Experiment 6: String Recognizer

Hardik Panchal
Roll Number 200070054
EE-214, WEL, IIT Bombay
October 5, 2021

Overview of the experiment:

In this experiment, we will design a string detector using a Mealy-type FSM to detect the occurrence of the covid word in a string of letters. The design accepts a sequence of letters coded in binary and outputs a ’1’ if the required word is detected. The letters of covid can be present anywhere in the string but should be in sequence.
e.g.,letters a = ”00001”, b = ”00010” and so on.
For instance, ”lclolvlildl” is the input text then
the output sequence would be ”00000000010”.

Approach to the experiment:

We will make a state transition table from this given below figure of an FSM. In architecture, we have defined three different processes.

State Transition Table

Reset	Input	P.S.	N.S.	Output
1	X	XXX	RST	0
0	'c'	RST	1	0
0	'o'	1	2	0
0	'v'	2	3	0
0	'i'	3	4	0
0	'd'	4	RST	1

First is the clock process, second is the state transition process, and third is the output process.
Each process will run concurrently.
The output will be ‘1’ if we reach state four and encounter ‘d’ as an input. In all other cases, the output is ‘0’.

Design document and VHDL code if relevant:

Design:

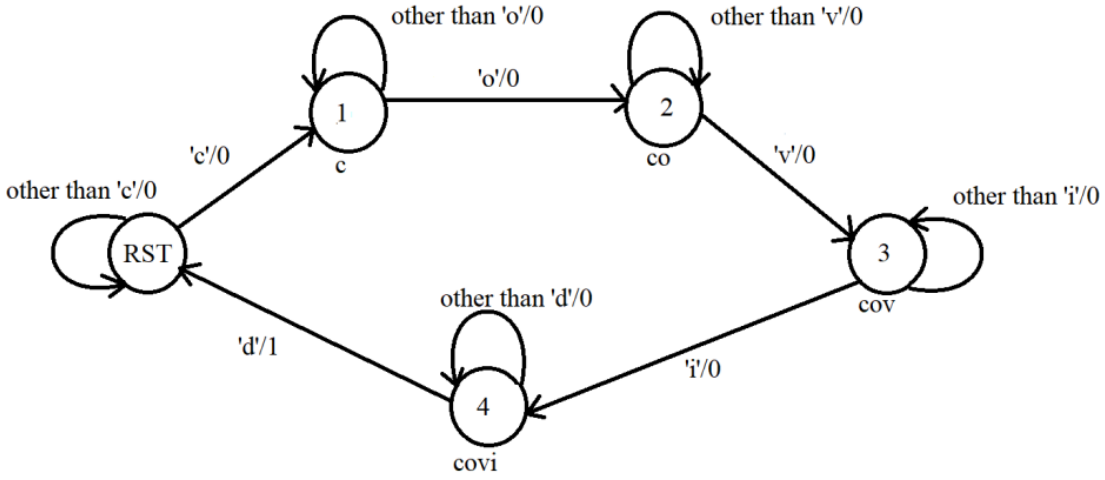


Figure 1: State diagram for detecting word "*covid*" in the given string of alphabets.

State Transition Table

Reset	Input	P.S.	N.S.	Output
1	X	XXX	RST	0
0	'c'	RST	1	0
0	'o'	1	2	0
0	'v'	2	3	0
0	'i'	3	4	0
0	'd'	4	RST	1

The architecture of main logic:

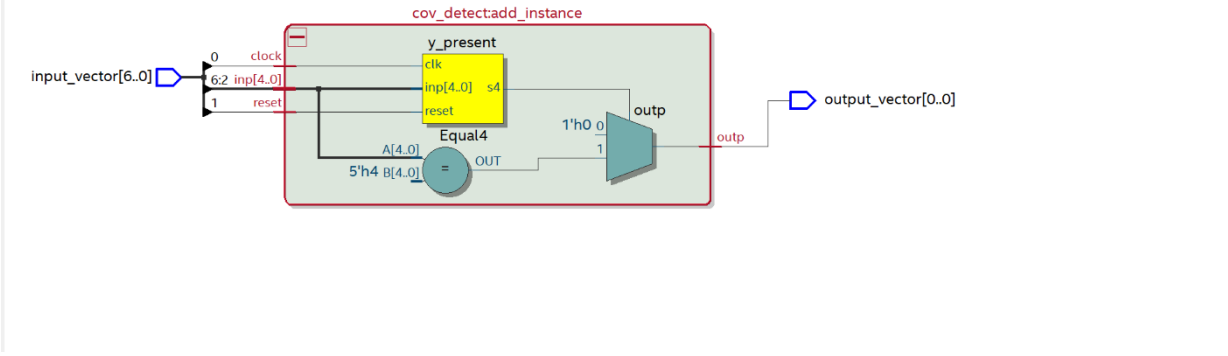
```
architecture rch of cov_detect is

type state is (rst,s1,s2,s3,s4);
gnal 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)=3) then    --c
                y_next<=s1;
                else
                     y_next<=rst;
                end if;

        when s1=>
            if(unsigned(inp)=15) then    --o
                y_next<=s2
                else
                     y_next<=s1;
                end if;

        when s2=>
            if(unsigned(inp)=22) then    --v
                y_next<=s3;
                else
                     y_next<=s2;
                end if;

        when s3=>
            if(unsigned(inp)=9) then    --i
                y_next<=s4;
                else
                     y_next<=s3;
                end if;

        when s4=>
            if(unsigned(inp)=4) then    --d
```

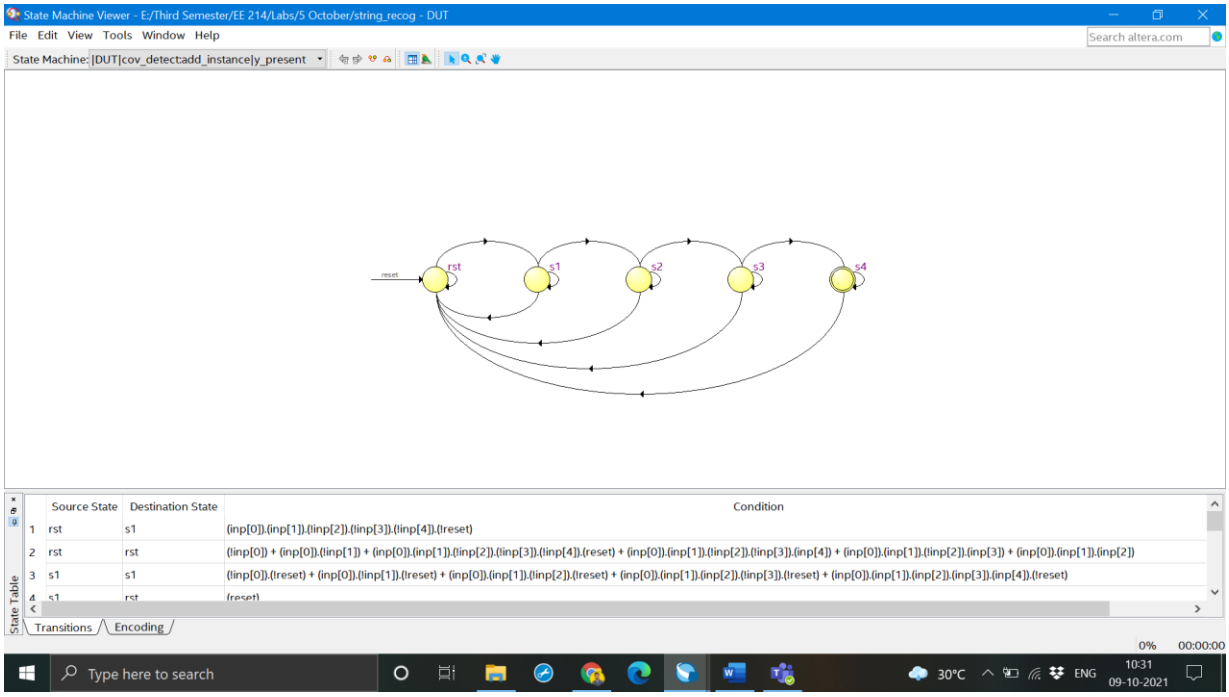
```
                y_next<=rst;
                else
                     y_next<=s4;
                end if;

        end case;
end process;

output_proc:process(inp,y_present)
begin
    case y_present is
        when rst=>
            if(unsigned(inp)=3) then    --c
                outp<='0';
                else
                    outp<='0';
                end if;
        when s1=>
            if(unsigned(inp)=15) then    --o
                outp<='0';
                else
                    outp<='0';
                end if;
        when s2=>
            if(unsigned(inp)=22) then    --v
                outp<='0';
                else
                    outp<='0';
                end if;
        when s3=>
            if(unsigned(inp)=9) then    --i
                outp<='0';
                else
                    outp<='0';
                end if;
        when s4=>
            if(unsigned(inp)=4) then    --d
                outp<='1';
                else
                    outp<='0';
                end if;
        end case;
end process;
```

```
end rch;
```

RTL View:



State Machine Viewer:



DUT Input/Output Format:

Input: 5-bit input reset clock **LSB** = clock **MSB** = inp(4)
Output: outp **LSB** = **MSB** = outp

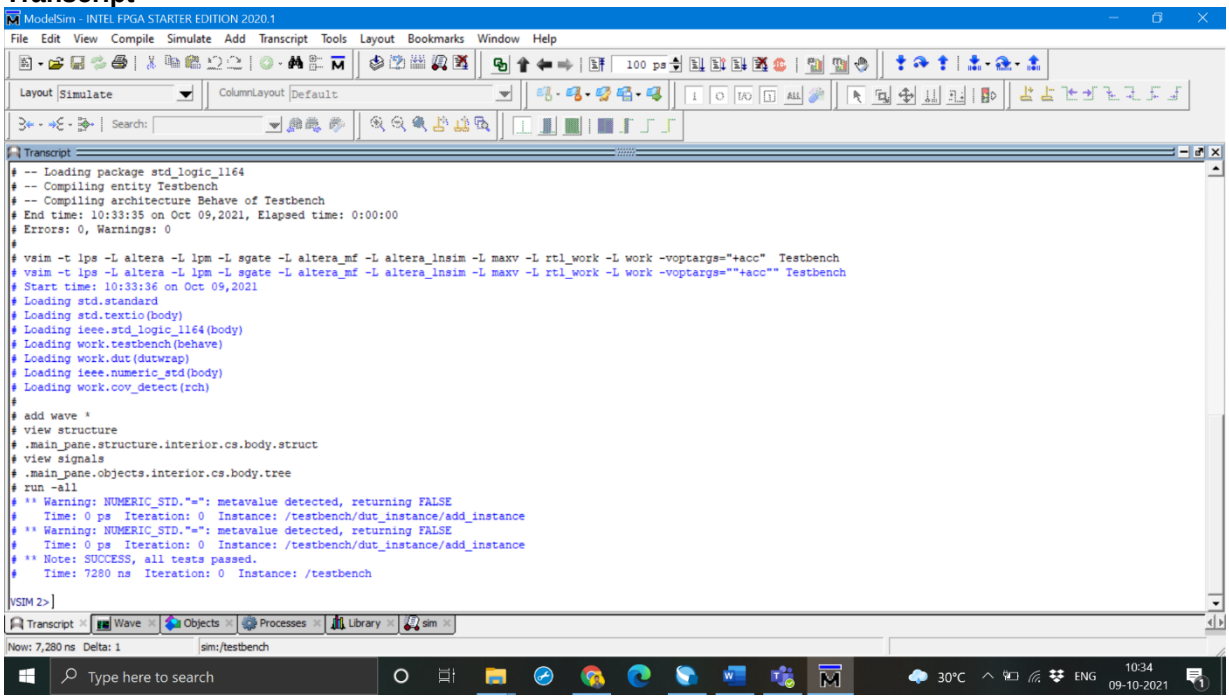
Some Test Cases from TRACEFILE.txt

Format: 5-bit input reset clock outp mask-bit

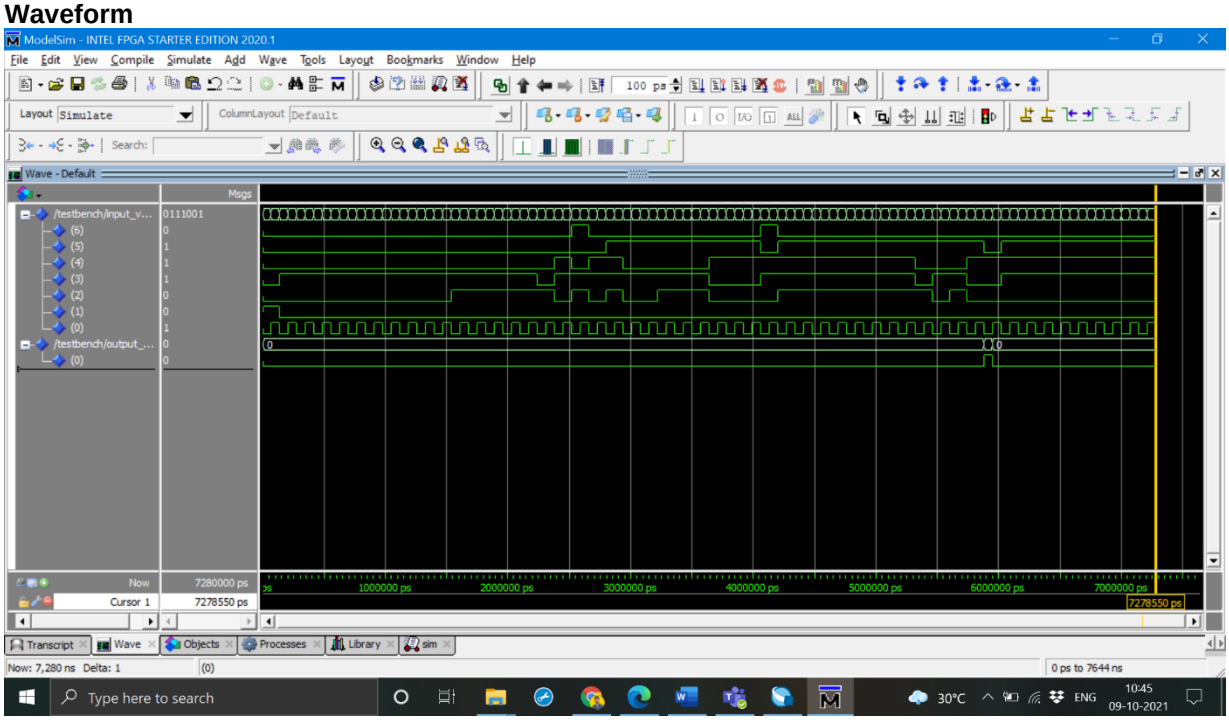
0011000 0 1
0011001 0 1
1001100 0 1
1001101 0 1
0011000 0 1
0011001 0 1
0111100 0 1
0111101 0 1
0101000 0 1
0101001 0 1
0101000 0 1
0101001 0 1
0101100 0 1
0101101 0 1
0101100 0 1
0101101 0 1
0101100 0 1
0101101 0 1
0110000 0 1

RTL Simulation:

Transcript

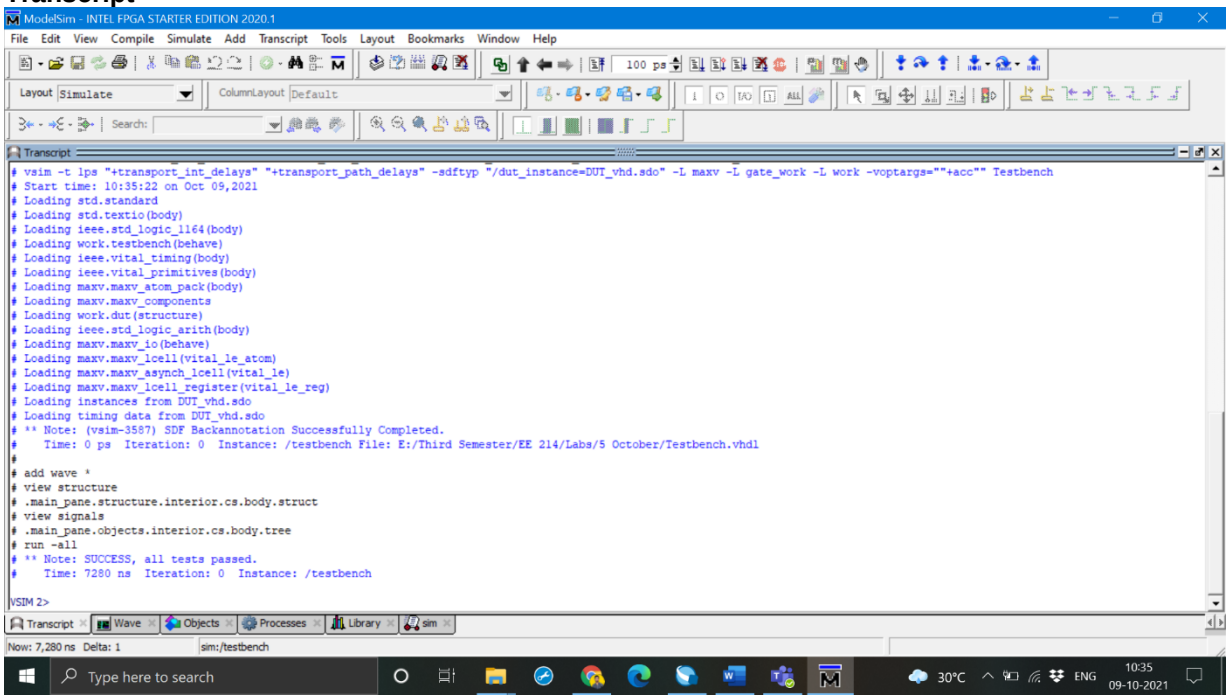


Waveform

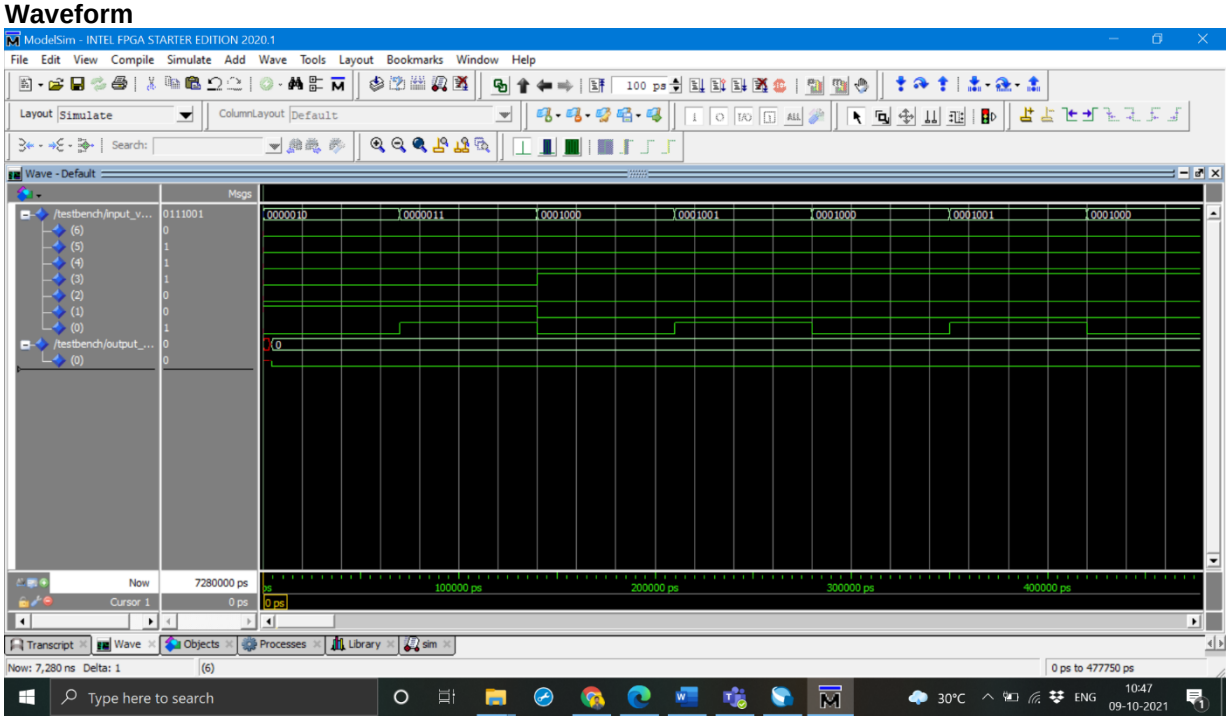


Gate-level Simulation:

Transcript

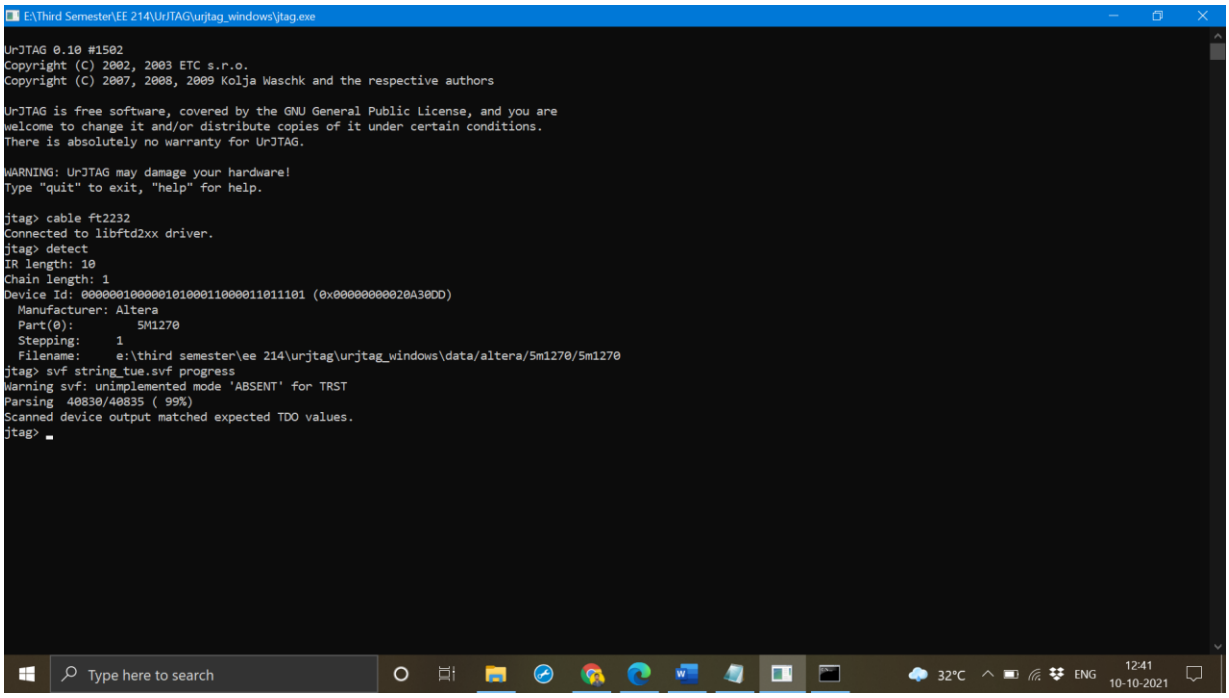


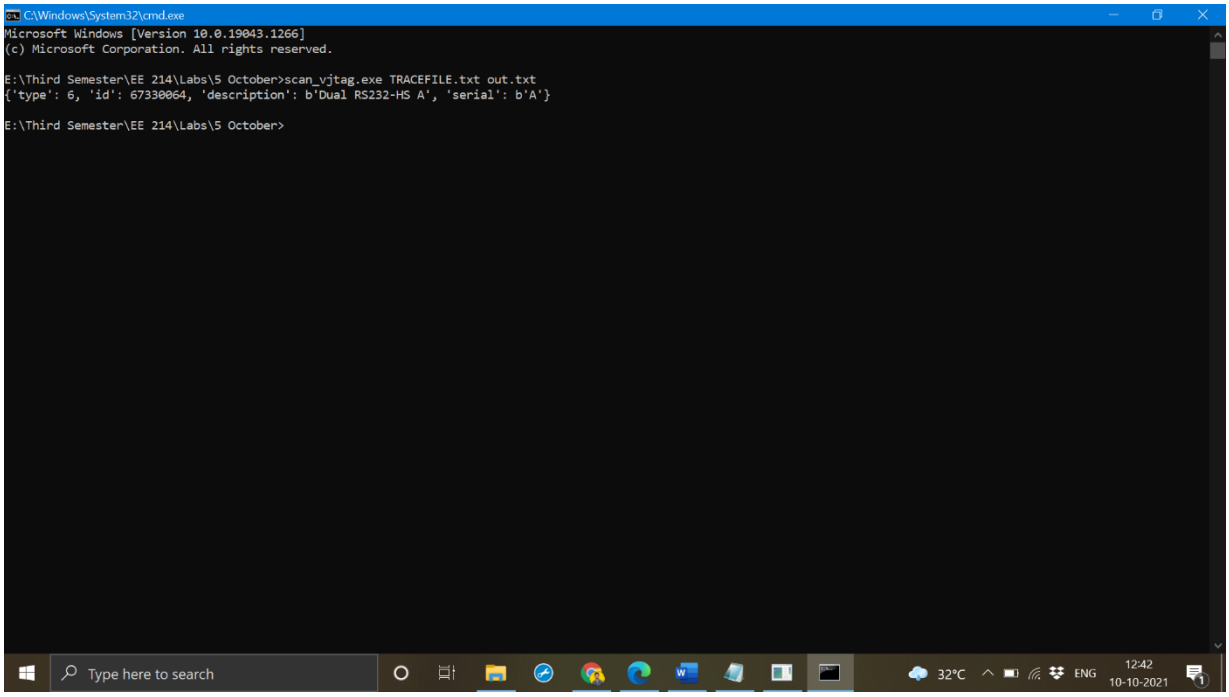
Waveform



Krypton board:

We have used a scan chain for this experiment. So **out.txt** has an output which I got using scan chain.





Some outputs from out.txt

0001001 0 Success
0001000 0 Success
0001001 0 Success
0001100 0 Success
0001101 0 Success
0001100 0 Success
0001101 0 Success
0001100 0 Success
0001101 0 Success
0001100 0 Success
0001101 0 Success
0001100 0 Success
0001101 0 Success
0000100 0 Success
0000101 0 Success
0011000 0 Success
0011001 0 Success

Observations:

The main observation and learning outcome from this experiment was how to write the logic of a Mealy-type FSM. And also how to write different concurrent processes and how to assign them their task.

References:

My primary reference was our course webpage; it contained many valuable things, such as a sample code and many other specifications. It also had one handout for this experiment which was very helpful. This I used as my reference