CSE1003-Digital Logic Design

Module:5 Sequential Circuits-I

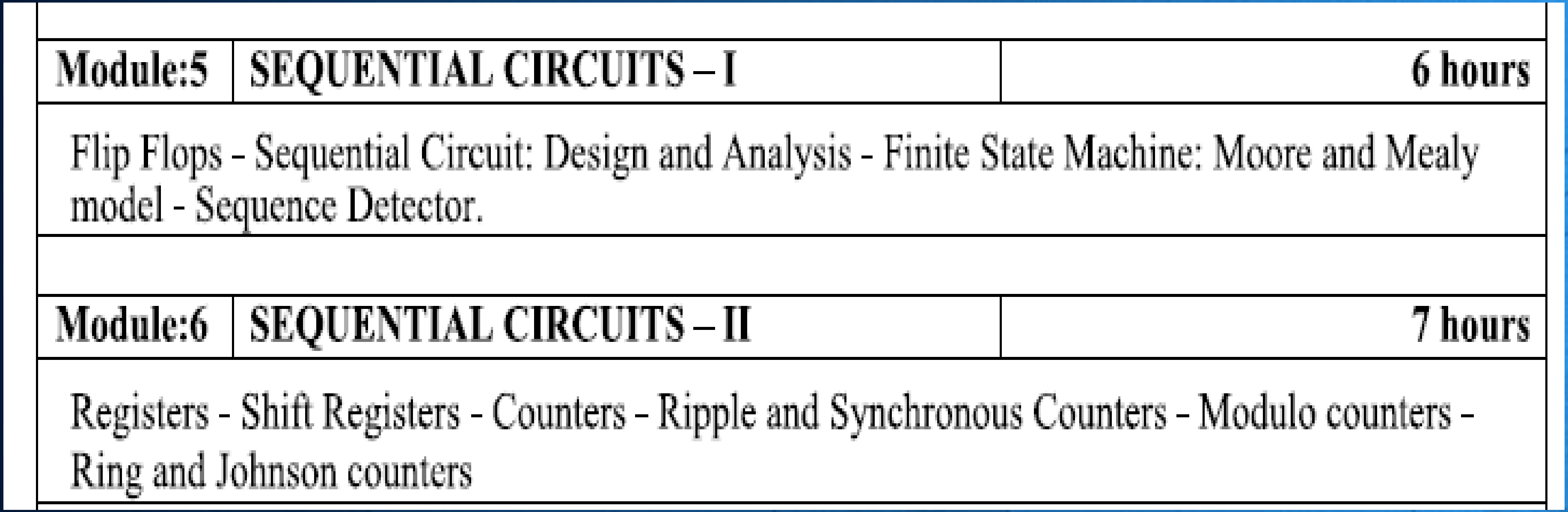
Part-2

Dr.Penchalaiah Palla

Dept. of Micro and Nanoelectronics

School of Electronics Engineering,VIT, Vellore

Module:5 Sequential Circuits-I



Module:5	**SEQUENTIAL CIRCUITS – I**	**6 hours**
Flip Flops - Sequential Circuit: Design and Analysis - Finite State Machine: Moore and Mealy model - Sequence Detector.		
Module:6	**SEQUENTIAL CIRCUITS – II**	**7 hours**
Registers - Shift Registers - Counters - Ripple and Synchronous Counters - Modulo counters - Ring and Johnson counters		

Introduction:Sequential Circuits

- Combinational
  - The outputs depend only on the current input values
  - It uses only logic gates
- Sequential
  - The outputs depend on the current and past input values
  - It uses logic gates and storage elements
  - Example
    - ✓ Vending machine
  - They are referred as finite state machines since they have a finite number of states

Sequential Logic Design with Flip-flops

Sequential Logic Design with Flip-flops

Introduction

- Sequential circuits has an extra dimension – *time*.
- Combinational circuit output depends only on the present inputs
- Sequential circuit output depends on the history of past inputs as well
- More powerful than combinational circuit, able to model situations that cannot be modeled by combinational circuits
- Building blocks of synchronous sequential logic circuits: *gates* and *flip-flops.*
- Flip-flops make up the *memory M* while the gates form one or more combinational subcircuits C_1, C_2, …, C_q.

Difference Between Analysis and Design

- *Analysis*: Starting from a circuit diagram, derive the state table or state diagram.
- *Characteristic tables* are used in analysis.

- *Design*: Starting from a set of specifications (in the form of state equations, state table, or state diagram), derive the logic circuit.
- *Excitation tables* are used in design.

Flip-flop Characteristic Tables

- Each type of flip-flop has its own behavior. The **characteristic tables** for the various types of flip-flops are shown below:

J	*K*	*Q(t+1)*	Comments
0	0	*Q(t)*	No change
0	1	0	Reset
1	0	1	Set
1	1	*Q(t)'*	Toggle

JK Flip-flop

S	*R*	*Q(t+1)*	Comments
0	0	*Q(t)*	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

SR Flip-flop

D	*Q(t+1)*	
0	0	Reset
1	1	Set

D Flip-flop

T	*Q(t+1)*	
0	*Q(t)*	No change
1	*Q(t)'*	Toggle

T Flip-flop

Sequential Circuit Analysis

- Given a sequential circuit diagram, analyze its behaviour by deriving its *state table* and hence its *state diagram*.
- Requires *state equations* to be derived for the flip-flop inputs, as well as *output functions* for the circuit outputs other than the flip-flops (if any).
- We use $A(t)$ and $A(t+1)$ to represent the present state and next state, respectively, of a flip-flop represented by A.
- Alternatively, we could simply use A and A^+ for the present state and next state respectively.

Sequential Circuit Analysis

- Example #1 (using D flip-flops):

State equations:

$A^{+} = A.x + B.x$

$B^{+} = A'.x$

Output function:

$y = (A + B).x'$

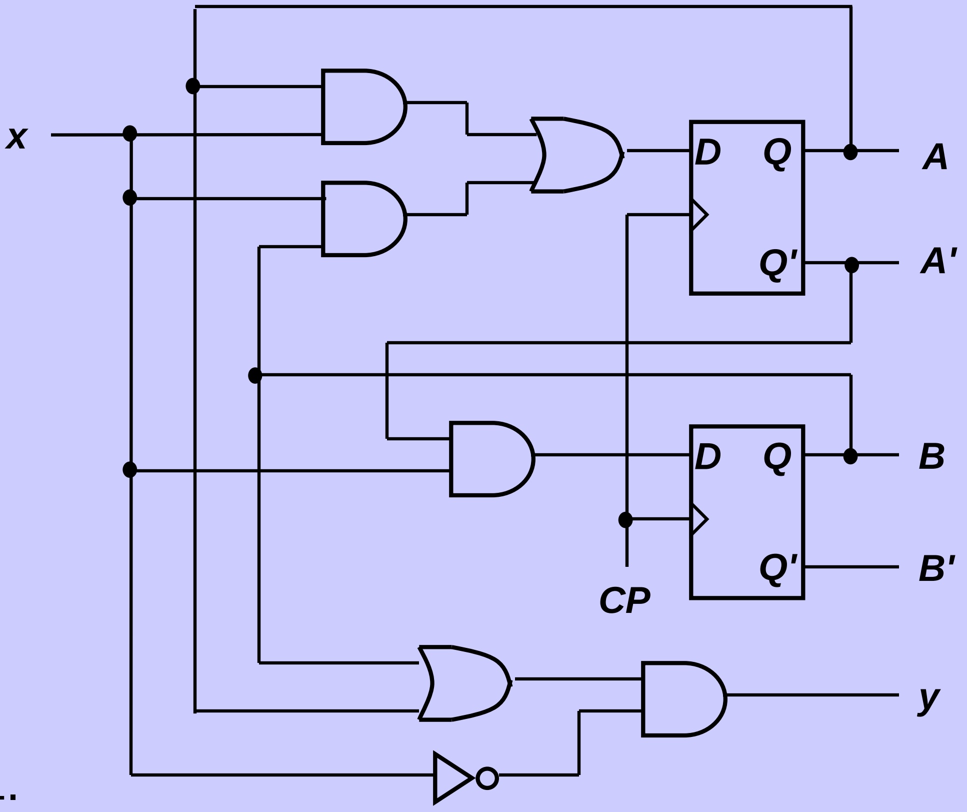


Figure 1.

Sequential Circuit Analysis

- From the *state equations* and *output function*, we derive the *state table*, consisting of all possible binary combinations of present states and inputs.
- State table
  - Similar to truth table.
  - Inputs and present state on the left side.
  - Outputs and next state on the right side.
- m flip-flops and n inputs $\rightarrow 2^{m+n}$ rows.

Sequential Circuit Analysis

- State table for the circuit of Figure 1:

  State equations:

  $A^+ = A.x + B.x$

  $B^+ = A'.x$

  Output function:

  $y = (A + B).x'$

Present State		Input	Next State		Output
A	*B*	*x*	A^+	B^+	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Sequential Circuit Analysis

- Alternate form of state table:

Present State		Input	Next State		Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
AB	A^+B^+	A^+B^+	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

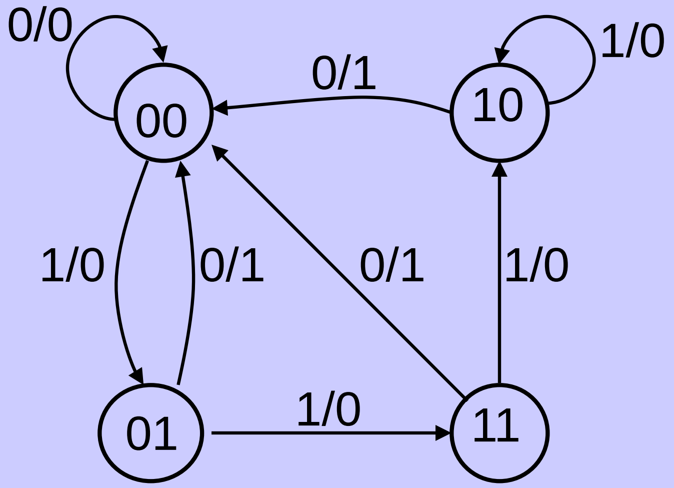
Sequential Circuit Analysis

- From the *state table*, we can draw the *state diagram*.
- State diagram
  - Each state is denoted by a circle.
  - Each arrow (between two circles) denotes a transition of the sequential circuit (a row in state table).
  - A label of the form *a*/*b* is attached to each arrow where *a* denotes the inputs while *b* denotes the outputs of the circuit in that transition.
- Each combination of the flip-flop values represents a state. Hence, m flip-flops $\rightarrow$ up to 2^m states.

Sequential Circuit Analysis

- State diagram of the circuit of Figure 1:

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
AB	A^+B^+	A^+B^+	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0



Flip-flop Input Functions

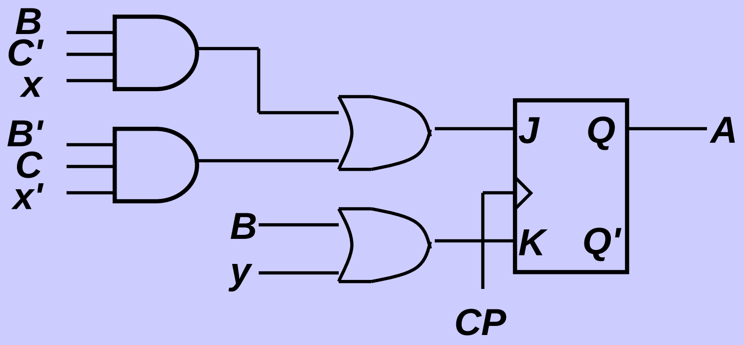
- The outputs of a sequential circuit are functions of the present states of the flip-flops and the inputs. These are described algebraically by the *circuit output functions*.
  - In Figure 1: $y = (A + B).x'$
- The part of the circuit that generates inputs to the flip-flops are described algebraically by the *flip-flop input functions* (or *flip-flop input equations*).
- The flip-flop input functions determine the next state generation.
- From the flip-flop input functions and the characteristic tables of the flip-flops, we obtain the next states of the flip-flops.

Flip-flop Input Functions

- Example: circuit with a JK flip-flop.
- We use 2 letters to denote each flip-flop input: the first letter denotes the input of the flip-flop (*J* or *K* for JK flip-flop, *S* or *R* for SR flip-flop, *D* for D flip-flop, *T* for T flip-flop) and the second letter denotes the name of the flip-flop.

$JA = B.C'.x + B'.C.x'$

$KA = B + y$



Flip-flop Input Functions

- In Figure 1, we obtain the following state equations by observing that $Q^+ = DQ$ for a D flip-flop:

$A^+ = A.x + B.x$ (since $DA = A.x + B.x$)

$B^+ = A'.x$ (since $DB = A'.x$)

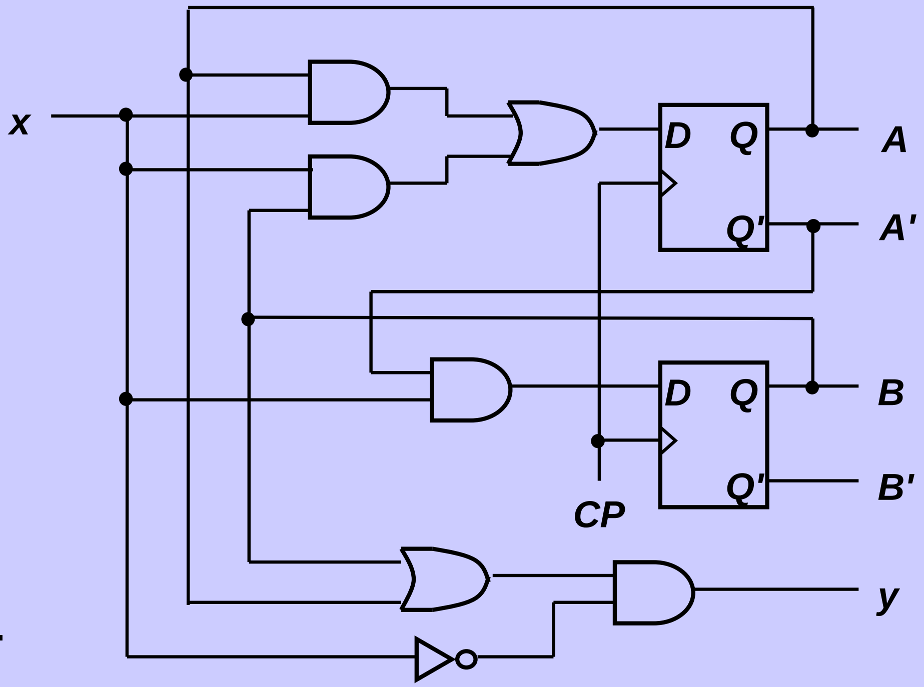


Figure 1.

Analysis: Example #2

- Given Figure 2, a sequential circuit with two JK flip-flops *A* and *B*, and one input *x*.

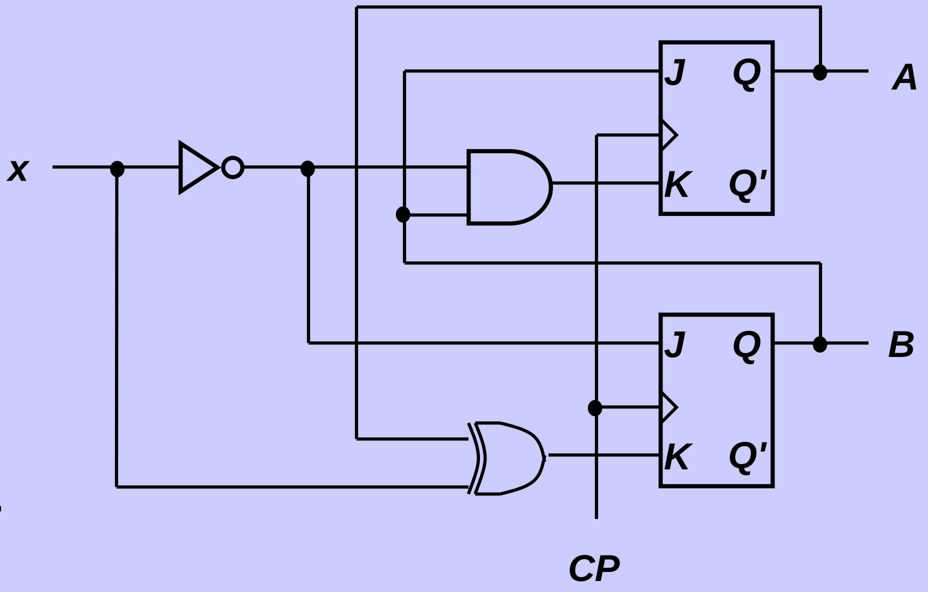


Figure 2.

- Obtain the flip-flop input functions from the circuit:

$JA = B$ $\qquad$ $JB = x'$

$KA = B.x'$ $\qquad$ $KB = A'.x + A.x' = A \oplus x$

Analysis: Example #2

- Flip-flop input functions:

  $JA = B$ $\qquad$ $JB = x'$

  $KA = B.x'$ $\qquad$ $JB = A'.x + A.x' = A \oplus x$

- Fill the state table using the above functions, knowing the characteristics of the flip-flops used.

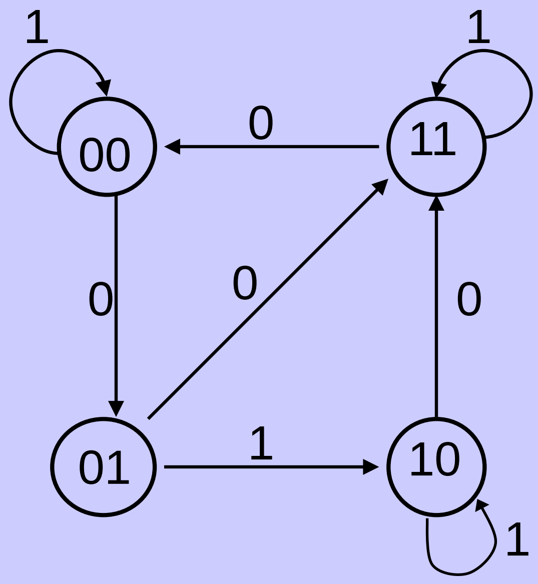
J	K	Q(t+1)	Comments
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q(t)'	Toggle

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

Analysis: Example #2

- Draw the state diagram from the state table.

Present state		Input	Next state		Flip-flop inputs			
A	*B*	*x*	A^+	B^+	*JA*	*KA*	*JB*	*KB*
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0



Analysis: Example #3

- Derive the state table and state diagram of the following circuit.

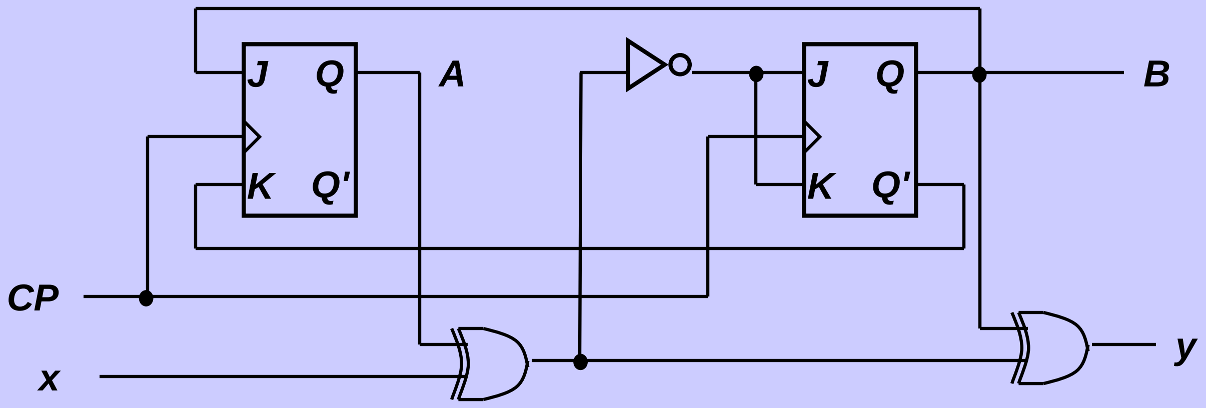


Figure 3.

- Flip-flop input functions:

  $JA = B$ $\quad\quad JB = KB = (A \oplus x)' = A.x + A'.x'$

  $KA = B'$

Analysis: Example #3

- Flip-flop input functions:

  $JA = B$ $\quad\quad JB = KB = (A \oplus x)' = A.x + A'.x'$

  $KA = B'$

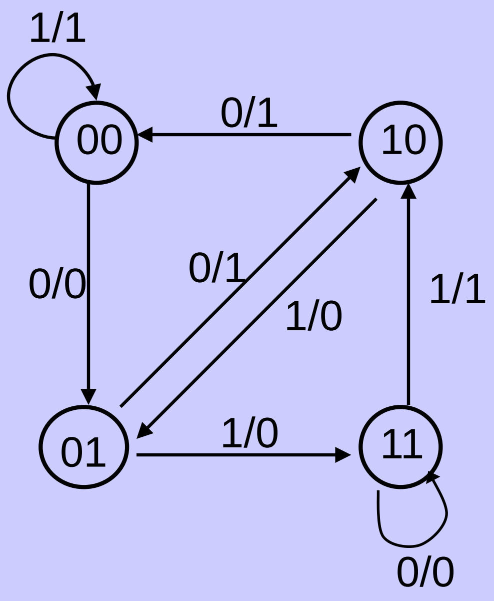
- State table:

Present state		Input	Next state		Output	Flip-flop inputs			
A	B	x	A^+	B^+	y	JA	KA	JB	KB
0	0	0	0	1	0	0	1	1	1
0	0	1	0	0	1	0	1	0	0
0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0
1	0	1	0	1	0	0	1	1	1
1	1	0	1	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1	1

Analysis: Example #3

- State diagram:

Present state		Input	Next state		Output	Flip-flop inputs			
A	*B*	*x*	A^+	B^+	*y*	*JA*	*KA*	*JB*	*KB*
0	0	0	0	1	0	0	1	1	1
0	0	1	0	0	1	0	1	0	0
0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0
1	0	1	0	1	0	0	1	1	1
1	1	0	1	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1	1



Flip-flop Excitation Tables

- *Design*: Starting from a set of specifications (in the form of state equations, state table, or state diagram), derive the logic circuit.
- *Excitation tables* are used in design.

Flip-flop Excitation Tables

- *Excitation tables*: given the required transition from present state to next state, determine the flip-flop input(s).

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

SR Flip-flop

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

D Flip-flop

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

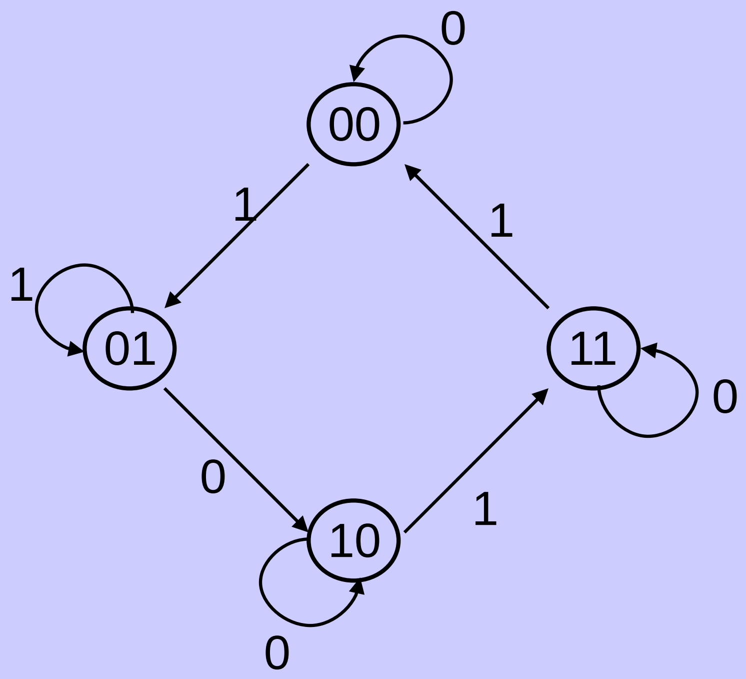
T Flip-flop

Sequential Circuit Design

- Design procedure:
  - Start with circuit specifications – description of circuit behaviour.
  - Derive the state table.
  - Perform state reduction if necessary.
  - Perform state assignment.
  - Determine number of flip-flops and label them.
  - Choose the type of flip-flop to be used.
  - Derive circuit excitation and output tables from the state table.
  - Derive circuit output functions and flip-flop input functions.
  - Draw the logic diagram.

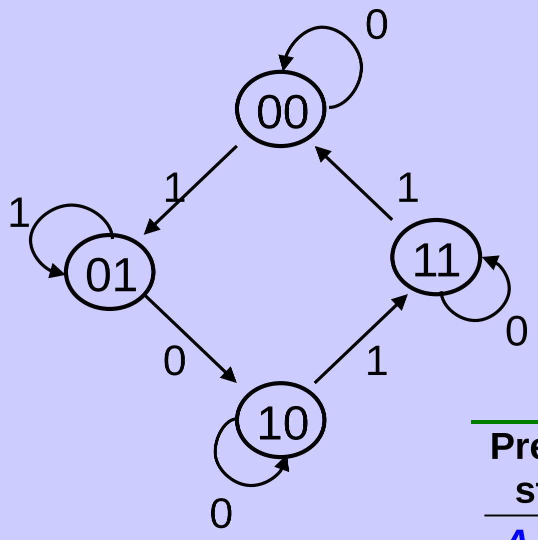
Design: Example #1

- Given the following state diagram, design the sequential circuit using JK flip-flops.



Design: Example #1

- Circuit state/excitation table, using JK flip-flops.



Present State	Next State	
	x=0	x=1
AB	A^+B^+	A^+B^+
00	00	01
01	10	01
10	10	11
11	11	00

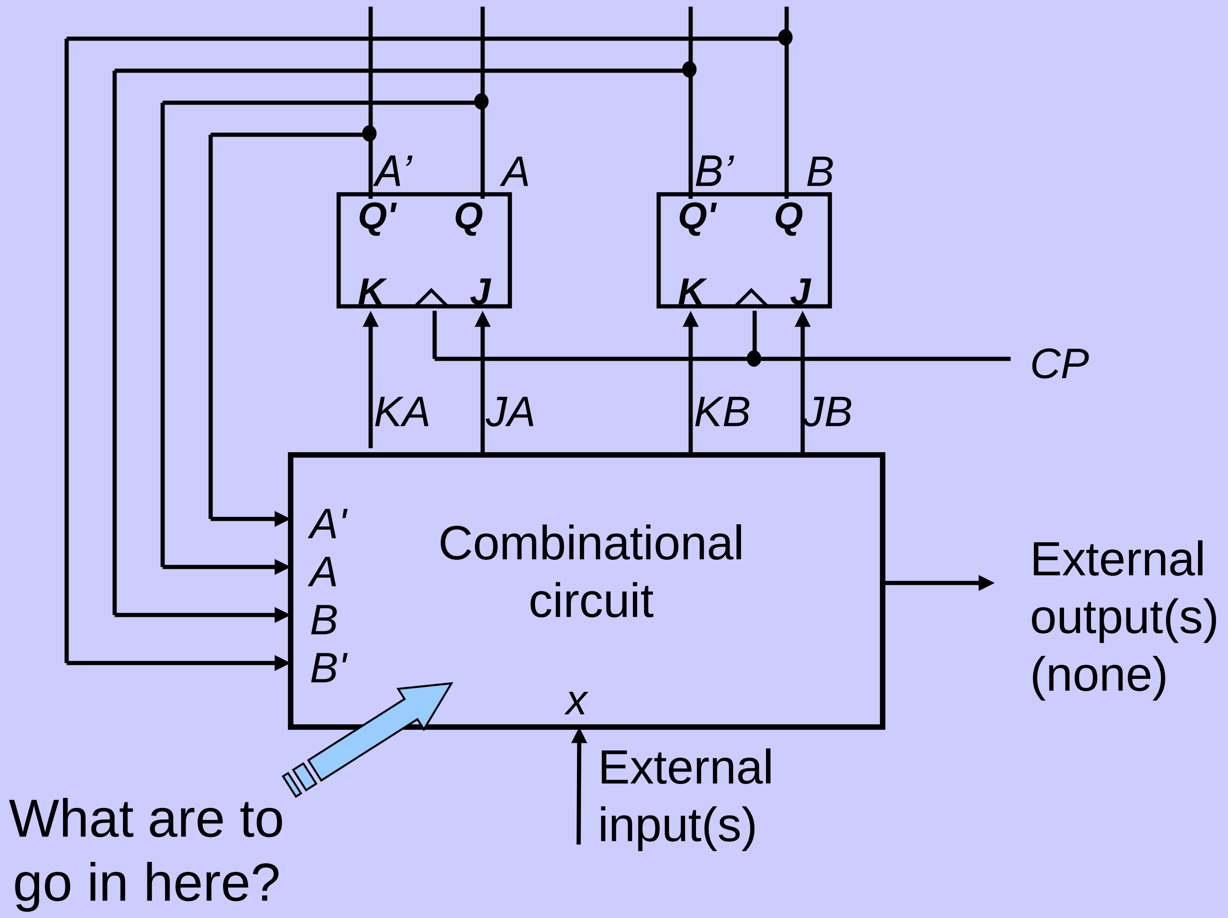
Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop's excitation table.

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Design: Example #1

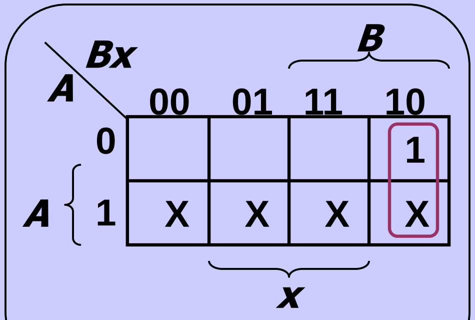
- Block diagram.



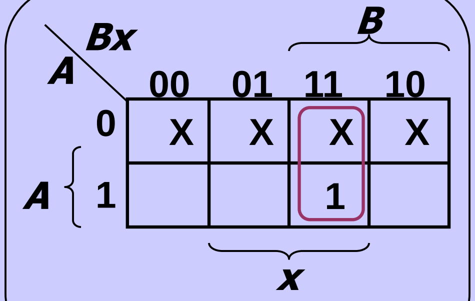
Design: Example #1

- From state table, get flip-flop input functions.

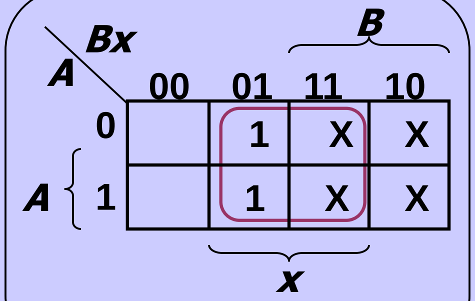
Present state		Input	Next state		Flip-flop inputs			
A	*B*	*x*	A^+	B^+	*JA*	*KA*	*JB*	*KB*
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1



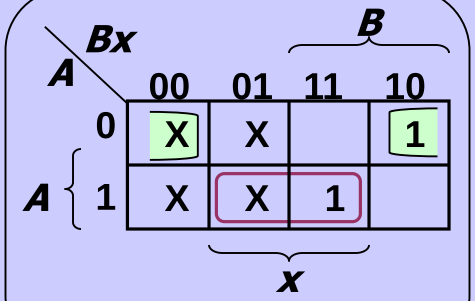
JA = B.x'



KA = B.x



JB = x



KB = (A ⊕ x)'

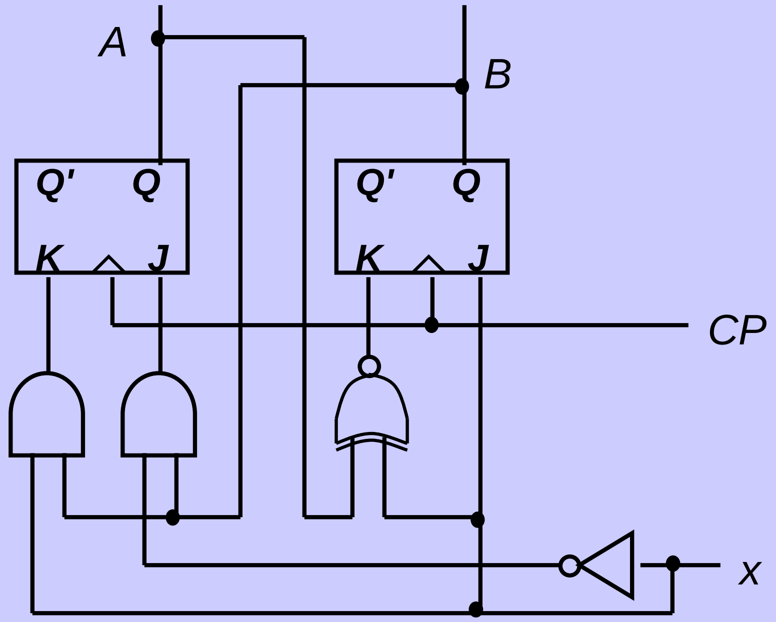
Design: Example #1

- Flip-flop input functions.

  $JA = B.x'$ $\qquad JB = x$

  $KA = B.x$ $\qquad KB = (A \oplus x)'$

- Logic diagram:



Design: Example #2

- Design, using D flip-flops, the circuit based on the state table below. (Exercise: Design it using JK flip-flops.)

Present state		Input	Next state		Output
A	*B*	*x*	A^+	B^+	*y*
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

Design: Example #2

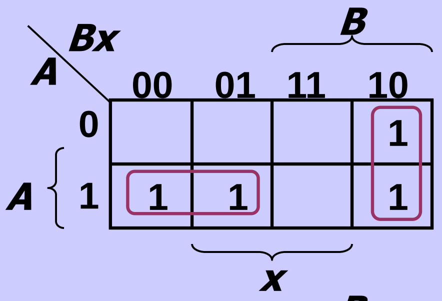
- Determine expressions for flip-flop inputs and the circuit output *y*.

Present state		Input	Next state		Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

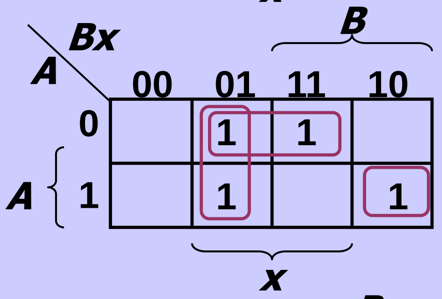
$DA(A, B, x) = \Sigma\ m(2,4,5,6)$

$DB(A, B, x) = \Sigma\ m(1,3,5,6)$

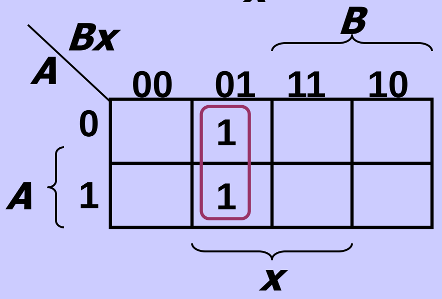
$y(A, B, x) = \Sigma\ m(1,5)$



DA = A.B' + B.x'



DB = A'.x + B'.x + A.B.x'



y = B'.x

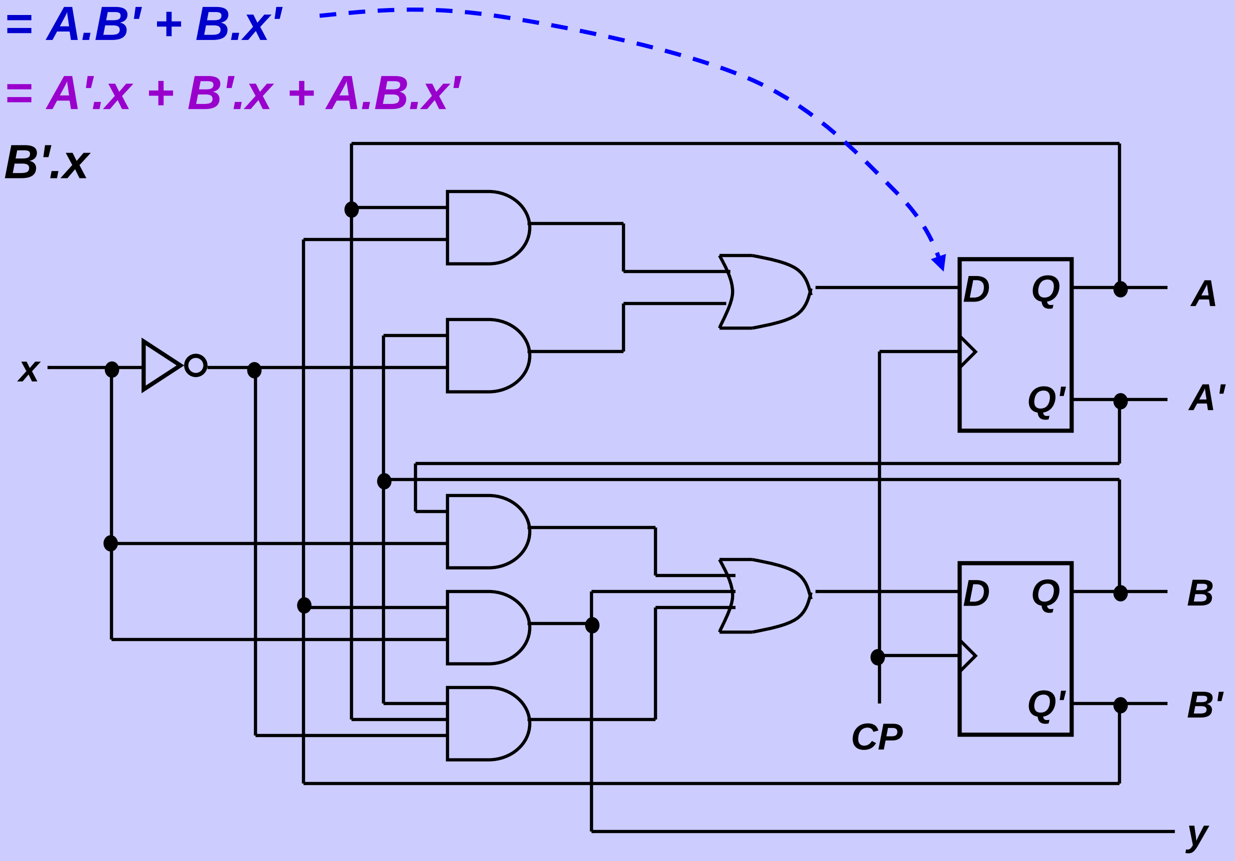
Design: Example #2

- From derived expressions, draw logic diagram:

DA = A.B' + B.x'

DB = A'.x + B'.x + A.B.x'

y = B'.x



Design: Example #3

- Design with unused states.

Present state			Input	Next state			Flip-flop inputs						Output
A	*B*	*C*	*x*	A^+	B^+	C^+	*SA*	*RA*	*SB*	*RB*	*SC*	*RC*	*y*
0	0	1	0	0	0	1	0	X	0	X	X	0	0
0	0	1	1	0	1	0	0	X	1	0	0	1	0
0	1	0	0	0	1	1	0	X	X	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	X	0
0	1	1	0	0	0	1	0	X	0	1	X	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	X	0	0	X	1	0	0
1	0	0	1	1	0	0	X	0	0	X	0	X	1
1	0	1	0	0	0	1	0	1	0	X	X	0	0
1	0	1	1	1	0	0	X	0	0	X	0	1	1

Given these

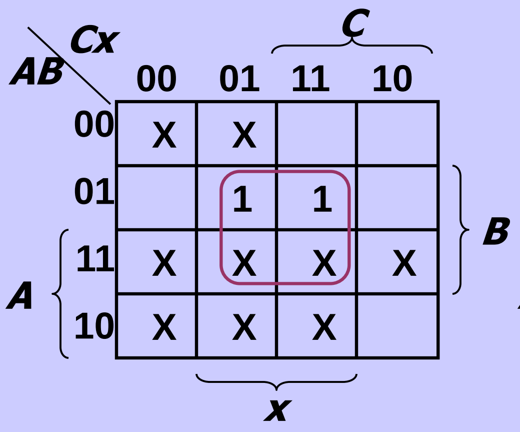
Derive these

Unused state 000:

0	0	0	0	X	X	X	X	X	X	X	X	X	X
0	0	0	1	X	X	X	X	X	X	X	X	X	X

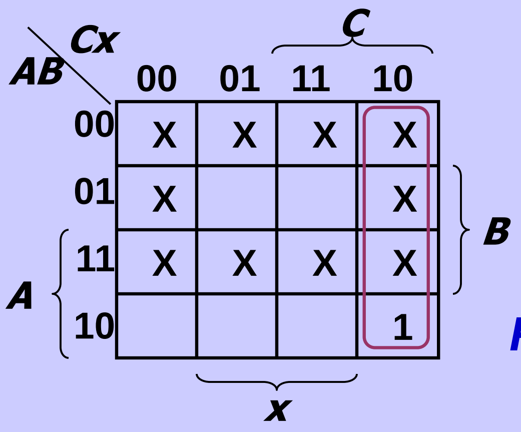
Design: Example #3

- From state table, obtain expressions for flip-flop inputs.



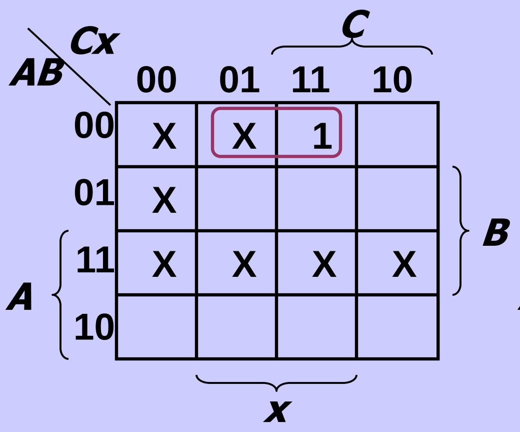
AB \ Cx	00	01	11	10
00	X	X		
01		1	1	
11	X	X	X	X
10	X	X	X	

SA = B.x



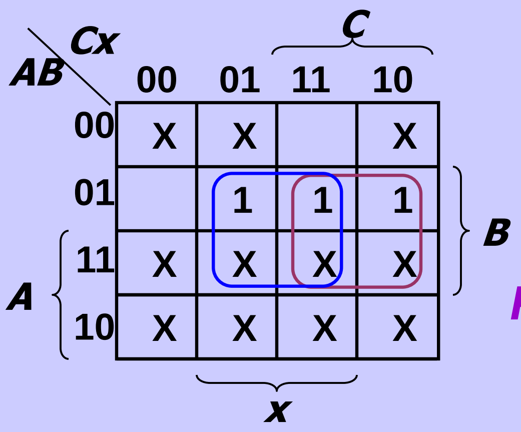
AB \ Cx	00	01	11	10
00	X	X	X	X
01	X			X
11	X	X	X	X
10				1

RA = C.x'



AB \ Cx	00	01	11	10
00	X	X	1	
01	X			
11	X	X	X	X
10				

SB = A'.B'.x

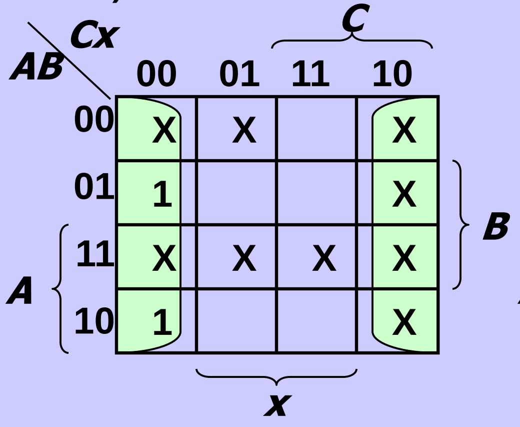


AB \ Cx	00	01	11	10
00	X	X		X
01		1	1	1
11	X	X	X	X
10	X	X	X	X

RB = B.C + B.x'

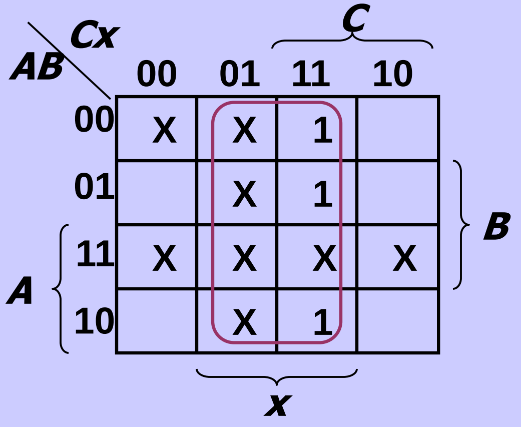
Design: Example #3

- From state table, obtain expressions for flip-flop inputs (cont'd).



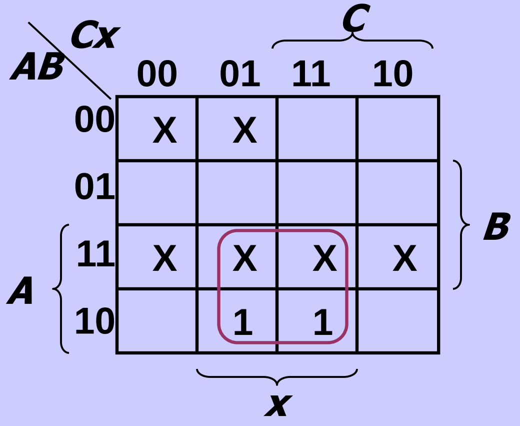
AB \ Cx	00	01	11	10
00	X	X		X
01	1			X
11	X	X	X	X
10	1			X

SC = *x'*



AB \ Cx	00	01	11	10
00	X	X	1	
01		X	1	
11	X	X	X	X
10		X	1	

RC = *x*



AB \ Cx	00	01	11	10
00	X	X		
01				
11	X	X	X	X
10		1	1	

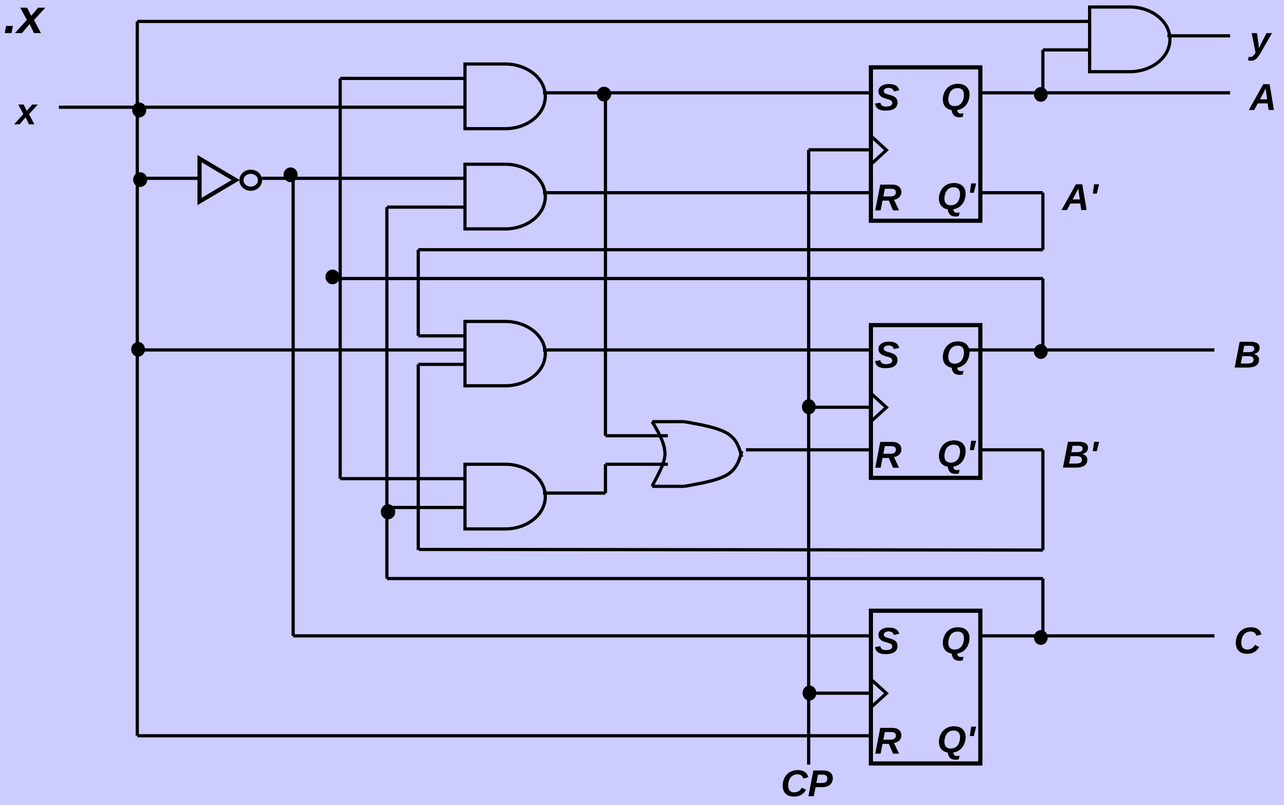
y = *A.x*

Design: Example #3

- From derived expressions, draw logic diagram:

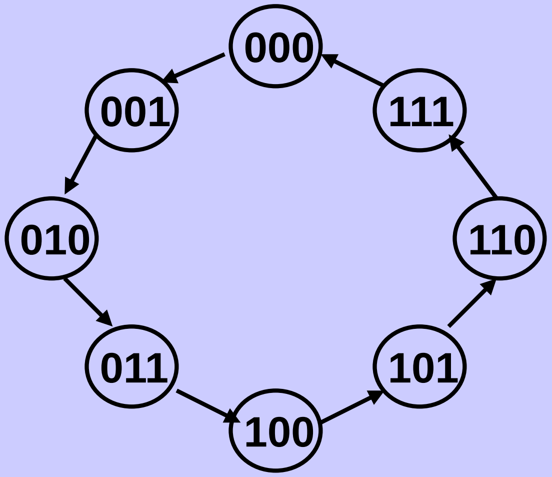
SA = B.x	*SB = A'.B'.x*	*SC = x'*
RA = C.x'	*RB = B.C + B.x'*	*RC = x*

y = A.x



Design of Synchronous Counters

- Counter: a sequential circuit that cycles through a sequence of states.
- Binary counter: follows the *binary sequence*. An *n*-bit binary counter (with *n* flip-flops) counts from 0 to 2^n-1.
- Example 1: A 3-bit binary counter (using T flip-flops).

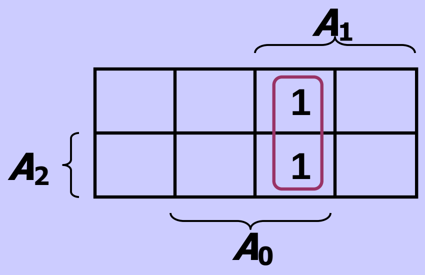

$$\begin{tikzcd} 000 \arrow[r] & 001 \arrow[r] & 010 \arrow[r] & 011 \arrow[r] & 100 \arrow[r] & 101 \arrow[r] & 110 \arrow[r] & 111 \arrow[lllllll, bend left] \end{tikzcd}$$

Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

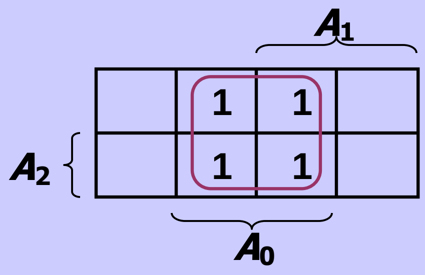
Design of Synchronous Counters

- 3-bit binary counter (cont'd).

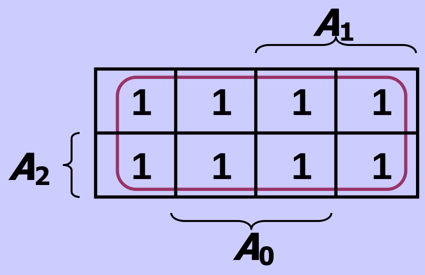
Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



$TA_2 = A_1.A_0$



$TA_1 = A_0$

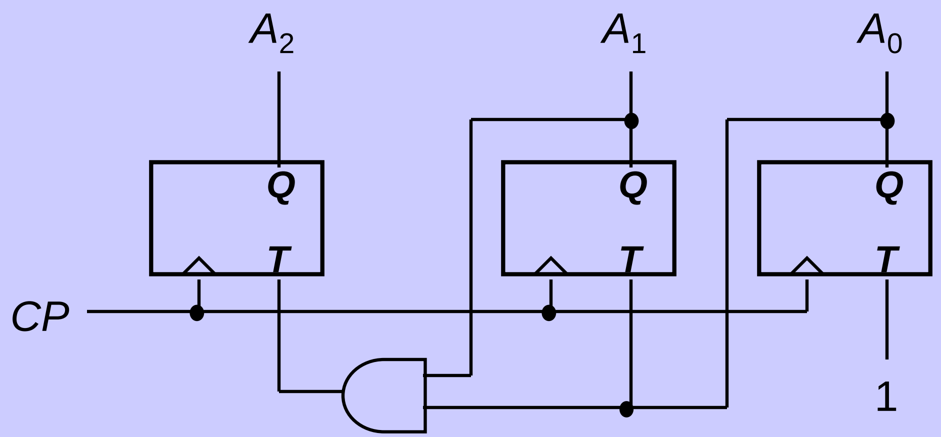


$TA_0 = 1$

Design of Synchronous Counters

- 3-bit binary counter (cont'd).

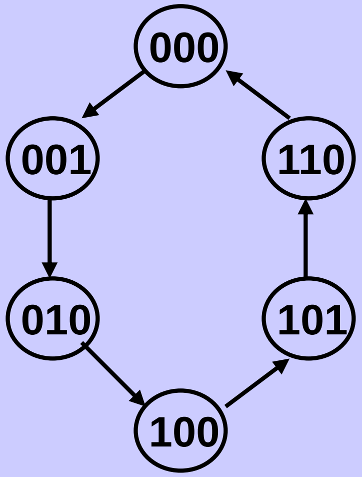
$TA_2 = A_1.A_0$ $TA_1 = A_0$ $TA_0 = 1$



Design of Synchronous Counters

- Example 2: Counter with non-binary sequence:

  $000 \rightarrow 001 \rightarrow 010 \rightarrow 100 \rightarrow 101 \rightarrow 110$ and back to 000



Present state			Next state			Flip-flop inputs					
A	*B*	*C*	A^+	B^+	C^+	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

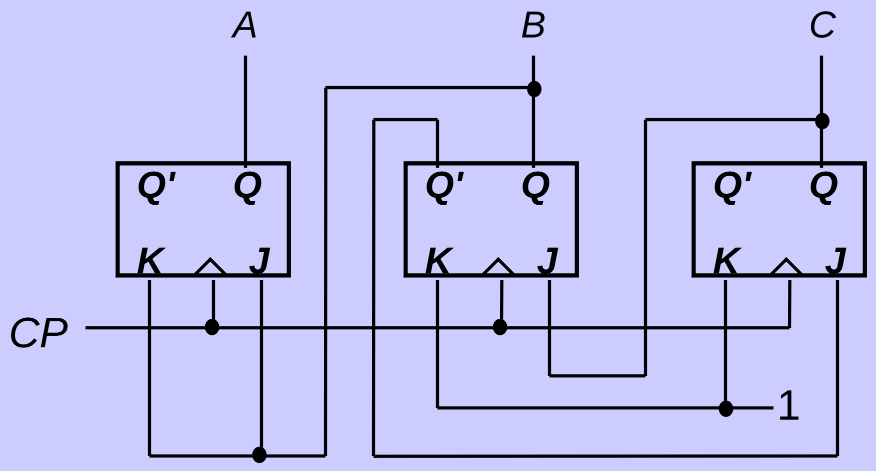
$JA = B$ $\quad$ $JB = C$ $\quad$ $JC = B'$

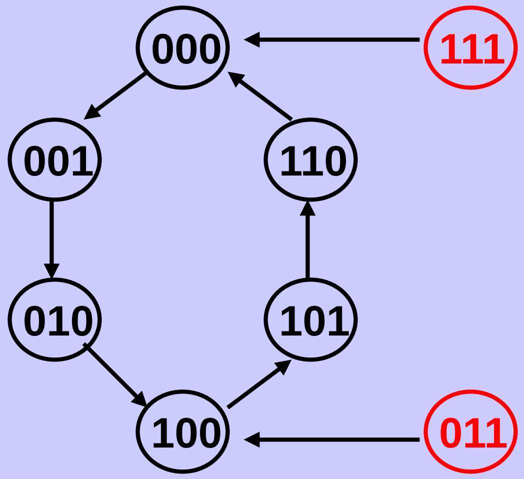
$KA = B$ $\quad$ $KB = 1$ $\quad$ $KC = 1$

Design of Synchronous Counters

- Counter with non-binary sequence (cont'd).

$JA = B$	$JB = C$	$JC = B'$
$KA = B$	$KB = 1$	$KC = 1$





Summary

- Sequential circuits have memory and they are more powerful than combinational circuits.
- Analyzing sequential circuits
  - Flip-flop characteristic table
  - State Table
  - State diagram
- Designing sequential circuits
  - Flip-flop excitation table
  - State assignment
  - Circuit output function
  - Flip-flop input function

- Finite State Machine (FSM)
  - This is what we have been waiting for in this class. Using combinational and sequential logics, now you can design a lot of clever digital logic circuits for functional products. We will learn different steps you take to go from word problems to logic circuits. We first talk about a simplified version of FSM which is a counter.

- Moore/Mealy machines
  - There are two different ways to express the FSMs with respect to the output. Both have different advantages so it is good to know them.

Finite state machines: more than counters

- FSM: A system that visits a finite number of logically distinct states
- Counters are simple FSMs
  - Outputs and states are identical
  - Visit states in a fixed sequence without inputs
- FSMs are typically more complex than counters
  - Outputs can depend on current state and on inputs
  - State sequencing depends on current state and on inputs

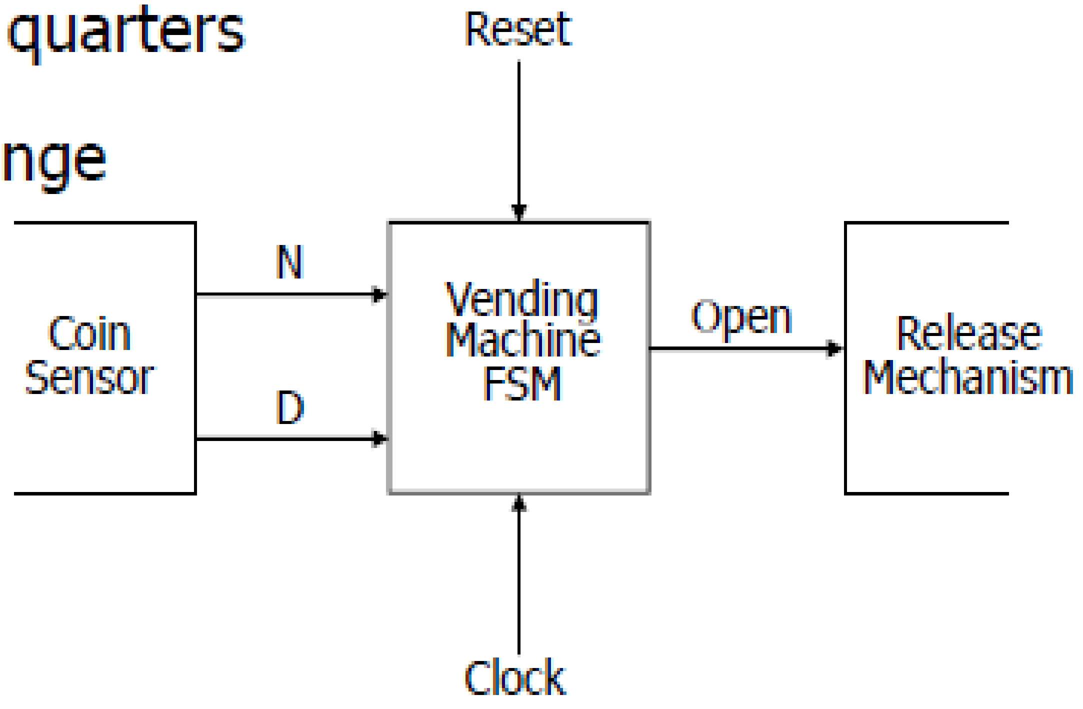
FSM design

- Counter-design procedure
  1. State diagram
  2. State-transition table
  3. Next-state logic minimization
  4. Implement the design

- FSM-design procedure
  1. State diagram
  2. state-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

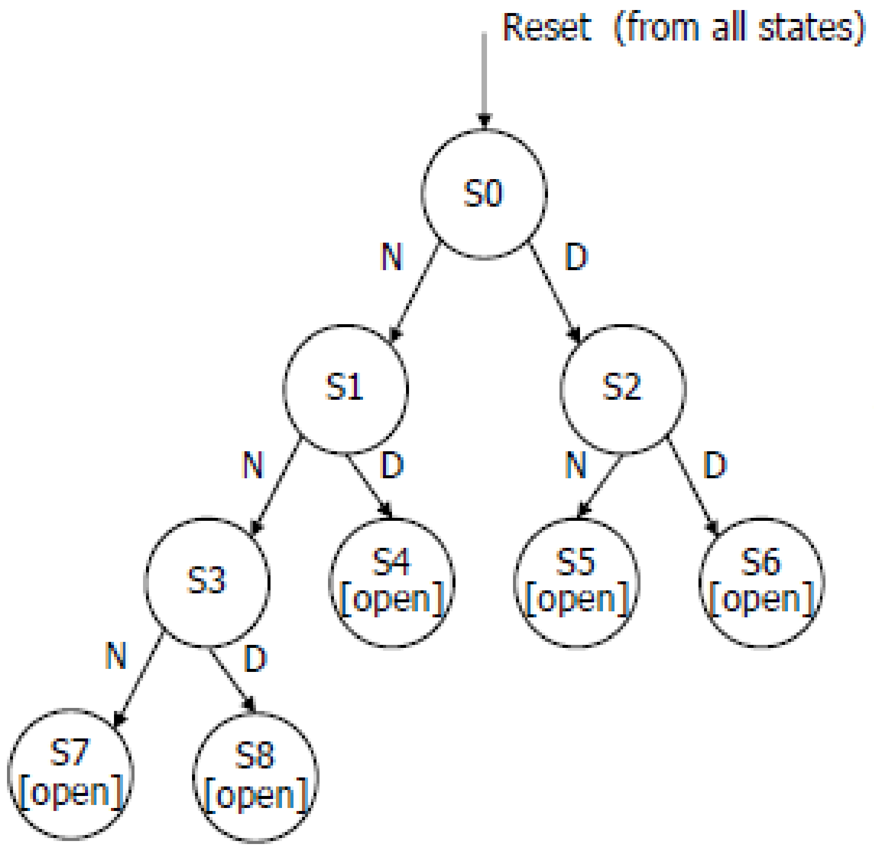
Example: A vending machine

- 15 cents for a cup of coffee
- Doesn't take pennies or quarters
- Doesn't provide any change



- FSM-design procedure
  1. State diagram
  2. state-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

A vending machine: (conceptual) state diagram



Draw self-loops for N′ + D′ for S0 to S3

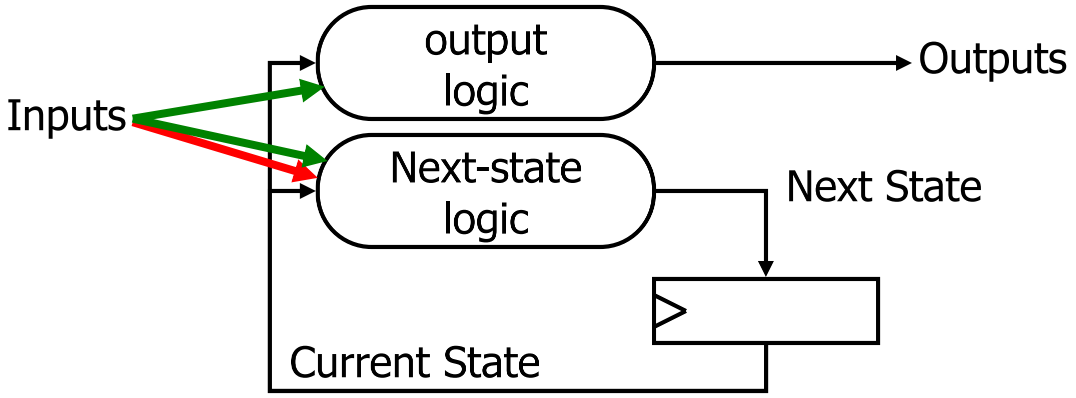
Also draw self-loops for 1 for S4 to S8

A vending machine: State transition table

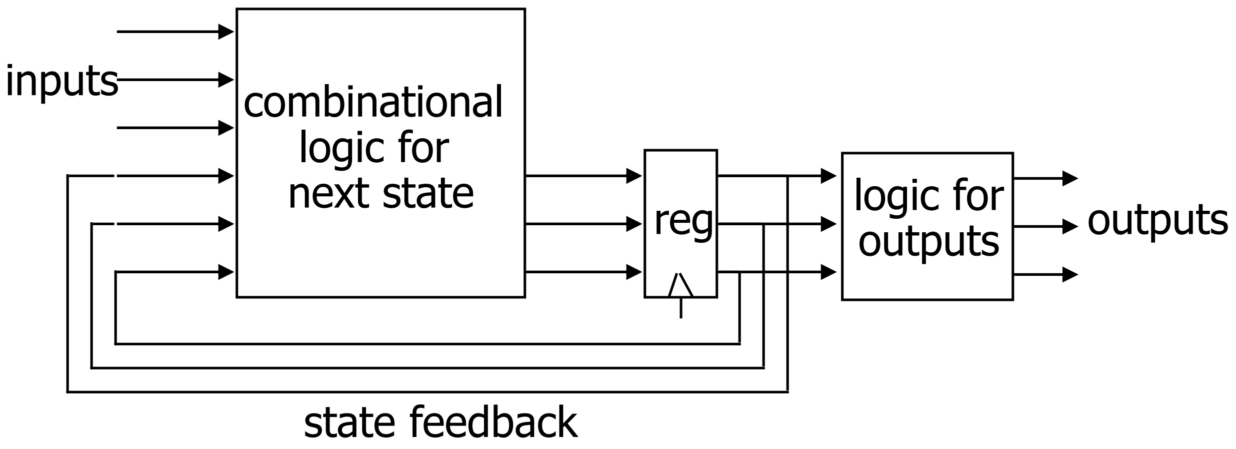
present state	inputs D	inputs N	next state	output open
S0	0	0	S0	0
	0	1	S1	0
	1	0	S2	0
	1	1	X	X
S1	0	0	S1	0
	0	1	S3	0
	1	0	S4	0
	1	1	X	X
S2	0	0	S2	0
	0	1	S5	0
	1	0	S6	0
	1	1	X	X
S3	0	0	S3	0
	0	1	S7	0
	1	0	S8	0
	1	1	X	X
S4	X	X	S4	1
S5	X	X	S5	1
S6	X	X	S6	1
S7	X	X	S7	1
S8	X	X	S8	1

Generalized FSM model: Moore and Mealy

- Combinational logic computes next state and outputs
  - Next state is a function of current state and inputs
  - Outputs are functions of
    - Current state (**Moore** machine)
    - Current state and inputs (**Mealy** machine)

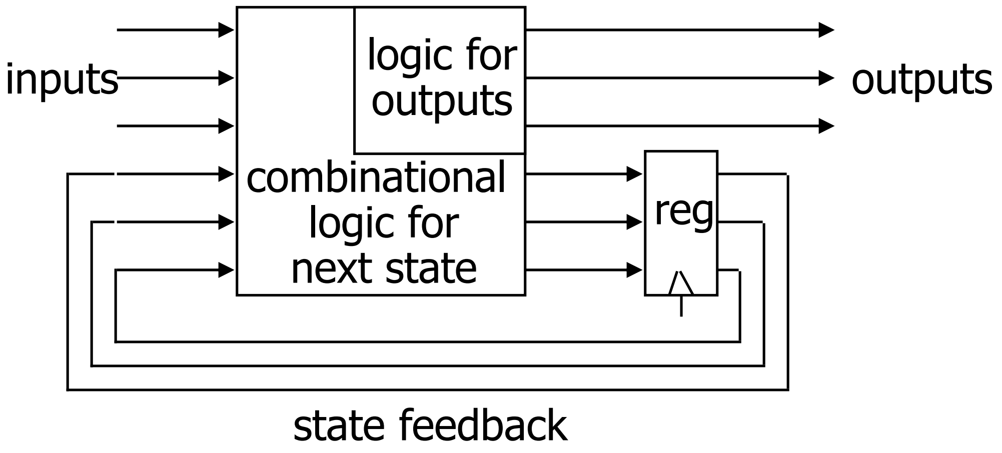


Moore versus Mealy machines



Moore machine
Outputs are a function of current state

Outputs change synchronously with state changes



Mealy machine
Outputs depend on state and on inputs

Input changes can cause immediate output changes (**asynchronous**)

Impacts start of the FSM design procedure

- Counter-design procedure
  1. State diagram
  2. State-transition table
  3. Next-state logic minimization
  4. Implement the design

- FSM-design procedure
  1. State diagram
  2. State-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

State Diagrams

- Moore machine
  - Each state is labeled by a pair:

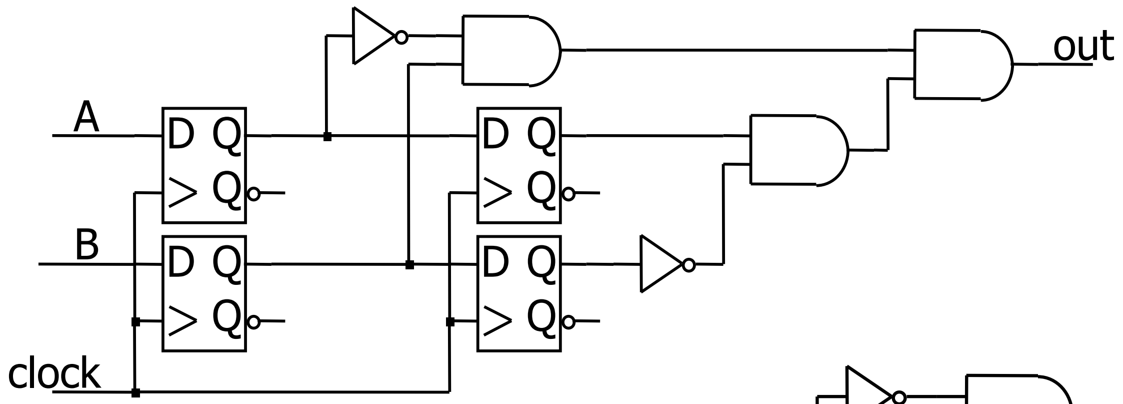
    state-name/output   or   state-name [output]

- Mealy machine
  - Each transition arc is labeled by a pair:

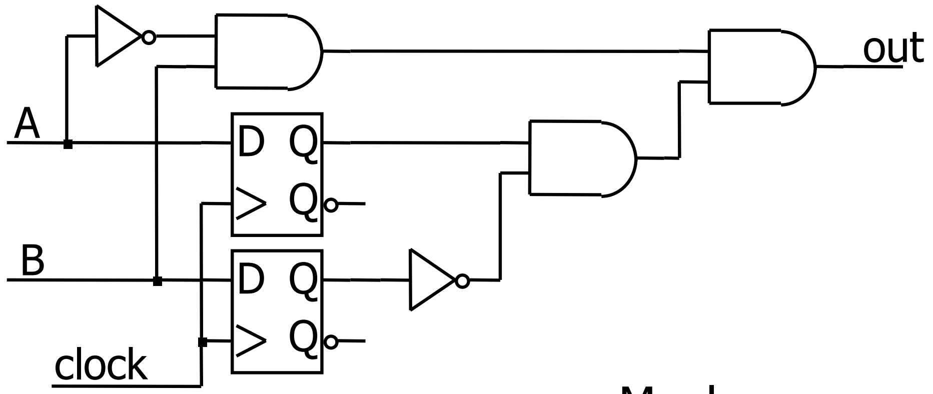
    input-condition/output

Example 10 → 01: Moore or Mealy?

- Circuits recognize AB=10 followed by AB=01
  - What kinds of machines are they?



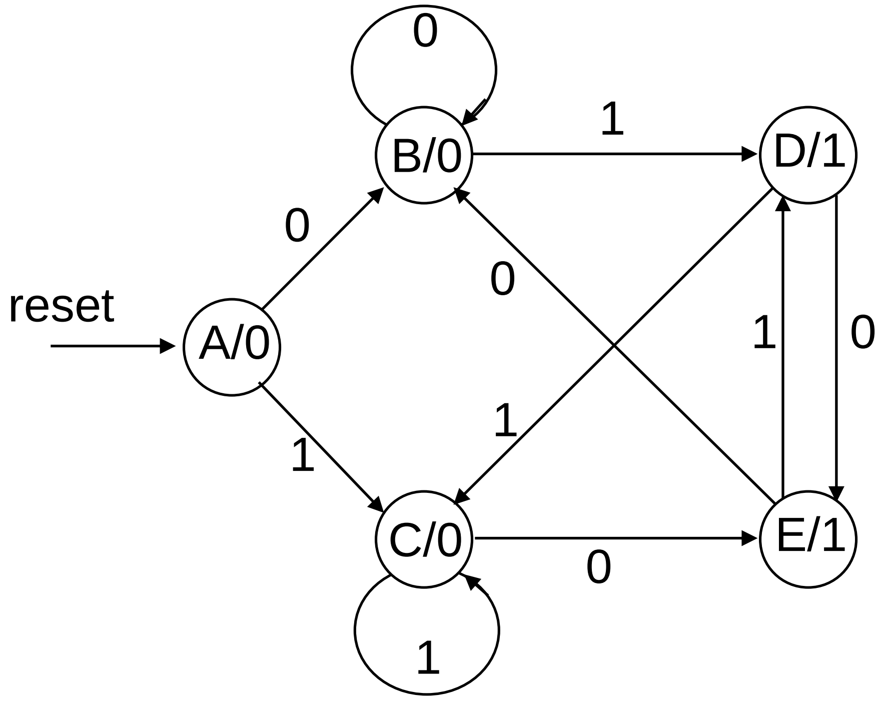
Moore



Mealy

Example "01 or 10" detector: a Moore machine

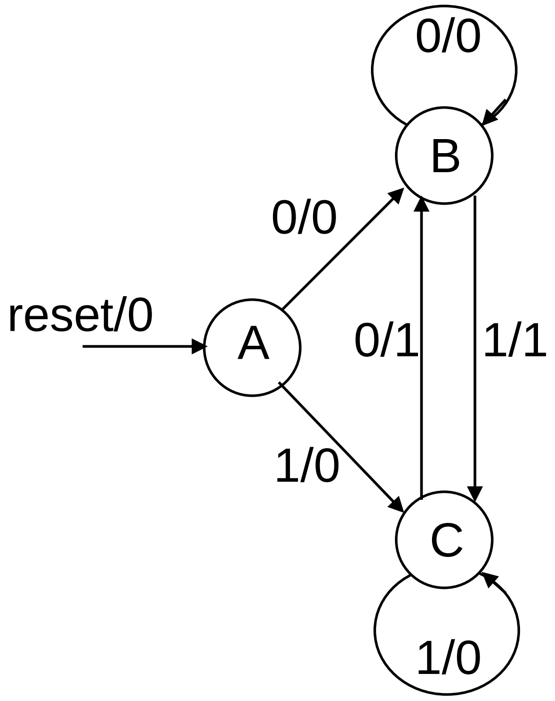
- Output is a function of state only
  - Specify output in the state bubble



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	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

Example "01 or 10" detector: a Mealy machine

- Output is a function of state and inputs
  - Specify outputs on transition arcs



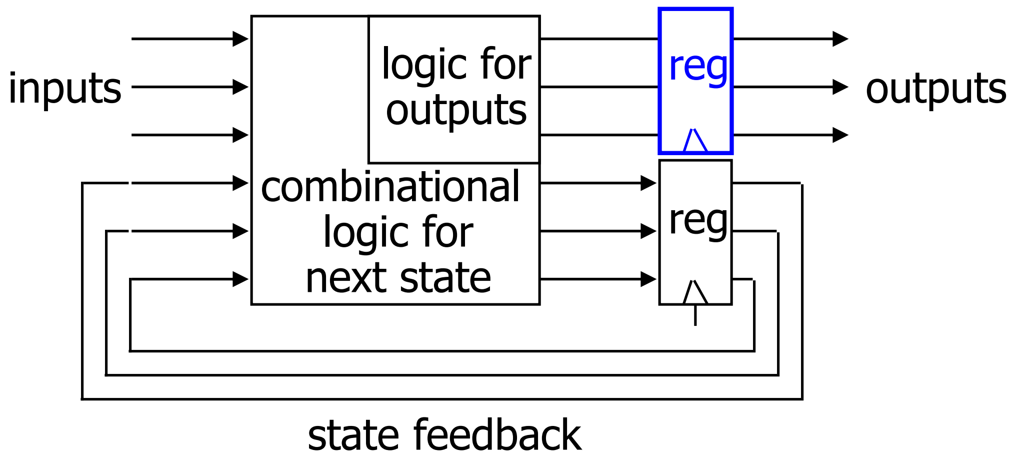
reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Comparing Moore and Mealy machines

- Moore machines
  - + Safer to use because outputs change at clock edge
  - – May take additional logic to decode state into outputs
- Mealy machines
  - + Typically have fewer states
  - + React faster to inputs — don't wait for clock
  - – Asynchronous outputs can be dangerous
- We often design synchronous Mealy machines
  - Design a Mealy machine
  - Then register the outputs

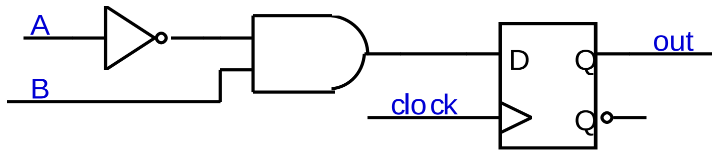
Synchronous (registered) Mealy machine

- Registered state and registered outputs
  - No glitches on outputs
  - No race conditions between communicating machines

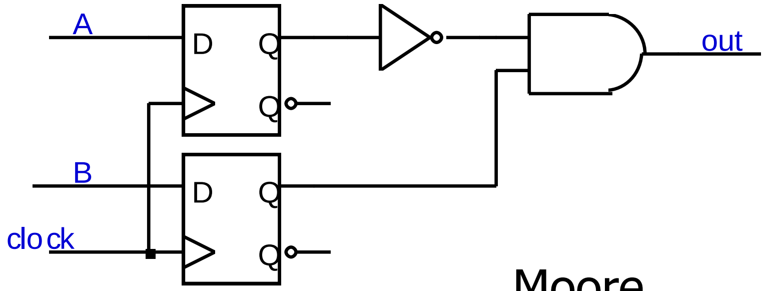


Example "=01": Moore or Mealy?

- Recognize AB = 01
  - Mealy or Moore?



Registered Mealy
(actually Moore)



Moore

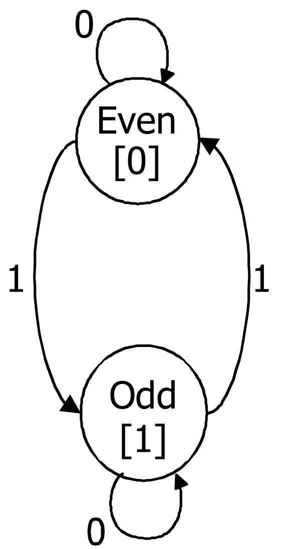
Example: A parity checker

- Serial input string
  - OUT=1 if odd # of 1s in input
  - OUT=0 if even # of 1s in input
- Let’s do this for Moore and Mealy

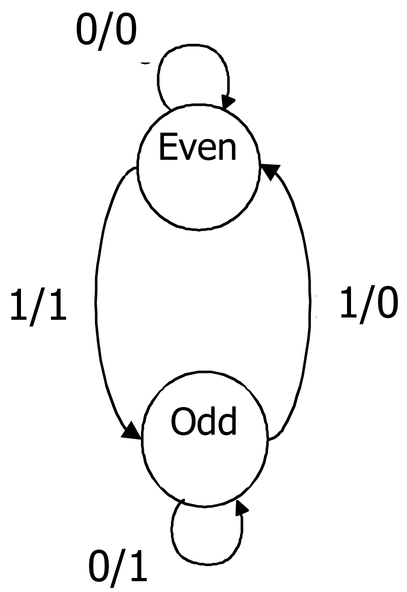
Example: A parity checker

1. State diagram

Moore



Mealy



Example: A parity checker

1. State-transition table

Moore

Present State	Input	Next State	Present Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Mealy

Present State	Input	Next State	Present Output
Even	0	Even	0
Even	1	Odd	1
Odd	0	Odd	1
Odd	1	Even	0

Example: A parity checker

3. State minimization: Already minimized
   - Need both states (even and odd)
   - Use one flip-flop

Example: A parity checker

4. State encoding

Moore

Present State	Input	Next State	Present Output
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Assignment	
Even	0
Odd	1

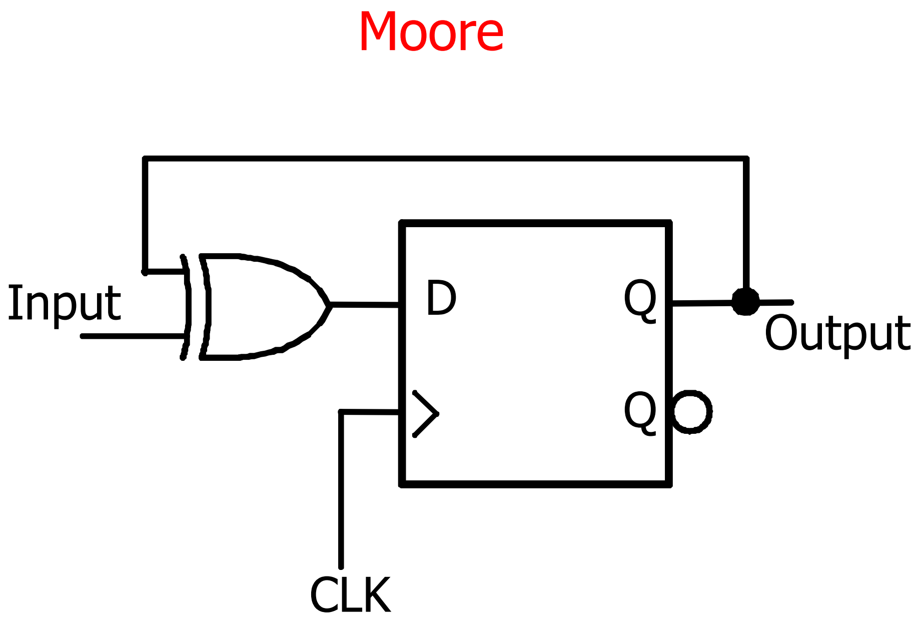
Mealy

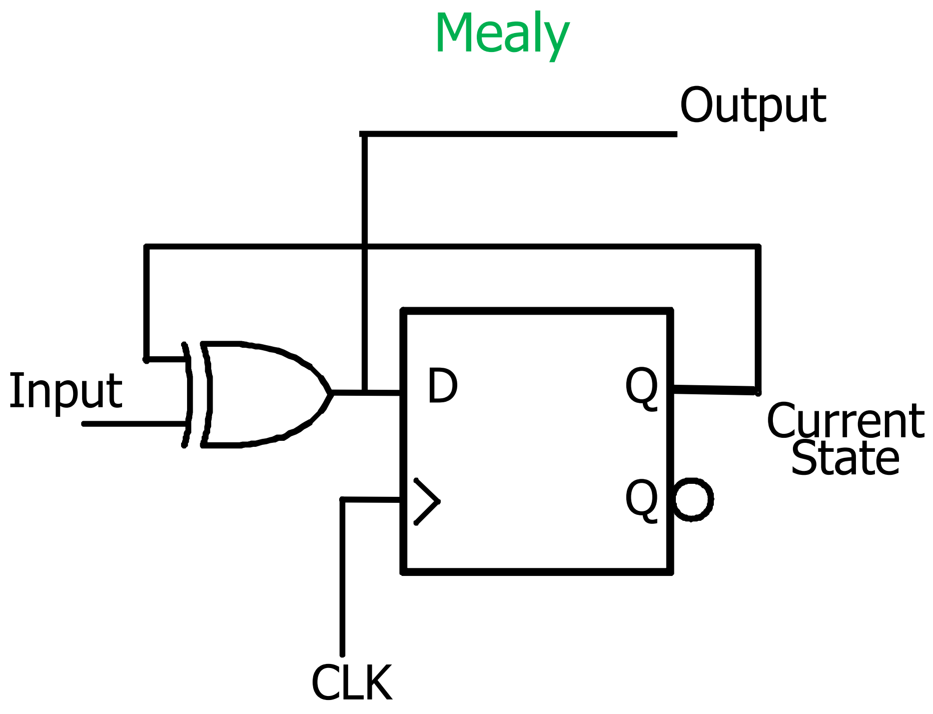
Present State	Input	Next State	Present Output
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Example: A parity checker

5. Next-state logic minimization
   - Assume D flip-flops
   - Next state = (present state) XOR (present input)

6. Implement the design





What was covered so far on FSM

- Finite state machines
  - FSM design procedure
    1. State diagram
    2. State-transition table
    3. State minimization
    4. State encoding
    5. Next-state logic minimization
    6. Implement the design
  - No Mealy machines