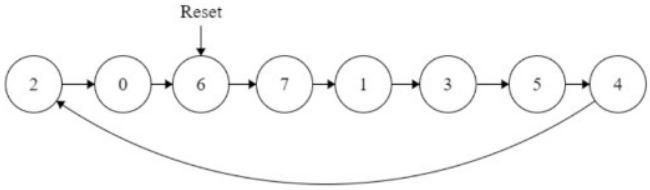
Experiment 5: A Sequence Generator

Karrthik Arya Roll Number 200020068
EE-214, WEL, IIT Bombay
September 29, 2021

Overview of the experiment:

This week we were suppose to make a sequence generator using structural as well as behavioural modelling. It was supposed to generate a certain sequence upon every input clock pulse and on reset the sequence should go into default sequence. This is the sequence that was to be generated.



Approach to the experiment:

For behavioural modelling the process block had to be run when either there was a clock'event from 0 to 1 or reset was set as 1. Thus clock and reset had to go in the sensitivity list. For generating the sequence we had to see what was the earlier output and where is it supposed to go from that output. This should happen when clock goes from 0 to 1. Then we need to have a separate case when reset is 1 where the output should go to the default state which is 6 here. With this logic circuit can be coded using behavioural modelling.

For structural modelling we need to use filpflops. So the firsti thing to do is create flipflop components. We need both set and reset flipflops. Set flipflop is when the reset pin goes to 1 the output becomes 1 irrespective of the clock and reset flipflop is when the reset pin goes to 1 the output becomes 0. We use behavioural modelling to describe these components. The skeleton code was given where we had to only complete the circuit for reset flipflop.

We will need 3 flipflops for each of the 3 bits. For bits 2 and 1 the flipflops should be set and for bit 0 it should be reset. This should be since when reset is 1 it should generate sequence 6 i.e. "110". Now to get the circuits for the data pin we draw the excitation table.

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

Using this we can get the boolean expressions for D2,D1,D0 and draw the circuits for each of them. These will go as input to the data pins in the flipflops. We can draw the kmaps and get the boolean expressions.

Q_2 \ Q_1Q_0	00	01	11	10
0	1	0	1	0
1	0	1	0	1

$D_2 = Q_2 \text{ xnor } (Q_1 \text{ xor } Q_0)$

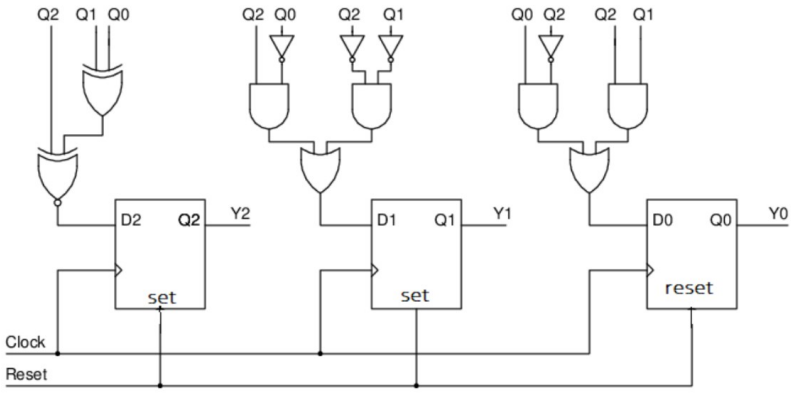
Q_2 \ Q_1Q_0	00	01	11	10
0	1	1	0	0
1	1	0	0	1

$D_1 = Q_2.\overline{Q_0} + \overline{Q_2}.\overline{Q_1}$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$D_0 = \overline{Q_2}.Q_0 + Q_2.Q_1$

Now using these expressions we can draw the circuit for each of the data inputs and draw the complete circuit. This would look like this:



Here each of the block here is a flipflop set or reset.Now for structural modelling we just have to code this in vhdl.

Design document and VHDL code if relevant:

For behavioural modelling the code of the main process block is this:

```
if(reset='1')then
```

```
state<= "110"; -- write the reset state
elsif(clock'event and clock='1')then
case state is
     when s_6=>
               state<=s_7;
          when s_0 =>
               state<= s_6;
               when s_1 =>
               state<=s_3;
               when s_2 =>
               state<=s_0;
               when s_3 =>
               state<=s_5;
               when s_4 =>
               state<=s_2;
               when s_5 =>
               state<=s_4;
               when s_7 =>
               state<=s_1;
--this is needed  because std_logic can take values other than 0,1 i.e high imepedance
      when others=>
        state<= "110"; -- write the reset state
      end case;
end if;
end process reg_process;
```

The same logic described above is coded here. Here I had declared s_1,s_2,s_3...s_7 as the states corresponding to each of the sequences. So when the clock goes from 0 to 1 it sees which state was it in earlier and accordingly goes to the next one. Before checking for the clock'event it also checks for the reset pin. If that is 1 then it directly gives the default state 6 as the output. This is the main code logic for behavioural modelling.

For structural modelling we first describe the flipflops.
Here is the code for that:
For set flipflop

```
        entity dff_set is port(D,clock,set:in std_logic;Q:out std_logic);
        end entity dff_set;

        architecture behav of dff_set is
        begin
        dff_set_proc: process (clock,set)
        begin
        if(set='1')then -- set implies flip flip output logic high
            Q <= '1';
        elsif (clock'event and (clock='1')) then
            Q <= D;
        end if ;
        end process dff_set_proc;
        end behav;
```

For reset flipflop:

```
        entity dff_reset is port(D,clock,reset:in std_logic;Q:out std_logic);
        end entity dff_reset;

        architecture behav of dff_reset is
        begin
        dff_reset_proc: process (clock, reset) --write the sensitivity list
```

```
begin
if(reset='1')then -- reset implies flip flip output logic low
    Q <= '0' ; -- write the D Flipflop output when reset
elsif (clock'event and (clock='1')) then
    Q <= D; ---- write the D Flipflop output when posedge clock is triggered end if ;
end if;
end process dff_reset_proc;
end behav;
```

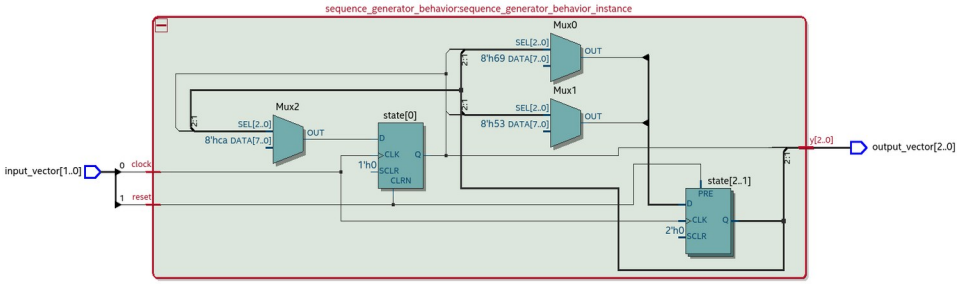
Here the logic described above has just been coded to make the 2 types of flipflops.
We then use these components to code in the complete circuit like this:

```
architecture struct of Sequence_generator_stru_dataflow is
signal D :std_logic_vector(2 downto 0); -- D flip flop inputs
signal Q:std_logic_vector(2 downto 0);  -- D flip flop outputs
begin

-- write the eqations in dataflow  e.g z=a+bc written z<= a or (b and c)
D(2)<= Q(2) xnor (Q(1) xor Q(0));

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

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

y(2 downto 0)<= Q(2 downto 0);

--Q0
dff_0  : dff_reset port map(D => D(0), clock => clock, reset => reset, Q => Q(0));

--Q1
dff_1  : dff_set port map(D => D(1), clock => clock, set => reset, Q=> Q(1));

--Q2
dff_2  : dff_set port map(D => D(2), clock => clock, set => reset, Q=>Q(2));
end struct;
```

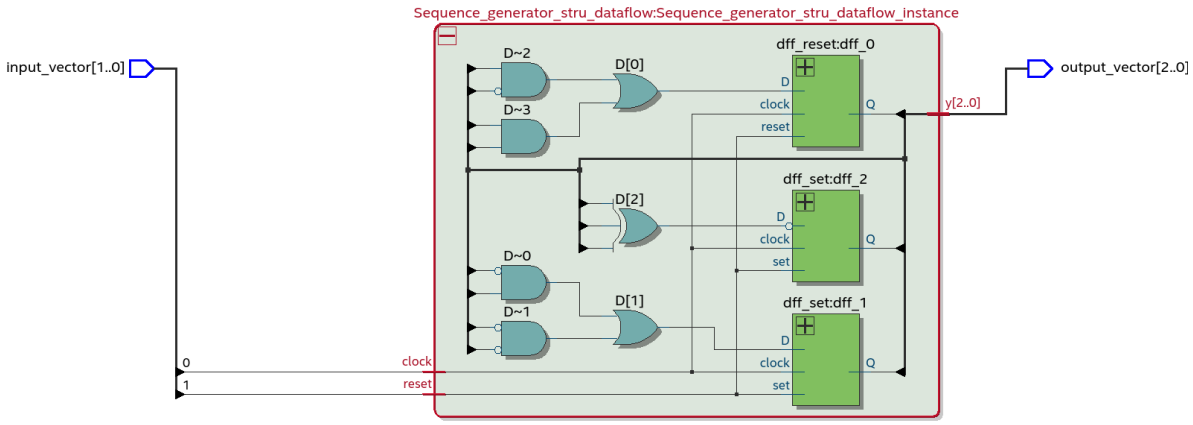
Using the boolean expressions that we got we code in for D(2), D(1) and D(0). We use the signa Q to store the outputs so that they can be used the next time to go to the next sequence. Dff_1,2,3 are the flip flops that have been declared with the logic that we described earlier.

RTL View:

For behavioural modelling RTL view looks like:



For structural modelling RTL view looks like:



DUT Input/Output Format:

The input/output format and the tracefile will be same for both of them since we are modelling the same circuit just in different ways.
So the format is : <resetclock> <y2y1y0> 111

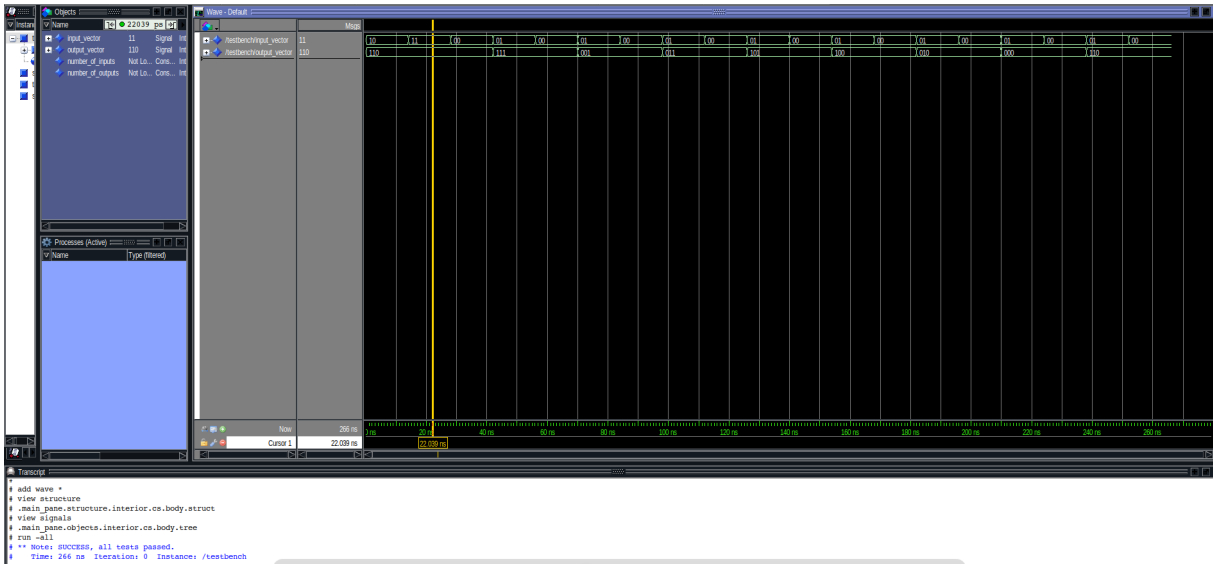
Reset and clock are the input given. Clock is according to which the sequence is supposed to change when it goes from 0 to 1. Reset is to set the sequence back to the default state. Y vector is the output here. 111 are the mask bits to check which cases to verify and which to not. The trace file would look like this:

10 110 000
11 110 000
00 110 111
01 111 000
00 111 111
01 001 000
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

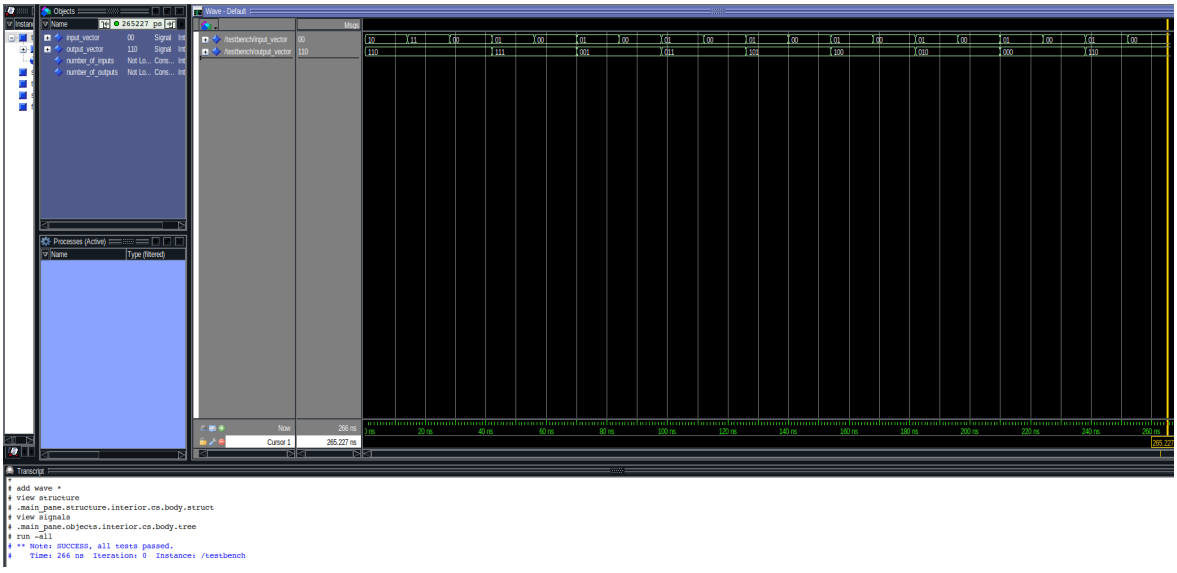
You can see how the sequence changes. You can also see only some of the mask bits are 111 because we are only checking the test cases after a 0 to 1 transition has happened. Since its at thos transitions that the circuit's output changes.

RTL Simulation:

For behavioural modelling:



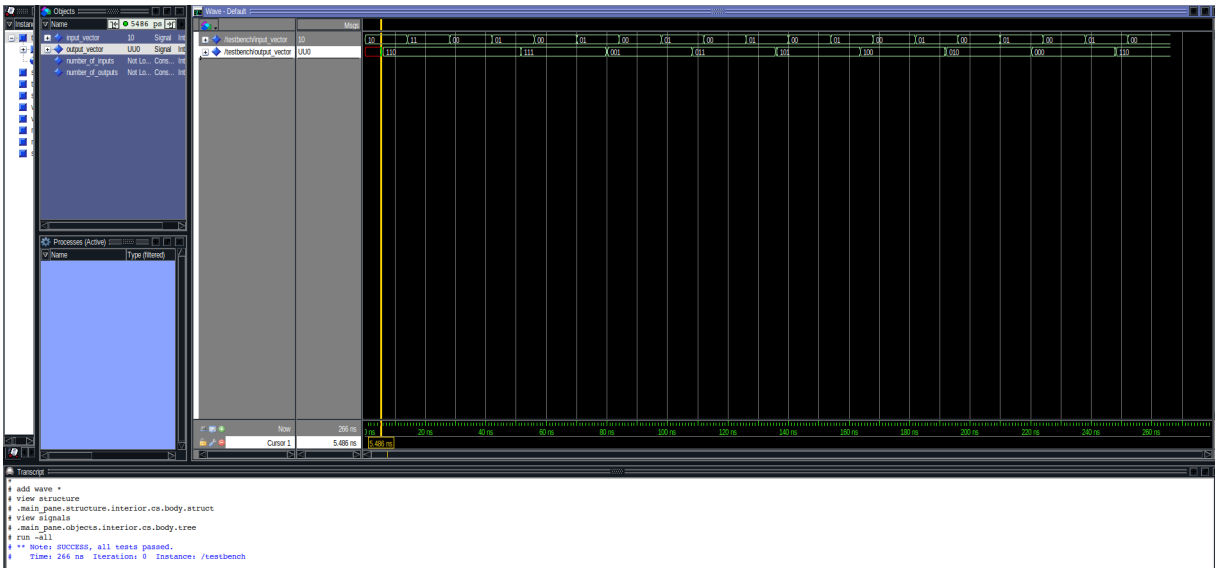
For structual modelling:



Gate-level Simulation:

For behavioural modelling:

For Structural modelling:



Krypton board*:

I used scanchain to test both the designs on the Krypton board. I added the apprорpriate files, set TopLevel.vhdl as the top level entity, compiled and created the svf file. I then uploaded the svf file on the Krypton board and ran the scanchain script.

Observations*:

All the test cases passed with both the designs. Similar out.txt files were generated as below:
10 110 Success
11 110 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
01 000 Success
00 000 Success
01 110 Success
00 110 Success

References:

N/A

* To be submitted after the tutorial on ”Using Krypton.