LAB #5 Introduction to State Machines

LAB OBJECTIVES

1. Gain experience designing state machines via state tables and K-map techniques
2. Implement state machine design in PLDs using equations
3. Simulate a state machine
4. Design state machines using "case" statements
5. Experience the difference between Moore and Mealy state machines

PART 1 – Design a state machine solution for the state diagram in Figure 5-1. Use D flip-flops and only equations (assign statements) to implement this design. (No "if else" OR "case" statements can be used here, [just for RESET]). Use K-maps to find the equations for each of the inputs to the D flip-flops. Implement your equations from the K-maps into Verilog code, and bring your Verilog file to lab. Compile your file and assign pins, download and test your solution. Since this type of state machine has an "implied" system clock you will have to provide a clock for your testing. Monitor the outputs of your state machine.

You will have four types of outputs: the present state of the flip-flops, the Next State logic outputs for the D-F/Fs the Moore output and the Mealy output. These should all be assigned to LEDs.

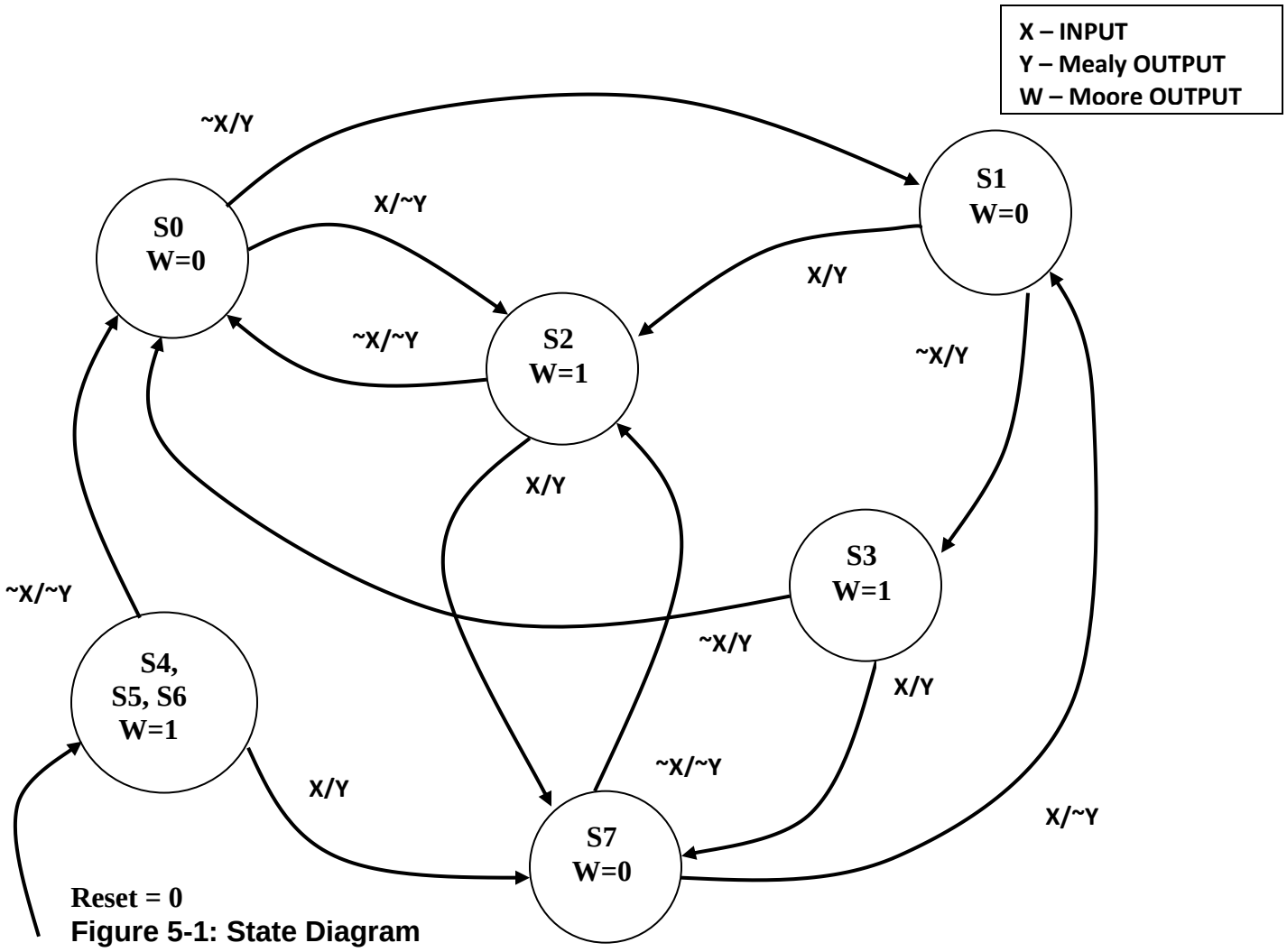


Figure 5-1: State Diagram

PART 2 – Now you can use the Verilog's "if else" or "case" statement to implement the state machine in Part 1. (see Verilog examples).

QUESTION #1: What is the advantage of writing the description using this method?

PART 3 - This is a useful state machine design with Mealy and Moore outputs. Design a solution to the following sequence detector. There is one input called **A**.
For example, the sequence **A, A, /A, A, /A,** would be represented by 1 1 0 1 0.

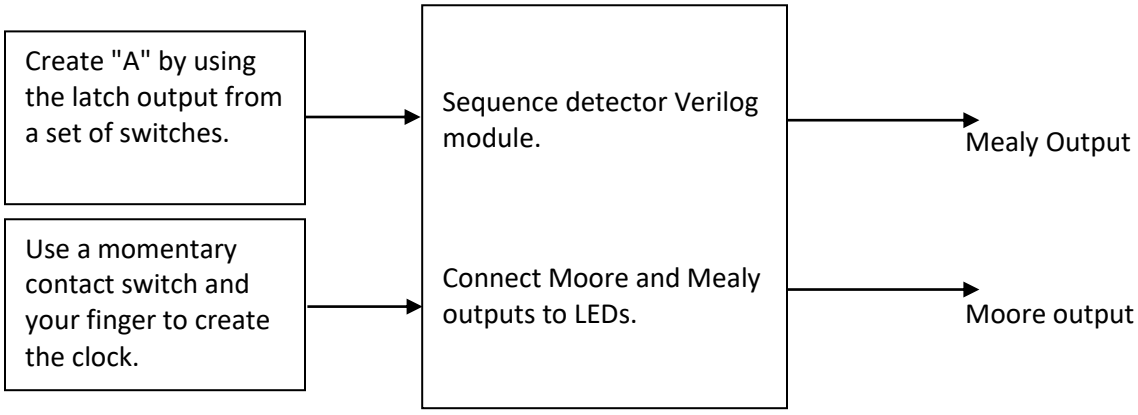
Find this sequence 0 1 1 0 1 1 0 1 0

The Moore output is active on the last "0" of the successful sequence.

The Mealy output goes active whenever the state machine is in the state just before the Moore output goes active and A= "1". The Mealy output remains active if A='1'; if A goes to '0' the Mealy output goes inactive (even without a clock transition). The Mealy output "can be" active where the "^" is located:
that is at the second to the last state and when A=1. 0 1 1 0 1 1 0 1 0
^

When the clock changes the flip-flops to another state, the Mealy out goes inactive. Remember the use of the word "ACTIVE". This could mean a **high** state or a **low** state. (Active High or Active Low)

Remember to draw a state diagram for your lab report. You should do this before you start your Verilog code.



Use 4 switches (similar to 4 keys labeled '0', '1', '2', '3') and NO clock input switch. Use the 32 MHz free running clock instead. Implement a design to detect the following sequence: 1, 2, 3, 0, 0, 3, 2, 1 Show an Active High Moore output when the correct sequence has been entered.

Lab 5 – Introduction to State Machines Demo List

Part Number	Requirements	Completed
Part 1 – Boolean Logic State Machine		
	K-Maps and Work	
	Verilog: Use “data flow” modeling	
	Waveform	
Part 2 – State Machine using Behavioral Modeling		
	Verilog: Use “behavioral” modeling	
	Waveform	
Part 3 – Special State Machine Design		
	Verilog	
	Waveform	

Note 1: “Verilog” is referring to testing the code on your FPGA (Hardware)

Note 2: “Waveform” is referring to creating a test bench then producing a waveform.

Useful EXAMPLES of Verilog HDL for Lab 5 and Lab 6

```
module stmachine(zmoore,x,reset,clk,zstate,sevenseg);    //example of a state machine coded in Verilog

input               x,reset,clk;
output              zmoore;
output[1:0]          zstate;              // optional, will indicate which "state" the state machine is presently in
output[6:0]          sevenseg;

reg [1:0]           current_state,next_state;     // internal  signals for the state machine ... but not outputs
reg                 zmoore;                       // zmoore is one of the desired outputs
reg[6:0]            sevenseg;

parameter                   s0          = 2'b00,
                    s1          = 2'b01,            s2          = 2'b10,
                    s3          = 2'b11;

always@(x or current_state)
 begin              // the control logic of the machine
  case(current_state)
    s0 : begin
           zmoore<=1;
           sevenseg<=7'b1111110;
           if(x==1)  next_state<=s1;
             else    next_state<=s3;
         end

    s1 : begin
           zmoore<=1;
           sevenseg<=7'b0110000;
           if(x==1)  next_state<=s2;
             else    next_state<=s0;
         end

    s2 : begin
           zmoore<=1;
           sevenseg<=7'b1101101;
           if(x==1)  next_state<=s3;
             else    next_state<=s1;
         end

    s3 : begin
           zmoore<=0;
           sevenseg<=7'b1111001;
           if(x==1)  next_state<=s0;
             else    next_state<=s2;
         end

  endcase
 end

always@(posedge clk or posedge reset)
 begin      // the register part of the machine
    if(reset)          current_state<=s0;
    else               current_state<=next_state;
 end

assign zstate = current_state;
endmodule
```