EXPERIMENT 8

Encoders & Application to a 7-Segment Display Driver

Department of Electrical & Computer Engineering

I. OBJECTIVES:

- Examine encoders and their applications.
- Examine the characteristics of a decimal-to-BCD priority encoder.
- Design an 8-to-3 line priority encoder in schematic mode and test it on a target board.
- Create a macro for a 7-segment decoder with active-HIGH outputs using VHDL.
- Utilizing a created macro, design a circuit in schematic mode to display a decimal digit on a 7-segment display.

II. MATERIALS:

- Xilinx Vivado software, student or professional edition V2018.2 or higher.
- IBM or compatible computer with Pentium III or higher, 128 M-byte RAM or more, and 8 G-byte Or larger hard drive.
- BASYS 3 Board.

III. DISCUSSION:

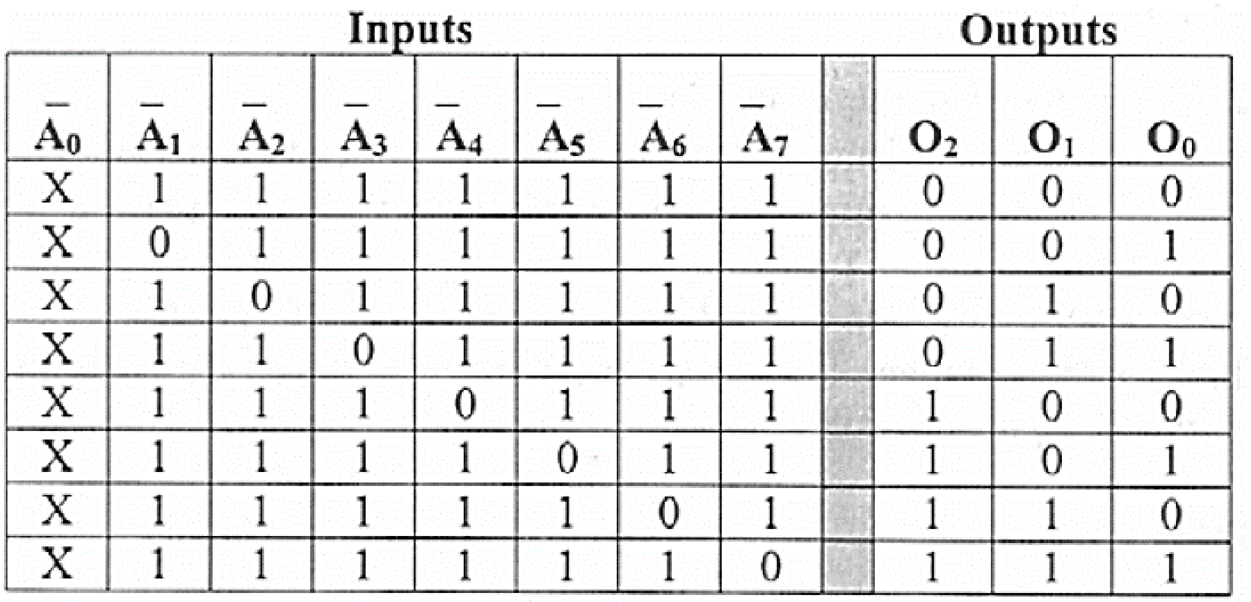
As we saw in the previous experiment, a decoder identifies or detects a particular binary number or code. Encoding is the opposite process of decoding. An encoder has a number of input lines (up to 2n), only one of which is activated at any given time, and produces the n-bit output code for the input selected. For example, consider an 8-to-3 encoder. When one of its eight inputs is activated, the output will be a 3-bit binary number (code) corresponding to that input.

If two or more inputs are activated at the same time, which one of the inputs should be encoded and reflected on the outputs? This is when priority is used in the encoder design. When multiple inputs are activated, priority specifies which input will get selected to produce the output code.

A common application of encoders is in the keyboards of calculators and computer systems to convert key-presses to binary numbers or to codes such as BCD or ASCII.

The 8-to-3 Line Encoder with Active-LOW Inputs

The 8-to-3 (octal-to-binary) encoder accepts eight input lines and produces a unique 3-bit output code for each set of inputs. Table 9.1 below describes the function of this encoder. Note the active-low inputs, as could be obtained from a keypad with normally-open contacts to ground.



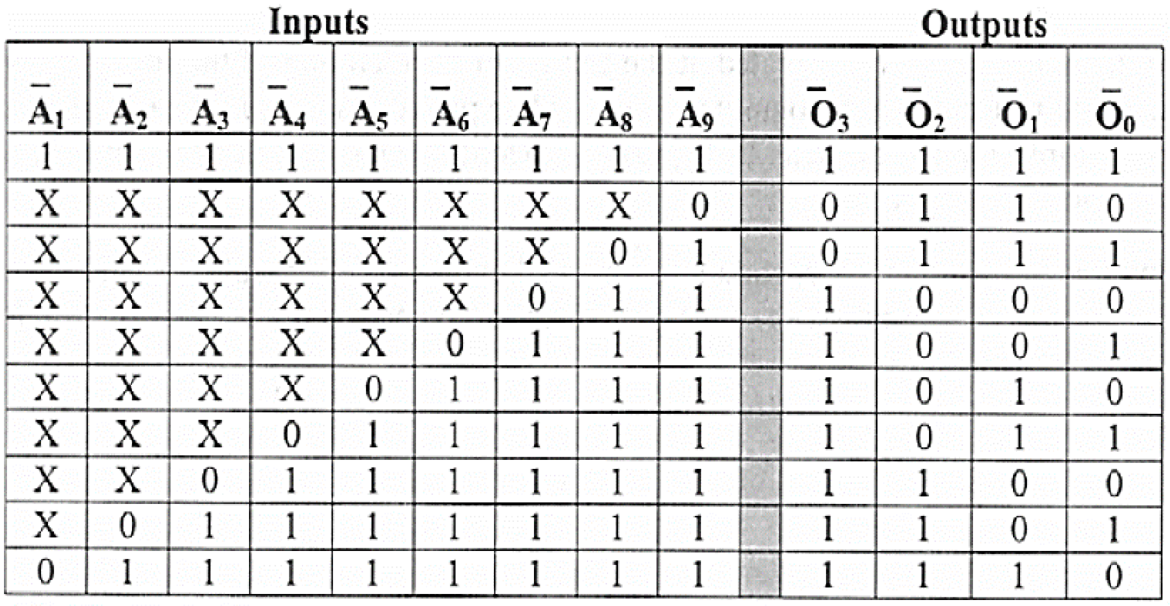
Inputs									Outputs		
$\overline{A_0}$	$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{A_5}$	$\overline{A_6}$	$\overline{A_7}$		O_2	O_1	O_0
X	1	1	1	1	1	1	1		0	0	0
X	0	1	1	1	1	1	1		0	0	1
X	1	0	1	1	1	1	1		0	1	0
X	1	1	0	1	1	1	1		0	1	1
X	1	1	1	0	1	1	1		1	0	0
X	1	1	1	1	0	1	1		1	0	1
X	1	1	1	1	1	0	1		1	1	0
X	1	1	1	1	1	1	0		1	1	1

Table 9.1 Truth Table for the Octal-to-Binary Encoder

The Decimal-to-BCD Priority Encoder

The SN54/74LS147 and the SN54/74LS148 are Priority Encoders. They provide priority decoding of the inputs to ensure that only the highest order data line is encoded. Both devices have data inputs and outputs which are active at the low logic level.

The LS147 encodes nine data lines to four-line (8-4-2-1) BCD. The implied decimal zero condition does not require an input condition because zero is encoded when all nine data lines are at a high logic level. Figure 9.2 is the truth-table for a decimal-to-BCD priority-encoder (such as the 74147 TTL chip). It has nine active-low inputs representing decimal numbers 1 through 9. The encoder produces the inverted BCD code corresponding to which of the nine inputs is activated.



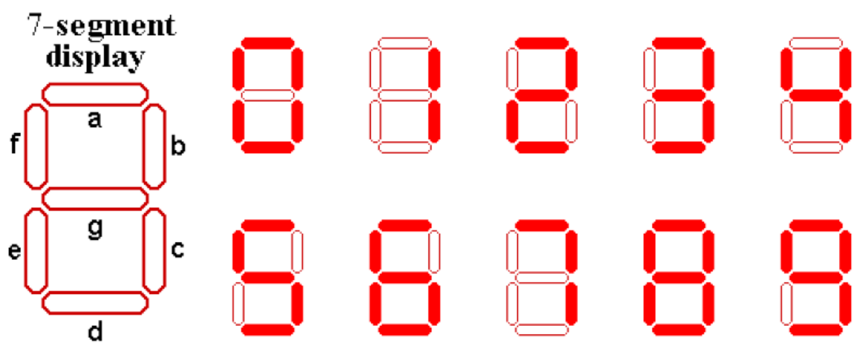
Inputs										Outputs			
$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{A_5}$	$\overline{A_6}$	$\overline{A_7}$	$\overline{A_8}$	$\overline{A_9}$		$\overline{O_3}$	$\overline{O_2}$	$\overline{O_1}$	$\overline{O_0}$
1	1	1	1	1	1	1	1	1		1	1	1	1
X	X	X	X	X	X	X	X	0		0	1	1	0
X	X	X	X	X	X	X	0	1		0	1	1	1
X	X	X	X	X	X	0	1	1		1	0	0	0
X	X	X	X	X	0	1	1	1		1	0	0	1
X	X	X	X	0	1	1	1	1		1	0	1	0
X	X	X	0	1	1	1	1	1		1	0	1	1
X	X	0	1	1	1	1	1	1		1	1	0	0
X	0	1	1	1	1	1	1	1		1	1	0	1
0	1	1	1	1	1	1	1	1		1	1	1	0

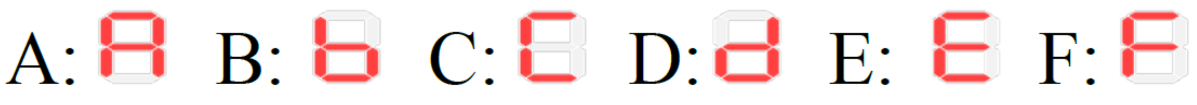
Table 9.2 Truth Table for a Decimal-to-BCD Priority Encoder (74147)

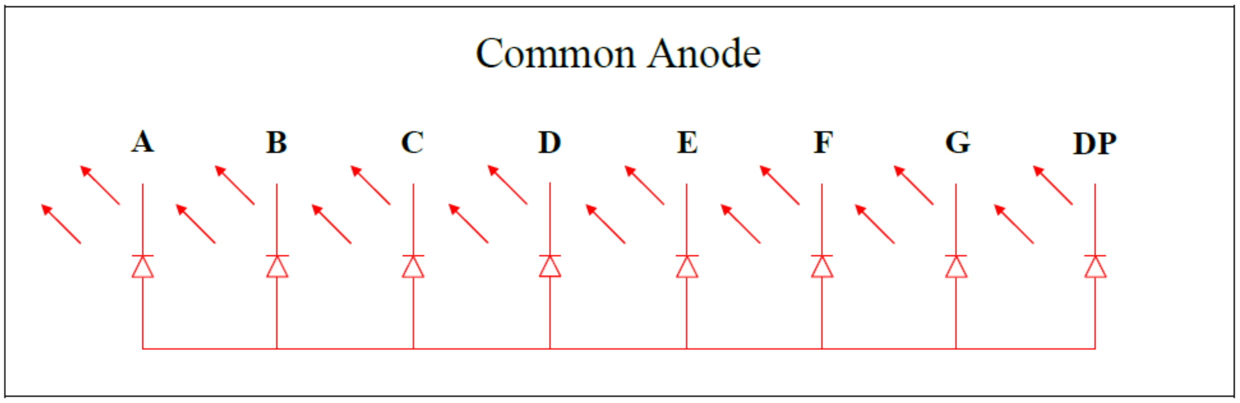
Note the “don’t-cares” (Xs) in the truth table. They imply that, if two inputs are activated simultaneously, only the highest data line is encoded. For example, if lines A1 and A5 are activated at the same time, A5 will be encoded producing the output 1010 (which is 0101 inverted, or BCD 5). That’s why it’s called a “priority” encoder. Moreover, the implied decimal zero condition requires no inputs since zero is encoded when all nine data lines are at HIGH. For a more detailed discussion of encoder circuits, refer to your digital textbook.

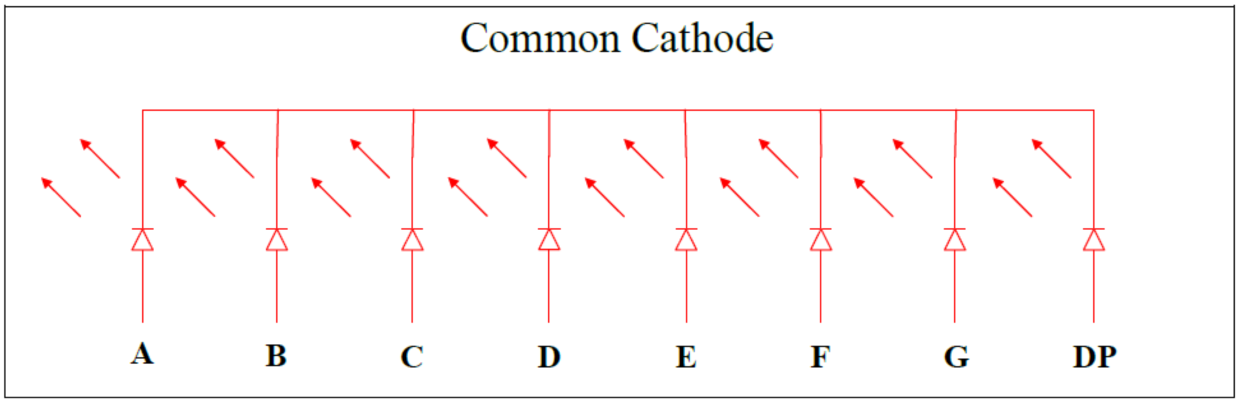
7-Segment Display

A 7-segment display is composed of seven bars (the segments; either LED or LCD) that can be individually activated to emit light. Such a display can show digits from 0 to 9, as well as a few letters (A, b, C, d, E, F, H, L, P, S, U, Y), a minus sign (-) and a decimal point. For a common-cathode 7-segment LED display, the “common” input is connected to GND, and a HIGH on any segment-input will light up that segment. For a common- anode display, the common-input is connected to a HIGH, and a LOW on a segment- input lights the segment. Figure 9.1 (a) and (b) show how the seven segments are arranged.









The Xilinx target board is using Common Cathode 7-segment display chips.

7-Segment Decoder

7-segment decoder is not available in the symbol library of the Xilinx software, so we will design one. We will design the decoder with active-high outputs for a common- cathode display. The lit display segments for each digit from 0 to 9 are given in the following truth table:

BCD to 7-Segment Decoder Driver (74LS47) Table

Decimal Number	Inputs				Outputs						
	D_3	**D_2**	**D_1**	**D_0**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0											
1											
2											
3											
4											
5											
6											
7											
8											
9											
Invalid											
Invalid											
Invalid											
Invalid											
Invalid											
Invalid											

Table 9.3 Truth Table for a 7-Segment Decoder with Active-High Outputs

Note that the outputs of this decoder are all LOW for invalid (mv) BCD codes. Based on the above truth table, we can derive the Boolean equations for the outputs using K-maps or Boolean algebra:

$$a = \overline{D}_3D_1 + \overline{D}_2\overline{D}_1\overline{D}_0 + D_3\overline{D}_2\overline{D}_1 + \overline{D}_3D_2D_0$$

$$b = \overline{D}_3\overline{D}_2 + \overline{D}_2\overline{D}_1 + \overline{D}_3\overline{D}_1\overline{D}_0 + \overline{D}_3D_1D_0$$

$$c = \overline{D}_3D_2 + \overline{D}_2\overline{D}_1 + \overline{D}_3D_0$$

$$d = \overline{D}_2\overline{D}_1\overline{D}_0 + \overline{D}_3\overline{D}_2D_1 + \overline{D}_3D_1\overline{D}_0 + \overline{D}_3D_2\overline{D}_1D_0$$

$$e = \overline{D}_2\overline{D}_1\overline{D}_0 + \overline{D}_3D_1\overline{D}_0$$

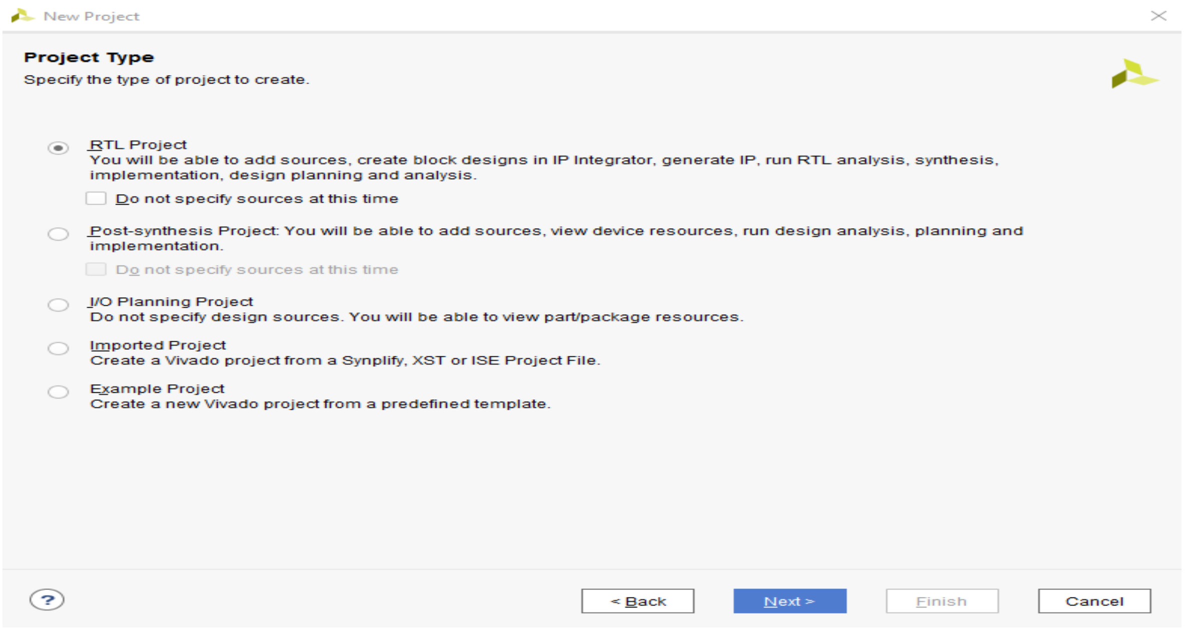
$$f = \overline{D}_2\overline{D}_1\overline{D}_0 + \overline{D}_3D_2\overline{D}_1 + \overline{D}_3D_2\overline{D}_0 + D_3\overline{D}_2\overline{D}_1$$

$$g = \overline{D}_3\overline{D}_2D_1 + \overline{D}_3D_2\overline{D}_1 + D_3\overline{D}_2\overline{D}_1 + \overline{D}_3D_1\overline{D}_0$$

IV. PROCEDURE:

Section 1:

1. Open Xilinix Vivado and in the **Xilinx-Project Navigator** window, Quick start, **New Project**.
2. Choose "RTL Project" and check the "Do not specify sources at this time" as we will configure all the settings manually through the navigator from inside the project.



Project Type

Specify the type of project to create.

[● RTL Project] You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ Do not specify sources at this time

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

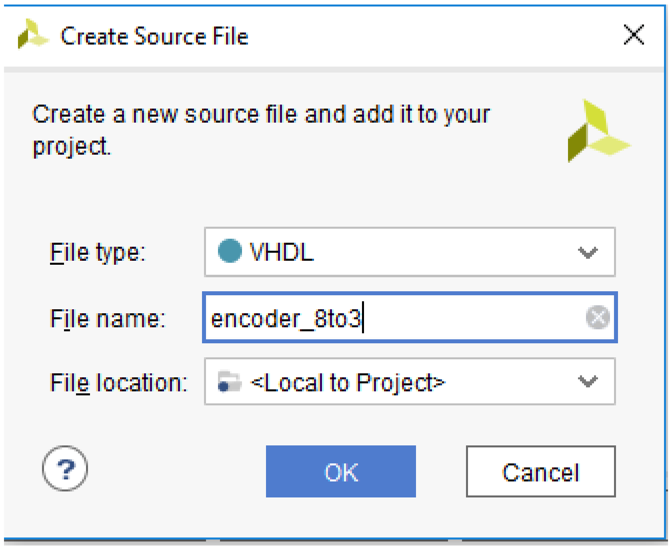
☐ Do not specify sources at this time

○ I/O Planning Project: Do not specify design sources. You will be able to view part/package resources.

○ Imported Project: Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project: Create a new Vivado project from a predefined template.

3. Select **New Source…** and the **New** window appears. In the New window, choose Schematic, type your file name (such as ***encoder_8to3***) in the File Name editor box, click on OK, and then click on the Next button.


Create Source File

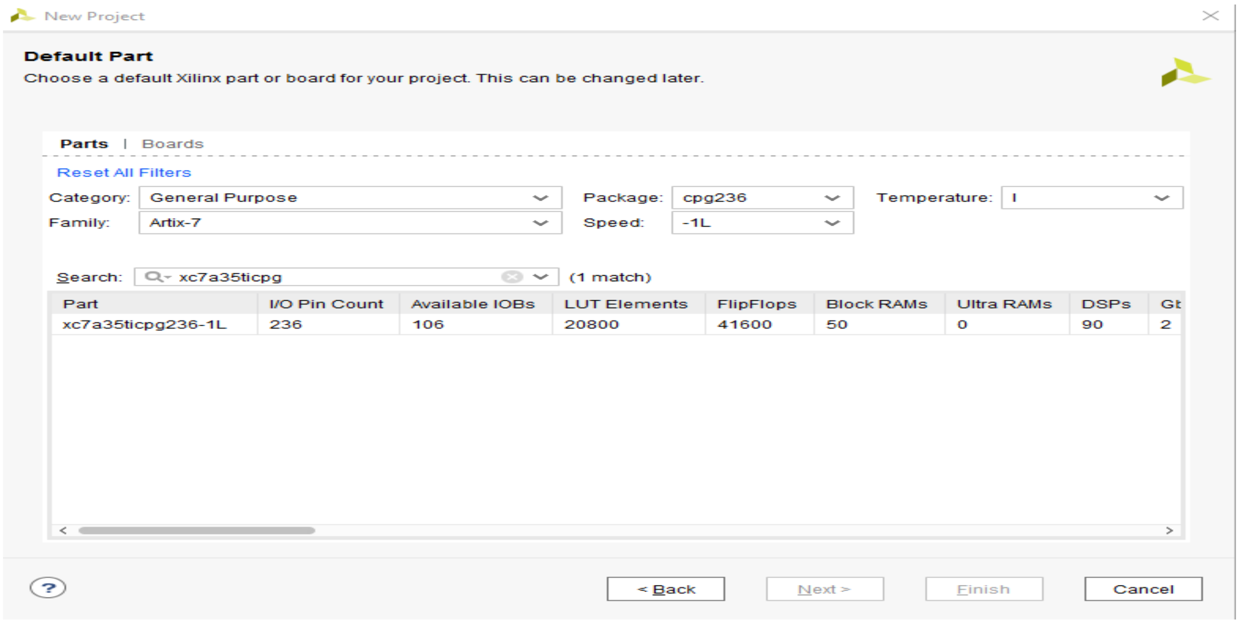
Create a new source file and add it to your project.

File type: VHDL

File name: encoder_8to3

File location: <Local to Project>

4. In the **Xilinx - Project Navigator** window, select the following
   - Category: “General Purpose”
   - Family: “Artix-7”
   - Package: “cpg236”
   - Speed: “-1”
   - Choose “xc7a35tcpg236-1” that corresponds to the board we are using.
   - Then Choose Finish.


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Category: General Purpose

Family: Artix-7

Package: cpg236

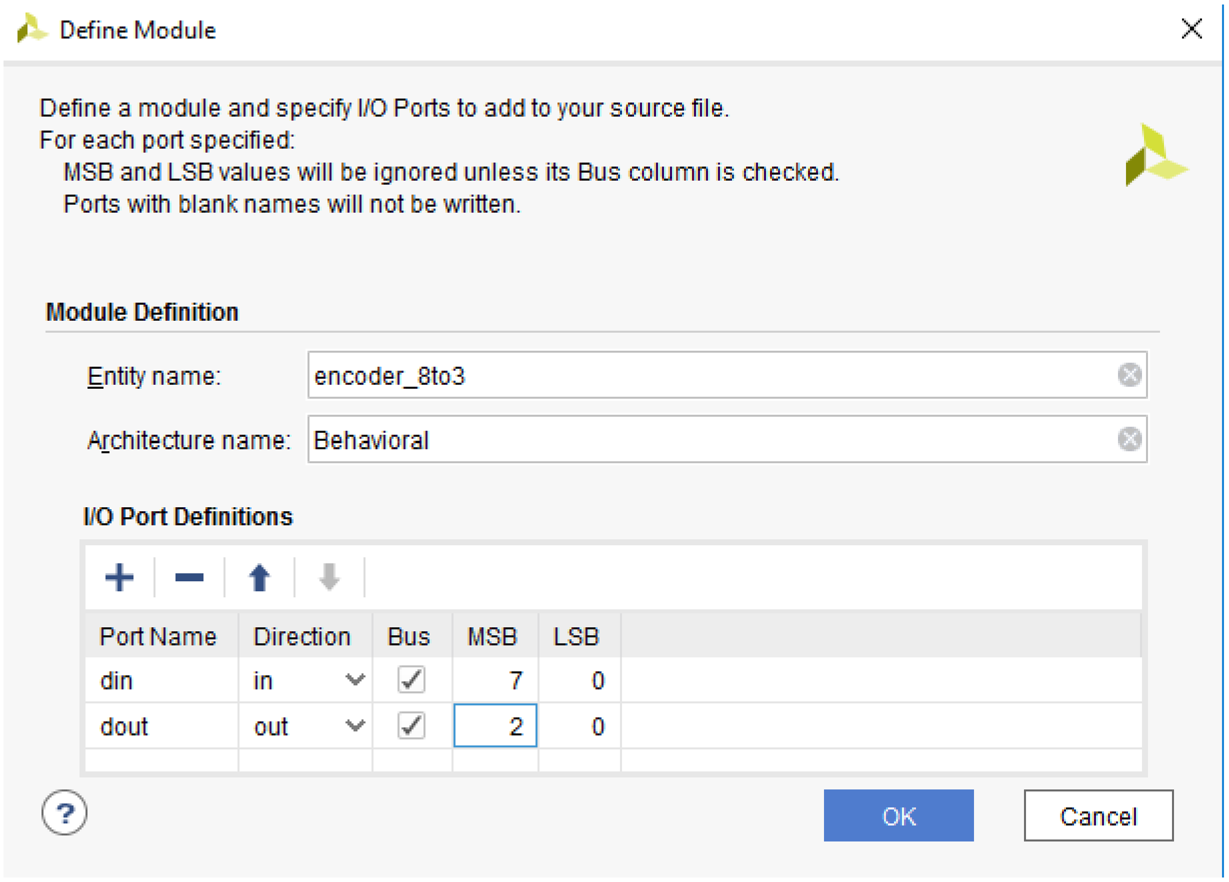
Speed: -1L

Temperature: I

Search: xc7a35ticpg (1 match)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7a35ticpg236-1L	236	106	20800	41600	50	0	90	2

5. The Define Module Window that will appear, we will choose the input and output labels for the gates under investigation in this experiment.



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

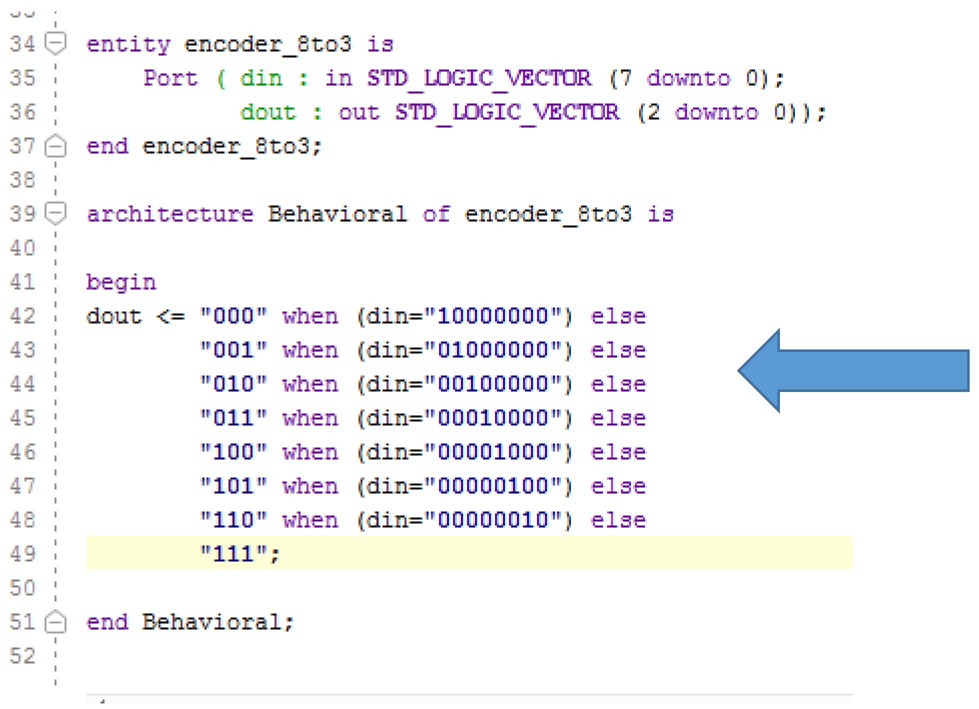
Entity name: encoder_8to3

Architecture name: Behavioral

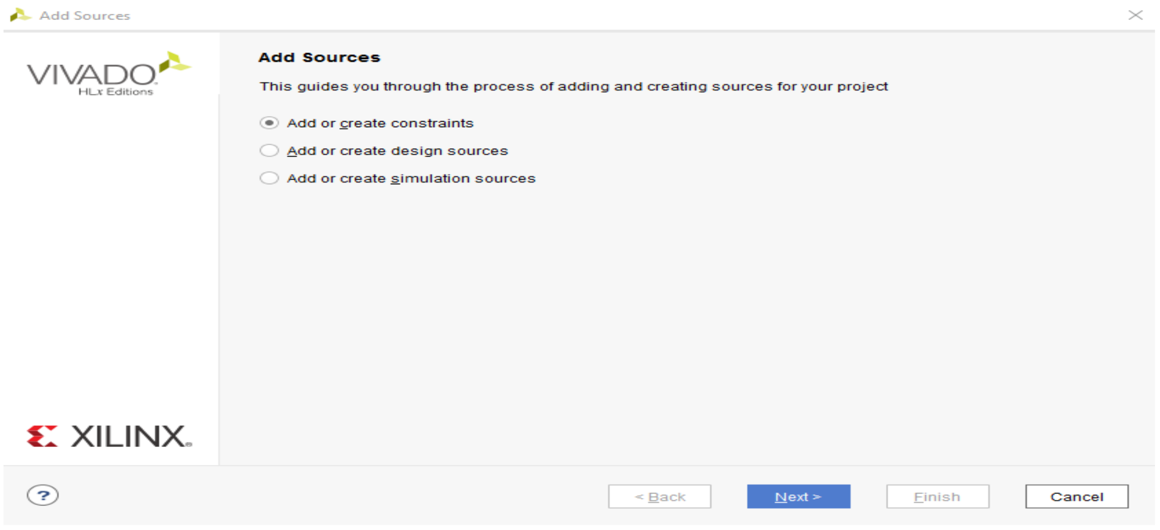
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
din	in	✓	7	0
dout	out	✓	2	0

6. In the “***encoder_8to3***.vhd” created file, type the gates equivalent VHDL code as follows and then save the file.


```
entity encoder_8to3 is
    Port ( din : in STD_LOGIC_VECTOR (7 downto 0);
           dout : out STD_LOGIC_VECTOR (2 downto 0));
end encoder_8to3;

architecture Behavioral of encoder_8to3 is

begin
dout <= "000" when (din="10000000") else
        "001" when (din="01000000") else
        "010" when (din="00100000") else
        "011" when (din="00010000") else
        "100" when (din="00001000") else
        "101" when (din="00000100") else
        "110" when (din="00000010") else
        "111";

end Behavioral;
```

7. Next, we need to add To add a constraint file with the".xdc" extension, as following: Go to "Flow Navigator" and from "Project Manager" select "Add Sources" then "Add or create constraints". Next, choose "Create File" and enter the file name "lab_2" then "OK" followed by "Finish".



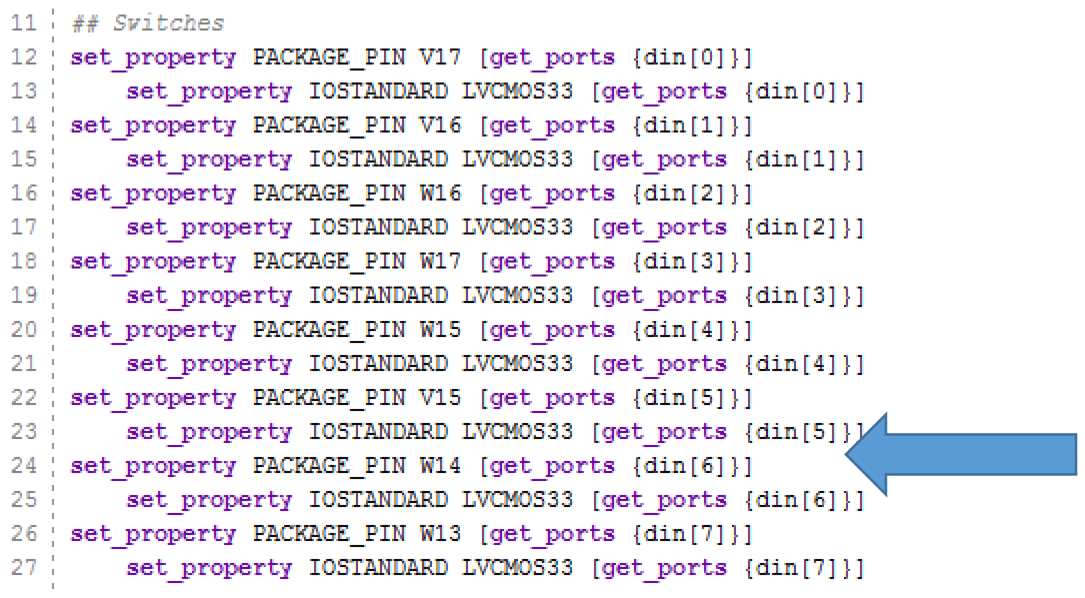
Add Sources

This guides you through the process of adding and creating sources for your project

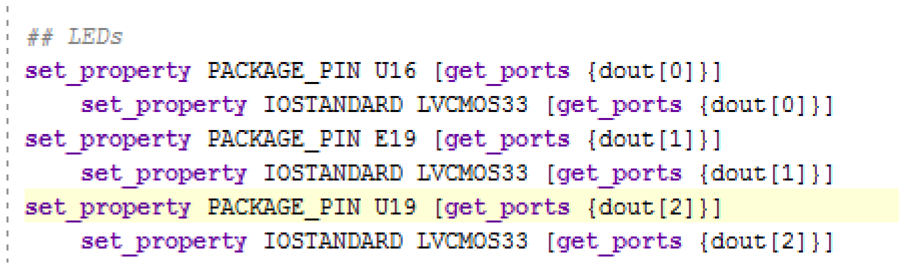
- (●) Add or create constraints
- ( ) Add or create design sources
- ( ) Add or create simulation sources

8. Then, we need to get a template xdc file that is going to be edited according to the different experiments. Google "basys 3 xdc file" and choose the "xilinix" link that

appears (https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/Basys3/Supporting%20Material/Basys3_Master.xdc). Copy the whole file and paste it into the "port_assign.xdc" that you have just created in the last step. Then uncomment and edit the input Switches and the output LEDs as in the next step.

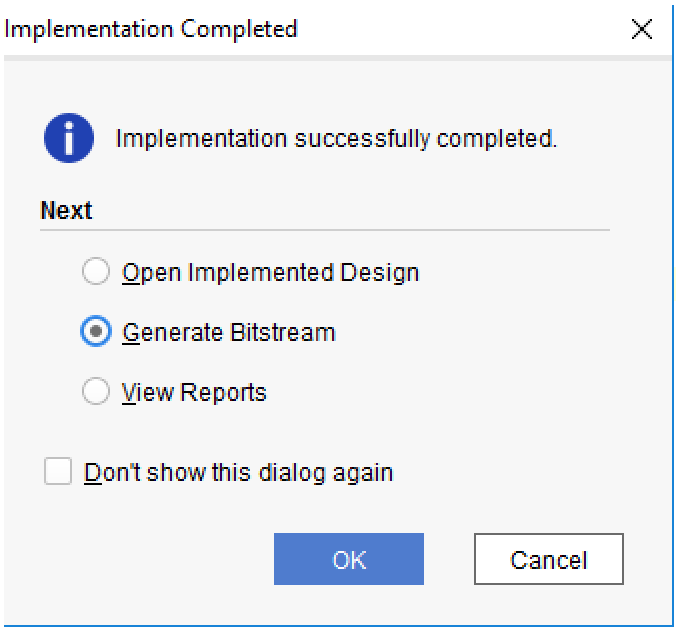


```
## Switches
set_property PACKAGE_PIN V17 [get_ports {din[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[0]}]
set_property PACKAGE_PIN V16 [get_ports {din[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[1]}]
set_property PACKAGE_PIN W16 [get_ports {din[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[2]}]
set_property PACKAGE_PIN W17 [get_ports {din[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[3]}]
set_property PACKAGE_PIN W15 [get_ports {din[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[4]}]
set_property PACKAGE_PIN V15 [get_ports {din[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[5]}]
set_property PACKAGE_PIN W14 [get_ports {din[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[6]}]
set_property PACKAGE_PIN W13 [get_ports {din[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {din[7]}]
```



```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {dout[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {dout[0]}]
set_property PACKAGE_PIN E19 [get_ports {dout[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {dout[1]}]
set_property PACKAGE_PIN U19 [get_ports {dout[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {dout[2]}]
```

9. From the tool tab choose the play button and then "Run Implementation". Select "Number of jobs" =1 and then press OK.
10. The implementation errors window will appear if any or the successfully completed window. From this window select "Generate Bitstream" and then OK. This will make the software generate ".bin" file to be used in programing the hardware BAYAS 3.


Implementation Completed

Implementation successfully completed.

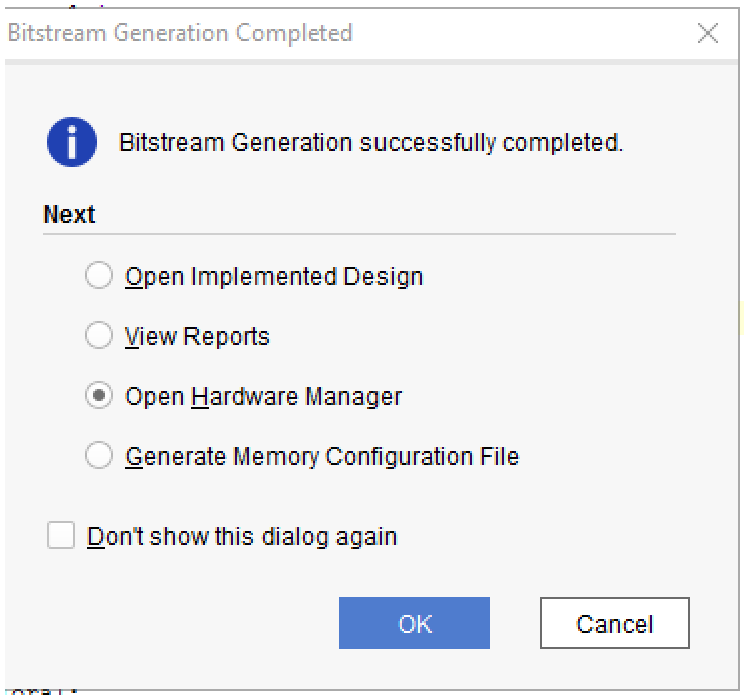
Next

- ( ) Open Implemented Design
- [✓ Generate Bitstream]
- ( ) View Reports

☐ Don't show this dialog again

OK Cancel

11. The next window will appear in which choose "Open Hardware Manger", connect the Hardware Kit to the USB port and then press OK.


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

OK Cancel

12. From the window appears, select the ".bin" file from the Project you create by browsing for the generated ".bit file" under the ".runs" folder and program the board then press OK.

13. Check your board and fill the truth table,

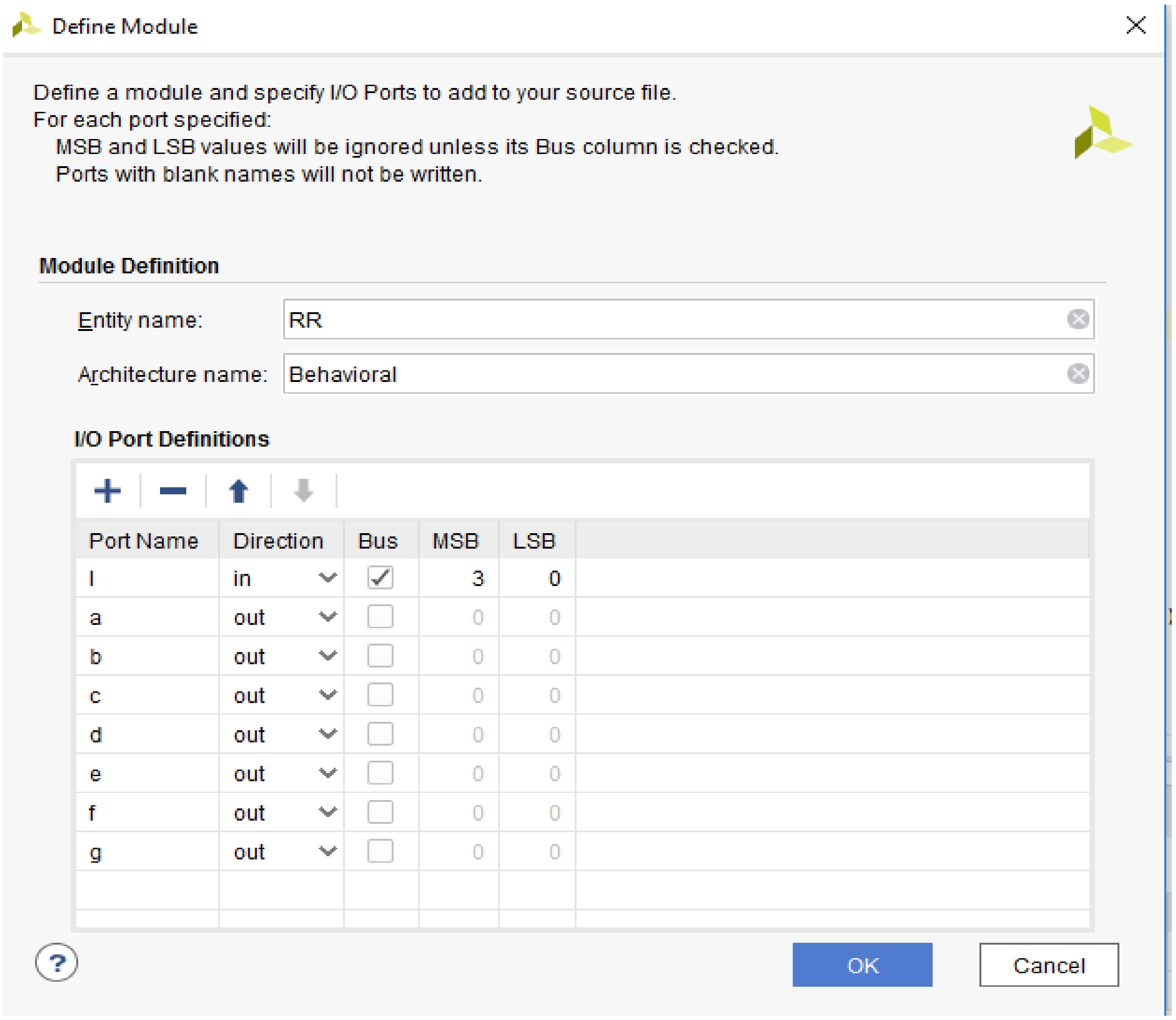
Input									Output		
din$_0$	din$_1$	din$_2$	din$_3$	din$_4$	din$_5$	din$_6$	din$_7$		dout$_0$	dout$_1$	dout$_2$
X	**X**	**X**	**X**	**X**	**X**	**X**	**X**				
H	**H**	**H**	**H**	**H**	**H**	**H**	**H**				
L	**H**	**H**	**H**	**H**	**H**	**H**	**H**				
X	**L**	**H**	**H**	**H**	**H**	**H**	**H**				
X	**X**	**L**	**H**	**H**	**H**	**H**	**H**				
X	**X**	**X**	**L**	**H**	**H**	**H**	**H**				
X	**X**	**X**	**X**	**L**	**H**	**H**	**H**				
X	**X**	**X**	**X**	**X**	**L**	**H**	**H**				
X	**X**	**X**	**X**	**X**	**X**	**L**	**H**				
X	**X**	**X**	**X**	**X**	**X**	**X**	**L**				

14. List your comments from last step

Checked by______________________________ Date ____________

Section 2:

1. Follow section 1 from step 1 to 4 but use a different project and source name (such as ***deco_7seg***).
2. The Define Module Window that will appear, we will choose the input and output labels for the gates under investigation in this experiment.



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

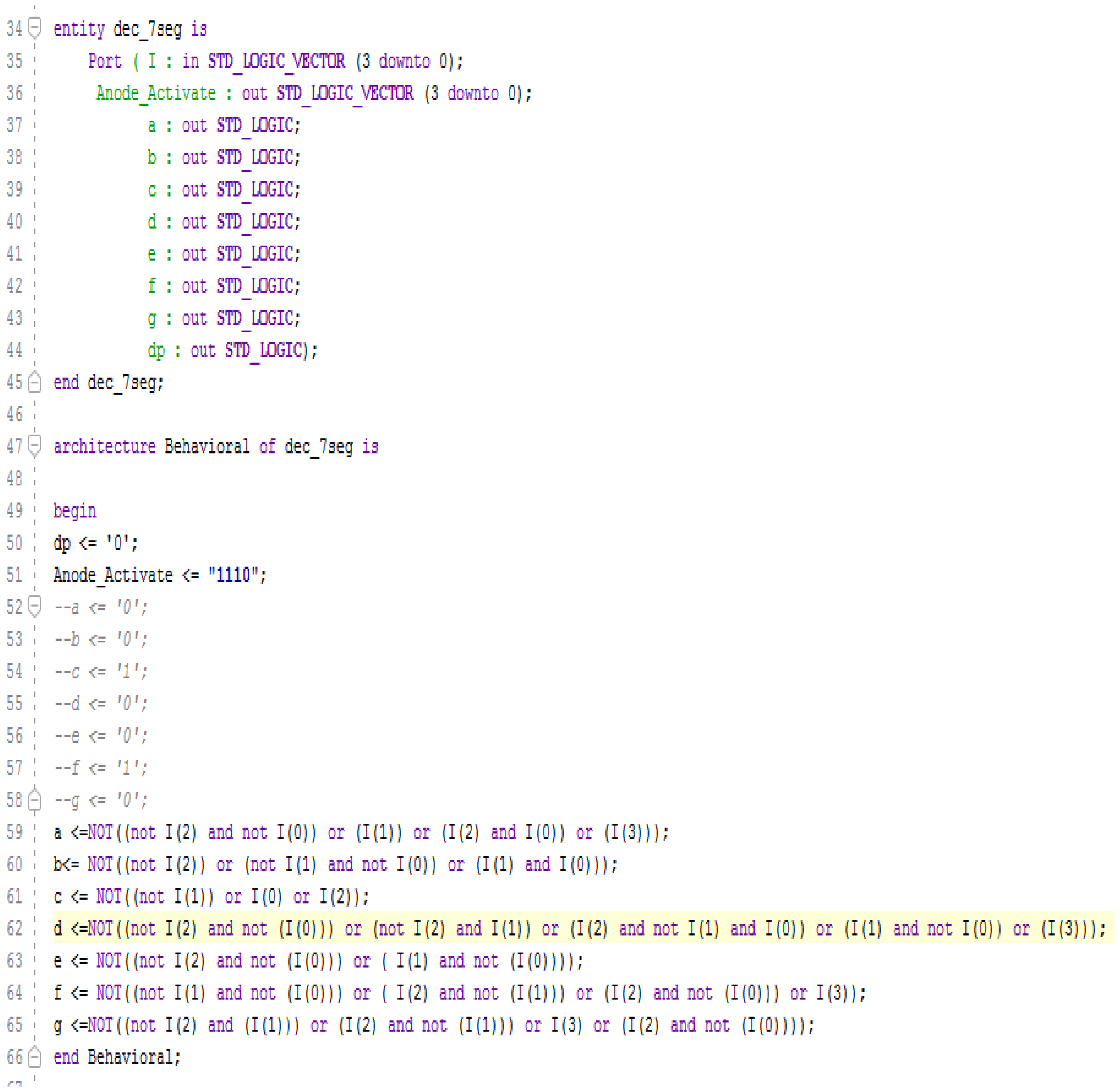
Entity name: RR

Architecture name: Behavioral

I/O Port Definitions

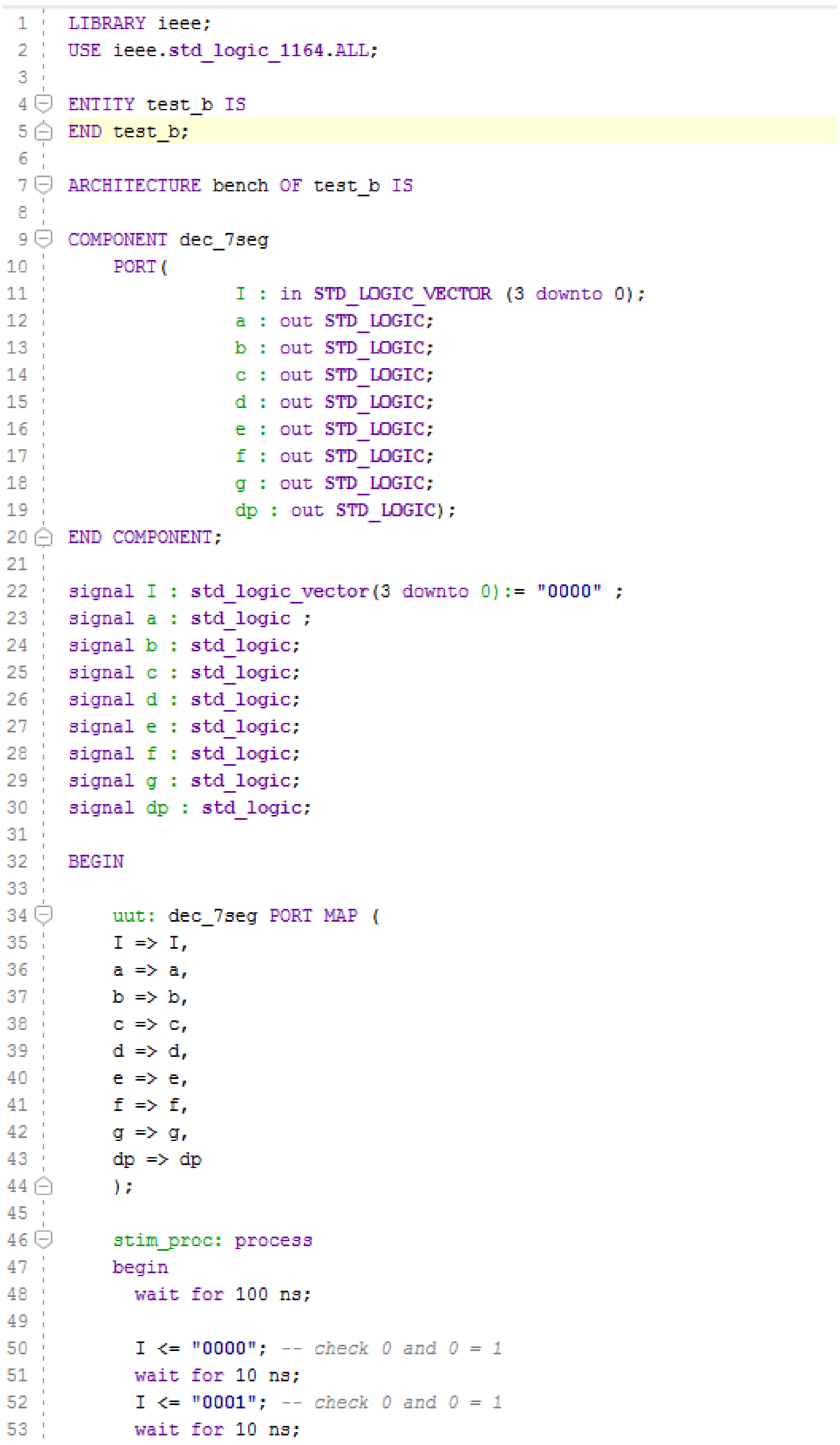
Port Name	Direction	Bus	MSB	LSB
I	in	✓	3	0
a	out		0	0
b	out		0	0
c	out		0	0
d	out		0	0
e	out		0	0
f	out		0	0
g	out		0	0

3. In the "***encoder_8to3***.vhd" created file, type the gates equivalent VHDL code as follows and then save the file.

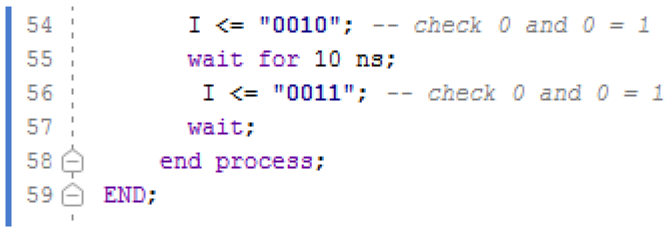


```
entity dec_7seg is
    Port ( I : in STD_LOGIC_VECTOR (3 downto 0);
     Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
           a : out STD_LOGIC;
           b : out STD_LOGIC;
           c : out STD_LOGIC;
           d : out STD_LOGIC;
           e : out STD_LOGIC;
           f : out STD_LOGIC;
           g : out STD_LOGIC;
           dp : out STD_LOGIC);
end dec_7seg;

architecture Behavioral of dec_7seg is

begin
dp <= '0';
Anode_Activate <= "1110";
--a <= '0';
--b <= '0';
--c <= '1';
--d <= '0';
--e <= '0';
--f <= '1';
--g <= '0';
a <=NOT((not I(2) and not I(0)) or (I(1)) or (I(2) and I(0)) or (I(3)));
b<= NOT((not I(2)) or (not I(1) and not I(0)) or (I(1) and I(0)));
c <= NOT((not I(1)) or I(0) or I(2));
d <=NOT((not I(2) and not (I(0))) or (not I(2) and I(1)) or (I(2) and not I(1) and I(0)) or (I(1) and not I(0)) or (I(3)));
e <= NOT((not I(2) and not (I(0))) or ( I(1) and not (I(0))));
f <= NOT((not I(1) and not (I(0))) or ( I(2) and not (I(1))) or (I(2) and not (I(0))) or I(3));
g <=NOT((not I(2) and (I(1))) or (I(2) and not (I(1))) or I(3) or (I(2) and not (I(0))));
end Behavioral;
```

4. Start a test bench as follows



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY test_b IS
END test_b;

ARCHITECTURE bench OF test_b IS

COMPONENT dec_7seg
    PORT(
               I : in STD_LOGIC_VECTOR (3 downto 0);
               a : out STD_LOGIC;
               b : out STD_LOGIC;
               c : out STD_LOGIC;
               d : out STD_LOGIC;
               e : out STD_LOGIC;
               f : out STD_LOGIC;
               g : out STD_LOGIC;
               dp : out STD_LOGIC);
END COMPONENT;

signal I : std_logic_vector(3 downto 0):= "0000" ;
signal a : std_logic ;
signal b : std_logic;
signal c : std_logic;
signal d : std_logic;
signal e : std_logic;
signal f : std_logic;
signal g : std_logic;
signal dp : std_logic;

BEGIN

    uut: dec_7seg PORT MAP (
    I => I,
    a => a,
    b => b,
    c => c,
    d => d,
    e => e,
    f => f,
    g => g,
    dp => dp
    );

    stim_proc: process
    begin
      wait for 100 ns;

      I <= "0000"; -- check 0 and 0 = 1
      wait for 10 ns;
      I <= "0001"; -- check 0 and 0 = 1
      wait for 10 ns;
```

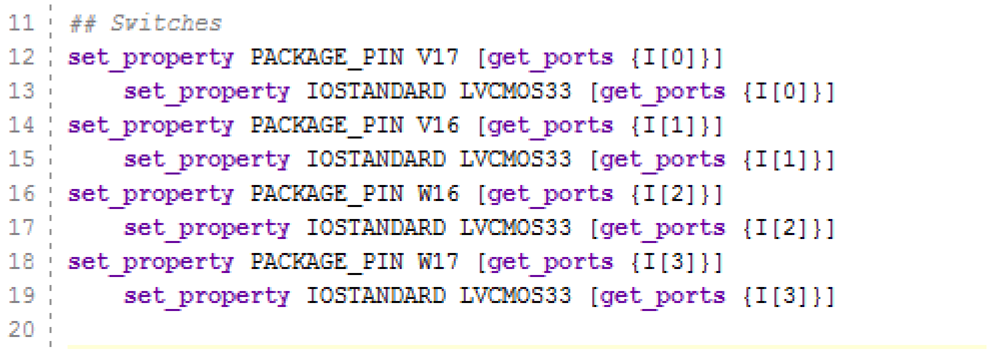


```
        I <= "0010"; -- check 0 and 0 = 1
        wait for 10 ns;
         I <= "0011"; -- check 0 and 0 = 1
        wait;
      end process;
END;
```

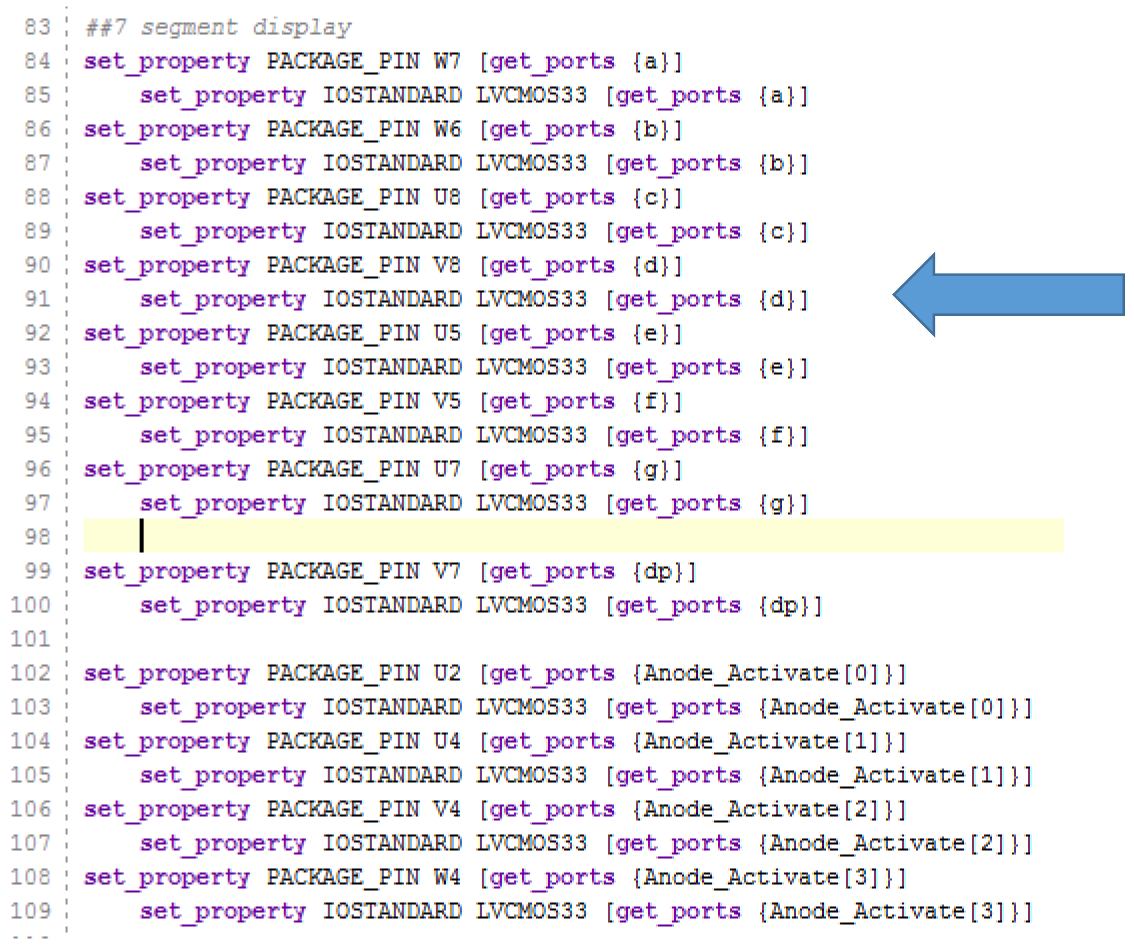
5. What is your observation ?

6. Next, we need to add To add a constraint file with the".xdc" extension, as following:
   Go to "Flow Navigator" and from "Project Manager" select "Add Sources" then "Add or create constraints". Next, choose "Create File" and enter the file name "lab_2" then "OK" followed by "Finish".

7. Then, we need to get a template xdc file that is going to be edited according to the different experiments. Google "basys 3 xdc file" and choose the "xilinix" link that appears (https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/Basys3/Supporting%20Material/Basys3_Master.xdc). Copy the whole file and paste it into the "port_assign.xdc" that you have just created in the last step. Then uncomment and edit the input Switches and the output LEDs as in the next step.

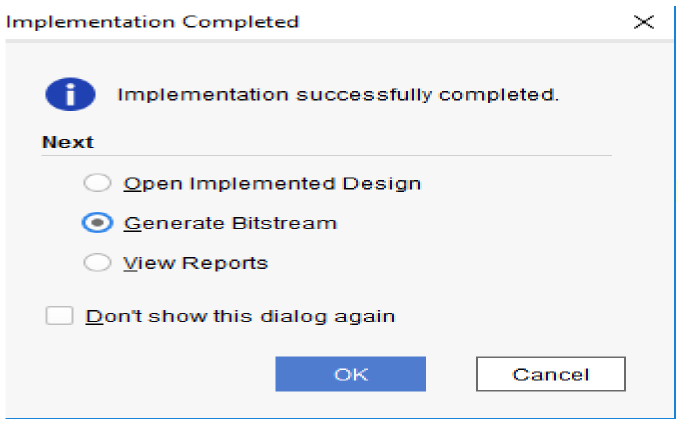


```
## Switches
set_property PACKAGE_PIN V17 [get_ports {I[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {I[0]}]
set_property PACKAGE_PIN V16 [get_ports {I[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {I[1]}]
set_property PACKAGE_PIN W16 [get_ports {I[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {I[2]}]
set_property PACKAGE_PIN W17 [get_ports {I[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {I[3]}]
```



```
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {a}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property PACKAGE_PIN W6 [get_ports {b}]
    set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property PACKAGE_PIN U8 [get_ports {c}]
    set_property IOSTANDARD LVCMOS33 [get_ports {c}]
set_property PACKAGE_PIN V8 [get_ports {d}]
    set_property IOSTANDARD LVCMOS33 [get_ports {d}]
set_property PACKAGE_PIN U5 [get_ports {e}]
    set_property IOSTANDARD LVCMOS33 [get_ports {e}]
set_property PACKAGE_PIN V5 [get_ports {f}]
    set_property IOSTANDARD LVCMOS33 [get_ports {f}]
set_property PACKAGE_PIN U7 [get_ports {g}]
    set_property IOSTANDARD LVCMOS33 [get_ports {g}]

set_property PACKAGE_PIN V7 [get_ports {dp}]
    set_property IOSTANDARD LVCMOS33 [get_ports {dp}]

set_property PACKAGE_PIN U2 [get_ports {Anode_Activate[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode_Activate[0]}]
set_property PACKAGE_PIN U4 [get_ports {Anode_Activate[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode_Activate[1]}]
set_property PACKAGE_PIN V4 [get_ports {Anode_Activate[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode_Activate[2]}]
set_property PACKAGE_PIN W4 [get_ports {Anode_Activate[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Anode_Activate[3]}]
```

8. From the tool tab choose the play button and then "Run Implementation". Select "Number of jobs" =1 and then press OK.
9. The implementation errors window will appear if any or the successfully completed window. From this window select "Generate Bitstream" and then OK. This will make the software generate ".bin" file to be used in programing the hardware BAYAS 3.



Implementation Completed

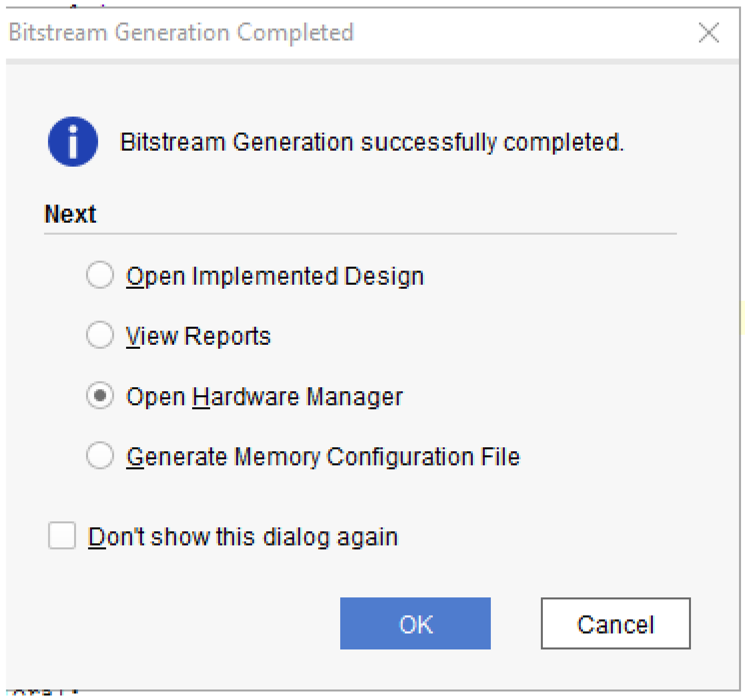
Implementation successfully completed.

Next

( ) Open Implemented Design
[✓ Generate Bitstream]
( ) View Reports

☐ Don't show this dialog again

10. The next window will appear in which choose "Open Hardware Manger", connect the Hardware Kit to the USB port and then press OK.



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

11. From the window appears, select the ".bin" file from the Project you create by browsing for the generated ".bit file" under the ".runs" folder and program the board then press OK.

12. Check your board for the 7-segment display.

Checked by_________________________ Date_______________________

V. Questions:

1.) Priority encoders are much more common than non-priority encoders. What do you think the reasons are for that?

2.) Which bit has the highest priority? I_0 or I_9? (Why is there no EO input in the decimal-to-BCD encoder?

3.) If a common-anode 7-segment is used, what modifications must you make on the project design in Section III? If you have more than one, list them all.

4.) Name two applications for encoders.