Defne Yaz Kılıç
22102167
7.11.22
Section 2

EEE102-Lab4 Report
Arithmetic Logic Unit (ALU)

Purpose

The purpose of this lab was to design an arithmetic logic unit (ALU) that has 8 different functionalities. In order to make it more usable it is expected to deploy the ALU on FPGA board and control is functionality manually with switches whose outputs is displayed on LEDs of Basys3. By creating, simulating and deploying a working ALU it is expected that one should have gotten familiar with bitwise arithmetic operations, how to write them in Vhdl and how to simulate their results.

Methodology

An ALU is designed for 3-bit number operations which can compute the 8 functions listed below:

- Addition
- Substraction
- One's complement
- Two's complement
- Left & right logic shift
- Bitwise and
- Bitwise nand
- Bitwise xor

1. Addition

In order to create a 3-bit adder, firstly a half adder (Fig.1) is designed and incorporated in full adder (Fig.2) using Port Map and a general 3 bit adder (Fig.3) is designed using 3 full adders. It includes 6 input signals, each three for 3 bit number, three output signals and one overflow signal.

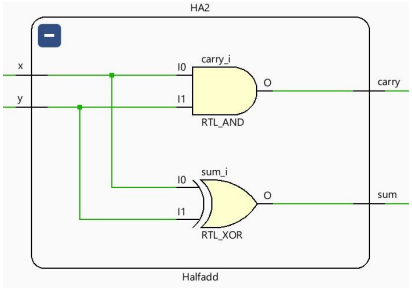


Fig.1 Half Adder

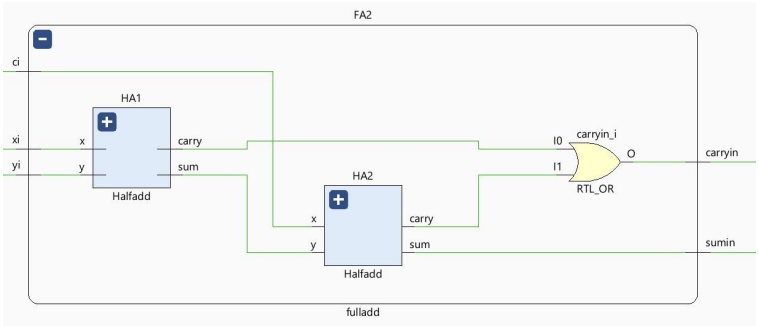


Fig.2 Full Adder

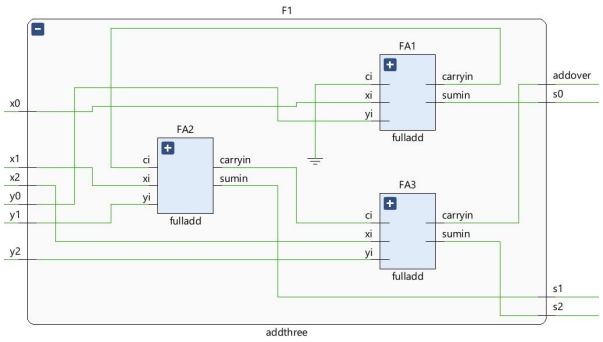


Fig.3 3-bit adder

2. Substraction

Same method used in addition is utilized for substraction except in this case the overflow signal is replaced with borrow signal, the figures of half substractor (Fig.4), full substractor (Fig.5) and 3 bit substractor (Fig.6) can be found below.

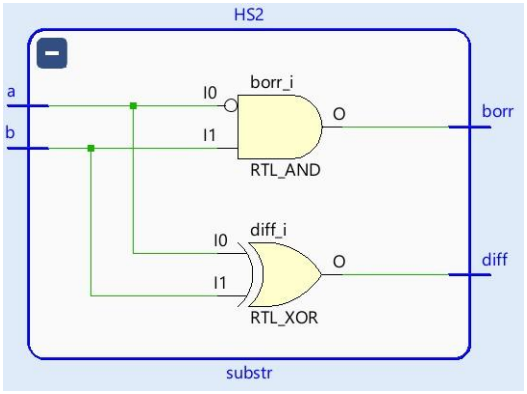


Fig.3 Half Substractor

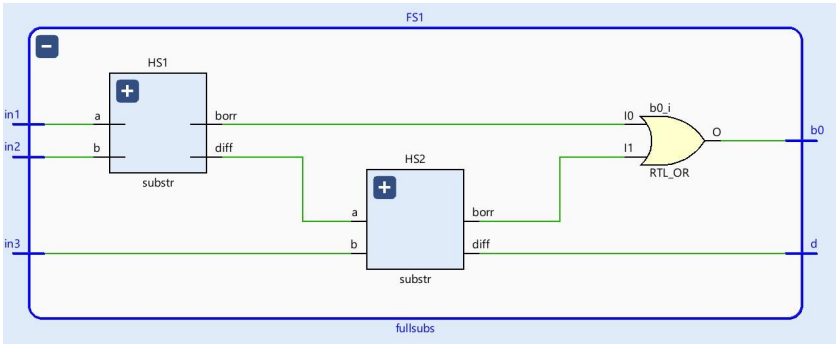


Fig.4 Full Substractor

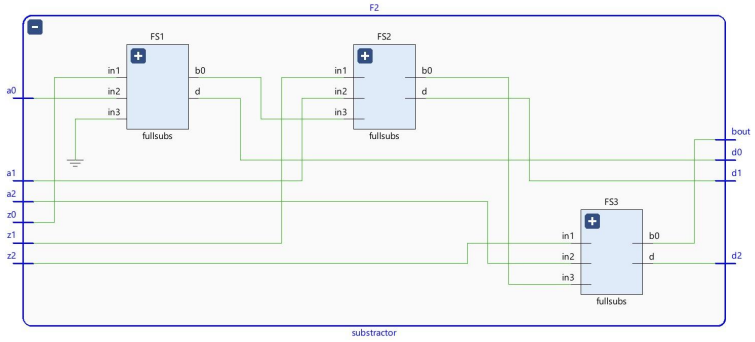


Fig.5 3-bit Substractor

3. One's Complement

For one's complement the inverses of the inputs are taken.(Fig.6)

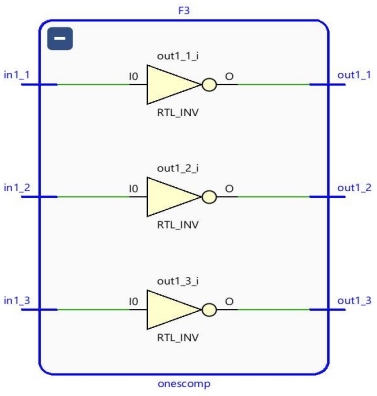


Fig.6 1's complement

4. Two's Complement

To display 2's complement, one's complement and 3- bit adder is combined. (Fig.7)

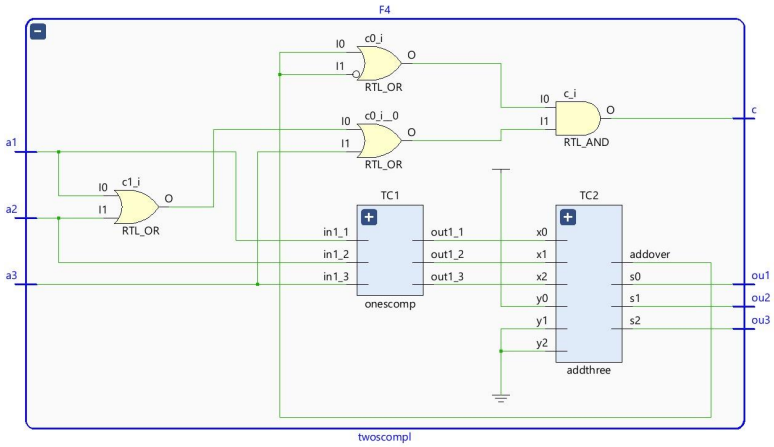


Fig.7 Two's complement

5. Left & Right Logic Shift

The left or right shift states are defined by a select signal defined as 'd', when d <= 1 left shift takes place, when d <= 0 right shift takes place. (Fig.8)

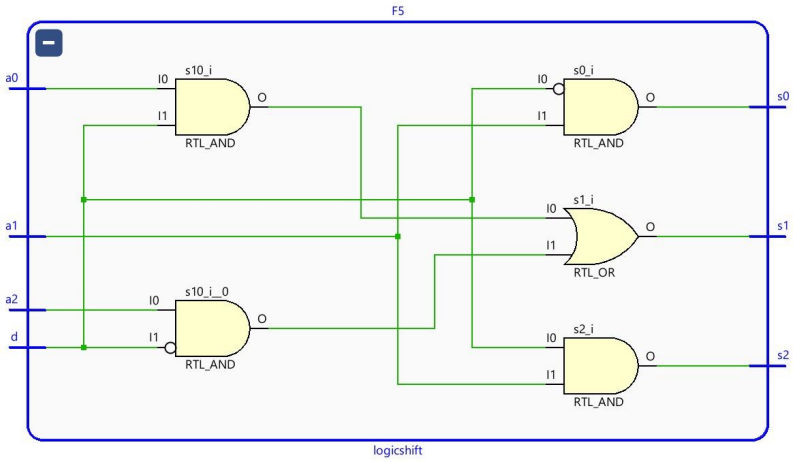


Fig.8 Left and Right Shift

6. Bitwise And

Bits of same significance of each 3 bit number are used as inputs of and gate. (Fig.9)

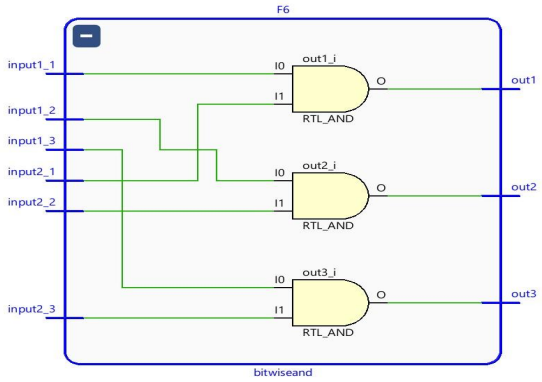


Fig.9 Bitwise And

7. Bitwise Nand

Same procedure is applied as in bitwise and, and gates are replaced wih nand gates. (Fig.10)

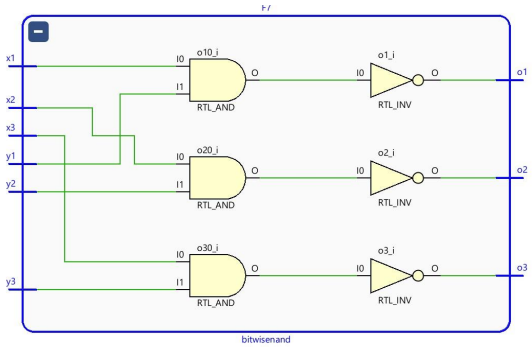


Fig.10 Bitwise Nand

8. Bitwise Xor

Same procedure is applied as in bitwise and, and gates are replaced wih xor gates. (Fig.11)

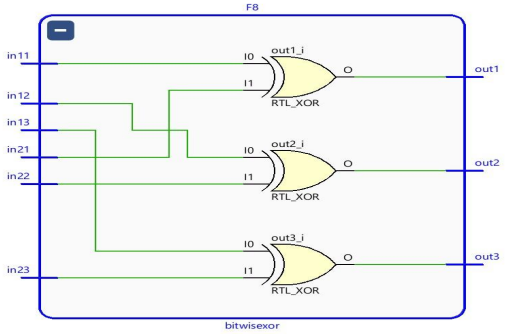


Fig.11 Bitwise Xor

Further Information about Top Module

In order to choose between functions three select signals are used to represent eight different function possibilities (Table 1.). A decoder is used to link the signal coming from select signals and the functions (Fig 12). All the operations are united in a top module (Fig 13.) using Port Map.

Select Signals				
Sel_3	**Sel_2**	**Sel_1**	**Decoder**	**Operation**
0	0	0	m0=1	Adder
0	0	1	m1=1	Subtractor
0	1	0	m2=1	1's comp.
0	1	1	m3=1	2's comp.
1	0	0	m4=1	Logic shift
1	0	1	m5=1	And
1	1	0	m6=1	Nand
1	1	1	m7=1	Xor

Table 1. Operation Select

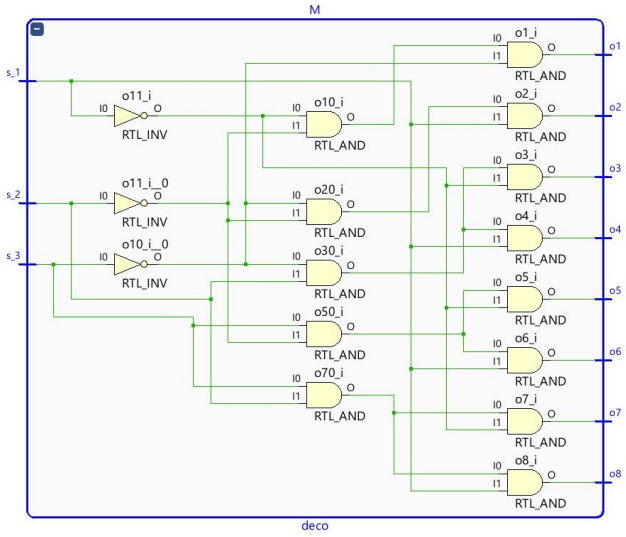


Fig.12 Decoder

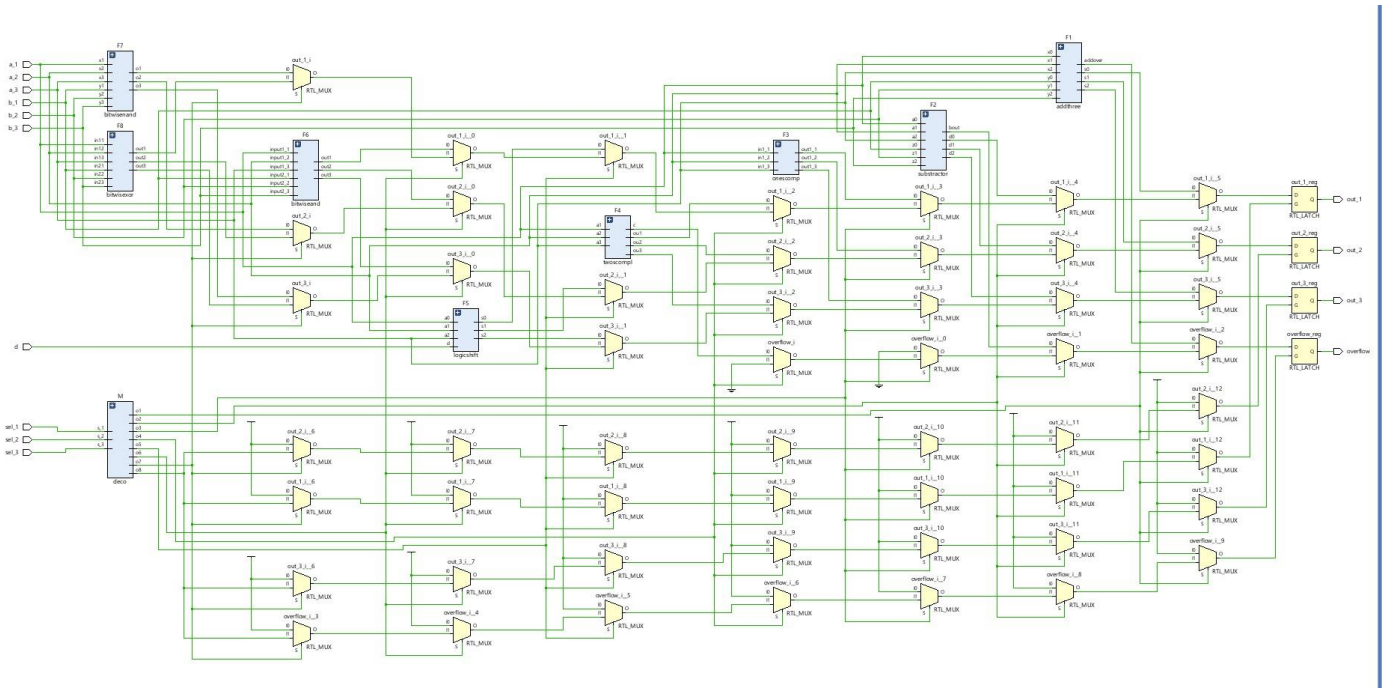


Fig.13 General Schematic of Top Module

Results

For the Bays3 implementation, places of select switches: sel_1, sel_2, sel_3, d (for L or R logic shift) ; places of input switches a_1, a_2, a_3, b_1, b_2, b_3and lastly output LEDs: out_1, out_2, out_3 are assigned using constraints file. Their places can be seen below.

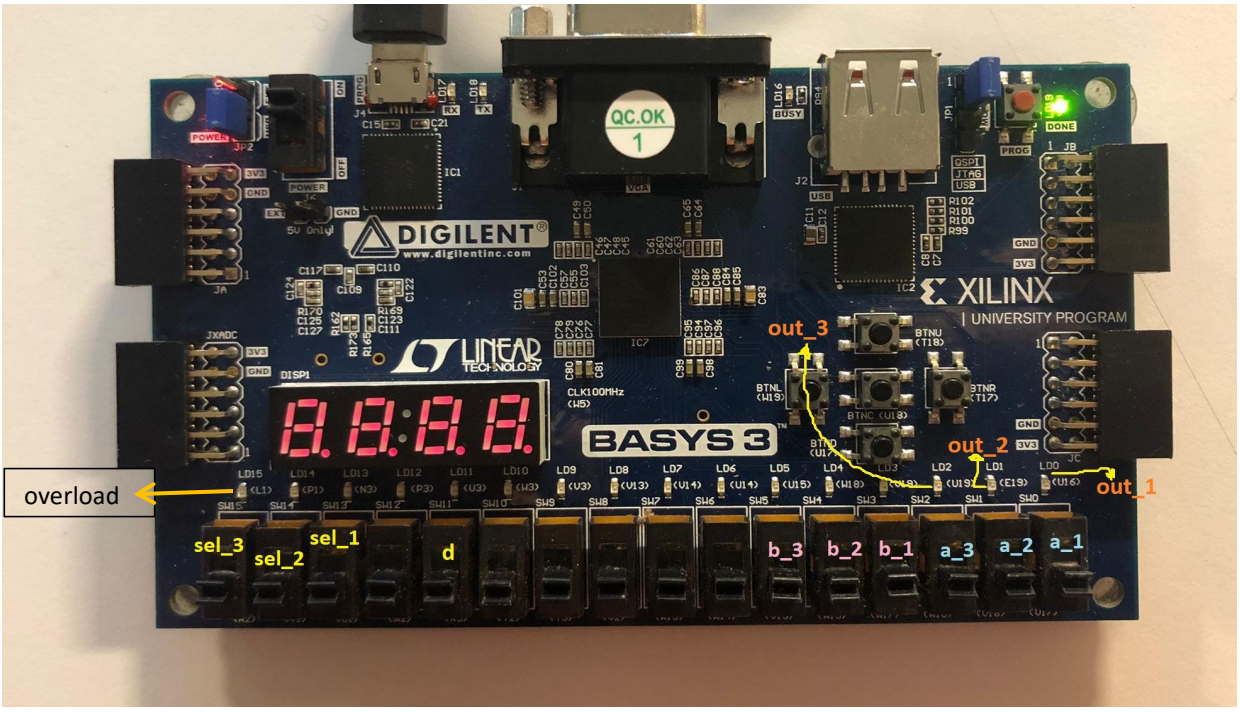
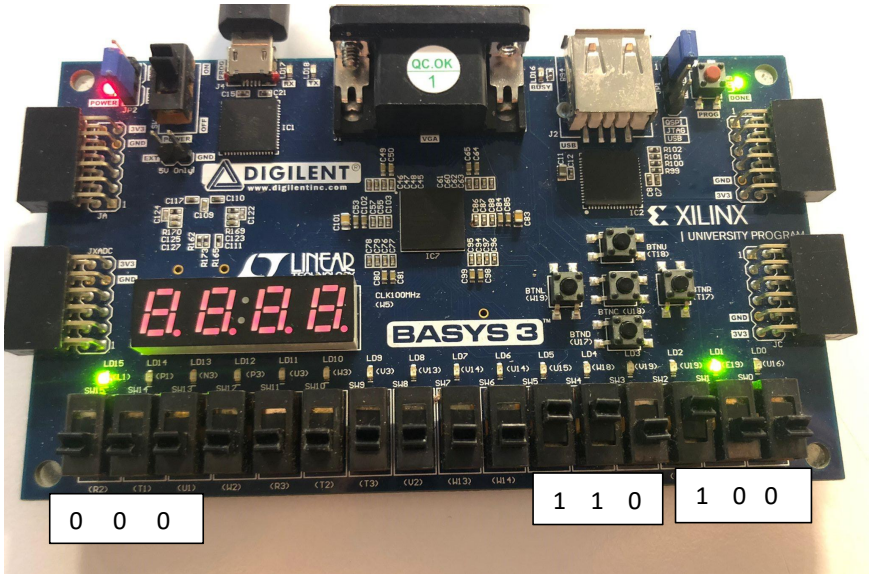


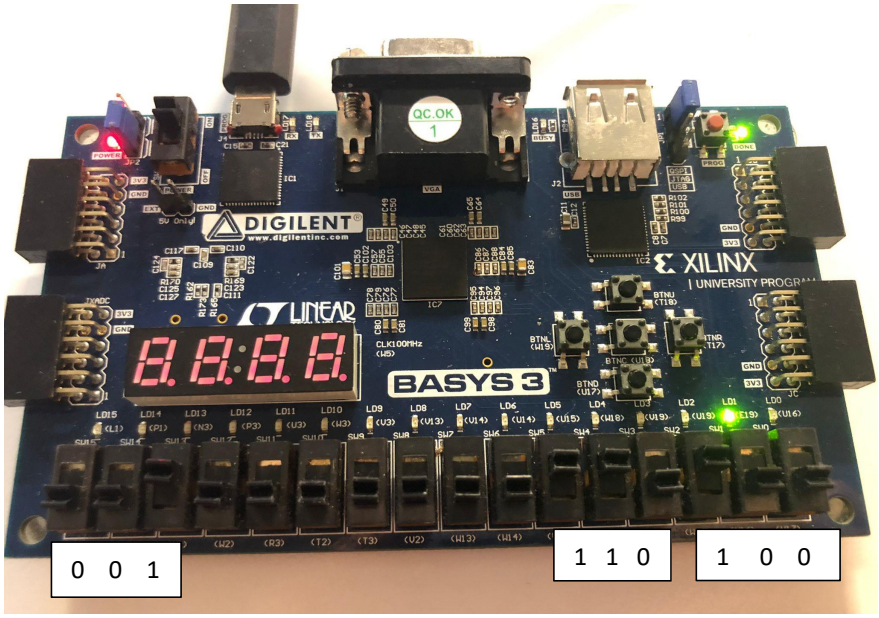
Fig.14 Implementation on Basys3

An example on Basys3 for each operation can be seen below:



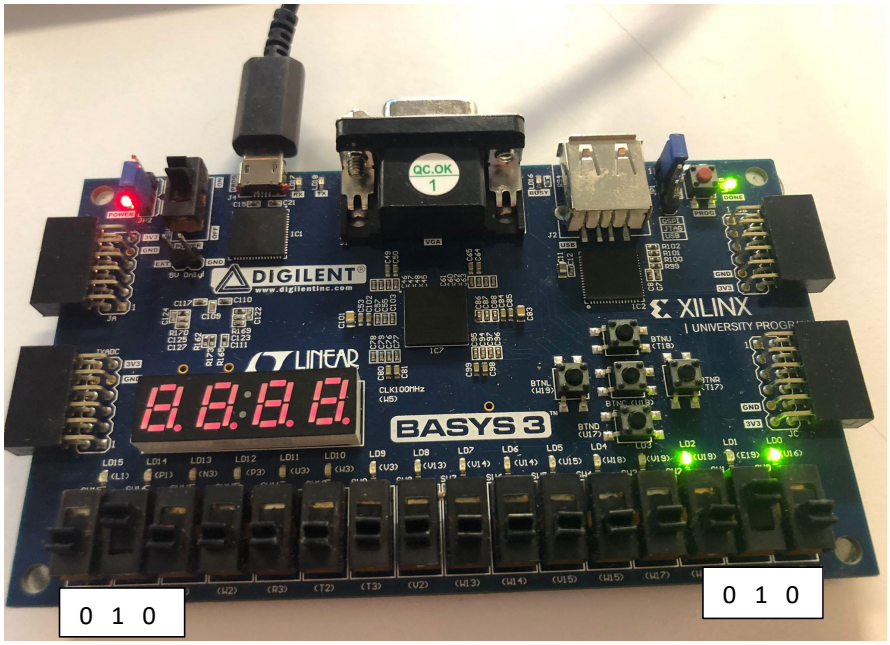
Adder
6+4 =10

Out: 010
Overload: 1



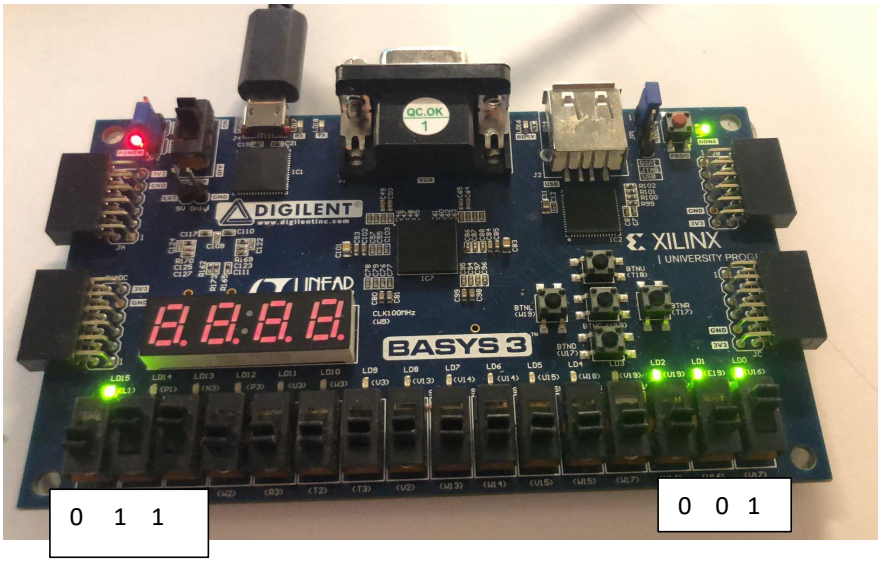
Subtractor
6-4=2

Out: 010
Borrow: 0



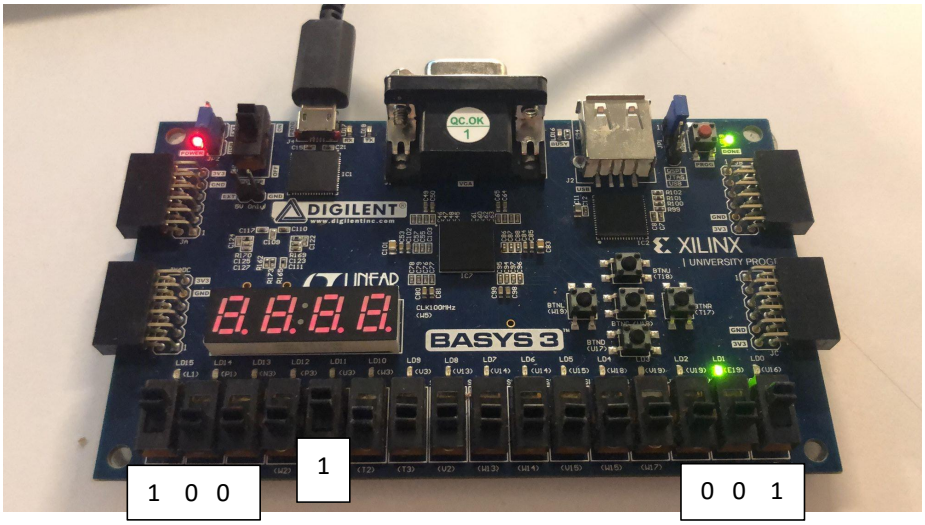
1's complement

Out: 101



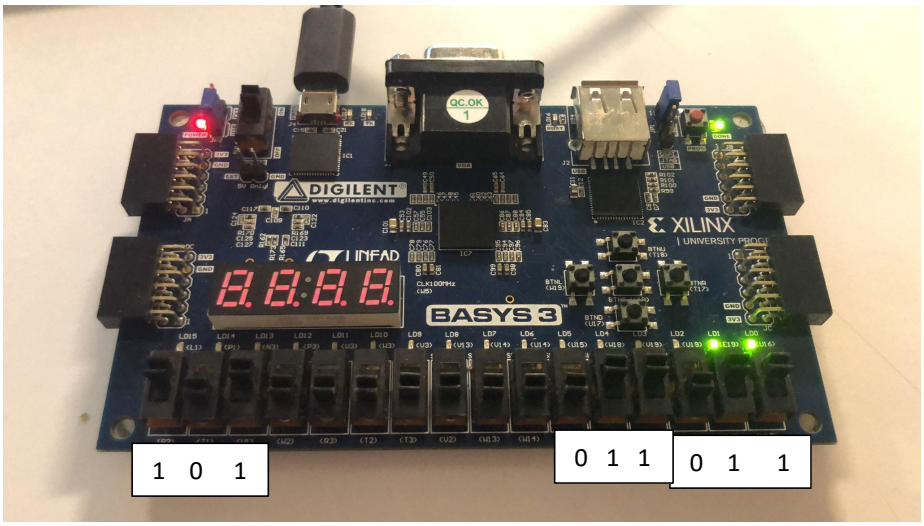
2’s comp.

Out:111
Overflow:1



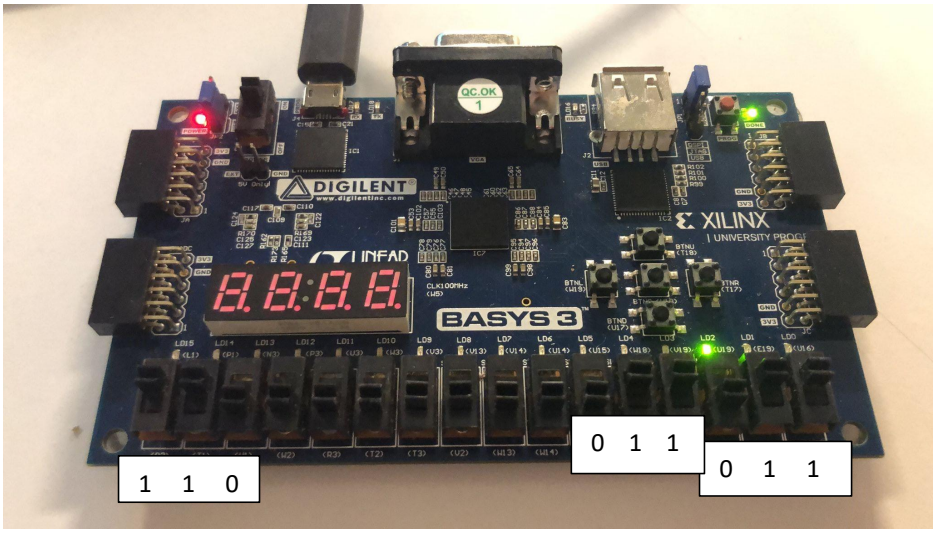
Logic Shift
(d=1, left shift)

Out: 010



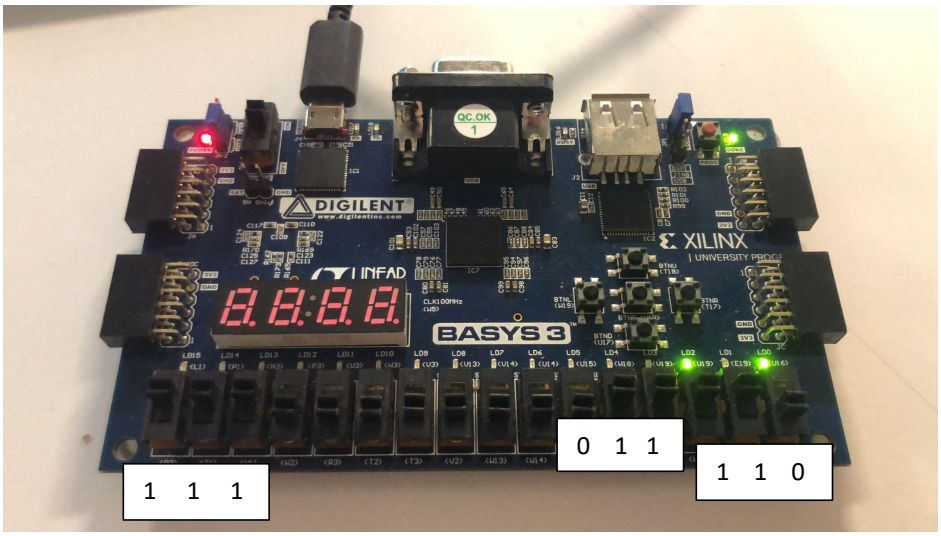
And

Out: 011



Nand

Out: 100



Xor

Out:101

ALU is also simulated in a test bench, whole simulation result can be seen below (Fig.15), separate and more detailed version can be found in Appendix. Design and simulation codes can also be found in Appendix.

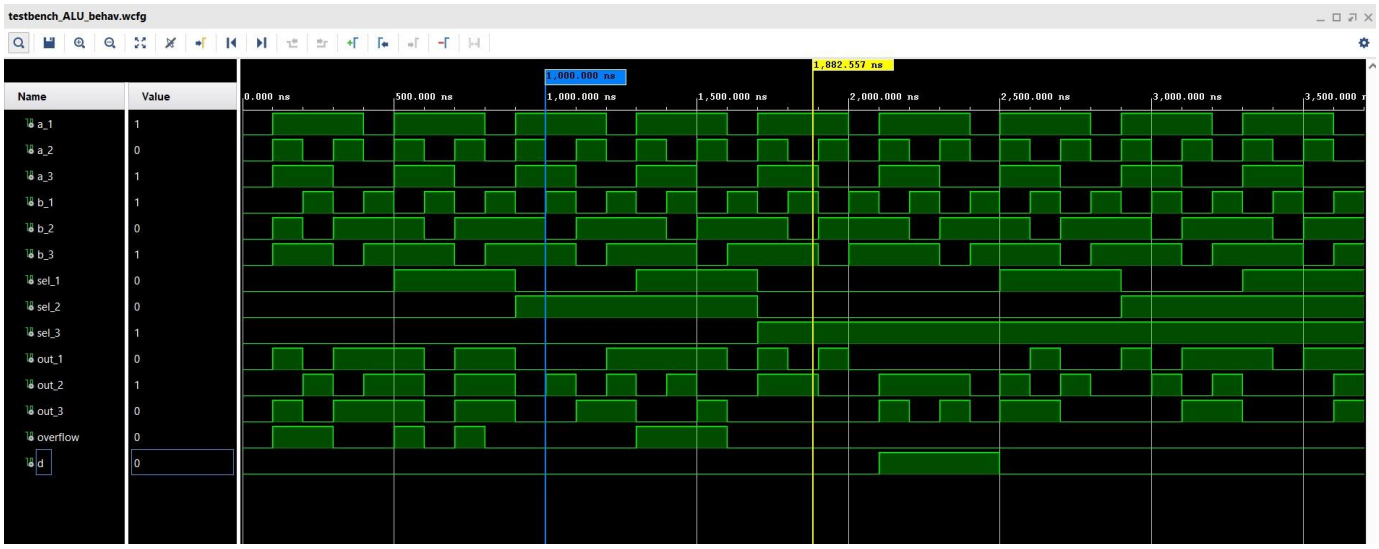
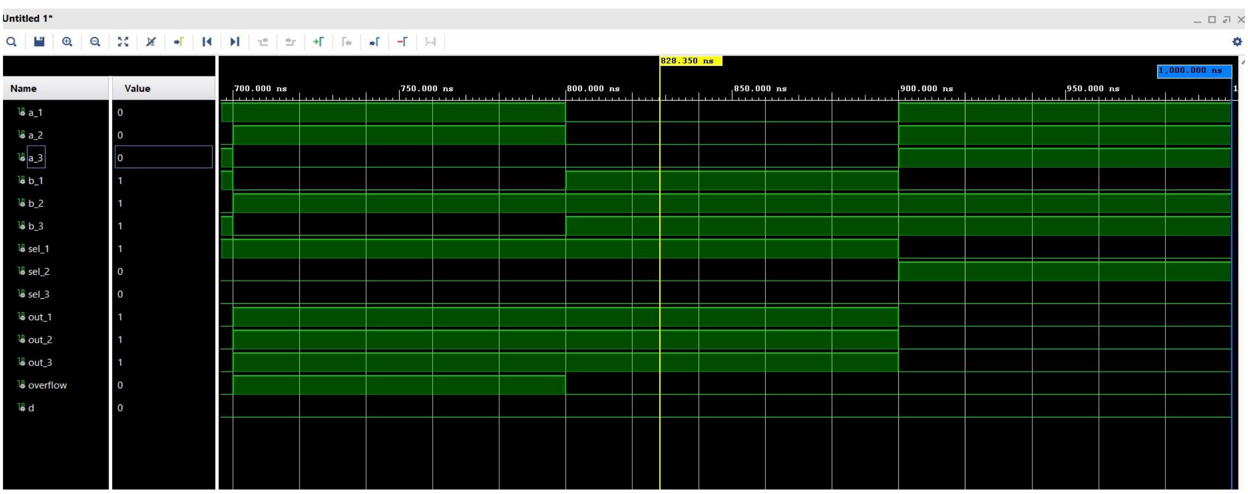


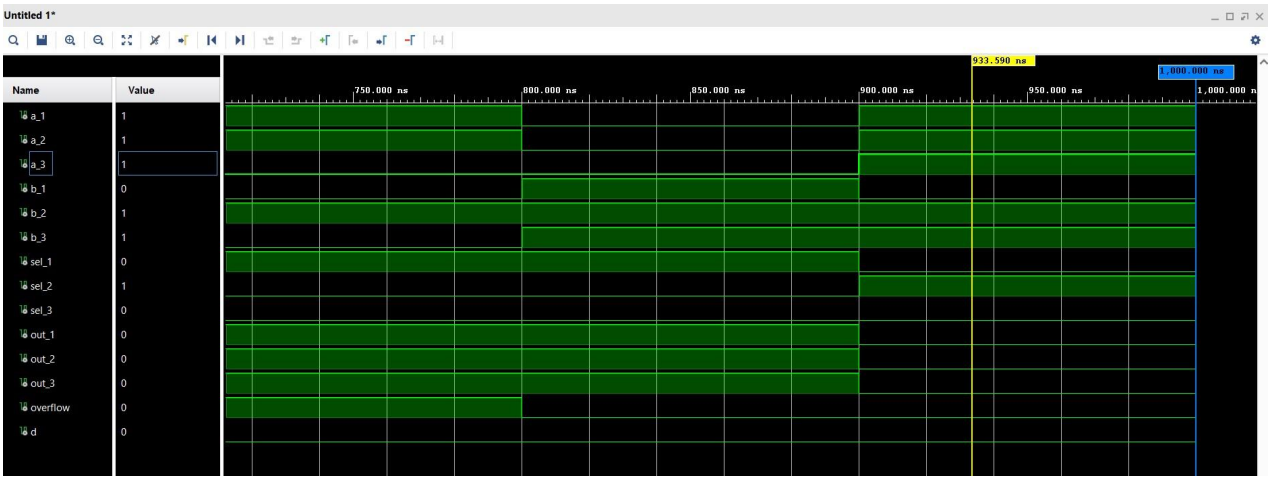
Fig.15 Test Bench

Conclusion

With this lab I learned how to create an ALU and increased my knowledge about arithmetic operations. I could have been also used vectors which could have made the task easier however it didn't came to my mind in the first place therefore I used a decoder instead. I also increased my knowledge about Port mapping.

Appendix





HALF ADDER

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


entity Halfadd is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           sum : out STD_LOGIC;
           carry : out STD_LOGIC);
end Halfadd;

architecture half of Halfadd is

begin
 sum <= x xor y;
 carry <= x and y;



end half;
```

FULL ADDER

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


entity fulladd is
    Port ( ci : in STD_LOGIC;
           xi : in STD_LOGIC;
           yi : in STD_LOGIC;
           sumin : out STD_LOGIC;
           carryin : out STD_LOGIC);
end fulladd;

architecture full of fulladd is

    component Halfadd is
        Port ( x : in STD_LOGIC;
               y : in STD_LOGIC;
               sum : out STD_LOGIC;
```

```
            carry : out STD_LOGIC);
    end component;

 signal s1,c1,c2: std_logic;

 begin
 HA1: Halfadd port map(xi,yi,s1,c1);
 HA2: Halfadd port map (ci,s1,sumin,c2);

 carryin <= c1 or c2;

 end full;
```

3-BIT ADDER

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


entity addthree is
   Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          y0 : in STD_LOGIC;
          y1 : in STD_LOGIC;
          y2 : in STD_LOGIC;
          addover : out STD_LOGIC;
          s0 : out STD_LOGIC;
          s1 : out STD_LOGIC;
          s2 : out STD_LOGIC);
end addthree;

architecture add of addthree is
 component fulladd is
 Port ( ci : in STD_LOGIC;
        xi : in STD_LOGIC;
        yi : in STD_LOGIC;
        sumin : out STD_LOGIC;
        carryin : out STD_LOGIC);

 end component;

signal c0,c1: std_logic ;
begin
FA1: fulladd port map ('0',x0,y0,s0,c0);
FA2: fulladd port map (c0,x1,y1,s1,c1);
FA3: fulladd port map (c1,x2,y2,s2,addover);

end add;
```

HALF SUBTRACTOR

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


entity substr is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          diff : out STD_LOGIC;
          borr : out STD_LOGIC);
```

```
end substr;

architecture Behavioral of substr is

begin
diff <= a xor b;
borr <= not a and b;


end Behavioral;
```

FULL SUBTRACTOR

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


entity fullsubs is
    Port ( in1 : in STD_LOGIC;
           in2 : in STD_LOGIC;
           in3 : in STD_LOGIC;
           d : out STD_LOGIC;
           b0 : out STD_LOGIC);
end fullsubs;

architecture fulsub of fullsubs is
    component substr is
        Port ( a : in STD_LOGIC;
               b : in STD_LOGIC;
               diff : out STD_LOGIC;
               borr : out STD_LOGIC);
    end component;
signal differ,borr1,borr2: std_logic;



begin
HS1:substr port map (in1,in2,differ,borr1);
HS2: substr port map (differ,in3,d,borr2);
b0 <= borr1 or borr2;



end fulsub;
```

3BIT SUBTRACTOR

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



entity substractor is
    Port ( a0 : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           z0 : in STD_LOGIC;
           z1 : in STD_LOGIC;
           z2 : in STD_LOGIC;
```

```
        d0 : out STD_LOGIC;
        d1 : out STD_LOGIC;
        d2 : out STD_LOGIC;
        bout : out STD_LOGIC);
end substractor;

architecture subs of substractor is
component fullsubs is
Port ( in1 : in STD_LOGIC;
       in2 : in STD_LOGIC;
       in3 : in STD_LOGIC;
       d : out STD_LOGIC;
       b0 : out STD_LOGIC);
end component;
signal s0,s1 : std_logic ;

begin
FS1: fullsubs port map (z0,a0,'0',d0,s0);
FS2: fullsubs port map (z1,a1,s0,d1,s1);
FS3: fullsubs port map (z2,a2,s1,d2,bout);


end subs;
```

1'S COMPLEMENT

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



entity onescomp is
   Port ( in1_1 : in STD_LOGIC;
          in1_2: in STD_LOGIC;
          in1_3: in STD_LOGIC;


          out1_1: out STD_LOGIC;
          out1_2 : out STD_LOGIC;
          out1_3: out STD_LOGIC);
end onescomp;

architecture Behavioral of onescomp is

begin
out1_1 <= not in1_1;
out1_2 <= not in1_2;
out1_3 <= not in1_3;



end Behavioral;
```

2'S COMPLEMENT

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

```
entity twoscompl is
   Port ( a1 : in STD_LOGIC;
          a2 : in STD_LOGIC;
          a3 : in STD_LOGIC;
          c : out STD_LOGIC;
          ou1 : out STD_LOGIC;
          ou2 : out STD_LOGIC;
          ou3 : out STD_LOGIC);
end twoscompl;

architecture twocomp of twoscompl is
component onescomp is
   Port ( in1_1 : in STD_LOGIC;
          in1_2: in STD_LOGIC;
          in1_3: in STD_LOGIC;


          out1_1: out STD_LOGIC;
          out1_2 : out STD_LOGIC;
          out1_3: out STD_LOGIC);
end component;
component addthree is
   Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          y0 : in STD_LOGIC;
          y1 : in STD_LOGIC;
          y2 : in STD_LOGIC;
          addover : out STD_LOGIC;
          s0 : out STD_LOGIC;
          s1 : out STD_LOGIC;
          s2 : out STD_LOGIC);
end component;
signal Ax,Ay,Az: std_logic;


begin
TC1: onescomp port map (a1,a2,a3,Ax,Ay,Az);
TC2: addthree port map (Ax,Ay,Az,'1','0','0',c,ou1,ou2,ou3);




end twocomp;
```

LOGIC SHIFT

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


entity logicshift is
   Port ( a0 : in STD_LOGIC;
          a1 : in STD_LOGIC;
          a2 : in STD_LOGIC;
          d : in STD_LOGIC;
          s0 : out STD_LOGIC;
          s1 : out STD_LOGIC;
          s2 : out STD_LOGIC);
end logicshift;
```

```
architecture logshift of logicshift is


begin
s0 <= not d and a1;
s1 <= (a0 and d) or ( a2 and not d);
s2 <= d and a1;


end logshift;
```

BITWISEAND

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


entity bitwiseand is
   Port ( input1_1 : in STD_LOGIC;
          input1_2 : in STD_LOGIC;
          input1_3 : in STD_LOGIC;
          input2_1 : in STD_LOGIC;
          input2_2 : in STD_LOGIC;
          input2_3 : in STD_LOGIC;

          out1 : out STD_LOGIC;
          out2 : out STD_LOGIC;
          out3 : out STD_LOGIC);
end bitwiseand;


architecture Behavioral of bitwiseand is


begin
out1 <= input1_1 and input2_1;
out2 <= input1_2 and input2_2;
out3 <= input1_3 and input2_3;


end Behavioral;
```

BITWISENAND

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

entity bitwisenand 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;
          o1 : out STD_LOGIC;
          o2 : out STD_LOGIC;
          o3 : out STD_LOGIC);
```

```
end bitwisenand;

architecture bitand of bitwisenand is

begin
o1 <= x1 nand y1;
o2 <= x2 nand y2;
o3 <= x3 nand y3;


end bitand;
```

BITWISEXOR

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

entity bitwisexor is
   Port ( in11 : in STD_LOGIC;
          in12 : in STD_LOGIC;
          in13 : in STD_LOGIC;
          in21 : in STD_LOGIC;
          in22 : in STD_LOGIC;
          in23 : in STD_LOGIC;
          out1 : out STD_LOGIC;
          out2 : out STD_LOGIC;
          out3 : out STD_LOGIC);
end bitwisexor;

architecture bitxor of bitwisexor is

begin
out1 <= in11 xor in21;
out2 <= in12 xor in22;
out3 <= in13 xor in23;



end bitxor;
```

DECODER

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


entity deco is
   Port ( s_1 : in STD_LOGIC;
          s_2 : in STD_LOGIC;
          s_3 : in STD_LOGIC;
          o1 : out STD_LOGIC;
          o2 : out STD_LOGIC;
          o3 : out STD_LOGIC;
          o4 : out STD_LOGIC;
          o5 : out STD_LOGIC;
          o6 : out STD_LOGIC;
          o7 : out STD_LOGIC;
          o8 : out STD_LOGIC);
end deco;

architecture dec of deco is
```

```
begin
o1 <= not s_1 and not s_2 and not s_3;
o2 <= not s_3 and not s_2 and s_1;
o3 <= not s_3 and s_2 and not s_1;
o4 <= not s_3 and s_2 and s_1;
o5 <= s_3 and not s_2 and not s_1;
o6 <= s_3 and not s_2 and s_1;
o7 <= s_3 and s_2 and not s_1;
o8 <= s_3 and s_2 and s_1;
end dec;
```

TOP MODULE

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



entity TOP is
   Port (
      --inputs--
      a_1 : in STD_LOGIC;
      a_2 : in STD_LOGIC;
      a_3 : in STD_LOGIC;
      b_1 : in STD_LOGIC;
      b_2 : in STD_LOGIC;
      b_3 : in STD_LOGIC;
      --logicshiftleftrightsignal--
      d : in STD_LOGIC;
      --output--
      out_1 : out STD_LOGIC;
      out_2 : out STD_LOGIC;
      out_3 : out STD_LOGIC;
      --overflow--
      overflow : out STD_LOGIC;
      --selectinput--
      sel_1 : in STD_LOGIC;
      sel_2 : in STD_LOGIC;
      sel_3 : in STD_LOGIC );


end TOP;

architecture topmodule of TOP is


--ADDER--
component addthree is
   Port ( x0 : in STD_LOGIC;
       x1 : in STD_LOGIC;
       x2 : in STD_LOGIC;
       y0 : in STD_LOGIC;
       y1 : in STD_LOGIC;
       y2 : in STD_LOGIC;
       addover : out STD_LOGIC;
       s0 : out STD_LOGIC;
       s1 : out STD_LOGIC;
       s2 : out STD_LOGIC);
end component;
```

```
--SUBSTRACTOR--
component substractor is
   Port ( a0 : in STD_LOGIC;
       a1 : in STD_LOGIC;
       a2 : in STD_LOGIC;
       z0 : in STD_LOGIC;
       z1 : in STD_LOGIC;
       z2 : in STD_LOGIC;
       d0 : out STD_LOGIC;
       d1 : out STD_LOGIC;
       d2 : out STD_LOGIC;
       bout : out STD_LOGIC);
end component;

--1's COMPLEMENT--
component onescomp is
   Port ( in1_1 : in STD_LOGIC;
       in1_2: in STD_LOGIC;
       in1_3: in STD_LOGIC;


       out1_1: out STD_LOGIC;
       out1_2 : out STD_LOGIC;
       out1_3: out STD_LOGIC);
end component;


--2'S COMPLEMENT--
component twoscompl is
   Port ( a1 : in STD_LOGIC;
       a2 : in STD_LOGIC;
       a3 : in STD_LOGIC;
       c : out STD_LOGIC;
       ou1 : out STD_LOGIC;
       ou2 : out STD_LOGIC;
       ou3 : out STD_LOGIC);
end component;


--LOGICAL SHIFT--
component logicshift is
   Port ( a0 : in STD_LOGIC;
       a1 : in STD_LOGIC;
       a2 : in STD_LOGIC;
       d : in STD_LOGIC;
       s0 : out STD_LOGIC;
       s1 : out STD_LOGIC;
       s2 : out STD_LOGIC);
end component;


--BITWISE AND--
component bitwiseand is
   Port ( input1_1 : in STD_LOGIC;
       input1_2 : in STD_LOGIC;
       input1_3 : in STD_LOGIC;
       input2_1 : in STD_LOGIC;
       input2_2 : in STD_LOGIC;
       input2_3 : in STD_LOGIC;
```

```
        out1 : out STD_LOGIC;
        out2 : out STD_LOGIC;
        out3 : out STD_LOGIC);
end component;


--BITWISE NAND--
component bitwisenand 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;
        o1 : out STD_LOGIC;
        o2 : out STD_LOGIC;
        o3 : out STD_LOGIC);
end component;


--BITWISE XOR--
component bitwisexor is
    Port ( in11 : in STD_LOGIC;
        in12 : in STD_LOGIC;
        in13 : in STD_LOGIC;
        in21 : in STD_LOGIC;
        in22 : in STD_LOGIC;
        in23 : in STD_LOGIC;
        out1 : out STD_LOGIC;
        out2 : out STD_LOGIC;
        out3 : out STD_LOGIC);
end component;

-- DECODER--
component deco is
    Port ( s_1 : in STD_LOGIC;
        s_2 : in STD_LOGIC;
        s_3 : in STD_LOGIC;
        o1 : out STD_LOGIC;
        o2 : out STD_LOGIC;
        o3 : out STD_LOGIC;
        o4 : out STD_LOGIC;
        o5 : out STD_LOGIC;
        o6 : out STD_LOGIC;
        o7 : out STD_LOGIC;
        o8 : out STD_LOGIC);
end component;

signal
 add1,add2,add3,overflow1,
 subs1,subs2,subs3,borrow1,
 oc1,oc2,oc3,
 tc1,tc2,tc3,overflow2,
 lsh1,lsh2,lsh3,
 ba1,ba2,ba3,
 bn1,bn2,bn3,
 bxo1,bxo2,bxo3,
 m0,m1,m2,m3,m4,m5,m6,m7 : std_logic ;
begin
```

```
F1: addthree port map(a_1,a_2,a_3,b_1,b_2,b_3,overflow1,add1,add2,add3);
F2: substractor port map (a_1,a_2,a_3,b_1,b_2,b_3,subs1,subs2,subs3,borrow1);
F3: onescomp port map(a_1,a_2,a_3,oc1,oc2,oc3);
F4: twoscompl port map(a_1,a_2,a_3,overflow2,tc1,tc2,tc3);
F5: logicshift port map(a_1,a_2,a_3,d,lsh1,lsh2,lsh3);
F6: bitwiseand port map(a_1,a_2,a_3,b_1,b_2,b_3,ba1,ba2,ba3);
F7: bitwisenand port map (a_1,a_2,a_3,b_1,b_2,b_3,bn1,bn2,bn3);
F8: bitwisexor port map (a_1,a_2,a_3,b_1,b_2,b_3,bxo1,bxo2,bxo3);

M: deco port map (sel_1,sel_2,sel_3,m0,m1,m2,m3,m4,m5,m6,m7);


process (m0,m1,m2,m3,m4,m5,m6,m7,add1,add2,add3,overflow1,borrow1,
subs1,subs2,subs3,borrow1,oc1,oc2,oc3,overflow2,tc1,tc2,tc3,lsh1,lsh2,lsh3,ba1,ba2,ba3,bn1,bn2,
bn3,bxo1,bxo2,bxo3)

begin

   if m0='1' then
   overflow <= overflow1;
   out_1 <= add1;
   out_2 <= add2;
   out_3 <= add3;

   elsif m1= '1' then
   overflow <= borrow1;
   out_1 <= subs1;
   out_2 <= subs2;
   out_3 <= subs3;

   elsif m2 = '1' then
   overflow <= '0';
   out_1 <= oc1;
   out_2 <= oc2;
   out_3 <= oc3;

   elsif m3= '1' then
   overflow <= overflow2;
   out_1 <= tc1;
   out_2 <= tc2;
   out_3 <= tc3;

   elsif m4= '1' then
   overflow <= '0';
   out_1 <= lsh1;
   out_2 <= lsh2;
   out_3 <= lsh3;

   elsif m5= '1' then
   overflow <= '0';
   out_1 <= ba1;
   out_2 <= ba2;
   out_3 <= ba3;


   elsif m6= '1' then
   overflow <= '0';
   out_1 <= bn1;
   out_2 <= bn2;
```

```
   out_3 <= bn3;


   elsif m7= '1' then
   overflow <= '0';
   out_1 <= bxo1;
   out_2 <= bxo2;
   out_3 <= bxo3;


   end if;


end process;

end topmodule;
```

TEST BENCH

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



entity testbench_ALU is

end testbench_ALU;

architecture tb of testbench_ALU is
component TOP is
   Port (
      --inputs--
      a_1 : in STD_LOGIC;
      a_2 : in STD_LOGIC;
      a_3 : in STD_LOGIC;
      b_1 : in STD_LOGIC;
      b_2 : in STD_LOGIC;
      b_3 : in STD_LOGIC;
      --logicshiftleftrightsignal--
      d : in STD_LOGIC;
      --output--
      out_1 : out STD_LOGIC;
      out_2 : out STD_LOGIC;
      out_3 : out STD_LOGIC;
      --overflow--
      overflow : out STD_LOGIC;
      --selectinput--
      sel_1 : in STD_LOGIC;
      sel_2 : in STD_LOGIC;
      sel_3 : in STD_LOGIC );


end component;
signal a_1, a_2, a_3,b_1,b_2,b_3 : std_logic;
signal sel_1,sel_2,sel_3 : std_logic;
signal out_1,out_2,out_3 :std_logic;
signal overflow :std_logic;
signal d : std_logic;

begin
```

```
def:  TOP port map
(a_1 => a_1, a_2 => a_2, a_3 => a_3, b_1 => b_1, b_2 => b_2,b_3 => b_3,
sel_1 => sel_1, sel_2 => sel_2, sel_3 => sel_3,
out_1 => out_1, out_2 => out_2, out_3 => out_3,
overflow => overflow, d => d);

Testbench: process
begin

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '0';
b_2 <= '0';
b_3 <= '0';
d <= '0';
wait for 100ns;


--addition--
sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;



sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;




sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
```

```
wait for 100ns;


sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

--substraction--

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '0';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;
```

```
-- 1'scomplement--

sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;


sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;


sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;


sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

--2's complement--

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
```

```
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '0';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

--logic shift--

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
d <= '0';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
d <= '0';
wait for 100ns;
```

```
sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
d <= '0';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
d <= '0';
wait for 100ns;



sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
d <= '1';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
d <= '1';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
```

```
d <= '1';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
d <= '1';
wait for 100ns;
```

```
--bitwiseand--

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '0';
sel_3 <= '1';
a_1 <= '0';
a_2 <= '0';
```

```
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

--bitwisenand--

sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;

sel_1 <= '0';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

--bitwisexor--

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '1';
```

```
b_1 <= '0';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '0';
a_3 <= '1';
b_1 <= '1';
b_2 <= '0';
b_3 <= '1';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '1';
a_2 <= '1';
a_3 <= '0';
b_1 <= '0';
b_2 <= '1';
b_3 <= '0';
wait for 100ns;

sel_1 <= '1';
sel_2 <= '1';
sel_3 <= '1';
a_1 <= '0';
a_2 <= '0';
a_3 <= '0';
b_1 <= '1';
b_2 <= '1';
b_3 <= '1';
wait for 100ns;

end process;

end tb;
```