Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

LAB 04: ARITHMETIC LOGIC UNIT

Purpose

The purpose of this experiment is to design an arithmetic logic unit (ALU) that can perform 8 operations including addition, subtraction, and at least one bitwise and one shift operation. The ALU is designed using VHDL in a modular fashion and it is implemented on the BASYS3 board.

Methodology

I first decided on the eight operations that would be implemented. They are addition, multiplication, logical shift, rotate, ones' complement, bitwise AND, bitwise OR, and bitwise XOR. The operations will function with two 3-bit numbers X and Y. To implement the switch that decides the operation, I first created separate modules for the operations then I used them as sub-modules of the top module. The operation is then selected based on the value of the switch input. I generated the RTL schematics for the ALU and the operators to see whether the pins are connected appropriately. I also used test bench codes for the operations to check if there exists any problem in their functioning. After creating the constraint file and generating the bitstream, I was able to implement the ALU on the BASYS3 board. I used the switches for the inputs and displayed the results using the LEDs.

Design Specifications

There are 7 inputs and 1 output. The operation is selected by the user using the 3-bit input, sel(sel(0), sel(1), sel(2)). Then the bits of the first 3-bit number X are separately input by the user to inputs x1, x2, x3 and likewise for the second number Y to the inputs y1, y2, y3. The 4-bit output out_alu is displayed by the LEDs of the BASYS3 board.

The code for the ALU is written in a modular fashion, in other words, the eight operations of the ALU are designed as sub-modules of the main module, "alu.vhd". The main module outputs the result of the operation selected by the user. The eight sub-modules under the main modules are:

- addition.vhd
- subtraction.vhd
- logical_shift.vhd
- rotate.vhd
- ones_complement.vhd
- and_gate.vhd
- or_gate.vhd
- xor_gate.vhd

For the addition and subtraction modules, I designed two sub-modules that function as one-bit half adder and one-bit full adder:

- half_adder.vhd
- full_adder.vhd

Every operation except for subtraction inputs and outputs unsigned numbers. The subtractor inputs two unsigned numbers and outputs a 3-bit signed number. Figure 1 shows the select inputs and the corresponding operations.

Select (sel)	**Operation**	**Inputs**	**Outputs (from MSB to LSB)**	**Example**
“000”	Addition	X (x1, x2, x3), Y (y1, y2, y3)	X+Y	“110” + ”011” = “1001”
“001”	Subtraction	X, Y	X-Y	“110” – “011” = “011”
“010”	Logical Shift	X	“0”X2 X1 “0”	“110” => “100”
“011”	Rotate	X	“0” X2 X1 X3	“110” => “101”
“100”	Ones’ Complement	X	“0” (NOT x3) (NOT x2) (NOT x1)	“110” =>”001”
“101”	Bitwise AND Gate	X, Y	“0” (x3 AND y3) (x2 AND y2) (x1 AND y1)	“110” AND “011” => “010”
“110”	Bitwise OR Gate	X, Y	“0” (x3 OR y3) (x2 OR y2) (x1 OR y1)	“110” OR “011” => “111”
“111”	Bitwise XOR Gate	X, Y	“0” (x3 XOR y3) (x2 XOR y2) (x1 XOR y1)	“110” XOR “011” => “101”

Figure 1: Table of selections and operations

The ALU is then implemented on the BASYS3 board. The switches and LEDs of the board corresponding to the inputs and outputs are:

x1: V17 switch

x2: V16 switch

x3: W16 switch

y1: W15 switch

y2: V15 switch

y3: W14 switch

sel(0): U1 switch

sel(1): T1 switch

sel(2): R2 switch

out_alu(0): U16 LED

out_alu(1): E19 LED

out_alu(2): U19 LED

out_alu(3): V19 LED

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

Results

After writing the code, I generated the RTL schematics for the main module and the operator modules. Figure 2 shows the RTL schematic for the arithmetic logic unit. The cells (blue boxes) at the bottom represent the sub-modules (operators) and the operation is chosen by the multiplexers above (based on the value of sel). The cells from right to left represent addition, multiplication, logical shift, rotate, ones' complement, bitwise AND, bitwise OR, and bitwise XOR respectively.

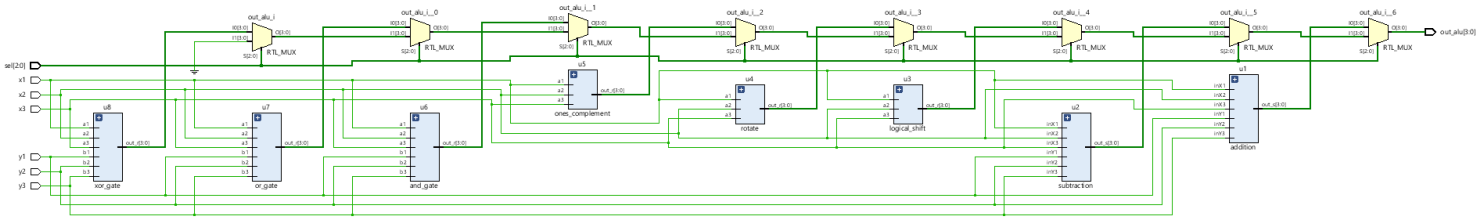

Figure 2: RTL schematic for the ALU

Figures 3-12 display the RTL schematics for the operators.

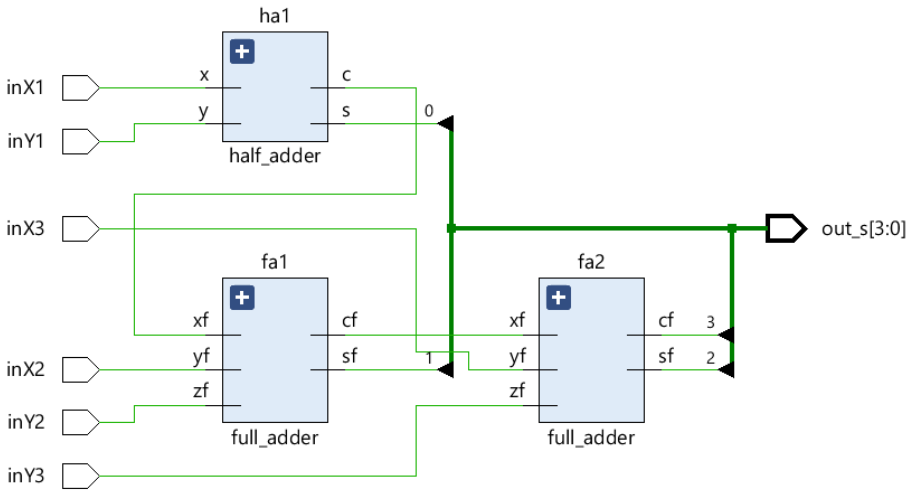

Figure 3: RTL schematic of Addition Operator

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

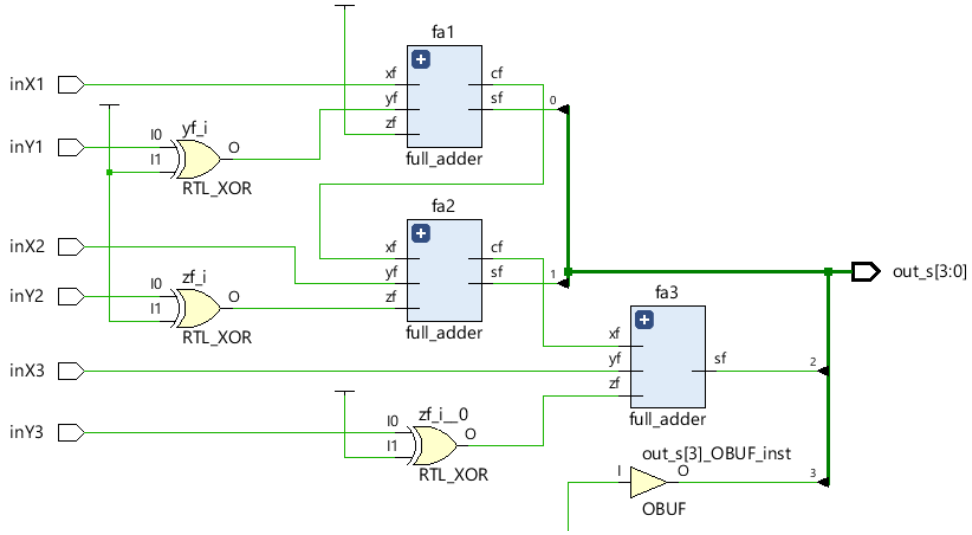

Figure 4: RTL schematic of Subtraction Operator

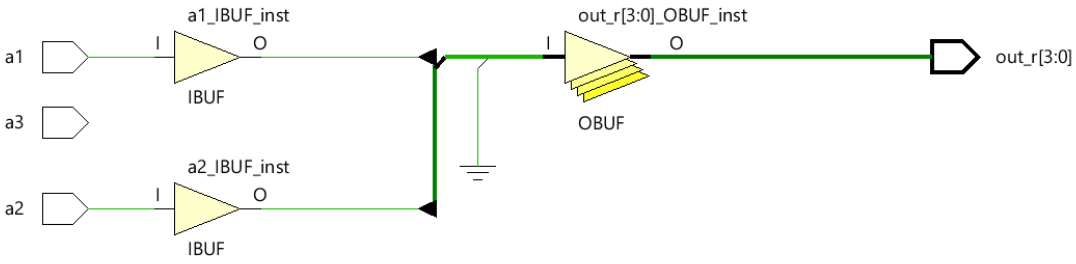

Figure 5: RTL schematic of Logical Shift

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

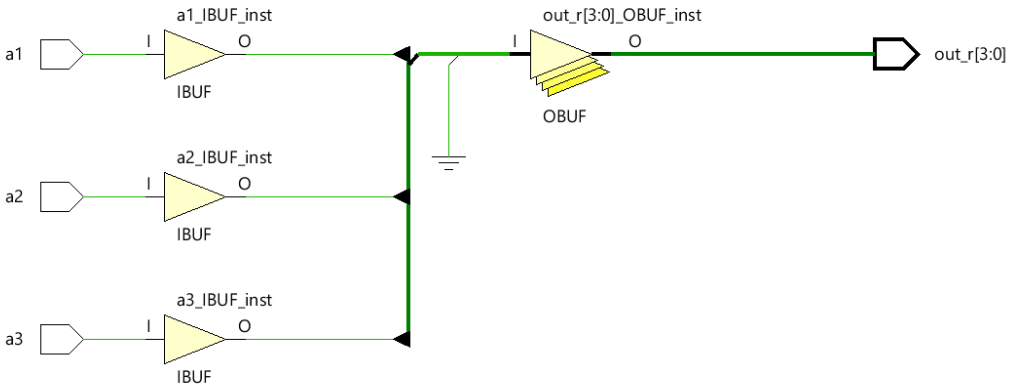

Figure 6: RTL schematic of Rotate

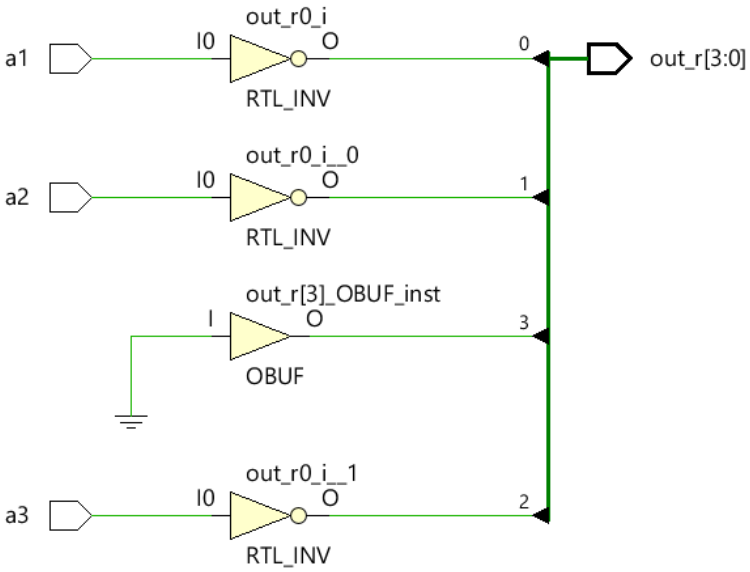

Figure 7: RTL schematic of Ones' Complement

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

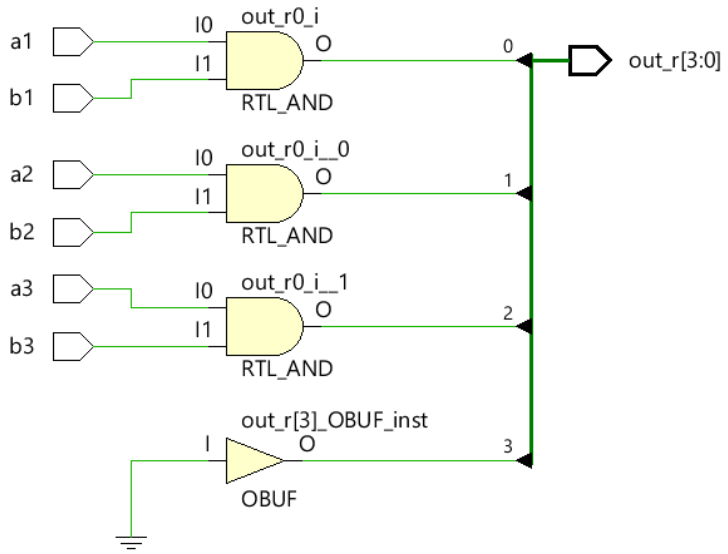

Figure 8: RTL schematic of Bitwise AND

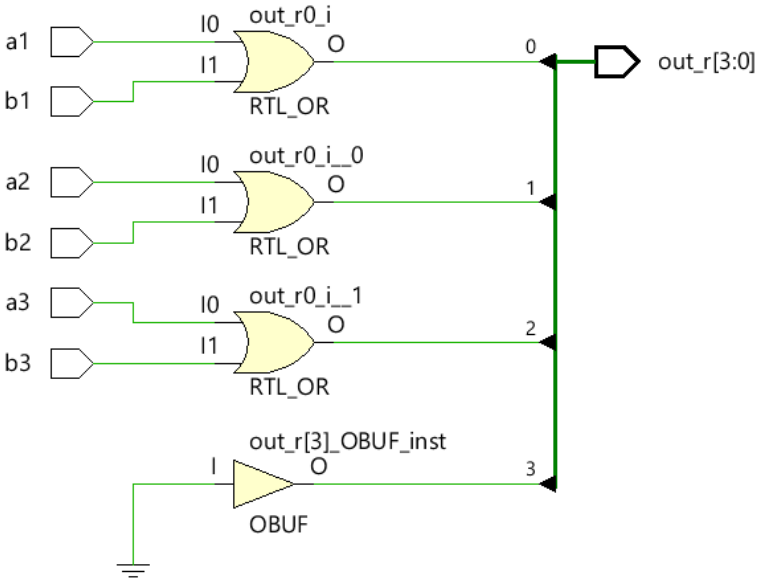

Figure 9: RTL schematic of Bitwise OR

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

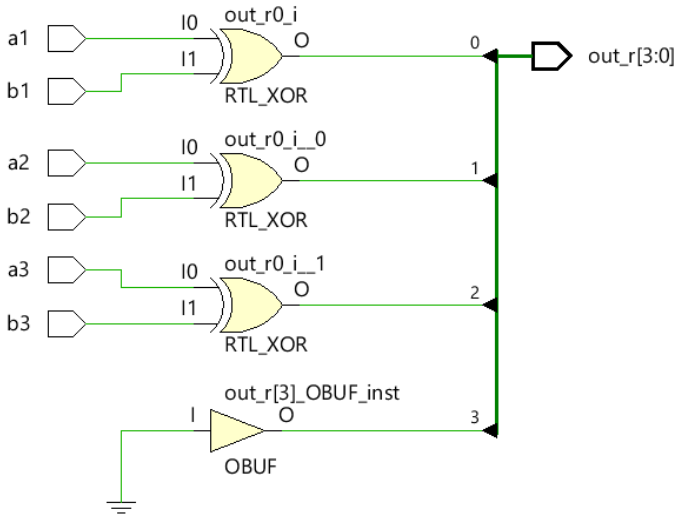

Figure 10: RTL schematic of Bitwise XOR

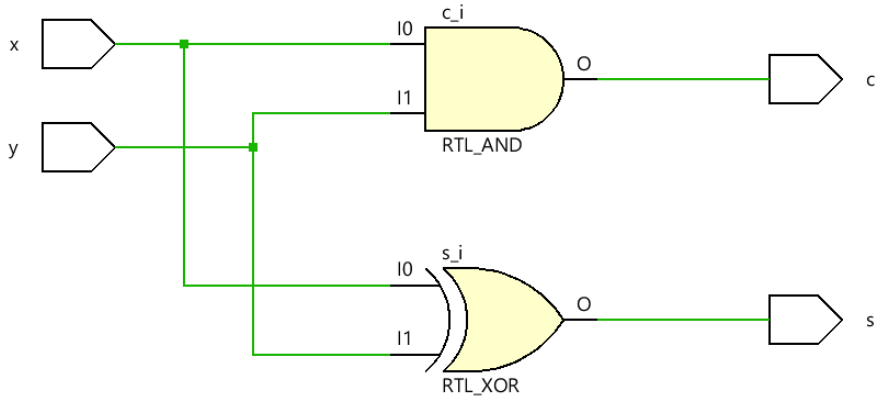

Figure 11: RTL schematic of Half Adder

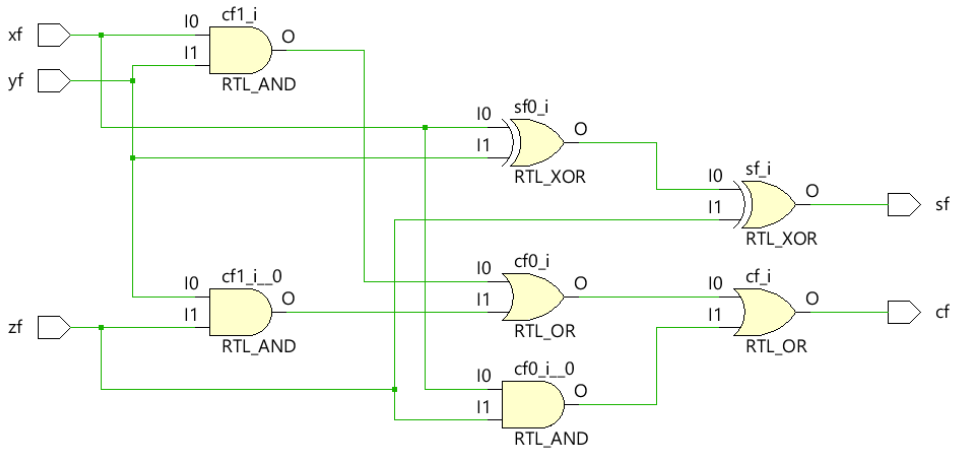

Figure 12: RTL schematic of Full Adder

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

To simulate the modules, I used four different combinations of numbers X and Y. Figures 13-16 show the simulation results for different values of X and Y. In the test bench codes, all the possible values of sel is tested and the outputs for all the operators are displayed.

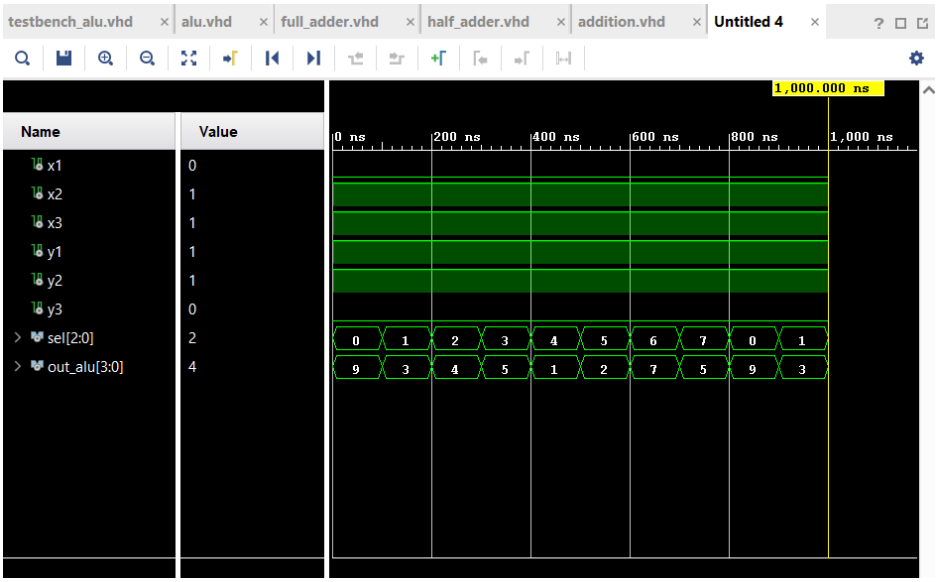

Figure 13: Simulation results with X= "110" and Y="011"

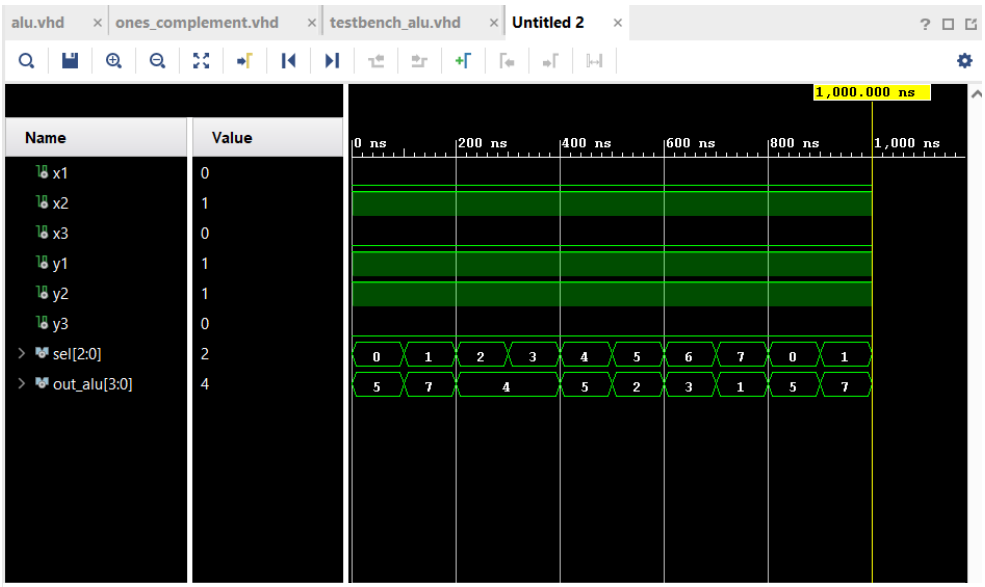

Figure 14: Simulation results with X= "010" and Y="011"

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

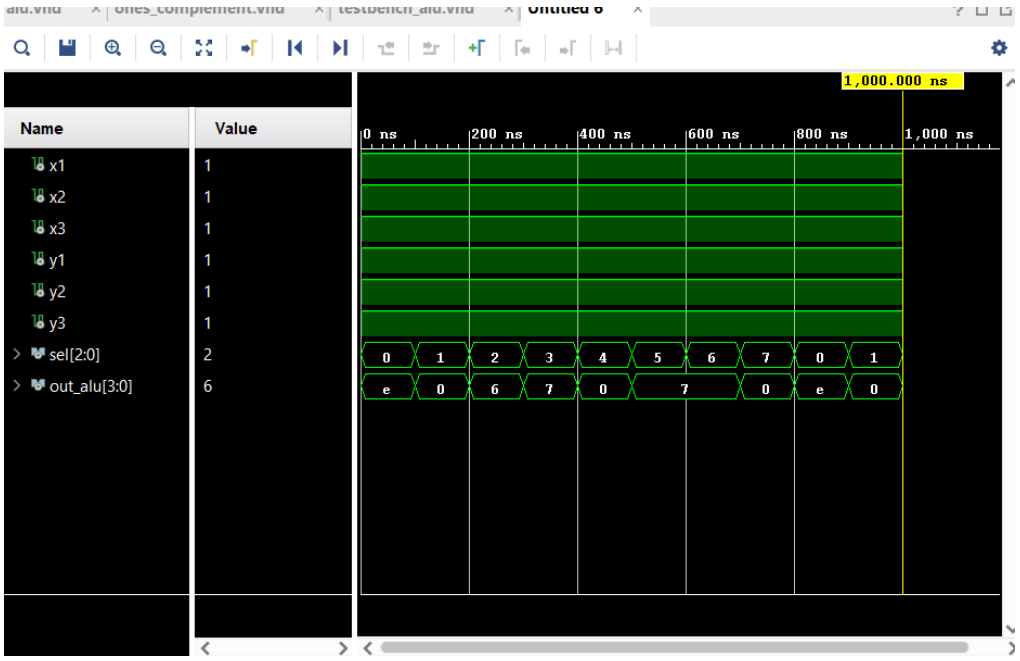

Figure 15: Simulation results with X= "111" and Y="111"

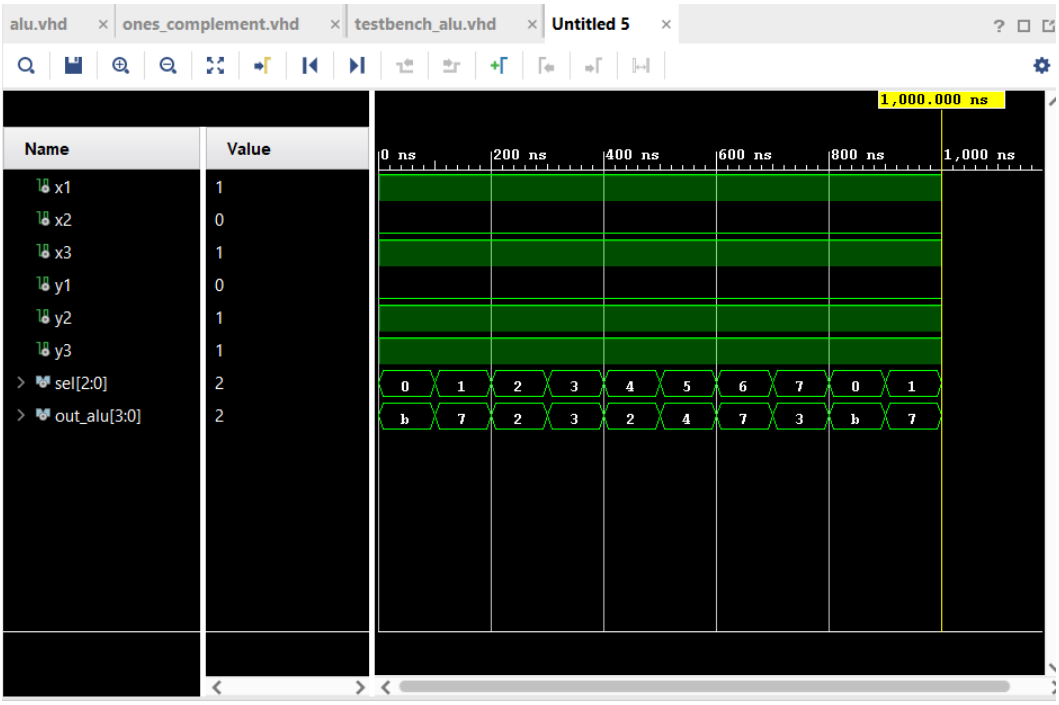

Figure 16: Simulation results with X= "101" and Y="110"

The code is then implemented on the BASYS3 board. Figures 17-24 show the outputs on the BASYS3 board when X=”110” and Y=”011”. The operation is selected with the three leftmost switches.

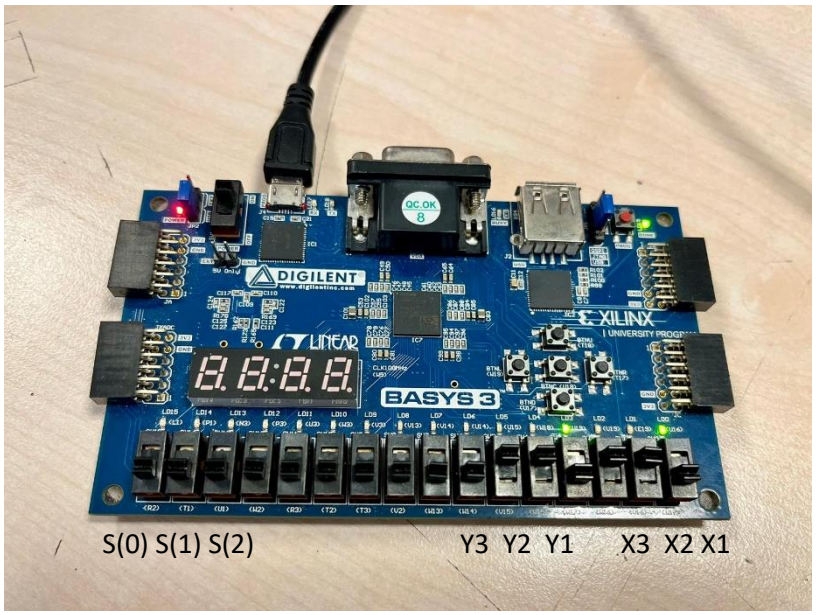

Figure 17: Operation is addition when sel=”000” and the output is “1001”

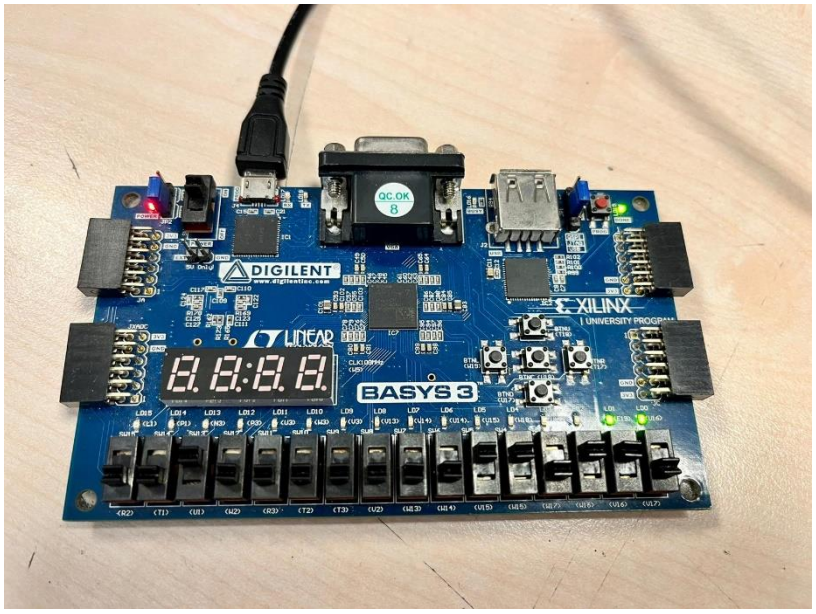

Figure 18: Operation is subtraction when sel=”001” and the output is “0011”

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

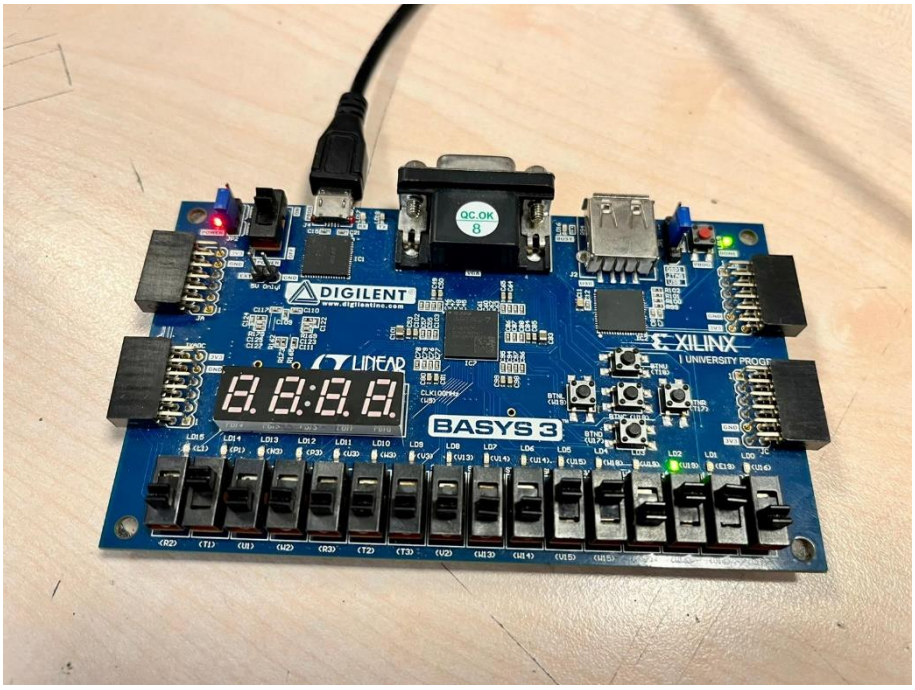

Figure 29: Operation is logical shift when sel="010" and the output is "0010"

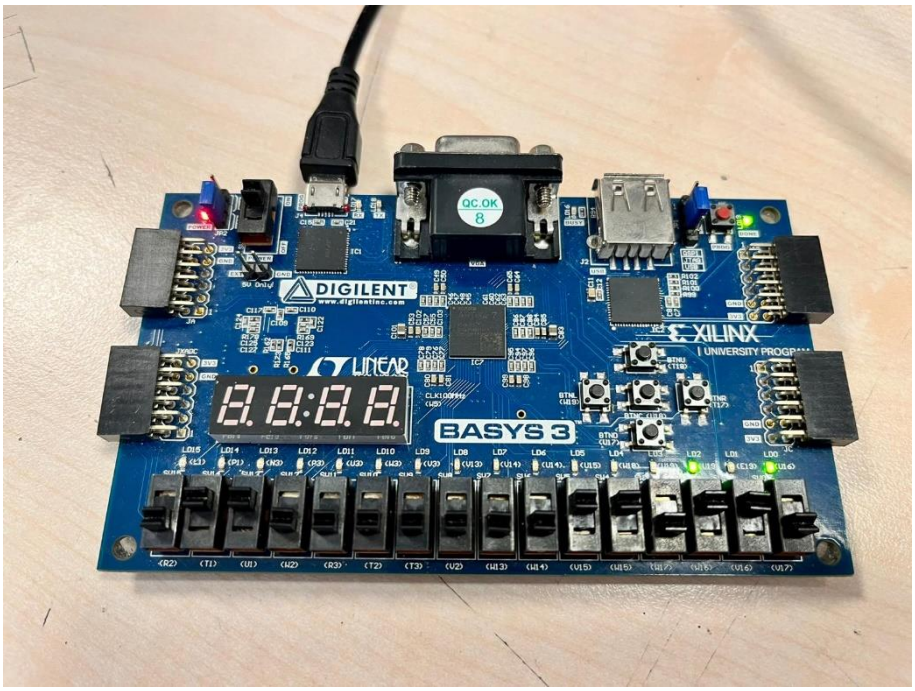

Figure 20: Operation is rotate when sel="011" and the output is "0101"

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

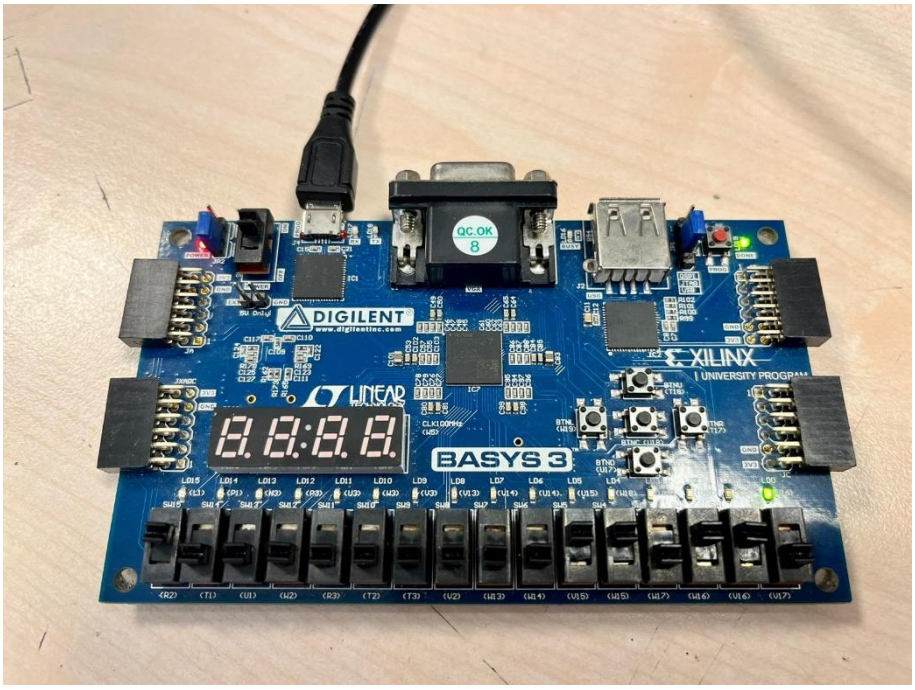

Figure 21: Operation is ones' complement when sel="100" and the output is "0001"

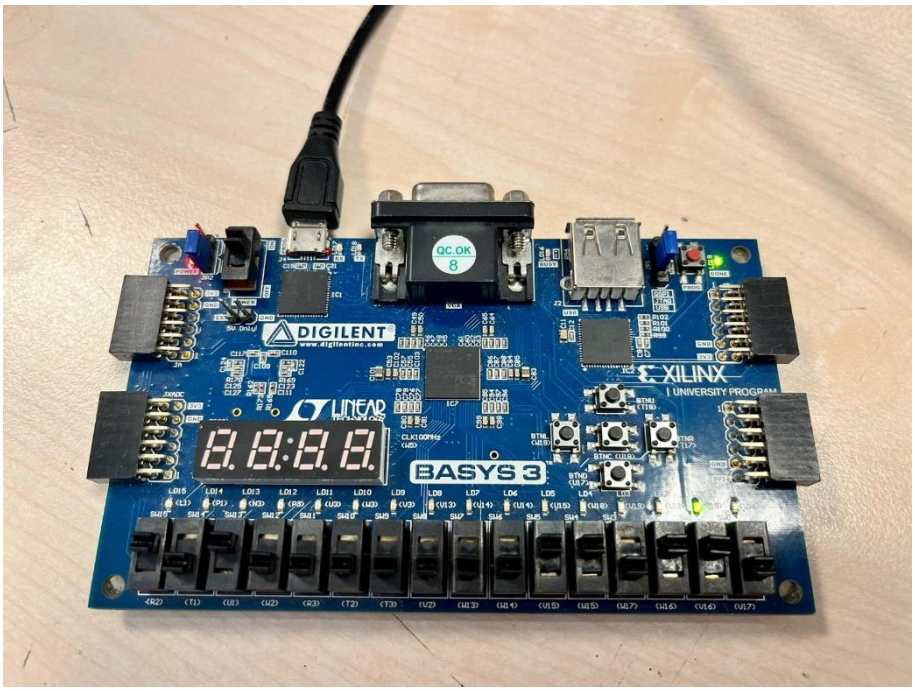

Figure 22: Operation is bitwise AND when sel="101" and the output is "0010"

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

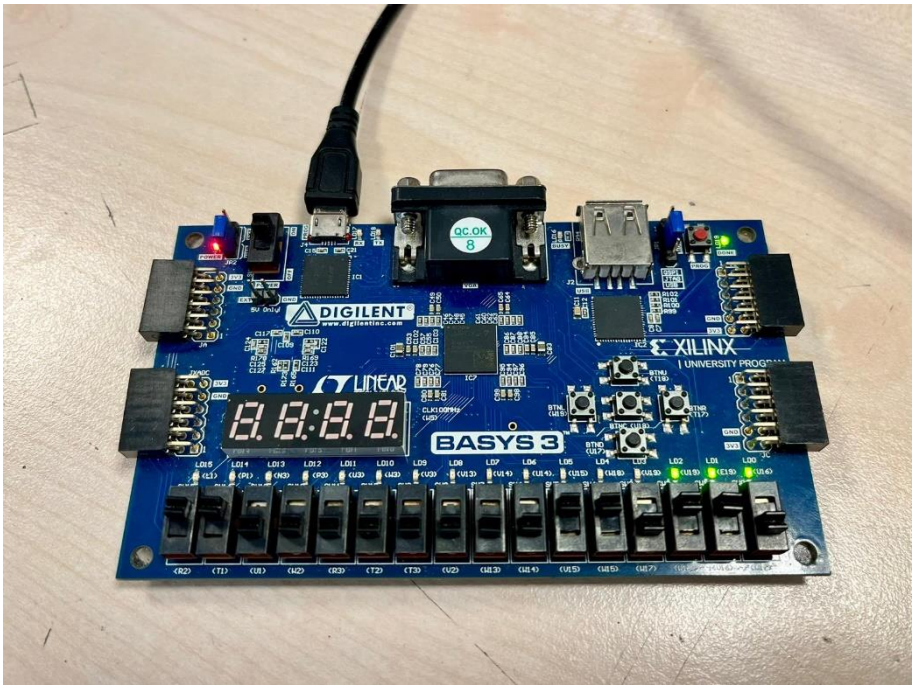

Figure 23: Operation is bitwise OR when sel=”110” and the output is “0111”

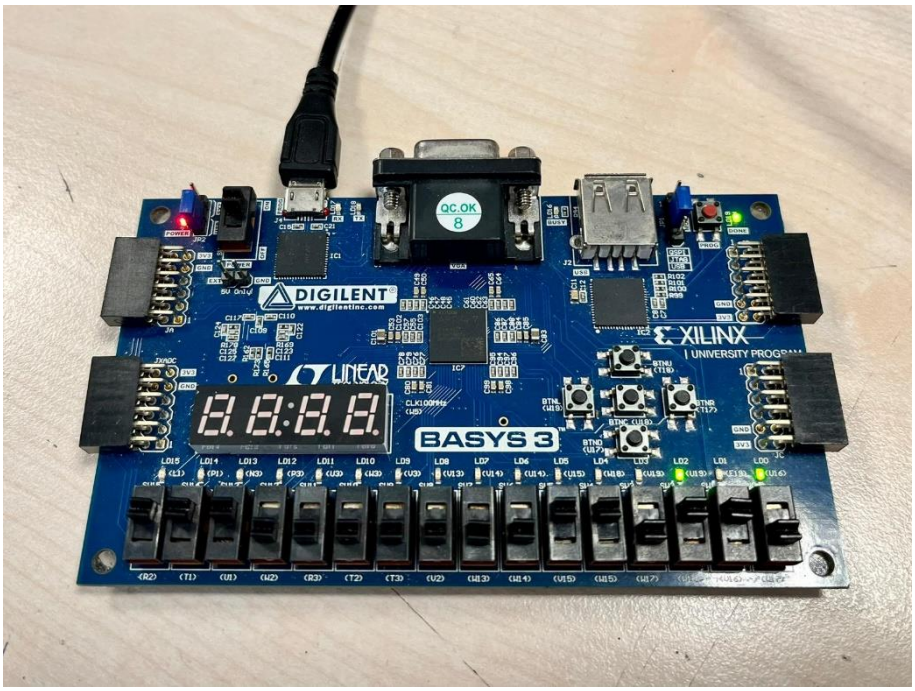

Figure 24: Operation is bitwise XOR when sel=”111” and the output is “0101”

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

Conclusion

In this lab, we were aiming to design an Arithmetic Logic Unit that can perform eight operations. The operators take two 3-bit numbers as inputs and the result is displayed as a 4-bit output. The operation is selected through the “sel” input by the user. The simulation results and the outputs on the BASYS3 board were all as expected and the operations gave the correct results. The design was made in a modular fashion, and I found it time-consuming to implement all the operations as separate modules. I also had difficulties in declaring the ports of the components while writing the code. However, it was very beneficial for me because I learned how to use modules and implement circuits with them in VHDL. I also gained a lot more experience in VHDL because this is the first time that I’ve written very long VHDL codes.

Appendices:

alu.vhd

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


entity alu is
    Port ( x1 : in STD_LOGIC;
           x2 : in STD_LOGIC;
           x3 : in STD_LOGIC;
           y1 : in STD_LOGIC;
           y2 : in STD_LOGIC;
           y3 : in STD_LOGIC;
           sel : in STD_LOGIC_VECTOR (2 downto 0);
           out_alu : out STD_LOGIC_VECTOR (3 downto 0));
end alu;


architecture Behavioral of alu is


component addition
 Port ( inX1,inX2, inX3, inY1, inY2, inY3 : in std_logic;
 out_s: out std_logic_vector(3 downto 0));
```

```
end component;

component subtraction
 Port (  inX1,inX2, inX3, inY1, inY2, inY3 : in std_logic;
 out_s: out std_logic_vector(3 downto 0));
end component;

component logical_shift
 Port (a1,a2, a3: in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;

component rotate
 Port (a1,a2, a3: in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;

component ones_complement
 Port (a1,a2, a3: in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;

component and_gate
 Port ( a1,a2, a3, b1, b2, b3 : in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;

component or_gate
 Port ( a1,a2, a3, b1, b2, b3 : in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;
```

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

```
component xor_gate
 Port ( a1,a2, a3, b1, b2, b3 : in std_logic;
 out_r: out STD_LOGIC_vector(3 downto 0));
end component;

signal ina1,ina2,ina3,inb1,inb2,inb3: std_logic;
signal out1, out2, out3, out4, out5, out6, out7, out8: std_logic_vector(3 downto 0);
begin
ina1 <= x1;
ina2 <= x2;
ina3 <= x3;
inb1 <= y1;
inb2 <= y2;
inb3 <= y3;
u1: addition
port map(inX1 => ina1,inX2 => ina2, inX3 => ina3,
 inY1 => inb1,inY2 => inb2, inY3 => inb3,
 out_s => out1);

u2: subtraction
port map(inX1 => ina1,inX2 => ina2, inX3 => ina3,
 inY1 => inb1,inY2 => inb2, inY3 => inb3,
 out_s => out2);

u3: logical_shift
port map(a1 => ina1,a2 => ina2, a3 => ina3,
 out_r => out3);

u4: rotate
port map(a1 => ina1,a2 => ina2, a3 => ina3,
```

```
 out_r => out4);

u5: ones_complement
port map(a1 => ina1,a2 => ina2, a3 => ina3,
 out_r => out5);

u6: and_gate
port map(a1 => ina1,a2 => ina2, a3 => ina3,
 b1 => inb1,b2 => inb2, b3 => inb3,
 out_r => out6);

u7: or_gate
port map(a1 => ina1,a2 => ina2, a3 => ina3,
 b1 => inb1,b2 => inb2, b3 => inb3,
 out_r => out7);

u8: xor_gate
port map(a1 => ina1,a2 => ina2, a3 => ina3,
 b1 => inb1,b2 => inb2, b3 => inb3,
 out_r => out8);

process(sel, out1, out2, out3, out4, out5, out6, out7, out8)
begin
 if sel = "000" then
    out_alu <= out1;
 elsif sel = "001" then
    out_alu <= out2;
 elsif sel = "010" then
    out_alu <= out3;
 elsif sel = "011" then
    out_alu <= out4;
```

```
 elsif sel = "100" then
    out_alu <= out5;
 elsif sel = "101" then
    out_alu <= out6;
 elsif sel = "110" then
    out_alu <= out7;
 elsif sel = "111" then
    out_alu <= out8;
 else
    out_alu <= "0000";
 end if;
end process;

end Behavioral;
```

addition.vhd

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

entity addition is
    Port ( inX1 : in STD_LOGIC;
           inX2 : in STD_LOGIC;
           inX3 : in STD_LOGIC;
           inY1 : in STD_LOGIC;
           inY2 : in STD_LOGIC;
           inY3 : in STD_LOGIC;
           out_s : out STD_LOGIC_VECTOR (3 downto 0));
end addition;

architecture Behavioral of addition is
component half_adder
```

```
 Port ( x : in STD_LOGIC;
 y : in STD_LOGIC;
 s : out STD_LOGIC;
 c : out STD_LOGIC);
end component;

component full_adder
 Port ( xf : in STD_LOGIC;
 yf : in STD_LOGIC;
 zf : in STD_LOGIC;
 sf : out STD_LOGIC;
 cf : out STD_LOGIC);
end component;

signal hac,fac1: std_logic;
begin
 ha1:half_adder
 port map(x => inX1, y => inY1,
 s => out_s(0), c => hac);
 fa1:full_adder
 port map(xf => hac, yf => inX2, zf => inY2,
 sf => out_s(1), cf => fac1);
 fa2:full_adder
 port map(xf => fac1, yf => inX3, zf => inY3,
 sf => out_s(2), cf => out_s(3));


end Behavioral;
```

subtraction.vhd

```
library IEEE;
```

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

```
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity subtraction is
    Port ( inX1 : in STD_LOGIC;
           inX2 : in STD_LOGIC;
           inX3 : in STD_LOGIC;
           inY1 : in STD_LOGIC;
           inY2 : in STD_LOGIC;
           inY3 : in STD_LOGIC;
           out_s : out STD_LOGIC_VECTOR (3 downto 0));
end subtraction;

architecture Behavioral of subtraction is
component full_adder
 Port ( xf : in STD_LOGIC;
 yf : in STD_LOGIC;
 zf : in STD_LOGIC;
 sf : out STD_LOGIC;
 cf : out STD_LOGIC);
end component;

signal d,fc1,fc2: std_logic;
```

```
begin

 fa1:full_adder
 port map(xf => inX1, yf => (inY1 xor '1'), zf => '1',
 sf => out_s(0), cf => fc1);
 fa2:full_adder
 port map(xf => fc1, yf => inX2, zf => (inY2 xor '1'),
 sf => out_s(1), cf => fc2);
 fa3:full_adder
 port map(xf => fc2, yf => inX3, zf => (inY3 xor '1'),
 sf => out_s(2), cf => d);

 out_s(3) <= '0';

end Behavioral;
```

logical_shift.vhd

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

entity logical_shift is
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR (3 downto 0));
end logical_shift;

architecture Behavioral of logical_shift is
signal x,y,z: std_logic;
begin
 x <= a2;
```

```
 y <= a1;
 z <= '0';
 out_r <= '0' & x & y & z;

end Behavioral;
```

rotate.vhd

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

entity rotate is
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR (3 downto 0));
end rotate;


architecture Behavioral of rotate is
signal x,y,z: std_logic;
begin
 x <= a2;
 y <= a1;
 z <= a3;
 out_r <= '0' & x & y & z;

end Behavioral;
```

ones_complement.vhd

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

entity ones_complement is
```

```
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR (3 downto 0));
end ones_complement;

architecture Behavioral of ones_complement is

begin

 out_r(0) <= not a1;
 out_r(1) <= not a2;
 out_r(2) <= not a3;
 out_r(3) <= '0';

end Behavioral;
```

and_gate.vhd

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

entity and_gate is
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           b3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR(3 downto 0));
end and_gate;

architecture Behavioral of and_gate is
```

```
begin
out_r(0) <= a1 and b1;
out_r(1) <= a2 and b2;
out_r(2) <= a3 and b3;
out_r(3) <= '0';

end Behavioral;
```

or_gate.vhd

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

entity or_gate is
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           b3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR (3 downto 0));
end or_gate;

architecture Behavioral of or_gate is

begin
out_r(0) <= a1 or b1;
out_r(1) <= a2 or b2;
out_r(2) <= a3 or b3;
out_r(3) <= '0';

end Behavioral;
```

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

xor_gate.vhd

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

entity xor_gate is
    Port ( a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           b3 : in STD_LOGIC;
           out_r : out STD_LOGIC_VECTOR (3 downto 0));
end xor_gate;

architecture Behavioral of xor_gate is
begin
out_r(0) <= a1 xor b1;
out_r(1) <= a2 xor b2;
out_r(2) <= a3 xor b3;
out_r(3) <= '0';



end Behavioral;
```

half_adder.vhd

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

entity half_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : out STD_LOGIC;
```

```
        c : out STD_LOGIC);
end half_adder;

architecture Behavioral of half_adder is

begin

s <= x xor y;
c <= x and y;

end Behavioral;
```

full_adder.vhd

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

entity full_adder is
    Port ( xf : in STD_LOGIC;
           yf : in STD_LOGIC;
           zf : in STD_LOGIC;
           sf : out STD_LOGIC;
           cf : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is

begin
sf <= xf xor yf xor zf;
cf <= (xf and yf) or (yf and zf) or (xf and zf);

end Behavioral;
```

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

testbench_alu.vhd

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

entity testbench_alu is
end testbench_alu;

architecture Behavioral of testbench_alu is

component alu
 Port (x1,x2, x3, y1, y2, y3: in STD_LOGIC;
 sel: in std_logic_vector(2 downto 0);
 out_alu: out STD_LOGIC_vector(3 downto 0));
end component;
 signal x1, x2,x3,y1,y2,y3: std_logic;
 signal sel: std_logic_vector(2 downto 0);
 signal out_alu: std_logic_vector(3 downto 0);
begin
 u1: alu
 Port map(x1 => x1, x2 => x2, x3 => x3, y1 => y1, y2 =>
y2, y3 => y3,
 sel => sel, out_alu => out_alu);
testbench_alu: process
begin
sel <= "000";
 x1 <= '0';
 x2 <= '1';
 x3 <= '1';
 y1 <= '1';
 y2 <= '1';
 y3 <= '0';
```

```
wait for 100ns;
sel <= "001";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "010";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "011";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "100";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
```

```
y3 <= '0';
wait for 100ns;
sel <= "101";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "110";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "111";
x1 <= '0';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
-----------------------
sel <= "000";
x1 <= '0';
x2 <= '1';
x3 <= '0';
```

```
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "001";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "010";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "011";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "100";
x1 <= '0';
x2 <= '1';
```

```
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';

wait for 100ns;
sel <= "101";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "110";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
sel <= "111";
x1 <= '0';
x2 <= '1';
x3 <= '0';
y1 <= '1';
y2 <= '1';
y3 <= '0';
wait for 100ns;
----------------------
```

Yiğit Narter
22102718
Section 102-01
Lab Report 4
7.11.2022

```
sel <= "000";
 x1 <= '1';
 x2 <= '1';
 x3 <= '1';
 y1 <= '1';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "001";
 x1 <= '1';
 x2 <= '1';
 x3 <= '1';
 y1 <= '1';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "010";
 x1 <= '1';
 x2 <= '1';
 x3 <= '1';
 y1 <= '1';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "011";
 x1 <= '1';
 x2 <= '1';
 x3 <= '1';
 y1 <= '1';
 y2 <= '1';
 y3 <= '1';
```

```
wait for 100ns;
sel <= "100";
x1 <= '1';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "101";
x1 <= '1';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "110";
x1 <= '1';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "111";
x1 <= '1';
x2 <= '1';
x3 <= '1';
y1 <= '1';
y2 <= '1';
```

```
 y3 <= '1';
 wait for 100ns;
----------------------------
sel <= "000";
 x1 <= '1';
 x2 <= '0';
 x3 <= '1';
 y1 <= '0';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "001";
 x1 <= '1';
 x2 <= '0';
 x3 <= '1';
 y1 <= '0';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "010";
 x1 <= '1';
 x2 <= '0';
 x3 <= '1';
 y1 <= '0';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;
 sel <= "011";
 x1 <= '1';
 x2 <= '0';
 x3 <= '1';
```

```
y1 <= '0';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "100";
x1 <= '1';
x2 <= '0';
x3 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "101";
x1 <= '1';
x2 <= '0';
x3 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "110";
x1 <= '1';
x2 <= '0';
x3 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '1';
wait for 100ns;
sel <= "111";
x1 <= '1';
x2 <= '0';
```

```
 x3 <= '1';
 y1 <= '0';
 y2 <= '1';
 y3 <= '1';
 wait for 100ns;

end process;
end Behavioral;
```

pins1.xdc

```
set_property PACKAGE_PIN V17 [get_ports {x1}]
 set_property IOSTANDARD LVCMOS33 [get_ports {x1}]
set_property PACKAGE_PIN V16 [get_ports {x2}]
 set_property IOSTANDARD LVCMOS33 [get_ports {x2}]
set_property PACKAGE_PIN W16 [get_ports {x3}]
 set_property IOSTANDARD LVCMOS33 [get_ports {x3}]
set_property PACKAGE_PIN W15 [get_ports {y1}]
 set_property IOSTANDARD LVCMOS33 [get_ports {y1}]
set_property PACKAGE_PIN V15 [get_ports {y2}]
 set_property IOSTANDARD LVCMOS33 [get_ports {y2}]
set_property PACKAGE_PIN W14 [get_ports {y3}]
 set_property IOSTANDARD LVCMOS33 [get_ports {y3}]
set_property PACKAGE_PIN U1 [get_ports {sel[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sel[0]}]
set_property PACKAGE_PIN T1 [get_ports {sel[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property PACKAGE_PIN R2 [get_ports {sel[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sel[2]}]
set_property PACKAGE_PIN U16 [get_ports {out_alu[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {out_alu[0]}]
set_property PACKAGE_PIN E19 [get_ports {out_alu[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {out_alu[1]}]
```

Yiğit Narter
22102718
Section 102-01


```
set_property PACKAGE_PIN U19 [get_ports {out_alu[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {out_alu[2]}]
set_property PACKAGE_PIN V19 [get_ports {out_alu[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {out_alu[3]}]
```