*COS 116*

*The Computational Universe*

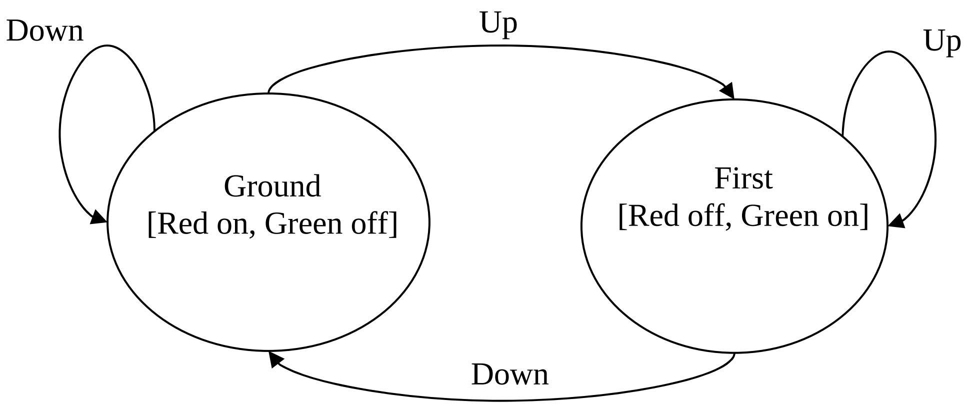
# ***How To Design A Finite State Machine***

Here is an example of a designing a finite state machine, worked out from start to finish.

<u>Step 1: Describe the machine in words.</u>

In this example, we'll be designing a controller for an elevator. The elevator can be at one of two floors: Ground or First. There is one button that controls the elevator, and it has two values: Up or Down. Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First. At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way.

<u>Step 2: Draw the FSM diagram</u>



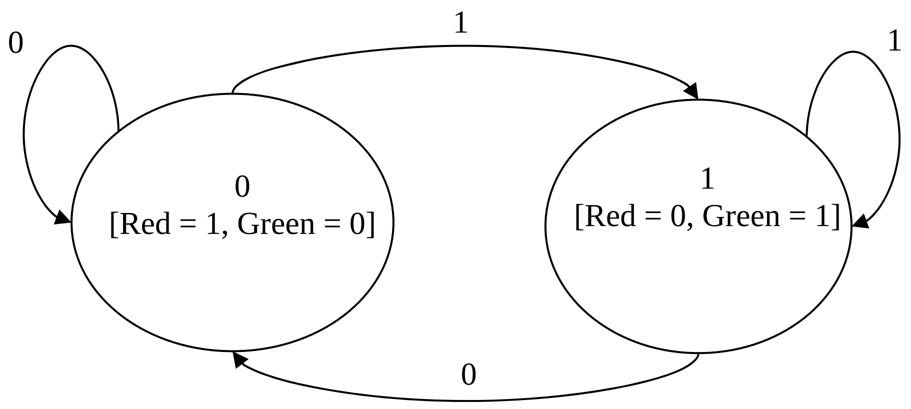
In this diagram, the bubbles represent the states, and the arrows represent state transitions. The arrow labels indicate the input value corresponding to the transition. For instance, when the elevator is in the Ground state, and the input is Up, the next state is First. The information in the brackets indicates the output values for the lights in each state.

<u>Step 3: Select numbers to represent states and values</u>

Before converting the above FSM diagram to a circuit, we need to represent every value in our example as a binary number. Here is some convenient numbers to use.

| | | |
|---|---|---|
| Ground = 0 | Down = 0 | Off = 0 |
| First = 1 | Up = 1 | On = 1 |

So here's the FSM diagram with the words replaced by numbers:



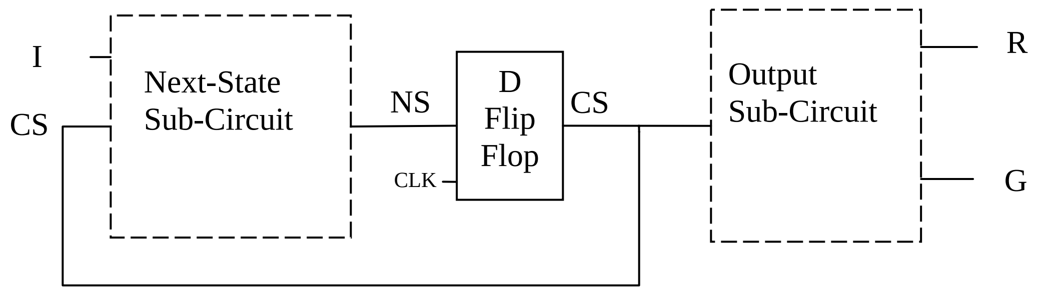
<u>Step 4: Write the truth table</u>

From the FSM diagram, it's easy to read off the correct truth table.

| CurrentState | Input | NextState | Red | Green |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

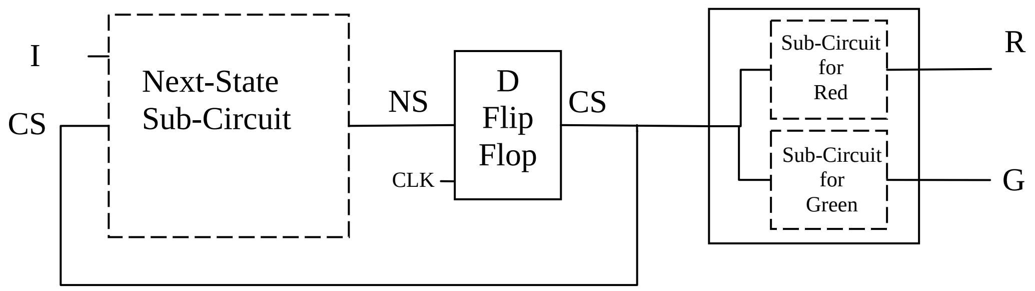
<u>Step 5: Draw a "big picture" view of the circuit</u>

Here is the finite-state machine circuit, with many details missing. The variable names have been abbreviated. The dashed boxes indicate the parts (let's call them "sub-circuits") that we still need to design.



All FSM circuits will have a form similar to this. Our example has two states, and so we need only one D flip-flop. An FSM with more states would need more flip-flops. Our example has one input (labeled "I" in the figure), but in general there may be many inputs, or none at all. Also, an FSM may not have any outputs, in which case the "Output Sub-Circuit" would be omitted.

In our example, the Output Sub-Circuit has two outputs, R and G. To make things simpler, let's break this into two further sub-circuits: a sub-circuit that computes R, and another sub-circuit that computes G. This is shown below.



After making this change, every dashed box (i.e. every sub-circuit that we still need to design) has exactly one output. This is the easiest form to work with.

Step 6: Find Boolean expressions

For each sub-circuit that we need to design, we'll write a Boolean expression that expresses its output as a function of its inputs. We derive these expressions from the truth table we wrote in Step 4. There is a very general method for doing this, which we'll illustrate on the Next-State Sub-Circuit.

The Next-State Sub-Circuit has two inputs, I and CS, and one output, NS. From the truth table, we see that NS is 1 in exactly two cases:

1. CS = 0 and I = 1
2. CS = 1 and I = 1

So we simply write a Boolean expression that covers all and only these cases:

NS = ((*not* CS) *and* I) *or* (CS *and* I)

Notice that each case is represented by an AND subexpression, and the whole expression is the OR of these subexpressions. This technique can be used on a truth table of any size.

Of course, this Boolean expression can be simplified. To do this, we need some rules of Boolean logic. Here are the most useful ones:

1. A *or* 1 = 1

2. A *or* 0 = A
3. A *and* 1 = A
4. A *and* 0 = 0
5. (*not* A) or A = 1
6. (*not* A) *and* A = 0
7. *not* (*not* A) = A
8. (A *or* B) *and* C = (A *and* C) *or* (B *and* C)

Now we can simplify our Boolean expression:

NS = ((*not* CS) *and* I) *or* (CS *and* I)
= ((*not* CS) *or* CS) *and* I [using rule 8]
= 1 *and* I [using rule 5]
= I [using rule 3]

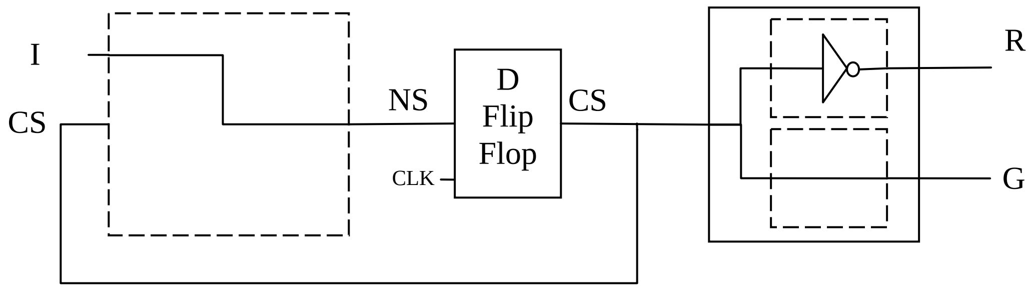
And so NS = I. Of course, for this simple example, this could have been easily seen just by inspecting the truth table.

Similarly, we find the Boolean expressions for the other sub-circuits are:

R = *not* CS
G = CS

<u>Step 7: Draw the rest of the circuit</u>

The only thing left to do is to draw the sub-circuits represented by our Boolean expressions.



Naturally, a more complicated example will require more gates, but the same methods will apply.