California State University Long Beach
Computer Engineering and Computer Science Department
CECS 201 – Computer Logic Design I
Lab 6 – Combinational Datapath Components: Adders and Subtractors

Objectives:

- To implement and test a design using Vivado SW and FPGA board
- To practice making **hierarchical** design in Verilog
- To learn about **delay** in Verilog
- To support learning about combinational datapath components: an Adder and a Subtractor

Theory:

This lab assignment will help you put in practice concepts covered in chapter "Combinational Datapath Components" particularly sub-sections "Adders", "Signed numbers in binary" and "Subtractors": please make sure to study the corresponding chapter and do corresponding homework assignment in preparation for this lab assignment.

Setup:

Please follow the setup instruction provided in the Lab 1 assignment.

Assignment:

This lab assignment supports learning about design of Combinational Datapath Components: Adders and Subtractors. You will need to create a module and a corresponding testbench for the following designs: Full Adder, 4-bit Adder, 4-bit Adder-Subtractor. The full adder will be created using structural design, where the remaining will be coded using hierarchical design. The 4-bit Adder will implement a Ripple Carry Adder, and you will learn about propagation of Carry output, and how it might affect Sum output using **delay** with an **assign** statement. Finally, you will **implement** a Full Adder (and 4-bit Adder, for Extra Credit) on the FPGA board and test it. Follow the instructions in the Steps section and the video tutorials to perform those steps in the same project.

In the conclusion portion of the report, write the expected results and reflect if the simulation shows the expected results. Also add if the FPGA board shows expected output value for each test case. If not, try to figure out why and write this in the report as well.

Steps:

Both Student Virtual Lab (SVL) and Vivado are installed on the computers in the lab. Feel free to install them on your laptop or desktop.

1. If you need to use SVL, connect to SVL, as per instructions on BB

2. Create a folder **lab6** in the **labs** folder for lab 6 assignment

3. Start Vivado 2016.x and follow the instructions on how to create a project (x=2 for CSULB lab computes and SVL). Project name should be lab5-v0 (later, you may wish to create more than one version of the lab assignments *if needed*)

4. For each module in this lab populate the top portion with
   // Company: CECS 201 – Fall 2019
   // Engineer: your name

5. Create a new module to implement a Full Adder with module name, inputs and outputs as shown in the Figure 1. The function for the full adder is described using following Boolean Equations:

   s = a ⊕ b ⊕ ci;
   co = a•b + a•ci + b•ci;

   Use assignment statement to implement the functions. Verilog operator for *bit-wise exclusive* or is **^**; please use **proper** Verilog bit-wise operators for • (and) and + (or).

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////
3  // Company:
4  // Engineer:
5  //////////////////////////////////
6  module full_adder(
7      input a,
8      input b,
9      input ci,
10     output s,
11     output co
12     );
13
14 // add your code here
15
16 endmodule
17
```

Figure 1. Module full_adder, its inputs and outputs, and the module body.

6. Create a testbench for Full Adder as per Figure 2. Instantiate the full adder using the variables defined at the beginning of the testbench. Add tests test inside the “initial” statement to perform comprehensive testing.

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////////////////////////////////

module full_adder_tb;

    // Inputs
    reg a_tb;
    reg b_tb;
    reg ci_tb;

    // Outputs
    wire s_tb;
    wire co_tb;

    // Instantiate the Unit Under Test (UUT)
    full_adder uut (              );

    initial begin
        // Initialize Inputs
        a_tb = 0;
        b_tb = 0;
        ci_tb = 0;
        // add your test cases here

    end
endmodule

```

Figure 2. Testbench full_adder _tb

7. Simulate the testbench and observe the outputs in the simulation window (run the simulation for additional time, if needed, to observe all test cases). Create a table listing expected output values for all test cases and include it in your report alongside the screenshot of the output.

8. Create a new module to implement a 4-bit Adder (Figure 3.) using 4 instances of a Full Adder. Use module name, inputs and outputs as shown in the Figure 4. Use Figure 3 to help you fill in the blanks in the module body. Note that you ***cannot*** use Co to compute V, because it is an **output port**. Therefore, you need to declare wire Co4, use it for computing V, and assign its value to Co.

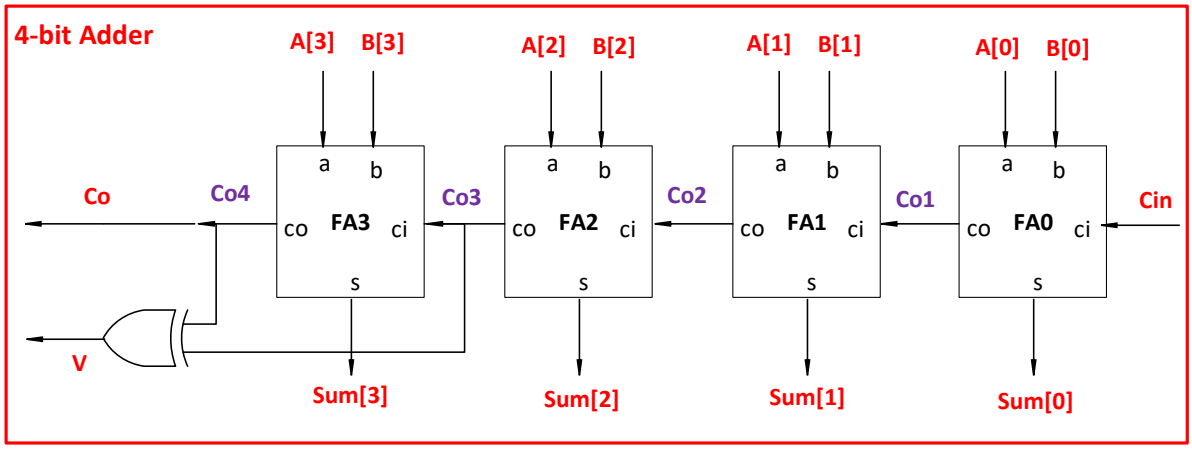

Figure 3. Block diagram of a 4-bit Adder

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //////////////////////////////////////////////////////////////////////////////////
6  module four_bit_adder(
7      input [3:0] A,
8      input [3:0] B,
9      input Cin,
10     output [3:0] Sum,
11     output Co,
12     output V
13     );
14
15 wire Co1, Co2, Co3, Co4;
16
17
18 assign V =          ;
19 assign Co =          ;
20
21 // instantiate 4 instances of mudules full_adder
22 // name them FA3, FA2, FA1, FA0 (left to right in the scematic)
23
24
25 endmodule
```

Figure 4. Module four_bit_adder, its inputs and outputs, and the module body.

9. Create a testbench for the 4-bit Adder as per Figure 5. Instantiate the 4-bit Adder using the variables defined at the beginning of the testbench. Study the code inside “initial” statement: what are the values for A, B, and M it produces?

```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //////////////////////////////////////////////////////////////////////////////////
7
8  module four_bit_adder_tb;
9
10     // Inputs
11     reg [3:0] A;
12     reg [3:0] B;
13     reg Cin;
14
15     // Outputs
16     wire [3:0] Sum;
17     wire Co;
18     wire V;
19
20     integer i;
21
22     // Instantiate the Unit Under Test (UUT)
23     four_bit_adder uut (    );
24
25     initial begin
26     for (i =0; i < 16; i = i+1)
27     begin
28       if(i%4) Cin = 1; else Cin = 0;
29         A = i;
30         B = i + 10;
31
32         if(i==15)  begin A = 1; B = 16'hFFFF; end
33
34         #100;
35
36         if (Sum !== A+B+Cin)
37           begin
38             $display("Error Sum = %d Co = %d V = % d for A= %d B= %d Cin=%d", Sum,  Co, V, A, B, Cin);
39           end
40       end
41     end
42
43 endmodule
44
```

Figure 5. Testbench for four_bit_adder

10. Simulate the testbench and observe the outputs in the simulation window (run the simulation for additional time if needed to observe all test cases). Create a table listing expected output values for all test cases and include it in your report alongside the screenshot of the output.

11. You will explore, now, propagation of Carry outputs of Full Adder modules and the effect it produces to outputs of the 4-bit Adder. To do so, you will use **assign** statement with **delay**. For example:

    assign #5 Q = x + y + z;

The statement will compute the value of (x + y + z) and assign it to Q 5ns after the value has been computed. Please remember that the right-hand side of the expression is (re)evaluated when any of the variables in the expression on the right-hand side changes its value. Delay is *used* in **simulation only** and it is *ignored* in **synthesis** (after the synthesis delay of physical components will be used in post-synthesis analysis and simulation, but that is out of scope of this class).

Open the file containing full_adder module and duplicate the original statements containing computation of s and co. Use "//" to comment out one copy of the assignment statements (to preserve it for later). Add 10ns delay to the second copy of both assign statements that compute s and co (like the code in Figure 6.) Save the file and simulate again the design for 4-bit Adder. What is the difference and why it appears? Include the explanation and the screenshot of the results in the repot.

```
14  // add your code here
15  // without the dealy
16  //assign       s =
17  //assign       co =
18  // with the delay
19  assign     #10 s =
20  assign     #10 co =
```

Figure 6. Adding delay to s and co

12. Remove the delay to proceed with creation of the next module: comment out the statements with the delay and remove the comments from the ones without. (Figure 7.)

```
14  // add your code here
15  // without the dealy
16  assign       s =
17  assign       co =
18  // with the delay
19  //assign     #10 s =
20  //assign     #10 co =
```

Figure 7. Removing the delay

13. Create a new module to implement a 4-bit Adder-Subtractor (Figure 8.) with module name, inputs and outputs as shown in the Figure 9. Fill in the blanks and to instantiate one instance of a four_bit_adder module and one instance of a 4-bit 2-to-1 mux, while connecting them to implement the schematic given in the Figure 8.

You have several options for implementing a 4-bit 2-to-1 mux. You may add a file that implements the mux: modify the implementation of 4-bit 4-to-1 mux from the previous lab to create a 4-bit 2-to-1 mux; then create one instance of the mux in this module.

Alternatively, to create a 4-bit 2-to-1 mux you may use an assignment statement with conditional operator (http://verilog.renerta.com/mobile/source/vrg00010.htm). The expression in the parenthesis (E == 0) is evaluated: if it is true, R gets value of T; and if it is false, R gets the value of F, as per example below. This operator can be used for scalar and vector variables.

assign R = (E == 0)? T : F;

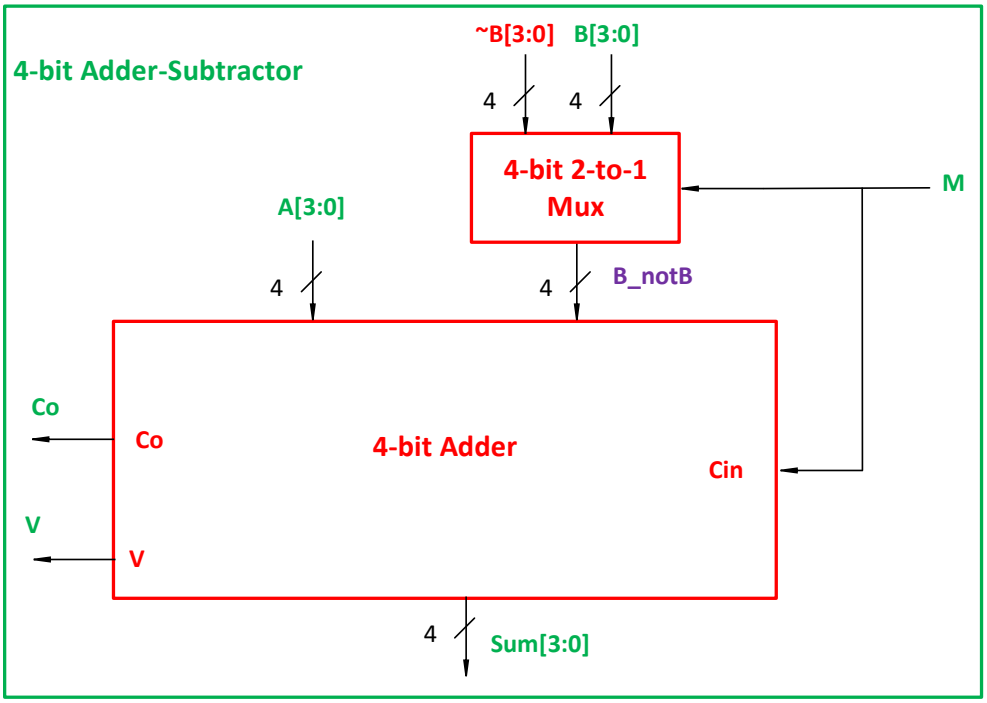

Figure 8. Block diagram of a 4-bit Adder-Subtractor module

14. Create a testbench for the 4-bit Adder-Subtractor as per Figure 9. Instantiate the 4-bit Adder- Subtractor using the variables defined at the beginning of the testbench. Study the code inside "initial" statement: what are the values for A, B, and M it produces?

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////////////////////////////////
module four_bit_add_sub_tb;
    // Inputs
    reg [3:0] A;
    reg [3:0] B;
    reg M;
    // Outputs
    wire [3:0] Result;
    wire Cout;
    wire V;
    integer i;
    // Instantiate the Unit Under Test (UUT)
    four_bit_add_sub uut (  );
    initial begin
    //testing addition
     for (i =0; i < 16; i = i+1)
      begin
        M= 0;
        A = i;
        B = i + 3;
        if(i==15)  begin A = 1; B = 16'hFFFF; end
        #100;
        if(Result !== A+B)
        begin
            $display("Error Result = %d for A= %d B= %d M=%", Result, A, B, M);
        end
     end
     // testing subtraction
     for (i = -7; i < 7; i = i+1)
      begin
        M= 0;
        A = i;
        B = i + 3;
        #100;
        M = 1;
        if((16-Result) !== A-B)
         begin
            $display("Error Result = %d for A= %d B= %d M=%", Result, A, B, M);
         end
        end
    end
endmodule
```

Figure 9. Testbench for four_bit_add_sub

15. When all modules created and added to the project, the “Sources” window will show the hierarchy as in Figure 10.

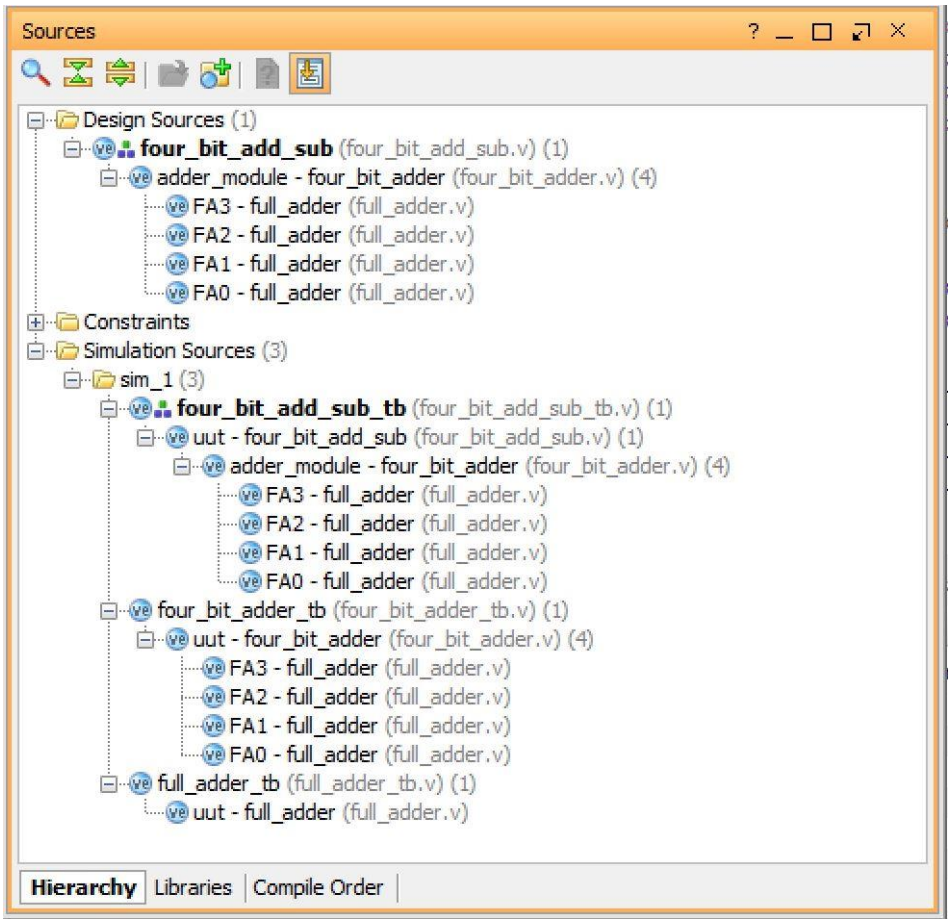

Figure 10. Sources window show correct hierarchy

16. Simulate the testbench and observe the outputs in the simulation window (run the simulation for additional time if needed to observe all test cases). Create a table listing expected output values for all test cases and include it in your report alongside the screenshot of the output.

Synthesis:

17. Make sure full_adder and full_adder_tb are both “set as top”, and that full_adder is functionally correct.

18. Constraint file is used to map inputs of your module to switches and LEDs on the FPGA board. Add a constraint file using “Add source” option “Add or create constraints”, press “Next”. Name the file “full_adder_NexysA7-100T”, press “OK” and “Finish”. In “Sources” window, expand “Constraints” and “consts_1”, to see the added file, and double click on it, to open it for editing.

19. Download file “MasterConstraint_NexysA7-100T.txt” from Lab 4 folder on BB and save it locally. Open the file with the text editor (Notepad or WordPad), copy its content to clipboard, and paste the content inside the “full_adder_NexysA7-100T.xdc” file in the editor.

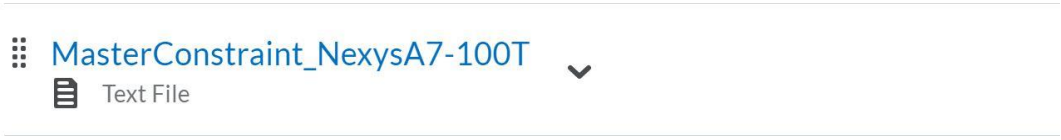

Figure 11. Master constraint file for Nexys A7-100T board, in txt format (for easier opening)

20. Modify the file to map the inputs of the full_adder to the switches, and the output to LED, by removing # at the beginning of an appropriate line, and changing parameters for get_ports. Save the file.

    Map switches:
    J15 – ci
    L16 – b
    M13 – a

    And map LEDs:
    U16 – s
    T15 – co

21. In Flow Navigator, under “Synthesis”, press “Run Synthesis”. Please note that all the steps from now till the end can take some time, especially if you are running the tool on SVL. The underlying algorithms are very complex.

Figure 12. a) Run Synthesis; b) The synthesis tool is running – top right corner of Vivado window

22. When the synthesis is completed successfully, the pop-up window (and the top right corner) will indicate so (Figure 13.). “Run Implementation” by pressing “OK” or by pressing “Run Implementation” under “Implementation” in “Flow Navigator”.

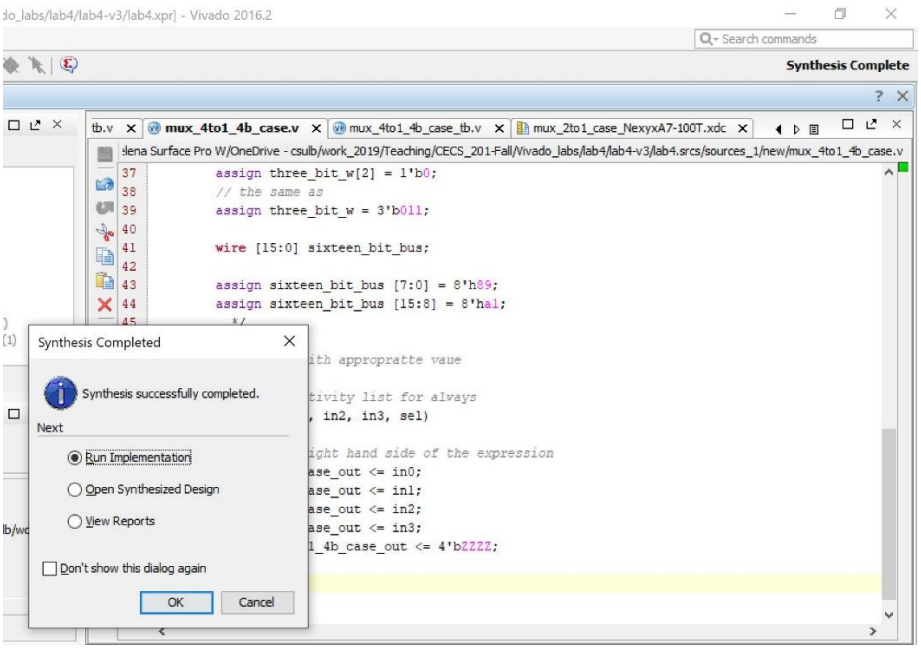

Figure 13. Successfully completed Synthesis and "Run Implementation" option

23. Upon successfully finished Implementation, pop-up window appears. Select "Generate Bitstream" by pressing "OK" or choosing "Generate Bitstream" under "Program and Debug" in "Flow Navigator".

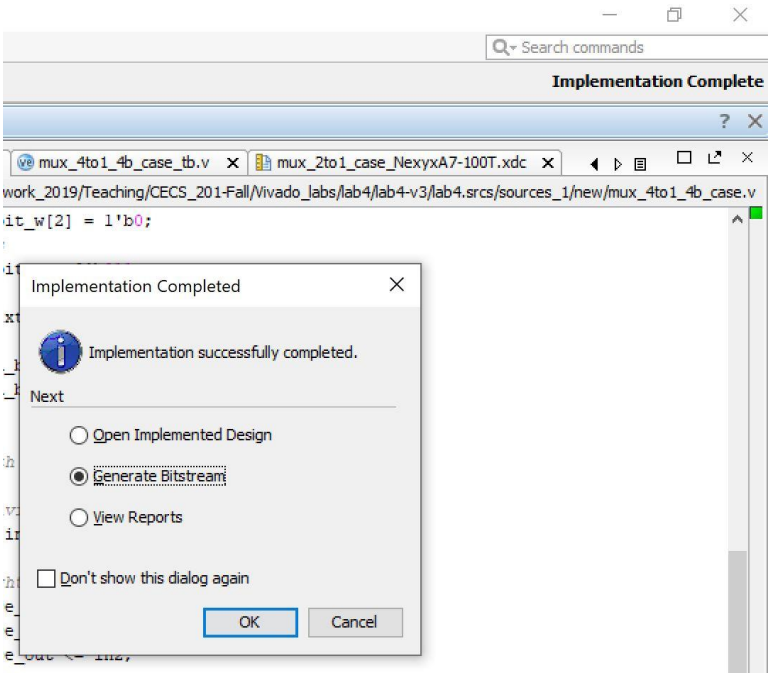

Figure 14. Implementation completed and the next step "Generate Bitstream"

24. Now it is time to connect and power on your Nexys A7-100T board. Handle the board gently, plug in the cable into USB port of your computer and in the board, place the

POWER switch in ON position. There will be several LEDs that turn on, and a pattern will be cycling on the seven segment displays.

25. Back to Vivado: “Open Hardware Manager” either by pressing “OK” (Figure 15.) on the pop-up window or by pressing “Open Hardware Manager” under “Program and Debug” in “Flow Navigator”.

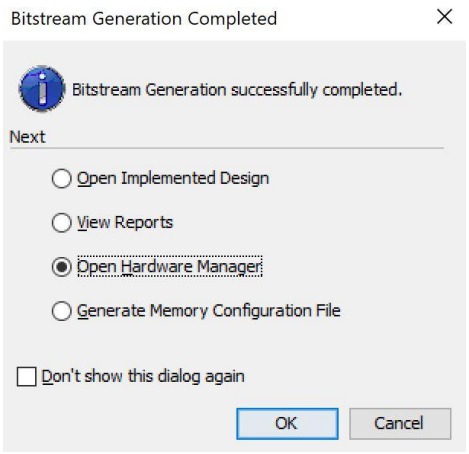

Figure 15. Bitstream generated successfully, Open Hardware Manager

26. Press “Open Target” and chose “Auto connect” option (Figure 17.a)). Once the tool recognized the board, choose the “Program device” option (Figure 17.b)). Press on the device name “xc7a100t_0” in the drop-down menu.

27. Press “Program” while leaving the default options as in Figure 16.

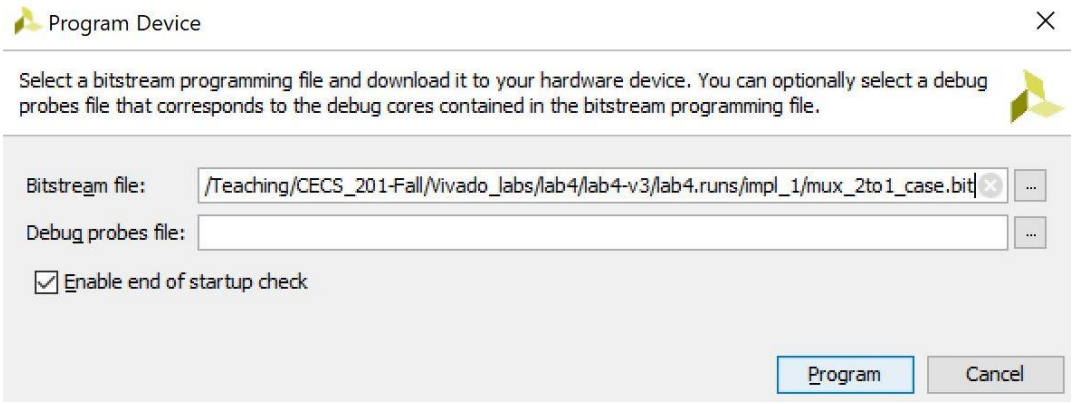

Figure 16. Programming the board

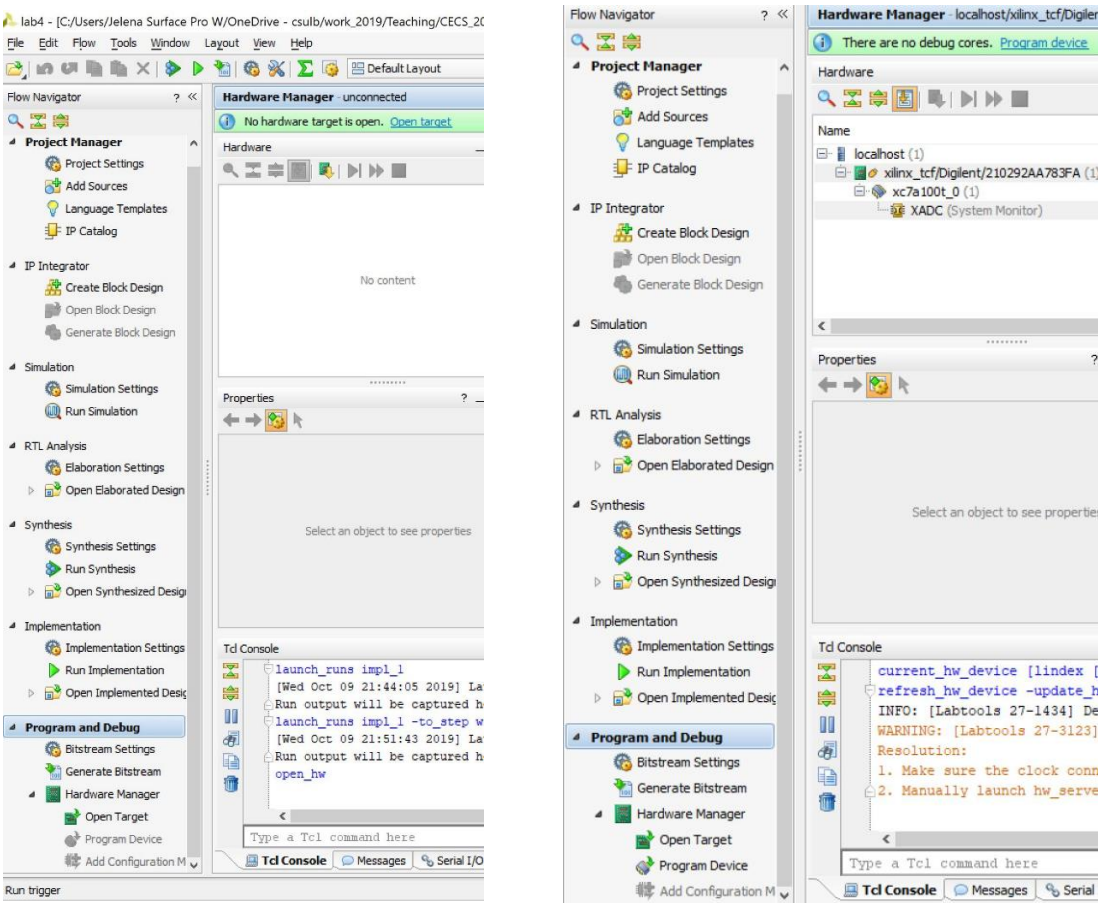

Figure 17. a) Connecting the board; b) Programming the board

28. The “light show” on the board will stop. Move the eight rightmost switches (see the .xdc file describing the connections) to test the design. Observe the LEDs “H17” – “U16”. Does your design work as per specification and simulation?

29. Close “Hardware Manager” by pressing X on the blue nav bar in top right (do not need to close the project or Vivado). Move “POWER” switch on the board to “OFF” position and unplug the board.

Extra Credit (20% more for this lab):

30. Synthesize 4-bit Adder such that operand A is mapped to 4 left most switches, operand B to 4 right most switches, and Cin to the switch in the middle. Sum should be mapped to middle LEDs, Co to the next LED right from the MSB of Sum, and V to the left most LED on the board.

What to submit:

Upload to Lab_6 Dropdox the following files:

a. full_adder.v , full_adder _tb.v,
b. four_bit_adder.v , four_bit_adder_tb.v,
c. add_sub.v, add_sub_tb.v, add_sub_NexysA7-100T.xdc, and
d. your lab report.

The ".v" files can be found in lab6-v0/lab6-v0.srcs/sim_1 and lab6-v0/lab6-v0.srcs/sources_1. Each folder contains a folder "new" and the sources are in that folder. ". xdc" files can be found in lab6-v0\lab6.srcs\constrs_1\new (Figure 20.).

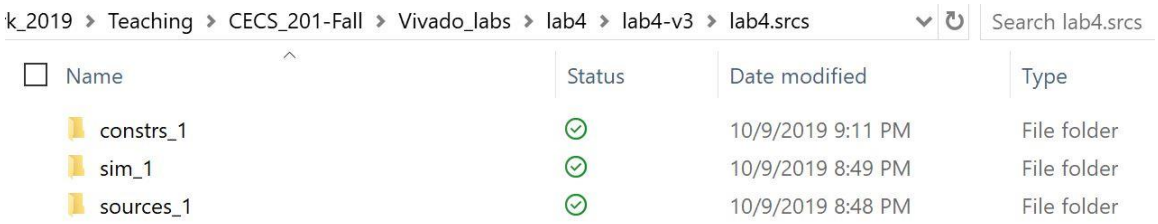

Figure 20. Location of sources and .xdc files for lab4 example

Questions:

The questions to be asked during the demo will be similar, but not limited to, the questions listed below:

1. Show the design being simulated: explain the waveform seen during the simulation.

2. Observe all testbenches: explain the inputs they generate?

3. Did you have any errors and how did you fix them?

4. Explain how the implemented designs work on FPGA board. Is this something you expected? Why or why not?

5. Explain **delay** in **assignment** statement on the example of encoder and decoder.