ECE 3270: Laboratory 1

LAB 1: BASIC GATES

February 8, 2019

Chintan Patel
Clemson University
Department of Electrical and Computer Engineering
`cdpatel@clemson.edu`

Abstract

Lab 1 introduces the student to programming FPGAs in VHDL. The student learns how to connect simple input and output devices to an FPGA device and implement a circuit that uses these devices. The laboratory was divided in five parts: D flip-flop, 2-to-1 Multiplexer, 5-to-1 Multiplexer, 7-segment Decoder and finally, part five involved putting it all together in a cohesive circuit to drive five 7-segment displays.

1 Introduction

Lab one introduces the student to the Quartus II and model sim software as well as programming in VHDL. The lab was divided in five parts. Part one consisted of designing a simple D flip-flop that would latch onto the user input when a pulse was sent through it. Parts two and three consisted on designing multiplexers that would be combined with the D flip-flop from part one in the final cohesive circuit design. Part four involved using the seven segment decoder to drive one of the seven segment displays on the FPGA device. Finally, the final part was to combine the circuits from parts one to four to come up with a cohesive design that would drive five seven segment displays. These displays will print out 'HELLO' and based on the input bits, the letters of the word would rotate in a circular fashion.

2 Design of the D flip-flop

In the first part of the laboratory, a D flip-flop was designed. A D flip-flop latches on to the input with a rising or falling edge. This was quite a simple circuit to design considering it only involved a single input signal. If a rising edge was detected, then the signal value will be latched. A rising edge was sent as a pulse input to allow control of what inputs to latch onto.

3 Design of the 2-to-1 Multiplexer

The second part of the lab involved designing an eight-bit wide 2-to-1 multiplexer with a select input s. Figure 1 below under Figures and Tables shows an implementation of a 2-to-1 multiplexer. If $s = 0$ the multiplexers output m is equal to the input x, and if $s = 1$ the output is equal to y. Part a shows the circuit, Part b gives a truth table for this multiplexer, and part c shows its circuit symbol. An eight-bit wide multiplexer follows the same logic as the implementation shown in Figure 1 below but with eight-bit inputs instead of single bit inputs.This design was then simulated with simulink before being programmed on the FPGA device to test on the actual board.

3.1 Simulation

The simulation involved setting x to "00000000" and y to "11111111" initially with s set to 1. The x and y inputs were later changed to arbitrary values with the select bit being toggled between 1 and 0. As can be seen in figure 2 below(under Figures and Tables), when $s = 0$, input x is selected and when $s = 1$, input y is selected.

3.2 Figures and Tables

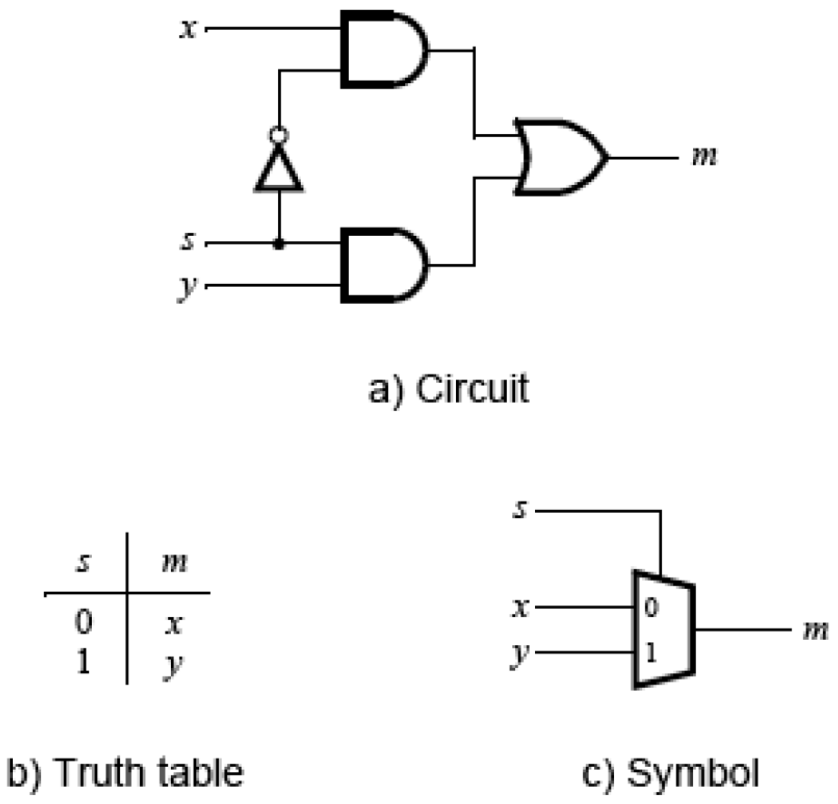


Figure 1: 2 to 1 Multiplexer

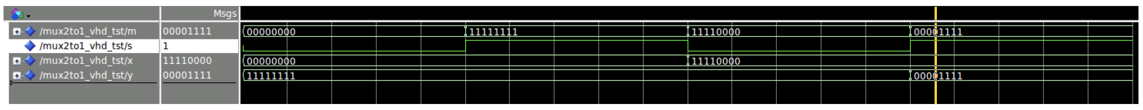


Figure 2: 2 to 1 Multiplexer Simulation

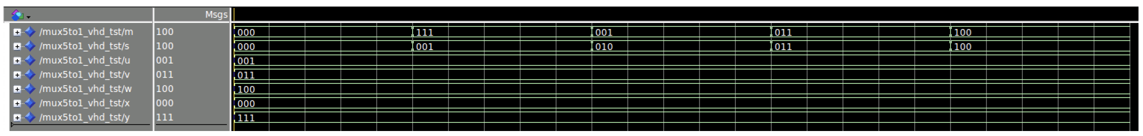
4 Design of the 5-to-1 Multiplexer

The 5-to-1 Multiplexer's design was based off of the 2-to-1 Multiplexer implementation from Figure 1 above. The only difference was that instead of single bit inputs we used three-bit inputs. Therefore, the select(s) was a three bit input. Based on the bit values of the select, one of the five inputs(x, y, u, v, w) is sent out through the output(m) pin. This multiplexer was then simulated with a test bench since it would have required too many pins on the board.

4.1 Simulation

The simulation involved setting inputs u, v, w, x and y to "001", "011", "100", "000" and "111" respectively. The select pin(s) was then toggled from "000" to "100" counting upwards. With each different select value, one of the five input values was output to m. As can be seen from Figure 3 below(under Figures and Tables subsection).

4.2 Figures and Tables



```
                       Msgs
/mux5to1_vhd_tst/m     100    000      111      001      011      100
/mux5to1_vhd_tst/s     100    000      001      010      011      100
/mux5to1_vhd_tst/u     001    001
/mux5to1_vhd_tst/v     011    011
/mux5to1_vhd_tst/w     100    100
/mux5to1_vhd_tst/x     000    000
/mux5to1_vhd_tst/y     111    111
```

Figure 3: 5 to 1 Multiplexer Simulation

5 Design of the 7-segment decoder

The 7-segment decoder produces seven outputs that are used to display a character on a 7-segment display. It takes a three bit input $c_2c_1c_0$. Table 1 below lists the characters that should be displayed for each valuation of $c_2c_1c_0$ (plus the blank character, which is selected for codes 101 - 111).

The seven segments in the display are identified by the indices 0 to 6 shown in Figure 4 below(under Figures and Tables subsection). Each segment is illuminated by driving it to the logic value 0. The display variable was set up with little endian-ness in mind. Therefore, the least significant bit was the rightmost bit and the most significant bit was the leftmost. The code snippet in Figure 5 below shows the design of the architecture and how the three bit input was used to map to a desired letter from Table 1. This design was then simulated with simulink before being programmed on the FPGA device to test on the actual board.

5.1 Figures and Tables

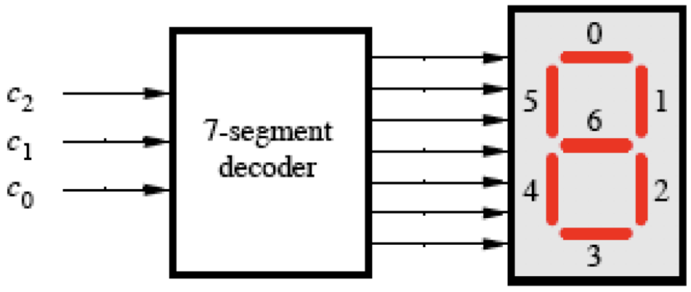
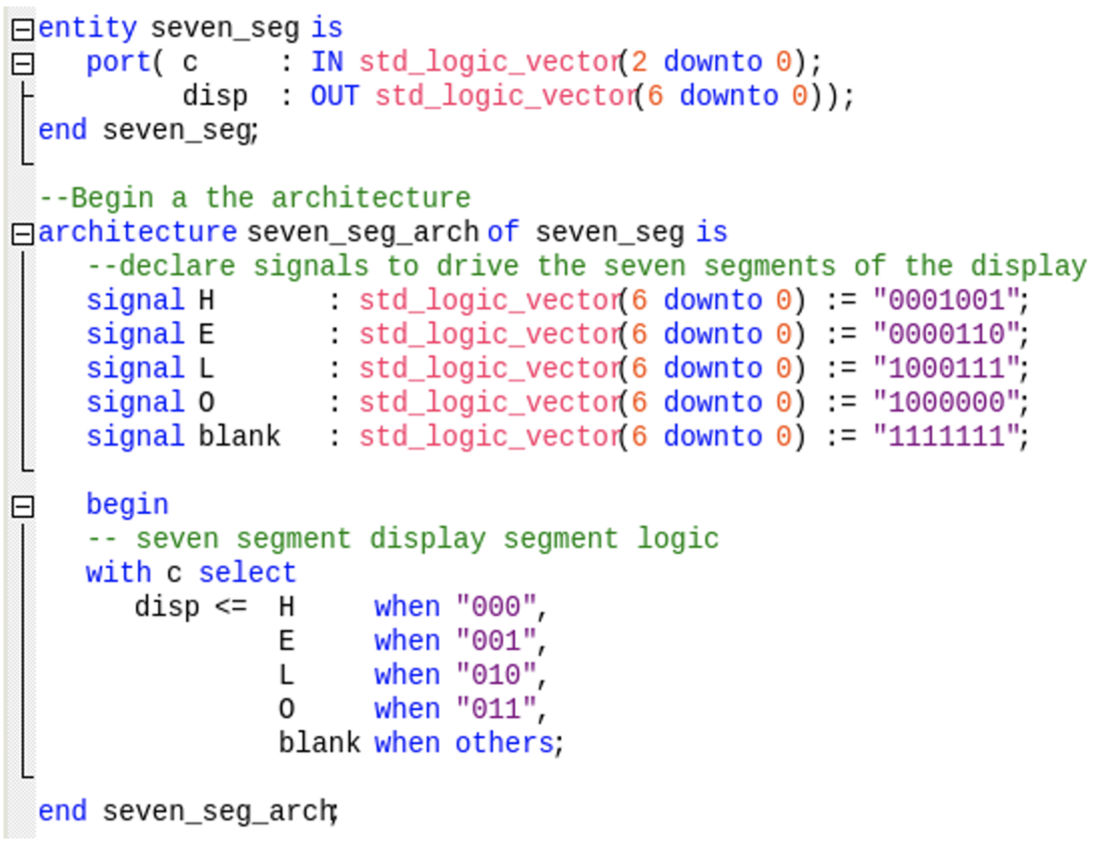


Figure 4: A 7-Segment Decoder



```
entity seven_seg is
   port( c      : IN std_logic_vector(2 downto 0);
         disp   : OUT std_logic_vector(6 downto 0));
end seven_seg;

--Begin a the architecture
architecture seven_seg_arch of seven_seg is
   --declare signals to drive the seven segments of the display
   signal H       : std_logic_vector(6 downto 0) := "0001001";
   signal E       : std_logic_vector(6 downto 0) := "0000110";
   signal L       : std_logic_vector(6 downto 0) := "1000111";
   signal O       : std_logic_vector(6 downto 0) := "1000000";
   signal blank   : std_logic_vector(6 downto 0) := "1111111";

   begin
   -- seven segment display segment logic
   with c select
      disp <=  H     when "000",
               E     when "001",
               L     when "010",
               O     when "011",
               blank when others;

end seven_seg_arch;
```

Figure 5: 7-Segment Decoder Architecture

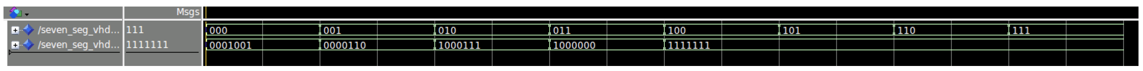


Figure 6: 7-Segment Decoder Simulation

Table 1: Input to Character Mapping

$c_2 c_1 c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

5.2 Simulation

The simulation involved initializing the input to "000" and then waiting for 10ns before changing it to "001" and so on. The input was changed every 10ns counting upwards until "111". Based on Figure 6 above , the simulation produced the desired results.

6 Putting it all together

The final part of the lab exercise involved designing a cohesive circuit that includes components from the previous parts. The final circuit uses a three-bit wide 5-to-1 multiplexer (Part 3) to enable the selection of five characters that are displayed on a 7-segment display. Using the 7-segment decoder from Part 4 this circuit can display any of the characters H, E, L, O and blank. The character codes are set according to Table 1 by using the switches SW2-0, are latched with KEY3-0, and a specific character is selected for display by setting the switches SW9-7. Five 7-segment displays will be used from HEX0-HEX4.

The purpose of the final circuit is to display any word on the five displays that is composed of the characters in Table 1, and be able to rotate this word in a circular fashion across the displays when the switches SW9-7 are toggled. As an example, if the displayed word is HELLO, then the circuit produces the output patterns illustrated in Table 2 below(under Figures and Tables). A portion of the circuit is shown below in Figure 7. As shown in Figure 7, the final circuit will use four D flip-flops to latch various inputs and forward them to the five multiplexers. The multiplexers will then select a certain input based on the value of the selection bits SW9-SW7. The five multiplexer outputs will then be forwarded to the 7-segment decoder which will use the five displays(HEX0-HEX4) to display the selected letters forming a desired word. Upon changing of the selection bits SW9-SW7, the letters will rotate in a circular fashion as shown in Figures 8-12(under the Figures and Tables subsection).

6.1 Simulation

The simulation involved initializing the input SW2-SW0 to "001" to select the letter E and latching it on to HEX3 display. Then the input was changed to "010" to select the letter L and latching it on to HEX2 and HEX1 displays and so on as shown in the code snippet below in Figure 13. Thereafter, the selection pins SW9-SW7 were toggled from "000" to "111" counting upwards to simulate the rotation of the letters in a circular fashion. The results of the simulation can be seen in Figure 14 below.

6.2 Figures and Tables

Table 2: Character Patterns Based on Input Selection

$SW_9SW_8SW_7$	**Character Pattern**				
000	H	E	L	L	O
001	O	H	E	L	L
010	L	O	H	E	L
011	L	L	O	H	E
100	E	L	L	O	H

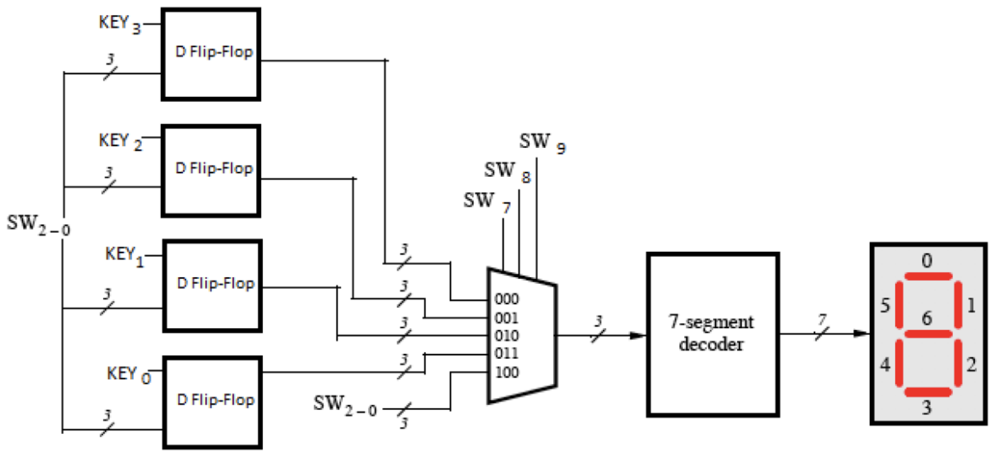


Figure 7: A circuit that can select and display one of the five characters.

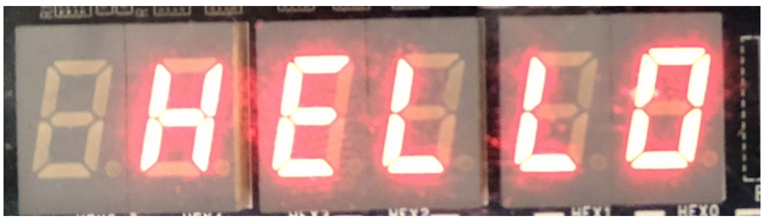


Figure 8: Display of HELLO.

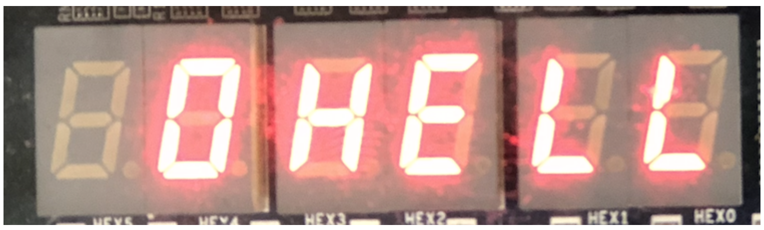


Figure 9: Display of OHELL.

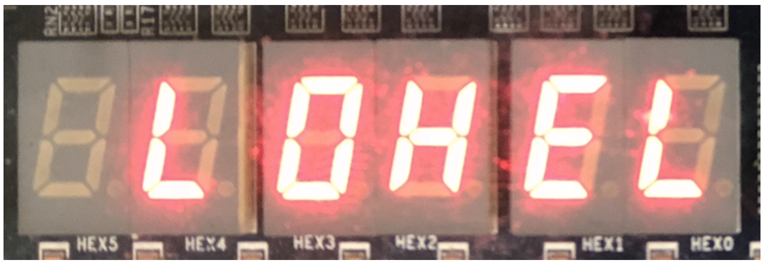


Figure 10: Display of LOHEL.

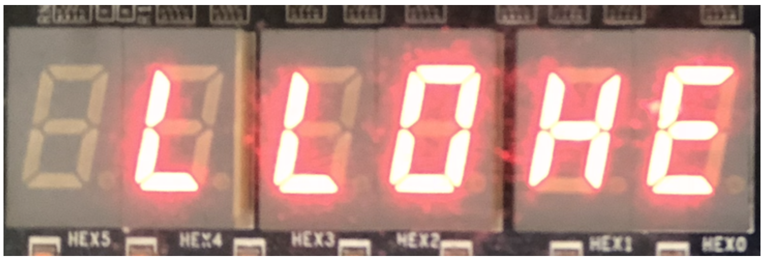


Figure 11: Display of LLOHE.

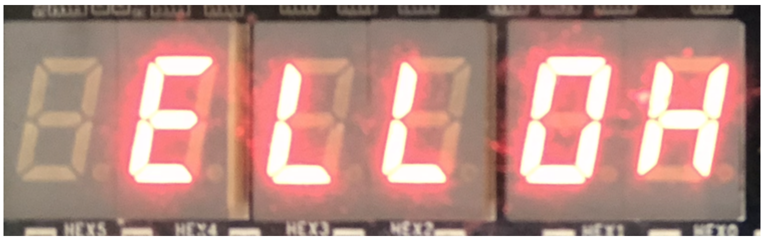
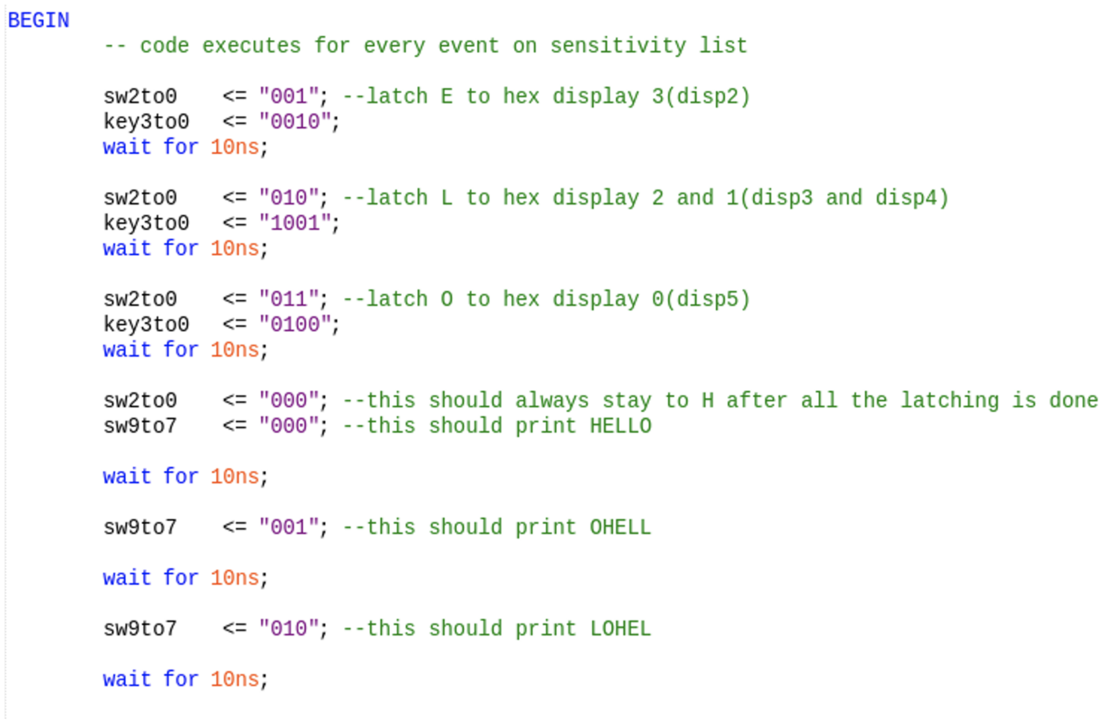


Figure 12: Display of ELLOH.



```
BEGIN
        -- code executes for every event on sensitivity list

        sw2to0    <= "001"; --latch E to hex display 3(disp2)
        key3to0   <= "0010";
        wait for 10ns;

        sw2to0    <= "010"; --latch L to hex display 2 and 1(disp3 and disp4)
        key3to0   <= "1001";
        wait for 10ns;

        sw2to0    <= "011"; --latch O to hex display 0(disp5)
        key3to0   <= "0100";
        wait for 10ns;

        sw2to0    <= "000"; --this should always stay to H after all the latching is done
        sw9to7    <= "000"; --this should print HELLO

        wait for 10ns;

        sw9to7    <= "001"; --this should print OHELL

        wait for 10ns;

        sw9to7    <= "010"; --this should print LOHEL

        wait for 10ns;
```

Figure 13: Code Snippet for the Simulation.

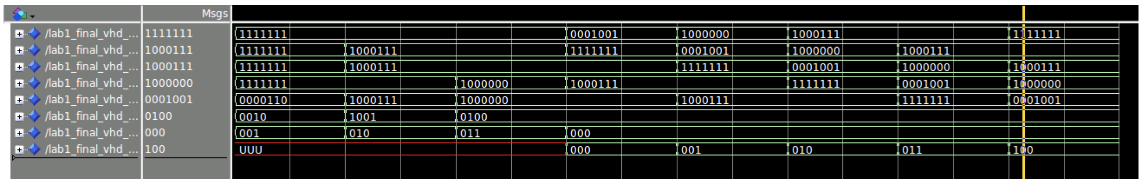

Msgs
/lab1_final_vhd_... 1111111 | 1111111 | 0001001 | 1000000 | 1000111 | 1111111
/lab1_final_vhd_... 1000111 | 1111111 | 1000111 | 1111111 | 0001001 | 1000000 | 1000111
/lab1_final_vhd_... 1000111 | 1111111 | 1000111 | 1111111 | 0001001 | 1000000 | 1000111
/lab1_final_vhd_... 1000000 | 1111111 | 1000000 | 1000111 | 1111111 | 0001001 | 1000000
/lab1_final_vhd_... 0001001 | 0000110 | 1000111 | 1000000 | 1000111 | 1111111 | 0001001
/lab1_final_vhd_... 0100 | 0010 | 1001 | 0100
/lab1_final_vhd_... 000 | 001 | 010 | 011 | 000
/lab1_final_vhd_... 100 | UUU | 000 | 001 | 010 | 011 | 100

Figure 14: Simulation Output.

7 Results

Overall the circuit design worked as expected. As seen in Figures 8-12 above, the word 'HELLO' was latched onto five seven segment displays and rotated successfully in a circular fashion based on the toggling of inputs SW9-SW7. The only hiccup was that the simulation for the final part of the lab did not work as expected. The letter H always ended up not showing in the simulation wave diagram and instead a 'blank' was shown(represented by "1111111"). Also the letter selected by SW2-SW0 did not show up in the wave diagram. However, when the same circuit was programmed on the board it worked as expected.

8 Conclusion

The first lab served as an introduction to programming the FPGA devices in VHDL. The lab teaches the student on how to set up a new project in Quartus II, create a testbench for simulation and finally compile and run the program on the FPGA device. It also teaches the student on how to create modular designs first and later incorporate them into a cohesive single circuit.