

## 1 Introduction

In this lab, you will understand the complete FPGA design cycle --- RTL to bitstream generation using FIR and IIR design. In addition, you will learn the transformations that happen to the design at each phase of the design cycle. You will also study the impact of timing and placement constraints in this lab.

### For this lab, you are expected to complete the following tasks:

- Run the example adder design with/without pin/timing/placement constraints and understand the impact of the constraints.
- Run the behavioral simulation of the example FIR design, understand the basic operations of the fix-point numbers.
- Implement the IIR design and run the behavioral simulation.

### 1.1 Introduction to Filters

The Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters are discrete-time filters and are the two classes of digital filters. Filters are mainly used to alter the frequency component of the time signals by either reducing or amplifying the magnitude of certain frequencies.

The output of FIR filters is a weighted sum of past inputs, whereas the output of IIR filters is based on both past inputs and previous outputs. The IIR is recursive in nature, as its filters use part of the outputs as inputs. Consequently, IIR is not always as computationally stable as FIR. From the computational perspective, FIR filters require more coefficients to execute similar filtering operations than the IIR filters, thus slower in computations than the IIR filters.

#### 1.1.1 FIR Filter

The order of a digital filter is the number of previous inputs used to calculate the current output. For a FIR filter with  $m$  orders, the general equation is defined as follows. Figure 1 demonstrates a simple FIR filter with  $m = 2$ .

$$y[n] = \sum_{k=0}^m b_k x[n - k]$$

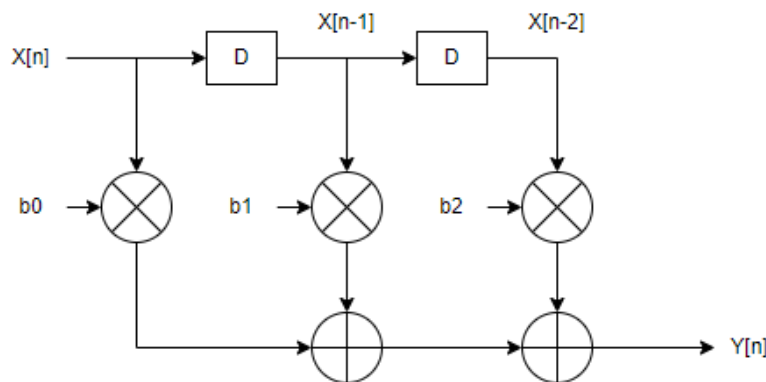


Figure 1: FIR Filter

### 1.1.2 IIR Filter

For an IIR filter with  $p$  and  $q$  orders, the general equation is defined below. Figure 2 shows a simple IIR filter with  $p = 1$  and  $q = 2$ .

$$y[n] = \sum_{k=0}^p b_k x[n-k] + \sum_{k=1}^q a_k y[n-k]$$

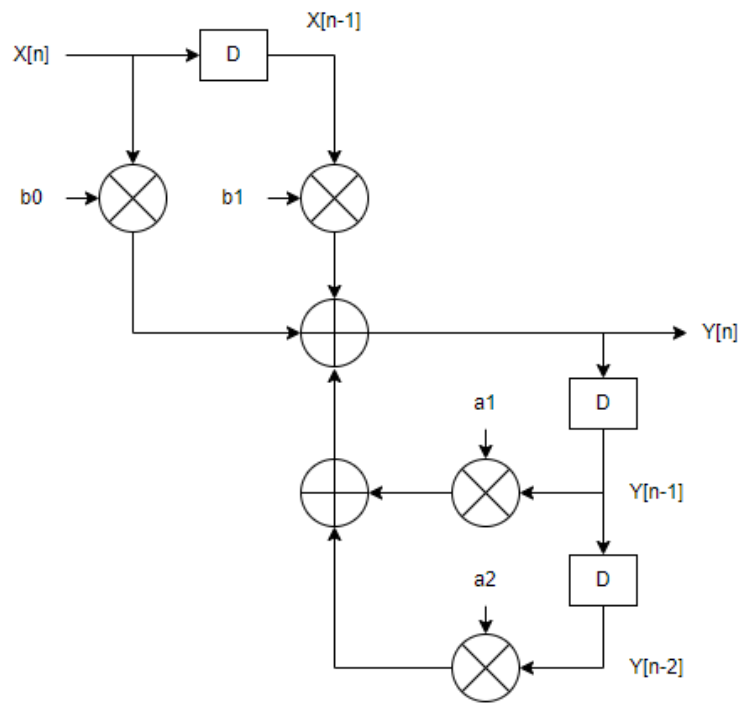


Figure 2: IIR Filter

## 1.2 Introduction to FPGA Resources

The FPGA resources are listed as follows:

1. Configurable logic blocks (CLBs). Any RTL design given by the user gets converted into these CLBs. Each CLB consists of two or four slices depending on the chosen FPGA family. These slices, in turn, consist of the following primitives:
  - a. Look-up-tables (LUTs)
  - b. Flip-flops: They can be configured as flip-flops or latches based on the RTL code given by the user
  - c. Multiplexers (MUXes)
  - d. Carry chain
2. Hard macros, such as Phase-locked loops (PLL), Transceivers, Peripheral component interconnect express (PCIe), Block RAMs (BRAMs), and Digital signal processing blocks (DSP). Unlike CLBs, which are configured based on the user's logic, these macros have fixed functionality.

## 2 FPGA Design Flow

In this section, we will be walking over the design cycle of Xilinx FPGAs using its development tool – Vivado IDE. Apart from Xilinx, there are other vendors such as Intel and Lattice. We will be focusing on the Xilinx FPGAs and their design flow. Hence, the terminology of the FPGA resources and the commands aiding the design flow corresponds to Xilinx.

### 2.1 Steps from RTL to bitstreams

#### 1. Design to RTL

The user transforms the microarchitecture to the hardware description language (HDL) such as Verilog, VHDL, and System Verilog. This microarchitecture can be converted to behavioral or structural RTL. However, predominantly the designers prefer behavioral RTL.

#### 2. Synthesis -- RTL to synthesized netlist

The resources in the FPGA vary from device to device. Hence, the Vivado tool translates the RTL design into a synthesized netlist based on the chosen device type. To understand this transformation, the user can save the synthesized netlist in a Verilog format. The RTL is converted to LUT-level and flip-flop level design, and hence, the Verilog netlist has instantiations of FPGA primitives such as LUTs and flip-flops.

#### 3. Place and route -- Synthesized netlist placed and routed on the FPGA

Once the synthesized design is placed and routed (implemented) by the Vivado tool, location constraints are given to each primitive in the netlist. Hence, the device view shows where each primitive in the synthesized netlist is placed on the FPGA. It also shows the routing path taken between any two primitives.

#### 4. Timing analysis

After step 3, the tool has the details of the propagation delay of every primitive and net in the implemented design. The timing analysis requires this information to determine if any routing path fails static timing analysis. The static timing analysis on each path should have a positive slack for the design to work successfully on the FPGA. If any routing path fails this analysis, the design will not function as intended onboard.

#### 5. Bitstream generation

Vivado generates the bitstream from the implemented netlist created by step 4. This bitstream will be loaded on the FPGA

### 2.2 Common mistakes

1. **Clock signal generation using HDL design:** The user should not write an HDL code to generate the clock. The PLLs must be configured to provide the necessary clock to the design. They are instantiated from the IP core generator. The PLL's output (clock signal) is routed on the pre-laid clock tree. These routes are designed to have the most negligible skew.

## ECEN 489/689 – Lab 2: Understanding the FPGA Design Cycle

- 2. Clock source to the PLL:** The clock from onboard oscillators/ethernet PHY (in the case of Zybo board) must be connected to the clock pin of the FPGA. These pins have the pre-routed path from the input port to the PLL inputs. Please do not generate the input clock to the PLL using RTL coding (e.g., clock generated using always block).

As shown in Section 5 in the following webpage: <https://reference.digilentinc.com/programmable-logic/zybo-z7/reference-manual>, please connect the pin E17 to the clock input of the PLL. Please check Section 2.4 for more information on pin constraints.

### 2.3 Must-knows in FPGA design

#### 1. How to instantiating the PLL:

- Step 1:** Click on the IP Catalog and then search for "Clocking Wizard," as shown below

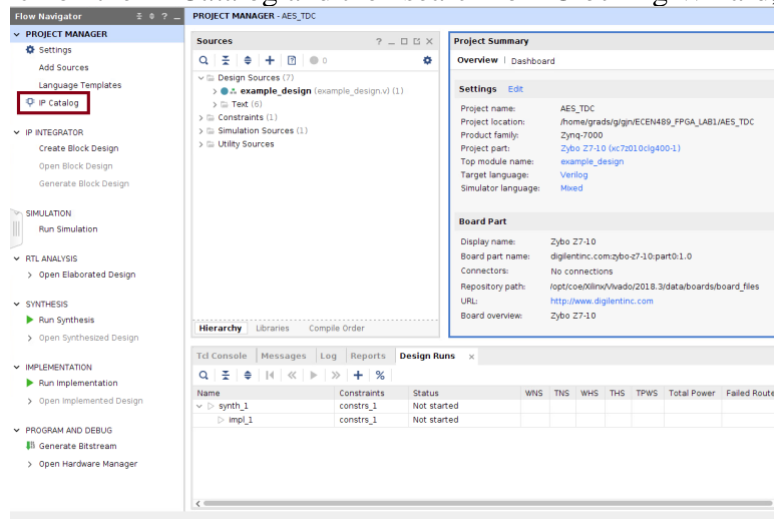


Figure 3: IP Catalog boxed in red

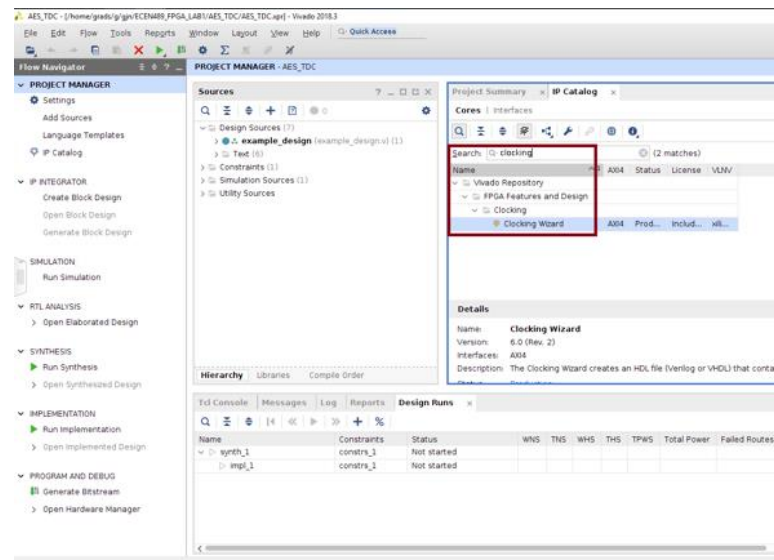


Figure 4: Clocking Wizard in IP Catalog boxed in red

- Step 2:** Set the input clock frequency. As shown below, choose "PLL" and update the primary clock with 125MHz as this is the fixed input clock to the FPGA. As the clock input is single-

ended, ensure that the "Source" type is chosen as "Single ended clock capable pin."

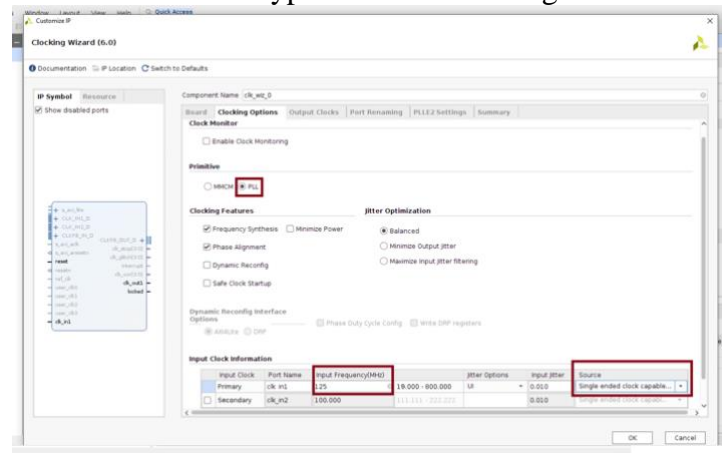


Figure 5: PLL input clock configurations

- **Step 3:** Set the output frequency. The output frequency is set to 200MHz, as shown in the below screen capture. Also, ensure that the input – reset and output – locked signal are enabled, as illustrated below. Then click on “OK” to finish the setting.

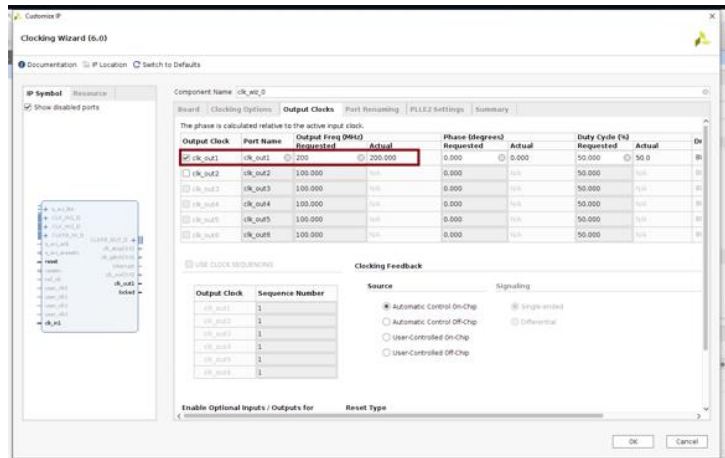


Figure 6: PLL output clock configurations (1)

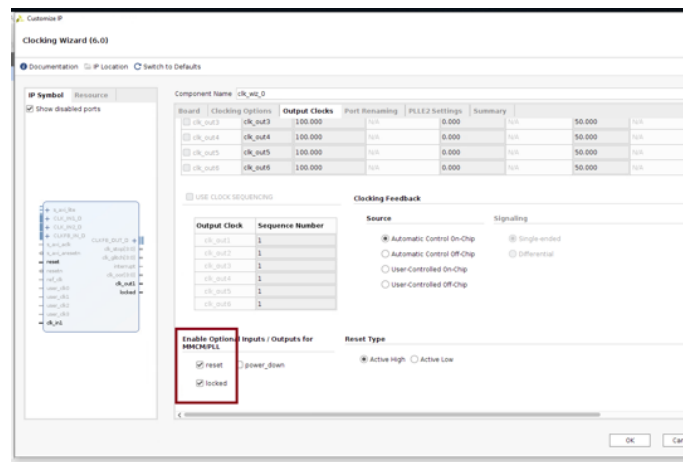


Figure 7: PLL output clock configurations (2)

### 2. Pin constraints:

- The RTL design's top-level input/output ports must be connected to the IN/OUT pins of the FPGA. The following command achieves this.

```
set_property PACKAGE_PIN <PIN_NUM> [get_ports <PIN_NM>]
```

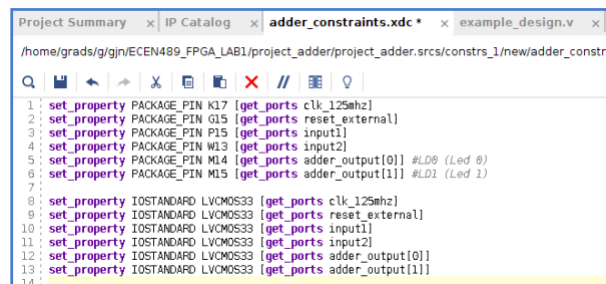
Here, the port with the port name <PIN\_NM> is connected to the physical FPGA pin <PIN\_NUM>.

- The in/out pins of the FPGA are connected to other chips/devices onboard. Therefore, the voltage level of the FPGA pins must be compatible with the voltage levels of the chips to which it is connected. The following command achieves this.

```
set_property IOSTANDARD <IO_STD_VARIANTS> [get_ports <PIN_NAME>]
```

Here, the port with the port name <PIN\_NM>, is set to the voltage level defined by the IO standard <IO\_STD\_VARIANTS>.

If there are no connections between certain FPGA pins and external chips onboard, it is safe to ignore these constraints for those FPGA pins. The pin constraints for the adder circuit are added to the `adder_constraints.xdc` file. This file should be added to the project using the "Add constraints file" option. The constraints for the adder circuit are given below:



```
1 set_property PACKAGE_PIN K17 [get_ports clk_125mhz]
2 set_property PACKAGE_PIN G15 [get_ports reset_external]
3 set_property PACKAGE_PIN P15 [get_ports input1]
4 set_property PACKAGE_PIN W13 [get_ports input2]
5 set_property PACKAGE_PIN M14 [get_ports adder_output[0]] #LD0 (Led 0)
6 set_property PACKAGE_PIN M15 [get_ports adder_output[1]] #LD1 (Led 1)
7
8 set_property IOSTANDARD LVCMOS33 [get_ports clk_125mhz]
9 set_property IOSTANDARD LVCMOS33 [get_ports reset_external]
10 set_property IOSTANDARD LVCMOS33 [get_ports input1]
11 set_property IOSTANDARD LVCMOS33 [get_ports input2]
12 set_property IOSTANDARD LVCMOS33 [get_ports adder_output[0]]
13 set_property IOSTANDARD LVCMOS33 [get_ports adder_output[1]]
14
```

Figure 8: User constraints file (`adder_constraints.xdc`)

### 3. Timing constraints:

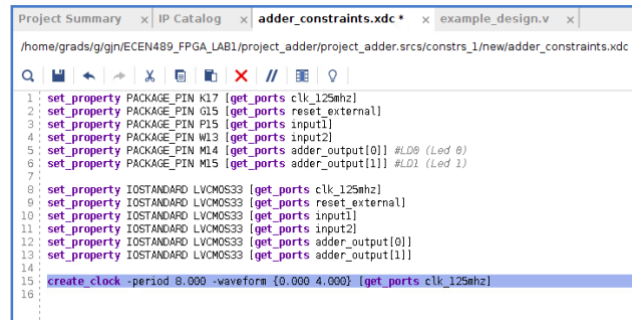
- The following command informs the timing analysis tool in Vivado that the clock <CLK\_NM> has a period of <TIME\_PERIOD> ns. The clock will be "high" starting <HIGH>ns and "low" starting <LOW>ns. The <HIGH> and <LOW>, in turn, controls the duty cycle of the clock.

```
create_clock -period <TIME_PERIOD> -waveform {<HIGH> <LOW>} [get_nets <CLK_NM>]
```

It is mandatory to include the "create\_clock" command for each clock that drives the PLL input. However, it is not necessary to include this command for the PLL's output clocks. This is because Vivado

will automatically update the timing constraints of the output clocks based on the timing constraints of the input clocks.

The timing constraint required for the "adder" project is given below (highlighted in blue):



```
Project Summary x IP Catalog x adder_constraints.xdc x example_design.v x
/home/grads/g/gjn/ECEN489_FPGA_LAB1/project_adder/project_adder.srsrcs/constrs_1/new/adder_constraints.xdc

1 set_property PACKAGE_PIN K17 [get_ports clk_125mhz]
2 set_property PACKAGE_PIN G15 [get_ports reset_external]
3 set_property PACKAGE_PIN P15 [get_ports input1]
4 set_property PACKAGE_PIN M13 [get_ports input2]
5 set_property PACKAGE_PIN M14 [get_ports adder_output[0]] #LD0 (Led 0)
6 set_property PACKAGE_PIN M15 [get_ports adder_output[1]] #LD1 (Led 1)
7
8 set_property IOSTANDARD LVCMOS33 [get_ports clk_125mhz]
9 set_property IOSTANDARD LVCMOS33 [get_ports reset_external]
10 set_property IOSTANDARD LVCMOS33 [get_ports input1]
11 set_property IOSTANDARD LVCMOS33 [get_ports input2]
12 set_property IOSTANDARD LVCMOS33 [get_ports adder_output[0]]
13 set_property IOSTANDARD LVCMOS33 [get_ports adder_output[1]]
14
15 create_clock -period 8.000 -waveform {0.000 4.000} [get_ports clk_125mhz]
16
```

Figure 9: Timing constraints added to the user constraints file (adder\_constraints.xdc)

#### 4. Placement constraints:

These constraints are not mandatory for every design. The placement and routing algorithm in the Vivado automatically looks for the optimal placement/wire length of the routing resource of the user design. However, the user can override Vivado's placement locations with his/her own locations. The manual placements can be done in module-level, LUT-level, and flip-flop-level. The following is a set of commands required for module-level (instance-level) placement.

- a) `create_pblock <partition_nm>`

This command creates a partition block named `<partition_nm>`.

- b) `resize_pblock [get_pblocks <partition_nm>] -add {SLICE_X<XC1>Y <YC1>:SLICE_X<XC2>Y <YC2>}`

This associates the partition block `<partition_nm>` with the physical region on the FPGA denoted by `SLICE_X<XC1>Y <YC1>:SLICE_X<XC2>Y <YC2>`. Here `XC1`, `YC1`, `XC2`, `YC2` are physical location coordinates on the FPGA.

- c) `add_cells_to_pblock [get_pblocks <partition_nm>] [get_cells -quiet [list <instance_nm>]]`

This command assigns the instance `<instance_nm>` to the partition block `<partition_nm>`.

### 2.4 Bitstream generation with and without user constraints. The example is based on a simple adder circuit.

1. **RTL example:** Please refer to `example_design.v` file in the example directory. The source file has the adder circuit and the PLL to generate the clock required by the adder circuit.



2. **Device View of placed and routed design:** After the bitstream is generated, click on "Open Implemented Design" from the dialog box (or) under the Implementation tab on the left pane of the Vivado software, as shown below.

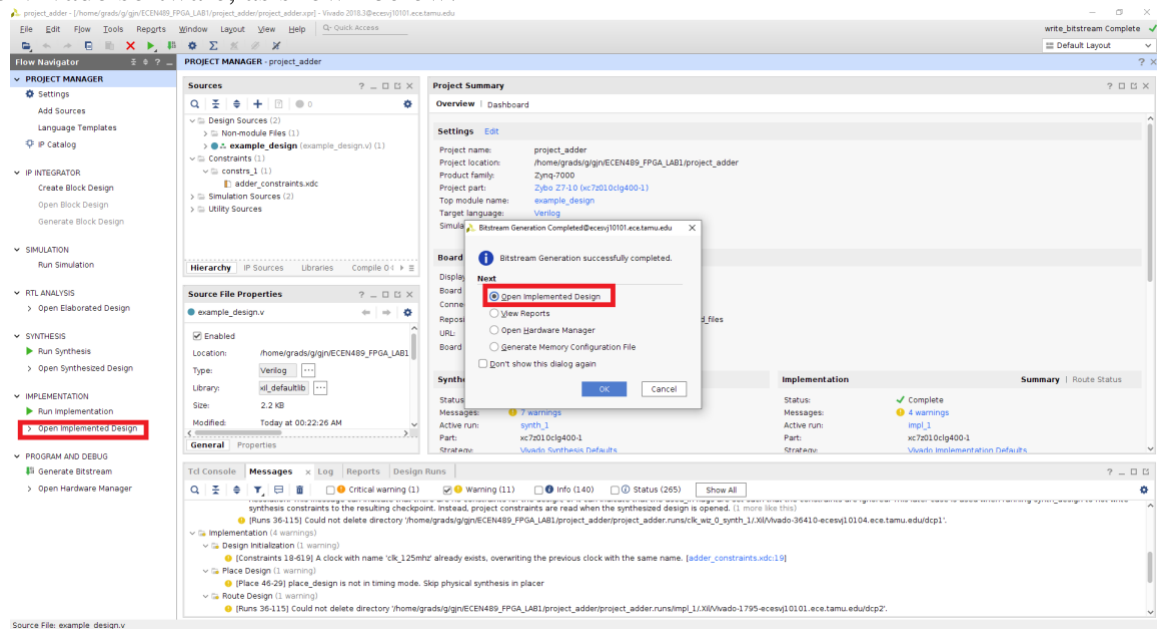


Figure 10 Dialog box after a successful bitstream generation.

3. This opens up the "Device View" window. This Device view shows how the RTL is translated into flip flops and LUTs. Please refer to the following figures to understand the impact of user constraints.

**Without pin/timing/placement constraints:** The adder circuit is implemented in the X1Y0 region of the FPGA by the Vivado software (Please check the region name in the left corner of the highlighted red box).

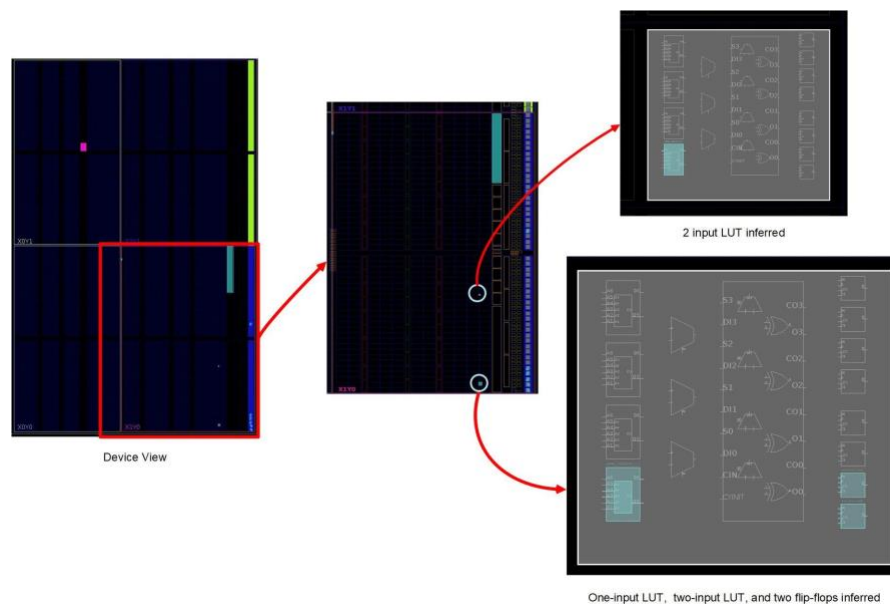


Figure 11 Device view with no user constraints



## With pin constraints and without timing/placement constraints:

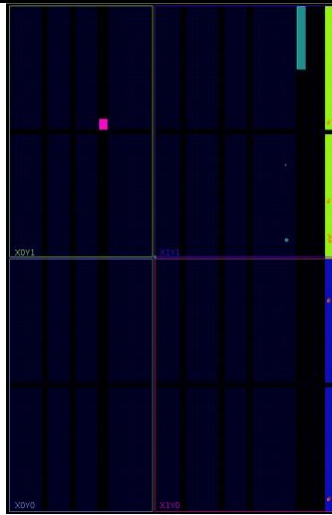


Figure 12 Device view with pin constraints (The location of the adder is moved from X1Y0 to X1Y1)

## With pin/timing constraints and without placement constraints:

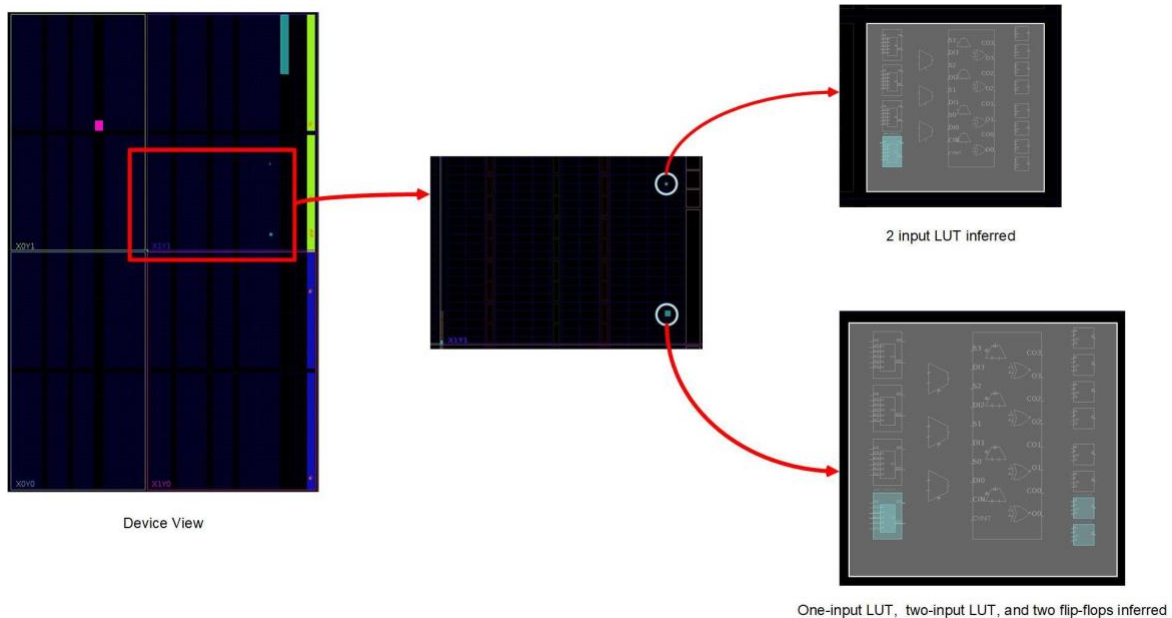


Figure 13 Device view with pin and timing constraints

## With pin/timing/placement constraints:

Here all the LUTs and flip-flops are placed in the pblock created by the user. The adder is moved to X0Y0 region of the FPGA.

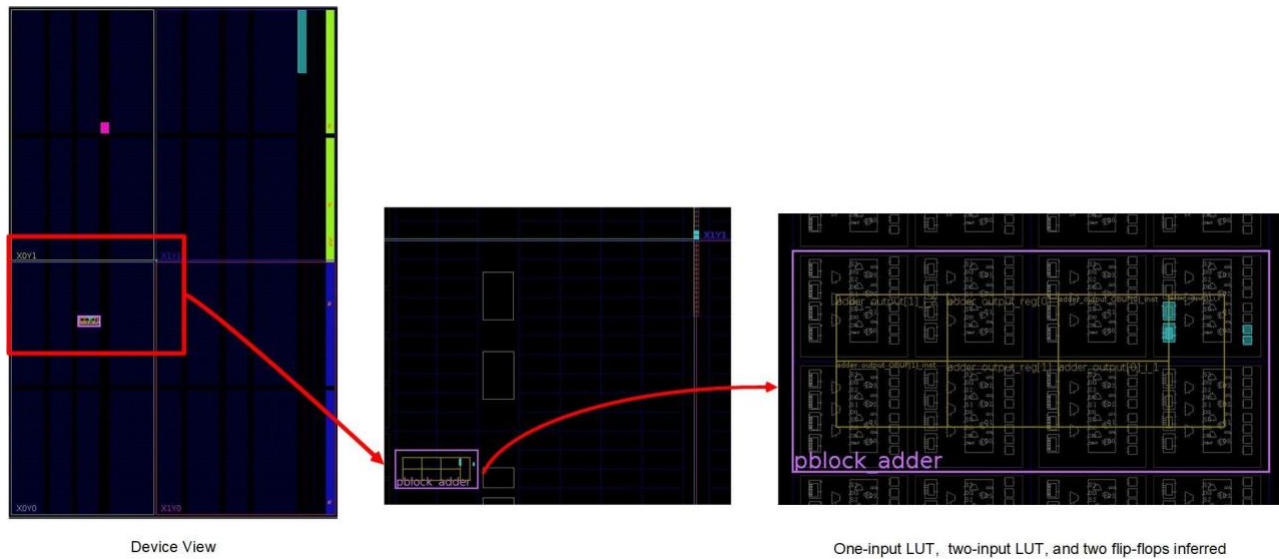


Figure 14 Device view with pin, timing, and placement constraints

### 3 Resources required:

#### 1. Understanding the FPGA internals:

- Class Notes.
- <https://courses.cs.washington.edu/courses/cse467/05wi/pdfs/lectures/18-FPGA-DesignTips.pdf>

#### 2. Design constraints (Pin, timing, and placement):

- [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2019\\_1/ug945-vivado-using-constraints-tutorial.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug945-vivado-using-constraints-tutorial.pdf) -- This link shares a detailed understanding of all available constraints that can be used on Xilinx FPGAs.

**3. Accessing Vivado:** Please refer to the lab 0 manual for detailed instructions on accessing Vivado.

**4. Base Verilog files:** The students will be provided with project files “**lab2.xpr.zip**”. Unzip this file to form the base Vivado directory. This folder has the example Verilog source code at “\$project\_path/project\_adder.srscs/sources/imports/ECEN489\_FPGA\_LAB2/example\_design.v.”

**5. FIR/IIR files:** The source code and testbench code are at “\$project\_path/lab2\_code/\*.v”. Beside, there is a python script “ref\_FIRIIR.py” for the results verification.

**6. Fixed point numbers in Verilog:** <https://projectf.io/posts/fixed-point-numbers-in-verilog/>

### 4 Task for students

Please download “Lab2.xpr.zip” from Piazza and extract the file before you proceed. In the extracted folder, you will find the Vivado project file “**project\_adder/project\_adder.xpr**”, and the source code in “project\_adder/lab2\_code/”.

#### 4.1 Understanding the constraints in the FPGA design.

- Open the project in Vivado by double-click on “project\_adder.xpr”.

- b) Follow the steps in section 2.4, understand the impact of the constraints.
- c) Open the constraint file “adder\_constraints.xdc”, modify the pin/timing/placement constraints by comment/uncomment the corresponding lines.
- d) Once the constraint file is ready, run the implementation by click on “Run Implementation” on the Flow Navigator Panel on the left.
- e) Open the implemented design when the implementation is complete. Check the positions of the primitives to understand the constrains. (No screenshots needed)
- f) Implement the design of IIR filter, run the behavioral simulation, share the screenshot of the simulation, and justify the correctness of your outputs.

### 4.2 Behavioral Simulation of FIR Design

- a) Close the implemented design in the previous section by click on the ‘X’ as shown in Figure 15.

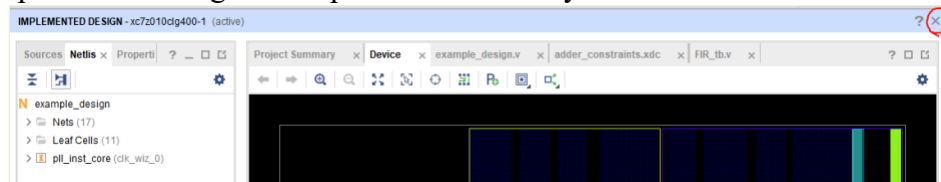


Figure 15 Closing the implemented design

- b) In the Design Sources Panel, open “FIR\_tb.v”, “FIR.v”, and “multiply.v”. Read through these files and understand how the FIR design and the testbench are implemented.
- c) Once understanding the design, we can do the behavioral simulation. Right-click on “SIMULATION” and select “Simulation Settings” as shown in Figure 16.
- d) In the Simulation Settings window, please change the simulation top module to “FIR\_tb” as shown in Figure 17.
- e) Click on the “Simulation” tab to modify the simulation runtime as shown in Figure 18. In this lab, 50ns is enough. Click on “OK” to confirm this setting.

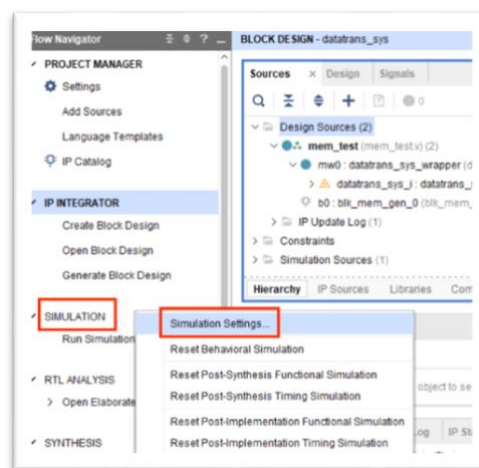


Figure 16 Opening the simulation settings

## ECEN 489/689 – Lab 2: Understanding the FPGA Design Cycle

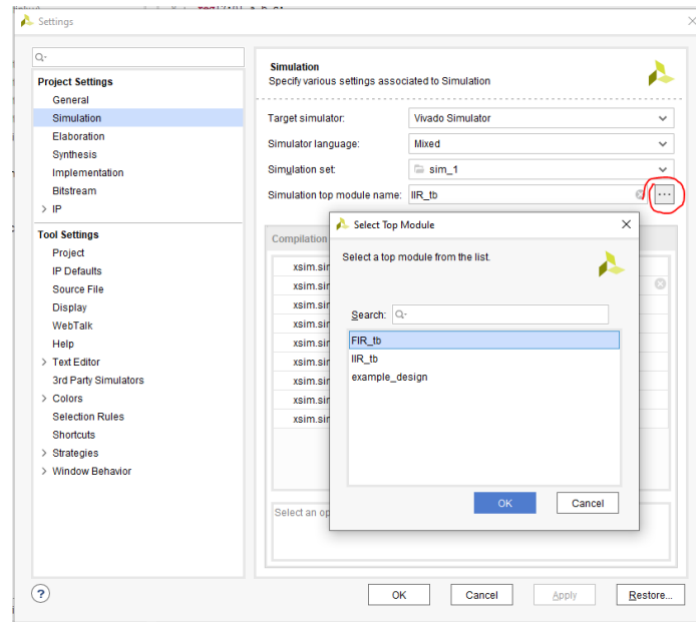


Figure 17 Selecting Top Module

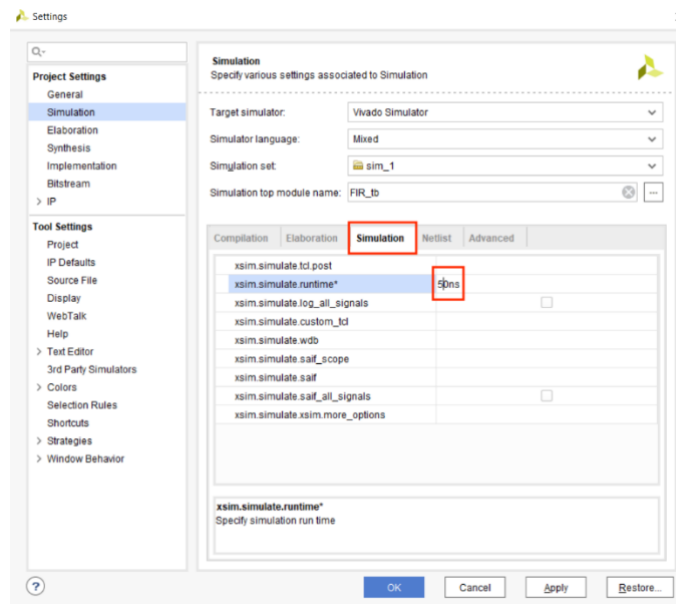


Figure 18 Setting simulation runtime

g) In the Flow Navigator Panel, click on “**Run Simulation->Run Behavioral simulation**” as shown in Figure 19.

h) Now you will see the waveforms of the behavioral simulation. Change the radix of the waveform to signed fixed-point number with four fractional bits. **Please take a screenshot of the waveform to the lab report.**

i) Run the python script “ref\_FIRIIR.py” to see if the simulation result matches with the python result.

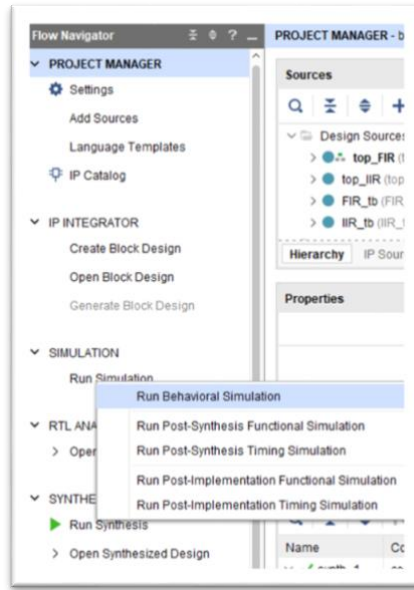


Figure 19 Running behavioral simulation

## 4.3 Implementation and simulation of the IIR design

- Open the source file in “IIR\_tb.v” and “IIR.v” and implement your design on the IIR filter.
- Once you finish the design, follow the instructions in section 4.2 to run the behavioral simulation of the IIR design. **Please take a screenshot of the waveform to the lab report.**
- Run the python script again and compare your simulation with the reference outputs.

## 5 Pre-lab submission

- Please only submit 1 PDF file, containing the following items:
  - Screenshots of your behavior simulation of FIR design.
  - A few words explaining the results in the screenshots.
- Please name the PDF file as “Lab#\_PreLab\_Section#\_LastName\_FirstName.pdf”.
- Please email the PDF file to the TA with the subject as “ECEN 489 Lab Report” or “ECEN 689 Lab Report”.
- Please submit before February 07 (Monday) 11:59 pm.

## 6 Post-lab Submission

- Please only submit 1 zip file, containing all the code files you used in this lab, and 1 PDF report. Name it as “Lab#\_PostLab\_Section#\_LastName\_FirstName.zip”.
- In the PDF report, please include the following items:
  - Screenshots of the behavioral simulations of IIR design.
  - A few words explaining the results in the screenshots.
- Please email the zip file to the TA with the subject as “ECEN 489 Lab Report” or “ECEN 689 Lab Report”.
- Please submit before February 11 (Friday) 11:59 pm.

