CPE 133: Digital Design

Prof John Callenes-Sloan
Luis Gomez & Richard Hua

Experiment 13: 3 Bit FSM Counter

Summary:
In exp 13 we successfully created a 3-bit counter for an arbitrary sequence 0,2,4,5,7... We created our own clock module to be able to display the sequence at a reasonable rate via the 7-Segment display module.

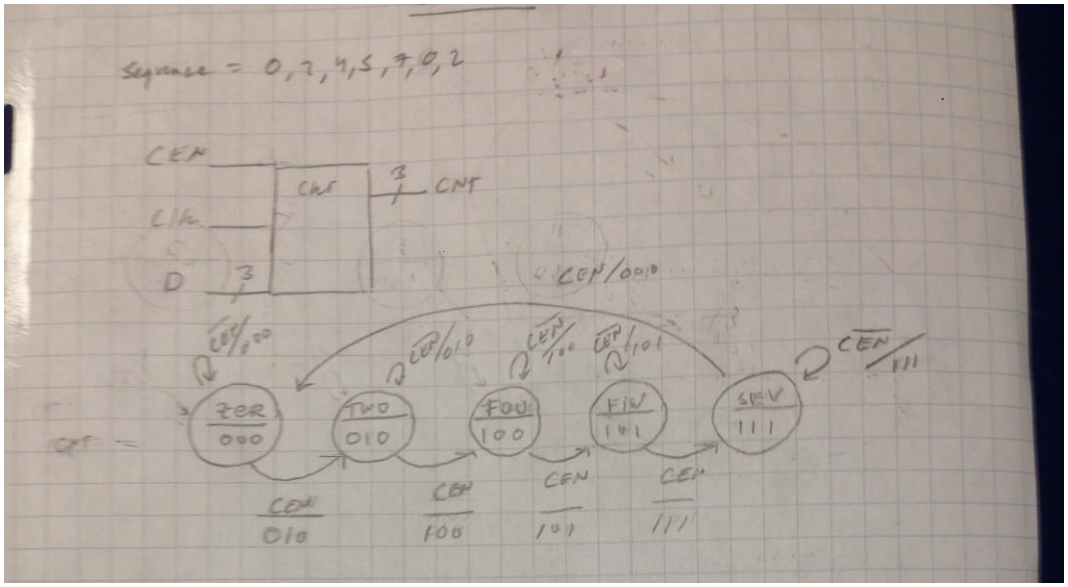


Figure 1: Moore State Diagram of 3-Bit FSM Counter

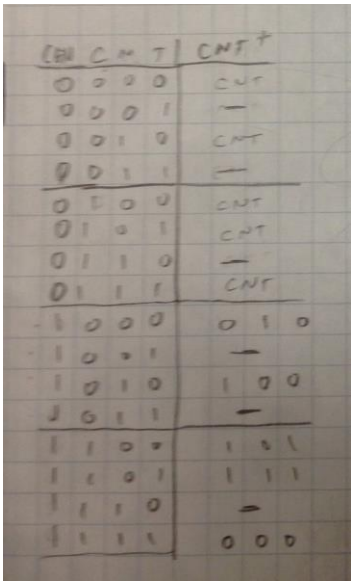


Figure 2: Excitation Table for 3-Bit FSM Counter

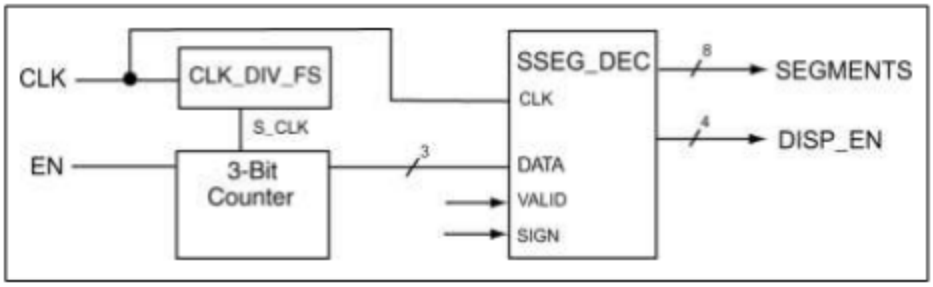


Figure 3: BBD for 3-Bit FSM Counter

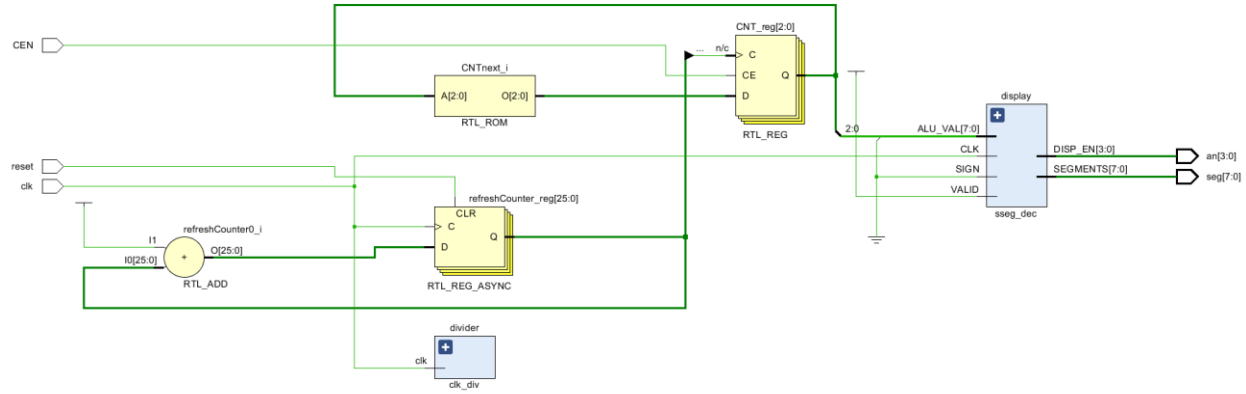


Figure 4: Elaborated Circuit Design for 3-Bit FSM Counter

Verification: In the video demo provided below, we demonstrate the function of our 3-bit counter. We configured our clock-module to provide a reasonable rate of edge-triggers to toggle the sequence.

https://youtu.be/sUVPceepQOk

Questions:

1. How many flip-flops did this FSM design use? Recall that even if you did not explicitly create a flip flop, the synthesizer would still create flip flops for you based on your behavioral model.

The FSM design used 3 flip-flops to represent the 5 different states.

2. How hard would it to add four more numbers into the counting sequence using the Verilog behavioral approach to FSM design?

This would not be very hard since all we have to do is add four more cases to represent the numbers that we want to add into the sequence.

3. What would happen if this FSM were to suddenly find itself displaying the count of ‘3’?

The FSM would change to zero on the next rising edge, effectively resetting itself, so that it can properly display the next correct number in the sequence.

Code:

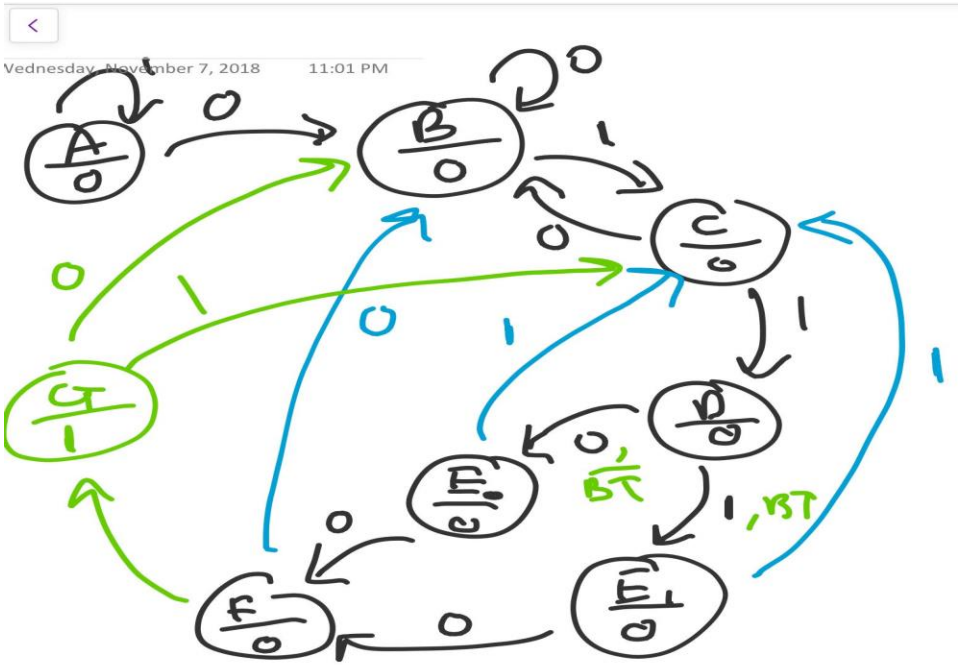
```
35  module fsm_counter(
36      input clk,
37      input CEN, reset,
38      output [3:0] an,
39      output [7:0] seg
40      );
41
42      wire sclk;
43      parameter [2:0] ZERO=3'b000, TWO=3'b010, FOUR=3'b100, FIVE=3'b101, SEVEN=3'b111;
44      reg [2:0] CNT = 3'b000;
45      reg [2:0] CNTnext ;
46
47      sseg_dec display( CNT, 0 , 1, clk, an, seg ) ;
48      clk_div divider( clk, sclk ) ;
49
50      wire LEDcounter;
51      reg [25:0] refreshCounter;
52      always @(posedge clk or posedge reset)
53          begin
54           if(reset == 1)
55            refreshCounter <= 0;
56           else
57            refreshCounter <= refreshCounter + 1;
58          end
59          assign LEDcounter = refreshCounter[25:24];
```

```
60
61      always @ ( posedge LEDcounter )
62      begin
63      case (CEN) // check Enable
64          1'b1: begin // Enable active-low
65              CNT <= CNTnext ;
66          end
67          1'b0: begin
68              CNT <= CNT ;
69          end
70      endcase
71      end
72
73      always @ ( CNT )
74      begin
75          case (CNT)
76                              ZERO: begin
77                                  CNTnext <= TWO;
78                                  end
79                              TWO: begin
80                                  CNTnext <= FOUR;
81                                  end
82                              FOUR: begin
83                                  CNTnext <= FIVE;
84                                  end
85                              FIVE: begin
86                                  CNTnext <= SEVEN;
87                                  end
88                              SEVEN: begin
89                                  CNTnext <= ZERO;
90                                  end
91                              default: CNTnext <= ZERO;
92         endcase
93      end
94  endmodule
```

Experiment 14: Sequence Detector FSM

Summary:

In exp 14 we attempted to create a Sequence Detector FSM utilizing the provided “SEQ_DVR” and “BC_DEC” modules. The sequence detector has the following inputs: a Clock signal ,push-Button , and a 8-bit switch vector. The Sequence Detector has 12 bits of output via the 7-segment display module. The push-button input is a form of external control that determines which of the two different sequences is to be detected.



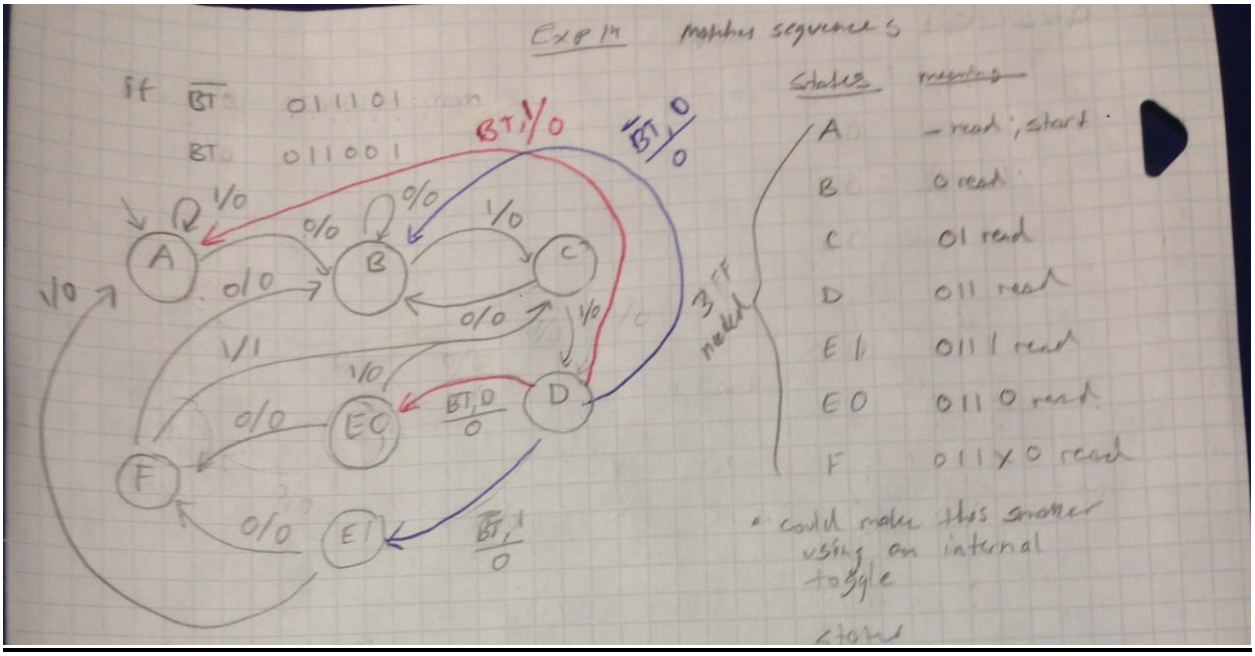


Figure 5: Mealy & Moore State Diagram of Sequence Detector & State Table

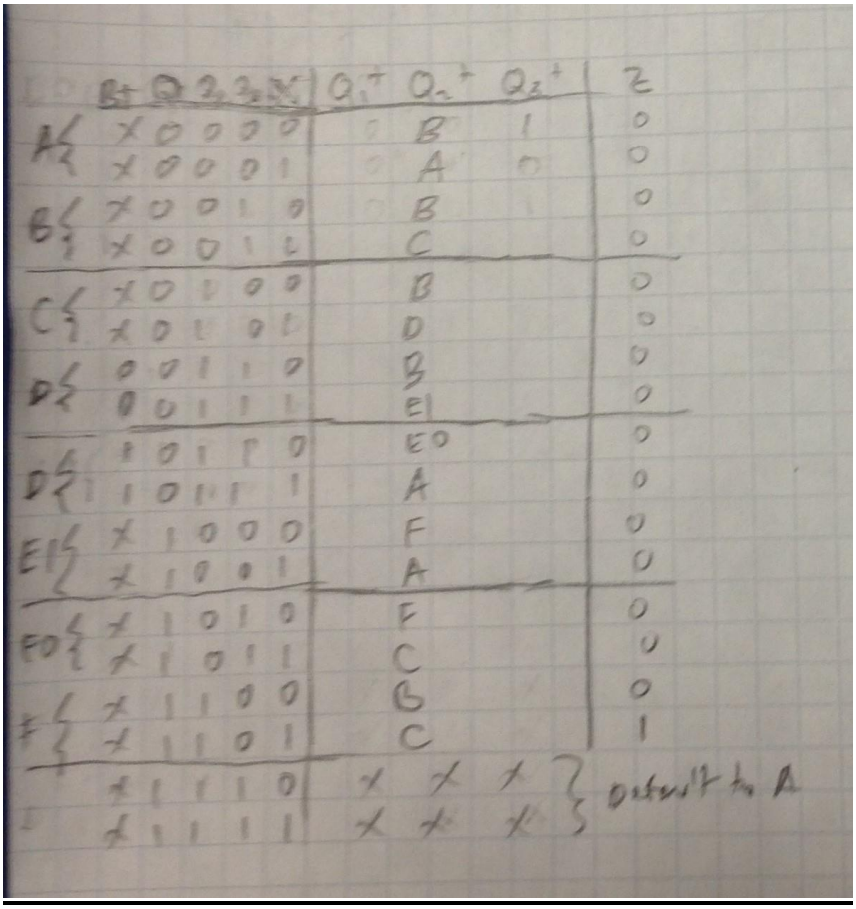


Figure 6: Next State Table for Sequence Detector, note that we only care about the Button toggle during the D-state

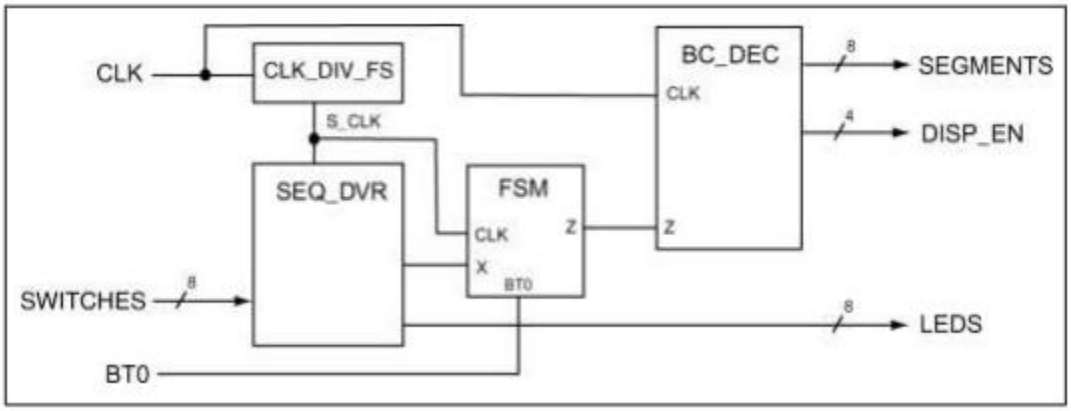


Figure 7: BBD of Sequence Detector

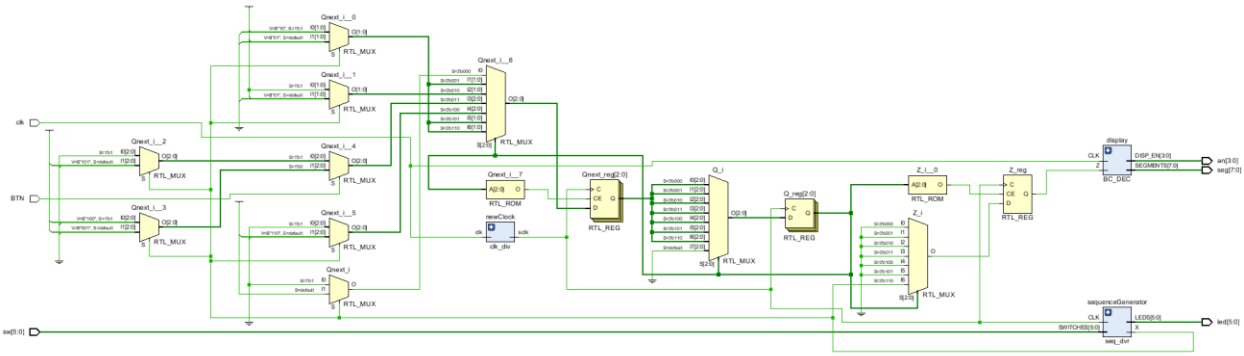


Figure 8: Elaborated Circuit Design for Sequence Detector

Verification: We provided a video demo below of our attempt at creating the Sequence Detector. We believe that our state-diagram and next-state table were correct but that our implementation in Vivado requires debugging and review. We were able to detect one of the two sequences provided but our display did not display COOL but rather C8AP and Cr8P, but we haven't discerned why.

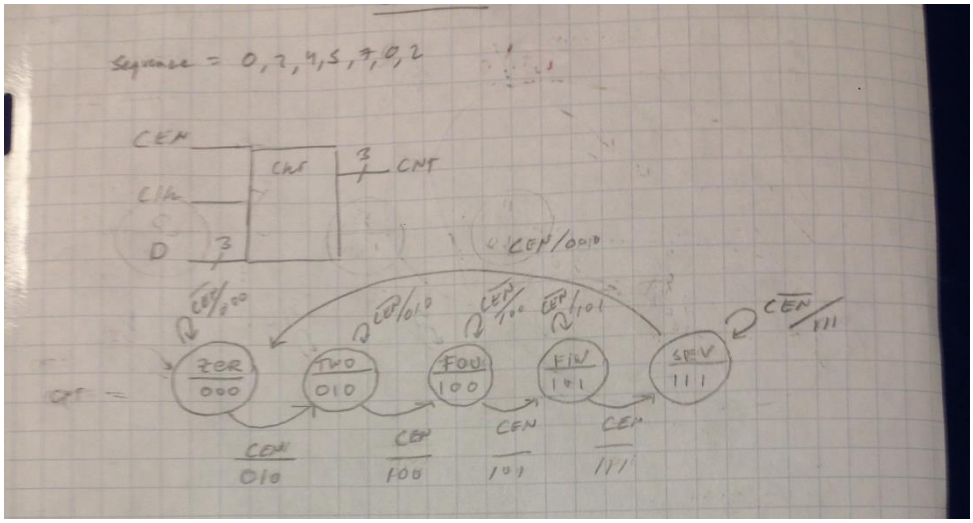
https://youtu.be/4I3TGv38HjY

Questions:

1. Describe at least two applications where sequence detectors could potentially be useful.

Sequence detectors could be used to decode packets of transmitted data or for very basic decryption, by mapping sequences of binary values to messages or letters of the alphabet. With a parity checker, a sequence detector could possibly do error correction/detection given a sufficiently capable hamm-code.

2. Provide Mealy-type state diagrams for the resetting version of the FSM you designed in this lab activity.



3. Briefly explain the function of the SEQ_DVR box in Figure 22. You'll need to take a look at the Verilog model for this question.

SEQ_DVR at a high level is a 3-bit counter module paired with an encoder. With each edge-trigger of the clock the module increments a 3-bit value and uses this value to call the index of 1 of 8 switch inputs. The reference element in the 8b-switch vector is then assigned to the output value X. Over the course of many cycles, the module will output the state of the switch inputs in order ,which we interpret as a sequence.

Code:

```
37
38  module fsm_sequence_detector(
39      input clk,
40      input BTN,
41      input [5:0] sw,
42      output [5:0] led,
43      output [3:0] an,
44      output [7:0] seg
45      );
46
47      wire sclk ;
48      wire X ;
49      parameter [2:0] A=3'b000, B=3'b001, C=3'b010, D=3'b011 ;
50      parameter [2:0] E1=3'b100, E0=3'b101, F=3'b110 ;
51      reg Z ;
52      reg [2:0] Q , Qnext = A;
53
54      // sseg_dec display ( CNT, 0 , 1, clk, an, seg ) ;
55      clk_div newClock ( clk, sclk ) ;
56      BC_DEC display ( clk, Z, an, seg ) ;
57      seq_dvr sequenceGenerator ( sclk, sw, led, X ) ;
58
59      always @ ( posedge sclk )
60      begin
61          Q <= Qnext ;
62
63          case (Q)
64                          A: begin
65                              Z <= 1'b0 ;
66                              if (X)
67                                  Qnext <= A ;
68                              else Qnext <= B ;
69                              end
70                          B: begin
71                              Z <= 1'b0 ;
72                              if (X)
73                                  Qnext <= C ;
74                              else Qnext <= B ;
75                              end
76                          C: begin
77                              Z <= 1'b0 ;
78                              if (X)
79                                  Qnext <= D ;
80                              else Qnext <= B ;
81                              end
```

```
                        D: begin
                            Z <= 1'b0 ;
                            case (BTN)
                                1'b1: begin
                                    if (X)
                                        Qnext <= A ;
                                    else Qnext <= E0 ;
                                end
                                1'b0: begin
                                    if (X)
                                        Qnext <= E1 ;
                                    else Qnext <= B ;
                                end
                            endcase
                            end
                        E1: begin
                            Z <= 1'b0 ;
                            if (X)
                                Qnext <= A ;
                            else Qnext <= F ;
                            end
                        E0: begin
                            Z <= 1'b0 ;
                            if (X)
                                Qnext <= C ;
                            else Qnext <= B ;
                            end
                        F: begin
                            if (X) begin
                                Qnext <= C ;
                                Z <= 1'b1;
                                end
                            else begin
                                Qnext <= B ;
                                Z <= 1'b0 ;
                                end
                            end
                        default: Q <= A ;
        endcase
    end
endmodule
```