CET 141: Day 14 -15

Dr. Noori KIM

Finite State Machine
-A quick guide first-

Dr. Noori Kim

Introduction

State machines are an important tool to use when designing complex sequential circuits. For example, to implement a USB protocol in hardware, the circuit must be "aware" and respond appropriately whether data is coming-in, coming-out, or under the error state. State machines can also connect different circuits that are operating asynchronously by placing part of the circuit into an "idle" state to wait for another circuit to send or receive data.
This lab serves as a springboard for topics that will be discussed in the Digital Electronics II course.

Pre-lab

You will derive and implement a Moore Machine state detector to look for the sequence '1001' with overlaps on a digital input line.
To complete this lab, your work must be checked by TAs during lab session and upload your report on the moodle. Failure to complete your demo will result in a reduction of the grade by 10% for the current lab.

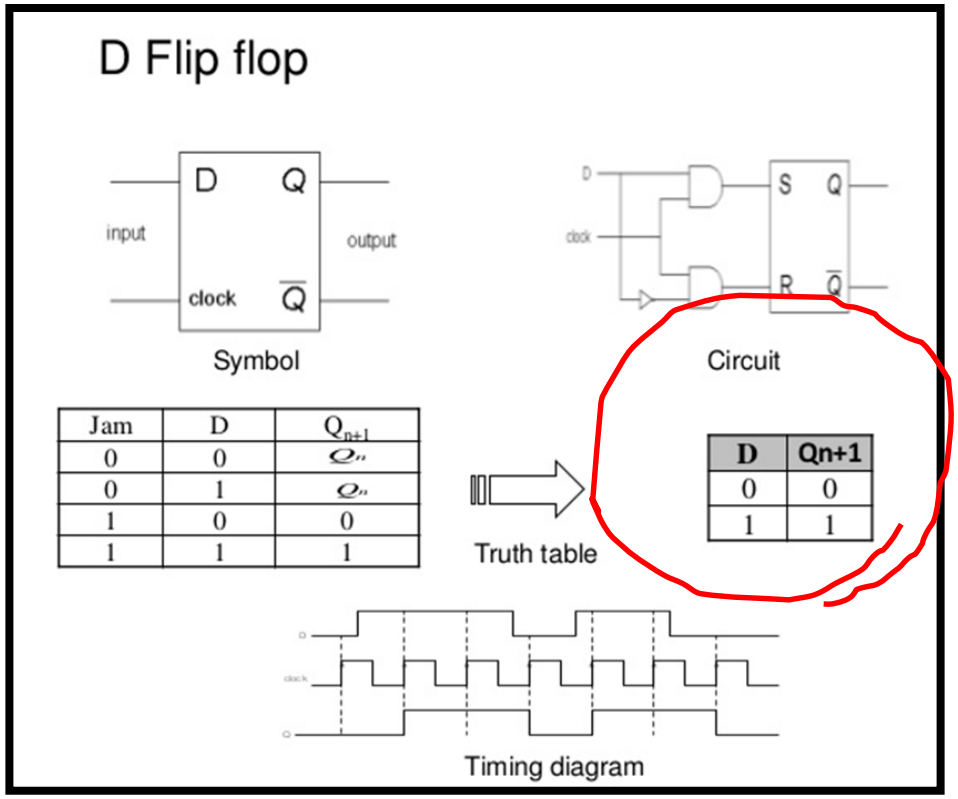
1) Derive a Moore machine state transition diagram for the sequence detector that looks for the sequence '1001' with overlaps on a digital input line.
2) Find the state transition table from the diagram.
3) Derive the logic for the flip-flop inputs based on the state table (using JK Flip-Flop or D Flip-Flop).
4) **Prepare a circuit schematic (Eagle).**

To complete our lab, we need to know the followings

a. State Transition Diagram
b. State Encoding and Output Encoding tables
c. State Transition Table (both without and with encoding)
d. Next State Logic Boolean (with K-maps)
e. Output Logic Boolean (with K-map and Boolean expression)

- Let me take an example and explain what they are
  - **A sequence detector for 01 or 10 <u>with overlap</u>**
  - Don’t forget we are using Moore Machine (The output is a function of states)
  - This example uses D flip-flops
    - If you want to use other FFs, you must consider different input Boolean logic customized to each FF.
    - **For your lab implementation, use D Flip-flops**

Excitation table of D FF



Flip-flop's memory property

Status		D
Present Q_t	Next Q_{t+1}	
0	0	0
0	1	1
1	0	0
1	1	1

A Side Bar

Excitation table of T FF

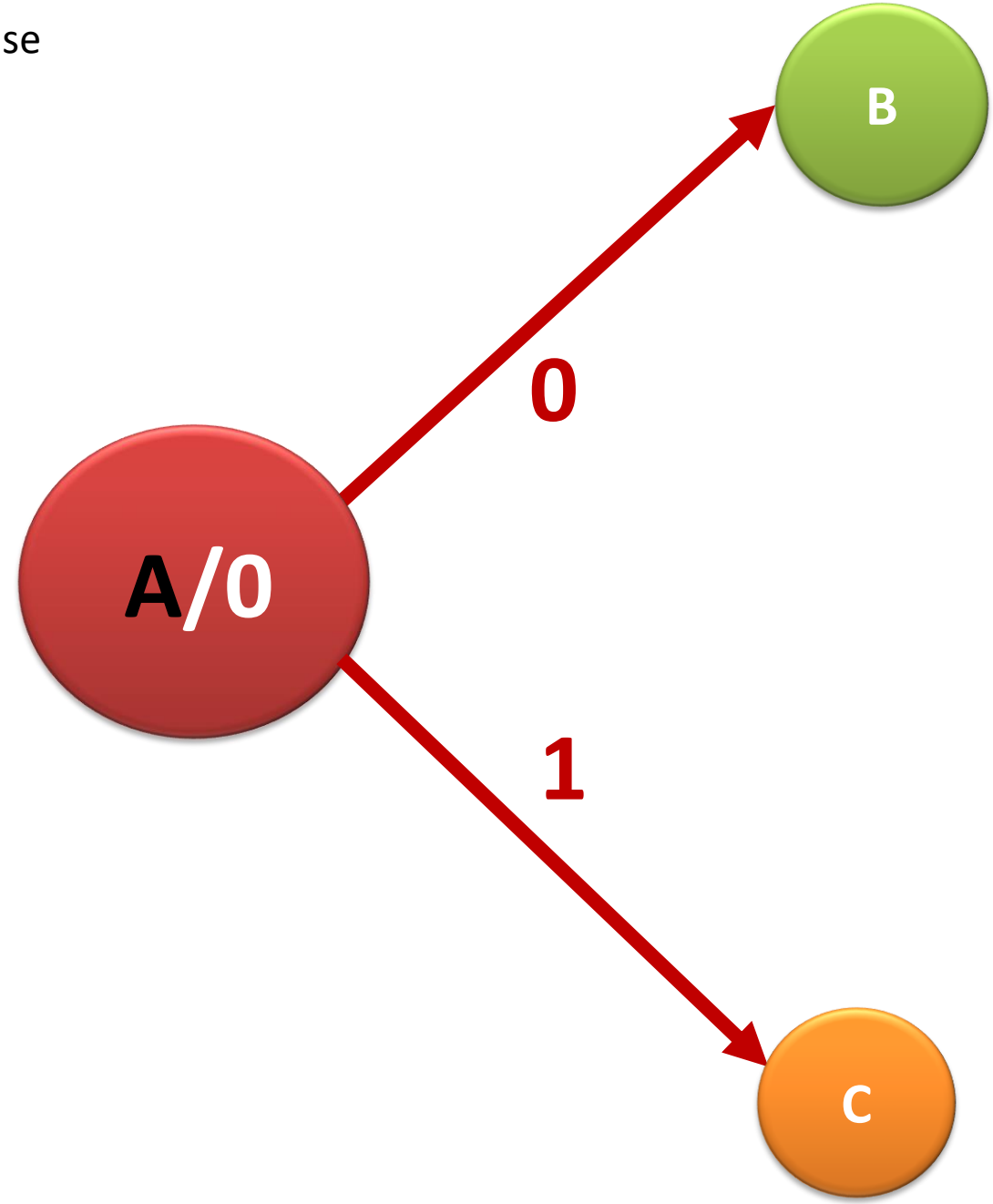
Status		T
Present Q_t	Next Q_{t+1}	
0	0	0
0	1	1
1	0	1
1	1	0

Excitation table of JK FF

Status		J	K
Present Q_t	Next Q_{t+1}		
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

State Transition Diagram
-Moore-

1 bit input case



01 or 10 detector

State Encoding (both state encoding table and output encoding table)

Encoding means we get rid of any words (alphabets) and translates them as binary numbers!!

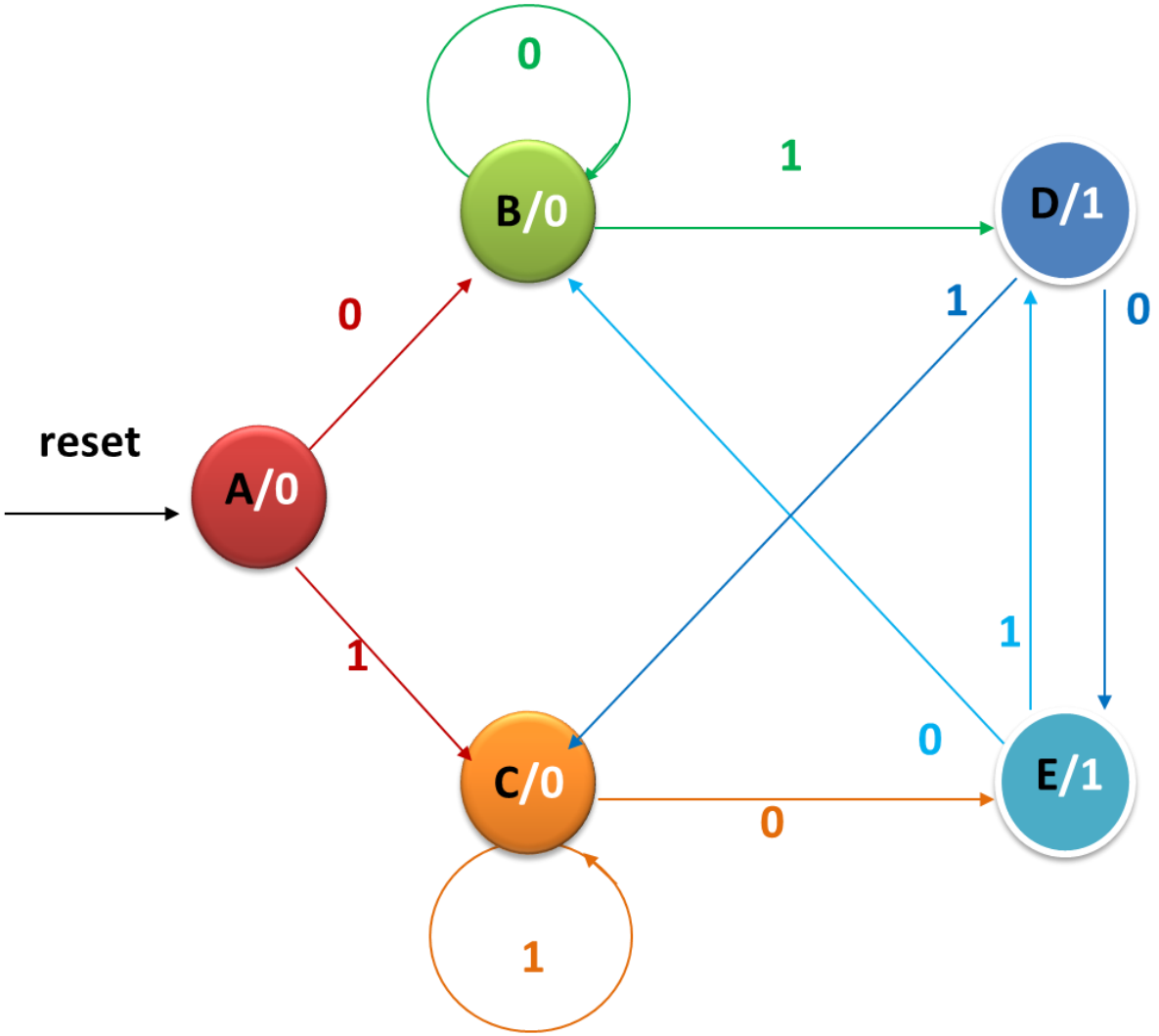
State	3-bits Encoding (D2 D1 D0)
A	000
B	001
C	010
D	011
E	100

State Encoding Table

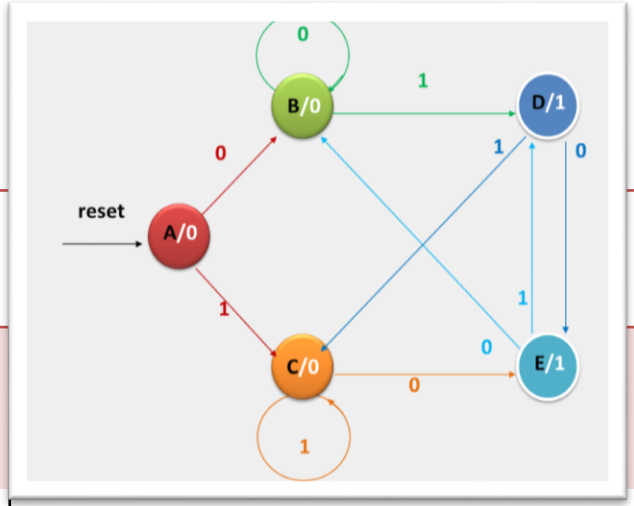
State	Output Y
000	0
001	0
010	0
011	1
100	1

Output Encoding Table

State Transition Table (both without and with encoding



Use this!!



Reset		Input	Next State, S'			Output
			$Q2_{next}$	$Q1_{next}$	$Q0_{next}$	
1	-	-	A			0
0	A	0	?			?
0	A	1				
0	B	0	?			?
0	B	1				
0	C	0	?			?
0	C	1				
0	D	0	?			?
0	D	1				
0	E	0	?			?
0	E	1				

State transition table without encoding

Then we encode each state column using our encoding

State
A
B
C
D
E

How many bits do we need?

	Current State, S			Input	Next State, S'			Output
Reset	Q2 (MSB)	Q1	Q0	I	$Q2_{next}$	$Q1_{next}$	$Q0_{next}$	Y
1	-	-		-	0	A	0	0
0	0	A	0	0	0	B	1	0
0	0	A	0	1	0	C	0	0
0	0	B	1	0	0	B	1	0
0	0	B	1	1	0	D	1	0
0	0	C	0	0	1	E	0	0
0	0	C	0	1	0	C	0	0
0	0	D	1	0	1	E	0	1
0	0	D	1	1	0	C	0	1
0	1	E	0	0	0	B	1	1
0	1	E	0	1	0	D	1	1

State transition table with encoding

Next State Logic Boolean (K-maps)

	Current State, S			Input	Nex
Reset	Q2 (MSB)	Q1	Q0	I	$Q2_{next}$
1	-			-	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0

Status		D
Present Q_t	Next Q_{t+1}	
0	0	0
0	1	1
1	0	0
1	1	1

Karnaugh map for D2

Q_0I \ Q_2Q_1	00	01	11	10
00	0	1	X	0
01	0	0	X	0
11	0	0	X	X
10	0	1	X	X

	Current State, S			Input	Next State, S	
Reset	Q2 (MSB)	Q1	Q0	I	$Q2_{next}$	$Q1_{next}$
1	-			-	0	0
0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	0	0
0	0	0	1	1	0	1
0	0	1	0	0	1	0
0	0	1	0	1	0	1
0	0	1	1	0	1	0
0	0	1	1	1	0	1
0	1	0	0	0	0	0
0	1	0	0	1	0	1

Status		D
Present Q_t	Next Q_{t+1}	
0	0	0
0	1	1
1	0	0
1	1	1

Karnaugh map for D1

Q_0I \ Q_2Q_1	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	X	X
10	0	0	X	X

	Current State, S			Input	Next State, S'		
Reset	Q2 (MSB)	Q1	Q0	I	$Q2_{next}$	$Q1_{next}$	$Q0_{next}$
1	-			-	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	0	1	1	1	0	1	0
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	1

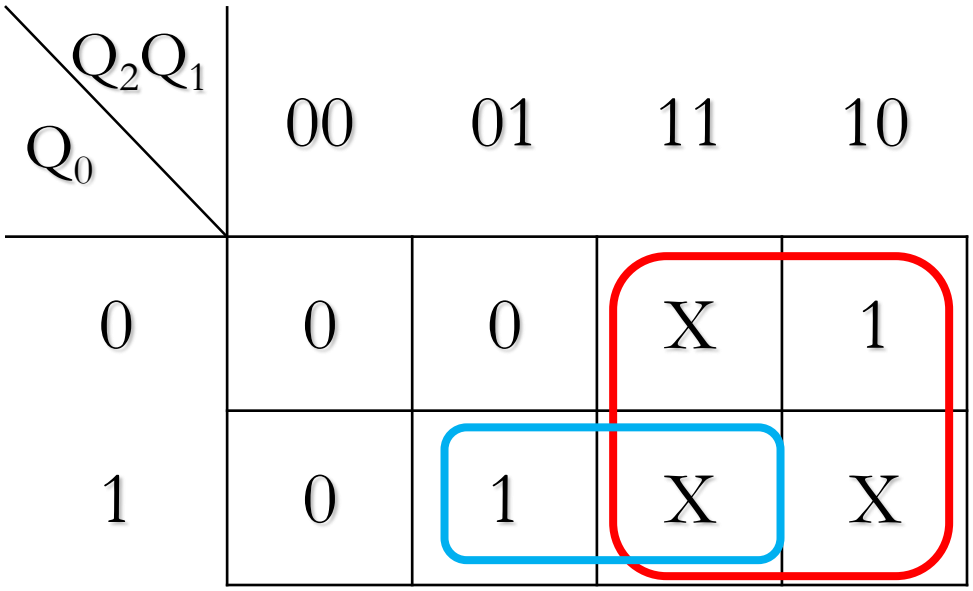
Status		D
Present Q_t	Next Q_{t+1}	
0	0	0
0	1	1
1	0	0
1	1	1

Karnaugh map for D0

Q_0I \ Q_2Q_1	00	01	11	10
00	1	0	X	1
01	0	0	X	1
11	1	0	X	X
10	1	0	X	X

Output Logic Boolean (with K-map and Boolean expression)

Current State, S			Input	Next State, S'			Output
Q2 (MSB)	Q1	Q0	I	$Q2_{next}$	$Q1_{next}$	$Q0_{next}$	Y
-			-	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	1	1
1	0	0	1	0	1	1	1



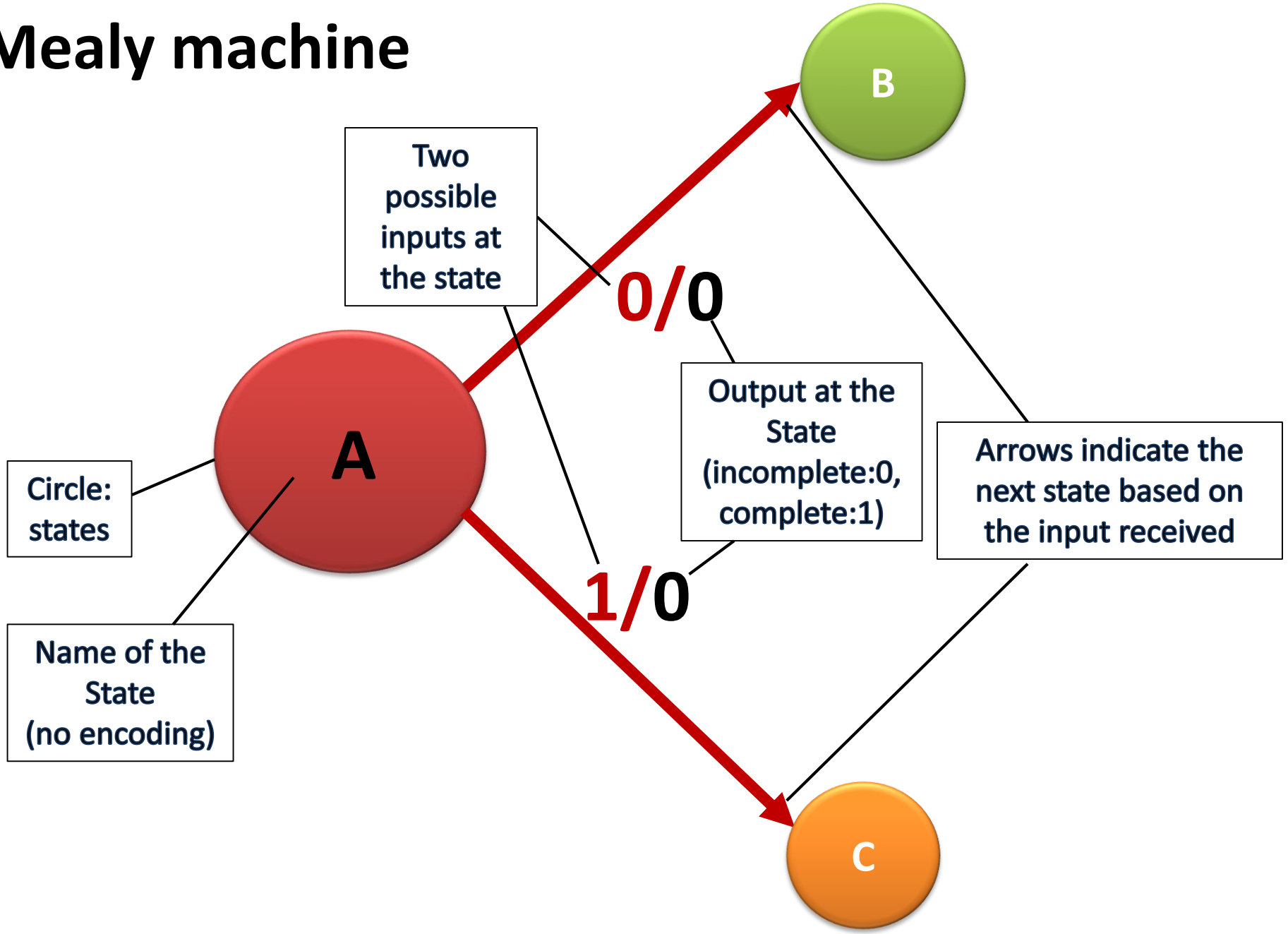
Karnaugh map for Output

Because it is a Moore machine, the output is not a function of input

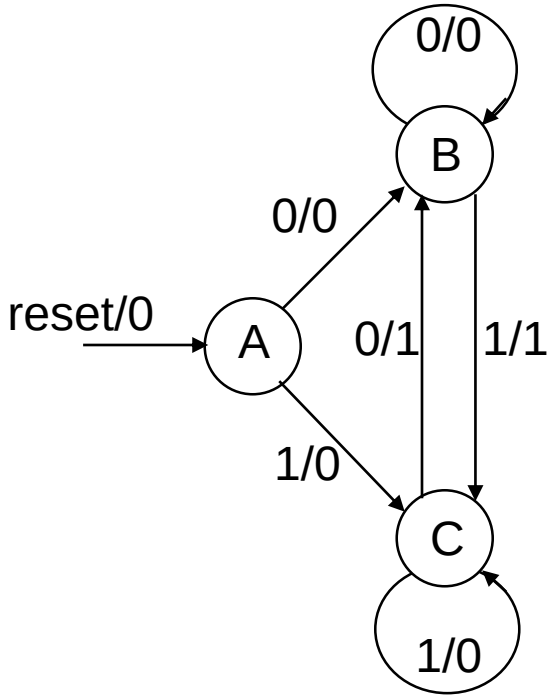
FYI

- Special attention to Preset and CLR pins on 7474 IC (datasheet please)
- Once you implement your circuit, give a slow clock (around 0.5-1 hz → 0.5-1 cycle per sec)
- Simulate your input 1001 (Vcc→GND→GND→Vcc)
- Watch your output response via LED

Mealy machine



FYI 2: The same example- Mealy implementation, you try



reset	input	current state (Q)	next state(Q')	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

	Current State, S		Input	Next State, S’		Output
Reset	Q1 (MSB)	Q0	I	$Q1_{next}$	$Q0_{next}$	Y
1	-		-	0	0	0
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	0	1
0	1	0	0	0	1	1
0	1	0	1	1	0	0

reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

D1=IQ1’Q0’+IQ1’Q0+IQ1Q0’=IQ1’(Q0’+Q0)+IQ1Q0’=I(Q1’+Q1Q0’)=**I(Q1’+Q0’)**

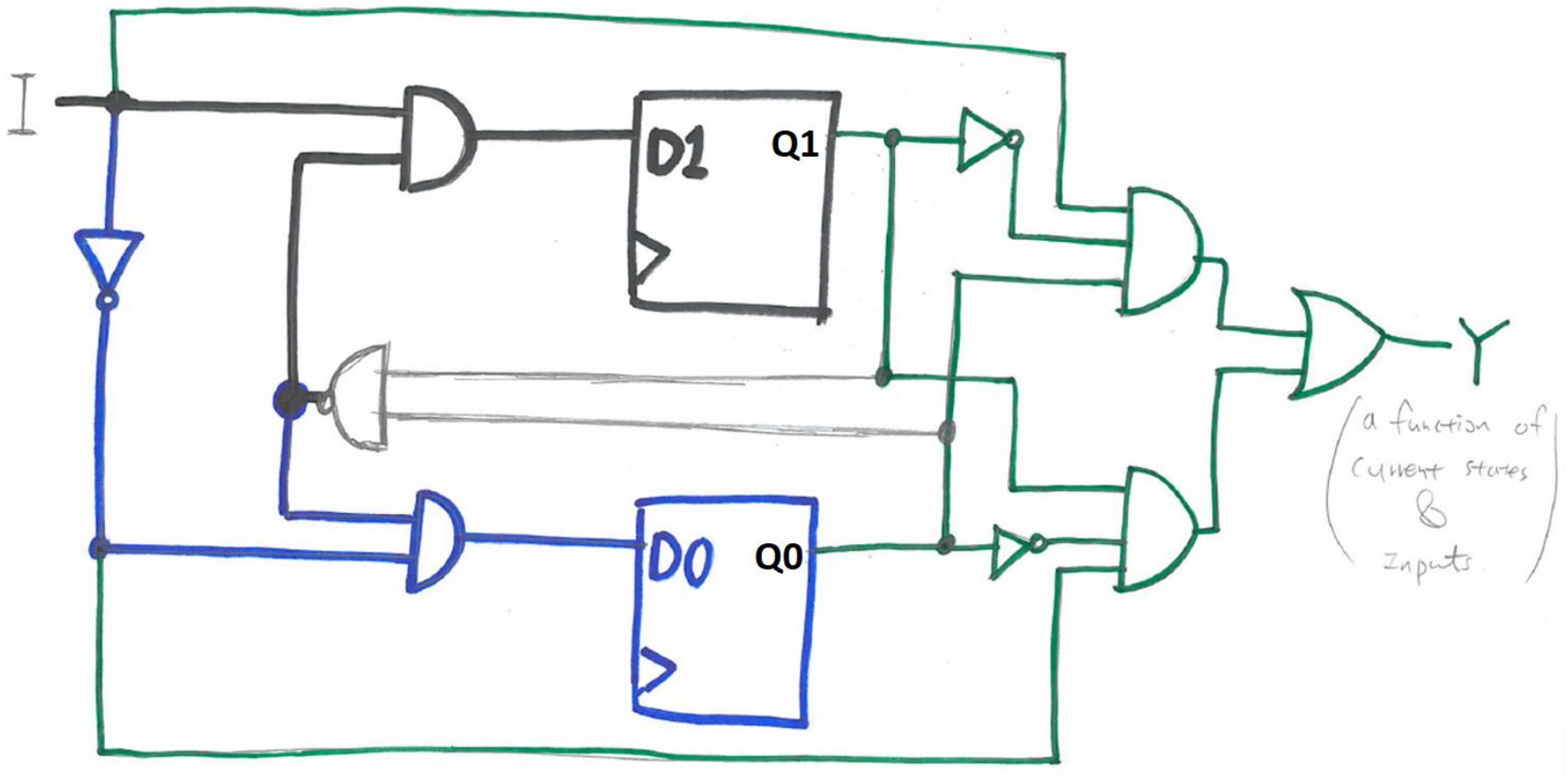
D0=I’Q1’Q0’+I’Q1’Q0+I’Q1Q0’=I’Q1’(Q0’+Q0)+I’Q1Q0’=I’(Q1’+Q1Q0’)=**I’(Q1’+Q0’)**

Y=Q1’Q0I+Q1Q0’I’

D1=I(Q1'+Q0')=I(Q1*Q2)'

D0=I'(Q1'+Q0')=I'(Q1*Q0)'

Y=Q1'Q0I+Q1Q0'I'



More on Finite state machine (Moore, Mealy)

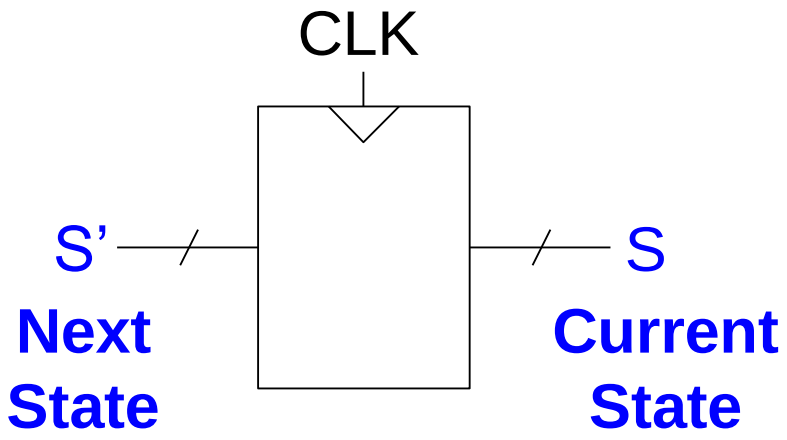
Edward F. Moore (1925– 2003)

George H. Mealy (1927 – 2010)

- George H. Mealy (1927 – 2010) worked at the Bell Laboratories in 1950's and was a Harvard University professor in 1970's http://boards.ancestry.com/surnames.mealy/56.1.1/mb.ashx

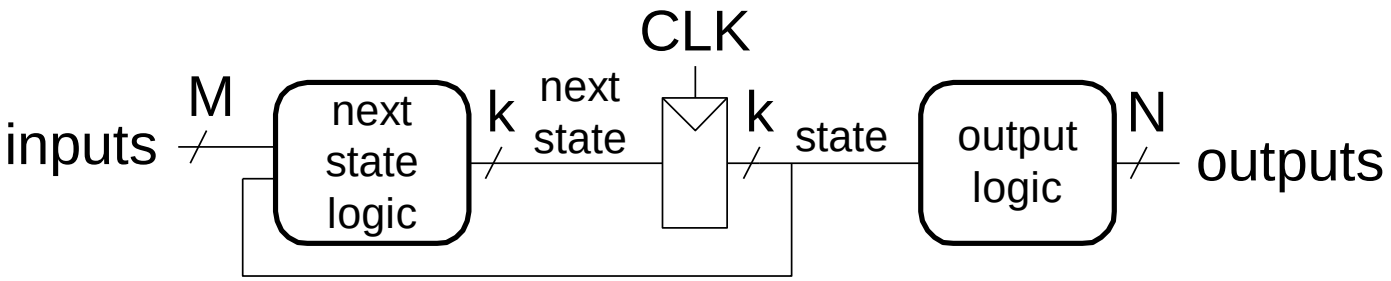
Finite State Machine (FSM)

- Consists of:
  - **State register**
    - Stores current state
    - Loads next state at clock edge
  - **Combinational logic**
    - Computes the next state
    - Computes the outputs

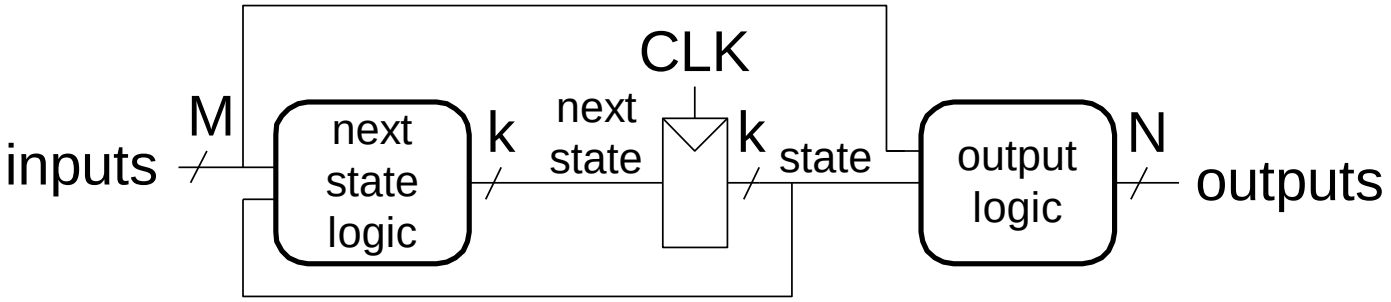


- **Next state** determined by current state and inputs
- Two types of finite state machines differ in **output logic**:
  - **Moore FSM:** outputs depend only on **current state**
  - **Mealy FSM:** outputs depend on **current state** *and* **inputs**

Moore FSM



Mealy FSM

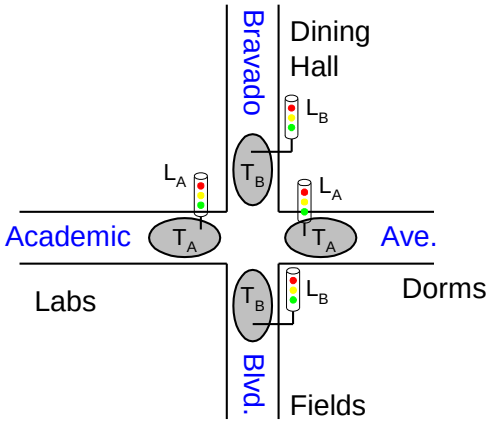


Mealy Machine	Moore Machine
Output depends both upon present state and present input.	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Mealy Machine.
Output changes at the clock edges.	Input change can cause change in output change as soon as logic is done.
Mealy machines react faster to inputs.	In Moore machines, more logic is needed to decode the outputs since it has more circuit delays.

A general procedure of finite state machine implementation (5 steps)

FSM Example 2 (page 124)

- Traffic light controller (using Moore)
  - Traffic sensors (**Inputs** or outputs?): T_A, T_B (TRUE when there's traffic)
  - Lights (Inputs or **outputs**?): L_A, L_B



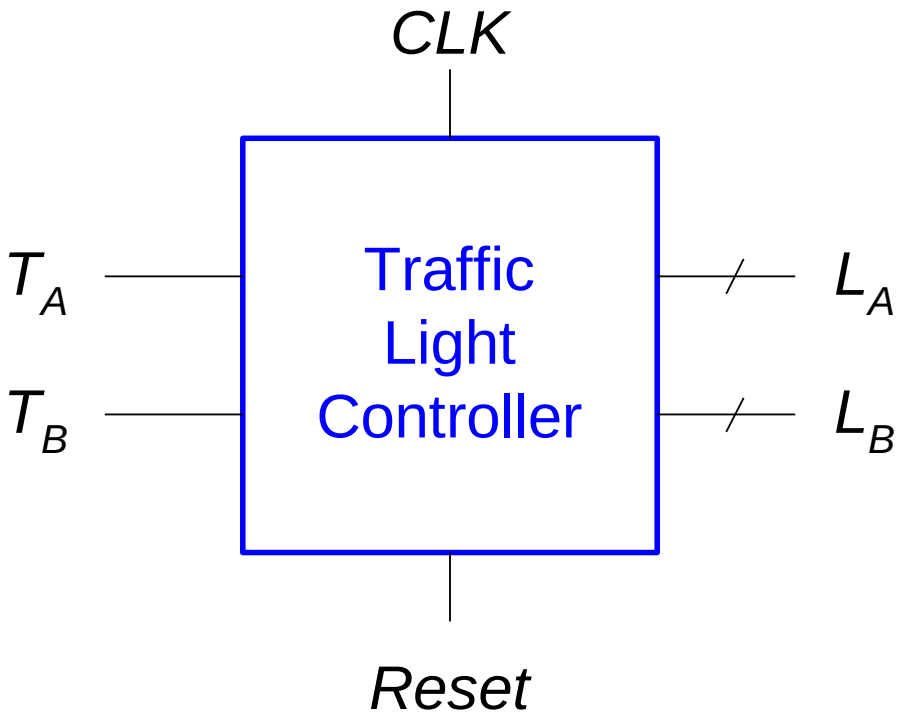
light before there are fatalities.

Ben decides to solve the problem with an FSM. He installs two traffic sensors, T_A and T_B, on Academic Ave. and Bravado Blvd., respectively. Each sensor indicates TRUE if students are present and FALSE if the street is empty. He also installs two traffic lights, L_A and L_B, to control traffic. Each light receives digital inputs specifying whether it should be green, yellow, or red. Hence, his FSM has two inputs, T_A and T_B, and two outputs, L_A and L_B. The intersection with lights and sensors is shown in Figure 3.23. Ben provides a clock with a 5-second period. On each clock tick (rising edge), the lights may change based on the traffic sensors. He also provides a reset button so that Physical Plant technicians can put the controller in a known initial state when they turn it on. Figure 3.24 shows a black box view of the state machine.

Ben's next step is to sketch the *state transition diagram*

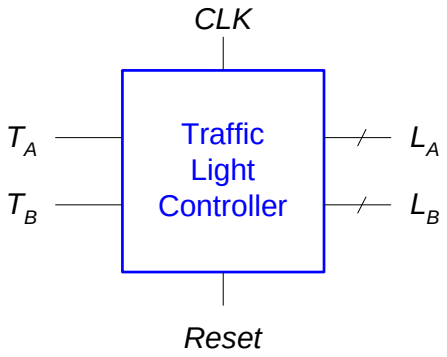
Black Box: in/out defs

- Inputs: *CLK*, *Reset*, T_A, T_B
- Outputs: L_A, L_B



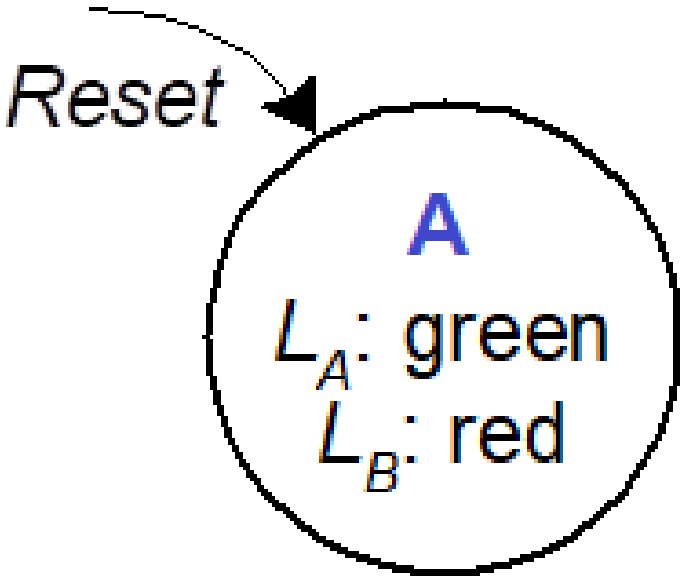
Ben's next step is to sketch the *state transition diagram*, shown in Figure 3.25, to indicate all the possible states of the system and the transitions between these states. When the system is reset, the lights are green on Academic Ave. and red on Bravado Blvd. Every 5 seconds, the controller examines the traffic pattern and decides what to do next. As long as traffic is present on Academic Ave., the lights do not change. When there is no longer traffic on Academic Ave., the light on Academic Ave. becomes yellow for 5 seconds before it turns red and Bravado Blvd.'s light turns green. Similarly, the Bravado Blvd. light remains green as long as traffic is present on the boulevard, then turns yellow and eventually red. ...sition diagram, circles represent states and arcs represent

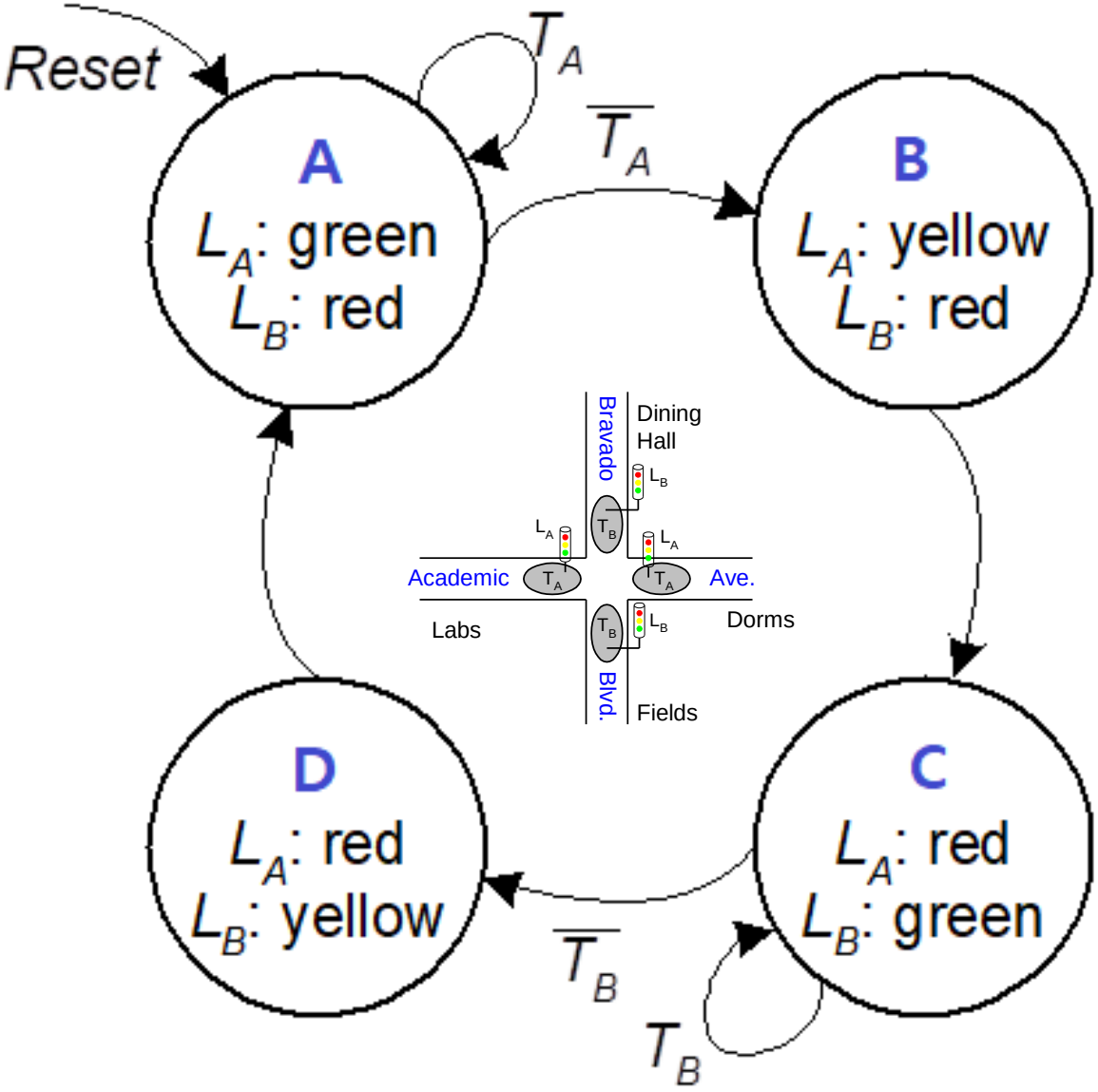
- Reset: Green on Academic/Red on Bravado
- If Yes traffic on Academic: Stay
- If No traffic on Academic: Yellow -> Red on Academic and Green on Bravado.
- If Yes traffic on Bravado: Stay
- After Green on Bravado, No traffic on Bravado: Yellow->Red on Bravado and Green on Academic



FSM State Transition Diagram

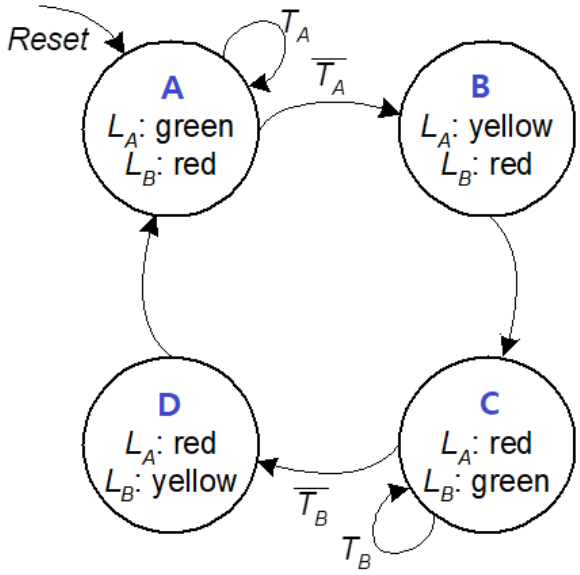
- **Moore FSM:** outputs labeled in each state (L_A and L_B)
- **States:** Circles (A,B,C, and D→ 4 kinds therefore we need 2 bits to represent all states, 2^2)
- **Transitions:** Arcs




Reset
T_A
$\overline{T_A}$
A
L_A: green
L_B: red
B
L_A: yellow
L_B: red
Bravado
Dining Hall
L_B
T_B
L_A
L_A
Academic
T_A
T_A
Ave.
Labs
T_B
L_B
Dorms
Blvd.
Fields
D
L_A: red
L_B: yellow
C
L_A: red
L_B: green
$\overline{T_B}$
T_B

FSM State Transition Table

Current State	Inputs		Next State
S	T_A	T_B	S'
A	0	X	
A	1	X	
B	X	X	
C	X	0	
C	X	1	
D	X	X	



FSM Encoded State Transition Table

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

State	Encoding
A	00
B	01
C	10
D	11

$S'_1 =$

$S'_0 =$

FSM Output Table

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

Output	Encoding
green	00
yellow	01
red	10

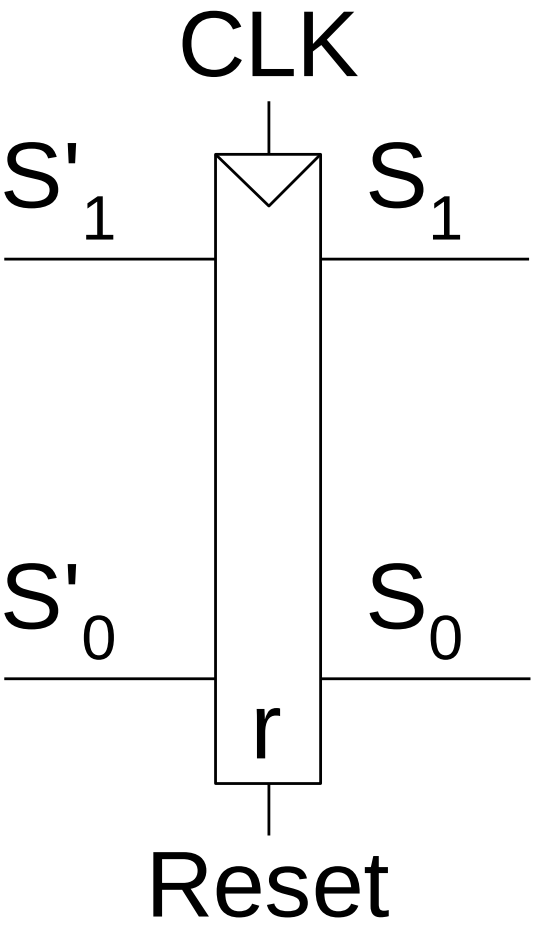
$$L_{A1} =$$

$$L_{A0} =$$

$$L_{B1} =$$

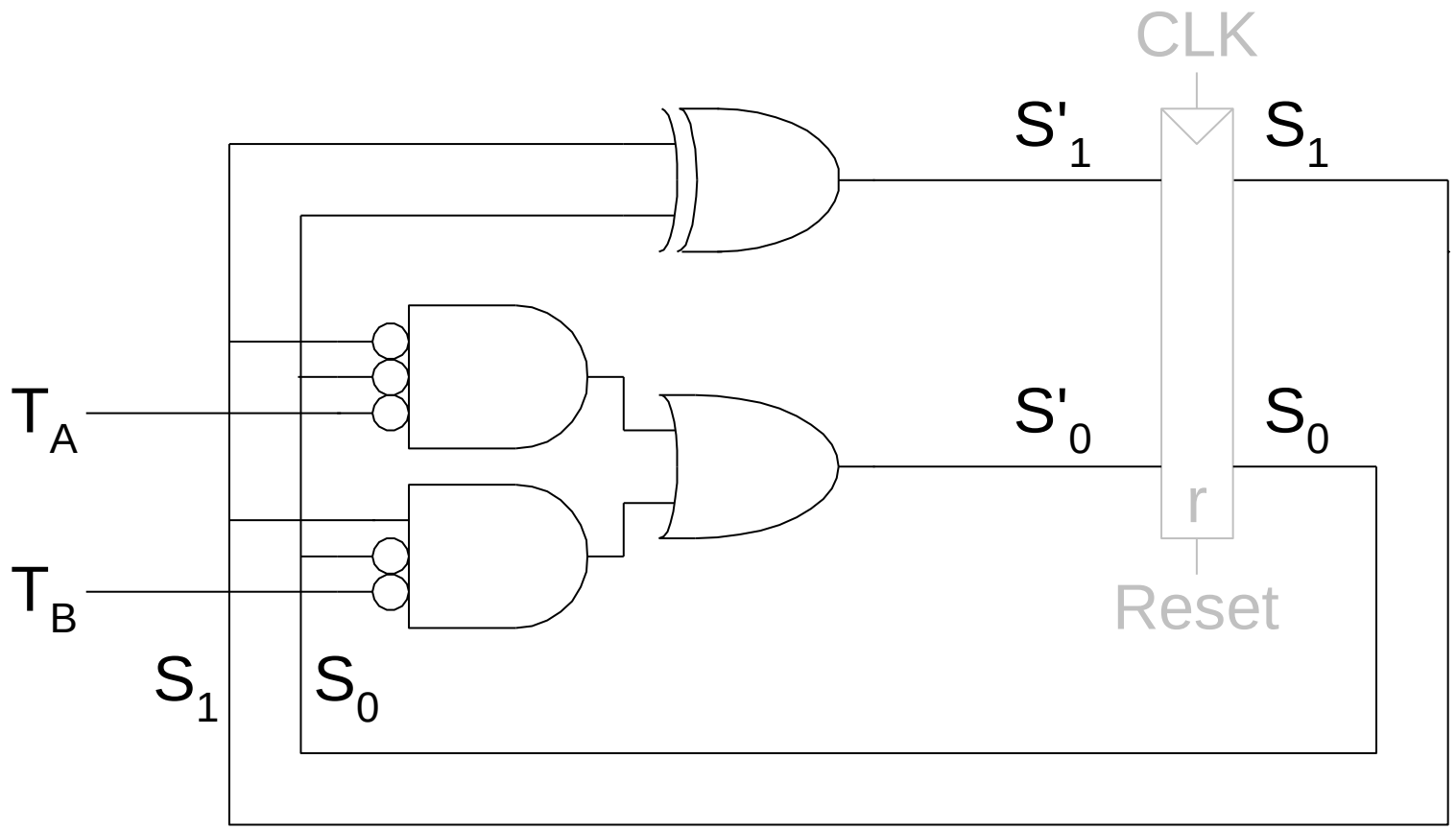
$$L_{B0} =$$

FSM Schematic: State Register



state register

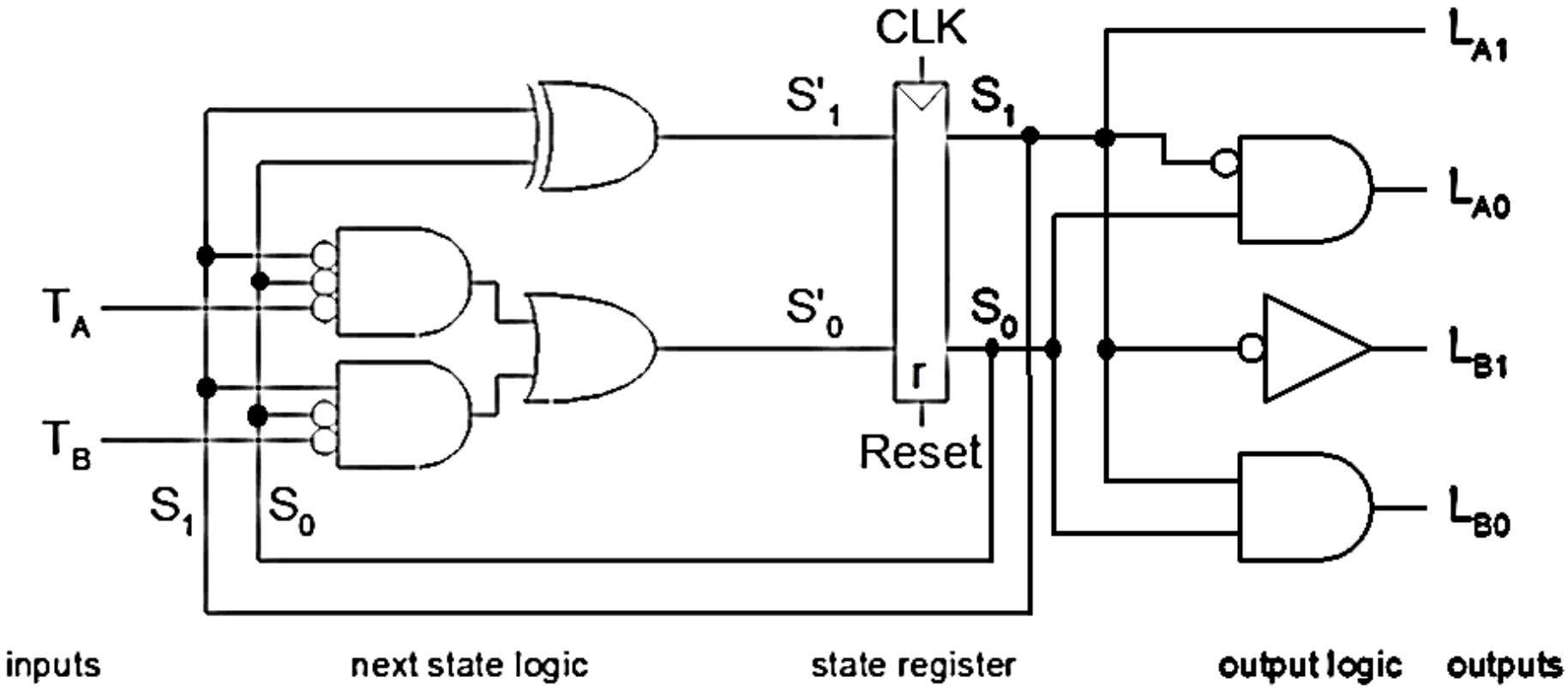
FSM Schematic: Next State Logic



inputs

next state logic

state register



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

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

$$L_{A1} = S_1$$

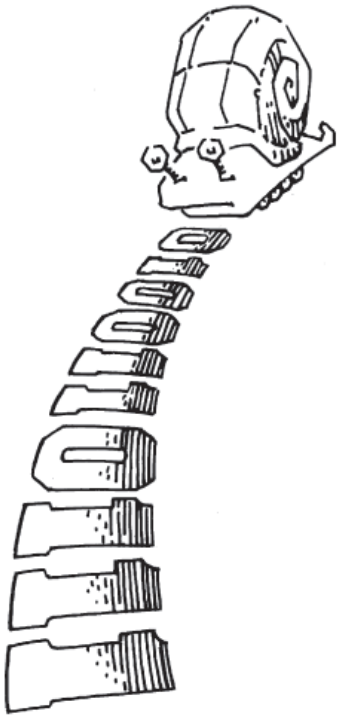
$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S_1}$$

$$L_{B0} = S_1 S_0$$

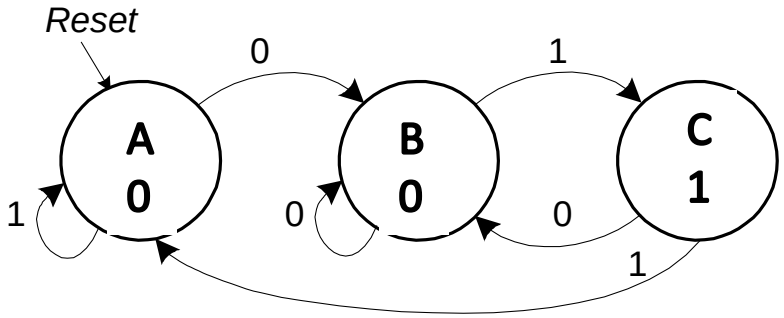
FSM Example 3

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it.
- The snail smiles whenever the last two digits it has crawled over are 01.
- Design Moore and Mealy FSMs of the snail

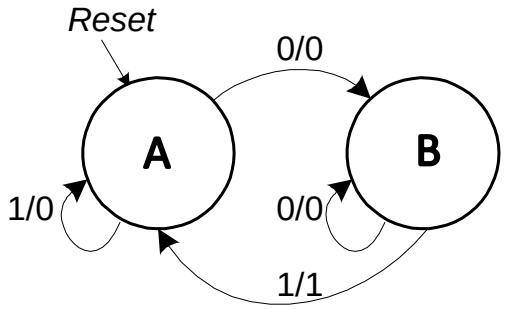


State Transition Diagrams

Moore FSM



Mealy FSM

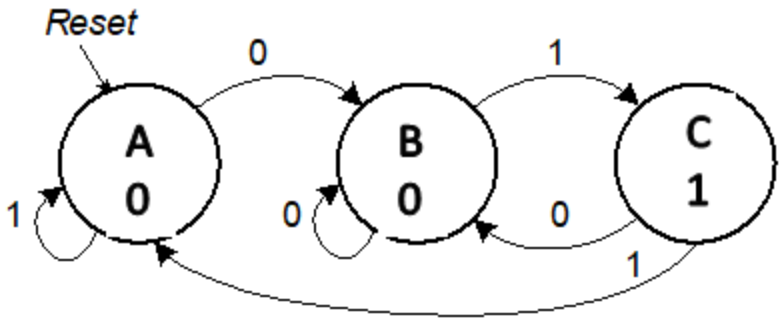


Mealy FSM: arcs indicate input/output

Moore FSM State Transition Table

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

Moore FSM



State	Encoding
A	00
B	01
C	10

$S_1' =$

$S_0' =$

?

Current State		Inputs	Next State	
S_1	S_0	X	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

S_1S_0 \ X	0	1
0 0	0	0
0 1	0	1
1 1	x	x
1 0	0	0

S_1S_0 \ X	0	1
0 0	1	0
0 1	1	0
1 1	x	x
1 0	1	0

Moore FSM Output Table

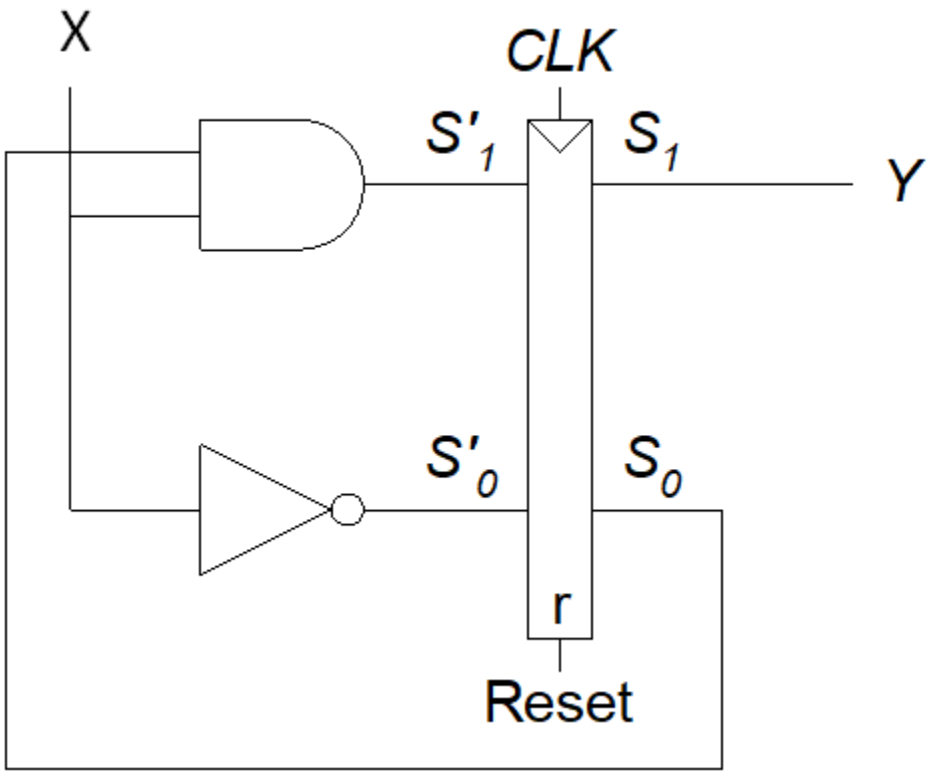
Current State S_1	Current State S_0	Output Y
0	0	
0	1	
1	0	

S_1 \ S_0	0	1
0	0	0
1	1	x

Moore FSM Schematic

$S_1' = S_0 X$

$S_0' = \overline{X}$



$Y = S_1$

Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
S_0	X	S'_0	Y
0	0		
0	1		
1	0		
1	1		

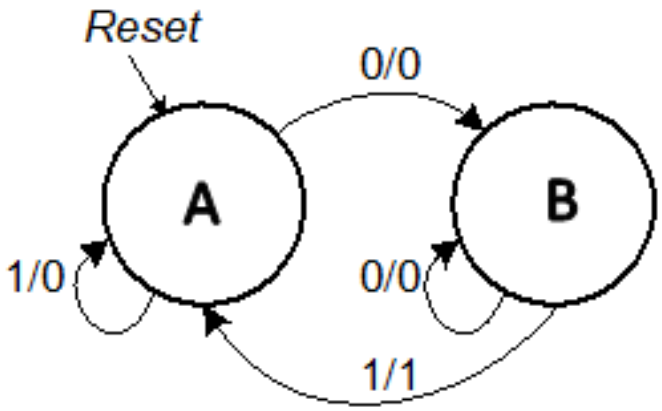
S_0 \ X	0	1
0	1	0
1	1	0

S_0'

S_0 \ X	0	1
0	0	0
1	0	1

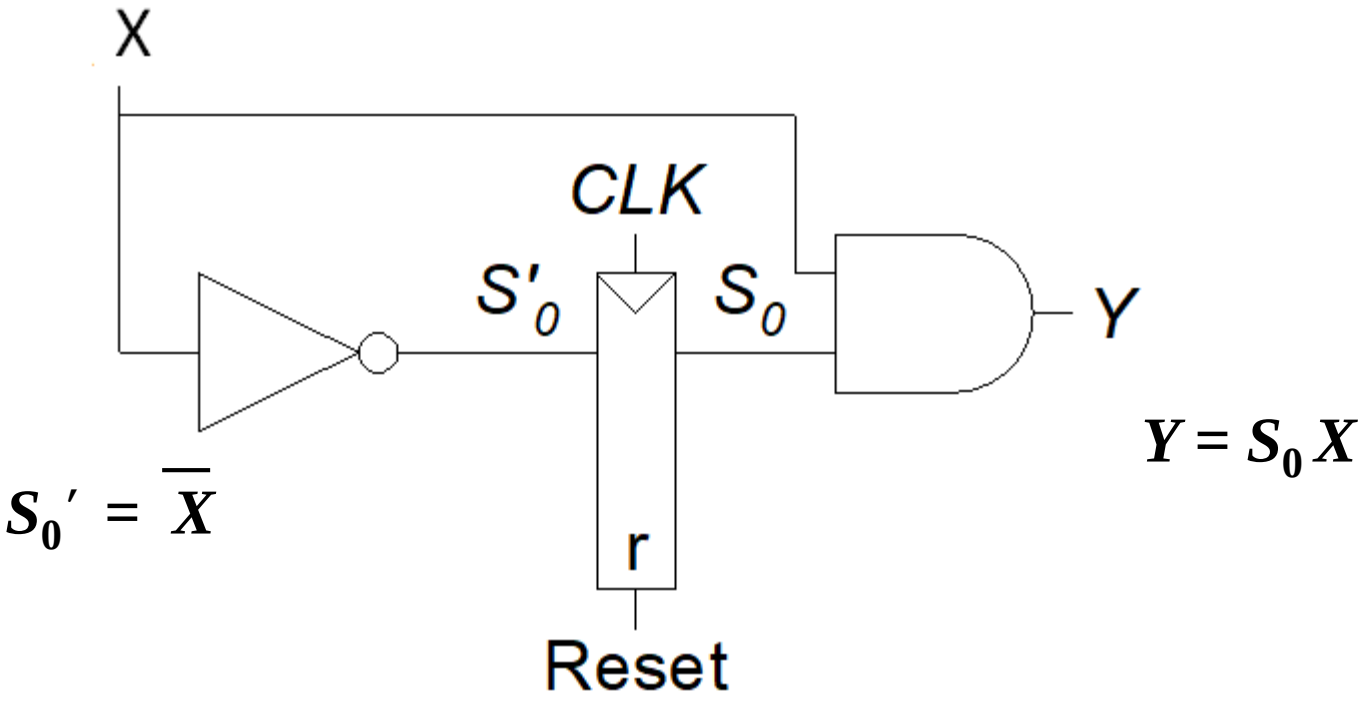
$Y =$

Mealy FSM



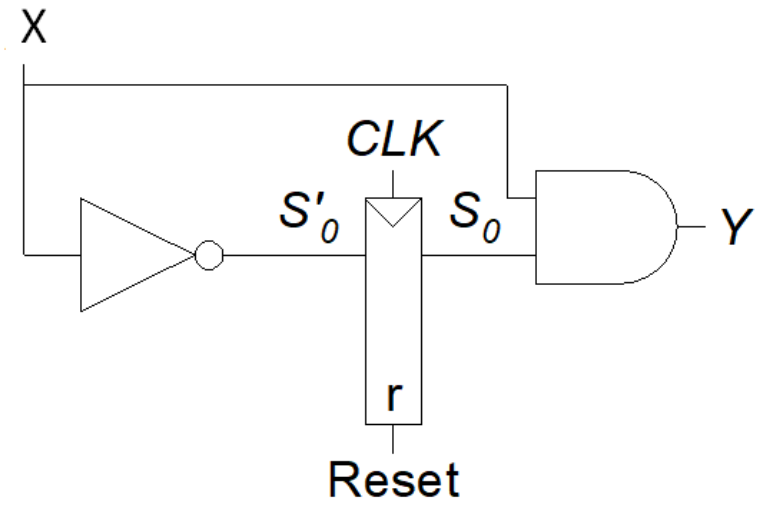
State	Encoding
A	0
B	1

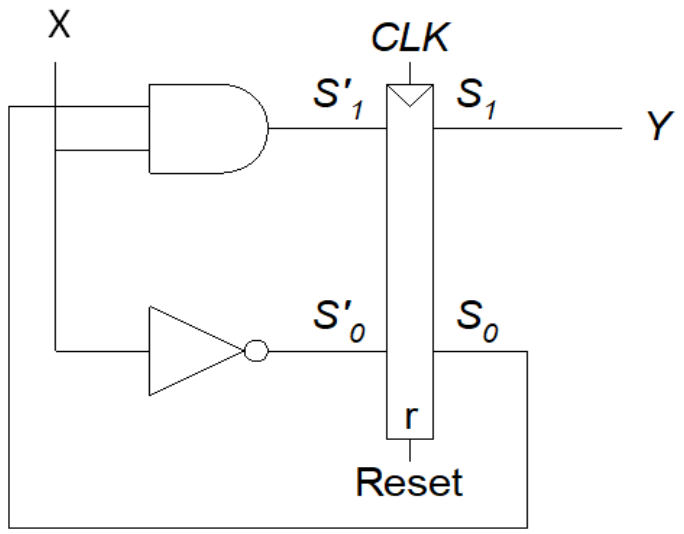
Mealy FSM Schematic



Check yourself!

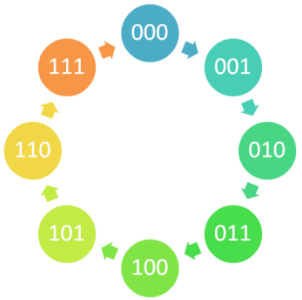
- By looking at circuit implementations, can you
  - Come up with a transition diagram?
  - Come up with next state's Boolean expression?
  - Determine whether it is Moore or Mealy FSM?
  - Determine output's Boolean expression?





Synchronous counter implementations steps:

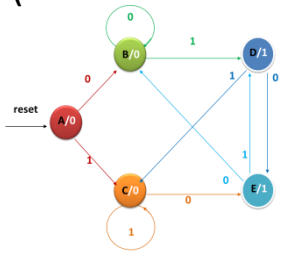
1. Decide the number of FFs (based on the states) and a kind of FF
   - i.e., 3 bits, JK FF
2. Excitation table of FF (relates Q_t and Q_{t+1})
3. State transition diagram
   - State encoding



4. Circuit excitation table
5. Obtain simplified equations using K map
6. Draw the logic diagram

A general procedure of finite state machine implementation

1. Defining Inputs/Outputs from given problems
2. State Transition Diagram
   - Decide the number of FFs (based on the states) and a kind of FF



3. Encoding (both states/inputs/outputs)
4. State Transition Table (both without and with encoding)
5. Next State Logic Boolean (with K-maps or Boolean simplification)
6. Output Logic Boolean (with K-map (or Boolean simplification) and Boolean expression)
7. Draw the logic diagram

- Note that asynchronous counter has a different implementation concept
- It is a synchronous counter which makes you confused with in-general FSM
- Counters in general:
  - A popular subset of FSM
  - No input FSM
  - Therefore transition between each state is fixed (only one arrow)