Experiment 2: Sequential Circuit – 2

Prateek Garg
20D070060
EE-214, WEL, IIT Bombay
August 18th, 2021

Overview of the experiment:

Main Objectives:

- Design a string detector using a Mealy type Finite State Machine which will detect the occurrence of word *'krypton*' in a string of letters. The design accepts a sequence of binary coded alphabets and outputs a '1' if the required word is detected. The letters of krypton can be present anywhere in the string but in sequence.
- Describe the designed circuit in vhdl using 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 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:

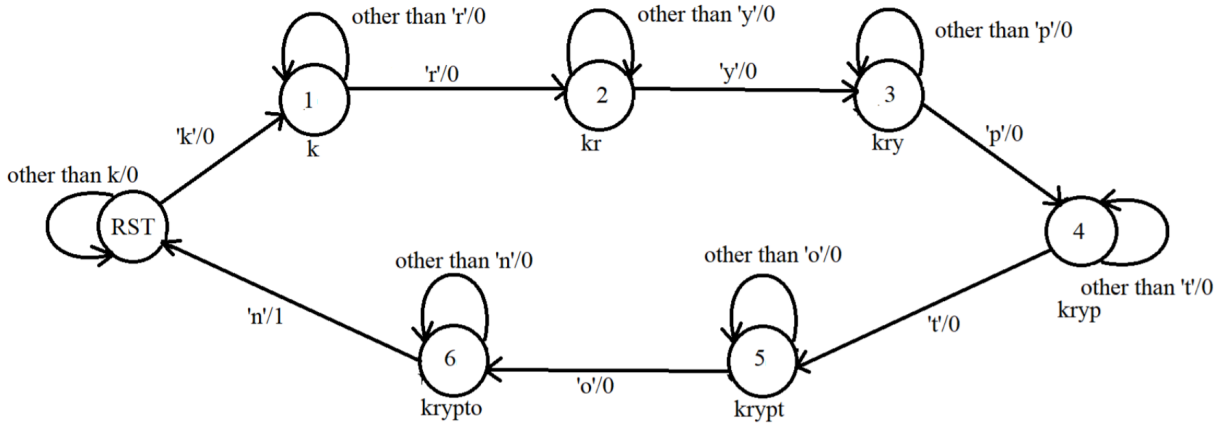

Fig: State Diagram of FSM

We proceed by making a state transition table.

Reset	Input	Present State	Next State	Output
1	X	XXX	RST	0
0	'k'	RST	S1	0
0	'r'	S1	S2	0
0	'y'	S2	S3	0
0	'p'	S3	S4	0
0	't'	S4	S5	0
0	'o'	S5	S6	0
0	'n'	S6	RST	1

Fig: State Transition Table

Each state remembers the letters encountered so far of the word to be detected. If some other letter appears, it remains in the same state and waits for the correct input to arrive.For example, state 2 remembers "kr" letters from the word "krypton". If any input other than y, the next state is chosen as 2.

The alphabets are encoded as 5-bit binary coded input

Design document and VHDL code:

Architecture:

```
entity krypton is
port(inp:in std_logic_vector(4 downto 0);
        reset,clock:in std_logic;
        outp: out std_logic);
end krypton;

architecture rch of krypton is
---------------state types-----------------------------
type state is (rst,s1,s2,s3,s4,s5,s6);
---------------Define signals of state type-----------------------
signal y_present,y_next: state:=rst;

begin
clock_proc:process(clock,reset)
begin
    if(clock='1' and clock' event) then
        if(reset='1') then
            y_present<= rst ;
        else
            y_present<= y_next ;
        end if;
    end if;

end process;

state_transition_proc:process(inp,y_present)
begin
    case y_present is
        when rst=>
            if(unsigned(inp)=11) then    --k
                y_next<= s1;
            else
                y_next <= y_present;
```

```
        end if;
      when s1=>
        if(unsigned(inp)=18) then    --r
            y_next<= s2;
        else
            y_next <= y_present;
        end if;
      when s2=>
        if(unsigned(inp)=25) then    --y
            y_next<= s3;
        else
            y_next <= y_present;
        end if;
      when s3=>
        if(unsigned(inp)=16) then    --p
            y_next<= s4;
        else
            y_next <= y_present;
        end if;
      when s4=>
        if(unsigned(inp)=20) then    --t
            y_next<= s5;
        else
            y_next <= y_present;
        end if;
      when s5=>
        if(unsigned(inp)=15) then    --o
            y_next<= s6;
        else
            y_next <= y_present;
        end if;
      when s6=>
        if(unsigned(inp)=14) then    --n
            y_next<= rst;
        else
            y_next <= y_present;
        end if;

      end case;
```

```
end process;
output_proc: process(y_present,inp)
begin
        if(unsigned(inp)=14 and y_present=s6) then
            outp<='1';
        else
            outp<='0';
        end if;

end process;
end rch;
```

RTL View:

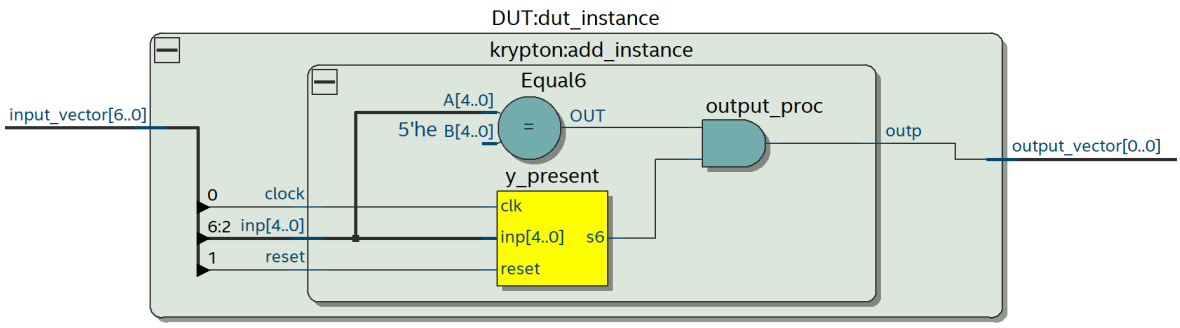

Fig: RTL View.

DUT Input/Output Format:

Port map:

```
-- order of inputs
inp     => input_vector(6 downto 2),
reset   => input_vector(1),
clock   => input_vector(0),
-- order of outputs
outp    => output_vector(0)
```

```
Some test cases from TRACEFILE:
TRACEFILE format
        <5-bit input> <reset> <clock> <space> <output> <space> 1
                                0111000 0 1
                                0111001 0 1
                                0111000 0 1
                                0111001 0 1
                                0110000 0 1
                                0110001 0 1
                                0110000 0 1
                                0110001 0 1
                                0111100 0 1
                                0111101 0 1
                                0111000 1 1
                                0111001 0 1
                                0111000 0 1
                                0111001 0 1
                                0111000 0 1
                                0111001 0 1
```

RTL Simulation:

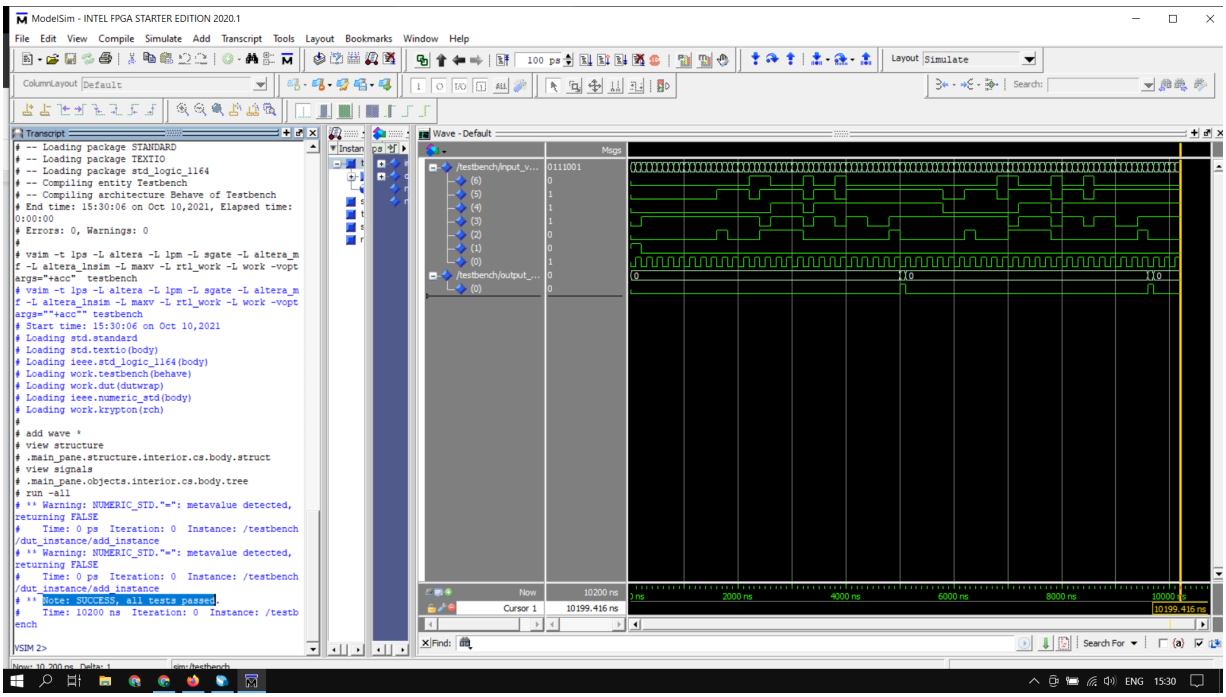

Fig: RTL Simulation

Gate-level Simulation:

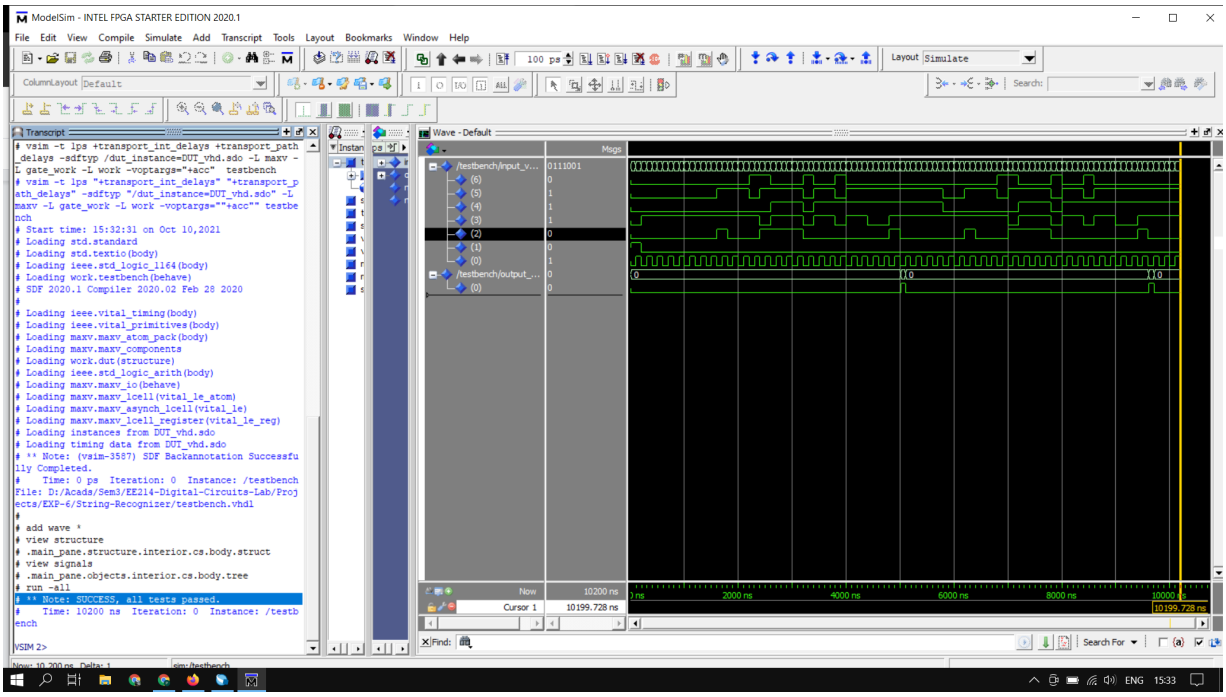

Fig: Gate Level Simulation

Observation:

Parts of the file out.txt generated by scanchain:

0000011 0 Success	1010000 0 Success	0111101 0 Success
0001000 0 Success	1010001 0 Success	0111100 0 Success
0001001 0 Success	0111000 0 Success	0111101 0 Success
0001000 0 Success	0111001 0 Success	1000000 0 Success
0001001 0 Success	0111000 0 Success	1000001 0 Success
0001000 0 Success	0111001 0 Success	0111000 0 Success
0001001 0 Success	0110000 0 Success	0111001 0 Success
0001000 0 Success	0110001 0 Success	0111000 0 Success
0001001 0 Success	0110000 0 Success	0111001 0 Success
0001000 0 Success	0110001 0 Success	1010000 0 Success
0001001 0 Success	0111100 0 Success	1010001 0 Success

References:

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