Report to Lab 2:
Introduction to VHDL

Arda Özkut
22101727
Section 2
Physics Department, Bilkent University
(Dated: October 3, 2022)

I. PURPOSE OF THE EXPERIMENT

The purpose of this experiment was to learn the basics of Vivado software, FPGA programming and the implementation of FPGA design to real life problems.

II. METHODOLOGY

The first step of the design is creating the truth tables of the desired outputs corresponding to the input signals as can be seen bellow. The truth tables are constructed to resemble the safety system of the motorised table of an industrial laser cutter.

x_1 models the switch to the laser diode. When the switch is closed the state flips to 1. x_2 represents the state of the water cooling system of the laser. $g(x_1, x_2)$ models the power to the laser. If x_2 and the diode switch signal are in the state of one, $g(x_1, x_2)$ is flipped on, turning on the laser diode. Lastly x_3 signal is flipped to 1 if the computer has finished cutting.

x_1	x_2	x_3	$g(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

We can calculate the simplest form of g with using sum-of-products.

$$g(x_1, x_2, x_3) = m_6 = x_1.x_2.\overline{x_3} \tag{1}$$

Since there is one minterm the canonical sum is the simplest.

In the truth table bellow the function $f(g(x_1, x_2, x_3), x_4, x_5)$ models the power to the motorized table.

- $f(g(x_1, x_2, x_3), x_4, x_5) = 1$ if the motorised table has power.
- $x_4 = 1$ if there is a sample on the motorised table.
- $x_5 = 1$ if the safety door is closed.
- $x_6 = 1$ if the emergency button is pressed.

If the laser power is on or there is a sample on the cutting table the table should be able to move. If the laser is off, the table can move freely without a sample. This way, the door is not required to be closed. If the door closed signal flips to 0 the process should stop if the laser is also on and if the emergency signal is pressed and the power should

be switched off.

g	x_4	x_5	x_6	$f(g(x_1,x_2,x_3),x_4,x_5,x_6)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

From here one can find the minterms and OR the terms to get the sum-of-products.

$$\begin{aligned} m_0 &= \overline{g}.\overline{x_4}.\overline{x_5}.\overline{x_6} \\ m_2 &= \overline{g}.\overline{x_4}.x_5.\overline{x_6} \\ m_4 &= \overline{g}.x_4.\overline{x_5}.\overline{x_6} \\ m_6 &= \overline{g}.x_4.x_5.\overline{x_6} \\ m_{10} &= g.\overline{x_4}.x_5.\overline{x_6} \\ m_{14} &= g.x_4.x_5.\overline{x_6} \end{aligned} \tag{2}$$

$$m_0.m_2.m_4.m_6.m_{10}.m_{14} = \overline{g}.\overline{x_4}.\overline{x_5}.\overline{x_6} + \overline{g}.\overline{x_4}.x_5.\overline{x_6} + \overline{g}.x_4.\overline{x_5}.\overline{x_6} + \overline{g}.x_4.x_5.\overline{x_6} + g.\overline{x_4}.x_5.\overline{x_6} + g.x_4.x_5.\overline{x_6} \tag{3}$$

Simplifying the equation we get

$$\begin{aligned} \Sigma m &= \overline{g}.\overline{x_4}.\overline{x_6}(\overline{x_5} + x_5) + \overline{g}.x_4\overline{x_6}(\overline{x_5} + x_5) + g.x_5\overline{x_6}(x_4 + \overline{x_4}) \\ &= \overline{g}.\overline{x_6}(x_4 + \overline{x_4}) + g.x_5.\overline{x_6} \\ &= \overline{x_6}(\overline{g} + g.x_5) \end{aligned} \tag{4}$$

After this I used Vivado to write the vhd file. We can do this by creating a design source where we define the inputs and outputs of the module. All the code can be seen in Appendix A.
The RTL schematic can be seen in figure 1.

After generating the RTL schematic I wrote a test bench file to check for possible errors or glitches.

Questions

1. One can specify the inputs and outputs by initiating definitions inside the entity of the main module (entity declaration). Each definition consists of the name of the signal, the type of the signal and datatype specification.

2. When designing a hierarchical set of modules, instead of writing a single module with complex design the functionality of PORT MAPs can be used to divide the code into sub modules that can be combined using PORT MAPs.

3. Constraint files allow the program to implement the specified inputs and outputs into the physical chip of the FPGA board. The user maps the desired input and output signals coming from the modules to pins, leds or buttons that can represent signals being turned on and off.

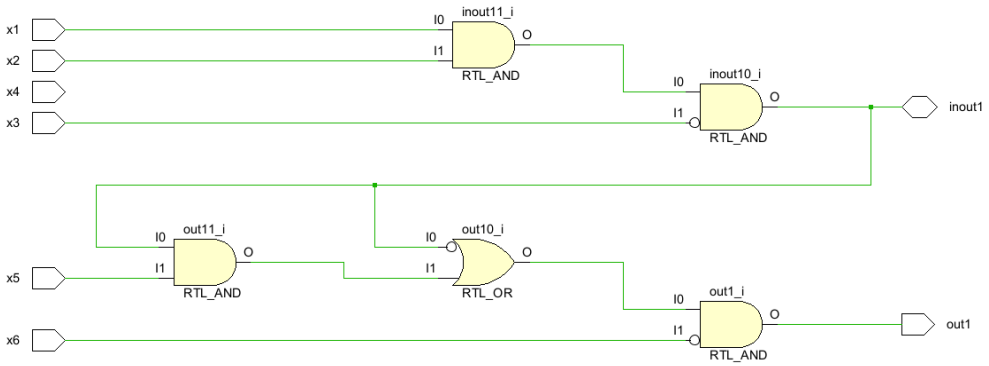

FIG. 1. RTL Schematic

4. Writing a test bench is an important step after synthesizing a network. A test bench produces signal to be sent into the desired module and test the outcome corresponding to the signal sent. The test bench, therefore, allows the designer or the simulator to see what the output is when applied a signal. This could help designers check for faults or glitches in their module and it could help others learn about the purpose of a logic network.

III. RESULTS

We can see that the test bench yields consistent results with the truth table. I tested all the 64 possible combinations with the test bench. The code can be found in Appendix 2A.
I then took photos of the board with all possible combinations. Some of the pictures can be found in Appendix B.

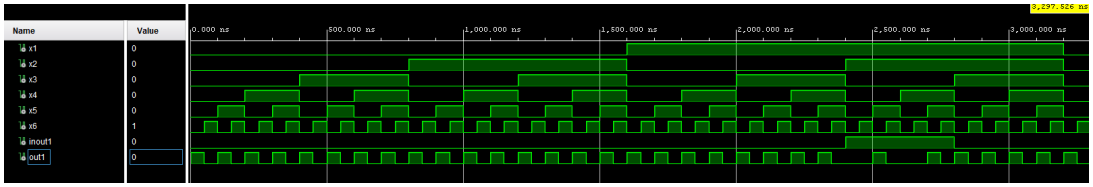

FIG. 2. The Testbench Results

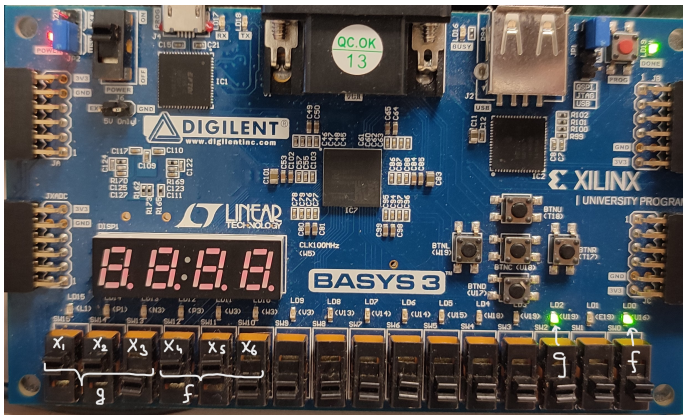

FIG. 3. 110110, $f = 1$, $g = 1$

The rest pictures can be found in Appendix B.

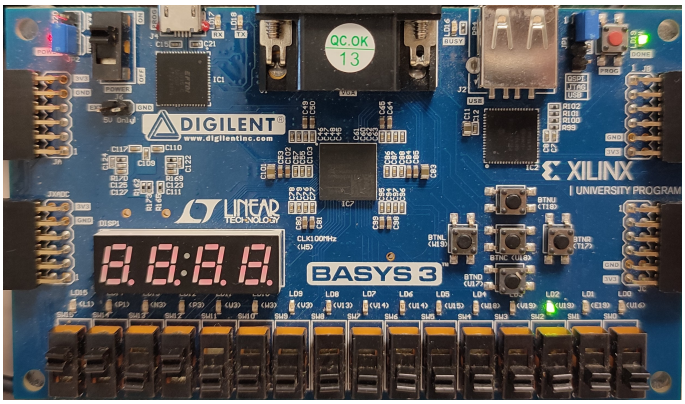

FIG. 4. 110100, $f = 0$, $g = 1$

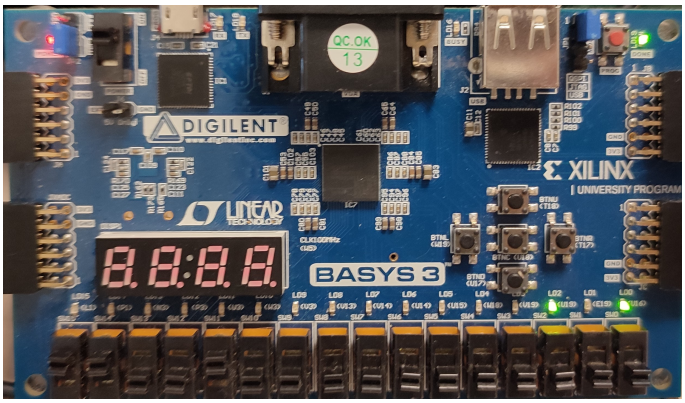

FIG. 5. 110010, $f = 1$, $g = 1$

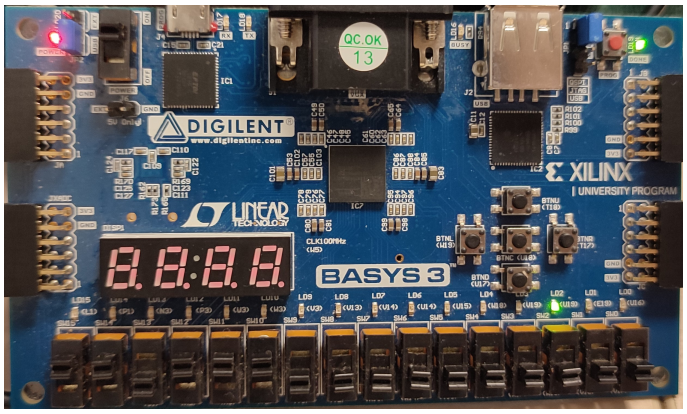

FIG. 6. 110111, $f = 0$, $g = 1$

IV. CONCLUSION

In this lab session I learned how to model real world problems using logic networks. I learned how to program modules, use PORT MAP, write constraints and run test benches using BASYS 3 and the Vivado software.

APPENDIX

A. Code

1. *Design Source*

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

entity source1 is
    Port ( x1 : in STD_LOGIC;
           x2 : in STD_LOGIC;
           x3 : in STD_LOGIC;
           x4 : in STD_LOGIC;
           x5 : in STD_LOGIC;
           x6 : in STD_LOGIC;
           inout1 : inout std_logic;
           out1 : out STD_LOGIC);
end source1;

architecture Behavioral of source1 is

begin
inout1 <= (x1 and x2 and not x3);
out1 <= ((not inout1) or (inout1 and x5)) and not x6;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TestBench_mainsource is
end TestBench_mainsource;
architecture Behavioral of TestBench_mainsource is
component source1
 Port (    x1 : in STD_LOGIC;
           x2 : in STD_LOGIC;
           x3 : in STD_LOGIC;
           x4 : in STD_LOGIC;
           x5 : in STD_LOGIC;
           x6 : in STD_LOGIC;
           inout1 : inout std_logic;
           out1 : out STD_LOGIC);
end component;
 signal x1: STD_LOGIC;
 signal x2: STD_LOGIC;
 signal x3: STD_LOGIC;
 signal x4: STD_LOGIC;
 signal x5: STD_LOGIC;
 signal x6: STD_LOGIC;
 signal inout1: std_logic;
 signal out1: STD_LOGIC;
begin
dut: source1 PORT MAP(
x1 => x1,
x2 => x2,
x3 => x3,
x4 => x4,
x5 => x5,
x6 => x6,
inout1 => inout1,
out1 => out1
);
TestBench_mainsource: PROCESS
 begin

 x1<='0';
 x2<='0';
 x3<='0';
 x4<='0';
 x5<='0';
 x6<='0';

   wait for 50 ns;
 x1<='0';
 x2<='0';
 x3<='0';
 x4<='0';
 x5<='0';
 x6<='1';

   wait for 50 ns;
 x1<='0';
 x2<='0';
```

```
 x3<='0';
 x4<='0';
 x5<='1';
 x6<='0';

 -- I didn't want to include all 64 inputs here.
  ...

   wait for 50 ns;
 x1<='1';
 x2<='1';
 x3<='1';
 x4<='1';
 x5<='1';
 x6<='0';

  wait for 50 ns;
 x1<='1';
 x2<='1';
 x3<='1';
 x4<='1';
 x5<='1';
 x6<='1';


  wait for 50 ns;
end PROCESS;
end Behavioral;
```

3. Constraints

```
set_property PACKAGE_PIN T2 [get_ports x6]
set_property IOSTANDARD LVCMOS33 [get_ports x6]
set_property PACKAGE_PIN R3 [get_ports x5]
set_property IOSTANDARD LVCMOS33 [get_ports x5]
set_property PACKAGE_PIN W2 [get_ports x4]
set_property IOSTANDARD LVCMOS33 [get_ports x4]
set_property PACKAGE_PIN U1 [get_ports x3]
set_property IOSTANDARD LVCMOS33 [get_ports x3]
set_property PACKAGE_PIN T1 [get_ports x2]
set_property IOSTANDARD LVCMOS33 [get_ports x2]
set_property PACKAGE_PIN R2 [get_ports x1]
set_property IOSTANDARD LVCMOS33 [get_ports x1]
## LEDs
set_property PACKAGE_PIN U16 [get_ports out1]
set_property IOSTANDARD LVCMOS33 [get_ports out1]
#
set_property PACKAGE_PIN U19 [get_ports inout1]
set_property IOSTANDARD LVCMOS33 [get_ports inout1]
```

B. Some of the Pictures of Results

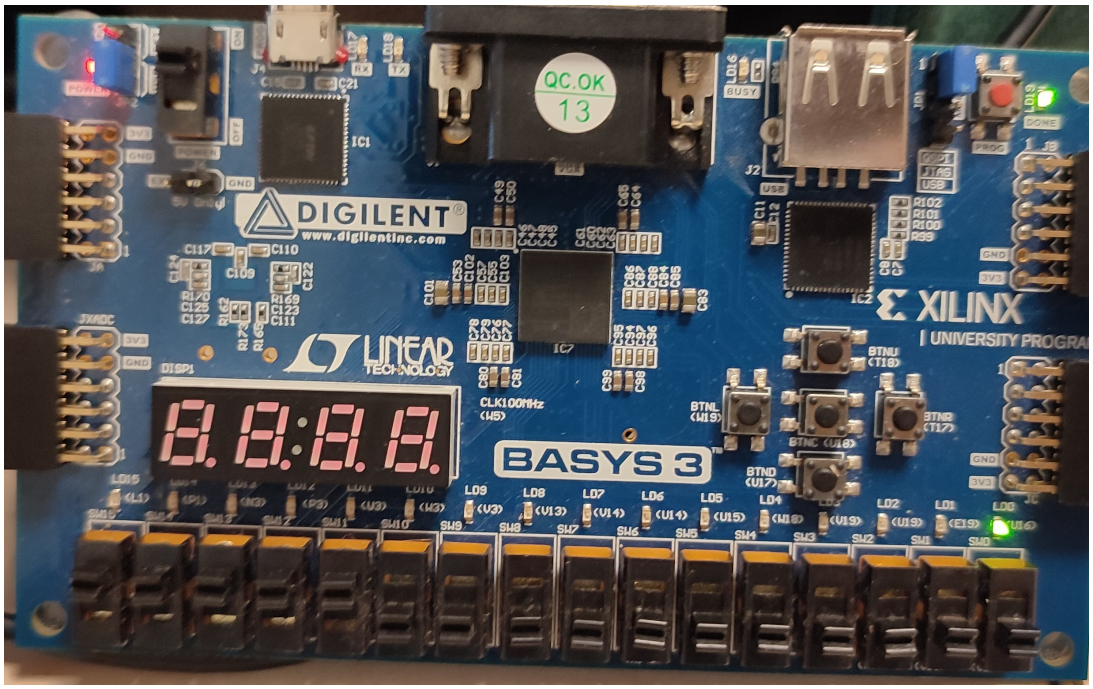

FIG. 7. 111110, $f = 1$, $g = 0$

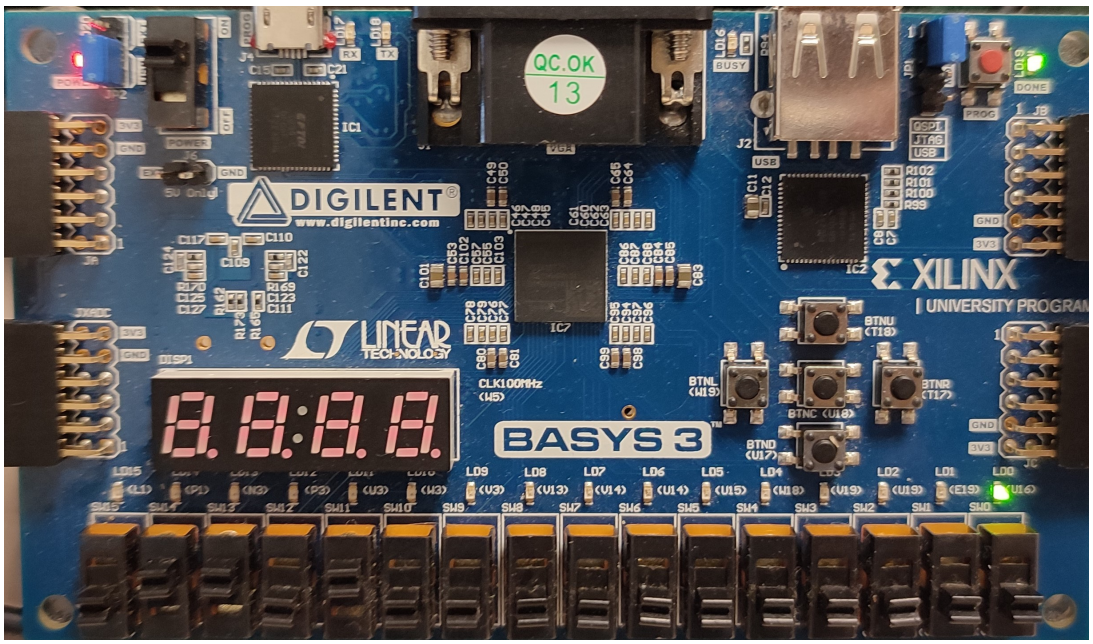

FIG. 8. 011010, $f = 1$, $g = 0$

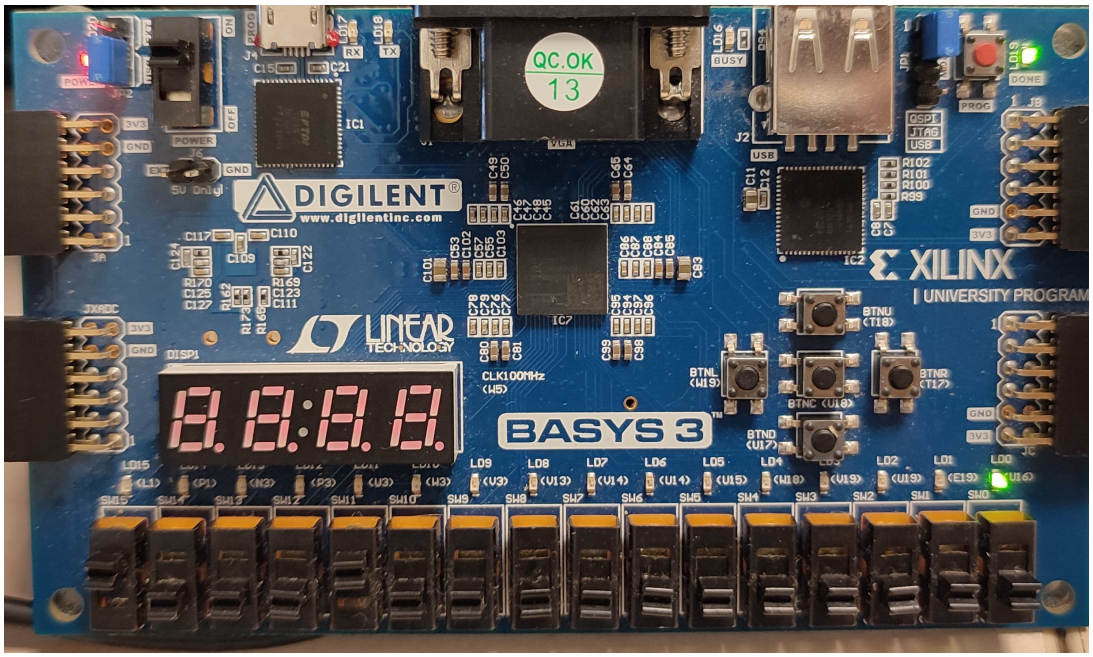

FIG. 9. 100010, $f = 1$, $g = 0$

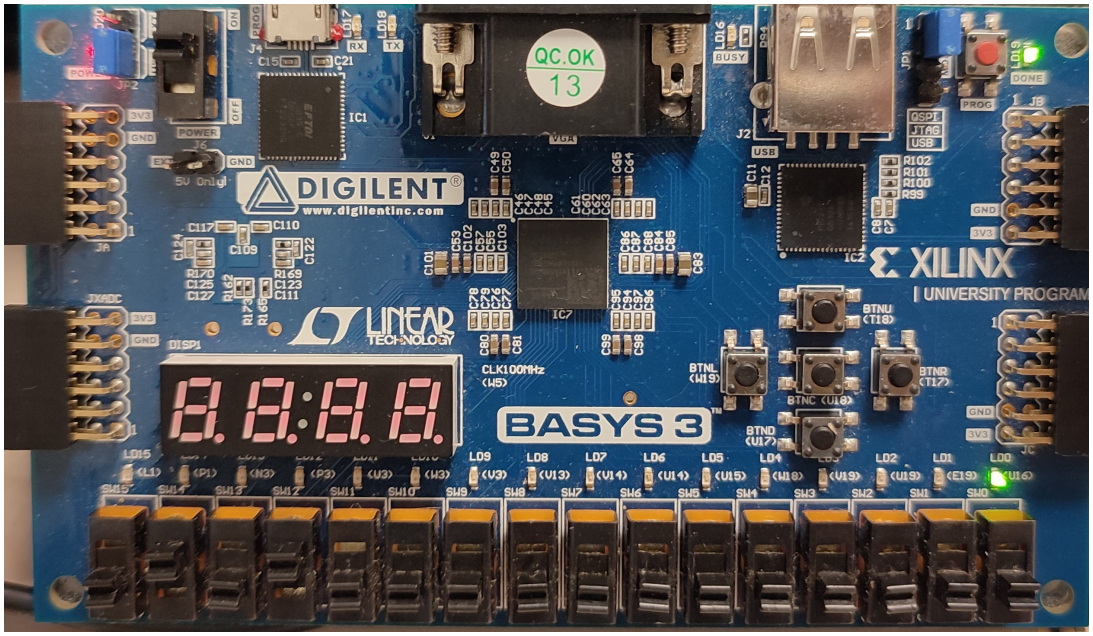

FIG. 10. 010100, $f = 1$, $g = 0$

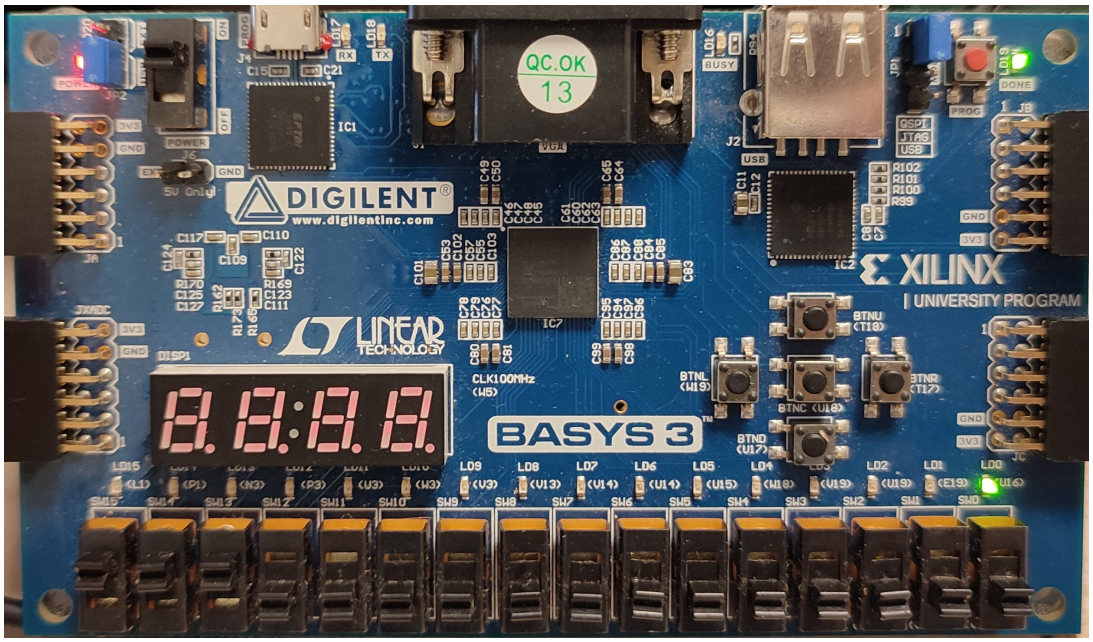

FIG. 11. 111000, $f = 1$, $g = 0$

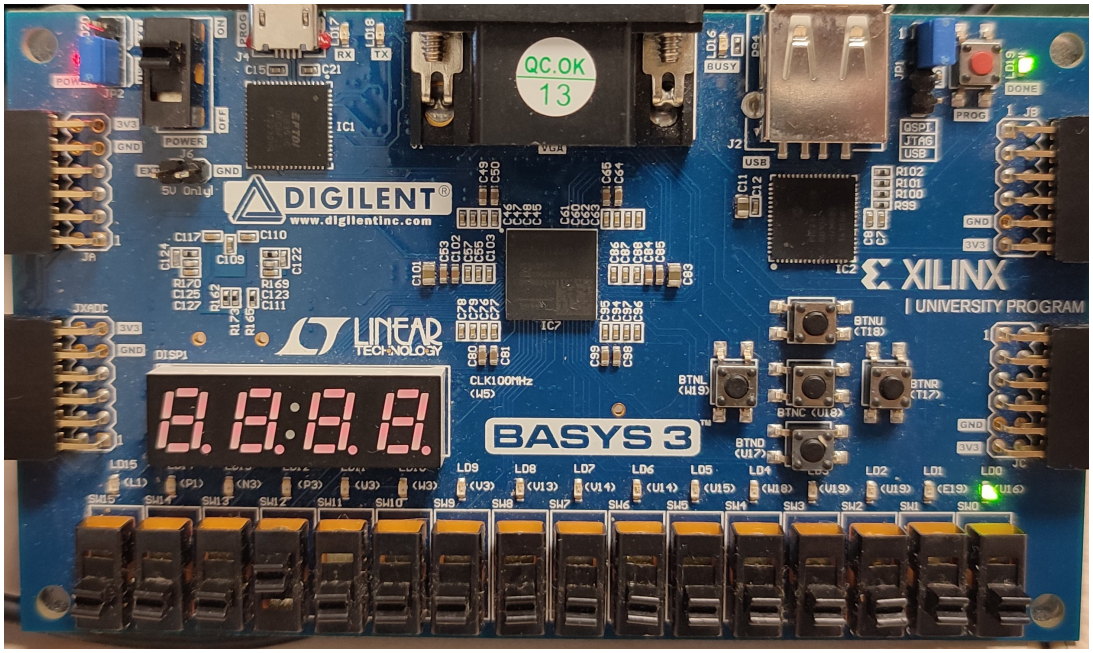

FIG. 12. 000100, $f = 1$, $g = 0$

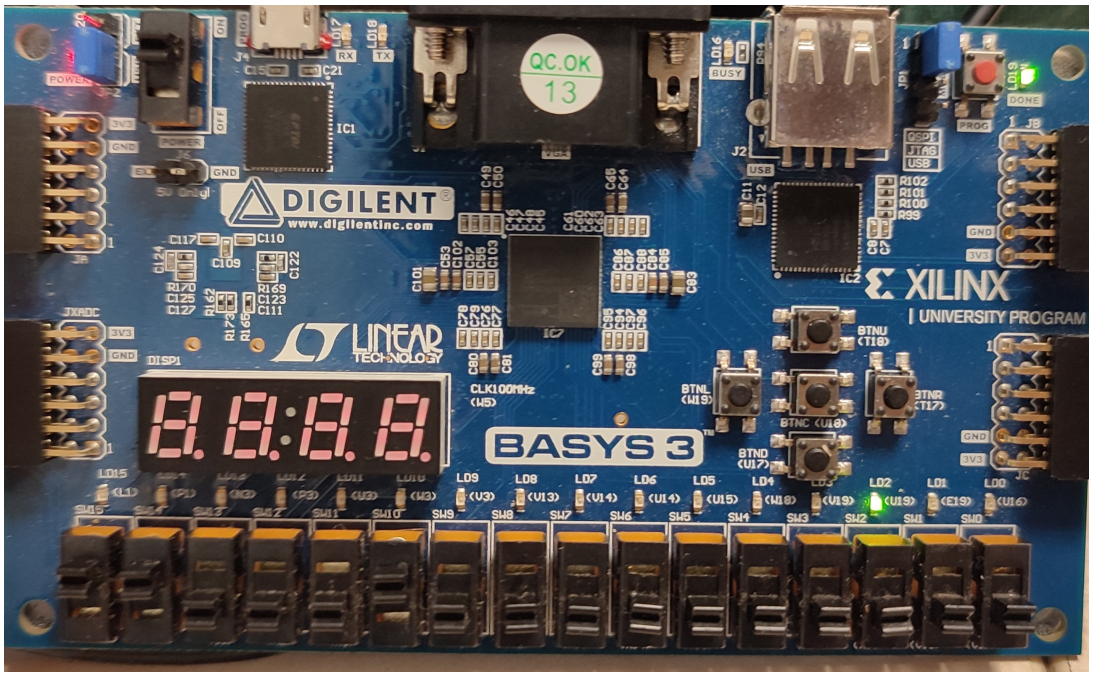

FIG. 13. 110001, $f = 0$, $g = 1$

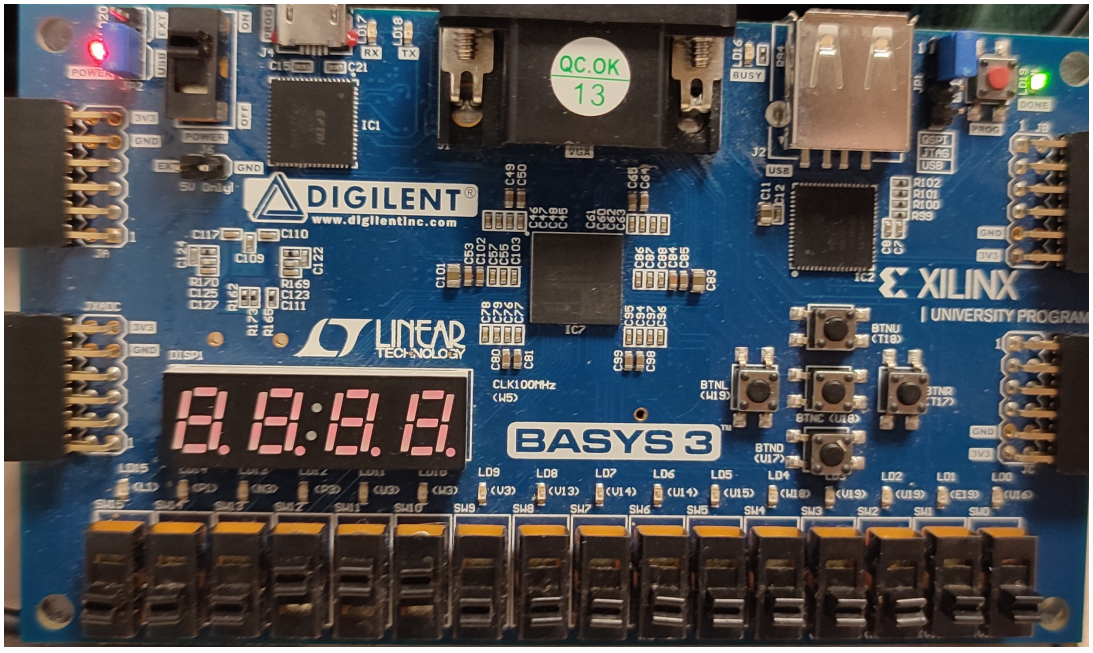

FIG. 14. 000111 1, $f = 0$, $g = 0$