Bilkent University Electrical and Electronics Department
EE102-03 Lab 2 Report:
Introduction to VHDL

Cankut Bora Tuncer – 22001770

Purpose:

The purpose of this lab was to get used to designing a combinational circuit by using the Vivado IDE to compose VHDL code. Furthermore, by compiling the VHDL code into the Basys3 board, learning the basics of an FPGA programing.

Methodology:

The initial decision we made was to simulate a real-life simulation. In my project, the case was the decision mechanism underlying a nuclear fallout which happened in Cuban Missile Crisis in 1962. After deciding on the problem, I recreated the case with AND and NOR gates. First I initialized my variables and constructed the logic of the combinational circuit with VHDL. Then, I synthesized the code and generated the logic circuit (Figure 1.1). In order to test the circuit design, first I created a test bench code to simulate all possible inputs combinations. Vivado IDE generated the waveform as a brief simulation of my VHDL code. Then I initiated the bitwise generator and loaded the VHDL code to Basys3. As for the last task, I implemented the scenario on the FPGA.

Design Specifications:

The design included in this lab was about simulation of a historical event happened in 1962, Cuban Missile Crisis. This design includes 2 AND and 1 NOR gate with 4 inputs and 1 output. The variables can be summarized as:

in1: Grand Assembly
in2: President
in3: Submarine Captain No.1
in4: Submarine Captain No.2
out1: Fate of Humanity

The ‘in1’ and ‘in2’ (Result A) are related to each other with AND gate as well as ‘in3’ and ‘in4’ (Result B). The results from the sets, A and B, are inputs of the NOR gate. The design is supposed to tell whether a nuclear apocalypse is inevitable or not. In this case, if either the collaborative decision made by Grand Assembly (‘in1’) and the President (‘in2’) or Submarine Captain No.1 (‘in3’) and Submarine Captain No.2 (‘in4’) is for using the nuclear missiles, which is 1, the fate of humanity (‘out1’) is doom, 0. If none of the parties decides to use nuclear weapons, the result is peace, 1.

Results:

The proposed logic combination is first designed via VHDL (Code 1). The given input variables are first initialized in the PORT function and then used after the BEGIN initializer. We assigned the inputs and outputs of our design at the PORT function as an element of ‘STD_LOGIC’ by indicating whether the parameter is input, ‘in’ or output, ‘out’ before the ‘STD_LOGIC’ indicator. The output is assigned to the operation ‘((in1 and in2) nor (in3 and in4))’ (main.vhdl).

After the VHDL design we ‘Synthesized’ the code and generated the RTL Schematic of the design. As seen in Figure 1.1 we can observe that ‘in1’ and ‘in2’ are inputted to AND

gate as well as ‘in3’ and ‘in4’. The Vivado generated the NOR gate as the combination of OR and NOT gates and the output is assigned to ‘out1’.

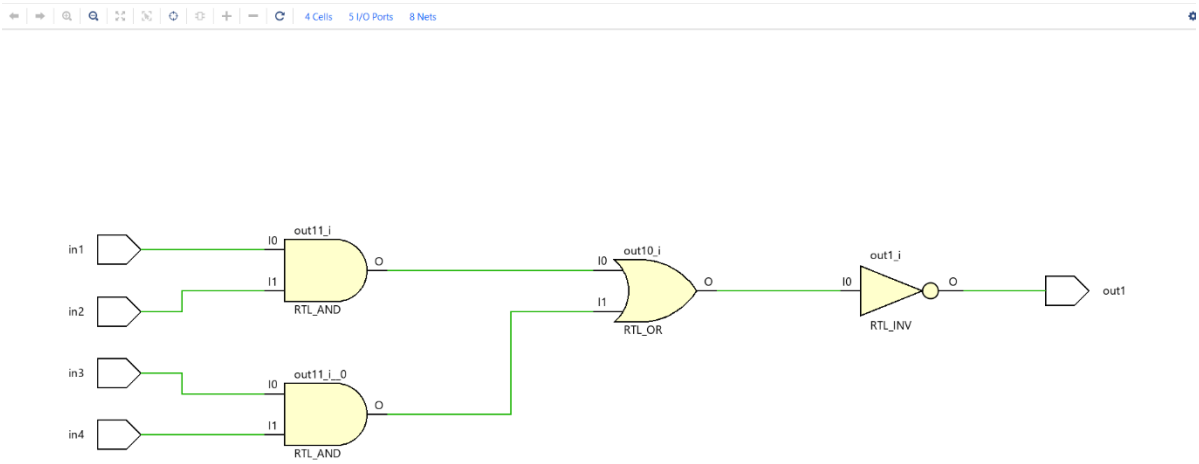

Figure 1.1 The RTL Schematic of the combinational logic gate configuration

In order to simulate the code as a waveform, we first wrote the test-bench code (testBench_main.vhdl) which initiated the simulation (Code 2). In the test-bench code, first we included the ‘main.vhdl’ code in order to create the simulation on top of that base design. To use the base design in the test-bench code we first initiated a label, ‘UUT’ and to that label we assigned the base code, ‘main’, before using the PORT MAP function. Since the test-bench code is self-contained (thus there is nothing in between the ‘entity’ and ‘end’ labels), we the connect our local signals to the module variables, inputs and output, by using PORT MAP function. After the PROCESS label we created the structure of our wave form by referencing the truth table (Figure 1.2) with a 50ns delay between each combinational result (Figure 1.3).

in1	in2	in3	in4	out1
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Figure 1.2 The truth table of the circuit combination

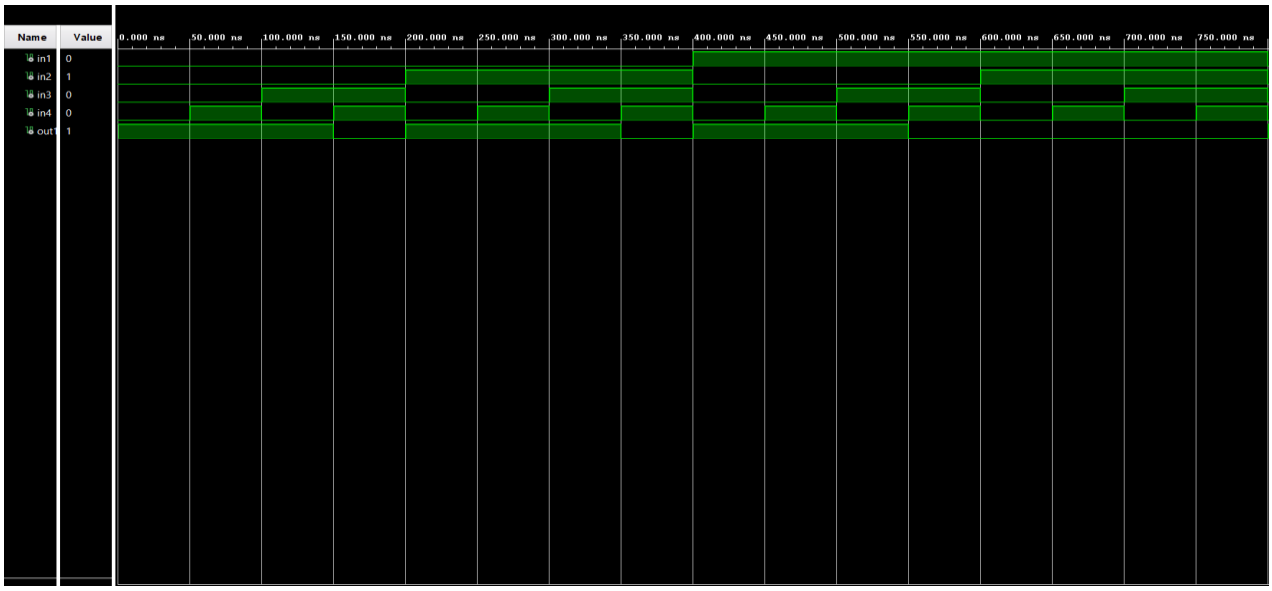

Figure 1.3 The corresponding waveform representation of the given truth table

In order to assign switches to inputs and led indicator to the output from the Basys3, we included the constraint file (Code 3). The assigned values are:

In1: V17 switch
In2: V16 switch
In3: W16 switch
In4: W17 switch
Out1: U16 LED

In the final step, we implemented the VHDL design on the Basys3. There we can see some observations from the FPGA.

When the combination is Figure 1.4:

in1: 0

in2: 0

in3: 0

in4: 0

out1: 1

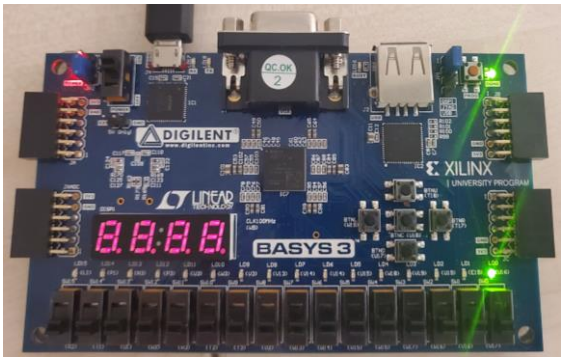

Figure 1.4 result of combination 0-0-0-0

When the combination is Figure 1.5:

in1: 1

in2: 0

in3: 1

in4: 1

out1: 0

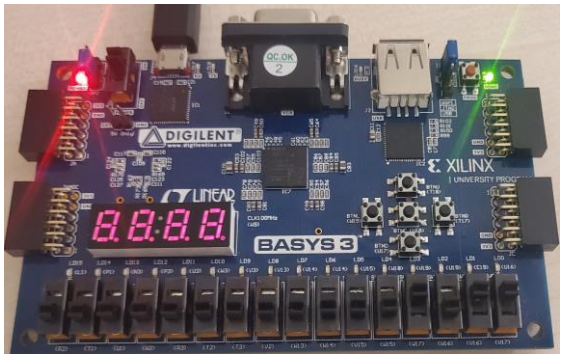

Figure 1.5 result of combination 1-0-1-1

When the combination is Figure 1.6:

in1: 1

in2: 1

in3: 1

in4: 1

out1: 0

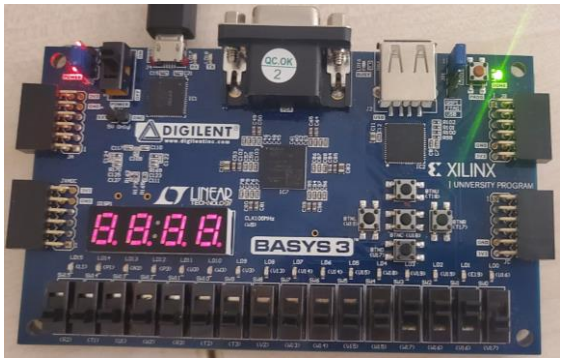

Figure 1.6 result of combination 1-1-1-0

When the combination is Figure 1.7:

in1: 1

in2: 0

in3: 0

in4: 1

out1: 1

Figure 1.7 result of combination 1-0-0-1

When the combination is Figure 1.8:

in1: 0

in2: 1

in3: 1

in4: 0

out1: 1

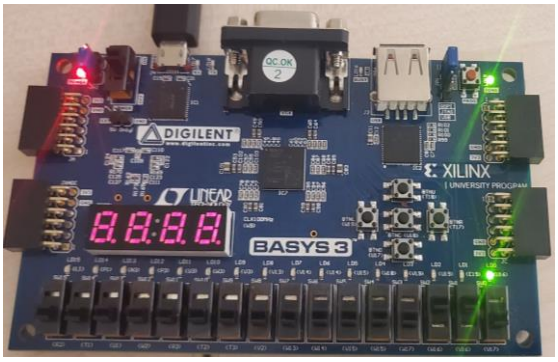

Figure 1.8 result of combination 0-1-1-0

When the combination is Figure 1.9:

in1: 1

in2: 1

in3: 1

in4: 1

out1: 0

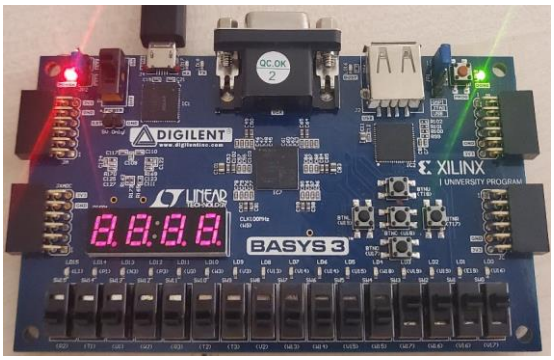

Figure 1.9 result of combination 1-1-1-1

Conclusion:

In this lab we are introduced to the basics of digital design via using VHDL and FPGA. In order to do so we implemented a real life case via initialing combinations of logic gates. The experiment results are matching with the theoretical outputs thus the results are successful. The conclusions from this experiment is closely related with the learnings we have in EE-102 class such that we had the opportunity to implement the methods when constructing a gate combination for the desired result by first implementing the truth table and using minterm rules to construct the gate schematic. Since this experiment had little to no place for human error, the only room for error is due to technical and implementation errors; which I have not observed any when comparing the results from the experiment with the theory.

Appendices:

Code 1: main.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity main is
    Port ( in1 : in STD_LOGIC;
           in2 : in STD_LOGIC;
           in3 : in STD_LOGIC;
           in4 : in STD_LOGIC;
           out1 : out STD_LOGIC);
end main;
architecture Behavioral of main is
begin
-- This is where we designed our gate combination.
out1 <= ((in1 and in2) nor (in3 and in4));
end Behavioral;
```

Code 2: testBench_main.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity testBench_main is
end testBench_main;
architecture Behavioral of testBench_main is
component main
    PORT(
        in1: in STD_LOGIC;
```

```
        in2: in STD_LOGIC;
        in3: in STD_LOGIC;
        in4: in STD_LOGIC;
        out1: out STD_LOGIC);
end component;
    signal in1: STD_LOGIC;
    signal in2: STD_LOGIC;
    signal in3: STD_LOGIC;
    signal in4: STD_LOGIC;
    signal out1: STD_LOGIC;
begin
UUT: main PORT MAP(
in1 => in1,
in2 => in2,
in3 => in3,
in4 => in4,
out1 => out1
 );
testBench_main: PROCESS
    begin
    -- 0000
        in1<='0';
        in2<='0';
        in3<='0';
        in4<='0';
```

```
-- 0001
wait for 50 ns;
    in1<='0';
    in2<='0';
    in3<='0';
    in4<='1';
-- 0010
wait for 50 ns;
    in1<='0';
    in2<='0';
    in3<='1';
    in4<='0';
-- 0011
wait for 50 ns;
    in1<='0';
    in2<='0';
    in3<='1';
    in4<='1';
-- 0100
wait for 50 ns;
    in1<='0';
    in2<='1';
    in3<='0';
    in4<='0';
-- 0101
```

```
wait for 50 ns;
    in1<='0';
    in2<='1';
    in3<='0';
    in4<='1';
-- 0110
wait for 50 ns;
    in1<='0';
    in2<='1';
    in3<='1';
    in4<='0';
-- 0111
wait for 50 ns;
    in1<='0';
    in2<='1';
    in3<='1';
    in4<='1';
-- 1000
wait for 50 ns;
    in1<='1';
    in2<='0';
    in3<='0';
    in4<='0';
-- 1001
wait for 50 ns;
```

```
    in1<='1';
    in2<='0';
    in3<='0';
    in4<='1';
-- 1010
wait for 50 ns;
    in1<='1';
    in2<='0';
    in3<='1';
    in4<='0';
-- 1011
wait for 50 ns;
    in1<='1';
    in2<='0';
    in3<='1';
    in4<='1';
-- 1100
wait for 50 ns;
    in1<='1';
    in2<='1';
    in3<='0';
    in4<='0';
-- 1101
wait for 50 ns;
    in1<='1';
```

```
        in2<='1';
        in3<='0';
        in4<='1';
    -- 1110
    wait for 50 ns;
        in1<='1';
        in2<='1';
        in3<='1';
        in4<='0';
    -- 1111
    wait for 50 ns;
        in1<='1';
        in2<='1';
        in3<='1';
        in4<='1';
    wait for 50ns;
end PROCESS;
end Behavioral;
```

Code 3: main_constraints.xdc

```
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 W17 [get_ports {in4}]

set_property IOSTANDARD LVCMOS33 [get_ports {in4}]

set_property PACKAGE_PIN U16 [get_ports {out1}]

set_property IOSTANDARD LVCMOS33 [get_ports {out1}]
```