California State University Long Beach
Computer Engineering and Computer Science Department
CECS 201 – Computer Logic Design I
Lab 5 – Combinational Datapath Components: Encoder and Decoder

Objectives:

- To practice programming an FPGA board using Vivado SW and test the implemented design
- To practice **vectors**, **always** and **case** statement in Verilog
- To lean about **module** instantiation and make **hierarchical** design in Verilog
- To support learning about combinational datapath components: an Encoder and a Decoder

Theory:

This lab assignment will help you put in practice concepts covered in chapter "Combinational Datapath Components" particularly sub-section "Decoder": 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: Encoder and Decoder. You will need to create three modules and test each of them: one module and corresponding testbench for a 3-to-8 encoder, another module and the corresponding testbench for a 8-to-3 decoder, and, finally, a module and the corresponding testbench for a top level design that consists of an encoder that is connected to a decoder (enc_dec). Within the body of an encoder and a decoder module you will describe the circuit using a behavioral model (using its desired behavior/function). However, the combined module enc_dec will be described using structural Verilog, i.e. using components and connection that are "inside" the enc_dec module. This is basic for developing a hierarchical design. Finally, you will **implement** (only) an enc_dec 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 **lab5** in the **labs** folder for lab 5 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 3-to-8 encoder (Figure 1.) with module name, inputs and outputs as shown in the Figure 2. The function for the decoder is given in the Table 1. Use it to help you fill in the blanks in the module body. Pay attention to "item" in the case statement: in the previous lab, we expressed the values of "item" in decimal form, where here we are using binary form: value of **In** will be compared to "0000_0001", "0000_0010", etc. Please note that "_" is added just to improve readability and doe not change the value of a binary number.

Table 1. Truth table for 3-to-8 encoder

In[7]	**In[6]**	**In[5]**	**In[4]**	**In[3]**	**In[2]**	**In[1]**	**In[0]**	**Out[2]**	**Out[1]**	**Out[0]**	**Valid**
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	1	0	0	0	0	1	1	1
0	0	0	1	0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	1	0	1	1
0	1	0	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	1	1

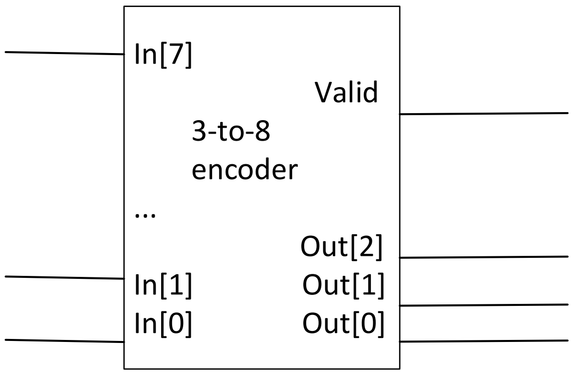
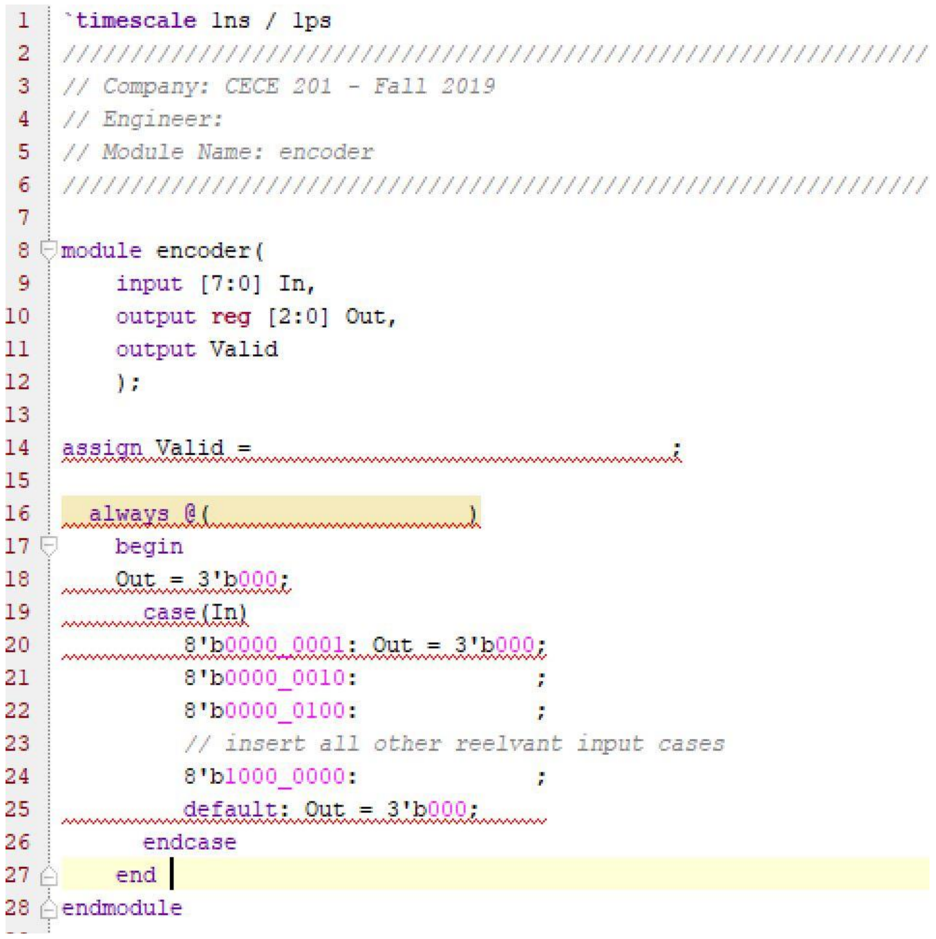


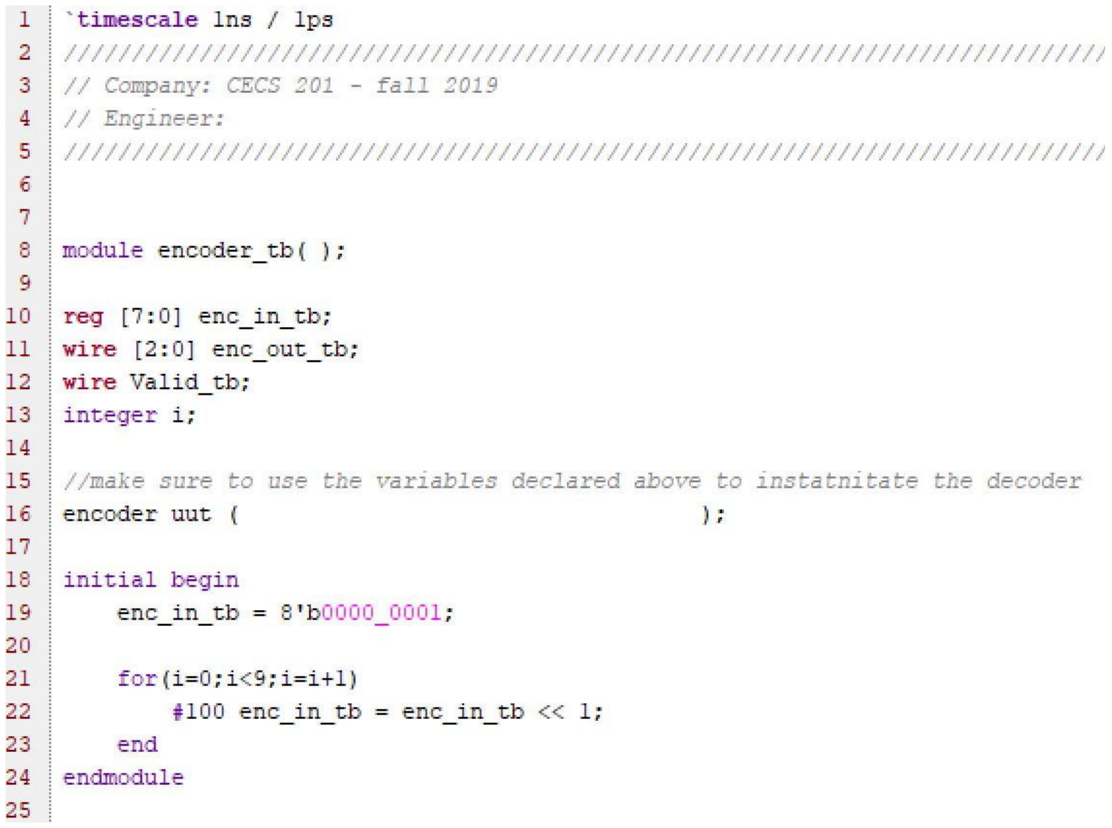
Figure 1. Block diagram of a 3-to-7 encoder



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////
// Company: CECE 201 - Fall 2019
// Engineer:
// Module Name: encoder
////////////////////////////////////////////////////////////////////

module encoder(
    input [7:0] In,
    output reg [2:0] Out,
    output Valid
    );

assign Valid =                                    ;

  always @(                            )
    begin
    Out = 3'b000;
      case(In)
         8'b0000_0001: Out = 3'b000;
         8'b0000_0010:                 ;
         8'b0000_0100:                 ;
         // insert all other reelvant input cases
         8'b1000_0000:                 ;
         default: Out = 3'b000;
      endcase
    end
endmodule
```

Figure 2. Module encoder, its inputs and outputs, and the module body. In the binary number "_" is added to increase readability.

6. Create a testbench for encoder as per Figure 3. Instantiate the encoder using the variables defined at the beginning of the testbench. Study the code inside "initial" statement: what are the values it produces? You'll need to figure out what does "<<" do.



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: CECS 201 - fall 2019
// Engineer:
//////////////////////////////////////////////////////////////////////////////////


module encoder_tb( );

reg [7:0] enc_in_tb;
wire [2:0] enc_out_tb;
wire Valid_tb;
integer i;

//make sure to use the variables declared above to instatnitate the decoder
encoder uut (                                  );

initial begin
    enc_in_tb = 8'b0000_0001;

    for(i=0;i<9;i=i+1)
        #100 enc_in_tb = enc_in_tb << 1;
    end
endmodule

```

Figure 3. Testbench encoder _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). Compare them to the values in Table 1. Include the screenshot of the output in the lab report.

8. Create a new module to implement a 3-to-8 decoder (Figure 4.) with module name, inputs and outputs as shown in the Figure 5. The function for the decoder is given in the Table 2. Use it to help you fill in the blanks in the module body.

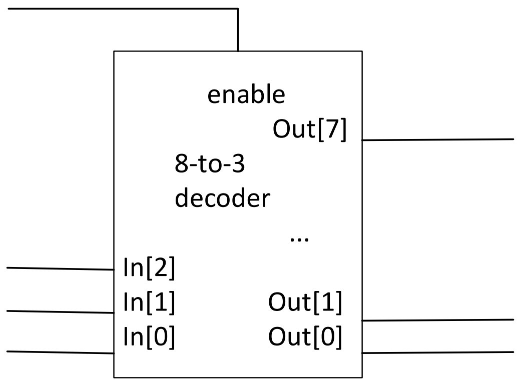
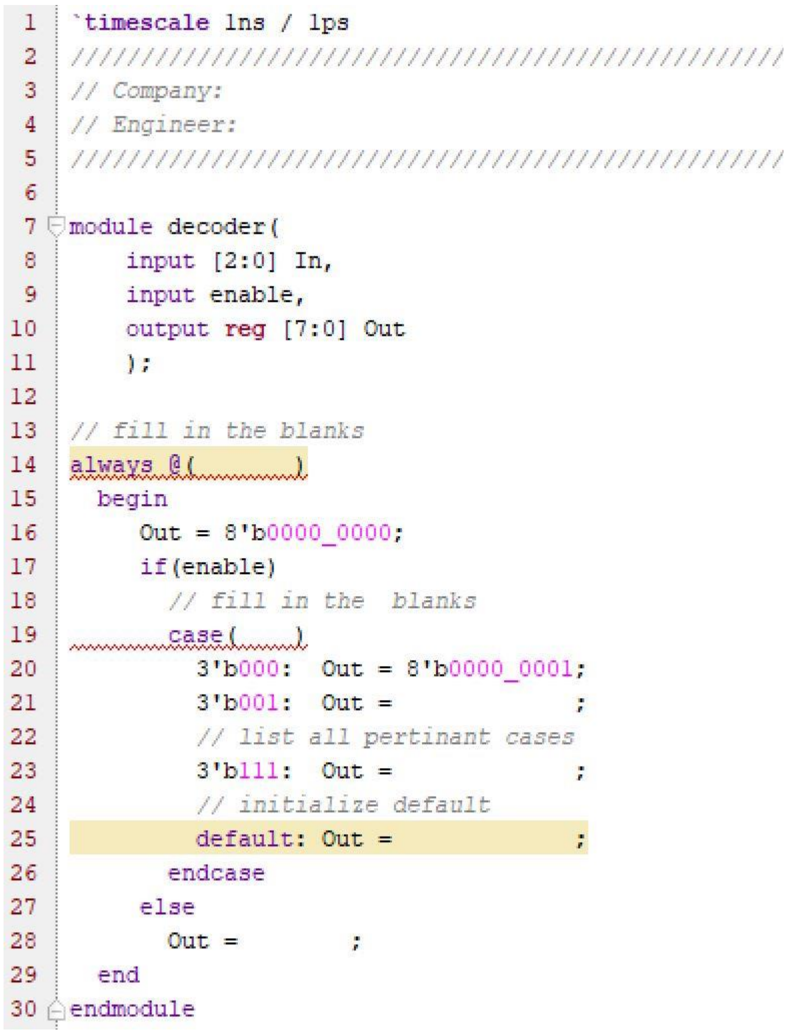


Figure 4. Block diagram of a 1-bit 4-to-1 mux

Table 2. Truth table for 8-to-3 decoder

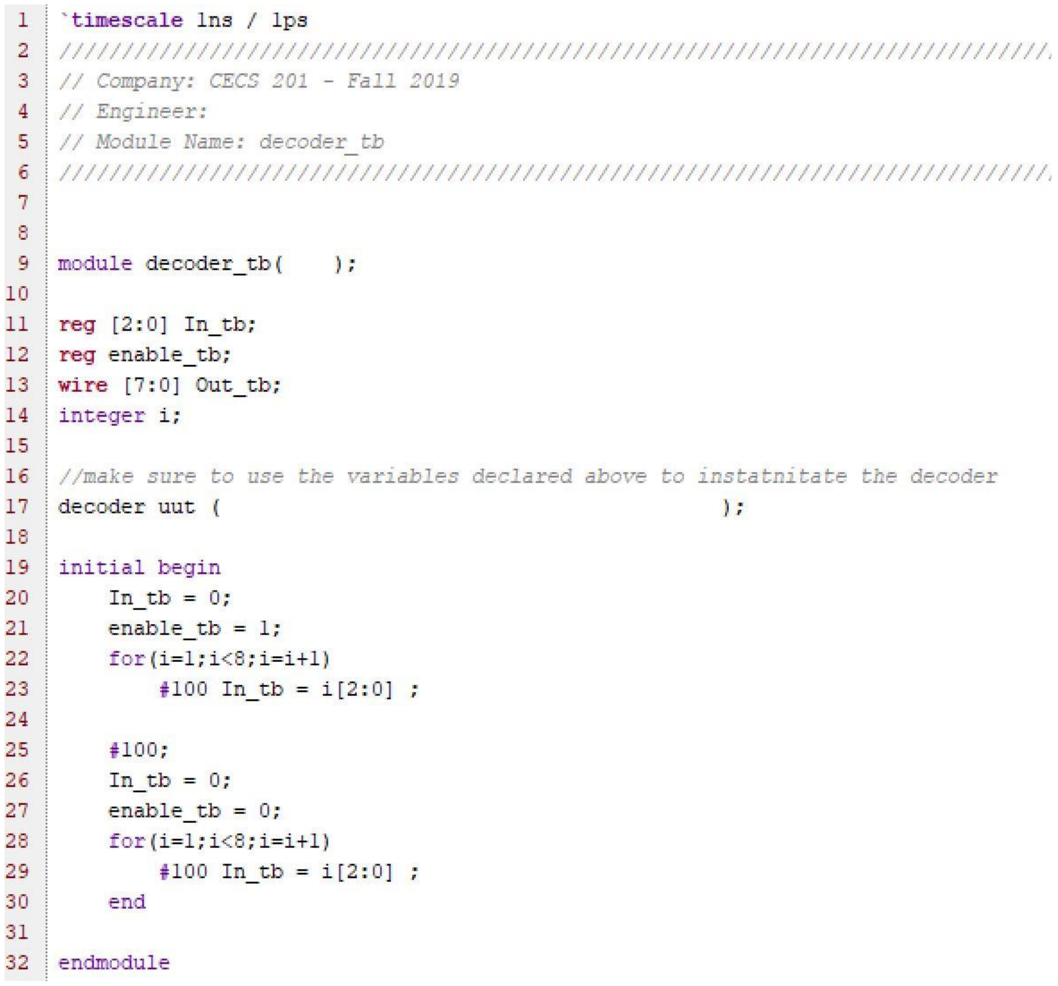
enable	In[2]	In[1]	In[0]	Out[7]	Out[6]	Out[5]	Out[4]	Out[3]	Out[2]	Out[1]	Out[0]
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////

module decoder(
    input [2:0] In,
    input enable,
    output reg [7:0] Out
    );

// fill in the blanks
always @(         )
  begin
     Out = 8'b0000_0000;
     if(enable)
       // fill in the  blanks
       case(     )
         3'b000:  Out = 8'b0000_0001;
         3'b001:  Out =            ;
         // list all pertinant cases
         3'b111:  Out =            ;
         // initialize default
         default: Out =            ;
       endcase
     else
       Out =        ;
  end
endmodule
```

Figure 5. Module decoder, its inputs and outputs, and the module body. In the binary number “_” is added to increase readability.

9. Create a testbench for decoder as per Figure 6. Instantiate the decoder using the variables defined at the beginning of the testbench. Study the code inside "initial" statement: what are the values it produces?



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: CECS 201 - Fall 2019
// Engineer:
// Module Name: decoder_tb
//////////////////////////////////////////////////////////////////////////////////


module decoder_tb(    );

reg [2:0] In_tb;
reg enable_tb;
wire [7:0] Out_tb;
integer i;

//make sure to use the variables declared above to instatnitate the decoder
decoder uut (                                        );

initial begin
    In_tb = 0;
    enable_tb = 1;
    for(i=1;i<8;i=i+1)
        #100 In_tb = i[2:0] ;

    #100;
    In_tb = 0;
    enable_tb = 0;
    for(i=1;i<8;i=i+1)
        #100 In_tb = i[2:0] ;
    end

endmodule
```

Figure 6. Testbench decoder _tb

10. "Set as top" files containing decoder and decoder_tb modules. Simulate the testbench and observe the outputs in the simulation window (run the simulation for additional time if needed to observe all test cases). Compare them to the values in Table 2. Include the screenshot of the output in the lab report.

11. Create a new module to implement an enc_ dec (Figure 7.) with module name, inputs and outputs as shown in the Figure 8. Fill in the blanks and to instantiate the encoder and a decoder and connect them to each other using the schematic given in the Figure 7.

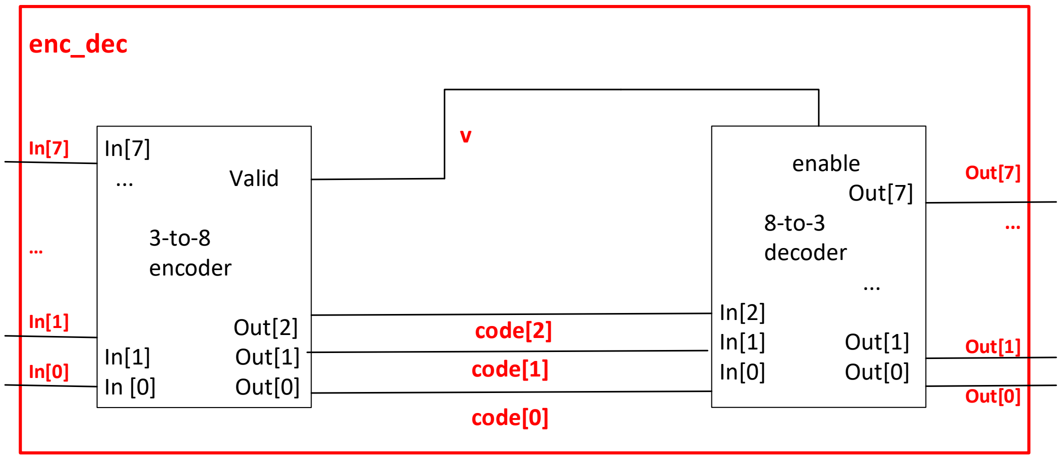
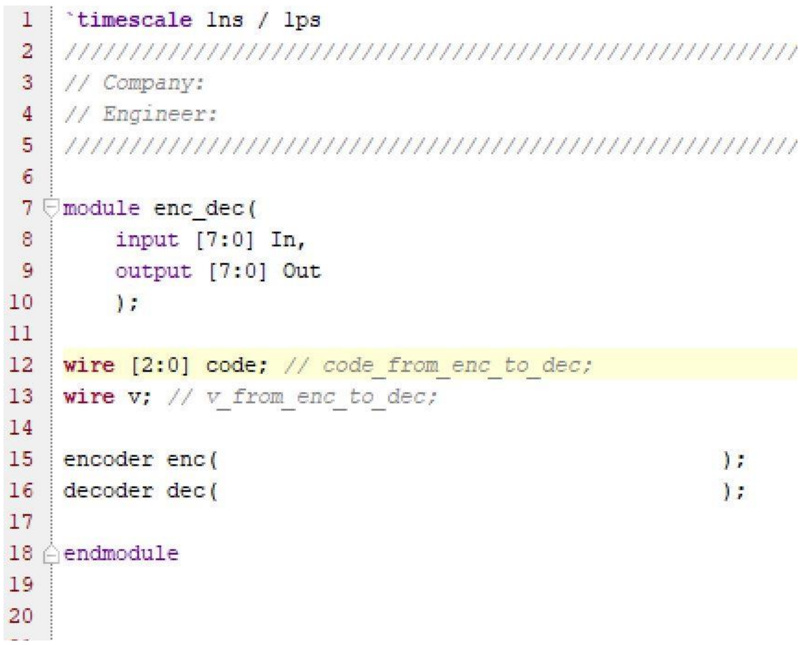


Figure 7. Block diagram of an enc_dec module



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
////////////////////////////////////////////////////////////////////

module enc_dec(
    input [7:0] In,
    output [7:0] Out
    );

wire [2:0] code; // code_from_enc_to_dec;
wire v; // v_from_enc_to_dec;

encoder enc(                                    );
decoder dec(                                    );

endmodule


```

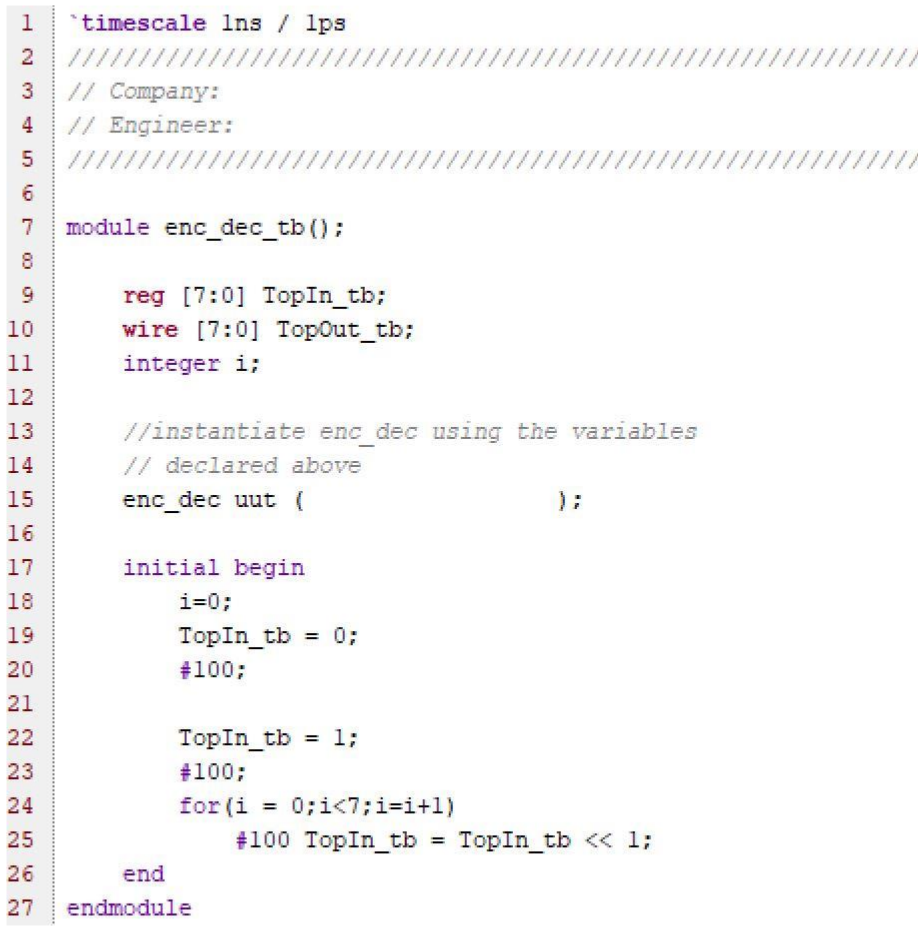
Figure 8. Module enc_dec, its inputs and outputs, and the module body.

12. Fill in the Table 3. and include it in the report. “code” and “v” are internal wires inside enc_dec module, used to connect the encoder with the decoder.

Table 2. Truth table for 8-to-3 decoder

In[7:0]	code[2:0]	v	Out[7:0]
0000_0000			▨
0000_0001			▨
0000_0010			▨
…			▨
1000_0000			▨
1100_0000			▨

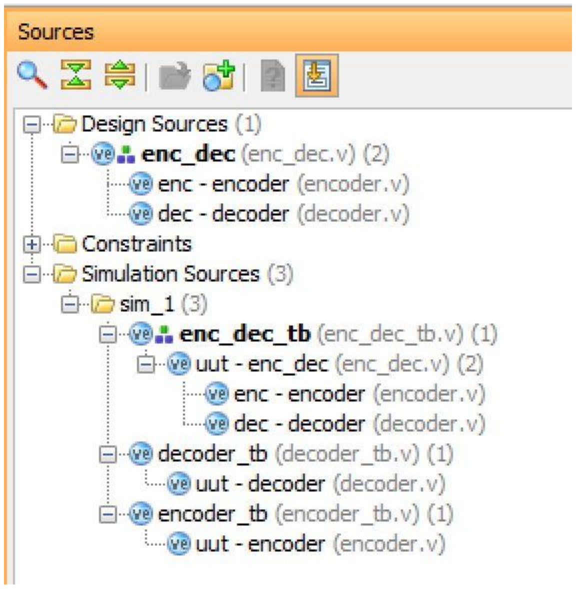
13. Create a testbench for enc_dec as per Figure 9. Instantiate the enc_dec using the variables defined at the beginning of the testbench. Study the code inside "initial" statement: what are the values it produces? Using Table 3. Determine the expected outputs of the testbench.



```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////////////////////

module enc_dec_tb();

    reg [7:0] TopIn_tb;
    wire [7:0] TopOut_tb;
    integer i;

    //instantiate enc_dec using the variables
    // declared above
    enc_dec uut (                    );

    initial begin
        i=0;
        TopIn_tb = 0;
        #100;

        TopIn_tb = 1;
        #100;
        for(i = 0;i<7;i=i+1)
            #100 TopIn_tb = TopIn_tb << 1;
    end
endmodule
```

Figure 9. Testbench for enc_dec

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



```
Sources
Design Sources (1)
    enc_dec (enc_dec.v) (2)
        enc - encoder (encoder.v)
        dec - decoder (decoder.v)
Constraints
Simulation Sources (3)
    sim_1 (3)
        enc_dec_tb (enc_dec_tb.v) (1)
            uut - enc_dec (enc_dec.v) (2)
                enc - encoder (encoder.v)
                dec - decoder (decoder.v)
        decoder_tb (decoder_tb.v) (1)
            uut - decoder (decoder.v)
        encoder_tb (encoder_tb.v) (1)
            uut - encoder (encoder.v)
```

Figure 10. Sources window show correct hierarchy

15. “Set as top” files containing enc_dec and enc_dec _tb. Simulate the testbench and observe the outputs in the simulation window (run the simulation for additional time if needed to observe all test cases). Compare them to the values you populated in Table 3. Include the screenshot(s) of the output in the lab report.

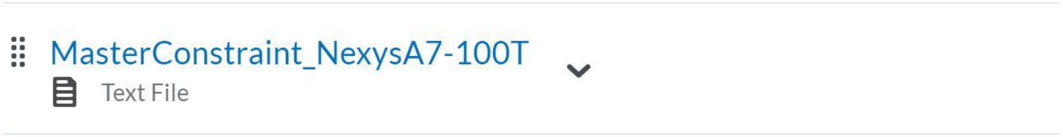
Synthesis:

16. Make sure enc_dec and enc_dec _tb are both “set as top”, and that enc_dec is functionally correct.

17. 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 “enc_dec_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.

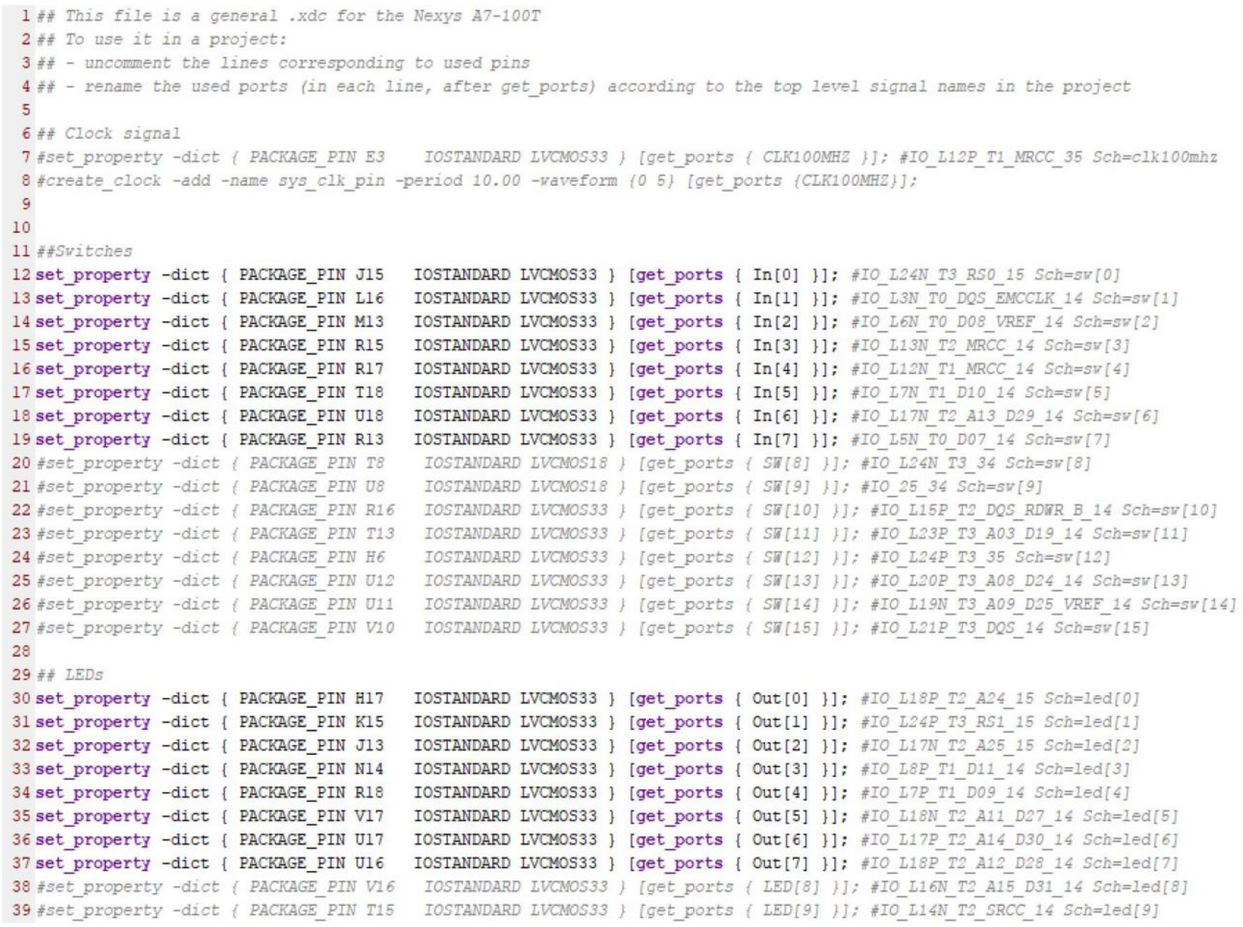
18. Download file “MasterConstraint_NexysA7-100T.txt” from Lab 4 folder on BB (Figure 11.) and save it locally. Open the file with the text editor (Notepad or WordPad), copy its content to clipboard, and paste the content inside the “enc_dec_NexysA7-100T.xdc” file in the editor.

19. Modify the file to map the inputs of the enc_dec to the switches, and the output to LED, as shown in the Figure 12. by removing # at the beginning of an appropriate line, and changing parameters for get_ports. Save the file.


MasterConstraint_NexysA7-100T

Text File

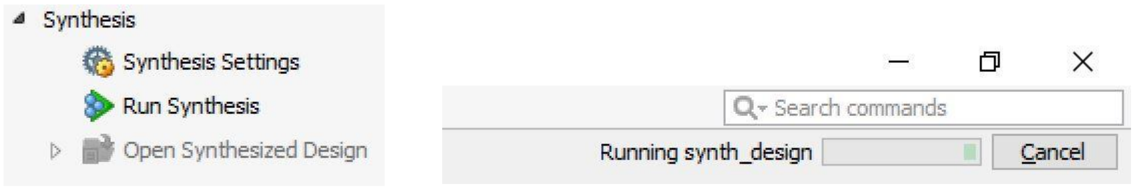
Figure 11. Master constraint file for Nexys A7-100T board, in txt format (for easier opening)



```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
#set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { In[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { In[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { In[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { In[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { In[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { In[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { In[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { In[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { Out[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { Out[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { Out[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { Out[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { Out[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { Out[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { Out[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { Out[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
```

Figure 12. Mapped switches and LEDs

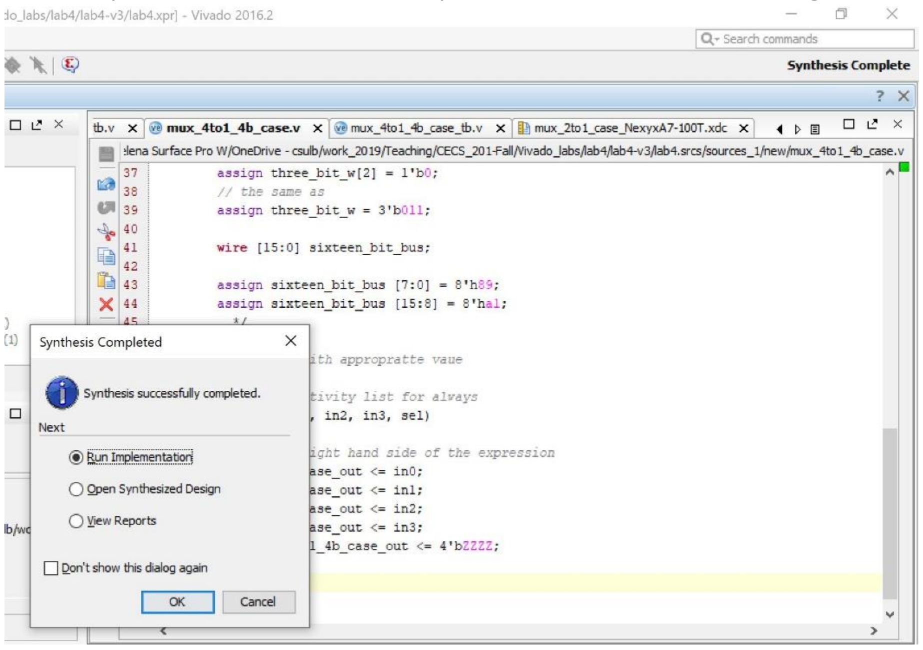
20. In Flow Navigator, under "Synthesis", press "Run Synthesis". Please note that all the steps from now till the end can take some time, especially if you are running the tool on SVL. The underlying algorithms are very complex.



Running synth_design

Figure 13. a) Run Synthesis; b) The synthesis tool is running – top right corner of Vivado window

21. When the synthesis is completed successfully, the pop-up window (and the top right corner) will indicate so (Figure 14.). “Run Implementation” by pressing “OK” or by pressing “Run Implementation” under “Implementation” in “Flow Navigator”.



do_labs/lab4/lab4-v3/lab4.xpr] - Vivado 2016.2

Synthesis Complete

tb.v | mux_4to1_4b_case.v | mux_4to1_4b_case_tb.v | mux_2to1_case_NexyxA7-100T.xdc

elena Surface Pro W/OneDrive - csulb/work_2019/Teaching/CECS_201-Fall/Vivado_labs/lab4/lab4-v3/lab4.srcs/sources_1/new/mux_4to1_4b_case.v

```
        assign three_bit_w[2] = 1'b0;
        // the same as
        assign three_bit_w = 3'b011;

        wire [15:0] sixteen_bit_bus;

        assign sixteen_bit_bus [7:0] = 8'h89;
        assign sixteen_bit_bus [15:8] = 8'ha1;
          */

ith appropratte vaue

tivity list for always
, in2, in3, sel)

ight hand side of the expression
ase_out <= in0;
ase_out <= in1;
ase_out <= in2;
ase_out <= in3;
1_4b_case_out <= 4'bZZZZ;
```

Synthesis Completed

Synthesis successfully completed.

Next

[✓ Run Implementation]

Open Synthesized Design

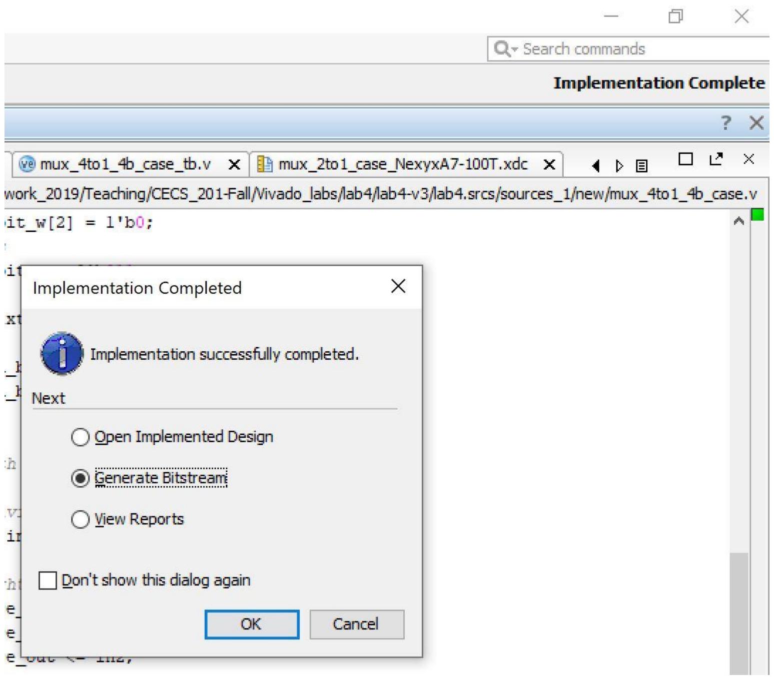
View Reports

Don't show this dialog again

OK Cancel

Figure 14. Successfully completed Synthesis and “Run Implementation” option

22. 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”.



Implementation Complete

mux_4to1_4b_case_tb.v | mux_2to1_case_NexyxA7-100T.xdc

work_2019/Teaching/CECS_201-Fall/Vivado_labs/lab4/lab4-v3/lab4.srcs/sources_1/new/mux_4to1_4b_case.v

```
it_w[2] = 1'b0;
```

Implementation Completed

Implementation successfully completed.

Next

Open Implemented Design

[✓ Generate Bitstream]

View Reports

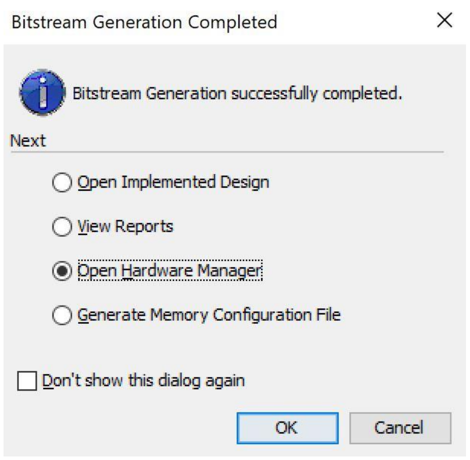
Don't show this dialog again

OK Cancel

Figure 15. Implementation completed and the next step “Generate Bitstream”

23. 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.

24. Back to Vivado: "Open Hardware Manager" either by pressing "OK" (Figure 16.) on the pop-up window or by pressing "Open Hardware Manager" under "Program and Debug" in "Flow Navigator".


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

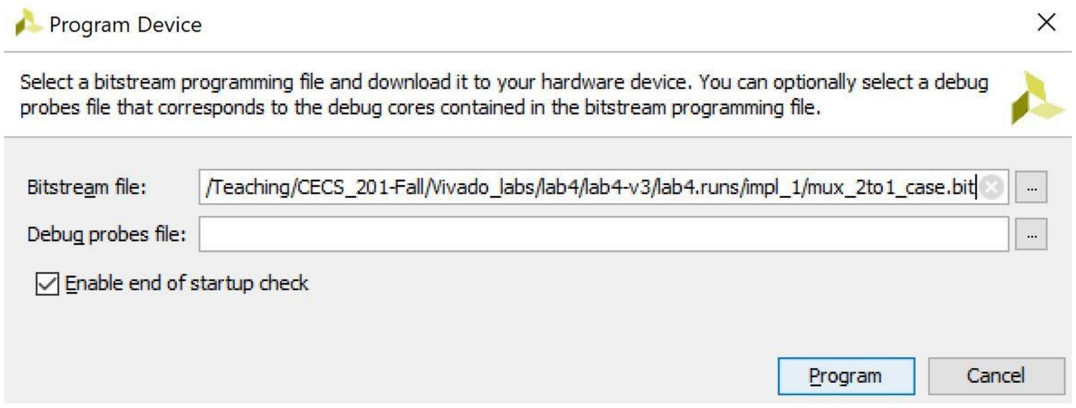
- Open Implemented Design
- View Reports
- [✓ Open Hardware Manager]
- Generate Memory Configuration File

☐ Don't show this dialog again

Figure 16. Bitstream generated successfully, Open Hardware Manager

25. Press "Open Target" and chose "Auto connect" option (Figure 18.a)). Once the tool recognized the board, choose the "Program device" option (Figure 18.b)). Press on the device name "xc7a100t_0" in the drop-down menu.

26. Press "Program" while leaving the default options as in Figure 17.


Program Device

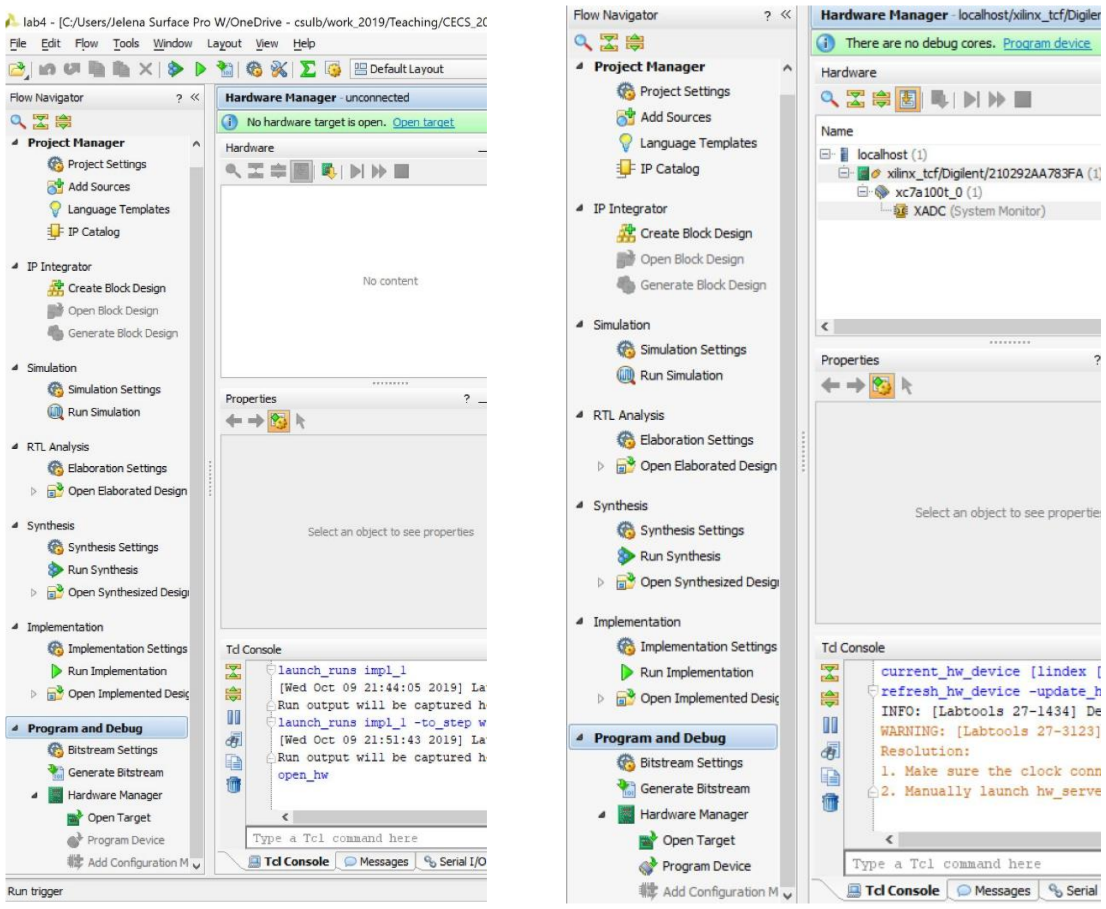
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: /Teaching/CECS_201-Fall/Vivado_labs/lab4/lab4-v3/lab4.runs/impl_1/mux_2to1_case.bit

Debug probes file:

☑ Enable end of startup check

Figure 17. Programming the board



```
launch_runs impl_1
[Wed Oct 09 21:44:05 2019] La
Run output will be captured h
launch_runs impl_1 -to_step w
[Wed Oct 09 21:51:43 2019] La
Run output will be captured h
open_hw
```

```
current_hw_device [lindex [
refresh_hw_device -update_h
INFO: [Labtools 27-1434] De
WARNING: [Labtools 27-3123]
Resolution:
1. Make sure the clock conn
2. Manually launch hw_serve
```

Figure 18. a) Connecting the board; b) Programming the board

27. 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?

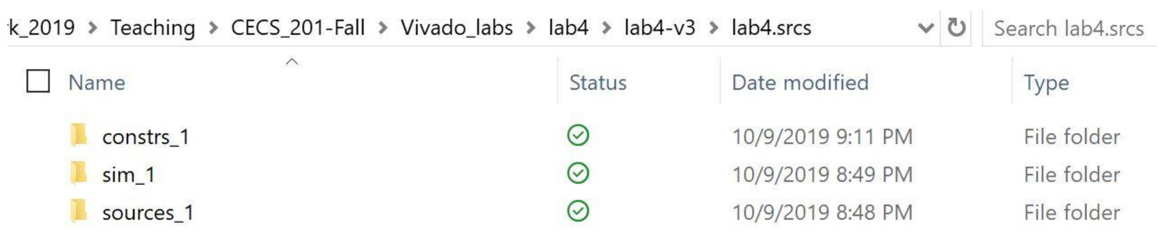
28. Close “Hardware Manager” by pressing X on the blue nav bar in top right (do not need to close the project or Vivado). Move “POWER” switch on the board to “OFF” position and unplug the board.

What to submit:

Upload to Lab_4 Dropdox the following files:

a. encoder.v , encoder_tb.v,
b. decoder.v , decoder_tb.v,
c. enc_dec.v, enc_dec_NexysA7-100T.xdc, and
d. your lab report.

The ".v" files can be found in lab5-v0/lab5-v0.srcs/sim_1 and lab5-v0/lab5-v0.srcs/sources_1. Each folder contains a folder "new" and the sources are in that folder. ". xdc" files can be found in lab5-v0\lab5.srcs\constrs_1\new (Figure 20.).


k_2019 > Teaching > CECS_201-Fall > Vivado_labs > lab4 > lab4-v3 > lab4.srcs

Search lab4.srcs

Name	Status	Date modified	Type
constrs_1	✓	10/9/2019 9:11 PM	File folder
sim_1	✓	10/9/2019 8:49 PM	File folder
sources_1	✓	10/9/2019 8:48 PM	File folder

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. If there is no "default" statement in encoder module, is there difference in its function when tested using encoder_tb? How about if in the enc_dec module when running enc_dec_tb or on the FPGA board?

4. If there is no "default" statement in decoder module, is there difference in its function when tested using decoder_tb? How about if in the enc_dec module when running enc_dec_tb or on the FPGA board?

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

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

6. Explain **always** block and **case** statement on the example of encoder and decoder.

7. Explain **if** statement on the example of decoder.