CE 3320.001 Project

Name: Fernando Portillo

Project Title: Mealy State Machine

Date: November 29, 2022

Table of Contents

1 Problem

Design a mealy state machine (1 input, 1 output) that is turned on when the input ”1010” is obtained.

2 Mealy State Machine Designed

2.1 Description

Mealy state machine can take detect the sequence ”1010” and turn on. It can also use previous bits to create a new sequence (overlaps). Resets if the reset is ON (0) mostly seen in the code. Using four states to determine the sequence.

2.2 Mealy State Machine Diagram

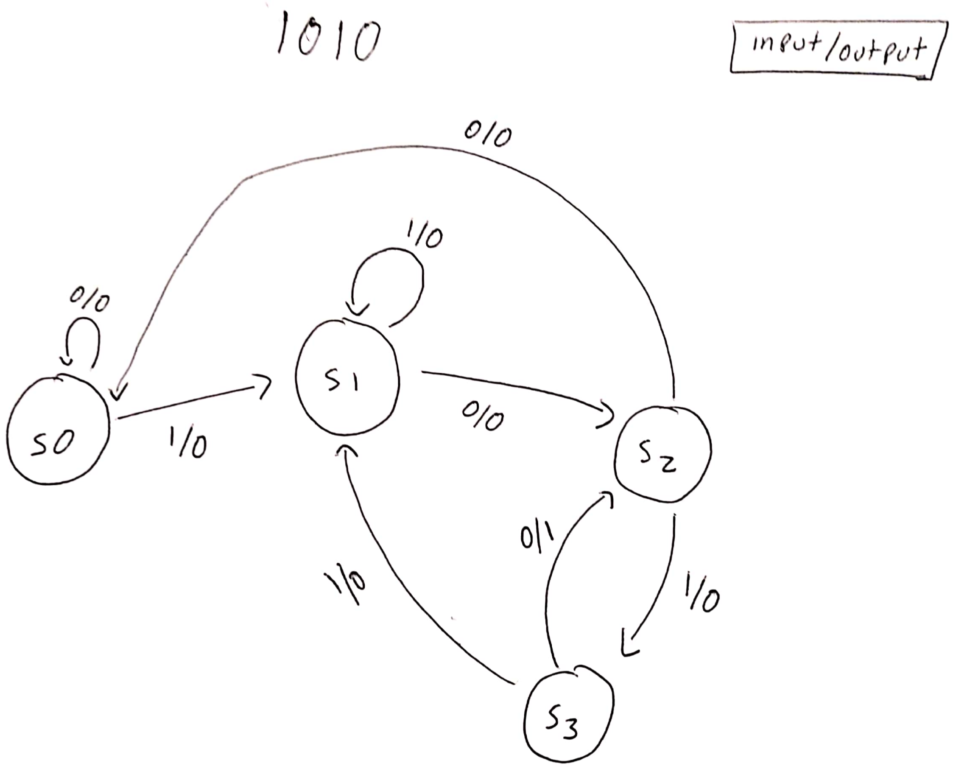


Figure 1: Mealy State Machine Diagram

3 Verilog Solutions

3.1 source.v

```verilog
`define CK2Q 5 // Defines the Clock - to - Q Delay of the flip flop.
module source(reset , clk , in_seq , out_seq);
input reset;
input clk;
input in_seq;
output out_seq;

reg out_seq;
reg in_seq_reg;

//---------------- Defining State machine states ----------------
parameter SIZE = 2;
parameter S0 = 2'b00 , S1 = 2'b01 , S2 = 2'b10 , S3 = 2'b11;

//---------------- Internal Variables ----------------
reg [SIZE - 1 : 0] state;
reg [SIZE - 1 : 0] next_state;

//---------------- Register the in_seq ----------------
always @(posedge clk)
  begin
    if (reset == 1'b0)
    begin
      in_seq_reg <= #`CK2Q 1'b0;
    end
    else
    begin
      in_seq_reg <= #`CK2Q in_seq;
    end
  end

```

Figure 2: source.v verilog code (Part 1)

```
//--------------- Mealy State machined Code Starts Here ----------------------------
//Determine the next state for each state in the state machine using the input
//sequence given to it. 'next_state' is combinatorial in nature. 1010
//--------------------------------------------------------------------------------
always @(state or in_seq_reg)
 begin
   next_state = 2'b00;
   case(state)
     S0 :
       if (in_seq_reg == 1'b1)
       begin
         next_state = S1;
       end
       else
       begin
         next_state = S0;
       end
     S1 :
       if (in_seq_reg == 1'b0)
       begin
         next_state = S2;
       end
       else
       begin
         next_state = S1;
       end
     S2 :
       if (in_seq_reg == 1'b1)
       begin
         next_state = S3;
       end
       else
       begin
         next_state = S1;
       end
     S3 :
       if (in_seq_reg == 1'b0)
       begin
         next_state = S2;
       end
       else
       begin
         next_state = S1;
       end
   endcase
 end


```

Figure 3: source.v verilog code (Part 2)

```
//-----------------------------------------------------------------------------------
//register the combinatorial next_state variable.
//-----------------------------------------------------------------------------------
always @(posedge clk)
 begin
   if (reset == 1'b0)
   begin
     state <= #`CK2Q S0;
   end
   else
   begin
     state <= #`CK2Q next_state;
   end
 end

//-----------------------------------------------------------------------------------
//Based on the combinatorial next_state signal and the input sequence, determine the
//out_seq of the finite state machine.
//-----------------------------------------------------------------------------------
always @(state or in_seq_reg or reset)
 begin
   if (reset == 1'b0)
   begin
     out_seq <= 1'b0;
   end
   else
   begin
     case(state)
       S3 :
         begin
           if (in_seq_reg == 1'b0)
           begin
             out_seq <= 1'b1;
           end
           else
             out_seq <= 1'b0;
           end
       default :
         begin
           out_seq <= 1'b0;
         end
     endcase
   end
 end
endmodule // end of Module Mealy state machine
```

Figure 4: source.v verilog code (Part 3)

3.2 testbench.v

```
module testbench();
//registers for inputs
reg reset, clk, in_seq;

//bit stream register
reg [15:0] data;

//bit shift indicator and output wire
integer shift;
wire out_seq;

//create source module
source seq_detector(reset, clk, in_seq, out_seq);

//reset and initial data setup
initial
  begin
    shift = 0;
    data = 16'b0010100110101010;
    reset = 1'b0;
    #1200;
    reset = 1'b1;
    #60000;
    $display("\n");
    $finish;
  end

```

Figure 5: testbench.v verilog code (Part 1)

```
    //generate clock
    initial
      begin
        clk = 1'b0;
        forever begin
          #600;
          clk = ~clk;
        end
      end

    //handle sequence shift
    always @(posedge clk)
      begin
        in_seq = data >> shift;
        shift = shift + 1;

        #50 //wait 50 nanoseconds

        //print information
        $write(in_seq);
        if(out_seq == 1'b1)
          $display("\nSequence with or without overlap detected\n");
      end
    endmodule

```

Figure 6: testbench.v verilog code (Part 2)

4 Output

Output in Vivado differs from the output seen in the command line. Unfortunately, I could not get the Vivado Design Suite to work, it shows all the inputs changing but the output does not seem to change. Command line output if the sequence was detected based on the output wire turning on. That is the correct functionality of the design. The command line shows the sequence up the point it was detected or when the program ends.

4.1 Vivado

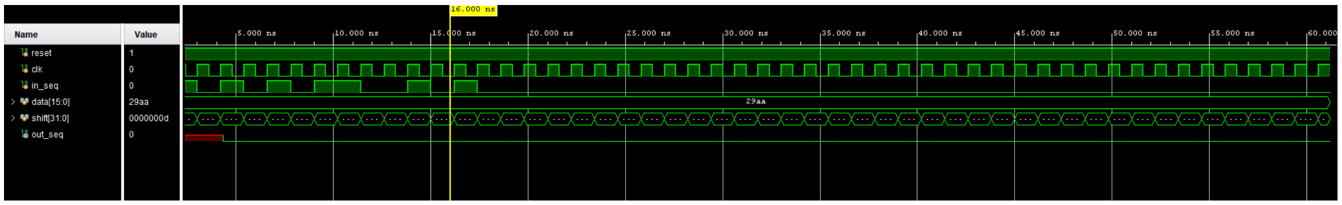


Figure 7: overall look of the vivado simulation output

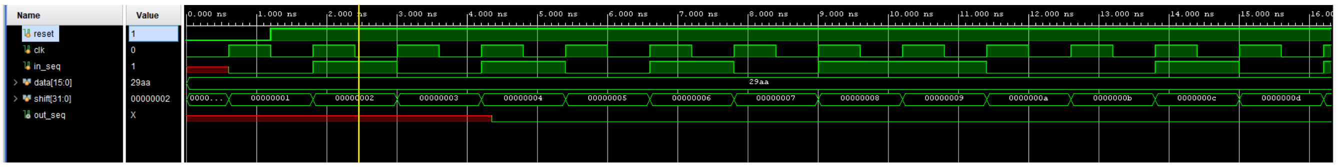
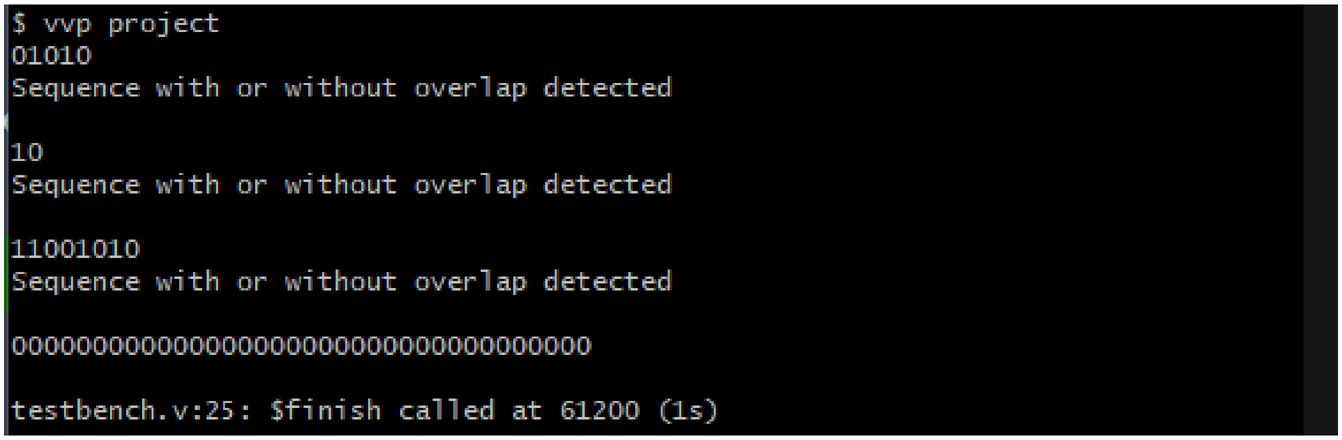


Figure 8: close up look of the vivado simulation output

4.2 Command Line (Icarus Verilog)



```
$ vvp project
01010
Sequence with or without overlap detected

10
Sequence with or without overlap detected

11001010
Sequence with or without overlap detected

000000000000000000000000000000000

testbench.v:25: $finish called at 61200 (1s)
```

Figure 9: icarus verilog command line output using display