TALLINN UNIVERSITY OF TECHNOLOGY

COURSE CODE: **IAS0600**

(DIGITAL SYSTEMS DESIGN WITH VHDL)

LABORATORY REPORT NO 4

(TOPIC: CREEPING LINE)

Student: Dismas EZECHUKWU (**184603IVEM**)

Instructors:

Alexander Sudnitson (Associate Professor)

Dmitri Mihhailov (Research Scientist)

DATE: 11/11/2018

1. INTRODUCTION

The aim of this laboratory work is to design, simulate and implement on an FPGA board, a creeping line running on eight (8) seven segment display. To efficiently implement this, it is required to understand many other sections of digital circuit designs such as:

- Operation and configuration of the seven-segment display
- Understanding and coding of a decoder circuitry in VHDL
- Implementation of counters, clock dividers and multiplexers will be handy in this work as well.

The overall steps needed to complete this project is presented in the TASK section below.

1.2 TASK

The task is to implement a creeping line, running on eight seven-segment LED displays. This will be done in three (3) steps as stated below:

a. Implement a decoder that converts a binary value that can be set with four switches to a hexadecimal value that is shown on one seven-segment LED display;

b. Increase the number of seven-segment LED displays to four, with each display being connected to a separate set of four switches (so each seven-segment LED display can be controlled independently);

c. Increase the number of seven-segment LED displays to eight, substitute switches for a 32-bit circular shift register (that shifts 4 bits at a time). Initialize the register with a value corresponding to e.g. student ID code. Shifting speed is not strictly defined, although the line should be readable. Optionally, the decoders can be changed to display other information besides hexadecimal numbers. If the modified decoder can output more than 16 characters, the shift register size should be changed accordingly.

2. BACKGROUND

Seven segment display

A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays. It has seven segments that can be used to form number and characters [1]. This is done by lighting up the light emitting diode (LED) in the segment we want to light up. A simple structure of the seven-segment display is shown in the Figure 1. below.

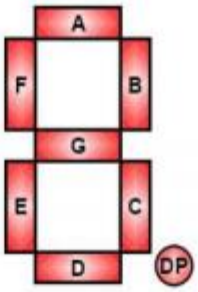


Figure 1: Seven-segment LED Display

BCD to 7-Segment Decoder

This decoder is also known as binary decoder: In digital electronics, a binary decoder is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs. They are used in a wide variety of applications, including data demultiplexing, seven segment displays, and memory address decoding. The figure 2. Below shows the operation of the decoder and generally depicts what will be achieved in the first task of this project.

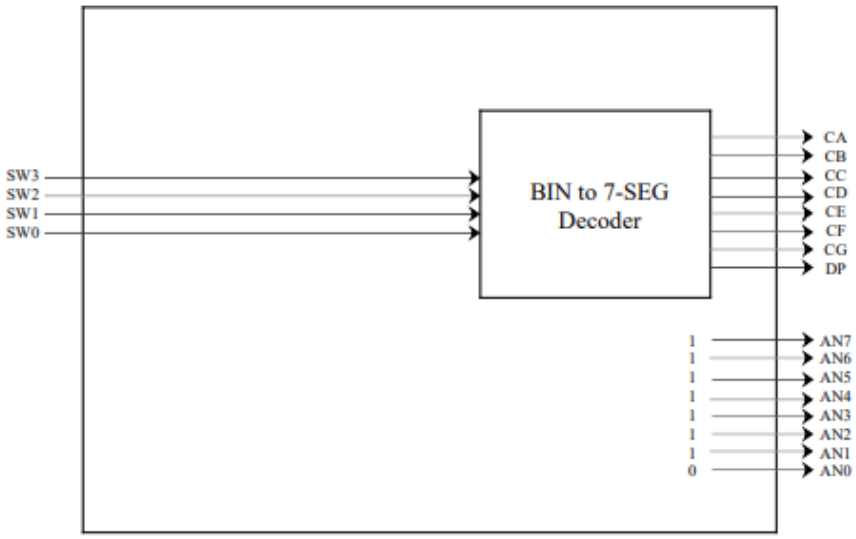


Figure 2: Design Structure for the First Step

From Figure 2. Above, the inputs SW1, SW2, SW3 and SW4 represents the binary input from the switches on the FPGA board provided. While the outputs CA, CB, CC through to CG represents the output of the decoder.

A table 1. Below shows the different input combination fed into the decoder and their corresponding outputs (both in binary and hexadecimal format). It is important to note that the Hex output of the table below will be the output on of the simulator while the Hex input will be displayed on the seven-segment display on the board.

TABLE 1: Decoder input and output

HEX Input	Binary Input				Binary output							HEX Output
	SW3	SW2	SW1	SW0	CA	CB	CC	CD	CE	CF	CG	
0	0	0	0	0	0	0	0	0	0	0	1	**01**
1	0	0	0	1	1	0	0	1	1	1	1	**4F**
2	0	0	1	0	0	0	1	0	0	1	0	**12**
3	0	0	1	1	0	0	0	0	1	1	0	**06**
4	0	1	0	0	1	0	0	1	1	0	0	**4C**
5	0	1	0	1	0	1	0	0	1	0	0	**24**
6	0	1	1	0	0	1	0	0	0	0	0	**20**
7	0	1	1	1	0	0	0	1	1	1	1	**0F**
8	1	0	0	0	0	0	0	0	0	0	0	**00**
9	1	0	0	1	0	0	0	0	1	0	0	**04**
A	1	0	1	0	0	0	0	1	0	0	0	**08**
B	1	0	1	1	1	1	0	0	0	0	0	**60**
C	1	0	0	0	0	1	1	0	0	0	1	**31**
D	1	1	0	1	1	0	0	0	0	1	0	**42**
E	1	1	1	0	0	1	1	0	0	0	0	**30**
F	1	1	1	1	0	1	1	1	0	0	0	**38**

It is important to clarify what the AN), AN1,…AN7 pins stand for. These pins are just like the enable pins of each 7-segment display. As seen in the figure 3. below, all the diodes of similar segment of the eight 7-segment displays are connected to just one pin from the FPGA. Hence to select which digit the signal should display on, we have to use the “AN” pins to select them.

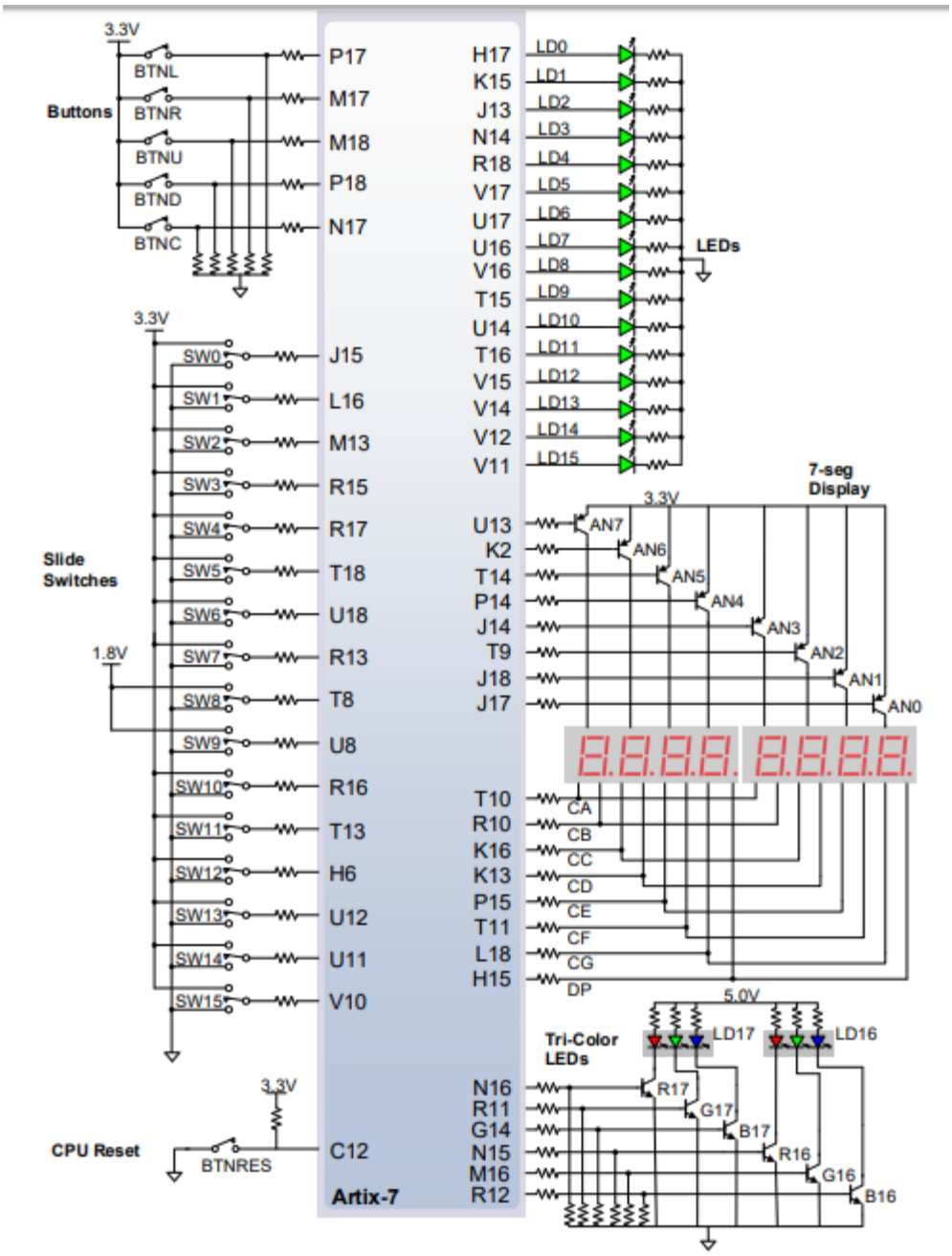


Figure 3. Showing the general purpose I/O devices of the Nexys4 DDR (Atrix - 7) [3]

For the task 2 and task 3, the figures 4 and 5 shows what is to be implemented in them.

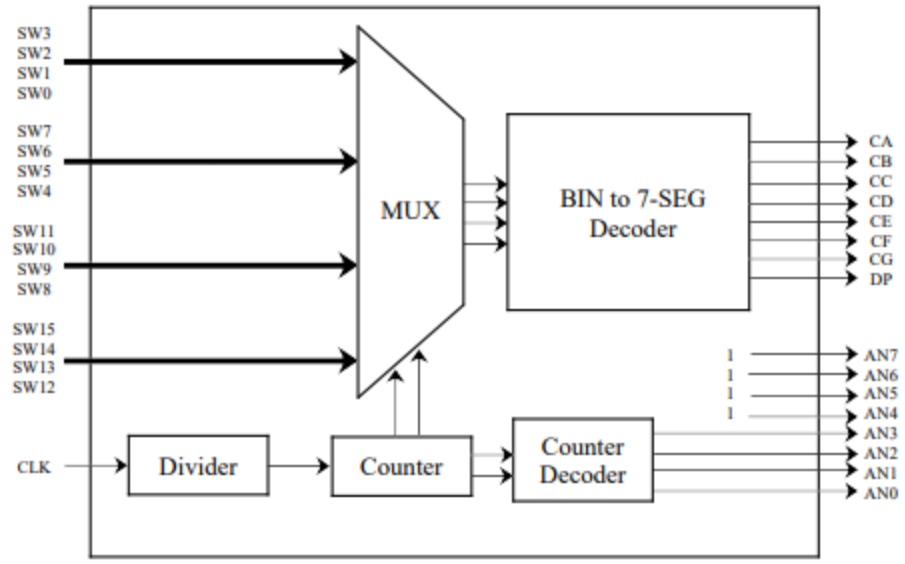


Figure 4. Task two implementation.

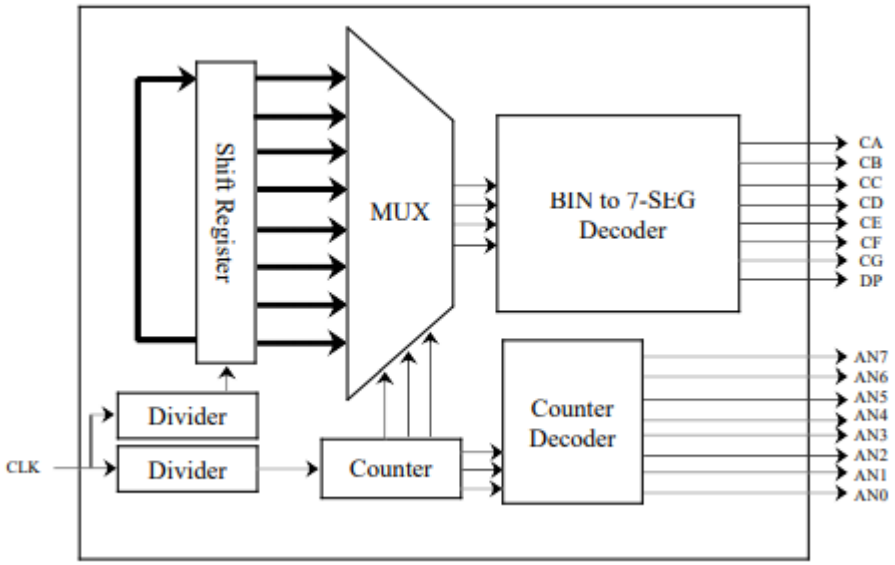


Figure 5. showing Task 3 implementation diagram [2].

Ut is clear from the diagrams that task two will involve 16 input pins while task three will make use of a 32-bit shift register.

3. WORKFLOW

Referencing to the Task section of this report, this project was carried out in Three (3) steps; Task one was coded, simulated and implemented and tested on the FPGA board provided. Task 2 and 3 was also coded, simulated and implemented and tested on the board just like task 1. Only that for task 2, 16 switches were used and for task 3 no switch was used instead a circular shift register is used in its place.

The Test Bench File

The purposes of the testbench file is for exercising and verifying the functional correctness of the creeping line models (tasks 1, 2 and 3). Its structure is described below;

```
clock: process -- period commensurate to 100MHz is 10 ns
begin
clock_100MHz_tb <= '1';
wait for 5 ns;
clock_100MHz_tb <= '0';
wait for 5 ns;
end process;

reset: process
begin
reset_tb <= '1';
wait for 1 ns;
reset_tb <= '0';
wait for 40 ms;
end process;

input: process
begin
input_1_tb <= "0001";
input_2_tb <= "0010";
input_3_tb <= "0011";
input_4_tb <= "0100";
wait for 80 ms;
wait;
end process;

end Behavioral;
```

Figure 6. test bench used for task 2

```
19  signal AN_tb:                STD_LOGIC_VECTOR (7 downto 0);
20
21  begin
22
23  uut: creeping_line_3
24
25  port map(
26           AN=>AN_tb,
27           clock_100MHz=>clock_100MHz_tb,
28           reset=>reset_tb
29  );
30
31  clk:process
32  begin
33  clock_100MHz_tb <= '1';
34  wait for 5 ns;
35  clock_100MHz_tb <= '0';
36  wait for 5 ns;
37  end process;
38
39  reset: process
40  begin
41  reset_tb <= '1';
42  wait for 1 ns;
43  reset_tb <= '0';
44  wait for 100 ms;
45  wait;
46  end process;
47
48  end Behavioral;
```

Figure 7. test bench used for task 3

4. RESULTS AND DISCUSSION

The results for this work will be presented in form of simulation results as well as code sections. They are presented below.

```
13      begin
14      AN<="01111111";
15      with input
16      select
17      output<="0000001" when "0000", --0
18              "1001111" when "0001",--1
19              "0010010" when "0010",--2
20              "0000110" when "0011",--3
21              "1001100" when "0100",--4
22              "0100100" when "0101",--5
23              "0100000" when "0110",--6
24              "0001111" when "0111",--7
25              "0000000" when "1000",--8
26              "0000100" when "1001",--9
27              "0001000" when "1010",--a
28              "1100000" when "1011",--b
29              "0110001" when "1100",--c
30              "1000010" when "1101",--d
31              "0110000" when "1110",--e
32              "0111000" when "1111",--f
33              "1111111" when others; --default
34      end Behavioral;
```

Figure 8. Code section of the decoder.

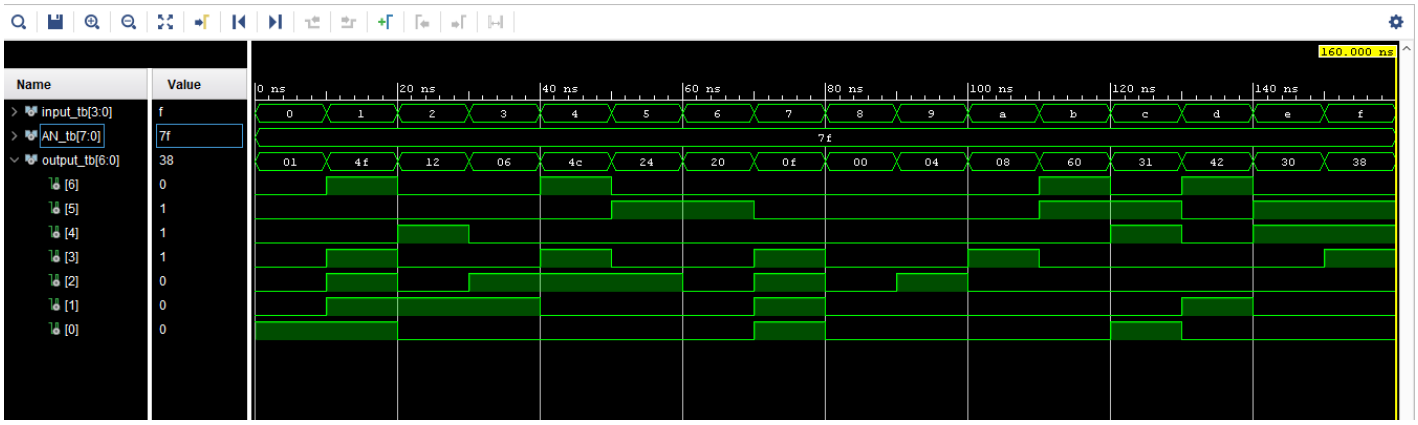


Figure 9. Simulation result for task 1.

In the simulation result in Figure 9. Above, the various outputs of the simulation result are in accordance with the table 1 presented for decoder in section 2. Of this report. It is seen that the AN remained as 7F throughout

the simulation, this means that only one bit of the AN was set ACTIVE so that only one 7-segment display will eventually display.

```
39      ---- counter
40      Counting: process (clock_100MHz)
41       begin
42          if clock_100MHz'event and clock_100MHz = '1' then
43            if reset = '1' then
44              Q <= "00";
45            elsif clock_250Hz = '1' then
46              Q <= std_logic_vector(unsigned(Q) + 1);
47             end if;
48          end if;
49      end process;
50
51      --multiplexer
52      with Q select
53      input <= input_1 when "00", -- for input digit 1
54               input_2 when "01", -- for input digit 2
55               input_3 when "10", -- for input digit 3
56               input_4 when "11", -- for input digit 4
57               "0000" when others; --default
58
59      --counter_decoder
60      with Q  select
61      AN<="01111111" when "00", -- for digit 1
62          "10111111" when "01", --for digit 2
63          "11011111" when "10", --for digit 3
64          "11101111" when "11", --for digit 4
65          "11111111" when others; --default
66
```

Figure 10. Code section of the counter, multiplexer and counter decoder.

The Figure 10 above features different sections (Counter, multiplexer and counter-decoder) of the task 2. Its simulation (for task 2) is shown below

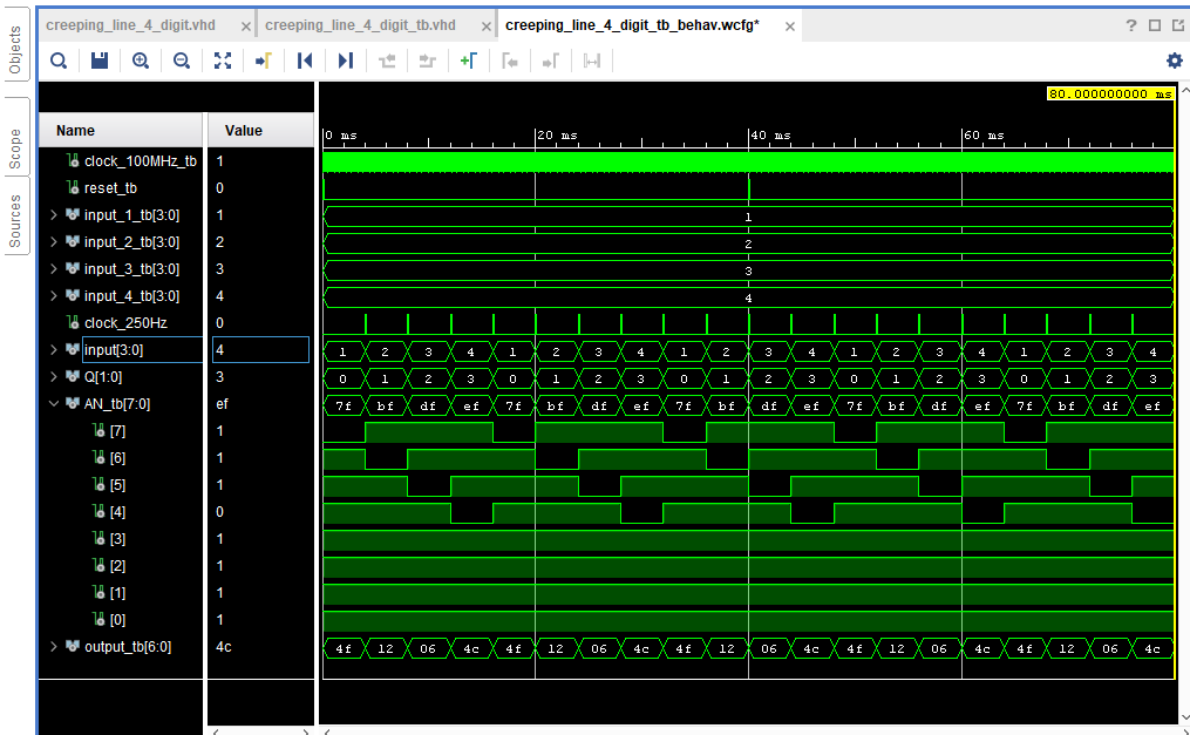


Figure 11. Simulation result for task 2.

From the simulation result (Figure 11) above, the following can be inferred:

- The **input_1_tb, input_2_tb, input_3_tb** and **input_4_tb** (each of which are for bits vectors) represents the input to the multiplexer and ther were all initialized to decimal value 1, 2, 3 and 4 respectively. The output of the multiplexer is **input.** It can be seen that the output of the multiplexer (which is denoted **input**) switches to different decimal inputs of the multiplexer as the outputs of the counter (**Q**) changes appropriately.
- The clock divider output (which is denoted with signal **clock_250Hz**) is used to control the increment of the counter output (**Q**)
- The AN also switches as it is controled by the counter output (**Q**). It switches to select the appropriate display on which the output will show on.
- The output also responds accordingly to the input as depicted by the decoder table. Only that this occurs in a cyclic form.
- Figure 10. Code section of the counter, multiplexer and counter decoder.
- Figure 10. Code section of the counter, multiplexer and counter decoder.

```
--shift register
Cir_shift_reg: process(clock_100MHz)
begin
 if clock_100MHz'event and clock_100MHz = '1' then
    if clock_1Hz = '1' then
          circular_shift_reg(4 to 31) <= circular_shift_reg(0 to 27);
          circular_shift_reg(0 to 3) <= circular_shift_reg(28 to 31);
    end if;
 end if;
end process;
```

Figure 12. Code section of the 32-bits circular shift register used for the task 3.

It si seen that the frequency of the clock used to synchronize the circular shift register is 1Hz, this means that the display will be **creeping** every 1 seconds of time. The simulation of task 3 is shown below.

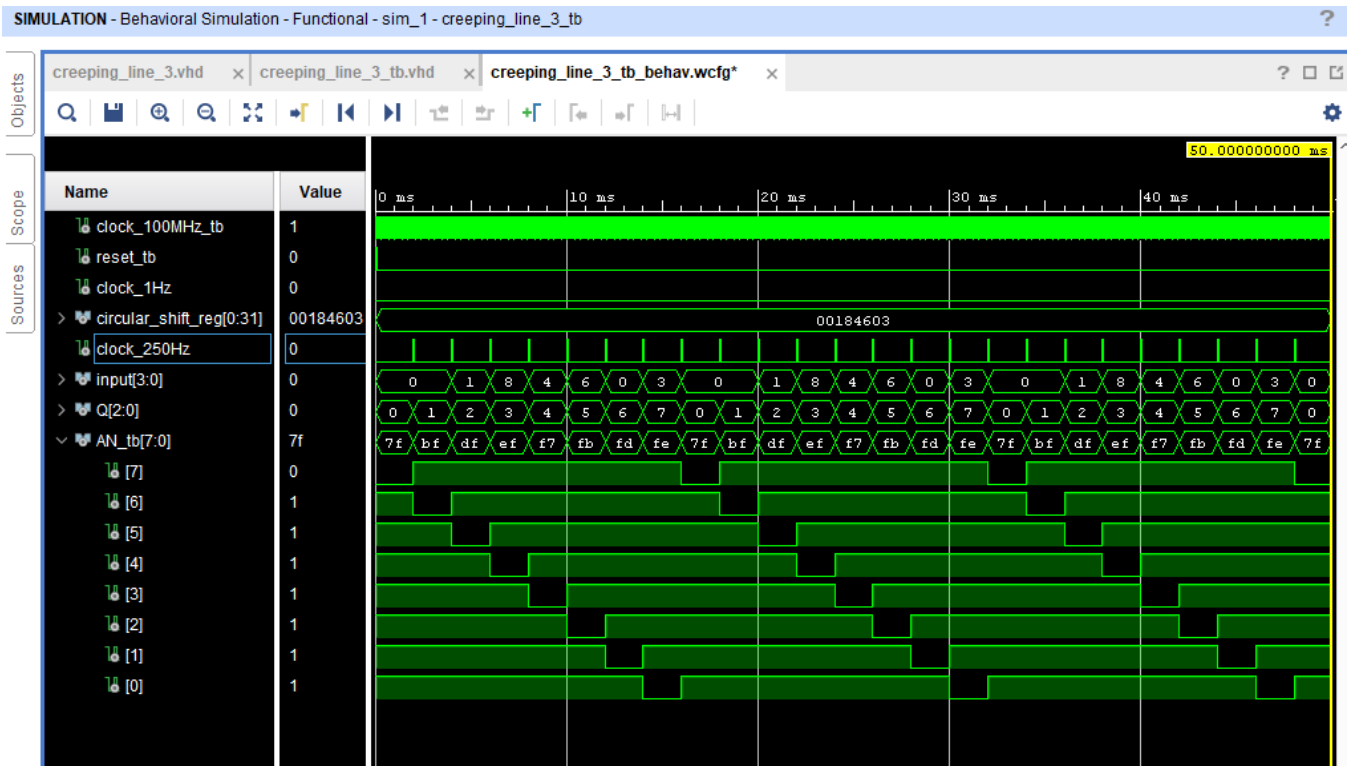


Figure 13. Simulation result for task 3.

The simulation above for the task 3 of this project shows the shift register with an initialized value of 00184603. The rest of it is similar to task 2 above. Only that the AN is now switching all the eight (8) displays.

Picture

Pictures from the tasks are shown below;

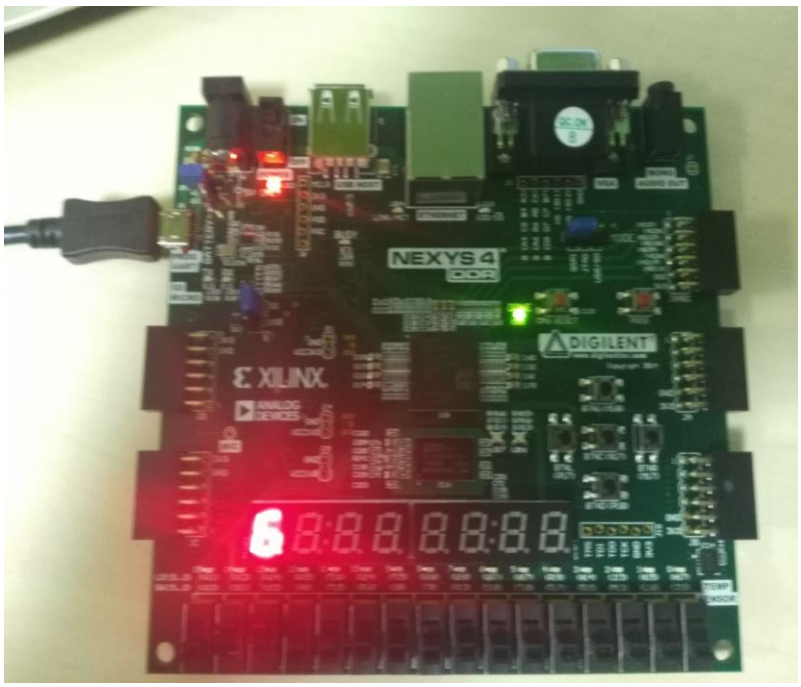


Figure 14. Picture of task 1 on the board.

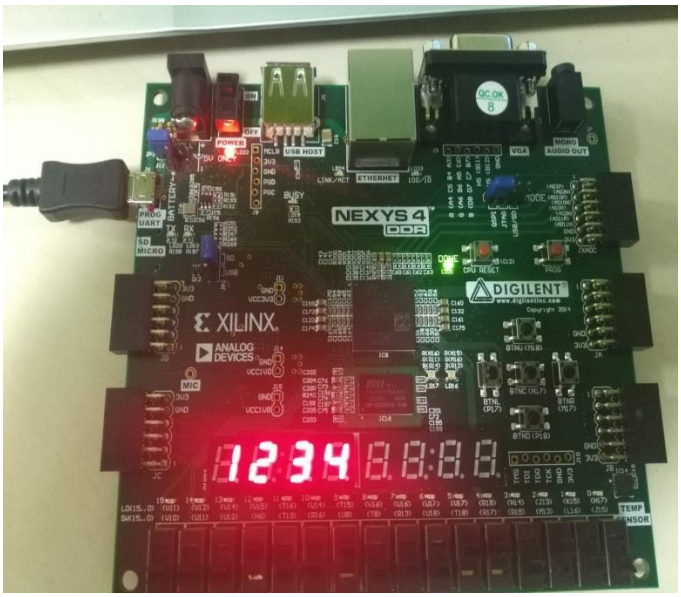
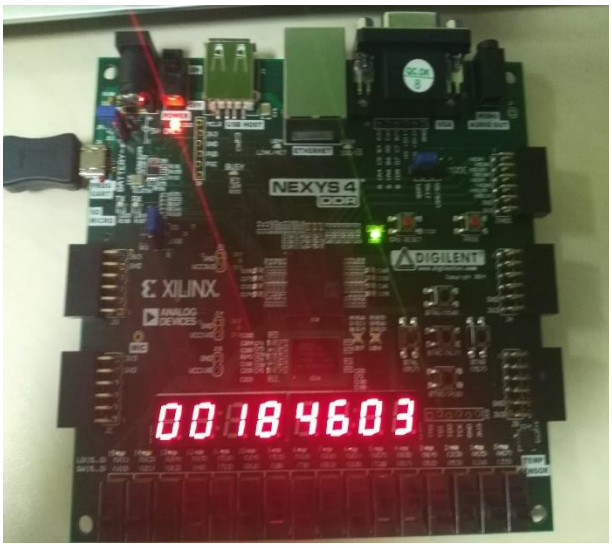


Figure 14. Picture of task 1 on the board.



.

Figure 16. Picture of task 3 on the board.

5. CONCLUSION

In this project, we have been able implement a creeping line on eight (8) 7-segmen display present in the Nexys4 FPGA board made by Xillinx. Implementing the tasks involved us building other little parts of the projects viz; counter, clock divider, multiplexers and decoders. We also simulated the task in vivado software and it worked

correctly. In the first task, four switches were used to only control one display, then 16 switches were used for the second one while in the third one, a 32-bit circular shift register was employed and they all worked correctly bot on the physical board and on simulation.

6. References

[1] Wikipedia, “Binary Decoder” 2018. https://en.wikipedia.org/wiki/Binary_decoder

[2] Alexander, “Creeping Line”. 2018 http://ati.ttu.ee/~alsu/DE_Creeping.pdf

[3] DILIGENT “Nexys4 DDR™ FPGA Board Reference Manual”. 2018

http://ati.ttu.ee/~alsu/DE_Nexys4DDR.pdf

7. Appendix

Link to a video file recorded during testing as well as link to the files is given below:

Video link:

https://drive.google.com/drive/folders/1pqELKAaofl9MMEYs5svuwAbpgYZ0kUc1?usp=sharing