

Report to Lab 4: Arithmetic Logic Unit

Arda Özkut
22101727
Section 2

Physics Department, Ihsan Doğramacı Bilkent University
(Dated: November 8, 2022)

I. PURPOSE OF THE LAB ASSIGNMENT

The purpose of this assignment was to design an arithmetic logic unit in VHDL that performs 8 operations. It was expected to control the signals using the switches on the card and light the LEDs on the board with the desired outputs.

II. DESIGN SPECIFICATIONS

I decided to write a 4-bit arithmetic logic unit with one overflow signal.

There are 8 inputs that correspond to two 4-bit integers and 6 inputs that select the operation. 3 of them select the operation type and two of them are the direction specifiers for the arithmetic and logic shift operations.

The pin configurations can be seen in [FIG. 1]

The ALU is constructed in a hierarchical design. The RTL schematics for all the sub-modules can be found in Appendix B. Below the Top Module that combines the operations there are following sub-modules (\rightarrow represents one lower in hierarchy):

- Addition[FIG. 8]
- \rightarrow Adder[FIG. 9]
- $\rightarrow\rightarrow$ Half Adder [FIG. 10]
For the addition I first designed the half adder and full subtractor respectively. Then used these to construct full adder and subtractor.
- Bit-Wise AND [FIG. 11]
I ANDed each signals of same significance.
- Increment [FIG. 12]
I used the adder module I designed and instead of the second integer I connected the inputs to ground or V_{dd} to add "0001" to the first input.
- Logical Shift [FIG. 13]
I also designed an overflow signal and a direction select signal for the shift functions.
- Subtraction [FIG. 14]
- \rightarrow Full Subtractor [FIG. 15]
- $\rightarrow\rightarrow$ Half Subtractor [FIG. 16]
- Arithmetic Shift [FIG. 8]
- Ones Compliment [FIG. 18]
- Twos Compliment [FIG. 19]
I used the Ones Compliment, the Increment module and specified an overflow signal to implement the Twos compliment function.

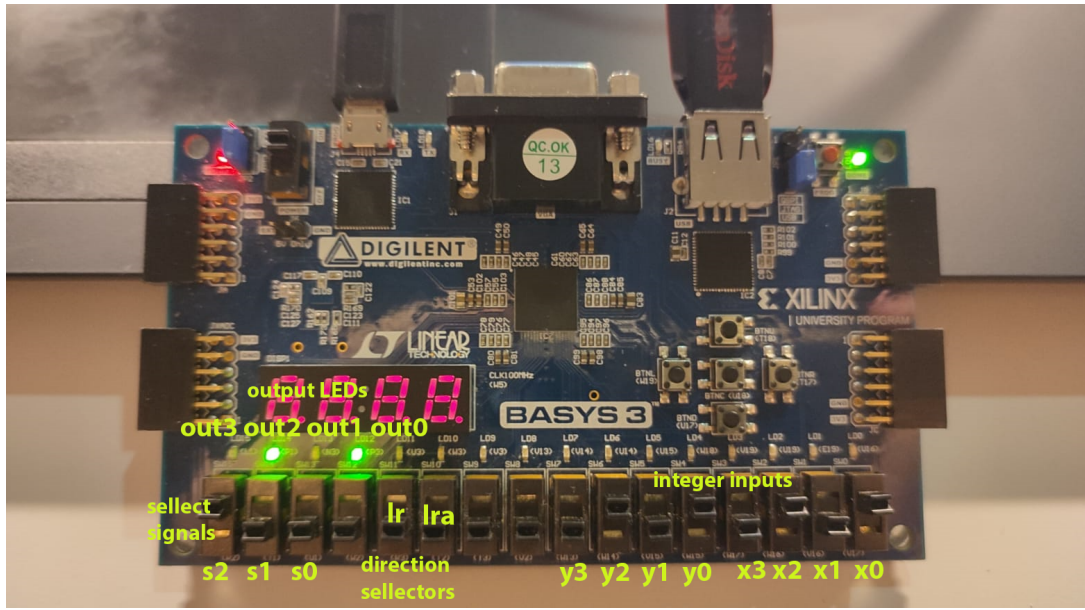


FIG. 1: Pin and LED configuration

III. RESULTS

As can be seen from [FIG. 2] the design is successfully implemented. Test Bench results, examples from the Basys 3 board and some of the signals from the test bench can be seen in the Appendix B. The overflow indicator signal and the operations work well and the waveform generated by the Vivado Simulator gives the expected results. Some of the results of both the simulation and examples on the card itself are included in the Appendix part B.

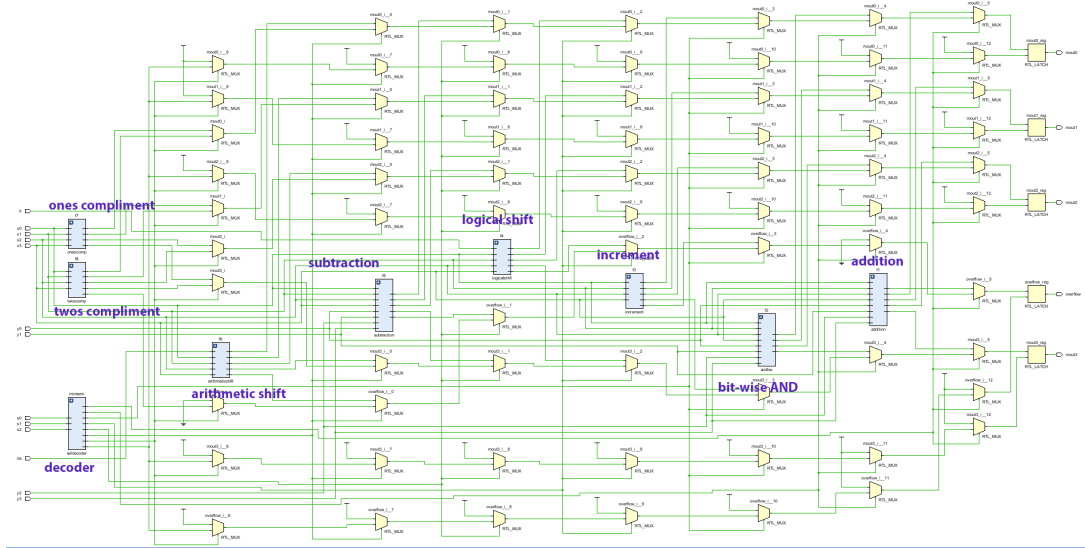


FIG. 2: RTL Schematic

IV. CONCLUSION

This lab the task was to implement an ALU that has 8 operations and implement them on Basys 3. I have learned and enhanced my understanding of the operations, the interface of Vivado Software and my understanding of how to code VHDL. I successfully implemented the ALU on the Basys 3 FPGA card. The design has no flaws as far as my tests found, however the design could have been simpler if I would have used vectors instead of using individual signals to choose between operations.

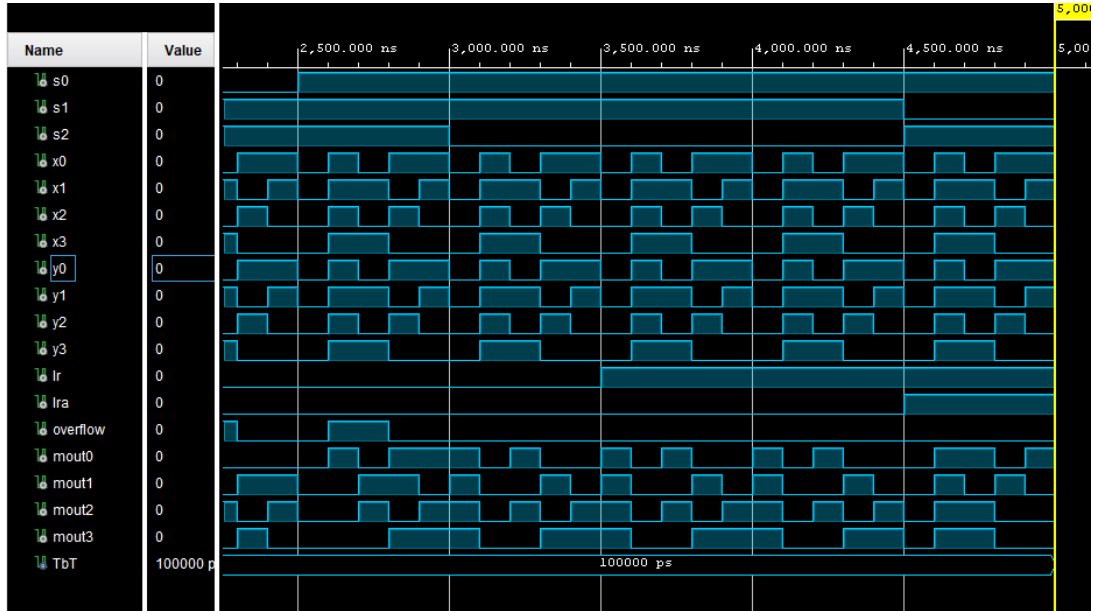


FIG. 3: Test Bench Results

V. APPENDIX

A. Code

1. Top Level (toplvl.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity toplvl is
    port(

        --select
        s0 : in std_logic;
        s1 : in std_logic;
        s2 : in std_logic;
        --inputs
        x0 : in STD_LOGIC;
        x1 : in STD_LOGIC;
        x2 : in STD_LOGIC;
        x3 : in STD_LOGIC;
        y0 : in STD_LOGIC;
        y1 : in STD_LOGIC;
        y2 : in STD_LOGIC;
        y3 : in STD_LOGIC;
        --lr for shiftfs
        lr : in std_logic;
        lra : in std_logic;
        --out
        overflow : out std_logic;
        --masterout
        mout0 : out std_logic;
        mout1 : out std_logic;
        mout2 : out std_logic;
        mout3 : out std_logic
    );
end toplvl;

architecture totalarch of toplvl is
    component seldecoder is
        Port ( s0 : in STD_LOGIC;
              s1 : in STD_LOGIC;
              s2 : in STD_LOGIC;
              m1 : out STD_LOGIC;
              m2 : out STD_LOGIC;
              m3 : out STD_LOGIC;
              m4 : out STD_LOGIC;
              m5 : out STD_LOGIC;
              m6 : out STD_LOGIC;
              m7 : out STD_LOGIC;
              m8 : out STD_LOGIC);
    end component;
    component increment is
        Port ( x0 : in STD_LOGIC;
              x1 : in STD_LOGIC;
              x2 : in STD_LOGIC;

```

```

        x3 : in STD_LOGIC;
        oi0 : out STD_LOGIC;
        oi1 : out STD_LOGIC;
        oi2 : out STD_LOGIC;
        oi3 : out STD_LOGIC;
        overflowi : out std_logic);
end component;
component addition is
    port(x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          y0 : in STD_LOGIC;
          y1 : in STD_LOGIC;
          y2 : in STD_LOGIC;
          y3 : in STD_LOGIC;
          o0 : out STD_LOGIC;
          o1 : out STD_LOGIC;
          o2 : out STD_LOGIC;
          o3 : out STD_LOGIC;
          overflow : out std_logic);
end component;
component andbw is
    Port ( x0 : in STD_LOGIC;
           x1 : in STD_LOGIC;
           x2 : in STD_LOGIC;
           x3 : in STD_LOGIC;
           y0 : in STD_LOGIC;
           y1 : in STD_LOGIC;
           y2 : in STD_LOGIC;
           y3 : in STD_LOGIC;
           oa0 : out STD_LOGIC;
           oa1 : out STD_LOGIC;
           oa2 : out STD_LOGIC;
           oa3 : out STD_LOGIC);
end component;
component logicalshift is
    port( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          lr : in STD_LOGIC;

          ol0 : inout STD_LOGIC;
          ol1 : inout STD_LOGIC;
          ol2 : inout STD_LOGIC;
          ol3 : inout STD_LOGIC;
          overflowl : inout std_logic
    );
end component;
component subtraction is
    Port (x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;

```

```

        y0 : in STD_LOGIC;
        y1 : in STD_LOGIC;
        y2 : in STD_LOGIC;
        y3 : in STD_LOGIC;
        os0 : out STD_LOGIC;
        os1 : out STD_LOGIC;
        os2 : out STD_LOGIC;
        os3 : out STD_LOGIC;
        overflows : out std_logic );
end component;
component arithmeticshift is
    Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          lra : in STD_LOGIC;
          oa0 : inout STD_LOGIC;
          oa1 : inout STD_LOGIC;
          oa2 : inout STD_LOGIC;
          oa3 : inout STD_LOGIC;
          overflowa : inout std_logic
        );
end component;
component onescomp is
    Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          oo0 : out STD_LOGIC;
          oo1 : out STD_LOGIC;
          oo2 : out STD_LOGIC;
          oo3 : out STD_LOGIC);
end component;
component twoscomp is
    Port (x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          ot0 : out STD_LOGIC;
          ot1 : out STD_LOGIC;
          ot2 : out STD_LOGIC;
          ot3 : out STD_LOGIC;
          overflow : out std_logic);
end component;

signal w00,w01,w02,w03,w10,w11,w12,w13,w20,w21,w22,w23,v0,v1,v2,v3
,m1,m2,m3,m4,m5,m6,m7,m8,overflowsig1,overflowsig2,overflowsig3,p0
,p1,p2,p3,overflowsig4,va0,va1,va2,va3,overflowsig5,g0,g1,g2,g3,j0
,j1,j2,j3,overflowsig6 : std_logic ;
begin

f1: addition port map(x0,x1,x2,x3,y0,y1,y2,y3,w00,w01,w02,w03,overflowsig1);
f2: andbw port map(x0,x1,x2,x3,y0,y1,y2,y3,w20,w21,w22,w23);
f3: increment port map(x0,x1,x2,x3,w10,w11,w12,w13,overflowsig2);

```

```

f4: logicalshift port map(x0,x1,x2,x3,lr,v0,v1,v2,v3,overflowsig3);
f5: subtraction port map(x0,x1,x2,x3,y0,y1,y2,y3,p0,p1,p2,p3,overflowsig4);
f6: arithmeticshift port map(x0,x1,x2,x3,lra,va0,va1,va2,va3,overflowsig5);
f7: onescomp port map(x0,x1,x2,x3,g0,g1,g2,g3);
f8: twoscomp port map(x0,x1,x2,x3,j0,j1,j2,j3,overflowsig6);

```

```

minterm: seldecoder port map(s0,s1,s2,m1,m2,m3,m4,m5,m6,m7,m8);

```

```

process(m1,m2,m3,m4,m5,m6,m7,m8,w00,w01,w02,w03,
overflowsig1,w20,w21,w22,w23,w10,w11,w12,w13,overflowsig2,v0
,v1,v2,v3,overflowsig3,va0,va1,va2,va3,overflowsig5,p0,p1,p2,p3
,overflowsig4,g0,g1,g2,g3,j0,j1,j2,j3,overflowsig6
)

```

```

begin

```

```

    if m1 = '1' then
        mout0 <= w00;
        mout1 <= w01;
        mout2 <= w02;
        mout3 <= w03;
        overflow <= overflowsig1;
    elsif m2 = '1' then
        mout0 <= w20;
        mout1 <= w21;
        mout2 <= w22;
        mout3 <= w23;
        overflow <= '0';
    elsif m3 = '1' then
        mout0 <= w10;
        mout1 <= w11;
        mout2 <= w12;
        mout3 <= w13;
        overflow <= overflowsig2;
    elsif m4 = '1' then
        mout0 <= v0;
        mout1 <= v1;
        mout2 <= v2;
        mout3 <= v3;
        overflow <= overflowsig3;
    elsif m5 = '1' then
        mout0 <= p0;
        mout1 <= p1;
        mout2 <= p2;
        mout3 <= p3;
        overflow <= overflowsig4;
    elsif m6 = '1' then
        mout0 <= va0;
        mout1 <= va1;
        mout2 <= va2;
        mout3 <= va3;
        overflow <= overflowsig5;
    elsif m7 = '1' then
        mout0 <= g0;
        mout1 <= g1;
        mout2 <= g2;
        mout3 <= g3;
        overflow <= '0';
    elsif m8 = '1' then
        mout0 <= j0;

```

```

        mout1 <= j1;
        mout2 <= j2;
        mout3 <= j3;
        overflow <= overflowsig6;
    end if;

end process;

end totalarch;

```

2. Addition (addition.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity addition is
    port(x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          y0 : in STD_LOGIC;
          y1 : in STD_LOGIC;
          y2 : in STD_LOGIC;
          y3 : in STD_LOGIC;
          o0 : out STD_LOGIC;
          o1 : out STD_LOGIC;
          o2 : out STD_LOGIC;
          o3 : out STD_LOGIC;
          overflow : out std_logic);
end addition;

architecture add of addition is
    signal c1,c2,c3 : std_logic;
    component oneadder is
        port(ci : in STD_LOGIC;
              x0 : in STD_LOGIC;
              x1 : in STD_LOGIC;
              cf : out STD_LOGIC;
              s : out STD_LOGIC);
    end component;

begin
    A1: oneadder port map('0',x0,y0,c1,o0);
    A2: oneadder port map(c1,x1,y1,c2,o1);
    A3: oneadder port map(c2,x2,y2,c3,o2);
    A4: oneadder port map(c3,x3,y3,overflow,o3);
end add;

```

3. Full Adder (oneadder.vhd)

```

library ieee;
use IEEE.STD_LOGIC_1164.ALL;

```



```

entity oneadder is
  Port (
    -- innitial carry
    ci : in STD_LOGIC;
    --two inputs
    x0 : in STD_LOGIC;
    x1 : in STD_LOGIC;
    --final carry and sum
    cf : out STD_LOGIC;
    s : out STD_LOGIC);
end oneadder;

architecture oneadd of oneadder is
  signal w0,w1,w2 : std_logic;
  component hadder is
    port( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          carry : out STD_LOGIC;
          sum : out STD_LOGIC);
    end component;
begin
  U1: hadder port map(x0,x1,w1,w0);
  U2: hadder port map(ci,w0,w2,s);

  cf <= w2 or w1;

end oneadd;

```

4. Half Adder (hadder.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity hadder is
  Port (
    x0 : in STD_LOGIC;
    x1 : in STD_LOGIC;
    carry : out STD_LOGIC;
    sum : out STD_LOGIC);
end hadder;

architecture hadd of hadder is
begin
  sum <= x0 XOR x1;
  carry <= x0 and x1;
end hadd;

```

5. Bit Wise AND (andbw.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity andbw is

```

```

Port ( x0 : in STD_LOGIC;
       x1 : in STD_LOGIC;
       x2 : in STD_LOGIC;
       x3 : in STD_LOGIC;
       y0 : in STD_LOGIC;
       y1 : in STD_LOGIC;
       y2 : in STD_LOGIC;
       y3 : in STD_LOGIC;
       oa0 : out STD_LOGIC;
       oa1 : out STD_LOGIC;
       oa2 : out STD_LOGIC;
       oa3 : out STD_LOGIC);
end andbw;

architecture archand of andbw is
begin
    oa0 <= x0 and y0;
    oa1 <= x1 and y1;
    oa2 <= x2 and y2;
    oa3 <= x3 and y3;

end archand;

\subsubsection{Increment (increment.vhd)}
\begin{lstlisting}[style = vhdl]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity increment is
Port (
    x0 : in STD_LOGIC;
    x1 : in STD_LOGIC;
    x2 : in STD_LOGIC;
    x3 : in STD_LOGIC;
    oi0 : out STD_LOGIC;
    oi1 : out STD_LOGIC;
    oi2 : out STD_LOGIC;
    oi3 : out STD_LOGIC;
    overflowi : out std_logic);
end increment;

architecture inc of increment is
    signal c1,c2,c3 : std_logic;
    component oneadder is
        port(ci : in STD_LOGIC;
             x0 : in STD_LOGIC;
             x1 : in STD_LOGIC;
             cf : out STD_LOGIC;
             s : out STD_LOGIC);
    end component;
begin
    A1: oneadder port map('0',x0,'1',c1,oi0);
    A2: oneadder port map(c1,x1,'0',c2,oi1);
    A3: oneadder port map(c2,x2,'0',c3,oi2);
    A4: oneadder port map(c3,x3,'0',overflowi,oi3);

```

```
end inc;
```

6. Logical Shift (logicalshift.vhd)

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

entity logicalshift is
    Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          lr : in STD_LOGIC;
          ol0 : inout STD_LOGIC;
          ol1 : inout STD_LOGIC;
          ol2 : inout STD_LOGIC;
          ol3 : inout STD_LOGIC;
          overflowl : inout std_logic
        );

end logicalshift;

architecture logshift of logicalshift is
    signal out1, out2, out3, out0 : std_logic;
begin
    process(x0,x1,x2,x3,lr,ol0,ol1,ol2,ol3,out1,out2,out3,out0,overflowl)
    begin
        -- x0 is the least significant bit
        if lr = '1' then
            ol0 <= x1;
            ol1 <= x2;
            ol2 <= x3;
            ol3 <= '0';
            overflowl <= '0';
        else
            overflowl <= x3;
            ol0 <= '0';
            ol1 <= x0;
            ol2 <= x1;
            ol3 <= x2;
        end if;
    end process;

end logshift;
```

7. Subtraction (subtraction.vhd)

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

entity subtraction is
    Port (x0 : in STD_LOGIC;
```

```

    x1 : in STD_LOGIC;
    x2 : in STD_LOGIC;
    x3 : in STD_LOGIC;
    y0 : in STD_LOGIC;
    y1 : in STD_LOGIC;
    y2 : in STD_LOGIC;
    y3 : in STD_LOGIC;
    os0 : out STD_LOGIC;
    os1 : out STD_LOGIC;
    os2 : out STD_LOGIC;
    os3 : out STD_LOGIC;
    overflows : out std_logic );
end subtraction;

architecture sub of subtraction is
    signal cs1,cs2,cs3,cond : std_logic;
    component fullsubtractor is
        Port ( A : in STD_LOGIC;
              B : in STD_LOGIC;
              Bin : in STD_LOGIC;
              D : out STD_LOGIC;
              Bout : out STD_LOGIC);
    end component;

begin
    S1: fullsubtractor port map(x0,y0,'0',os0,cs1);
    S2: fullsubtractor port map(x1,y1,cs1,os1,cs2);
    S3: fullsubtractor port map(x2,y2,cs2,os2,cs3);
    S4: fullsubtractor port map(x3,y3,cs3,os3,cond);
    overflows <= x3 and y3;

end sub;

```

8. Full Subtractor (fullsubtractor.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fullsubtractor is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          Bin : in STD_LOGIC;
          D : out STD_LOGIC;
          Bout : out STD_LOGIC);
end fullsubtractor;

architecture Behavioral of fullsubtractor is
    component halfsubtractor is
        Port ( a0 : in STD_LOGIC;
              b0 : in STD_LOGIC;
              D : out STD_LOGIC;
              B : out STD_LOGIC);
    end component;
    signal h1,h2,h3 : std_logic;

```

```

begin
    HS1 : halfsubtractor port map(A,B,h1,h2);
    HS2 : halfsubtractor port map(h1,Bin,D,h3);
    Bout <= h3 or h2;
end Behavioral;

```

9. Half Subtractor (halfsubtractor.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity halfsubtractor is
    Port ( a0 : in STD_LOGIC;
           b0 : in STD_LOGIC;
           D : out STD_LOGIC;
           B : out STD_LOGIC);
end halfsubtractor;

architecture halfsub of halfsubtractor is

begin
    D <= a0 xor b0;
    B <= (not a0) and b0;

end halfsub;

```

10. Arithmetic Shift (arithmeticshift.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity arithmeticshift is
    Port ( x0 : in STD_LOGIC;
           x1 : in STD_LOGIC;
           x2 : in STD_LOGIC;
           x3 : in STD_LOGIC;
           lra : in STD_LOGIC;
           oa0 : inout STD_LOGIC;
           oa1 : inout STD_LOGIC;
           oa2 : inout STD_LOGIC;
           oa3 : inout STD_LOGIC;
           overflowa : inout std_logic
    );
end arithmeticshift;

architecture arithm of arithmeticshift is

begin
    process(x0,x1,x2,x3,lra,oa0,oa1,oa2,oa3,overflowa)
    begin
        if lra = '1' then
            oa0 <= x1;
            oa1 <= x2;

```

```

        oa2 <= x3;
        oa3 <= x3;
        overflowa <= '0';
    else
        overflowa <= x3;
        oa0 <= '0';
        oa1 <= x0;
        oa2 <= x1;
        oa3 <= x2;
    end if;
end process;

end arithm;

```

11. Ones Complement (onescomp.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity onescomp is
    Port ( x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          oo0 : out STD_LOGIC;
          oo1 : out STD_LOGIC;
          oo2 : out STD_LOGIC;
          oo3 : out STD_LOGIC);
end onescomp;

architecture ones of onescomp is

begin
    oo0 <= not x0;
    oo1 <= not x1;
    oo2 <= not x2;
    oo3 <= not x3;

end ones;

```

12. Twos Complement (twoscomp.vhd)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity twoscomp is
    Port (x0 : in STD_LOGIC;
          x1 : in STD_LOGIC;
          x2 : in STD_LOGIC;
          x3 : in STD_LOGIC;
          ot0 : out STD_LOGIC;
          ot1 : out STD_LOGIC;
          ot2 : out STD_LOGIC;

```

```

        ot3 : out STD_LOGIC;

        overflow : out std_logic);
end twoscomp;

architecture twos of twoscomp is
    component onescomp is
        Port ( x0 : in STD_LOGIC;
              x1 : in STD_LOGIC;
              x2 : in STD_LOGIC;
              x3 : in STD_LOGIC;
              oo0 : out STD_LOGIC;
              oo1 : out STD_LOGIC;
              oo2 : out STD_LOGIC;
              oo3 : out STD_LOGIC);
    end component;
    component increment is
        Port ( x0 : in STD_LOGIC;
              x1 : in STD_LOGIC;
              x2 : in STD_LOGIC;
              x3 : in STD_LOGIC;
              oi0 : out STD_LOGIC;
              oi1 : out STD_LOGIC;
              oi2 : out STD_LOGIC;
              oi3 : out STD_LOGIC;
              overflowi : out std_logic);
    end component;
    signal s0,s1,s2,s3,l1,l2,l3,l4 : std_logic;
begin
    O1: onescomp port map(x0,x1,x2,x3,s0,s1,s2,s3);
    I1: increment port map(s0,s1,s2,s3,l1,l2,l3,l4);
    ot0 <= l1;
    ot1 <= l2;
    ot2 <= l3;
    ot3 <= l4;
    overflow <= not l4 and (x0 or x1 or x2 or x3) ;
end twos;

```

13. Decoder

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

entity seldecoder is
    Port ( s0 : in STD_LOGIC;
          s1 : in STD_LOGIC;
          s2 : in STD_LOGIC;
          m1 : out STD_LOGIC;
          m2 : out STD_LOGIC;
          m3 : out STD_LOGIC;
          m4 : out STD_LOGIC;
          m5 : out STD_LOGIC;
          m6 : out STD_LOGIC;
          m7 : out STD_LOGIC;

```

```

        m8 : out STD_LOGIC
    );
end seldecoder;

architecture decode of seldecoder is

begin
m1 <= not s0 and not s1 and not s2;
m2 <= not s0 and not s1 and s2;
m3 <= not s0 and s1 and not s2;
m4 <= not s0 and s1 and s2;
m5 <= s0 and not s1 and not s2;
m6 <= s0 and not s1 and s2;
m7 <= s0 and s1 and not s2;
m8 <= s0 and s1 and s2;

end decode;

```

14. Test Bench

```

-----
-- Company:
-- Engineer:
--
-- Create Date: 11/06/2022 03:03:25 PM
-- Design Name:
-- Module Name: test_b - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----

```

```

library IEEE;
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 test_b is
    -- Port ( );
end test_b;

```


architecture Behavioral **of** test_b **is**

```

component toplvl is
  port(
    --select
    s0 : in std_logic;
    s1 : in std_logic;
    s2 : in std_logic;
    --inputs
    x0 : in STD_LOGIC;
    x1 : in STD_LOGIC;
    x2 : in STD_LOGIC;
    x3 : in STD_LOGIC;
    y0 : in STD_LOGIC;
    y1 : in STD_LOGIC;
    y2 : in STD_LOGIC;
    y3 : in STD_LOGIC;
    --lr for shifts
    lr : in std_logic;
    lra : in std_logic;
    --out
    overflow : out std_logic;
    --masterout
    mout0 : out std_logic;
    mout1 : out std_logic;
    mout2 : out std_logic;
    mout3 : out std_logic
  );
end component;

signal s0,s1,s2,x0,x1,x2,x3,y0,y1,y2,y3,lr,lra : std_logic;

signal overflow : std_logic;
signal mout0    : std_logic;
signal mout1    : std_logic;
signal mout2    : std_logic;
signal mout3    : std_logic;

constant TbT : time := 100 ns;
begin

  DUT: toplvl port map(
    s0      => s0,
    s1      => s1,
    s2      => s2,
    x0      => x0,
    x1      => x1,
    x2      => x2,
    x3      => x3,
    y0      => y0,
    y1      => y1,
    y2      => y2,
    y3      => y3,
    lr      => lr,
    lra     => lra,
    overflow => overflow,

```

```

        mout0    => mout0,
        mout1    => mout1,
        mout2    => mout2,
        mout3    => mout3);

```

```
test_b: process
```

```
begin
```

```

    s0 <= '0';
    s1 <= '0';
    s2 <= '0';

```

```

    x0 <= '0';
    x1 <= '0';
    x2 <= '0';
    x3 <= '0';
    y0 <= '0';
    y1 <= '0';
    y2 <= '0';
    y3 <= '0';

```

```

    lr <= '0';
    lra <= '0';

```

```
wait for 100 ns;
```

```
---000
```

```

    s0 <= '0';
    s1 <= '0';
    s2 <= '0';

```

```

    x0 <= '1';
    x1 <= '1';
    x2 <= '1';
    x3 <= '1';
    y0 <= '1';
    y1 <= '1';
    y2 <= '1';
    y3 <= '1';

```

```
wait for 100 ns;
```

```

    x0 <= '0';
    x1 <= '1';
    x2 <= '0';
    x3 <= '1';
    y0 <= '0';
    y1 <= '1';
    y2 <= '0';
    y3 <= '1';

```

```
wait for 100 ns;
```

```

    x0 <= '1';
    x1 <= '0';
    x2 <= '1';
    x3 <= '0';
    y0 <= '1';
    y1 <= '0';
    y2 <= '1';

```

```
y3 <= '0';
wait for 100 ns;
```

```
x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;
```

```
---001
```

```
s0 <= '1';
s1 <= '0';
s2 <= '0';
```

```
x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;
```

```
x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;
```

```
x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;
```

```
x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
```

```

y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';

wait for 100 ns;
---010
s0 <= '0';
s1 <= '1';
s2 <= '0';

x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

```

```
x0 <= '1';  
x1 <= '1';  
x2 <= '0';  
x3 <= '0';  
y0 <= '1';  
y1 <= '1';  
y2 <= '0';  
y3 <= '0';  
wait for 100 ns;
```

```
---100
```

```
s0 <= '0';  
s1 <= '0';  
s2 <= '1';
```

```
x0 <= '0';  
x1 <= '0';  
x2 <= '0';  
x3 <= '0';  
y0 <= '0';  
y1 <= '0';  
y2 <= '0';  
y3 <= '0';  
wait for 100 ns;
```

```
x0 <= '1';  
x1 <= '1';  
x2 <= '1';  
x3 <= '1';  
y0 <= '1';  
y1 <= '1';  
y2 <= '1';  
y3 <= '1';  
wait for 100 ns;
```

```
x0 <= '0';  
x1 <= '1';  
x2 <= '0';  
x3 <= '1';  
y0 <= '0';  
y1 <= '1';  
y2 <= '0';  
y3 <= '1';  
wait for 100 ns;
```

```
x0 <= '1';  
x1 <= '0';  
x2 <= '1';  
x3 <= '0';  
y0 <= '1';  
y1 <= '0';  
y2 <= '1';  
y3 <= '0';  
wait for 100 ns;
```

```
x0 <= '1';  
x1 <= '1';  
x2 <= '0';
```

```

x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

---110

```

```

s0 <= '0';
s1 <= '1';
s2 <= '1';

```

```

x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';

```

```

y2 <= '0';
y3 <= '0';
wait for 100 ns;
---111
s0 <= '1';
s1 <= '1';
s2 <= '1';

x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;
---011

```

```

--left
s0 <= '1';
s1 <= '1';
s2 <= '0';

lr <= '0';
x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;
--right
s0 <= '1';
s1 <= '1';

```



```

s2 <= '0';
lr <= '1';

x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;
--101
--left
lra <= '0';
x0 <= '0';
x1 <= '0';
x2 <= '0';

```

```

x3 <= '0';
y0 <= '0';
y1 <= '0';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

--right
lra <= '1';
s0 <= '1';
s1 <= '0';
s2 <= '1';

```

```

x0 <= '0';
x1 <= '0';
x2 <= '0';
x3 <= '0';
y0 <= '0';
y1 <= '0';

```

```

y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '1';
x3 <= '1';
y0 <= '1';
y1 <= '1';
y2 <= '1';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '0';
x1 <= '1';
x2 <= '0';
x3 <= '1';
y0 <= '0';
y1 <= '1';
y2 <= '0';
y3 <= '1';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '0';
x2 <= '1';
x3 <= '0';
y0 <= '1';
y1 <= '0';
y2 <= '1';
y3 <= '0';
wait for 100 ns;

```

```

x0 <= '1';
x1 <= '1';
x2 <= '0';
x3 <= '0';
y0 <= '1';
y1 <= '1';
y2 <= '0';
y3 <= '0';
wait for 100 ns;

```

```

end process;

```

```

end Behavioral;

```

B. RTL and TestBench Results

1. Examples on Basys 3

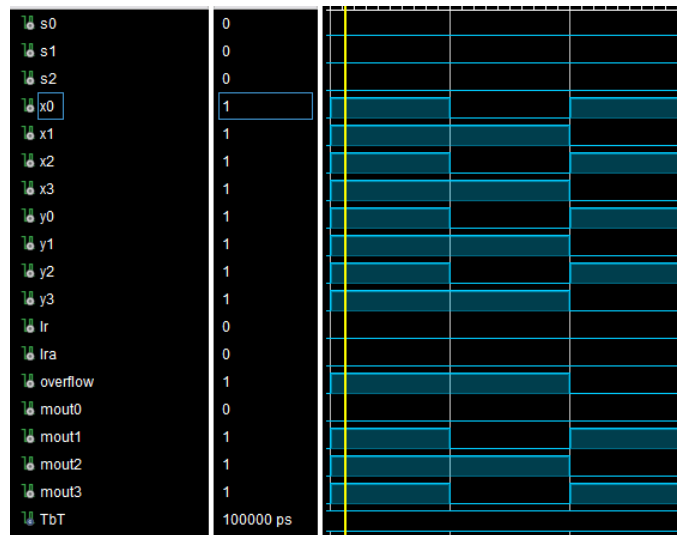


FIG. 4: select: addition

all inputs are 1

out: (1)1110

x:1010, y:1010

out:(1)0100

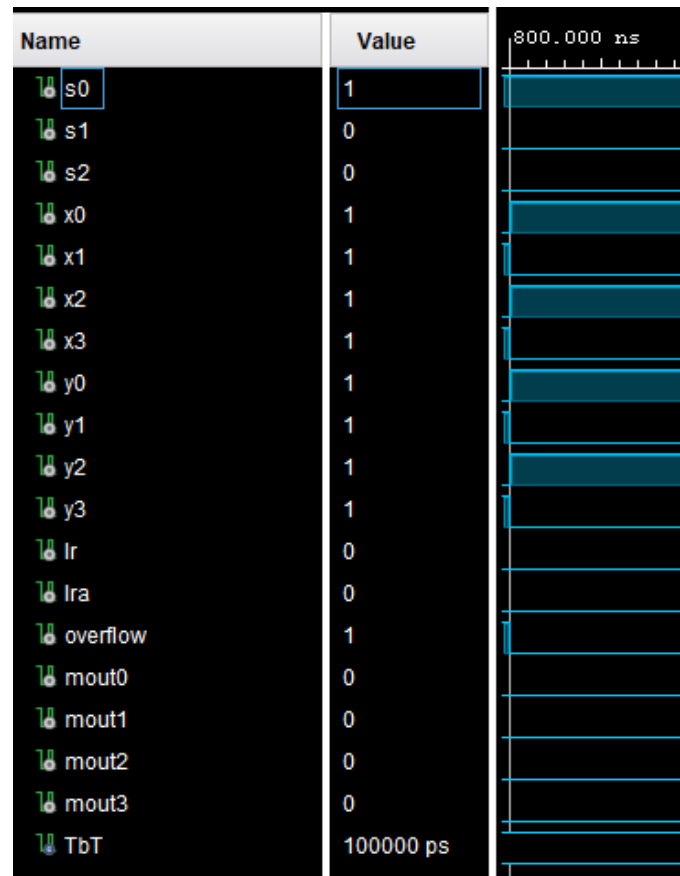


FIG. 5: select: Bit-Wise AND

x:0101, y:1010

out: 0000

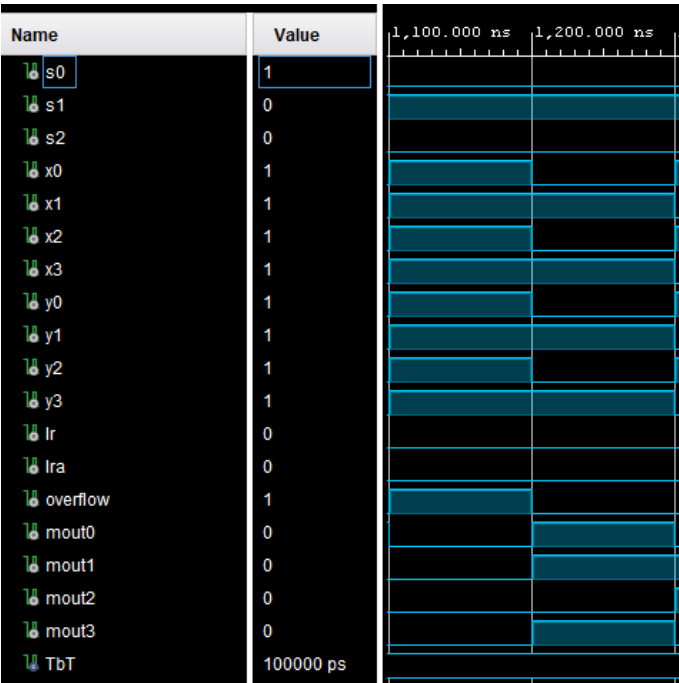


FIG. 6: select: Increment
x:1111
out: (1)0000
x:1010
out:1011




















Name	Value	
 s0	1	
 s1	1	
 s2	1	
 x0	1	
 x1	1	
 x2	1	
 x3	1	
 y0	1	
 y1	1	
 y2	1	
 y3	1	
 lr	0	
 lra	0	
 overflow	1	
 mout0	1	
 mout1	0	
 mout2	0	
 mout3	0	
 TbT	100000 p	

FIG. 7: select: twos compliment
x:1111
out: (1)0001

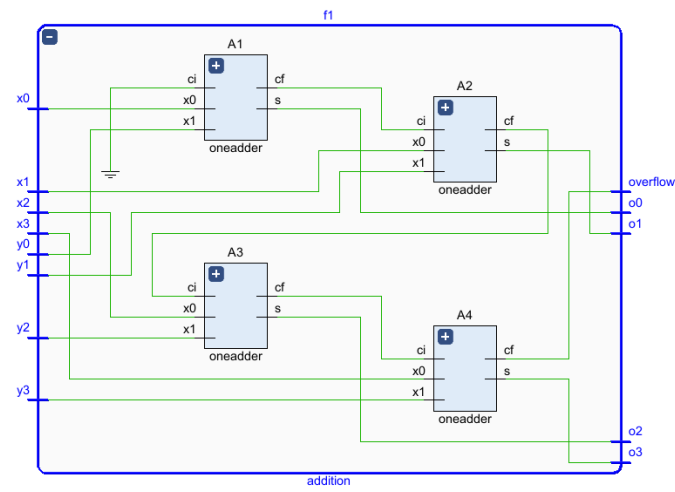


FIG. 8: 4-bit Adder

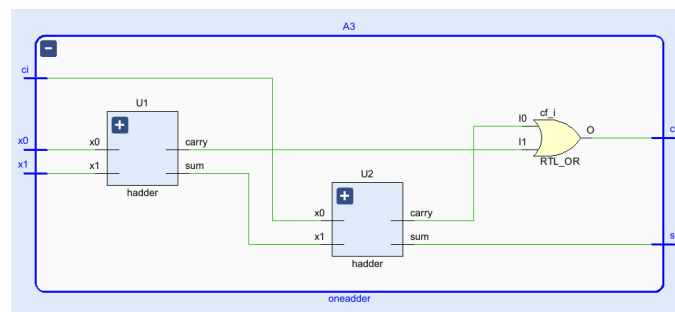


FIG. 9: Full Adder

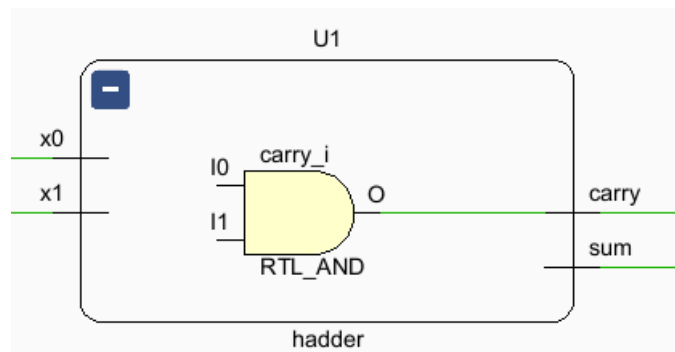


FIG. 10: Half Adder

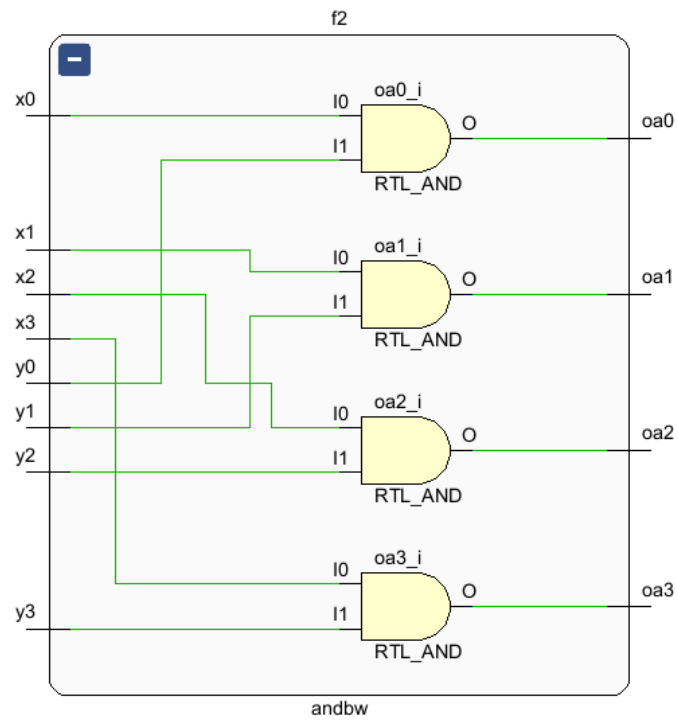


FIG. 11: Bit-wise AND

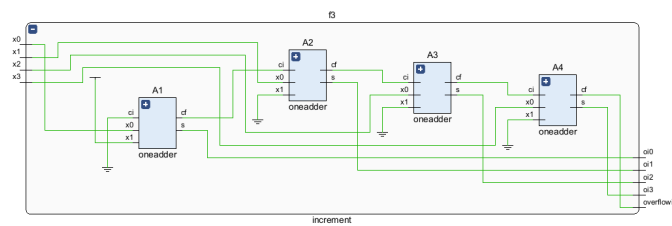


FIG. 12: Increment

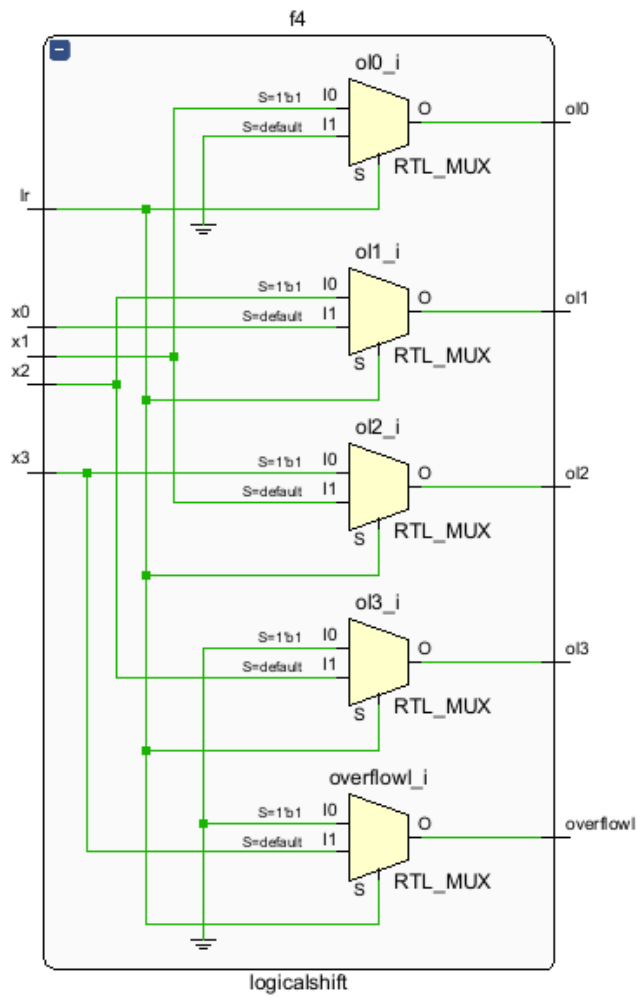


FIG. 13: Logical Shift

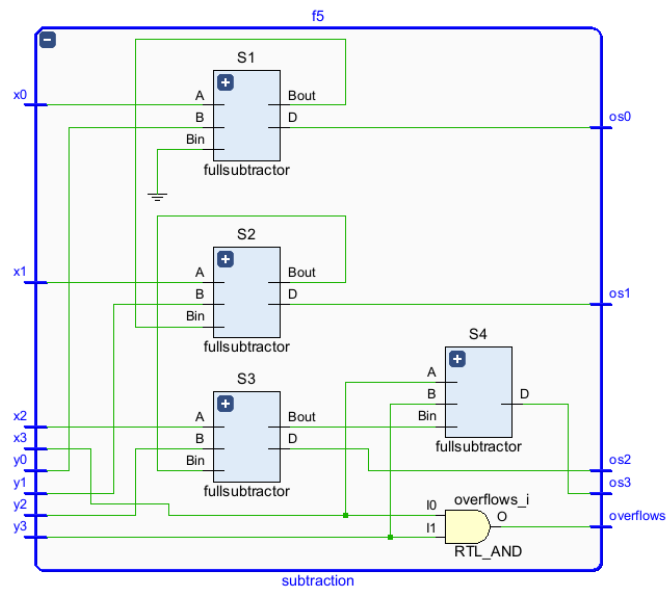


FIG. 14: 4-bit Subtractor

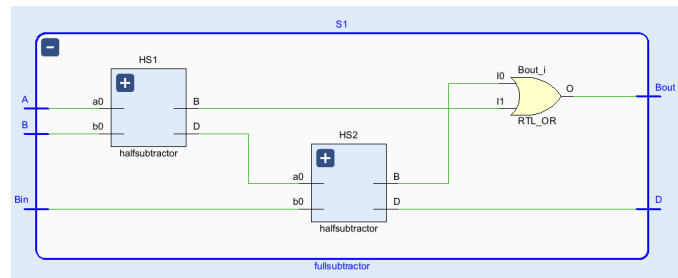


FIG. 15: Full Subtractor

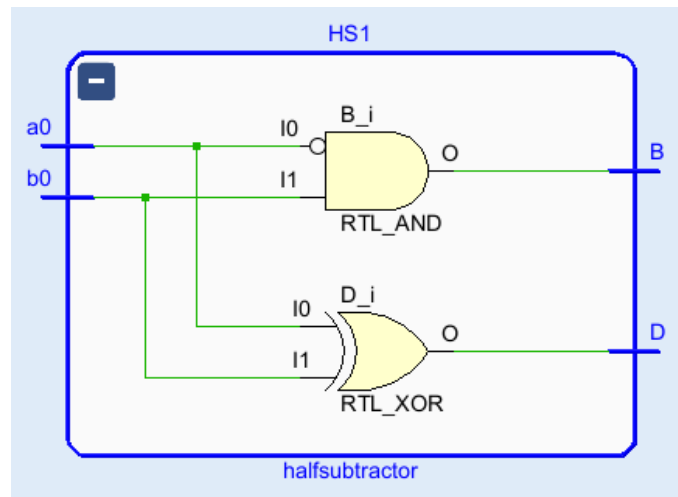


FIG. 16: Half Subtractor

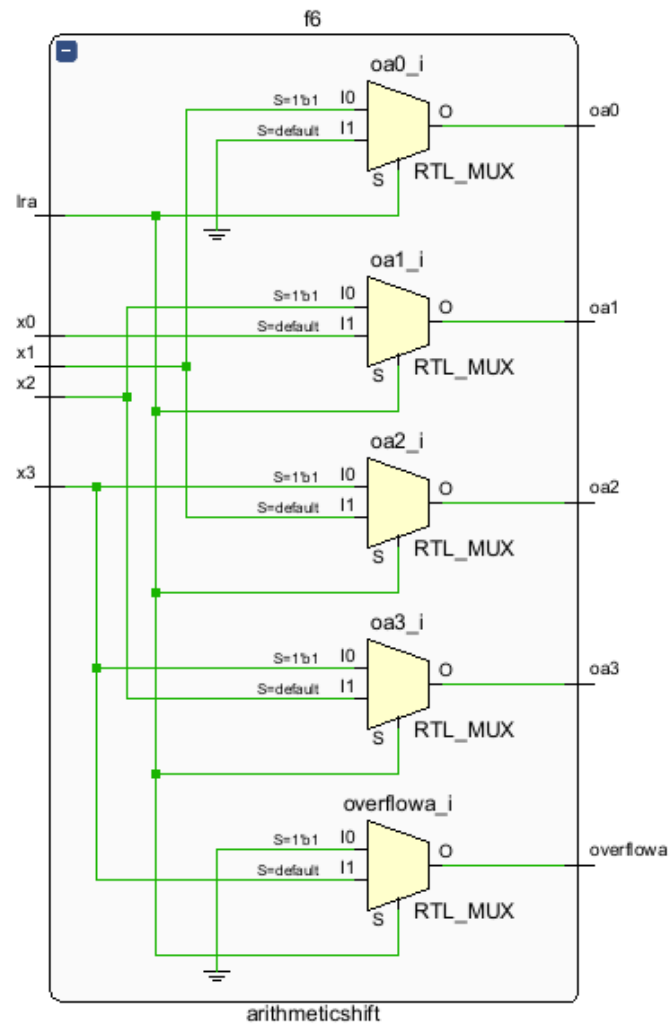


FIG. 17: Arithmetic Shift

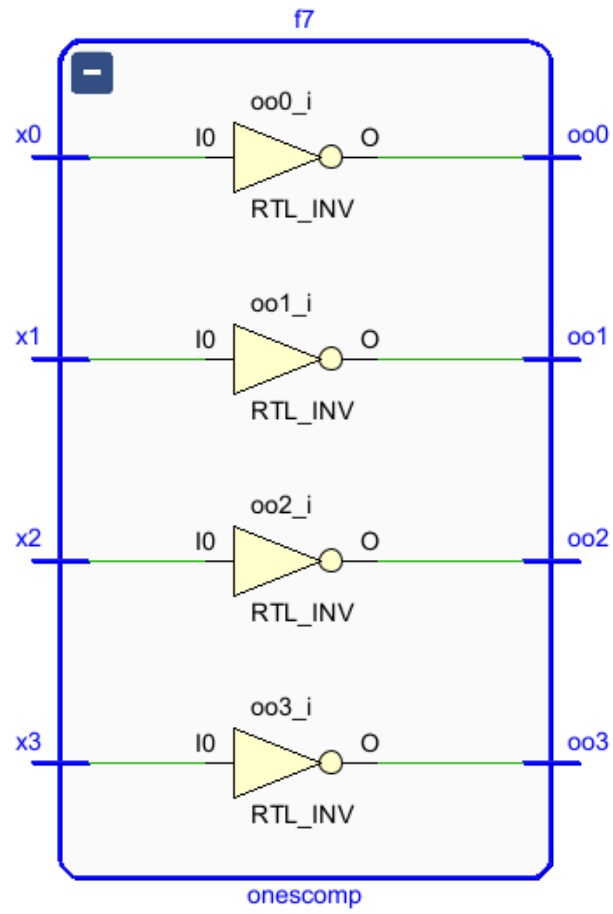


FIG. 18: Ones Compliment

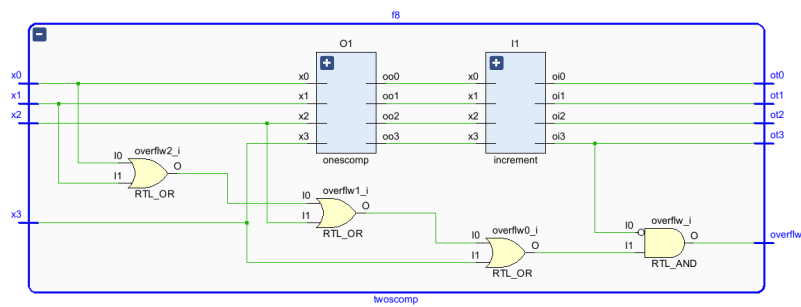


FIG. 19: Twos Compliment

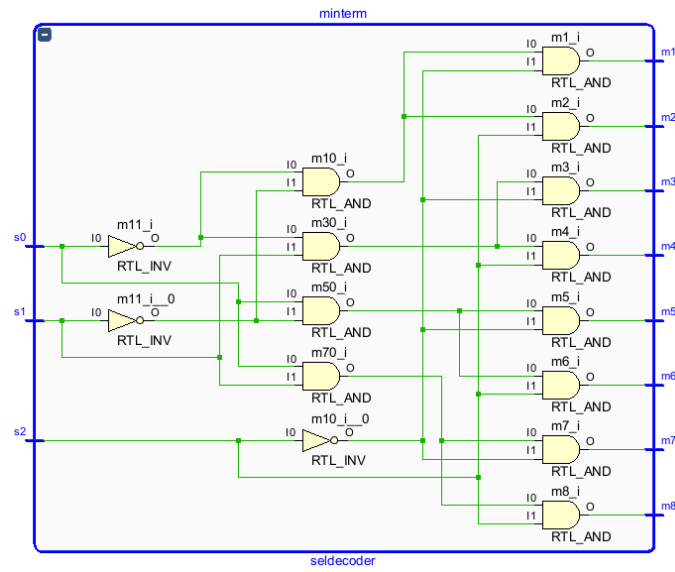


FIG. 20: Decoder

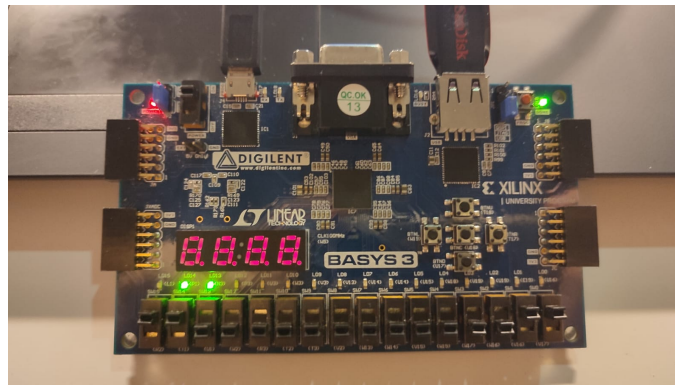


FIG. 21: Logical Shift
 x:0011
 out:1100

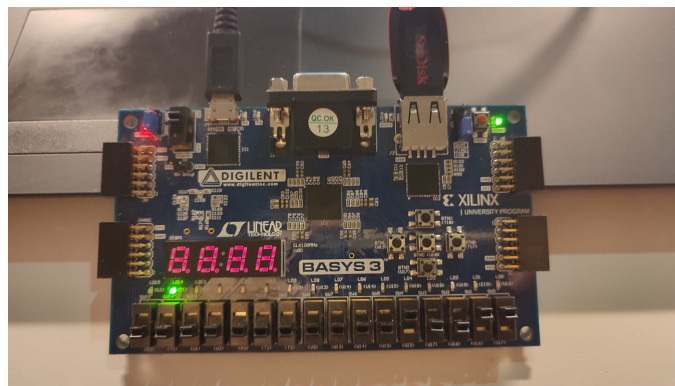


FIG. 22: Addition
 x:0011, y:0001
 out:0100

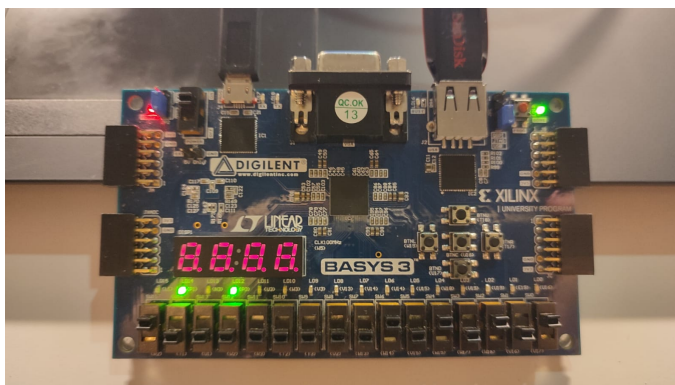


FIG. 23: AND
x:0101,y:0101
out:0101

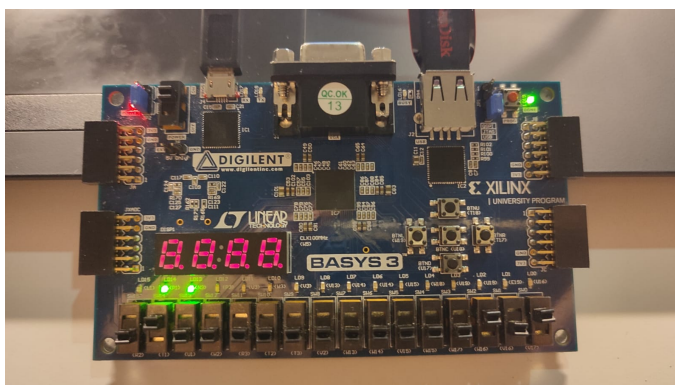


FIG. 24: Increment
x:0101
out:0110

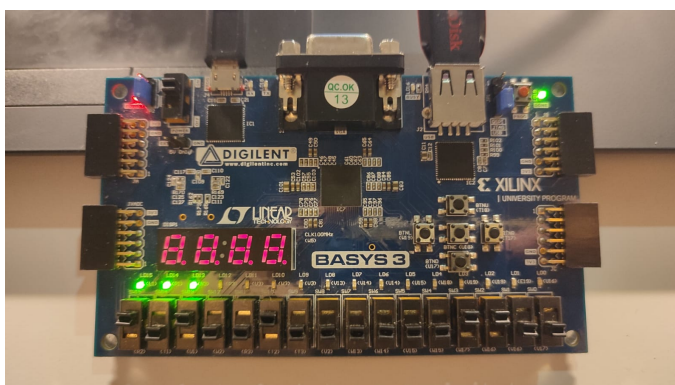


FIG. 25: Arithmetic Shift(R)
x:1100
out:1110

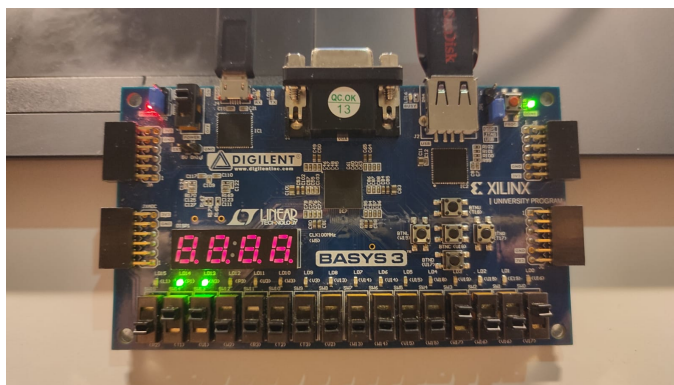


FIG. 26: Ones Compliment
 x:1001
 out:0110

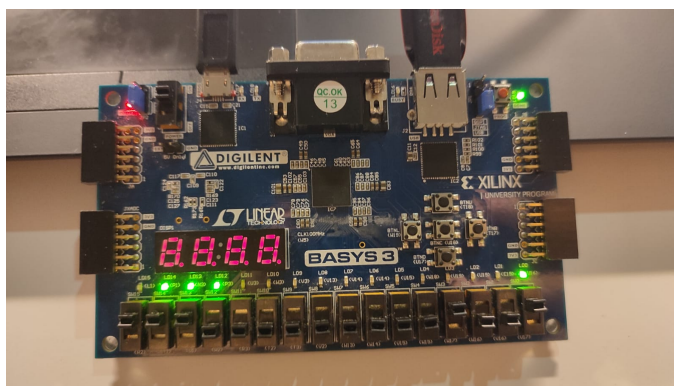


FIG. 27: Twos Compliment
 x:0001
 out:0111