New York City
College of technology

Component Subsystem Design II

Lab 3

Design Multi-Bit Multiplexer

CET 4805 SECTION:

INSTRUCTOR: Prof Y. Wang

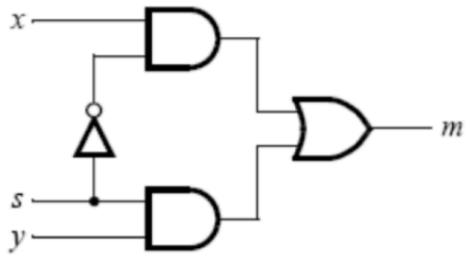
Design Multi-bit Multiplexer

Background

In figure one we can see a sum of products circuit that implements a 2-to-1 multiplexer with a select input s. If s = 0 the multiplexers output m is equal to the input x, and if s = 1 then the output is equal to y. Figure 1 also shows the truth table for this multiplexer and the circuit symbol for the 2-to-1 multiplexer.

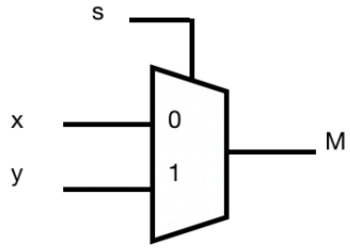
Figure 1. ***2 to 1 multiplexer***

Circuit



s (Select)	M
0	x
1	Y

Truth Table



Circuit Symbol

The single bit multiplexer can be described by the VHDL statement
M <=(NOT (s) AND x) OR (a AND y);

In this lab you will need to write a VHDL entity that includes 5 assignment statements like the one shown above to describe the circuit given

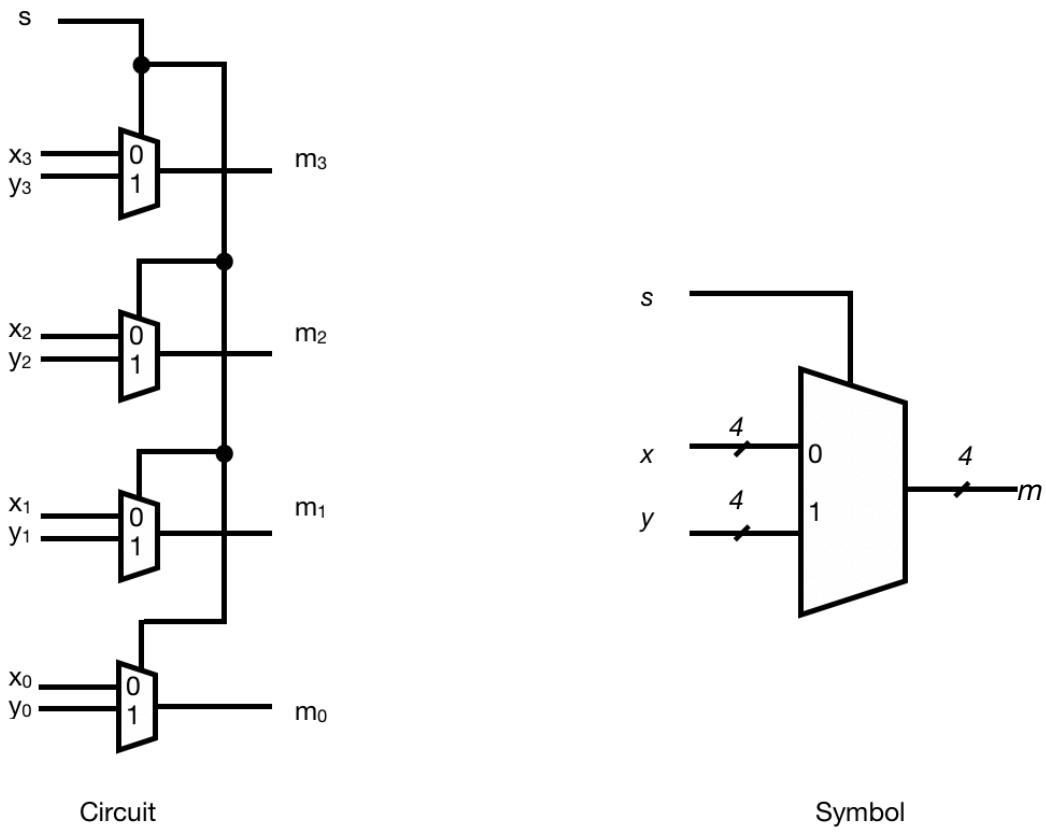


Figure 2. ***Four bit 2 to 1 multiplexer.***

1. Create a Quartus Prime project Multiplexer_1 for your circuit.

2. Include your VHDL file for the four bit wide 2 to 1 multiplexer In your project (follow the approach in the previous lab and generate a digital component that will be reused). Use switch **SW[9]** on the DE10 Board as the S (select input). Switches ***SW[0-3] will serve as the x*** input and ***SW[4-7] as the y input. Connect the output m to the red lights LEDR[0-3]***

(Note : when s = 0 it will choose all the x inputs and when S = 1 it will choose all the y inputs)

3. Include the required pin assignments for the DE10 board. These assignments ensure that the input ports of your VHDL code will use the pins on the Cyclone V FPGA. //FIX

4. Compile the project.

5. Build a vector waveform called Multiplexer_1.vwf and preform a simulation to observe and verify that the circuit is working correctly
6. Download the *.sof* file onto the FPGA chip. Design your test table and test the functionality of the four bit wide 2 to 1 multiplexer by toggling the switches and observing the LED outputs of m.

—Implements a 4 bit wide 2 to 1 multiplexer
—Inputs: SW[0-3] represents the 4 bit input x, and [4-7] represents the 4 bit input y
SW[9] selects either x or y to drive the Output LEDs
—Outputs: LEDR[0-3] represents the outputs of the multiplexers

Implementation I

```
-- LIBRARY NEEDED
ENTITY mux_4bit_2to1 IS
PORT ( s : IN STD_LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- switches
        M: OUT STD LOGIC VECTOR (3 DOWNTO 0); red lights
END mux_4bit_2to1;
ARCHITECTURE Behavior OF mux_4bit_2to1 IS
BEGIN
        M(0) <= (NOT (s) AND x(0)) OR (s AND y(0));
--your code is here
        END Behavior;
```

Implementation II

```
-- LIBRARY NEEDED ENTITY mux_4bit_2to1 IS
 PORT ( s : IN STD LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- switches
        M: OUT STD LOGIC VECTOR (3 DOWNTO 0); red lights

END mux_4bit_2to1;
ARCHITECTURE Behavior OF mux_4bit_2to1 IS
BEGIN
        --use when... else statement
        M       <= X when s = '0' else
                   Y when s= '1' else
```

```
            NULL; --default case
END Behavior;
```

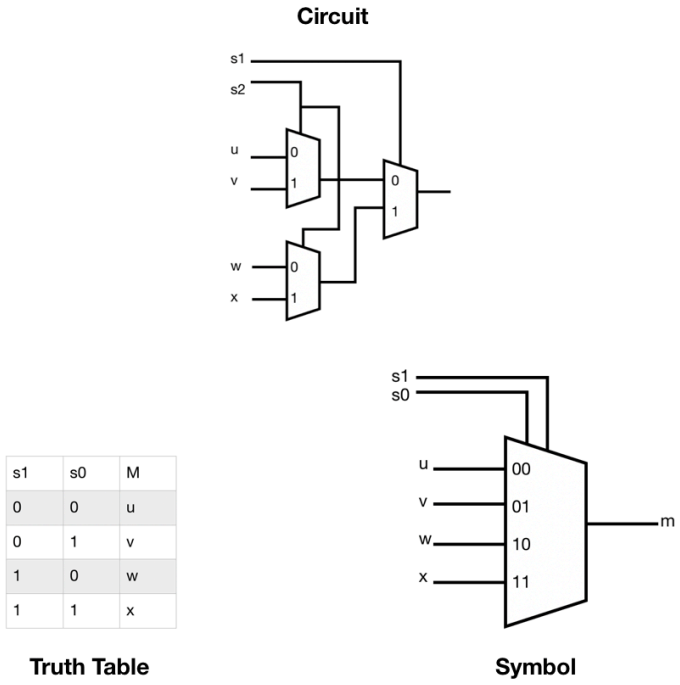
IMPLEMENTATION III

```
-- LIBRARY NEEDED
ENTITY mux_4bit_2to1 IS
PORT ( s : IN STD LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- switches
        M: OUT STD LOGIC VECTOR (3 DOWNTO 0); red lights
END mux_4bit_2to1;
ARCHITECTURE Behavior OF mux_4bit_2to1 IS
 BEGIN
        process (s) --use case statement
         begin
                case s is
                        when '0' => M <= X;
                        when '1' => M <= Y;
                        when others => M <= "0000";
                end case;
        end process;
END Behavior;
```

Part II

Implement a two bit wide 4 to 1 multiplexer. Figure 1 shoed a 2 to 1 multiplexer that selected between the two outputs x and y. Consider a circuit in which output m must be selected from 4 inputs u, v, w, x. Figure 3 shows how we can build the required 4 to 1 multiplexer by using a 3 2 to 1 multiplexers. The circuit used 2 bit select inputs s1 and s0 and implements the truth table shown in figure 1. A circuit symbol for the multiplexer is also shown below.

Figure 3. ***2 bit wide 4 to 1 multiplexer***



s1	s0	M
0	0	u
0	1	v
1	0	w
1	1	x

Recall Figure 2 a 4 bit wide 2 to 1 multiplexer can be build by using 4 instances of a 2 to 1 multiplexer. Figure 2 applies this concept to define a two bit wide 4 to 1 multiplexer.

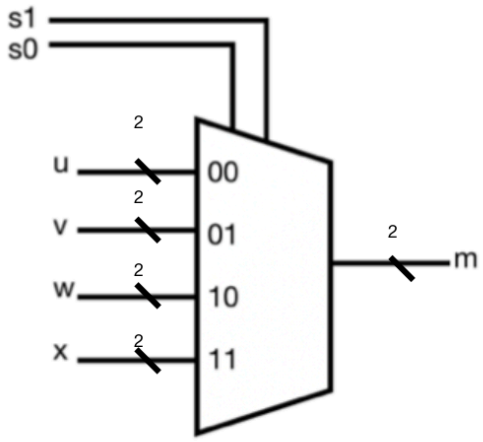


Figure 4. ***2 bit wide 4 to 1 multiplexer***

Preform the following steps to implement the two bit wide 4 to 1 multiplexer.

1. Create a new Quartus prime Multiplexer_2 project for your circuit.

2. Create a VHDL entity for the two bit wide 4 to 1 multiplexer (follow the approach in the previous lab and generate a digital component to be reused). Connect the select inputs to switches SW[8-9]. Use the remaining 8 switches SW [0-7] to provide the 4 two bit inputs u, v, w, x. Connect the M to the red lights LEDR[0-1].
3. Include the required pin assignments for the DE10 Board.

4. Compile the project.

5. Create a vector waveform file called **Multiplexer_2.vwf** and preform a simulation to verify that the circuit works correctly.

6. Download the *.sof* file onto the FPGA chip. Design your test table and test the functionality of the four bit wide 2 to 1 multiplexer by toggling the switches and observing the LED outputs of m.

—Implements a two bit wide 4 to 1 multiplexer

```
ENTITY mux_2bit_4to1 IS
PORT ( S, U, V, W, X,: IN STD LOGIC VECTOR(1 DOWNTO 0);
        M: OUT STD LOGIC VECTOR (1 DOWNTO 0)

ARCHITECTURE Behavior OF mux_2bit_4to1 IS
        --your code is here
END Behavior;
```

Part III

Preform the following steps to implement a three bit wide 5 to 1 multiplexer. You will need to wire your switches and connect them to GPIO pins using a breadboard.

1. Create a new Quartus Prime Project Multiplexer_3 for your circuit.

2. Create a VHDL entity for the three bit wide 5 to 1 multiplexer (Follow the approach in previous parts and generate a digital component to be reused). Connect the **Select to GPIO pins [0-2]**, use **9 switches (SW [0-8]) to provide 3-bit inputs for u, v, and w**. Use **GPIO pins [3 - 8] to provide 3 bit inputs for x, and y**. Provide power to the breadboard using GPIO PIN 11 (VCC 5v). Connect m to output to the red lights LEDR[0-2].

3. Include the required pin assignments for the DE10 Board.

4. Compile the project.

5. Download the *.sof* file onto the FPGA chip. Connect all required wires from the switches on the breadboard to the GPIO. Design your test table and test the functionality of your three bit wide 5 to 1 multiplexer by toggling the GPIO signals (selectors) and observing the LEDS for output m. Ensure the inputs of u, v, w, x, and y can be properly selected.

GPIO 0 (JP1)

PIN_V12	GPIO_0[0]	1		2	GPIO_0[1]	PIN_E8
PIN_W12	GPIO_0[2]	3		4	GPIO_0[3]	PIN_D11
PIN_D8	GPIO_0[4]	5		6	GPIO_0[5]	PIN_AH13
PIN_AF7	GPIO_0[6]	7		8	GPIO_0[7]	PIN_AH14
PIN_AF4	GPIO_0[8]	9		10	GPIO_0[9]	PIN_AH3
	5V	11		12	GND	
PIN_AD5	GPIO_0[10]	13		14	GPIO_0[11]	PIN_AG14
PIN_AE23	GPIO_0[12]	15		16	GPIO_0[13]	PIN_AE6
PIN_AD23	GPIO_0[14]	17		18	GPIO_0[15]	PIN_AE24
PIN_D12	GPIO_0[16]	19		20	GPIO_0[17]	PIN_AD20
PIN_C12	GPIO_0[18]	21		22	GPIO_0[19]	PIN_AD17
PIN_AC23	GPIO_0[20]	23		24	GPIO_0[21]	PIN_AC22
PIN_Y19	GPIO_0[22]	25		26	GPIO_0[23]	PIN_AB23
PIN_AA19	GPIO_0[24]	27		28	GPIO_0[25]	PIN_W11
	3.3V	29		30	GND	
PIN_AA18	GPIO_0[26]	31		32	GPIO_0[27]	PIN_W14
PIN_Y18	GPIO_0[28]	33		34	GPIO_0[29]	PIN_Y17
PIN_AB25	GPIO_0[30]	35		36	GPIO_0[31]	PIN_AB26
PIN_Y11	GPIO_0[32]	37		38	GPIO_0[33]	PIN_AA26
PIN_AA13	GPIO_0[34]	39		40	GPIO_0[35]	PIN_AA11