Lab 13 Submission

**Serial Parity Generator**

CPE 133 - 03

Michael Hegglin

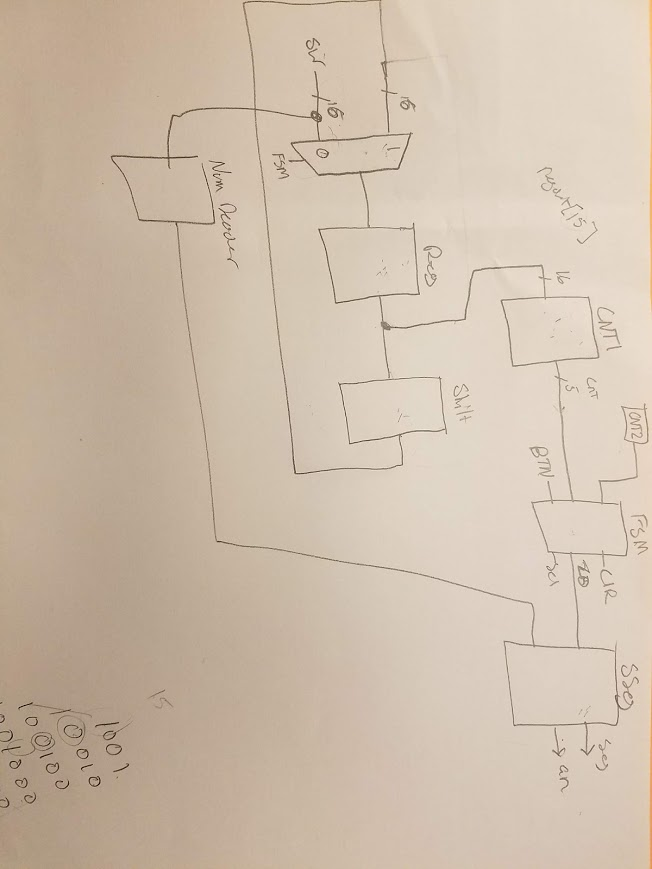
Jonathan Skelly

**Executive Summary:**

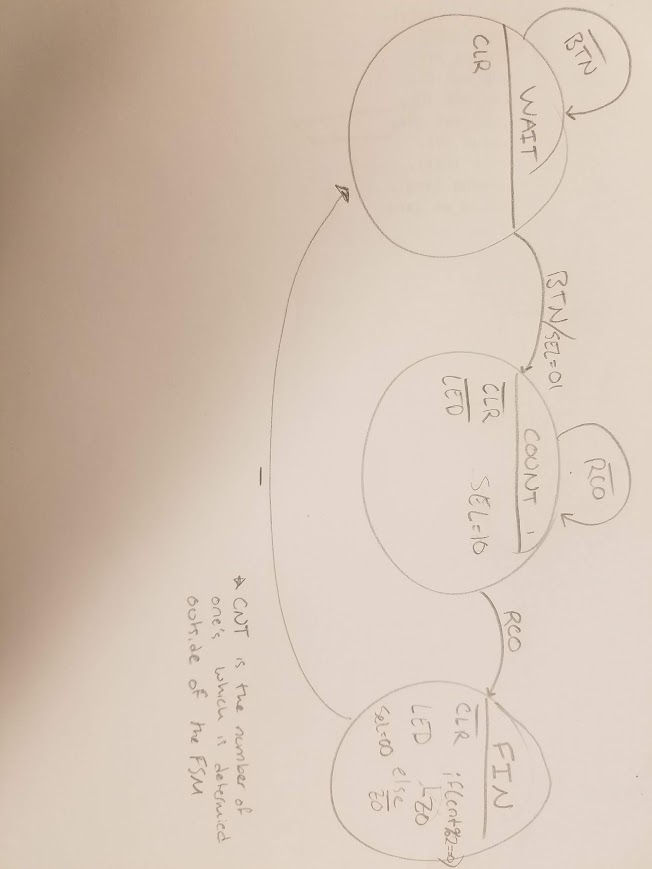
We designed an 16-bit even odd parity generator with a stone-age number. The generator used an FSM, a seven segment decoder, counters, muxes, and external inputs like buttons and switches. The logic decided which numbers would be counted to decide parity.



**High Level BBD**



**Lower Level BBD**

****

**FSM State Diagram**

**Questions:**

1. Briefly describe why the pros and cons of having two ways to calculate parity for a given set of bits. In other words, describe why we have both a “human version” (associated with methods that count the number of set bits) and a “hardware version”?

* People can calculate parity simply by counting. Hardware needs to feed each bit into an XOR gate to determine the parity. We see the human version as an easier way, however, hardware cannot count like us.

2. Briefly describe the main purpose of a parity generation and detection.

* Parity makes sure that no data is changed or lost. If the parity generated and detected do not match, then there was an error somewhere along the way.

3. Shift registers are synchronous circuits. Describe what limits how fast you can shift a simple shift register and still have the device operate properly.

* The limits of how fast you can shift a simple shift register depend on the setup and hold times of the flip-flops that make up the register. If the shift register is too fast, then the circuit will have an output that is neither high nor low.

4. Another standard function performed by shift register is “rotation”, such as rotate right and rotate left. Briefly describe the changes you would need to do to the shift register in this lab activity if you wanted the shift register to only rotate left.

* Shifting right → (when “01” sq <= sq[14:0] \* Din )
* Rotate right → (when “01” sq <= sq[14:0] \* sq[15])

5. One of the great selling points of many computer-type devices is that they do exclusively integer based math as opposed to using floating point math. Why would this be a good selling point for hardware? Briefly but completely support your answer.

* Integer based math is cheap, simple, fast, and easy to follow. Float based math is more expensive and harder to follow.

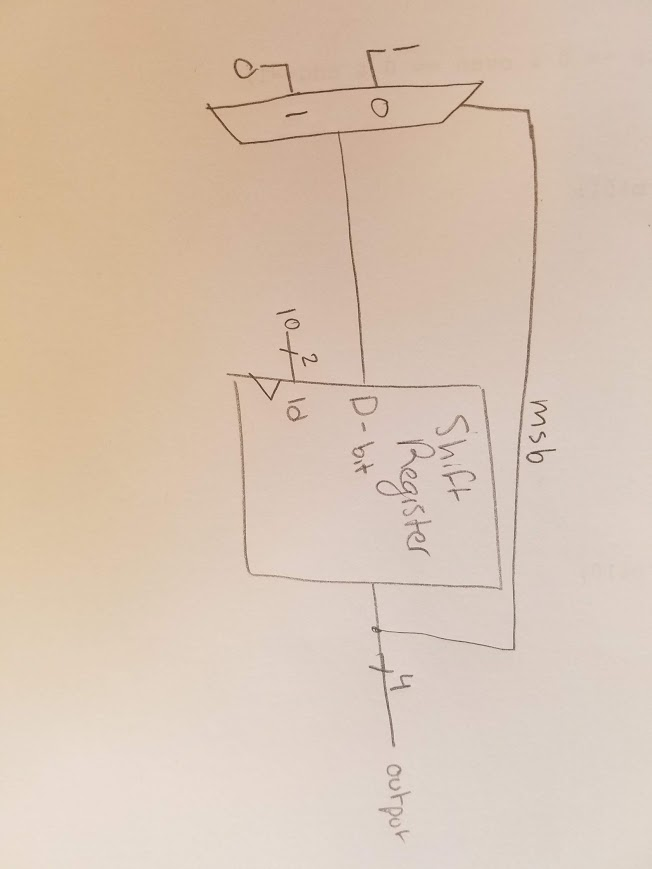
6. If a shift register was used with signed binary numbers (RC), briefly describe what you would have to do when you right-shift a number. For this problem, assume the shifted number result is always valid.

* The signed bit must always be copied and added onto the beginning of the new shifted number so the number keeps its original sign.

7. A shift-right operation officially performs truncation on the value in the shift register. Briefly explain what this means in the context of a shift register that shift right.

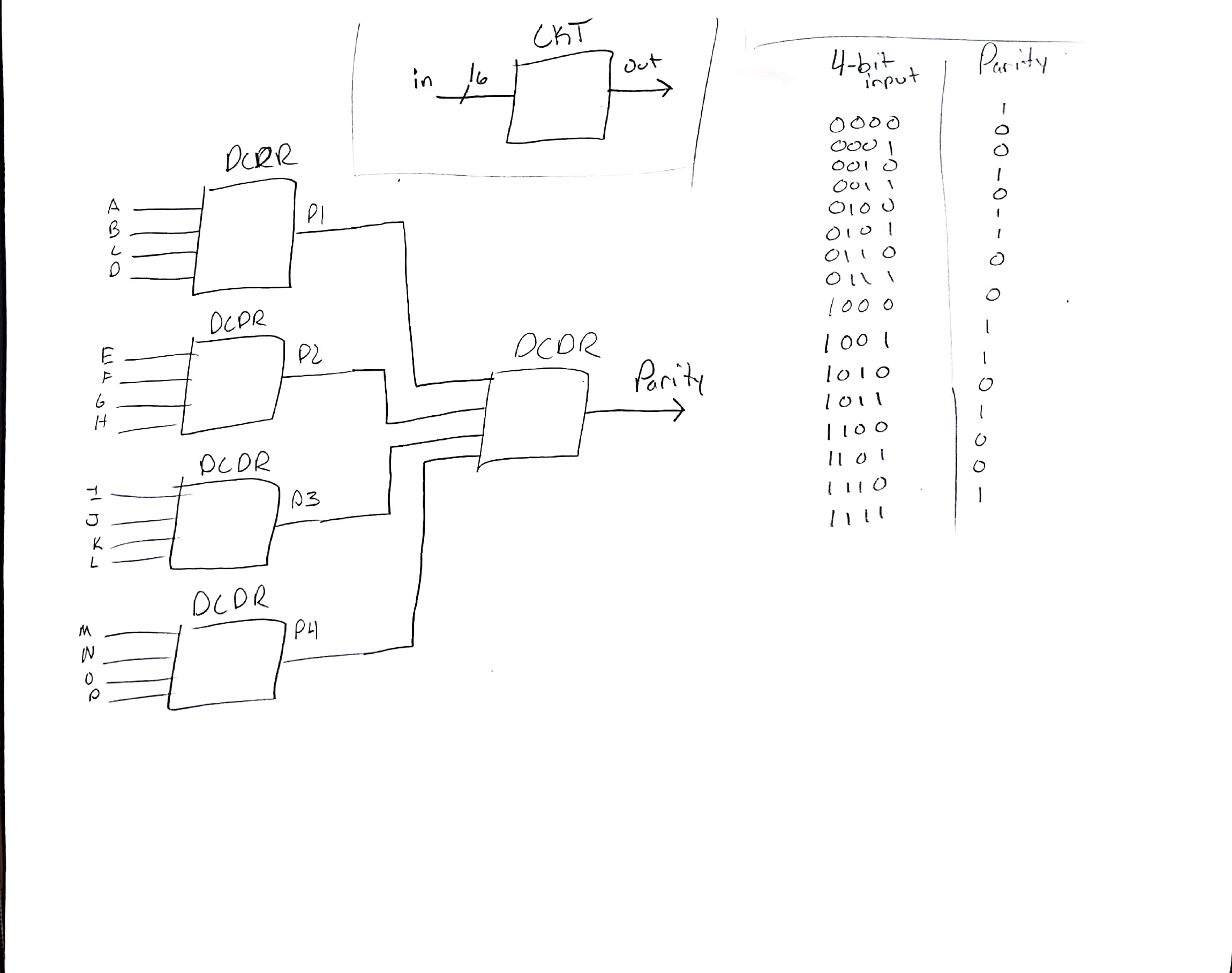
* If something is shifted to the right, data will be lost, and division is what occurred.

8. Show a diagram indicating how you would connect a shift register to obtain the following pattern on the shift register’s storage elements (example shows output for a four-bit shift register). Make sure your shift register has a way of getting into the first state listed below. Don’t use a FSM in your design; minimize your use of hardware in your design. Assume that the circuit starts at the 0000 value.



**Design Problems:**

1. Design a 16-bit parity checker using only 4-input 1-output decoders.



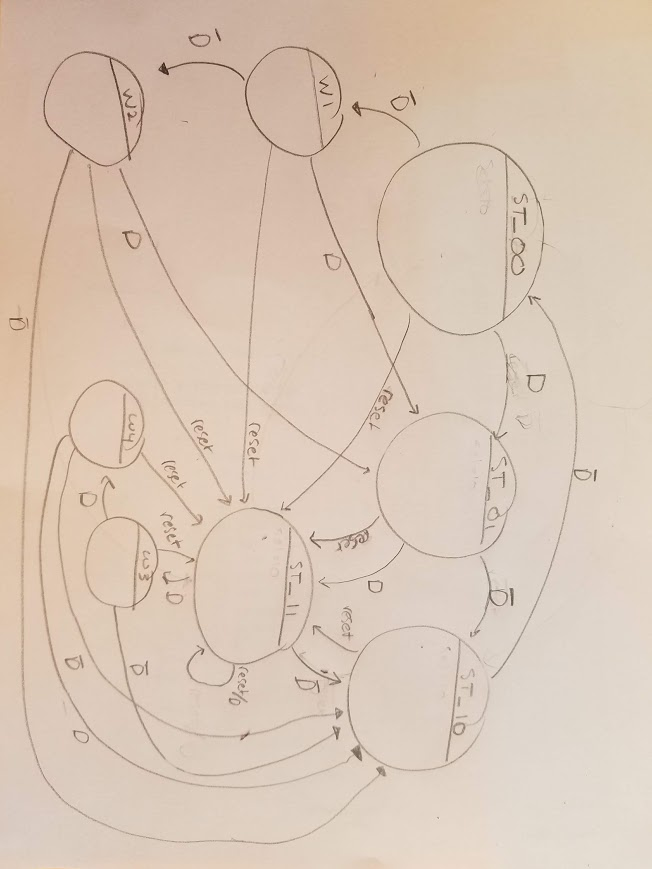
**High-level Black Body Diagram, next-level down, and decoder table for each of the 4-input 1-output decoders.**

2. Provide a state diagram that models the operation of a special 2-bit left shifting shift register. For this design, the shift register should never have the same output for more than three clock cycles. Avoid this condition by directing potential violations of this condition to state Q=”10”. Minimize the number of states in your design. Don’t use a counter in this design.

The D input represents the value that is synchronously shifted into the shift register

The Reset signal is an asynchronous input that places the input into the “11” state

The Q output is the value of the shift register



**Source Code:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: DogsWithJobs

// Engineer: Hegglin Skelly

//

// Create Date: 12/06/2018 08:05:03 AM

// Design Name: FSM module

// Module Name: fsm\_template

// Project Name: serial parity generator

// Target Devices:

// Tool Versions:

// Description: Generic FSM model with both Mealy & Moore outputs.

// Note: data widths of state variables are not specified

//

// Dependencies:

//

// Revision:

// Revision 1.00 - File Created (07-07-2018)

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module fsm\_template(reset\_n, rco, btn, clk, cnt1, ZO, clr, sel, LED);

input reset\_n, clk, btn;

input [4:0] cnt1, rco;

output reg ZO, clr, LED;

output reg [1:0] sel;

//- next state & present state variables

reg [1:0] NS, PS;

//- bit-level state representations

parameter [1:0] st\_WAIT=2'b00, st\_COUNT=2'b01, st\_END=2'b11;

//- model the state registers

always @ (negedge reset\_n, posedge clk)

if (reset\_n == 0)

PS <= st\_WAIT;

else

PS <= NS;

//- model the next-state and output decoders

always @ (\*)

begin

// ZO = 0; clr = 1; sel = 0; // assign all outputs

case(PS)

st\_WAIT:

begin

LED= 0;

clr = 1;

if (btn == 1)

begin

clr = 0;

sel = 01;

NS = st\_COUNT;

end

else

begin

NS = st\_WAIT;

end

end

st\_COUNT:

begin

clr = 0;

sel = 10;

if (rco == 5'b10001)

begin

NS = st\_END;

end

else

begin

NS = st\_COUNT;

end

end

st\_END:

begin

clr = 0;

sel = 00;

LED = 1;

case (cnt1)

0: ZO = 1;

1: ZO = 0;

2: ZO = 1;

3: ZO = 0;

4: ZO = 1;

5: ZO = 0;

6: ZO = 1;

7: ZO = 0;

8: ZO = 1;

9: ZO = 0;

10: ZO = 1;

11: ZO = 0;

12: ZO = 1;

13: ZO = 0;

14: ZO = 1;

15: ZO = 0;

16: ZO = 1;

default: ZO = 0;

endcase

NS = st\_WAIT;

end

default: NS = st\_WAIT;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: DogsWithJobs

// Engineer: Skelly/Hegglin

//

// Create Date: 11/02/2018 08:05:03 AM

// Design Name: Seven Seg Decoder

// Module Name: univ\_sseg

// Project Name: 4-Bit Up/Down Counter w/ Multiplexed 7-Segment

// Display

// Target Devices:

// Tool Versions:

// Description: Decoder for outputting binary values as decimal values // on the 7-segment LED display

//

// Dependencies:

//

// Revision:

// Revision 1.00 - File Created (07-07-2018)

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module main (sw, btn, clk, seg, an, LED);

input [15:0] sw;

input btn, clk;

output [7:0] seg;

output [3:0] an;

output LED;

reg [4:0] num;

wire [4:0] cnt2;

wire [4:0] cnt1;

wire ZO, clr, rco1, rco2, nclk;

wire [1:0] sel;

wire [15:0] shift, regout, muxout;

always @ (sw)

begin

case (sw)

16'b0000000000000000: num = 5'b00000;

16'b0000000000000001: num = 5'b00001;

16'b0000000000000011: num = 5'b00010;

16'b0000000000000111: num = 5'b00011;

16'b0000000000001111: num = 5'b00100;

16'b0000000000011111: num = 5'b00101;

16'b0000000000111111: num = 5'b00110;

16'b0000000001111111: num = 5'b00111;

16'b0000000011111111: num = 5'b01000;

16'b0000000111111111: num = 5'b01001;

16'b0000001111111111: num = 5'b01010;

16'b0000011111111111: num = 5'b01011;

16'b0000111111111111: num = 5'b01100;

16'b0001111111111111: num = 5'b01101;

16'b0011111111111111: num = 5'b01110;

16'b0111111111111111: num = 5'b01111;

16'b1111111111111111: num = 5'b10000;

default: num = 5'b00000;

endcase

end

cntr\_up\_clr\_nb #(.n(5)) MY\_CNTR1 (

.clk (nclk),

.clr (clr),

.up (shift[15]),

.ld (0),

.D (0),

.count (cnt1),

.rco (rco1) );

cntr\_up\_clr\_nb #(.n(5)) MY\_CNTR2 (

.clk (nclk),

.clr (clr),

.up (1),

.ld (0),

.D (0),

.count (cnt2),

.rco (rco2) );

usr\_nb #(.n(16)) MY\_USR (

.data\_in (sw),

.dbit (0),

.sel (sel),

.clk (nclk),

.clr (clr),

.data\_out (shift)

);

fsm\_template my\_fsm (

.reset\_n (1),

.rco (cnt2),

.btn (btn),

.clk (nclk),

.cnt1 (cnt1),

.ZO (ZO),

.clr (clr),

.sel (sel),

.LED (LED));

univ\_sseg mysseg (

.num (num),

.clk (clk),

.seg (seg),

.an (an),

.ZO (ZO));

clk\_divder\_nbit #(.n(23)) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

endmodule