Lab 4: Arithmetic Logic Unit

Date: 23.10.2023

Name: Şevval ERBAY

Section: 2

Purpose of the experiment:

The purpose of this lab is to design an arithmetic logic unit (ALU) capable of performing at least eight different operations including addition, subtraction, one bitwise and one shift operation by using VHDL. The VHDL code is implemented in modular fashion.

Methodology:

The selected operations are: addition, subtraction, bitwise OR-gate, bitwise AND-gate, pass through, ones' complement, left logical shift and right logical shift. A half-adder and a full-adder is created to use in the addition and subtraction operations. Main module alu.vhd has the other operations in a modular fashion. 2 different 4-bit signed numbers, **a** and **b**, 3-bit select option **sel**, 4-bit output **out0** and a carry **car** are used in the operations. RTL schematic is created, and a test bench is written to check whether the operations are working properly or not. In order to be sure, another test bench is utilized. Lastly, the constraint code is written and bit stream is generated. The process is simulated by using BASYS3's switches and LEDs.

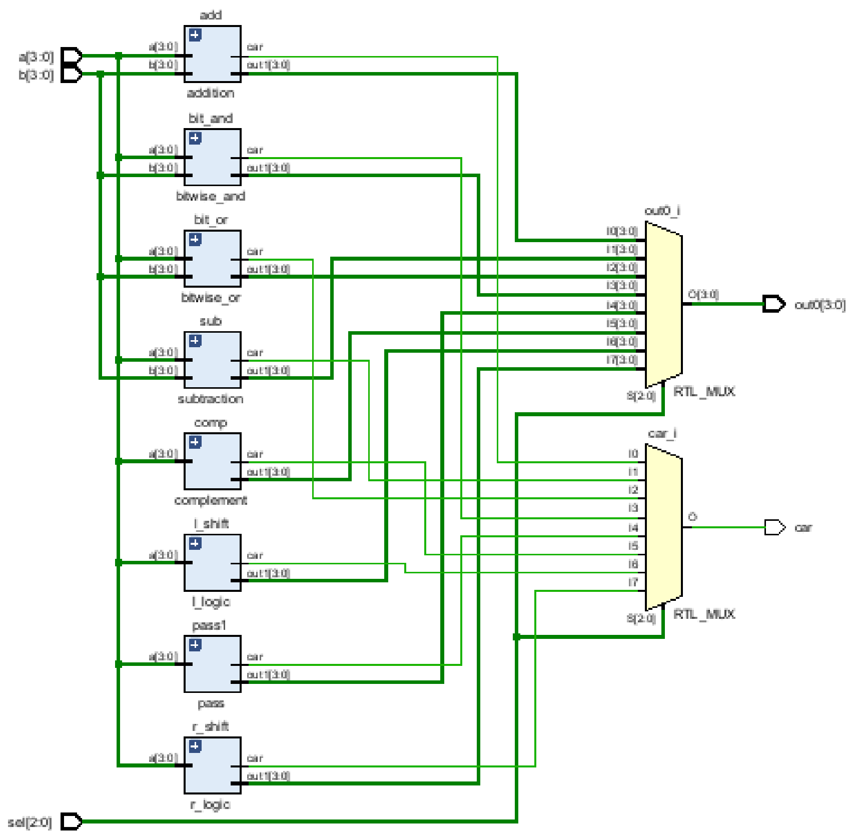


Figure.1 RTL Schematic of the Arithmetic Logic Unit (ALU)

ALU Operations:

Addition: This function adds two signed 4-bit inputs together. When **sel** is “000”, this operation becomes active.

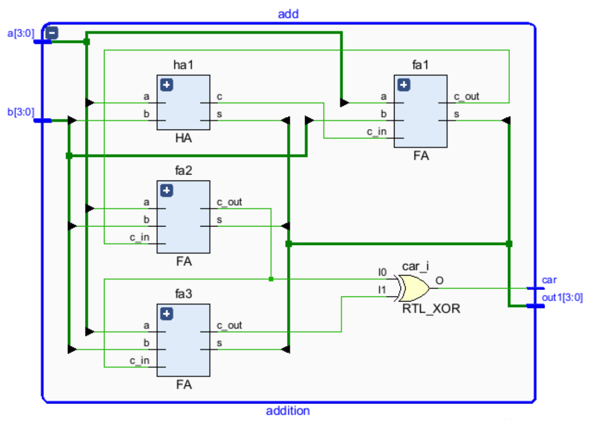


Figure.2 Addition schematic

Subtraction: This function subtracts **b** from **a**. When **sel** is “001”, this operation becomes active.

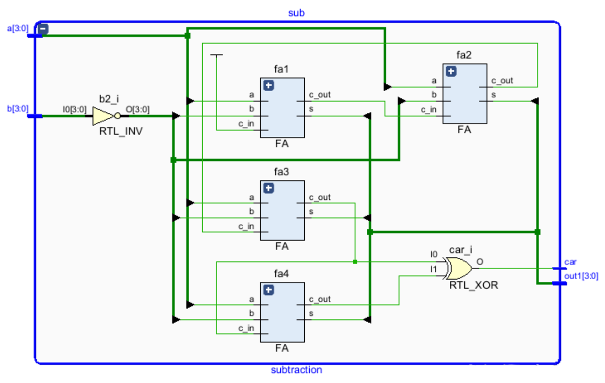


Figure.3 Subtraction schematic

Bitwise OR-gate: When **sel** is “010”, this operation becomes active.

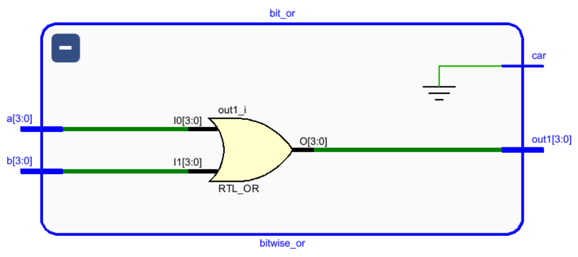


Figure.4 Bitwise OR-gate

Bitwise AND-gate: When **sel** is “011”, this operation becomes active.

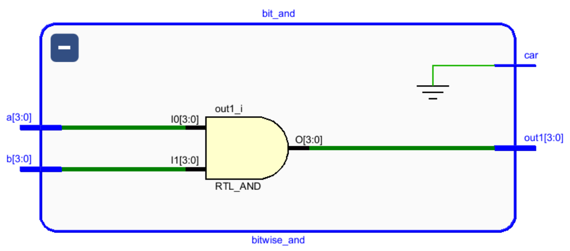


Figure.5 Bitwise AND-Gate

Pass Through: **a** appears unmodified at the output. When **sel** is “100”, this operation becomes active.

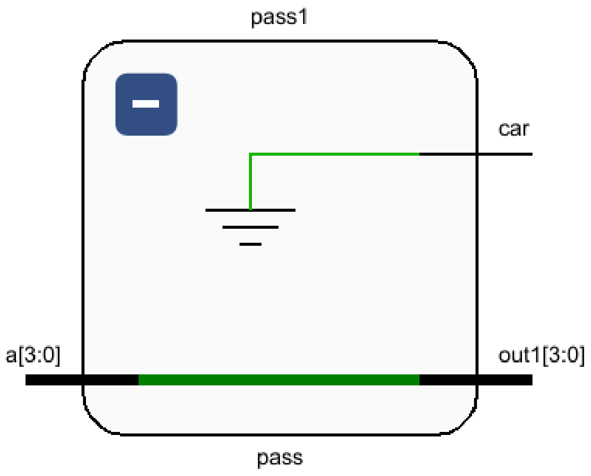


Figure.6 Pass Through

Ones’ Complement: All bits of **a** becomes inverted at the output. When **sel** is “101”, this operation becomes active.

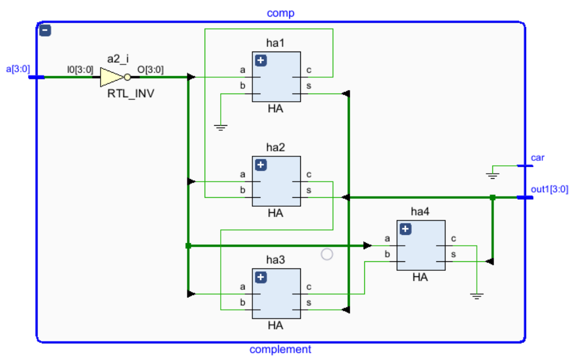


Figure.7 Ones’ Complement

Left Logic Shift: Moves each bit to one left bit and fills the space with “0”. When **sel** is “110”, this operation becomes active.

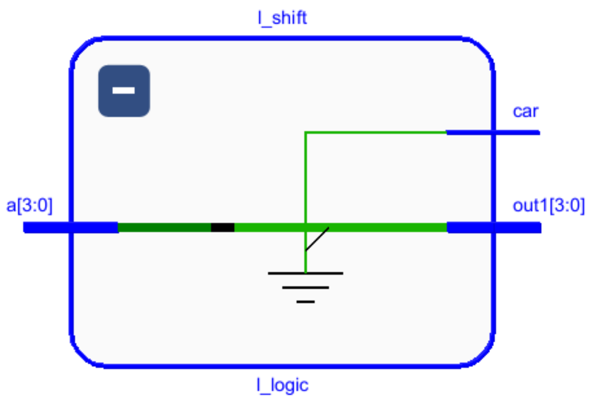


Figure.8 Left Logic Shift

Right Logic Shift: Moves each bit to one right bit and fills the space with “0”. When **sel** is “111”, this operation becomes active.

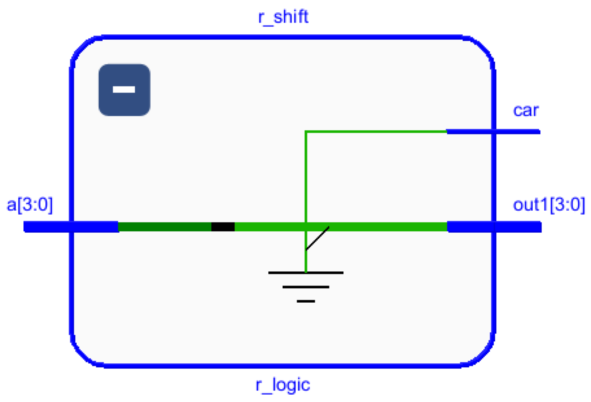


Figure.9 Right Logic Shift

Results:

Two different waveforms are observed by using two different test benches. The switches and LEDs on BASYS3 is chosen as:

a : V17, V16, W16, W17

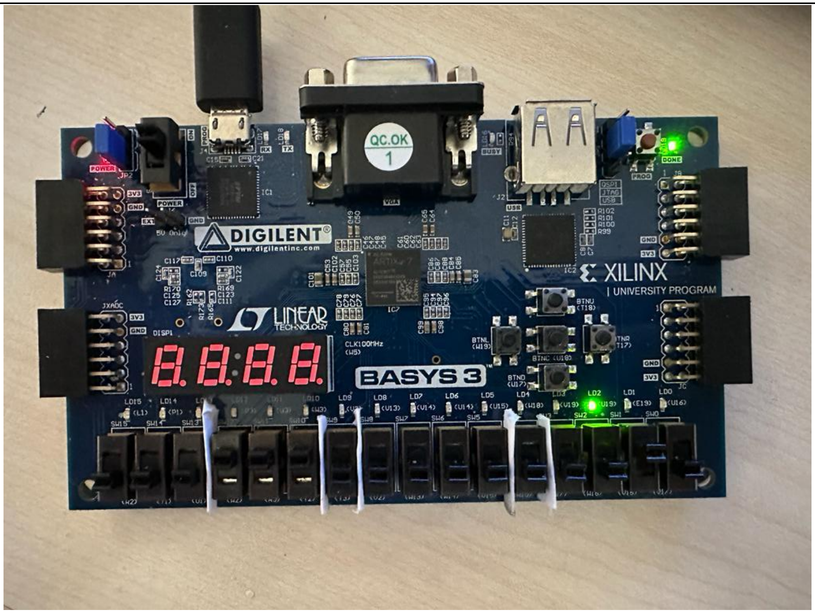
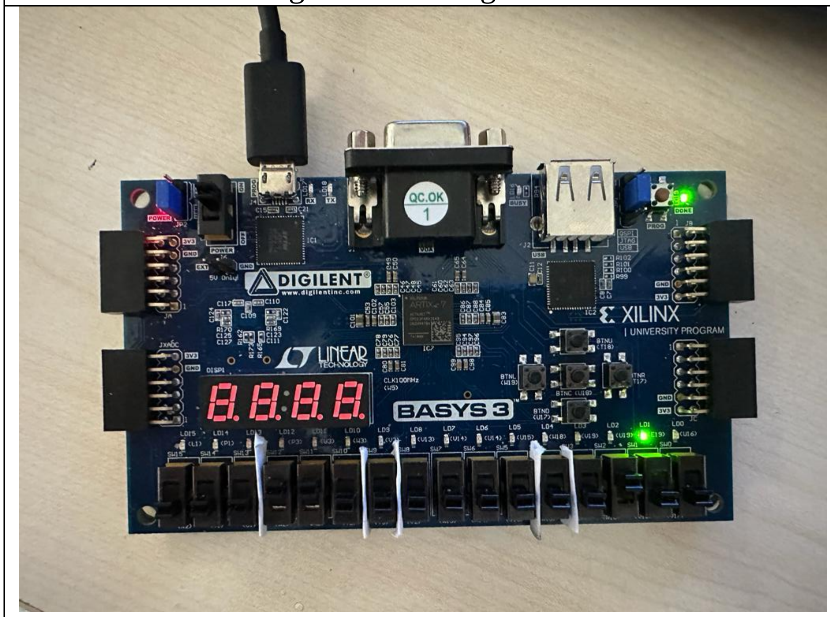
b : V15, W14, W13, V2

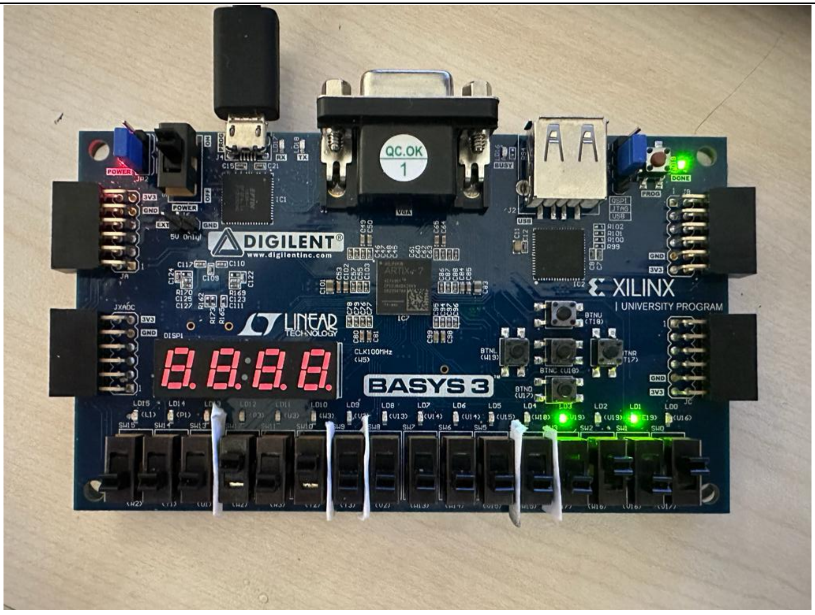
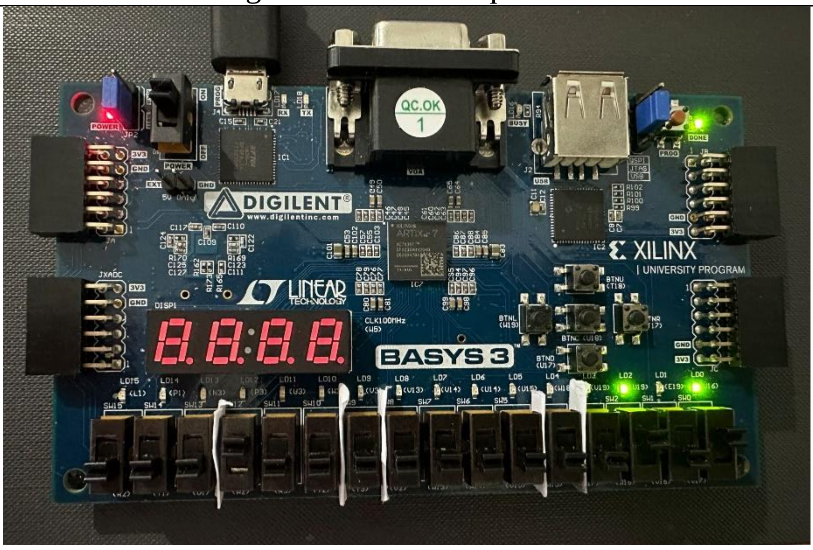
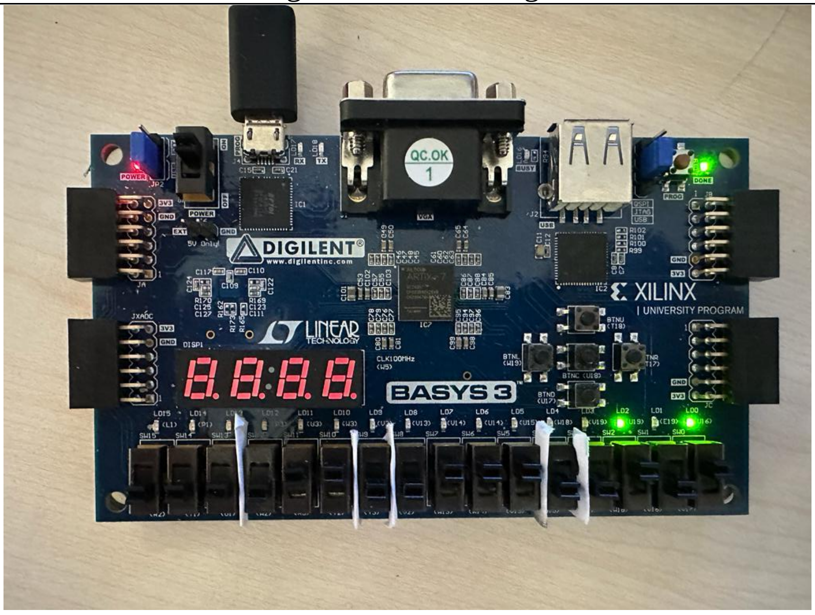
sel : T2, R3, W2

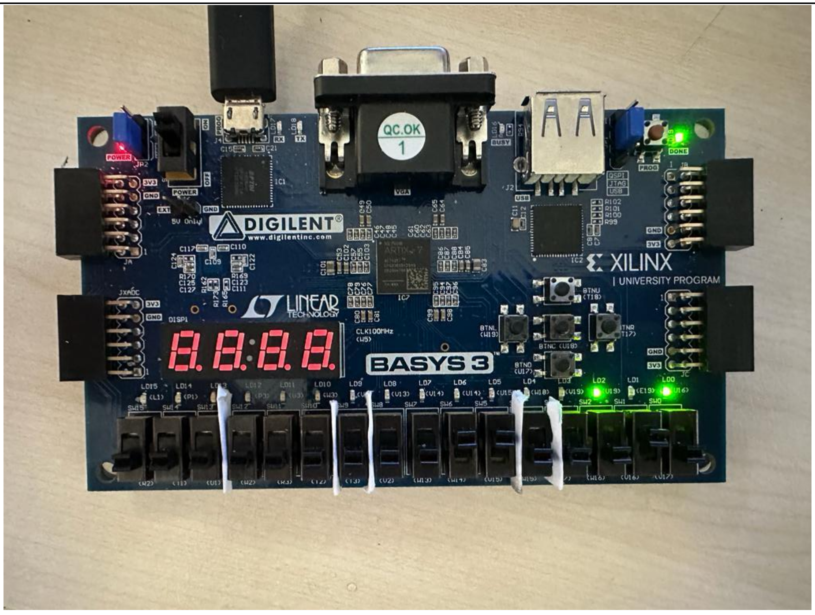
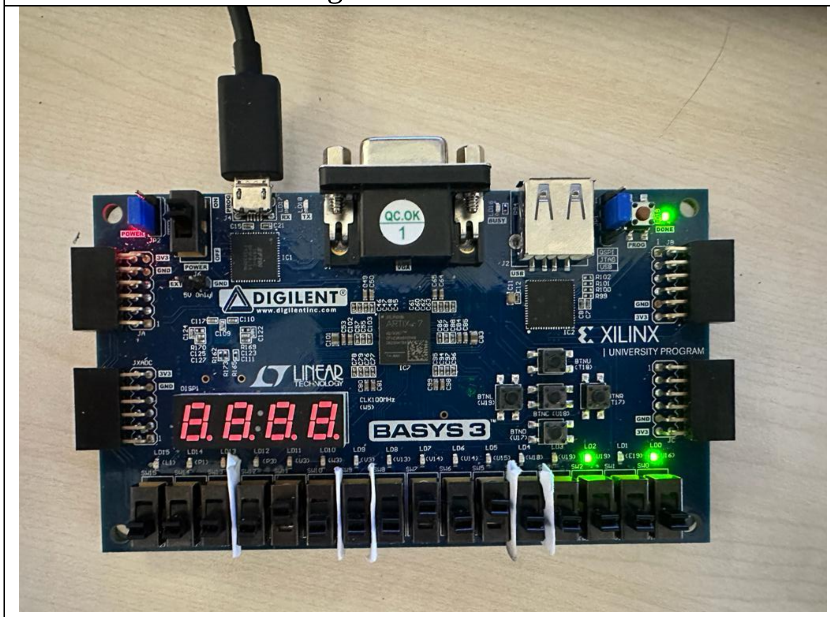
out0 : U16, E19, U19, V19

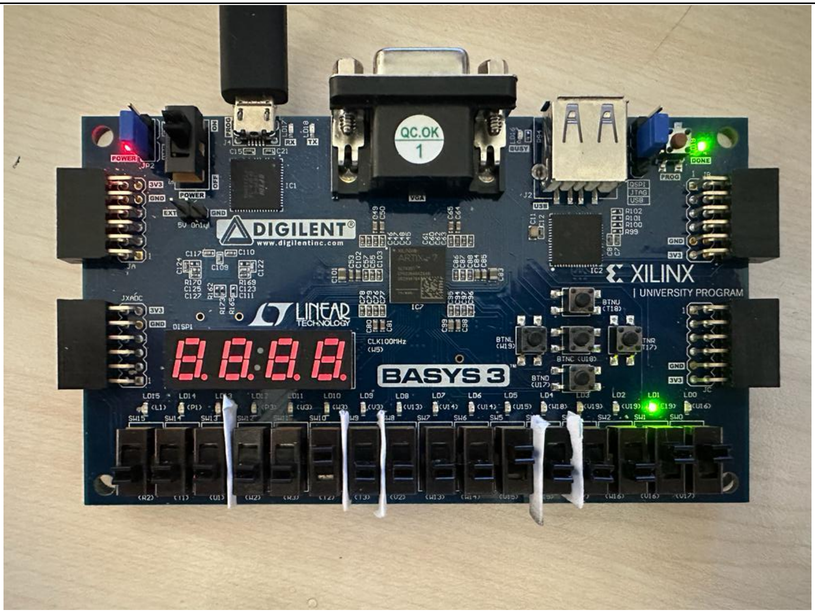
car : U15

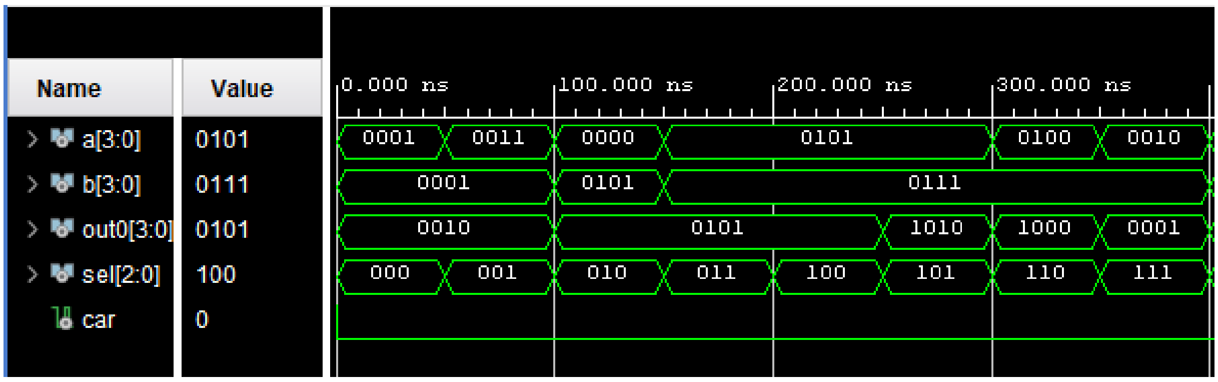
The waveforms and the BASYS3 representation are in parallel.

 Figure.10 Left logic shift	sel : 110 a : 0010 out0 : 0100
 Figure.11 Right logic shift	sel : 111 a : 0100 out0 : 0010

 Figure.12 Ones' complement	sel : 101 a : 0101 out0 : 1010
 Figure.13 Pass through	sel : 100 a : 0101 out0 : 0101
 Figure.14 Bitwise AND-gate	sel : 011 a : 0101 b : 0111 out0 : 0101

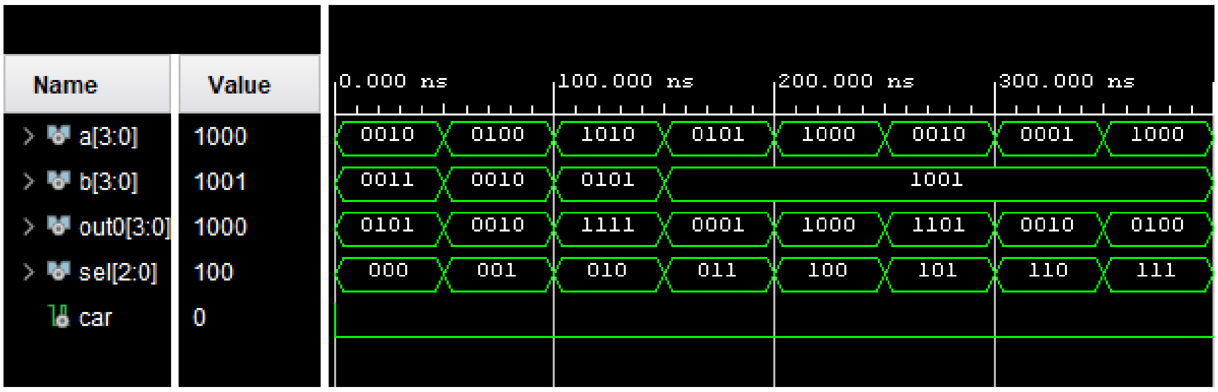
 Figure.15 Addition	sel : 000 a : 0010 b : 0011 out0 : 0101
 Figure.16 Bitwise OR-gate	sel : 010 a : 0000 b : 0101 out0 : 0101

 Figure.17 Subtraction	sel : 001 a : 0011 b : 0001 out0 : 0010



Name	Value
a[3:0]	0101
b[3:0]	0111
out0[3:0]	0101
sel[2:0]	100
car	0

Figure.18 Waveform 1, generated by the first test bench



Name	Value
a[3:0]	1000
b[3:0]	1001
out0[3:0]	1000
sel[2:0]	100
car	0

Figure.19 Waveform 2, generated by the second test bench

Conclusion:

In this lab, we learned how to design an arithmetic logic unit (ALU) capable of performing at least eight different operations including addition, subtraction, one bitwise and one shift operation by using VHDL. The selected operations are: addition, subtraction, bitwise OR-gate, bitwise AND-gate, pass through, ones' complement, left logical shift and right logical shift. A half-adder and a full-adder is created to use in the addition and subtraction operations. Main module alu.vhd has the other operations in a modular fashion. 2 different 4-bit signed numbers, **a** and **b**, 3-bit select option **sel**, 4-bit output **out0** and a carry **car** are used in the operations. I learned how to properly implement an ALU in real life. All in all, the experiment was successful at the end.

APPENDIX

addition.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity addition is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          out1 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
end addition;
architecture add_arch of addition is
   component fa is
      port(a, b, c_in : in std_logic;
           s, c_out : out std_logic);
   end component;
   component ha is
      port(a, b : in std_logic;
           s, c : out std_logic);
   end component;
   signal c : std_logic_vector(3 downto 0);
begin
   ha1 : ha port map (a => a(0),b => b(0),c => c(0), s => out1(0));
   fa1 : fa port map (a => a(1),b => b(1),c_in => c(0), c_out => c(1), s => out1(1));
   fa2 : fa port map (a => a(2),b => b(2),c_in => c(1), c_out => c(2), s => out1(2));
   fa3 : fa port map (a => a(3),b => b(3),c_in => c(2), c_out => c(3), s => out1(3));
   car <= c(2) xor c(3);
end add_arch;
```

subtraction.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity subtraction is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          out1 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
end subtraction;
architecture sub_arch of subtraction is
   signal b2  : std_logic_vector(3 downto 0);
   signal c   : std_logic_vector(3 downto 0);
   component fa is
      port(a, b, c_in : in std_logic;
           s, c_out : out std_logic);
   end component;
begin
   b2 <= not b;
   fa1: FA port map (a => a(0), b => b2(0) ,c_in => '1'  , s => out1(0), c_out => c(0) );
   fa2: FA port map (a => a(1), b => b2(1) ,c_in => c(0) , s => out1(1), c_out => c(1) );
   fa3: FA port map (a => a(2), b => b2(2) ,c_in => c(1) , s => out1(2), c_out => c(2) );
   fa4: FA port map (a => a(3), b => b2(3) ,c_in => c(2) , s => out1(3), c_out => c(3) );
   car <= c(2) xor c(3);
end sub_arch;
```

bitwise_or.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bitwise_or is
  Port (a,b  : in std_logic_vector(3 downto 0);
        out1 : out std_logic_vector(3 downto 0);
        car : out std_logic);
end bitwise_or;
architecture Behavioral of bitwise_or is
begin
    out1 <= a or b;
    car <= '0';
end Behavioral;
```

bitwise_and.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bitwise_and is
  Port (a,b  : in std_logic_vector(3 downto 0);
        out1 : out std_logic_vector(3 downto 0);
        car : out std_logic);
end bitwise_and;
architecture Behavioral of bitwise_and is
begin
    out1 <= a and b;
    car <= '0';
end Behavioral;
```

pass.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity pass is
  Port (a  : in std_logic_vector(3 downto 0);
        out1 : out std_logic_vector(3 downto 0);
        car : out std_logic);
end pass ;
architecture Behavioral of pass is
begin
    out1 <= a;
    car <= '0';
end Behavioral;
```

complement.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity complement is
    Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
           out1 : out STD_LOGIC_VECTOR(3 downto 0);
           car : out STD_LOGIC);
end complement;
architecture comp_arch of complement is
    component ha is
            port(a, b : in std_logic;
                 s, c : out std_logic);
        end component;
    signal c,a2 : std_logic_vector(3 downto 0);
begin
    a2 <= not a;
    ha1: HA port map (a => a2(0) , b => '0' , c => c(0) , s => out1(0));
    ha2: HA port map (a => a2(1) , b => c(0) , c => c(1) , s => out1(1));
    ha3: HA port map (a => a2(2) , b => c(1) , c => c(2) , s => out1(2));
    ha4: HA port map (a => a2(3) , b => c(2) , c => c(3) , s => out1(3));
    car <= c(3);
 car <= '0';
end comp_arch;
```

l_logic_shift.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity l_logic is
  Port (a : in std_logic_vector(3 downto 0);
        out1 : out std_logic_vector(3 downto 0);
        car : out std_logic);
end l_logic;
architecture l_log_arch of l_logic is
begin
    out1 <= a(2 downto 0) & '0';
    car <= '0';
end l_log_arch;
```

r_logic_shift.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity r_logic is
  Port (a : in std_logic_vector(3 downto 0);
        out1 : out std_logic_vector(3 downto 0);
        car : out std_logic);
end r_logic;
architecture r_log_arch of r_logic is
begin
    out1 <= '0' & a(3 downto 1);
    car <= '0';
end r_log_arch;
```

testbench.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity testbench is
end testbench;
architecture Behavioral of testbench is
component ALU is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          sel : in STD_LOGIC_VECTOR(2 downto 0);
          out0 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
   end component;
   signal a, b, out0 : std_logic_vector(3 downto 0);
   signal sel : std_logic_vector(2 downto 0);
   signal car : std_logic;
begin
aluu : ALU
port map(a => a , b => b , out0 => out0 , sel => sel, car => car);
testbench : process
begin
   sel <= "000"; -- add
   a <= "0101";
   b <= "0001";
   wait for 50ns; -- sub
   sel <= "001";
   a <= "0011";
   b <= "0001";
   wait for 50ns; -- bit_or
```

```
    sel <= "010";
    a <= "0000";
    b <= "0101";
    wait for 50ns; -- bit_and
    sel <= "011";
    a <= "0101";
    b <= "0111";
    wait for 50ns; -- pass1
    sel <= "100";
    a <= "0101";
    wait for 50ns; -- comp
    sel <= "101";
    a <= "0101";
    wait for 50ns; -- l-shift
    sel <= "110";
    a <= "0100";
    wait for 50ns; -- r-shift
    sel <= "111";
    a <= "0010";
    wait for 50ns;
end Behavioral;
```

alu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ALU is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          sel : in STD_LOGIC_VECTOR(2 downto 0);
          out0 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
end ALU;
architecture ALU_arch of ALU is
   component addition is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          out1 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
   end component;
   component subtraction is
   Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          out1 : out STD_LOGIC_VECTOR(3 downto 0);
          car : out STD_LOGIC);
   end component;
   component bitwise_or is
   Port (a,b  : in std_logic_vector(3 downto 0);
      out1 : out std_logic_vector(3 downto 0);
      car : out std_logic);
   end component;
   component bitwise_and is
   Port (a,b  : in std_logic_vector(3 downto 0);
      out1 : out std_logic_vector(3 downto 0);
      car : out std_logic);
   end component;
   component pass is
   Port (a  : in std_logic_vector(3 downto 0);
      out1 : out std_logic_vector(3 downto 0);
      car : out std_logic);
   end component;
   component complement is
      Port ( a : in STD_LOGIC_VECTOR(3 downto 0);
           out1 : out STD_LOGIC_VECTOR(3 downto 0);
           car : out STD_LOGIC);
   end component;
   component l_logic is
     Port (a : in std_logic_vector(3 downto 0);
         out1 : out std_logic_vector(3 downto 0);
         car : out std_logic);
   end component;
   component r_logic is
     Port (a : in std_logic_vector(3 downto 0);
         out1 : out std_logic_vector(3 downto 0);
         car : out std_logic);
   end component;
   signal out1,out2,out3,out4,out5,out6,out7,out8 : std_logic_vector(3 downto 0);
   signal car1,car2,car3,car4,car5,car6,car7,car8 : std_logic;
begin
   add : addition port map (a => a,b=> b,car=>car1,out1 => out1);
   sub : subtraction port map (a => a,b=> b,car=>car2,out1 => out2);
   bit_or : bitwise_or port map (a => a,b=> b,car=>car3,out1 => out3);
```

```
    bit_and : bitwise_and port map (a => a,b=> b,car=>car4,out1 => out4);

    pass1 : pass port map (a => a,car=>car5,out1 => out5);

    comp : complement port map (a => a,car=>car6,out1 => out6);

    l_shift : l_logic port map (a => a,car=>car7,out1 => out7);

    r_shift : r_logic port map (a => a,car=>car8,out1 => out8);

    mux_process                                                             :
process(out1,out2,out3,out4,out5,out6,out7,out8,car1,car2,car3,car4,car5,car6,car7,car8,sel)

    begin

        case sel is

            when "000" =>

                out0 <= out1;

                car <= car1;

            when "001" =>

                out0 <= out2;

                car <= car2;

            when "010" =>

                out0 <= out3;

                car <= car3;

            when "011" =>

                out0 <= out4;

                car <= car4;

            when "100" =>

                out0 <= out5;

                car <= car5;

            when "101" =>

                out0 <= out6;

                car <= car6;

            when "110" =>

                out0 <= out7;

                car <= car7;

            when "111" =>
```

```
                out0 <= out8;
                car <= car8;
            when others =>
                out0 <= out8;
                car <= car8;
        end case;
    end process;
end ALU_arch;
```

ha.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity HA is
  Port (a,b : in  std_logic;
        c , s : out std_logic);
end HA;
architecture Behavioral of HA is
begin
    s <= a xor b;
    c <= a and b;
end Behavioral;
```

fa.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FA is
    port(a, b, c_in : in std_logic;
         s, c_out : out std_logic);
end FA;
architecture Behavioral of FA is
begin
    s <= a xor b xor c_in;
    c_out <= ( (a xor b) and c_in ) or (a and b);
end Behavioral;
```

const.xdc

```
set_property PACKAGE_PIN V17 [get_ports {a[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property PACKAGE_PIN V16 [get_ports {a[1]}]
set_property PACKAGE_PIN W16 [get_ports {a[2]}]
set_property PACKAGE_PIN W17 [get_ports {a[3]}]

set_property PACKAGE_PIN V15 [get_ports {b[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property PACKAGE_PIN W14 [get_ports {b[1]}]
set_property PACKAGE_PIN W13 [get_ports {b[2]}]
set_property PACKAGE_PIN V2 [get_ports {b[3]}]

set_property PACKAGE_PIN T2 [get_ports {sel[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sel}]
set_property PACKAGE_PIN R3 [get_ports {sel[1]}]
set_property PACKAGE_PIN W2 [get_ports {sel[2]}]

set_property PACKAGE_PIN U16 [get_ports {out0[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {out0}]
set_property PACKAGE_PIN E19 [get_ports {out0[1]}]
set_property PACKAGE_PIN U19 [get_ports {out0[2]}]
set_property PACKAGE_PIN V19 [get_ports {out0[3]}]

set_property PACKAGE_PIN U15 [get_ports {car}]
        set_property IOSTANDARD LVCMOS33 [get_ports {car}]
```