Sait Sarper Özaslan

102-2

7/11/2022

Lab 4 Report: Arithmetic Logic Unit(ALU)

Purpose:

The purpose of this lab is to learn how to design an Arithmetic Logic Unit(ALU) that consists of addition, subtraction and 6 more operations which must include at least one bitwise and one bit shift operations.

Design Specifications and Methodology:

To make the ALU specified I needed 8 different operations to fit into specifications wanted. The ALU I made consists of Addition, Substractor, Comparator, Rotate Left, Ones complement, NAND gate, OR gate and XOR gate. While Rotate Left is a bit shift operation, the gate operations and Ones Complement are the bit wise operations in the ALU I made. The left-out operations are arithmetic operations. To visualize the connections between operators an RTL schematic should be used, and this schematic can be further supported through a testbench, which lets me see whether the operations are working properly or not.

To be able to specify between these operations I needed a 3-bit multiplexer which means I needed 3 inputs in the BASYS3 to pick between operations. I decided to use two 3-bit numbers to have more variety between the results of operations. Considering the addition operation being able to give a 4-bit number as its' end result, I decided to use a 4-bit number to show output. For each 3-bit input I needed to put 3 inputs as switches to the BASYS3 and as for outputs I needed to use 4 LEDs to display the output appropriately.

For two 3-bit input numbers' inputs I used X1_in, X2_in X3_in and Y1_in, Y2_in, Y3_in. As for multiplexer's inputs I used Mult(0), Mult(1) and Mult(2). Finally for outputs I used Z as a logic vector '3 downto 0 '.

All of the modules are coded in a modular fashion which gets combined into one big module which I named 'Topmod'.

In other words, I first have,

- Topmod.vhdl

Under this module I have:

- Addition.vhdl
- Substractor.vhdl
- Comparator.vhdl
- Rotateleft.vhdl
- Onescomplement.vhdl
- Nandgate.vhdl
- Orgate.vhdl
- Xorgate.vhdl

To be able to use addition and substractor module I also needed a half adder and full adder which are put under these 2 modules as:

- Half_adder.vhdl
- Full_adder.vhdl

Operations	Input Example	Output Example
“000” /Summation	010 /101	0101
“001”/ Substractor	110/ 101	0001
“010”/ Comparator	101/ 011	0010
“011”/RotateLeft	101	0011
“100”/Onescomplement	110	0001
“101”/Nandgate	101/110	0011
“110”/Orgate	010/100	0110
“111”/ Xorgate	101/011	0110

Figure 1.1 Operations

Results:

To begin with, the ALU was designed through VHDL. I designed the sub module half adder and full adder first to be able to do subtraction and summation. As I created each operations’

module I also added these modules to top module I have made so that I can display it's RTL schematic. The RTL schematic of the ALU is:

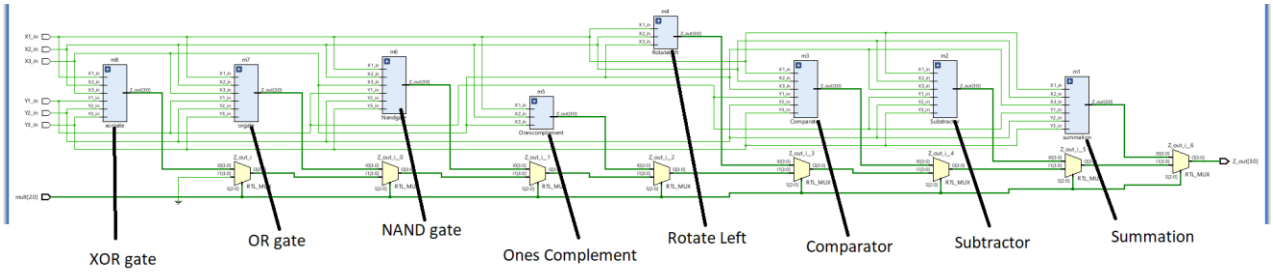

Figure 1.2: The RTL Schematic of ALU

On the simulation there were several options to display, however I only chose 3 of them:

Case 1: I used 111 for 3-bit number for X's inputs and 111 for 3-bit number Y's inputs which can be seen from the figure the results were appropriate to multiplexers results.

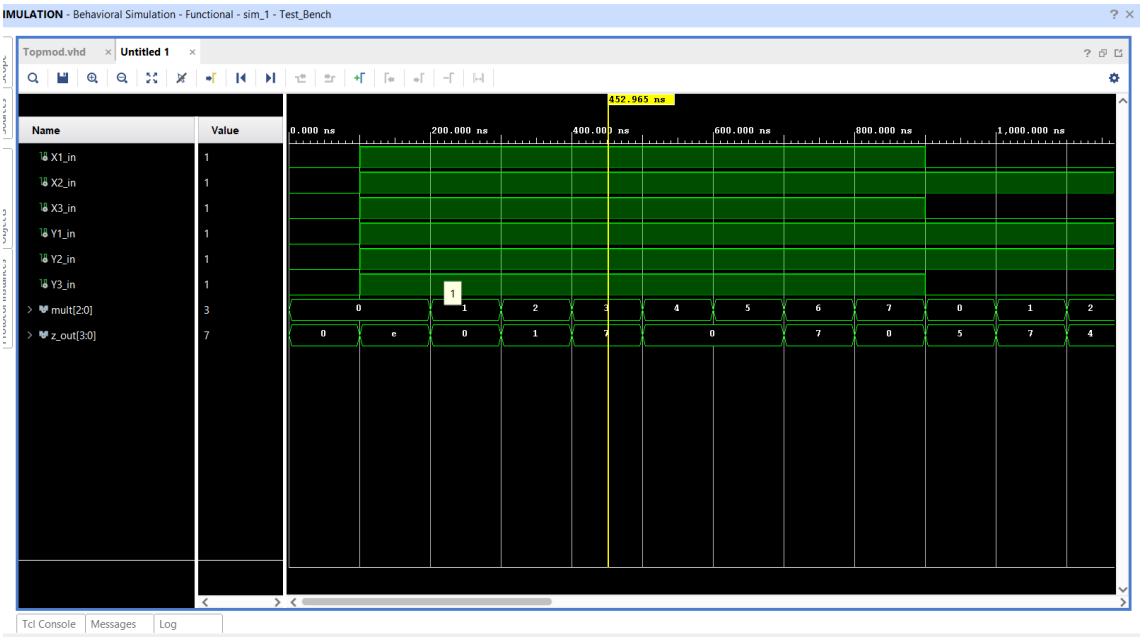

Figure 1.3: Case 111/111

Case 2: I used 010 for X and 111 for Y. There were no undefined values or errors in this part as well.

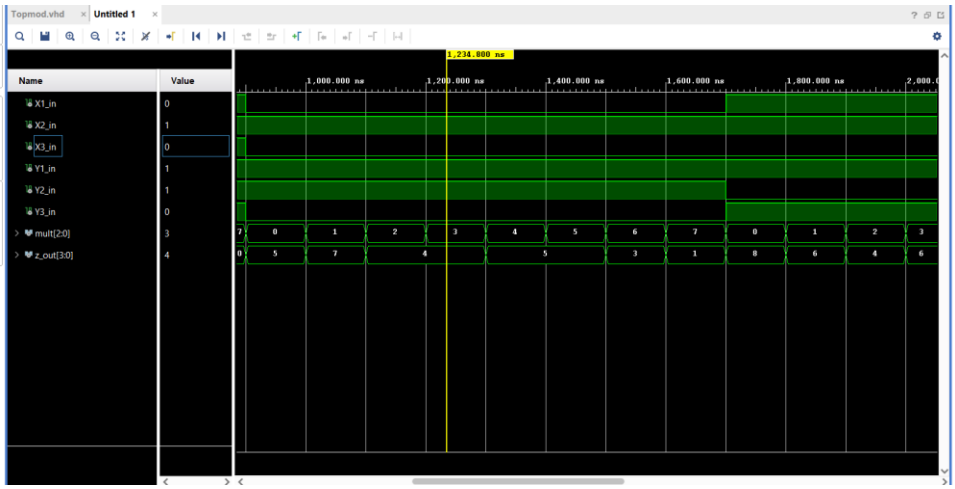

Figure 1.4: Case 010/111

Case 3: I used 011 and 101 for this case. To not get confused I put X3_in to X1_in in most significant bit to least significant bit order. Same for Y3_in to Y1_in as well.

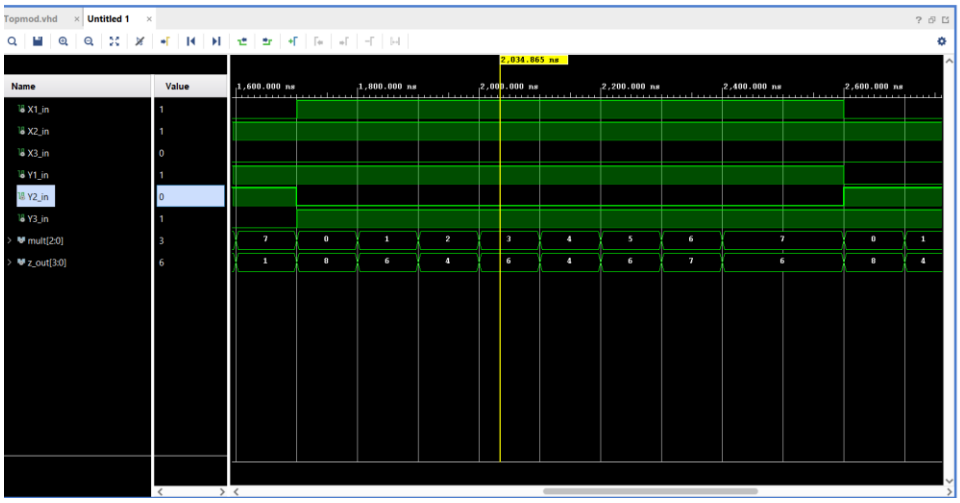

Figure 1.5: Case 011/101

After creating the simulation all there was left is to create the constraint file to implement it into BASYS3. The implementation I used for this lab was:

X1_in: V17 Switch,

X2_in: V16 switch,

X3_in W16 Switch

- I left W 17 switch empty to be able to differentiate easily between inputs

Y1_in: W15 Switch

Y2_in: V15 Switch

Y3_in: W14 Switch

- For multiplexer;

Mult(0) : V1 Switch

Mult(1) : T1 Switch

Mult(2) : R2 Switch

- For Outputs;

Z_out(0): U16 LED

Z_out(1): E19 LED

Z_out(2): U19 LED

Z_Out(3):V19 LED

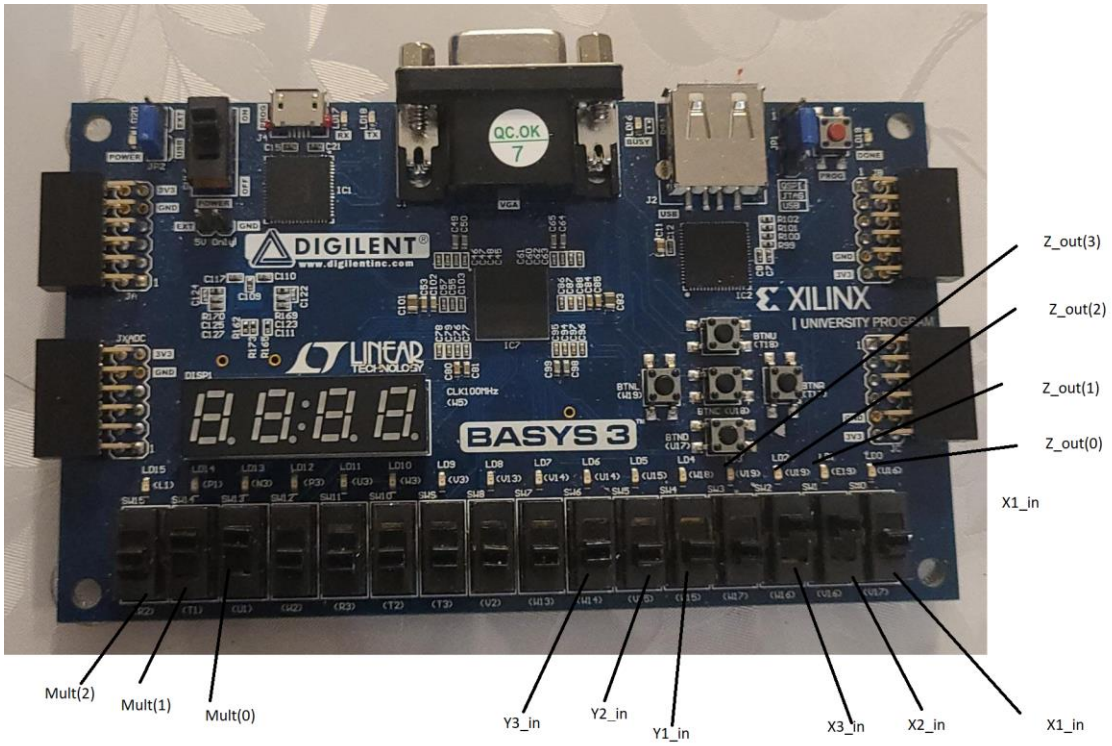

Figure 1.6 : BASYS3 Configuration

Then I started to test the operations I have written on the BASYS3. Here are all 8 cases and examples for each one of them. I have put small papers in between switches so that the observations can be made easier.

Summation:

Multiplexer signal: 000

3-bit X Input: 111

3-bitY Input:110

Expected Result:1101

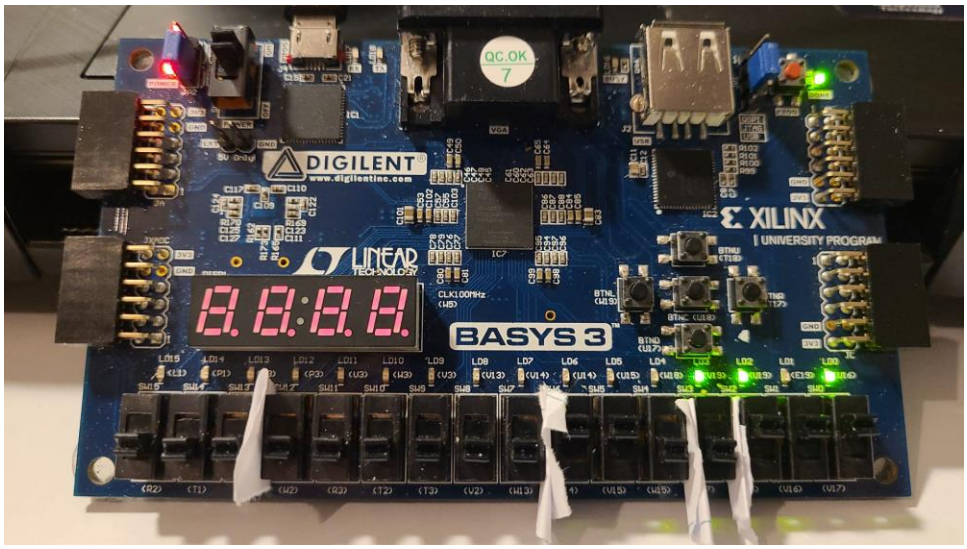

Figure 1.7: Case 111/101 on Summation

Subtractor:

Multiplexer Signal: 001

3-bit X Input:101

3-bit Y Input:011

Expected Result:0010

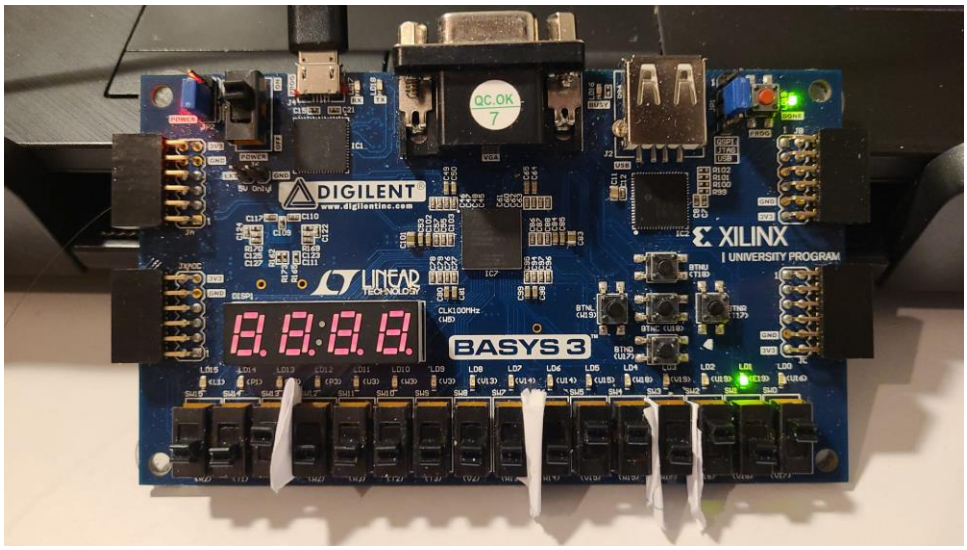

Figure 1.8: Case 101/011 on Subtraction

Comparator:

When Z_out(2) turns on it is less than, when Z_out(1) turns on the result is greater than and when Z_out(0) turns on it is equal.

Multiplexer Signal: 010

3-bit X Input: 101

3-bit Y Input:101

Expected Result:0001

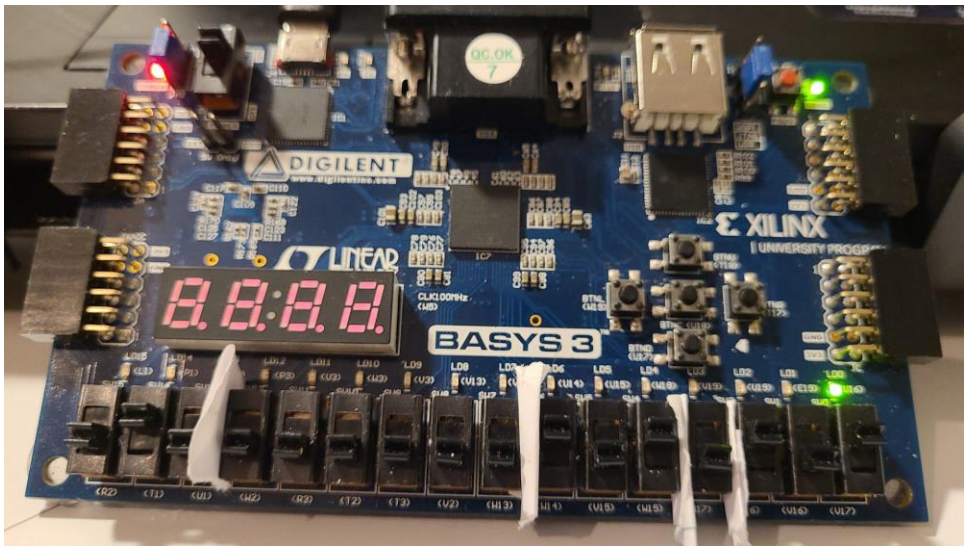

Figure 1.9: Case 101/101 on Comparator

Rotate Left

Multiplexer Signal:011

3-bit X Input:101

Expected Result:0011

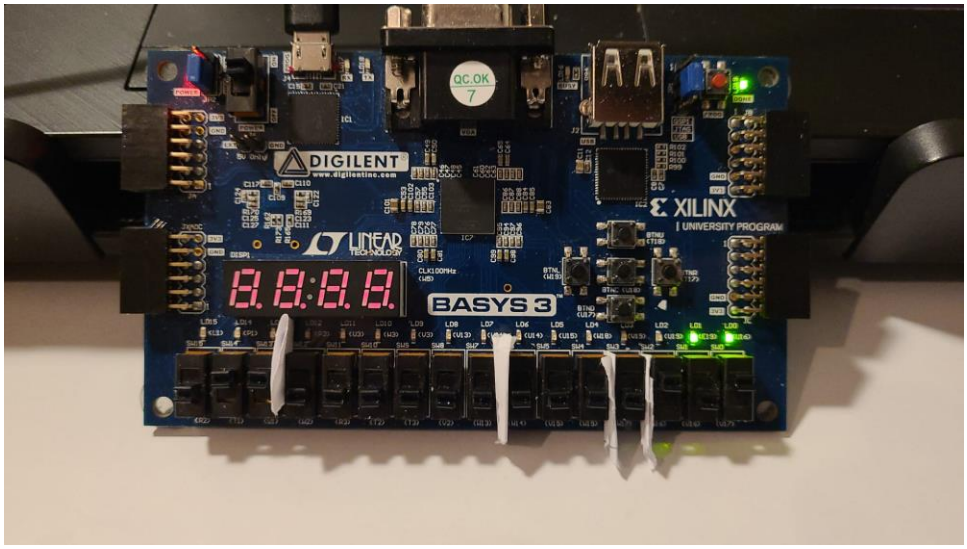

Figure 1.10: Case 101 on Rotate Left

Ones Complement:

Multiplexer Signal: 101

3-bit X Input: 110

Expected Result:0001

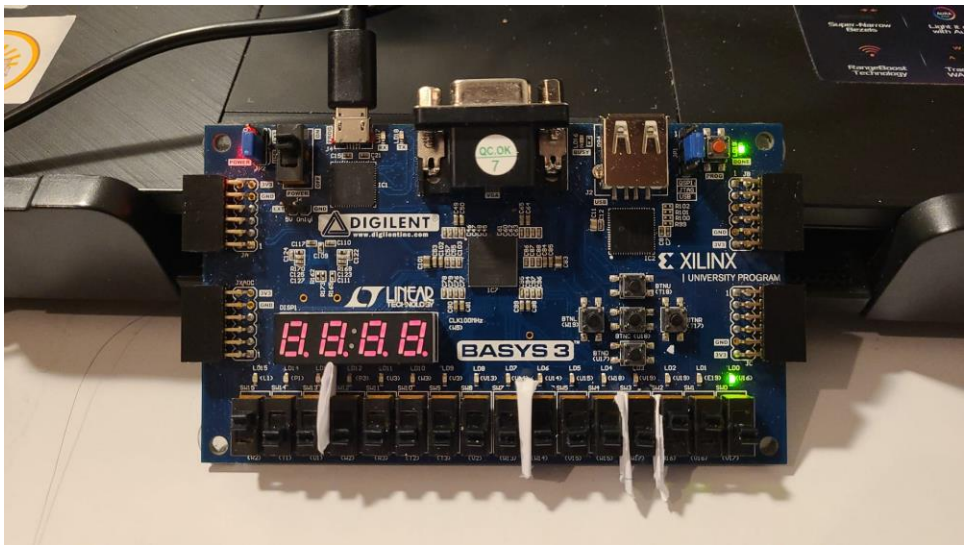

Figure 1.11 Case 110 on Ones Complement

NAND Gate

Multiplexer Signal: 101

3-bit X Input:110

3-bit Y Input:010

Expected Result:0101

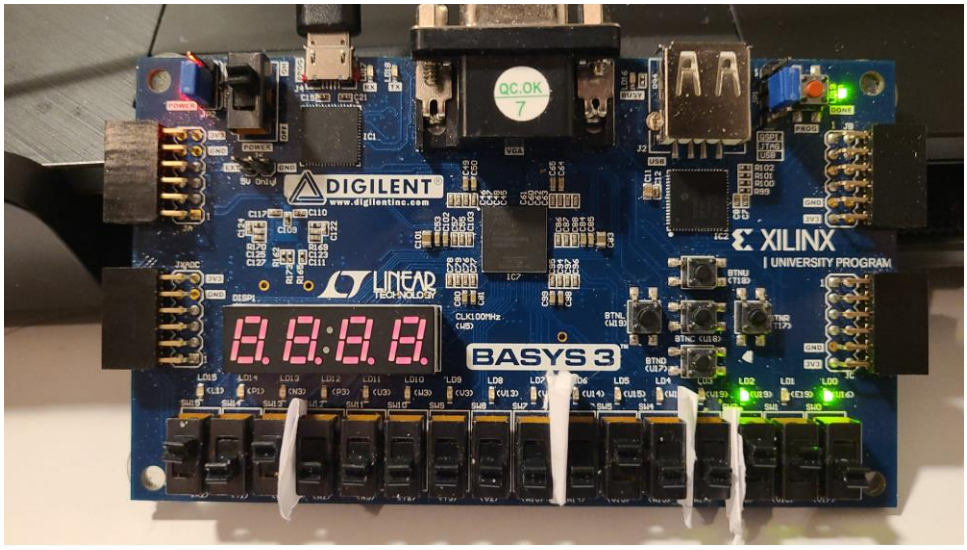

Figure 1.12: Case 110/010 on NAND Gate

OR Gate

Multiplexer Signal:110

3-bit X Input: 110

3-bit Y Input:101

Expected Result:0111

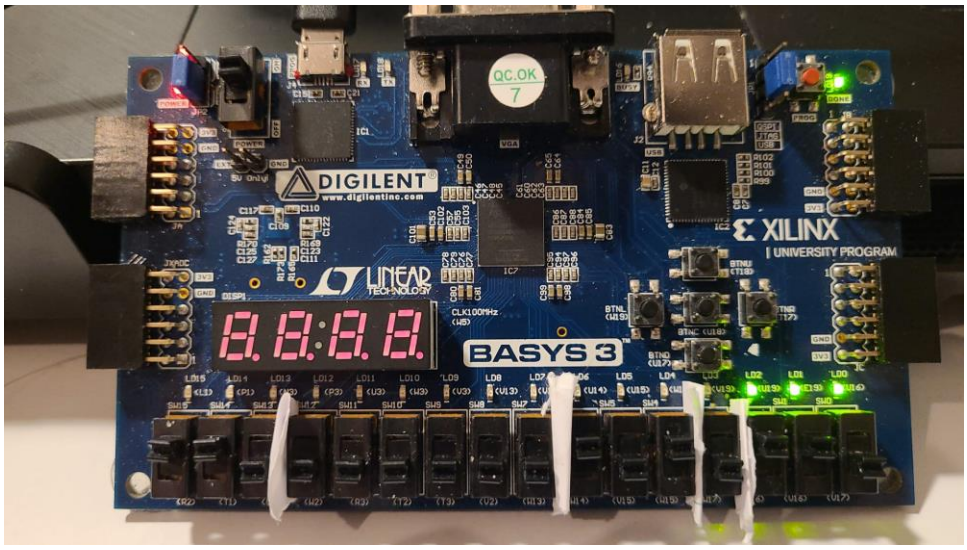

Figure 1.13: Case 110/101 on OR Gate

XOR Gate

Multiplexer Signal:111

3-bit X Input:101

3-bit Y Input:100

Expected Result:0001

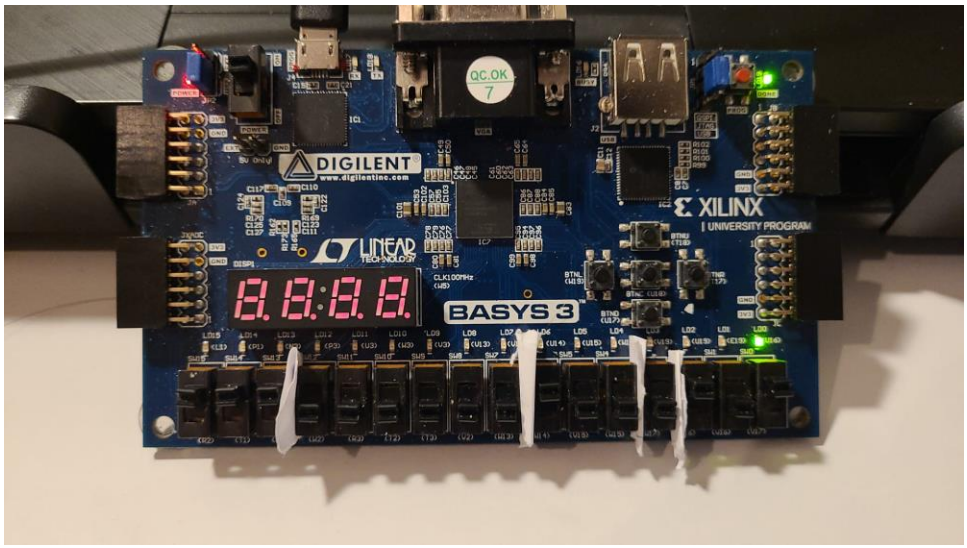

Figure 1.14: Case 101/100 on XOR Gate

It can be seen from the examples the expected results aligned with the actual results.

Conclusion:

Throughout this lab, I implemented an ALU into the BASYS3. To implemented this ALU I wrote codes for eight operations. One error I had to struggled with was the comparator's VHDL code as I firstly started the write the code through codes lengthy Boolean expression. This required me to add several signals to the code making it more confusing. To fix this issue I examined the comparators gate structure which we had learnt in class and implemented the comparator accordingly. One other issue I faced was with the subtractor as I realized there needs to be twos complement code and a comparator for the cases where 3-bit input Y is bigger than 3-bit X. Otherwise, the subtractor works perfectly fine. Finally, there were no significant errors in testbench which allowed me to observe the behavior of ALU easily.

Appendix:

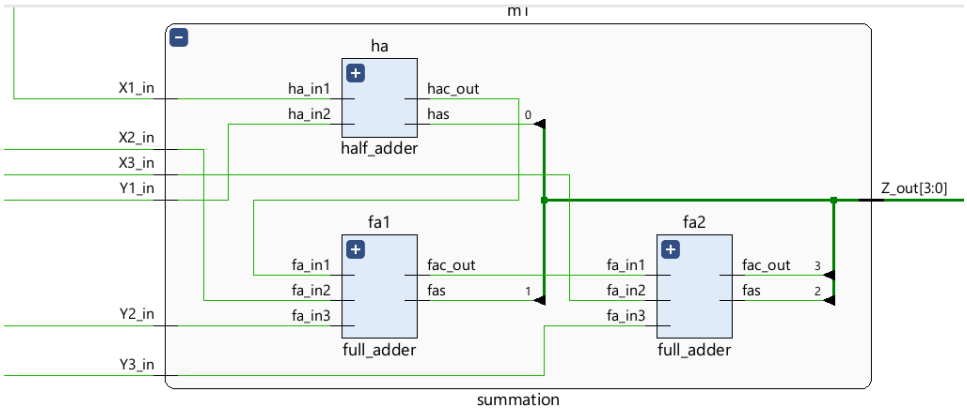

Figure 1.15: RTL Schematic for Summation

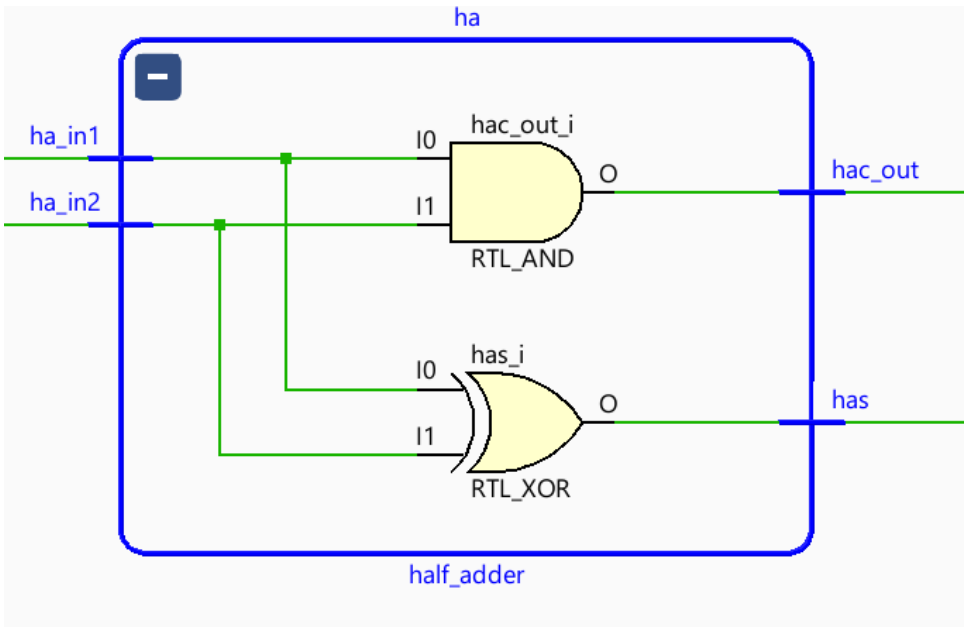

Figure 1.16: RTL Schematic of Half Adder

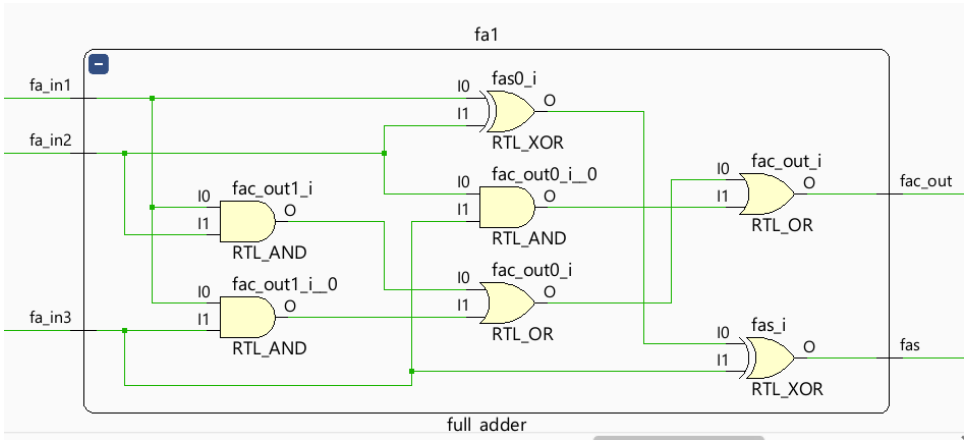

Figure 1.17: RTL Schematic of Full Adder

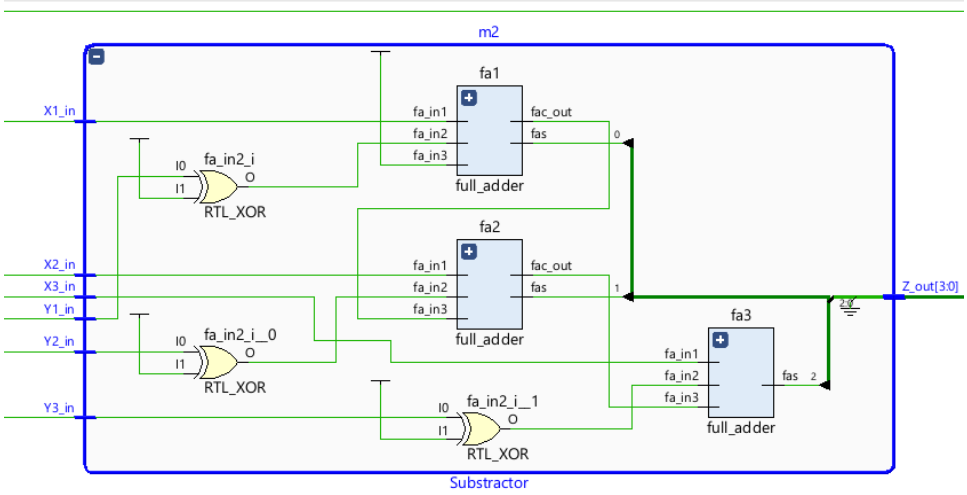

Figure 1.18: RTL Schematic of Subtractor

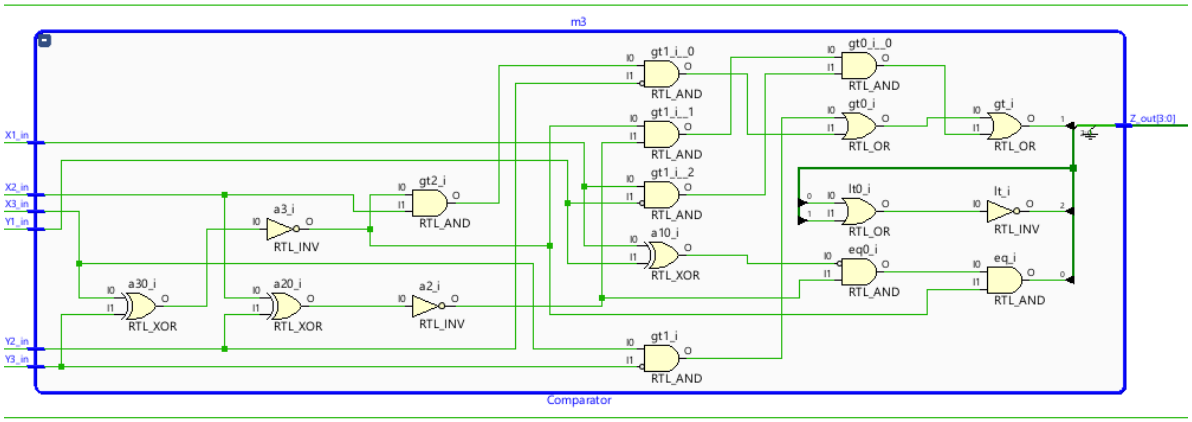

Figure 1.19: RTL Schematic of Comparator

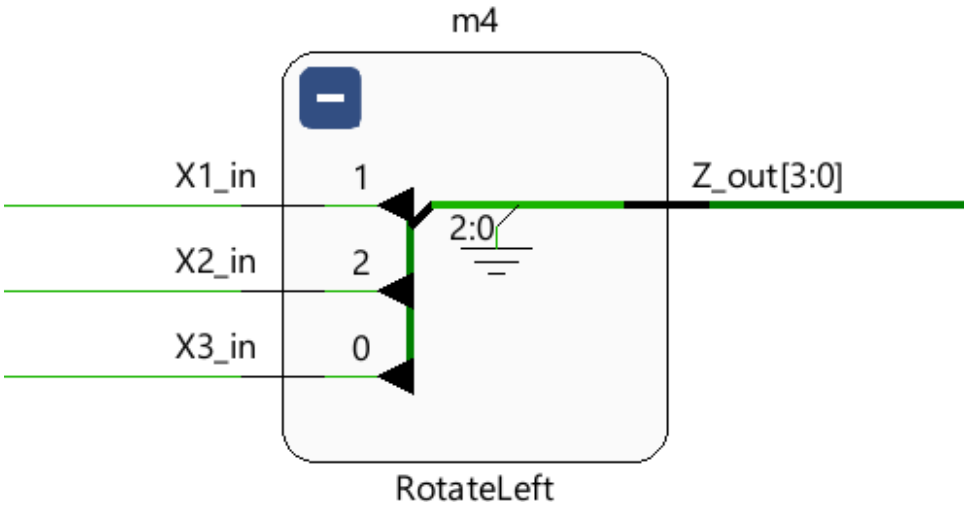

Figure 1.20: RTL Schematic of Rotate Left

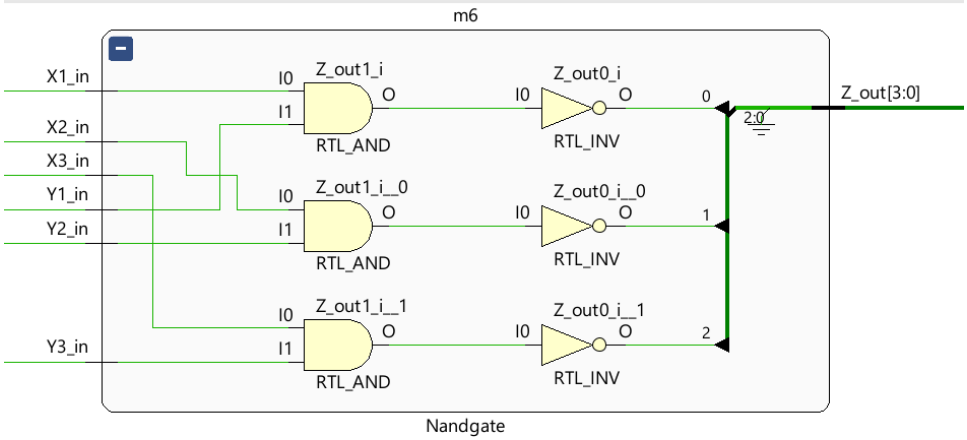

Figure 1.21: RTL Schematic of NAND Gate

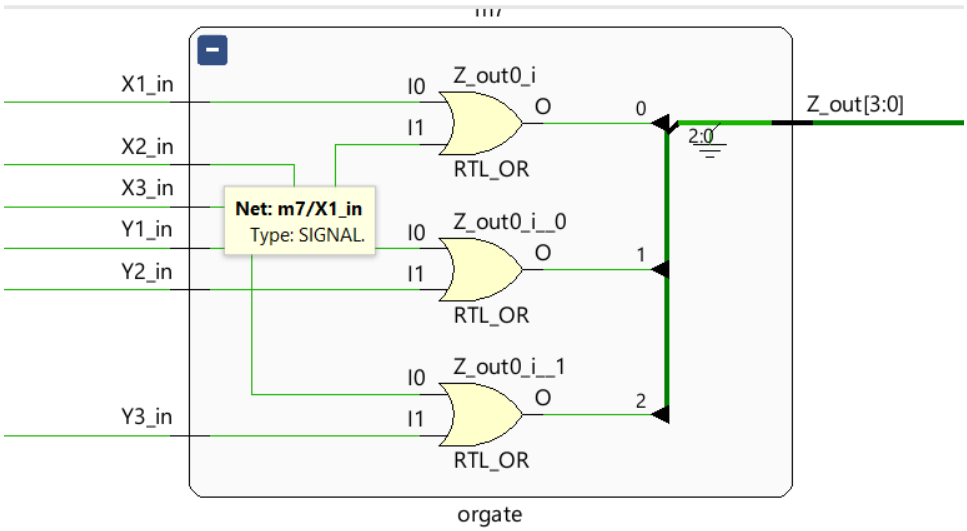

Figure 1.22: RTL Schematic of OR Gate

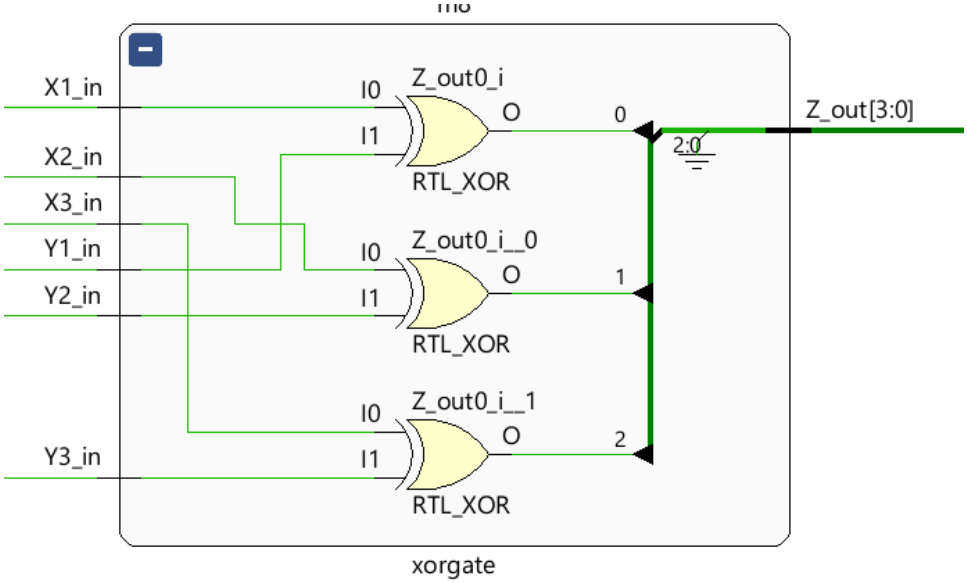

Figure 1.23: RTL Schematic of XOR Gate

Topmod.vhdl

```
entity Topmod is
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
           X3_in : in STD_LOGIC;
           Y1_in : in STD_LOGIC;
           Y2_in : in STD_LOGIC;
           Y3_in : in STD_LOGIC;
           mult : in STD_LOGIC_vector(2 downto 0);
           Z_out : out STD_LOGIC_vector(3 downto 0));
end Topmod;

architecture Behavioral of Topmod is
component summation
Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
           X3_in : in STD_LOGIC;
           Y1_in : in STD_LOGIC;
           Y2_in : in STD_LOGIC;
           Y3_in : in STD_LOGIC;
           Z_out : out STD_LOGIC_vector(3 downto 0));
end component ;
--substractor

component substractor
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
```

```
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end component;
-- comparator
component comparator
    Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;
-- Rotate left
component Rotateleft
Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;
-- Onescomplent
component Onescomplement
Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
```

```
        X3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;
--Nand gate
component Nandgate
Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;
--orgate
component orgate
Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;

--xorgate
component xorgate
Port ( X1_in : in STD_LOGIC;
```

```
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out std_logic_vector (3 downto 0));
end Component;



signal X1in,x2in,x3in,y1in,y2in,y3in: std_logic;
signal out1, out2,out3,out4,out5,out6,out7,out8: std_logic_vector(3 downto 0);
begin
X1in <= X1_in;
X2in <= X2_in ;
x3in <= X3_in;
y1in <= Y1_in ;
y2in <= y2_in;
y3in <= Y3_in;

m1: summation
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out1);

m2: substractor
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out2);
```

```
m3: comparator
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out3);

m4: Rotateleft
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
       Z_out => out4);

m5: Onescomplement
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
       Z_out => out5);
m6: Nandgate
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out6);

m7: orgate
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out7);

m8: xorgate
port map(X1_in =>X1in , X2_in => x2in , X3_in => x3in,
      Y1_in => Y1in, Y2_in =>y2in, Y3_in =>y3in , Z_out => out8);

process (mult,out1, out2,out3,out4,out5,out6,out7,out8)
begin
if mult = "000" then
```

```
Z_out <= out1;
elsif mult = "001" then
Z_out <= out2;
elsif mult = "010" then
Z_out <= out3;
elsif mult = "011" then
Z_out <= out4;
elsif mult = "100" then
Z_out <= out5;
elsif mult = "101" then
Z_out <= out6;
elsif mult = "110" then
Z_out <= out7;
elsif mult = "111" then
Z_out <= out8;
else
Z_out <= "0000";
end if;
end process;



end Behavioral;
```

Summation.vhdl

```
entity summation is
    Port ( X1_in : in STD_LOGIC;
```

```
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out STD_LOGIC_vector(3 downto 0));
end summation;

architecture Behavioral of summation is
--half adder
component half_adder
Port ( ha_in1 : in STD_LOGIC;
        ha_in2 : in STD_LOGIC;
        has : out STD_LOGIC;
        hac_out : out STD_LOGIC);
end component;
--component full_adder
component full_adder
Port ( fa_in1 : in STD_LOGIC;
        fa_in2 : in STD_LOGIC;
        fa_in3 : in STD_LOGIC;
        fas : out STD_LOGIC;
        fac_out : out STD_LOGIC);
end component;
signal hc, fac1: std_logic;

begin
```

```
ha: half_adder
   Port map(ha_in1=> x1_in, ha_in2 => y1_in, has=>Z_out(0), hac_out=> hc);
fa1: full_adder
   Port map(fa_in1 => hc, fa_in2 => x2_in, fa_in3 => y2_in, fas => Z_out(1), fac_out=>fac1);
fa2: full_adder
   Port map(fa_in1 => fac1, fa_in2 => x3_in, fa_in3 => y3_in, fas=> Z_out(2), fac_out =>
Z_out(3));



end Behavioral;
```

Half adder.vhdl

```
entity half_adder is
   Port ( ha_in1 : in STD_LOGIC;
          ha_in2 : in STD_LOGIC;
          has : out STD_LOGIC;
          hac_out : out STD_LOGIC);
end half_adder;

architecture Behavioral of half_adder is



begin

has <= ha_in1 xor ha_in2;
hac_out <= ha_in1 and ha_in2;
end Behavioral;
```

Full_adder.vhdl

```
entity full_adder is
__Port ( fa_in1 : in STD_LOGIC;
_____fa_in2 : in STD_LOGIC;
_____fa_in3 : in STD_LOGIC;
_____fas : out STD_LOGIC;
_____fac_out : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is

begin
fas<= fa_in1 xor fa_in2 xor fa_in3 ;
fac_out <= (fa_in1 and fa_in2) or (fa_in1 and fa_in3) or (fa_in2 and fa_in3) ;

end Behavioral;
```

Substractor.vhdl

```
entity Substractor is
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
           X3_in : in STD_LOGIC;
           Y1_in : in STD_LOGIC;
           Y2_in : in STD_LOGIC;
           Y3_in : in STD_LOGIC;
           Z_out : out std_logic_vector (3 downto 0));
```

```
end Substractor;


architecture Behavioral of Substractor is
component Full_adder
Port ( fa_in1 : in STD_LOGIC;
       fa_in2 : in STD_LOGIC;
       fa_in3 : in STD_LOGIC;
       fas : out STD_LOGIC;
       fac_out : out STD_LOGIC);
end component ;
signal fc1,fc2,s: std_logic;
begin
fa1: Full_adder
port map(fa_in1 => X1_in, fa_in2 => (Y1_in XOR '1'), fa_in3 => '1', fas => Z_out(0), fac_out =>
fc1);
fa2: Full_adder
port map(fa_in1 => X2_in, fa_in2 => (Y2_in Xor '1'), fa_in3 => fc1, fas => Z_out(1), fac_out =>
fc2);
fa3: Full_adder
port map(fa_in1 => X3_in , fa_in2 => (Y3_in XOR '1'), fa_in3 => fc2, fas => Z_out(2), fac_out =>
s);
Z_out(3) <='0';
end Behavioral;
```

Comparator.vhdl

```
entity Comparator is
   Port ( X1_in : in STD_LOGIC;
          X2_in : in STD_LOGIC;
          X3_in : in STD_LOGIC;
```

```
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out STD_LOGIC_VECTOR (3 downto 0));
end Comparator;

architecture Behavioral of Comparator is
signal a1, a2,a3,gt,lt,eq: std_logic ;
begin
a1 <= not (X1_in xor Y1_in);
a2 <= not (X2_in xor Y2_in);
a3 <= not (X3_in xor Y3_in );
eq <= a1 and a2 and a3;
gt <= (X3_in and (not Y3_in)) or (a3 and X2_in and(not Y2_in)) or (a3 and a2 and (X1_in and (not
Y1_in)));
lt <= not(eq or gt);
Z_out (0) <= eq;
Z_out (1) <= gt;
Z_out (2) <= lt;
Z_out(3)<='0';


end Behavioral;
```

Rotateleft.vhdl

```
entity RotateLeft is
    Port ( X1_in : in STD_LOGIC;
```

```
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Z_out : out STD_LOGIC_Vector(3 downto 0));
end RotateLeft;

architecture Behavioral of RotateLeft is

begin
Z_out(0)<= X3_in ;
Z_out(1)<= X1_in ;
Z_out(2)<= X2_in;
Z_out (3) <= '0';

end Behavioral;
```

Onescomplement.vhdl

```
entity Onescomplement is
    Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
        Z_out : out STD_LOGIC_Vector(3 downto 0));
end Onescomplement;

architecture Behavioral of Onescomplement is

begin
Z_out(0) <= not X1_in;
Z_out(1) <= not X2_in;
```

```
Z_out(2) <= not X3_in;
Z_out (3) <= '0';


end Behavioral;
```

Nandgate.vhdl

```
entity Nandgate is
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
           X3_in : in STD_LOGIC;
           Y1_in : in STD_LOGIC;
           Y2_in : in STD_LOGIC;
           Y3_in : in STD_LOGIC;
           Z_out : out STD_LOGIC_Vector(3 downto 0));
end Nandgate;

architecture Behavioral of Nandgate is

begin
Z_out(0) <= not (X1_in and Y1_in);
Z_out(1) <= not (X2_in and Y2_in);
Z_out(2) <= not (X3_in and Y3_in);
Z_out (3) <= '0';
end Behavioral;
```

orgate.vhdl

```
entity orgate is
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
           X3_in : in STD_LOGIC;
           Y1_in : in STD_LOGIC;
           Y2_in : in STD_LOGIC;
           Y3_in : in STD_LOGIC;
           Z_out : out STD_LOGIC_Vector(3 downto 0));
end orgate;

architecture Behavioral of orgate is

begin
Z_out(0) <=  (X1_in or Y1_in);
Z_out(1) <=  (X2_in or Y2_in);
Z_out(2) <=  (X3_in or Y3_in);
Z_out (3) <= '0';


end Behavioral;
```

Xorgate.vhdl

```
entity xorgate is
    Port ( X1_in : in STD_LOGIC;
           X2_in : in STD_LOGIC;
```

```
        X3_in : in STD_LOGIC;
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        Z_out : out STD_LOGIC_Vector(3 downto 0));
end xorgate;

architecture Behavioral of xorgate is

begin
Z_out(0) <=  (X1_in xor Y1_in);
Z_out(1) <=  (X2_in xor Y2_in);
Z_out(2) <=  (X3_in xor Y3_in);
Z_out (3) <= '0';
end Behavioral;
```

Test_bench.vhdl

```
entity Test_Bench is

end Test_Bench;
architecture Behavioral of Test_Bench is

component Topmod
    Port ( X1_in : in STD_LOGIC;
        X2_in : in STD_LOGIC;
        X3_in : in STD_LOGIC;
```

```
        Y1_in : in STD_LOGIC;
        Y2_in : in STD_LOGIC;
        Y3_in : in STD_LOGIC;
        mult : in STD_LOGIC_vector(2 downto 0);
        Z_out : out STD_LOGIC_vector(3 downto 0));
end component;
signal X1_in, X2_in, X3_in, Y1_in, Y2_in, Y3_in: std_logic ;
signal mult: std_logic_vector (2 downto 0);
signal z_out: std_logic_vector (3 downto 0);
begin
a1: Topmod
   port map( X1_in => X1_in, X2_in => X2_in, X3_in => X3_in, Y1_in => Y1_in,
   Y2_in => Y2_in, Y3_in => Y3_in ,mult=> mult ,z_out =>z_out );
Testbench: process
begin
mult <= "000";
X1_in <= '0';
X2_in <= '0';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '0';
Y3_in <= '0';
wait for 100ns;
mult <= "000";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
```

```
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "001";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "010";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "011";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
```

```
wait for 100ns;
mult <= "100";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "101";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "110";
X1_in <= '1';
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "111";
X1_in <= '1';
```

```
X2_in <= '1';
X3_in <= '1';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '1';
-----------------------------------------------
wait for 100ns;
mult <= "000";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
wait for 100ns;
mult <= "001";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
wait for 100ns;
mult <= "010";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
```

```
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
wait for 100ns;
mult <= "011";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
wait for 100ns;
mult <= "100";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';

wait for 100ns;
mult <= "101";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
```

```
Y3_in <= '0';
wait for 100ns;
mult <= "110";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
wait for 100ns;
mult <= "111";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '1';
Y3_in <= '0';
-----------------------------------------------
wait for 100ns;
mult <= "000";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
```

```
mult <= "001";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "010";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "011";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "100";
X1_in <= '1';
X2_in <= '1';
```

```
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "101";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "110";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
Y3_in <= '1';
wait for 100ns;
mult <= "111";
X1_in <= '1';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '1';
Y2_in <= '0';
```

```
Y3_in <= '1';
wait for 100ns;
-----------------------------------
wait for 100ns;
mult <= "000";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "001";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "010";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
```

```
wait for 100ns;
mult <= "011";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "100";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';

wait for 100ns;
mult <= "101";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "110";
```

```
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
wait for 100ns;
mult <= "111";
X1_in <= '0';
X2_in <= '1';
X3_in <= '0';
Y1_in <= '0';
Y2_in <= '1';
Y3_in <= '1';
end process;


end Behavioral;
```