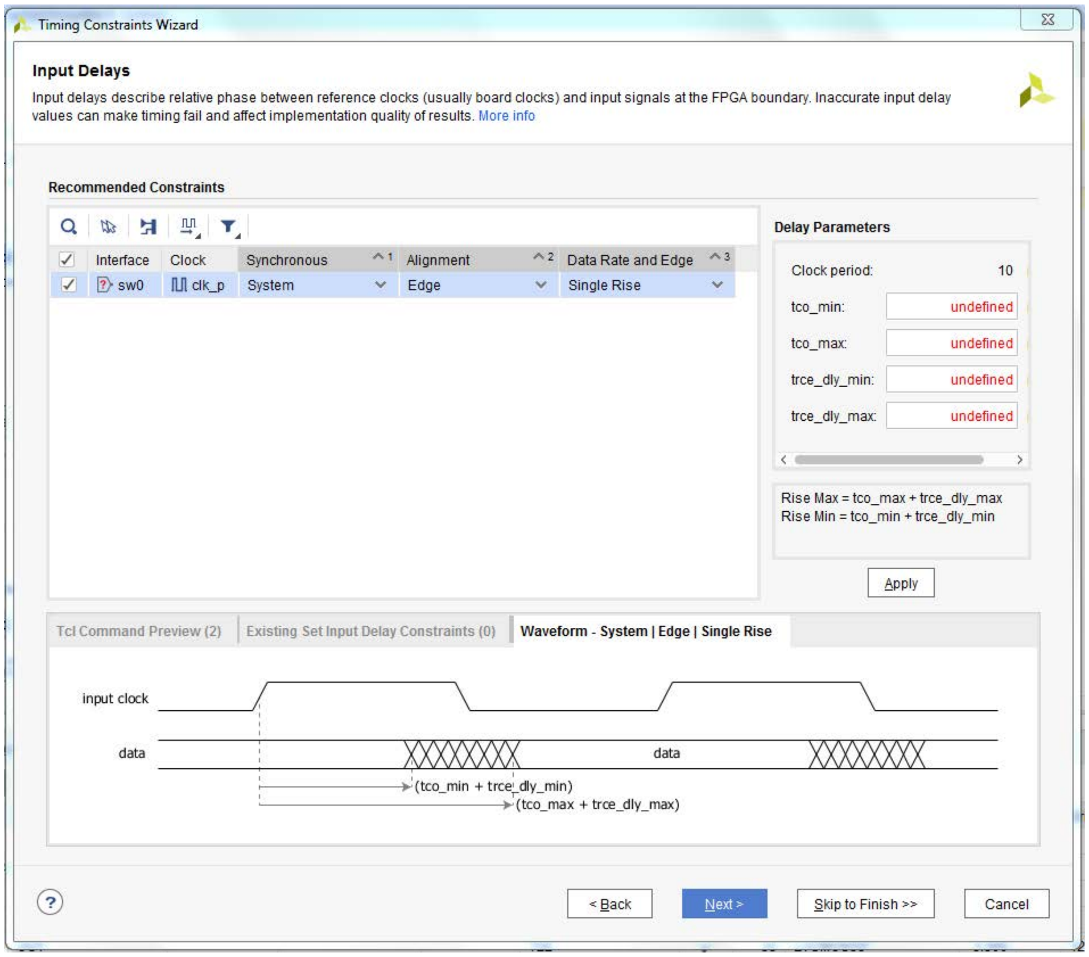
Fig 1.10: Primary Clocks dialog box

Step 4. The primary clock constraints have been added to the design. Next, the wizard looks for unconstrained generated clocks. Generated clocks are derived from primary clocks in the FPGA fabric. In our design, the wizard determined that there are no unconstrained generated clocks. In the **Generated Clocks** dialog box, click **Next** to continue.

Step 5. Next, the wizard looks for forwarded clocks. 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 our design. In the **Forwarded Clocks** dialog box, click **Next** to continue.

Step 6. Next, the wizard looks for external feedback delays. 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 delay in our design. In the **External Feedback Delays** dialog box, click **Next** to continue.

Step 7. Next, the wizard looks at the input delays. Fig 1.11 shows the **Input Delays** page of the Timing Constraints wizard. There are three sections on the page.



Timing Constraints Wizard

Input Delays

Input delays describe relative phase between reference clocks (usually board clocks) and input signals at the FPGA boundary. Inaccurate input delay values can make timing fail and affect implementation quality of results. More info

Recommended Constraints

✓	Interface	Clock	Synchronous	Alignment	Data Rate and Edge
✓	sw0	clk_p	System	Edge	Single Rise

Delay Parameters

Clock period:	10
tco_min:	undefined
tco_max:	undefined
trce_dly_min:	undefined
trce_dly_max:	undefined

Rise Max = tco_max + trce_dly_max
Rise Min = tco_min + trce_dly_min

Fig 1.11: Input Delays dialog box

- First section shows 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 constraints the input.
- In the second section you provide the delay values for the template. This section will change depending on the template chosen in the first section.
- In the third section there are three tabs:
  - **Tcl Command Preview** - previews the Tcl commands that will be used to constrain the design
  - **Existing Set Input Delay Constraints** - shows input delay constraints that exist in the design
  - **Waveform** - displays the waveform associated with the template

Step 8. Uncheck the ***sw0*** input port in the first section of the Input Delays dialog box, because we don't need a delay period for this input port. When you have successfully finished with all input constraint values, click **Next.**

Step 9. Next, the **Output Delays** page of the wizard displays all the outputs that are unconstrained in the design. The page layout is very similar to the inputs page. Uncheck the pwm_out output port in the first section of the Output Delays dialog box, because we don't need a delay period for this output port either. When you have successfully finished with all output constraint values, click **Next.**

Step 10. The wizard now 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. Our design doesn't contain any combinational paths. In the **Combinational Delays** dialog box, click **Next** to continue.

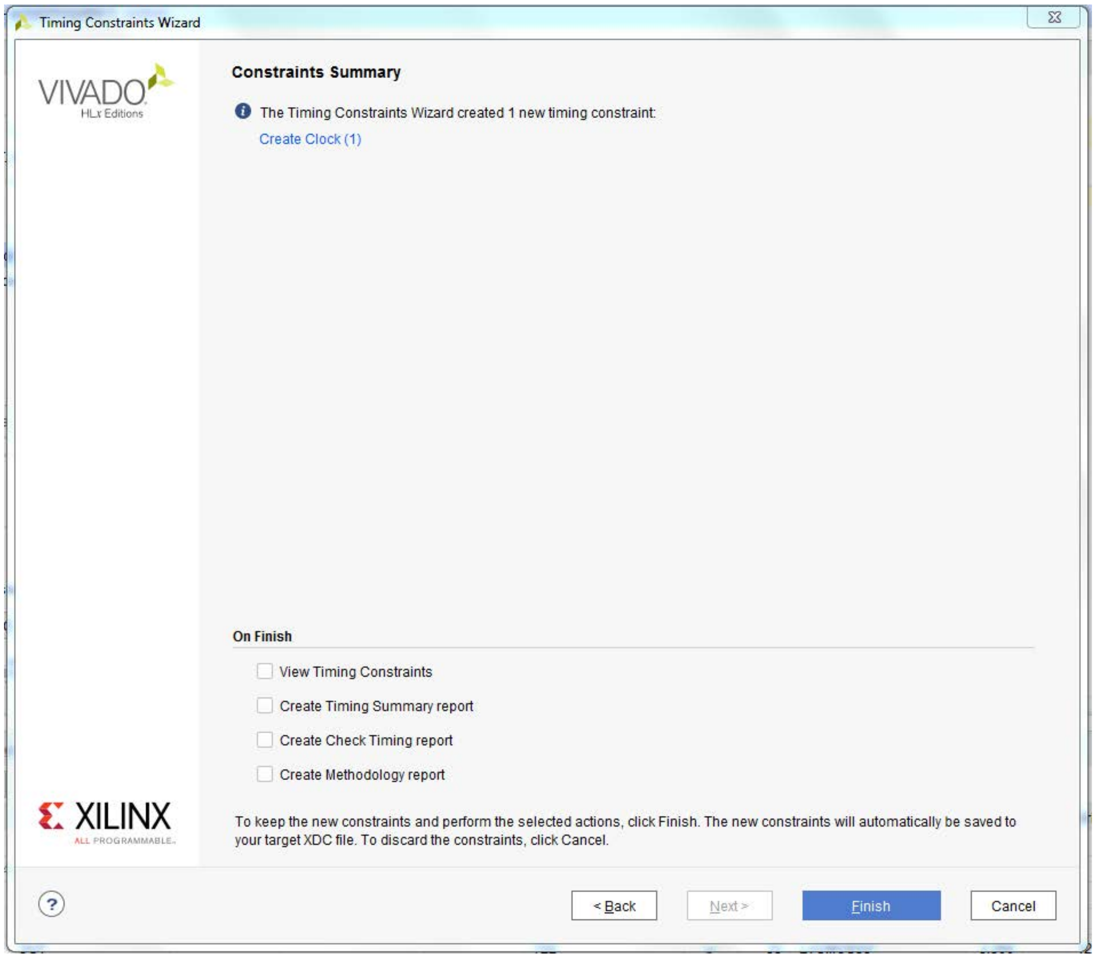
Step 11. Physically exclusive clock groups are clocks that do not exit in the design at the same time. There are no unconstrained physically exclusive clock groups in our design. In the **Physically Exclusive Clock Groups** dialog box, click **Next** to continue.

Step 12. 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 our design. In the **Logically Exclusive Clock Groups with No Interaction** dialog box, click **Next** to continue.

Step 13. 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 our design. In the **Logically Exclusive Clock Groups with Interaction** dialog box, click **Next** to continue.

Step 14. The **Asynchronous Clock Domain Crossings** page recommends constraints for safe clock domain crossings. Our design does not contain any unconstrained clock domain crossings. Click **Next** to continue.

Step 15. The **Constraints Summary** page is the final page of the Timing Constraints wizard, see Fig 1.12. 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 using the check boxes in the wizard. Click **Finish** to complete the Timing Constraints wizard.



Timing Constraints Wizard

Constraints Summary

The Timing Constraints Wizard created 1 new timing constraint:

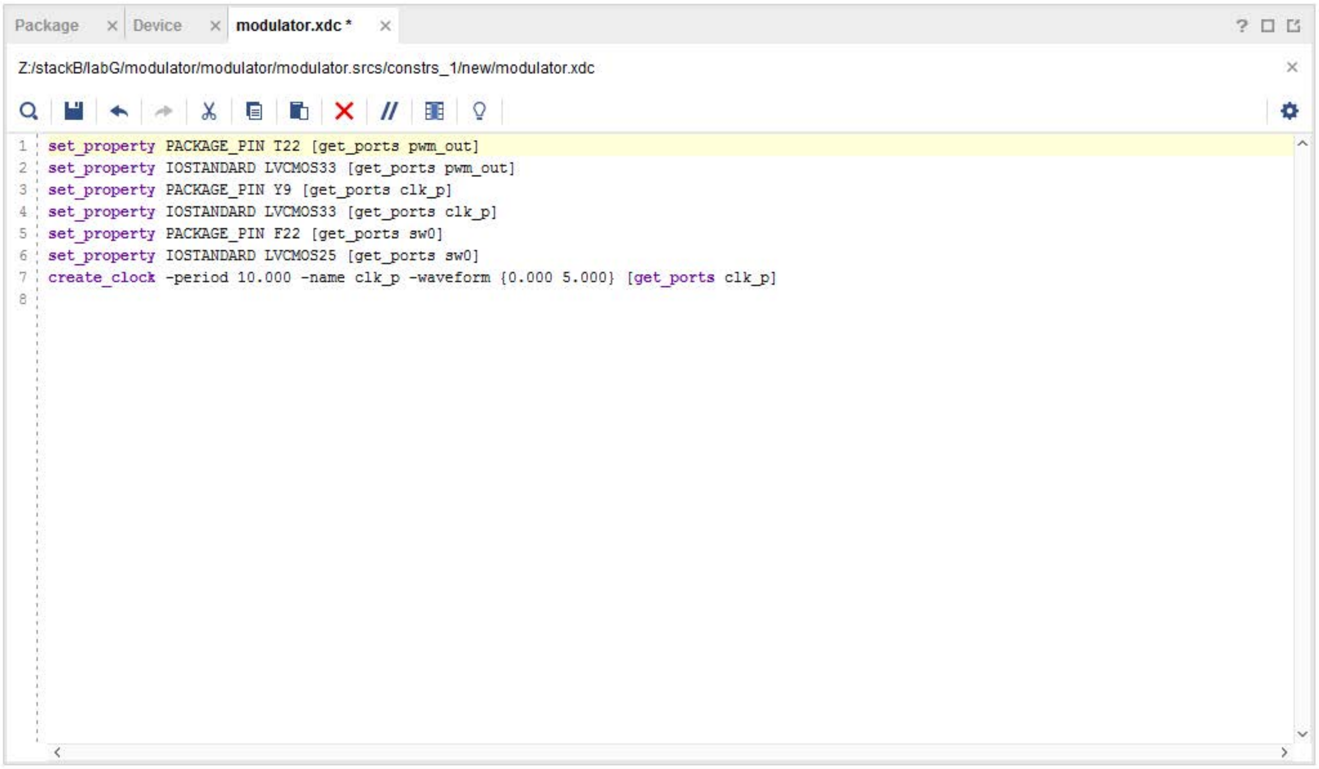
Create Clock (1)

On Finish

- [ ] View Timing Constraints
- [ ] Create Timing Summary report
- [ ] Create Check Timing report
- [ ] Create Methodology report

To keep the new constraints and perform the selected actions, click Finish. The new constraints will automatically be saved to your target XDC file. To discard the constraints, click Cancel.

Fig 1.12: Constraints Summary dialog box



Z:/stackB/labG/modulator/modulator/modulator.srcs/constrs_1/new/modulator.xdc

```
set_property PACKAGE_PIN T22 [get_ports pwm_out]
set_property IOSTANDARD LVCMOS33 [get_ports pwm_out]
set_property PACKAGE_PIN Y9 [get_ports clk_p]
set_property IOSTANDARD LVCMOS33 [get_ports clk_p]
set_property PACKAGE_PIN F22 [get_ports sw0]
set_property IOSTANDARD LVCMOS25 [get_ports sw0]
create_clock -period 10.000 -name clk_p -waveform {0.000 5.000} [get_ports clk_p]
```

Fig 1.13: Constraints File showing Timing Constraint

Defining timing constraints using Constraints Editor

To define timing constraints using Constraints Editor, follow the next steps:

Step 1. Select **Window -> Timing Constraints** option from the main Vivado IDE menu to open the Timing Constraints window, see Fig 1.14, or

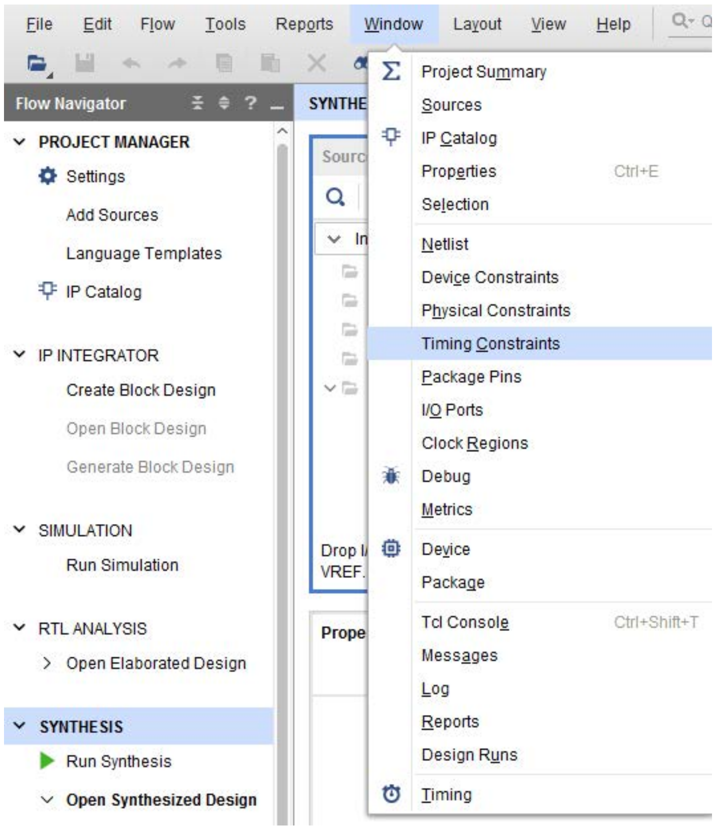


Fig 1.14: Timing Constraints option

select in the Flow Navigator, under the Synthesis Design section, second offered **Edit Timing Constraints** command.

The **Timing Constraints** window appears in the main window of the Vivado IDE, see Fig 1.15.

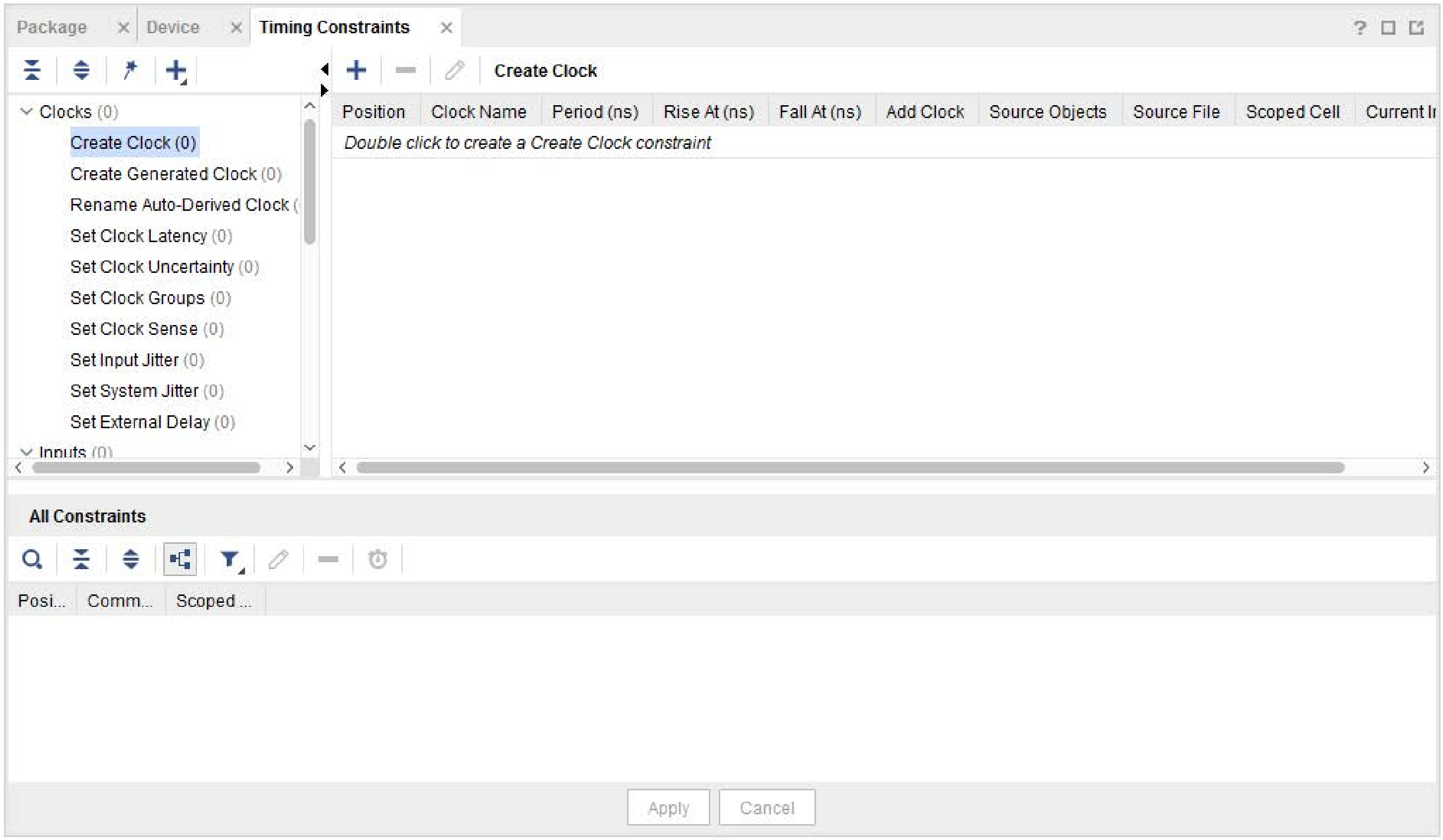


Fig 1.15: Timing Constraints window

There are three sections in the Timing Constraints window:

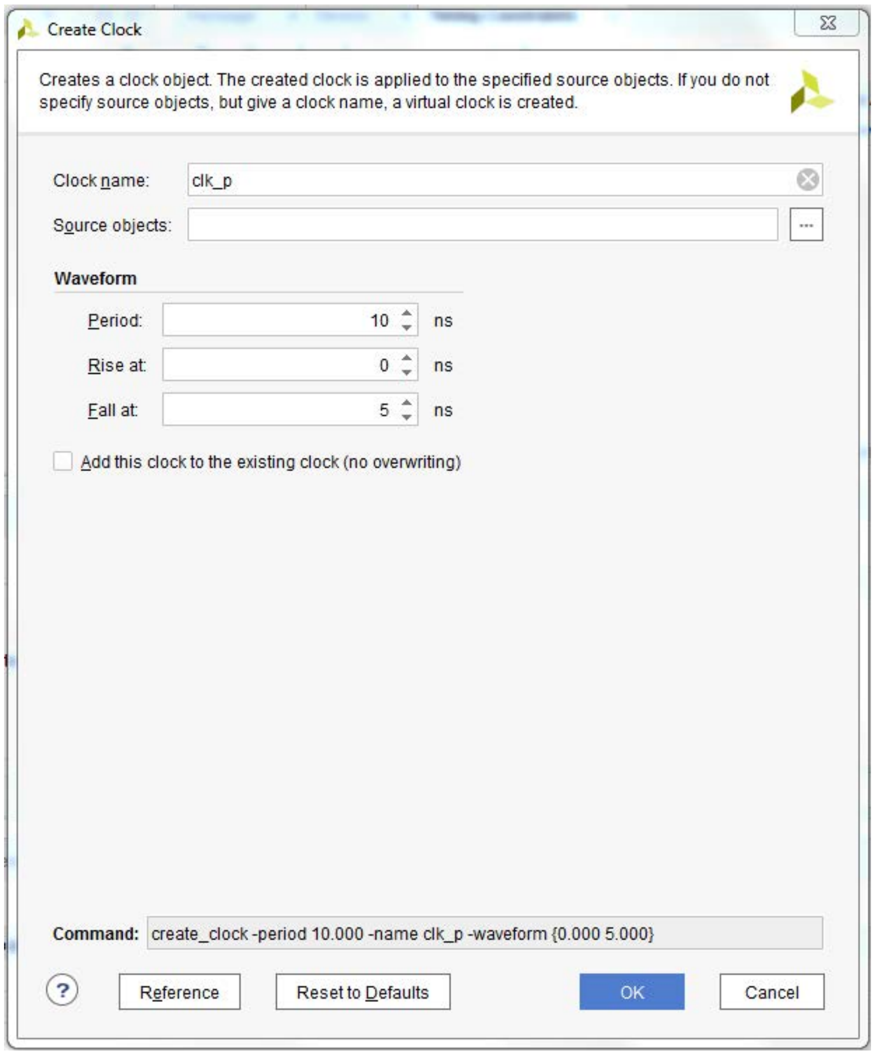
- **Constraints tree view** - displays standard timing constraints, grouped by category. Double-clicking a constraint in this section opens a new window to help you define the selected constraint.
- **Constraints Spreadsheet** - displays timing constraints of the type currently selected in the Constraints tree view. If you prefer, you can use this to directly define or edit constraints instead of using Constraints wizard.
- **All Constraints** - displays all the timing constraints that currently exist in the design.

The Timing Constraints wizard identifies missing clocks, I/O delays, and clock domain crossings exceptions, but it doesn't handle general timing exceptions. We will use the timing constraints editor to create the exceptions that exist in the design.

Define the primary clock constraint by creating a clock object with a specified period. The modulator design has a 100 MHz clock supplied through differential clock input ports on the FPGA. First define the primary clock object for the design and then define a PERIOD constraint for the clock object.

Step 2. In the **Constraints tree view** window of the Timing Constraint editor, double-click on the **Create Clock (0)** option under the Clocks (0) section to create a clock constraint.

Step 3. In the **Create Clock** dialog box, enter ***clock_name*** (**clk_p**) in the **Clock Name** field, see Fig 1.16.


Create Clock

Creates a clock object. The created clock is applied to the specified source objects. If you do not specify source objects, but give a clock name, a virtual clock is created.

Clock name: clk_p

Source objects:

Waveform

Period: 10 ns

Rise at: 0 ns

Fall at: 5 ns

☐ Add this clock to the existing clock (no overwriting)

Command: `create_clock -period 10.000 -name clk_p -waveform {0.000 5.000}`

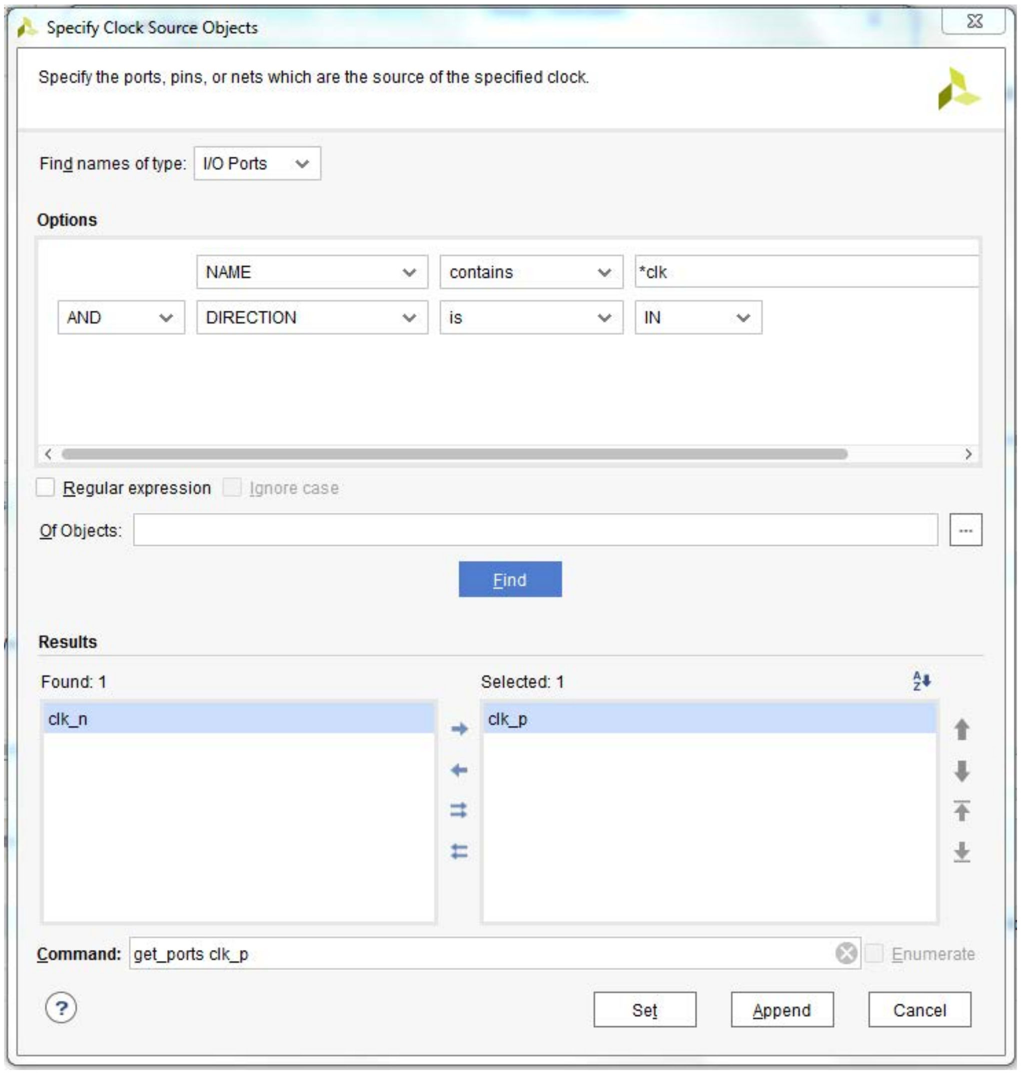
Fig 1.16 Create Clock dialog box

Step 4. Click the icon next to the **Source objects** field and **Specify Clock Source Objects** dialog box will appear, see Fig 1.17.

Note: This step is important to associate the clock input port to the clock definition.

Step 5. In the **Specify Clock Source Objects** dialog box (see Fig 1.17), do the following:

- Ensure that **I/O Ports** is selected from the **Find names of type** drop-down list
- Enter **clk** in the empty search field
- Click **Find**
- In the **Find results: 2** section, select **clk_p**
- **Click the -> icon to select clk_p**
- Click Set



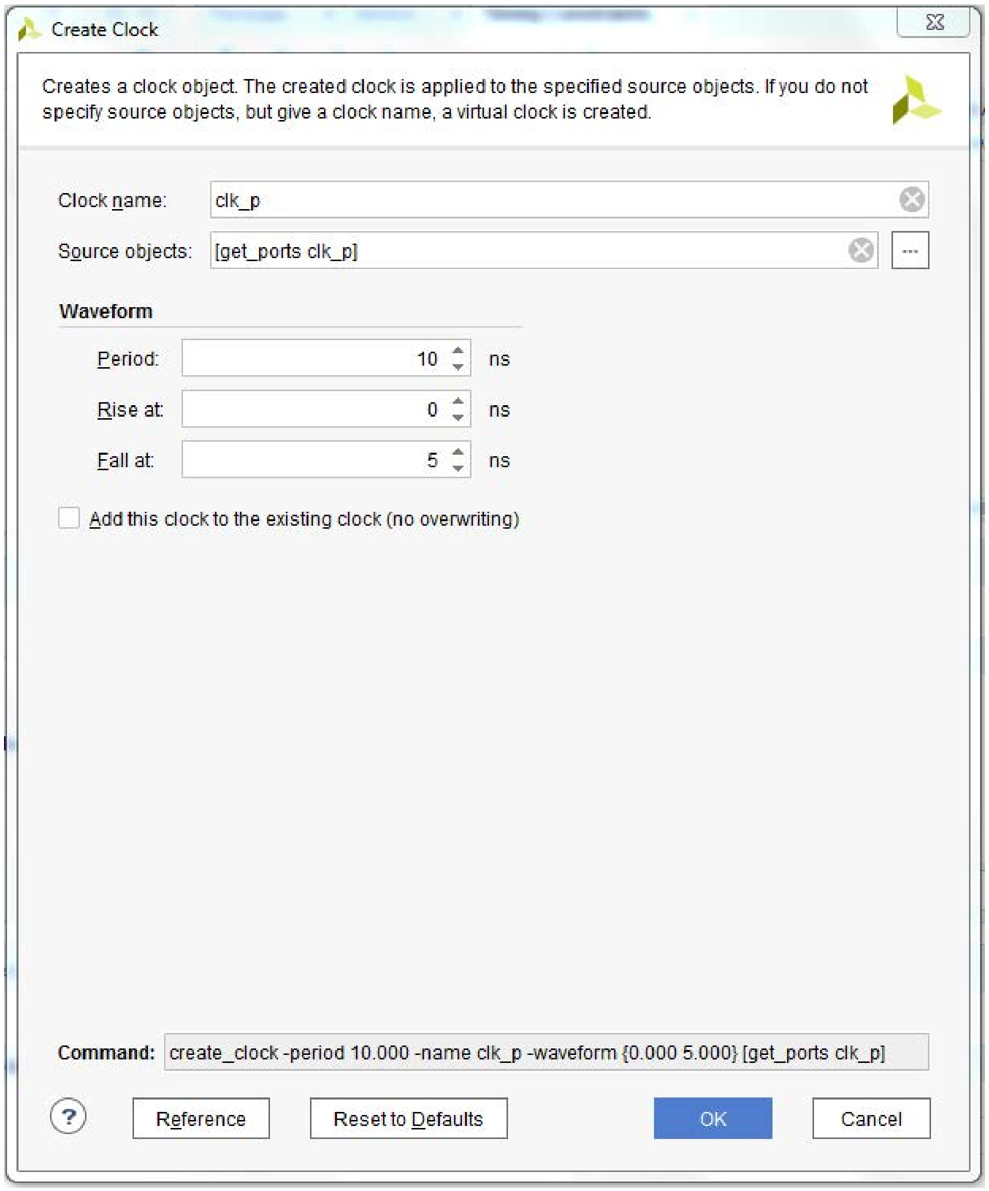
Command: get_ports clk_p

Fig 1.17: Specify Clock Source Objects dialog box

Step 6. In the **Create Clock** dialog box, specify the period by setting the period property of the clock. In this step, you will describe the period property and review the waveform details of the clock objects, see Fig 1.18:

- Enter **10 ns** in the **Period** field in the Waveform section, because 10 ns is the period of the 100 MHz input clock signal
- Ensure that the **Rise at** and **Fall at** fields are set to **0** and **5** respectively, which means that the duty cycle of the input clock signal will be 50%.

- Click **OK** to create the clock constraint.



Create Clock

Creates a clock object. The created clock is applied to the specified source objects. If you do not specify source objects, but give a clock name, a virtual clock is created.

Clock name: clk_p

Source objects: [get_ports clk_p]

Waveform

Period: 10 ns

Rise at: 0 ns

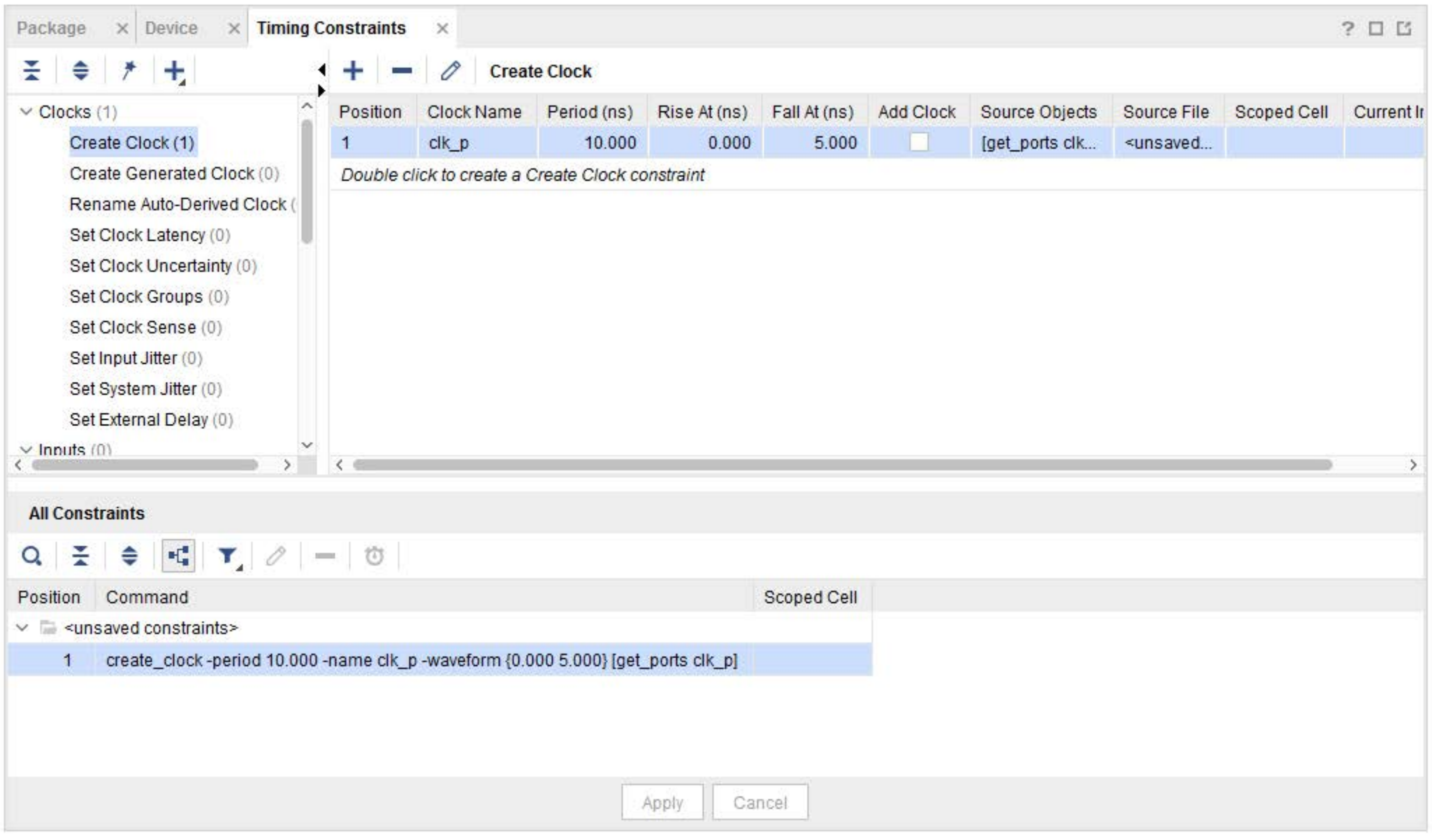
Fall at: 5 ns

Add this clock to the existing clock (no overwriting)

Command: `create_clock -period 10.000 -name clk_p -waveform {0.000 5.000} [get_ports clk_p]`

Fig 1.18: Create Clock dialog box after specifying the period for the clk_p

The Timing Constraints window now displays the timing constraint applied to the design, see Fig 1.19.



Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects	Source File	Scoped Cell	Current In
1	clk_p	10.000	0.000	5.000	☐	[get_ports clk...	<unsaved...		

Double click to create a Create Clock constraint

Position	Command	Scoped Cell
	<unsaved constraints>	
1	create_clock -period 10.000 -name clk_p -waveform {0.000 5.000} [get_ports clk_p]	

Fig 1.19: Timing Constraints window with the create_clock constraint

Notice that the create_clock XDC command for the created clock is also displayed in the All Constraints view of the Timing Constraints window.

All the timing constraints that have been run are applied to the design that is loaded in the memory. The applied constraints can be saved by writing them to the XDC file. All the timing constraints applied to the design are available in the **All Constraints** view of the Timing Constraints window, see Fig 1.19.

Step 7. To save your timing constraints to the **modulator.xdc** constraints file, click the **Save** button.

If you want to verify that the timing constraints have been applied to the **modulator.xdc** file, do the following:

1. If the **modulator.xdc** file is already open, click the **Reload** link in the banner of the ***modulator.xdc*** tab to reload the constraints file from disk.
2. If the **modulator.xdc** file is not open, select the **Sources** window, **Hierarchy** view
3. Expand **Constraints** folder

4. Double-click on the **modulator.xdc** file to open the file and you should see that your timing constraints were saved to the XDC file.

Implementation

About the Vivado Implementation Process

The Vivado Design Suite enables implementation of UltraScale FPGA and Xilinx 7 Series FPGA designs from the variety of design sources, including RTL designs, netlist designs and IP centric design flows.

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

The Vivado implementation is a timing-driven flow. It supports industry standard Synopsys Design Constraints (SDC) commands to specify design requirements and restrictions, as well as additional commands in the Xilinx Design Constraints (XDC) format.

The Vivado implementation process includes logical and physical transformations of the design. The implementation process consists of the following sub-processes:

- **Opt Design: Netlist Optimization**

  Optimizes the logical design to make it easier to fit onto the target Xilinx device:

  - Ensures optimal netlist for placement
  - Optional in non-project batch flow (but recommended)
  - Automatically enables in the project-based flow

  Because this is the first view of the assembled design (RTL and IP blocks), the design can usually be further optimized. The ***opt_design*** command is the next step and performs logic trimming, removing cells with no loads, propagating constant inputs, and combining LUTs for example LUTs in series that can be combined into fewer LUTs.

- **Power Opt Design: Power Optimization**

  Optimizes design elements to reduce the power demands of the target Xilinx device:

  - Disabled in project-based flow (can be set with implementation settings in GUI)
  - Power optimization includes a fine-grained clock gating solution that can reduce dynamic power by up to 30%
  - Intelligent clock gating optimizations are automatically performed on the entire design and will generate no changes to the existing logic or clocks
  - Algorithm performs analysis on all portions of the design

  Note: This step is optional.

- **Place Design: Placer**

  Places the design onto the target Xilinx device:

  - Project-based flow (included in implementation stage)

    - Non-project batch flow (place_design)
    - Can use an input XDEF as a starting point for placement

- **Phys Opt Design: Physical Synthesis**

  Optimizes design timing by replicating drivers of high-fanout nets to distribute the loads:

    - Post-placement timing-driven optimization (replicates and places drivers of high fanout nets with negative slack)
    - More features coming in future releases (register retiming)
    - Available in all flows and can be de-activated in the GUI
    - phys_opt_design (run between place_design and route_design)

  Note: This step is optional.

- **Route Design: Router**

  Routes the design onto the target Xilinx device:

    - Project-based flow (included in implementation stage)
    - Non-project batch flow (route_design)
    - Router reporting (report_route_status command)
    - Check route status of individual nets

The Vivado Design Suite includes a variety of design flows, and supports an array of design sources. In order to generate a bitstream that can be downloaded onto the FPGA device, the design must pass through implementation process.

Implementation is a series of steps that takes the logical netlist and maps it into the physical array of the target Xilinx device. These steps include:

- Logic optimization
- Placement of logic cells
- Routing of connections between cells

Run Implementation

Now we will run implementation process from the Flow Navigator, which will trigger synthesis followed by implementation in one step.

To run the implementation process, please do the following:

Step 1. In the **Flow Navigator**, click **Run Implementation** command and wait for implementation to be completed, see Fig 1.20.

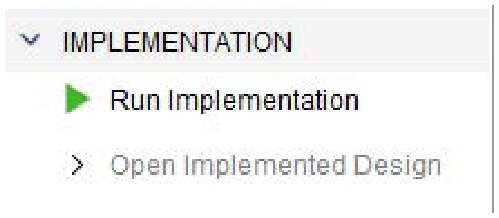
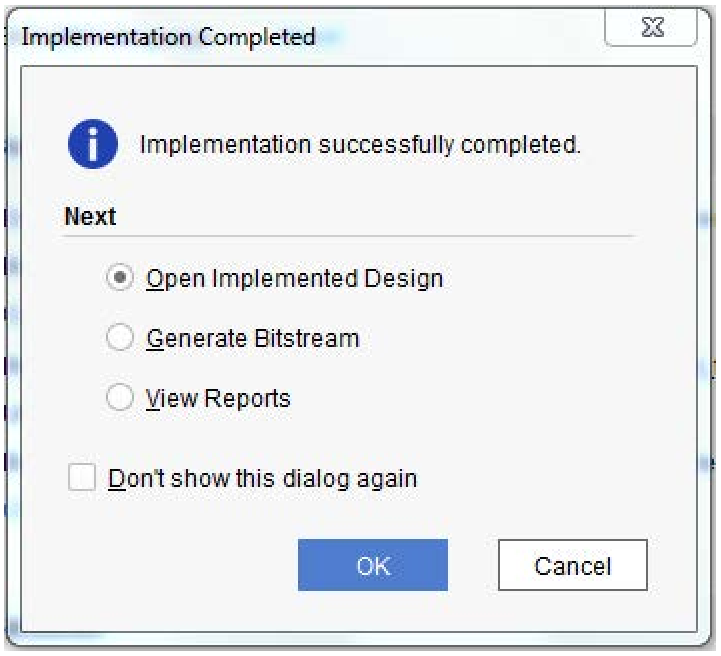


Fig 1.20: Run Implementation command

Note: You can monitor the Implementation progress in the bar in the upper right corner of the Vivado IDE.

Step 2. After the implementation is completed, the **Implementation Completed** dialog box will appear, see Fig 1.21.



Implementation Completed

Implementation successfully completed.

Next

(●) Open Implemented Design

( ) Generate Bitstream

( ) View Reports

☐ Don't show this dialog again

Fig 1.21 Implementation Completed dialog box

Step 3. Select **Open Implementation Design** option in the **Implementation Completed** dialog box and click **OK** to open the implemented design.

After Implementation

After implementation process:

- Sources and Netlist tabs do not change. Now as each resource is selected, it will show the exact placement of the resource on the die (Instance Properties view will show specific details about the resource).
- Timing results have to be generated with the Report Timing Summary
- As each path is selected, the placement of the logic and its connections is shown in the Device view. This is the cross-probing feature that helps with static timing analysis.

After you have implemented the design (or opened a project that only contains an implemented design), the Flow Navigator changes again, see Fig 1.22. Flow Navigator is optimized to provide quick access to the options most frequently used after implementation (note that most of these reports are the same, except with true-timing information):

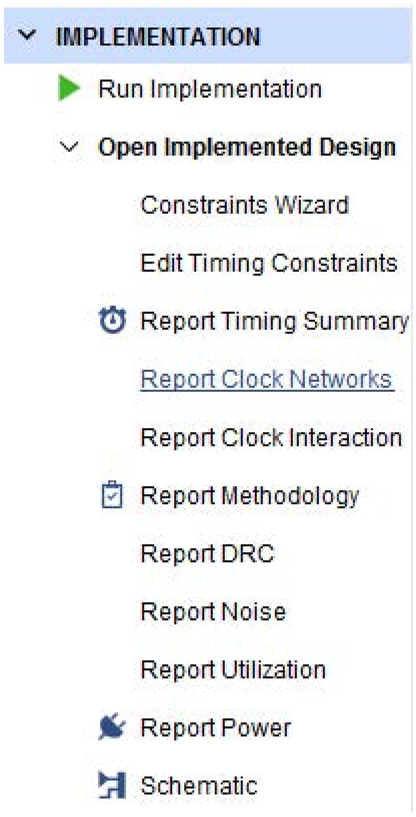
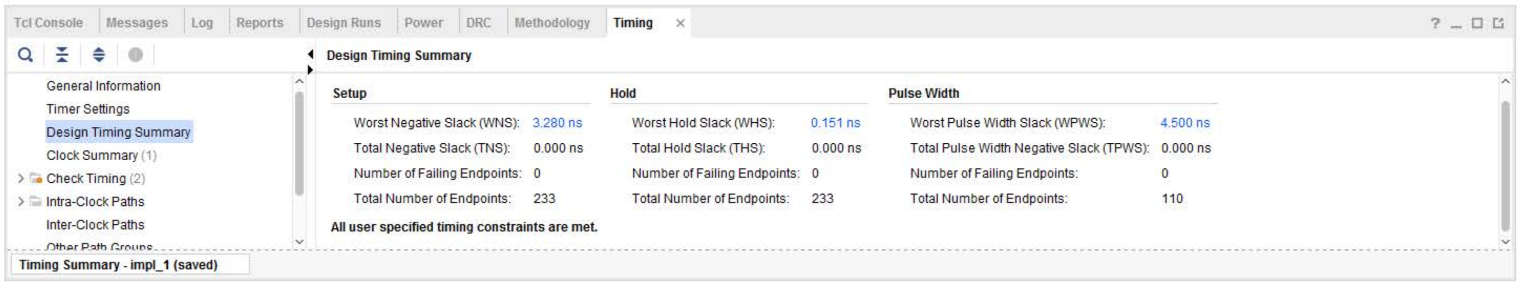


Fig 1.22: Implemented Design options

- ***Constraints Wizard***: Opens the Timing Constraints wizard
- ***Edit Timing Constraints***: Opens the Constraints viewer
- ***Report Timing Summary***: Generates a default timing report (using true timing information)
- ***Report Clock Networks***: Generates a clock tree for the design
- ***Report Clock Interaction***: Verifies constraint coverage on paths between clock domains
- ***Report Methodology***: Performs automated methodology checks and allows you to find design issues early in the design process
- ***Report DRC***: Performs design rule check on the entire design
- ***Report Noise***: Performs an SSO analysis of output and bidirectional pins in your design
- ***Report Utilization***: Generates a graphic version of the Utilization Report
- ***Report Power***: Provides detailed power analysis reports

Note that the Report Timing Summary is the most important default report because at this point what most designers are concerned about is meeting their timing objectives and only after completing an implementation does the designer know if they can actually do that.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.280 ns	Worst Hold Slack (WHS):	0.151 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	233	Total Number of Endpoints:	233	Total Number of Endpoints:	110

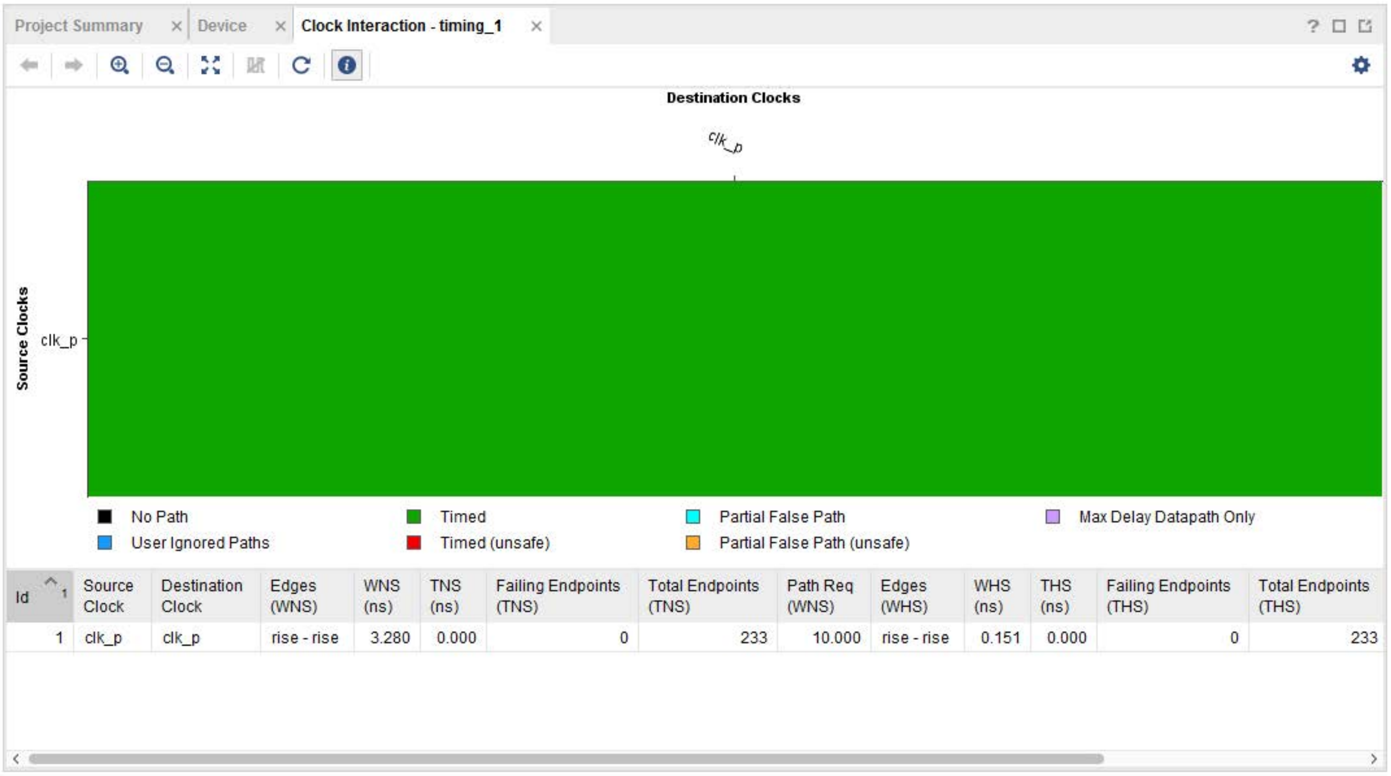
All user specified timing constraints are met.

Fig 1.23: Report Timing Summary tab

Step 1. To view the clock interaction of the design, expand **Implemented Design**, under the **Implementation** in the Flow Navigator, and select **Report Clock Interaction** command.

Step 2. In the **Report Clock Interaction** dialog box, type the name of the results in the **Results name** field and click **OK**

Step 3. The **Clock Interaction** report will display in the main Vivado IDE window, see Fig 1.24



⬛ No Path
🟦 User Ignored Paths
🟩 Timed
🟥 Timed (unsafe)
🟦 Partial False Path
🟧 Partial False Path (unsafe)
🟪 Max Delay Datapath Only

Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)
1	clk_p	clk_p	rise - rise	3.280	0.000	0	233	10.000	rise - rise	0.151	0.000	0	233

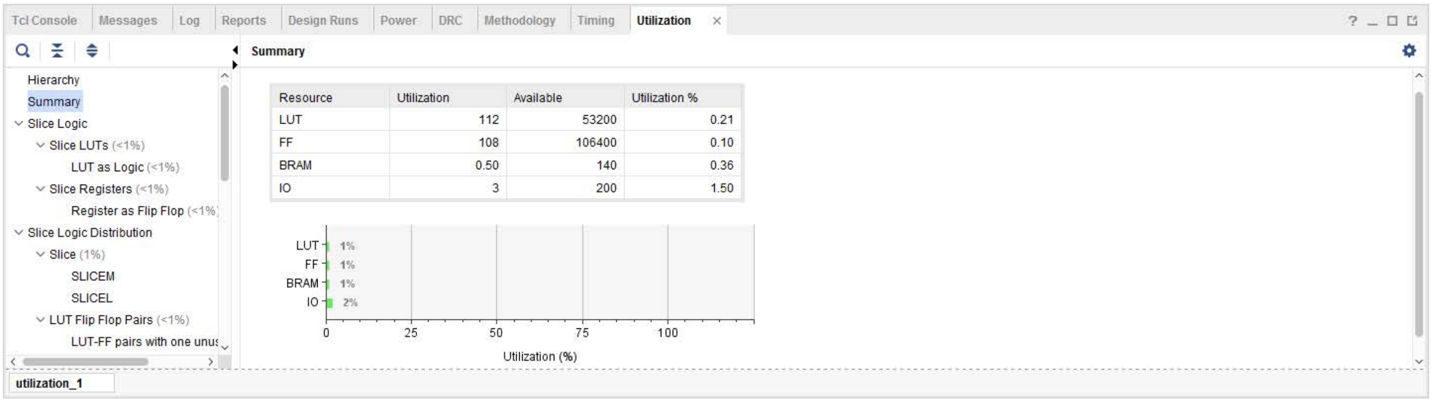
Fig 1.24: Report Clock Interaction tab

This report is helpful to tell us if timing is asynchronous (in case that we didn't include synchronization circuitry) and if paths are constrained (in case that we didn't add timing constraints to cover paths between unrelated clock domains). Green squares confirm that paths between the two clock domains are constrained.

Step 4. To view the resource utilization of the design, expand **Implemented Design**, under the **Implementation** in the Flow Navigator, and select **Report Utilization** command

Step 5. In the **Report Utilization** dialog box, type the name of the results in the **Results name** field and click **OK**

Step 6. The **Utilization** report will display at the bottom of the Vivado IDE, see Fig 1.25.



Resource	Utilization	Available	Utilization %
LUT	112	53200	0.21
FF	108	106400	0.10
BRAM	0.50	140	0.36
IO	3	200	1.50

Fig 1.25: Utilization Report tab

Note: You can maximize the utilization report and explore the results.

Implementation Reports

While the Flow Navigator points to the most important reports, the **Reports** tab contains several other useful reports.

Vivado Implementation Log - describes the implementation process and any issues it encountered

IO Report - Lists every signal, its attributes and its final location. It is always important to double-click pin assignments before implementing, because the tools can move any pin that is unassigned.

Utilization Report - describes the amount of FPGA resources used in a text format.

Control Sets Report - describes the number of unique control sets in the design Ideally this number will be as small as possible. Number of control sets describes how control signals were grouped. Control signals include clocks, clock enables, set, and reset signals. How the tools group them into slices and CLBs will dictate the density of the design in the FPGA.

DRC Report - Lists the DRC routing checks that were completed.

Power Report - describes the operating conditions and the estimated power consumption of your device.

Route Status Report - reports lists any nets that could not be routed

Timing Summary Report - identifies the default timing for the finished design (with true timing information)

The benefit of automatically generating these reports is that it encourages designers to read more about their design.