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.    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:**      **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 |