ALU

Devesh Kumar, 16D070044

March 9, 2018

1 Overview of the experiment

The aim of this experiment is to write a VHDl code for an Alu which does addition subtraction , shift left and shift right operations.then to burn this code in the **krypton**.

2 Design/algorithm

2.1 Addition

To add two binary numbers, I first constructed a full adder. **fulladder** In a full adder, the input is three bits let be X1, X2, C0 and the output is S0 and C1.

X1	X2	C0	S	C1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table: for S and C1

X1 X2 -> / C0	0 0	0 1	1 1	1 0
0	0	1	0	1
1	1	0	1	0

For S

X1 X2 -> / C0	0 0	0 1	1 1	1 0
0	0	0	1	0
1	0	1	1	1

for C1

Form the table
$S = (((X_1)\ \mathbf{XOR}(X_2))\mathbf{XOR}(C_0))$

$C_1 = ((a.b) + (a.c_0)) + (b.c_0)$
Components Used

- onebitfulladder-It takes two numbers of one bit and a carry bit . It gives out the sum of the numbers along with the carry.
- eightbit- It adds the two eight bit numbers. It calls onebitfull adder number eight times.

2.2 Subtraction

To do subtraction I used 2's complement. Two subtract two numbers X1 and X2 with 2's complement. I first inverted the bits of X2 with **Inverter**.Then added it with X1 using the **full wave adder** with initial carry bit = 1.
Components Used

- eightbit- Adds the two eight bit string with initial carry 1.
- inverter- It inverts the a eight string number by using not gate for each bit

2.3 Shift left and Shift right

For each bit of Y i constructed a shifted version of input signal and a normal signal then I used the MUX to decide which of the two string. If value is 1

then I will keep the shifted otherwise I will keep the original
To shift a string by one unit in left or right direction I used shfter and shiftR components which shifted the string by 1 unit in left or right direction. Then depending on the value of of string y we call the funtion shift.
For example if the y0 is one i will call shifter one time if it is y1 then i will call it it two times.If y3 or above bit is one then only 0 string will be released.
Components Used

- muxbit Implements a normal mux that gives out one of the two input
- MUX- It takes a two eight bit string and gives out one of the eight bit string depending on one bit string.It uses mux bit for each of the string separately.
- shifter- It shifts left the eight bit string by one unit .It assigns 0 to the least bit.
- shifter R- It shifts right the eight bit string by one unit .It assigns 0 to the most significant bit.

2.4 ALU

First I calculated all the four possible operations i.e. addition, subtraction, shift left, shift right. Then using three mux I will choose what the output is.

3 VHDL code (well commented)

```
----mux_bit---output of mux_singlebit_depending_on_Y------------
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;
entity MUX_bit is
     port(
      c,d,Y : in std_logic;
                c0 : out std_logic
    );
                  end entity;
architecture behave of  MUX_bit is
signal y0: std_logic;
```

```
begin
y0 <= NOT(Y);
c0 <= ((c AND Y) OR ( d AND y0));
end architecture behave;

----mux---outputof mux _the_whole_vector(a,b)_depending_on_y1-----
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;

entity MUX is
     port(
      a,b : in std_logic_vector(7 downto 0);
                y1 : in std_logic;
      z : out std_logic_vector(7 downto 0)
    );
         end entity;
architecture behave of  MUX is
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;
 ----using_the_mux_bit_component_to_do_implement_of_each_bit_saparately
begin
X1: MUX_bit port map(c=>a(0),d=>b(0),y=>y1,c0=>z(0));
X2: MUX_bit port map(c=>a(1),d=>b(1),y=>y1,c0=>z(1));
X3: MUX_bit port map(c=>a(2),d=>b(2),y=>y1,c0=>z(2));
X4: MUX_bit port map(c=>a(3),d=>b(3),y=>y1,c0=>z(3));
X5: MUX_bit port map(c=>a(4),d=>b(4),y=>y1,c0=>z(4));
X6: MUX_bit port map(c=>a(5),d=>b(5),y=>y1,c0=>z(5));
X7: MUX_bit port map(c=>a(6),d=>b(6),y=>y1,c0=>z(6));
X8: MUX_bit port map(c=>a(7),d=>b(7),y=>y1,c0=>z(7));
end architecture behave;

------------------shift left--------------------------
----shifter--shifts_left------
library  std ;
```

```
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;
entity shifter is
     port(
      ip : in std_logic_vector(7 downto 0);
      op: out std_logic_vector(7 downto 0)
    );

end entity;
architecture behave of shifter is
Begin
  op(7) <= ip(6);
        op(6) <= ip(5);
        op(5) <= ip(4);
        op(4) <= ip(3);
        op(3) <= ip(2);
        op(2) <= ip(1);
        op(1) <= ip(0);
        op(0) <= '0';
end architecture behave;
entity shift_left is
    port(
      x,y : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0)
    );
end entity;

------------------shift left main---------------------------
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;
architecture behave of  shift_left is
    component shifter is
    port(
      ip : in std_logic_vector(7 downto 0);
      op : out std_logic_vector(7 downto 0)
    );
    end component shifter;
    component mux is
```

```
      port(
      a,b : in std_logic_vector(7 downto 0);
                y1 : in std_logic;
      z : out std_logic_vector(7 downto 0)
    );
    end component mux;
        signal dum0,dum1,dum2,dum3,dum4,dum5,dum6,set,
        out1,out2,out3,out4,out5,out6,out7,out8,
        out9,out10,out11 :std_logic_vector(7 downto 0);
        --dum stores the value if shift is done ones
        --out stores the value the value depending on y
 begin
 ----set represent the value if shifting is done more than 8 times
set(0)<='0';
set(1)<='0';
set(2)<='0';
set(3)<='0';
set(4)<='0';
set(5)<='0';
set(6)<='0';
set(7)<='0';
--if y0=1 then we will do one bit shift
 shift_1: shifter port map ( ip=>x, op=>dum0 );
 mux_1:  mux port map(a=> dum0, b=> x, y1=> y(0),z=> out1);

 --if y1=1 then we will do two times shift
 shift_2: shifter port map ( ip=>out1, op=>dum1 );
 mux_2:  mux port map(a=> dum1, b=> out1, y1=> y(1),z=> out2);

 shift_3: shifter port map ( ip=>out2, op=>dum2 );
 mux_3:  mux port map(a=> dum2, b=> out2, y1=> y(1),z=> out3);

 --if y2=1 we will do shifting 4 times
 shift_4: shifter port map ( ip=>out3, op=>dum3 );
 mux_4:  mux port map(a=> dum3, b=> out3, y1=> y(2),z=> out4) ;

 shift_5: shifter port map ( ip=>out4, op=>dum4 );
 mux_5:  mux port map(a=> dum4, b=> out4, y1=> y(2),z=> out5);
```

```
 shift_6: shifter port map ( ip=>out5, op=>dum5 );
 mux_6:  mux port map(a=> dum5, b=> out5, y1=> y(2),z=> out6);

 shift_7: shifter port map ( ip=>out6, op=>dum6 );
 mux_7:  mux port map(a=> dum6, b=> out6, y1=> y(2),z=> out11);

 --if y3 or above is 1 the we can directly say that output is 0
 mux_8:  mux port map(a=> SET, b=> out11 , y1=> y(3),z=> OUT7) ;
 mux_9:  mux port map(a=> SET, b=> out7, y1=> y(4),z=> OUT8) ;
 mux_10:  mux port map(a=> SET, b=> out8, y1=> y(5),z=> out9) ;
 mux_11:  mux port map(a=> SET, b=> out9, y1=> y(6),z=> out10);
 mux_12:  mux port map(a=> SET, b=> out10, y1=> y(7),z=> q);
end architecture behave;

------------------------shift_right----------------------------

----shifter_R----shifts the input to right--------------------
library  std ;
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;
entity shifter_R is
     port(
      ip : in std_logic_vector(7 downto 0);
      op: out std_logic_vector(7 downto 0)
    );

end entity;
architecture behave of shifter_R is
Begin
        op(7) <= '0';
        op(6) <= ip(7);
        op(5) <= ip(6);
        op(4) <= ip(5);
        op(3) <= ip(4);
        op(2) <= ip(3);
        op(1) <= ip(2);
        op(0) <= ip(1);
```

```
end architecture behave;

----main_shift_Right-------------------
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;

entity shift_right is
    port(
      x,y : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0)
    );
end entity;
architecture behave of  shift_right is
    component shifter_R is
    port(
      ip : in std_logic_vector(7 downto 0);
      op : out std_logic_vector(7 downto 0)
    );
    end component shifter_R;
    component mux is
      port(
      a,b : in std_logic_vector(7 downto 0);
                y1 : in std_logic;
      z : out std_logic_vector(7 downto 0)
    );
    end component mux;
         signal dum0,dum1,dum2,dum3,dum4,dum5,dum6,set,
         out1,out11,out2,out3,out4,out5,out6,
         out7,out8,out9,out10 :std_logic_vector(7 downto 0);
         --dum stores the value if shift is done ones
         --out stores the value the value depending on y
 begin
 ----set represent the value if shifting is done more than 8 times
set(0)<='0';
set(1)<='0';
set(2)<='0';
set(3)<='0';
set(4)<='0';
```

```
set(5)<='0';
set(6)<='0';
set(7)<='0';
--if y0=1 then we will do one bit shift
 shift_1: shifter_R port map ( ip=>x, op=>dum0 );
 mux_1:  mux port map(a=> dum0, b=> x, y1=> y(0),z=> out1);

 --if y1=1 then we will do two bit shift
 shift_2: shifter_R port map ( ip=>out1, op=>dum1 );
 mux_2:  mux port map(a=> dum1, b=> out1, y1=> y(1),z=> out2);

 shift_3: shifter_R port map ( ip=>out2, op=>dum2 );
 mux_3:  mux port map(a=> dum2, b=> out2, y1=> y(1),z=> out3);

 --if y2=1 then we will do four bit shift
 shift_4: shifter_R port map ( ip=>out3, op=>dum3 );
 mux_4:  mux port map(a=> dum3, b=> out3, y1=> y(2),z=> out4) ;

 shift_5: shifter_R port map ( ip=>out4, op=>dum4 );
 mux_5:  mux port map(a=> dum4, b=> out4, y1=> y(2),z=> out5);

 shift_6: shifter_R port map ( ip=>out5, op=>dum5 );
 mux_6:  mux port map(a=> dum5, b=> out5, y1=> y(2),z=> out6);

 shift_7: shifter_R port map ( ip=>out6, op=>dum6 );
 mux_7:  mux port map(a=> dum6, b=> out6, y1=> y(2),z=> out11);

--if y0=1 then we will output the 0
 mux_8:  mux port map(a=> SET, b=> out11 , y1=> y(3),z=> OUT7) ;
 mux_9:  mux port map(a=> SET, b=> out7, y1=> y(4),z=> OUT8) ;
 mux_10:  mux port map(a=> SET, b=> out8, y1=> y(5),z=> out9) ;
 mux_11:  mux port map(a=> SET, b=> out9, y1=> y(6),z=> out10);
 mux_12:  mux port map(a=> SET, b=> out10, y1=> y(7),z=> q);
end architecture behave;

 ---------------onebitfull adder--------------
use  std.standard.all;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity onebit_fulladder is
   port (
          a,b,cin :in std_logic;
          s,co : out std_logic
   );
end entity onebit_fulladder;
architecture behave of onebit_fulladder is
begin
 s <= ((a XOR b) XOR cin);
 co<= ((a AND b) OR (a AND cin)) OR  (b AND cin);
 end architecture behave;

----------------eight-bitAdder-----------------------
library ieee;
use ieee.std_logic_1164.all;
 entity eightbit is
   port (
         x,y: in std_logic_vector(7 downto 0);
         cin: in std_logic;
         sum: out std_logic_vector(7 downto 0);
         cout: out std_logic
   );
end entity eightbit;
architecture behave of eightbit is
     signal w : std_logic_vector(7 downto 0);
          component onebit_fulladder is
          port(
          a,b,cin : in std_logic;
          s,co : out std_logic
          );
          end component onebit_fulladder;
begin
              -- adding first bits and prpagating the carry to the next
x1:onebit_fulladder port map (a=>x(0), b=>y(0), cin=>cin , s=>sum(0), co=>w(0));
x2:onebit_fulladder port map (a=>x(1), b=>y(1), cin=>w(0) , s=>sum(1), co=>w(1));
```

```
x3:onebit_fulladder port map (a=>x(2), b=>y(2), cin=>w(1) , s=>sum(2), co=>w(2));
x4:onebit_fulladder port map (a=>x(3), b=>y(3), cin=>w(2) , s=>sum(3), co=>w(3));
x5:onebit_fulladder port map (a=>x(4), b=>y(4), cin=>w(3) , s=>sum(4), co=>w(4));
x6:onebit_fulladder port map (a=>x(5), b=>y(5), cin=>w(4) , s=>sum(5), co=>w(5));
x7:onebit_fulladder port map (a=>x(6), b=>y(6), cin=>w(5) , s=>sum(6), co=>w(6));
x8:onebit_fulladder port map (a=>x(7), b=>y(7), cin=>w(6) , s=>sum(7), co=>cout);
end architecture behave;

----------------subtractor----------------
 -----inverter--------------------
library ieee;
use ieee.std_logic_1164.all;

entity inverter is
        port (
                ip: in std_logic_vector (7 downto 0);
                op: out std_logic_vector (7 downto 0)
        );
end entity inverter;
architecture behave of inverter is
begin
        op(0) <= NOT ip(0);
        op(1) <= NOT ip(1);
        op(2) <= NOT ip(2);
        op(3) <= NOT ip(3);
        op(4) <= NOT ip(4);
        op(5) <= NOT ip(5);
        op(6) <= NOT ip(6);
        op(7) <= NOT ip(7);
end architecture behave;
--------subtractor main-----------------
library ieee;
use ieee.std_logic_1164.all;
entity Subtractor is
        port (
                x,y: in std_logic_vector (7 downto 0);
                sum: out std_logic_vector (7 downto 0);
                cout: out std_logic
```

```
        );
end entity Subtractor;
architecture behave of Subtractor is
        component eightbit is
                port (
                x,y: in std_logic_vector (7 downto 0);
                cin: in std_logic;
                sum: out std_logic_vector (7 downto 0);
                cout: out std_logic
        );
        end component eightBit;
        component inverter is
                port (
                ip: in std_logic_vector (7 downto 0);
                op: out std_logic_vector (7 downto 0)
        );
        end component inverter;
        signal w: std_logic_vector (7 downto 0);
begin
        inv: inverter port map ( ip=>y, op=>w );--inverting y
        add1: eightBit port map ( x=>x, y=>w, cin=>'1', sum=>sum, cout=>cout);
        --adding the inverted signal, x, and 1
end architecture behave;

-----ALU main-------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity alu is
        port( m,n: in std_logic_vector(7 downto 0);
        x0,x1 : in std_logic ; p : out std_logic_vector(7 downto 0));
end entity;

architecture behave of alu is
        signal sig1,sig2,sig3,sig4,sig5,sig6 : std_logic_vector(7 downto 0);
         signal        q,w : std_logic;

        component eightbit is
```

```
                port (
                x,y: in std_logic_vector (7 downto 0);
                cin: in std_logic;
                sum: out std_logic_vector (7 downto 0);
                cout: out std_logic
        );
        end component eightBit;

        component shift_left is
           port(
      x,y : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0)
    );
        end component;

        component shift_right is
          port(
      x,y : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0)
    );
        end component;

        component subtractor is
        port (
                x,y: in std_logic_vector (7 downto 0);
                sum: out std_logic_vector (7 downto 0);
                cout: out std_logic
        );
        end component;

        component mux is
      port(
      a,b : in std_logic_vector(7 downto 0);
          y1 : in std_logic;
      z : out std_logic_vector(7 downto 0)
    );
         end component;
-----------------------------------------------
```

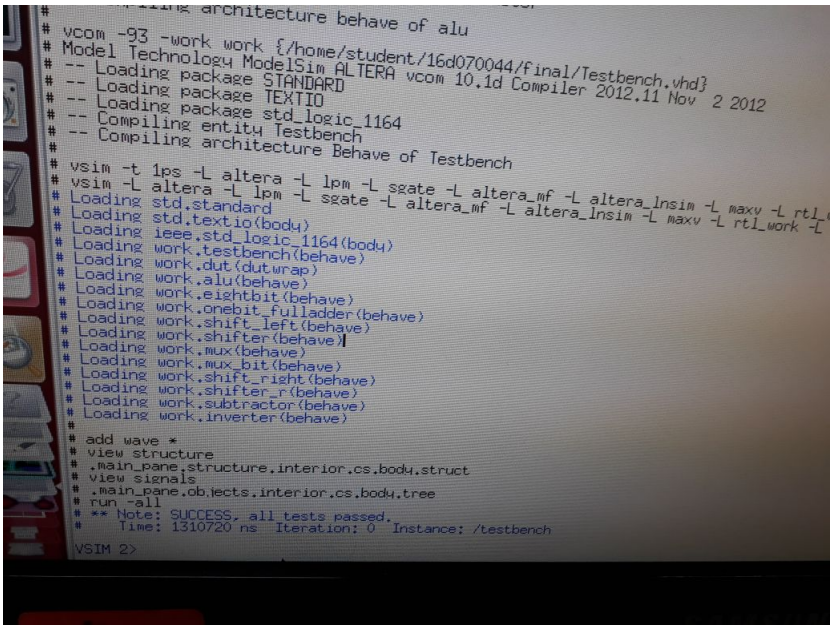
```
begin
a: eightbit   port map(x => m,  y=> n, cin=>'0', sum => sig1, cout => q);
b: shift_left port map(x => m, y => n, q => sig4);
c: shift_right port map(x => m, y => n, q => sig3);
d: subtractor   port map(x => m, y => n,sum  => sig2 ,cout => w);
--implementing 4 X 1 Mux using three 2 X 1 MUX
 mux_1:  mux port map(a=> sig2, b=> sig1, y1=> x0,z=> sig5);
 mux_2:  mux port map(a=> sig4, b=> sig3, y1=> x0,z=> sig6);
 mux_3:  mux port map(a=> sig6, b=> sig5, y1=> x1,z=> p);

        end architecture behave;
```

4 Test bench

I used the test bench uploaded in the moodle. I just changed the no of input to 18 and the location of the input and output tracefiles.

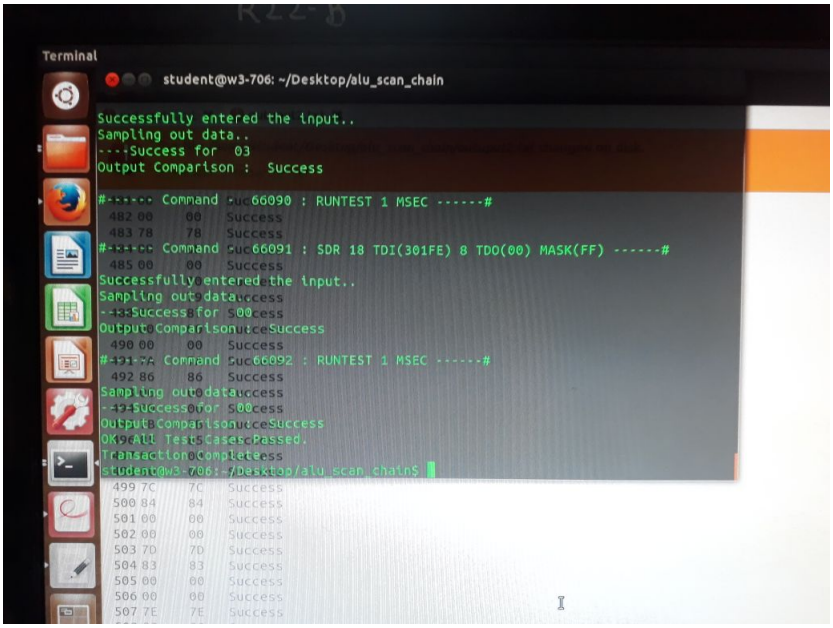
5 Simulation results: RTL viewer, waveforms if readable, simulation report after running test bench



We can see that all the test cases were passed.

6 Scan chain result

I showed this scan chain result to Deep Mistry (lab TA).


Terminal
student@w3-706: ~/Desktop/alu_scan_chain
Successfully entered the input..
Sampling out data..
----Success for 03
Output Comparison : Success
#------ Command Success 66090 : RUNTEST 1 MSEC ------#
482 00 00 Success
483 78 78 Success
#------ Command Success 66091 : SDR 18 TDI(301FE) 8 TDO(00) MASK(FF) ------#
485 00 00 Success
Successfully entered the input..
Sampling out data..Success
----Success for 00..Success
Output Comparison..Success
490 00 00 Success
#------ Command Success 66092 : RUNTEST 1 MSEC ------#
492 86 86 Success
Sampling out data..Success
----Success for 00..Success
Output Comparison..Success
OK..All Tests Cases Passed.
Transaction Complete..Success
student@w3-706:~/Desktop/alu_scan_chain$
499 7C 7C Success
500 84 84 Success
501 00 00 Success
502 00 00 Success
503 7D 7D Success
504 83 83 Success
505 00 00 Success
506 00 00 Success
507 7E 7E Success