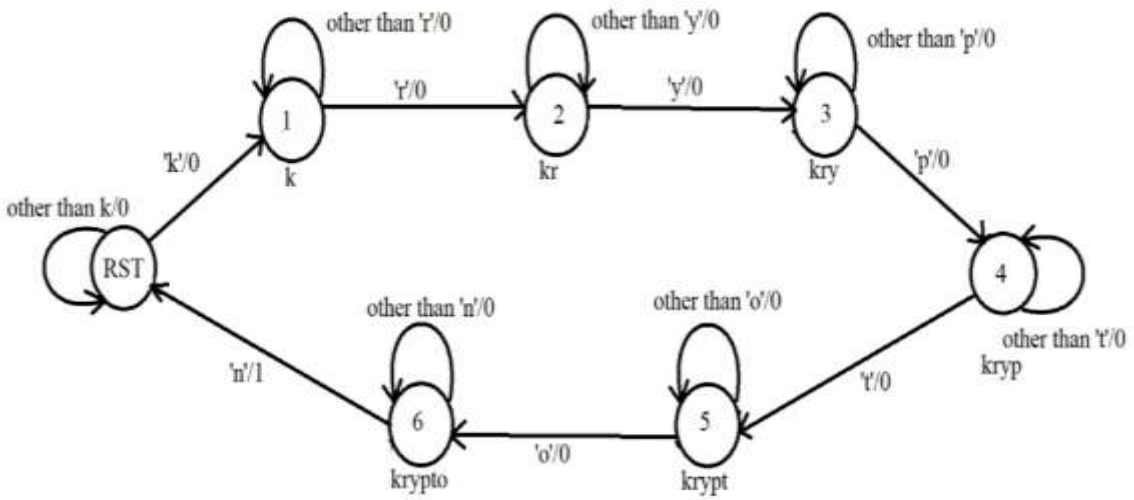
Experiment 6: String Recognizer

Narne Avinash Chowdary Roll Number 200070047
EE-214, WEL, IIT Bombay
October 10, 2021

Overview of the experiment:

- design a string detector using a Mealy type FSM which will detect the occurrence of krypton 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.
- Described behavioral model of the string detector Mealy type FSM in VHDL.
- Considered 5 different state variables

Approach to the experiment:



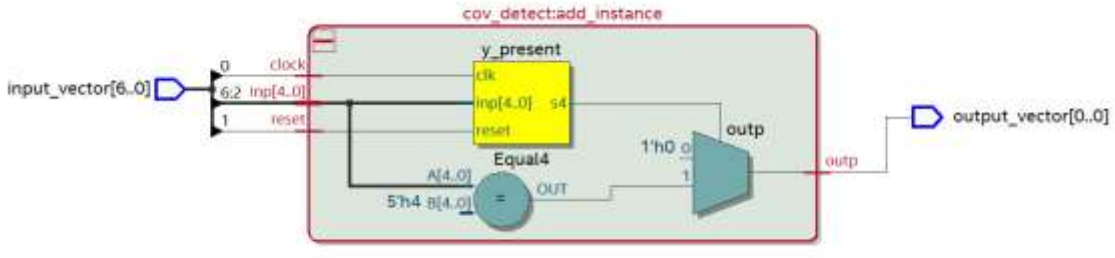
Design document and VHDL code if relevant:

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

entity cov_detect is
port(inp:in std_logic_vector(4 downto 0);
      reset,clock:in std_logic;
```

```
        outp: out std_logic);
end cov_detect;

architecture rch of cov_detect is

---------------Define state type here----------------------------
type state is (init,s1,s2,s3,s4); -- Fill the code
---------------Define signals of state type-----------------------
signal y_present,y_next: state:=init;

begin
clock_proc:process(clock,reset)
begin
   if(clock='1' and clock' event) then
      if(reset='1') then
         y_present<=init; -- Fill the code
      else
                  y_present<=y_next;
       -- Fill the code
      end if;
   end if;

end process;

state_transition_proc:process(inp,y_present)
begin
   case y_present is
      when init=>
         if(unsigned(inp)=3) then    --c
            y_next<=s1;
                                          outp<='0';
                                 else
                                          y_next<=init;
                                          outp<='0';
                                 end if;-- Fill the code
                                 when s1=>
                                 if(unsigned(inp)=15) then    --c
            y_next<=s2;
                                          outp<='0';
                                 else
                                          y_next<=s1;
                                          outp<='0';
                                 end if;
```

```
                        when s2=>
                        if(unsigned(inp)=22) then    --c
y_next<=s3;
                                outp<='0';
                        else
                                y_next<=s2;
                                outp<='0';
                        end if;
                        when s3=>
                        if(unsigned(inp)=9) then    --c
y_next<=s4;
                                outp<='0';
                        else
                                y_next<=s3;
                                outp<='0';
                        end if;
                        when s4=>
                        if(unsigned(inp)=4) then    --c
y_next<=init;
                                outp<='1';
                        else
                               y_next<=s4;
                                outp<='0';
                        end if;
                                end case;
                                end process;
                                end rch;
```

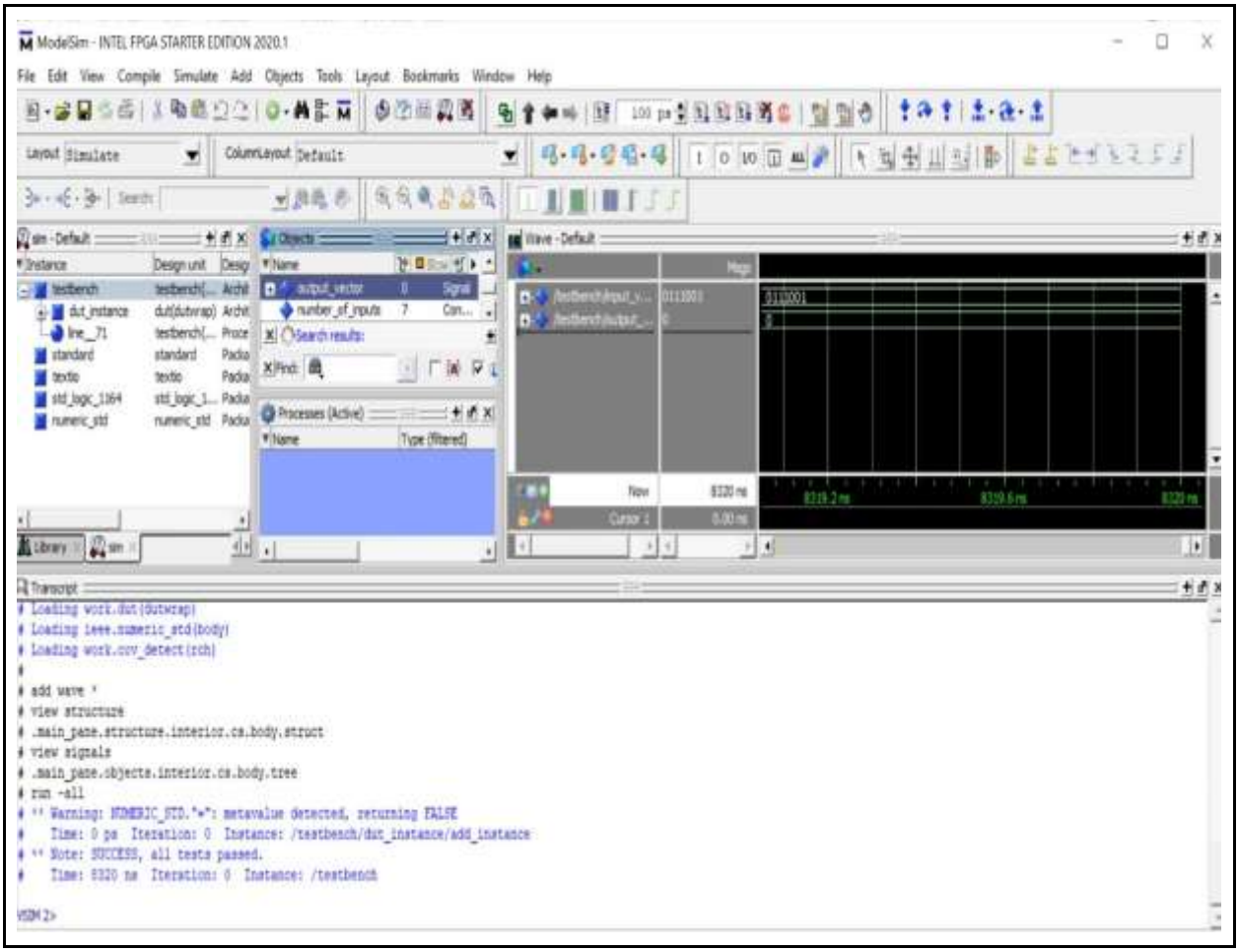
RTL View:



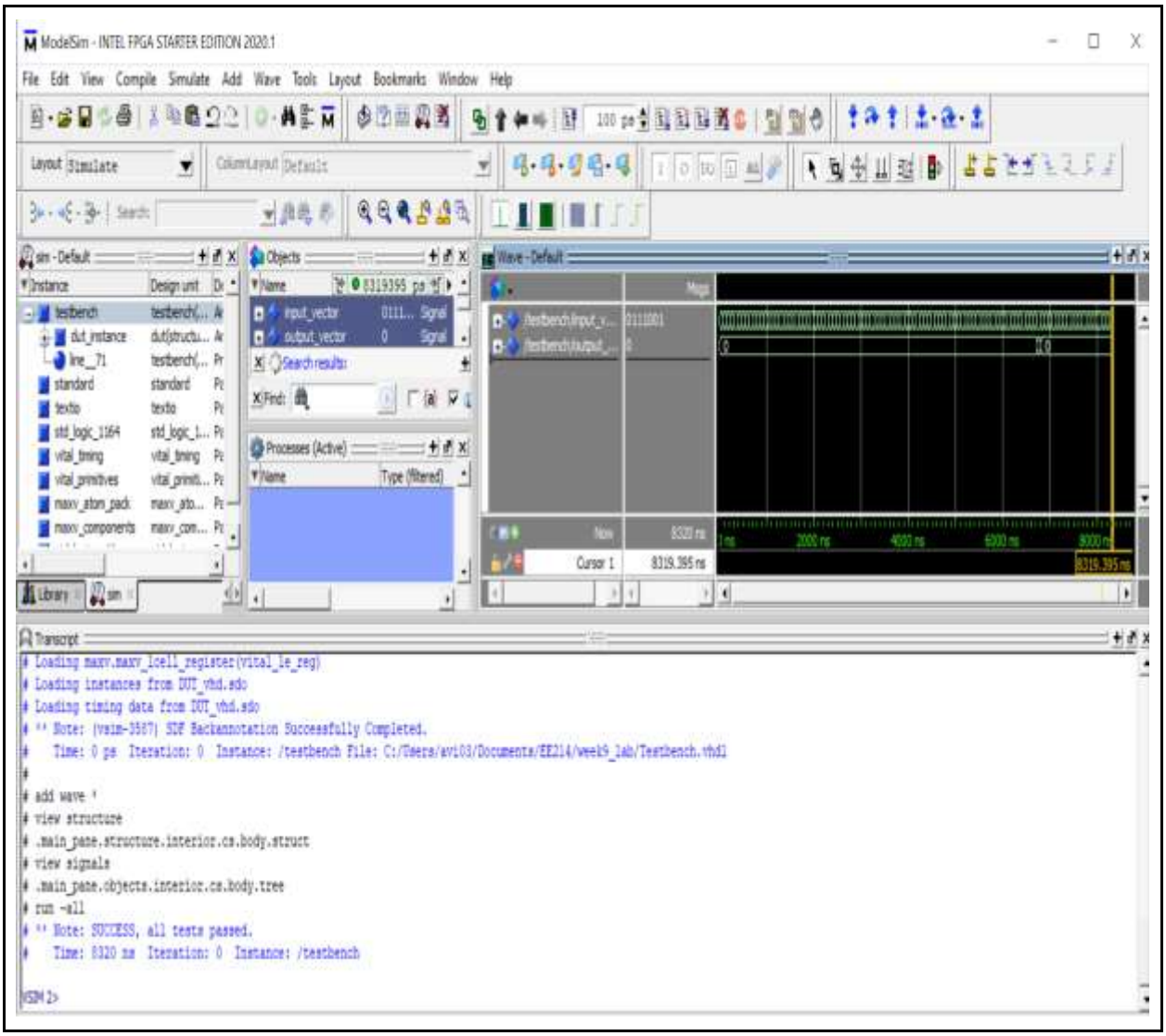
DUT Input/Output Format:

```
inp(4)=>input_vector(6),
  inp(3)=>input_vector(5),
   inp(2)=>input_vector(4),
   inp(1)=>input_vector(3),
  inp(0)=>input_vector(2),
  reset=>input_vector(1),
  clock=>input_vector(0),

  outp=>output_vector(0));
```

RTL Simulation:



Gate-level Simulation:



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

0000010 0 Success
0000011 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
0001001 0 Success
0001000 0 Success
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
1001100 0 Success
1001101 0 Success

0011000 0 Success
0011001 0 Success
0111100 0 Success
0111101 0 Success
0101000 0 Success
0101001 0 Success
0101000 0 Success
0101001 0 Success
0101100 0 Success
0101101 0 Success
0101100 0 Success
0101101 0 Success
0101100 0 Success
0101101 0 Success
0110000 0 Success
0110001 0 Success
0110000 0 Success
0110001 0 Success
0110000 0 Success
0110001 0 Success
1011000 0 Success
1011001 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0111100 0 Success
0111101 0 Success
0100100 0 Success
0100101 0 Success
0101000 0 Success
0101001 0 Success
0101100 0 Success

0101101 0 Success
0110000 0 Success
0110001 0 Success
0010000 1 Success
0010001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success
0111000 0 Success
0111001 0 Success

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