Lab 10 Submission

**Full-Feature 3-Bit Up/Down Counter**

CPE 133 - 03

Michael Hegglin

Jonathan Skelly

**Executive Summary:**

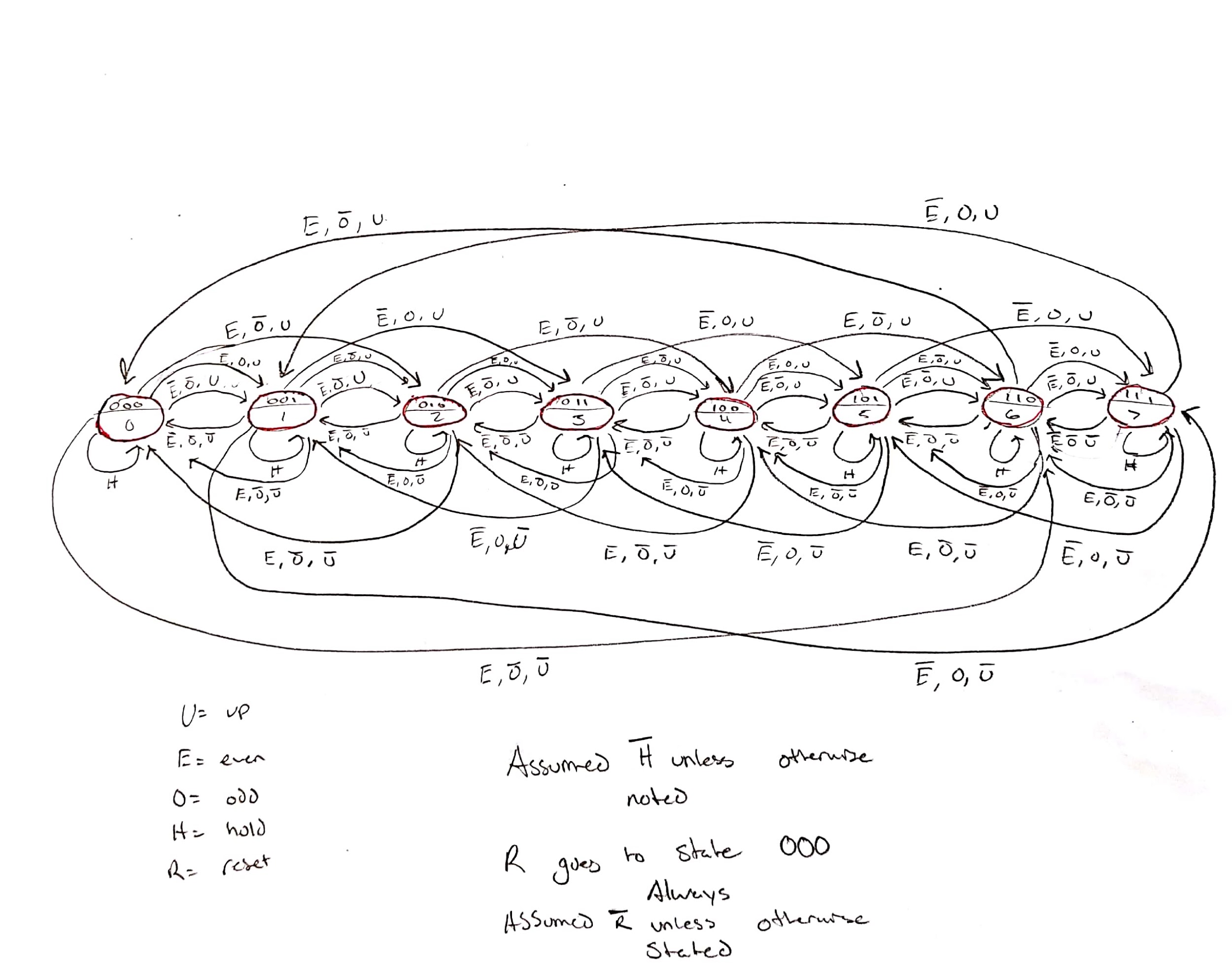
We designed a 3-bit up/down counter with an unsigned binary number. The counter used a mealy states, a seven segment decoder, and external inputs like buttons and switches. The mealy logic kept track of the current and transitioning state depends on the external inputs.



Upper Level Black Body Diagram



Lower Level Black Body Diagram

State diagram of our full-feature 3-bit up/down counter, with state transitions for counting up, down, even up, even down, odd up, odd down, and hold.

**Questions:**

1. From your state diagram, what would happen if both the ODD and EVEN input were simultaneously asserted?

* If both odd and even inputs where asserted our counter went to 0. This was due to how the code was structured in our project, in which it defaulted to 0.

2. From your HDL model, what would happen if any other input is asserted at the same time as the RESET input?

* It did not matter, if the reset button was pressed it would take precedence over any other input and revert to the 000 state.

3. Briefly describe the symmetry present in both the state diagram and HDL model of this FSM.

* All of the states are basically the same in this design. For example every state took the current state and outputted a new state depending on the external inputs. The only difference being what states each switched to.

4. Self-correcting hardware is a great feature in digital design. Briefly describe what the term “illegal state recovery” means and the specific condition that it attempts to avoid.

* Illegal state recovery is when a circuit correct itself after falling into an unused state. It does this by pointing unused states back to relevant states. This is all done to avoid the program from hanging.

5. Did the FSM model you used for the circuit in this lab activity use some form of illegal state recovery? Briefly explain.

* Our FSM used illegal state recovery when in a state not specified. For example, when the switch to make the counter count by odds was flipped, but an even number was currently displayed, it would change the state to a relevant state (an odd number). As well, when even and odd are asserted, 0 is always displayed.

6. Briefly state why it is that state diagrams never includes clock signals.

* It is implied that the clock is what causes the transitions.

7. If someone told you they encoded their FSM using eight flip-flops, would you know how many unique states were in their state diagram? Briefly explain.

* 2^8 would be the number of unique states that a person could have in their diagram. However, it is not possible to know how many of these states they use in their design.

8. Write a closed form formula that relates the maximum number of unique states in a sequential circuit to the number of 1-bit storage elements in the sequential circuit.

* f(x) = 2^x

9. Write a closed form formula that shows how many storage elements (n) would need to implement a circuit requiring Q unique states. This formula should include a floor or ceiling function.

* Ceiling function
* n = log2(Q)

**Design Problems:**

1. Design a counter that continuously counts in the following sequence: {2, 5, 7, 1, 4}.

* The counter has a hold input that prevents the counter from counting. The counter’s output shows the sequence as given when the hold input is not asserted (positive logic), or negative values of the sequence when the hold input is asserted. In other words, the hold input pauses the counter and causes the output to display a negative version (RC format) of the current count.
* Make the design self-correcting by sending any unused states to state associated with count 7. The counter’s output for unused states is zero.

Provide a high-level BBD, a PS/NS table, and a state diagram for this problem. Minimize your use of hardware.



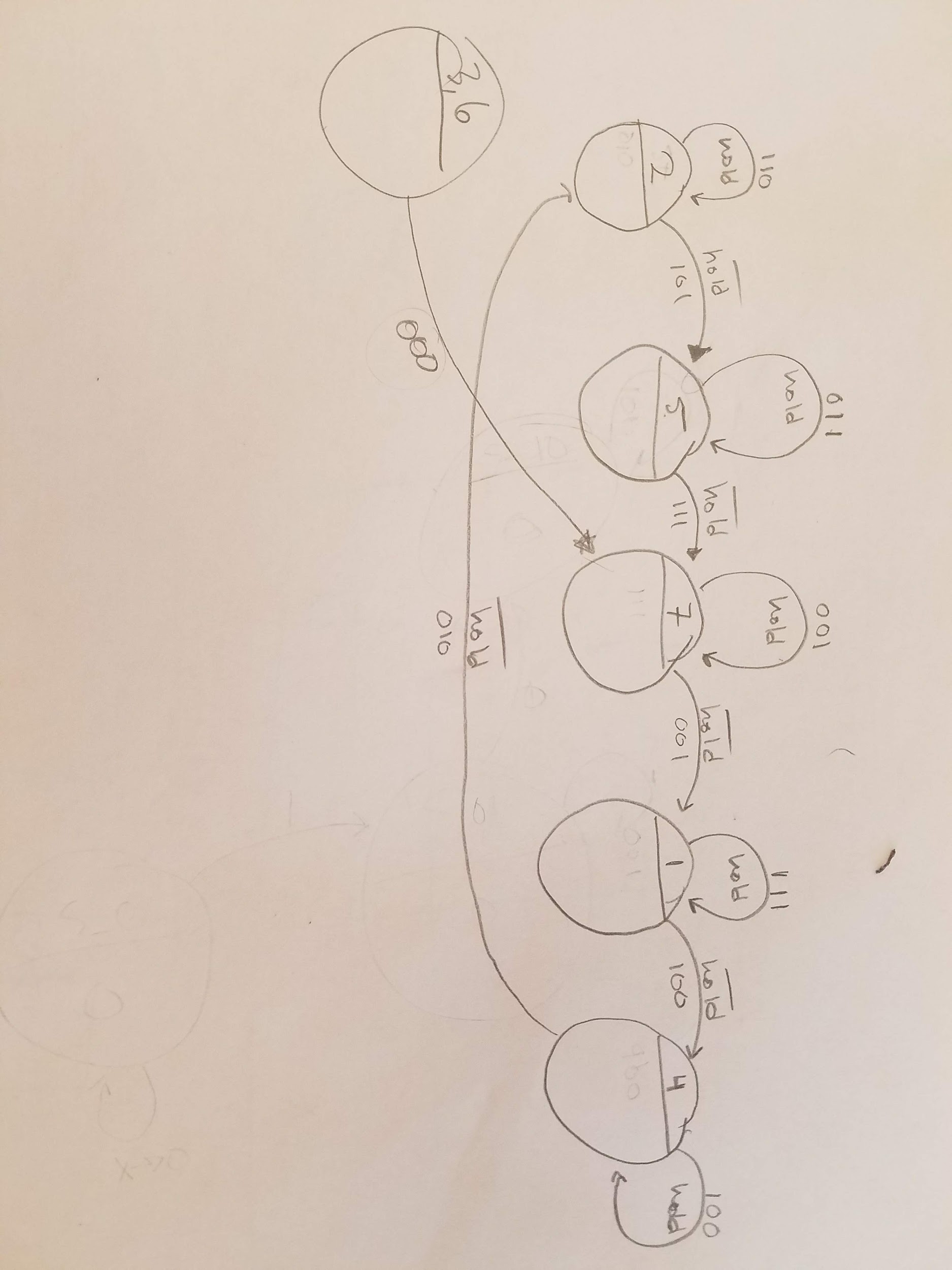
**Upper Level BBD**



**Lower Level BBD**

|  |  |  |
| --- | --- | --- |
| PS | H | NS |
| 000 | 0 | 111 |
| 000 | 1 | 111 |
| 001 | 0 | 100 |
| 001 | 1 | 111 |
| 010 | 0 | 011 |
| 010 | 1 | 110 |
| 011 | 0 | 000 |
| 011 | 1 | 000 |
| 100 | 0 | 010 |
| 100 | 1 | 100 |
| 101 | 0 | 111 |
| 101 | 1 | 011 |
| 110 | 0 | 000 |
| 110 | 1 | 000 |
| 111 | 0 | 001 |
| 111 | 1 | 001 |

**PS/NS Table**



**State Diagram**

**Source Code:**

`timescale 1ns / 1ps

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

// Company: Ratner Surf Designs

// Engineer: James Ratner

//

// Create Date: 07/07/2018 08:05:03 AM

// Design Name: Counter FSM

// Module Name: fsm\_template

// Project Name: Full-Feature 3-Bit Up/Down Counter

// Target Devices:

// Tool Versions:

// Description: FSM module for controlling all state transitions

// 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, up, odd, even, hold, clk, mealy, moore);

input reset\_n, up, odd, even, hold, clk;

output reg [2:0] mealy, moore;

//- next state & present state variables

reg [2:0] NS, PS;

//- bit-level state representations

parameter [2:0] st\_0=3'b000, st\_1=3'b001, st\_2=3'b010, st\_3=3'b011, st\_4=3'b100, st\_5=3'b101, st\_6=3'b110, st\_7=3'b111;

wire nclk;

clk\_divder\_nbit #(.n(25)) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

//- model the state registers

always @ (negedge reset\_n, posedge nclk)

if (reset\_n == 1)

PS <= st\_0;

else

PS <= NS;

//- model the next-state and output decoders

always @ (up, odd, even, hold,PS, reset\_n)

begin

mealy = 3'b000; moore = 3'b000; // assign all outputs

case(PS)

st\_0:

begin

moore = 3'b000;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b000;

NS = st\_1;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b000;

NS = st\_7;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b000;

NS = st\_2;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b001;

NS = st\_3;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b000;

NS = st\_6;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b001;

NS = st\_7;

end

end

else

begin

mealy = 3'b000;

NS = PS;

end

end

st\_1:

begin

moore = 3'b001;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b001;

NS = st\_2;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b001;

NS = st\_0;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b000;

NS = st\_2;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b001;

NS = st\_3;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b000;

NS = st\_6;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b001;

NS = st\_7;

end

end

else

begin

mealy = 3'b001;

NS = PS;

end

end

//verliog adds extra spaces when copy/pasting, causing wrapping of our code

st\_2:

begin

moore = 3'b010;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b010;

NS = st\_3;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b010;

NS = st\_1;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b010;

NS = st\_4;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b011;

NS = st\_5;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b010;

NS = st\_0;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b011;

NS = st\_1;

end

end

else

begin

mealy = 3'b010;

NS = PS;

end

end

st\_3:

begin

moore = 3'b011;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b011;

NS = st\_4;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b011;

NS = st\_2;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b010;

NS = st\_4;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b011;

NS = st\_5;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b010;

NS = st\_0;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b011;

NS = st\_1;

end

end

else

begin

mealy = 3'b011;

NS = PS;

end

end

st\_4:

begin

moore = 3'b100;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b100;

NS = st\_5;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b100;

NS = st\_3;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b100;

NS = st\_6;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_7;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b100;

NS = st\_2;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_3;

end

end

else

begin

mealy = 3'b100;

NS = PS;

end

end

st\_5:

begin

moore = 3'b101;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b101;

NS = st\_6;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b101;

NS = st\_4;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b100;

NS = st\_6;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_7;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b100;

NS = st\_2;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_3;

end

end

else

begin

mealy = 3'b101;

NS = PS;

end

end

st\_6:

begin

moore = 3'b110;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b110;

NS = st\_7;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b110;

NS = st\_5;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b110;

NS = st\_0;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_7;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b110;

NS = st\_4;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b101;

NS = st\_3;

end

end

else

begin

mealy = 3'b110;

NS = PS;

end

end

st\_7:

begin

moore = 3'b111;

if (hold == 0)

begin

if (up == 1 & even == 0 & odd==0)

begin

mealy = 3'b111;

NS = st\_0;

end

else if (up == 0 & even == 0 & odd==0)

begin

mealy = 3'b111;

NS = st\_6;

end

else if (up == 1 & even == 1 & odd==0)

begin

mealy = 3'b110;

NS = st\_0;

end

else if (up == 1 & even == 0 & odd==1)

begin

mealy = 3'b111;

NS = st\_1;

end

else if (up == 0 & even == 1 & odd==0)

begin

mealy = 3'b110;

NS = st\_4;

end

else if (up == 0 & even == 0 & odd==1)

begin

mealy = 3'b111;

NS = st\_5;

end

end

else

begin

mealy = 3'b111;

NS = PS;

end

end

default: NS = st\_0;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company: DogsWithJobs

// Engineer: Hegglin/Skelly

//

// Create Date: 11/07/2018 08:05:03 AM

// Design Name: Counter FSM

// Module Name: mainLogic

// Project Name: Full-Feature 3-Bit Up/Down Counter

// Target Devices:

// Tool Versions:

// Description: Main source code for full-feature counter

//

// Dependencies:

//

// Revision:

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

// Additional Comments:

//

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

module mainLogic(reset, up, odd, even, hold, clk, seg, an);

input up, odd, even, reset, hold, clk;

output [7:0] seg;

output [3:0] an;

wire [2:0] mealy, moore;

fsm\_template myFSM(

.reset\_n (reset),

.up (up),

.odd (odd),

.even (even),

.hold (hold),

.clk (clk),

.mealy (mealy),

.moore (moore)

);

univ\_sseg my\_univ\_sseg (

.cnt1 (mealy),

.cnt2 (0),

.valid (1),

.dp\_en (0),

.dp\_sel (0),

.mod\_sel (0),

.sign (0),

.clk (clk),

.ssegs (seg),

.disp\_en (an)

);

endmodule