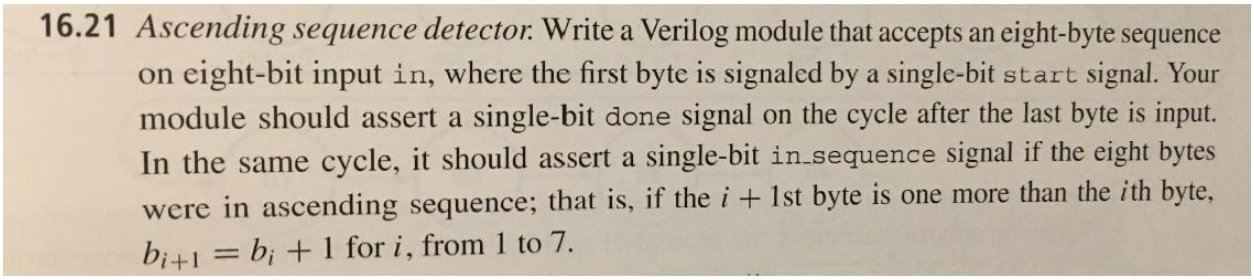
UNIVERSITY OF CALIFORNIA—DAVIS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
EEC180 — DIGITAL SYSTEMS II — WINTER 2023

HOMEWORK 3

Tobin Joseph

1) Problem 16.21 from the textbook. (30 points)



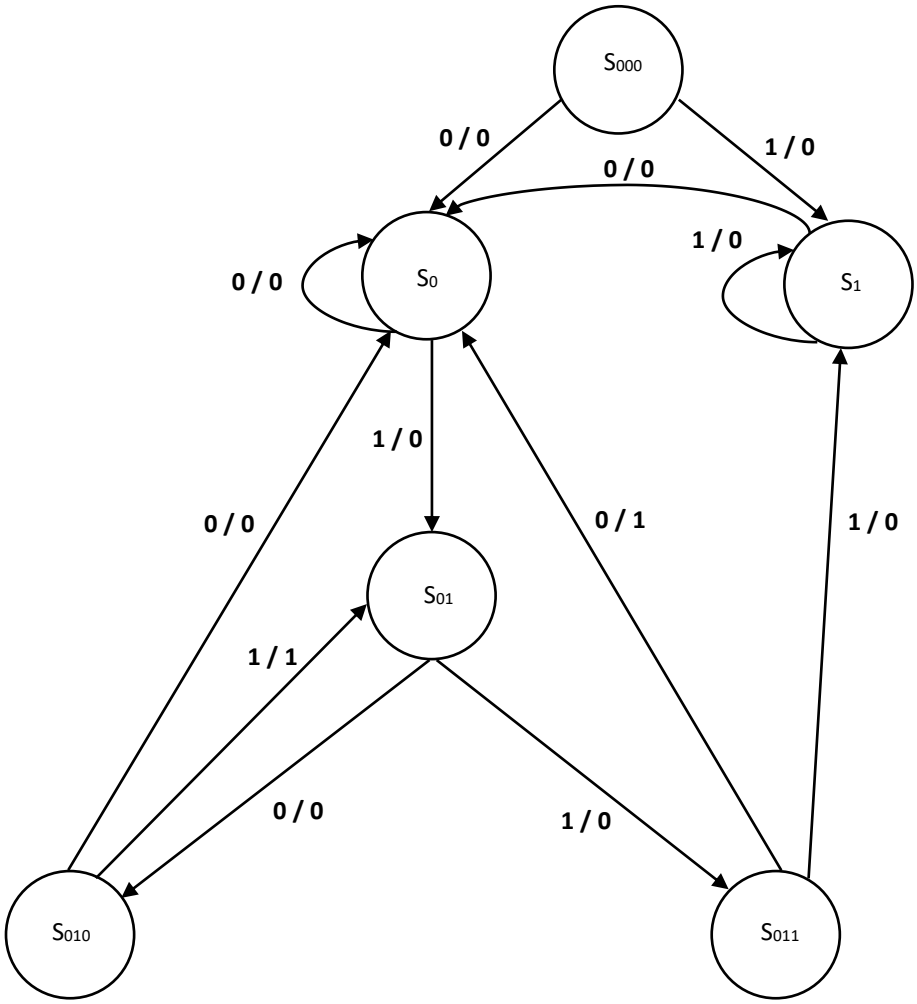
16.21 *Ascending sequence detector.* Write a Verilog module that accepts an eight-byte sequence on eight-bit input `in`, where the first byte is signaled by a single-bit `start` signal. Your module should assert a single-bit `done` signal on the cycle after the last byte is input. In the same cycle, it should assert a single-bit `in_sequence` signal if the eight bytes were in ascending sequence; that is, if the $i + 1$st byte is one more than the ith byte, $b_{i+1} = b_i + 1$ for i, from 1 to 7.

```
module edge_detection(
      input clock,
      input input_signal,
      output reg rising_transition
      );
      // declarations
      reg n;
      wire rising_transition_c;
      // logic to detect 0 in previous cycle and 1 in current cycle
      assign rising_transition_c = ~n & input_signal;
      // flip-flop instantiations
      always @(posedge clock) begin
            n <= #1 input_signal;
            rising_transition <= #1 rising_transition_c;
      end
endmodule

module hw3p1(input clk, input [7:0]in, input start, output reg done, output
reg in_sequence);

      parameter [1:0]WAIT = 2'b00;
      parameter [1:0]SEQ_START = 2'b01;
      parameter [1:0]IN_SEQ = 2'b11;
      parameter [1:0]DONE = 2'b10;

      reg [1:0]state;
      reg [1:0]state_c;
      reg [7:0]prev_in;
      reg done_c, in_sequence_c;
```

```
	wire edge_start;
	reg in_seq_flag;

	reg [3:0]i;
	reg [3:0]i_c;

	edge_detection e(clk, start, edge_start);

	initial begin
		in_seq_flag = 0;
		i_c = 0;
		done_c = 0;
		in_sequence_c = 0;
	end

	always @(in) begin
		// default stay in the same state
		state_c = state;
		case (state)
			WAIT: begin
				done_c = 0;
				in_sequence_c = 0;
				if (edge_start == 1'b1) begin
					state_c = SEQ_START;
					i_c = 1;
					in_seq_flag = 1;
				end
			end
			SEQ_START: begin
				state_c = IN_SEQ;
				i_c = i + 1;
				in_seq_flag = 1;
			end
			IN_SEQ: begin
				i_c = i + 1;
				//$display("in = %d, prev_in = %d, i_c = %d, i = %d,
state = %b, in_seq = %b", in, prev_in, i_c, i, state, in_seq_flag);
				if (i_c > 7) begin
					state_c = DONE;
					done_c = 1;
					in_sequence_c = in_seq_flag;
				end
				else begin
					if (!(in == (prev_in + 1))) begin
						in_seq_flag = 0;
					end
					else begin
						in_seq_flag = 1;
```

```
                        end
                    end
                end
                DONE: begin
                    state_c = WAIT;
                    done_c = 0;
                    in_sequence_c = 0;
                    i_c = 0;
                end
                default: begin
                    state_c = WAIT;
                    in_seq_flag = 0;
                    i_c = 0;
                    done_c = 0;
                    in_sequence_c = 0;
                end
            endcase
        end

        always @(posedge clk) begin
            state <= #1 state_c;
            prev_in <= #1 in;
            done <= #1 done_c;
            in_sequence <= #1 in_sequence_c;
            i <= #1 i_c;
        end

endmodule
```

2) FSM Design.
A Mealy finite state machine has one input *X* and two outputs *Y* and *Z*. The output *Y* = 1 occurs when 0110 is observed on the input *X*. The output *Z* = 1 occurs when 0101 is observed on the input *X*. Draw the state diagram of this machine using the minimum number of states.
A typical input-output behavior is shown below:
X 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0
Y 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
Z 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
a) Draw the state diagram of this machine using the minimum number of states.
(30 points)



b) Write a Verilog model of the finite state machine. (40 points)

```
module hw3p2(input clk, input X, output Y, output Z);
	// register for sequence
	reg S2, S1, S0;

	// output signals - Mealy, so no registered outputs
	assign Y = (~X) & S2 & S1 & (~S0);  // sequence 0110
	assign Z = (X) & (~S2) & S1 & (~S0);  // sequence 0101

	// shift signals in sequence X->S2->S1->S0
	always @(posedge clk) begin
		S0 <= #1 S1;
		S1 <= #1 S2;
		S2 <= #1 X;
	end

endmodule
```