Lab 8: 32-bit ALU and Testbench

Lina Mi

ID:@01377283

Purpose:

1) Get familiar with ALU and Understand the ALU function
2) Be able to programming with VHDL to implement design of the ALU.
3) Understand the efficiency of VHDL encoding in hardware design.
4) Be able to write VHDL testbench to test the correctness of the ALU
5) Use ModelSim software to simulate the designed ALU, and obtain the simulation results (simulation waves).

Content:

1) ALU Symbol and Operations Table :

The symbol of ALU to design is shown as following:

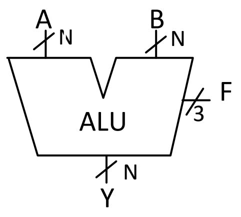


Figure 1 Symbol of ALU

The ALU to design has the operations as following table

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	Not used
100	A and $\bar{B}$
101	A OR $\bar{B}$
110	A - B
111	SLT

Table 1 ALU Operations

2) VHDL code

- The VHDL code used to design ALU is shown in the following VHD file:

ALU.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.TEXTIO.all;

entity alu is


port(a,b: in std_logic_vector(31 downto 0);
        f:      in std_logic_vector(2 downto 0);
        y:      out std_logic_vector(31 downto 0);
        zero: out std_logic
       );
end entity;


architecture sim of alu is
signal y1: std_logic_vector(31 downto 0);
 begin
   y<= y1;
process(f, a, b) begin
  case f is
    when "000" =>
      y1 <= a and b;
    when "001" =>
      y1 <= a or b;
    when "010" =>
       y1 <= std_logic_vector(signed(a) + signed(b));
```

```
     when "100" =>
       y1 <= a and not b;


    when "101" =>
       y1 <= a or not b;
    when "110" =>
       y1 <= std_logic_vector(signed(a) - signed(b));
    when "111"=>
        if (signed(a) < signed(b)) then
                              y1 <= "00000000000000000000000000000001";
        else
                             y1 <= "00000000000000000000000000000000";
      end if;
   when others =>
         y1 <= a;


end case;


if(y1 = "00000000000000000000000000000000")then
 zero<= '1';
else
zero<= '0';
end if;
end process;
end ;
```

3) Simulation and Testing

The following table shows the input vectors that used to feed the ALU.VHD fill to test the correctness of design.

Test	F[2:0]	A	B	Y	Zero
ADD 0+0	2	00000000	00000000	00000000	1
ADD 0+(-1)	2	00000000	FFFFFFFF	FFFFFFFF	0
ADD 1+(-1)	2	00000001	FFFFFFFF	00000000	1
ADD FF+1	2	000000FF	00000001	00000100	0
SUB 0-0	6	00000000	00000000	00000000	1
SUB 0-(-1)	6	00000000	FFFFFFFF	00000001	0
SUB 1-1	6	00000001	00000001	00000000	1
SUB 100-1	6	00000100	00000001	000000FF	0
SLT 0,0	7	00000000	00000000	00000000	1
SLT 0,1	7	00000000	00000001	00000001	0
SLT 0,-1	7	00000000	FFFFFFFF	00000000	1
SLT 1,0	7	00000001	00000000	00000000	1
SLT -1,0	7	FFFFFFFF	00000000	00000001	0
AND FFFFFFFF,FFFFFFFF	0	FFFFFFFF	FFFFFFFF	FFFFFFFF	0
AND FFFFFFFF,12345678	0	FFFFFFFF	12345678	12345678	0
AND 12345678,87654321	0	12345678	87654321	02244220	0
AND 00000000,FFFFFFFF	0	00000000	FFFFFFFF	00000000	1
OR FFFFFFFF,FFFFFFFF	1	FFFFFFFF	FFFFFFFF	FFFFFFFF	0
OR 12345678,87654321	1	12345678	87654321	97755779	0
OR 00000000,FFFFFFFF	1	00000000	FFFFFFFF	FFFFFFFF	0
OR 00000000, 00000000	1	00000000	00000000	00000000	1

Table 2 Test Vectors (values are expressed in heximal)

The following VHB fill is used to implement the test on designed ALU

TESTBENCH.VHD

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_TEXTIO.ALL;

use STD.TEXTIO.all;


entity testbench3 is

end;


architecture sim of testbench3 is
```

```
 component alu
  port (a,b: in std_logic_vector(31 downto 0);
        f:     in std_logic_vector(2 downto 0);
        y:     out std_logic_vector(31 downto 0);
        zero:  out std_logic
       );
       end component;
 signal a,b,y: std_logic_vector(31 downto 0);
 signal f: std_logic_vector(2 downto 0);
 signal zero: std_logic;
 signal y_expected:     std_logic_vector(31 downto 0);
 signal zero_expected:  std_logic;
 signal clk, reset:     std_logic;

begin
-- instantiate device under test
 dut:   alu port map(a,b,f,y,zero);

-- generate clock
 process begin
   clk <= '1' ; wait for 5 ns;
   clk <= '0' ; wait for 5 ns;
 end process;


 -- at start of test, pulse reset
 process begin
   reset <= '1';  wait for 27 ns; reset <= '0';
   wait;
 end process;
```

```
-- run tests
process is
  file tv: text;
  variable L: line;
  variable vector_a: std_logic_vector(31 downto 0);
  variable dummy1:   character;
  variable vector_b: std_logic_vector(31 downto 0);
  variable dummy2:   character;
  variable vector_f: std_logic_vector(3 downto 0);
  variable dummy3:   character;
  variable vector_y_ex: std_logic_vector(31 downto 0);
  variable dummy4:   character;
  variable vector_z_ex: std_logic_vector(3 downto 0);
  variable vectornum: integer := 0;
  variable errors: integer := 0;
begin
  FILE_OPEN(tv, "J:\Lab8\alu.tv", READ_MODE);
  while not endfile(tv) loop

        -- change vectors on rising edge
        wait until rising_edge(clk);

        -- read the next line of testvectors split it up
        readline(tv, L);
        hread(L,vector_f);
        read(L, dummy3);
        hread(L, vector_a);
        read(L, dummy1);
```

```
            hread(L,vector_b);
            read(L, dummy2);
            hread(L,vector_y_ex);
            read(L, dummy4);
            hread(L,vector_z_ex);
            a <= vector_a after 1 ns;
            b <= vector_b;
            f <= vector_f(2 downto 0);
            y_expected <= vector_y_ex;
            zero_expected <= vector_z_ex(0);


            wait until falling_edge(clk);
            if y /= y_expected then
              report("Error with line: " & integer'image(vectornum));
             -- report string'image(L);
              report("zero = " & std_logic'image(zero));
             -- report("Error: y=" & std_logic_vector'image(y));
             -- report("Error: a=" & std_logic_vector'image(a));
             -- report("Error: b=" & std_logic'image(b));
              --report("Error: f=" & std_logic'image(f));
              errors := errors + 1;
            end if;
            vectornum := vectornum + 1;
            end loop;

    if errors=0 then

      report "NO ERRORS" & integer'image(vectornum)  & "tests completed" severity failure;
```

```
  else

report integer'image(errors) & " ERRORS in " &

    integer'image(vectornum) & "tests" severity failure;

  end if;

        end process;
end;
```

the input test vectors are listed in the following alu.tv file:

```
alu.tv
2_00000000_00000000_00000000_1
2_00000000_FFFFFFFF_FFFFFFFF_0
2_00000001_FFFFFFFF_00000000_1
2_000000FF_00000001_00000100_0
6_00000000_00000000_00000000_1
6_00000000_FFFFFFFF_00000001_0
6_00000001_00000001_00000000_1
6_00000100_00000001_000000FF_0
7_00000000_00000000_00000000_1
7_00000000_00000001_00000001_0
7_00000000_FFFFFFFF_00000000_1
7_00000001_00000000_00000000_1
7_FFFFFFFF_00000000_00000001_0
0_ FFFFFFFF _ FFFFFFFF _ FFFFFFFF _0
```

0_ FFFFFFFF _12345678_12345678_0

0_ 12345678 _87654321_02244220_0

0_ 00000000 _FFFFFFFF_00000000_1

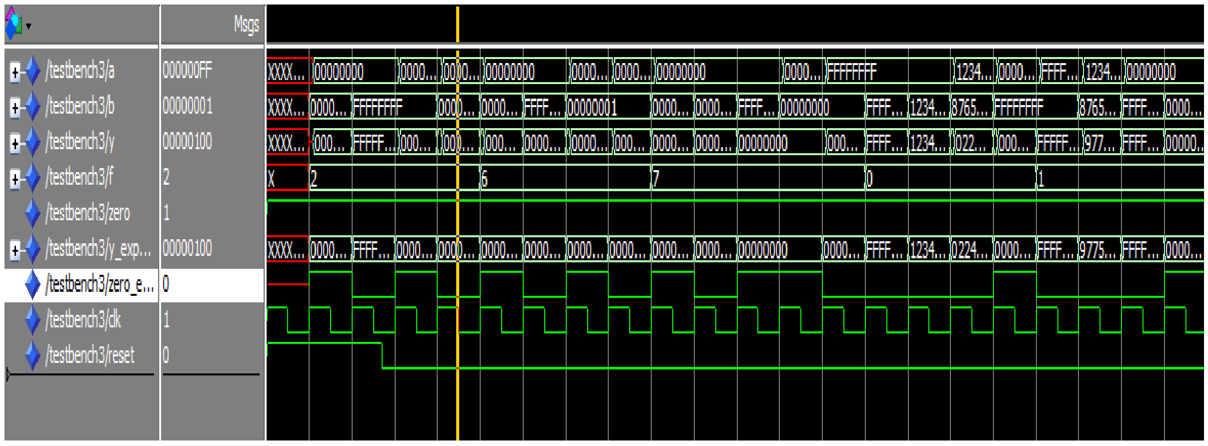
1_ FFFFFFFF _FFFFFFFF_FFFFFFFF_0

1_ 12345678 _87654321_97755779_0

1_00000000_FFFFFFFF_FFFFFFFF_0

1_00000000_00000000_00000000_1

4) Simulation Result



In the figure above, the waves of output signals Y and Zero are “y_export” and “zero_export”. Comparing the table of test vectors with the simulation waves, it can be seen that the output signals Y and Zero are correct under different combination of input signals f, a, and b. That means that the designed ALU has correct operations and the design is successful.

This lab cost my lab partners and me about 4 hours.