Experiment 4: CC_4

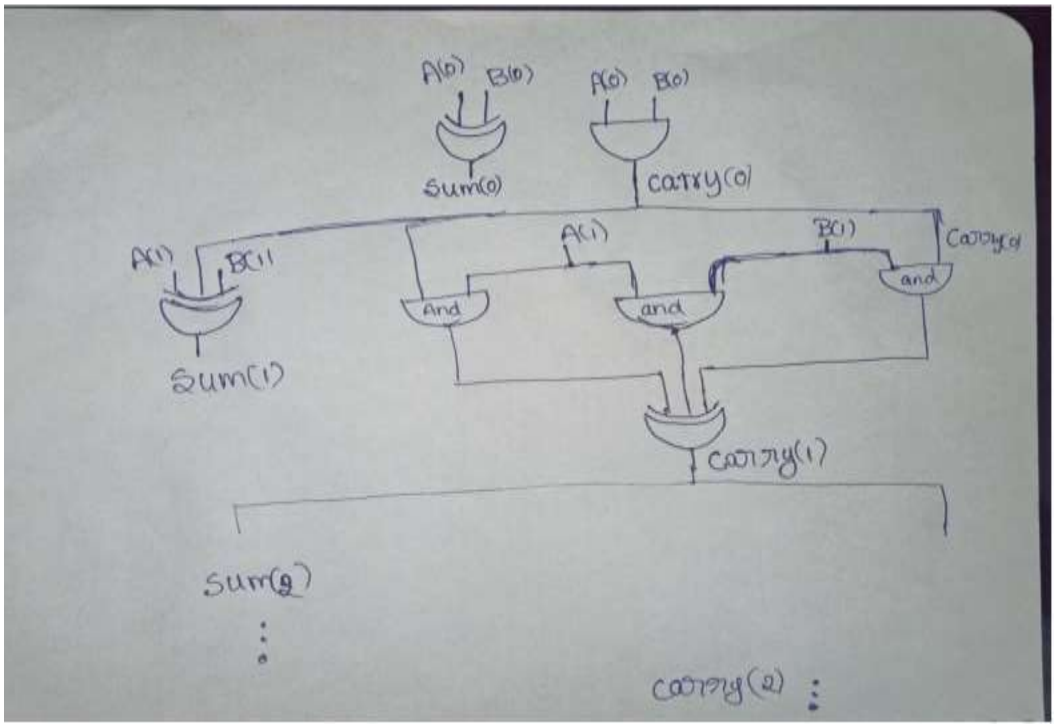
Narne Avinash Chowdary, Roll Number:200070047
EE-214, WEL, IIT Bombay
September 26, 2021

Overview of the experiment:

The purpose of the experiment is starting learning of **Behavioral and Dataflow modelling** Using Behavioral model we required to create a model for like 4,2 mux using sel a 2 bit vector for choosing 4 different process for A, B two 4 bit vectors, namely Concatenate two 4-bit inputs A and B, Performs A+B Operation, Performs A xor B Operation, Produces output as 2*A

To perform the experiment I completed the function add by taking for loop in consideration allotted different 4 different sel vectors for 4 different functions stated above by using the if , elsif and else statements,

Approach to the experiment:

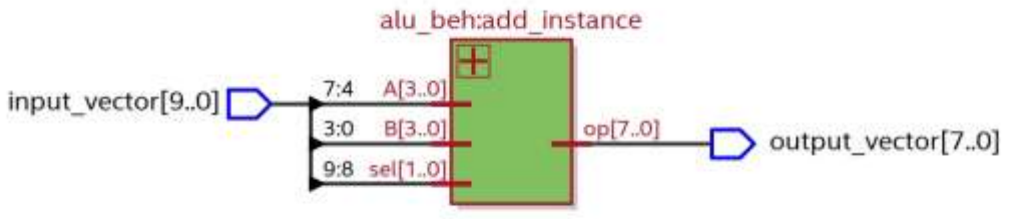


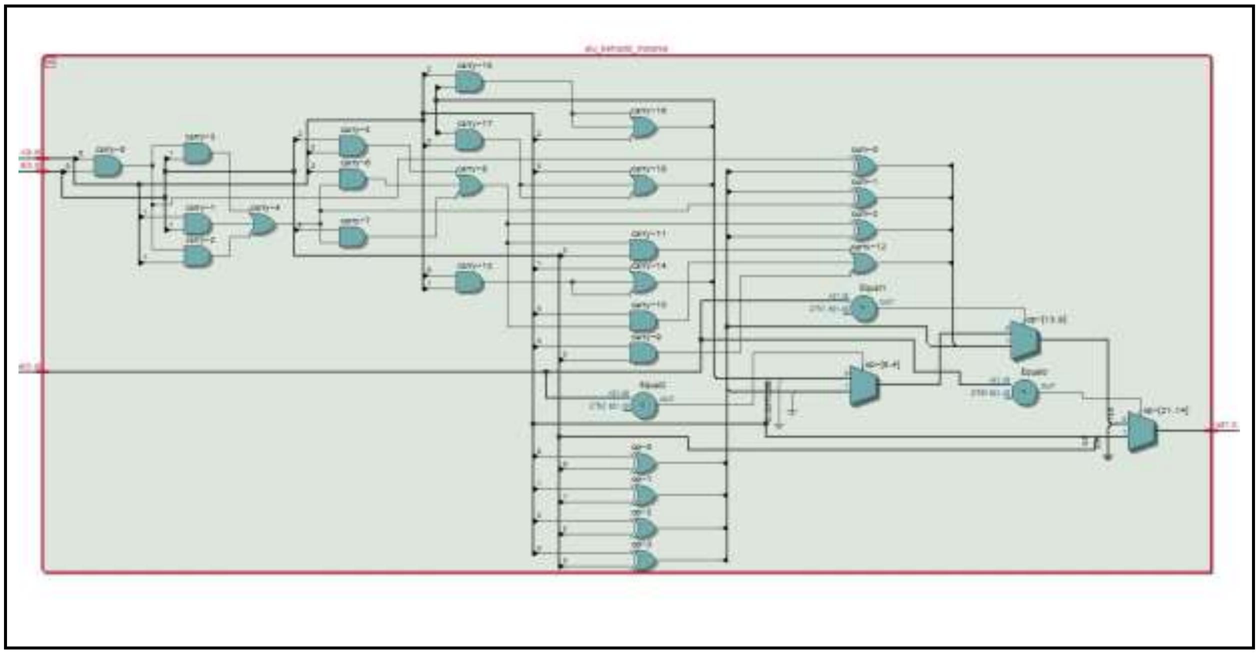
The add function is shown all the remaining is if statements, for add function collect all the sum values up to sum(4) and assign to output and remaining of the output bits as 0, Design document and VHDL code if relevant:

```
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
                          variable sum: std_logic_vector((operand_width*2)-1 downto 0) := (others=> '0');
                          variable carry: std_logic_vector(operand_width-1 downto 0) := (others=> '0');
                          variable i: integer;
            -- Declare "sum" and "carry" variable
            -- you can use aggregate to initialize the variables & signals as shown below
            --    variable variable_name : std_logic_vector(3 downto 0) := (others => '0');
        begin
                          sum(0) := A(0) xor B(0);
                          carry(0) :=A(0) and B(0);
                          Addition : for i in 1 to operand_width-1 loop
                          carry(i) := (A(i) and B(i)) or(A(i) and carry(i-1))or (B(i) and carry(i-1)) ;
                           sum(i) := A(i) xor B(i) xor carry(i-1);
                          end loop;
                          sum(4) := carry(3);
            -- write logic for addition
            -- Hint: Use for loop
            return sum;

    end add;
```

```
begin
alu : process( A, B, sel )
begin
 if (sel = "00") then
 op<= A&B;
 elsif (sel = "01") then
 op<= add(A,B);
 elsif (sel ="10") then
 op <= "0000" & (A xor B) ;
 else
 op <= add(A,A);
 end if;
   -- 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)
   --
   -- concatenate operator usage:
   --    "0000"&A
end process ;
end a1 ;
```

RTL View:

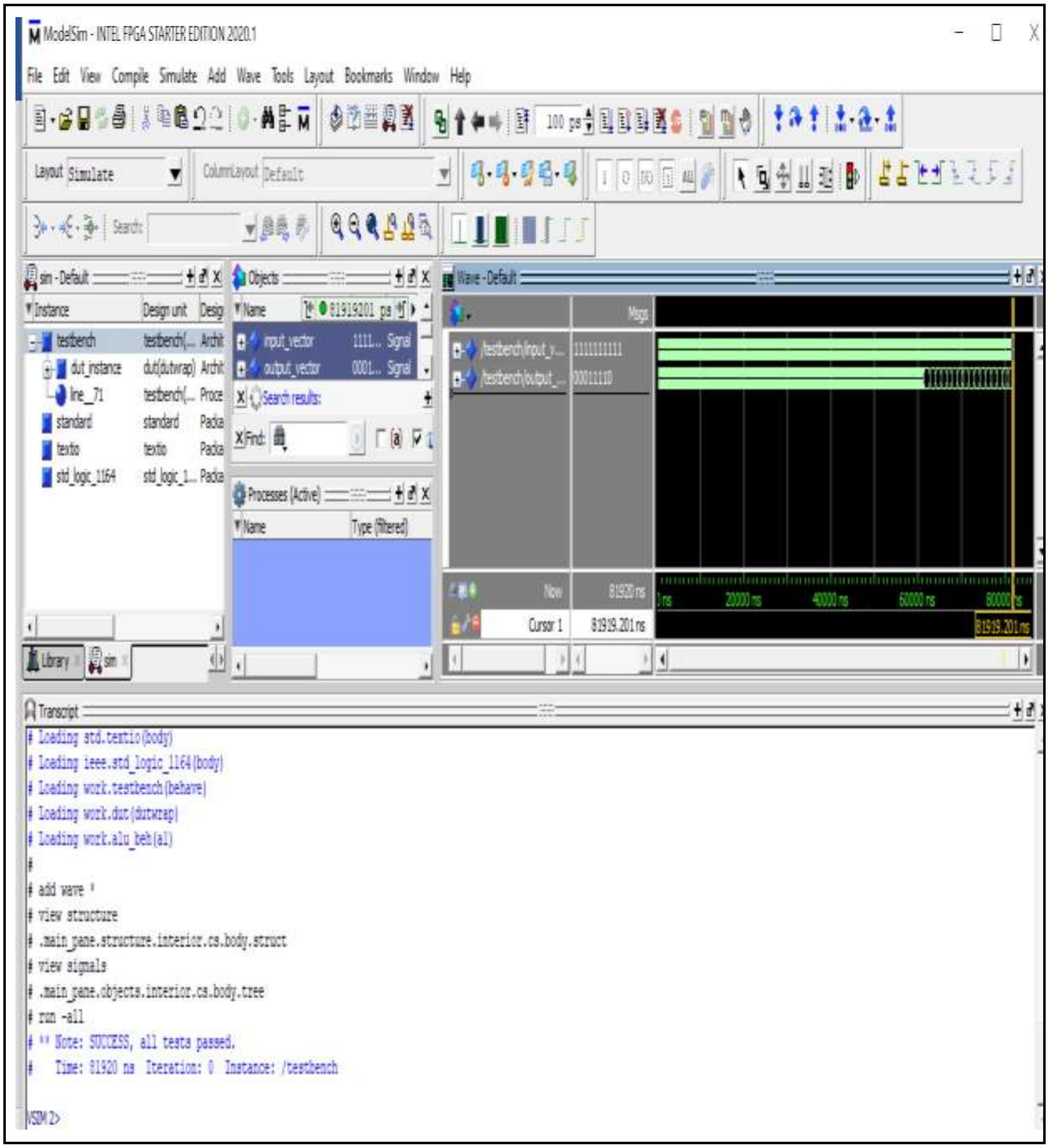




DUT Input/Output Format:

```
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),
                              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));
<S1 S0 A3 A2 A1 A0 B3 B2 B1 B0> <Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0> 11111111

Tracefile examples
0000000000 00000000 11111111
0000000001 00000001 11111111
0000000010 00000010 11111111
0000000011 00000011 11111111
0000000100 00000100 11111111
0000000101 00000101 11111111
```

RTL Simulation:



ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript

Instance	Design unit	Desig
testbench	testbench(...	Archit
dut_instance	dut(dutwrap)	Archit
line__71	testbench(...	Proce
standard	standard	Packa
textio	textio	Packa
std_logic_1164	std_logic_1...	Packa

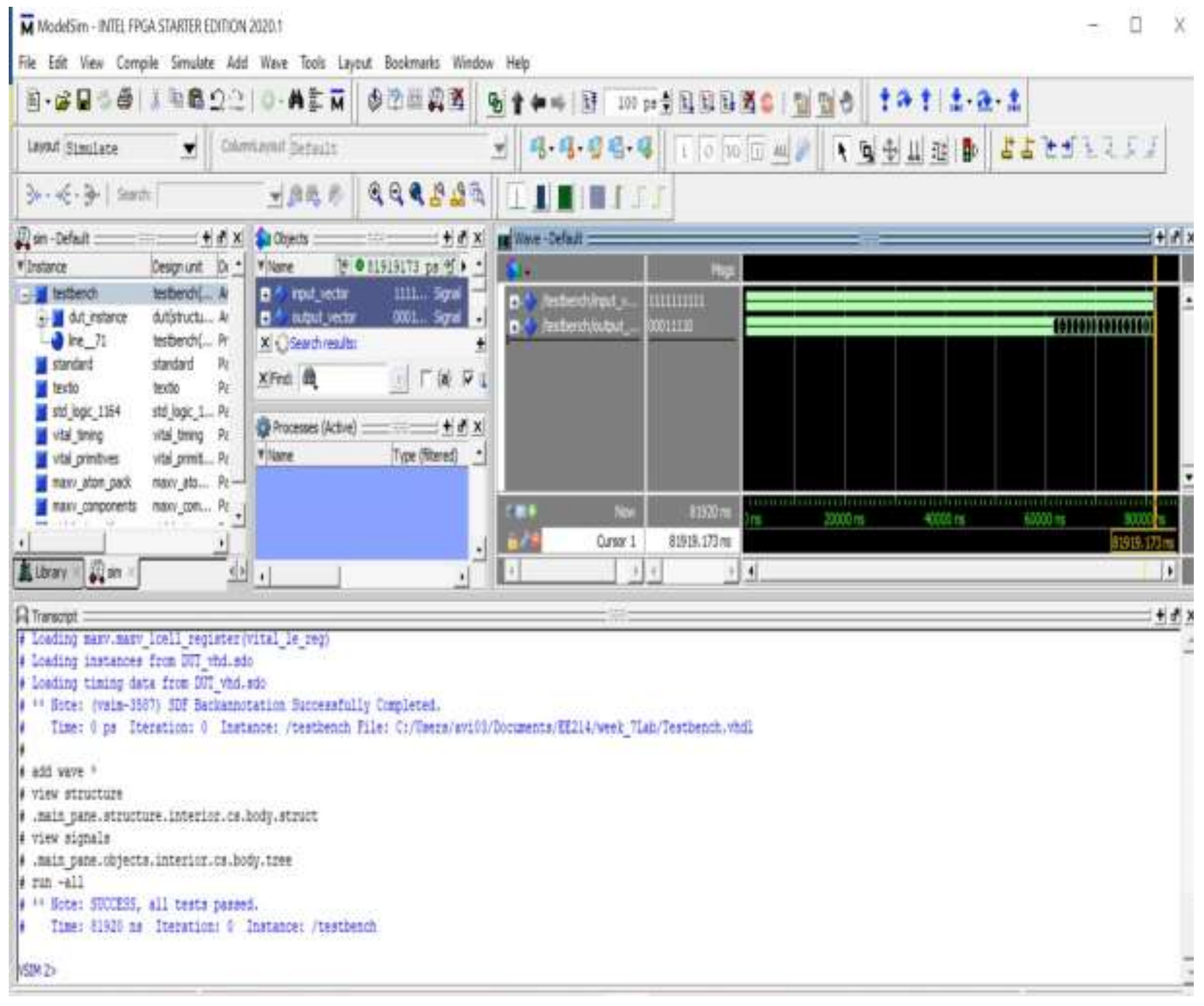
Name	81919201 ps	
input_vector	1111...	Signal
output_vector	0001...	Signal

	Msgs
/testbench/input_v...	111111111
/testbench/output_...	00011110

```
Now       81920 ns
Cursor 1  81919.201 ns
```

```
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.testbench(behave)
# Loading work.dut(dutwrap)
# Loading work.alu_beh(al)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 81920 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```

Gate-level Simulation:


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript

```
# Loading maxv.maxv_lcell_register(vital_le_reg)
# Loading instances from DUT_vhd.sdo
# Loading timing data from DUT_vhd.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /testbench File: C:/Users/avi03/Documents/EE214/week_7Lab/Testbench.vhdl
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 81920 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```

Krypton board*:

NA

Observations*:

It is observed that design is correct and it is successfully verified using ScanChain by viewing the TRACEFILE cases in the out.txt file generated in project directory.

Below are some of the cases in out.txt file

```
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
```