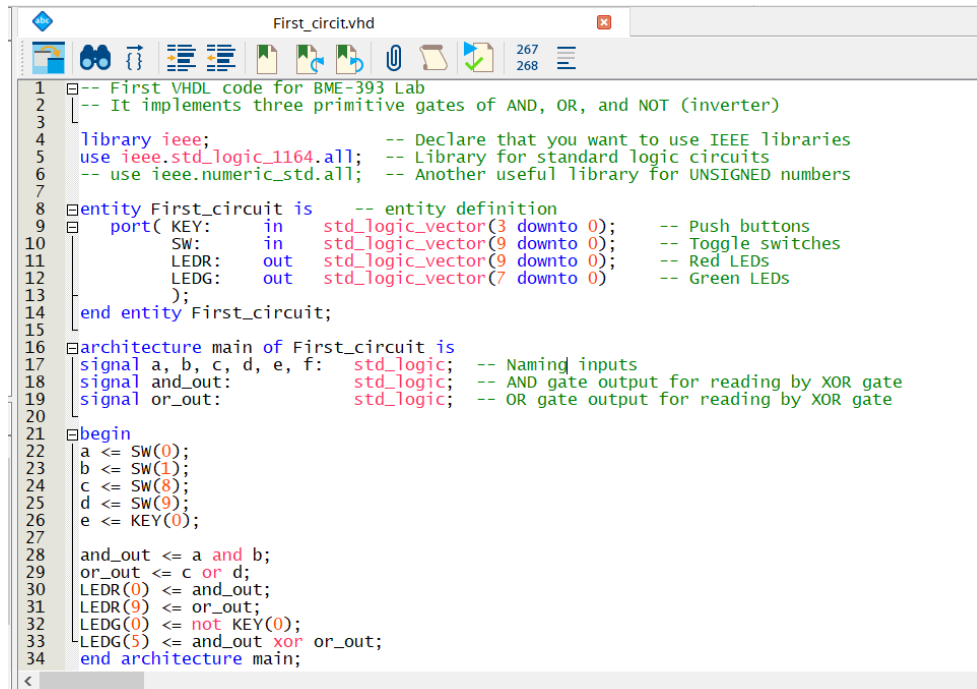


BME 393L: Lab 1

4.1. Simulation of a VHDL code

First, the VHDL code provided to us was used to verify that the Quartus Prime software and the necessary setup procedures have been done correctly. This was done by compiling the code.



```
1 -- First VHDL code for BME-393 Lab
2 -- It implements three primitive gates of AND, OR, and NOT (inverter)
3
4 library ieee;           -- Declare that you want to use IEEE libraries
5 use ieee.std_logic_1164.all; -- Library for standard logic circuits
6 -- use ieee.numeric_std.all; -- Another useful library for UNSIGNED numbers
7
8 entity First_circuit is  -- entity definition
9 port(
10     KEY: in std_logic_vector(3 downto 0); -- Push buttons
11     SW: in std_logic_vector(9 downto 0); -- Toggle switches
12     LEDR: out std_logic_vector(9 downto 0); -- Red LEDs
13     LEDG: out std_logic_vector(7 downto 0) -- Green LEDs
14 );
15 end entity First_circuit;
16
17 architecture main of First_circuit is
18     signal a, b, c, d, e, f: std_logic; -- Naming inputs
19     signal and_out: std_logic; -- AND gate output for reading by XOR gate
20     signal or_out: std_logic; -- OR gate output for reading by XOR gate
21
22 begin
23     a <= SW(0);
24     b <= SW(1);
25     c <= SW(8);
26     d <= SW(9);
27     e <= KEY(0);
28
29     and_out <= a and b;
30     or_out <= c or d;
31     LEDR(0) <= and_out;
32     LEDR(9) <= or_out;
33     LEDG(0) <= not KEY(0);
34     LEDG(5) <= and_out xor or_out;
35 end architecture main;
```

Figure 1: The First_circuit VHDL script used to verify setup and produce the output waveform

Next, a simulation was done to verify the obtained output with the output given in the lab procedure document. The inputs, which were predetermined, were: square wave input on SW[0] with a period of 10.0 ns, 0 ns offset, and 50% duty cycle, and the same for SW[1] (except with a period of 20.0 ns). The output obtained on LEDR[0], LEDR[9], LEDG[0], and LEDG[5], were as expected.

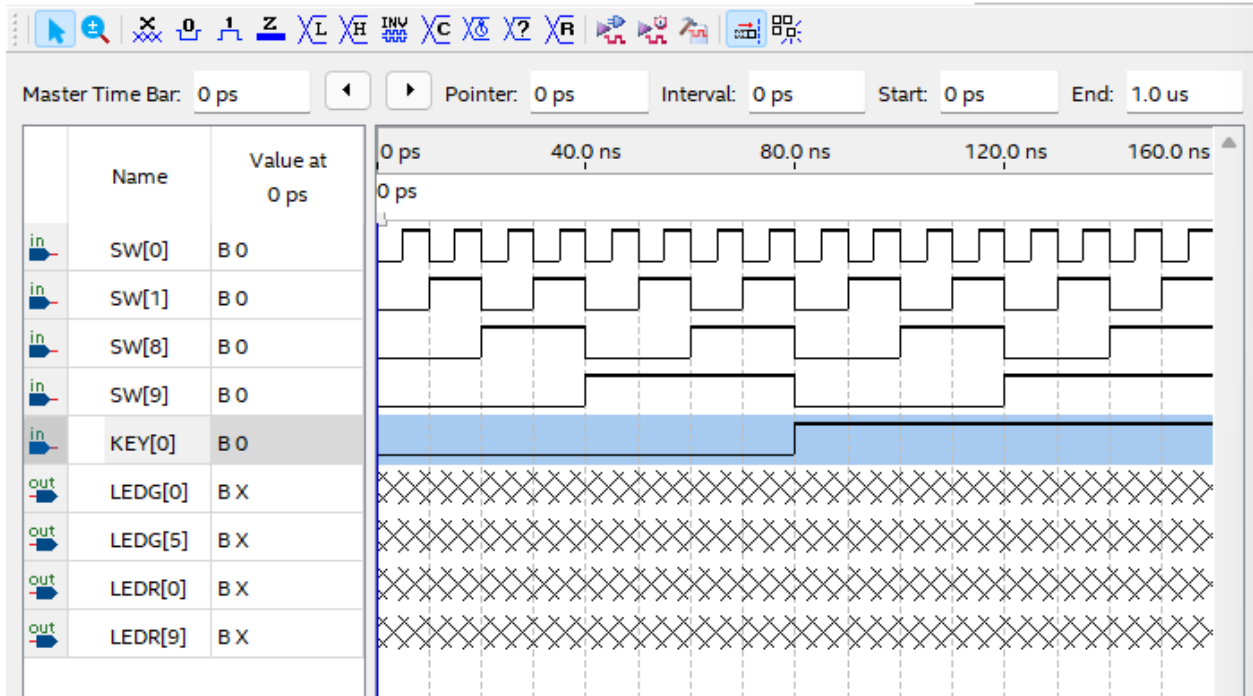


Figure 2: Input Waveform of First_circuit VHDL script

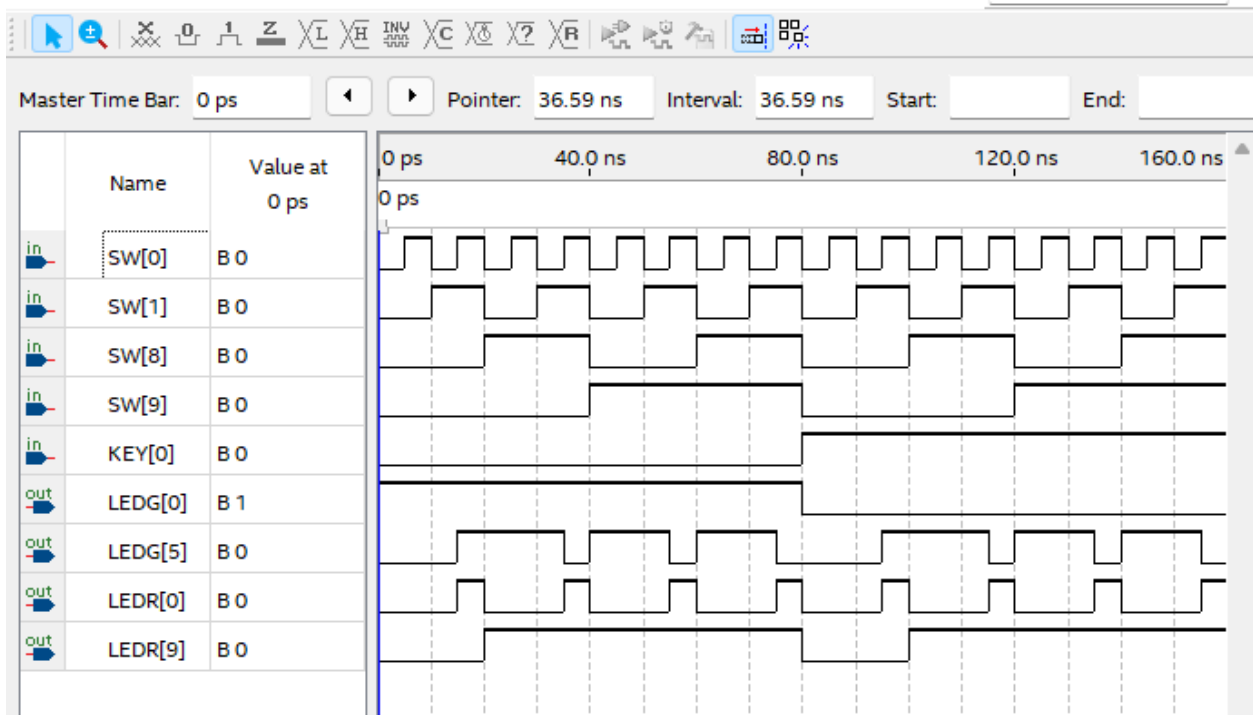


Figure 3: Output Waveform of First_circuit VHDL script (Result)

The output of the VHDL code simulation was then reformatted into a truth table (Table 1).

Table 1: Truth table for the First_circuit VHDL script

Input					Output			
SW[0]	SW[1]	SW[8]	SW[9]	KEY[0]	LEDG[0]	LEDG[5]	LEDR[0]	LEDR[9]
0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	1	1	0
0	0	1	0	0	1	1	0	1
1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	1	0	1
1	1	1	0	0	1	0	1	1
0	0	0	1	0	1	1	0	1
1	0	0	1	0	1	1	0	1
0	1	0	1	0	1	1	0	1
1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	0	1
1	1	1	1	0	1	0	1	1
0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
1	1	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1

1	1	1	0	1	0	0	1	1
0	0	0	1	1	0	1	0	1
1	0	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0	1
1	1	0	1	1	0	0	1	1
0	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	0	1
0	1	1	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1

The next step was to verify that the order of the lines of code within the body of the architecture has no effect on the behaviour of the circuit, as seen in Figure 4. The simulation with the same inputs, as shown in Figure 2, was run again. The input and output of the simulation, which can be seen in Figures 5 and 6, did not change, which was as expected.

```

1  -- First VHDL code for BME-393 Lab
2  -- It implements three primitive gates of AND, OR, and NOT (inverter)
3
4  library ieee;           -- Declare that you want to use IEEE libraries
5  use ieee.std_logic_1164.all; -- Library for standard logic circuits
6  -- use ieee.numeric_std.all; -- Another useful library for UNSIGNED numbers
7
8  entity First_circuit is -- entity definition
9      port( KEY:      in  std_logic_vector(3 downto 0); -- Push buttons
10          SW:       in  std_logic_vector(9 downto 0); -- Toggle switches
11          LEDR:    out std_logic_vector(9 downto 0); -- Red LEDs
12          LEDG:    out std_logic_vector(7 downto 0); -- Green LEDs
13      );
14  end entity First_circuit;
15
16  architecture main of First_circuit is
17      signal a, b, c, d, e, f: std_logic; -- Naming inputs
18      signal and_out:         std_logic; -- AND gate output for reading by XOR gate
19      signal or_out:          std_logic; -- OR gate output for reading by XOR gate
20
21  begin
22      a <= SW(0);
23      c <= SW(8);
24      d <= SW(9);
25      e <= KEY(0);
26      b <= SW(1);
27
28      LEDG(0) <= not KEY(0);
29      LEDR(0) <= and_out;
30      LEDG(5) <= and_out xor or_out;
31      and_out <= a and b;
32      or_out <= c or d;
33      LEDR(9) <= or_out;
34  end architecture main;

```

Figure 4: Scrambled the body of the architecture within First_circuit VHDL script

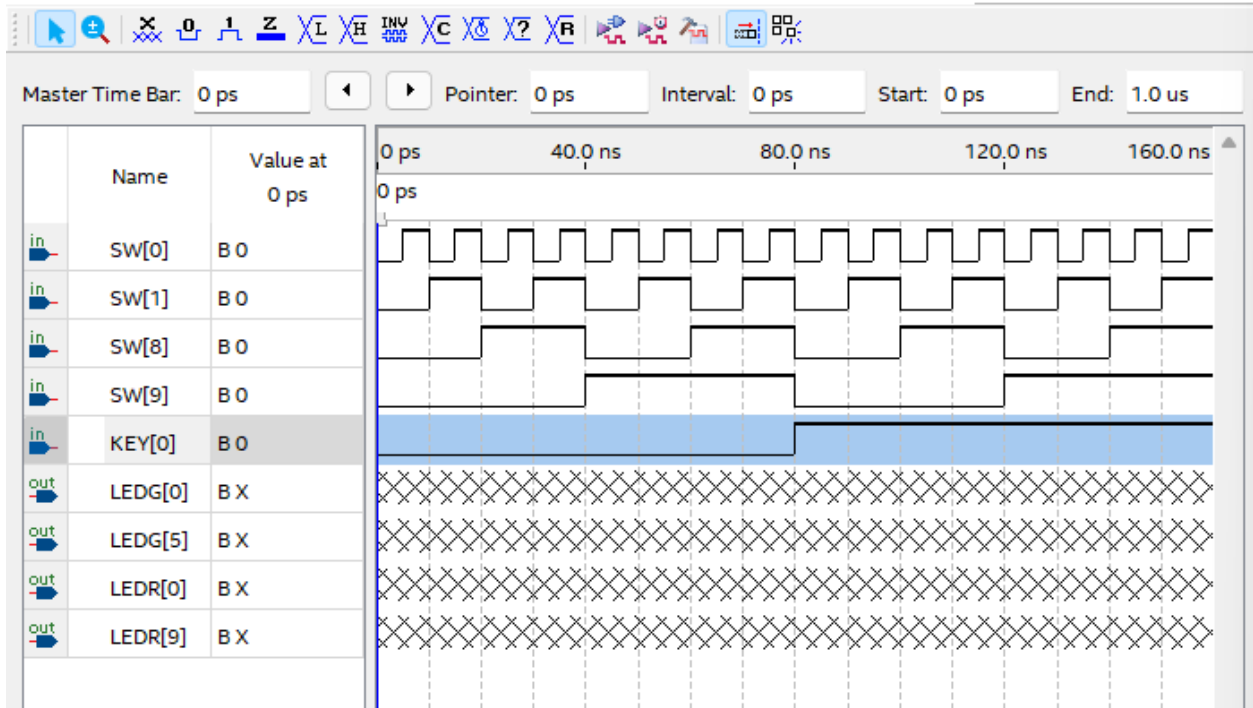


Figure 5: Input Waveform of the scrambled First_circuit VHDL script

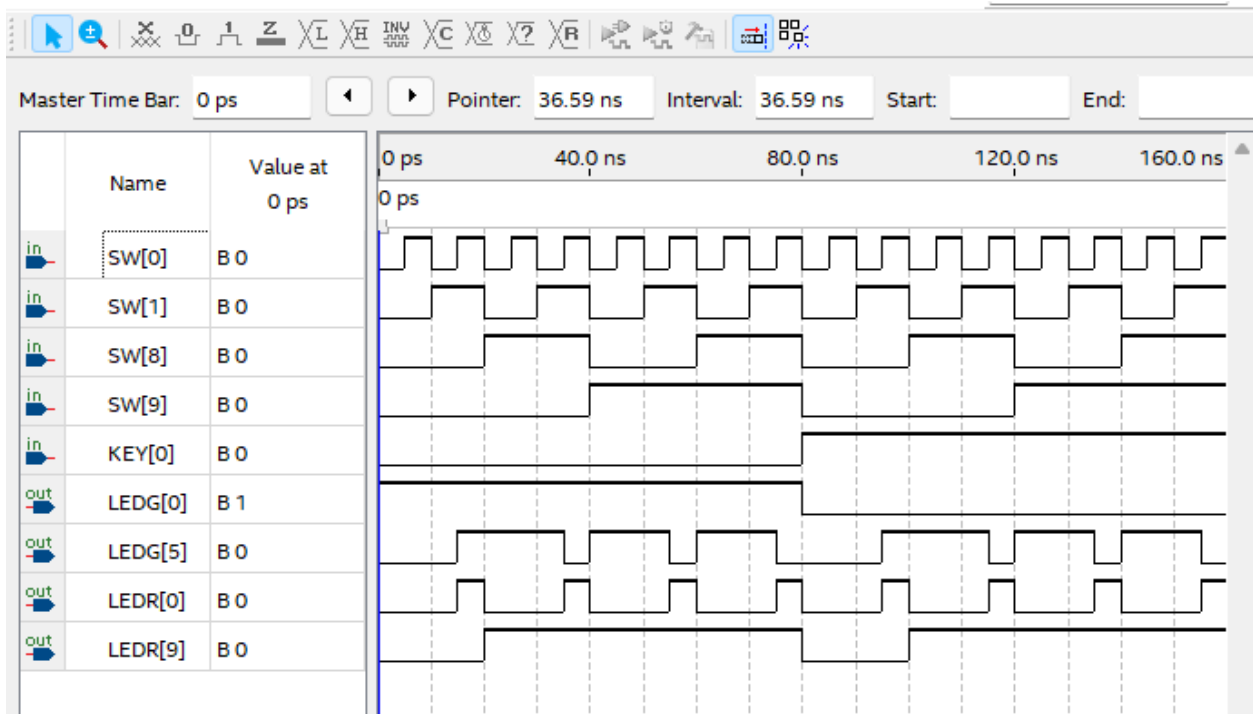


Figure 6: Output Waveform of the scrambled First_circuit VHDL script (Result)

The output of the scrambled VHDL code simulation was then reformatted into a truth table (Table 2).

Table 2: Truth table for the scrambled First_circuit VHDL script

Input					Output			
SW[0]	SW[1]	SW[8]	SW[9]	KEY[0]	LEDG[0]	LEDG[5]	LEDR[0]	LEDR[9]
0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	1	1	0
0	0	1	0	0	1	1	0	1
1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	1	0	1
1	1	1	0	0	1	0	1	1
0	0	0	1	0	1	1	0	1
1	0	0	1	0	1	1	0	1
0	1	0	1	0	1	1	0	1
1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	0	1
1	1	1	1	0	1	0	1	1
0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
1	1	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1

1	1	1	0	1	0	0	1	1
0	0	0	1	1	0	1	0	1
1	0	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0	1
1	1	0	1	1	0	0	1	1
0	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	0	1
0	1	1	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1

Comparing the results from Table 1 and Table 2, it is evident that scrambling the order of the lines of code within the body of the architecture has no effect on the output.

4.2. Schematic entry in digital circuits

The next step was to design the circuit behind the VHDL code shown in Figure 1 using a different method: schematic entry. The final schematic design can be seen in Figure 7. Additionally, a simulation was done, using the same inputs (Figure 8) as the VHDL code method, to verify that the schematic entry was done properly. The results, which can be seen in Figure 9, were as expected: the same as the simulation results of the VHDL code.

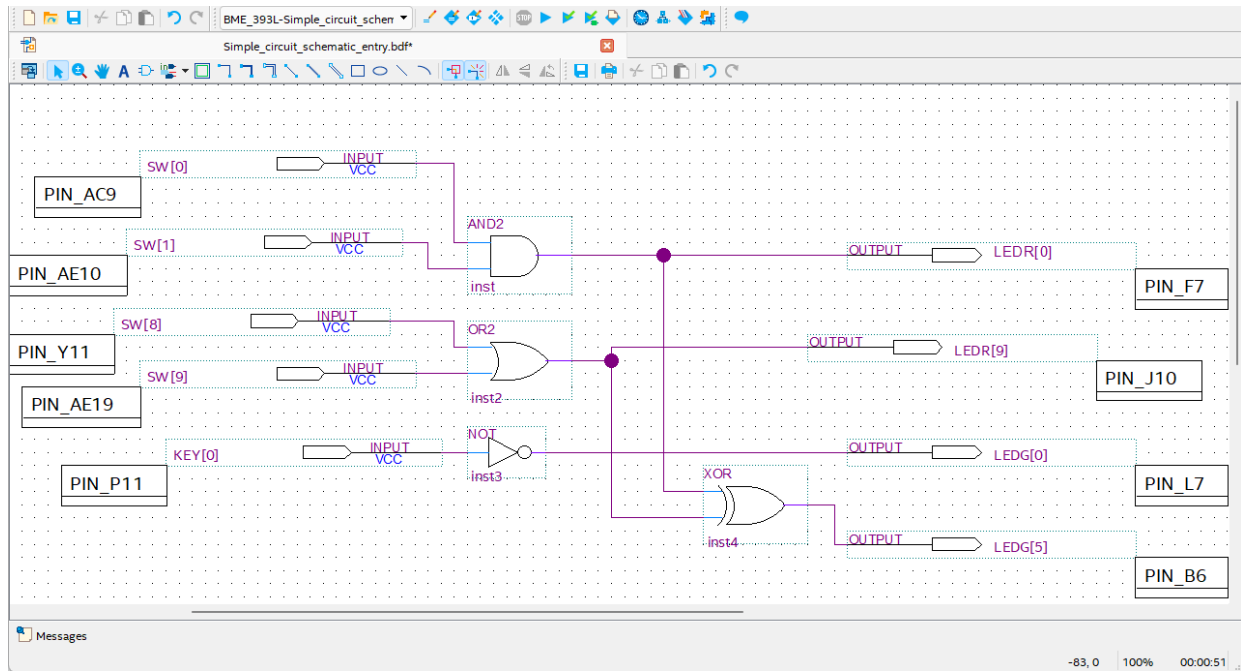


Figure 7: Figure 4-1 schematic used to produce the output waveform

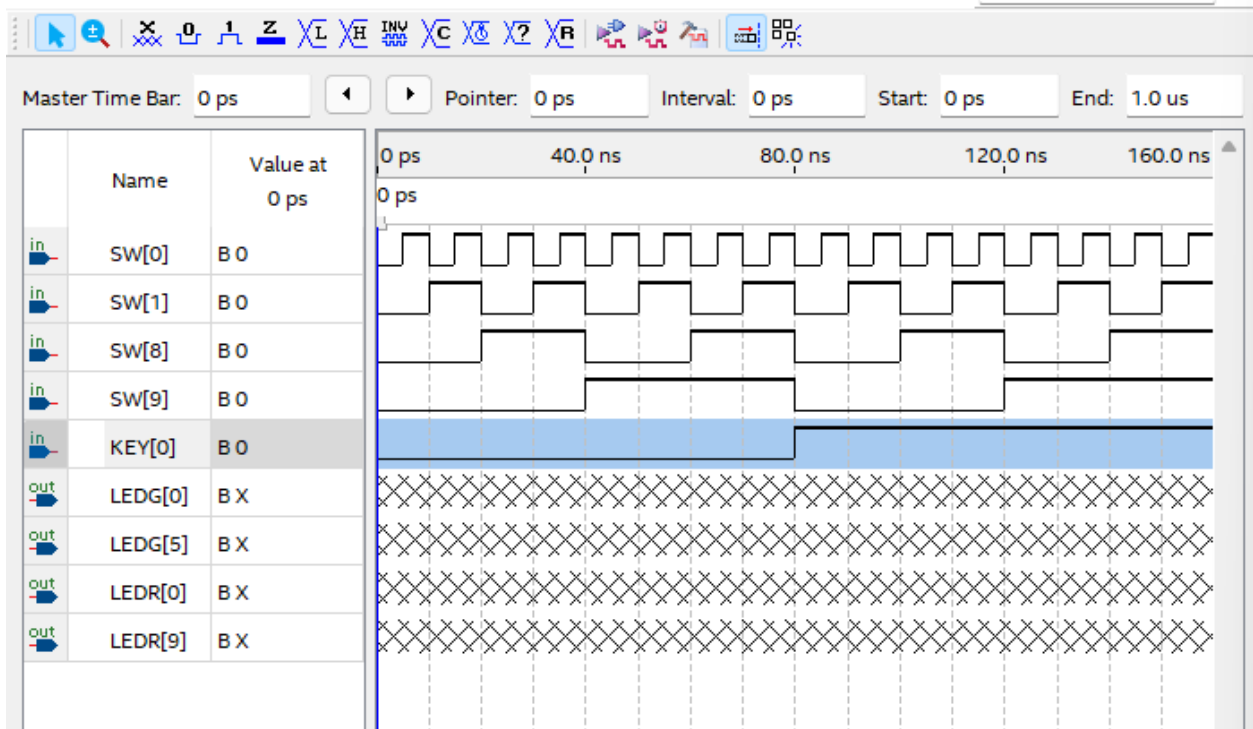


Figure 8: Input Waveform of the Figure 4-1 schematic

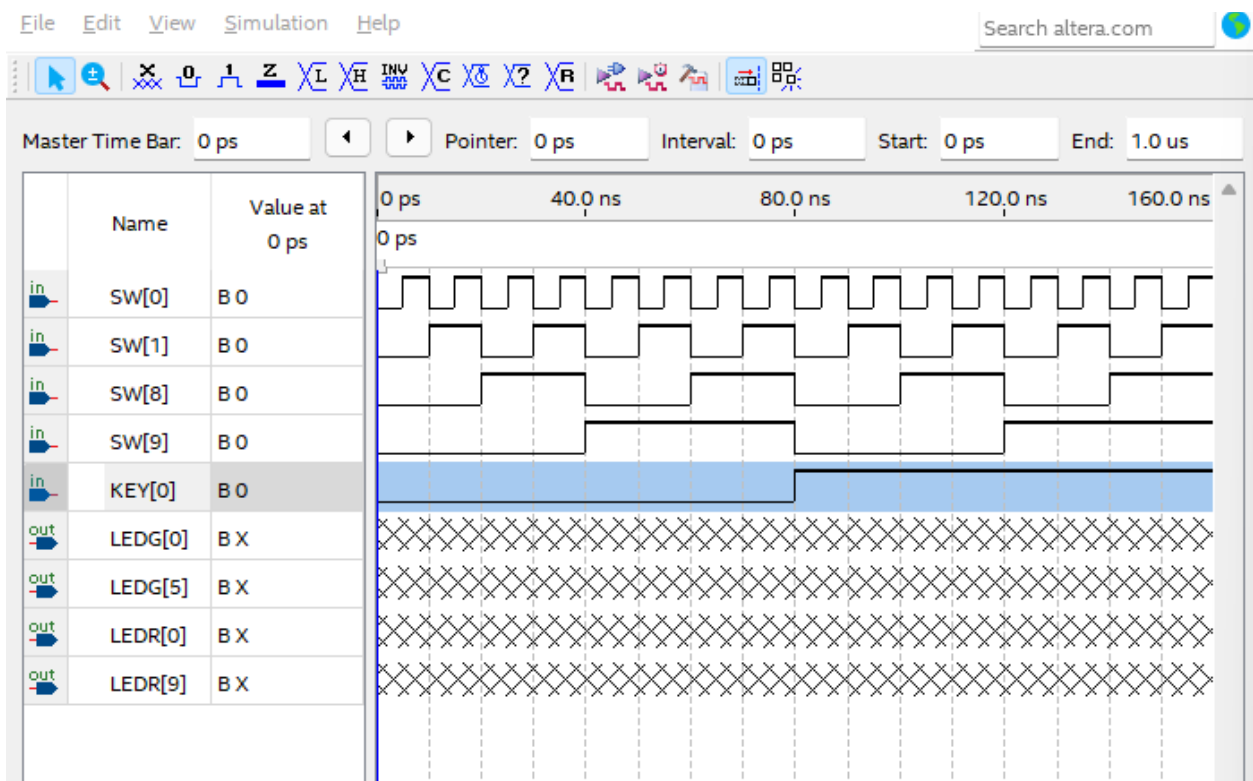


Figure 9: Output waveform of the Figure 4-1 schematic (Result)

The output of the schematic entry simulation was then reformatted into a truth table (Table 3).

Table 3: Truth table for the Figure 4-1 schematic

Input					Output			
SW[0]	SW[1]	SW[8]	SW[9]	KEY[0]	LEDG[0]	LEDG[5]	LEDR[0]	LEDR[9]
0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	1	1	0
0	0	1	0	0	1	1	0	1
1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	1	0	1
1	1	1	0	0	1	0	1	1
0	0	0	1	0	1	1	0	1
1	0	0	1	0	1	1	0	1
0	1	0	1	0	1	1	0	1
1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	0	1
1	1	1	1	0	1	0	1	1
0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
1	1	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1

1	1	1	0	1	0	0	1	1
0	0	0	1	1	0	1	0	1
1	0	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0	1
1	1	0	1	1	0	0	1	1
0	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	0	1
0	1	1	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1

Compare your truth-table from the two sections of 4.1 and 4.2. Do you see any difference between the two outputs?

The truth tables from section 4.1 (Table 1 & Table 2) and section 4.2 (Table 3) all showed the same results. There was no difference between the two outputs because the First_circuit (original and scrambled) script and the Figure 4-1 schematic were both describing the same circuit.

5. Lab Report

VHDL Method

The circuit shown in Figure 5.1 in the lab document is modelled using VHDL code, which can be seen in Figure 10 below.

```
library ieee;                                -- Declare that you want to
use IEEE libraries
use ieee.std_logic_1164.all; -- Library for standard logic circuits
-- use ieee.numeric_std.all; -- Another useful library for UNSIGNED
numbers

entity First_circuit is                      -- entity definition
    port( SW: in std_logic_vector(3 downto 0);    -- Toggle switches
          LEDG: out std_logic_vector(2 downto 0)    -- Green LEDs
    );
end entity First_circuit;

architecture main of First_circuit is
    signal sw_input: std_logic_vector(3 downto 0); -- Naming inputs
    signal intermediate: std_logic_vector(3 downto 0); -- Naming inputs
    signal first_bit_out: std_logic; -- output first bit -> value * 2^0
    signal second_bit_out: std_logic; -- output second bit -> value * 2^1
    signal third_bit_out: std_logic; -- output third bit -> value * 2^2

begin
    sw_input(0) <= SW(0);
    sw_input(1) <= SW(1);
    sw_input(2) <= SW(2);
    sw_input(3) <= SW(3);

    intermediate(0) <= sw(0) and sw(2);
    intermediate(1) <= sw(1) xor sw(3);
    intermediate(2) <= sw(1) and sw(3);
    intermediate(3) <= intermediate(0) and intermediate(1);

    first_bit_out <= sw(0) xor sw(2);
    second_bit_out <= intermediate(0) xor intermediate(1);
    third_bit_out <= intermediate(2) or intermediate(3);

    LEDG(0) <= first_bit_out;
```

```

LEDG(1) <= second_bit_out;
LEDG(2) <= third_bit_out;
end architecture main;

```

```

1  |library ieee;           -- Declare that you want to use IEEE libraries
2  |use ieee.std_logic_1164.all; -- Library for standard logic circuits
3  |-- use ieee.numeric_std.all; -- Another useful library for UNSIGNED numbers
4
5  |entity First_circuit is  -- entity definition
6  |    port( SW:           in  std_logic_vector(3 downto 0);  -- Toggle switches
7  |          LEDG:         out std_logic_vector(2 downto 0)    -- Green LEDs
8  |        );
9  |end entity First_circuit;
10
11 |architecture main of First_circuit is
12 |    signal sw_input: std_logic_vector(3 downto 0); -- Naming inputs
13 |    signal intermediate: std_logic_vector(3 downto 0); -- Naming inputs
14 |    signal first_bit_out: std_logic; -- output first bit -> value * 2^0
15 |    signal second_bit_out: std_logic; -- output second bit -> value * 2^1
16 |    signal third_bit_out: std_logic; -- output third bit -> value * 2^2
17
18 |begin
19 |    sw_input(0) <= SW(0);
20 |    sw_input(1) <= SW(1);
21 |    sw_input(2) <= SW(2);
22 |    sw_input(3) <= SW(3);
23
24 |    intermediate(0) <= sw_input(0) and sw_input(2);
25 |    intermediate(1) <= sw_input(1) xor sw_input(3);
26 |    intermediate(2) <= sw_input(1) and sw_input(3);
27 |    intermediate(3) <= intermediate(0) and intermediate(1);
28
29 |    first_bit_out <= sw_input(0) xor sw_input(2);
30 |    second_bit_out <= intermediate(0) xor intermediate(1);
31 |    third_bit_out <= intermediate(2) or intermediate(3);
32
33 |    LEDG(0) <= first_bit_out;
34 |    LEDG(1) <= second_bit_out;
35 |    LEDG(2) <= third_bit_out;
36 |end architecture main;

```

Figure 10: VHDL script for the Postlab Circuit

A simulation was then run with all potential input sets. The input and output of the simulation can be seen in Figure 11.

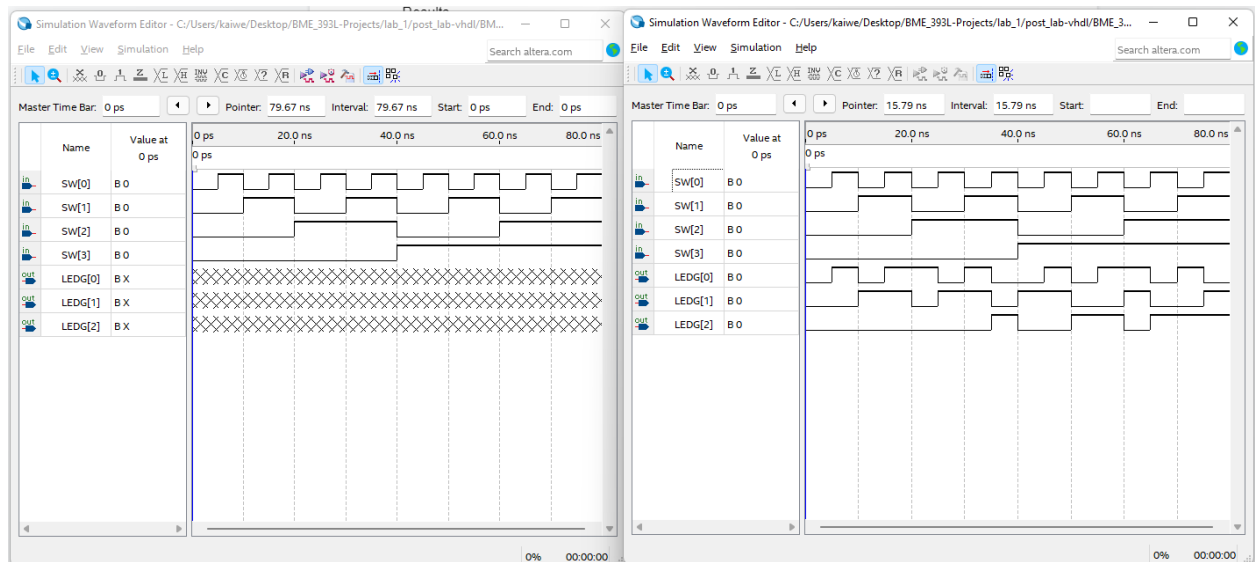


Figure 11: Input and Output Waveforms for the Postlab Circuit VHDL script (Result)

The results of the simulation have been reformatted into a truth table, as shown in Table 4.

Table 4: Truth table for the Postlab Circuit VHDL script

Input				Output		
SW[3]	SW[2]	SW[1]	SW[0]	LEDG[2]	LEDG[1]	LEDG[0]
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Schematic Method

The circuit shown in Figure 5.1 in the lab document is modelled using the schematic entry method, which can be seen in Figure 12 below. A simulation was then run with all potential input sets. The input and output of the simulation can be seen in Figure 13.

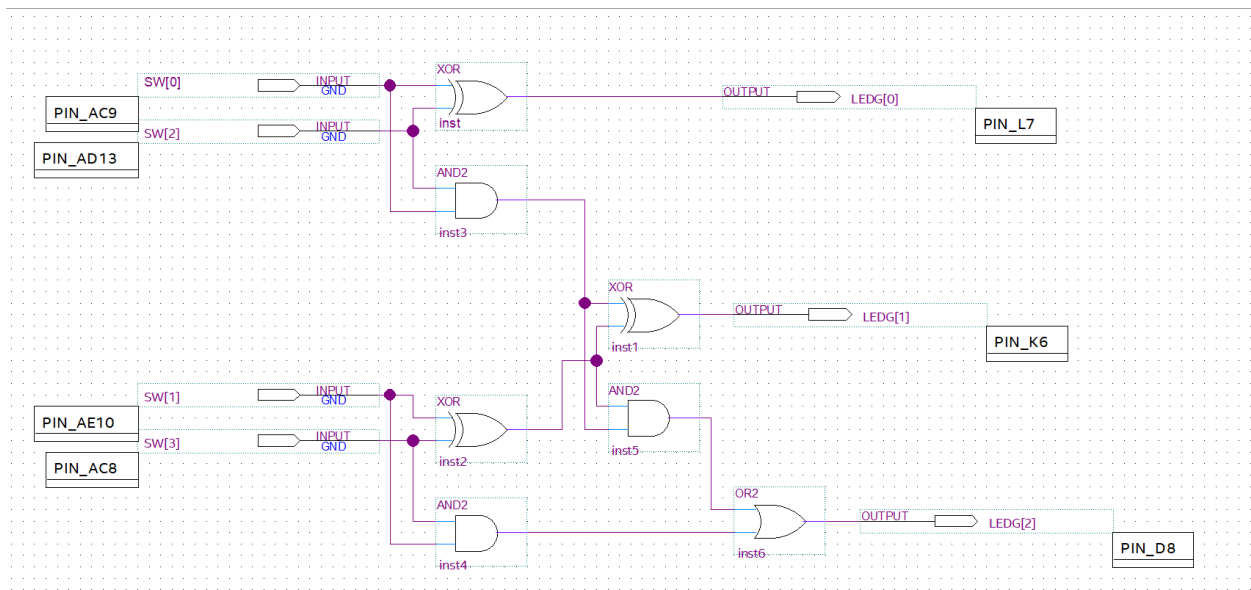


Figure 12: Schematic for the Postlab circuit

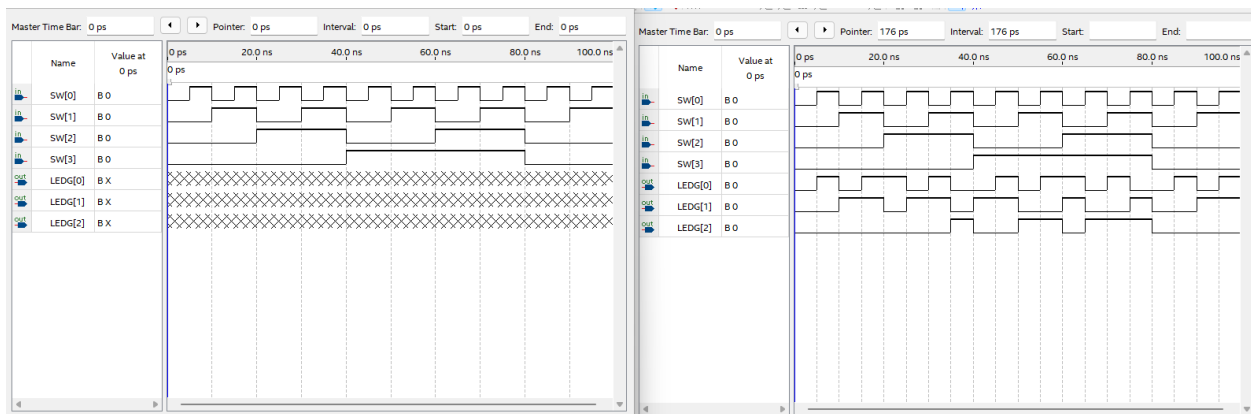


Figure 13: Input and Output Waveforms for the Postlab circuit schematic (Result)

The results of the simulation have been reformatted into a truth table, as shown in Table 5.

Table 5: Truth Table for the Postlab circuit schematic

Input				Output		
SW[3]	SW[2]	SW[1]	SW[0]	LEDG[2]	LEDG[1]	LEDG[0]
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

What is this circuit doing?

This circuit is adding together the binary values of A and B, and storing the result in C.

Input A, which can be thought of as a 2 bit binary number, is represented by the inputs: SW[0] and SW[1]. Here, SW[0] would be the Least Significant Bit (LSB) and SW[1] would be the Most Significant Bit (MSB).

Input B, which can also be thought of as a 2 bit binary number, is represented by the inputs: SW[2] and SW[3]. Here, SW[2] would be the LSB and SW[3] would be the MSB.

The output, C, which would store the result of the sum of A and B, is represented using a 3 bit binary number. Having 3 bits will ensure that any overflow is accounted for. This can be shown by using the range formula for a binary number (that is a whole number): with n being the number of bits, the range of the number would be: $[0, 2^n - 1]$. The range of a 2 bit binary number would then be: $[0, 3]$. The sum of two 2 bit binary numbers would be: $[0, 6]$. Therefore, a 3 bit binary number can be used to represent the sum, as it would have a range of $[0, 7]$. Starting with the MSB and ending with the LSB, the equivalent output would be: LEDG[2], LEDG[1], and LEDG[0].