Experiment 4: **ALU**

Prajwal Kishor Kalpande, Roll Number:**200070028**

EE214, WEL, IIT BOMBAY

September 21, 2021

Overview of the Experiment :

In this experiment we have to design an ALU circuit using behavioral modelling to perform certain operations based on select lines. The input includes a 2-bit binary number denoting select line and based on this number operations are to be performed on 4-bit binary numbers A and B to give a 8-bit output binary number.The operations include Concatenation, Addition, EX-ORing and Multiplication by 2.

Approach to the Experiment :

The ALU has two 4-bit inputs i.e. A3,A2,A1,A0 and B3,B2, B1, B0 and one two bit input S1,S0 which indicate the select signal. Based on the select line we have to perform one of four operations on the two four bit inputs as follows :

S1	**S0**	**ALU Output**
0	0	Concatenate two 4-bit inputs A and B
0	1	Perform A+B Operation
1	0	Perform A xor B Operation
1	1	Produce output as 2*A

If-else statements have been used to implement this.When S1 = 0, S0 = 0 we just have to concatenate the two inputs which is easily done using the concatenation operator ‘&’ and then the concatenated result is given as output.
When S1 = 0, S0 = 1 we have to perform addition operation A+B. For this a function add is declared in which two variables sum and carry are calculated using the boolean expressions which we already know for sum and carry. The temporary carry generated each time one bit of A and one bit of B is added is stored in variable temp. After doing this four times using a for loop, we get the final carry and sum which are then concatenated to get the output. Since the carry is only one bit, three zeros are added in the beginning.

When S1 = 1, S0 =0 we have to perform bit wise XOR which is done using a four loop and XOring A3 with B3 , A2 with B2 and so on.
Finally when S1= 1 and S0 = 1 we have to multiply A by 2. We know that multiplication by 2 is just shifting the whole number in binary by one place to the left and adding an extra zero at the end. To do this a 8-bit variable is declared with all its bits set to zero and then the four bits of this variable starting from the second one till the fifth one are assigned the value A (A3, A2,A1,A0).

Design document and VHDL code :

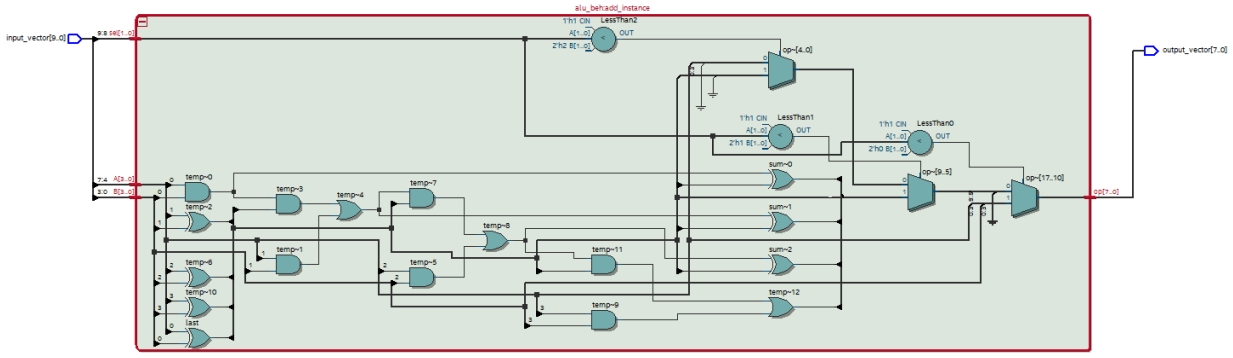
The ALU has two 4-bit inputs i.e. A3,A2,A1,A0 and B3,B2, B1, B0 and one two bit input sel1,sel0 which is the select signal(denoted by S1,S0 above). The output is 8-bit op8, op7, op6, op5, op4, op3, op2, op1 and op0. The VHDL code is as follows:

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

entity alu_beh is
   generic(
      operand_width : integer:=4;
      sel_line : integer:=2
      );
   port (
      A: in std_logic_vector(operand_width-1 downto 0);
      B: in std_logic_vector(operand_width-1 downto 0);
      sel: in std_logic_vector(sel_line-1 downto 0);
      op: out std_logic_vector((operand_width*2)-1 downto 0)
   ) ;
end alu_beh;

architecture a1 of alu_beh is
   function add(A: in std_logic_vector(operand_width-1 downto 0); B: in
std_logic_vector(operand_width-1 downto 0))
      return std_logic_vector is
```

```
        variable sum,carry: std_logic_vector(3 downto 0);
                                variable i : integer;
                                variable temp : std_logic;

        -- you can use aggregate to initialize the variables & signals as shown below
        --    variable variable_name : std_logic_vector(3 downto 0) := (others => '0');
      begin
                                temp:= '0';
        addition : for i in 0 to 3 loop
                                        sum(i):=(A(i) xor B(i) ) xor temp;
                                        temp :=(A(i) and B(i)) or (temp and (A(i) xor B(i) ));
                                end loop addition;
                                carry :="000"&temp;
        return carry&sum;
    end add;


begin
alu : process( A, B, sel )
variable last : std_logic_vector (3 downto 0);
variable mul : std_logic_vector ( 7 downto 0):= (others => '0');
begin

                if(sel<= "00")then
                        op <=  A&B;
                elsif(sel<= "01")then
                        op <= add(A,B);
                elsif(sel<= "10")then
                        exor:for i in 0 to 3 loop
                                        last(i) := A(i) xor B(i);
                                end loop exor;
                        op <= "0000"&last;

                else
                        mul(4 downto 1) := A;
                        op<= mul;

                end if;
   --
   -- add function usage :
```

```
   --   signal_name <= add(A,B)
   --   variable_name := add(A,B)
   --
   -- concatenate operator usage:
   --    "0000"&A
end process ; -- alu
end a1 ; -- a1
```

RTL View :



DUT Input/Output Format :

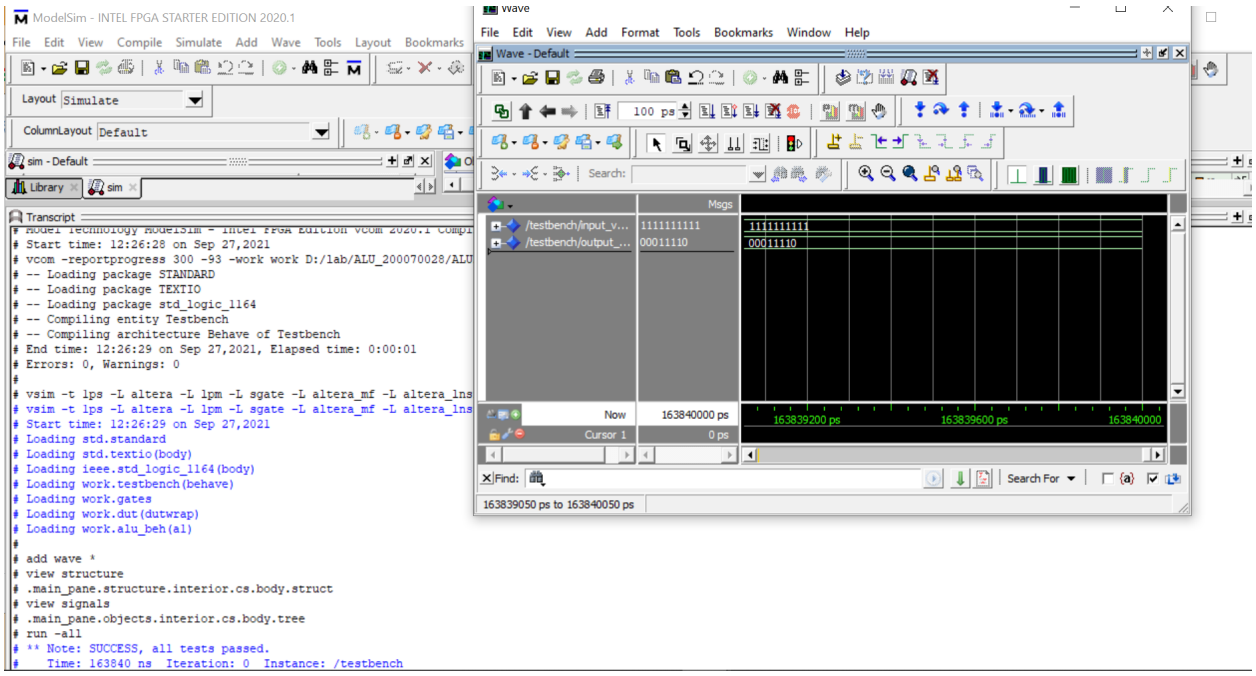
Input :

```
sel(1) => input_vector(9),
sel(0) => input_vector(8),

A(3) => input_vector(7),
A(2) => input_vector(6),
A(1) => input_vector(5),
A(0) => input_vector(4),
B(3) => input_vector(3),
B(2) => input_vector(2),
B(1) => input_vector(1),
B(0) => input_vector(0),
```

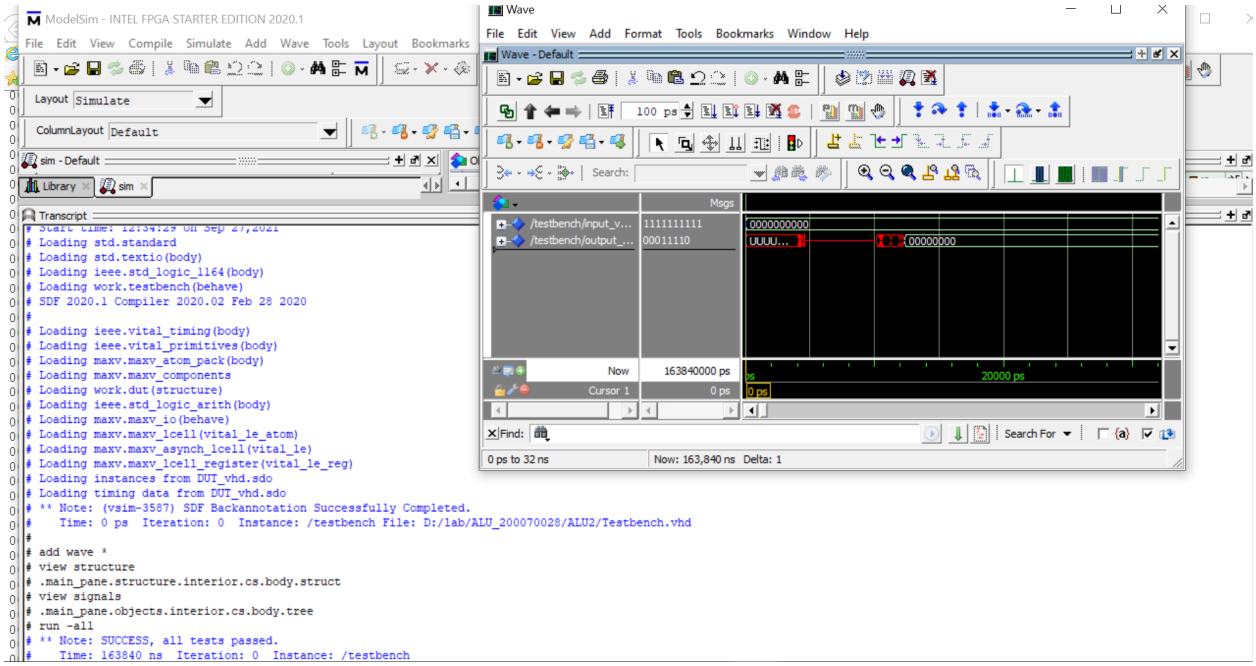
Output :

```
op(7) => output_vector(7),
op(6) => output_vector(6),
op(5) => output_vector(5),
op(4) => output_vector(4),
op(3) => output_vector(3),
op(2) => output_vector(2),
op(1) => output_vector(1),
op(0) => output_vector(0));
```

RTL Simulation :



Gate-Level Simulation :



Krypton Board :

After doing Scan Chain we get the following outputs in the out.txt file and no failures are found :

```
0000000000 00000000 Success
0000000001 00000001 Success
0000000010 00000010 Success
0000000011 00000011 Success
0000000100 00000100 Success
0000000101 00000101 Success
0000000110 00000110 Success
0000000111 00000111 Success
0000001000 00001000 Success
0000001001 00001001 Success
0000001010 00001010 Success
0000001011 00001011 Success
0000001100 00001100 Success
0000001101 00001101 Success
0000001110 00001110 Success
0000001111 00001111 Success
0000010000 00010000 Success
0000010001 00010001 Success
0000010010 00010010 Success
0000010011 00010011 Success
0000010100 00010100 Success
0000010101 00010101 Success
0000010110 00010110 Success
0000010111 00010111 Success
0000011000 00011000 Success
0000011001 00011001 Success
0000011010 00011010 Success
0000011011 00011011 Success
0000011100 00011100 Success
0000011101 00011101 Success
0000011110 00011110 Success
0000011111 00011111 Success
0000100000 00100000 Success
0000100001 00100001 Success
0000100010 00100010 Success
0000100011 00100011 Success
0000100100 00100100 Success
0000100101 00100101 Success
0000100110 00100110 Success
0000100111 00100111 Success
0000101000 00101000 Success
0000101001 00101001 Success
0000101010 00101010 Success
0000101011 00101011 Success
0000101100 00101100 Success
0000101101 00101101 Success
0000101110 00101110 Success
0000101111 00101111 Success
0000110000 00110000 Success
0000110001 00110001 Success
0000110010 00110010 Success
0000110011 00110011 Success
0000110100 00110100 Success
0000110101 00110101 Success
0000110110 00110110 Success
0000110111 00110111 Success
0000111000 00111000 Success
0000111001 00111001 Success
0000111010 00111010 Success
0000111011 00111011 Success
0000111100 00111100 Success
0000111101 00111101 Success
0000111110 00111110 Success
0000111111 00111111 Success
0001000000 01000000 Success
0001000001 01000001 Success
0001000010 01000010 Success
0001000011 01000011 Success
0001000100 01000100 Success
0001000101 01000101 Success
0001000110 01000110 Success
0001000111 01000111 Success
0001001000 01001000 Success
0001001001 01001001 Success
0001001010 01001010 Success
```

0001001011 01001011 Success
0001001100 01001100 Success
0001001101 01001101 Success
0001001110 01001110 Success
0001001111 01001111 Success
0001010000 01010000 Success
0001010001 01010001 Success
0001010010 01010010 Success
0001010011 01010011 Success
0001010100 01010100 Success
0001010101 01010101 Success
0001010110 01010110 Success
0001010111 01010111 Success
0001011000 01011000 Success
0001011001 01011001 Success
0001011010 01011010 Success
0001011011 01011011 Success
0001011100 01011100 Success
0001011101 01011101 Success
0001011110 01011110 Success
0001011111 01011111 Success
0001100000 01100000 Success
0001100001 01100001 Success
0001100010 01100010 Success
0001100011 01100011 Success
0001100100 01100100 Success
0001100101 01100101 Success
0001100110 01100110 Success
0001100111 01100111 Success
0001101000 01101000 Success
0001101001 01101001 Success
0001101010 01101010 Success
0001101011 01101011 Success
0001101100 01101100 Success
0001101101 01101101 Success
0001101110 01101110 Success
0001101111 01101111 Success
0001110000 01110000 Success

1111011011 00011010 Success
1111011100 00011010 Success
1111011101 00011010 Success
1111011110 00011010 Success
1111011111 00011010 Success
1111100000 00011100 Success
1111100001 00011100 Success
1111100010 00011100 Success
1111100011 00011100 Success
1111100100 00011100 Success
1111100101 00011100 Success
1111100110 00011100 Success
1111100111 00011100 Success
1111101000 00011100 Success
1111101001 00011100 Success
1111101010 00011100 Success
1111101011 00011100 Success
1111101100 00011100 Success
1111101101 00011100 Success
1111101110 00011100 Success
1111101111 00011100 Success
1111110000 00011110 Success
1111110001 00011110 Success
1111110010 00011110 Success
1111110011 00011110 Success
1111110100 00011110 Success
1111110101 00011110 Success
1111110110 00011110 Success
1111110111 00011110 Success
1111111000 00011110 Success
1111111001 00011110 Success
1111111010 00011110 Success
1111111011 00011110 Success
1111111100 00011110 Success
1111111101 00011110 Success
1111111110 00011110 Success
1111111111 00011110 Success