The Design Methodology:

In this laboratory work, I modelled a logical circuit of laptop's power source. I have four inputs and one output that symbolizes with four switches and one led light on FPGA. The logic algebra of laptop is "working = (electric + (batteryplug . charge)) . switch" So laptop will work if switch is ON and there is electric or battery. In order to battery operates laptop, it must be charged and plugged to laptop.

Logic Algebra of Working:

F = (A + (B . C)) . D

A = Power Cable – city electric => Switch 3

B = Battery plugged or not => Switch 2

C = Battery charged or not => Switch 1

D = Laptop switch => Switch 0

F = Laptop working or not => Led 0

Results:

After writing top module code, RTL schematic is correct. So we can continue to write test bench code.

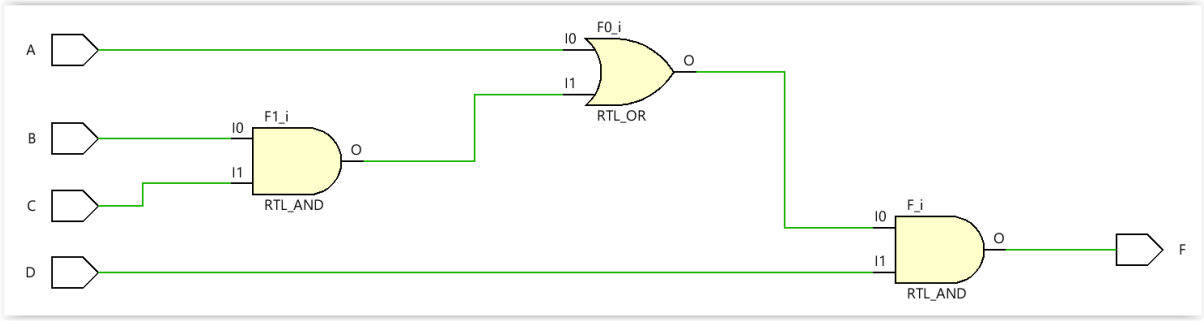

Figure-1 : RTL Schematic

After we write test bench code, we run behavioural simulation and it is same as our truth table. Hence we can continue to program BASYS-3.

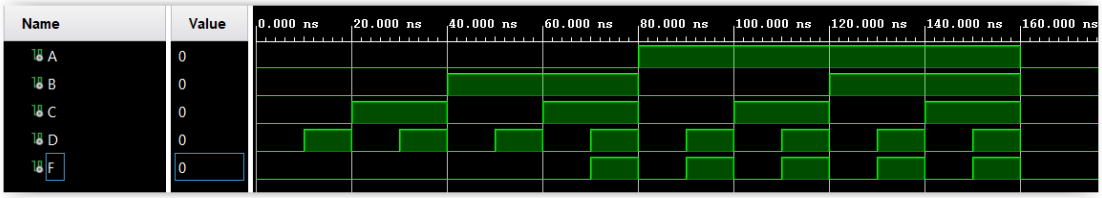

Figure-2 : Waveform of Behavioral Simulations

	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Figure-3 : Truth Table of our Logic Circuit

Running Codes:

After we run synthesis and implementation, it is the time to generate bitstream. Hence, we can generate bitstream, click auto-connect and Program Device. Hence our codes will be transferred to BASYS-3.

Example Photos of Working FPGA:

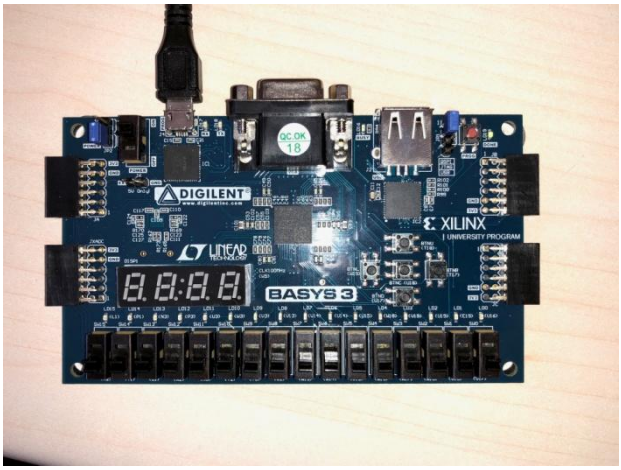

Figure-4 : Switches => 0-0-0-0 / Led => not working

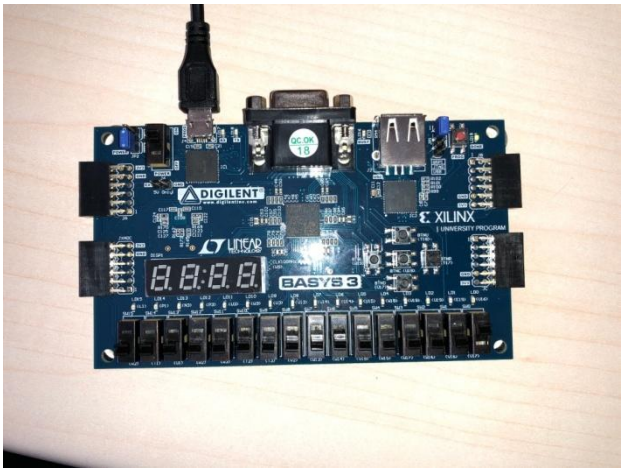

Figure-5 : Switches => 0-0-0-1 / Led => not working

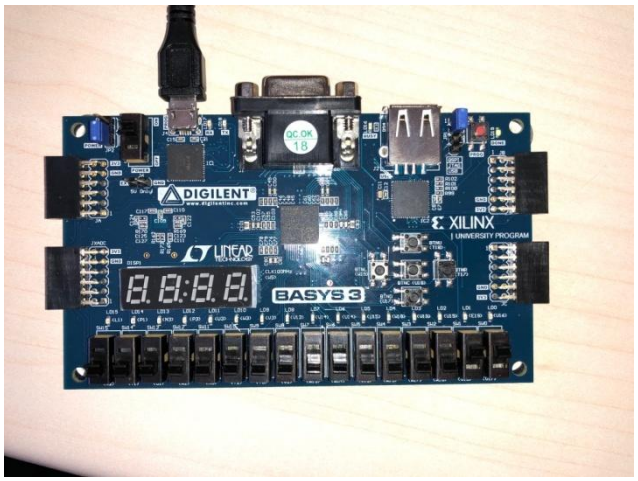

Figure-6 : Switches => 0-0-1-1 / Led => not working

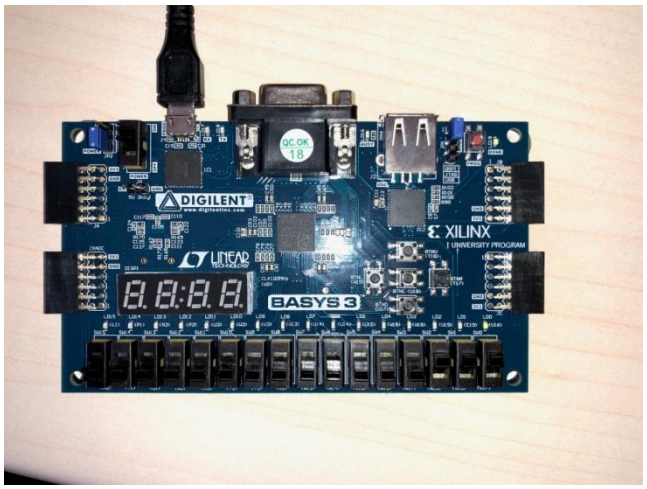

Figure-7 : Switches => 0-1-1-1 / Led => working

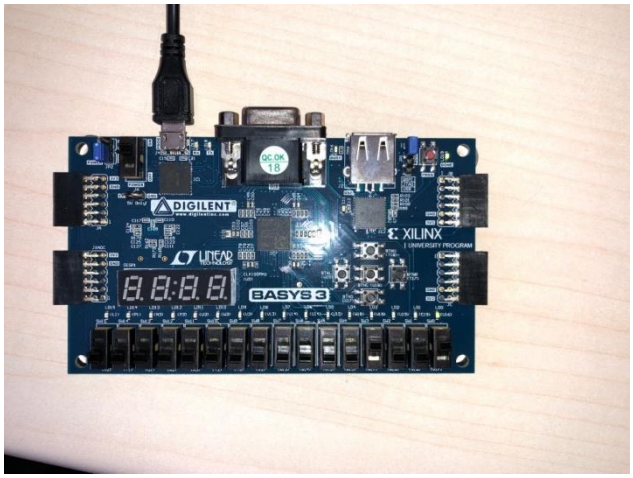

Figure-8 : Switches => 1-0-0-1 / Led => working

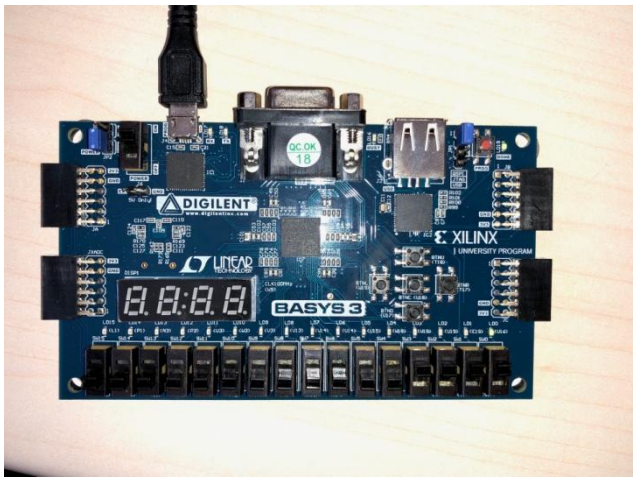

Figure-9 : Switches => 1-1-1-1 / Led => working

Questions:

1- **How does one specify the inputs and outputs of a module in VHDL?**

We can declare the input signals and output signals by using the structure ENTITY. We can use such a code like Port (A : in bit; F : out bit); then our inputs and outputs will be defined. Also we can specify the inputs and outputs when adding sources in vivado, but I do not prefer that way.

2- **How does one use a module inside another code/module? What does PORT MAP do?**

We can use many modules under one main TOP MODULE. With using PORT MAP, we can match inputs and outputs in modules with inputs and outputs in TOP MODULE. Hence, we do not need to use same names for inputs and outputs.

3- What is a constraint file? How does it relate your code to the pins on your FPGA?

Without constraint file, our code means nothing for FPGA. We are using constraints to match inputs and outputs with switches and leds etc. Hence we can see results on FPGA BASYS-3.

4- What is the purpose of writing a testbench?

If we write testbench, we can run Behavioral Simulation and we can simulate the inputs and display the output waveform

Conclusion:

In this lab work, I learned to design basic logical circuit, writing code of basic logical circuit and proggamming FPGA. I also experienced using Vivado program to write VHDL code. Writing code of logical circuit was not hard but writing test bench code was troublesome. I created truth table for test bench with using python code. Before this lab I did not know the difference between BIT and STD_LOGIC. I learned that BIT is kind of old version of STD_LOGIC. Also synthesis and implementation was not familiar to me. I learned about these things. The purpose of synthesis and implementation is converting the code to another that FPGA can read.

Appendices:

1) Design Code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity lab2 is

  Port (A,B,C,D : in bit;

        F : out bit );

end lab2;

architecture Behavioral of lab2 is

begin

F <= (A OR (B AND C) )AND D;

end Behavioral;
```

2) Test Bench Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.all;
```

```
USE ieee.numeric_std.ALL;

entity tb_lab2 is
--  Port ( );
end tb_lab2;
architecture Behavioral of tb_lab2 is
component lab2
port(   A : in bit;
        B : in bit;
        C : in bit;
        D : in bit;
        F : out bit
);
end component;
signal A: bit :='0';
signal B: bit :='0';
signal C: bit :='0';
signal D: bit :='0';
signal F: bit ;

begin

   dut : lab2
      port map (A => A,
                B => B,
                C => C,
                D => D,
                F => F);
   stimuli: process
   begin
      A <= '0';
      B <= '0';
      C <= '0';
      D <= '0';
      wait for 10 ns;
      A <= '0';
      B <= '0';
      C <= '0';
      D <= '1';
      wait for 10 ns;
      A <= '0';
      B <= '0';
      C <= '1';
      D <= '0';
      wait for 10 ns;
```

```
A <= '0';
B <= '0';
C <= '1';
D <= '1';
wait for 10 ns;
A <= '0';
B <= '1';
C <= '0';
D <= '0';
wait for 10 ns;
A <= '0';
B <= '1';
C <= '0';
D <= '1';
wait for 10 ns;
A <= '0';
B <= '1';
C <= '1';
D <= '0';
wait for 10 ns;
A <= '0';
B <= '1';
C <= '1';
D <= '1';
wait for 10 ns;
A <= '1';
B <= '0';
C <= '0';
D <= '0';
wait for 10 ns;
A <= '1';
B <= '0';
C <= '0';
D <= '1';
wait for 10 ns;
A <= '1';
B <= '0';
C <= '1';
D <= '0';
wait for 10 ns;
A <= '1';
B <= '0';
C <= '1';
D <= '1';
wait for 10 ns;
```

```
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '0';
        wait for 10 ns;
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '1';
        wait for 10 ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '0';
        wait for 10 ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '1';
        wait for 10 ns;
        A <= '0';
        B <= '0';
        C <= '0';
        D <= '0';
        wait;
    end process;
end;
```