LAB 8: STRING DETECTOR

ARYA AGARWAL
Roll Number - 210070012

7th October 2022

1. Overview of the Experiment / Assignment:

- This experiment was related to Mealy type Machine. It a Finite State Machine(FSM) which depends on both current states and inputs.
- Using this Mealy type machine we have to detect a word from the given input string.
- Then using Quartus and TRACEFILE provided we have to write our VHDL code for our design. In this design we have to only use Behavioural dataflow.
- After writing the VHDL code , we have to check our design using SCANCHAIN.

2. Experiment Setup or Approach to the Assignment:

- In this experiment we have to take 7 inputs including clock and reset. We will correspondingly will get 1 output.If the reset bit is high the sequence will again start from its first state. We have to detect the word ‘run’ . We will require at least 3 states to detect the word run. Then we will draw the state diagram and then using that we can design a logic to detect the word ‘run’. For this experiment, letter ‘a’ is encoded as “00001”, ‘b’ is encoded as “00010” and so on. So, as soon as we detect ‘r’ i.e. if the input bits correspond to 18 which is the unsigned decimal of “10010” we move on to the next state to detect the letter ‘u’ else we will remain in the same state. The process goes on until we detect the word ‘run’. In this experiment we had given 3 words to detect ‘run’,’cry’ and ‘broom’. So, 3 FSM will be used and then we will take ‘OR’ of them for the final output.

- We were given an incomplete code which contained some hints and basics of our VHDL code. We have to complete that code and then check our program using SCANCHAIN.

For example suppose the input string is: ***Bring UNO cards from my bag***
This string contains the subsequence "run" "cry" and "broom".
Thus, the input and output sequences when lined up will look like:

Bring UNO cards from my bag
00000001000000000010010000

2.1 Design Code and Documentation:

Now once we understood all the functions and their logic, it's time to describe the circuit using code in VHDL language in Quartus. The DUT, gates files will almost be same except some minor changes in both cases.

The code for the main file is written below which is used to describe the circuit.

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

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

architecture bhv of word_detection is
--------------Define state type here-----------------------------
   type state is (rst,s1,s2,s3,s4); -- Fill other states here
---------------Define signals of state type-----------------------
   signal y_present1,y_next1,y_present2,y_present3,y_next2,y_next3: state:=rst;
   signal otp_1,otp_2,otp_3: std_logic := '0';
   begin
      clock_proc:process(clock,reset)
      begin
         if(clock='1' and clock' event) then
            if(reset = '1') then
               y_present1<=rst;
               y_present2<=rst;
               y_present3<=rst;
--Here Reset is synchronous
-- Fill the code here
            else
               y_present1<= y_next1;
               y_present2<= y_next2;
               y_present3<= y_next3;
 -- Fill the code here
            end if;
         end if;
      end process;
      state_transition_run:process(inp,y_present1,otp_1)
      begin
```

```
---------------------------FOR RUN-------------------------------------------
   case y_present1 is
   when rst=>
      if(unsigned(inp)=18) then --r has been detected
         y_next1<= s1;
         otp_1 <= '0';
      else
         y_next1<= rst;
         otp_1 <= '0';
      end if;

   when s1=>

       if(unsigned(inp)=21) then --u has been detected
         y_next1<= s2;
         otp_1 <= '0';
      else
         y_next1<= s1;
         otp_1 <= '0';
      end if;

   when s2=>

       if(unsigned(inp)=14) then --n has been detected
         y_next1<= rst;
         otp_1 <= '1';
      else
         y_next1<= s2;
         otp_1 <= '0';
      end if;

   when s3=>
     y_next1 <= rst;

            when s4=>
     y_next1 <= rst;
```

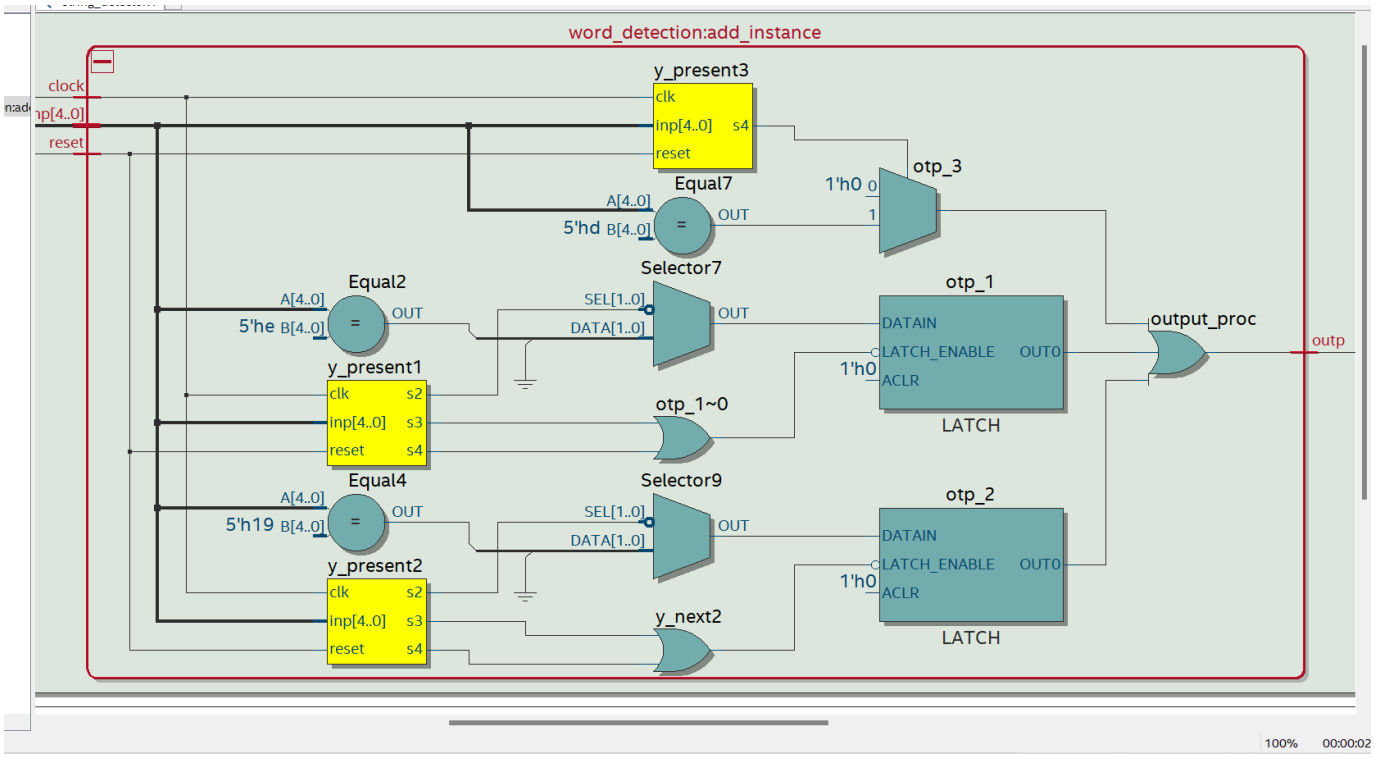
```
        end case;
      end process;

   ------------------------------------FOR CRY---------------------------   -
state_transition_cry:process(inp,y_present2,otp_2)
 begin

        case y_present2 is

        when rst=>

           if(unsigned(inp)=3) then --c has been detected
              y_next2<= s1;
              otp_2 <= '0';
           else
              y_next2<= rst;
              otp_2 <= '0';
           end if;

        when s1=>

            if(unsigned(inp)=18) then --r has been detected
              y_next2<= s2;
              otp_2 <= '0';
           else
              y_next2<= s1;
              otp_2 <= '0';
           end if;

        when s2=>

            if(unsigned(inp)=25) then --y has been detected
              y_next2<= rst;
              otp_2 <= '1';
           else
              y_next2<= s2;
              otp_2 <= '0';
```

```
              y_next2<= s2;
              otp_2 <= '0';
            end if;

        when s3=>
          y_next2 <= rst;

        when s4=>
          y_next2 <= rst;


        end case;
      end process;

   ---------------------------FOR BROOM---------------------------------------
state_transition_broom:process(inp,y_present3,otp_3)
begin

        case y_present3 is

        when rst=>

           if(unsigned(inp)=2) then --b has been detected
              y_next3<= s1;
              otp_3 <= '0';
           else
              y_next3<= rst;
              otp_3 <= '0';
           end if;
        when s1=>

            if(unsigned(inp)=18) then --r has been detected
              y_next3<= s2;
              otp_3 <= '0';
           else
              y_next3<= s1;
              otp_3 <= '0';
              end if;
```

```
            end if;
        when s2=>

            if(unsigned(inp)=15) then --o has been detected
              y_next3<= s3;
              otp_3 <= '0';
           else
              y_next3<= s2;
              otp_3 <= '0';
              end if;

        when s3=>

            if(unsigned(inp)=15) then --o has been detected
              y_next3<= s4;
              otp_3 <= '0';
           else
              y_next3<= s3;
              otp_3 <= '0';
              end if;
        when s4=>

            if(unsigned(inp)=13) then --m has been detected
              y_next3<= rst;
              otp_3 <= '1';
           else
              y_next3<= s4;
              otp_3 <= '0';
            end if;
        end case;
   end process;
-------------------------------------------------------------------------------
---------Fill rest of the code here---------

output_proc:process(y_present1,y_present2,y_present3, inp,otp_1,otp_2,otp_3)
begin
```

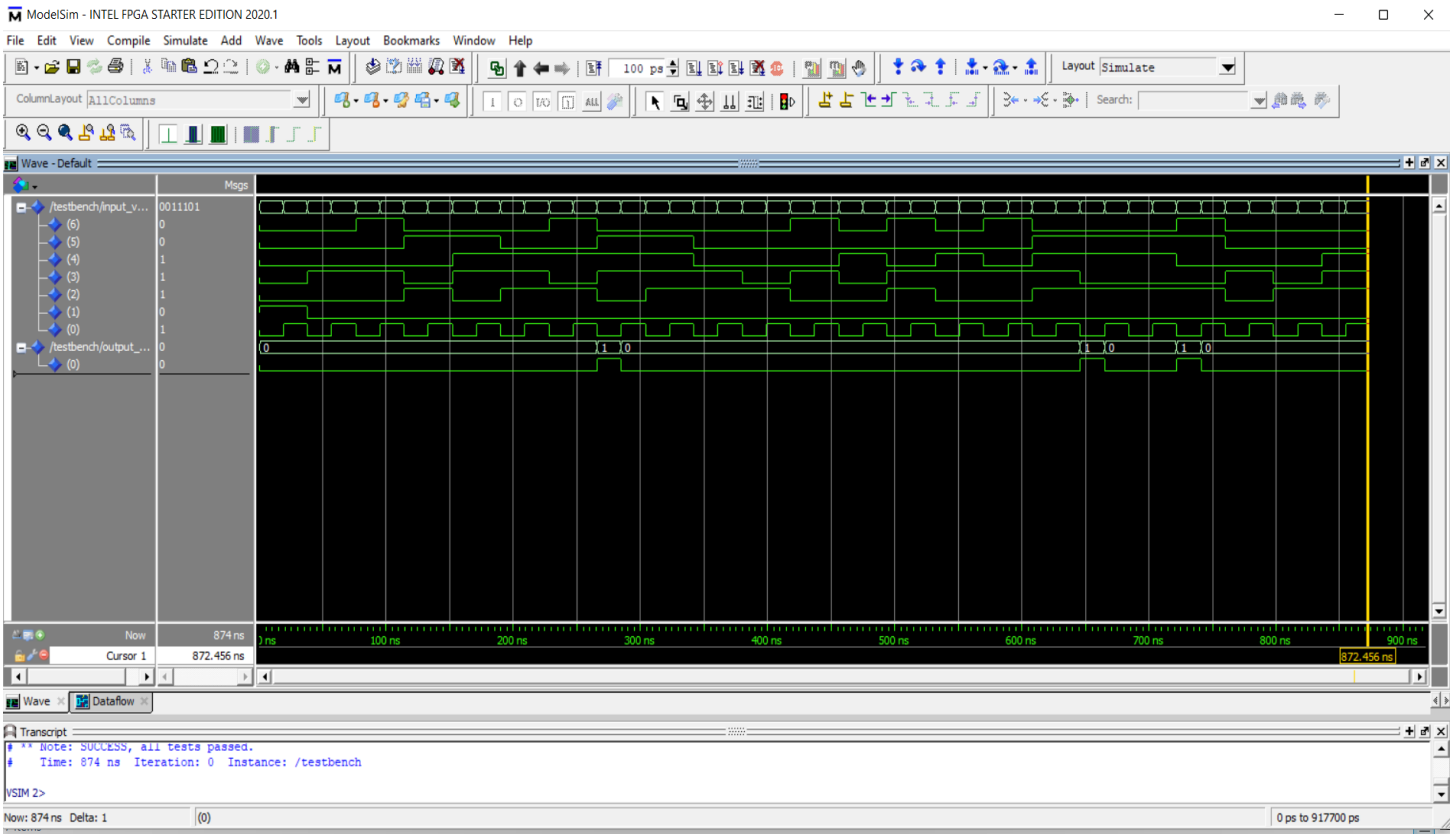
```
   end process;
-------------------------------------------------------------------------------
---------Fill rest of the code here---------

output_proc:process(y_present1,y_present2,y_present3, inp,otp_1,otp_2,otp_3)
begin

if (otp_1='1' or otp_2='1' or otp_3='1') then
   outp <= '1';
else
   outp <= '0';
 end if;
   end process;
   end bhv;
```

2.2 RTL VIEW

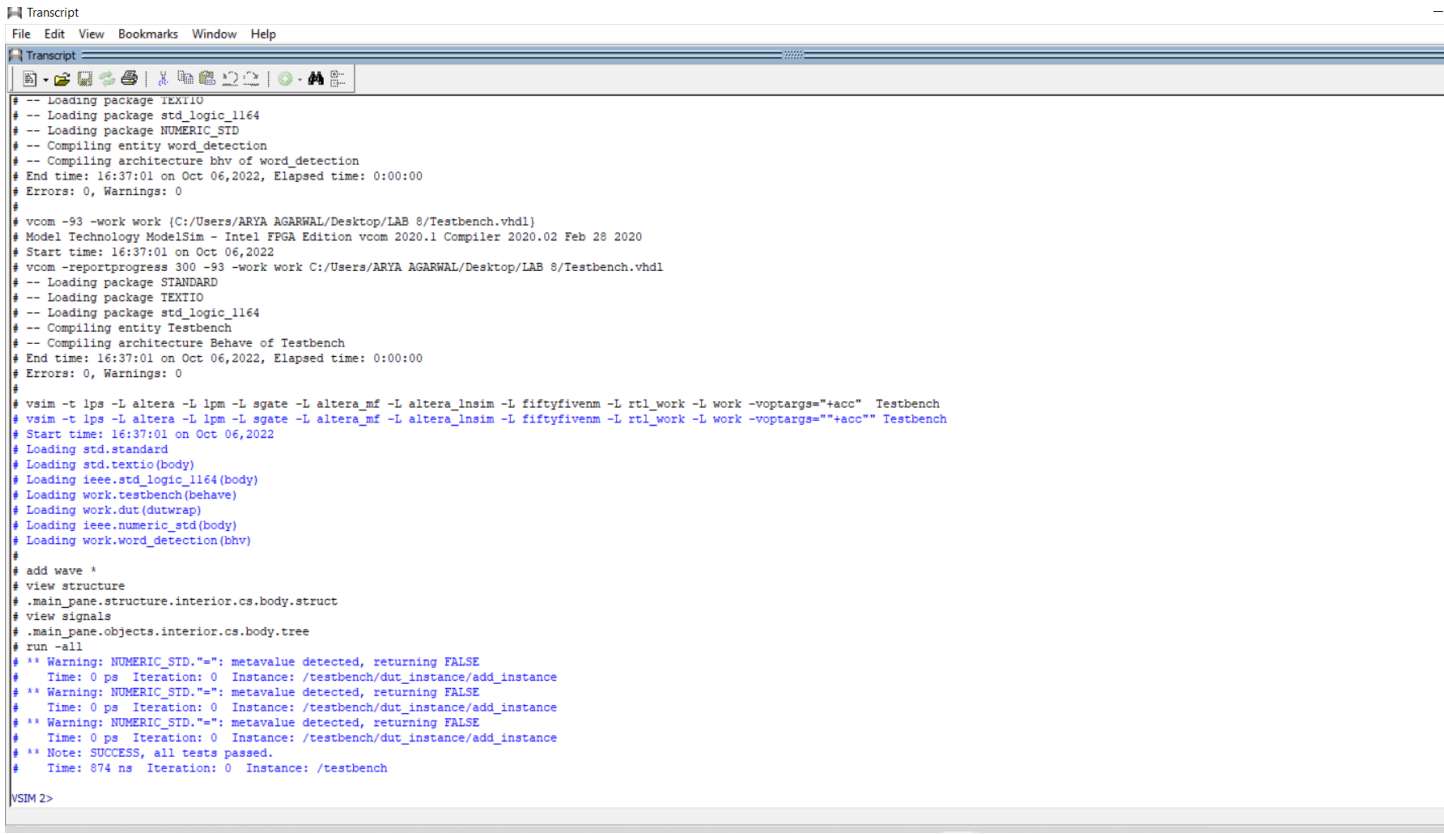


3. Observations:

After you run the analysis of code with no error. The next step is to run the simulation. For this we again use the MODELSIM - Altera Software. We also need the Testbench and the TRACEFILE (already provided) to run the simulation. The simulation is shown below.



3.1 TRANSCRIPT(all test cases passed successfully)



4. SCANCHAIN

After running the code successfully, we have to connect our design to Xenon. Then we were provided with the SCANCHAIN files. After adding the Top-level file from the SCANCHAIN files to our main code file. Then making some minor changes in the top level file and then set it as top level entity. Then Compile the whole design. Once the design is compiled make the .svf file and throw it in our URJTAG after connecting the Xen10 board to our PC. Before Using the Xenon Board, it should be checked thoroughly. We have to perform all the tests before we use it. Once the Testing is done we can throw our svf file in our Xen10 board using URJTAG. Then using SCANCHAIN we can check our logic.

The benefit of using SCANCHAIN over Xen10 is that if we want to check a 9 output bit device we can't check it using Xen10 board because it only contains 8 LEDs. Another benefit is that SCANCHAIN is much faster than Xen10.

After doing all the necessary steps SCANCHAIN will create an output file which will show how many cases passed.
Here is my output file. My design did not contain any test case failure.