Experiment 5: sequence generator

Narne Avinash Chowdary , Roll Number-200070047

EE-214 WEL, IIT Bombay

October 2, 2021

Overview of the experiment:

Sequence generator :

In this experiment we have to generate the given sequence shown in fig1 by using flipflops and inputs as clock and reset outputs are 3 bit number.

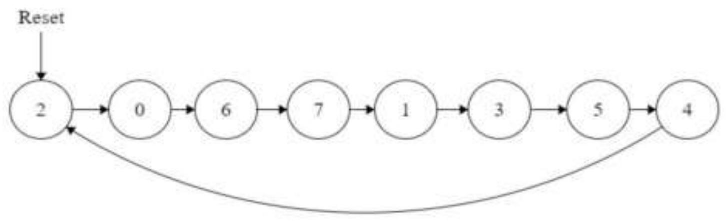
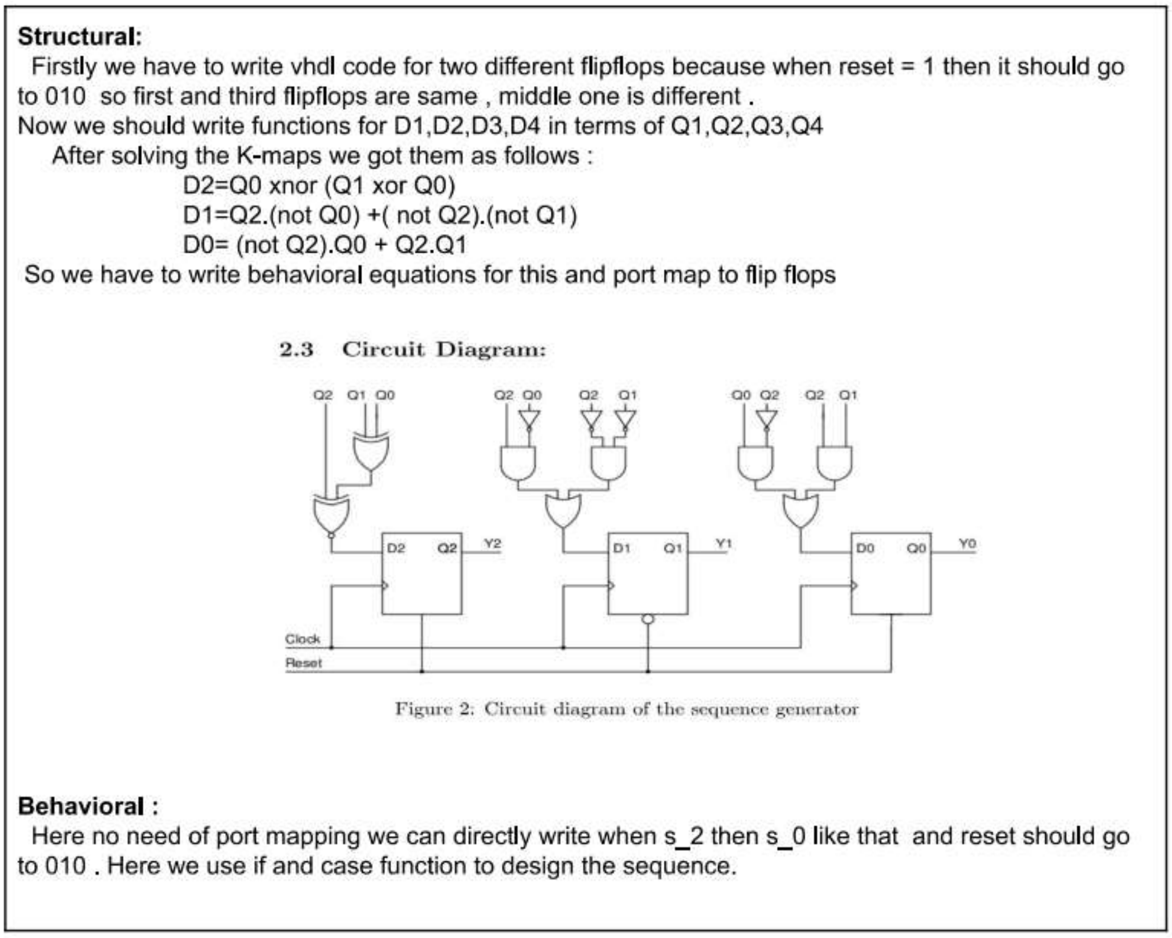


Figure 1: Sequence generator

> In this lab we have to do both structural and behavioral models .
> In structural model we have to do port maps for flipflops and also we have to define functions for flip flop inputs in terms for signals
> Where as in behavioral one we can directly write by using case and if function.

Approach to the Experiment:



Structural:

Firstly we have to write vhdl code for two different flipflops because when reset = 1 then it should go to 010 so first and third flipflops are same , middle one is different .
Now we should write functions for D1,D2,D3,D4 in terms of Q1,Q2,Q3,Q4
After solving the K-maps we got them as follows :

D2=Q0 xnor (Q1 xor Q0)
D1=Q2.(not Q0) +(not Q2).(not Q1)
D0= (not Q2).Q0 + Q2.Q1

So we have to write behavioral equations for this and port map to flip flops

2.3 Circuit Diagram:

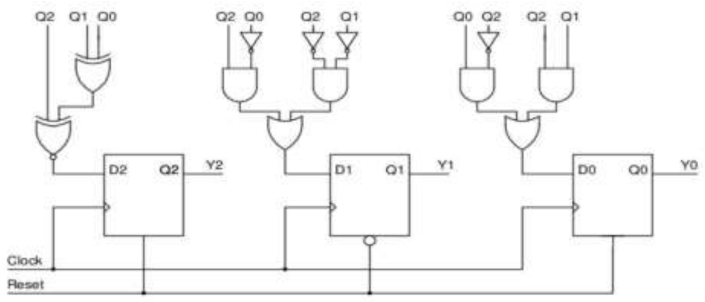


Figure 2: Circuit diagram of the sequence generator

Behavioral :

Here no need of port mapping we can directly write when s_2 then s_0 like that and reset should go to 010 . Here we use if and case function to design the sequence.

Design document and VHDL code:

Structural:

```
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
-- write the equations here
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<= ((Q(0) and (not Q(2))) or (Q(2) and Q(1)));
y(2)<=Q(2);
y(1)<= Q(1);
y(0)<=Q(0);

-- Do the port mapping                          --asynchronous reset
--Q0
dff_0  : dff2 port map(D0,clock,reset,Q(0));

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

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



end struct;
```

Behavioral:

```
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
--state binary encoding
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";
--write the remaining constant declarations
begin
-- process for next state and output logic
reg_process: process(clock,reset)
```

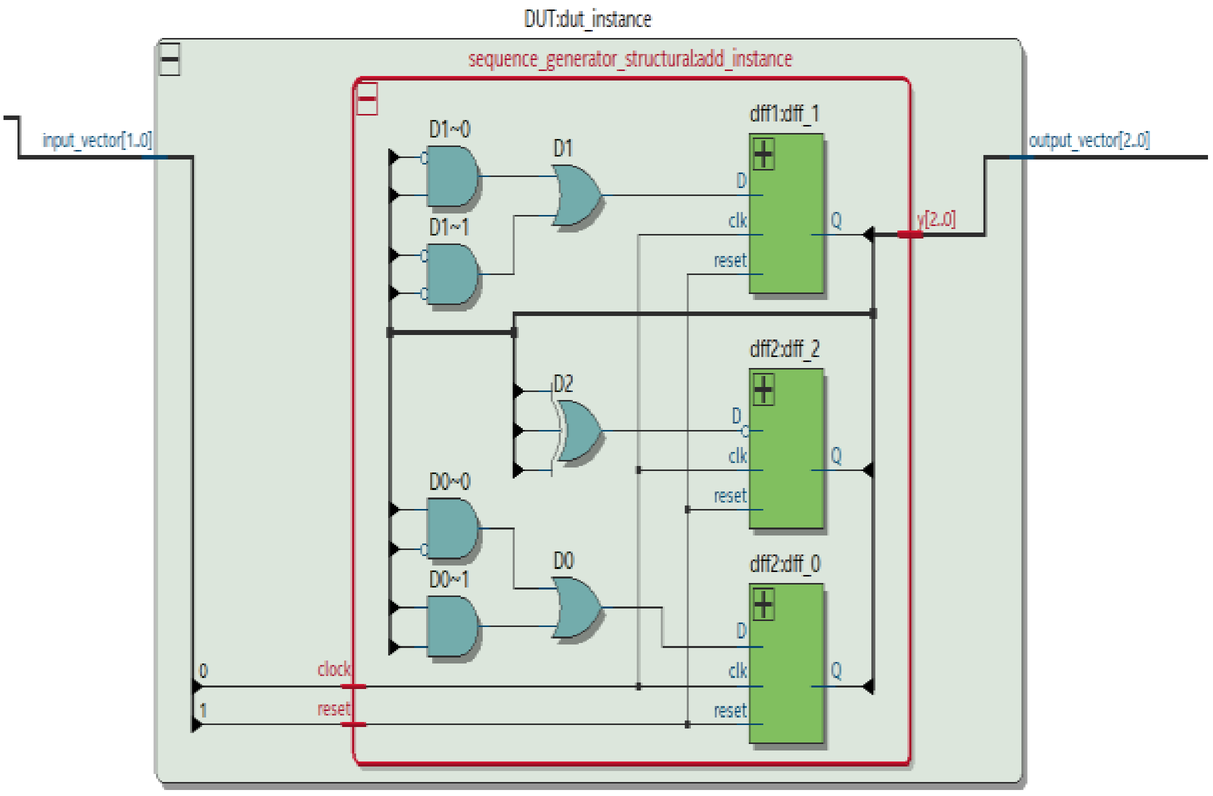
```
begin
if(reset='1')then
state<=s_2; -- write the reset state
elsif(clock'event and clock='1')then
case state is
      --reset
      when s_2=>
     state<=s_0;


            when s_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 s_0=>
     state<=s_6;
     -- write the remaining choices
```

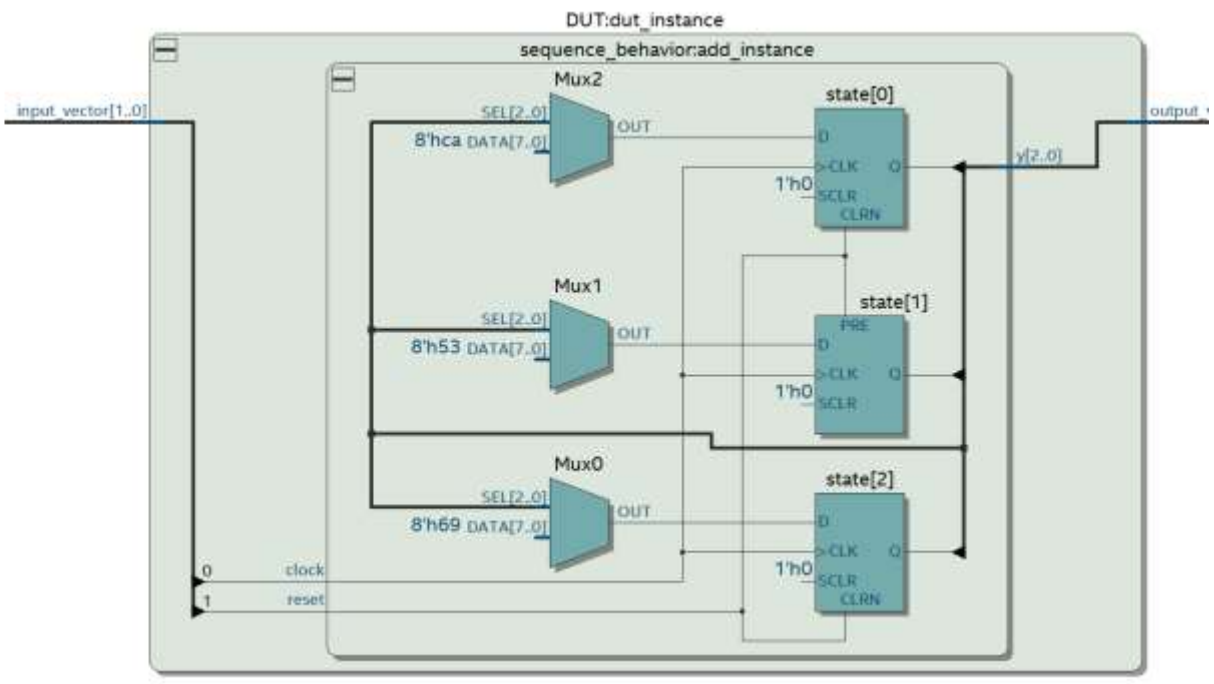
```
         --DEFAULT CASE
       when others=>
         state<="010";-- write the reset state
       end case;
end if;
end process reg_process;
-- output logic concurrent statemet or one more process
y<=state;
end behav;
```

RTL VIEW:

Structural:



Behavioral:



DUT Input/Output Format:

reset => input_vector(1),

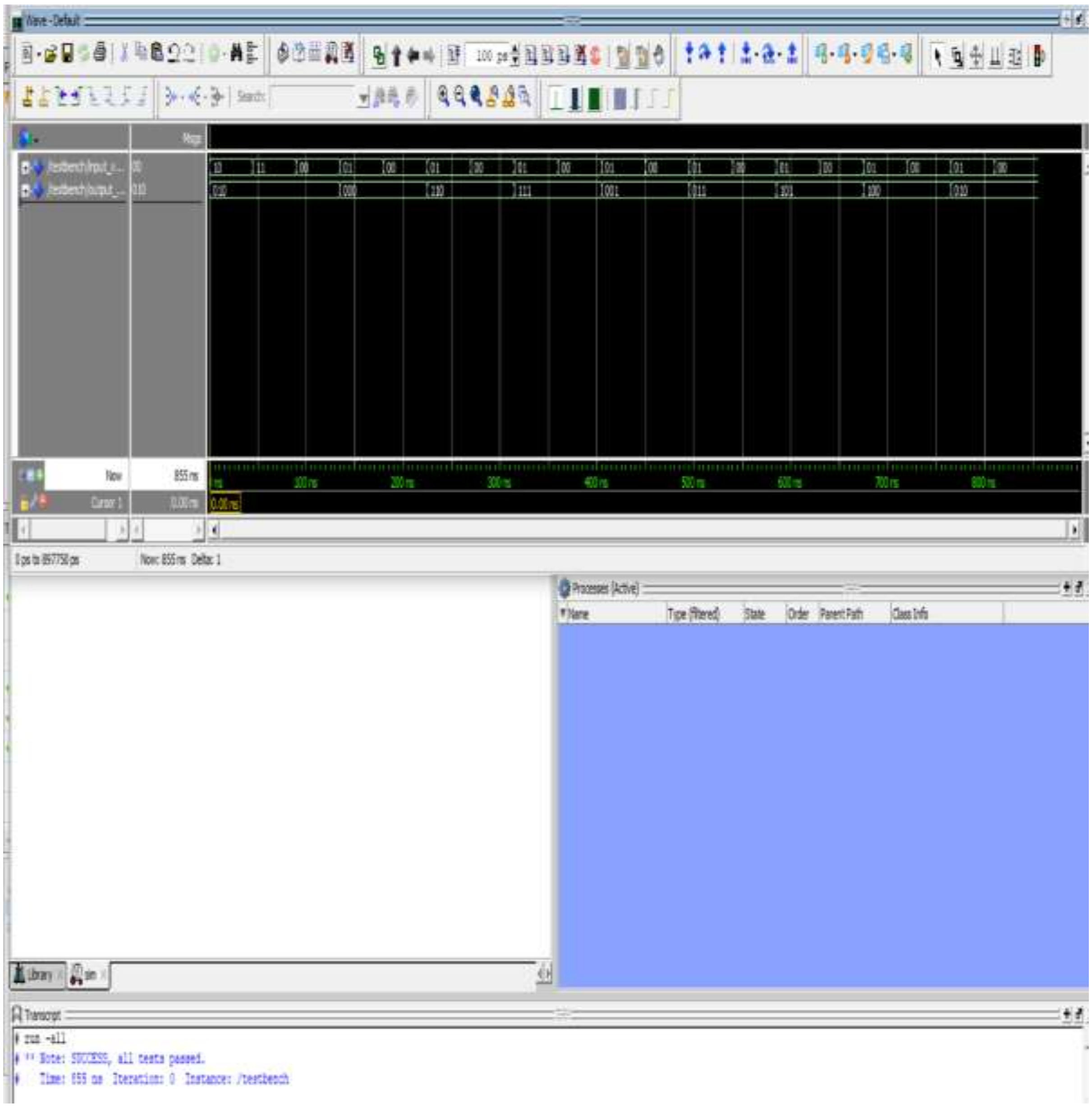
clock=> input_vector(0),

y(2)=>output_vector(2),

y(1)=>output_vector(1),

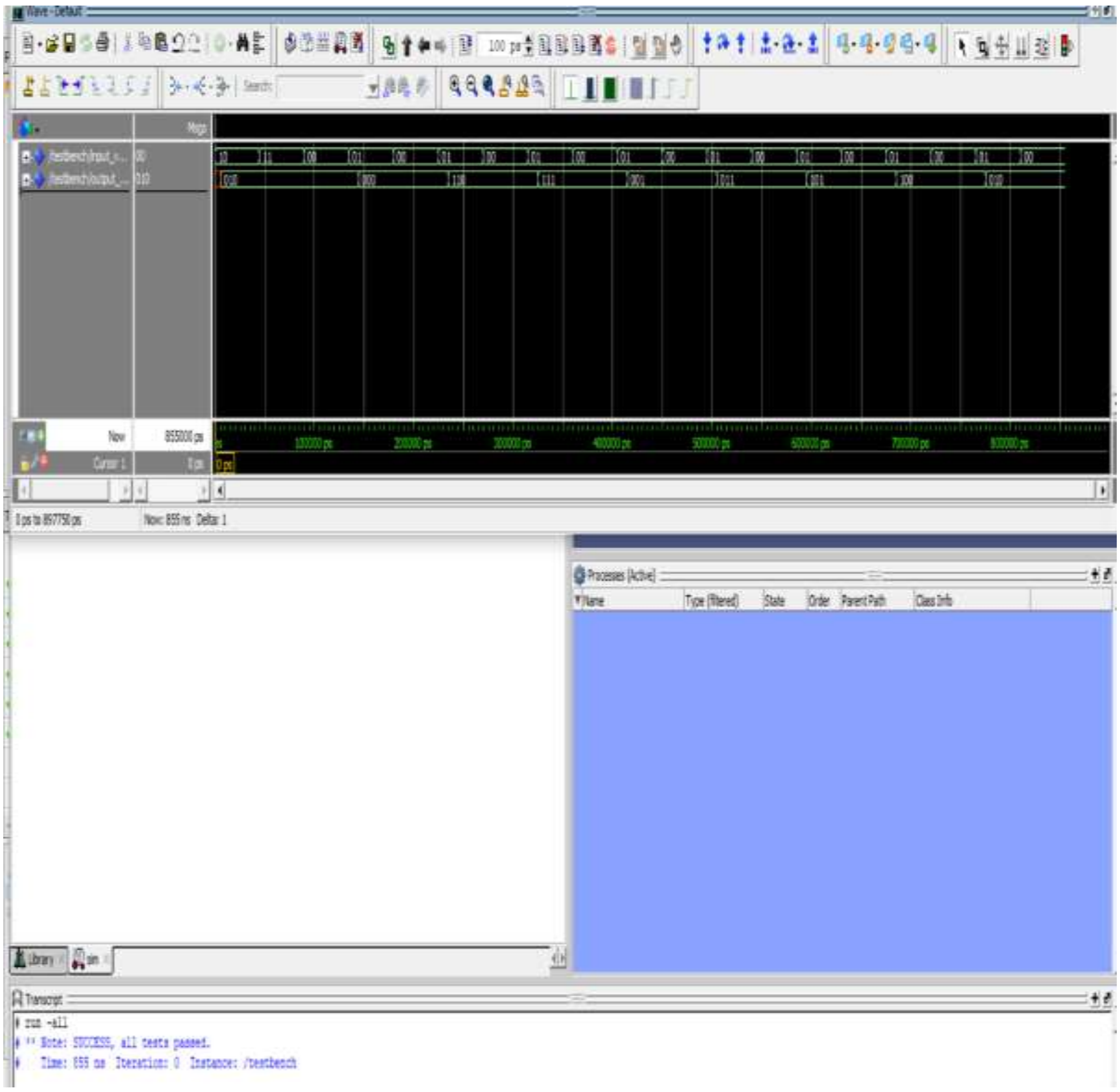
y(0)=>output_vector(0)

RTL SIMULATION:



```
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 855 ns  Iteration: 0  Instance: /testbench
```

Gate – Level Simulation:



```
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 855 ns  Iteration: 0  Instance: /testbench
```

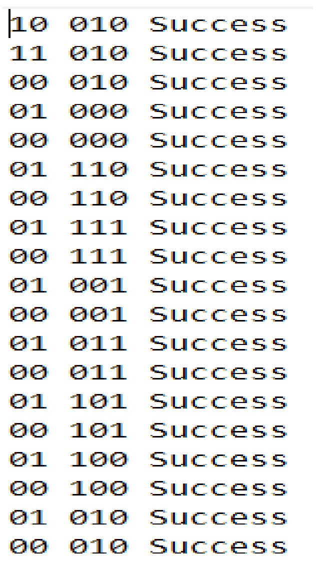
Krypton Board:

After completion of simulation on Quartus, the sequence generator is implemented on Krypton board using UrJTAG. Later we verify this using ScanChain and ideal values that Krypton board implement to this circuit. We confirm whether the design is correctly implemented while implementing UrJATG and the svf file obtained.

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.



```
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
```