Experiment 4: **Combinational Circuit 4**

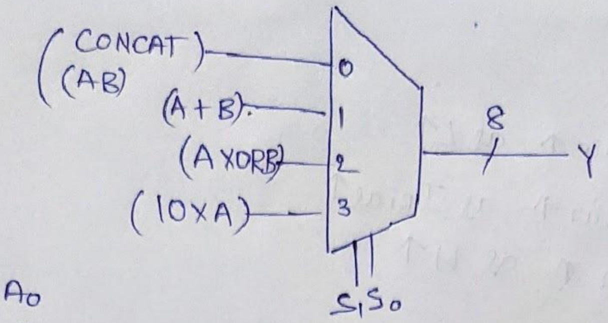
ALU

Gnanendar Reddy palagiri Roll Number 200070053
EE-214, WEL, IIT Bombay
September 24, 2021

Overview of the experiment:

- The purpose of this experiment is to make us familiar with how to use Behavioral and Dataflow modelling to build circuits .
- We modelled an ALU using behavioural and dataflow modelling .
- I will be presenting the report in the following order approach,code followed by the results.

Approach to the experiment:



$A = A_3 A_2 A_1 A_0$

$B = B_3 B_2 B_1 B_0$

$S_1 S_0$	Y
0 0	$A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0$
0 1	$0 0 0 C\ S_3 S_2 S_1 S_0$
1 0	$0 0 0 0\ X_3 X_2 X_1 X_0$
1 1	$0 0 0\ A_3 A_2 A_1 A_0\ 0$

$$\rightarrow A+B = \begin{array}{ccccc} & A_3 & A_2 & A_1 & A_0 \\ & B_3 & B_2 & B_1 & B_0 \\ \hline C & C_{iii} & C_{ii} & C_i & \\ C & S_3 & S_2 & S_1 & S_0 \\ \hline \end{array}$$

$\rightarrow A \text{ XOR } B = X_3 X_2 X_1 X_0$

$$\begin{array}{r} 10 \times A_3 A_2 A_1 A_0 \\ \hline 0000 \\ A_3 A_2 A_1 A_0 \\ \hline A_3 A_2 A_1 A_0 0 \end{array}$$

→ to make rest of the o/p through Y 8bits we concat remaining Bits (towards MSB) with '0''s.

Design document and VHDL code if relevant:

```
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
            -- Declare "sum" and "carry" variable
                                        variable sum,carry: std_logic_vector(4 downto 0);
        begin
            -- write logic for addition
            -- Hint: Use for loop


                                                    sum(0):=(A(0) xor B(0));
                                                    carry(0):=(A(0) and B(0));

                                                    adder: for i in 1 to 3 loop
                                                    sum(i):=(A(i) xor B(i) xor carry(i-1));
                                                    carry(i):=(((A(i) xor B(i)) and carry(i-1)) or (A(i) and B(i)));

                end loop;
                                                                sum(4):= carry(3);





            return sum;
    end add;


begin
alu : process( A, B, sel )
variable temp : std_logic_vector(3 downto 0);
```

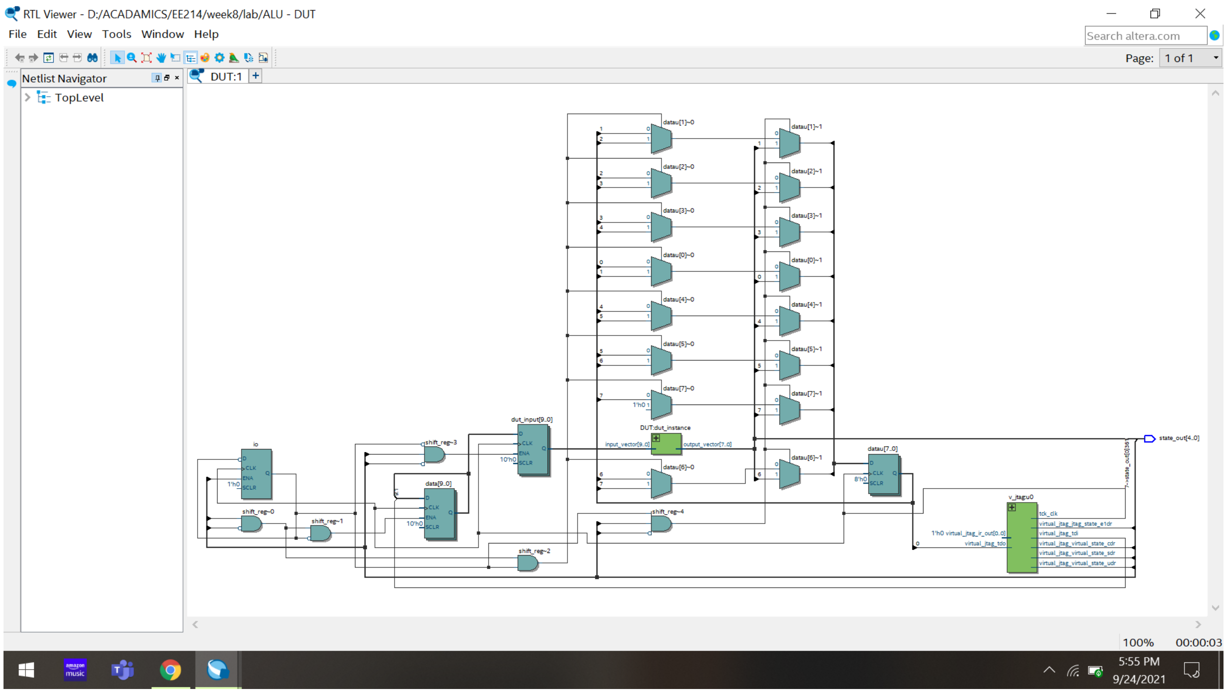
```
begin
   -- complete VHDL code for various outputs of ALU based on select lines
   -- Hint: use if/else statement
   -- add function usage :
   --    signal_name <= add(A,B)
   --    variable_name := add(A,B)
             if (sel="00") then
               op<= (A&b);
             elsif (sel="01") then
               op<=("000"&add(A,B));
             elsif (sel="10") then
               temp:= A xor B;
               op<=("0000"&temp);
             elsif (sel="11") then
              op<=("000"&A&"0");
             end if;



end process ; -- alu
end a1 ; -- a1

.
```

RTL View:



DUT Input/Output Format:

Input format:

```
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 format:

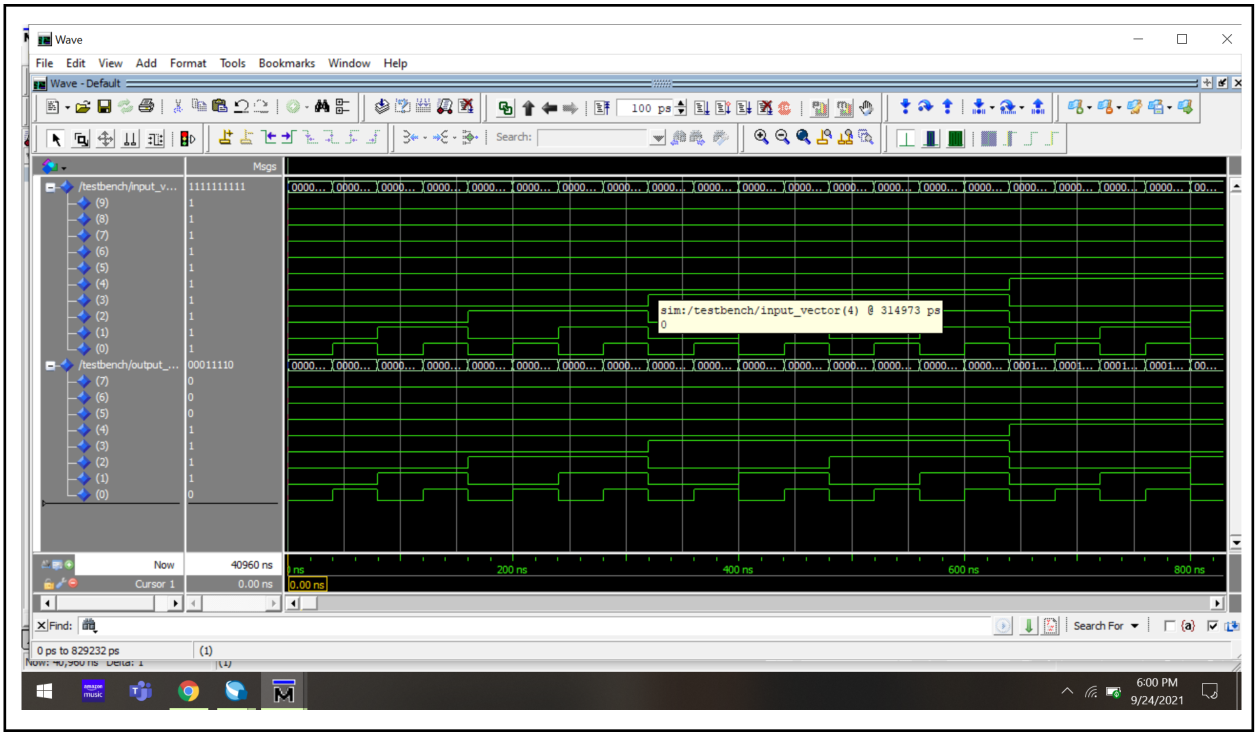
```
op(7) => output_vector(7),
op(6) => output_vector(6),
op(5) => output_vector(5),
op(4) => output_vector(4),
op(3) => output_vector(3),
op(2) => output_vector(2),
op(1) => output_vector(1),
op(0) => output_vector(0)
```

Model tracefile:

```
0000000000 00000000 11111111
0000000001 00000001 11111111
0000000010 00000010 11111111
0000000011 00000011 11111111
```

.

RTL Simulation:



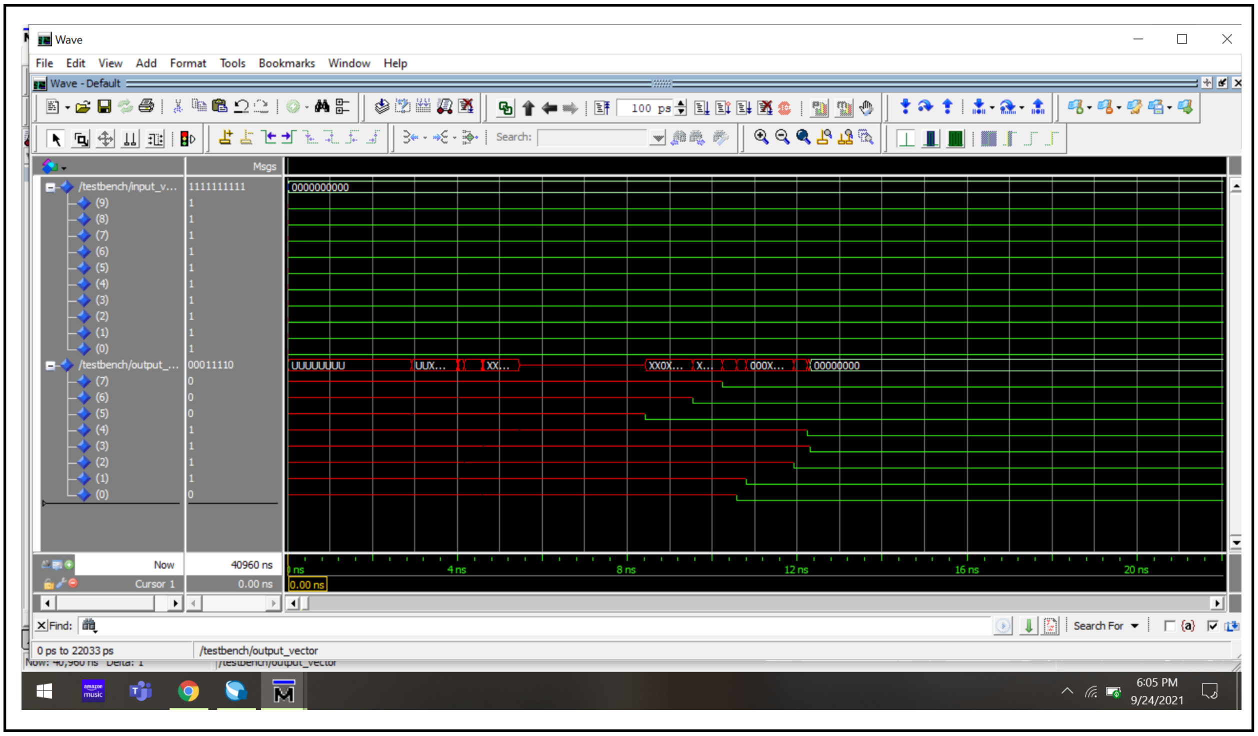
	Msgs
/testbench/input_v...	1111111111
(9)	1
(8)	1
(7)	1
(6)	1
(5)	1
(4)	1
(3)	1
(2)	1
(1)	1
(0)	1
/testbench/output_...	00011110
(7)	0
(6)	0
(5)	0
(4)	1
(3)	1
(2)	1
(1)	1
(0)	0
Now	40960 ns
Cursor 1	0.00 ns

```
sim:/testbench/input_vector(4) @ 314973 ps
0
```

0.00 ns

0 ps to 829232 ps (1)

Gate-level Simulation:



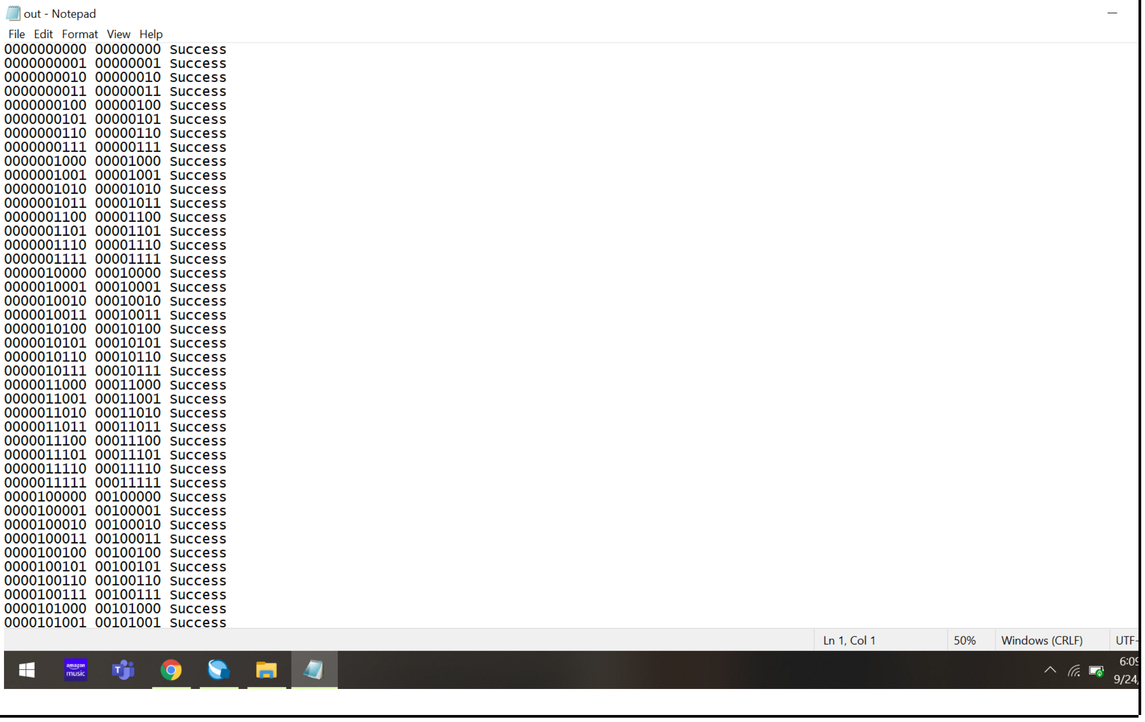
	Msgs
/testbench/input_v...	1111111111
(9)	1
(8)	1
(7)	1
(6)	1
(5)	1
(4)	1
(3)	1
(2)	1
(1)	1
(0)	1
/testbench/output_...	00011110
(7)	0
(6)	0
(5)	0
(4)	1
(3)	1
(2)	1
(1)	1
(0)	0
Now	40960 ns
Cursor 1	0.00 ns

0.00 ns

0 ps to 22033 ps /testbench/output_vector

Observations:

- I have observed that the quartus automatically convert the behavioural description we have written into available logic circuits by based on our selected board model.
- Scanchain result sample :



```
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
```