Lab 11 Submission

**Sequence Detector FSMs**

CPE 133 - 03

Michael Hegglin

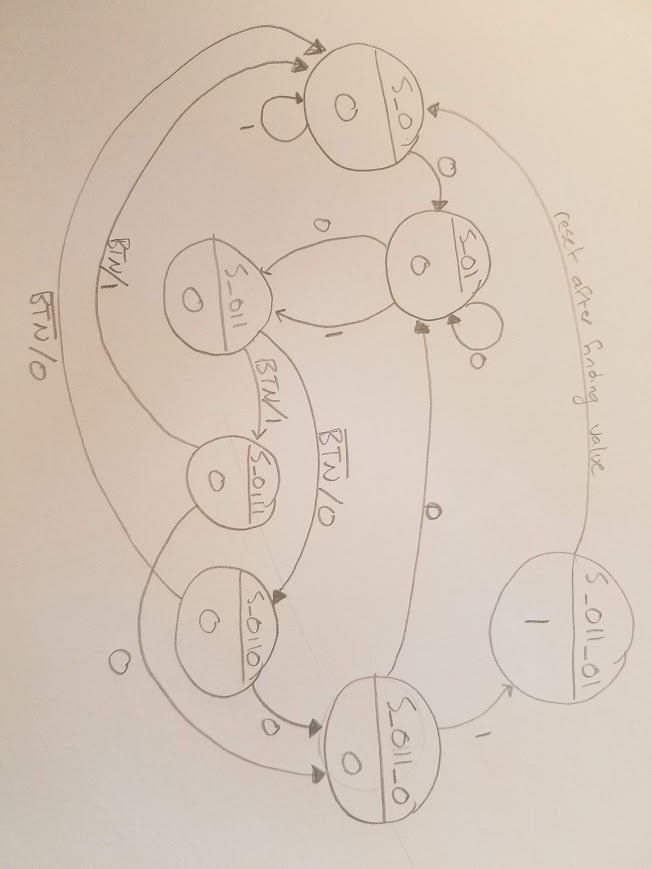
Jonathan Skelly

**Executive Summary:**

We designed an 8-bit sequence detector with an unsigned binary number. The detector used moore states, a seven segment decoder, and external inputs like buttons and switches. The moore logic kept track of the current state depends on the current bit input.



**High Level BBD**



**State Diagram**

**Questions:**

1. Describe an application where sequence detectors could potentially be useful.

* Sequence detectors are potentially useful in the timing of something. For example, a process is waiting for a certain sequence before it is executed.

2. Given two FSMs that perform the same basic function, a Mealy-type model typically has fewer states than Moore-type models. Briefly describe why that is generally the case.

* A mealy type model typically has less states than a moore model due to the fact that it is signal based off an external input. This means arrows can be used for this transition. In a moore model, a new state must be made for this transition, resulting in more states.

3. Mealy-type FSM are able to “react” more quickly than Moore-type FSMs. Briefly describe what characteristic of a Mealy-type FSM makes this the case.

* Mealy models are able to react faster due to the fact that they rely on an external input, and can immediately change states depending on the input.

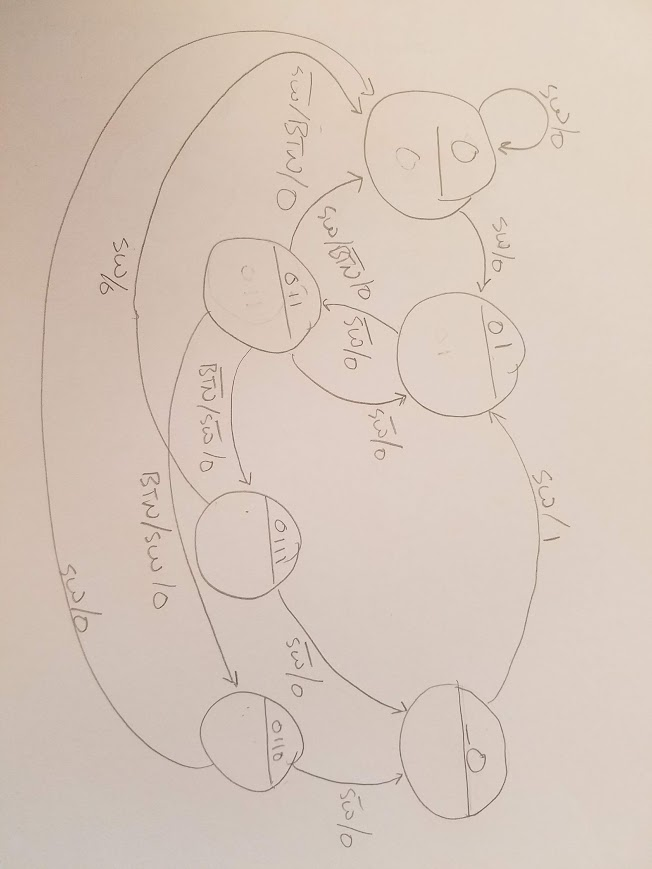
4. In the previous question, briefly describe what exactly the term “react” refers to?

* The term “react” refers to the changing of states.

5. Slower clock speeds of any circuit are generally considered better because they save power. Briefly describe the main factor in deciding the minimum clock speed for your FSM and still have it do the job you need it to do.

* The worst timing condition of the circuit will lead directly to the maximum clock frequency.

6. Provide a Mealy-type state diagram for the resetting version of the FSM you designed in this lab activity.



7. Show one approach (draw a diagram) to change an XOR gate into an inverter. Include a truth table that clearly shows the foundation of this approach.

* If one of the inputs in the XOR gate is a one, the output will be inverted.

|  |  |  |
| --- | --- | --- |
| 1 used to invert | input | Out |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

8. Briefly describe why an XOR gate is not functionally complete.

* An XOR gate is not functionally complete because a person can’t make all boolean expressions with it.

9. Show two approaches (draw a diagram) of how you can change a NAND gate into an inverter.

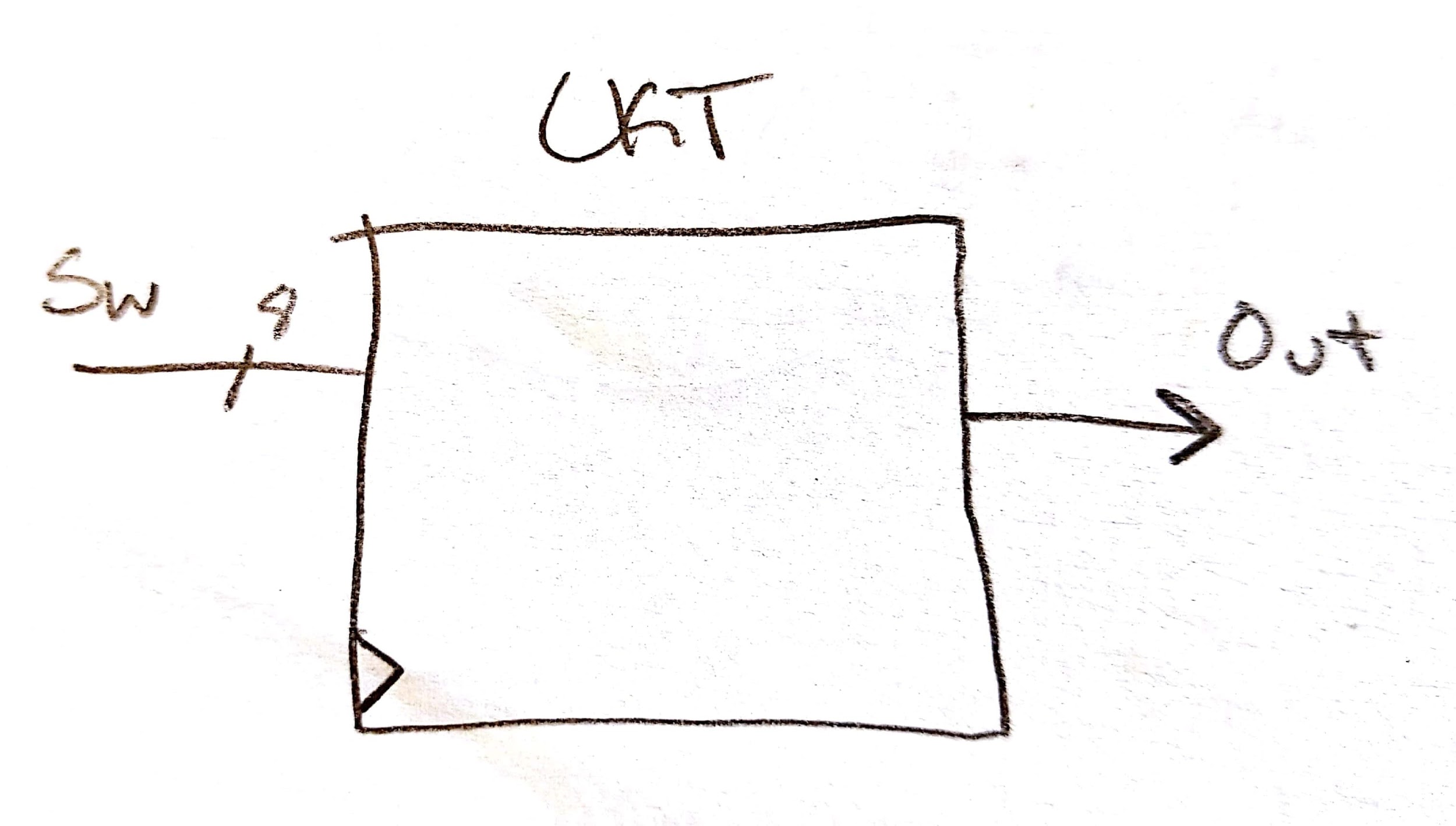


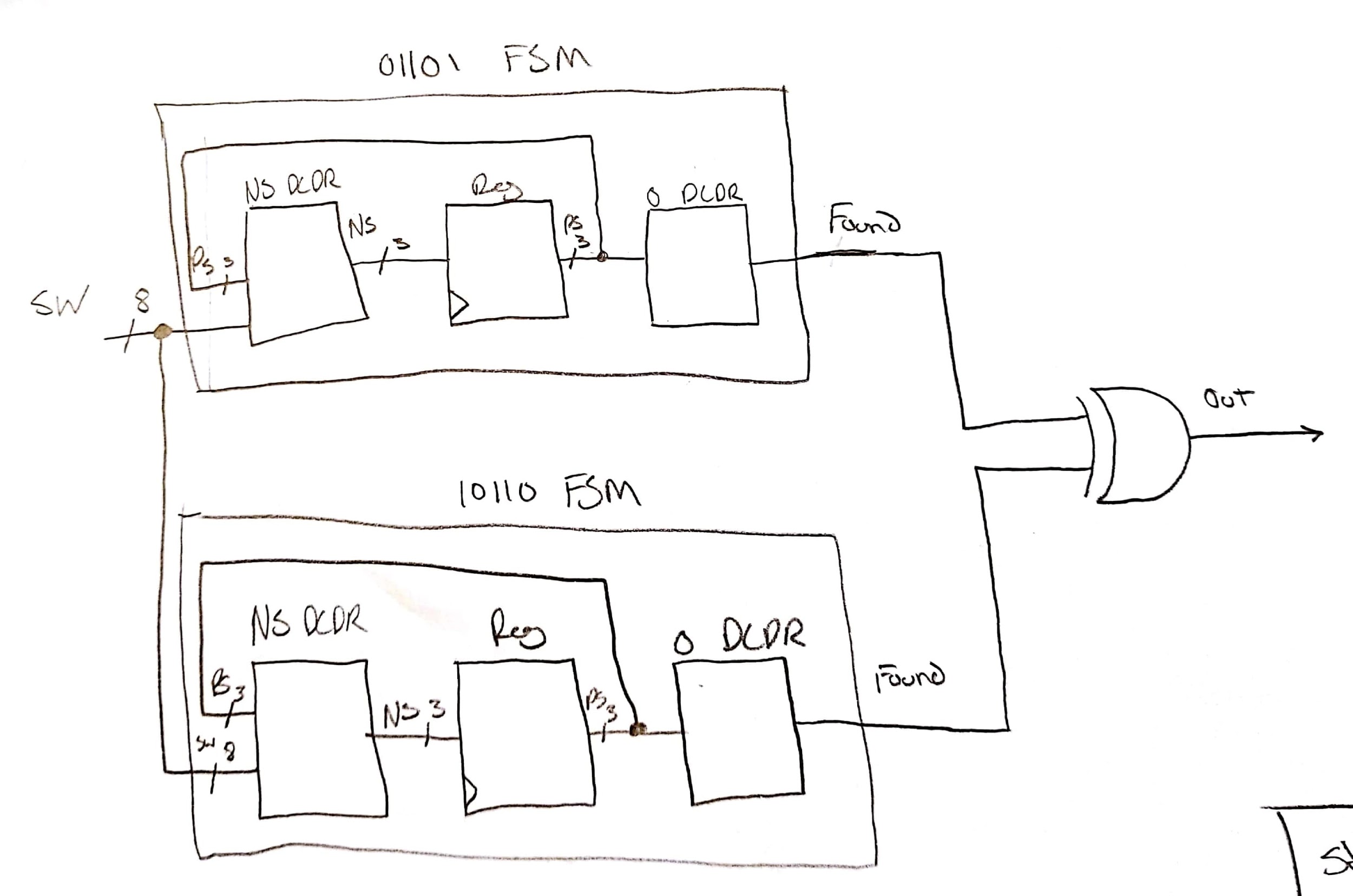
10. Show two approaches (draw a diagram) of how you can change a NOR gate into an inverter.

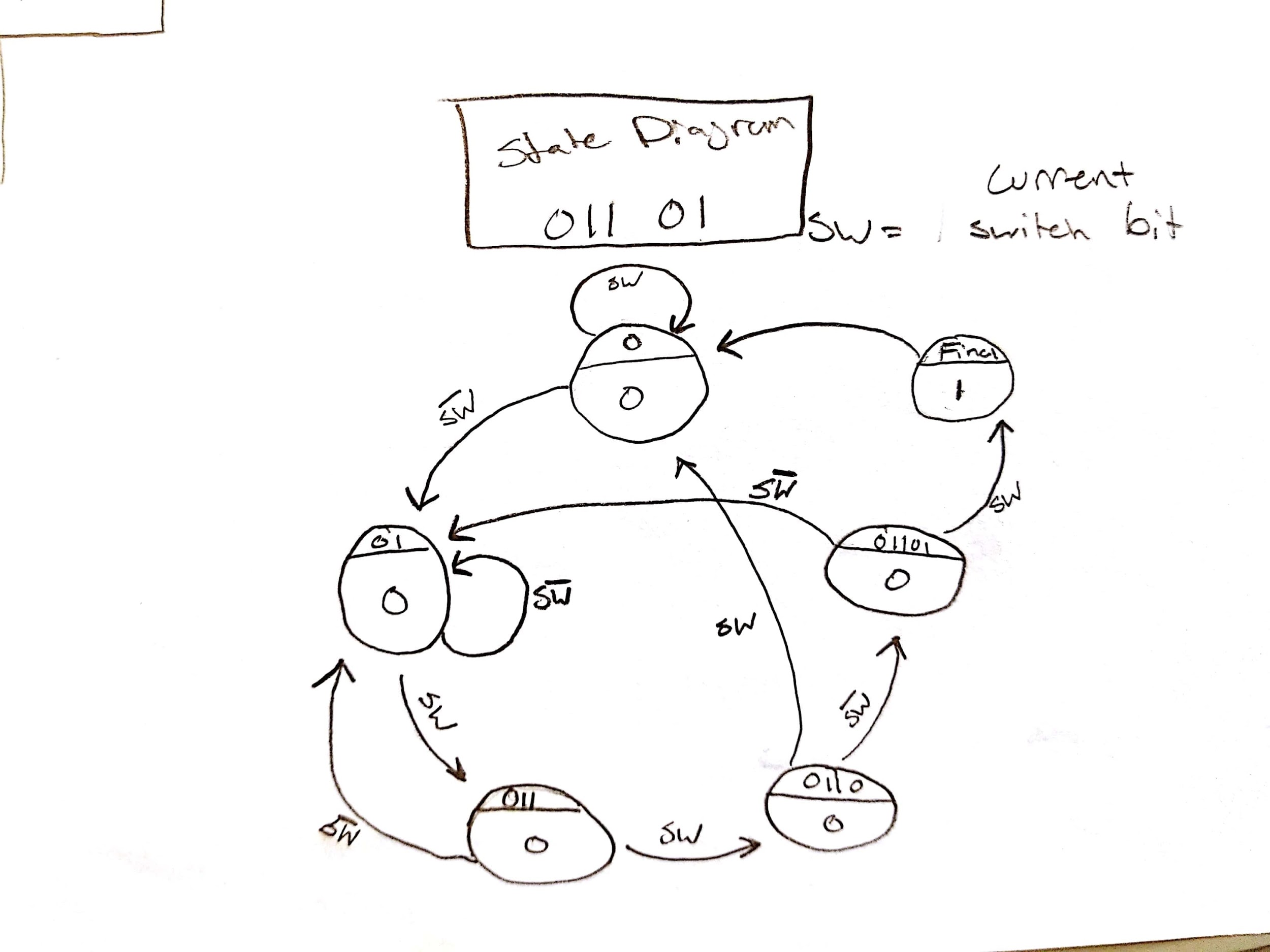


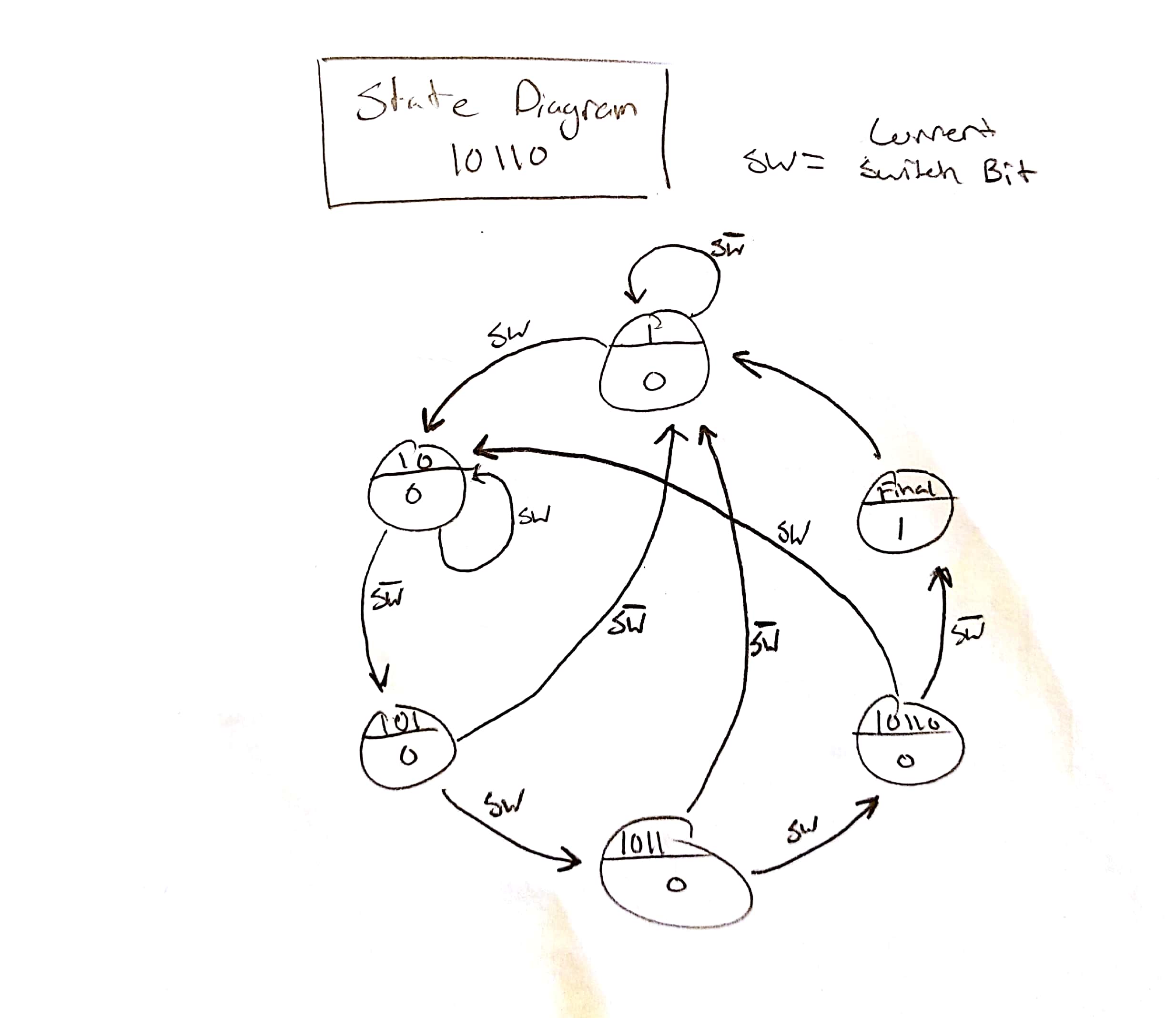
**Design Problems:**

Design a circuit that indicates when it finds one and only one of the two following sequences “01101” and “10110” on a single serial input. The most straight-forward approach is to use two FSMs in your; be sure to provided state diagrams for any FSM you use in your design. Assume the serial input does not change more than once per clock cycle. Minimize your use of hardware in your design. State how the circuit is controlled.

BBD of 01101/10110 detector.

Next-level down of 01101/10110 detector.

State diagram of 01101 FSM.

State diagram of 10110 FSM.

**Source Code:**

`timescale 1ns / 1ps

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

// Company: DogsWithJobs

// Engineer: Hegglin/Skelly

//

// Create Date: 11/26/2018 08:05:03 AM

// Design Name: Detector FSM

// Module Name: fsm\_template

// Project Name: Sequence Detector FSMs

// Target Devices:

// Tool Versions:

// Description: FSM for detecting given sequences.

//

// Dependencies:

//

// Revision:

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

// Additional Comments:

//

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

module fsm\_template(reset\_n, sw, clk, mealy, moore, btn);

input reset\_n, clk, btn;

input sw;

output reg 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\_f=3'b110, st\_7=3'b111;

//- model the state registers

always @ (negedge reset\_n, posedge clk)

if (reset\_n == 0)

PS <= st\_0;

else

PS <= NS;

//- model the next-state and output decoders

always @ (sw,PS,btn)

begin

mealy = 0; moore = 0; // assign all outputs

case(PS)

st\_0:

begin

moore = 0;

if (sw == 0)

begin

NS = st\_1;

end

else

begin

NS = st\_0;

end

end

st\_1:

begin

moore = 0;

if (sw == 1)

begin

NS = st\_2;

end

else

begin

NS = st\_1;

end

end

st\_2:

begin

moore = 0;

if (sw == 1)

begin

NS = st\_3;

end

else

begin

NS = st\_1;

end

end

st\_3:

begin

moore = 0;

if (sw == 1 && btn == 1)

begin

NS = st\_4;

end

else if (sw == 0 && btn == 0)

begin

NS = st\_4;

end

else if (sw == 0 && btn == 1)

begin

NS = st\_1;

end

else if (sw == 1 && btn ==0)

begin

NS = st\_0;

end

end

st\_4:

begin

moore = 0;

if (sw == 0)

begin

NS = st\_5;

end

else

begin

NS = st\_0;

end

end

st\_5:

begin

moore = 0;

if (sw == 1)

begin

NS = st\_f;

end

else

begin

NS = st\_1;

end

end

st\_f:

begin

moore = 1;

NS = st\_0;

end

default: NS = st\_0;

endcase

end

endmodule

//clk divider

module clk\_divder\_nbit(clockin, clockout);

input clockin;

output wire clockout;

parameter n = 13;

reg [n:0] count;

always@(posedge clockin)

begin

count <= count + 1;

end

assign clockout = count[n];

endmodule