Lab Report : Arithmetic Logic Unit

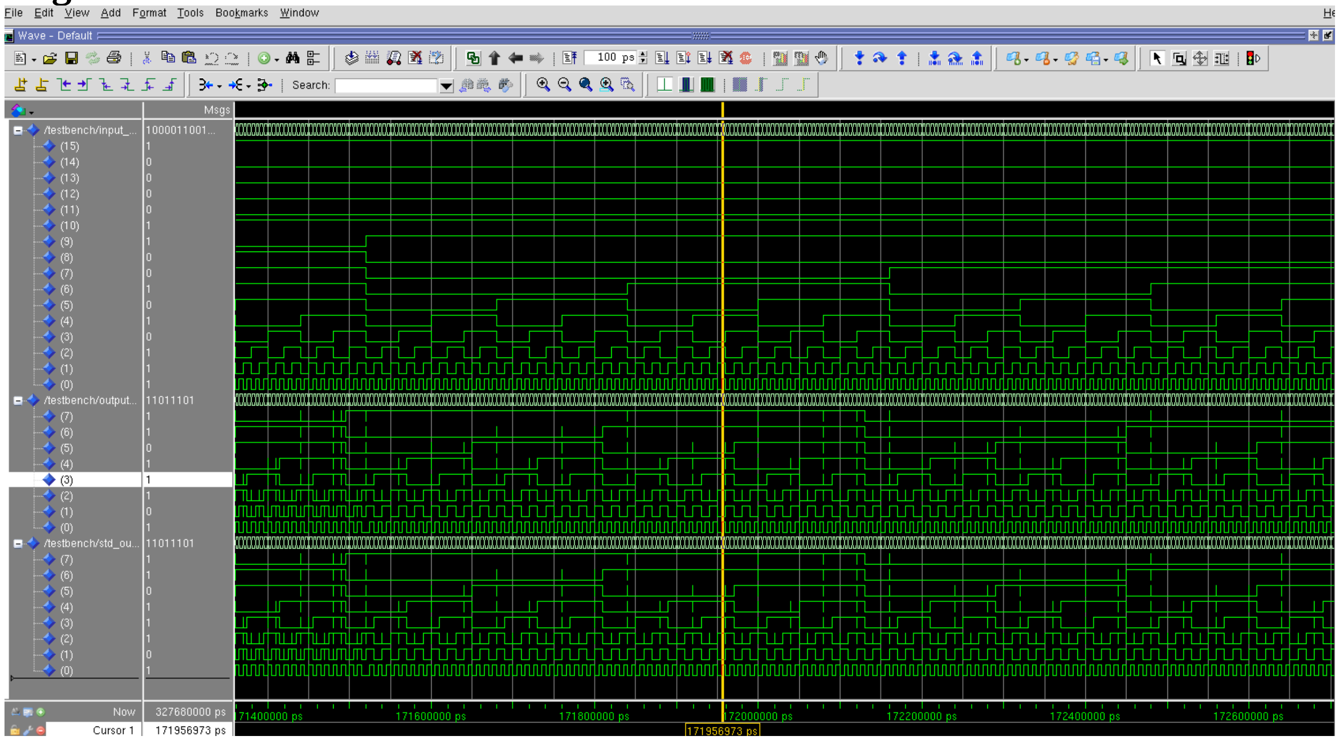
Name : Saqib Azim
Roll No. : 150070031

Eight Bit Adder:

eightbitadder.vhd

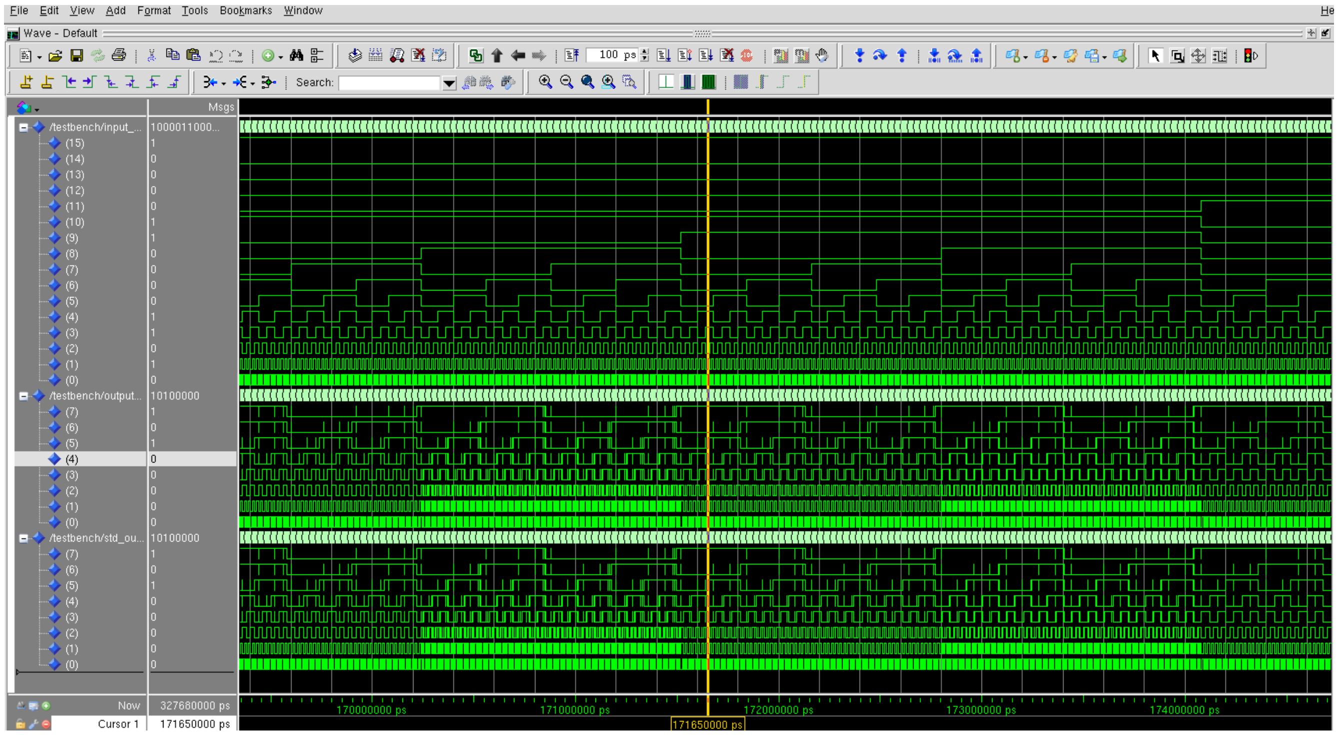
```
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;

entity eightbitadder is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of eightbitadder is
        component onebitadder is
        port(x,y,cin : in std_logic;
        z,cout : out std_logic);
        end component;
        signal c:std_logic_vector(7 downto 0);
begin
 adder0:onebitadder port map(x=>x(0),y=>y(0),cin=>'0',z=>z(0),cout=>c(0));
 adder1:onebitadder port map(x=>x(1),y=>y(1),cin=>c(0),z=>z(1),cout=>c(1));
 adder2:onebitadder port map(x=>x(2),y=>y(2),cin=>c(1),z=>z(2),cout=>c(2));
 adder3:onebitadder port map(x=>x(3),y=>y(3),cin=>c(2),z=>z(3),cout=>c(3));
 adder4:onebitadder port map(x=>x(4),y=>y(4),cin=>c(3),z=>z(4),cout=>c(4));
 adder5:onebitadder port map(x=>x(5),y=>y(5),cin=>c(4),z=>z(5),cout=>c(5));
 adder6:onebitadder port map(x=>x(6),y=>y(6),cin=>c(5),z=>z(6),cout=>c(6));
 adder7:onebitadder port map(x=>x(7),y=>y(7),cin=>c(6),z=>z(7),cout=>c(7));
end behave;

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

entity onebitadder is
port (x,y,cin: in std_logic;
                    z,cout:out std_logic);
end entity;

architecture behave of onebitadder is
begin
          z <= (cin xor (x xor y));
          cout <= ((cin and (x xor y)) or (x and y));
end behave;
```

Register Transfer Level Results-



```
/testbench/input_...   1000011001...
  (15)  1
  (14)  0
  (13)  0
  (12)  0
  (11)  0
  (10)  1
  (9)   1
  (8)   0
  (7)   0
  (6)   1
  (5)   0
  (4)   1
  (3)   0
  (2)   1
  (1)   1
  (0)   1
/testbench/output...   11011101
  (7)   1
  (6)   1
  (5)   0
  (4)   1
  (3)   1
  (2)   1
  (1)   0
  (0)   1
/testbench/std_ou...   11011101
  (7)   1
  (6)   1
  (5)   0
  (4)   1
  (3)   1
  (2)   1
  (1)   0
  (0)   1

Now       327680000 ps
Cursor 1  171956973 ps

171400000 ps   171600000 ps   171800000 ps   172000000 ps   172200000 ps   172400000 ps   172600000 ps
171956973 ps
```

Gate Level Simulation

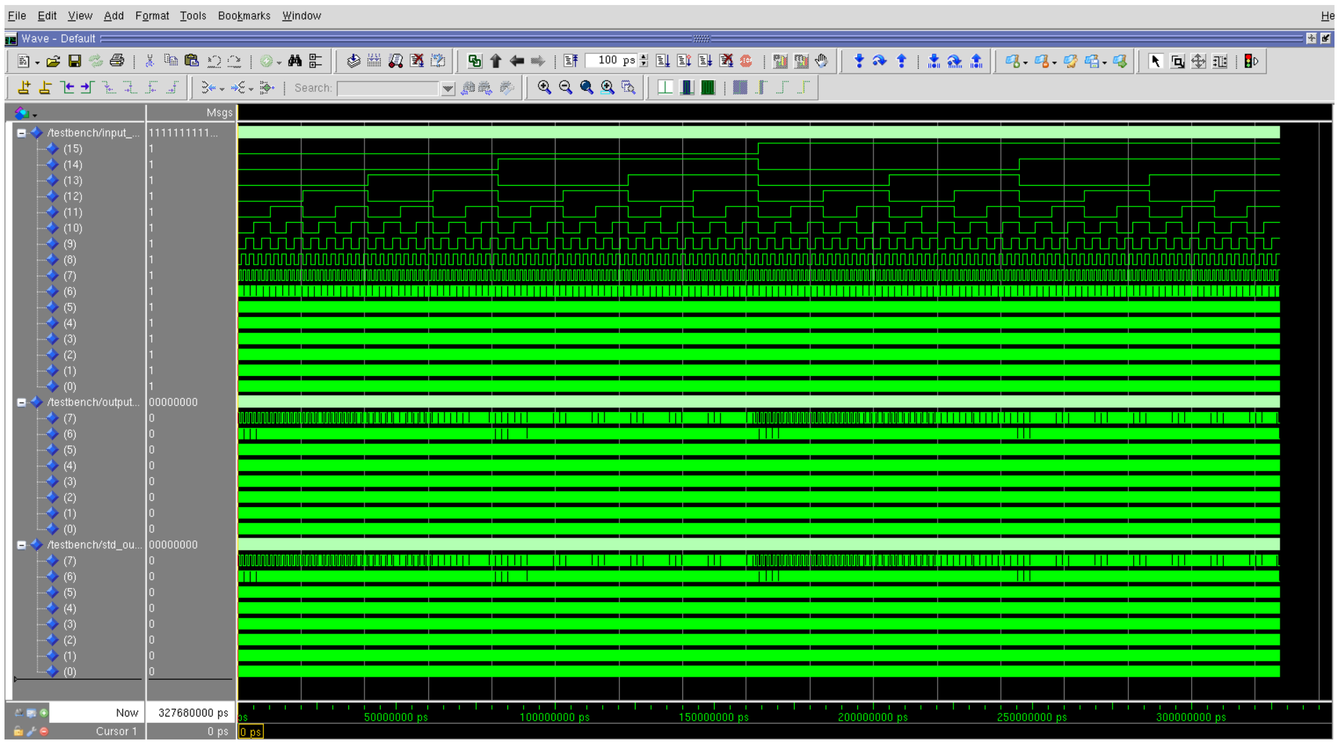


Eight Bit Subtractor:

eightbitsubtractor.vhd

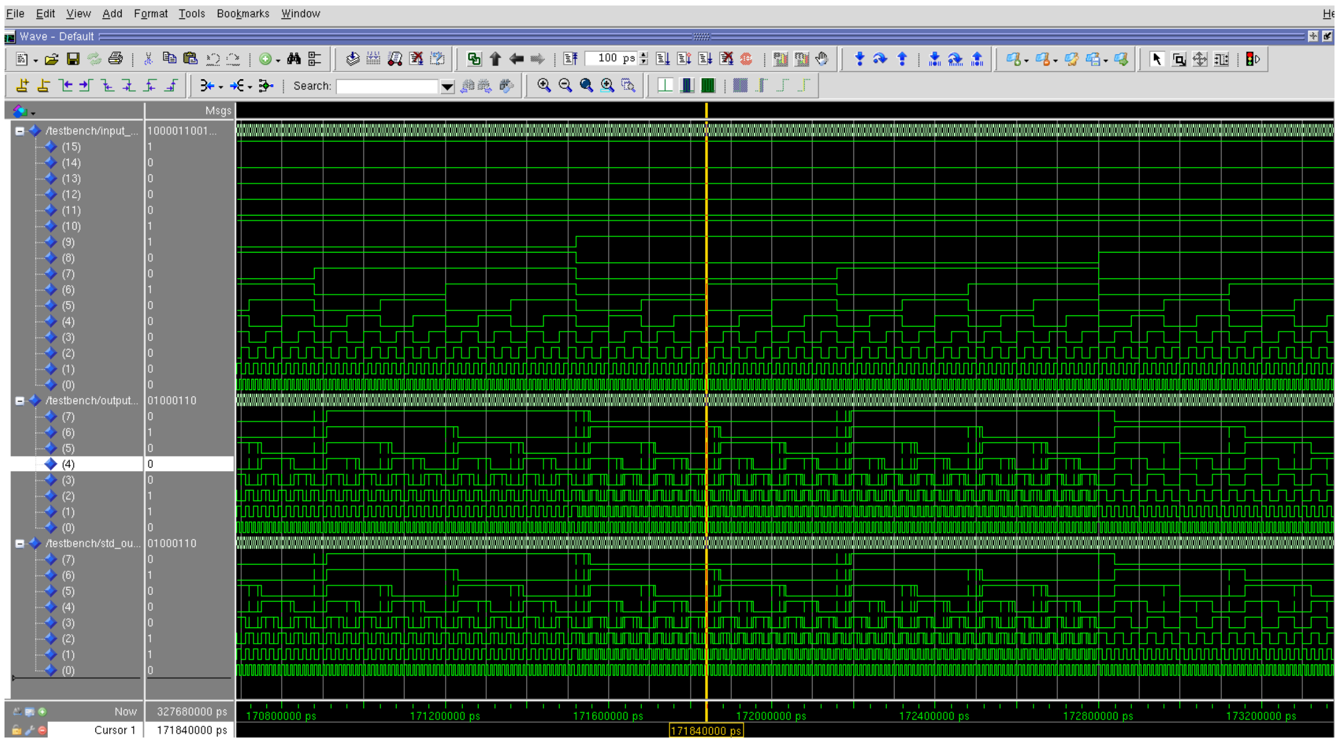
```
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;

entity eightbitsubtractor is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of eightbitsubtractor is
        component onebitsubtractor is
        port(x,y,cin : in std_logic;
        z,cout : out std_logic);
        end component;
        signal c:std_logic_vector(7 downto 0);
begin

 subtractor0:onebitsubtractor port
map(x=>x(0),y=>y(0),cin=>'0',z=>z(0),cout=>c(0));
 subtractor1:onebitsubtractor port
map(x=>x(1),y=>y(1),cin=>c(0),z=>z(1),cout=>c(1));
 subtractor2:onebitsubtractor port
map(x=>x(2),y=>y(2),cin=>c(1),z=>z(2),cout=>c(2));
 subtractor3:onebitsubtractor port
map(x=>x(3),y=>y(3),cin=>c(2),z=>z(3),cout=>c(3));
 subtractor4:onebitsubtractor port
map(x=>x(4),y=>y(4),cin=>c(3),z=>z(4),cout=>c(4));
 subtractor5:onebitsubtractor port
map(x=>x(5),y=>y(5),cin=>c(4),z=>z(5),cout=>c(5));
 subtractor6:onebitsubtractor port
map(x=>x(6),y=>y(6),cin=>c(5),z=>z(6),cout=>c(6));
 subtractor7:onebitsubtractor port
map(x=>x(7),y=>y(7),cin=>c(6),z=>z(7),cout=>c(7));
end behave;

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

entity onebitsubtractor is
port (x,y,cin: in std_logic;
                z,cout:out std_logic);
end entity;

architecture behave of onebitsubtractor is
begin
        z <= (cin xor (x xor y));
        cout <= ((not x) and cin) or ((not x) and y) or (cin and y);
end behave;
```

Register Transfer Level Results-



```
/testbench/input_...  1111111111...
  (15)  1
  (14)  1
  (13)  1
  (12)  1
  (11)  1
  (10)  1
  (9)   1
  (8)   1
  (7)   1
  (6)   1
  (5)   1
  (4)   1
  (3)   1
  (2)   1
  (1)   1
  (0)   1
/testbench/output...  00000000
  (7)   0
  (6)   0
  (5)   0
  (4)   0
  (3)   0
  (2)   0
  (1)   0
  (0)   0
/testbench/std_ou...  00000000
  (7)   0
  (6)   0
  (5)   0
  (4)   0
  (3)   0
  (2)   0
  (1)   0
  (0)   0
Now       327680000 ps
Cursor 1  0 ps
```

Gate Level Simulation Results-



```
/testbench/input_...  1000011001...
  (15)  1
  (14)  0
  (13)  0
  (12)  0
  (11)  0
  (10)  1
  (9)   1
  (8)   0
  (7)   0
  (6)   1
  (5)   0
  (4)   0
  (3)   0
  (2)   0
  (1)   0
  (0)   0
/testbench/output...  01000110
  (7)   0
  (6)   1
  (5)   0
  (4)   0
  (3)   0
  (2)   1
  (1)   1
  (0)   0
/testbench/std_ou...  01000110
  (7)   0
  (6)   1
  (5)   0
  (4)   0
  (3)   0
  (2)   1
  (1)   1
  (0)   0
Now       327680000 ps
Cursor 1  171840000 ps
```

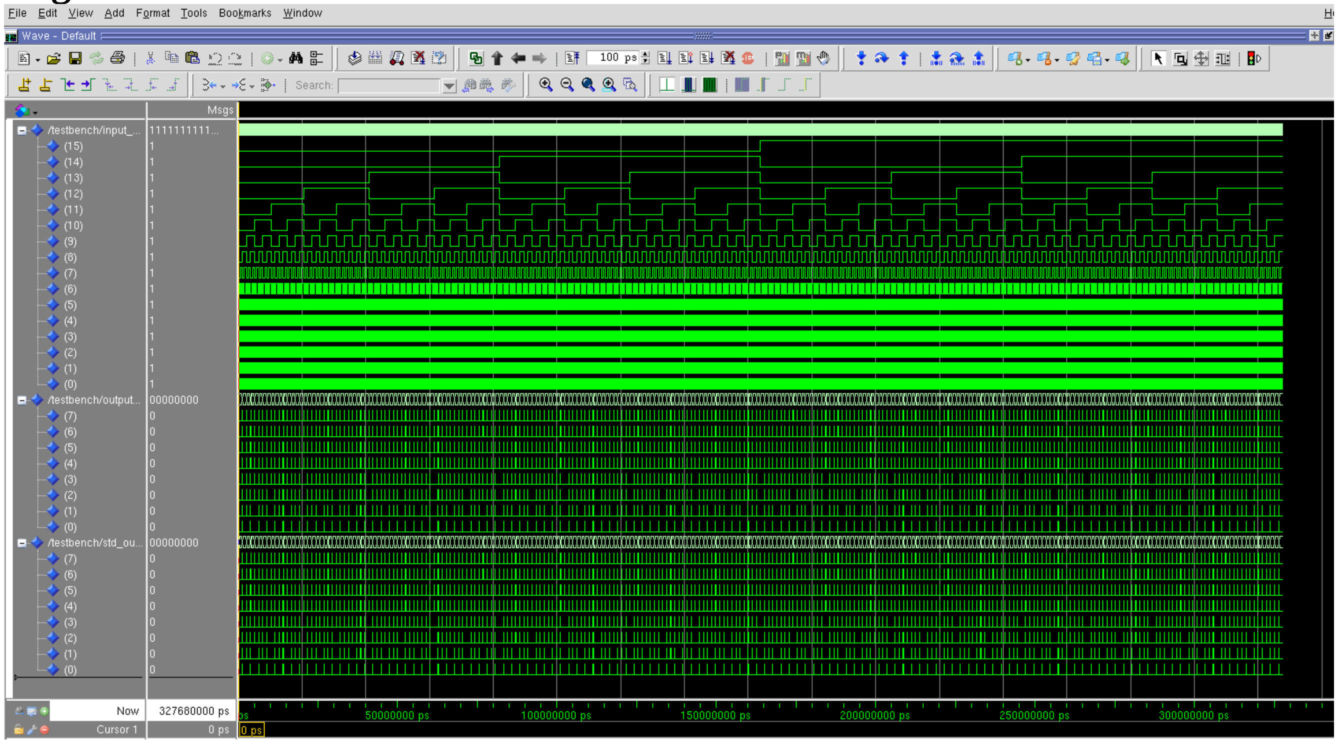
Left Shifter:

leftshift.vhd

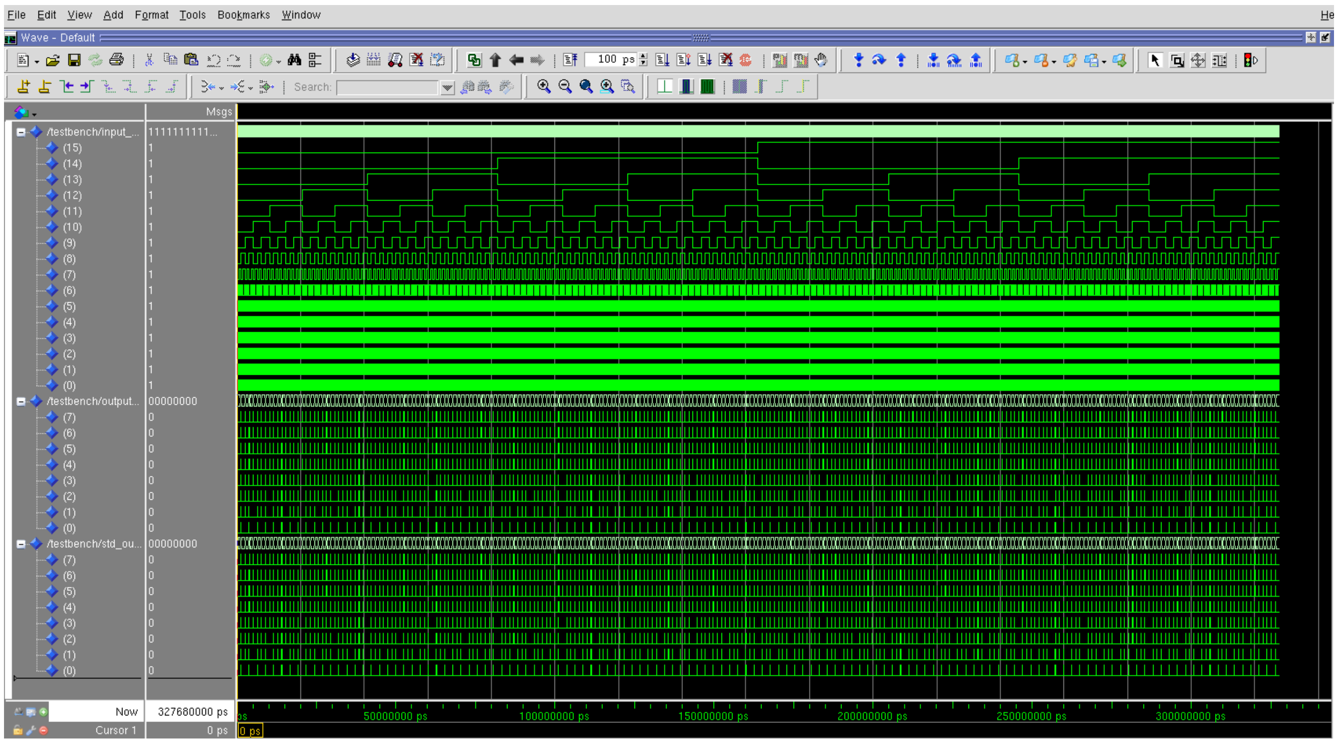
```
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;

entity leftshift is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of leftshift is
        signal a, b:std_logic_vector(7 downto 0);
        signal c:std_logic;
begin
c <= (not (y(7) or y(6) or y(5) or y(4) or y(3)));

a(7) <= (x(3) and y(2)) or ((not y(2)) and x(7));
a(6) <= (x(2) and y(2)) or ((not y(2)) and x(6));
a(5) <= (x(1) and y(2)) or ((not y(2)) and x(5));
a(4) <= (x(0) and y(2)) or ((not y(2)) and x(4));
a(3) <= ((not y(2)) and x(3));
a(2) <= ((not y(2)) and x(2));
a(1) <= ((not y(2)) and x(1));
a(0) <= ((not y(2)) and x(0));

b(7) <= (a(5) and y(1)) or ((not y(1)) and a(7));
b(6) <= (a(4) and y(1)) or ((not y(1)) and a(6));
b(5) <= (a(3) and y(1)) or ((not y(1)) and a(5));
b(4) <= (a(2) and y(1)) or ((not y(1)) and a(4));
b(3) <= (a(1) and y(1)) or ((not y(1)) and a(3));
b(2) <= (a(0) and y(1)) or ((not y(1)) and a(2));
b(1) <= ((not y(1)) and a(1));
b(0) <= ((not y(1)) and a(0));
```

```
z(7) <= ((b(6) and y(0)) or ((not y(0)) and b(7))) and c;
z(6) <= ((b(5) and y(0)) or ((not y(0)) and b(6))) and c;
z(5) <= ((b(4) and y(0)) or ((not y(0)) and b(5))) and c;
z(4) <= ((b(3) and y(0)) or ((not y(0)) and b(4))) and c;
z(3) <= ((b(2) and y(0)) or ((not y(0)) and b(3))) and c;
z(2) <= ((b(1) and y(0)) or ((not y(0)) and b(2))) and c;
z(1) <= ((b(0) and y(0)) or ((not y(0)) and b(1))) and c;
z(0) <= (((not y(0)) and b(0))) and c;

end behave;
```

Register Transfer Level Results-



Gate Level Simulation Results-



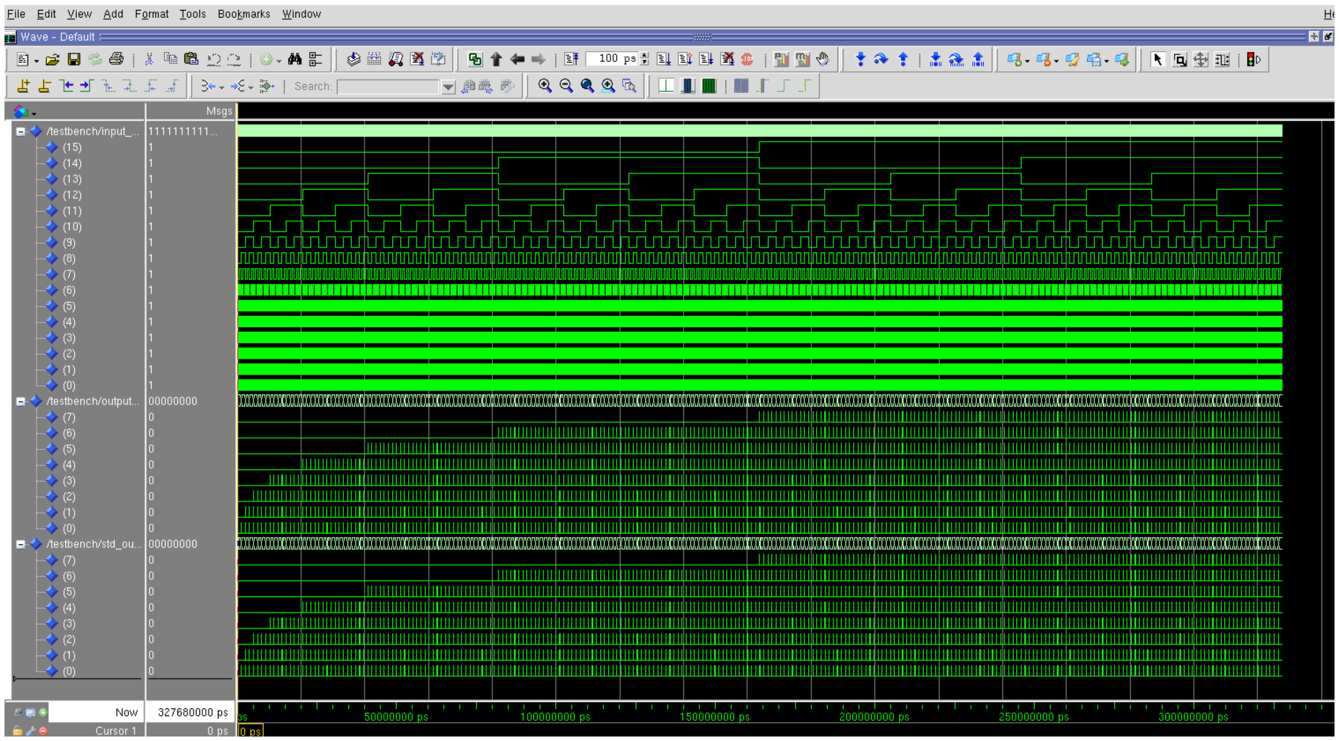
RightShifter:

rightshift.vhd

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

entity rightshift is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of rightshift is
        signal a, b:std_logic_vector(7 downto 0);
        signal c:std_logic;
```

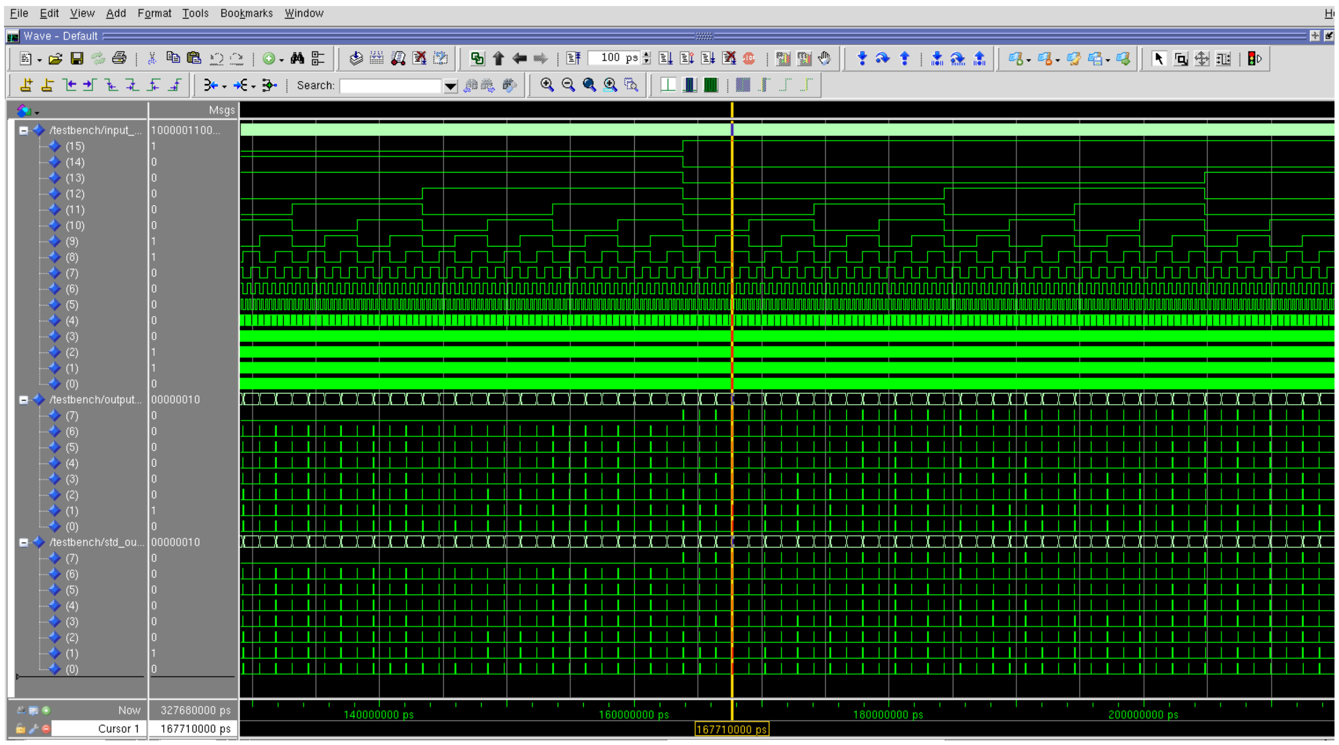
```
begin
c <= (not (y(7) or y(6) or y(5) or y(4) or y(3)));

a(0) <= (x(4) and y(2)) or ((not y(2)) and x(0));
a(1) <= (x(5) and y(2)) or ((not y(2)) and x(1));
a(2) <= (x(6) and y(2)) or ((not y(2)) and x(2));
a(3) <= (x(7) and y(2)) or ((not y(2)) and x(3));
a(4) <= ((not y(2)) and x(4));
a(5) <= ((not y(2)) and x(5));
a(6) <= ((not y(2)) and x(6));
a(7) <= ((not y(2)) and x(7));

b(0) <= (a(2) and y(1)) or ((not y(1)) and a(0));
b(1) <= (a(3) and y(1)) or ((not y(1)) and a(1));
b(2) <= (a(4) and y(1)) or ((not y(1)) and a(2));
b(3) <= (a(5) and y(1)) or ((not y(1)) and a(3));
b(4) <= (a(6) and y(1)) or ((not y(1)) and a(4));
b(5) <= (a(7) and y(1)) or ((not y(1)) and a(5));
b(6) <= ((not y(1)) and a(6));
b(7) <= ((not y(1)) and a(7));

z(0) <= ((b(1) and y(0)) or ((not y(0)) and b(0))) and c;
z(1) <= ((b(2) and y(0)) or ((not y(0)) and b(1))) and c;
z(2) <= ((b(3) and y(0)) or ((not y(0)) and b(2))) and c;
z(3) <= ((b(4) and y(0)) or ((not y(0)) and b(3))) and c;
z(4) <= ((b(5) and y(0)) or ((not y(0)) and b(4))) and c;
z(5) <= ((b(6) and y(0)) or ((not y(0)) and b(5))) and c;
z(6) <= ((b(7) and y(0)) or ((not y(0)) and b(6))) and c;
z(7) <= (((not y(0)) and b(7))) and c;

end behave;
```

Register Transfer Level Results-



```
/testbench/input_...   1111111111...
  (15) 1
  (14) 1
  (13) 1
  (12) 1
  (11) 1
  (10) 1
  (9) 1
  (8) 1
  (7) 1
  (6) 1
  (5) 1
  (4) 1
  (3) 1
  (2) 1
  (1) 1
  (0) 1
/testbench/output...   00000000
  (7) 0
  (6) 0
  (5) 0
  (4) 0
  (3) 0
  (2) 0
  (1) 0
  (0) 0
/testbench/std_ou...   00000000
  (7) 0
  (6) 0
  (5) 0
  (4) 0
  (3) 0
  (2) 0
  (1) 0
  (0) 0
Now        327680000 ps
Cursor 1   0 ps
```

Gate Level Simulation Results-



```
/testbench/input_...   1000001100...
  (15) 1
  (14) 0
  (13) 0
  (12) 0
  (11) 0
  (10) 0
  (9) 1
  (8) 1
  (7) 0
  (6) 0
  (5) 0
  (4) 0
  (3) 0
  (2) 1
  (1) 1
  (0) 0
/testbench/output...   00000010
  (7) 0
  (6) 0
  (5) 0
  (4) 0
  (3) 0
  (2) 0
  (1) 1
  (0) 0
/testbench/std_ou...   00000010
  (7) 0
  (6) 0
  (5) 0
  (4) 0
  (3) 0
  (2) 0
  (1) 1
  (0) 0
Now        327680000 ps
Cursor 1   167710000 ps
```

Arithmetic Logic Unit:

alu.vhd

```
----------------------------------------------
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------
entity alu is
        port( X,Y : in std_logic_vector(7 downto 0); x0,x1 : in std_logic ; Z : out
std_logic_vector(7 downto 0));
end entity;

architecture behave of alu is
        signal sig1,sig2,sig3,sig4 : std_logic_vector(7 downto 0);

        component eightbitadder is
                port(x,y:in std_logic_vector(7 downto 0); z:out std_logic_vector(7
downto 0));
        end component;
        --- Add component of eightbitsubtractor

        component leftshift is
                port( x,y : in std_logic_vector(7 downto 0);
             z: out std_logic_vector(7 downto 0));
        end component;
        --- Add component of rightshift
        component rightshift is
                port( x,y : in std_logic_vector(7 downto 0);
             z: out std_logic_vector(7 downto 0));
        end component;

        component eightbitsubtractor is
```

```
                port(x,y:in std_logic_vector(7 downto 0); z:out std_logic_vector(7
downto 0));
    end component;
----------------------------------------------
begin
a: eightbitadder      port map(x => X, y => Y, z => sig1);
b: leftshift        port map(x => X, y => Y, z => sig4);
c: rightshift         port map(x => X, y => Y, z => sig3);
d: eightbitsubtractor  port map(x => X, y => Y, z => sig2);
----------------------------------------------

process(x0, x1,sig1, sig2, sig3, sig4)
begin
------------------------------
if (x0 = '0' and x1 = '0') then
z<= sig1;
elsif(x0 = '1') and (x1 = '0') then
z<= sig2;
elsif(x0 = '0') and (x1 = '1') then
z<= sig3;
else
z<= sig4;
end if;
------------------------------
end process;

end behave;

-----------------Eight Bit Adder-----------------------------------
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;

entity eightbitadder is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;
```

```
architecture behave of eightbitadder is
        component onebitadder is
        port(x,y,cin : in std_logic;
        z,cout : out std_logic);
        end component;
        signal c:std_logic_vector(7 downto 0);
begin

 adder0:onebitadder port map(x=>x(0),y=>y(0),cin=>'0',z=>z(0),cout=>c(0));
 adder1:onebitadder port map(x=>x(1),y=>y(1),cin=>c(0),z=>z(1),cout=>c(1));
 adder2:onebitadder port map(x=>x(2),y=>y(2),cin=>c(1),z=>z(2),cout=>c(2));
 adder3:onebitadder port map(x=>x(3),y=>y(3),cin=>c(2),z=>z(3),cout=>c(3));
 adder4:onebitadder port map(x=>x(4),y=>y(4),cin=>c(3),z=>z(4),cout=>c(4));
 adder5:onebitadder port map(x=>x(5),y=>y(5),cin=>c(4),z=>z(5),cout=>c(5));
 adder6:onebitadder port map(x=>x(6),y=>y(6),cin=>c(5),z=>z(6),cout=>c(6));
 adder7:onebitadder port map(x=>x(7),y=>y(7),cin=>c(6),z=>z(7),cout=>c(7));
end behave;


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

entity onebitadder is
port (x,y,cin: in std_logic;
                z,cout:out std_logic);
end entity;

architecture behave of onebitadder is
begin
        z <= (cin xor (x xor y));
        cout <= ((cin and (x xor y)) or (x and y));
end behave;

-----------------------------------------------------------------

--------------------------------Left Shifter-------------------------
```

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

entity leftshift is
port (x,y: in std_logic_vector(7 downto 0);
                z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of leftshift is
        signal a, b:std_logic_vector(7 downto 0);
        signal c:std_logic;
begin
c <= (not (y(7) or y(6) or y(5) or y(4) or y(3)));

a(7) <= (x(3) and y(2)) or ((not y(2)) and x(7));
a(6) <= (x(2) and y(2)) or ((not y(2)) and x(6));
a(5) <= (x(1) and y(2)) or ((not y(2)) and x(5));
a(4) <= (x(0) and y(2)) or ((not y(2)) and x(4));
a(3) <= ((not y(2)) and x(3));
a(2) <= ((not y(2)) and x(2));
a(1) <= ((not y(2)) and x(1));
a(0) <= ((not y(2)) and x(0));

b(7) <= (a(5) and y(1)) or ((not y(1)) and a(7));
b(6) <= (a(4) and y(1)) or ((not y(1)) and a(6));
b(5) <= (a(3) and y(1)) or ((not y(1)) and a(5));
b(4) <= (a(2) and y(1)) or ((not y(1)) and a(4));
b(3) <= (a(1) and y(1)) or ((not y(1)) and a(3));
b(2) <= (a(0) and y(1)) or ((not y(1)) and a(2));
b(1) <= ((not y(1)) and a(1));
b(0) <= ((not y(1)) and a(0));

z(7) <= ((b(6) and y(0)) or ((not y(0)) and b(7))) and c;
z(6) <= ((b(5) and y(0)) or ((not y(0)) and b(6))) and c;
z(5) <= ((b(4) and y(0)) or ((not y(0)) and b(5))) and c;
z(4) <= ((b(3) and y(0)) or ((not y(0)) and b(4))) and c;
z(3) <= ((b(2) and y(0)) or ((not y(0)) and b(3))) and c;
```

```
z(2) <= ((b(1) and y(0)) or ((not y(0)) and b(2))) and c;
z(1) <= ((b(0) and y(0)) or ((not y(0)) and b(1))) and c;
z(0) <= (((not y(0)) and b(0))) and c;

end behave;
---------------------------------------------------------------------
-------------------------------Right Shifter---------------------------


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

entity rightshift is
port (x,y: in std_logic_vector(7 downto 0);
              z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of rightshift is
        signal a, b:std_logic_vector(7 downto 0);
        signal c:std_logic;
begin
c <= (not (y(7) or y(6) or y(5) or y(4) or y(3)));

a(0) <= (x(4) and y(2)) or ((not y(2)) and x(0));
a(1) <= (x(5) and y(2)) or ((not y(2)) and x(1));
a(2) <= (x(6) and y(2)) or ((not y(2)) and x(2));
a(3) <= (x(7) and y(2)) or ((not y(2)) and x(3));
a(4) <= ((not y(2)) and x(4));
a(5) <= ((not y(2)) and x(5));
a(6) <= ((not y(2)) and x(6));
a(7) <= ((not y(2)) and x(7));

b(0) <= (a(2) and y(1)) or ((not y(1)) and a(0));
b(1) <= (a(3) and y(1)) or ((not y(1)) and a(1));
b(2) <= (a(4) and y(1)) or ((not y(1)) and a(2));
b(3) <= (a(5) and y(1)) or ((not y(1)) and a(3));
b(4) <= (a(6) and y(1)) or ((not y(1)) and a(4));
```

```
b(5) <= (a(7) and y(1)) or ((not y(1)) and a(5));
b(6) <= ((not y(1)) and a(6));
b(7) <= ((not y(1)) and a(7));

z(0) <= ((b(1) and y(0)) or ((not y(0)) and b(0))) and c;
z(1) <= ((b(2) and y(0)) or ((not y(0)) and b(1))) and c;
z(2) <= ((b(3) and y(0)) or ((not y(0)) and b(2))) and c;
z(3) <= ((b(4) and y(0)) or ((not y(0)) and b(3))) and c;
z(4) <= ((b(5) and y(0)) or ((not y(0)) and b(4))) and c;
z(5) <= ((b(6) and y(0)) or ((not y(0)) and b(5))) and c;
z(6) <= ((b(7) and y(0)) or ((not y(0)) and b(6))) and c;
z(7) <= (((not y(0)) and b(7))) and c;

end behave;

---------------------------Eight Bit Subtractor--------------------------------


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

entity eightbitsubtractor is
port (x,y: in std_logic_vector(7 downto 0);
              z:out std_logic_vector(7 downto 0));
end entity;

architecture behave of eightbitsubtractor is
        component onebitsubtractor is
        port(x,y,cin : in std_logic;
        z,cout : out std_logic);
        end component;
        signal c:std_logic_vector(7 downto 0);
begin

 subtractor0:onebitsubtractor port
map(x=>x(0),y=>y(0),cin=>'0',z=>z(0),cout=>c(0));
```

```
 subtractor1:onebitsubtractor port
map(x=>x(1),y=>y(1),cin=>c(0),z=>z(1),cout=>c(1));
 subtractor2:onebitsubtractor port
map(x=>x(2),y=>y(2),cin=>c(1),z=>z(2),cout=>c(2));
 subtractor3:onebitsubtractor port
map(x=>x(3),y=>y(3),cin=>c(2),z=>z(3),cout=>c(3));
 subtractor4:onebitsubtractor port
map(x=>x(4),y=>y(4),cin=>c(3),z=>z(4),cout=>c(4));
 subtractor5:onebitsubtractor port
map(x=>x(5),y=>y(5),cin=>c(4),z=>z(5),cout=>c(5));
 subtractor6:onebitsubtractor port
map(x=>x(6),y=>y(6),cin=>c(5),z=>z(6),cout=>c(6));
 subtractor7:onebitsubtractor port
map(x=>x(7),y=>y(7),cin=>c(6),z=>z(7),cout=>c(7));
end behave;



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

entity onebitsubtractor is
port (x,y,cin: in std_logic;
                z,cout:out std_logic);
end entity;

architecture behave of onebitsubtractor is
begin
        z <= (cin xor (x xor y));
        cout <= ((not x) and cin) or ((not x) and y) or (cin and y);
end behave;
```

Testbench.vhd

```
library std;
use std.textio.all;
```

```
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity Testbench is
end entity;
architecture Behave of Testbench is

  ---------------------------------------------------------------
  --  edit the following lines to set the number of i/o's of your
  --  DUT.
  ---------------------------------------------------------------
  constant number_of_inputs  : integer := 18;  -- # input bits to your design.
  constant number_of_outputs : integer := 8;  -- # output bits from your design.

  -- component port widths..
  component DUT is
   port(input_vector: in std_logic_vector(number_of_inputs-1  downto 0);
        output_vector: out std_logic_vector(number_of_outputs-1 downto 0));
  end component;

  -- end editing.
  ---------------------------------------------------------------
  ---------------------------------------------------------------

  signal input_vector  : bit_vector(number_of_inputs-1 downto 0);
  signal output_vector : bit_vector(number_of_outputs-1 downto 0);
  signal std_output_vector : std_logic_vector(number_of_outputs-1 downto 0);

  -- create a constrained string outof
  function to_string(x: string) return string is
     variable ret_val: string(1 to x'length);
     alias lx : string (1 to x'length) is x;
  begin
     ret_val := lx;
     return(ret_val);
  end to_string;

begin
  process
    variable err_flag : boolean := false;
    File INFILE: text open read_mode is
"/home/saqib1707/Acads/4thsem/EE214_Digital_Circuits_Lab/ALU/ALU_exp/sr
c/alu_TRACEFILE.txt";
    FILE OUTFILE: text  open write_mode is
"/home/saqib1707/Acads/4thsem/EE214_Digital_Circuits_Lab/ALU/ALU_exp/sr
c/OUTPUTS.txt";

    -------------------------------------------------
    -- edit the next two lines to customize
    -------------------------------------------------
    variable input_vector_var: bit_vector (number_of_inputs-1 downto 0);
    variable output_vector_var: bit_vector (number_of_outputs-1 downto 0);
    variable output_mask_var: bit_vector (number_of_outputs-1 downto 0);
    variable output_comp_var: bit_vector (number_of_outputs-1 downto 0);
    constant ZZZZ : bit_vector(number_of_outputs-1 downto 0) := (others => '0');
    --------------------------------------------------

    variable INPUT_LINE: Line;
    variable OUTPUT_LINE: Line;
    variable LINE_COUNT: integer := 0;


  begin
    while not endfile(INFILE) loop
          -- will read a new line every 5ns, apply input,
          -- wait for 1 ns for circuit to settle.
          -- read output.



        LINE_COUNT := LINE_COUNT + 1;



          -- read input at current time.
          readLine (INFILE, INPUT_LINE);
          read (INPUT_LINE, input_vector_var);
          read (INPUT_LINE, output_vector_var);
          read (INPUT_LINE, output_mask_var);

           -- apply input.
          input_vector <= input_vector_var;

           -- wait for the circuit to settle
           wait for 1 ns;

           -- check output.
          output_comp_var := (output_mask_var and (output_vector xor
output_vector_var));
           if (output_comp_var  /= ZZZZ) then
            write(OUTPUT_LINE,to_string("ERROR: line "));
            write(OUTPUT_LINE, LINE_COUNT);
            writeline(OUTFILE, OUTPUT_LINE);
            err_flag := true;
          end if;

          write(OUTPUT_LINE, input_vector);
          write(OUTPUT_LINE, to_string(" "));
          write(OUTPUT_LINE, output_vector);
          writeline(OUTFILE, OUTPUT_LINE);

           -- advance time by 4 ns.
           wait for 4 ns;
    end loop;

    assert (err_flag) report "SUCCESS, all tests passed." severity note;
    assert (not err_flag) report "FAILURE, some tests failed." severity error;

    wait;
  end process;

         output_vector <= to_bitvector(std_output_vector);
  dut_instance: DUT
         port map(input_vector => to_stdlogicvector(input_vector), output_vector
=>std_output_vector );
```

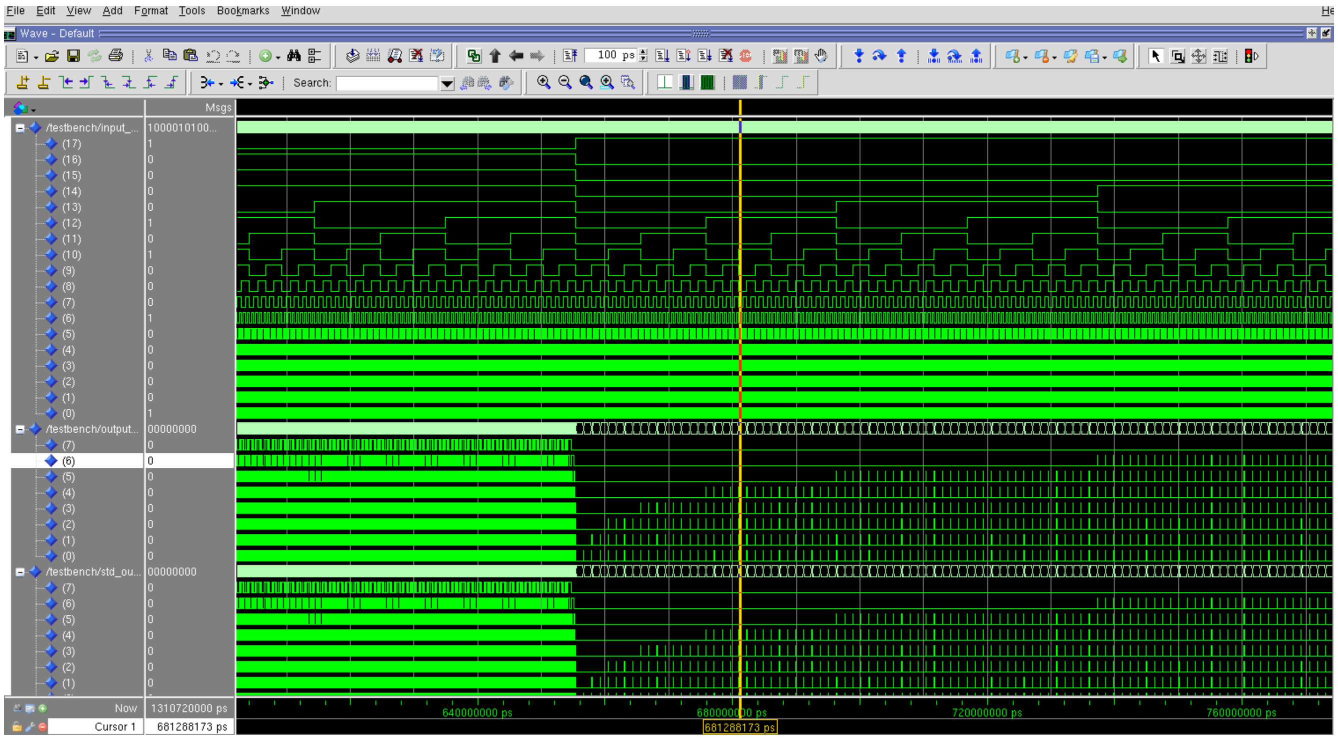
```
end Behave;
```

DUT.vhd

```
-- A DUT entity is used to wrap your design.
--  This example shows how you can do this for the
--  eight-bit adder.
library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity DUT is
   port(input_vector: in std_logic_vector(17 downto 0); ---Note: for alu testing (17
downto 0) for others (15 downto 0)
        output_vector: out std_logic_vector(7 downto 0));
end entity;

architecture DutWrap of DUT is

        --- Add component of eightbitadder
        --- Add component of eightbitsubtractor
        --- Add component of leftshift
        --- Add component of rightshift
        component alu is
        port( X,Y : in std_logic_vector(7 downto 0); x0,x1 : in std_logic ; Z : out
std_logic_vector(7 downto 0));
        end component;

begin

   -- input/output vector element ordering is critical,
   -- and must match the ordering in the trace file!
--a: eightbitadder port map(x => input_vector(15 downto 8), y =>input_vector(7
downto 0) , z=> output_vector);
--b: leftshift              port map(x => input_vector(15 downto 8), y
=>input_vector(7 downto 0) , z=> output_vector);
```

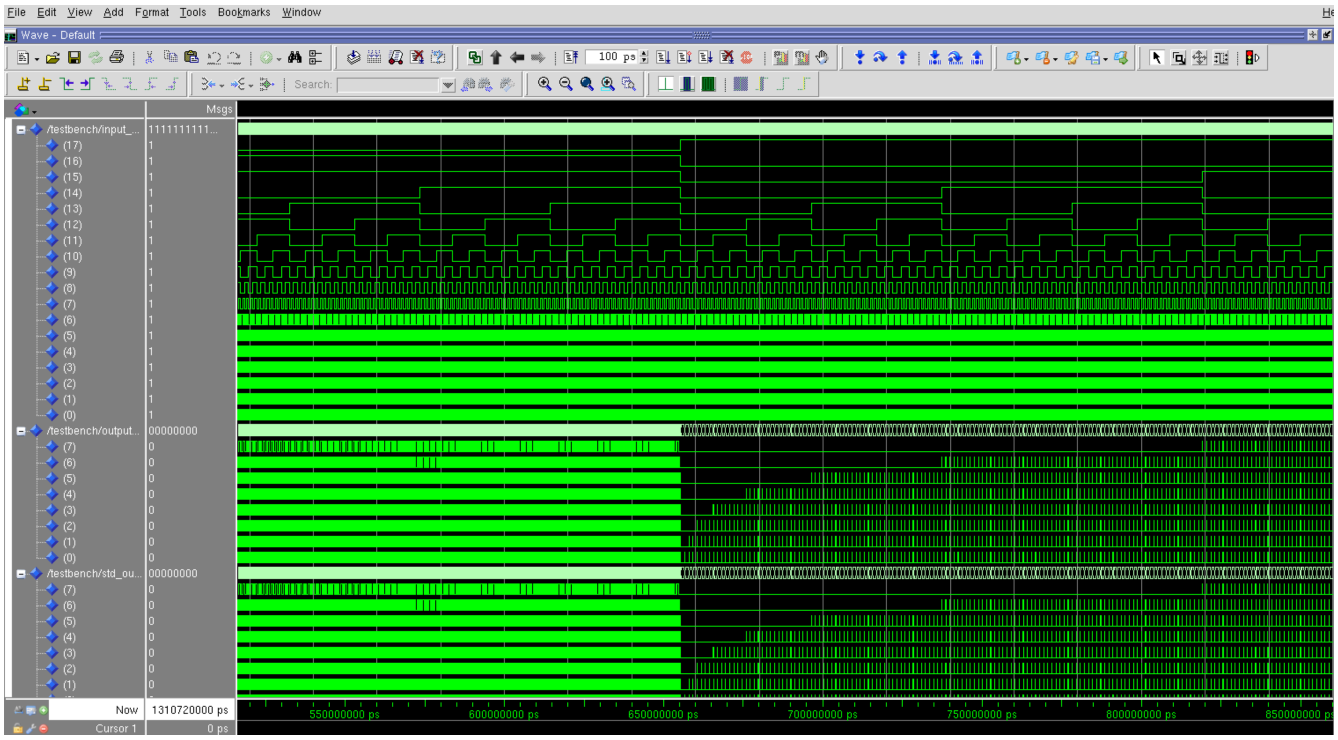
--c: rightshift port map(x => input_vector(15 downto 8), y =>input_vector(7 downto 0) , z=> output_vector);
--d: eightbitsubtractor port map(x => input_vector(15 downto 8), y =>input_vector(7 downto 0) , z=> output_vector); --- Note: z = x- y
add_instance: alu port map(X => input_vector(15 downto 8), Y => input_vector(7 downto 0) , x0 => input_vector(16) , x1 => input_vector(17), Z => output_vector);
end DutWrap;

Register Transfer Level Results-



	Msgs
/testbench/input_...	1000010100...
(17)	1
(16)	0
(15)	0
(14)	0
(13)	0
(12)	1
(11)	0
(10)	1
(9)	0
(8)	0
(7)	0
(6)	1
(5)	0
(4)	0
(3)	0
(2)	0
(1)	0
(0)	1
/testbench/output...	00000000
(7)	0
(6)	0
(5)	0
(4)	0
(3)	0
(2)	0
(1)	0
(0)	0
/testbench/std_ou...	00000000
(7)	0
(6)	0
(5)	0
(4)	0
(3)	0
(2)	0
(1)	0
Now	1310720000 ps
Cursor 1	681288173 ps

Gate Level Simulation Results-


File Edit View Add Format Tools Bookmarks Window
Wave - Default
100 ps
Search:
Msgs
/testbench/input_... 1111111111...
(17) 1
(16) 1
(15) 1
(14) 1
(13) 1
(12) 1
(11) 1
(10) 1
(9) 1
(8) 1
(7) 1
(6) 1
(5) 1
(4) 1
(3) 1
(2) 1
(1) 1
(0) 1
/testbench/output... 00000000
(7) 0
(6) 0
(5) 0
(4) 0
(3) 0
(2) 0
(1) 0
(0) 0
/testbench/std_ou... 00000000
(7) 0
(6) 0
(5) 0
(4) 0
(3) 0
(2) 0
(1) 0
Now 1310720000 ps
Cursor 1 0 ps
550000000 ps
600000000 ps
650000000 ps
700000000 ps
750000000 ps
800000000 ps
850000000 ps