ECE3270 Digital System Design
Lab 1: Fibonacci Check and Sequence Generator

Lab Overview: The purpose of this exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches (SW_{9-0}) on the DE1-SoC board as inputs to the circuit. We will use light emitting diodes ($LEDR_{9-0}$) and 7-segment displays (*HEX0*-*HEX5*) as output devices. The requirements for this lab consist of completing the QUARTUS designs, demoing the designs, and completing the laboratory report using the LaTeX template. For more information regarding pins and I/O for the DE1-SoC board (in these and all labs), please refer to the lab manual available on Canvas.

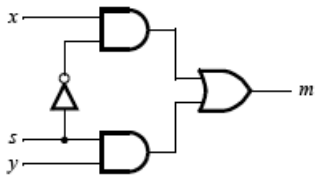
You will submit your report to Turnitin on Canvas and your code to the Assign Server as Assignment 1 by the due date.

Background: The DE1-series board provides 10 toggle switches, called SW_{9-0}, that can be used as inputs to a circuit, and 10 red lights, called $LEDR_{9-0}$, that can be used to display output values. Since there are 10 switches and lights, it is convenient to represent them as arrays (*STD_LOGIC_VECTOR*) in VHDL code.

The DE1-series board has hardwired connections between its FPGA chip and the switches and lights. To use SW_{9-0} and $LEDR_{9-0}$ it is necessary to include in your Quartus project the correct pin assignments, which are given on Canvas. For example, on the DE1-SoC board, SW_0 is connected to the FPGA pin *AB12* and $LEDR_0$ is connected to pin *V16*. A good way to make the required pin assignments is to import into the Quartus software the file called *DE1.qsf* for the DE1-SoC board, which is provided on Canvas. The procedure for making pin assignments is described in the Lab0, given in the class lab manual.

It is important to realize that the pin assignments in the .qsf file are useful only if the pin names given in the file are exactly the same as the port names used in your VHDL entity. The file uses the names *SW[0] : : : SW[9]* and *LEDR[0] : : : LEDR[9]* for the switches and lights (note that the Quartus software uses [ ] square brackets for array elements, while the VHDL syntax uses ( ) round brackets). The 7-segment displays behave differently, with each labeled as *HEX0 – HEX5*. Each display has 7 segments (whoah!), indexed by *HEX0[0] : : : HEX0[6]*.

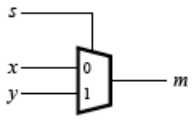
Part I: The figure below shows a sum-of-products circuit that implements a 2-to-1 *multiplexer* with a select input *s*. If $s = 0$ the multiplexer's output *m* is equal to the input *x*, and if $s = 1$ the output is equal to *y*. Part b of the figure gives a truth table for this multiplexer, and part c shows its circuit symbol.



a) Circuit

s	m
0	x
1	y

b) Truth table

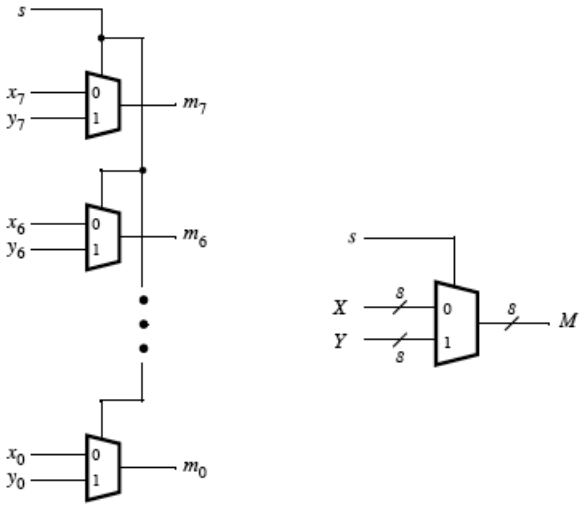


c) Symbol

The multiplexer can be described by the following *Data Flow* VHDL statement:

```
m <= (NOT (s) AND x) OR (s AND y);
```

1. Write a VHDL entity that describes the circuit given in below figure (an eight-bit wide 2-to-1 multiplexer). This circuit has two eight-bit inputs, *X* and *Y*, and produces the eight-bit output *M*. If $s = 0$ then $M = X$, while if $s = 1$ then $M = Y$. It has the circuit symbol shown in *b*, in which *X*, *Y*, and *M* are depicted as eight-bit wires (vectors). Note, it will be very difficult to describe this as data flow! Refer to lecture notes for some good designs that demonstrate multiplexer code!
2. Simulate with a test bench.



a) Circuit b) Symbol

Part II: Create a circuit that accepts a four-bit input and outputs a single bit. This circuit will output logic high if the input bits are a valid Fibonacci number (0, 1, 2, 3, 5, 8, 13) and logic low otherwise.

1. Write the VHDL entity/architecture as described.
2. Simulate with a testbench.
3. Wrap this (as shown in Lab0) in another entity to compile for board use. Map the input to switches and map the output to one of the *LEDR* outputs.

Part III: Create a circuit that accepts a four-bit input and outputs five bits. This circuit will output the next Fibonacci number if the input is a valid number. This circuit will begin at *"0001"* instead of *"0000"* (this is to protect against the difficult beginning of the sequence: 0, 1, 1…). So for *"0001"* the output will be *"00010"* up to *"1101"* outputting *"10101"*. For invalid inputs, output *"11111"*.

1. Write the VHDL entity/architecture as described.
2. Simulate with a testbench.
3. Wrap this (as shown in Lab0) in another entity to compile for board use. Map the input to switches and map the output to *LEDR* outputs.

Part IV: The figure below shows a *7-segment decoder* module that has the four-bit input $c_3c_2c_1c_0$. This decoder produces fifteen outputs that are used to display a character on a 7-segment display. Table 1 below lists the characters that should be displayed for each valuation of $c_3c_2c_1c_0$. Note only valid Fibonacci numbers from 0-8 are displayed, *E* is displayed (for Error) when a non-Fibonacci number is selected.

The seven segments in the display are identified by the indices 0 to 6 shown in Figure 8. Each segment is illuminated by driving it to the logic value 0.

1. Write a VHDL entity/architecture to activate each of the seven segments. Use only simple VHDL assignment statements in your code to specify each logic function.

2. Simulate with a test bench.
3. Wrap this (as shown in Lab0) in another entity to compile for board use. Map the inputs to switches and connect the outputs of the decoder to the *HEX0* display on the DE1-series board. The segments in this display are called $HEX0_0$, $HEX0_1$, : : :, $HEX0_6$, corresponding to Figure 8. You should declare the 7-bit port:

HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);

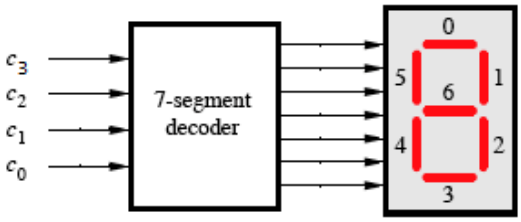


Table 1.

$c_3c_2c_1c_0$	Character
0000	0
0001	1
0010	2
0011	3
0100	E
0101	5
0110	E
0111	E
1000	8
1001	E
…	E
1111	E

Part V: Using all of the components above, create a structural design that links all of the components. Your input to the entity will be a four-bit code for the current number and a one-bit select line for the multiplexer. Your outputs will be a single bit that is high for valid Fibonacci numbers on the input and a seven-bit output that represents the output of the 7-segment decoder. The multiplexer (note you will need to modify the multiplexer shown to the correct bit width) will be used to control if the current Fibonacci input or the next number is sent to the 7-segment decoder (when select is 0 the input is displayed, when 1 the next Fibonacci number is displayed). This only displays 0-8, ignoring the 13 from the next Fibonacci output when selected.

1. Write the VHDL entity/architecture as described.
2. Simulate with a testbench.
3. Wrap this (as shown in Lab0) in another entity to compile for board use. Map the input to switches (you pick which to use) and map the single bit output to one bit of *LEDR* (pick one) and the output from the 7-segment decoder to *HEX0*.

Extra Credit: Modify the circuit to display the tens digit and be able to show *13*. For full extra credit, also be able to display *21* when selecting the next Fibonacci number as the output value. Show the tens digit on *HEX1* and the ones on *HEX0*. Note this will require modifying the multiplexer and creating a new decoder to handle each digit separately.

Rubric	
Report - Proper format - All sections included - Valid images where applicable - Proper grammar, punctuation, and spelling	**50%** - 10% - 30% - 5% - 5%

Demo
- Live Demonstration
  o Includes working code and answering questions from the TA
- Comments
  o Thoughtful comments, not English translations of code
	40%
	- 35%
	
	- 5%

Proper Assign Server Code Submission	**10%**