Lab 3 Tutorial

32-bit ALU Design

OVERVIEW

In this lab we will implement, test, and simulate a 32-bit Arithmetic Logic Unit (ALU) capable of performing the following operations:

Operation Name	ALU-Op		Operation Performed
	Neg/Tsel	*ALU-Select*	
AND (Logical)	0	00	Result <= a AND b
Or (Logical)	0	01	Result <= a OR b
ADD	0	10	Result <= a + b
SUB	1	10	Result <= a – b
ROL	1	00	Result <= a << 1
ROR	1	01	Result <= a >> 1

As per the lab specifications outlined in the lab manual:

"The addition and subtraction operations should be performed using a structural approach ***(i.e, A+B and A-B VHDL statements are not acceptable).***"

Please refer to the lab manual for more information on this.

We will design the following components to complete this lab:

1. 1-bit full adder.
2. 4-bit adder built using four 1-bit full adders.
3. 16-bit adder built using four 4-bit adders.

4. 32-bit adder/subtractor built using two 16-bit adders. We will use this 32-bit adder/subtractor to implement the addition and subtraction operations performed by the 32-bit ALU.
5. 32-bit ALU.

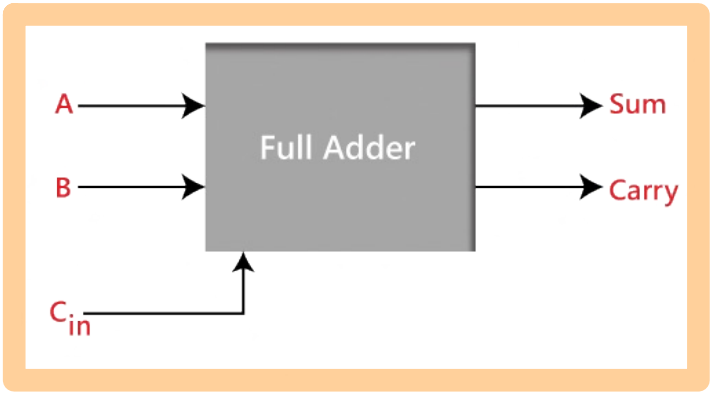
   Only the simulation waveforms for the 32-bit ALU will be provided in this tutorial.

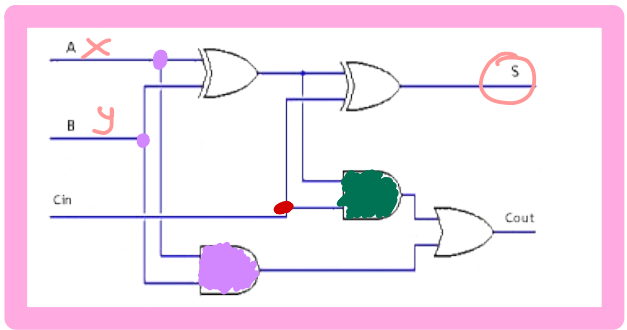
   MAKE SURE THAT YOU CHOOSE THE "EP4CE115F29C7" DEVICE IN THE PROJECT WIZARD.

PROCEDURE

1. 1-bit Full Adder (fulladd.vhd)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity fulladd is
6   port(
7      Cin, x, y : in  std_logic;
8      s, Cout   : out std_logic
9      );
10  end fulladd;
11
12  architecture Behavior of fulladd is
13  begin
14    s <= x xor y xor Cin;
15    Cout <= (x and y) or (Cin and x) or (Cin and y);
16  end Behavior;
17
```





equivalent

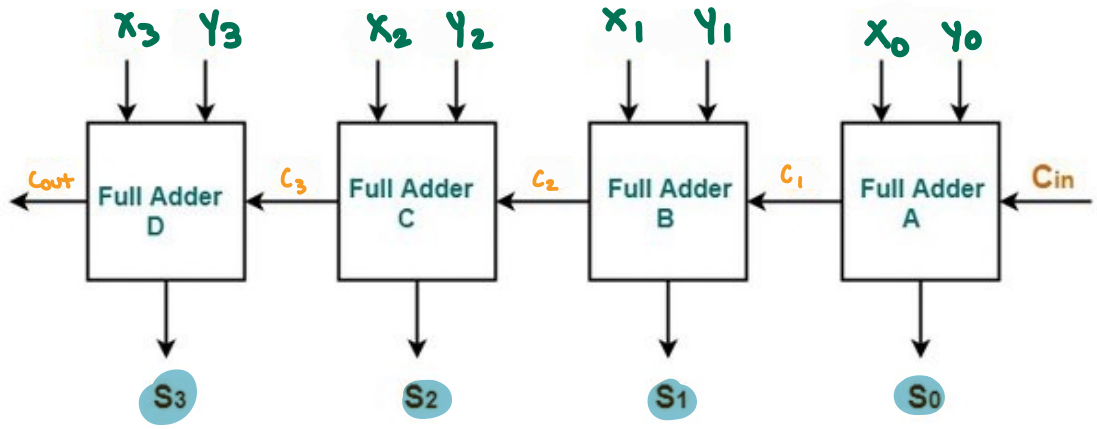
((x xor y) and cin) or (x and y)

2. 4-bit Adder (adder4.vhd)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity adder4 is
6   port(
7      Cin      :  in std_logic;
8      X,Y      :  in  std_logic_vector(3 downto 0);
9      S     :  out   std_logic_vector(3 downto 0);
10     Cout  :  out   std_logic
11     );
12  end adder4;
13
14  architecture Behavior of adder4 is
15     component fulladd
16     port(
17     Cin, x, y : in  std_logic;
18     s, Cout   : out std_logic
19     );
20     end component;
21
22     signal C : std_logic_vector (1 to 3);
23  begin
24     stage0:  fulladd port map (Cin,  X(0), Y(0), S(0), C(1));
25     stage1:  fulladd port map (C(1), X(1), Y(1), S(1), C(2));
26     stage2:  fulladd port map (C(2), X(2), Y(2), S(2), C(3));
27     stage3:  fulladd port map (C(3), X(3), Y(3), S(3), Cout);
28  end Behavior;
29
```

Portmap goes in order (Cin, X, Y, S, Cout)

Taking component 1 bit full adder (Prev slide) to make 4-bit full adder



4-bit Ripple Carry Adder

3. 16-bit Adder (adder16.vhd)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity adder16 is
6   port(
7      Cin      :  in std_logic;
8      X,Y      :  in  std_logic_vector(15 downto 0);
9      S     :  out   std_logic_vector(15 downto 0);
10     Cout  :  out   std_logic
11     );
12  end adder16;
13
14  architecture Behavior of adder16 is
15     component adder4
16     port(
17     Cin      :  in std_logic;
18     X,Y      :  in  std_logic_vector(3 downto 0);
19     S     :  out   std_logic_vector(3 downto 0);
20     Cout  :  out   std_logic
21     );
22     end component;
23
24     signal C : std_logic_vector (1 to 3);
25  begin
26     stage0:  adder4 port map (Cin, X(3  downto  0), Y(3 downto   0), S(3 downto   0), C(1
   ));
27     stage1:  adder4 port map (C(1), X(7  downto  4), Y(7 downto   4), S(7 downto   4), C(2
   ));
28     stage2:  adder4 port map (C(2), X(11 downto  8), Y(11 downto  8), S(11 downto  8), C(3
   ));
29     stage3:  adder4 port map (C(3), X(15 downto 12), Y(15 downto 12), S(15 downto 12),
   Cout);
30  end Behavior;
31
```

4. 32-bit Adder/Subtractor (adder32.vhd)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity adder32 is
6   port(
7      Cin      :  in std_logic;
8      X,Y      :  in  std_logic_vector(31 downto 0);
9      S     :  out   std_logic_vector(31 downto 0);
10     Cout  :  out   std_logic
11     );
12  end adder32;
13
14  architecture Behavior of adder32 is
15     component adder16
16     port(
17     Cin      :  in std_logic;
18     X,Y      :  in  std_logic_vector(15 downto 0);
19     S     :  out   std_logic_vector(15 downto 0);
20     Cout  :  out   std_logic
21     );
22     end component;
23
24     signal C : std_logic;
25  begin
26     stage0:  adder16 port map (Cin,   X(15  downto  0), Y(15 downto   0), S(15 downto   0),
       C);
27     stage1:  adder16 port map (C, X(31 downto 16), Y(31 downto 16), S(31 downto 16), Cout);
28  end Behavior;
29
```

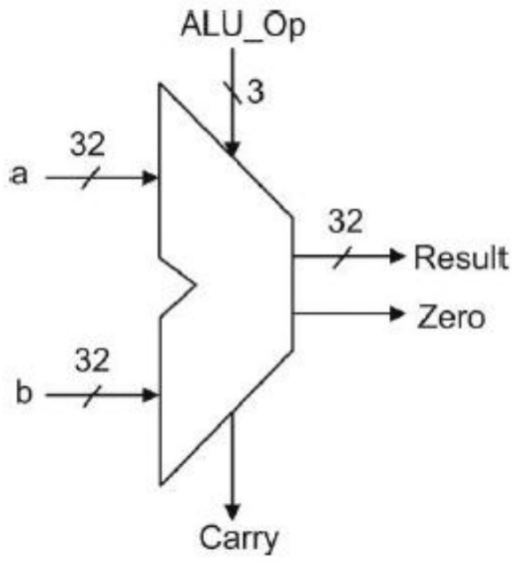


Figure 1: 32-bit ALU

5. 32-bit ALU (alu.vhd)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5   use ieee.numeric_std.all;
6
7   entity alu is
8   port(
9      a      :  in  std_logic_vector(31 downto 0);
10     b      :  in  std_logic_vector(31 downto 0);
11     op     :  in  std_logic_vector(2 downto 0);
12     result :  out  std_logic_vector(31 downto 0);
13     zero   :  out std_logic;
14     cout   :  out std_logic
15     );
16  end alu;
17
18  architecture Behavior of alu is
19     component adder32
20           port(
21              Cin      :  in std_logic;
22              X,Y      :  in  std_logic_vector(31 downto 0);
23              S        :  out   std_logic_vector(31 downto 0);
24              Cout     :  out   std_logic
25           );
26     end component;
27
28     signal result_s: std_logic_vector(31 downto 0):= (others => '0');
29     signal result_add: std_logic_vector(31 downto 0):= (others => '0');
30     signal result_sub:  std_logic_vector(31 downto 0):= (others => '0');
31     signal cout_s : std_logic := '0';
32     signal cout_add : std_logic := '0';
33     signal cout_sub : std_logic := '0';
34     signal zero_s : std_logic;
35
36  begin
37        add0 : adder32 port map (op(2), a, b, result_add, cout_add);
38        sub0 : adder32 port map (op(2), a, not b, result_sub, cout_sub);
39
40     process (a, b, op)
41     begin
42        case (op) is
43           when "000" =>           -- "000" a and b
44              result_s<= a and b;
45              cout_s <= '0';
46           when "001" =>           -- "001" a or b
47              result_s <= a or b;
48              cout_s <= '0';
49           when "010" =>           -- "010" a + b
50              result_s <= result_add;
51              cout_s <= cout_add;
52           when "011" =>           -- "011" b
53              result_s <= b;
54              cout_s <= '0';
55           when "110" =>           -- "110" a - b
56              result_s <= result_sub;
57              cout_s <= cout_sub;
58           when "100" =>           -- a sll 1
59              result_s <= a(30 downto 0) & '0';
60              cout_s <= a(31);
61           when "101" =>           -- a srl 1
```

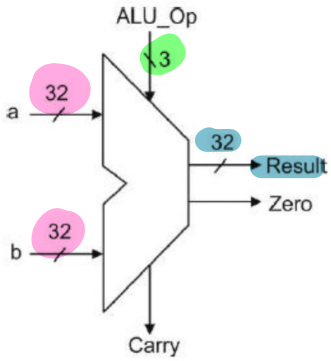
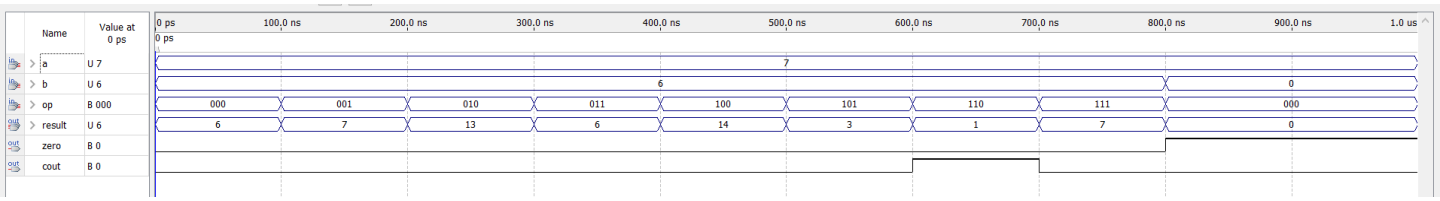


Figure 1: 32-bit ALU

Operation Name	ALU-Op		Operation Performed
	Neg/Tsel	*ALU-Select*	
AND (Logical)	0	00	Result <= a AND b
Or (Logical)	0	01	Result <= a OR b
ADD	0	10	Result <= a + b
SUB	1	10	Result <= a – b
ROL	1	00	Result <= a << 1
ROR	1	01	Result <= a >> 1

```
62              result_s <= '0' & a(31 downto 1);
63              cout_s <= '0';
64           when others =>
65              result_s <= a;
66              cout_s <= '0';
67        end case;
68
69        case (result_s) is
70           when (others => '0') =>
71              zero_s <= '1';
72           when others =>
73              zero_s <= '0';
74        end case;
75     end process;
76
77     result <= result_s;
78     cout <= cout_s;
79     zero <= zero_s;
80  end Behavior;
81
82
```


Name
Value at 0 ps
a
U 7
b
U 6
op
B 000
result
U 6
zero
B 0
cout
B 0
0 ps
100,0 ns
200,0 ns
300,0 ns
400,0 ns
500,0 ns
600,0 ns
700,0 ns
800,0 ns
900,0 ns
1.0 us
7
6
0
000
001
010
011
100
101
110
111
000
6
7
13
6
14
3
1
7
0