Satılmış Furkan Kahraman 22002075 EE 102 Section 1 07.11.2022

Lab 4 : Arithmetic Logic Unit Design

Objective

The purpose of this lab was to design an arithmetic logic unit that can perform 8 different operations that we specify with a 4-bit two or a number, and select one of the results of these operations and display it with the help of LED. All of the operations used while making this design are done in separate units, and the results to be displayed with the help of a multiplexer are selected.

Methodology

Before starting the design, it was first chosen which eight processes would be defined in the ALU. These 8 functions are one's complement, and gate, or gate, xnor gate, left shifter, right shifter, 4-bit subtractor and 4-bit adder. Then the inputs and outputs of the ALU were determined. ALU has two 4-bit number inputs (Ain and Bin) and one select input (S). The select input will be used to determine which of the 8 operations is shown in the result section. Table 1 shows which value the select input takes for which operation.

S (select input)	Function	Input	Output
"000"	Summation	A & B	Xout
"001"	AND operation	A & B	Xout
"010"	OR operation	A & B	Xout
"011"	One's complement	A	Xout
"100"	Subtraction	A & B	Xout
"101"	XNOR operation	A & B	Xout
"110"	Left Shift	A	Xout
"111"	Right Shift	A	Xout

Table 1 (Command Table)

ALU has 4 output to show the result. Xout is 4-bit number which equals to result of functions. There are also cout, overflow and cout_substracter. These will be used for addition and subtraction to see if there is any left over and if there is overflow. Then, the codes of 8 different components and 8-to-1 multiplexer were written in separate design sources. These components are defined in the main ALU file. The signals that will establish a connection between these components and the inputs and outputs of the ALU have been defined and connections have been established. All the result signals from the components were connected to the inputs of the multiplexer. Select input is connected to the select part of the multiplexer and Xout gives the desired result. Then, constraint files were written to make the code work with the buttons and LEDs on the board. In the last step, the code is made ready to be sent to the card by generating bitstream and uploaded to the card. It has been tested on the card whether the code works or not.

Design

a) Arithmetic Logic Unit

The ALU is the main unit where the sub-units that perform all the operations are collected. There are 8 different sub-units that enable 8 different operations to be performed here. The Arithmetic Logic Unit consist 8 different operator units. These are one's complement, AND-gate, OR-gate, left shift, XNOR-gate, right shift, subtractor and adder respectively. All sub-units are connected a 8-to-1 multiplexer to select one of the operation.

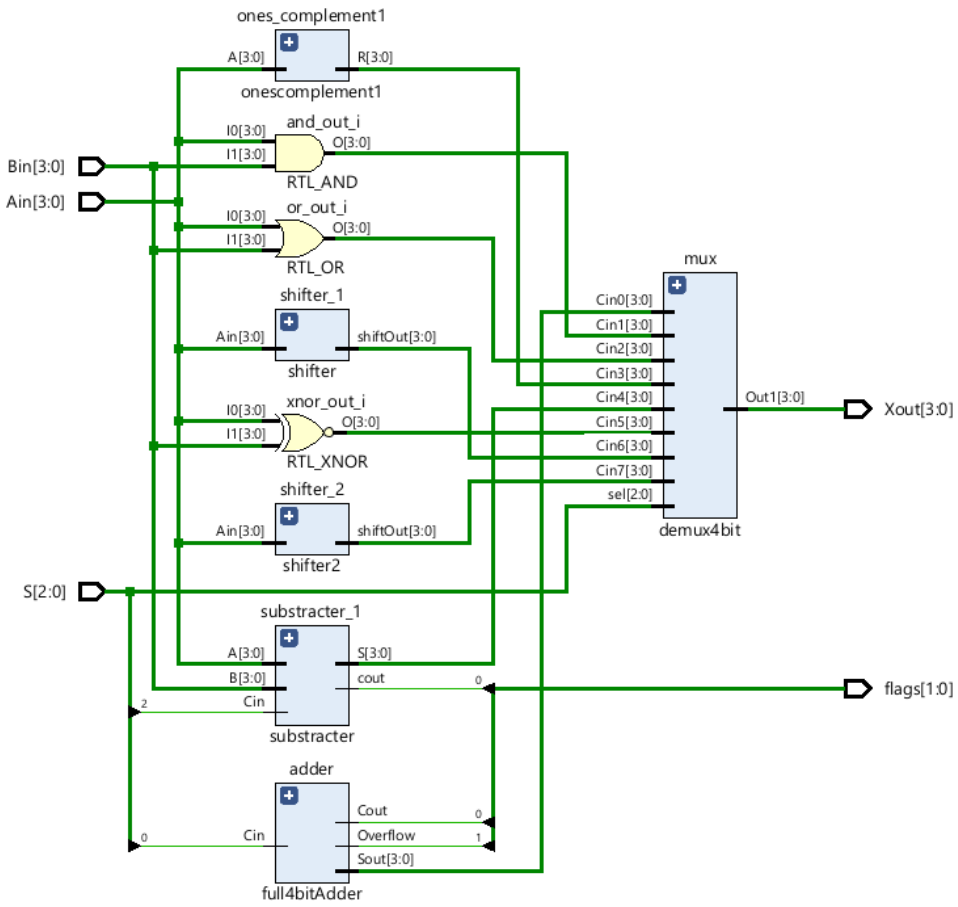

Figure 1 – ALU design

b) 4 Bit Full Adder

4-bit full adder takes three inputs Ain, Bin and Cin. Ain and Bin used for summation operation. Cin used for carry in and it should takes the value of zero. So bit 0 of Select input is connected with Cin to get zero values. Because the addition operation is performed when the select is "000" and in this case Cin always takes the value "0". There are 4 full adder in 4-bit full adder. In each full adder, a step of the addition process is done. And each bit is connected to Sout output which holds the result of the summation. There are also Cout which holds the most significant bits summation's carry out value and Overflow which shows if the summation overflowed or not.

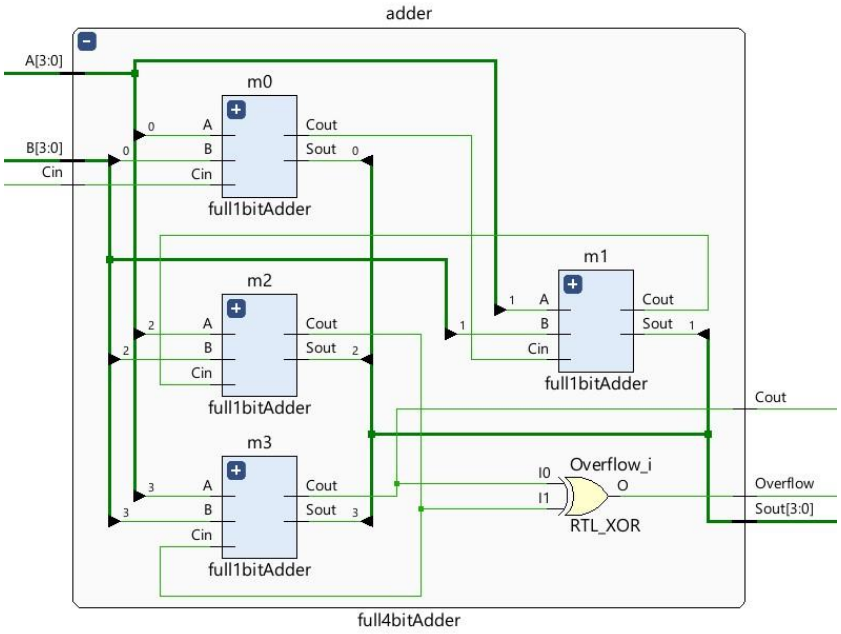

Figure 2 – 4 Bit Adder

c) AND-gate

AND gate has only two 4-bit input and one 4-bit output. Each bit compares with its own equivalent bit. For example Ain(0) is comparing with Bin(0) and if the both values are 1, the AND gate will return 1.

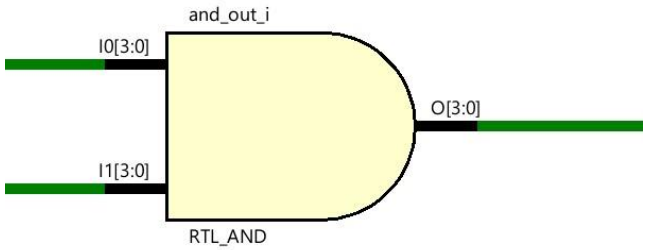

Figure 3 – 4 Bit AND-gate

d) OR-gate

OR gate has only two 4-bit input and one 4-bit output. Each bit compares with its own equivalent bit. For example Ain(0) is comparing with Bin(0) and if the both values are 1, the OR gate will return 1 or Ain(1) = 0 and Bin(1) = 1 then the result will be 1 again.

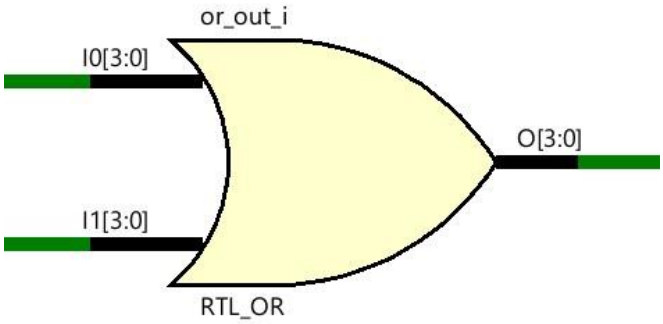

Figure 4 – 4 Bit OR-gate

e) One's Complement

One's complement takes one 4-bit input and gives one 4-bit output. The only operation it performs on the given input is inverting each bit.

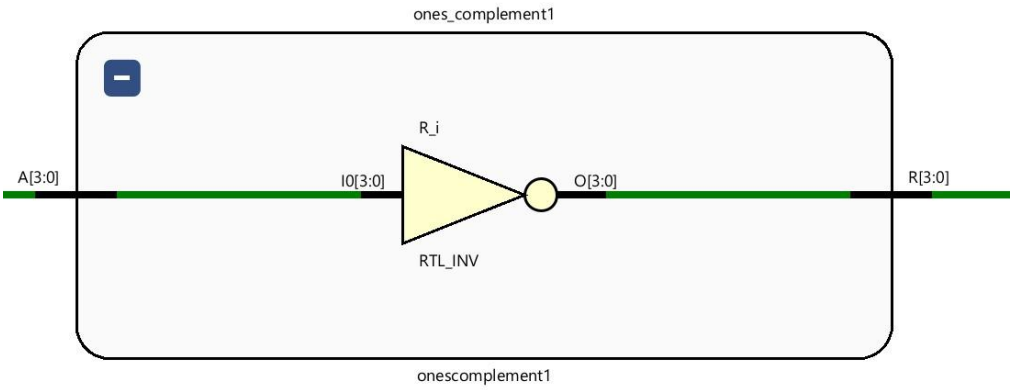

Figure 5 – One's complement

f) Subtraction

4-bit subtractor takes three inputs Ain, Bin and Cin. Ain and Bin used for subtraction operation. Cin used for carry in and it should takes the value of one. So bit 2 of Select input is connected with Cin to get one value. Because the subtraction operation is performed when the select is "100" and in this case Cin always takes the value "0". There are 4 full adder and 4 XOR-gates in 4-bit subtractor. In each full adder, a step of the subtraction process is done. And each bit is connected to Sout output which holds the result of the subtraction. There are also Cout which indicate that the result is positive or negative. If Cout = “1” then the result will be negative.

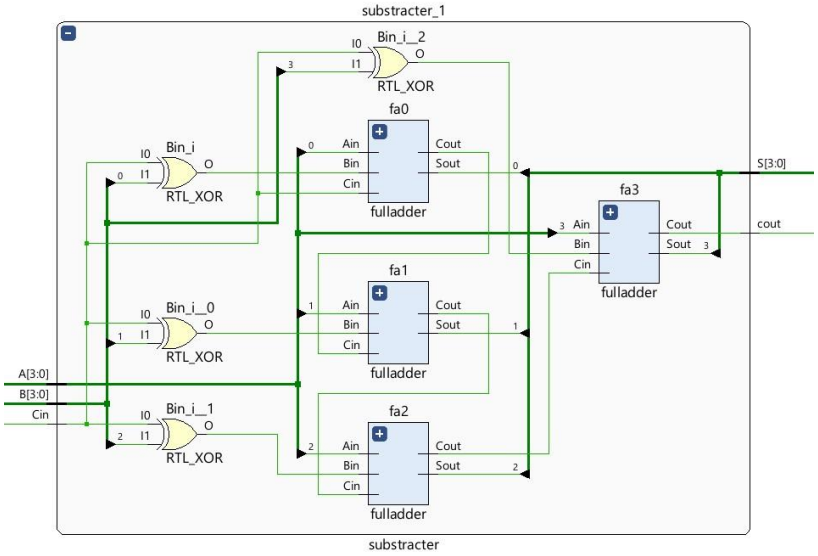

Figure 6 – Subtraction Circuit

g) Right Shift

Left shift takes one input and it gives one output. It adds a zero to the left of the given number and deletes the rightmost number. For example let the given number be Ain = “1010”. Then the taken output will be shiftOut = “0101”.

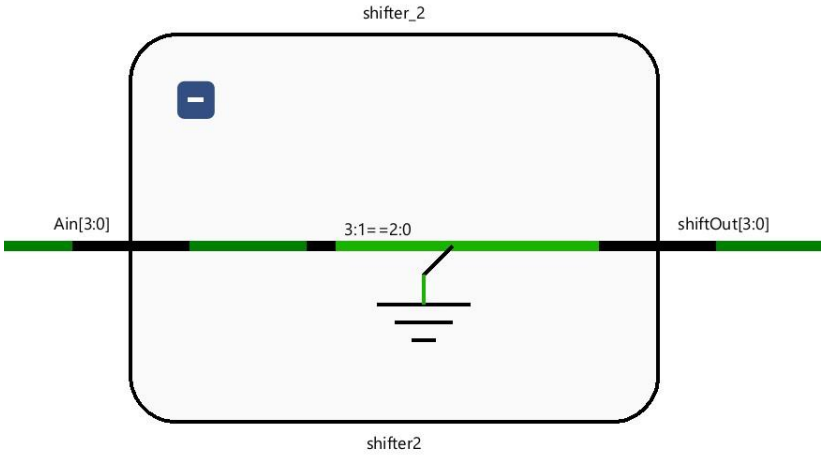

Figure 7 – Right shift

h) Left Shift

Left shift takes one input and it gives one output. It adds a zero to the right of the given number and deletes the leftmost number. For example let the given number be Ain = “1010”. Then the taken output will be shiftOut = “0100”.

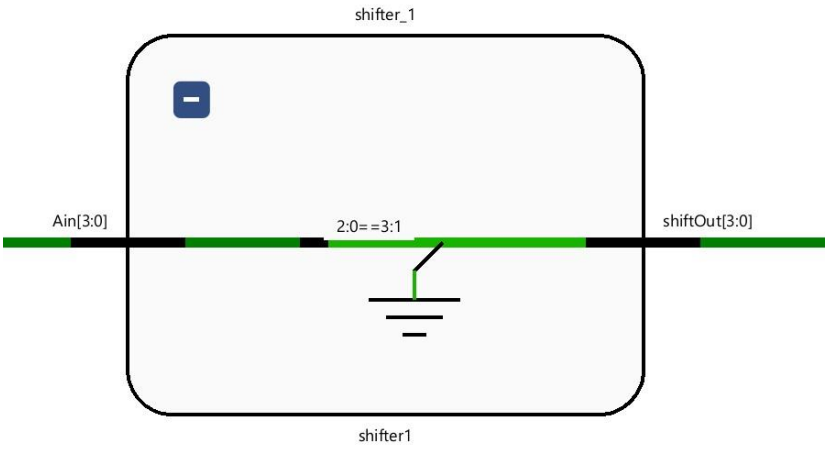

Figure 8 - Left Shift

i) XNOR-gate

XNOR gate has only two 4-bit input and one 4-bit output. Each bit compares with its own equivalent bit. For example Ain(0) is comparing with Bin(0) and if the both values are 1, the XNOR gate will return 01 or Ain(1) = 0 and Bin(1) = 1 then the result will be 0 in this circumstance.

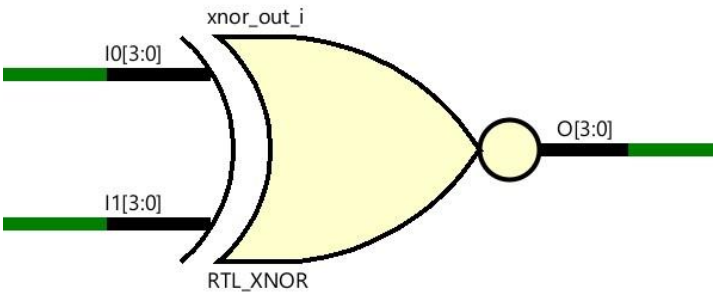

Figure 9 – XNOR gate

Results

The designed Arithmetic logic unit worked as expected in the simulations and on the board. As a result of the inputs entered into the buttons on the card, the LED turned on as we expected it to be. The result are shown below figures.

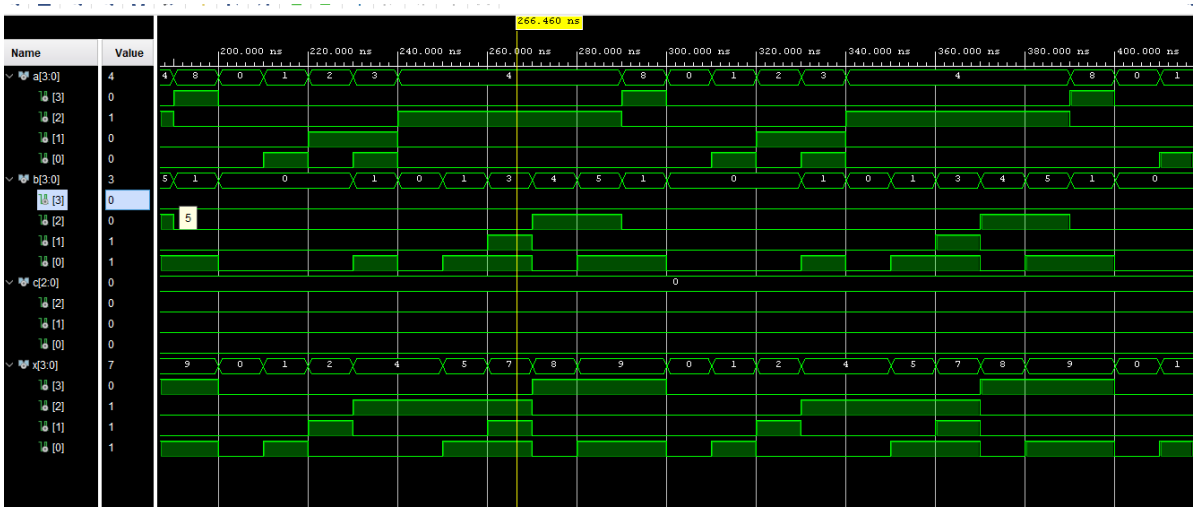

Figure 10 – Summation simulation

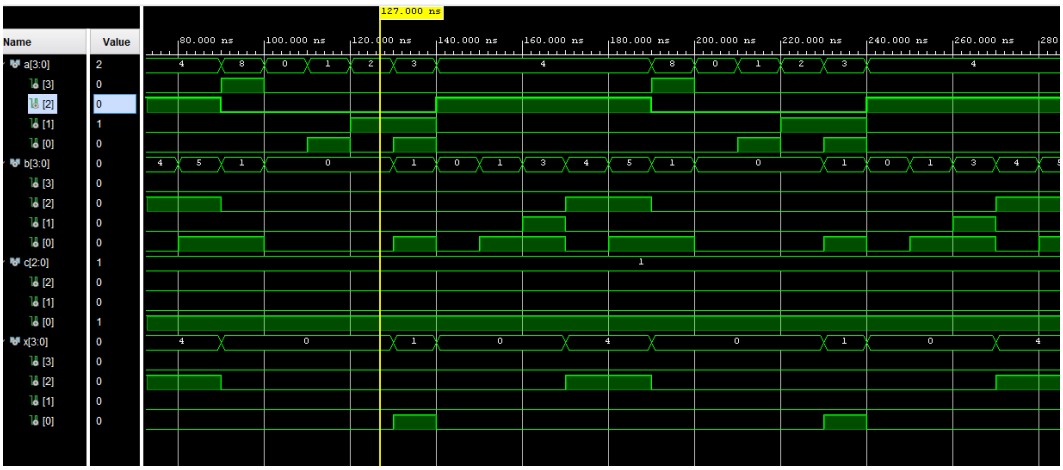

Figure 11 – AND-gate simulation

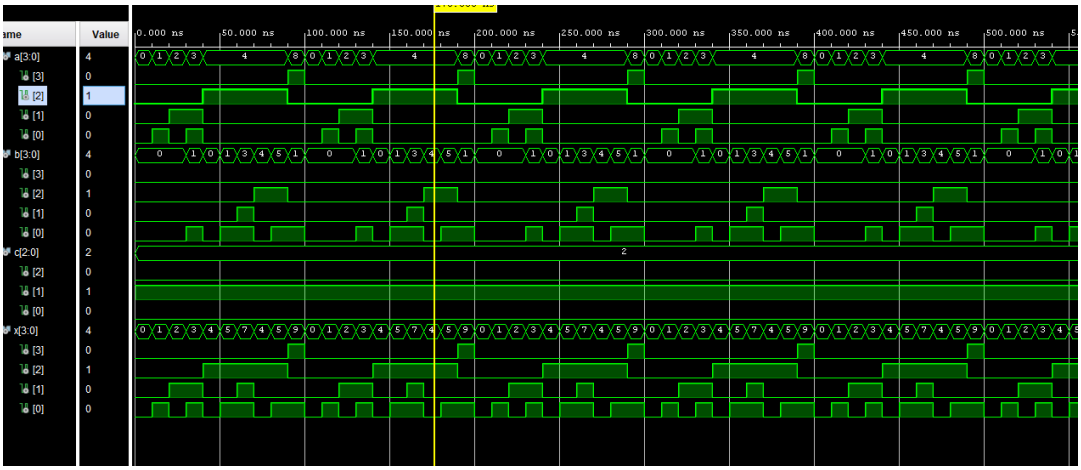

Figure 12 – OR-gate simulation

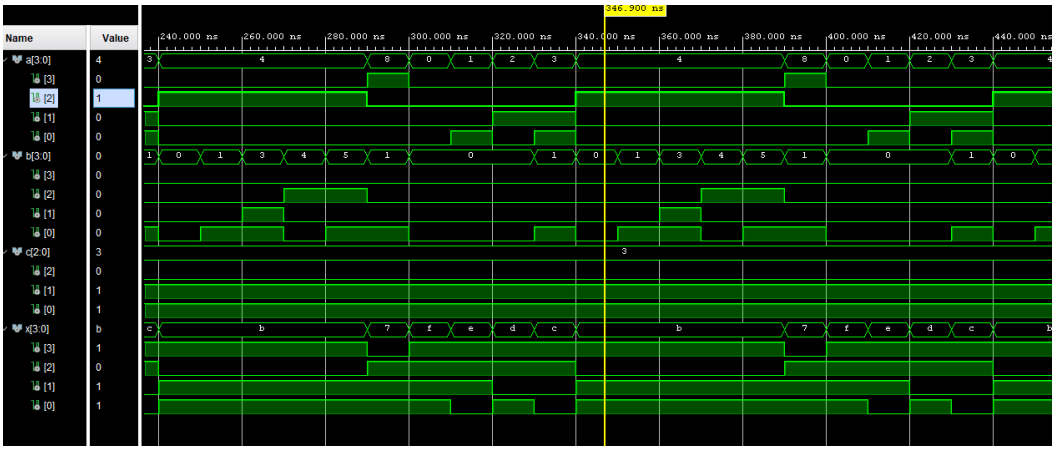

Figure 13 – One’s complement simulation

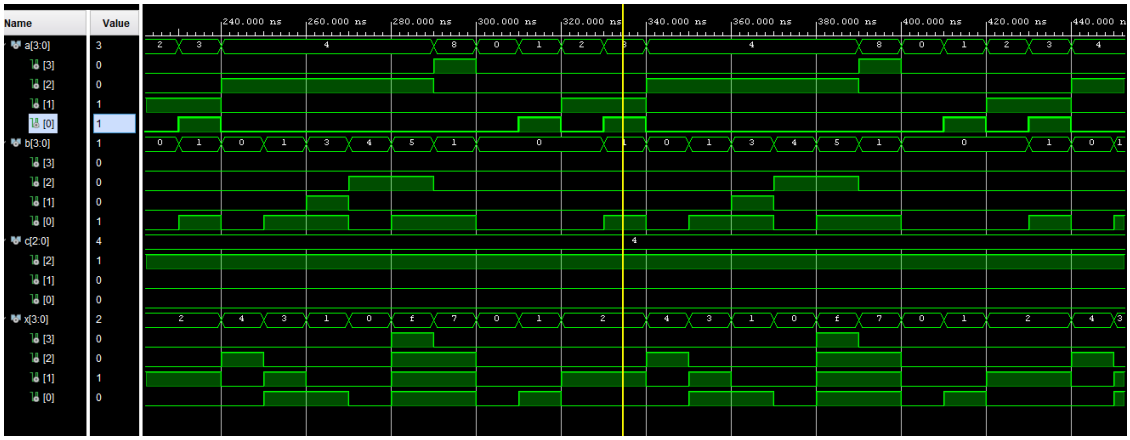

Figure 14 – Subtraction simulation

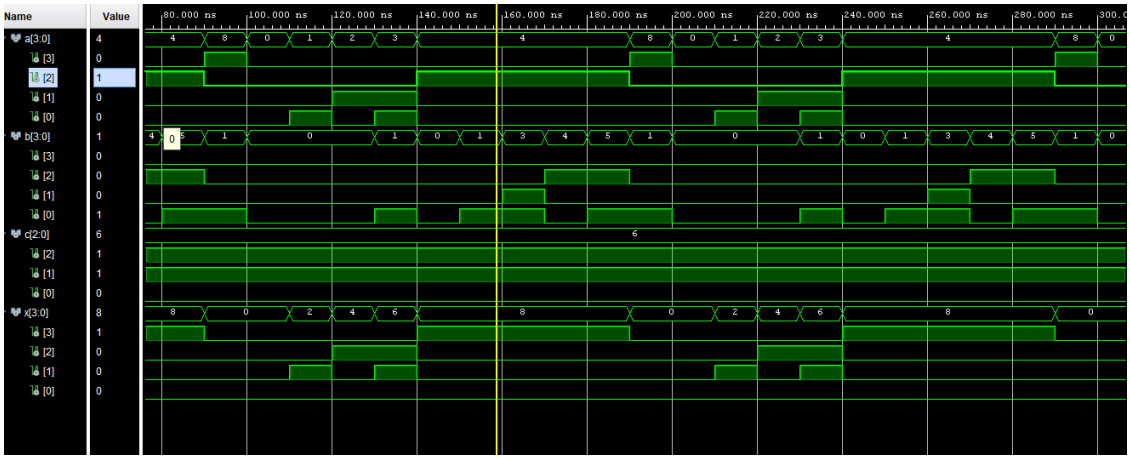

Figure 15 – Left Shift simulation

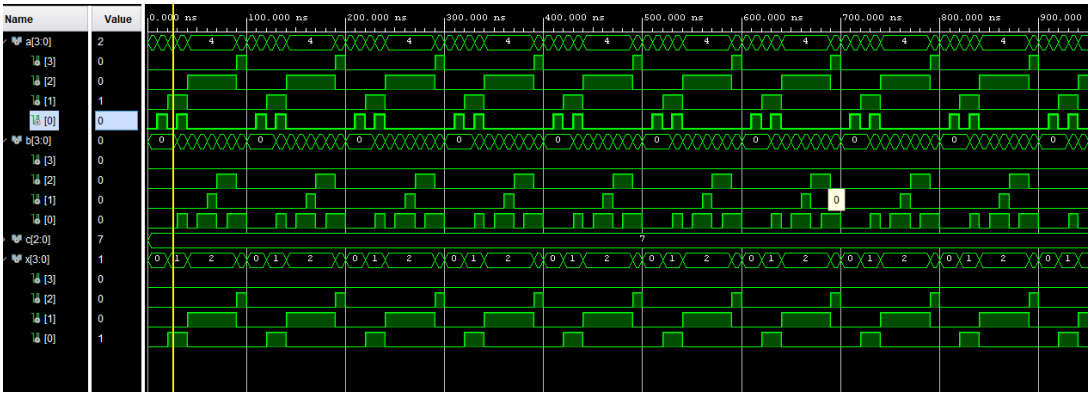

Figure 16 – Right Shift simulation

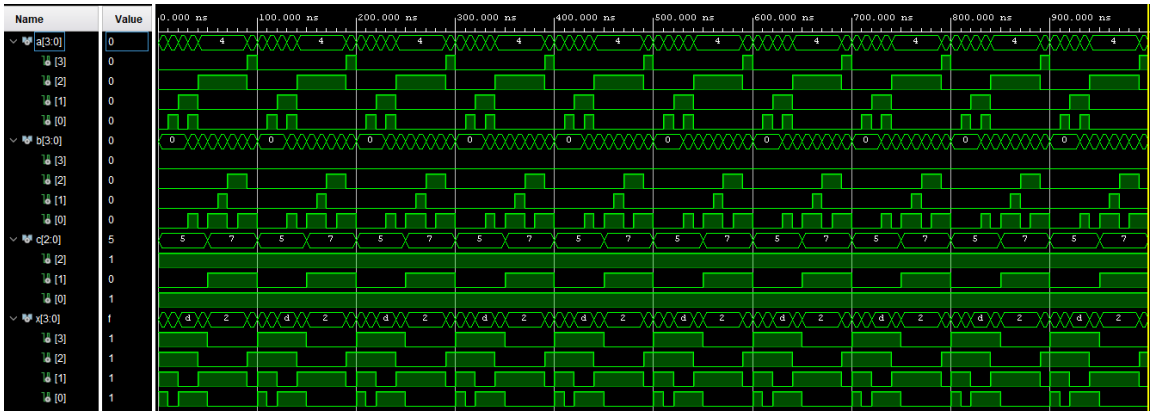

Figure 17 – XOR-gate simulation

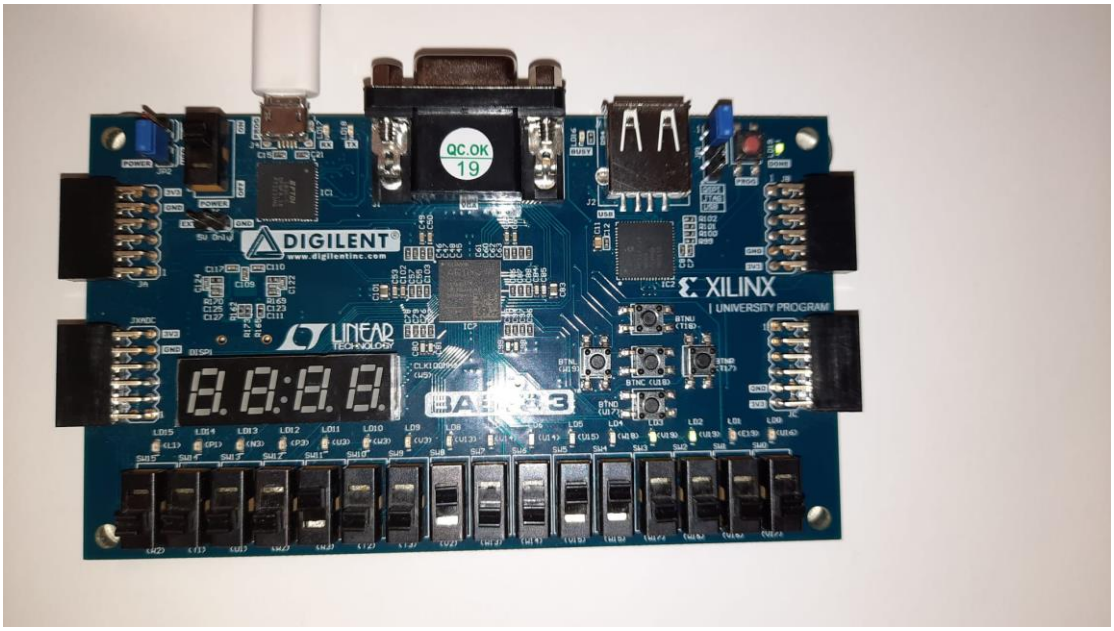

Figure 18 – S = “000” , A = “0011”, B = “1001”, X = “1001 ”

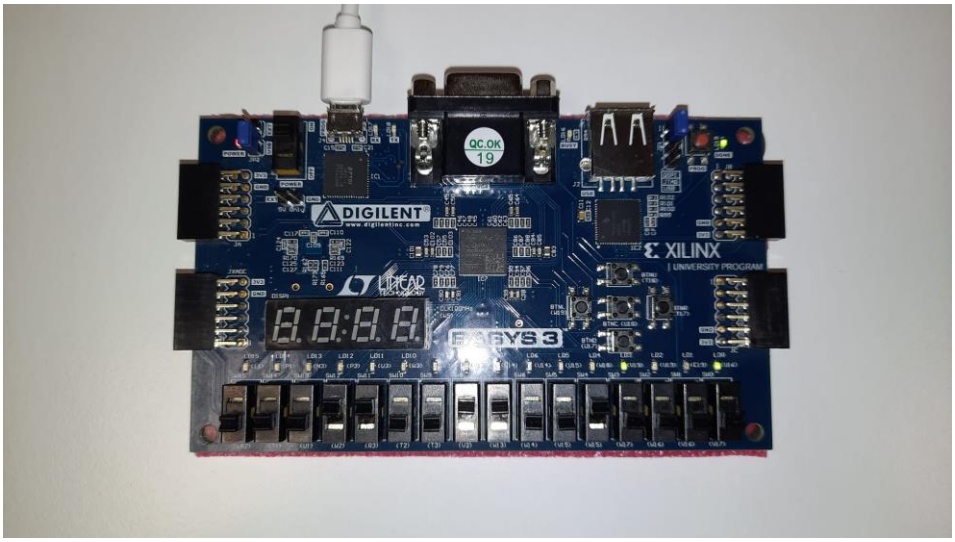

Figure 19 – S = “001” , A = “1001”, B = “1001”, X = “1100 ”

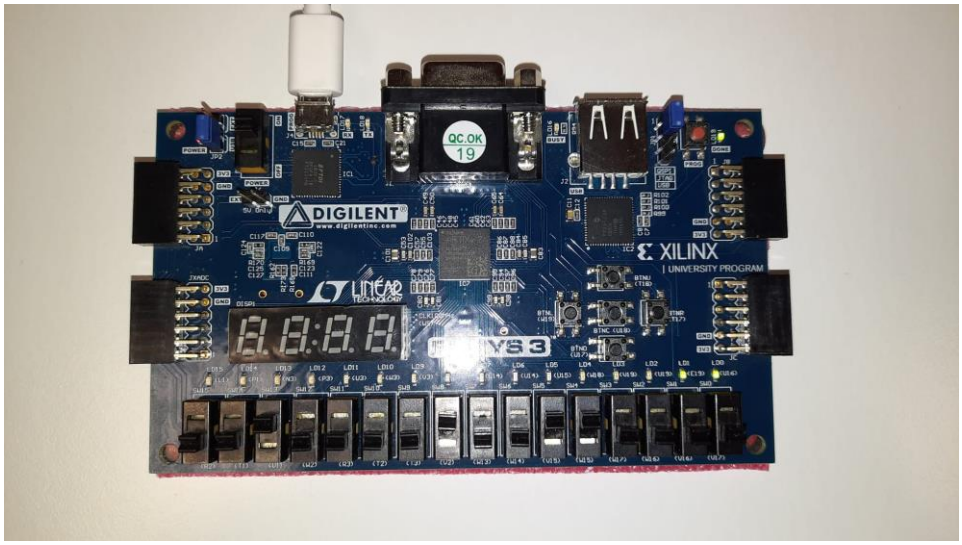

Figure 20 – S = “010” , A = “0011”, B = “0001”, X = “0022 ”

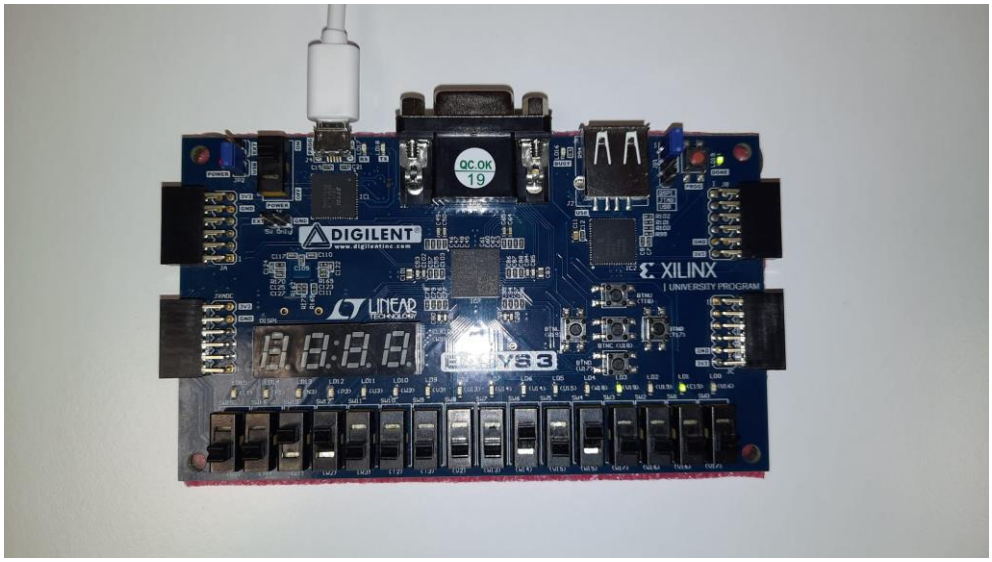

Figure 21 – S = “011” , A = “0101”, B = “0000”, X = “1010 ”

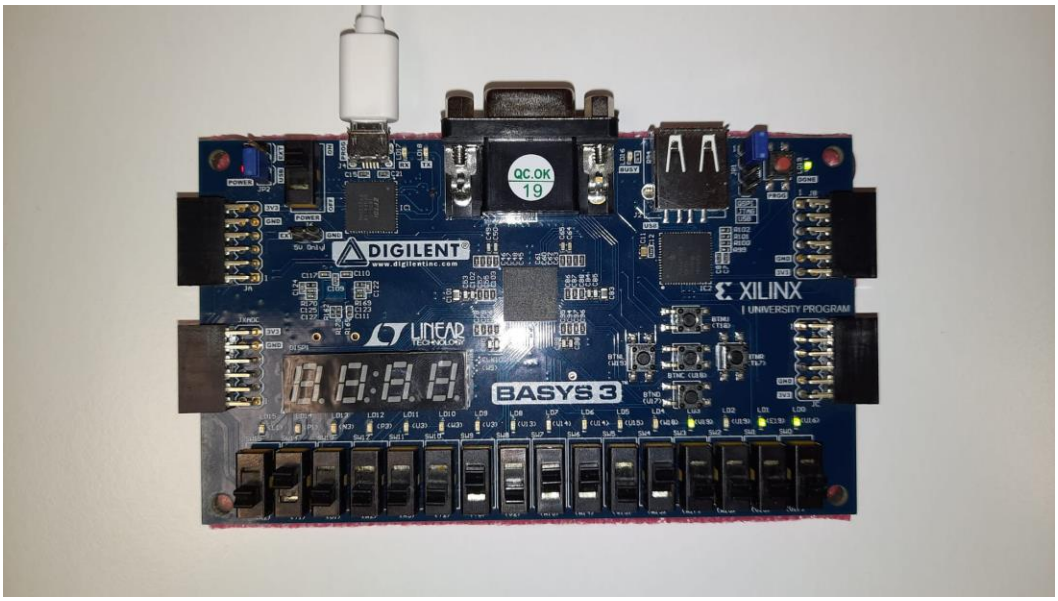

Figure 22 – S = “100” , A = “1101”, B = “0010”, X = “1011 ”

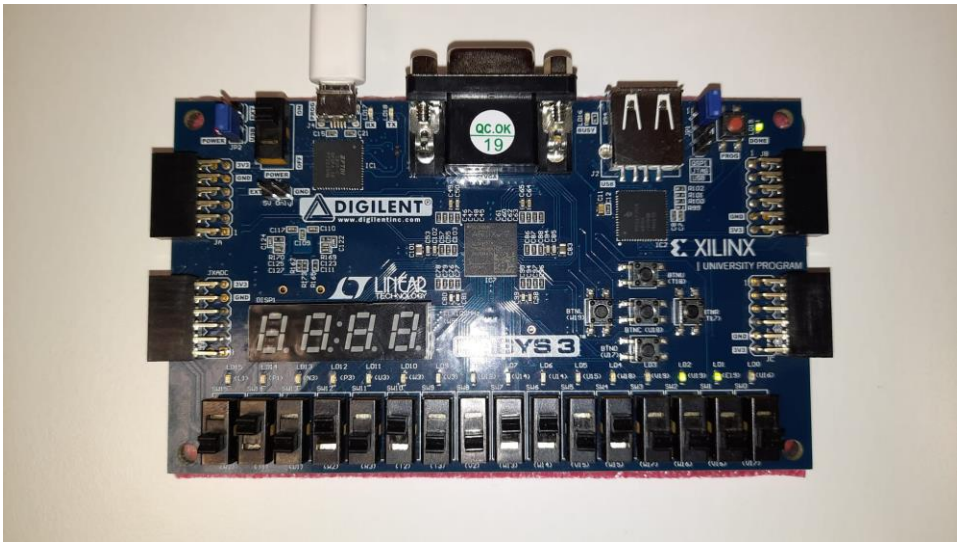

Figure 23 – S = “101” , A = “1101”, B = “0100”, X = “0110 ”

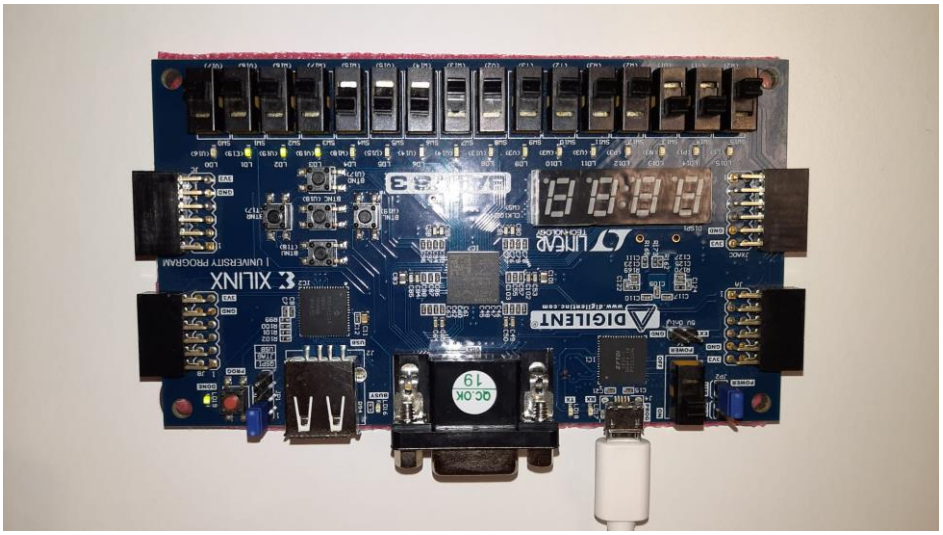

Figure 24 – S = “110” , A = “0111”, B = “0000”, X = “1110 ”

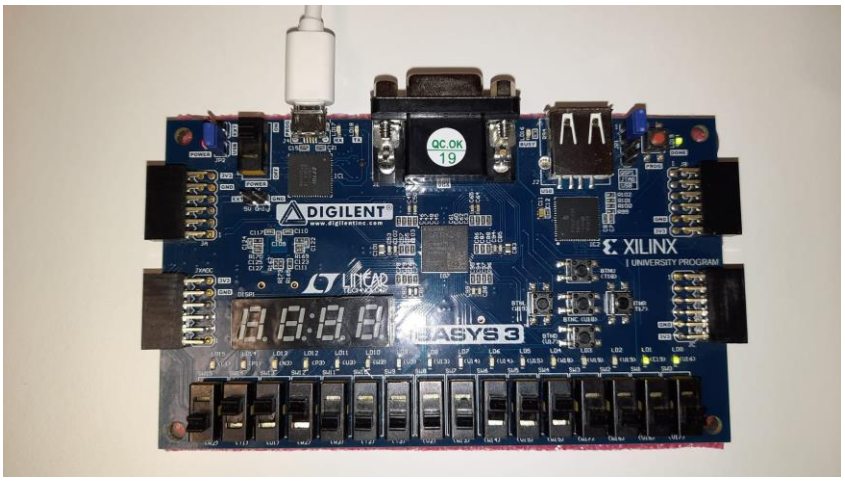

Figure 25 – S = “111” , A = “0111”, B = “0000”, X = “0011 ”

Conclusion

As expected, the arithmetic logic unit showed the results of the necessary operations with LEDs according to the selected values. Expected results expected results were also obtained in the simulation. There is no error in this lab.

Appendices

a) Arithmetic Logic Unit Code

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

entity ALU_design is
   PORT( Ain : in std_logic_vector (3 downto 0);
         Bin : in std_logic_vector (3 downto 0);
         S : in std_logic_vector (2 downto 0);
         Xout : out std_logic_vector (3 downto 0);
         result : out std_logic_vector (1 downto 0));
end ALU_design;

architecture ALU_architecture of ALU_design is

component shifter PORT(
         Ain : in std_logic_vector(3 downto 0);
         shift_Out_1 : out std_logic_vector(3 downto 0));
end component;

component shifter2 PORT(
         Ain : in std_logic_vector(3 downto 0);
         shift_Out_2 : out std_logic_vector(3 downto 0));
end component;

component subtracter PORT( Cin : in std_logic;
         Ain   : in std_logic_vector (3 downto 0);
         Bin   : in std_logic_vector (3 downto 0);
         Sout   : out std_logic_vector(3 downto 0);
         Cout: out std_logic);
end component;

component demux4bit PORT(
       Cin0,Cin1,Cin2,Cin3,Cin4,Cin5,Cin6,Cin7 : in std_logic_vector(3 downto 0);
       Out1 : out std_logic_vector(3 downto 0);
       sel : in std_logic_vector(2 downto 0));
end component;

component onescomplement1 PORT(
       A : in std_logic_vector(3 downto 0);
       D : out std_logic_vector(3 downto 0));
end component;

component full4bitAdder PORT(
       A, B : in std_logic_vector(3 downto 0);
       Cin: in std_logic;
       Cout: out std_logic;
```

```
        Sout : out std_logic_vector(3 downto 0);
        Overflow: out std_logic);
end component;


signal binary_adder_1: std_logic_vector(3 downto 0);
signal or_out: std_logic_vector(3 downto 0);
signal and_out: std_logic_vector(3 downto 0);
signal shift_out_1: std_logic_vector(3 downto 0);
signal alu_out: std_logic_vector(3 downto 0);
signal log_out: std_logic_vector(3 downto 0);
signal adder_out: std_logic_vector(3 downto 0);
signal subtracter1: std_logic_vector(3 downto 0);
signal xnor_out : std_logic_vector (3 downto 0);
signal shift_out_2 : std_logic_vector (3 downto 0);
signal ones_complement : std_logic_vector (3 downto 0);


begin


ones_complement1 : onescomplement1 port map(A => Ain, D => ones_complement);
shifter_1 : shifter port map(Ain=>Ain, shift_Out_1 => shift_out);
substracter_1 : substracter port map(Cin => Sout(2), Ain => Ain, Bin => Bin,  Sout => substracter1,
Cout => result(0));
mux: demux4bit port map(Cin0=>adder_out,Cin1=>and_out,Cin2=>or_out,Cin3=>
ones_complement,Cin4 =>subtracter1, Cin5 => xnor_out, Cin6 => shift_out_1 ,Cin7 =>
shift_out_2 ,Out1=>Xout,sel=>S);
adder: full4bitAdder port
map(A=>Alu_out,B=>binary_adder_1,Cin=>S(0),Cout=>result(0),Overflow=>result(1),
Sout=>adder_out);
shifter_2 : shifter port map(Ain=>Ain, shift_Out_2 => shift_out2);
and_out <= Ain and Bin;
or_out <= Ain or Bin;
xnor_out <= Ain xnor Bin;

end ALU_architecture;
```

b) One's Complement

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


entity onescomplement1 is
    Port (A : in STD_LOGIC_VECTOR(3 downto 0);
          D : out STD_LOGIC_VECTOR(3 downto 0));
end onescomplement1;
```

```
architecture Behavioral of onescomplement1 is

begin

R <= not A;

end Behavioral;
```

c) Left Shift

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

entity shifter is

    PORT( Ain : in std_logic_vector(3 downto 0);
          shiftOut : out std_logic_vector(3 downto 0));

end shifter;

architecture behavioral of shifter is

begin

shiftOut<= Ain(2 downto 0)&'0';

end behavioral;
```

d) Right Shift

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

entity shifter2 is

    PORT( Ain : in std_logic_vector(3 downto 0);
          shiftOut : out std_logic_vector(3 downto 0));

end shifter2;

architecture behavioral of shifter2 is

begin

shiftOut<= '0' & Ain(3 downto 1);

end behavioral;
```

e) Subtraction

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

entity halfadder is
    PORT ( A, B : in std_logic;
           Cout, Sout : out std_logic);
end halfadder;

architecture arch_ha of halfadder is
begin
Cout <= A and B;
Sout <= A xor B;
end arch_ha;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity fulladder is
    PORT (Ain, Bin, Cin : in std_logic;
          Cout, Sout: out std_logic);

end fulladder;

architecture arch_fa of fulladder is

component halfadder
    PORT (A, B : in std_logic;
          Cout, Sout: out std_logic);

end component;

signal sumAB : std_logic;
signal carry1: std_logic;
signal carry2: std_logic;

begin

halfadder1: halfadder port map(A =>Ain , B =>Bin, Sout=>sumAB, Cout=>carry1);
halfadder2: halfadder port map(A=>sumAB, B=>Cin, Sout=>Sout, Cout=>carry2);
Cout <= carry1 or carry2;
end arch_fa;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity subtracter is
```

```
    PORT( Cin : in std_logic;
          A   : in std_logic_vector (3 downto 0);
          B   : in std_logic_vector (3 downto 0);
          S   : out std_logic_vector(3 downto 0);
          cout: out std_logic);

end substracter;

architecture Behavioral of substracter is

component fulladder
    PORT (Ain, Bin, Cin : in std_logic;
          Cout, Sout: out std_logic);
end component;

signal c0: std_logic;
signal c1: std_logic;
signal c2: std_logic;
signal xor0 : std_logic;
signal xor1 : std_logic;
signal xor2 : std_logic;
signal xor3 : std_logic;

begin

fa0 : fulladder port map( Ain => A(0), Bin => Cin xor B(0), Cin => Cin, Cout => c0, Sout => S(0));
fa1 : fulladder port map( Ain => A(1), Bin => Cin xor B(1), Cin => c0, Cout => c1, Sout => S(1));
fa2 : fulladder port map( Ain => A(2), Bin => Cin xor B(2), Cin => c1, Cout => c2, Sout => S(2));
fa3 : fulladder port map( Ain => A(3), Bin => Cin xor B(3), Cin => c2, Cout => cout, Sout => S(3));

end Behavioral;
```

f) Multiplexer

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

entity demux4bit is
    PORT( Cin0 : in std_logic_vector(3 downto 0);
          Cin1 : in std_logic_vector(3 downto 0);
          Cin2 : in std_logic_vector(3 downto 0);
          Cin3 : in std_logic_vector(3 downto 0);
          Cin4 : in std_logic_vector(3 downto 0);
          Cin5 : in std_logic_vector(3 downto 0);
          Cin6 : in std_logic_vector(3 downto 0);
          Cin7 : in std_logic_vector(3 downto 0);
          Out1 : out std_logic_vector(3 downto 0);
          sel : in std_logic_vector(2 downto 0));
end demux4bit;
```

```
architecture behavioral of demux4bit is

begin
 process (sel, Cin0, Cin1, Cin2, Cin3, Cin4, Cin5, Cin6, Cin7)
  begin
   case sel is
   when "000" => Out1 <= Cin0;
   when "001" => Out1 <= Cin1;
   when "010" => Out1 <= Cin2;
   when "011" => Out1 <= Cin3;
   when "100" => Out1 <= Cin4;
   when "101" => Out1 <= Cin5;
   when "110" => Out1 <= Cin6;
   when "111" => Out1 <= Cin7;
   end case;
 end process;

end behavioral;
```

g) 4 Bit Adder

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

entity halfAdder is
        PORT( A, B : in std_logic;
              Cout, Sout : out std_logic);
end halfAdder;

architecture behavioral of halfAdder is

begin
Sout <=A xor B;
Cout <=A and B;
end behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full1bitAdder is

    PORT( A, B, Cin : in std_logic;
          Cout, Sout : out std_logic);
end full1bitAdder;

architecture behavioral of full1bitAdder is

component halfAdder
```

```
    PORT( A, B : in std_logic;
          Cout, Sout : out std_logic);
end component;

signal sum_AB: std_logic;
signal carry1: std_logic;
signal carry2: std_logic;

begin

halfAdder1: halfAdder port map(A=>A,B=>B,Sout=>sum_AB,Cout=>carry1);
halfAdder2: halfAdder port map(A=>sum_AB,B=>Cin,Sout=>Sout,Cout=>carry2);
Cout <= carry1 or carry2;

end behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full4bitAdder is
    PORT( A, B : in std_logic_vector(3 downto 0);
          Cin: in std_logic;
          Cout: out std_logic;
          Sout: out std_logic_vector(3 downto 0);
          Overflow: out std_logic);
end full4bitAdder;

architecture behavioral of full4bitAdder is

component full1bitAdder
    PORT( A, B, Cin : in std_logic;
          Cout, Sout : out std_logic);
end component;

signal CoutInside:std_logic_vector(4 downto 1);

begin
m0: full1bitAdder port map(A=>A(0),B=>B(0),Cin=>Cin,Sout=>Sout(0),Cout=>CoutInside(1));
m1: full1bitAdder port
map(A=>A(1),B=>B(1),Cin=>CoutInside(1),Sout=>Sout(1),Cout=>CoutInside(2));
m2: full1bitAdder port
map(A=>A(2),B=>B(2),Cin=>CoutInside(2),Sout=>Sout(2),Cout=>CoutInside(3));
m3: full1bitAdder port
map(A=>A(3),B=>B(3),Cin=>CoutInside(3),Sout=>Sout(3),Cout=>CoutInside(4));
Overflow<= CoutInside(4) xor CoutInside(3);
Cout <= CoutInside(4);

end behavioral;
```

h) Constraint

```
set_property PACKAGE_PIN W15 [get_ports {Ain[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Ain[0]}]
set_property PACKAGE_PIN V15 [get_ports {Ain[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Ain[1]}]
set_property PACKAGE_PIN W14 [get_ports {Ain[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Ain[2]}]
set_property PACKAGE_PIN W13 [get_ports {Ain[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Ain[3]}]
set_property PACKAGE_PIN V2 [get_ports {Bin[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Bin[0]}]
set_property PACKAGE_PIN T3 [get_ports {Bin[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Bin[1]}]
set_property PACKAGE_PIN T2 [get_ports {Bin[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Bin[2]}]
set_property PACKAGE_PIN R3 [get_ports {Bin[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Bin[3]}]
set_property PACKAGE_PIN W2 [get_ports {S[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {S[0]}]
set_property PACKAGE_PIN U1 [get_ports {S[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {S[1]}]
set_property PACKAGE_PIN T1 [get_ports {S[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {S[2]}]
set_property PACKAGE_PIN U16 [get_ports {Xout[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Xout[0]}]
set_property PACKAGE_PIN E19 [get_ports {Xout[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Xout[1]}]
set_property PACKAGE_PIN U19 [get_ports {Xout[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Xout[2]}]
set_property PACKAGE_PIN V19 [get_ports {Xout[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {Xout[3]}]
set_property PACKAGE_PIN U15 [get_ports {result[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[0]}]
set_property PACKAGE_PIN U14 [get_ports {result[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[1]}]
```

I) Simulation Code

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.Numeric_Std.all;

entity test is

end test;
```

```
architecture Behavioral of test is

component ALU_design
    PORT( Ain : in std_logic_vector (3 downto 0);
      Bin : in std_logic_vector (3 downto 0);
      S : in std_logic_vector (2 downto 0);
      Xout : out std_logic_vector (3 downto 0);
      result : out std_logic_vector (1 downto 0));
end component;

signal Ain:std_logic_vector(3 downto 0);
signal Bin:std_logic_vector(3 downto 0);
signal S:std_logic_vector(2 downto 0);
signal Xout:std_logic_vector(3 downto 0);
signal result : std_logic_vector(1 downto 0);

begin

 UUT: ALU_design PORT MAP(
     Ain=> Ain,
     Bin=> Bin,
     S => S,
     Xout => Xout,
     flags => flags );

 test : PROCESS

   BEGIN

    Ain <= "0000";
    Bin <= "0010";
    S <= "000";
    wait for 10ns;
    Ain <= "0000";
    Bin <= "0100";
    S <= "000";
    wait for 10ns;
    Ain <= "0100";
    Bin <= "0100";
    S <= "000";
    wait for 10ns;
    Ain <= "1110";
    Bin <= "0100";
    S <= "000";



   end process;
```

```
end Behavioral;
```

(Because of test code part is too long I just shared the summation part of it.)