Experiment 5: Sequential Circuits 1

Krutheeka R K J, Roll Number: 200070038
EE-214, WEL, IIT Bombay
September 28, 2021

Overview of the experiment:

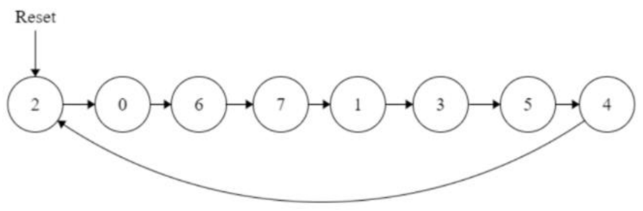
The objective of this experiment is to design and construct **Sequence Generator** in VHDL environment using Quartus software. We synthesize and simulate a device which generates a certain sequence upon every input clock pulse and on reset, the sequence will go into default sequence i.e. 2 in this case. In this experiment, we will also use **Scan chain** to test the correctness of the design.

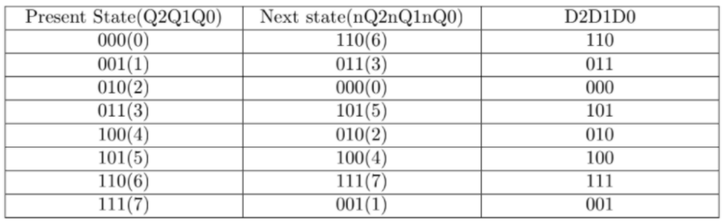
Approach to the experiment:

We will be using both the approaches: Structural and Behavioral.

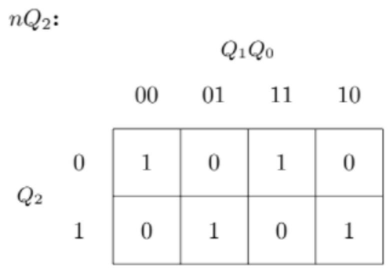
Structural modelling is used where we instantiate the components and describe the interconnection of the components we defined. Behavioral and Data Flow modelling is used where we describe the behavior of the circuit and define sequential statements.

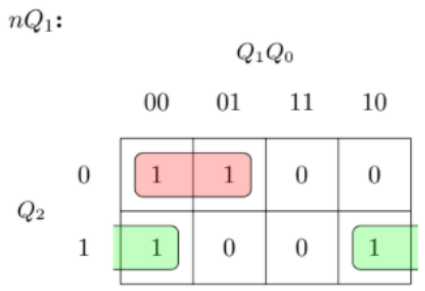
We used Quartus to compile the designs, synthesize the RTL view and ModelSim to simulate our digital design. We will test the design on hardware (Krypton board) using Scan chain.

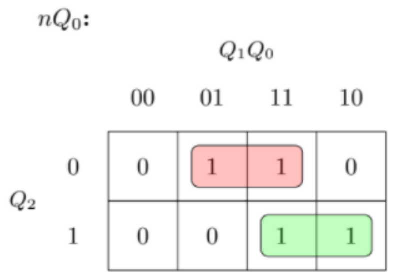




Present State(Q2Q1Q0)	Next state(nQ2nQ1nQ0)	D2D1D0
000(0)	110(6)	110
001(1)	011(3)	011
010(2)	000(0)	000
011(3)	101(5)	101
100(4)	010(2)	010
101(5)	100(4)	100
110(6)	111(7)	111
111(7)	001(1)	001







D2 = Q2 xnor (Q1 xor Q2)
D1 = Q2. Q0' + Q2'. Q1'
D0 = Q2'. Q0 + Q2. Q1

We will also define the architecture of a D-flipflop as a package to help us in the main design in structural modelling.

Design document and VHDL code:

STRUCTURAL MODELLING:

1.D-FLIP FLOP:

```
library ieee;
use ieee.std_logic_1164.all;
package flipflops is

  component dff2 is port(D,clk,reset:in std_logic;Q:out std_logic);
  end component dff2;

  component dff1 is port(D,clk,reset:in std_logic;Q:out std_logic);
  end component dff1;

  component dff0 is port(D,clk,reset:in std_logic;Q:out std_logic);
  end component dff0;


end package flipflops; --we define different d-flipflops for 3 bits
```

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

entity dff2 is port(D,clk,reset:in std_logic;Q:out std_logic);
end entity dff2;
architecture behav of dff2 is
begin
dff2: process (clk,reset)
begin
--on reset make Q = 0 or 1 based on requirement
if(reset='1')then
Q <= '0';
--when there is a change in clock pulse and the present clock value is '1'
elsif (CLK'event and (CLK='1')) then
Q <= D;
end if;
end process dff2;
end behav;

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
port(D,clk,reset:in std_logic;
       Q:out std_logic);
end entity dff1;
architecture behav of dff1 is
begin
dff1: process (clk,reset)
begin
if(reset='1')then
Q <= '1';
elsif (CLK'event and (CLK='1')) then
Q <= D;
end if;
end process dff1;
end behav;

library ieee;
use ieee.std_logic_1164.all;

entity dff0 is port(D,clk,reset:in std_logic;Q:out std_logic);
```

```
end entity dff0;
architecture behav of dff0 is
begin
dff0: process (clk,reset)
begin
if(reset='1')then
Q <= '0';
elsif (CLK'event and (CLK='1')) then
Q <= D;
end if;
end process dff0;
end behav;
```

2. SEQUENCE GENERATOR:

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

entity sequence_generator_structural is
      port (reset,clock: in std_logic;
                   y:out std_logic_vector(2 downto 0));
end entity sequence_generator_structural;

architecture struct of sequence_generator_structural is
      signal D2,D1,D0 :std_logic;
      signal Q:std_logic_vector(2 downto 0);
begin
--The logic expressions
D2<= Q(2) xnor (Q(1) xor Q(0));
D1<= (Q(2) and (not Q(0))) or ((not Q(2)) and (not Q(1)));
D0<= ((not Q(2)) and Q(0)) or (Q(2) and Q(1));
Y(2) <= Q(2);
Y(1) <= Q(1);
Y(0) <= Q(0);

dff_0  : dff0 port map(D => D0, clk => clock, reset => reset, Q => Q(0));

--Q1
dff_1  : dff1 port map(D => D1, clk => clock, reset => reset, Q => Q(1));

--Q2
dff_2  : dff2 port map(D => D2, clk => clock, reset => reset, Q => Q(2));

end struct;
```

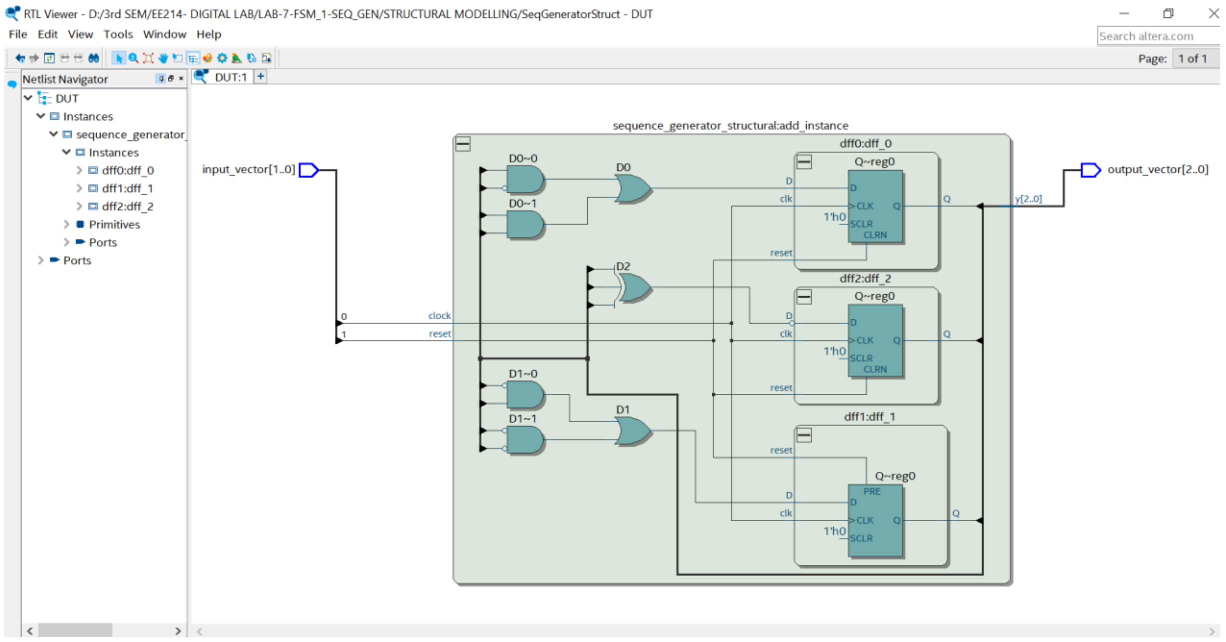
BEHAVIORAL MODELLING:

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

entity sequence_behavior is
port (reset,clock: in std_logic;
y:out std_logic_vector(2 downto 0));
end entity sequence_behavior;

architecture behav  of sequence_behavior is
--defines the signal that gets involved in sequential process
signal state:std_logic_vector(2 downto 0);
constant s_2:std_logic_vector(2 downto 0):="010";
constant s_0:std_logic_vector(2 downto 0):="000";
constant s_1:std_logic_vector(2 downto 0):="001";
constant s_3:std_logic_vector(2 downto 0):="011";
constant s_4:std_logic_vector(2 downto 0):="100";
constant s_5:std_logic_vector(2 downto 0):="101";
constant s_6:std_logic_vector(2 downto 0):="110";
constant s_7:std_logic_vector(2 downto 0):="111";

begin
-- we use case statements to tell which state comes after the present state
reg_process: process(clock,reset)
begin
if(reset='1')then
state<= s_2; -- write the reset state
elsif(clock'event and clock='1')then
case state is
      --reset
      when s_2=>
                    state<=s_0;
      when s_0 =>
                    state<=s_6;
             when s_1 =>
                    state<=s_3;
             when s_3 =>
                    state<=s_5;
             when s_4 =>
                    state<=s_2;
             when s_5 =>
                    state<=s_4;
             when s_6 =>
                    state<=s_7;
             when s_7 =>
                    state<=s_1;
```

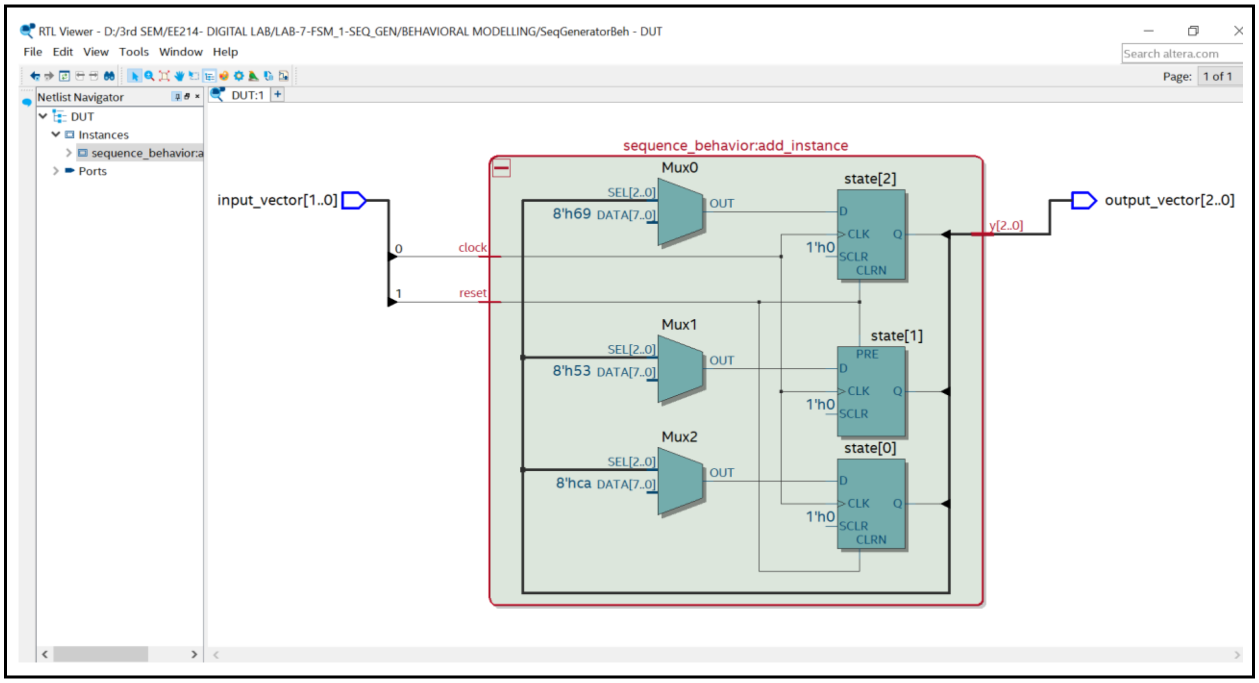
```
       when others=>
         state<= s_2;-- write the reset state
       end case;
end if;
end process reg_process;
y<=state;
end behav;
```

RTL View:

STRUCTURAL MODELLING:



BEHAVIORAL MODELLING:



DUT Input/Output Format:

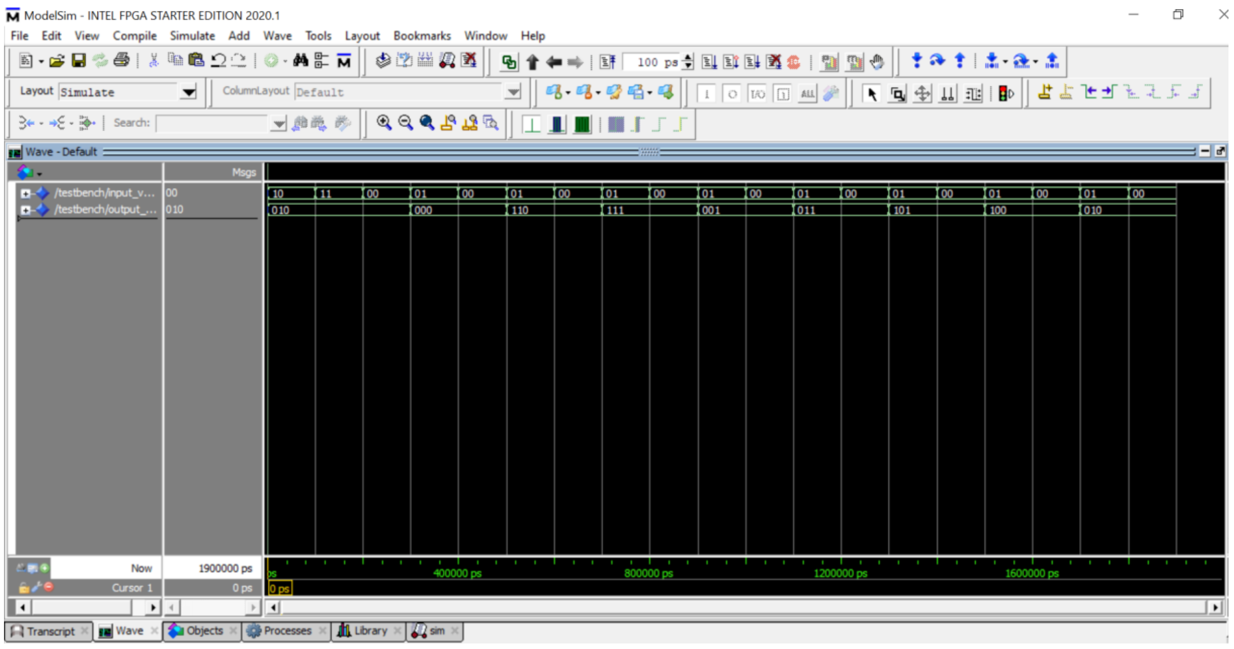
Input: <reset><clock>
LSB: <clock>
MSB: <reset>
Output: <Y2><Y1><Y0>
Test cases: 01 000 000
00 000 111
01 110 000

Scan Chain input/output:

Input	**Output**	**Remarks**
00	011	Success
01	101	Success
00	101	Success

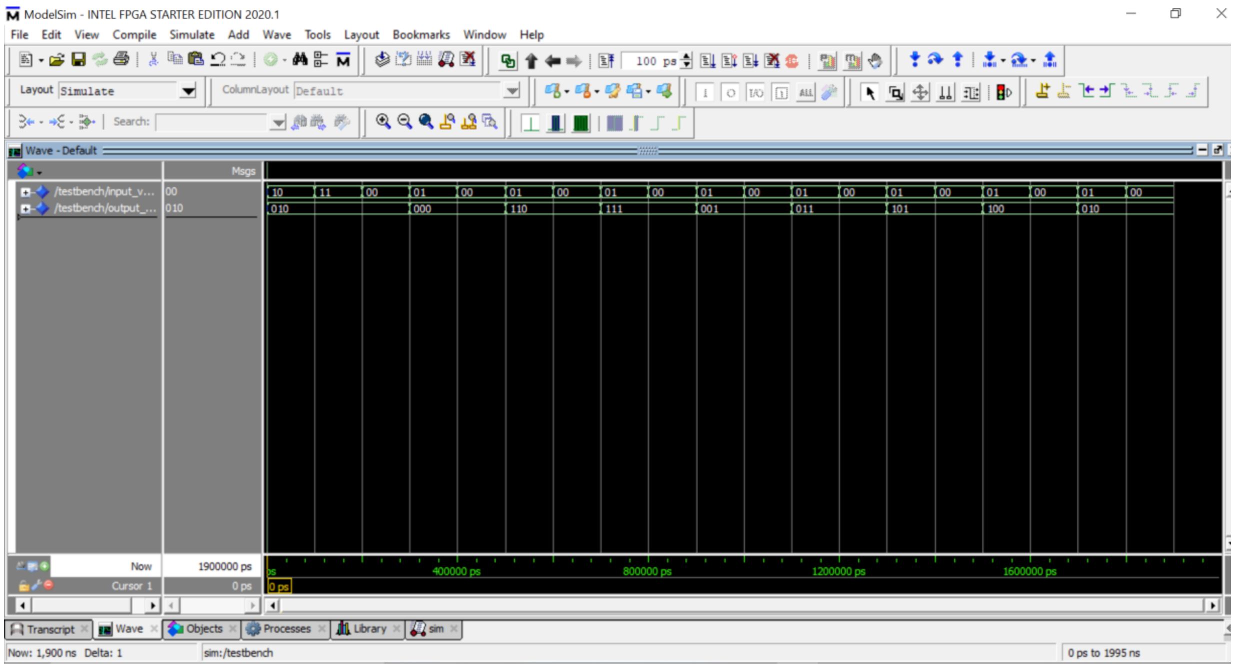
RTL Simulation:

STRUCTURAL MODELLING:


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
Search:
Wave - Default
Msgs
Transcript
Wave
Objects
Processes
Library
sim

```
/testbench/input_v...   00    10 11 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
/testbench/output_...   010   010 000 110 111 001 011 101 100 010

Now       1900000 ps    400000 ps  800000 ps  1200000 ps  1600000 ps
Cursor 1  0 ps          0 ps
```

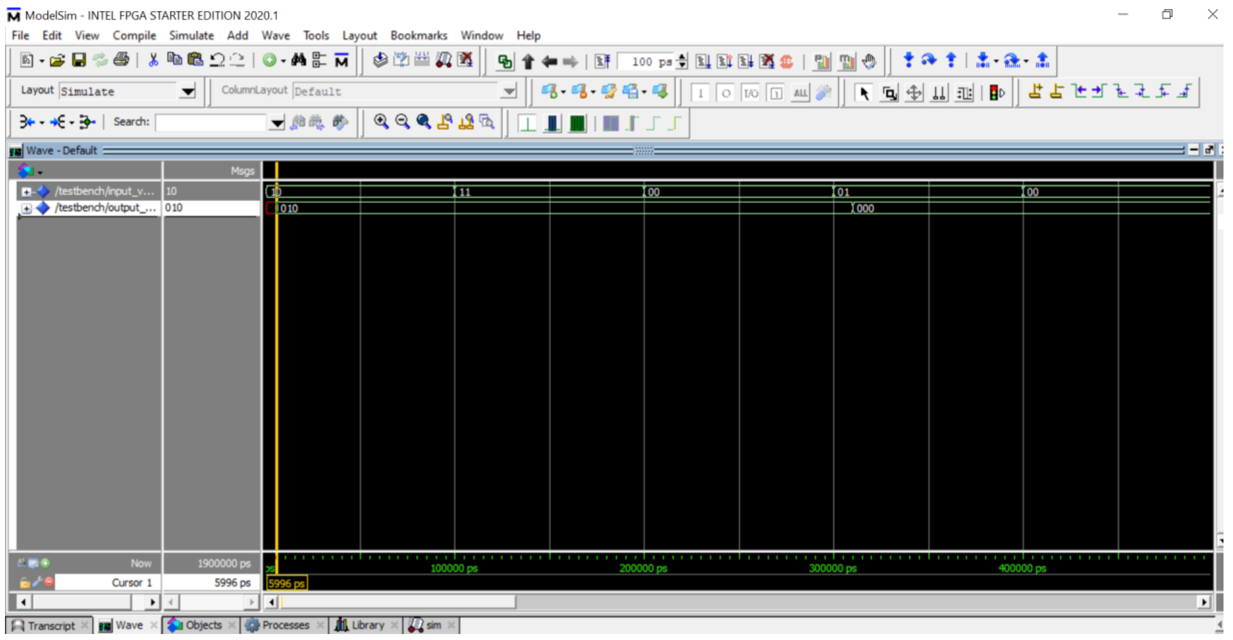
BEHAVIORAL MODELLING:


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
Search:
Wave - Default
Msgs
Transcript
Wave
Objects
Processes
Library
sim

```
/testbench/input_v...   00    10 11 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00
/testbench/output_...   010   010 000 110 111 001 011 101 100 010

Now       1900000 ps    400000 ps  800000 ps  1200000 ps  1600000 ps
Cursor 1  0 ps          0 ps

Now: 1,900 ns  Delta: 1     sim:/testbench     0 ps to 1995 ns
```

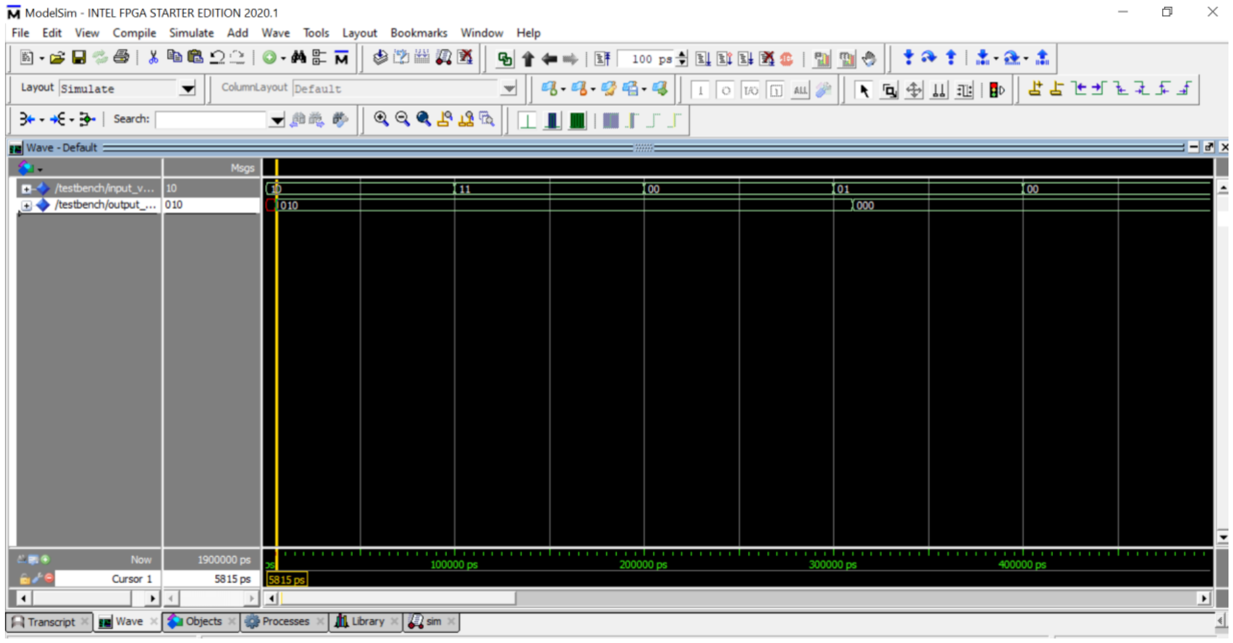
Gate-level Simulation:

STRUCTURAL MODELLING:


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
Search:
Wave - Default
Msgs
/testbench/input_v... 10
/testbench/output_... 010
10
11
00
01
00
010
000
Now 1900000 ps
Cursor 1 5996 ps
100000 ps
200000 ps
300000 ps
400000 ps
5996 ps
Transcript Wave Objects Processes Library sim

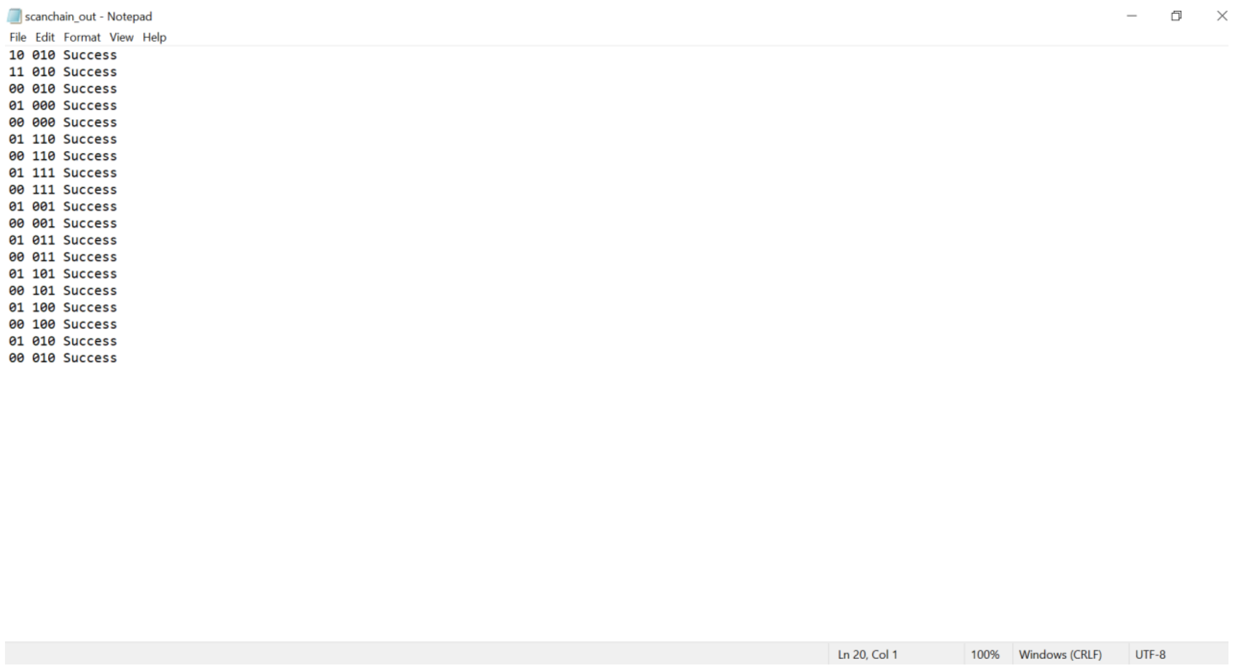
Time delay = 5.996 ns

BEHAVIORAL MODELLING:


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
Search:
Wave - Default
Msgs
/testbench/input_v... 10
/testbench/output_... 010
10
11
00
01
00
010
000
Now 1900000 ps
Cursor 1 5815 ps
100000 ps
200000 ps
300000 ps
400000 ps
5815 ps
Transcript Wave Objects Processes Library sim

Time delay = 5.816 ns

Observations:



```
10 010 Success
11 010 Success
00 010 Success
01 000 Success
00 000 Success
01 110 Success
00 110 Success
01 111 Success
00 111 Success
01 001 Success
00 001 Success
01 011 Success
00 011 Success
01 101 Success
00 101 Success
01 100 Success
00 100 Success
01 010 Success
00 010 Success
```

We see that for different values of input, the scan chain outputs resulted in “Success” i.e., the scan chain outputs matches with the tracefile outputs (for both Structural and Behavioral modelling)