Multi-Output Circuits: Encoders, Decoders, and Memories

Introduction

Boolean expressions are used to output a Boolean function of number of variables. Dataflow construct can be used to model such functions. There are number of circuits in use which have multiple outputs and multiple inputs. In this lab you will design encoders, decoders, and read only memories. *Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits*.

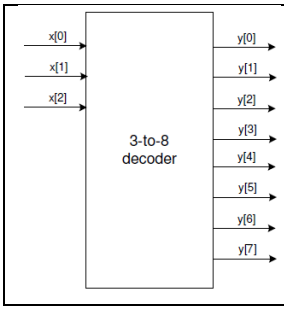
Objectives

After completing this lab, you will be able to:

- Design multi-output decoder circuits using behavioral modeling
- Design encoders using behavioral modeling
- Design read only memories

Multi-output Decoder Circuits Part 1

Decoders are combinatorial circuits which have multiple outputs. They are widely used in memory chips to select one of the words addressed by the address input. For example, an 8-words memory will have three bit address input. The decoder will decode the 3-bit address and generate a select line for one of the eight words corresponding to the input address. The 3-to-8 decoder symbol and the truth table are shown below.



x_0 x_1 x_2	y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
0 0 0	0 0 0 0 0 0 0 1
0 0 1	0 0 0 0 0 0 1 0
0 1 0	0 0 0 0 0 1 0 0
0 1 1	0 0 0 0 1 0 0 0
1 0 0	0 0 0 1 0 0 0 0
1 0 1	0 0 1 0 0 0 0 0
1 1 0	0 1 0 0 0 0 0 0
1 1 1	1 0 0 0 0 0 0 0

Such circuits, also known as binary decoders, can be modeled using dataflow statements as each output is true for a unique input combination.

1-1. Design a 3-to-8 line decoder. Let the input be through SW2-SW0 and output be on LED7-LED0. Use dataflow modeling constructs.

1-1-1. Open Vivado and create a blank project called lab3_1_1**.**

1-1-2. Create and add the VHDL module, naming it decoder_3to8_dataflow.vhd that defines the 3-to-8 line decoder with three-bit input *x* and 8-bit output *y*. Use dataflow modeling constructs.

1-1-3. Add the provided testbench (decoder_3to8_dataflow_tb.vhd) to the project.

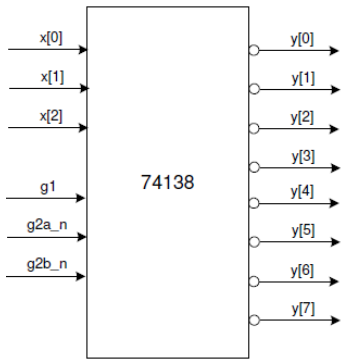
1-1-4. Simulate the design for 50 ns and verify that the design works.

1-1-5. Create and add the XDC file to the project. Assign *x* to ***SW2-SW0*** and *y* to ***LED7-LED0***. Note that one and only one LED will be turned ON for a given input combination.

1-1-6. Synthesize and implement the design.

1-1-7. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

1-2. Design and implement a popular IC, 74138, functionality using dataflow modeling and the decoder you used in 1-1. The IC symbol and truth table are given below.



g_1 g_{2a_n} g_{2b_n}	x_0 x_1 x_2	y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7
0 x x	x x x	1 1 1 1 1 1 1 1
x 1 x	x x x	1 1 1 1 1 1 1 1
x x 1	x x x	1 1 1 1 1 1 1 1
1 0 0	0 0 0	0 1 1 1 1 1 1 1
1 0 0	0 0 1	1 0 1 1 1 1 1 1
1 0 0	0 1 0	1 1 0 1 1 1 1 1
1 0 0	0 1 1	1 1 1 0 1 1 1 1
1 0 0	1 0 0	1 1 1 1 0 1 1 1
1 0 0	1 0 1	1 1 1 1 1 0 1 1
1 0 0	1 1 0	1 1 1 1 1 1 0 1
1 0 0	1 1 1	1 1 1 1 1 1 1 0

X = don't care

Note that this is very similar to the one you had created in 1-1, It has additional control (Enable) signals g1, g2a_n and g2b_n. These enable signals simplify decoding in some systems.

1-2-1. Open Vivado and create a blank project called lab3_1_2**.**

1-2-2. Create and add the VHDL module, named decoder_74138_dataflow, instantiating the model you had developed in 1-1. Add additional logic, by using the dataflow modeling constructs, to model the desired functionality.

1-2-3. Add the provided testbench (decoder_74138_dataflow_tb.vhd) to the project.

1-2-4. Simulate the design for 200 ns and verify that the design works.

1-2-5. Add the XDC file you had created in 1-1 to the project. Modify the XDC file to assign *g1* to **SW7**, *g2a_n* to ***SW6***, and *g2b_n* to ***SW5***.

1-2-6. Synthesize and implement the design.

1-2-7. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

Multi-output Encoder Circuits Part 2

Encoder circuit converts information from one format (code) to another for the purposes of standardization, speed, secrecy, security, or saving space by shrinking size. In digital circuits, encoding information may reduce size and/or prioritize functions. Widely used encoder circuits examples include priority encoders, Huffman encoders, etc.

2-1. Design an 8-to-3 priority encoder, whose truth table is given below. Use behavioral modeling.

INPUTS									OUTPUTS				
E1	0	1	2	3	4	5	6	7	A2	A1	A0	GS	E0
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

X : Don't Care

2-1-1. Open Vivado and create a blank project called lab3_2_1**.**

2-1-2. Create and add the VHDL module with *v* and *en_in_n* input; *y***,** *en_out***,** and *gs* output. The *v* input will be 8-bit data inputs (labeled 0 to 7 in the table), *en_in_n* input will be one bit (E1), *y* output will be 3-bit (A2, A1, A0), *en_out* will be one bit output (GS), and **en_out** will be one bit output (E0).

2-1-3. Create and add the XDC file to the project. Assign *x* input to ***SW7-SW0***, *en_in_n* to ***SW15***, *y* to ***LED2-LED0***, *en_out* to ***LED7***, and *gs* to ***LED6***.

2-1-4. Synthesize and implement the design.

2-1-5. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

Read-Only Memories Part 3

Read-only memories (ROM) consist of interconnected arrays to store an array of binary information. Once the binary information is stored it can be read any time but cannot be altered. Large ROMs are typically used to store programs and/or data which will not change by the other circuitry in the system. Small ROMs can be used to implement combinatorial circuits. A ROM uses a decoder, similar to one designed in 1-1 earlier, to address a particular location.

A ROM will have m address input pins and n information output pins to store 2^m words information, each word being n bit in length. The content is accessed by placing an address and the content of the corresponding word is read at the output pins.

In VHDL, memories can be defined as a two dimensional array using **`array`** data type, as illustrated below:

```
architecture behavioral of ROM_16x4 is
type rom is array (0 to 2**4 – 1) of std_logic_vector (3 downto 0);

constant MY_ROM : rom := (
      0 => "0000",
      1 => "0001",
      …
      15 => "1111"
);
```

where **array** is the data type, MY_ROM is a 16x4 memory with 16 locations each location being 4-bit wide. If the memory is to be modeled as read only then two things must happen: (i) memory should only be read and not written into, and (ii) memory should somehow be initialized with the desired content. The constant declarations immediately after the type declaration initializes the memory to the desired content, in this case binary bits counting from 0000 (decimal 0) to 1111 (decimal 15). Following is an example of definition and usage of 4x2 ROM.

```
entity ROM_4x2 is port (
      ROM_addr : in std_logic_vector(1 downto 0);
      ROM_data : out std_logic_vector(1 downto 0)
);
end entity ROM_4x2;

architecture behavioral of ROM_4x2 is
type rom is array (0 to 2**2 – 1) of std_logic_vector (1 downto 0);

constant MY_ROM : rom := (
      0 => "00",
      1 => "01",
      2 => "10",
      3 => "11"
);

begin
      process (ROM_addr)
      begin
            case ROM_addr is
                  when "00" => ROM_data <= MY_ROM(0);
                  when "01" => ROM_data <= MY_ROM(1);
                  when "10" => ROM_data <= MY_ROM(2);
                  when "11" => ROM_data <= MY_ROM(3);
                  when others => data <= "00";
            end case;
      end process;
end behavioral;
```

3-1. Design a 2-bit comparator that compares two 2-bit numbers and asserts outputs indicating whether the decimal equivalent of word A is less than, greater than, or equal to that of word B. You will model a ROM and initialize it.

3-1-1. Open Vivado and create a blank project called lab3_3_1**.**

3-1-2. Create and add the VHDL module with two inputs (*a, b*) and three outputs (*lt***,** *gt***,** and *eq*) using a ROM.

3-1-3. Create and add the XDC file, assigning *a* to ***SW3 to SW2***, *b* to ***SW1 to SW0***, *lt* to ***LED2***, *gt* to ***LED1*** and *eq* to ***LED0***.

3-1-4. Synthesize and implement the design.

3-1-5. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

3-2. Implement 2-bit by 2-bit multiplier using a ROM. Output the product in binary on four LEDs.

3-2-1. Open Vivado and create a blank project called lab3_3_2**.**

3-2-2. Create and add the VHDL module with two 2-bit inputs (*a, b*), a 4-bit *product* output using ROM.

3-2-3. Create and add the XDC file, assigning ***a*** to ***SW3-SW2***, *b* to ***SW1-SW0***, and *product* to ***LED3-LED0***.

3-2-4. Synthesize and implement the design.

3-2-5. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

Conclusion

In this lab, you learned how to model multiple output circuits such as decoders, encoders, and ROM. You also learned how to initialize ROM memory.