Memo 6

Drexel University

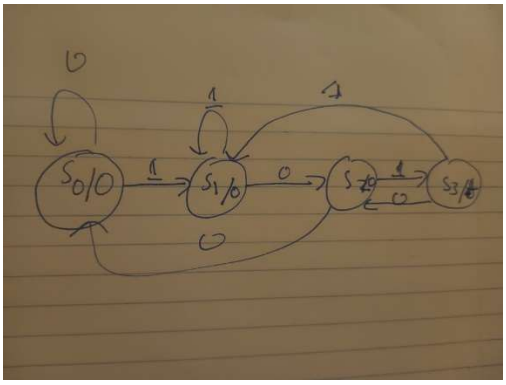
To: Dr Christopher Peters
From: Mahad Faisal
Date: 02/29/2024
Re: Lab 6- Finite State Machines

Purpose

The purpose of this lab is to demonstrate how a finite state machine can be used to detect a sequence ‘101’ with overlapping. For this project, the finite state machine will first be designed using a state transition drawing, table, circuit and a state table.

Methods

The first step in designing this finite state machine is to draw a state transition diagram. Since the sequence is 101 with an overlap using a Moore machine is suitable and the state transition diagram looks like this:



State 0 is when the input is 0 and out is zero as well. The next state is reached when the input is 1 and that state is defined as State 1. Since the next number in the sequence is 0, the input must be zero for the state to transition to State 2 or else it returns to state 2. At state 3 the input must be 0 to complete the sequence and return an output of 1. This helps us design the state transition table which is used in writing the Verilog code seen in Fig 1.1. Using the state transition table 3 equations were derived: F= AB, A* = x’B + xAB’ and B* = x. This was used to develop a set of inputs and outputs to determine hat the correct output of the code should be.

In the Verilog there is the clock signal (CLK), which serves as the timing reference for the entire design. It toggles at regular intervals determined by the specified clock period (CLK_PERIOD), ensuring synchronous operation and precise timing control within the FSM simulation environment.

Accompanying the clock signal is the binary input stream (x), which represents the input data provided to the FSM. By systematically manipulating x within the simulation, different input scenarios are emulated, allowing for the evaluation of the FSM's response under varying conditions.

The FSM's state is encapsulated by the state variable throughout the simulation, reflecting its current state at any given moment. Governed by a series of state transition logic, state dynamically transitions between different states based on input signals and clock edges, offering critical insights into the FSM's operational behavior.

Driving the FSM's operation is logic block, implemented within an always @(posedge CLK) construct. This FSM logic dictates the FSM's behavior, orchestrating state transitions based on specific conditions. By evaluating input conditions and responding accordingly, the FSM logic ensures the accurate detection of the "101" sequence and the fulfillment of its design objectives.

The application of stimulus within the simulation is a crucial aspect of the verification process. By systematically manipulating x at specified time intervals, the testbench trggers state transitions within the FSM, simulating real-world input scenarios and facilitating a comprehensive assessment of the FSM's performance.

Critical to the verification process are the display statements embedded within the simulation. These statements provide real-time monitoring of key signals (CLK, x, state), enabling observation of signal behavior throughout the simulation and aiding in debugging and verification efforts.

This implementation of a 101 overlapping detector was successful as the waveform matched the expected results.

The second part of the project was a hardware implementation using:

(1) Large breadboard (1) Arduino Mega 2560 (3) LEDs (3) 1 kΩ resistors (1) 74HC04 Hex Inverter package (1) 74HC08 AND gate package (1) 74HC32 OR gate package (1) 74HC595 8-bit shift register with 8-bit output register.

The process of building and verifying the "101" detector circuit involves several key steps, each contributing to the overall functionality and performance of the system.

In the initial step, a common ground is established on the breadboard, and three LEDs with current-limiting resistors are added to visually indicate the circuit's operation. This step ensures a stable electrical reference and provides a visual indication of the circuit's output.

Next, logic gates are integrated into the circuit, including shift registers, inverters, AND gates, and OR gates. These gates play crucial roles in processing input signals and generating output signals based on predefined logic conditions, forming the foundation of the finite state machine (FSM) operation.

Testing and validation of specific circuit components, such as input signals (x) and the complementary output (B*), are conducted to ensure proper functionality and alignment with design specifications. This step involves wiring connections and Arduino code implementation to accurately read and set input states and monitor output responses.

Further testing focuses on verifying the output states of the shift register, LEDs, and computed variables (A, B, and F). By establishing proper wiring connections and implementing Arduino code to interface with the shift register and monitor output states, the circuit's behavior can be systematically evaluated and validated.

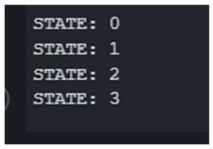
The final step involves constructing additional circuit components to compute the complementary output (A*) using NOT gates, AND gates, and an OR gate. This step enhances the circuit's functionality by providing a complementary output signal based on predefined logic conditions.

Throughout the process, meticulous attention to wiring connections, Arduino code implementation, and verification of output states is essential to ensure the circuit operates as intended. By following a systematic step-by-step approach, any discrepancies or errors in the circuit's operation can be identified and addressed effectively, leading to the successful creation of a Moore-style FSM.

The final circuit can be seen in Fig 1.2.

Results

The EPwaveform output of the Verilog code clearly shows that the correct sequence ‘101’ was detected as the output become 1 once the sequence is detected.



The results for the hardware lab seen in the video also match the expected outcomes detailed in the project PDF and this lab was therefore a success.

Figure 1.1

```
  // FSM logic
  always @(posedge CLK) begin
    // State transition logic
    if (state == 2'b00) begin // State S0
      if (x) state <= 2'b01; // Transition to S1 if input x is 1
    end
    else if (state == 2'b01) begin // State S1
      if (x) state <= 2'b10; // Transition to S2 if input x is 1
      else state <= 2'b00; // Reset to S0 if input x is 0
    end
    else if (state == 2'b10) begin // State S2
      if (x) state <= 2'b11; // Transition to S3 if input x is 1
      else state <= 2'b00; // Reset to S0 if input x is 0
    end
    else begin // State S3
      if (x) state <= 2'b01; // Transition to S1 if input x is 1 (overlap)
      else state <= 2'b00; // Reset to S0 if input x is 0
    end
  end

  // Display signals
  always @(posedge CLK) begin
    $display("Clock signal CLK: %b", CLK);
    $display("Data x: %b", x);
    $display("State: %b", state);
    // Additional signals A, B, A*, B* can be displayed here if available
  end

  // Stimulus
  initial begin
    $dumpfile("dump.vcd"); // Dump waveform to a file
    $dumpvars(0, testbench); // Dump all variables

    // Initialize inputs
    x = 0;

    // Apply stimulus
    #5; x = 1; // Sequence: 0 1
    #5; x = 0; // Sequence: 0 1 0
    #5; x = 1; // Sequence: 0 1 0 1
    #5; x = 1; // Sequence: 0 1 0 1 1 (sequence detected)

    // End simulation
    #10; $finish;
  end

endmodule

```

Figure 1.2