Lab 9 Submission

**4-Bit Up/Down Counter w/ Multiplexed 7-Segment Display**

CPE 133 - 03

Michael Hegglin

Jonathan Skelly

**Executive Summary:**

We designed a 4-bit up/down counter with an unsigned binary number. The counter used a clock, a register, two logic modules, and a seven segment decoder. The register held the data, while the logic modules helped display the LED and segment outputs, and decide which number was happening next.

BBD of our 4-bit LED display counter.

Next-level down diagram of our 4-bit LED display counter.

**Questions:**

1. Briefly describe the FSM output-type classification of the stoneage unary output you used in the previous experiment and then in this experiment.

* In the past experiment our FSM had a moore output. This is due to the fact that there was no external input being used to control the output. This experiment had a mealy output. This was due to the fact that the button was an external controlling input for the output of the logic box.

2. Briefly describe the FSM output-type classification of the decimal output you used in this experiment.

* The FSM output-type classification for this experiment was a moore output. This is due to the fact that no external inputs controlled it’s output.

3. Briefly describe the particular attribute of a digital circuit (not of an HDL model) that gives the circuit the ability to store data.

* A circuit can store data by making its input the output. This is called feedback, and is usually controlled by a register.

4. Briefly describe the relation, if any, between a sequential circuit in digital design and a sequential statement in Verilog?

* A sequential statement in verilog executes its steps in sequential order within the block. Likewise, a sequential circuit outputs a set of values based on the sequence of inputs. A sequential statement can be used to create a sequential circuit.

5. Briefly describe what the term “sequential” refers to in the term “sequential circuit”.

* Sequential circuits are circuits whose outputs depend on the sequence of inputs. This basically means that they have memory.

6. Briefly describe why an AND gate is not functionally complete.

* It’s not functionally complete because it is not possible to create an INVERT gate out of the AND gate.

7. Explicitly show how you can obtain an OR function from an AND gate. Show both the derivation and the resulting circuit element.

AND gate ⇒ F=A\*B

OR gate ⇒ F = ~(A\*B) ⇒ F = ~A + ~B



8. Explicitly show how you can obtain an AND function from an OR gates. Show both the derivation and the resulting circuit element.

OR gate ⇒ F=A + B

AND gate ⇒ F = ~(A + B) ⇒ F = ~A \* ~B



9. An EXNOR gate is often referred to as an “equivalence gate”. Briefly explain how it would get such a name.

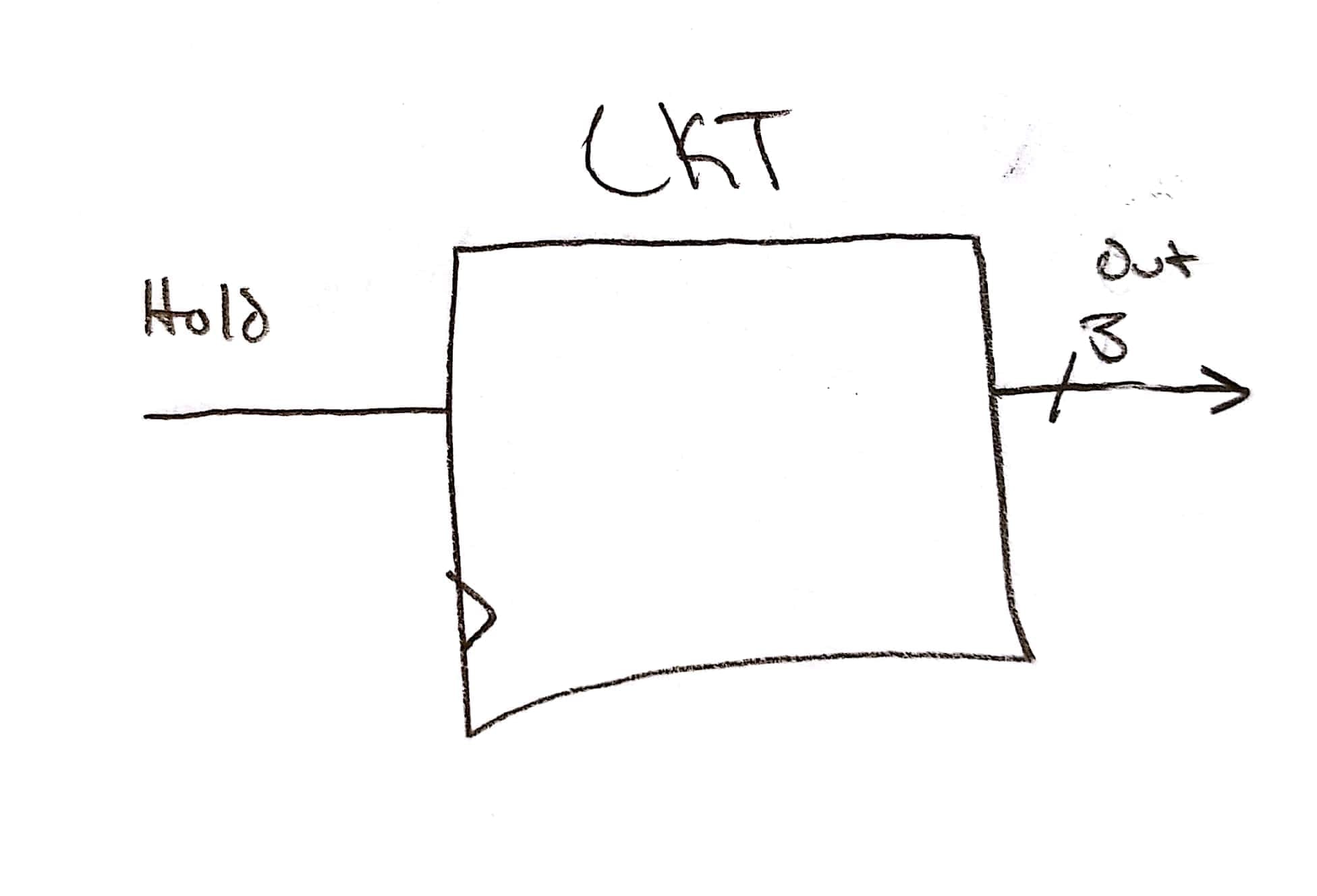
* The output of an EXNOR gate is only ever a 1 if both the inputs are the same, or “equivalent”. This is why it is called an “equivalence gate”.

10. Briefly explain what characteristic of a circuit makes it a “mixed logic” circuit?

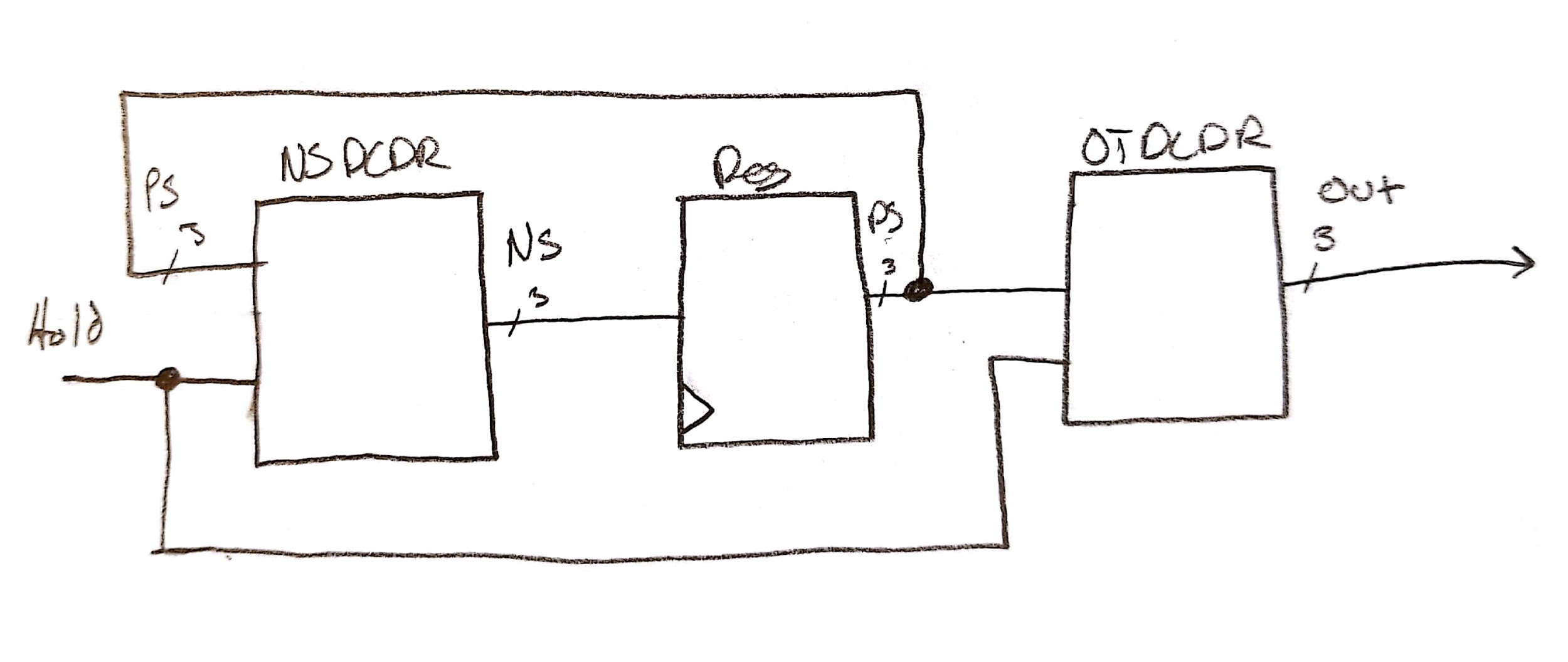
* Mixed logic is using both 0 and 1 when in an active state. In other words, it uses both positive and negative logic.

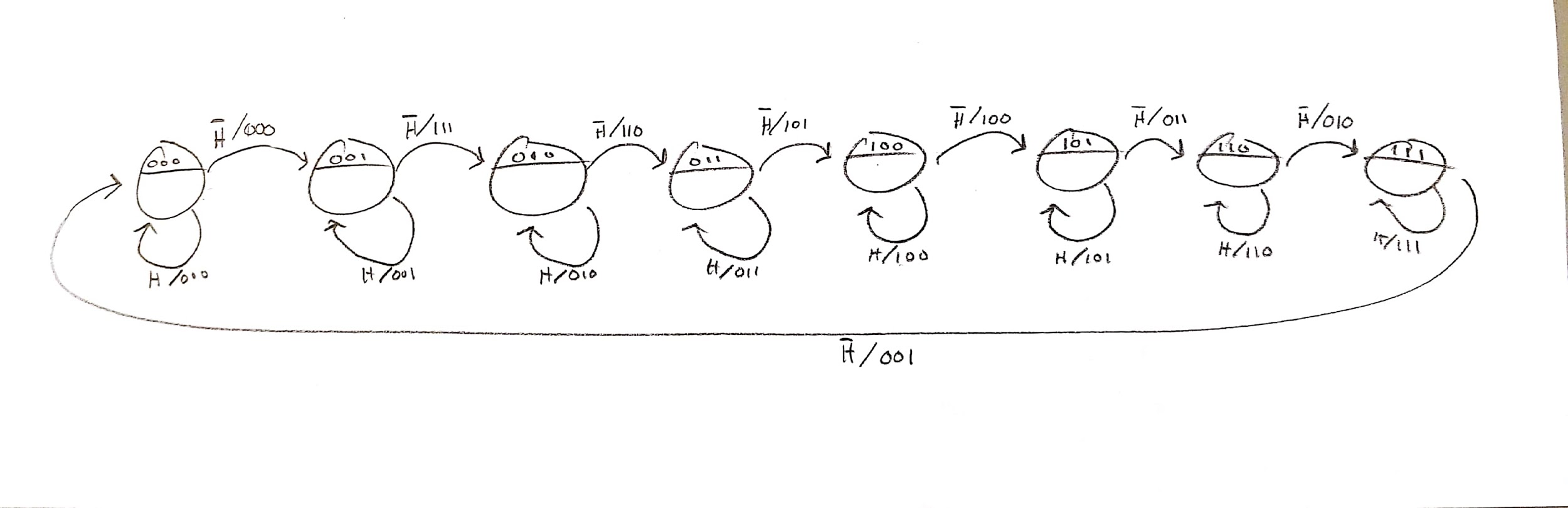
**Design Problems:**

Design a 3-bit synchronous up counter. Include a HOLD input that when asserted, prevents the circuit from counting. This output of this counter should be a 3-bit value in binary [0-7] format when the HOLD is asserted; otherwise a compliment of that 3-bit value when the HOLD is not asserted. This counter should contain all three basic FSM modules. Be sure to include a state diagram with your solution. Include a state diagram with your solution. State how the circuit is controlled.



BBD of 3-bit value/compliment counter.

Next-level down of 3-bit value/compliment counter. Externally controlled via the “Hold” input.



State diagram of 3-bit value/compliment counter.

**Source Code:**

`timescale 1ns / 1ps

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

// Company: DogsWithJobs

// Engineer: Skelly/Hegglin

//

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

// Design Name: 4-bit LED Display Counter

// Module Name: MainLogic

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

// Display

// Target Devices:

// Tool Versions:

// Description: 4-bit counter that displays decimal value on 7-seg

// display and stoneage unary value on LEDs. Counts backward on button

// press

//

// Dependencies:

//

// Revision:

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

// Additional Comments:

//

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

module MainLogic(clk, button, led, seg, an);

input clk, button;

wire[3:0] run\_around, run\_out;

wire nclk;

output [14:0] led;

output [7:0] seg;

output [3:0] an;

clk\_divder\_nbit #(26) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

initialDecoder D1(

.run\_out(run\_out),

.button(button),

.run\_around(run\_around)

);

reg\_nb #(4) MY\_REG (

.data\_in (run\_out),

.ld (1),

.clk (nclk),

.clr (0),

.data\_out (run\_around)

);

mealyMoore myMM (

.reg\_output (run\_around),

.led\_nums (led),

.button (button)

);

univ\_sseg my\_univ\_sseg (

.num (run\_around),

.clk (clk),

.seg (seg),

.an (an)

);

endmodule

module mealyMoore(reg\_output, led\_nums, button);

input [3:0] reg\_output;

input button;

output reg [14:0] led\_nums;

always @ (reg\_output, button)

begin

if(button == 0)

begin

case (reg\_output)

0: led\_nums = 15'b000000000000000;

1: led\_nums = 15'b000000000000001;

2: led\_nums = 15'b000000000000011;

3: led\_nums = 15'b000000000000111;

4: led\_nums = 15'b000000000001111;

5: led\_nums = 15'b000000000011111;

6: led\_nums = 15'b000000000111111;

7: led\_nums = 15'b000000001111111;

8: led\_nums = 15'b000000011111111;

9: led\_nums = 15'b000000111111111;

10: led\_nums = 15'b000001111111111;

11: led\_nums = 15'b000011111111111;

12: led\_nums = 15'b000111111111111;

13: led\_nums = 15'b001111111111111;

14: led\_nums = 15'b011111111111111;

15: led\_nums = 15'b111111111111111;

endcase

end

else

begin

case (reg\_output)

0: led\_nums = 15'b111111111111111;

1: led\_nums = 15'b111111111111110;

2: led\_nums = 15'b111111111111100;

3: led\_nums = 15'b111111111111000;

4: led\_nums = 15'b111111111110000;

5: led\_nums = 15'b111111111100000;

6: led\_nums = 15'b111111111000000;

7: led\_nums = 15'b111111110000000;

8: led\_nums = 15'b111111100000000;

9: led\_nums = 15'b111111000000000;

10: led\_nums = 15'b111110000000000;

11: led\_nums = 15'b111100000000000;

12: led\_nums = 15'b111000000000000;

13: led\_nums = 15'b110000000000000;

14: led\_nums = 15'b100000000000000;

15: led\_nums = 15'b000000000000000;

endcase

end

end

endmodule

module initialDecoder(run\_out, button, run\_around);

input button;

input [3:0] run\_around;

output reg [3:0] run\_out;

always @ (run\_around, button)

begin

if(button == 0)

begin

case (run\_around)

0: run\_out = 4'b0001;

1: run\_out = 4'b0010;

2: run\_out = 4'b0011;

3: run\_out = 4'b0100;

4: run\_out = 4'b0101;

5: run\_out = 4'b0110;

6: run\_out = 4'b0111;

7: run\_out = 4'b1000;

8: run\_out = 4'b1001;

9: run\_out = 4'b1010;

10: run\_out = 4'b1011;

11: run\_out = 4'b1100;

12: run\_out = 4'b1101;

13: run\_out = 4'b1110;

14: run\_out = 4'b1111;

15: run\_out = 4'b0000;

default: run\_out = 4'b0000;

endcase

end

else

begin

case (run\_around)

2: run\_out = 4'b0001;

3: run\_out = 4'b0010;

4: run\_out = 4'b0011;

5: run\_out = 4'b0100;

6: run\_out = 4'b0101;

7: run\_out = 4'b0110;

8: run\_out = 4'b0111;

9: run\_out = 4'b1000;

10: run\_out = 4'b1001;

11: run\_out = 4'b1010;

12: run\_out = 4'b1011;

13: run\_out = 4'b1100;

14: run\_out = 4'b1101;

15: run\_out = 4'b1110;

0: run\_out = 4'b1111;

1: run\_out = 4'b0000;

default: run\_out = 4'b0000;

endcase

end

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 univ\_sseg(num, clk, seg, an );

input [4:0] num;

input clk;

wire nclk;

reg [7 :0] tens, ones;

output reg [7:0] seg;

output reg [3:0] an;

reg [1:0] m\_cnt;

clk\_divder\_nbit #(10) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

always @(num, an, seg)

begin

//an = 4'b0111;

case (num)

0: ones = 8'b00000011;

1: ones = 8'b10011111;

2: ones = 8'b00100101;

3: ones = 8'b00001101;

4: ones = 8'b10011001;

5: ones = 8'b01001001;

6: ones = 8'b01000001;

7: ones = 8'b00011111;

8: ones = 8'b00000001;

9: ones = 8'b00011001;

10: ones = 8'b00000011;

11: ones = 8'b10011111;

12: ones = 8'b00100101;

13: ones = 8'b00001101;

14: ones = 8'b10011001;

15: ones = 8'b01001001;

default: ones = 8'b11111111;

endcase

end

always @(num, an, seg)

begin

//an = 4'b0111;

case (num)

0: tens = 8'b11111111;

1: tens = 8'b11111111;

2: tens = 8'b11111111;

3: tens = 8'b11111111;

4: tens = 8'b11111111;

5: tens = 8'b11111111;

6: tens = 8'b11111111;

7: tens = 8'b11111111;

8: tens = 8'b11111111;

9: tens = 8'b11111111;

10: tens = 8'b10011111;

11: tens = 8'b10011111;

12: tens = 8'b10011111;

13: tens = 8'b10011111;

14: tens = 8'b10011111;

15: tens = 8'b10011111;

default: tens = 8'b11111111;

endcase

end

always @ (m\_cnt)

begin

case (m\_cnt)

0: an = 4'b1101;

1: an = 4'b1111;

2: an = 4'b1110;

3: an = 4'b1111;

default an = 0;

endcase

end

always @ (m\_cnt)

begin

case (m\_cnt)

0: seg = ones;

1: seg = 16'b1111111111111111;

2: seg = tens;

3: seg = 16'b1111111111111111;

default an = 0;

endcase

end

//- counter that drives the mulitplexed display

always @ (posedge nclk)

begin

m\_cnt <= m\_cnt + 1'b1;

end

endmodule