CPE 462
VHDL: Simulation and Synthesis

Topic #07 - a) Finite State Machines

What is it?

- A finite-state machine (FSM) is a mathematical model used to design computer programs and digital logic circuits.
- It is an abstract machine that can be in one of a finite number of states.
- The machine is in only one state at a time; the state it is in at any given time is called the current state.
- It can change from one state to another when initiated by a triggering event or condition, this is called a transition.

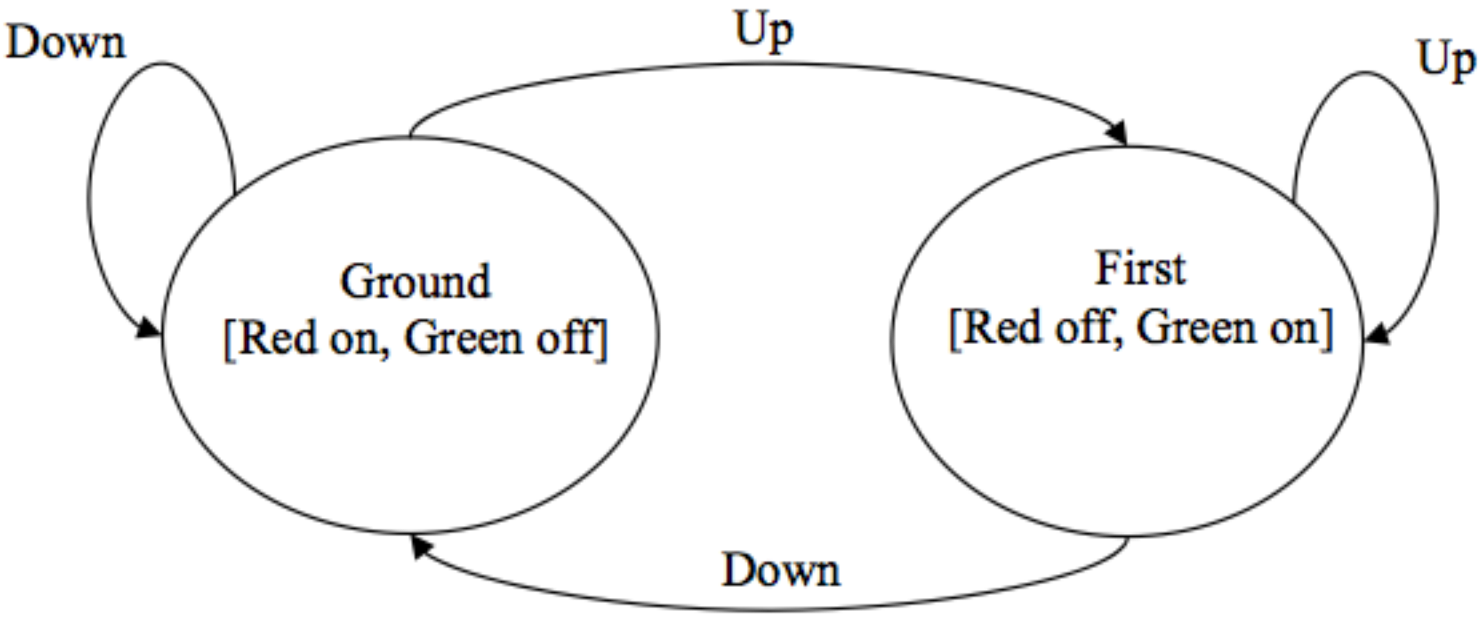
Lets design a FSM

Step 1: Describe the machine in words.

- We want to design the control 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.
- 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.

FSM diagram

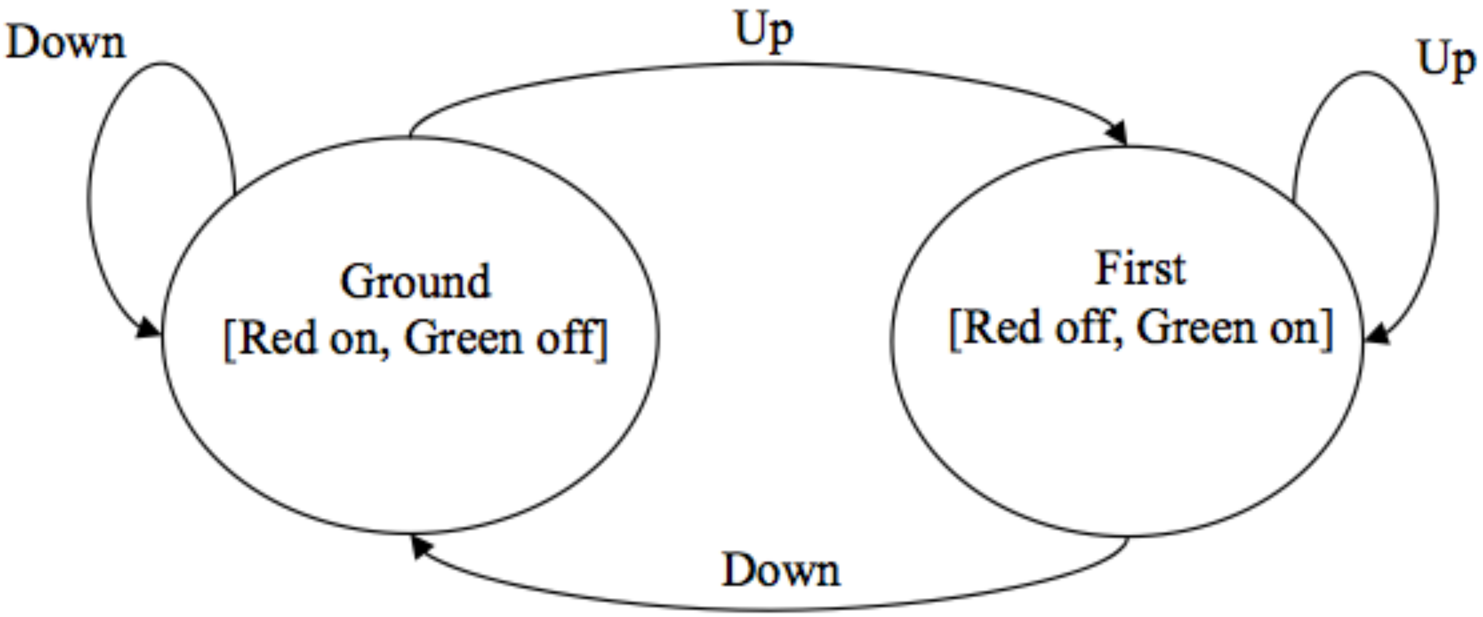
Step 2: Draw the FSM diagram



- Circles represent the states
- Arrows represent state transitions
- The arrow labels indicate the input value corresponding to the transition

FSM diagram

Step 2: Draw the FSM diagram



- Example: 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.

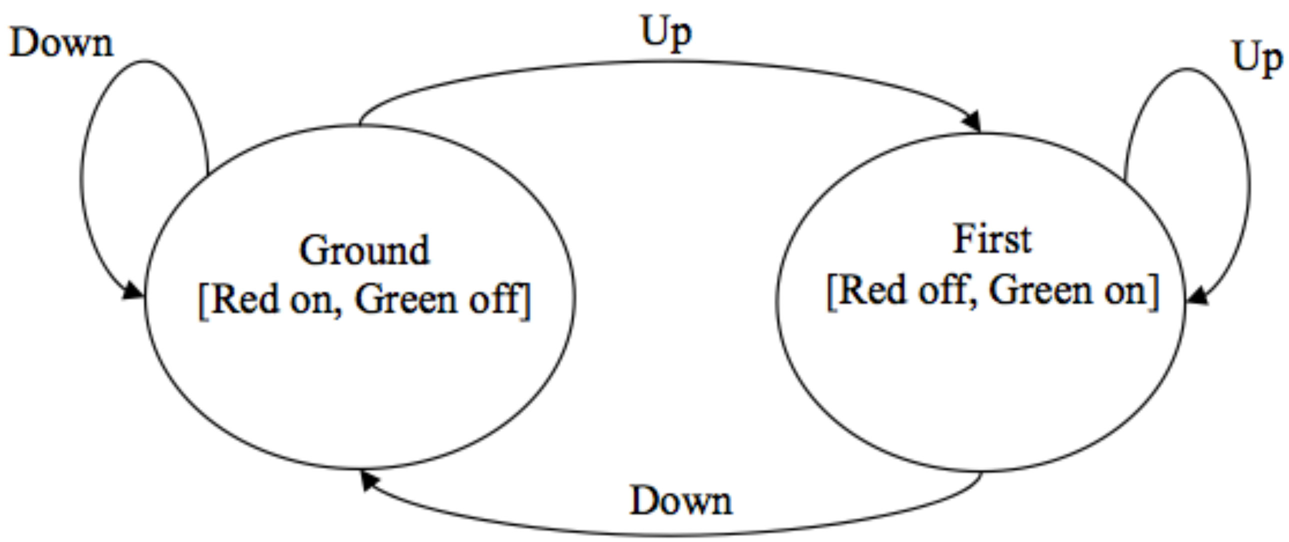
Representing states and values

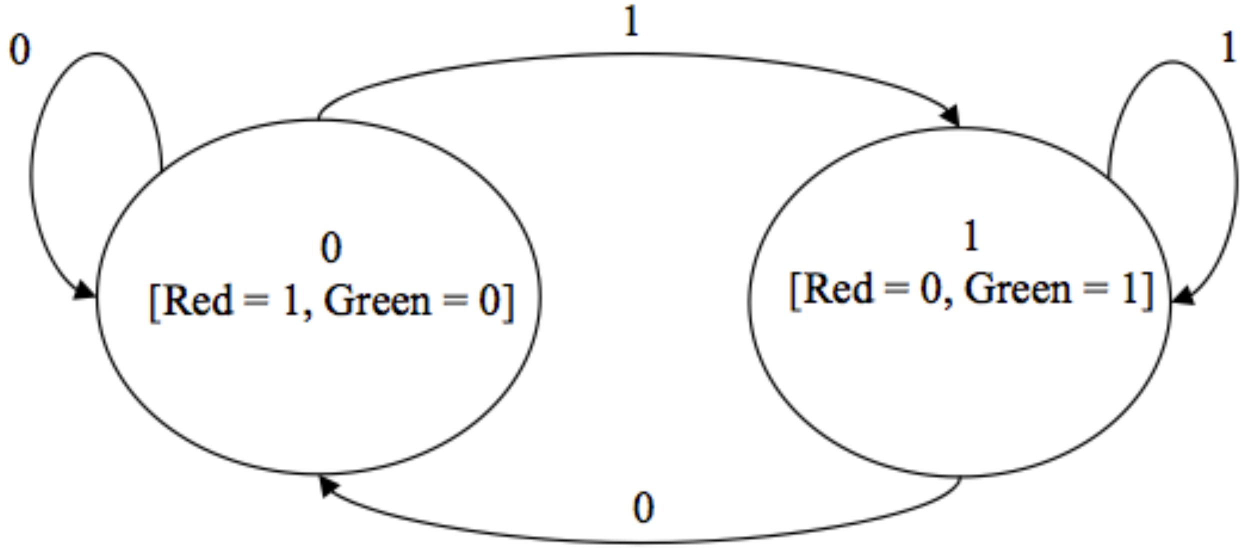
Step 3: Select binary numbers to represent states and values

State Identifier: Ground = 0 First = 1

Transition: Down=0, Up = 1

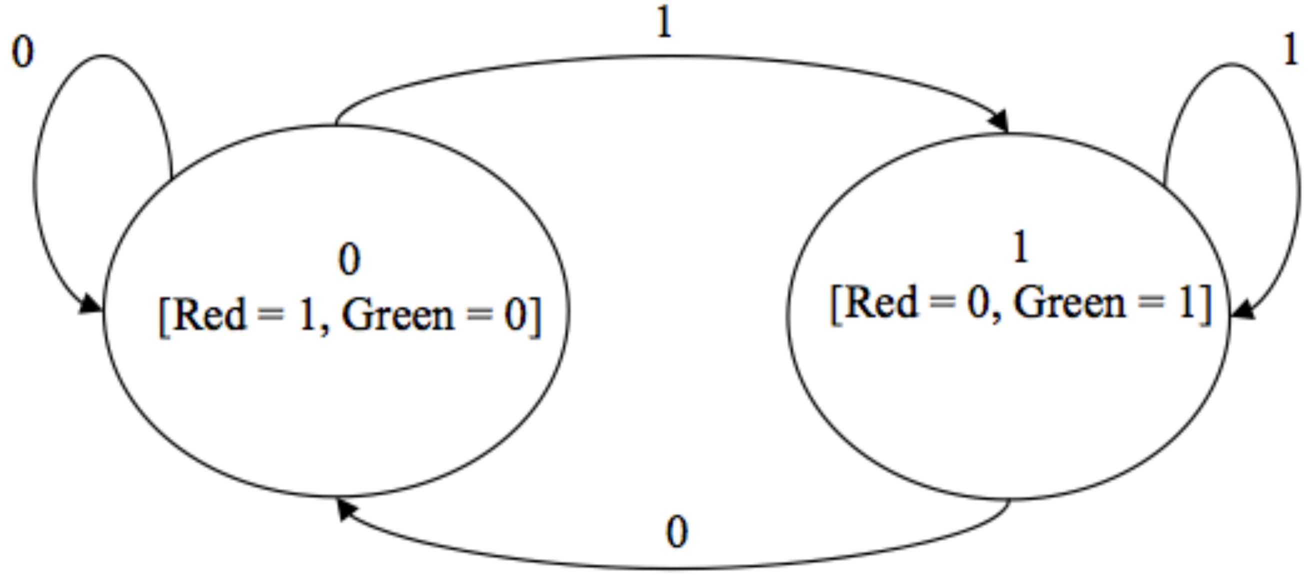
Outputs: Red / Green = 1 or 0

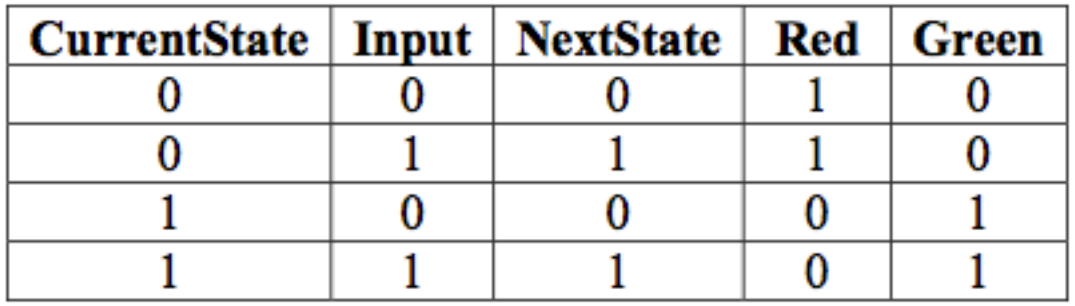




Write truth table

Step 4: Write truth table



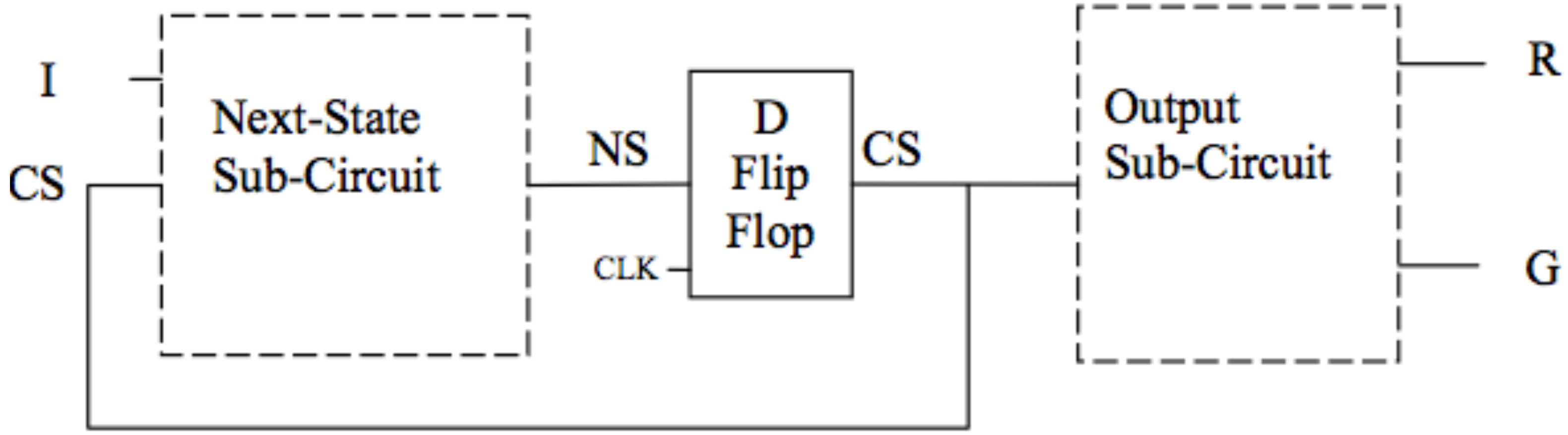


CurrentState	Input	NextState	Red	Green
0	0	0	1	0
0	1	1	1	0
1	0	0	0	1
1	1	1	0	1

“Big picture” view

Step 5: Draw a “big picture” view of the circuit

CS	I	NS	R	G
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

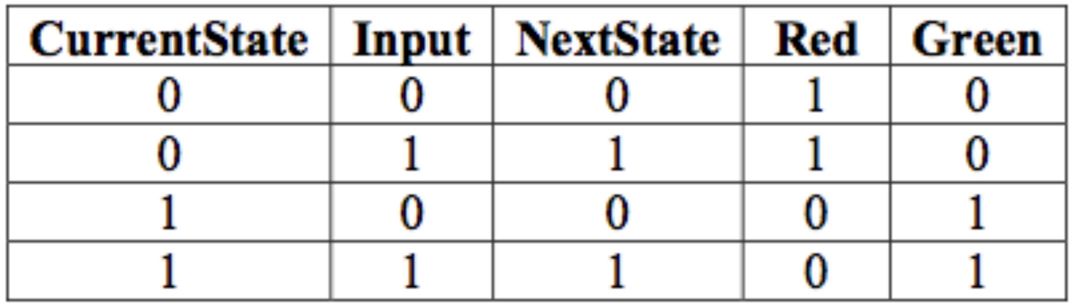


- The dashed boxes indicate the parts that we still need to design.
- All FSM circuits will have a form similar to this. Our example has two states, and so we need only one D flip-flop.
- We normally represent the states in binary, so a FSM with 6 states requires 3 FF (6 in binary is 101).
- An FSM with more states would need more flip-flops.

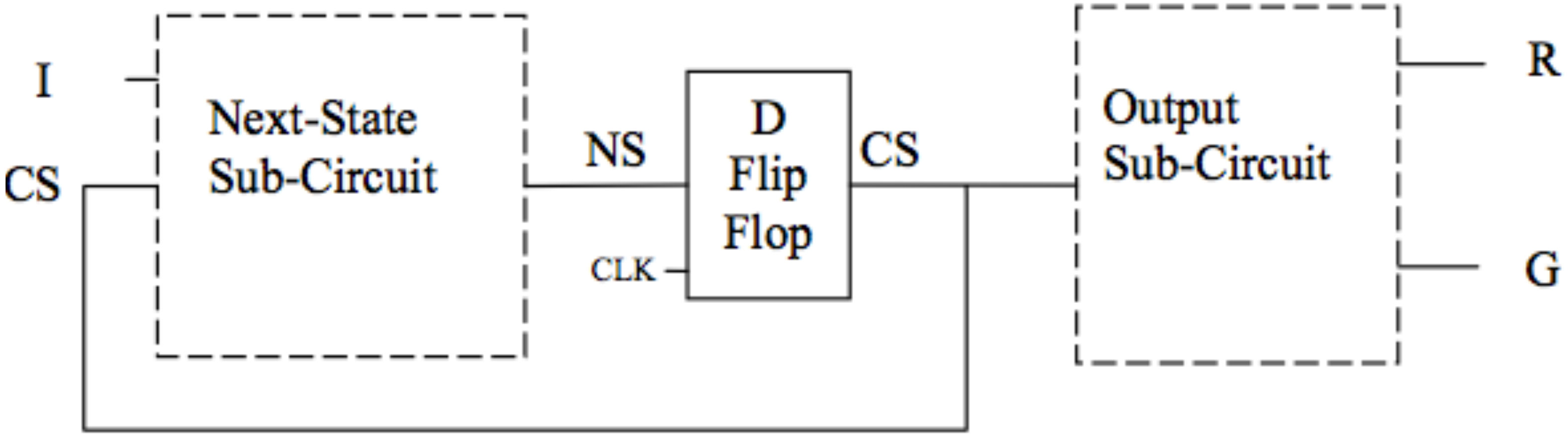
“Big picture” view

Step 5: Draw a “big picture” view of the circuit

CS I NS R G



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



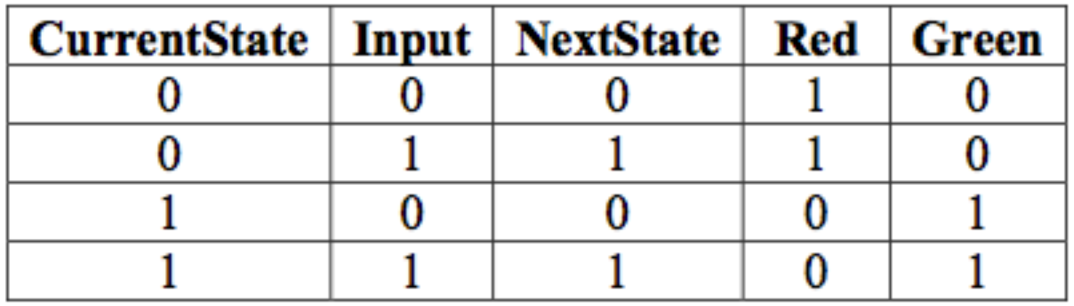
- This example has one input (labeled “I” in the figure), but in general there may be many inputs, or none at all.

- An FSM may not have any outputs, in which case the “Output Sub-Circuit” would be omitted.

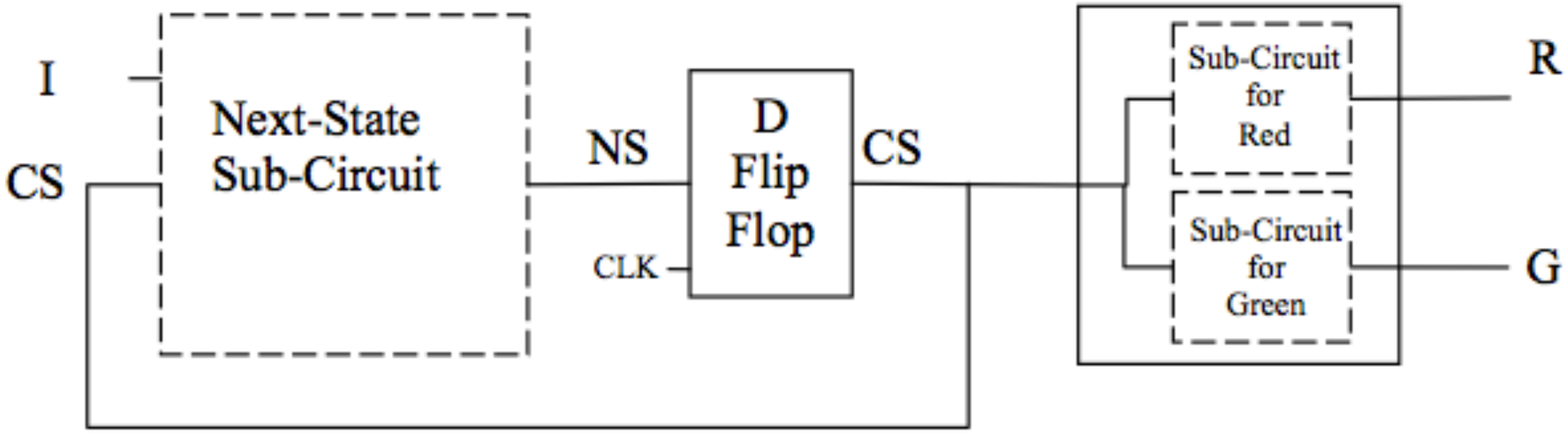
“Big picture” view

Step 5: Draw a “big picture” view of the circuit

CS I NS R G



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

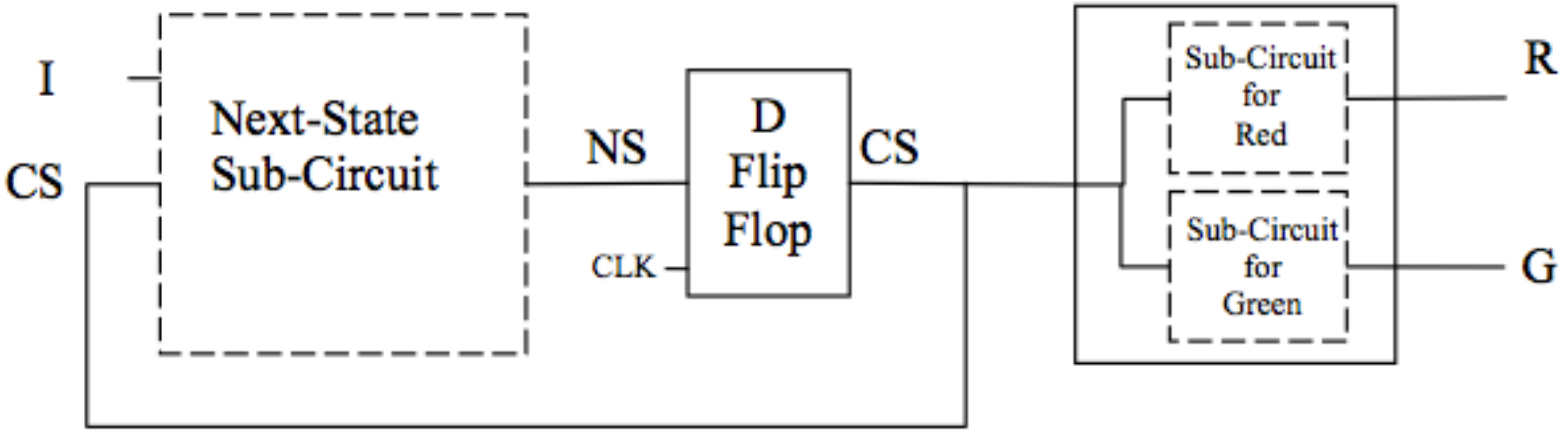


- In our example, the Output Sub-Circuit has two outputs, R and G.
- To make things simpler, let’s break this into two further sub-circuits: a sub-circuit that computes R, and another sub-circuit that computes G.

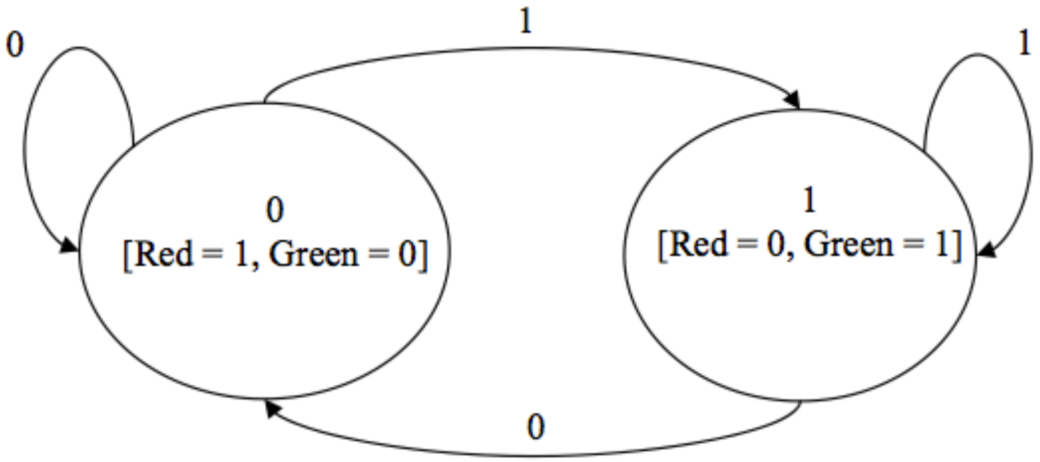
“Big picture” view

Step 5: Draw a “big picture” view of the circuit

CS	I	NS	R	G
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



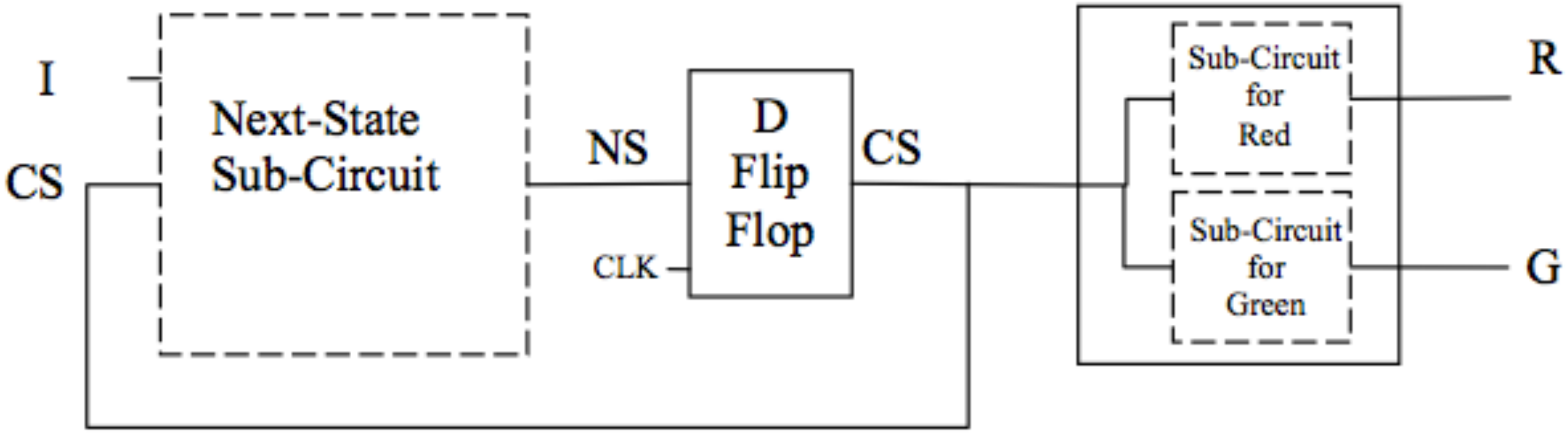
- NextState becomes CurrentState after the flip-flop.



Find Boolean expressions

Step 6: Find Boolean expressions

CS	I	NS	R	G
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



- 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.

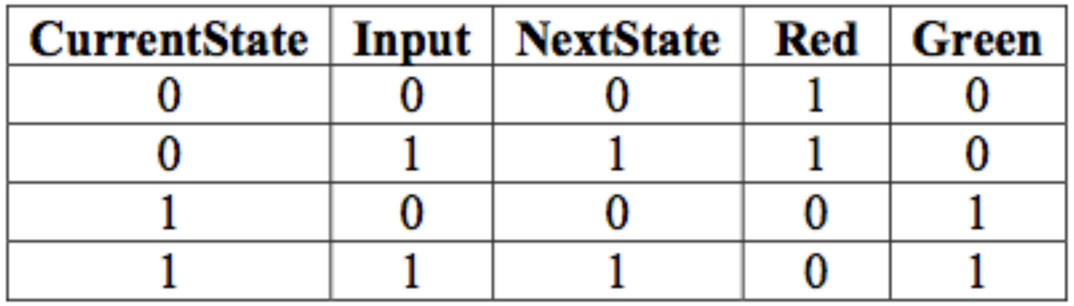
- We need to relate the **NextState** with **Input** and **CurrentState**.

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

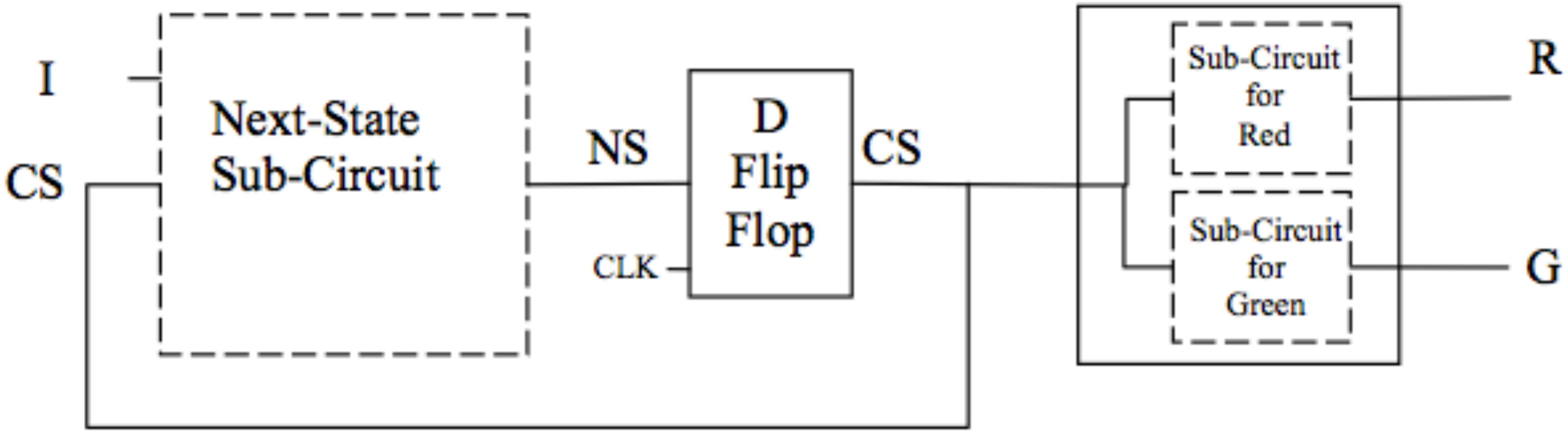
Find Boolean expressions

Step 6: Find Boolean expressions

CS I NS R G



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



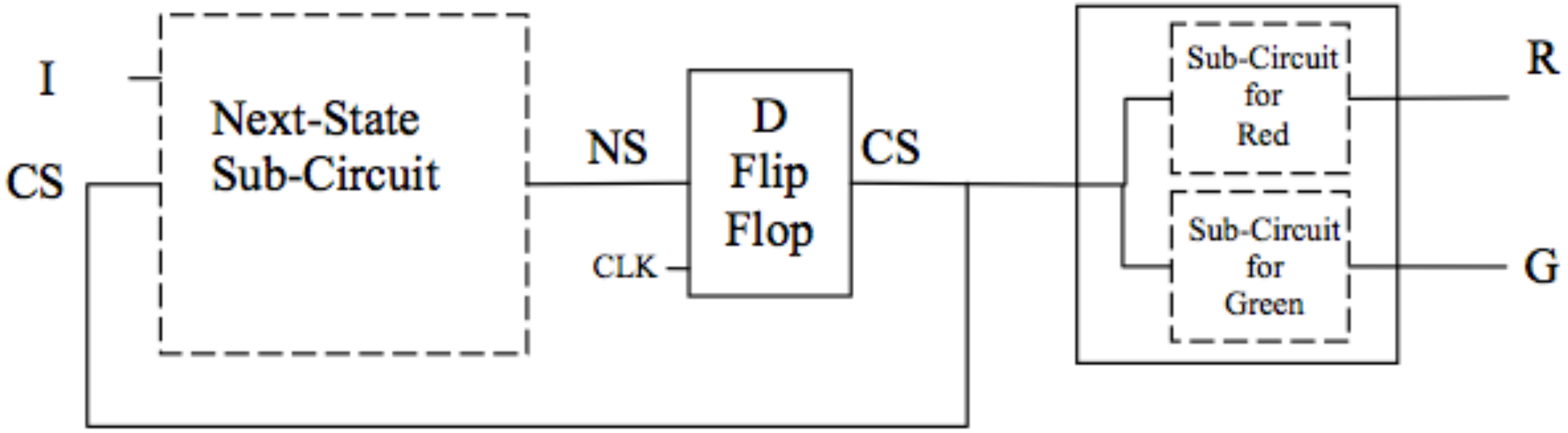
- Can we simplify **NS = ((not CS) and I) or (CS and I)** ?
  - Yes... **NS = I**

- The Boolean expressions for the other sub-circuits are:
  - R = not CS
  - G = CS

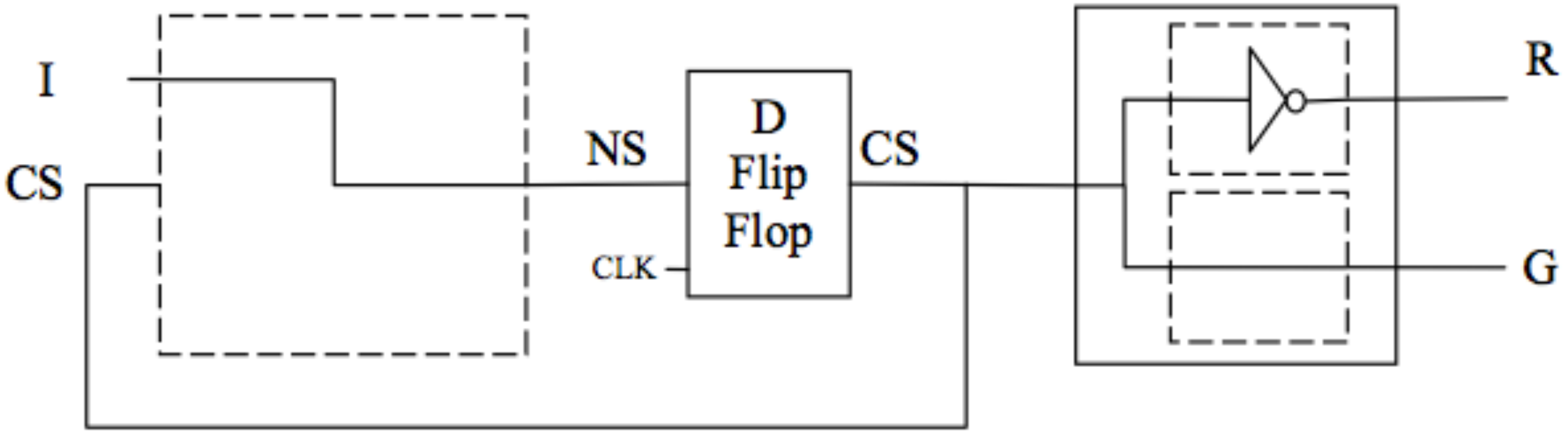
Draw the rest of the circuit

Step 7: Draw the entire circuit

CS	I	NS	R	G
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



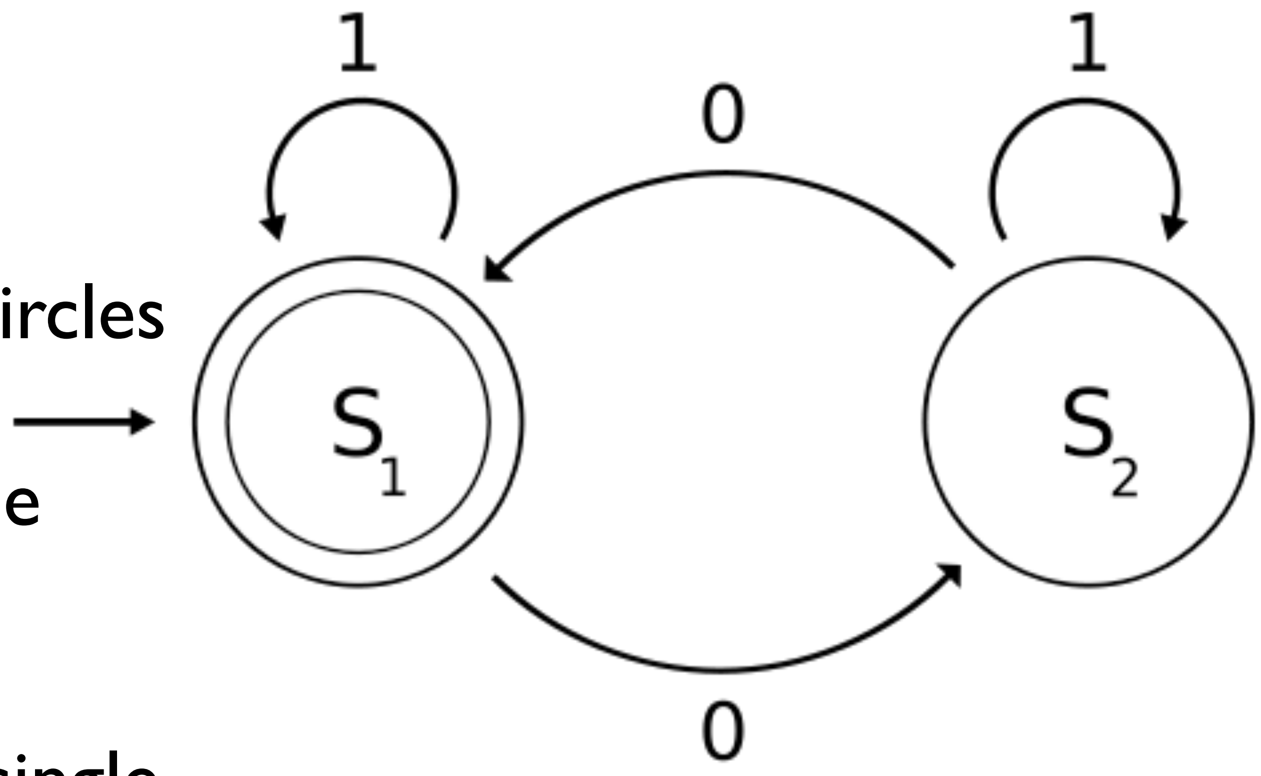
Becomes...



Moore machine

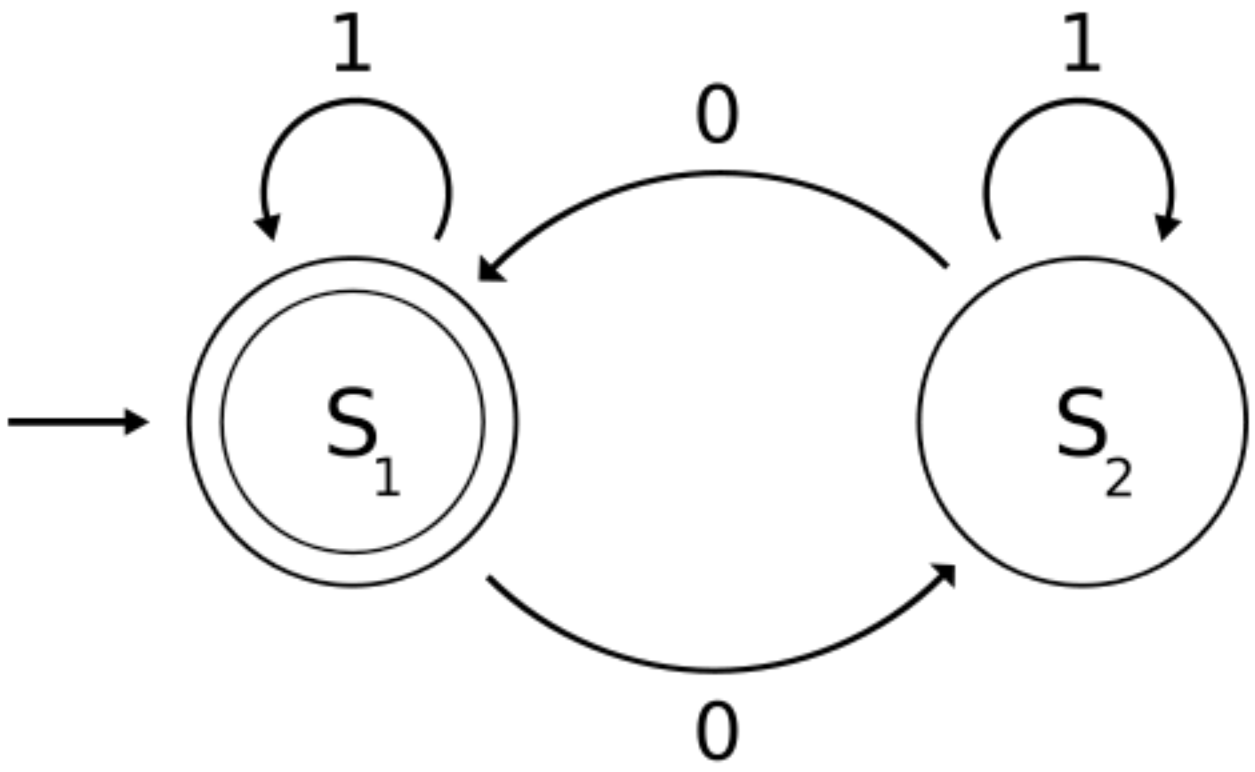
Another representation: Moore machine

- A Moore machine is a FSM. It takes some input and it generates a **single** output.
- Arrow points to the starting state
- The state names are usually inside the circles
- The transition conditions are next to the transition arrows
- The sub-circle indicates the final state (single output)



Moore machine state diagram

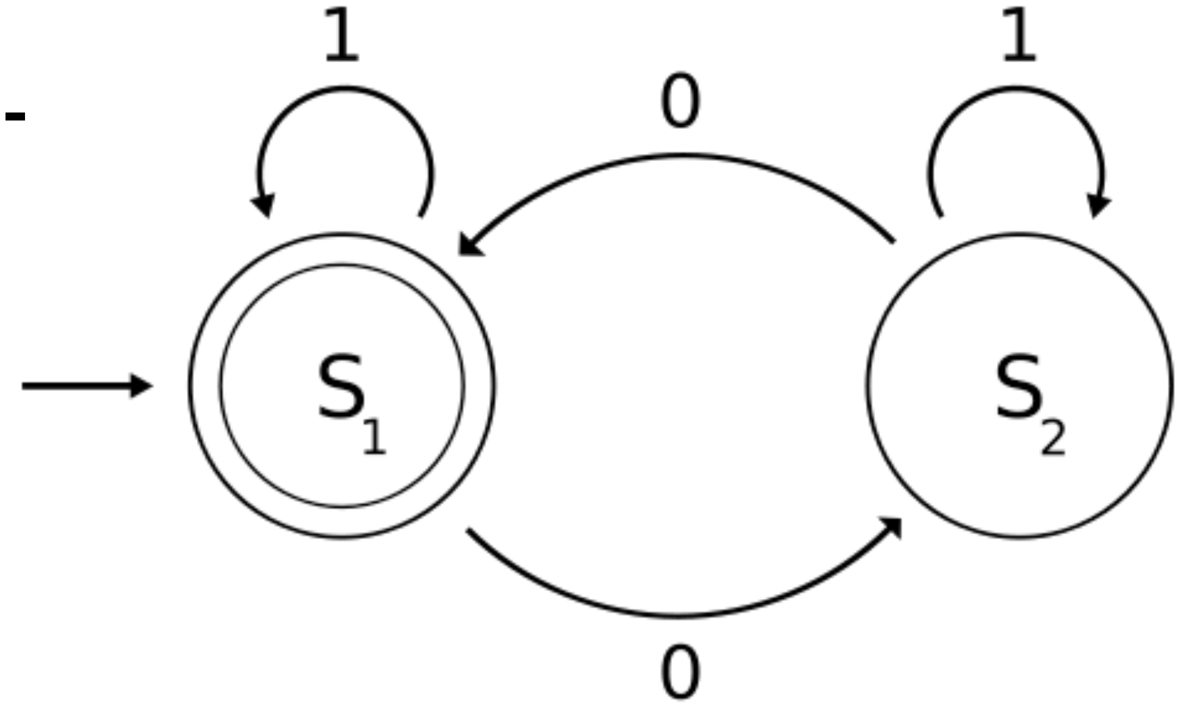
- The input to this machine is a string of binary values.
- This FSM will tell us if there is an even number of 0's in the binary string.
- State (S1) corresponds to an even number of 0’s



Formal representation of a FSM

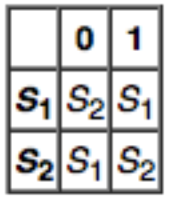
A deterministic finite automaton M is a 5-tuple, (Q, Σ, δ, q0, F), consisting of:

- A finite set of states (Q)
- A finite set of input symbols called the alphabet (Σ)
- A transition function (δ : Q × Σ → Q)
- A start state (q0 ∈ Q)
- A set of accept states (F ⊆ Q)



$M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{S_1, S_2\}$,
- $\Sigma = \{0, 1\}$,
- $q_0 = S_1$,
- $F = \{S_1\}$,
- $\delta =$



	0	1
S_1	S_2	S_1
S_2	S_1	S_2

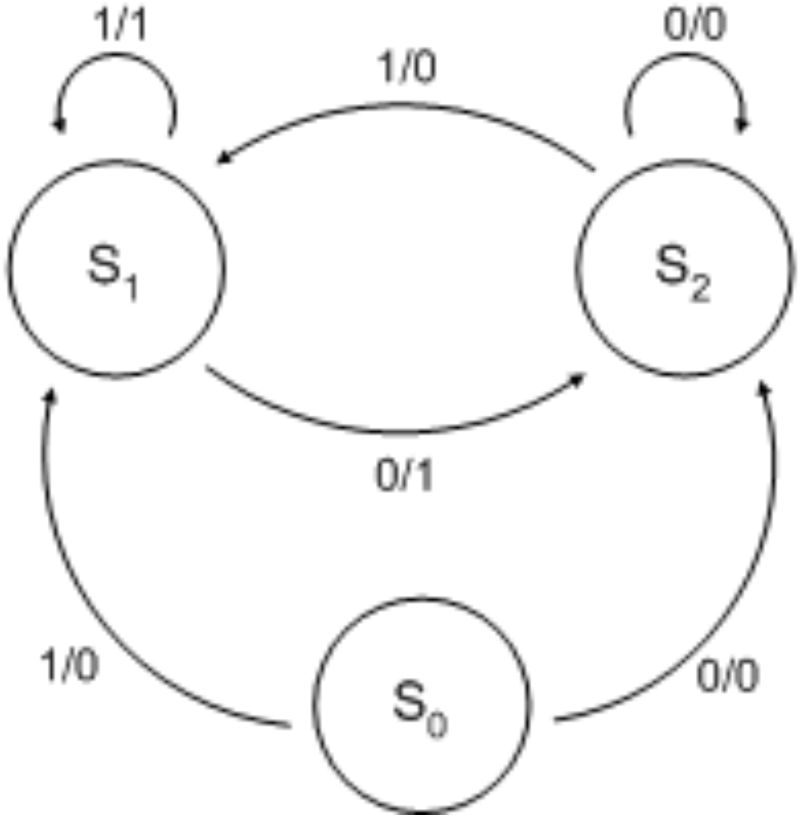
Mealy machine

Mealy machine

- A Mealy machine is a FSM. It takes some input and it generates an output every time is changes state.

- S0, S1, and S2 are states.

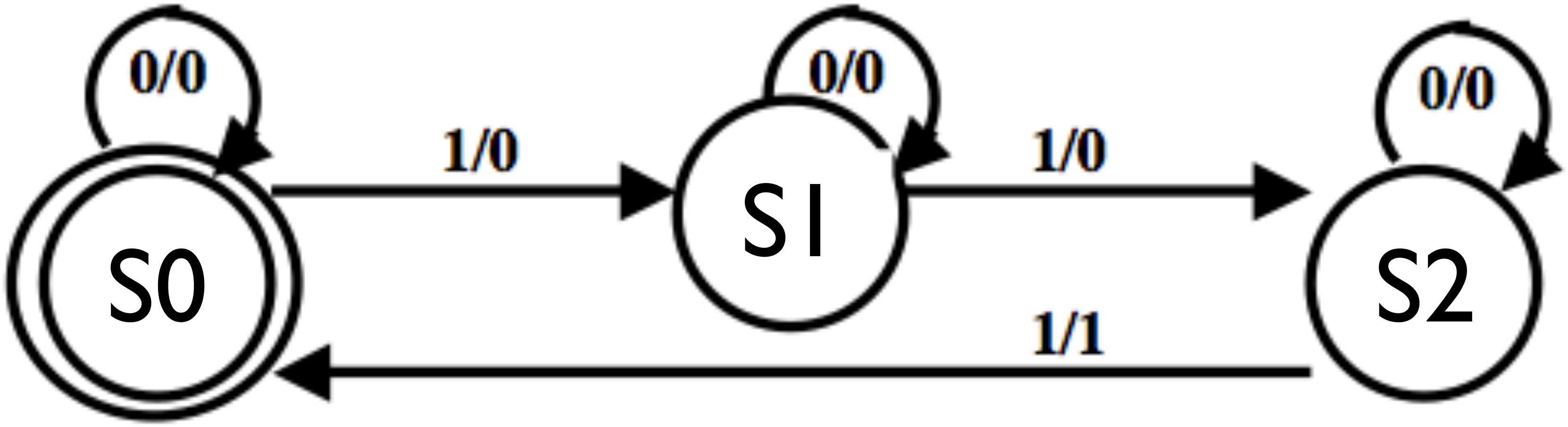
- Each edge is labeled with "**j** / **k**" where **j** is the input and **k** is the output.



Mealy machine FSM example

- State diagram for a Mealy machine with 3 states that outputs a "1" on every third "1" received as input, no matter how many "0"s are intermingled.
- For example:

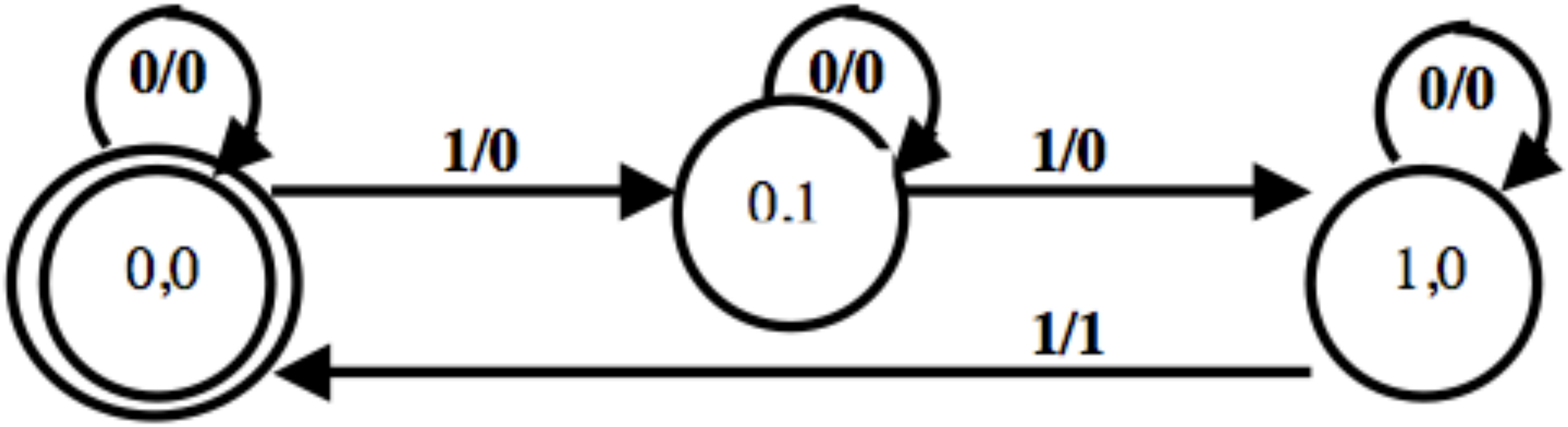
```
Input:   010110100101110110110
Output:  000010000001000100010
```



The circuit output can be for example an LED!

Mealy machine FSM example

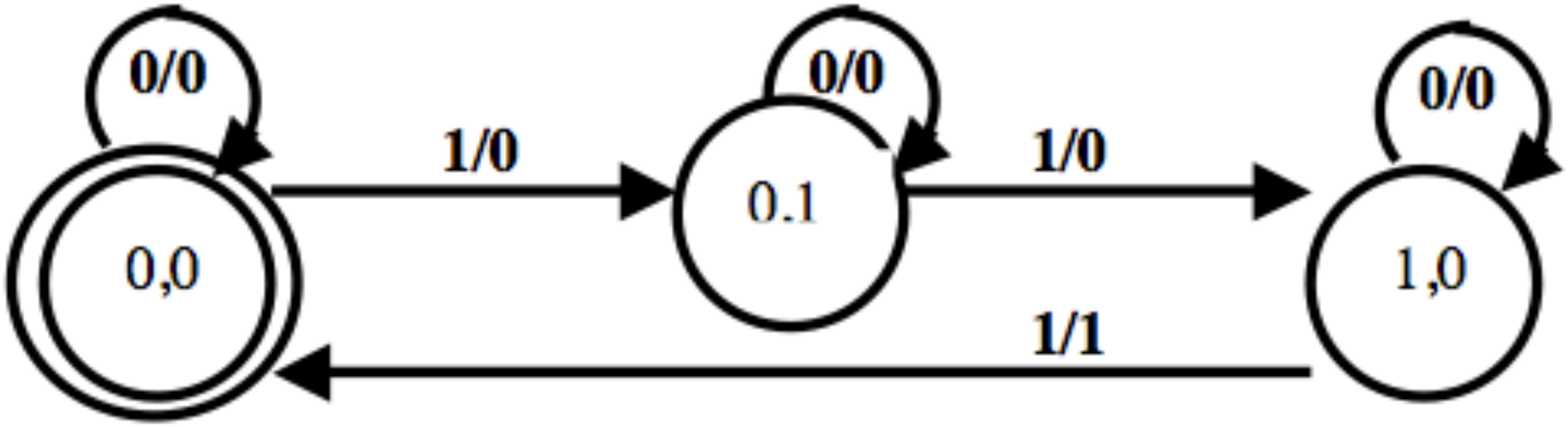
```
Input:   010110100101110110110
Output:  000010000001000100010
```



- Let (P1,P0) represent the current state
- S0 = (0,0) ; S1= (0,1) ; S2=(1,0)
- Let N1,N0 represent the next state

Mealy machine FSM example

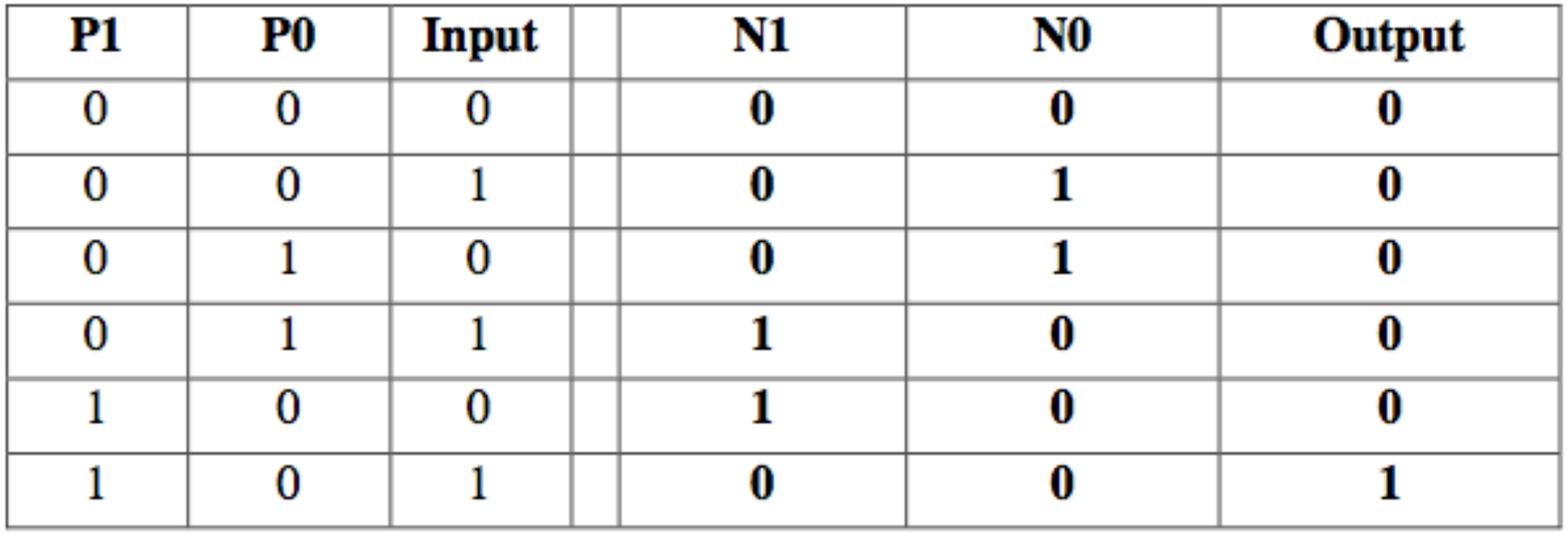
```
Input:   010110100101110110110
Output:  000010000001000100010
```



P1	P0	Input		N1	N0	Output
0	0	0		0	0	0
0	0	1		0	1	0
0	1	0		0	1	0
0	1	1		1	0	0
1	0	0		1	0	0
1	0	1		0	0	1

We need to relate the **next state** with the **current state** and **input**!

Mealy machine FSM example



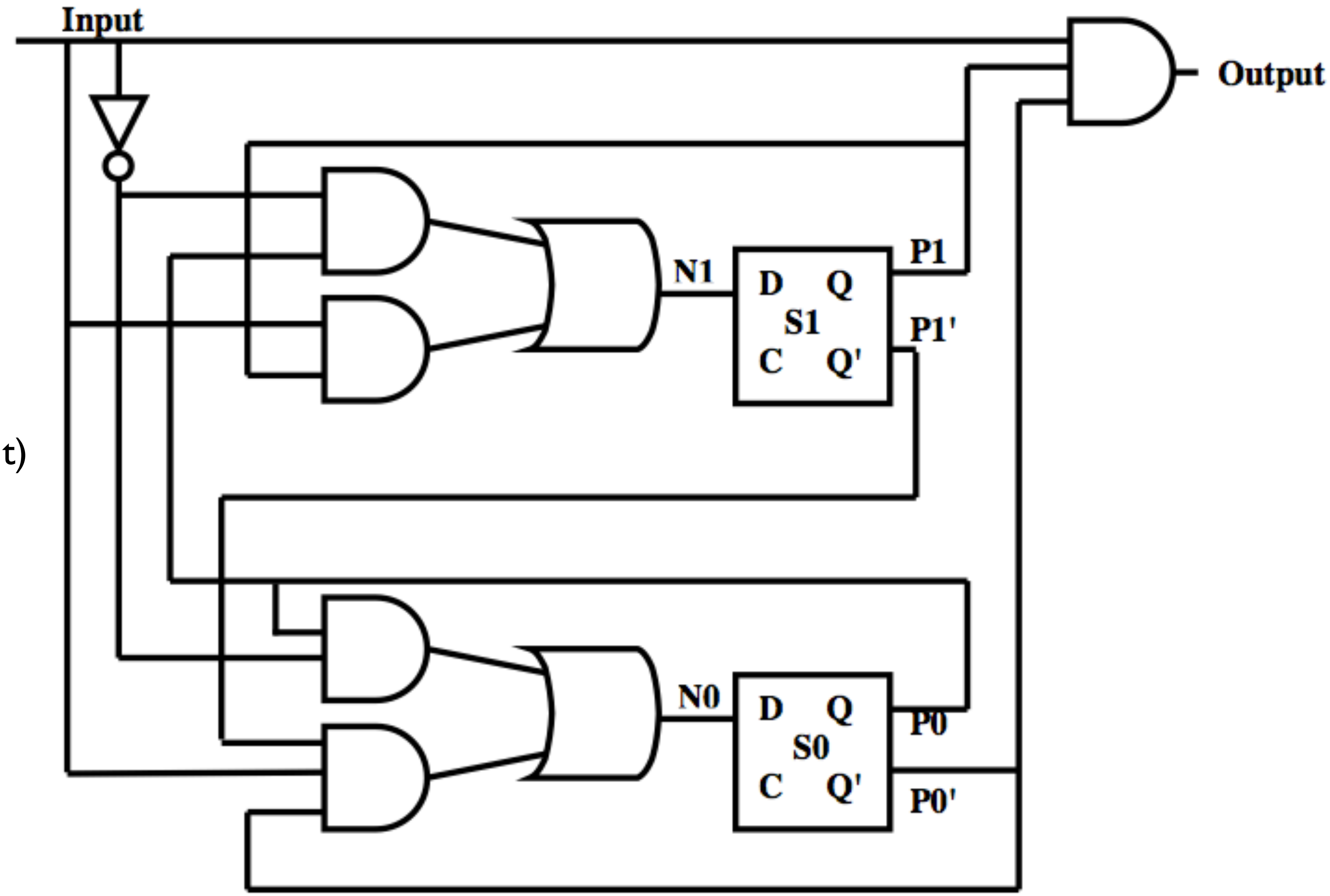
P1	P0	Input		N1	N0	Output
0	0	0		**0**	**0**	**0**
0	0	1		**0**	**1**	**0**
0	1	0		**0**	**1**	**0**
0	1	1		**1**	**0**	**0**
1	0	0		**1**	**0**	**0**
1	0	1		**0**	**0**	**1**

We need to relate the **next state** with the **current state** and **input**!

After some simplification I get:

N1 = (P1 • Input') + (P0 • Input)
N0 = (P0 • Input') + (P1' • P0' • Input)
Output = P1 • P0' • Input

Mealy machine FSM example



N1 = (P1 • Input') + (P0 • Input)
N0 = (P0 • Input') + (P1' • P0' • Input)
Output = P1 • P0' • Input