Laboratory 1

Lab Objective:

The aims of this laboratory session are:

(1) To familiarize with Xilinx Virtex-5 Evaluation Platform with a series of simple exercises
(2) To describe a digital system using Verilog and implement it on a Xilinx FPGA, which is Virtex 5 in the present case, using Xilinx ISE tool

Learning Outcomes:

At the end of this lab session, you would have learned the following:

1. Describing a digital design in Verilog with associated project directories
2. Functional and timing simulation of your design using ISim
3. Creating a User Constraints File (UCF) for pin assignment and timing constraints
4. Synthesizing and generating FPGA bit-stream for Xilinx FPGA using Xilinx ISE 14.7
5. Programming the FPGA using Xilinx iMPACT tool
6. Overall idea of FPGA Electronic Design Automation Tool flow

Note:

You will be required to carry out the steps listed under "Learning Outcomes" in future lab sessions as well. The instructions provided in Lab 1 will not be repeated in the other labs with respect to these steps. So, you are expected to finish lab 1 diligently as it lays the foundation for other labs.

Familiarization with Xilinx Virtex 5 Evaluation Platform:

The Xilinx XUPV5-LX110T Evaluation Platform features a Xilinx Virtex-5 FPGA. This evaluation platform is meant to help users evaluate their designs area and timing performance on a Virtex-5 FPGA. The platform features V5-LX110T device from the Virtex-5 family of FPGAs.

The evaluation platform also has other devices like DRAM, USB ports, Ethernet ports, compact flash card reader etc. You can know more about the additional devices on the board by reading Xilinx UG347 (ML505/506/507 User Guide; v3.1.2; May 16, 2011). Xilinx UG 347 does not feature V5-LX110T device but another Virtex-5 device. However, the board layout and design for XUPV5-LX110T Evaluation Platform is exactly the same as for ML505/506/507. Hence, Xilinx has not published any other user guide specific to XUPV5-LX110T Evaluation Platform.

The basic block diagram of XUPV5-LX110T Evaluation Platform is shown in Fig.1. The red box shows General Purpose IO (GPIO) switches available on the board. These can be used to provide simple inputs to the FPGA and see simple outputs coming out of FPGA after processing the input. Complex inputs, requiring a number of bits for representation, can be provided to FPGA

through other interfaces like RS-232, VGA Input Codec, Ethernet PHY (all shown in green box) etc.

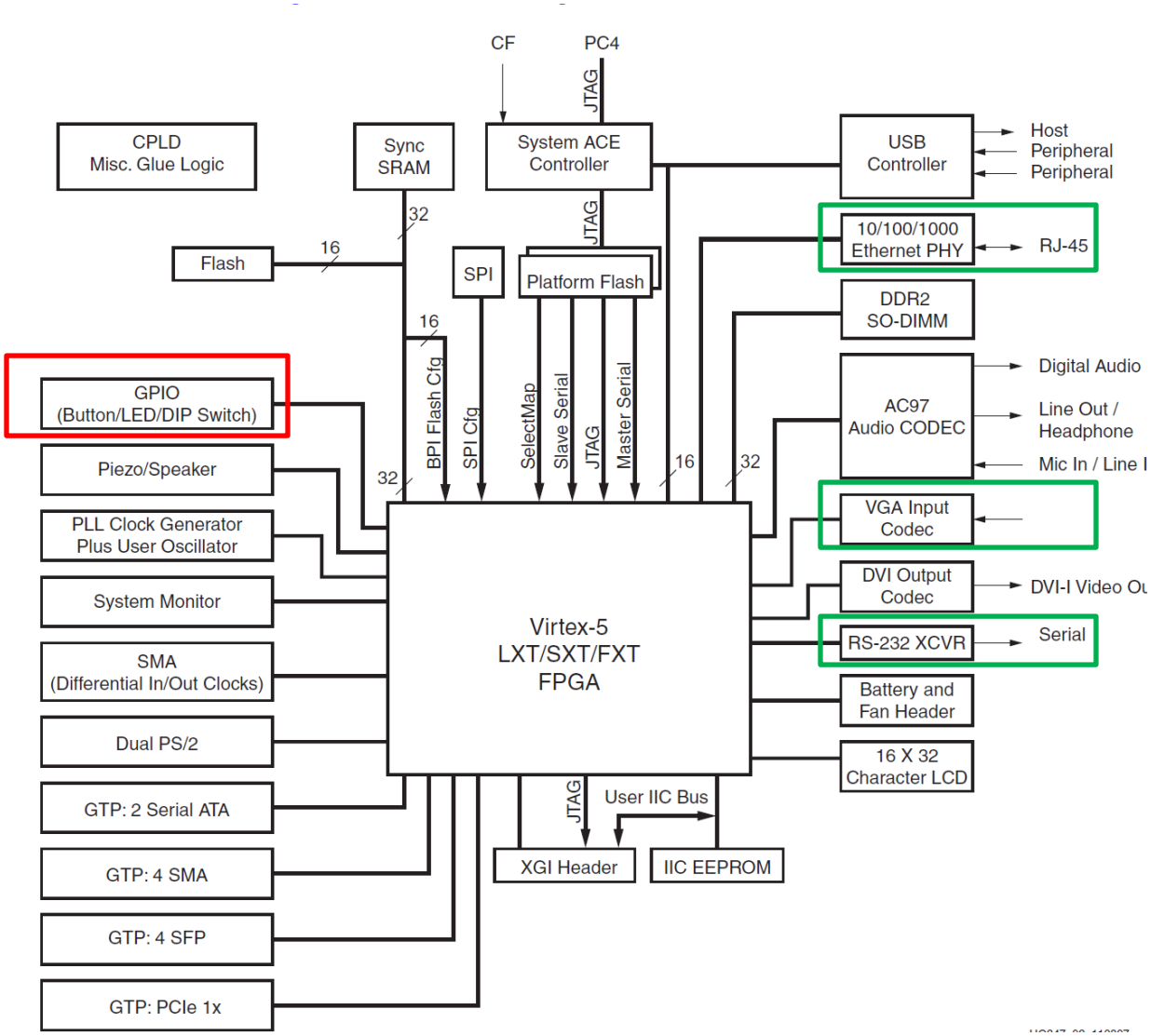


Fig.1: Block diagram of XUPV5-LX110T Evaluation Platform

Laboratory Exercises:

In the exercises in Lab1, you will control the 8 LEDs on the board using the 8 DIP switches. You can read more about DIP switch at the following link: http://en.wikipedia.org/wiki/DIP_switch

Notice the location of the 8 LEDs and the 8 DIP switches on Fig.2. Number 6 refers to the 8 DIP switches while number 7 refers to the 8 LEDs. The 8 DIP switches are connected to 8 user pins on the FPGA through traces on the PCB and the 8 LEDs are connected to another 8 user pins on the FPGA. It is these pins (each pin has a pin number) which will need to be specified in your

user constraints file (UCF). To learn more about other items on the board, refer to UG 347. To learn about the pin locations for DIP switches and LEDs, see Table 1-5 (page 20) and Table 1-6 (page 21) in UG 347. These tables are also present in Appendix I of this manual.

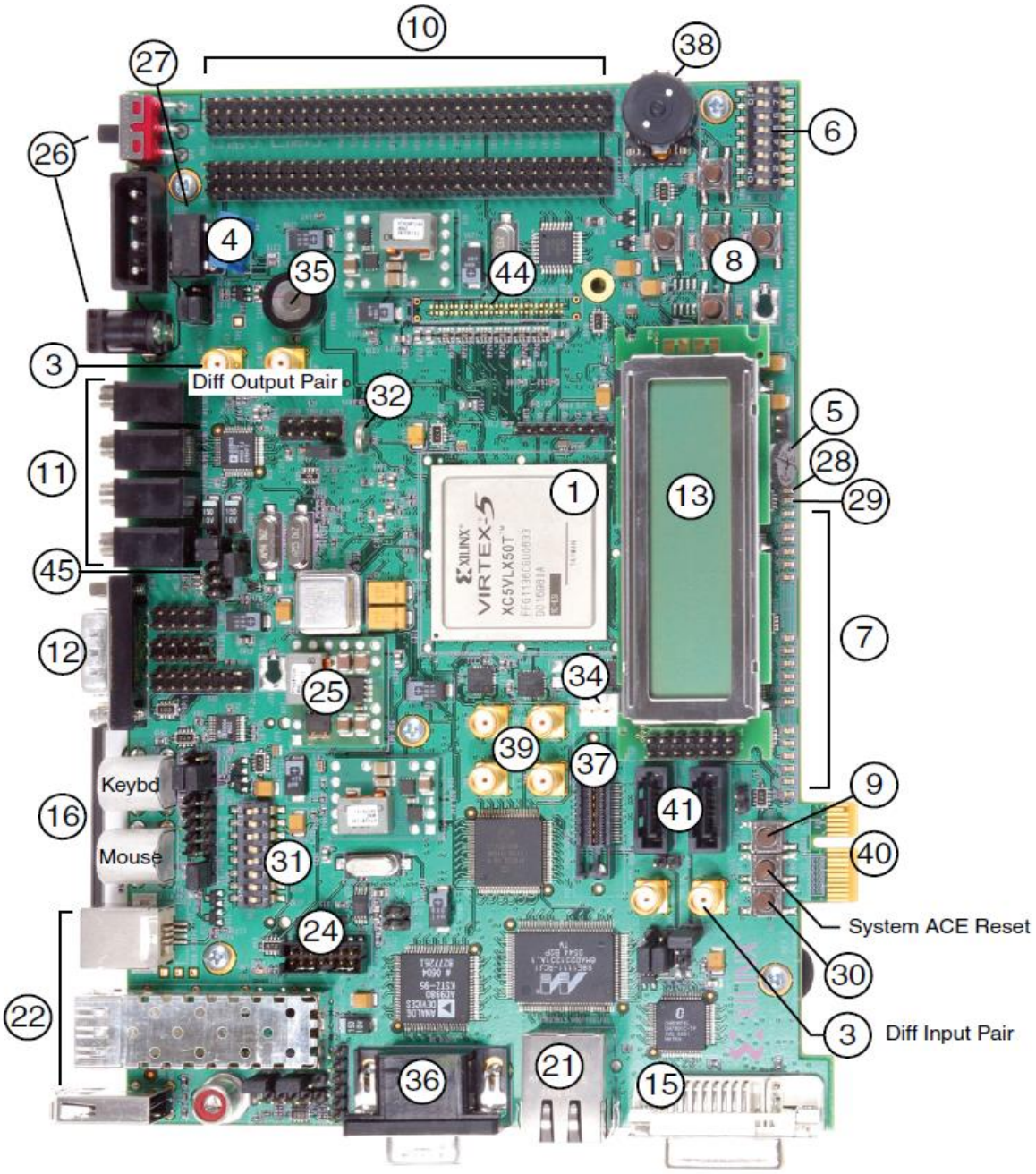


Fig.2: Front part of XUPV5-LX110T Evaluation Platform

Exercise 1A:

Controlling LEDs through DIP switches:

You are required to write a Verilog module that takes as input 8 bits and shows their value as output. The inputs will come through the DIP switch. Using the knowledge that an input bit value of 1 when connected to a LED will drive the LED high making it glow, you can write the Verilog module. The reference code (***control_LED.v***) is provided where you simply need to make the appropriate assignment to output.

Steps to be carried out:

1. Create a directory ***Lab1A***
2. Inside ***Lab1A***, create sub-directories ***src, sim*** and ***synth***. The ***src*** sub-directory will have the Verilog source file for your design while ***synth*** sub-directory will be used for synthesis using Xilinx ISE. The ***sim*** sub-directory will be used for simulation using ModelSim and will contain the necessary script and testbench files (This directory is not used in Lab1)
3. Copy the ***control_LED.v*** file to ***src*** directory. In future exercises, you will be required to create and save your source files in this directory. You can use any text editor to create your source file. Notepad++ (http://notepad-plus-plus.org/) is a good one as it allows language based code highlighting. Select ***Verilog*** from the ***Language*** menu if you use Notepad++.
4. Copy the ***control_LED.ucf*** file to ***synth*** directory. This file contains the constraints for pin location for the input and output. In future exercises, you will be required to create and save a similar UCF file in this directory.
5. Start Xilinx ISE Design Suite 14.7 by double clicking on ISE Design Suite icon on your desktop.
6. Go to **File** -> **New project** **(**A Create new project dialog box will pop up)
7. Under **Location**, browse to your drive:\Lab1\synth directory
8. Keep **Working directory** same as **Location**
9. Under **Name**, type control_LED
10. Select **HDL** in **Top-level source type**
11. Click **Next**
12. Select **Virtex 5** in **Family**
13. Select **XC5VLX110T** in **Device**
14. Select **FF1136** in **Package**
15. Select **-1** in **Speed Grade**
16. Select **Isim** in **Simulator**
17. Leave other options as they are
18. Click **Next** -> **Finish**
19. Go to **Project** -> **Add source**

20. Browse to drive:\Lab1\src and select ***control_LED.v*** and ***control_LED.ucf*** **(**You can also modify ***control_LED.v*** to make the proper assignment to output. UCF file contains the pin constraint information. It connects the input/output to the pin name. Please refer to the appendix table for the mapping between the switch name and pin name.)
21. You should see your design under **Implementation** view as shown in Fig. 3.

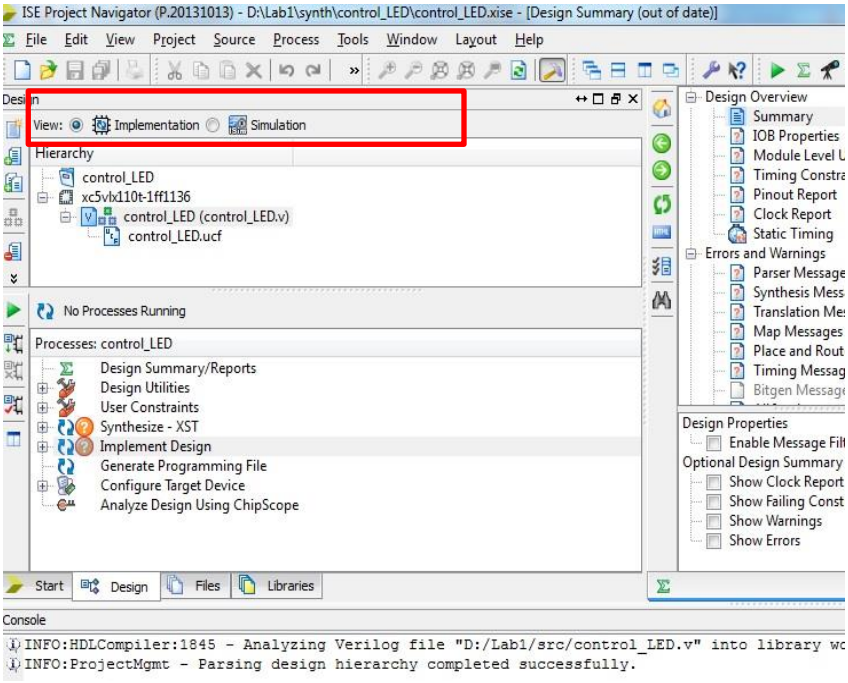


Fig.3. Implementation View

22. Change to **Simulation** view to see Fig. 4. Select **Behavioral** from drop down menu (red box)

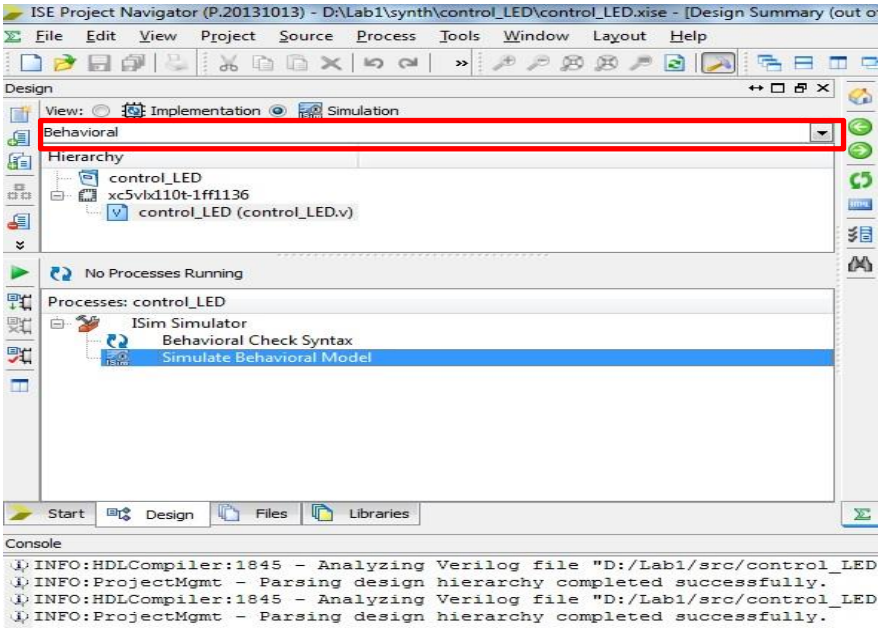


Fig.4. Simulation View

23. Right click on **Simulate Behavioral Model**, then click **Run Properties** in the popped-up menu; uncheck **Run for Specified Time** ; click **apply** and **ok**
24. Double click on **Simulate Behavioral Model;** a separate simulation window will pop up as shown in Fig. 5. You can see input and output in the waveform window (red box)

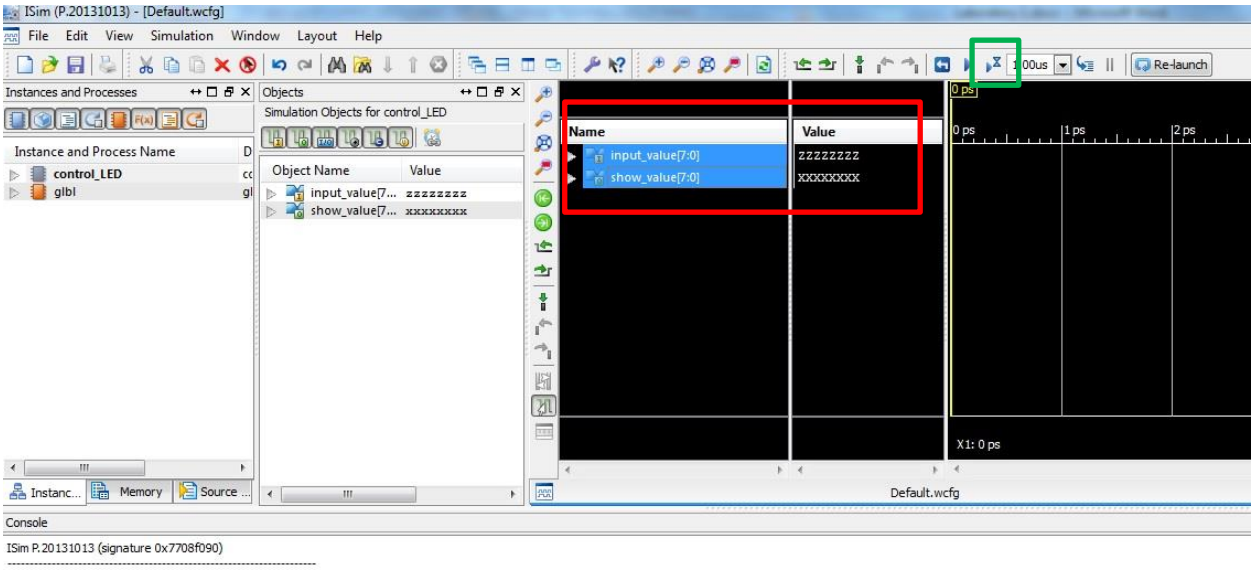


Fig.5: Simulation Window

25. Right click on **input_value** and select **Force Constant** and type 10000001 in **Force to Value**; click **Apply** and **Ok**
26. Click on the hour glass symbol (green box)

27. You should be able to see Fig. 6. You can see that the inputs and the outputs match. Play with the zoom buttons to zoom in or out the waveform. If inputs and outputs are same, your code is functionally correct. You just now performed "**Functional Simulation**" to test the functional correctness of your code. Try repeating steps 25 and 26 for other input values.

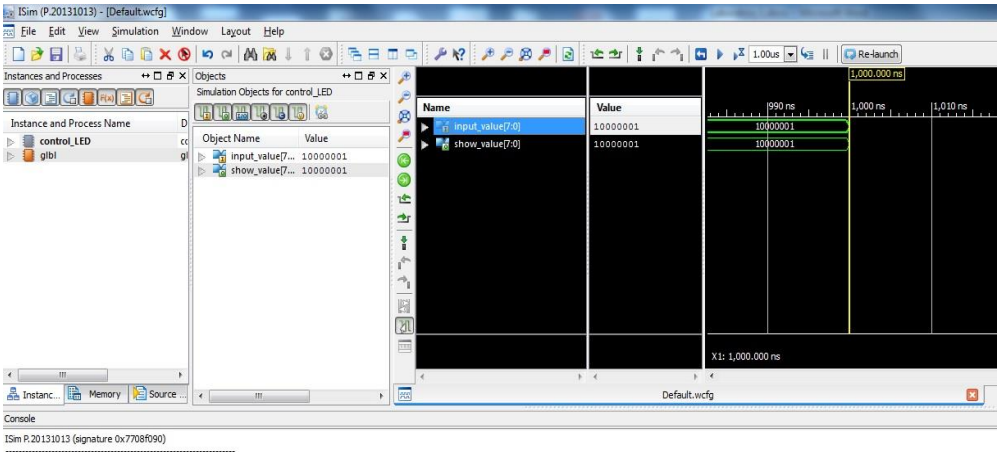


Fig.6: Functional Simulation

This method to create stimulus is convenient but we strongly unrecommend. We need to accomplish it in a more formal way. In the following several steps we will create a testbench file and use that to do all the simulations.

28. In **Implementation view**, right click on the window below and select **"add source file".** Then select "**Verilog Test Fixture"**. Name the file as **led_tb**. Then a testbench template will show on the screen as Fig.6-1. **Spend some time** to have an overview of the format of this file. In the succeeding labs, you need to complete the testbench yourself.

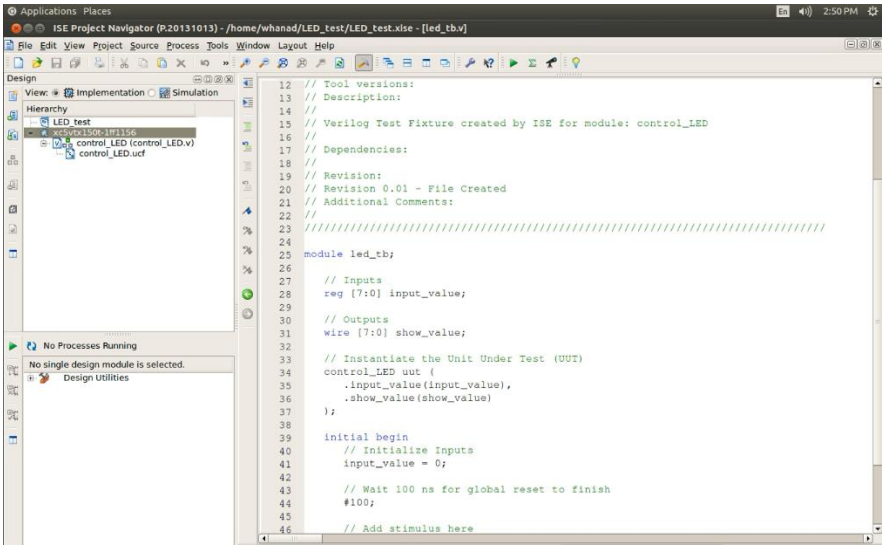


Fig.6-1: Testbench Template

29. Now you need to add stimulus, type **input_value=8'b1000_0001** under the comment "add stimulus here".
30. Go back to **Simulation** view and Select **led_tb** file. you should note that it has been set as the top module. Repeat the operation from 23 to 27 but this time you don't need to force constant value on the input. You will observe the same waveform as Fig.6 with a little difference-a 100ns delay on the initialization of signal of **input_value** and there is a **respond latency** between **input_value** and **show_value**, because in this simulation logic gate and wire latency are considered by the simulator.

31. Close the simulator window and exit the application.
32. Go back to **Implementation** view. Now, you will proceed to synthesizing your design now that you have verified your design for functional correctness using functional simulation. You will see Fig. 3 again.
33. Double click on **Synthesize.** You should see **a green tick mark** next to Synthesize after the process completes. If it is not green but **a yellow triangle with a note of exclamation**, click on W**arnings** tab at the very bottom of your screen to see if the tool identified some problem with your code.
34. Double click on **Implement.** Your design will now be implemented on the target FPGA. You should see something similar to Fig. 7 on the left of your screen. On the right side of your screen, you will see Fig. 8 which shows design related data.

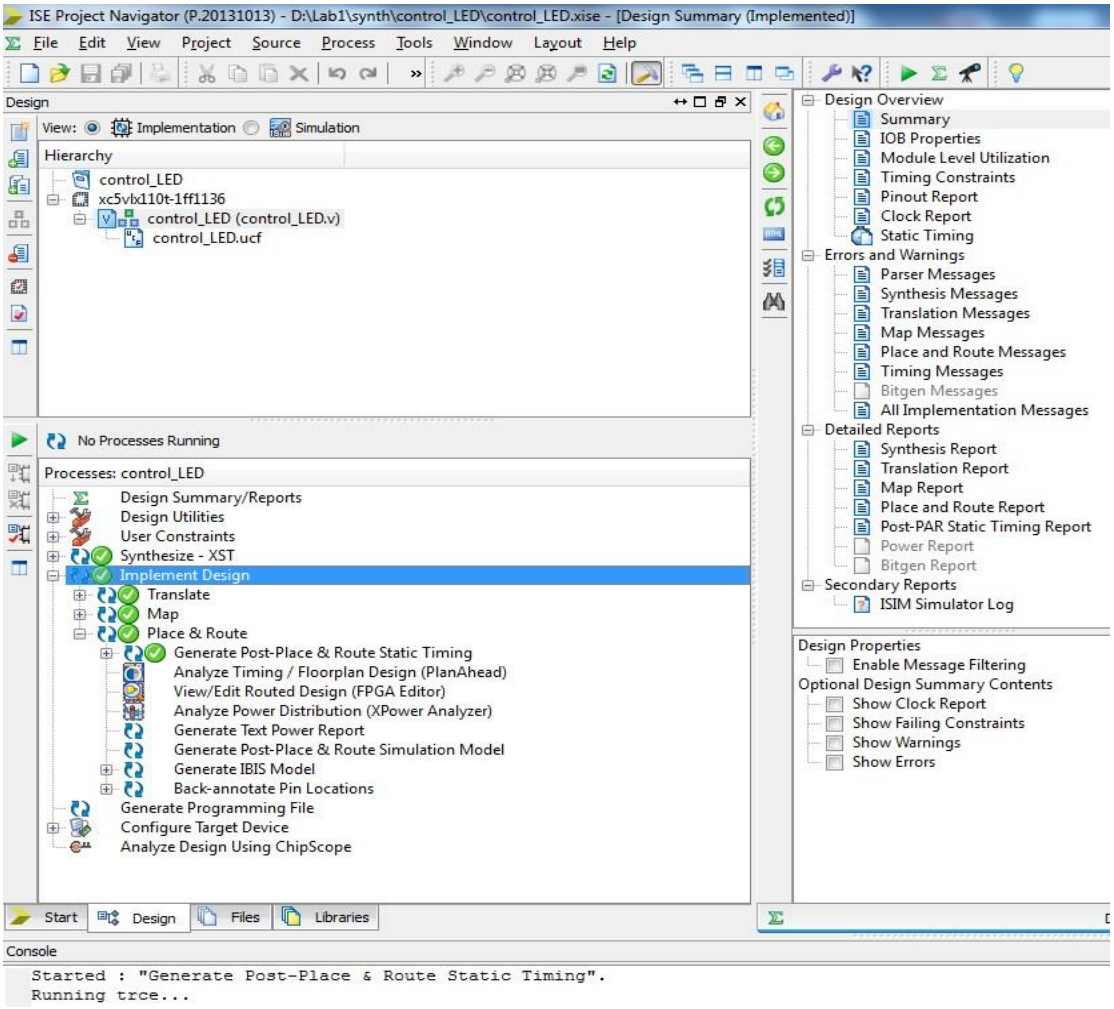
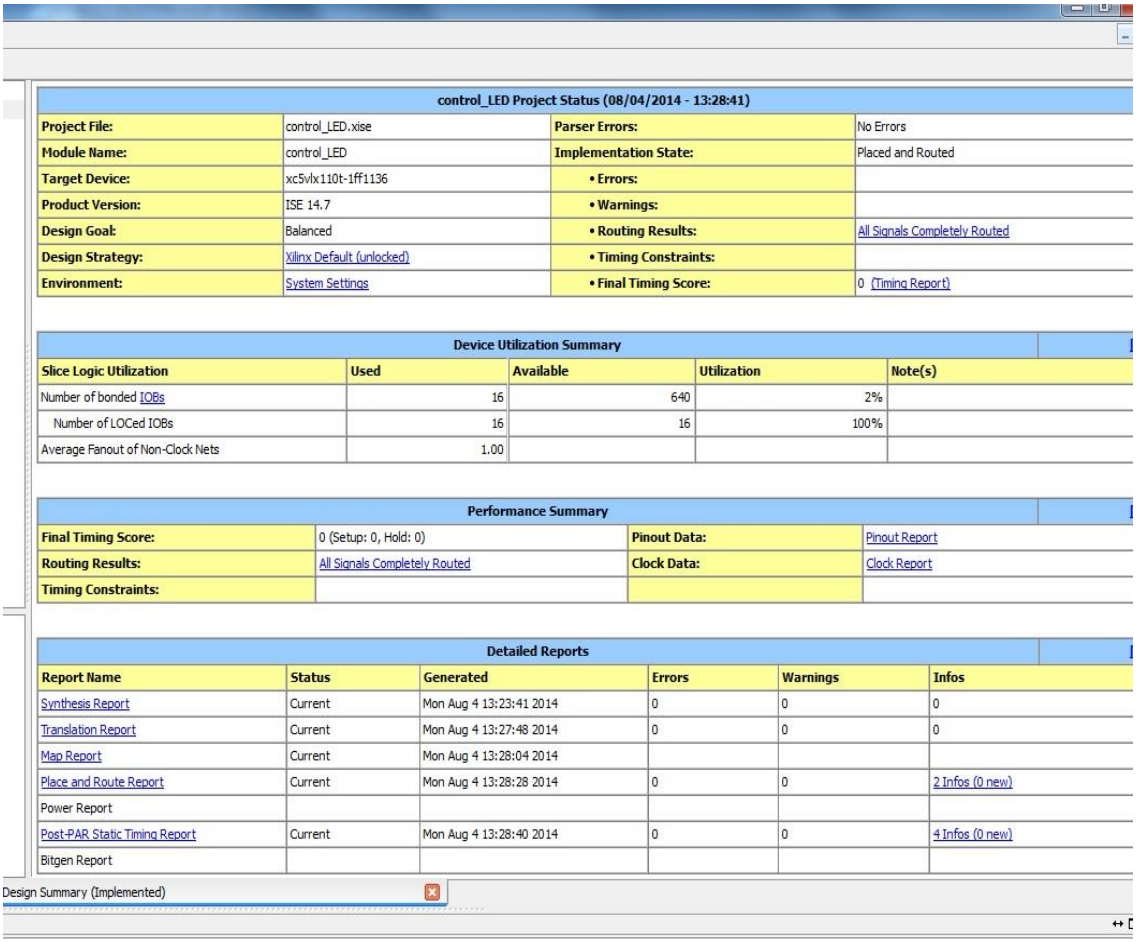


Fig.7: Implemented Design



control_LED Project Status (08/04/2014 - 13:28:41)

Project File:	control_LED.xise	Parser Errors:	No Errors
Module Name:	control_LED	Implementation State:	Placed and Routed
Target Device:	xc5vlx110t-1ff1136	• Errors:	
Product Version:	ISE 14.7	• Warnings:	
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of bonded IOBs	16	640	2%	
Number of LOCed IOBs	16	16	100%	
Average Fanout of Non-Clock Nets	1.00			

Performance Summary

Final Timing Score:	0 (Setup: 0, Hold: 0)	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report
Timing Constraints:			

Detailed Reports

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Mon Aug 4 13:23:41 2014	0	0	0
Translation Report	Current	Mon Aug 4 13:27:48 2014	0	0	0
Map Report	Current	Mon Aug 4 13:28:04 2014			
Place and Route Report	Current	Mon Aug 4 13:28:28 2014	0	0	2 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Mon Aug 4 13:28:40 2014	0	0	4 Infos (0 new)
Bitgen Report					

Design Summary (Implemented)

Fig.8: Design Data after implementation Design

35. Double click **on Generate Post-Place & Route Simulation Model** under **Place & Route**. (This will generate the placed and routed model of your design. This model will have information regarding actual delays on the device for interconnects and resources used by your design. This model is needed for **Post-Place and Route Timing Simulation** which verifies that your design actually works on the device under given timing constraints. Note that in the exercise 1A, there is no timing constraint, but you will need a timing constraint in later labs)
36. Go back to **Simulation** view (Fig.4) and select **Post route** from the drop down menu (red box in Fig.4). You should see Fig. 9.
37. Right click on **Simulate Post-Place and Route Model** and uncheck **Run for Specified Time** ; click A**pply** and **Ok**
38. Double click on **Post-Place and Route Model;** a new window will pop up as shown in Fig. 10. You can see a number of signals listed (red box). Except for the top two, the rest are internal signals.

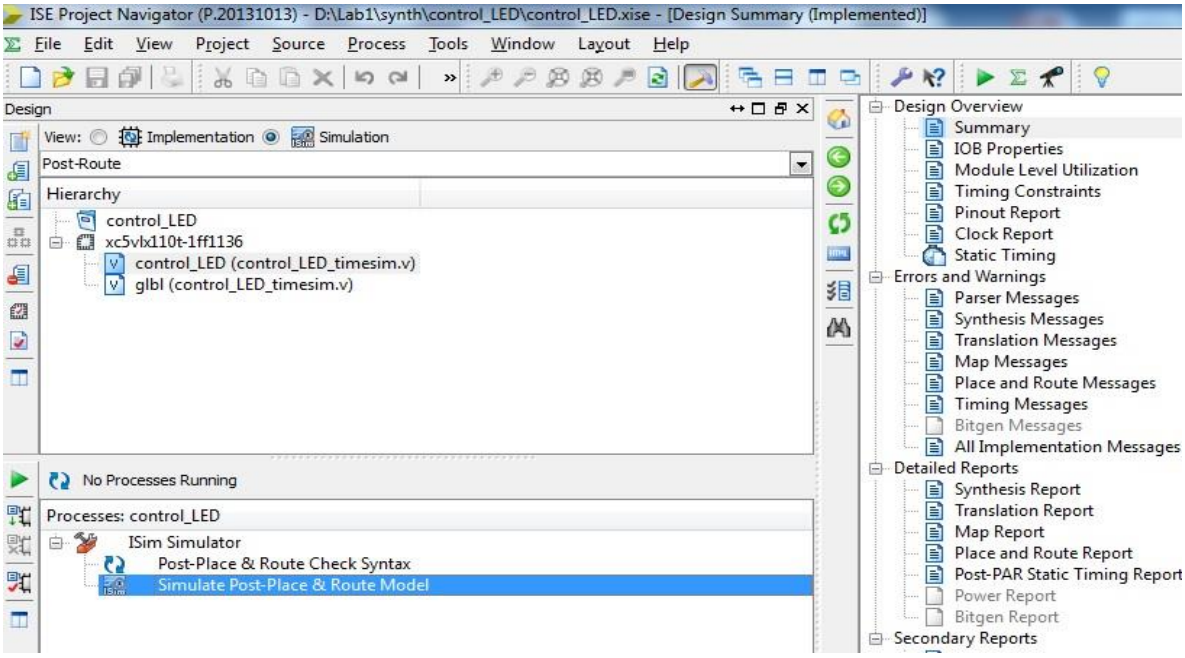


Fig. 9: Post-place and route timing simulation

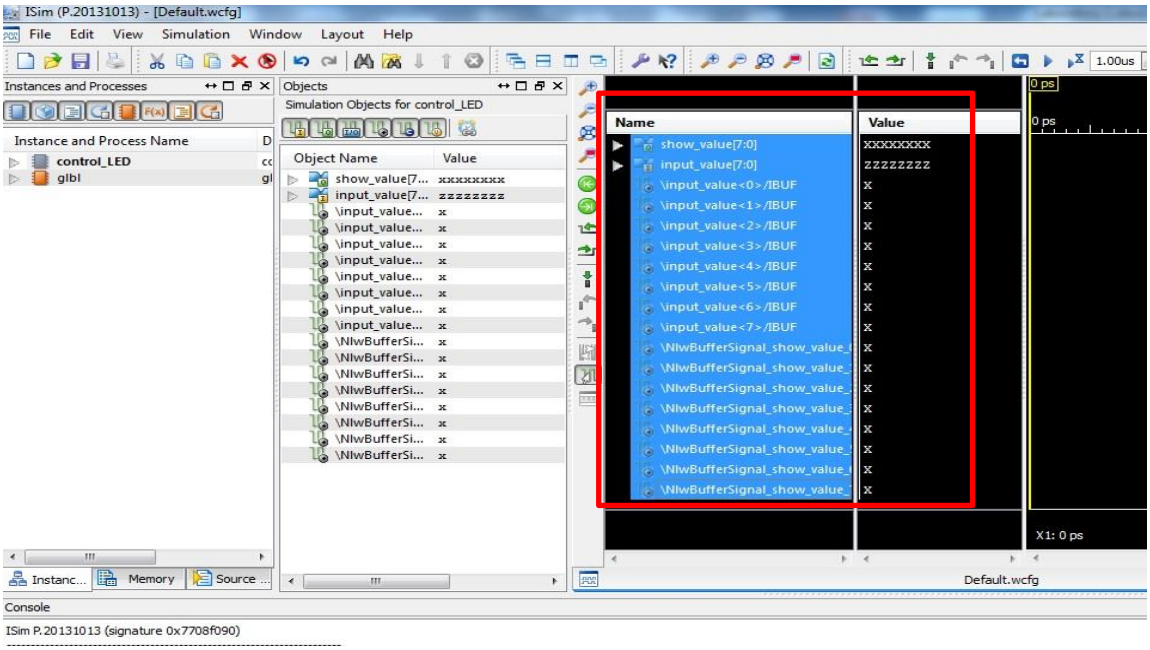


Fig. 9: Post-place and route timing simulation window

39. The simulation use testbench file as input so the waveform is quietly like that in behavior model. However, because it's a more realistic simulation, wire latency will be considered. Zoom in at around 100ns and you will see **an clear latency** from input_value to show_value.

40. Go back to **Implementation** view and double click on **Generate Programming File** in Fig.7. (You have now successfully generated the bit-stream to program your FPGA. Go back to ***synth*** directory to see if a ***control_LED.bit*** file exists or not. This file is the bit-stream file. You are now ready to program your FPGA)
41. Connect the power supply to your board (board designator P20). Turn the power switch (SW1) ON. The power good LED (DS41) should turn green. INIT LED should turn green to indicate that the FPGA has successfully powered up.
42. Hold the DIP switch SW3in the configuration: 00010100 (1 to 8). Connect the USB-JTAG Programming Cable to a USB port on your computer and toPC4 JTAG on the board.
43. Double click **Configure Target Device** (red box) under **Processes** in Xilinx ISE as shown in Fig. 10. Click OK on the resulting dialog box. This will open iMPACT which is Xilinx's tool for programming the FPGA.

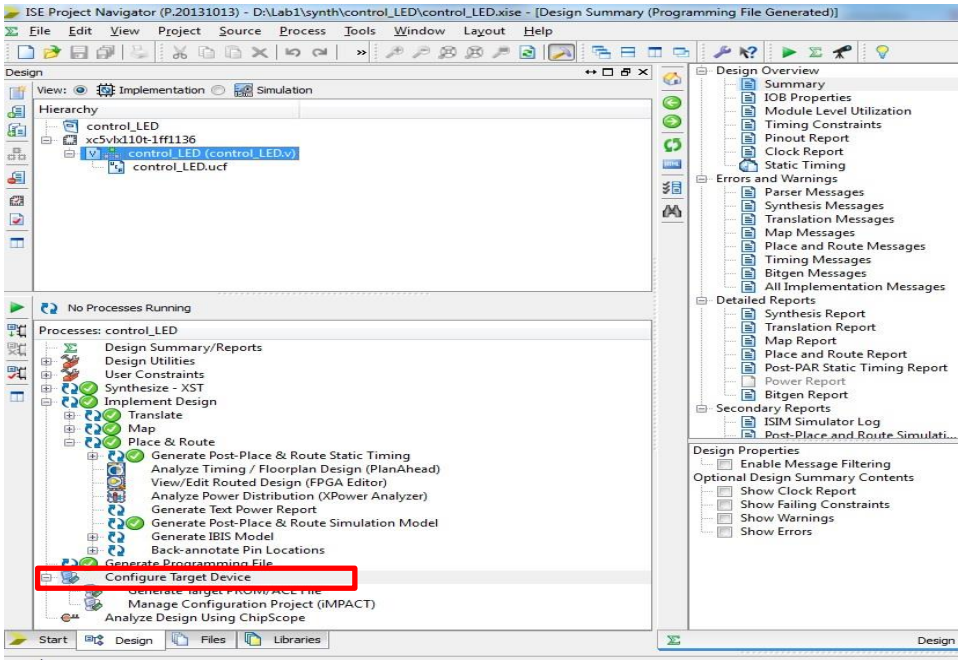


Fig. 10: Configure target device

44. Double click on **Boundary Scan** under **iMPCAT flows** in the iMPACT GUI.
45. Right click and select Initialize chain as shown in Fig. 11.

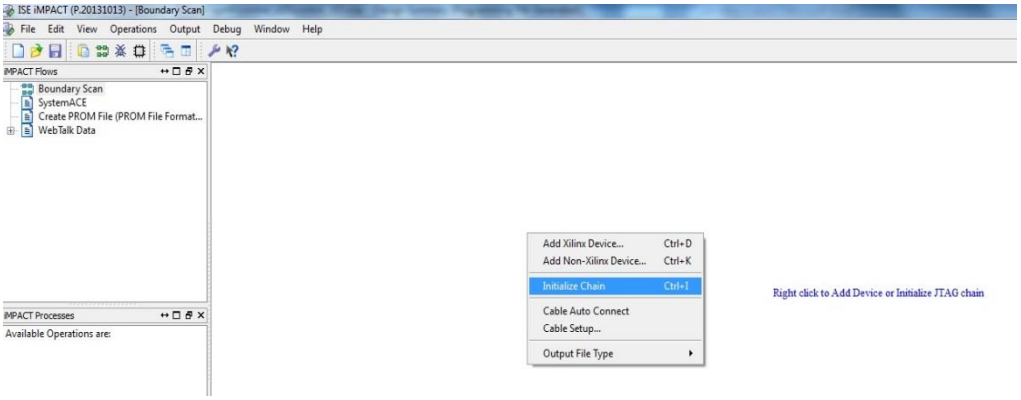


Fig. 11: Initializing JTAG Chain

You should see something similar to Fig. 12. Click NO and cancel on any subsequent dialog box that pops up. You can see the XC5VLX110T device identified in the chain.

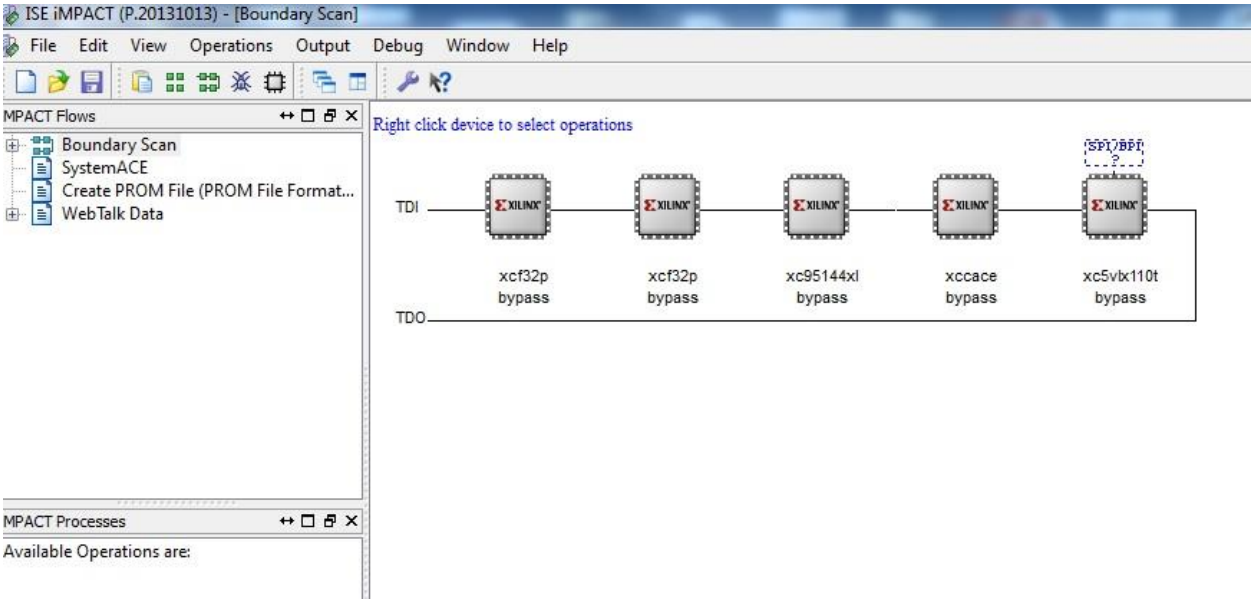


Fig.12: The JTAG Chain on the Evaluation Platform

46. Right click on the FPGA device and select **Get Device ID**. You should see **ReadIdcode Succeeded message.** This gives you the JTAG ID code of your device in the console window for informative purpose. You can skip this step in later exercises.
47. Right click on the FPGA device and select **Assign New Configuration File**. Browse to ***synth*** directory and select the ***control_LED.bit*** file. Select NO on any dialog box that may pop up related to SPI or BPI PROM.
48. Right click on the FPGA device and select **Program.** Click **Apply** and **OK** for device 5 on the resulting dialog box that pops up. You should see **Program Succeeded** below the JTAG chain and **Programmed Successfully** in the console. You should also see the

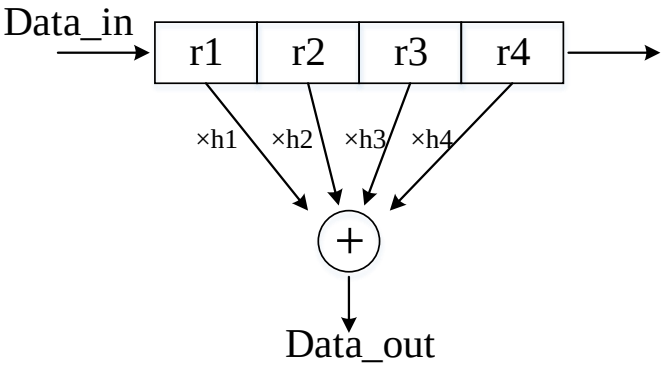
DONE and the INIT LEDs green on the board. You have successfully programmed the FPGA with your design.

49. Use the GPIO DIP switches in the extreme right corner (near the LCD display) to input different values to the FPGA and see the corresponding green LEDs light up. Note that switch 1 on the GPIO DIP switch corresponds to LED0 in the example UCF. Individual switches on the GPIO DIP switch are marked 1 to 8 on the switch panel. Show your results to your TA/Instructor.

After you have done Exercise 1A above, proceed to do exercises 1B. All the steps (1-46) listed above will need to be repeated for 1B. Exercises 1B are described next.

Exercise 1B:

1) You are required to implement a simple FIR filter and pass through behavioral simulation. The FIR filter is shown below. There is a line buffer with four elements. In every clock cycle, the design accept one input data Data_in, r1 will buffer its value, and the original data in the line buffer will shift right. For example, data in r2 will be stored in r3. The data in the line buffer will multiply with a coefficient and at last you need to sum up the four multiplication result to get the final result. We provide you with an example code "fir_example" together with the testbench "fir_test" but you are required to fill in some verilog code in the missing part under the comments to complete the function described above. Finally you need to pass through simulation to get correct result.



2) You are required to implement a 3-to-8 decoder. We provide you with an example code "decoder.v" but you are required to fill in some verilog code in the missing part under the comments to complete the decoder. You will be required to use the 3 DIP switches for the 3 inputs and show the decoder output by driving the LEDs. Also remember that you need to add *.ucf* file for IO configuration.

Additional questions for you to figure out if interested (optional):

1. Can you list the clocks available on the board along with their board designations and related FPGA pin numbers?
2. Can you reset the FPGA (there is a PROG switch on the board for this) and reprogram it using any of the available bit-streams from exercises 1A, 1B and 1C?
3. Can you show where a fan can be placed on the board to remove the heat generated when a FPGA is executing a design?
4. Can you implement a simple 3 bit adder and show the result of addition using the user LEDs just the way you used LEDs in the above exercises?
5. In the lab1, you forced signals to do the simulation. Instead, if you want to use a testbench to simulate your design, can you figure out how to use it using ISim? Try reading Xilinx UG682 (Isim in-depth tutorial) and pay special attention to page 2, Fig. 2-2 and pages 16-17, Fig. 2-7. If you could understand these two figures and related text, you can apply the concept to Lab 1 exercises for simulation using testbench in ISim.

APPENDIX I

Table 1-5: **DIP Switch Connections (SW8)**

SW4	FPGA Pin
GPIO_DIP_SW1	U25
GPIO_DIP_SW2	AG27
GPIO_DIP_SW3	AF25
GPIO_DIP_SW4	AF26
GPIO_DIP_SW5	AE27
GPIO_DIP_SW6	AE26
GPIO_DIP_SW7	AC25
GPIO_DIP_SW8	AC24

Table 1-6: **User and Error LED Connections**

Reference Designator	Label/Definition	Color	FPGA Pin	Buffered
DS20	LED North	Green	AF13	Yes
DS21	LED East	Green	AG23	Yes
DS22	LED South	Green	AG12	Yes
DS23	LED West	Green	AF23	Yes
DS24	LED Center	Green	E8	Yes
DS17	GPIO LED 0	Green	H18	Yes
DS16	GPIO LED 1	Green	L18	Yes
DS15	GPIO LED 2	Green	G15	Yes
DS14	GPIO LED 3	Green	AD26	No
DS13	GPIO LED 4	Green	G16	Yes
DS12	GPIO LED 5	Green	AD25	No
DS11	GPIO LED 6	Green	AD24	No
DS10	GPIO LED 7	Green	AE24	No
DS6	Error 1	Red	F6	No
DS6	Error 2	Red	T10	No