EEE102 Lab 2: Introduction to VHDL

Lab Report

Nizam Ercan 22302317 Section 2

1- Purpose:

The purpose of this lab is to get used to reading and writing VHDL codes and learn to use the Vivado software for designing simple combinational circuits. Also, simulating the design by writing testbench codes and implementing our code to our FPGA board is also aimed in this lab.

2- Methodology:

A file named LAB2_buggy was uploaded to Moodle. We were initially asked to alter some code of a submodule according to our Bilkent student ID. After that we were asked to find and fix 6 bugs inside the code using the Vivado software. After the debugging we were asked to demonstrate the fixed circuit with our Basys3 FPGA. Then we wrote the testbench code for our design and successfully simulated all the inputs sequentially and displayed all the inputs and the outputs in a time diagram. After that, we drew the RTL schematics, schematics under "Synthesized Design" and the schematics under "Implemented Design" and observed their differences.

We were also asked to answer some simple questions about Vivado and VHDL which are answered below:

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

In VHDL, we can specify inputs and outputs in a module by using entity declaration. We can look at the following example:

```
entity example is
    Port (
        input1 : in  std_logic;
        output1 : out std_logic;
    );
end entity example;
```

In this example, an entity called "example" is defined and one input named "input1" and one output named "output1" is defined in VHDL.

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

In order to use a module inside another module in a hierarchal VHDL code, we need to instantiate the module we want to use by declaring them as a component before the architecture code begins and we can map this component's ports in our top module using a "port map" statement as shown in this example:

```
entity TopModule is
    Port (
        input1_top : in  std_logic;
        output1_top : out std_logic;);
end entity TopModule;
```

```
architecture Behavioral of TopModule is

    component example is
        Port (
            input1 : in  std_logic;
            output1 : out std_logic;);
    end component example;

    signal internal_sig1 : std_logic;
    signal internal_sig2 : std_logic;

begin

    U1: example port map (
        input1 => internal_sig1,
        output1 => internal_sig2);

    input1_top <= internal_sig1;
    output1_top <= internal_sig2;

end architecture Behavioral;
```

In this example, we firstly defined the entity "TopModule" with its inputs and outputs and we declared a component "example" inside the architecture of our top module "TopModule" (shown with red). We defined and used the signals "internal_sig1" and "internal_sig2" in order to connect the outputs of "example" to the outputs of "TopModule" (shown with blue). We then instantiate "example" within "TopModule" using the component declaration and map its ports using the port map statement (shown with green). Finally, we connect the internal signals to the output ports of our top module (shown with black).

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

A constraint file connects my VHDL code to the physical implementation on my Basys3 FPGA such as switches and LEDs. Without any constraint file, neither Vivado nor my Basys3 can know which ports in my VHDL code will be assigned to the physical pins on the FPGA board. We can generate a constraint file by either writing it by our hand manually entering all the ports or by using the I/O Ports menu on the "Elaborated Design" section in RTL analysis section in Vivado.

4- What is the purpose of writing a testbench?

Writing a testbench code helps us to simulate our design before implementing it to our FPGA. A testbench code allows us to verify the and test the functions of our design by checking whether our design gives the correct outputs for the given inputs. We need to simulate our designs since synthesizing and implementing our design into a real FPGA can take a lot of resources and time. Therefore, making sure that our design works correctly is very important. We can use a testbench also for debugging.

3- Results:

When I first opened the "LAB2_buggy" document in Vivado, I encountered a critical error message as you can see in Figure 1. This error means that this project is made with a board other than my Basys3. In order to solve this bug, I go to the "Settings" under the "Project Manager" section which is located at the

top left of the screen and changed the project device from “xc7a12ticsg325-1L” to “Basys3 (xc7a35tcpg236-1)” which made the first bug disappear.

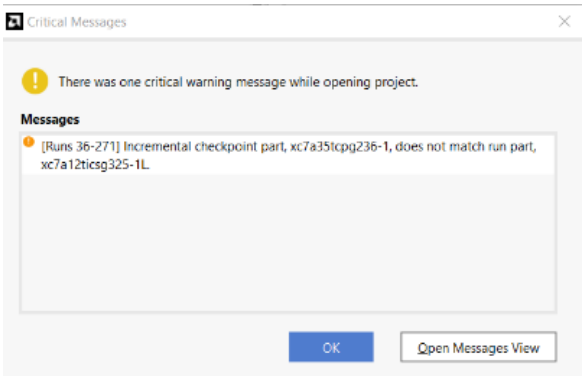

Figure 1: The initial error message

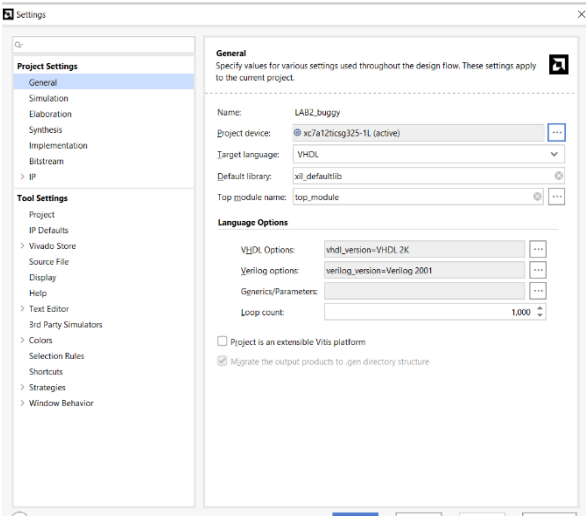

Figure 2: Wrong project device is selected

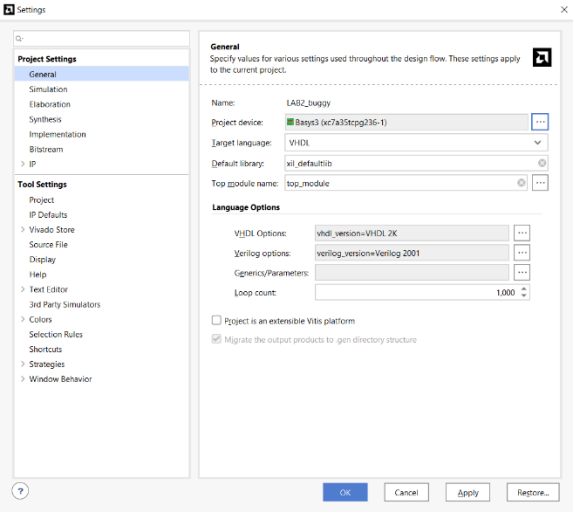

Figure 3: Project device is changed to Basys3

After fixing the bug about the project device, I changed the code of submodule1.vhd according to my Bilkent ID no 22302317. To do that I opened submodule1.vhd from the sources section of Vivado and changed the logic gate “and” in line 14 to “xor” as you can see in Figure 4 and 5 which is the first task in the lab manual.

```
12
13  begin
14      o_output_byte(0)          <= i_input_byte(0) and i_input_byte(1);
15      o_output_byte(1)          <= i_input_byte(2) or i_input_byte(3);
16      o_output_byte(2)          <= i_input_byte(4) xor i_input_byte(5);
17      o_output_byte(5 downto 3) <= "010";
18      o_output_byte(7 downto 6) <= (others => '0');
19
```

Figure 4: Initial logic gates

```
13  begin
14      o_output_byte(0)          <= i_input_byte(0) xor i_input_byte(1);
15      o_output_byte(1)          <= i_input_byte(2) or i_input_byte(3);
16      o_output_byte(2)          <= i_input_byte(4) xor i_input_byte(5);
17      o_output_byte(5 downto 3) <= "010";
18      o_output_byte(7 downto 6) <= (others => '0');
```

Figure 5: Altered logic gates according to ID no

Then, I tried to synthesize the code given in the file to see whether it can be converted into a netlist of hardware components that can be implemented on a real FPGA board. However, Vivado gave me 3 error messages which can be seen on Figure 6 which are:

1- [Synth 8-36] 's_ouput_1' is not declared
2- [Synth 8-9114] actual of formal out port 'o_output_byte' cannot be an expression
3- [Common 17-69] Command failed: Synthesis failed - please see the console or run log file for details

Next to the first two error messages, Vivado guides me to the location of the sources of the errors which are shown in lines 36 and 40 respectively in Figure 7. In line 36 a typo is made when mapping the internal signal s_output_1 to the output of sub_module1 component, s_ouput_1 is written instead of s_output_1. This is the second bug I fixed as you can see in Figure 8. When I looked at line 40, I saw that the explicit form of writing port map by using "=>", which is highly recommended to be used by Vivado, is not used. This resulted with an error with the inputs and the outputs being switched by "i_SW" port which is an input being connected to "o_output_vector" which is an output. We can fix this bug by explicitly showing which port of the component is being connected to which port or internal signal by using the "=>" operator. This fixes the third bug as you can see in Figure 8.

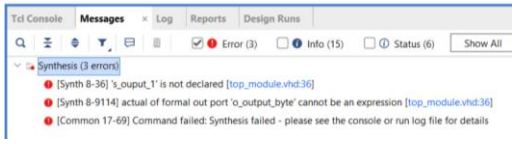

Figure 6: The error messages

```
33    sub_module1_2 : sub_module1
34        port map (
35            i_input_byte  => i_SW,
36            o_output_byte => s_ouput_1
37        );
38
39    sub_module2_1 : sub_module2
40        port map (s_output_2, i_SW);
41    s_output_3 <= unsigned(s_output_2) + 25;
```

Figure 7: Location of the two errors

```
33    sub_module1_2 : sub_module1
34        port map (
35            i_input_byte  => i_SW,
36            o_output_byte => s_output_1
37        );
38
39    sub_module2_1 : sub_module2
40        port map (i_switch_inputs => i_SW,
41                  o_output_vector => s_output_2);
42    s_output_3 <= unsigned(s_output_2) + 25;
```

Figure 8: Fixed code

After fixing these bugs I could succesfully synthesize my design. However, following the synthesize I couldn't run the implementation and Vivado reported that the implementation failed and gave me the error messages that you can see in Figure 9.

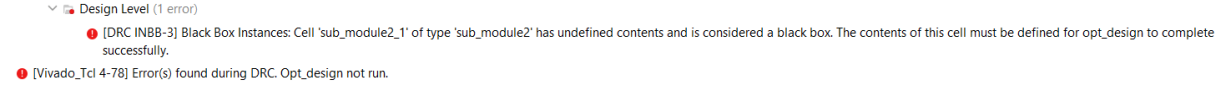

Figure 9: Error messages after the implementation

According to this error message, the cell “sub_module2_1” of type “sub_module2” in “top_module.vhd” is an instance of a black box which means that my code couldn’t access the internal details of “sub_module2”. The reason for this error is that the name of the entity we defined in “sub_module2.vhd” is not named “sub_module2” but “sub_module2_the_beast”. Therefore when we wrote “sub_module2” in “top_module.vhd” to refer the entity in “sub_module2.vhd” our code couldn’t detect it as you can see in Figure 10. In order to fix this bug, in “top_module.vhd”, I changed all the “sub_module2”s to “sub_module2_the_beast” as you can see in Figure 11 which solves the fourth bug.

```
21      component sub_module2 is
22          port (
23              i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
24              o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
25          );
26      end component sub_module2;
27
28      signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
29      signal s_output_3             : unsigned(7 downto 0)         := (others => '0');
30
31  begin
32
33      sub_module1_2 : sub_module1
34          port map (
35              i_input_byte  => i_SW,
36              o_output_byte => s_output_1
37          );
38
39      sub_module2_1 : sub_module2
40          port map (i_switch_inputs => i_SW,
41                    o_output_vector => s_output_2);
42      s_output_3 <= unsigned(s_output_2) + 25;
```

Figure 10: Wrong name is written for sub_module2

```
21      component sub_module2_the_beast is
22          port (
23              i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
24              o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
25          );
26      end component sub_module2_the_beast;
27
28      signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
29      signal s_output_3             : unsigned(7 downto 0)         := (others => '0');
30
31  begin
32
33      sub_module1_2 : sub_module1
34          port map (
35              i_input_byte  => i_SW,
36              o_output_byte => s_output_1
37          );
38
39      sub_module2_1 : sub_module2_the_beast
40          port map (i_switch_inputs => i_SW,
41                    o_output_vector => s_output_2);
42      s_output_3 <= unsigned(s_output_2) + 25;
```

Figure 11: The correct name is entered

After fixing this bug, I could succesfully synthesize and implement my code. However, this time Vivado gave me an error about the line 28 of my constraint file as you can see in Figure 12. According to this error, unlike other lines the command “set_property PACKAGE_PIN U19 [get_ports {o_LED[2]]” seems to be unrecognized by Vivado. From Figure 13, we can see that a curly bracket is missing in the line and in Figure 14 you can see the fixed code which solves the fifth bug.

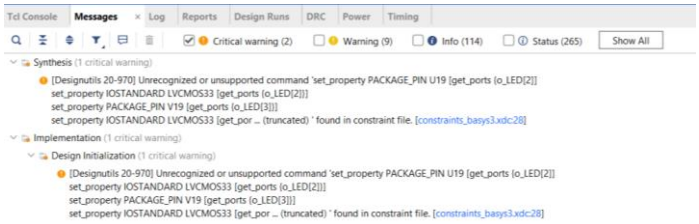

Figure 12: The error message

```
27      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
28  set_property PACKAGE_PIN U19 [get_ports {o_LED[2]]
29      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
```

Figure 13: Line 28 with missing bracket

```
27      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
28  set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]
29      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
```

Figure 14: Fixed line 28

After fixing the bug in the constraint file, I checked the I/O ports to see how my design will be interpreted by my Basys3. When I opened the I/O ports menu I saw that the LED pins 0 and 7 were swapped which you can see in Figure 15. This meant that when I implement my design on my Basys3, it will show the outputs in a different order. I fixed this bug, which is the sixth and the last bug, by swapping back these ports as you can see in Figure 16.

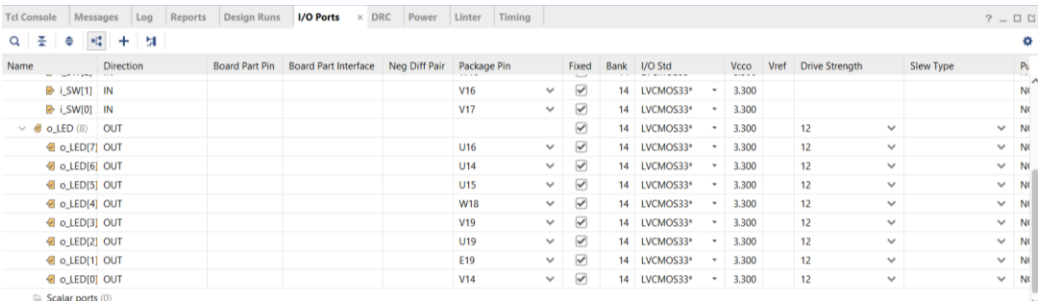

Figure 15: o_LED[0] and [7] are swapped

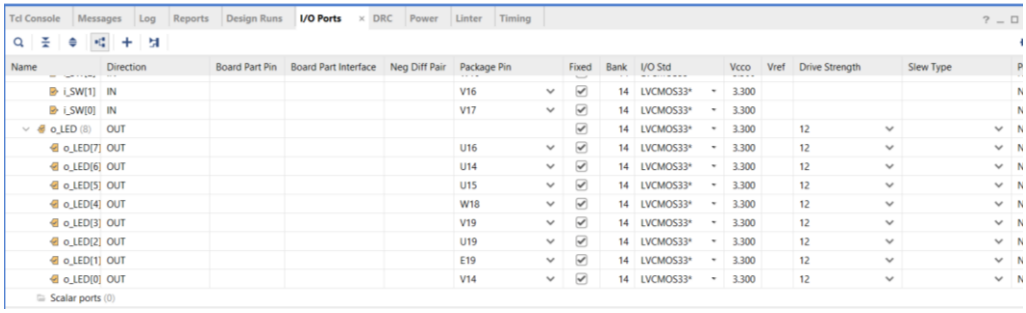

Figure 16: Fixed ports

After fixing all the six bugs in the code, which is the second task, I finally generated the bitstream, connected my Basys3 to my computer and loaded this bitstream to my Basys3 to test my code. Below, you can see 5 examples of my Basys3 running up the code.

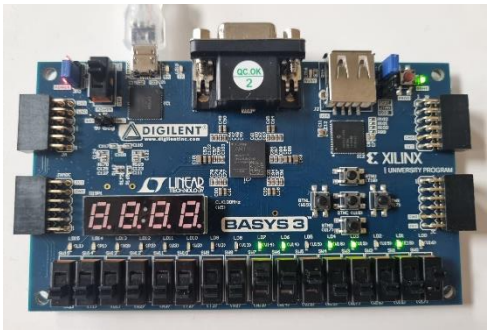

Figure 17: The output of input "11100001"

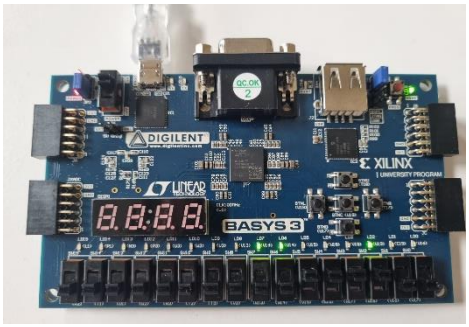

Figure 18: The output of input "00111100"

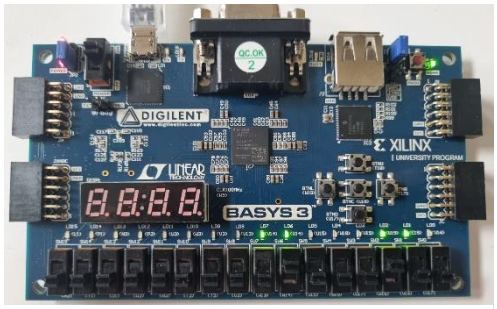

Figure 19: The output of input "01000100"

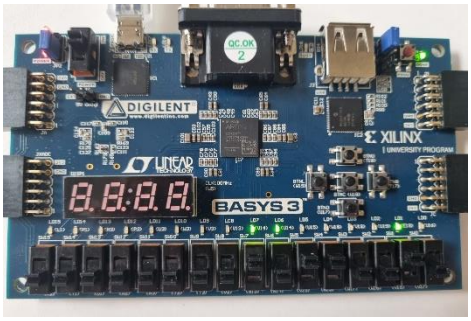

Figure 20: The output of input "10110010"

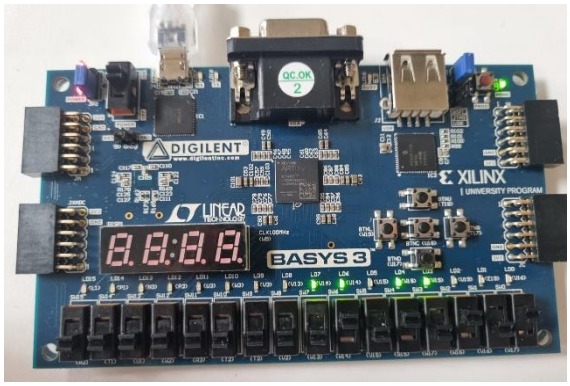

Figure 21: The output of input "01010101"

After demonstrating my circuit on my Basys3, which is the third task, I wrote a testbench code so that I can simulate all the inputs and the outputs of my circuit and compare it to the output LEDs of my Basys3. This makes sure that the demonstration on my Basys3 works as intended. In order to write the testbench code, I added a simulation source named "Lab2_testbench.vhd" to the LAB2_buggy file and I added my top module "top_module.vhd" as a component of my testbench. Then I defined two internal signals "input_tb" and "output_tb" so that I can map the ports of "top_module.vhd". Then I used the reserved word "port map" to connect the inputs and the outputs of "top_module.vhd" to my testbench. Instead of writing all 256 potential inputs, I used a for loop from 0 to 255 to simulate all the inputs. I turned the values from the for loop into an unsigned with 8 digits by using "to_unsigned(i,8)" then converted it to an std_logic_vector by using "std_logic_vector(to_unsigned(i,8))". Then I ended the for loop and the process which finished the testbench code. Finally, I ran my simulation which you can see in Figure 22.

Figure 22: My simulation for all the possible inputs

After that I used my simulation to check that the demonstration of my circuit on Basys3 is correct. In Table 1, you can see that the 5 example inputs I choose, the outputs from my Basys3 and the outputs according to my simulation. As you can see, all the outputs match with each other which indicates that my design is working as intended which is what the task 4 asks from us.

Example Inputs	Output from Basys3	Output according to the simulation
11100001	11011010	11011010
00111100	11000100	11000100
01000100	11000110	11000110
10110010	11000010	11000010
01010101	11011000	11011000

Table 1: Comparison of the outputs of Basys3 and the Simulation

After completing task 4, I started task 5 which asks me to draw 3 different design schematics and compare them. From Figure 23, 24 and 25, you can see RTL design, synthesized design, and the implemented design respectively. RTL design is the initial phase of the design where the system is described at a higher level without considering the actual physical implementation of the system. During the synthesis, the RTL design is transformed into a netlist representation that consist of lower-level logic gates and the synthesized design shows our RTL design analyzed, optimized and mapped onto specific hardware resources. In implemented design, the synthesized design is further processed to show the physical implementation of my design which includes placement and routing. The implementation process also involves mapping my synthesized netlist onto Basys3's resources.

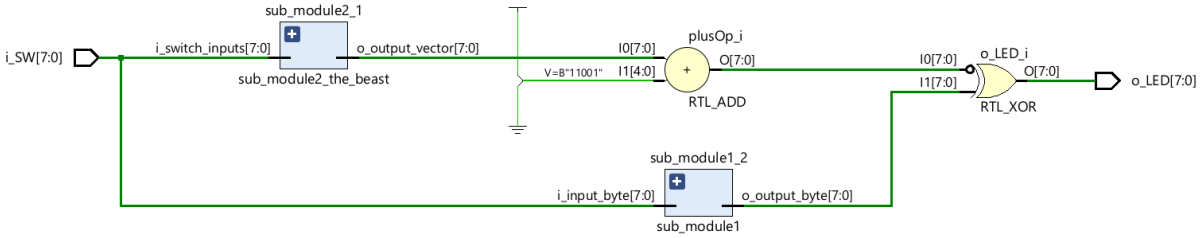

Figure 23: My RTL design

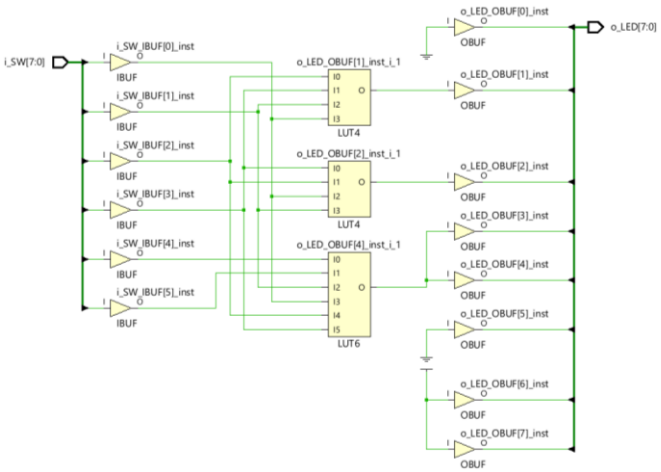

Figure 24: My synthesized design

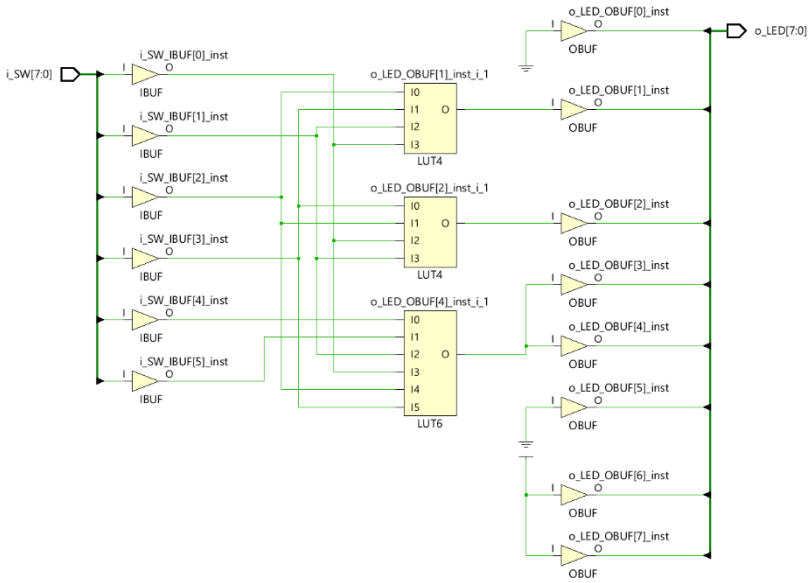

Figure 25: My implemented design

Conclusion:

The purpose of this lab was to get used to Vivado and VHDL by debugging a hierarchal VHDL code then synthesizing and implementing this code and successfully simulating and demonstrating the system on our Basys3 FPGA board. In the end, I think that I was able to achieve all of the lab's goals by successfully writing a testbench code and simulating my design and implementing this design to my Basys3.

References:

1- https://www.wevolver.com/article/rtl-design-a-comprehensive-guide-to-unlocking-the-power-of-register-transfer-level-design
2- https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug945-vivado-using-constraints-tutorial.pdf
3- https://surf-vhdl.com/vhdl-for-loop-statement/

Appendices:

sub_module1.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity sub_module1 is
	Port (
		i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
		o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
```

```
	);
end sub_module1;

architecture Structural_sub1 of sub_module1 is

begin
	o_output_byte(0)          <= i_input_byte(0) xor i_input_byte(1);
	o_output_byte(1)          <= i_input_byte(2) or i_input_byte(3);
	o_output_byte(2)          <= i_input_byte(4) xor i_input_byte(5);
	o_output_byte(5 downto 3) <= "010";
	o_output_byte(7 downto 6) <= (others => '0');

end Structural_sub1;
```

sub_module2.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sub_module2_the_beast is
    Port (
        i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
        o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
    );
end sub_module2_the_beast;

architecture Structural_sub2 of sub_module2_the_beast is
    component sub_module1 is
        port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module1;

    signal s_inv_input : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin
```

```
    s_inv_input <= not i_switch_inputs;

    sub_module1_1 : sub_module1
        port map (
            i_input_byte  => s_inv_input,
            o_output_byte => o_output_vector
        );
end Structural_sub2;
```

top_module.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top_module is
    Port (
        i_SW  : in  STD_LOGIC_VECTOR (7 downto 0);
        o_LED : out STD_LOGIC_VECTOR (7 downto 0)
    );
end top_module;

architecture Structural_top of top_module is

    component sub_module1 is
        port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module1;

    component sub_module2_the_beast is
        port (
            i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
```

```
        );
    end component sub_module2_the_beast;

    signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal s_output_3             : unsigned(7 downto 0)         := (others => '0');

begin

    sub_module1_2 : sub_module1
        port map (
            i_input_byte  => i_SW,
            o_output_byte => s_output_1
        );

    sub_module2_1 : sub_module2_the_beast
        port map (i_switch_inputs => i_SW,
                  o_output_vector => s_output_2);
    s_output_3 <= unsigned(s_output_2) + 25;

    o_LED <= (not std_logic_vector(s_output_3)) xor s_output_1;

end Structural_top;
```

Lab2_testbench.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top_module is
    Port (
        i_SW  : in  STD_LOGIC_VECTOR (7 downto 0);
        o_LED : out STD_LOGIC_VECTOR (7 downto 0)
    );
end top_module;
```

```
architecture Structural_top of top_module is

    component sub_module1 is
        port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module1;

    component sub_module2_the_beast is
        port (
            i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module2_the_beast;

    signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal s_output_3             : unsigned(7 downto 0)         := (others => '0');

begin

    sub_module1_2 : sub_module1
        port map (
            i_input_byte  => i_SW,
            o_output_byte => s_output_1
        );

    sub_module2_1 : sub_module2_the_beast
        port map (i_switch_inputs => i_SW,
                  o_output_vector => s_output_2);
    s_output_3 <= unsigned(s_output_2) + 25;

    o_LED <= (not std_logic_vector(s_output_3)) xor s_output_1;
```

```
end Structural_top;
```

constraints_basys3.xdc:

```
# Constraint file sets which input/output in the VHDL code connects to which pin in FPGA.

# Three characters after PACKAGE_PIN gives the FPGA pin which is also written next to all LEDs/Switches on the board.

# The second line (IOSTANDARD LVCMOS33) sets the voltage standard for the pins which is constant and always the same for BASYS3.


set_property PACKAGE_PIN V17 [get_ports {i_SW[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[0]}]

set_property PACKAGE_PIN V16 [get_ports {i_SW[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[1]}]

set_property PACKAGE_PIN W16 [get_ports {i_SW[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[2]}]

set_property PACKAGE_PIN W17 [get_ports {i_SW[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[3]}]

set_property PACKAGE_PIN W15 [get_ports {i_SW[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[4]}]

set_property PACKAGE_PIN V15 [get_ports {i_SW[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[5]}]

set_property PACKAGE_PIN W14 [get_ports {i_SW[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[6]}]

set_property PACKAGE_PIN W13 [get_ports {i_SW[7]}]

set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[7]}]



# LEDs

set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[0]}]

set_property PACKAGE_PIN E19 [get_ports {o_LED[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]

set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]

set_property PACKAGE_PIN V19 [get_ports {o_LED[3]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {o_LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[4]}]
set_property PACKAGE_PIN U15 [get_ports {o_LED[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[5]}]
set_property PACKAGE_PIN U14 [get_ports {o_LED[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[6]}]
set_property PACKAGE_PIN V14 [get_ports {o_LED[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[7]}]

set_property PACKAGE_PIN U16 [get_ports {o_LED[0]}]
```