ALU

Use following entity definitions for your design.
Top Entity(ALU):

```
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
--...
begin
--...
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
--...
begin
--...
end behave;
```

ALU

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
--...
Begin
--...
end behave;
```

Eight bit Right Shift:

```
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
--...
Begin
--...
end behave;
```

ALU

Eight bit Left Shift:

```
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
--...
Begin
--...
end behave;
```

ALU

DUT File:

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

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.components.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
--components declarations for
alu,eightbitadder,eightbitsubtractor,leftshift and rightshift .
--...
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

--dut: 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;
```

ALU

Use following TRACEFILES to test your design:

```
1)Eight Bit Adder:            adder_TRACEFILE.txt
 (Input size 16 bits( x(15 to 8), y(7 to 0)), output size 8bits(z))
2)Eight Bit Subtractor:       sub_TRACEFILE.txt
 (Input size 16 bits( x(15 to 8), y(7 to 0)), output size 8bits(z))
3)Right Shift:                rightshift_TRACEFILE.txt
 (Input size 16 bits( x(15 to 8), y(7 to 0)), output size 8bits(z))
4)Left Shift                  leftshift_TRACEFILE.txt
 (Input size 16 bits( x(15 to 8), y(7 to 0)), output size 8bits(z))
5)ALU(Final Design)           alu_TRACEFILE.txt
 (Input size 18 bits(x1(17),x0(16),x(15 to 8), y(7 to 0)), output size
8bits(z))
```

For ALU tracefile:

Op x y z Mask

00000000**1**110111101 11000000 11111111

For Others:

x y z Mask

0000000**1**110111101 11000000 11111111

Updated Testbench file for std logic is in zip folder by name Testbench.vhd.

You can find templates for your design in following files.

DUT.vhd, alu.vhd.