ECE2029 Introduction to Digital Circuit Design

Lab # 3 – Designing a 4-bit adder with overflow detector

(Implementing Combinational Functional Blocks)

1. Introduction

This laboratory introduces you to combinational logic design and its implementation on FPGA. You will practice all the techniques we learned so far from the lectures. In addition, you will also learn structure design in Verilog. For a large design, we typically design each component and test it. Then, we put all these components together in Verilog.

2. Prelab

In general, good design practice means that you do your design, simulation and synthesis **before** you even get near the actual hardware! This saves a lot of time and frustration in the lab, because you know your circuits have the right logic behavior before you start! Please have you prelab completed and checked off by the TA when you enter the lab.

Watch Tutotial(s): https://youtu.be/ql3llzXIqVM | https://youtu.be/k7Y_Mejmid4

3. Verilog Basic for Logic Design

In Verilog the Boolean operations are written as:

& for AND, | for OR, ~ for NOT and ^ for XOR.

Important Note – Generally there is no implied order of operations among the Boolean operators in Verilog. You must enforce order of operations with brackets ( ).

Ex: F = (A B' + C)' + A'C

```
// Verilog assignment for F
assign F = (~((A & (~B)) | C)) | ((~A) & C);
```

In this lab, you can use the assign statement to implement all the logic expression that you have derived in the prelab using K-map.

OR

You can also derive the expression using Logisim. Go to project, select analyze circuit, enter variable names for inputs and outputs, select the truth table tab and enter the output. Select expression tab, you can use the expression and modify it to create a Verilog code.

NOTE: Take help from lab# 2 if needed, this lab write-up doesn't include all the steps.

4. Design hexadecimal 7-segment display for 4-bit binary input

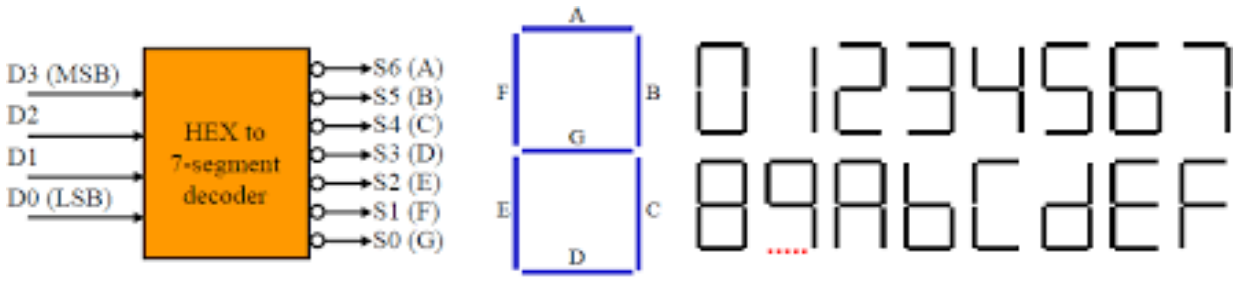


Figure 1. Diagram of the hexadecimal 7-segment display module

Figure 1 above shows the diagram of the function block.

From Figure 2 below, you should notice that each 7-segment display also has an enable signal, AN0, AN1, AN2, AN3 respectively. These enable signals are active LOW, which means AN0 must be set to 0 if you want to send your display to the left most digit. Since all 7-segment digits share the same input pins (CA to CG), they actually will display the same data if you set AN0 = AN1 = AN2 = AN3 = 0. For this lab, that is perfectly acceptable. You can certainly pick the left most or right most digit only by setting the other 3 AN signals to 1, if you prefer to do so. In future labs, we will learn a "trick" to use all 4 segments and make them to display 4 different digits.

As often, in this lab, we'll use switches and buttons as inputs to the FPGA. We also use LEDs and 7-segment displays as outputs of the FPGA. Figure 2 below shows the actual schematics of the switches, LEDs, pushbuttons, etc. on the Basys 3 board. You can also refer to the Basys 3 reference manual (Page 15).

You will use dipswitch (SW3-SW0) as 4-bit inputs (D3-D0) and 7-segment (A, B, C, D, E, F, G OR Seg[6], Seg[5],….Seg[0]) as outputs. The truth table and logic expression is available from your prelab. We also did this in the class but with active high logic (Lecture # 12 Decoders).
See the screenshot of constraint and source files for this part in figure 3.

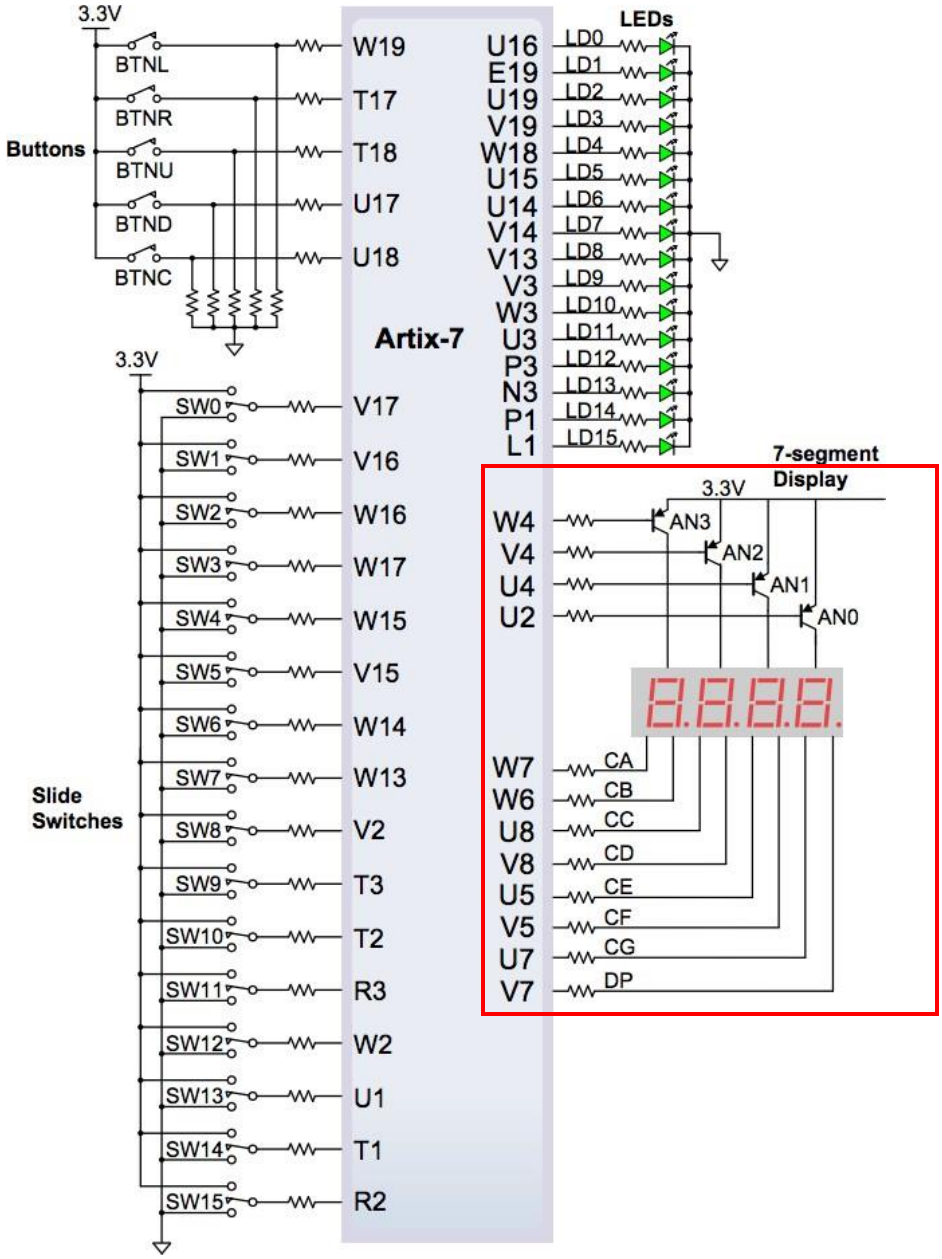


Figure 2. FPGA connections to the switches, buttons, LEDS, and 7-segment display

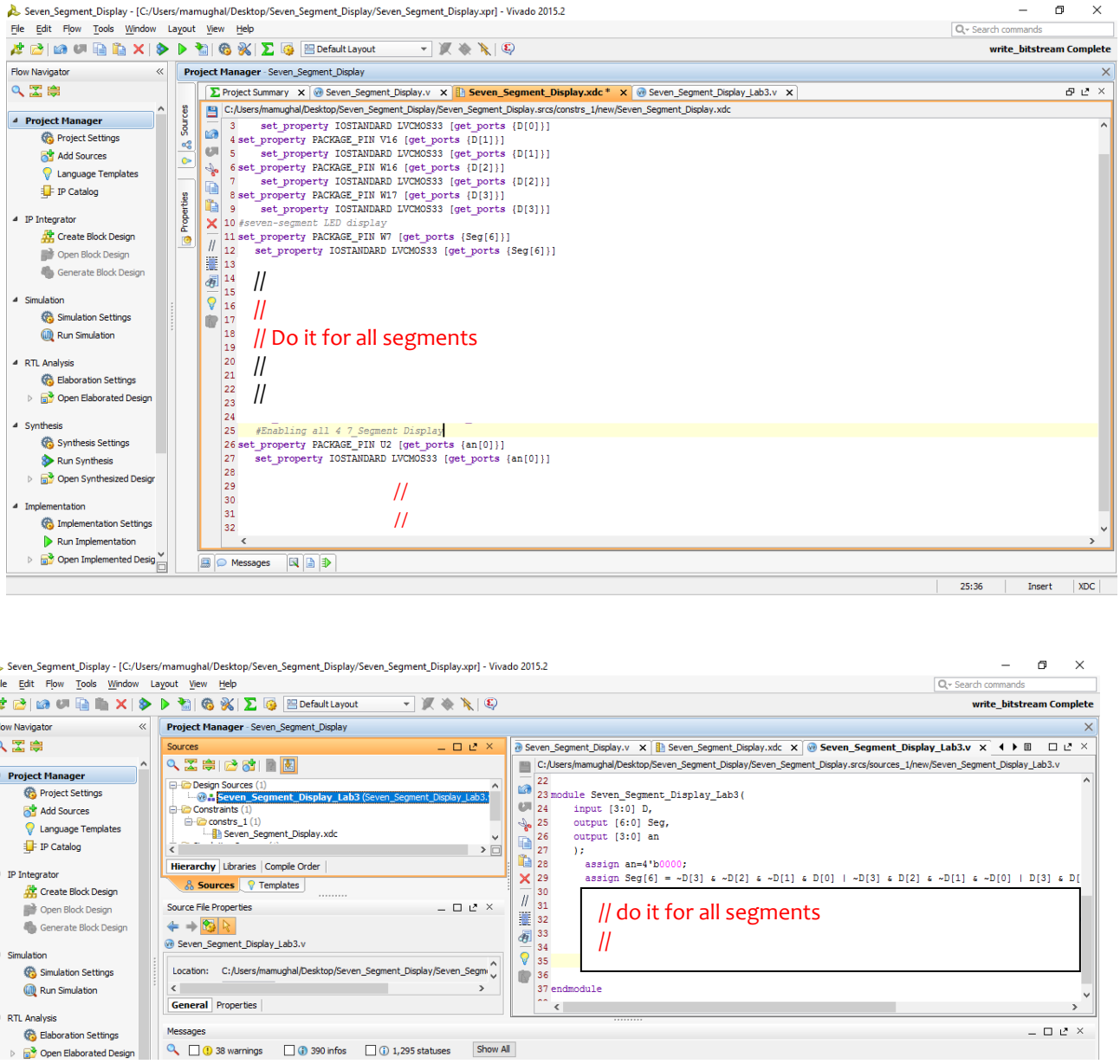

Seven_Segment_Display - [C:/Users/mamughal/Desktop/Seven_Segment_Display/Seven_Segment_Display.xpr] - Vivado 2015.2
write_bitstream Complete
Project Manager - Seven_Segment_Display
C:/Users/mamughal/Desktop/Seven_Segment_Display/Seven_Segment_Display.srcs/constrs_1/new/Seven_Segment_Display.xdc
// Do it for all segments
Seven_Segment_Display - [C:/Users/mamughal/Desktop/Seven_Segment_Display/Seven_Segment_Display.xpr] - Vivado 2015.2
write_bitstream Complete
C:/Users/mamughal/Desktop/Seven_Segment_Display/Seven_Segment_Display.srcs/sources_1/new/Seven_Segment_Display_Lab3.v
// do it for all segments

Figure 3

5. Design an one-bit full adder [Skip steps a to c, if you've already done it]

a) **Follow steps 1 to 9 from lab 2**, to create, design, and synthesize your project, fuller adder. Apply the logic expression as you developed in the prelab. [see Figure 4]

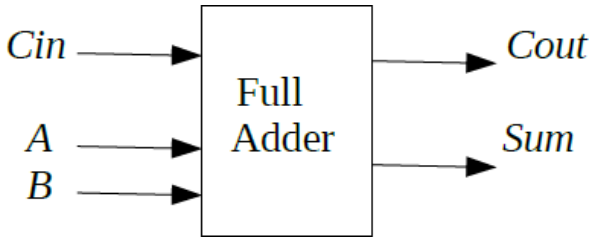


Diagram of the 1-bit fuller adder

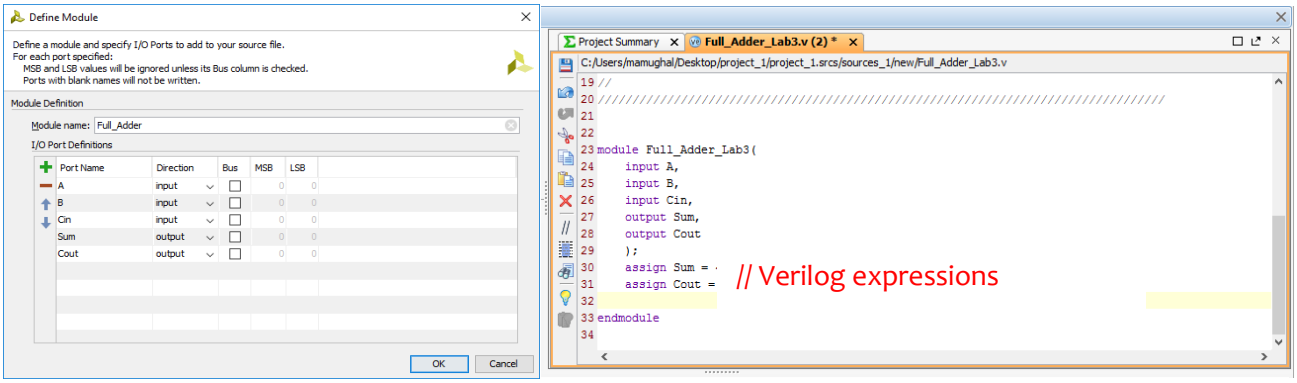


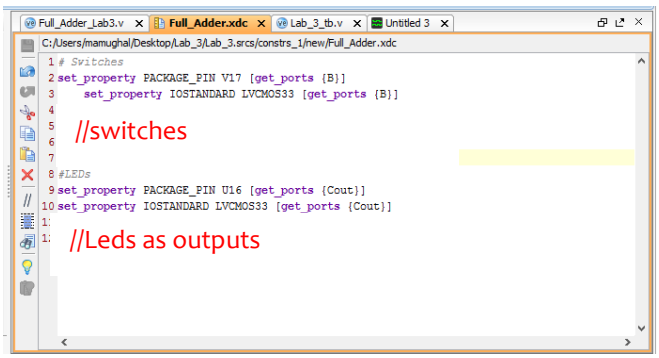
Figure 4

b) Next click Add Sources and select Add or create constraints.

Associate switches SW2 [W16], SW1 [V16], and SW0 [V17] with inputs "C_{in}", "A", and "B" and LEDs 0 [U16] and 1 [E19] with outputs "Cout" and "Sum in the user constraints file (.xdc). [See Figure 5]

Note: **User_Constraint_File(UCF)_Basys_3.txt**

c) Generating a Test-bench Waveform for Functional Verification



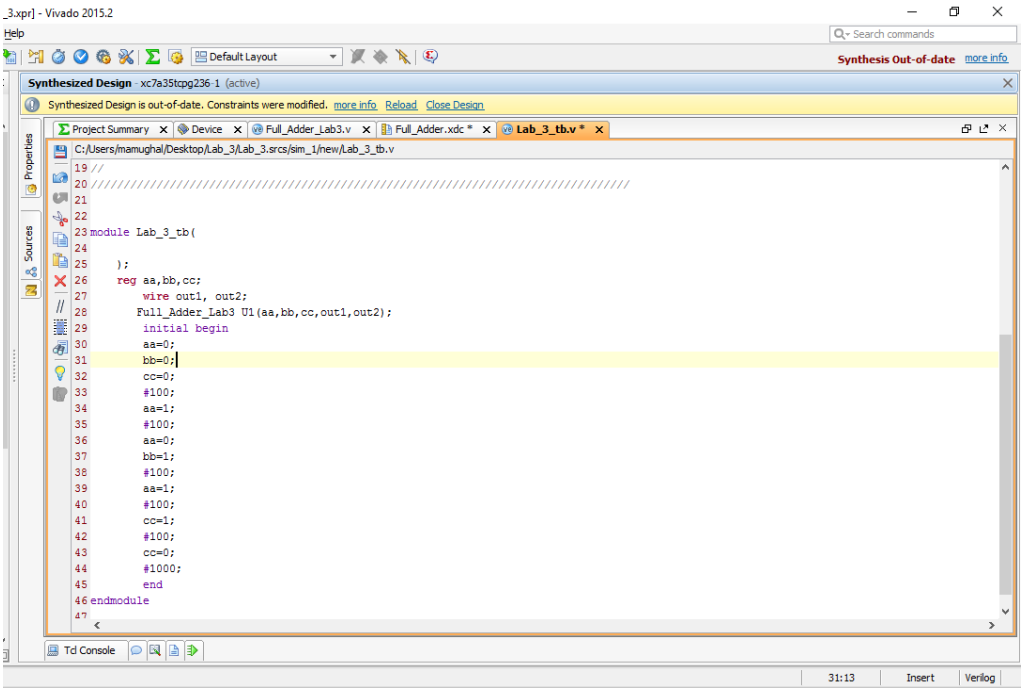


Figure 6

Under the Project Manager Simulation Sources, right click on sim_1(1) and Add Sources. Choose Add or create simulation sources. Create a testbench file and click finish.

Right click the new testbench file and Set it as Top module. Double click on the file name to open it in the editor window and add the following Verilog code. Notice that your are *instantiating* the Full_Adder_Lab_3 module you will define as U1. We are using simulator input signals called aa, bb and cc as inputs A, B, and Cin respectively and have assigned simulator outputs out1 and out2 as Sum and Cout respectively. Then we set the values of aa, bb and cc to test the all the settings we wish to test. [see Figure 6]

Select your testbench file then click on Run Simulation then Run Behavioral Simulation. In the simulation window hit the Run All button then right click in the window and select full view. Now verify the inputs and outputs in the timing diagram. [see Figure 7]

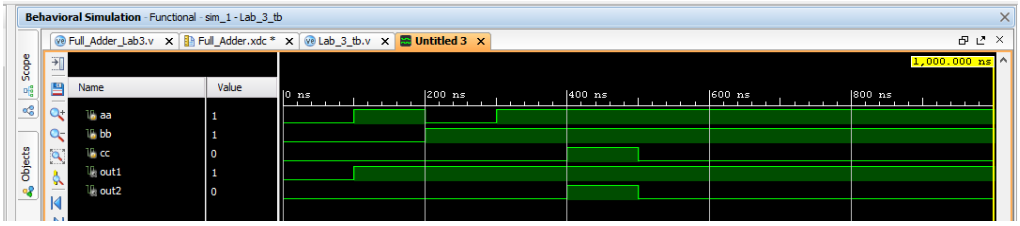


Figure 7

d)Under Design Sources, select your Full_Adder_Lab3 module and follow steps 22 to 25 from lab 2, perform implementation and generate a programming (.bit) file. Download your full adder to the Basys 3 board. Test your circuit and then show your Full Adder to the TA before continuing on.

6. Design an 4-bit 2's complement adder

In this Verilog module you will **learn how to hierarchical design**. You will design a 4-bit adder by "instantiating" copies of the Full_Adder_Lab3 module from step 5.

a) Start a new project. Enter the inputs as Bus, each with 4 bits. The Sum output as a bus of 4-bits. The signed overflow (OF) and C_MSB as one-bit output. [see Figure 8]

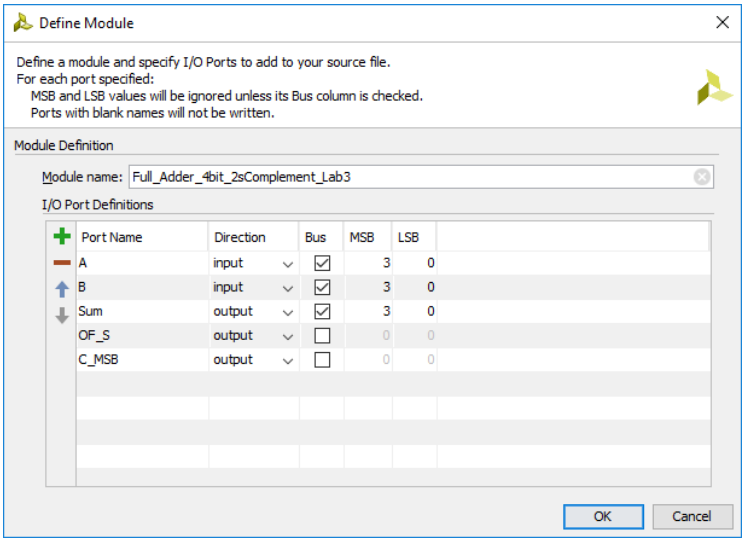


Figure 8

b) Once Vivado tool generates the code template, **modify the code to use the Full Adder modules 4 times (use exactly the same file name that you used for creating .v file for one-bit full adder, here we used "Full_Adder_Lab3"** [see Figure 9]. You need to copy and add the "fulladd.v" file from Step 5 to this project also (Click add sources, locate .v file and add it). Otherwise, the tool cannot find your one-bit fuller adder design.

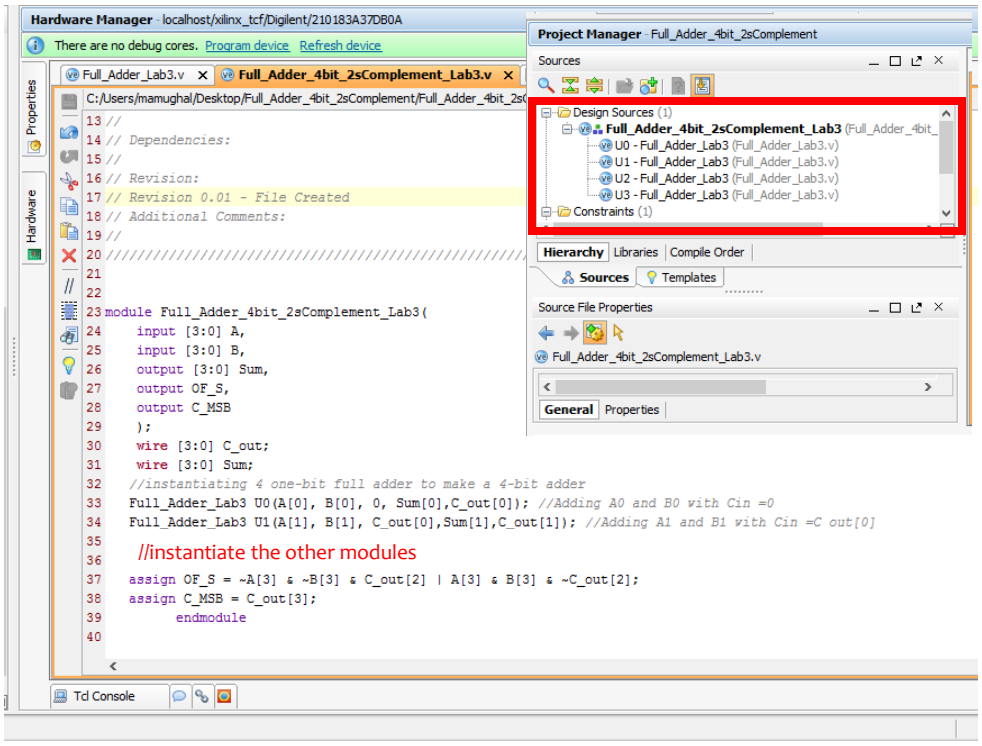
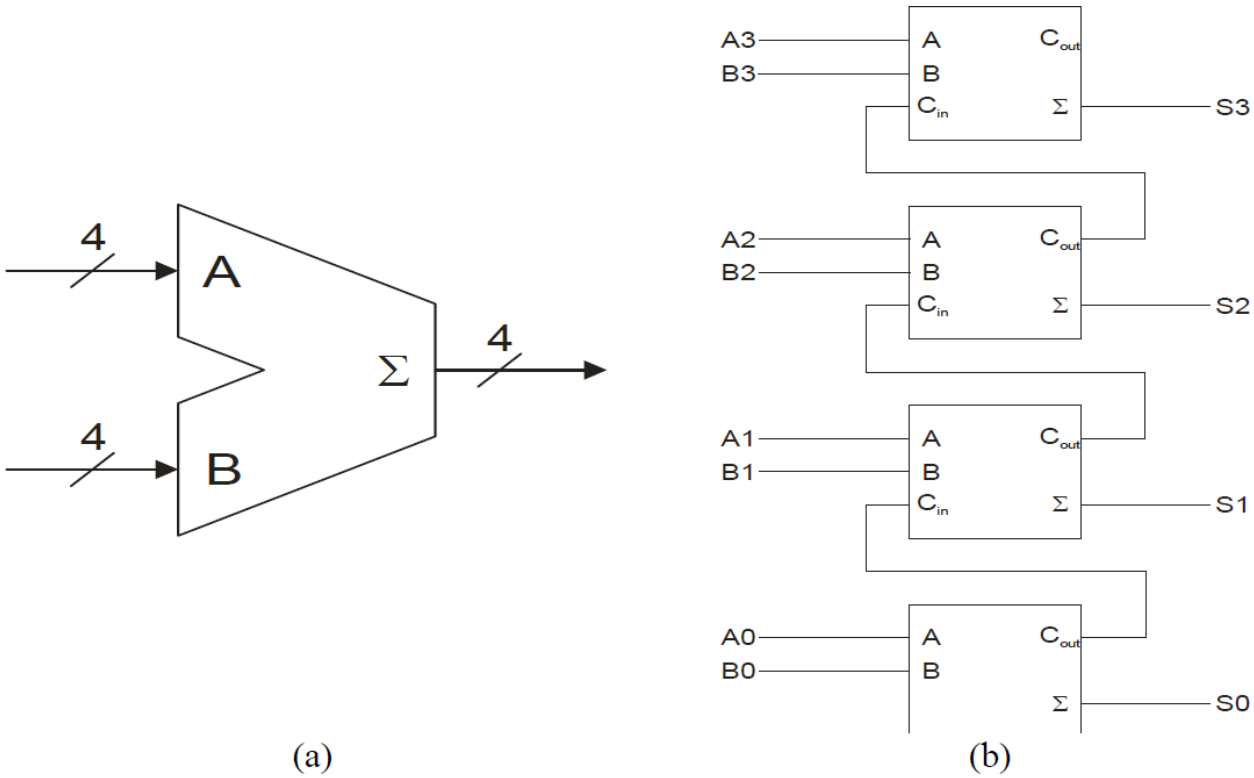


Figure 9

c) Now **create a constraint file (.xdc).** Use 8 DIP switches (SW0~SW7) as inputs and 4 LEDs (LD0~LD3) as 4-bit output. For the two different overflow detector logic that you designed in Prelab, we can use LD7 and LD8. Map your design appropriately using the .xdc file. [see Figure 10]



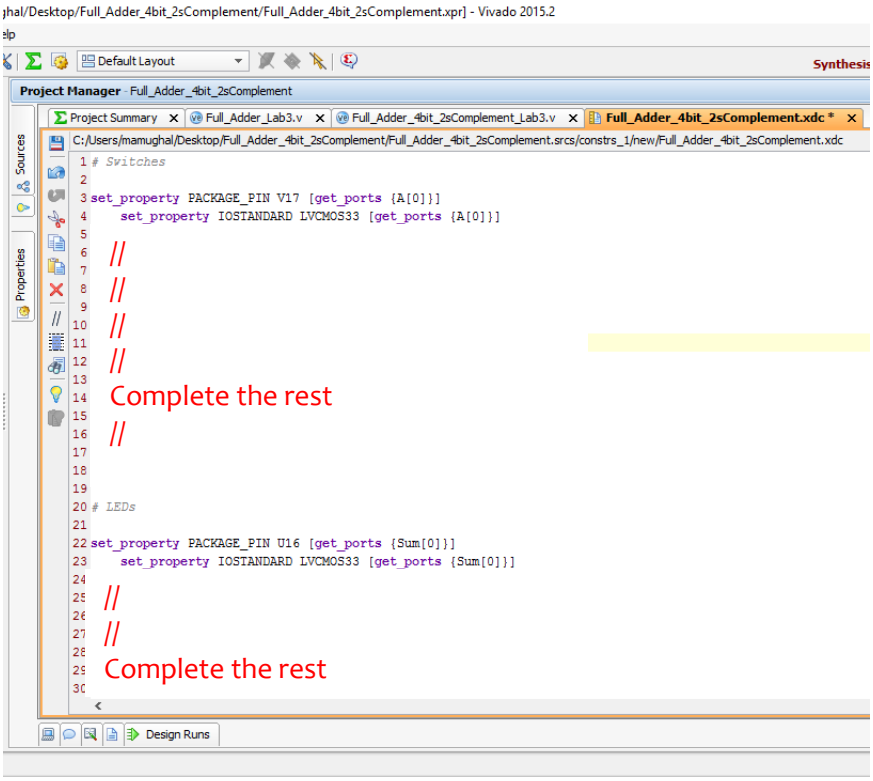


Figure 10

d) Generate a programming (.bit) file and download your 4-bit adder to the Basys 3 board. Test your circuit with 2's complement inputs and then show your Full Adder to the TA before continuing on.

It is very important to know that the computer can only perform addition. If there is a subtraction, it basically performs the 2's complement on the 2nd operands, followed by an addition operation. This also applies to your homework and exam problems. Do NOT perform binary subtraction directly. As we shown in class, it will give you wrong results which is the fundamental reason why we use 2's complement numbers today.

7. Top Module: Output 4-Bit Adder to 7-Segment Display

a) This is the final integration of Lab3. Create another new project to perform 4-bit addition with 7-segment display. Below is a list of the input and output signals.

	Input A	4 bits	A(3) ~ A(0)
	Input B	4 bits	B(3) ~ B(0)
	Output Segs7 (a-g)	7 bits	Segs7(6) ~ Segs7(0)
Overflow	Output OF_S	1 bit	OF_S
	Output OF_U		OF_U
	Output ANo	1 bit	An(0)

b) Following the same approach as in Step 6, instantiate two components 4-bit adder and hexadecimal display (see example below), and then connect them together use 4 wires Sum(3)~Sum(0). Map your design using the appropriate .xdc file.

Example Code

```
Full_Adder_4-bit_2sComplement_Lab3  U4(A, B, Sum, OF_S, OF_U);
Seven_Segment_Display_Lab3 U5(Seg, Sum, An);
```

c) Generate a programming (.bit) file and download your design to the Basys 3 board. Test your circuit with 2's complement inputs and then save the bit file for the TA for complete sign-off.