Bilkent University

Electrical and Electronics Department

EE102-01 Lab 2 Report:

“Introduction to VHDL”

02/10/2023

Fatih Mehmet Çetin - 22201689

Purpose:

The main purpose of this laboratory experiment was to get used to the VHDL coding language, the BASYS3 board operations and the Vivado software. Particularly, finding out a solution to a daily-life problem was asked in the lab. The specific daily-life problem examined in this study is one of the biggest problems gym community has been facing: Skipping the leg day workout. A tracker was built to help the gym-bros keep track of their leg day workouts and not lose the fit shape.

Methodology:

In this experiment, finding a solution to a real-life problem was the main concern. The problem that was examined in this particular experiment was something many people overlooked in the gym: the leg day. After coming up with the problem, a logical expression of the solution to the problem was made using AND and OR gates. The experiment consisted of two parts.

Firstly, all the variables were set and the logic behind the circuit was coded. Rather than manually creating the variables, variables can be created by clicking the “Add Sources” and “Create Design Sources” buttons in Vivado. Then the synthesis and implementation steps of the code was done. Then, an RTL schematic of the circuit was created **(Figure 1.1)**. To test the circuit design, a test bench code was generated to simulate all the possible input possibilities. Then, a behavioural simulation was conducted by the Vivado app, and a waveform of the simulation was generated (**Figure 1.2)**.

For the second part of the experiment, the constraints file was added to the project. After that, clicking on the “Generate Bitstream” button generated a “.bit” file which will be imported on BASYS3. Then, BASYS3 board was connected to the computer and the VHDL codes were loaded onto the BASYS3 board. Lastly, every single scenario- which in our case 8 different scenarios- was implemented on the BASYS3 board **(Figures 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8)**.

Q1: How does one specify the inputs and outputs of a module in VHDL?

The VHDL code lines below can be used to designate output and input variables:

```
-------------------------------------
entity asd is
   Port ( in1 : in STD_LOGIC;
          in2 : in STD_LOGIC;
          in3 : in STD_LOGIC;
          out1 : out STD_LOGIC);
end asd;
---------------------------------------
```

The port function is responsible for identifying the input and output variables of a logic function. In the code, the names of each input and output variable are represented by blue codes. Orange coloured codes are used to indicate whether the variable serves as an input or an output. The most common type used in VHDL is the “std_logic”. For this lab, it is desired that the digital interface of the wire to either have the value “1” or “0”. These two are the only values a binary digit can have. Nevertheless, in reality a physical digital can be in a number of states. Therefore, “std_logic” means the system will only use standard logic algebra.

Q2: How does one use a module inside another code/module? What does PORT MAP do?

A module is a self-contained unit of VHDL code. Modules communicate with the outside world through the entity. Port map is the part of the module instantiation where you declare which local signals the module’s inputs and outputs shall be connected to. A module without any input or output signals cannot be used in a real design. Its only purpose is to allow us to run VHDL code in a simulator. Therefore, it is referred to as a testbench. To simulate a module with input and output signals we have to instantiate it in a testbench. The port map function is called at the test bench step of the operation, and the variables and functions are copied to the test bench module with the source module itself. So basically, a port map; maps signals in an architecture to ports on an instance within that architecture. Here is an example port map code:

```
---------------------------------------------
UUT: asd PORT MAP( in1 => in1,
                   in2 => in2,
                   in3 => in3,
                   out1 => out1 );
--------------------------------------------
```

The “asd” module is called into a testbench module. Port map function is used to use a module in another module, and every variable is set to another variable one by one via “=>".

Q3: What is a constraint file? How does it relate your code to the pins on your FPGA?

Constraint files are simple documents within the system and can be compiled to add more intelligence to the whole process. A constraint document contains a list of statements, known as constraint groups, each of which targets one or more objects and contains one or more constraints. For this specific lab, variables were assigned to various LEDs on the BASYS3 board. Here is an example code in a constraint file:

```
------------------------------------------

set_property PACKAGE_PIN V17 [get_ports {in1}]

        set_property IOSTANDARD LVCMOS33 [get_ports {in1}]
```

The blue coloured code means that the red coloured variable has been assigned to the V17 pin on the BASYS3 board by using the set_property function.

Q4: What is the purpose of writing a testbench?

Test benches are used to simulate your design without the need of any physical hardware. The biggest benefit of this is that you can actually inspect every signal that is in your design. Test benches show the outcome of the design code, and they are very useful since the user doesn’t have to upload the code to the FPGA board again and again each time an error occurs.

Design Specifications:

In the fitness community, the most popular way to perform a weekly workout is the Push-Pull-Legs split -commonly known as PPL split. It consists of three different workout routines, each of them is performed at least twice a week in different days, six days in total. In a push day, a bodybuilder works out three main muscle groups: chest, shoulder and triceps. In a pull day, two main muscle groups are activated: back and biceps. The last exercise routine leg day, involves heavy exercises targeting the lower body muscles, including the quadriceps, hamstrings, glutes, and calves. Unfortunately, some bodybuilders tend to skip the entire leg day. This can cause problems because the legs are essential for overall aesthetics and balance for the human body. In a PPL-split; Mondays and Thursdays are push days, Tuesdays and Fridays are pull days. That leaves us with at least two out of three possible days to workout legs in order to keep a fit shape.

In the project we should design a circuit with three inputs (in1, in2 and in3) and one output (out1). Here is the summary of what each variable stands for:

in1: Having a leg day workout on Wednesday

in2: Having a leg day workout on Saturday

in3: Having a leg day workout on Sunday

out1: Having a fit body and a decent shape

Since for a decent shape a bodybuilder should at least workout legs twice a week, the logical expression for this problem is the following:

out1 = (in1)*(in2) + (in1)*(in3) + (in2)*(in3) **(Eq1)**

Here is the truth table of the logical expression:

in1	in2	in3	out1
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Table 1.1: Truth Table for Eq1

Results:

After writing the design code which has 3 inputs (in1, in2 and in3) and a single output (out1), the logical equation was written in VHDL language. The synthesis and implementation steps were completed successfully. The next step was getting the elaborated design which is also called as the RTL schematic or RTL design.

This was the step where a problem occurred. Whenever I clicked the "Open Elaborated Design" button, the Vivado application crashed without any crash report. After hours of research and visiting countless internet forums about this issue, I was able to find a solution to this problem. The root of this problem lied within the username of my personal account. I use a Win11 64-bit system. My windows username was "EXCALİBUR". Since there is a non-alphabetical character in mu username, whenever I tried to view the RTL schematic, my username gave Vivado indigestion **(Link 1.1)**. Also, changing the windows username did not change the user folder name in hard drive. Therefore, I had to manually create a new account by the name "MEHMET" -since it doesn't have any non-alphabetical character- and transfer all my previous files from the previous account. This was a major issue that was resolved in the end, but it took a couple of hours and lots of effort. I also contacted some of my friends and there was a significant number of students that had the same problem. I believe creating a tutorial about this common issue and uploading it into Moodle system will save hundreds of hours to tens of students every semester.

After solving this major problem, an expected RTL schematic of the specific logic equation appeared on the screen **(Figure 1.1)**:

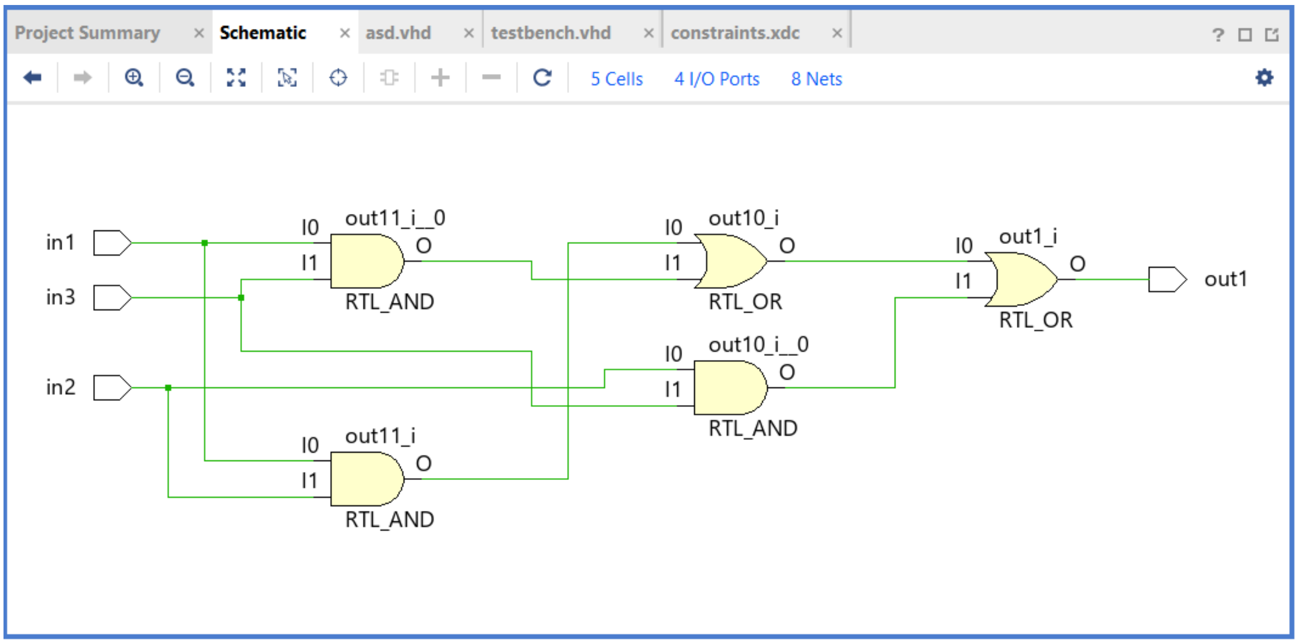

Figure 1.1: The RTL Schematic of Eq1

The synthesis and implementation steps were not necessary for the RTL schematic, but they are needed for the signal simulation of the function. In order to make a simulation, a testbench code is written first. Since the equation contains 3 inputs, there were 8 different scenarios that needs to be tested. All these conditions were coded in the testbench code with a period of 100ns. After all these steps, a successful simulation took place and an expected waveform signal appeared on the screen **(Figure 1.2)**:

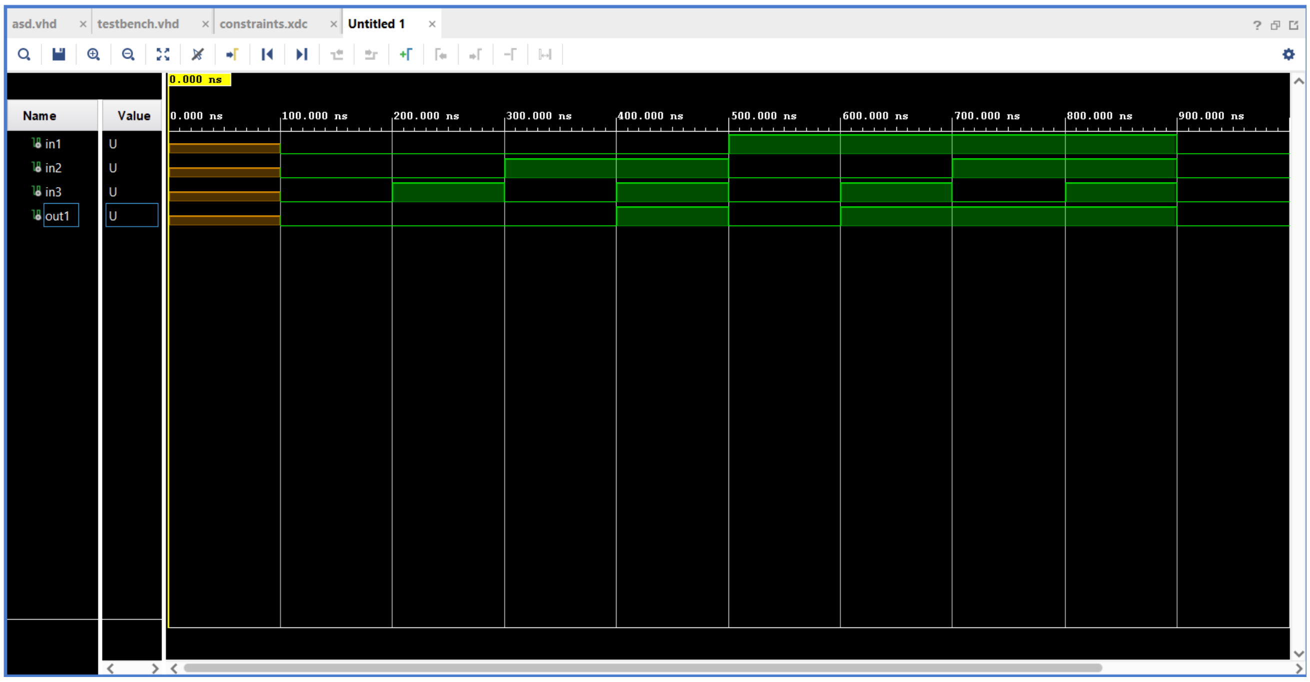

Figure 1.2: The Signal Waveform Simulation of Eq1

After completing the synthesis, implementation and simulation steps without any error, the constraints file was added to the project. In the constraints file, the 3 switches from the right (W17, W16 and V16) was assigned to the three inputs (in1, in2 and in3). The output (out1) was assigned to the green led at the far right (U16). After successfully writing the constraints file, the code was ready for the bitstream generation. Then, the computer got connected to BASYS3 via a micro-USB cable. Finally, the project was uploaded onto BASYS3, and the board was ready to work. Here are the implementations of all the possible input combinations on the BASYS3 board **(Figures 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8)**:

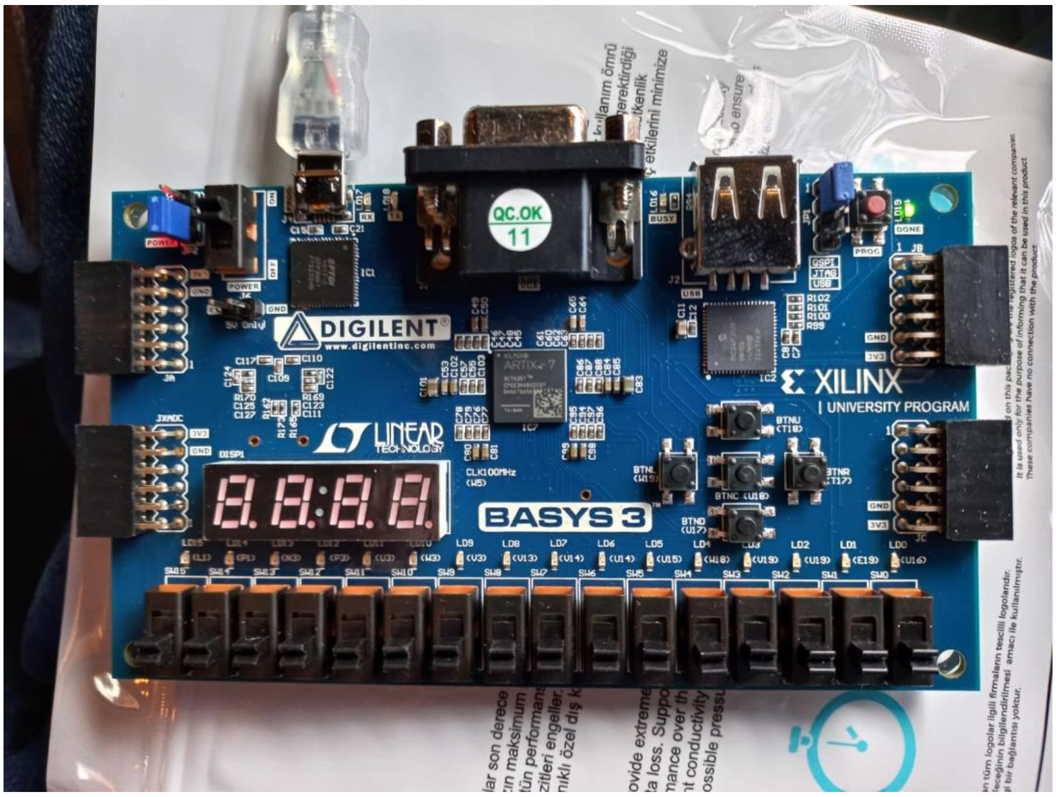

Figure 2.1: (in3 <= 0, in2 <= 0, in1 <= 0, out1 <= 0)

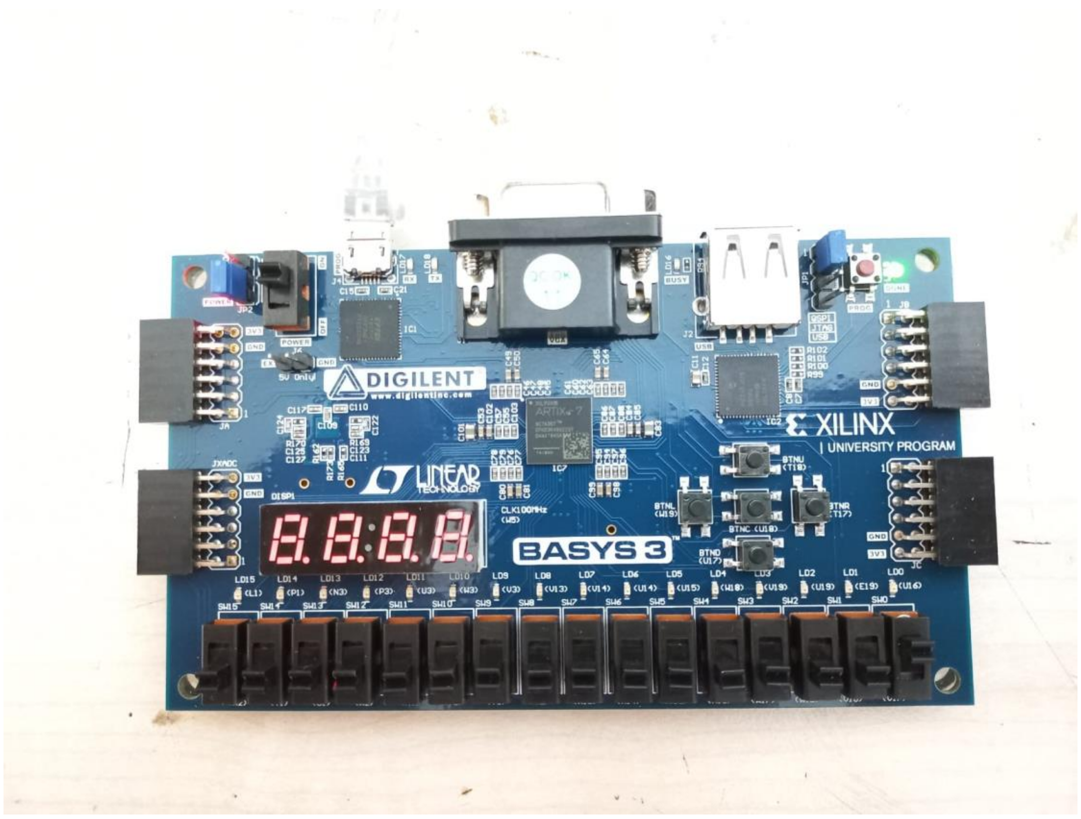

Figure 2.2: (in3 <= 0, in2 <= 0, in1 <= 1, out1 <= 0)

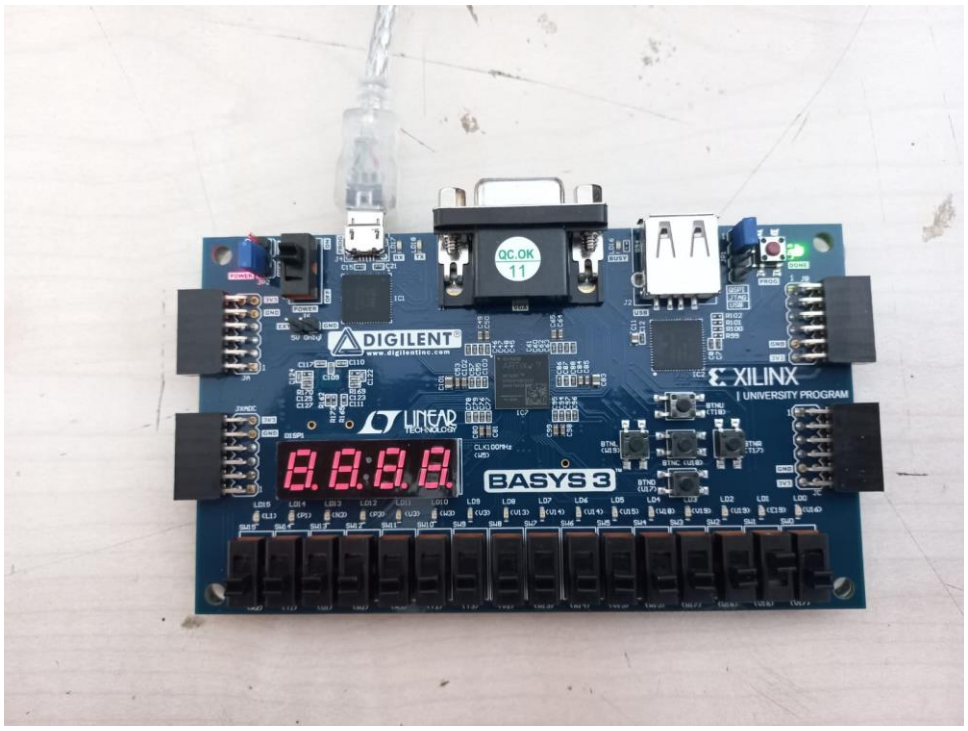

Figure 2.3: (in3 <= 0, in2 <= 1, in1 <= 0, out1 <= 0)

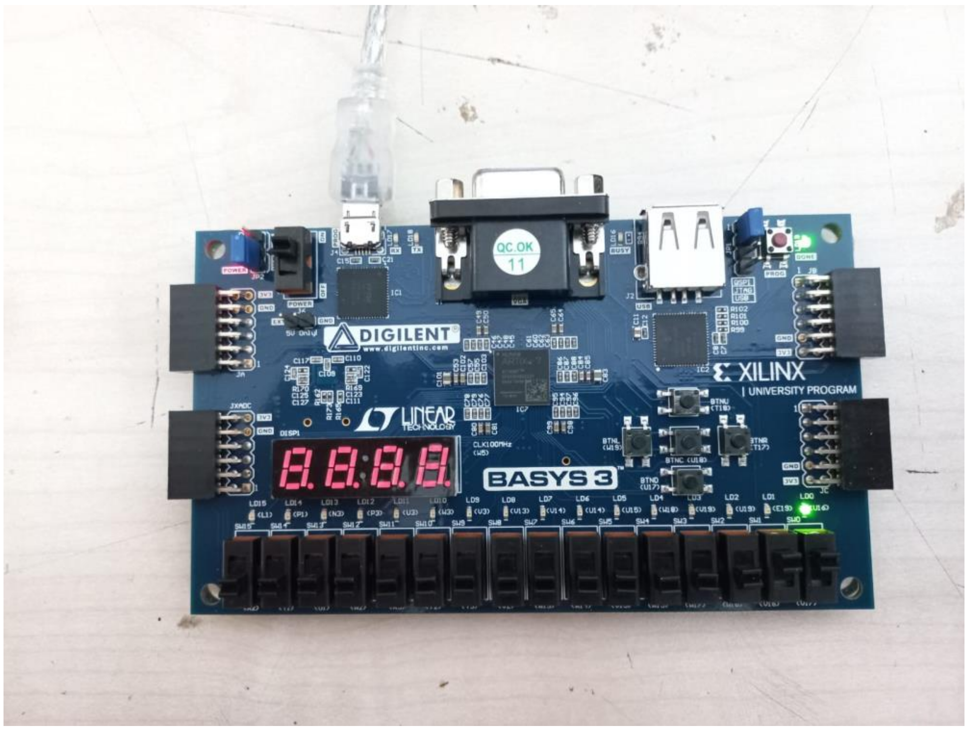

Figure 2.4: (in3 <= 0, in2 <= 1, in1 <= 1, out1 <= 1)

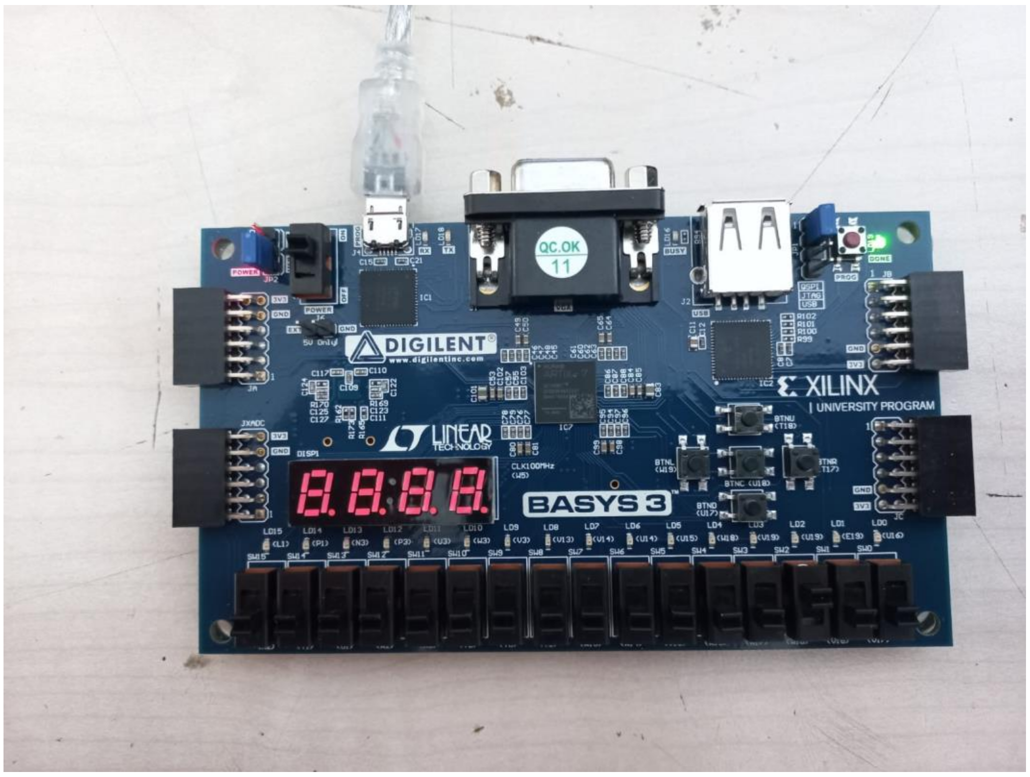

Figure 2.5: (in3 <= 1, in2 <= 0, in1 <= 0, out1 <= 0)

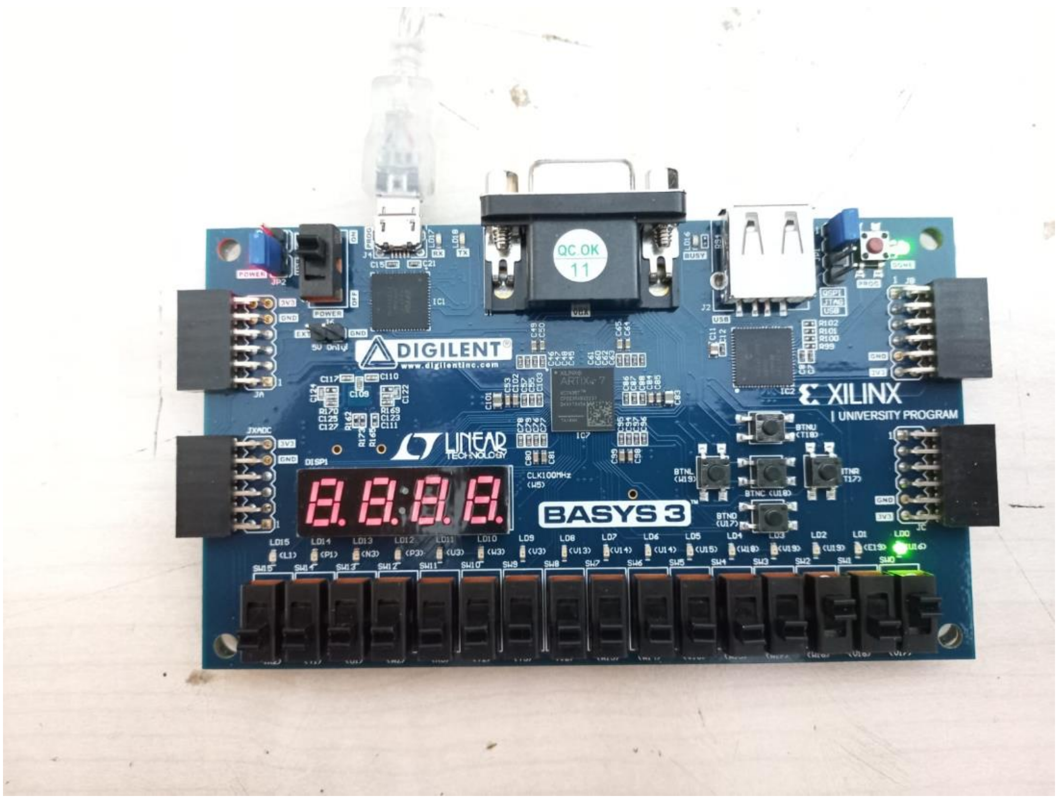

Figure 2.6: (in3 <= 1, in2 <= 0, in1 <= 1, out1 <= 1)

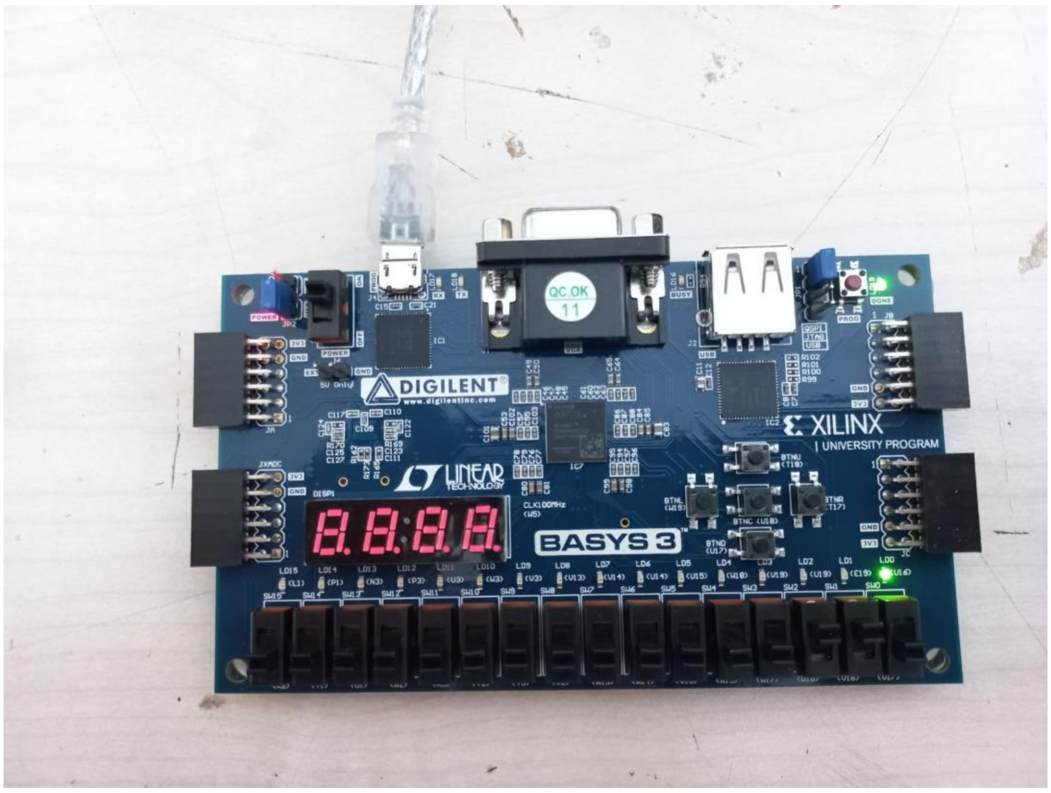

Figure 2.7: (in3 <= 1, in2 <= 1, in1 <= 0, out1 <= 1)

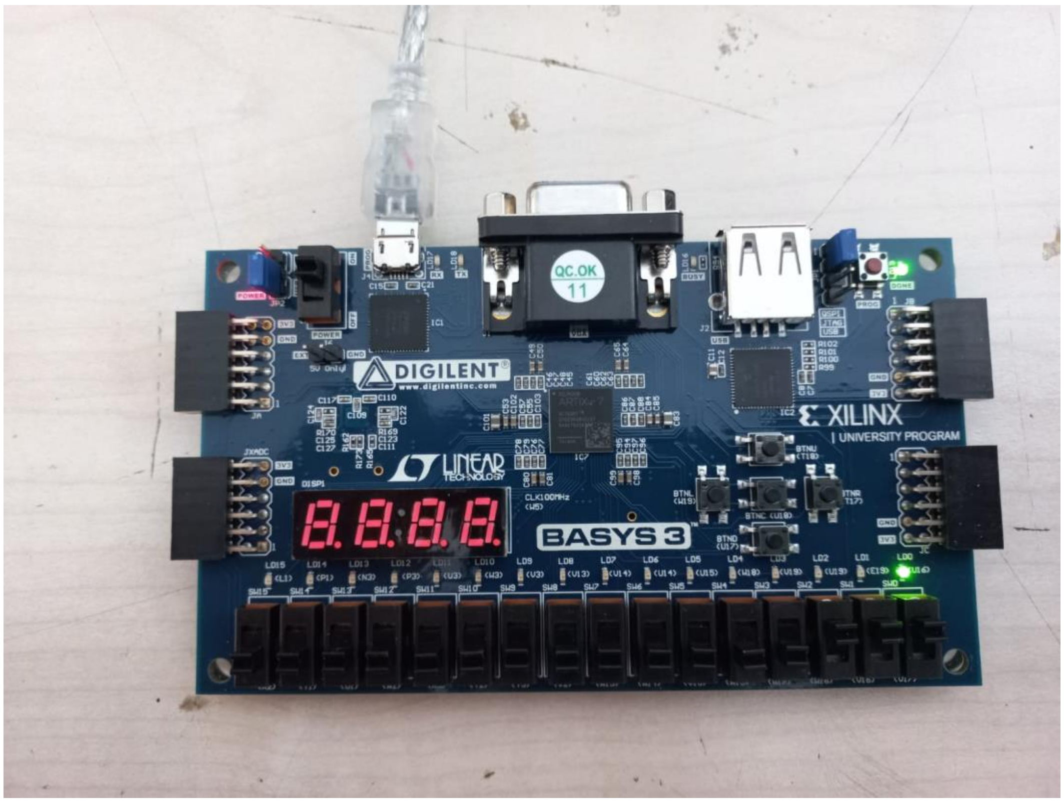

Figure 2.8: (in3 <= 1, in2 <= 1, in1 <= 1, out1 <= 1)

Conclusion:

In this lab, the basics and structures of BASYS3, Vivado and VHDL was examined. A real-life problem was designed on paper and then implemented on BASYS3 board by using the Vivado software and VHDL. This lab was a starting step to digital circuit design, and it showed us how fast and effective VHDL and Vivado can be if used correctly. There was not much of a chance for human errors, all the findings matched the expected outcomes. Lastly, the lab showed us how logic gates can be used in real-life situations.

Appendices:

Link1.1: https://support.xilinx.com/s/question/0D52E00006hpj9PSAQ/vivado-20182-crash-on-elaborating-design?language=en_US

VHDL Code:

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

entity asd is
   Port ( in1 : in STD_LOGIC;
          in2 : in STD_LOGIC;
          in3 : in STD_LOGIC;
          out1 : out STD_LOGIC);
end asd;

architecture Behavioral of asd is

begin
out1 <= (in1 and in2) or (in1 and in3) or (in2 and in3);
end Behavioral;
```

Test Bench Code:

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

entity testbench is
end testbench;

architecture Behavioral of testbench is

component asd
Port(in1: in std_logic;
    in2: in std_logic;
    in3: in std_logic;
    out1: out std_logic);
End component;

signal in1: std_logic;
signal in2: std_logic;
signal in3: std_logic;
signal out1: std_logic;

begin
UUT: asd port map(
in1 => in1,
in2 => in2,
in3 => in3,
out1 => out1
);

testbench: process
begin
```

```
wait for 100ns;
in1 <= '0';
in2 <= '0';
in3 <= '0';
wait for 100ns;
in1 <= '0';
in2 <= '0';
in3 <= '1';
wait for 100ns;
in1 <= '0';
in2 <= '1';
in3 <= '0';
wait for 100ns;
in1 <= '0';
in2 <= '1';
in3 <= '1';
wait for 100ns;
in1 <= '1';
in2 <= '0';
in3 <= '0';
wait for 100ns;
in1 <= '1';
in2 <= '0';
in3 <= '1';
wait for 100ns;
in1 <= '1';
in2 <= '1';
in3 <= '0';
wait for 100ns;
in1 <= '1';
in2 <= '1';
in3 <= '1';
```

```
end process;

end Behavioral;
```

Constraints:

```
set_property PACKAGE_PIN V17 [get_ports {in1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in1}]
set_property PACKAGE_PIN V16 [get_ports {in2}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in2}]
set_property PACKAGE_PIN W16 [get_ports {in3}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in3}]
set_property PACKAGE_PIN U16 [get_ports {out1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1}]
```