LAB 7

Group C

Edward Enriquez, eenriquez@cpp.edu, ID 013242619
Lorenzo Antonio Fabian Lopez, lfabianlopez@cpp.edu, ID 014903070
Rafael Gaeta, rgaeta@cpp.edu, ID 014522794

Contribution:
Edward Enriquez
50% Code creation and analysis

Lorenzo Antonio Fabian Lopez
25% Analysis of results and conclusion

Rafael Gaeta
25% Report and graphics

Abstract

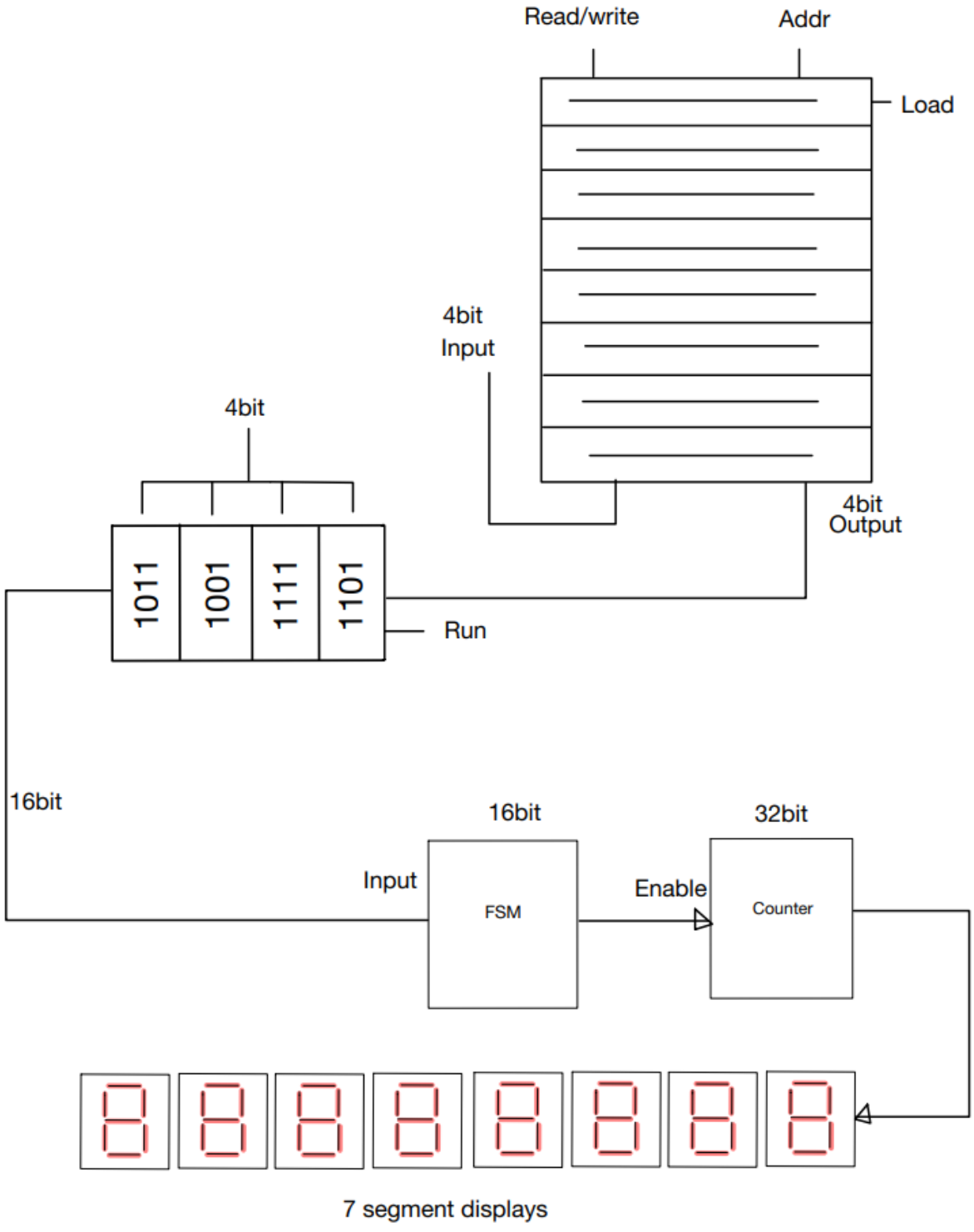
This lab consists in implementing a digital circuit design that mimics the specifications provided. The device will have 4 input switches to the Random Access Memory (RAM) with 3 switches to select in which location the 4-bit input gets loaded, along with 2 switches that read/write the input. The output is run through a register, followed by the Finite State Machine (FSM) where sequence detection is enabled and then using the up/down counter it is displayed on the seven segment display depending on how many times the sequence was detected in the FSM.

Specifications,

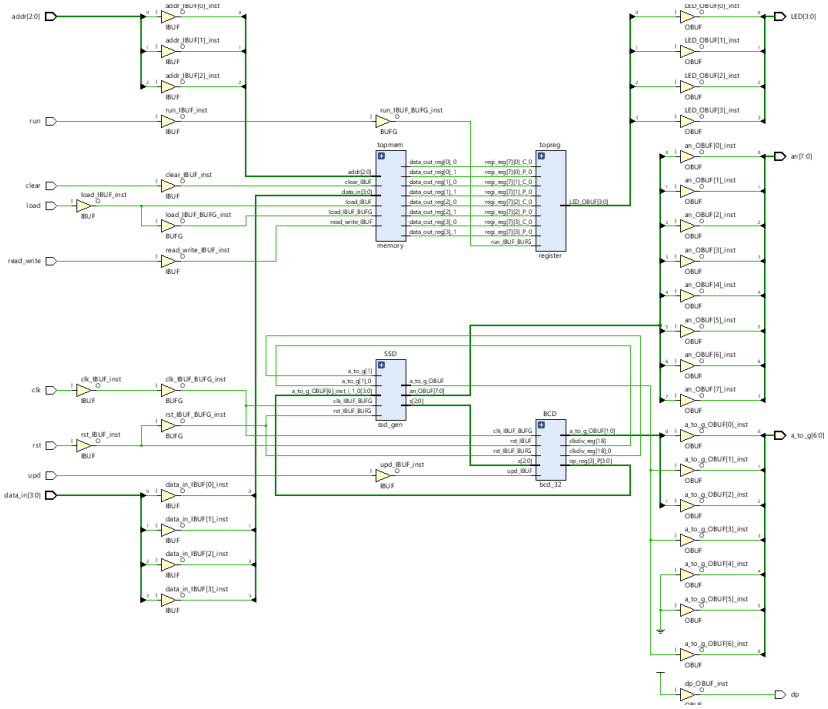
- Define Circuit Schematic/Diagram
- Structural Modeling Method
- Total power = No Specifications Given
- Number of LUT = No Specifications Given

Circuit Design

Diagram



Verilog Generated Schematic



Code Detail

The memory.v file modeled the memory of the system and had 8 address locations which were accessed using 3 bits. Each address was 4 bits. The load input was a button which either wrote data from the input to a specific address or read data into the output.

```
1  `timescale 1ns / 1ps
2  module memory#(parameter DATA_WIDTH=4, parameter ADDR_WIDTH=3)
3  (
4      input [DATA_WIDTH-1:0] data_in,
5      input [ADDR_WIDTH-1:0] addr,
6      output reg [DATA_WIDTH-1:0] data_out,
7      input load,
8      input read_write
9      );
10
11     localparam RAM_DEPTH=1<<ADDR_WIDTH;
12
13     reg [DATA_WIDTH-1:0] mem[0:RAM_DEPTH-1];
14
15     always@(posedge load)
16     begin
17     if(read_write)//write
18     mem[addr]=data_in;
19     else //read
20     data_out=mem[addr];
21     end
22 endmodule
23
```

memory.v

The next module in our system was the register.v. This file modeled a 2D array which stored four 4 bit words. Once the array/register is full, the whole register can be read sequentially to be inputted in the FSM.

```
`timescale 1ns / 1ps
module register#(parameter DATA_WIDTH=4)
    (
    input load,
    input clk,
    input run,
    input [3:0] ip,
    input clear,
    input read_write,
    output wire [15:0] op
    );

    integer n;
    reg [3:0] regi [0:3];
    reg in;
    reg [1:0] count;

    genvar i;
    generate
    for(i=0;i<4;i=i+1)
    begin
    assign op[(i*4)+3:i*4]=regi[i];
    end
    endgenerate

    initial
    in=0;


    always@(posedge load or posedge clk)
    begin
    if(clk)
    begin
    if(in==1)
            begin
            in=0;
                  if(!read_write)
                begin
                    for(n=1;n<4;n=n+1)
                    regi[n-1]<=regi[n];
                    regi[3]<=ip;
                    in=0;
                end
            end

    end
    if(load)
    begin
    in=in+1;
    end
    end




endmodule
```

register.v

The FSM.v modeled the sequence detector which we used to detect the bit sequence 1101. The input of this module are the 16 bit input passed on from the register, the clock which timed how fast the input was traversed, and the run button which starts the operation.

```
`timescale 1ns / 1ps

`define zero 3'b000
`define one 3'b001
`define two_ones 3'b010
`define two_ones_zero 3'b011
`define found 3'b100

module FSM(
    input [15:0] ip,
    input run,
    input clk,
    output reg value,
    output reg start,
    output reg [4:0] i,
    output reg test,
    output reg current_bit,
    output reg [3:0] state
    );

    reg [3:0] next_state;
    reg current_bit;
    reg [4:0] count;

    initial
    begin
    start=0;
    count=0;
    end


    always@(posedge clk or posedge run)
    begin
        if(run)
        begin
        start=1;
        current_bit=ip[0];
        i=0;
        state=0;
        end
    else
    begin
    if(start)
    begin
    test=1;
    i=i+1;
    current_bit=ip[i];
    state=next_state;
    if(i==17)
    begin
    i=0;
    start=0;
    if(state==`found)
    state=`zero;
    end
    end
    end
    end


    always@(current_bit or state)
    begin
    case(state)
    `zero:
        begin
        if(current_bit) next_state=`one;
        else next_state=`zero;
        end
    `one:
        begin
        if(current_bit) next_state=`two_ones;
        else next_state=`zero;
        end
    `two_ones:
        begin
        if(current_bit) next_state=`two_ones;
        else next_state=`two_ones_zero;
        end
    `two_ones_zero:
        begin
        if(current_bit) next_state=`found;
        else next_state=`zero;
        end
    `found:
        begin
        if(current_bit) next_state=`one;
        else next_state=`zero;
        end
    endcase
    end

    always@(posedge run)
    begin

    end

    always@(state)
    begin
        case(state)
        `zero:value=0;
        `one:value=0;
        `two_ones:value=0;
        `two_ones_zero:value=0;
        `found:value=1;
        endcase
    end
endmodule
```

FSM.v

The top.v file held all the instantiations including the files from the previous labs. For the input switches, there was the speed select of the clock *SW,* the address pointer, *data_in*, read/write switch, and the up counter/down counter select. For the input buttons we have a load button which reads and writes to the register and a run button which inputs the register sequentially into the sequence detector finite state machine.

```
1  `timescale 1ns / 1ps
2  module top#(parameter DATA_WIDTH=4, parameter ADDR_WIDTH=3)
3              (
4               input [DATA_WIDTH-1:0] data_in,
5               input [ADDR_WIDTH-1:0] addr,
6               input load,
7               input read_write,
8               input run,
9               input clear,
10              input clk,
11              input upd,
12              input [4:0] SW,
13              input rst,
14              output wire [6:0] a_to_g,
15              output wire [7:0] an,
16              output wire dp,
17              output wire [15:0] LED
18              );
19
20
21              wire [23:0] count_wire;
22              wire [DATA_WIDTH-1:0] data_out_wire;
23              wire [15:0] stream_wire;
24              wire [3:0] size_wire;
25              wire enable_wire;
26              wire [31:0] display;
27
28              assign display={4'b0,4'b0,count_wire};
29              assign LED=stream_wire[15:0];
30
31
32          wire clk_div;
33          clk_divider  GEN    (
34                               .clk(clk),
35                               .rst(rst),
36                               .SW(SW),
37                               .clk_div(clk_div)
38                               );
39
40             memory topmem (
41                            .data_in(data_in),
42                            .addr(addr),
43                            .load(load),
44                            .read_write(read_write),
45                            .data_out(data_out_wire)
46                            );
47
48               register topreg(
49                               .load(load),
50                               .clk(clk_div),
51                               .run(run),
52                               .read_write(read_write),
53                               .ip(data_out_wire),
54                               .clear(clear),
55                               .op(stream_wire)
56                              );
57
58              FSM topfsm(
59                      .ip(stream_wire),
60                      .run(run),
61                      .clk(clk_div),
62                      .value(enable_wire)
63                      );
64
65           bcd_32  BCD    (
66                           .clk(clk_div),
67                           .rst(rst),
68                           .en(enable_wire),
69                           .upd(upd),
70                           .count(count_wire),
71                           .load(1'b0),
72                           .value(4'b0000),
73                           .select(3'b000)
74                           );
75
76          ssd_gen   SSD   (
77                            .SW(display),
78                            .clk(clk),
79                            .rst(rst),
80                            .a_to_g(a_to_g),
81                                           .an(an),
82                                           .dp(dp)
83                                          );
84
85
86 endmodule
```

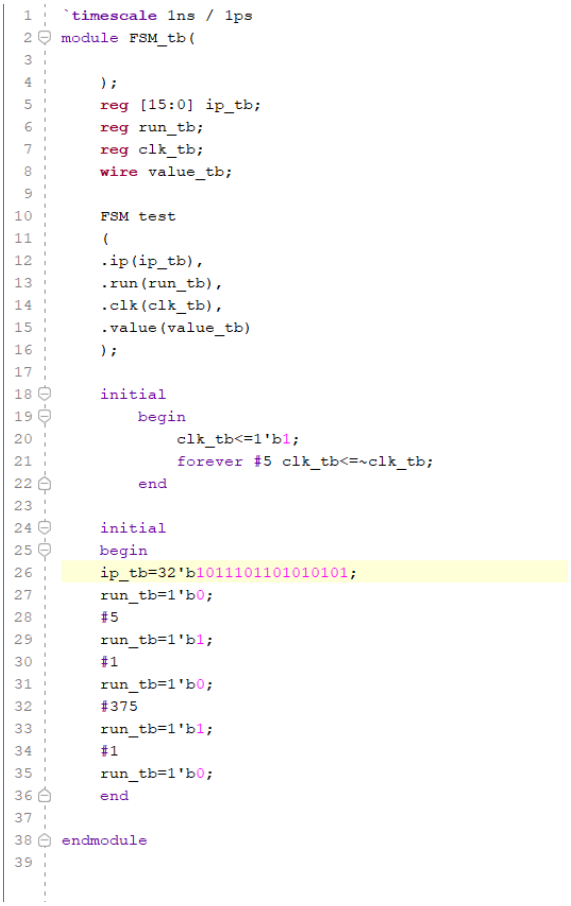
top.v

Test Bench

A testbench module was created for each of the new modules in this lab. Each module was sequential so it was important for the previous module to work so that the ones following it could also work properly.

```
`timescale 1ns / 1ps
module memory_tb#(parameter DATA_WIDTH_TB=4, parameter ADDR_WIDTH_TB=3)
    (
    );
    reg [DATA_WIDTH_TB-1:0] data_in_tb;
    reg [ADDR_WIDTH_TB-1:0] addr_tb;
    wire [DATA_WIDTH_TB-1:0] data_out_tb;
    reg load_tb;
    reg read_write_tb;
    memory testbench
    (
        .data_in(data_in_tb),
        .addr(addr_tb),
        .data_out(data_out_tb),
        .load(load_tb),
        .read_write(read_write_tb)
    );
    initial
    begin
    data_in_tb=4'b1011;
    load_tb=1'b0;
    read_write_tb=1'b1;
    addr_tb=3'b000;
    #5
    load_tb=1'b1;
        #1 load_tb=1'b0;
        #1
        addr_tb=3'b010;
        data_in_tb=4'b1111;
        #5
        load_tb=1'b1;
        #1 load_tb=1'b0;
        #1
        addr_tb=3'b000;
        read_write_tb=1'b0;
        #5
        load_tb=1'b1;
        #1 load_tb=1'b0;
        #1
        addr_tb=3'b010;
        #5
        load_tb=1'b1;
        #1 load_tb=1'b0;
        #1;
        end
endmodule
```

memory_tb.v



```
`timescale 1ns / 1ps
module FSM_tb(

    );
    reg [15:0] ip_tb;
    reg run_tb;
    reg clk_tb;
    wire value_tb;

    FSM test
    (
    .ip(ip_tb),
    .run(run_tb),
    .clk(clk_tb),
    .value(value_tb)
    );

    initial
        begin
            clk_tb<=1'b1;
            forever #5 clk_tb<=~clk_tb;
        end

    initial
    begin
    ip_tb=32'b1011101101010101;
    run_tb=1'b0;
    #5
    run_tb=1'b1;
    #1
    run_tb=1'b0;
    #375
    run_tb=1'b1;
    #1
    run_tb=1'b0;
    end

endmodule

```

FSM_tb.v

```
`timescale 1ns / 1ps

module top_tb#(parameter DATA_WIDTH_TB=4, parameter ADDR_WIDTH_TB=3)
(

    );
    reg [DATA_WIDTH_TB-1:0] data_in_tb;
    reg [ADDR_WIDTH_TB-1:0] addr_tb;
    reg load_tb;
    reg read_write_tb;
    reg run_tb;
    reg clear_tb;
    reg clk_tb;
    reg upd_tb;
    reg rst_tb;
    wire [6:0] a_to_g_tb;
    wire [7:0] an_tb;
    wire dp_tb;


    top toptest
    (
    .data_in(data_in_tb),
    .addr(addr_tb),
    .load(load_tb),
    .read_write(read_write_tb),
    .run(run_tb),
    .clear(clear_tb),
    .clk(clk_tb),
    .upd(upd_tb),
    .rst(rst_tb),
    .a_to_g(a_to_g_tb),
    .an(an_tb),
    .dp(dp_tb)
    );

    initial
        begin
            clk_tb<=1'b1;
            forever #5 clk_tb<=~clk_tb;
        end
            forever #5 clk_tb<=~clk_tb;
        end

    initial
    begin
    data_in_tb=4'b1011;
    addr_tb=3'b000;
    load_tb=1'b0;
    read_write_tb=1'b1;
    run_tb=1'b0;
    clear_tb=1'b0;
    upd_tb=1'b0;
    rst_tb=1'b0;
    #10
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    read_write_tb=1'b0;
    #5
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
    #8
    load_tb=1'b1;
    #10
    load_tb=1'b0;
       end

endmodule

```

top_tb.v

```
`timescale 1ns / 1ps
module register_tb(

    );
    reg load_tb;
    reg run_tb;
    reg clear_tb;
    reg [3:0] ip_tb;
    wire [31:0] op_tb;
    wire [2:0] count_tb;

    register test
    (
        .load(load_tb),
        .run(run_tb),
        .clear(clear_tb),
        .ip(ip_tb),
        .op(op_tb),
        .count(count_tb)
    );

    initial
    begin
    clear_tb=1'b0;
    #1
    clear_tb=1'b1;
    #1
    clear_tb=1'b0;
    load_tb=1'b0;
    ip_tb=4'b0101;
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #1
    ip_tb=4'b0111;
    #1
    ip_tb=4'b0111;
    #4
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    clear_tb=1'b1;
    #1
    clear_tb=1'b0;
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5
    load_tb=1'b1;
    #1
    load_tb=1'b0;
    #5;
    end
endmodule
```

register_tb.v

Corner Cases

This design has some very important corner cases. The finite state machine (FMS) has to save the state for the next loaded value coming from the register. If this does not happen, it will cause the system to behave unexpectedly.

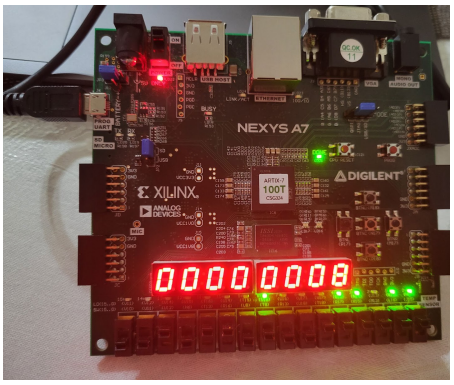
Technical Report

After the simulation was performed. The results obtained from the report are as follows. The total power consumption is 0.111W. The power obtained this time compared to previous labs is lower than expected. This is likely due to timing and the critical path which suggests that the design was optimized effectively as the simulation was instantiated. Although the results show very low power compared to previous labs, the number of lookup tables was significantly bigger. A total of 118 LUTs were used which also suggests that the design can be further implemented to improve performance and decrease power usage even lower.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
✓ synth_1	constrs_1	synth_design Complete!								121	121	0.0	0	0	11/8/21, 9:01 PM	00:02:43	Vivado Synthesis Default
✓ impl_1	constrs_1	write_bitstream Complete!	7.299	0.000	0.252	0.000	0.000	0.111	0	118	133	0.0	0	0	11/8/21, 9:04 PM	00:05:14	Vivado Implementation

Synthesis results for design

Once the simulation was complete and the bitstream created the file was transferred to the Nexys 7. The board was tested and the design worked according to the specification requirements provided in the abstract.



Programmed Nexys 7

Conclusion

After programming the board we were able to load the RAM with 4 bit inputs using the 3-bit selector to determine which slot in the memory. Using the read button it was able to output the numbers selected from the memory and loaded into the register where then the FSM successfully detected the sequence specified (w/out overlap) and displayed the count on the seven segment display.