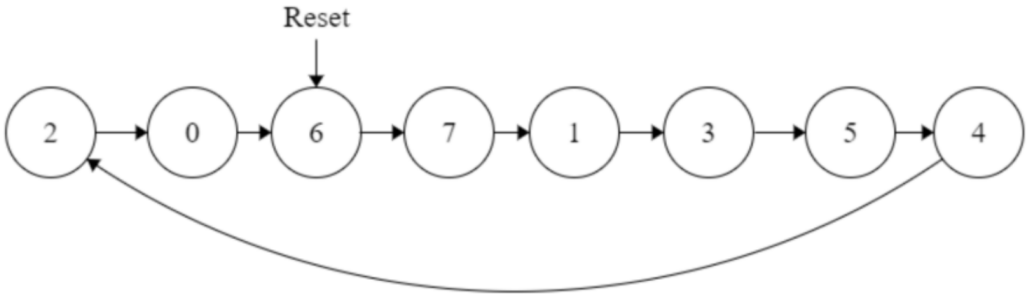
# Experiment 3: Sequential Circuit – 1

Prateek Garg
20D070060
EE-214, WEL, IIT Bombay
September 29th, 2021

## Overview of the experiment:

**Main Objectives:**

- Design a circuit which generates a certain sequence upon every input clock pulse and on reset the sequence will into default sequence.



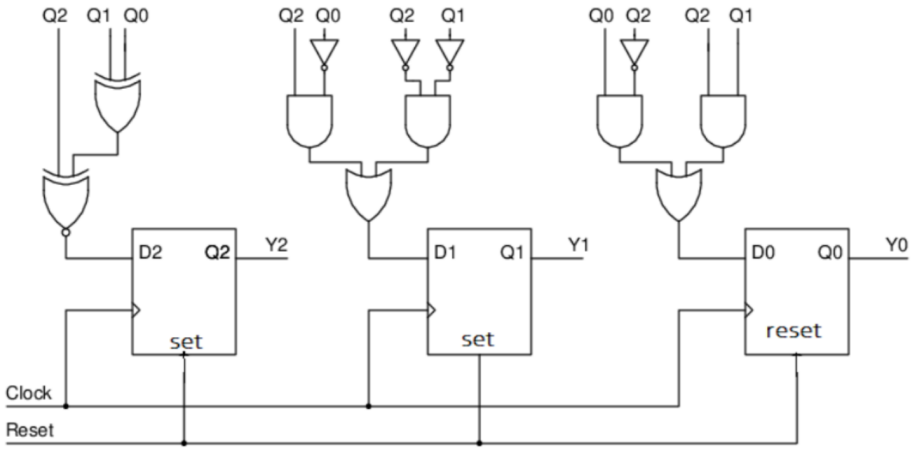

- Describe the designed circuit in vhdl using both Structural and Behavioral Modelling.
- Simulate the design using the generic testbench in ModelSim.
- Test the correctness of design using Scanchain.

The design was described in VHDL using the software Quartus Prime. For structural modelling, components were instantiated and port mapping is used to describe connections. For behavioral modelling, the process with a sensitivity list was described inside the architecture. The design was then simulated using ModelSim checked against every possible testcase. Then the design was simulated on Krypton board and checked against every possible test case using Scanchain.

## Approach to the experiment:

### Design:

#### Structural:




We proceed by making a state table in which we map the present state to the next state. Then using this tale we determine the inputs to be given to D0,D1,D2 using K-map minimisation.
The reset functionality is implemented using the set and reset options of the flip flops. We choose our flip-flops in such a way that whenever reset is high 2 flip-flops are high and other is low. So we get 110= 6 at reset. The flip-flops are designed using behavioral modelling.
For the flip-flops, we instantiated a process inside the architecture which contains a sensitivity list with clock and set/reset option included. When any of these values change the process will rerun to evaluate the output.
Flip-flops were then instantiated inside the architecture of the sequence generator and port mapping was used to describe the connections.
.

#### Behavioral:

For behavioral modelling we proceed by describing the entity and then architecture. In the architecture we create a list of logic vectors corresponding to the sequence to be generated.
And then, we instantiated a process inside the architecture which contains a sensitivity list with clock and set/reset option included. When any of these values change the process will rerun to evaluate the output. The process is described as follows: We first look at reset. If it's high then output is default value "110"=6. Else when there's a positive edge in the clock we change the value of output depending upon the present value at the output and next value is assigned from the sequence to be generated.

## Design document and VHDL code:

### Structural:

#### Sequence-Generator:

```
entity Sequence_generator_struct is
    port (
        reset:   in std_logic;
        clock:   in std_logic;
        Y:   out std_logic_vector(2 downto 0)
    ) ;
end Sequence_generator_struct;

architecture structa of Sequence_generator_struct is
  component dff_set is port(
                    D   :in std_logic;
                    clock  :in std_logic;
                    set    :in std_logic;
                    Q      :out std_logic
                    );
  end component dff_set;

  component dff_reset is port(
                    D   :in std_logic;
                    clock  :in std_logic;
                    reset    :in std_logic;
                    Q      :out std_logic
                    );
  end component dff_reset;

signal D: std_logic_vector(2 downto 0);
signal Q: std_logic_vector(2 downto 0);

begin
  dff_0: dff_reset port map(D => D(0) ,clock=>clock ,reset=> reset
,Q=>Q(0));
  dff_1: dff_set port map(D => D(1) ,clock=>clock ,set=> reset
,Q=>Q(1));
  dff_2: dff_set port map(D => D(2) ,clock=>clock ,set=> reset
,Q=>Q(2));
```

```
D(2) <= Q(2) xnor (Q(1) xor Q(0));
D(1) <= (Q(2) and not(Q(0))) or (not(Q(1)) and not(Q(2)));
D(0) <= ( ( Q(0) and not(Q(2)) ) or ( Q(2) and Q(1) ));

Y <= Q;

end structa ;
```

**Flip-Flops:**

```
library ieee;
use ieee.std_logic_1164.all;
package Flipflops is
    component dff_set is port(
                                        D     :in std_logic;
                                        clock  :in std_logic;
                                        set    :in std_logic;
                                        Q      :out std_logic
                                        );
    end component dff_set;

    component dff_reset is port(
                                        D     :in std_logic;
                                        clock  :in std_logic;
                                        reset    :in std_logic;
                                        Q      :out std_logic
                                        );
  end component dff_reset;

end package Flipflops;




library ieee;
use ieee.std_logic_1164.all;
entity dff_set  is port(
                              D        :in std_logic;
                              clock  :in std_logic;
                         set    :in std_logic;
```

```
                                Q       :out std_logic
                               );
end entity dff_set;

architecture struct of dff_set is
begin
dff_set_proc: process (clock,set)
begin
    if(set='1')then
    Q <= '1';
    elsif (clock'event and (clock='1')) then
    Q <=D;
    end if;

end process dff_set_proc;
end struct;
library ieee;
use ieee.std_logic_1164.all;
entity dff_reset    is port(
                                        D     :in std_logic;
                                        clock  :in std_logic;
                                        reset    :in std_logic;
                                        Q       :out std_logic
                                        );
end entity dff_reset;

architecture struct of dff_reset is
begin
dff_reset_proc: process (clock,reset)
begin
    if(reset='1')then
    Q <= '0';
    elsif (clock'event and (clock='1')) then
    Q <=D;
    end if;

end process dff_reset_proc;
end struct;
```

## Behavioral:

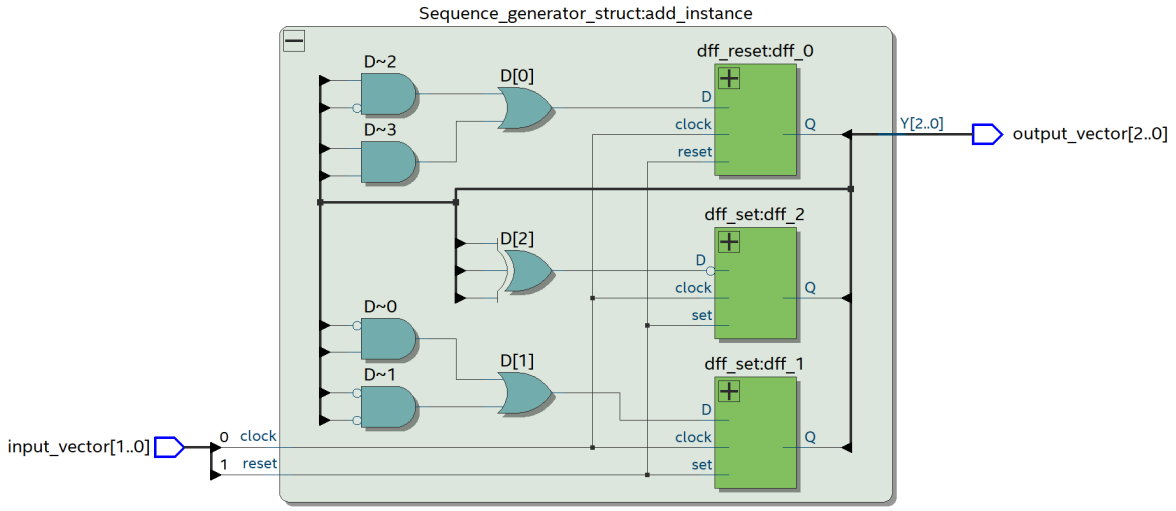
### Sequence-Generator:

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

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

end entity sequence_generator_behavior;

architecture behav  of sequence_generator_behavior is

signal state:std_logic_vector(2 downto 0);

constant s_6:std_logic_vector(2 downto 0):="010";
constant s_7:std_logic_vector(2 downto 0):="000";
constant s_0:std_logic_vector(2 downto 0):="110";
constant s_1:std_logic_vector(2 downto 0):="111";
constant s_2: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):="101";
constant s_5:std_logic_vector(2 downto 0):="100";

begin

reg_process: process(clock,reset)

begin
if(reset='1')then
    state<= s_0 ; -- write the reset state
elsif(clock'event and clock='1')then

    case state is
      when s_6=>
        state<=s_7;
        when s_7=>
```

```
        state<=s_0;
        when s_0=>
        state<=s_1;
        when s_1=>
        state<=s_2;
        when s_2=>
        state<=s_3;
        when s_3=>
        state<=s_4;
        when s_4=>
        state<=s_5;
        when s_5=>
        state<=s_6;

--this is needed  because std_logic can take values other than 0,1 i.e
high impedance
      when others=>
        state<= s_0;
      end case;
end if;
end process reg_process;
-- output logic concurrent statement or one more process
y<=state;
end behav;
```

# RTL View:

## Structural:




## Behavioral:

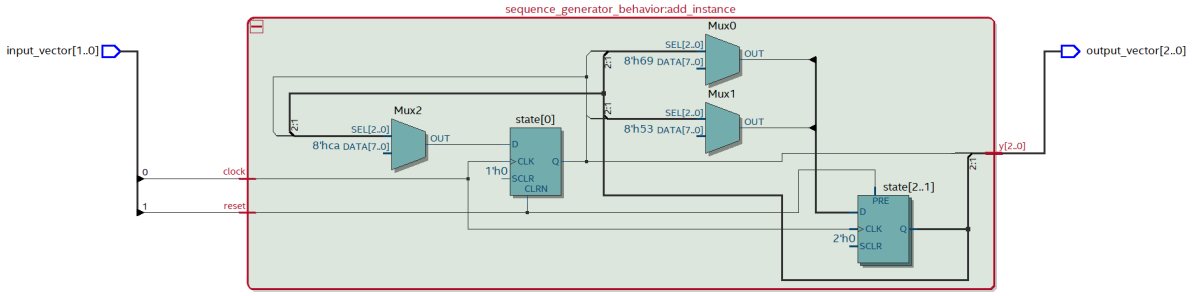



Fig: RTL View.

## DUT Input/Output Format:

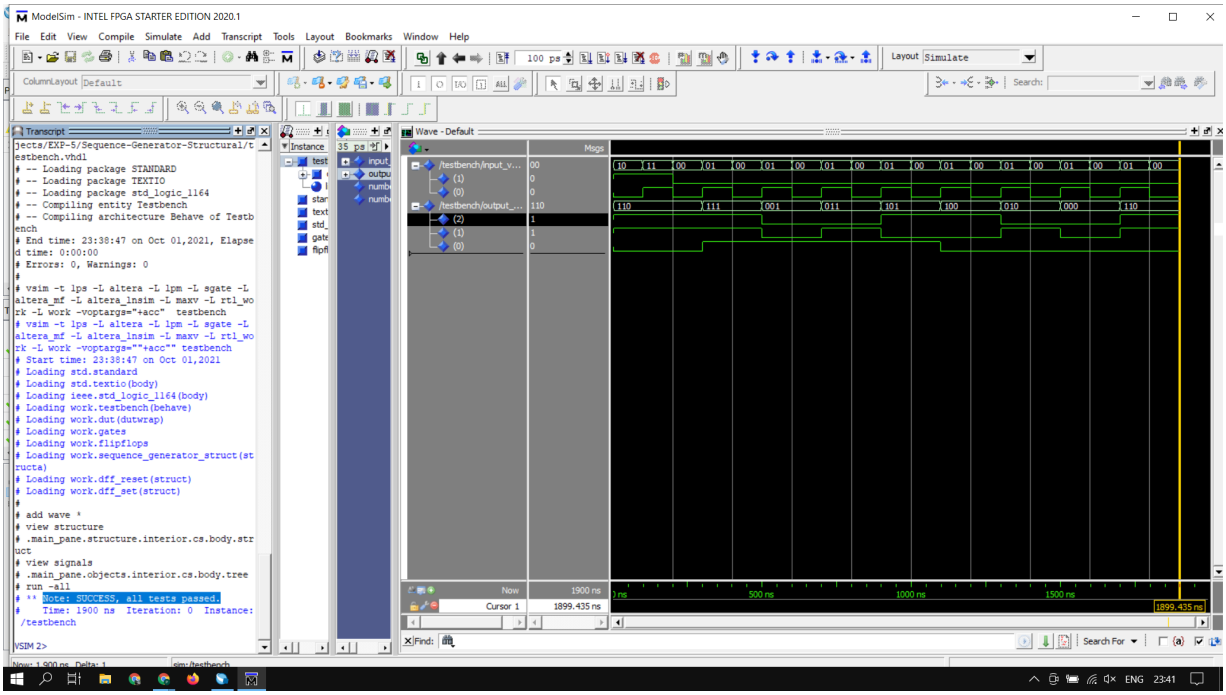
### Port map:

```
                        -- order of inputs
                        reset   => input_vector(1),
                        clock   => input_vector(0),
                        -- order of outputs
                        Y       => output_vector
```

```
Some test cases from TRACEFILE:
TRACEFILE format
                    < reset clock >< y2y1y0 > 111

                              00 001 111
                              01 011 000
                              00 011 111
                              01 101 000
                              00 101 111
                              01 100 000
                              00 100 111
                              01 010 000
                              00 010 111
                              01 000 000
                              00 000 111
                              01 110 000
                              00 110 111
```

# RTL Simulation:

## Structural:




## Behavioral:

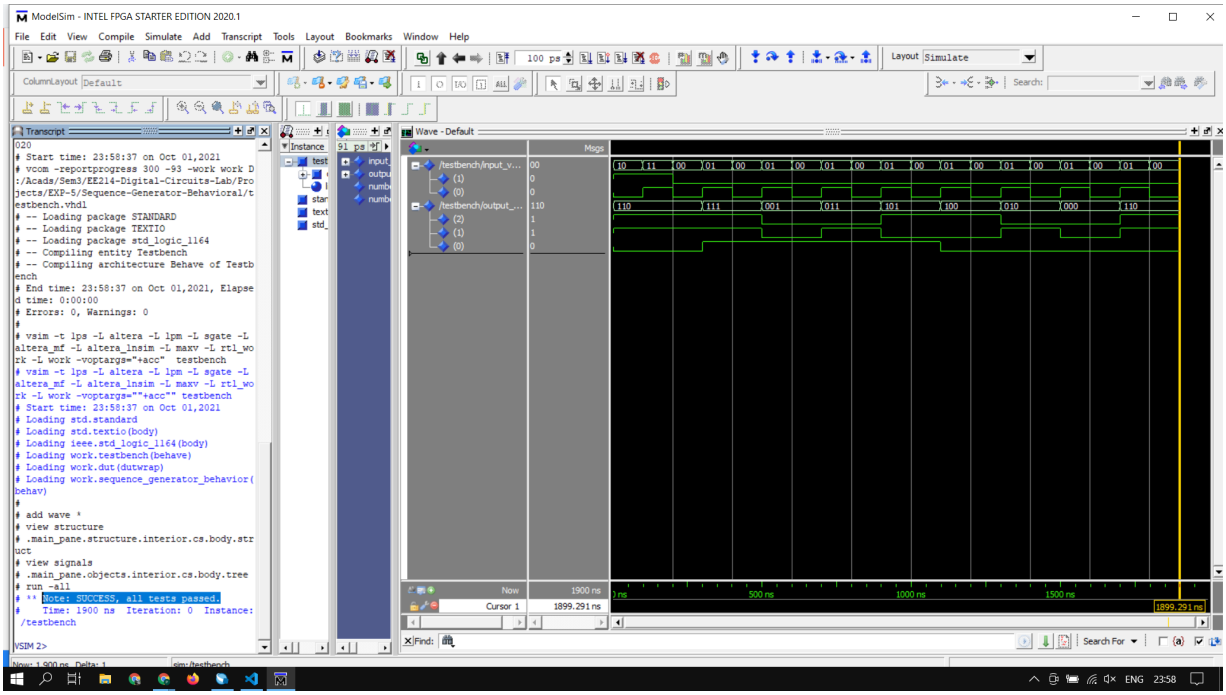
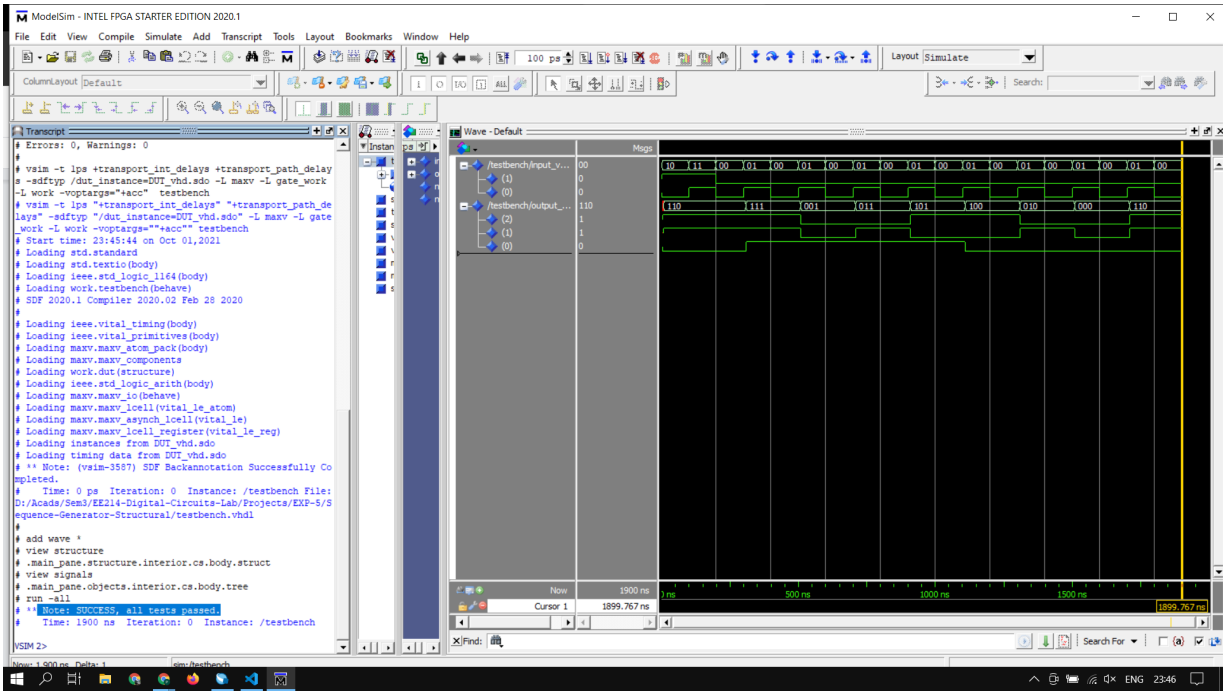



Fig: RTL Simulation

# Gate-level Simulation:

## Structural:




## Behavioral:

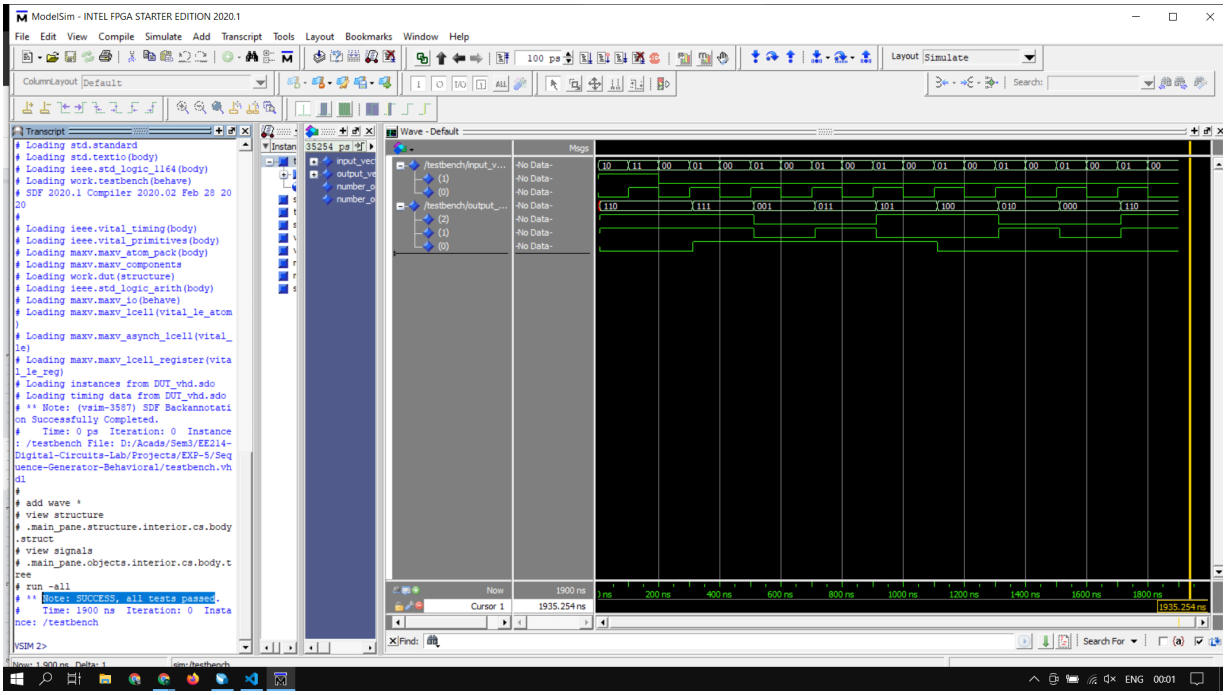



Fig: Gate Level Simulation

## Observation:

Parts of the file out.txt generated by scanchain:

| Structural: | Behavioral: |
|---|---|
| 10 110 Success | 10 110 Success |
| 11 110 Success | 11 110 Success |
| 00 110 Success | 00 110 Success |
| 01 111 Success | 01 111 Success |
| 00 111 Success | 00 111 Success |
| 01 001 Success | 01 001 Success |
| 00 001 Success | 00 001 Success |
| 01 011 Success | 01 011 Success |
| 00 011 Success | 00 011 Success |
| 01 101 Success | 01 101 Success |
| 00 101 Success | 00 101 Success |
| 01 100 Success | 01 100 Success |
| 00 100 Success | 00 100 Success |
| 01 010 Success | 01 010 Success |

## References:


1. J.F.Wakerly: Digital Design, Principles and Practices,4th Edition,Pearson Education, 2005