Computer Architecture

Digital logic: Part 4

Finite state machine

Agenda

- Why we need a finite state machine
- What is a FSM?
    - Types of FSM
    - FSM in digital circuits
    - State transition diagram
    - State transition table
    - Moore machine vs Mealy machine
    - Factoring a FSM
- Design a FSM

Finite state machine

Finite State Machine (FSM): it is a mathematical model used to describe a system that transitions between different states based on inputs.

- It provides a structured way to combine the combinational and sequential logic in a digital system.
- We can use it to design the control unit of the processor with a multiple-cycle microarchitecture.

Finite state machine

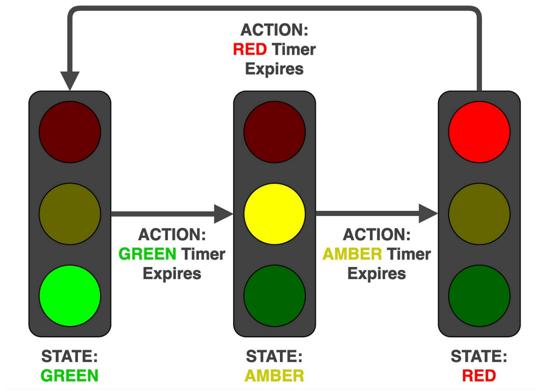
A finite state machine (FSM) model **breaks down** the system into the following components:

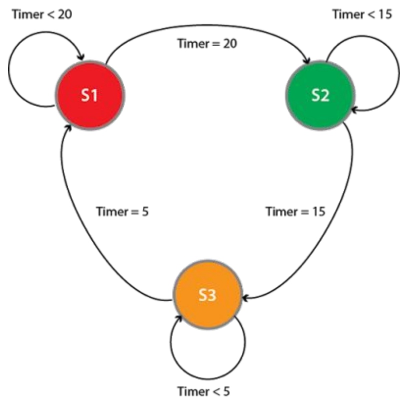
- **States**: different configurations or modes of the system
- **Transitions**: rules or conditions that determine how the system moves from one state to another
- **Input (action)**: signals or data that affect state transitions
- **Output**: the response of the system, which may be dependent on the current state and/or input
- **Initial state**: the starting state when the system is first powered on or reset.

FSM example: Traffic light controller

- **States**: green, amber, red.
- **Transitions**: Time-based transitions that change the light from green to yellow to red.
- **Actions**: the signals from the timer
- **Outputs**: the color of the light, which is determined solely by the current state.

The states transitions after a fixed amount of time, with no immediate response input.





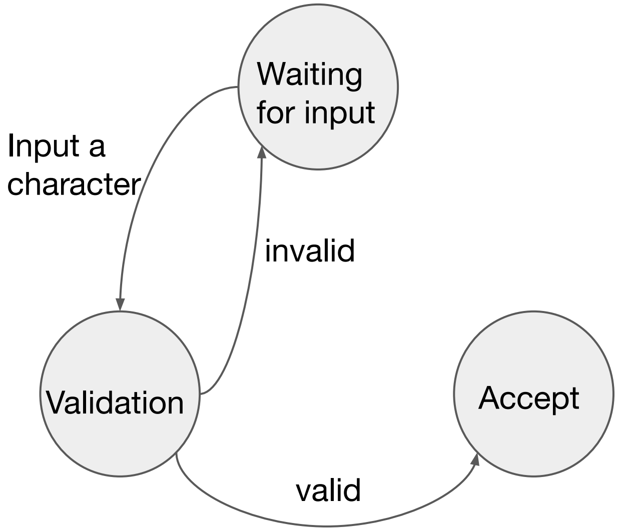
FSM example: Password validator

A valid password must meet the following criteria:

- Have more than 8 characters
- Contains a capital letter
- Have no spaces

It can be described as the following FSM:

- **States**: waiting for input, validation, accept
- **Transitions**: based on the input (each character typed)
- **Input**: each character entered by the user
- **Outputs**: The system immediately checks the validity of each character based on the current state and the input character.



In a state transition diagram, circles represent states and arcs represent transitions between states.

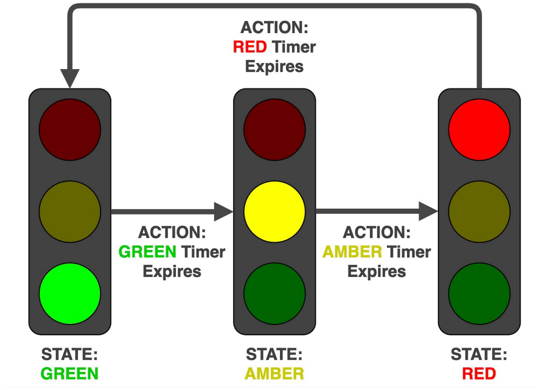
Types of FSMs

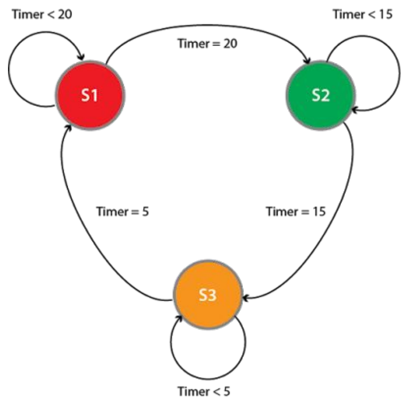
- **Moore machine:** outputs depend solely on the current state, regardless of the input.
  - It is simpler but sometime less flexible than a Mealy machine.
- **Mealy machine:** outputs depend on both the current state and the input.
  - It can react quickly to inputs, as outputs can change immediately upon receiving input.

Moore machine : Traffic light controller

- States: green, amber, red.
- Transitions: Time-based transitions that change the light from green to yellow to red.
- Actions: the signals from the timer
- Outputs: the color of the light, which is determined solely by the current state.

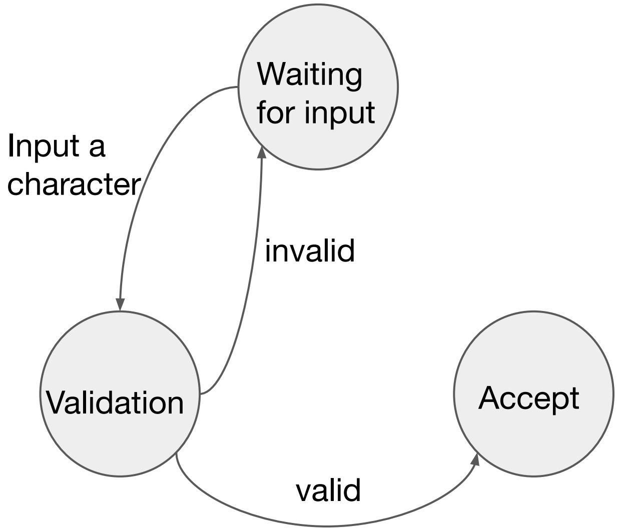
The states transitions after a fixed amount of time, <u>with no immediate response input</u>.





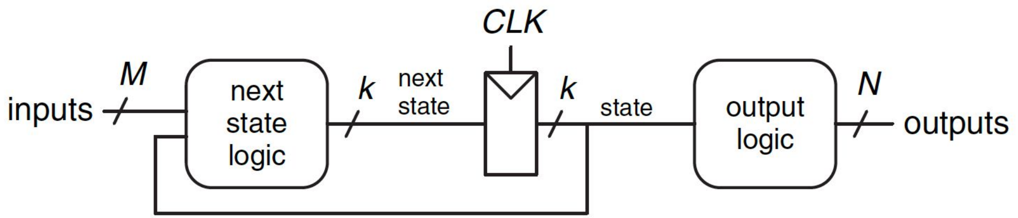
Mealy machine: Password validator

- **States**: waiting for input, validation, accept
- **Transitions**: based on the input (each character typed)
- **Input**: each character entered by the user
- **Outputs**: The system immediately checks the validity of each character based on the current state and the input character.

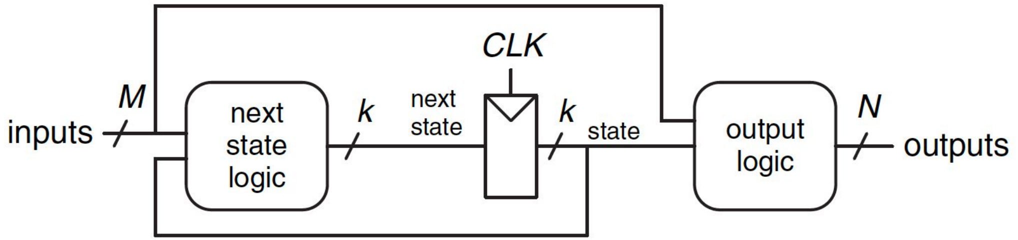


In a state transition diagram, circles represent states and arcs represent transitions between states.

FSM in digital circuits



(a) Moore machine

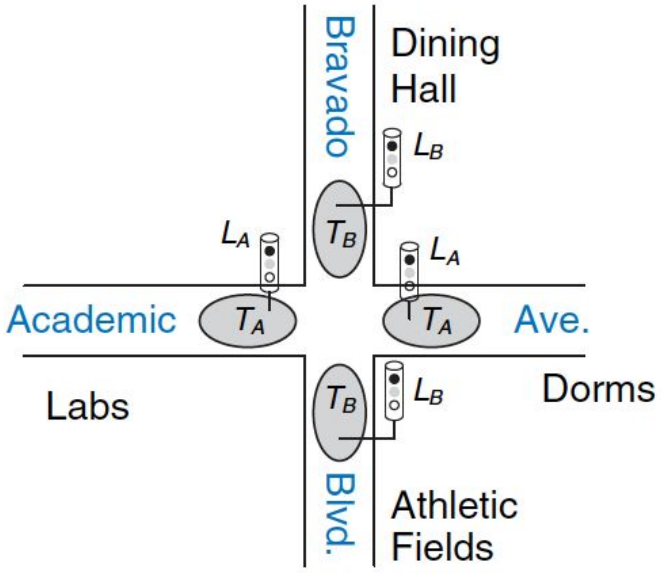


(b) Mealy machine

FSM is a circuit with k registers can have 2^k unique states (the number is finite.). It has

- two blocks of combinatorial logic: *next state logic* and *output logic*;
- a *register* that stores the state.
- In Moore machine, the output only depends the current state.
- In Mealy machine, the output depends on both the current state and the current inputs.

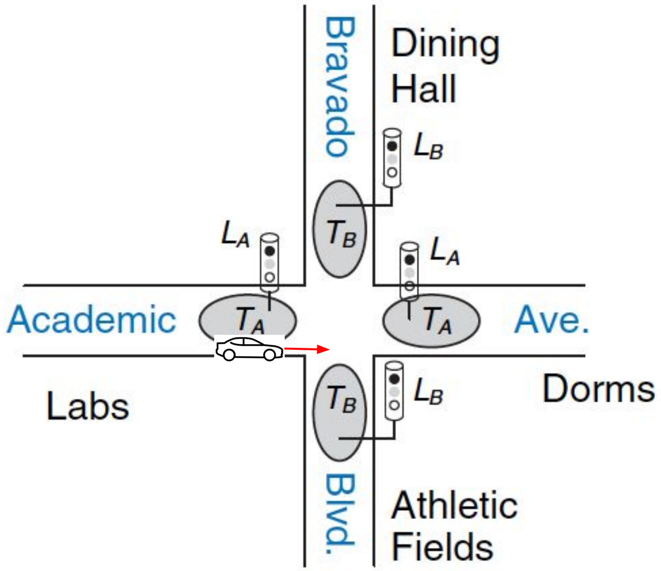
FSM Design Example



There are two streets: Academic Ave., and Bravado Blvd.

- Each street has a set of traffic lights: $\boldsymbol{L_A}$ and $\boldsymbol{L_B}$;
- Each street has traffic sensors: $\boldsymbol{T_A}$ and $\boldsymbol{T_B}$
- $\boldsymbol{T_A}$ and $\boldsymbol{T_B}$ will indicate **True** if passages present on the streets they installed and **False** if the street is empty.
- The system has a clock signal with a 5-second period; on each clock tick (rising edge), the lights may change based on the traffic sensors.

FSM Design Example

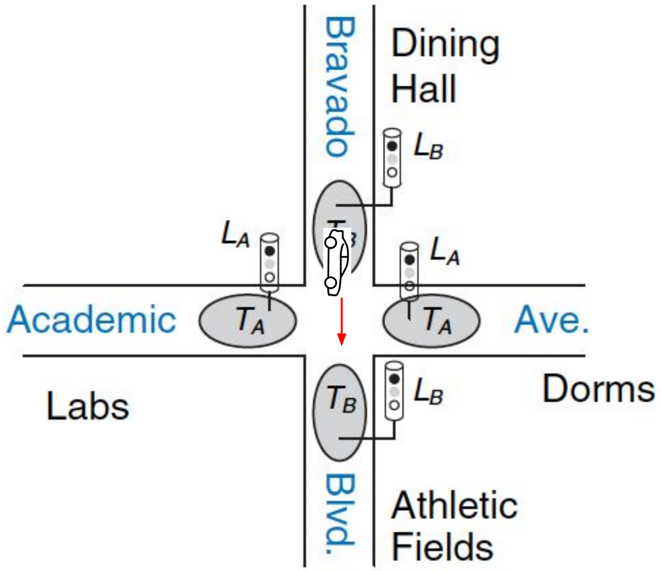


Every 5 seconds, the controller examines the traffic pattern and decides what to do next.

For Academic Ave.,

- if there is traffic on it, $\boldsymbol{T_A}$ is true, and the lights do not change;
- if there is no traffic on it, $\boldsymbol{T_A}$ is false, its lights become yellow for 5 seconds, and then, they turn into red; in the meanwhile, the lights of the Bravado Blvd. behave similarly, but turn into green.

FSM Design Example

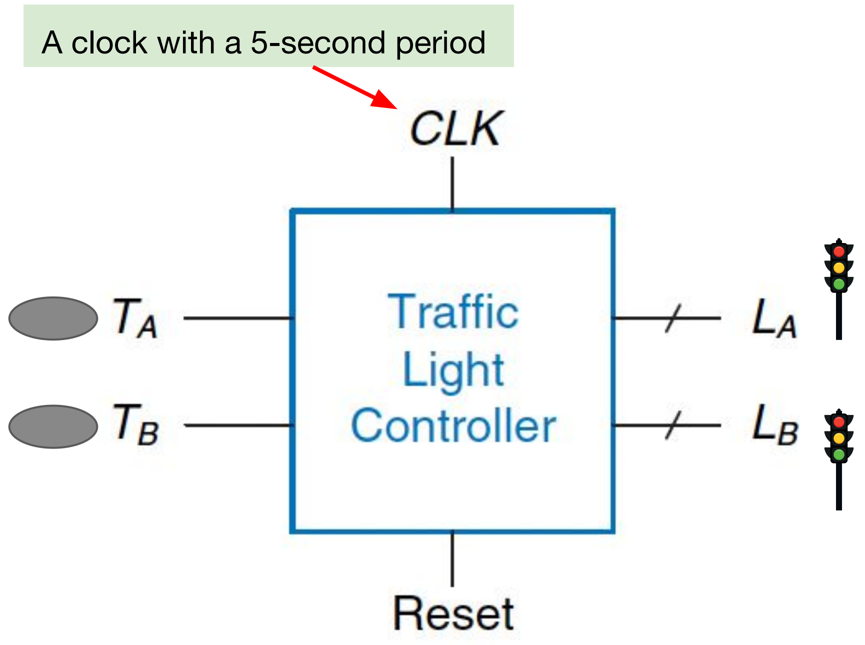


Every 5 seconds, the controller examines the traffic pattern and decides what to do next.

For Bravado Blvd.,

- if there is traffic on it, $\boldsymbol{T_B}$ is true, and the lights do not change;
- if there is no traffic on it, $\boldsymbol{T_B}$ is false, its lights become yellow for 5 seconds, and then, they turn into red; in the meanwhile, the lights of the Academic Ave. behave similarly, but turn into green.

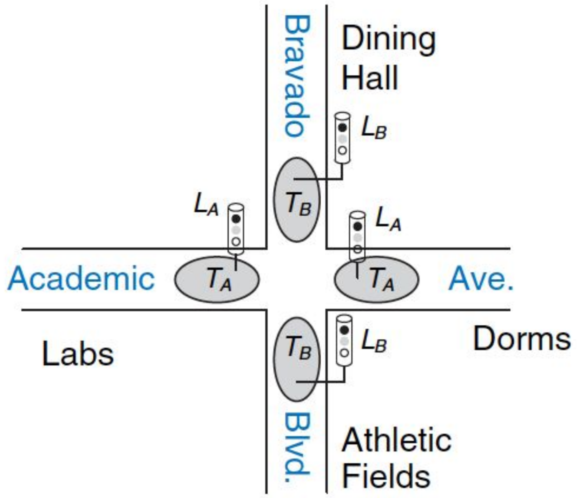
Black box view of the lighter controller



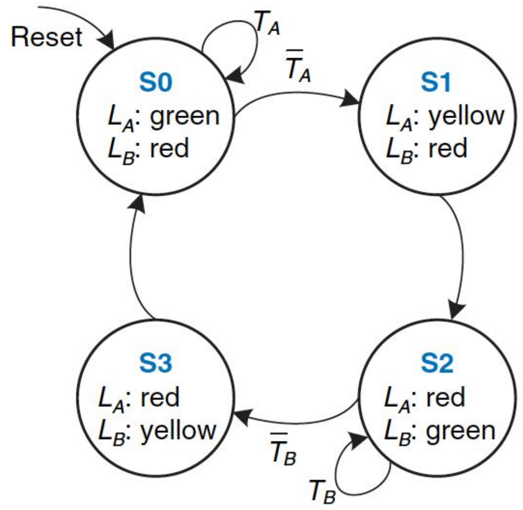
- A reset button is added, which allows people to put the controller in a known initial state when they turn it on.
- When the system is reset, the lights are green on Academic Ave. and red on Bravado Blvd.
- Every 5 seconds, the controller examines the traffic pattern and decides what to do next.
- When a traffic light changes its color from green to red or vice versa, it will become yellow for 5 seconds then change to the target color.

The FSM of the lights on Academic Ave. and Bravado Blvd.

- States
  - S0: L_A=green, L_B=red
  - S1: L_A=yellow, L_B=red
  - S2: L_A=red, L_B=green
  - S3: L_A=red, L_B=yellow
- Transitions:
  - T_A becomes false, and,
  - T_B becomes false.
  - Clock signal: every 5 seconds (depending on T_A, T_B)
  - Reset



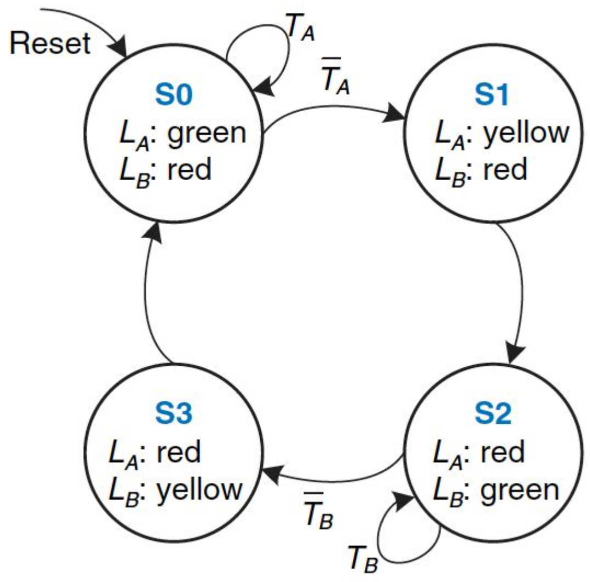
State transition diagram



It is a Moore FSM. The outputs are determined by the current state only. So, a circle is labeled by the state and its outputs; an arc is labeled by the input.

- Circles represent states and arcs represent transitions between states.
- The transitions take place on the rising edge of the clock;
- The clock signal is omitted from the diagram for two reasons: it is always present in a synchronous sequential circuit, and it simply controls when the transitions should occur, whereas the diagram indicates which transitions occur.
- The Reset arc pointing from outer space into S0 indicates that the system should enter that state upon reset.

State transition table

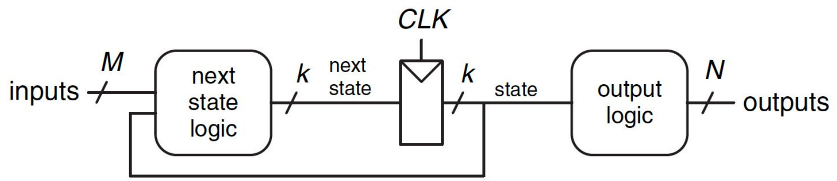


Current State S	Inputs T_A	T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

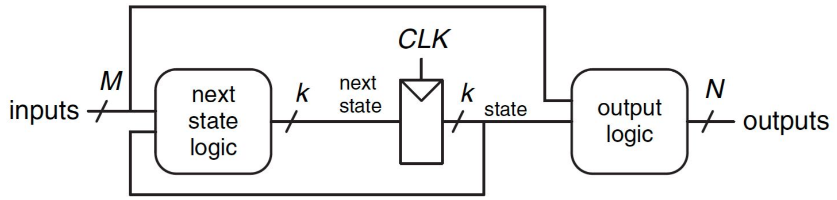
Two blocks of combinational logic

An FSM consists of two blocks of combinational logic, ***next state logic*** and ***output logic***, and a register that stores the state.

- The current state is the input of two blocks of combinational logic.



(a)



(b)

State transition table with binary encodings

Table 3.1 State transition table

Current State S	Inputs T_A	T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

+

Table 3.2 State encoding

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Table 3.4 State transition table with binary encodings

Current State S_1	S_0	Inputs T_A	T_B	Next State S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

From Table 3.4, we have the Boolean equations for next state in sum-of-products form:

$$S'_1 = \overline{S}_1 S_0 + S_1 \overline{S}_0 \overline{T}_B + S_1 \overline{S}_0 T_B$$

$$S'_0 = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

Using Karnaugh maps, we have the simplified *next state equations*:

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

Output table and output equations

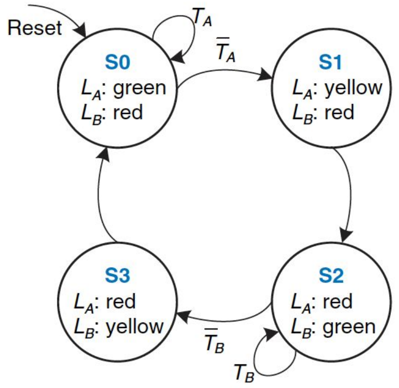


Table 3.2 State encoding

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Table 3.3 Output encoding

Output	Encoding $L_{1:0}$
green	00
yellow	01
red	10

Table 3.5 Output table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Boolean equations for the outputs:

$$L_{A1} = S_1$$

$$L_{A0} = \overline{S}_1 S_0$$

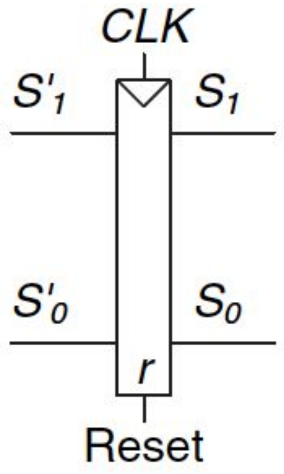
$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$

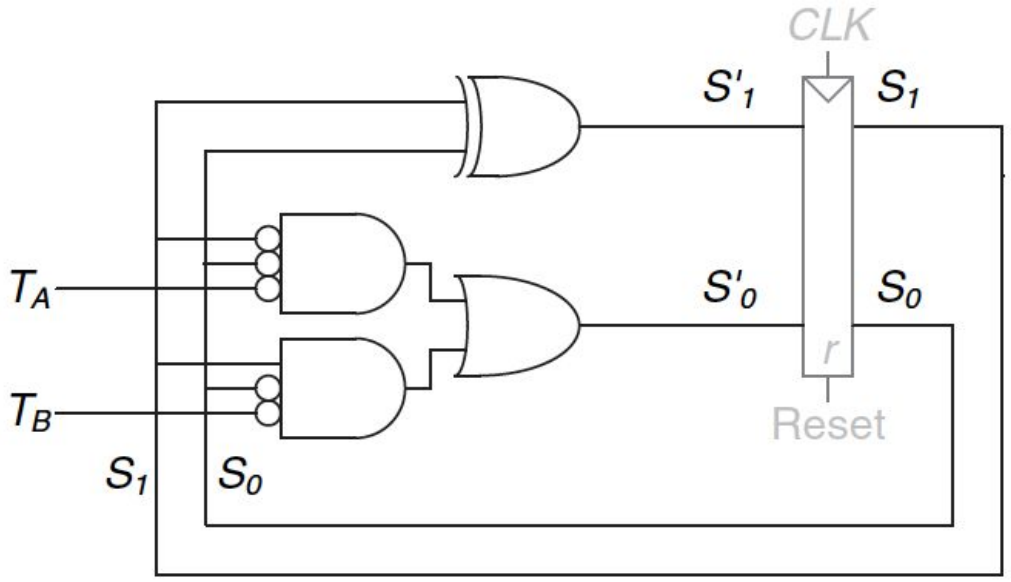
Sketching the Moore FSM: State register

2-bit state register:

- it should store two bits because of the state encoding used in the design;
- on each clock edge, it copies the next state $S'_{1:0}$ to become the "current" state $S_{1:0}$;
- it has a reset to initialize the FSM at startup.



Sketching the Moore FSM: next state logic



inputs

next state logic

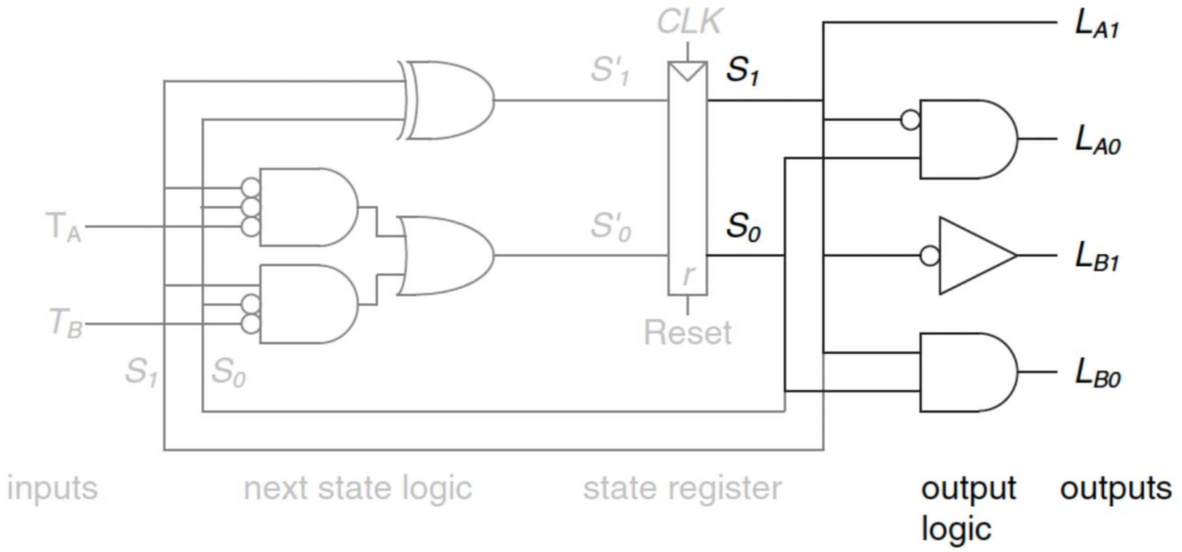
state register

Next state equations:

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

Sketching the Moore FSM: output logic



Output equations:

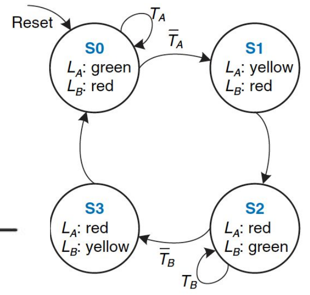
$$L_{A1} = S_1$$

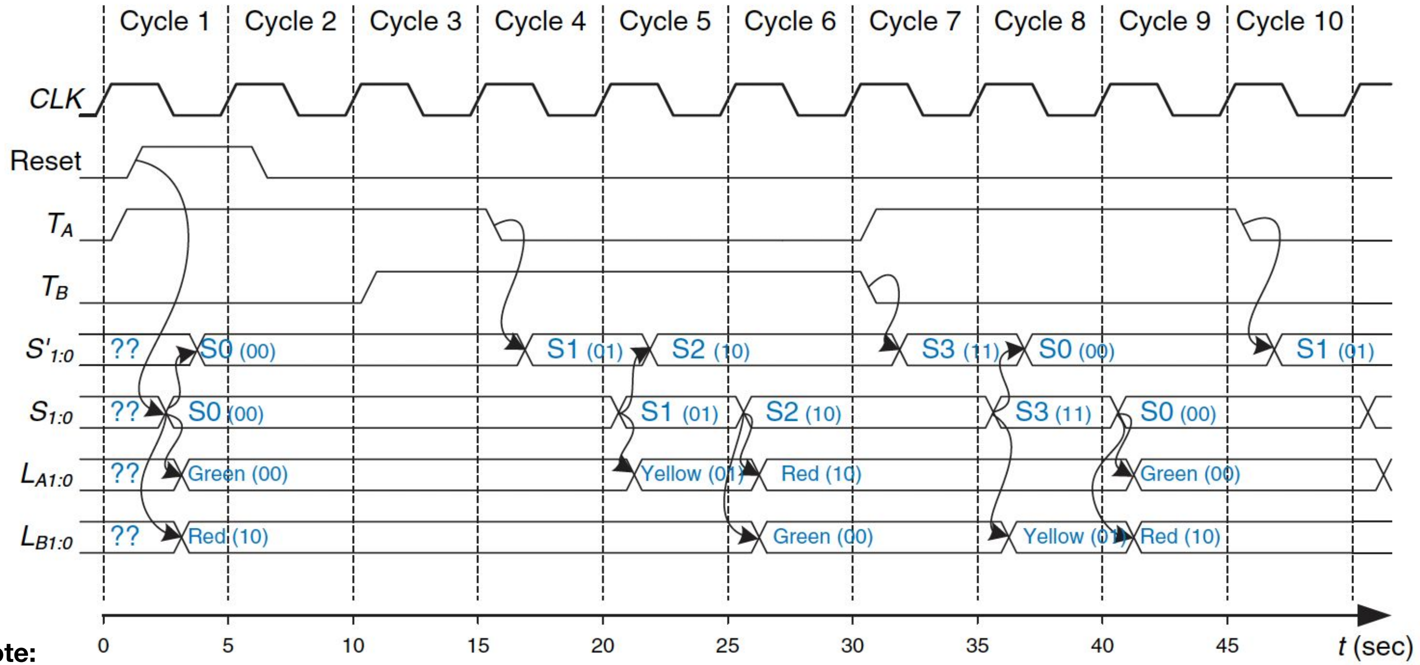
$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$

The timing diagram





Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Cycle 9
Cycle 10
CLK
Reset
T_A
T_B
$S'_{1:0}$
?? S0 (00) S1 (01) S2 (10) S3 (11) S0 (00) S1 (01)
$S_{1:0}$
?? S0 (00) S1 (01) S2 (10) S3 (11) S0 (00)
$L_{A1:0}$
?? Green (00) Yellow (01) Red (10) Green (00)
$L_{B1:0}$
?? Red (10) Green (00) Yellow (01) Red (10)
0
5
10
15
20
25
30
35
40
45
t (sec)

Note:

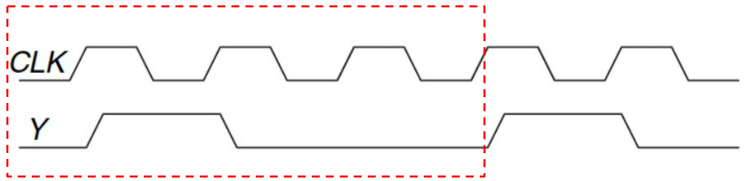
- *S, S', L_A,* and *L_B* are variables representing multiple bits, not a single bit. So, we can use a stripe, not a single line, to show their states in the timing diagram. We should label the state on the stripes.
- *S* is the output of the register which changes only at the rising edge (except when being reset), while the others are the outputs of combinational circuits.

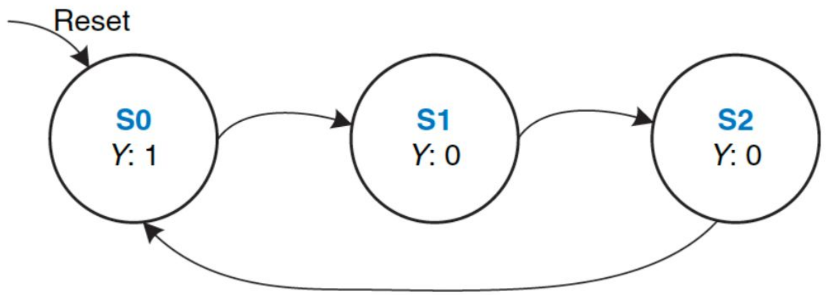
State encodings

- A different choice of state and output encodings would have resulted in a different circuit.
    - No simple way to find the best encoding that using the fewest logic gates or shortest propagation delay. (only can solve by brutal force, trying all possibilities)
- But two options often use:
    - **Binary encoding**: each state is represented as a binary number; K binary numbers can be represented by log_2K bits ⇒ a system with K states only needs log_2K bits of state.
    - **One-hot encoding**: a separate bit of state is used for each state; in the representation, only one bit is *TRUE* (i.e., 1) at any time.

Example: a divide-by-3 counter

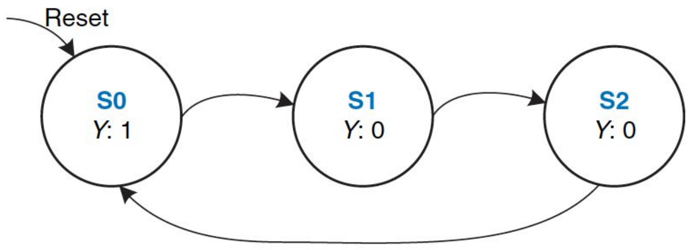
A divide-by-3 counter has one output and no inputs. Its waveform and state transition diagram is shown below. The output ***Y*** is HIGH for one clock cycle out of every 3.



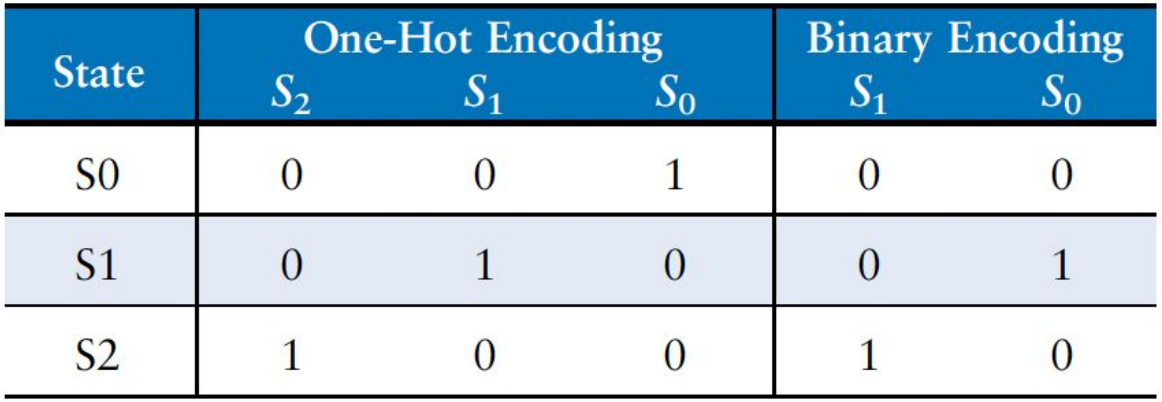


State transition table

Current State	Next State
S0	S1
S1	S2
S2	S0



State encoding



State	One-Hot Encoding			Binary Encoding	
	S_2	S_1	S_0	S_1	S_0
S0	0	0	1	0	0
S1	0	1	0	0	1
S2	1	0	0	1	0

State transition table with binary encoding

Current State S_1 S_0	Next State S'_1 S'_0	Output Y
0 0	0 1	1
0 1	1 0	0
1 0	0 0	0

The state transition equations and output equations:

$$S'_1 = \overline{S}_1 S_0$$
$$S'_0 = \overline{S}_1 \overline{S}_0$$

$$Y = \overline{S}_1 \overline{S}_0$$

State transition table with one-hot encoding

Current State S_2 S_1 S_0	Next State S'_2 S'_1 S'_0	Output Y
0 0 1	0 1 0	1
0 1 0	1 0 0	0
1 0 0	0 0 1	0

The state transition equations and output equations:

$$S'_2 = S_1$$
$$S'_1 = S_0$$
$$S'_0 = S_2$$

$$Y = S_0$$

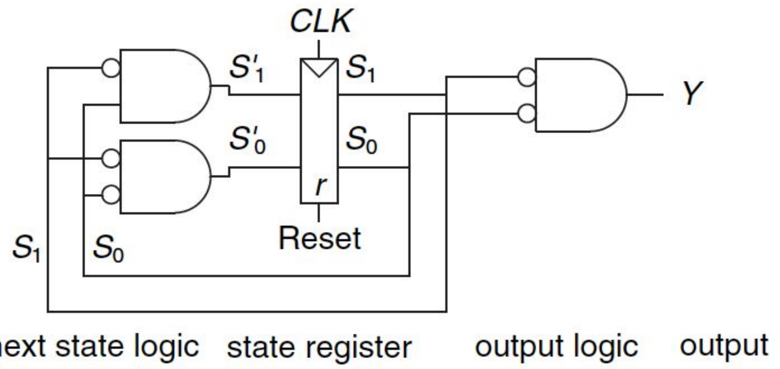
Circuits of the two different encodings

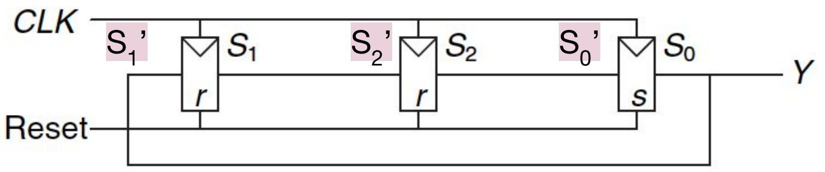
$$S'_1 = \overline{S}_1 S_0$$
$$S'_0 = \overline{S}_1 \overline{S}_0$$

$$Y = \overline{S}_1 \overline{S}_0$$

$$S'_2 = S_1$$
$$S'_1 = S_0$$
$$S'_0 = S_2$$

$$Y = S_0$$





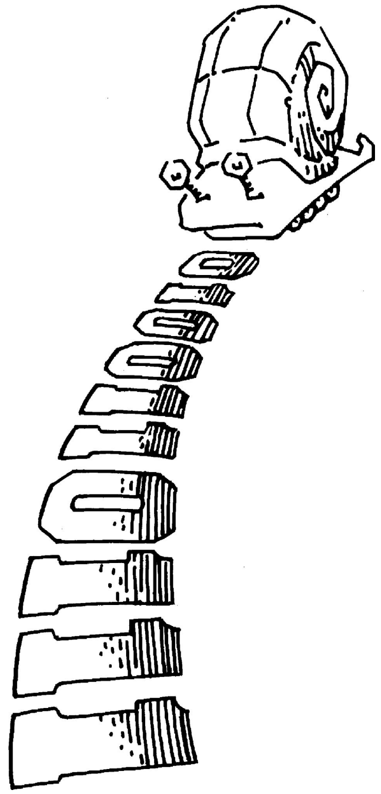
It can be implemented by three 1-bit registers. The settable and resettable flip-flops to initialize the machine to S0 on reset.

Moore vs. Mealy machines (H&H Section 3.4.3)

Alyssa P. Hacker owns a pet robotic **snail** with an FSM brain.

The snail crawls from left to right along a paper tape containing a sequence of 1's and 0's. On each clock cycle, the snail crawls to the next bit. The snail smiles when the last two bits that it has crawled over are, from left to right, 01.

- Design the FSM to compute when the snail should smile. The input *A* is the bit underneath the snail's antennae. The output *Y* is *TRUE* when the snail smiles.
- Compare Moore and Mealy state machine designs.
- Sketch a timing diagram for each machine showing the input, states, and output as Alyssa's snail crawls along the sequence 0100110111.



Analysis

- What are the patterns of the last two bits the snail crawled over? (i.e., the states)
  - 00, 01 (smile), 10, 11
- What are the transitions?
  - 00→0⇒00, 00→1⇒01, 01→0⇒10, 01→1⇒11, 10→0⇒00, 10→1⇒01, 11→0⇒10, 11→1⇒11;
- States reduction: Duplicate states have the same outputs and the same transitions (inputs).
  - 00 and 10 have the same inputs and outputs (00 and 01, both are not-smile), so remove one of them.
  - 01 and 11 have the same inputs but different outputs (10 is smile, 11 is not-smile), so both should be kept.

The Moore machine

Let's set the states as the following,

- S0 be the state where the last two bits are 11;
- S1 be the state where the last two bits are x0; (x means don't care)
- S2 be the state where the last two bits are 01.

We have the following state transition diagram. It is a Moore machine, so the output is tied to the state. Every possible output combination needs its own state.

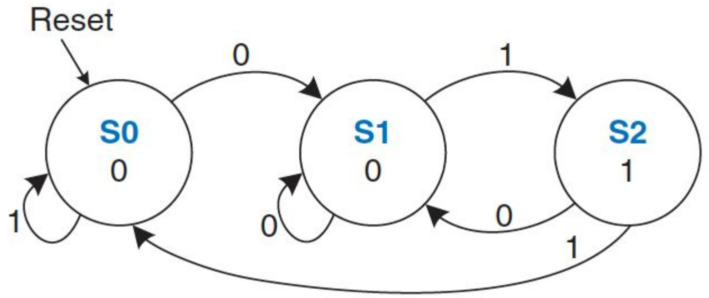


Table 3.11 Moore state transition table

Current State *S*	Input *A*	Next State *S′*
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2
S2	0	S1
S2	1	S0

Table 3.12 Moore output table

Current State *S*	Output *Y*
S0	0
S1	0
S2	1

The circuit of the Moore machine

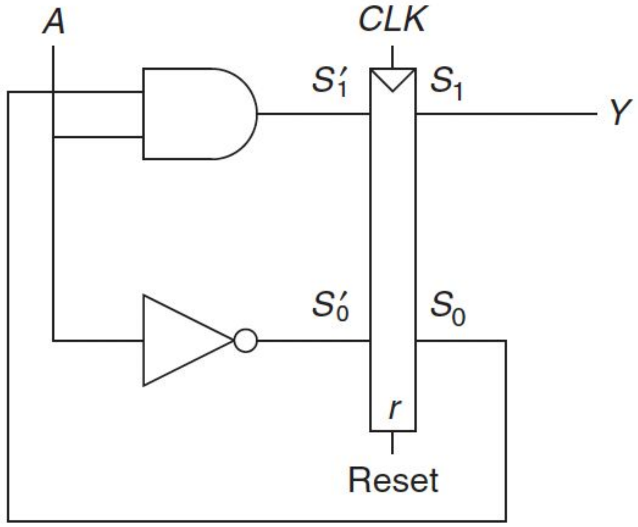


Table 3.13 Moore state transition table with state encodings

Current State S_1	S_0	Input A	Next State S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

Table 3.14 Moore output table with state encodings

Current State S_1	S_0	Output Y
0	0	0
0	1	0
1	0	1

The equations:

$$S'_1 = S_0 A$$
$$S'_0 = \overline{A}$$

$$Y = S_1$$

The Mealy machine

In a Mealy machine, the output can change immediately when an input is received. So, we only need to store the last bit the snail crawled over, not the last two bit,

- S0 is the state where the last bits is 1;
- S1 is the state where the last bit is 0.

In the state transition diagram, each arc is labeled as *A*/*Y*, where *A* is the input that causes the transition, and *Y* is the corresponding output; each circle is labeled as the state.

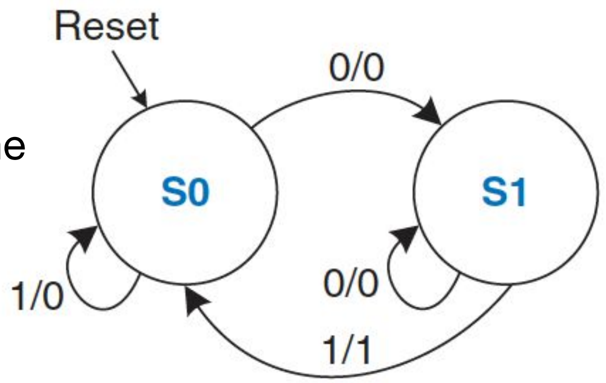


Table 3.15 Mealy state transition and output table

Current State *S*	Input *A*	Next State *S′*	Output *Y*
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S0	1

The circuit of the Mealy machine

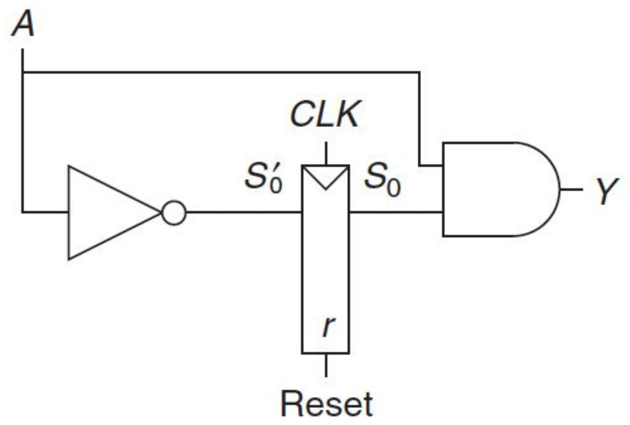


Table 3.16 Mealy state transition and output table with state encodings

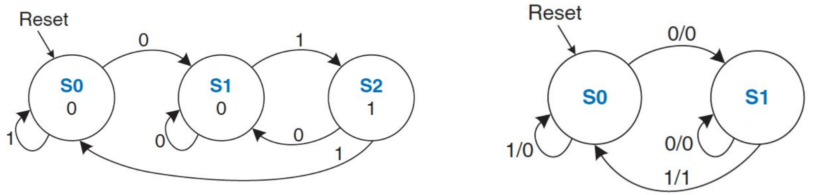
Current State S_0	Input A	Next State S_0'	Output Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

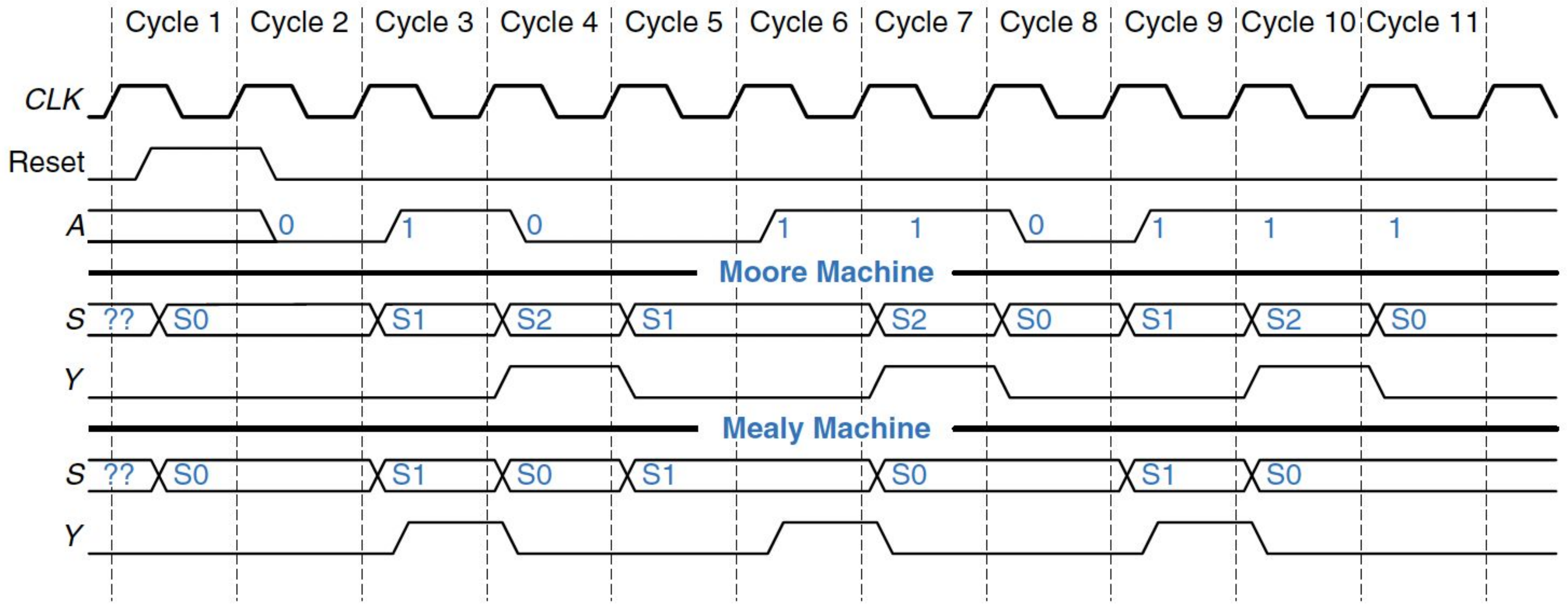
The equations:

$$S_0' = \overline{A}$$

$$Y = S_0 A$$

Timing diagram





Note: the Mealy machine's output rises a cycle sooner because it responds to the input rather than waiting for the state change.

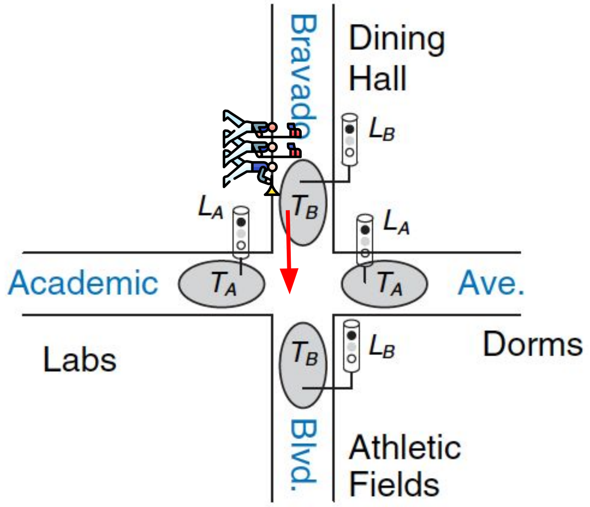
Factoring state machines

Designing complex FSM is often easier if they can be broken down into multiple interacting simpler state machine such that the output of some machines is the input of others.

Modification of the traffic light controller

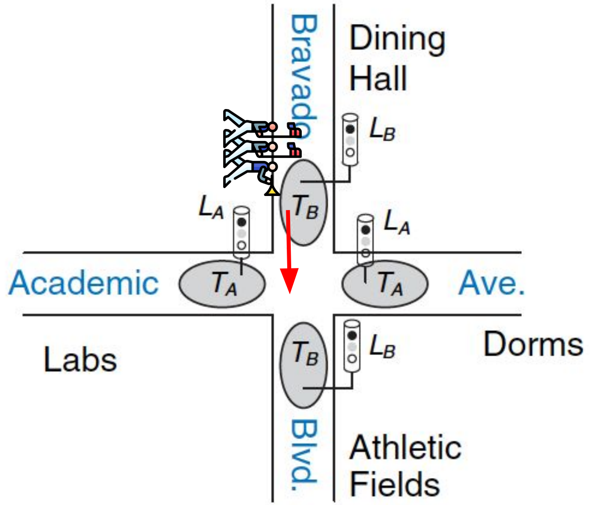
Adding a parade mode to the controller, which keeps the Bravado Blvd. light green while spectators and the band march to football games in scattered groups.

Now, the controller receives two more inputs: ***P*** and ***R***.



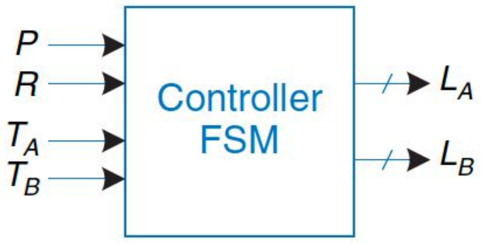
Modification of the traffic light controller

- Asserting ***P*** for at least one cycle enters parade mode.
- Asserting ***R*** for at least one cycle leaves parade mode.
- When in parade mode, the controller proceeds through its usual sequence until ***L*_B** turns green, then remains in that state with ***L*_B** green until parade mode ends.



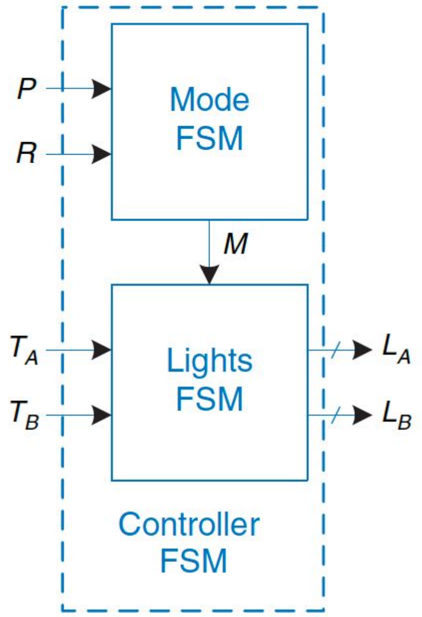
Sketching a transition system

Single FSM design

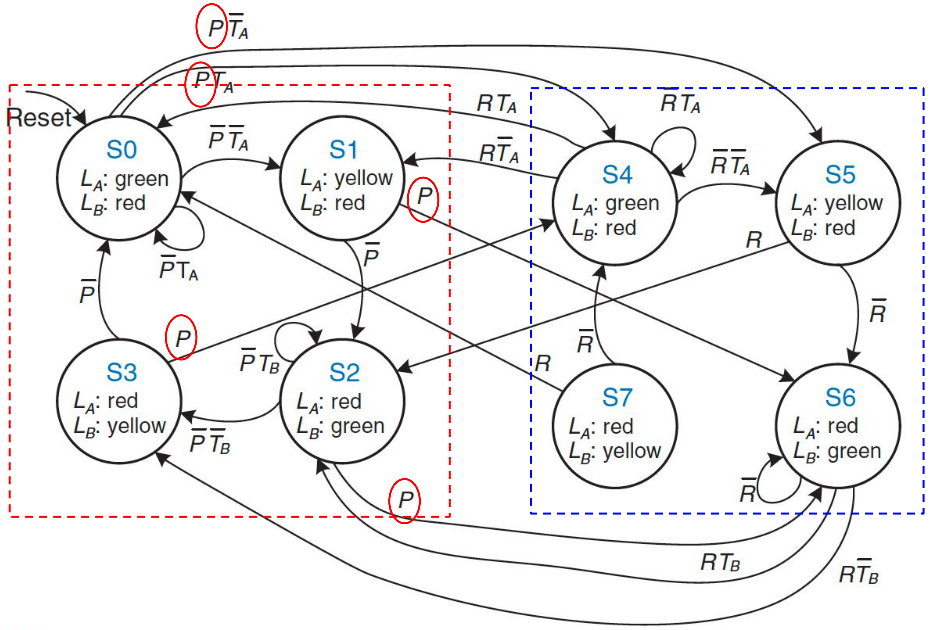


In the factored design, the Mode FSM asserts the output M when it is in parade mode. The Lights FSM controls the lights based on M and the traffic sensors, T_A and T_B.

Factored FSM design

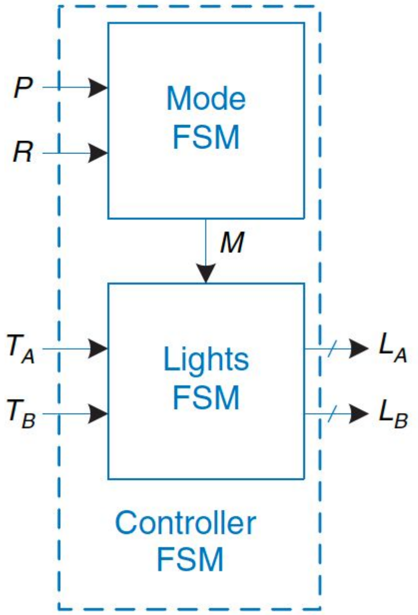


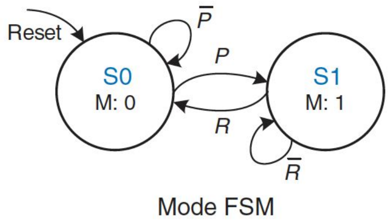
Transition system for a single design

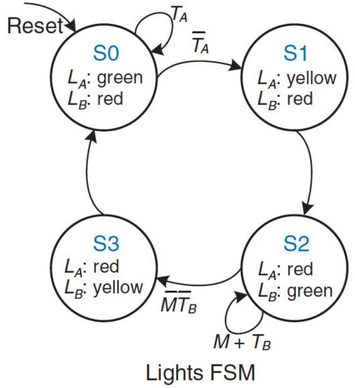


- Normal mode: S0 to S3 handle the normal mode.
- Parade mode: S4 to S7 handle the parade mode.
- The ***P*** and ***R*** control movement between these two modes.

Transition system for a factored design







In the factored version, the FSM has two states to track whether the lights are in normal or parade mode.

The lights FSM is modified to remain in S2 while M is True.

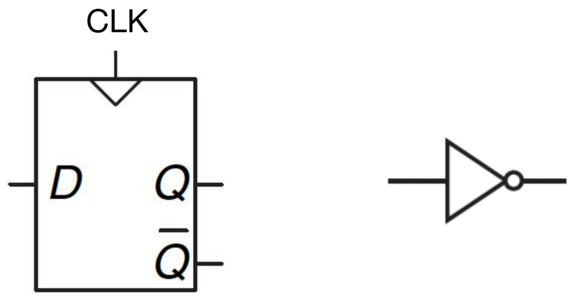
Deriving an Finite State Machine Review

- Identify the inputs and outputs
- Sketch a state transition diagram
- For a Moore machine:
  - Write a state transition table
  - Write an output table
- For a Mealy machine:
  - Write a combined state transition and output table
- Select state encodings: your selection affects the hardware design
- Write Boolean equations for the next state and output logic
- Sketch the circuit schematic

Exercise (Exercise 3.9 in H&H)

The toggle (T) flip-flop has one input, CLK, and one output, Q. On each rising edge of CLK, Q toggles to the complement of its previous value.

Draw a schematic for a T flip-flop using a D flip-flop and an inverter.



Exercise (Exercise 3.24 in H&H)

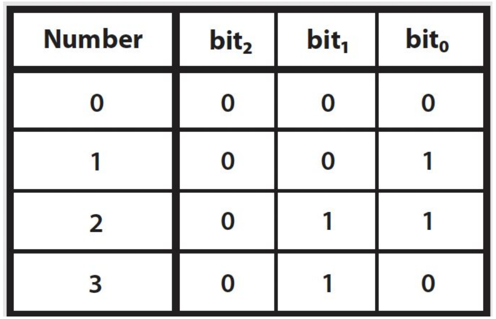
You have been enlisted to design a soda machine dispenser for your department lounge. Sodas are partially subsidized by the student chapter of the IEEE, so they cost only 25 cents. The machine accepts nickels(5), dimes(10), and quarters(15). When enough coins have been inserted, it dispenses the soda and returns any necessary change.

- Design an FSM controller for the soda machine. The FSM inputs are Nickel, Dime, and Quarter, indicating which coin was inserted. Assume that exactly one coin is inserted on each cycle. The outputs are Dispense, ReturnNickel, ReturnDime, and ReturnTwoDimes.
- When the FSM reaches 25 cents, it asserts Dispense and the necessary Return outputs required to deliver the appropriate change. Then it should be ready to start accepting coins for another soda.

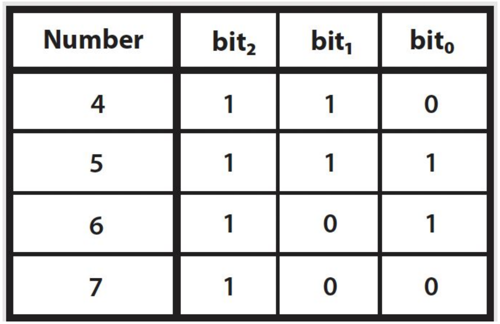
Exercise (Exercise 3.27 in H&H)

Gray codes have a useful property in that consecutive numbers di er in only a single bit position. The table below lists a 3-bit Gray code representing the numbers 0 to 7.

Design a 3-bit modulo 8 Gray code counter FSM with no inputs and three outputs. A modulo N counter counts from 0 to N – 1, then repeats. For example, a watch uses a modulo 60 counter for the minutes and seconds that counts from 0 to 59. When reset, the output should be 000. On each clock edge, the output should advance to the next Gray code. After reaching 100, it should repeat with 000.



Number	bit_2	bit_1	bit_0
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0



Number	bit_2	bit_1	bit_0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0