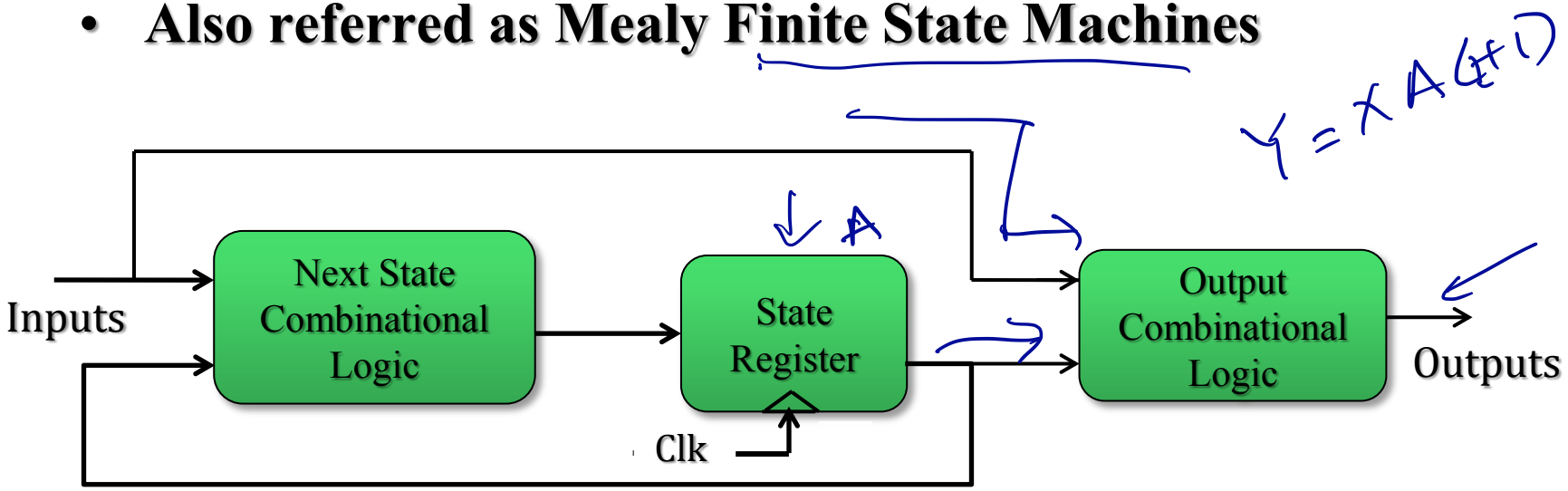
WEEK 7-2017

Synchronous Sequential Circuit III

Mealy and Moore Models

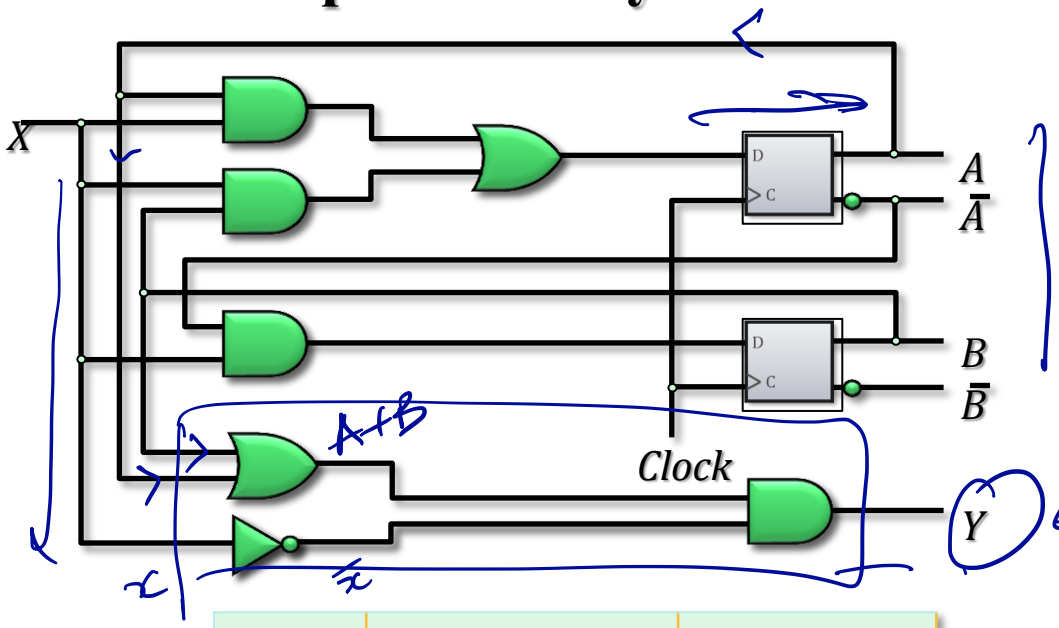
- **Two models of sequential circuits: Mealy and Moore**
- **Mealy models: the output is a function of both the present state and the input.**
- **Also referred as Mealy Finite State Machines**



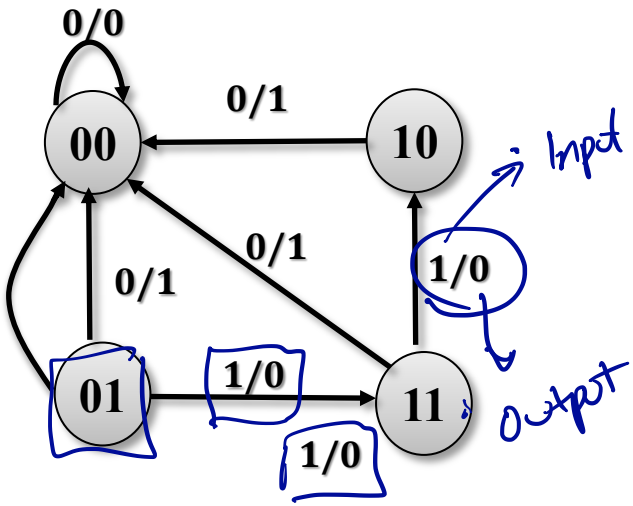
Mealy Finite Machine

Mealy Example

- **Example of Mealy FSM**

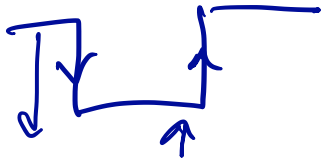


$$Y(t) = [A + B]\bar{x}$$



Present state		Next state				Output	
		X=0		X=1		X=0	X=1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

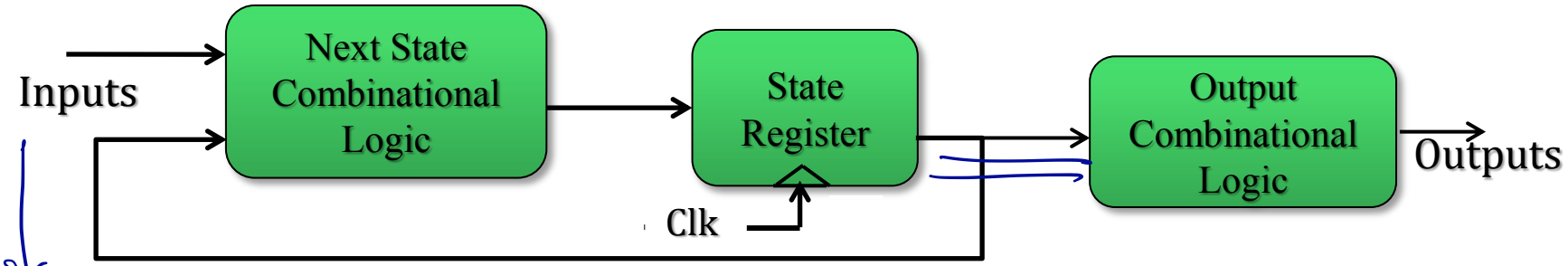
Mealy Models



- The outputs may change during the clock cycle.
- Momentary false values (glitches) may occur due to the delay encountered from the time that the inputs change and from the time that flip-flop outputs change.
- In order to solve mitigate these problems,
  - Inputs must be synchronized with the clock
  - Outputs must be sampled immediately before the clock edge
  - Inputs are changed at the inactive edge of the clock
- Thus, in mealy circuit, the output value is the value that is present immediately before the active edge of the clock.

Moore Model

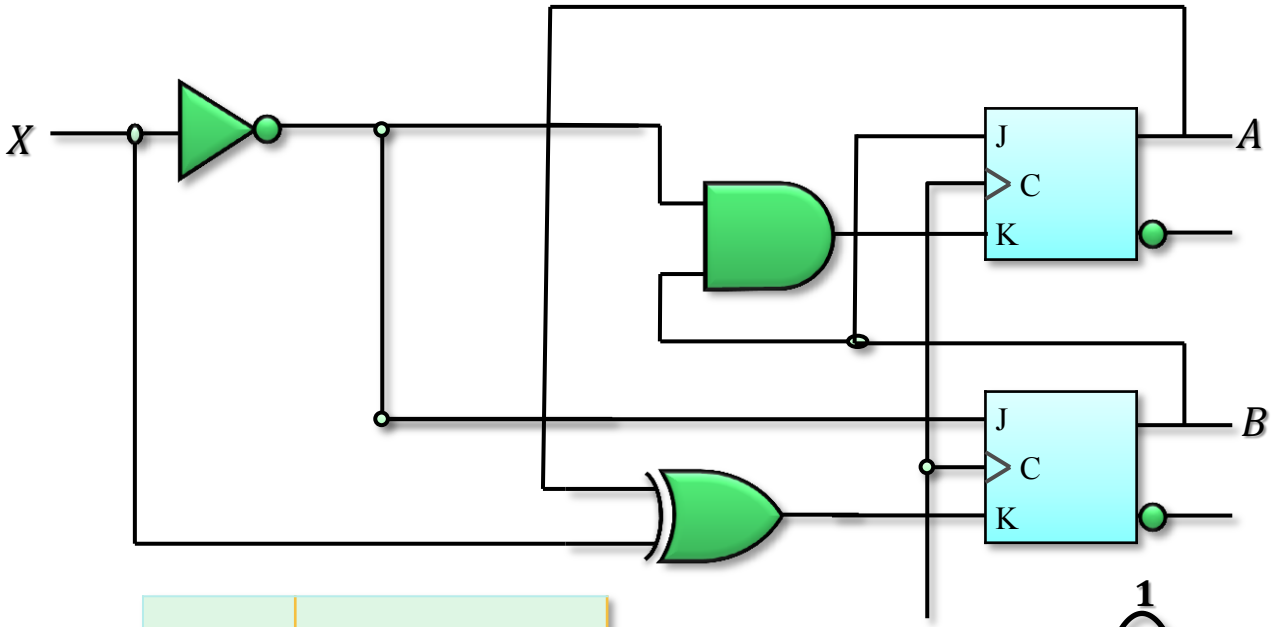
- **Moore Model: the output is a function of only the present state.**
- **Also known as Moore Finite State Machine.**
- **State tables do not need separate output listing for various input combinations.**
- **Output value is indicated inside the circle together with the present state.**



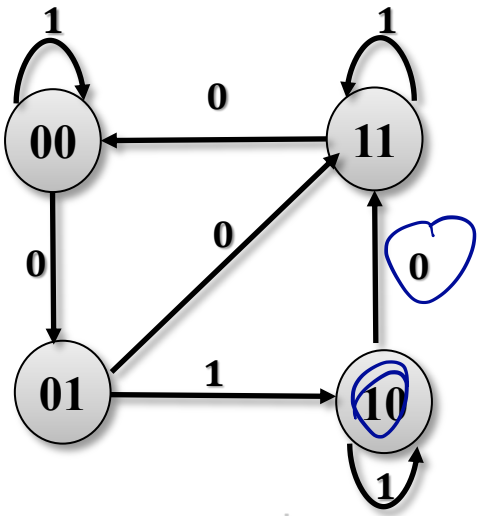
Moore Finite Machine

Moore Example

- **Example of Moore FSM**

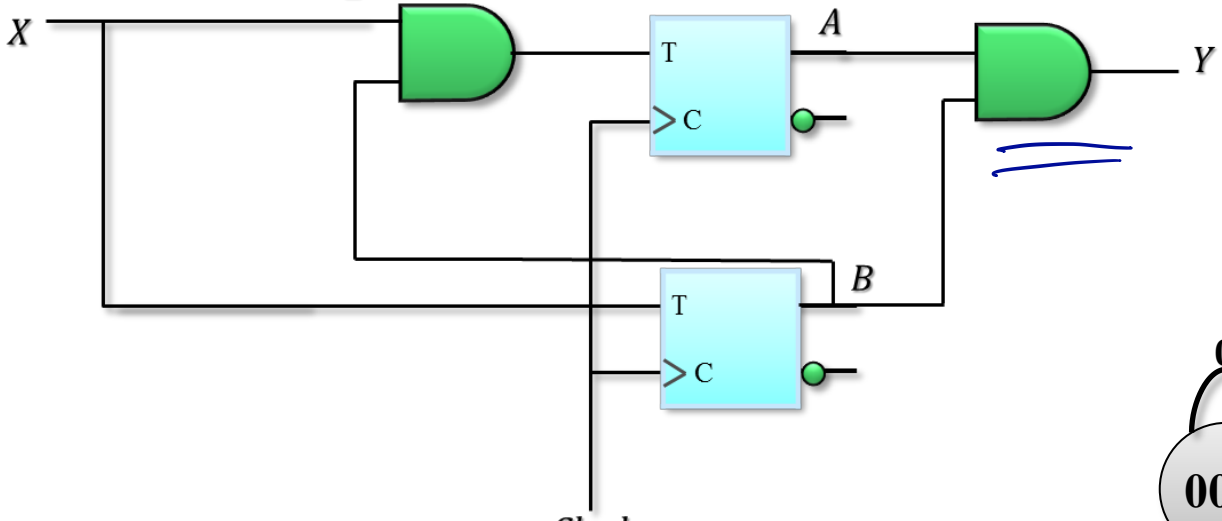


Present state		Next state			
		X=0		X=1	
A	B	A	B	A	B
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1

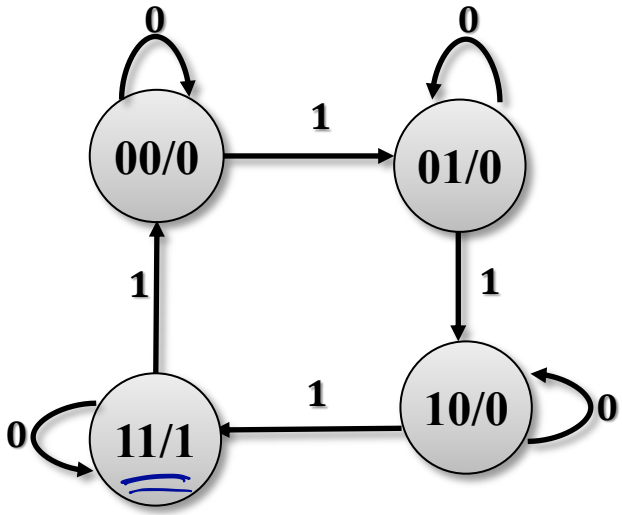


Moore Example

- **Example of Moore FSM**

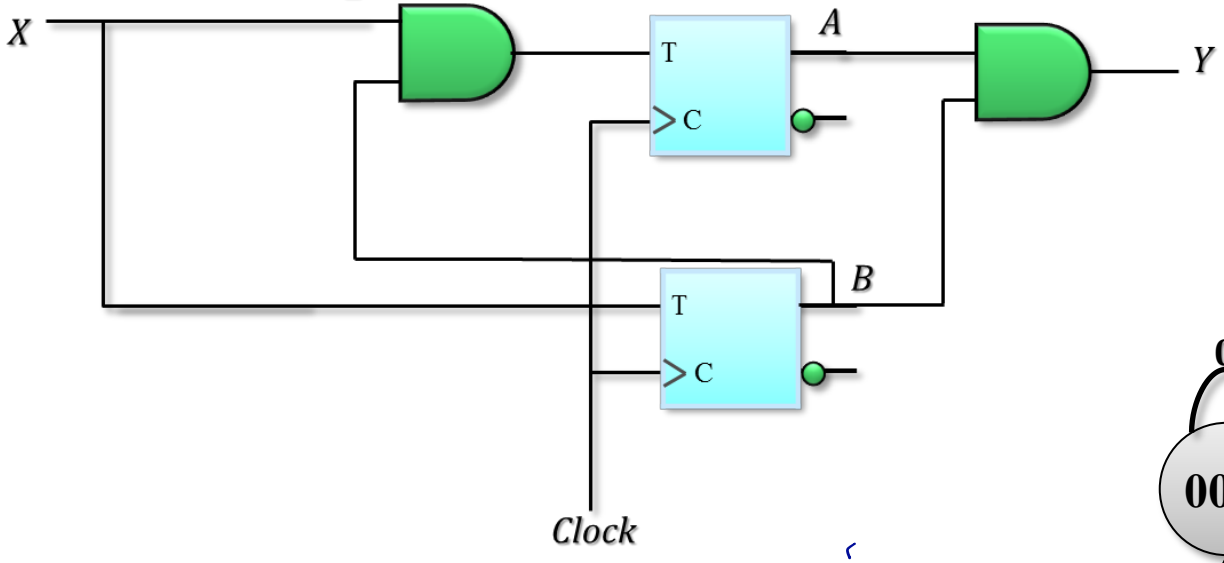


Present state		Next state				Output
		X=0		X=1		
A	B	A	B	A	B	Y
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1

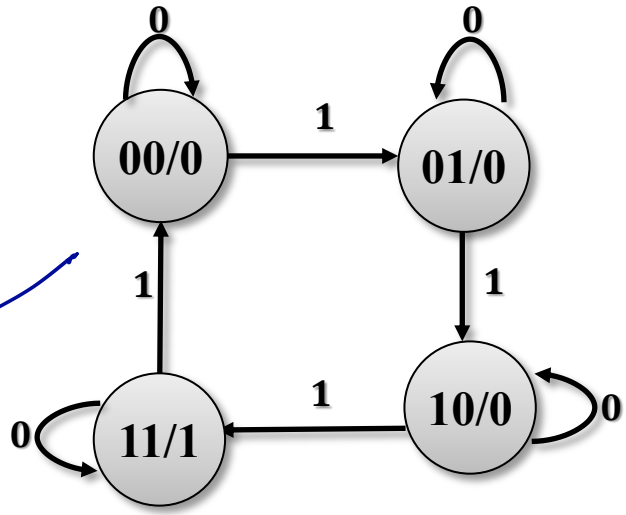


Moore Example

- **Example of Moore FSM**



Present state	Next state		Output
	X=0	X=1	
AB	AB	AB	Y
00	00	01	0
01	01	10	0
10	10	11	0
11	11	00	1



State Reduction

- **State reduction is a necessary step in realization of sequential circuits as it results in fewer numbers of flip-flops and hence reduction of the cost of a circuit.**

- **State-reduction algorithms are concerned with procedures for reducing the number of states in a state table while maintaining the same input-output requirements and relationships.**

- **Steps in state reduction**
  - **Produce state table from the state diagram**
  - **Apply the following algorithm:**

    "*Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state*"
  - **Construct implication table to check each pair of states for possible equivalence**

Implication table

- **Implication table is a chart that consists of squares, one for every possible pair of states, that provide spaces for listing any possible implied states.**
- **Consider the following state table:**

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

- **The present state a and b have the same output for the same input.**
- **Their next states are c and d for x=0 and b and a for x=1.**
- **If we can show c and d are equivalent, then a and b will be equivalent.**

Implication table

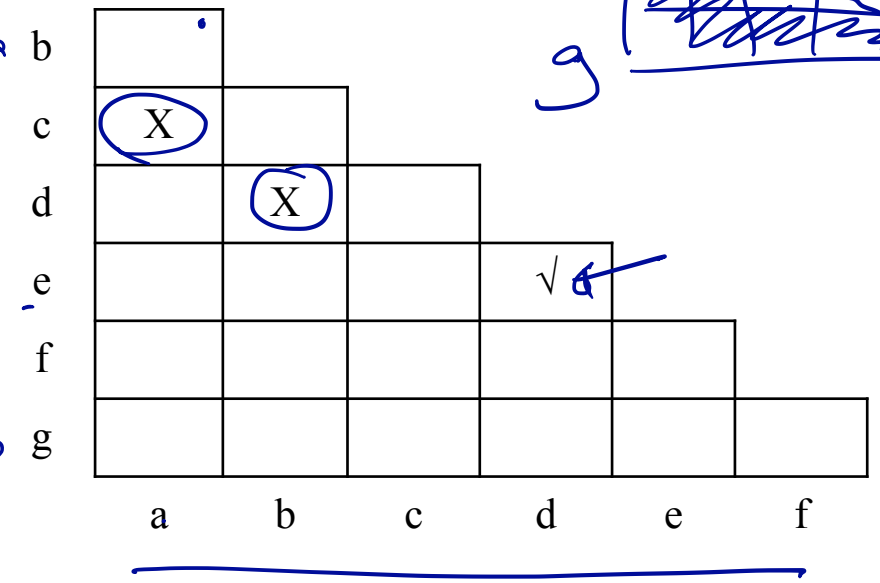
- When this relationship exist, we say (a,b) imply (c,d)
- That is if a and b are equivalent, then c and d have to equivalent.
- We can also observe that (c,d) imply (a,b).

"The characteristics of equivalent states is that if (a,b) imply (c,d) and (c,d) imply (a,b), then both pairs of states are equivalent"

- The previous state table can be reduced to only two states.

Implication table

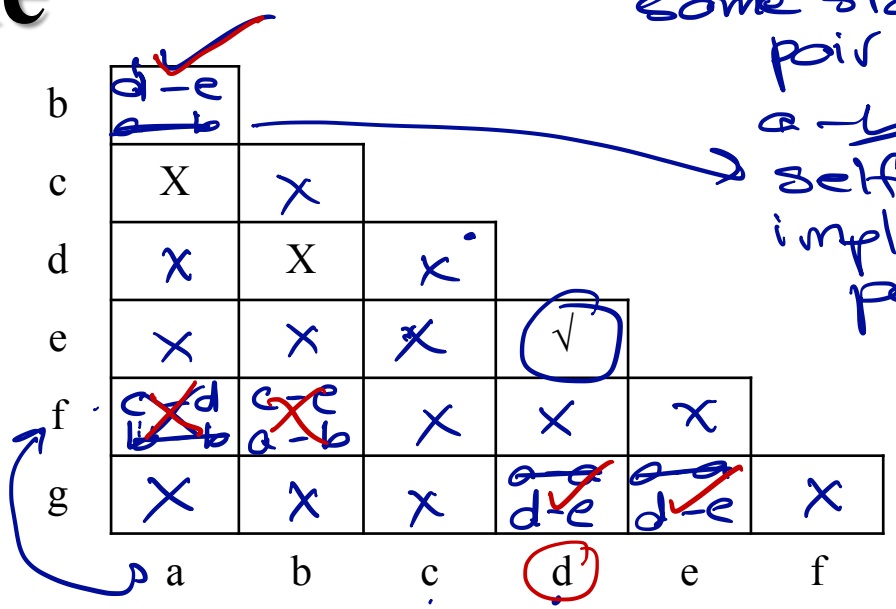
Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



- **Two states having different outputs for the same inputs are not equivalent.**
- **Two states that are not equivalent are marked with a cross (X) in the corresponding square whereas their equivalence is recorded with a check mark (√)**

Implication table

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



- **The steps in filling the implication table:**
  - **Place a cross in any square corresponding to non-equivalent states.**
  - **Enter in the remaining squares the implied states (Top-down and then proceed with next column to the right)**
  - **Record check marks for equivalent states**
  - **Make successive passes through the table to place a cross and check marks**
  - **Continue the procedure until no additional squares can be crossed out.**

For the above example: we have the following equivalent states (a,b) (c) (d,e,g) (f)

State reduced from seven to four states.

Implication table

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	a	0	0
a	d	a	0	0
c	d	f	0	1
d	a	d	1	0
d	a	d	1	0
f	c	a	0	0
d	a	d	1	0

7

3 FFs

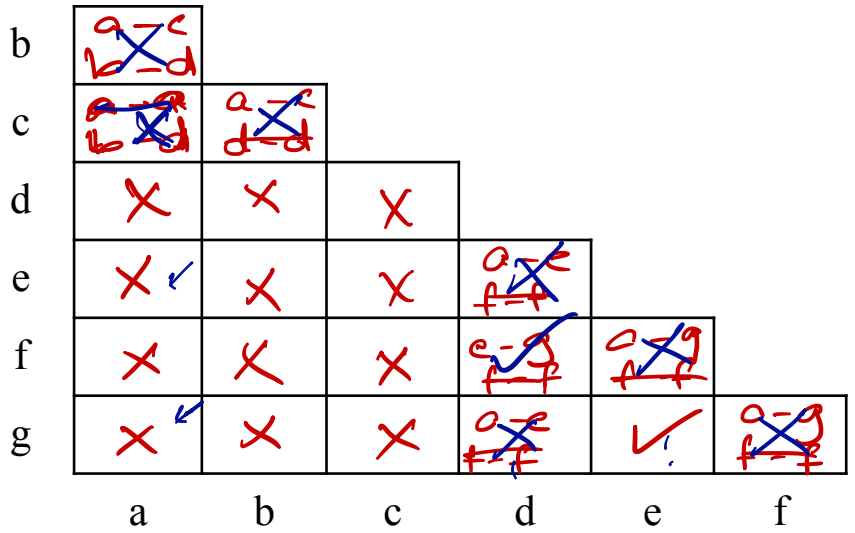
Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	a	0	0
c b	d	f	0	1
d c	a	d	1	0
f d	c	a	0	0

4

2 FFs

Implication table – Example 2

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f d	0	1
e	a	f d	0	1
f d	g e	f d	0	1
g e	a	f d	0	1



e ≡ g ✓

d ≡ f ✓

a
b
c
d
e

Implication table – Example 2

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1
d	e	d	0	1
e	a	d	0	1

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

State Assignment

- It is necessary to assign unique binary code to the states in order to design a sequential circuit with physical components.
- For m states, the codes must contain n bits, where $2^n \geq m$.
- Eight states can be assigned with three bits codes, binary numbers from 000 through 111.
- If the state table has seven states, the binary numbers from 000 to 110 can be used to assign the states; the remaining state is unused.
- For five states, five binary numbers can be used. The remaining three states are unused.
- The unused states are treated as “Don’t Care conditions” during design.

State Assignment

state	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	100	10000

- **Assignment 1 is often used and easy to apply.**
- **Assignment 2 makes it easier for boolean function to be placed in the map for simplification.**
- **Assignment 3 provides faster machines and simpler decoding logic for the next state and output.**

State Assignment - Example

- **Using binary code for state assignment, the reduced state table in the last example can be given as**

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Present state	Next state		Output	
ABC	X=0 A(t+1)	X=1	X=0	X=1
000 ←	000	001	0	0
001 ←	010	011	0	0
010 ←	000	011	0	0
011 ←	100	011	0	1
100 ←	000	011	0	1

2^n

Sequential Circuit Design

Procedure

1. Derive the state diagram from the word description and specifications of the desired operation.
2. Reduce the number of states.
3. Assign binary values to the states.
4. Obtain the binary-coded state table.
5. Choose the type of flip-flops to be used.
6. Derive the simplified flip-flop input equations and output equations.
7. Draw the logic diagram

Sequential Circuit Design-Example

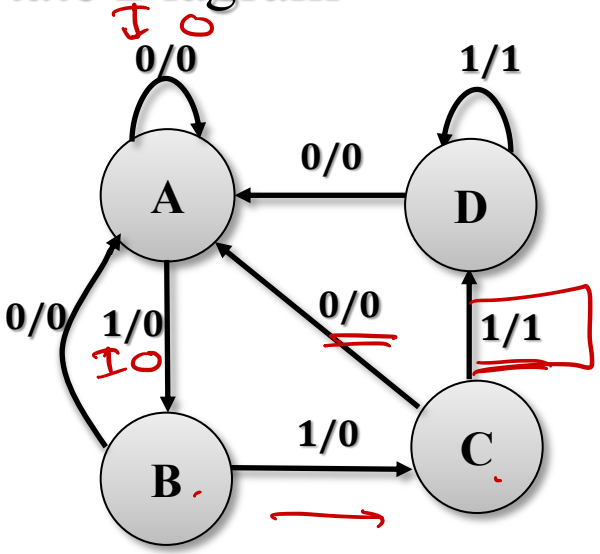
Specification:

Suppose we wish to design a circuit that detects a sequence of three or more consecutive 1's in a string of bits coming through an input line (ie the input is in serial bit stream)

0 1 0 1 1 0 1 1 1 0 1 1 1 1 0 0

0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0

State Diagram



C ≡ D

A – 0 1's

B = 1 1's

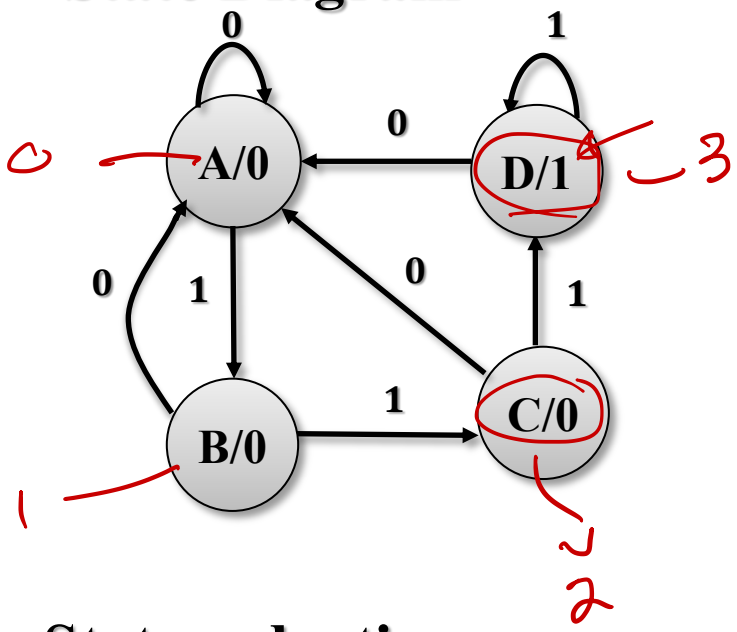
C = 2 1's

D = 3 1's

present state	next state x=0	next state x=1	output x=0	output x=1
A	A	B	0	0
B	A	C	0	0
C	A	D	0	1
D	A	D	0	1

Sequential Circuit Design-Example

State Diagram



State table

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
A	A	B	0	0
B	A	C	0	0
C	A	D	0	0
D	A	D	1	1

State reduction

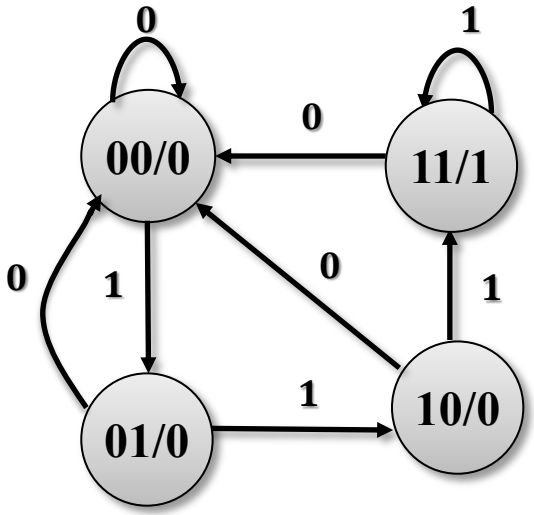
B	B,C x		
C	B,D x	C,D X	
D	X	X	X
	A	B	C

State assignment

A=00
B=01
C=10
D=11

Sequential Circuit Design-Example

State Diagram



State table

Present state	Next state		Output	
	X=0	X=1	X=0	X=1
00	00	01	0	0
01	00	10	0	0
10	00	11	0	0
11	00	11	1	1

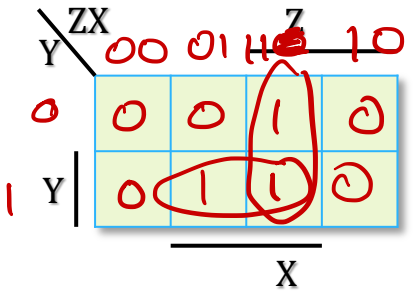
- **Synthesis using D Flip-Flops**
- **The next state equation is the same as the D flip-flop input equation.**
- **Let us Y and Z be the name of the two D flip-flops required**
- **$D_Y = Y(t+1)$ and $D_Z = Z(t+1)$**

Sequential Circuit Design-Example

Present state		Input	Next state		Output
Y	Z	X	Y(t+1)	Z(t+1)	F
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

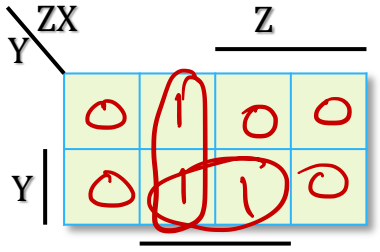
Present state	Next state		Output	
	X=0	X=1	X=0	X=1
00	00	01	0	0
01	00	10	0	0
10	00	11	0	0
11	00	11	1	1

D_Y:



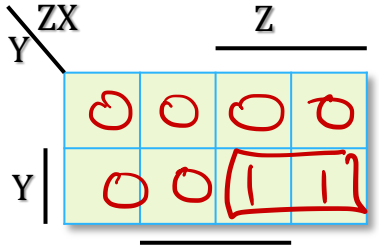
$D_Y = YX + XZ$

D_Z:



$D_Z = YX + \bar{Z}X$

F:



$F = YZ$

Sequential Circuit Design-Example

D_Y: $XZ + YX$

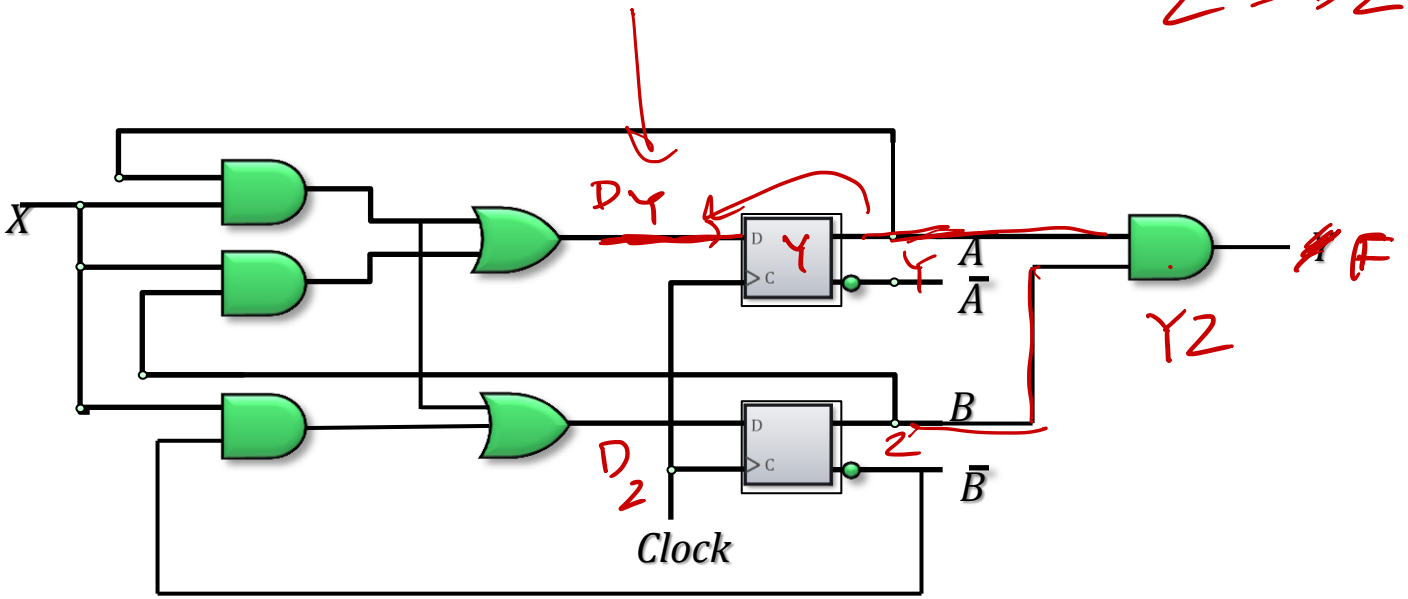
D_Z: $YX + \bar{Z}X$

F: YZ

- **Logic diagram:**

$Y = D_Y$

$Z = D_Z$



Excitation Tables

- When D flip-flops are employed, the input equations are obtained directly from the next state.
- However, for JK and T flip-flops, the input equations need to be derived indirectly from the state table.

- Therefore, a table that lists the required inputs for a given change of state in state table is needed. Such a table is called an excitation table

Excitation table for JK flip-flop

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation table for T flip-flop

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

J	K	Q (t+1)	
0	0	$Q(t)$	Unchanged
0	1	$\overline{0}$	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	complement

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

T	Q (t+1)	
0	$Q(t)$	unchanged
1	$\overline{Q}(t)$	complemented

T	$Q(t)$	$Q(t+1)$	T
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Synthesis using JK flip-flops

- **The excitation table must be used to obtain the required input equations from the state table.**
- **For previous design problem:**

Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	Y(t+1)	Z(t+1)	J_Y	K_Y	J_Z	J_Z
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

Excitation

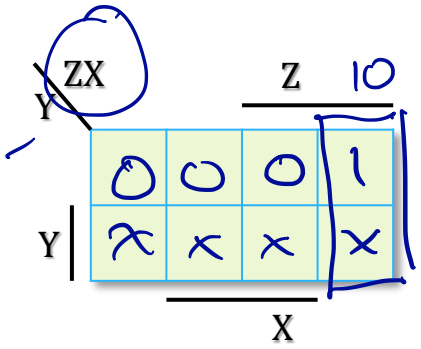
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Synthesis using JK flip-flops

- **The excitation table must be used to obtain the required input equations from the state table.**
- **Consider the following state table:**

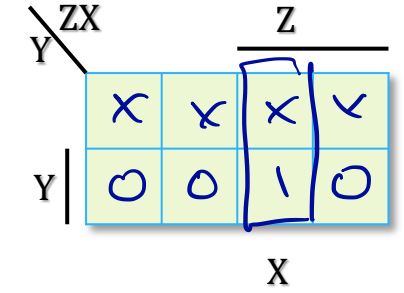
Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	Y(t+1)	Z(t+1)	J_Y	K_Y	J_Z	K_Z
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

J_Y:



$J_Y = ZX$

K_Y:



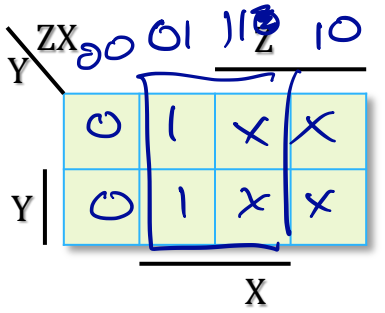
$K_Y = ZX$

Synthesis using JK flip-flops

- **The excitation table must be used to obtain the required input equations from the state table.**
- **Consider the following state table:**

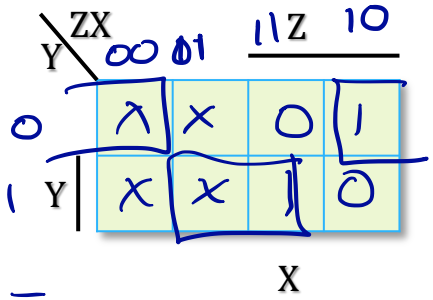
Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	Y(t+1)	Z(t+1)	J_Y	K_Y	J_Z	K_Z
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

J_Z:



$J_Z = X$

K_Z:



$K_Z = \bar{Y}\bar{X} + YX$

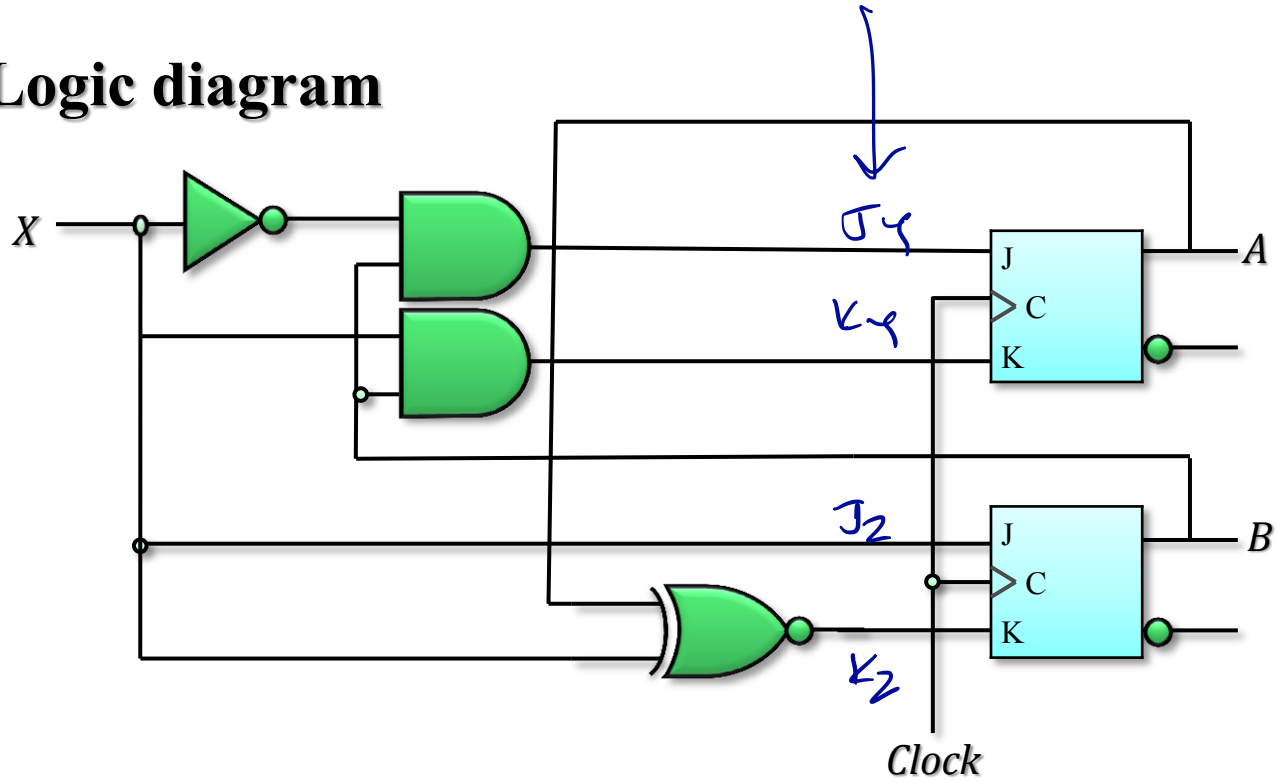
$= Y \oplus X$

Synthesis using JK flip-flops

J_Z: x K_Z: $\overline{Y \oplus x}$

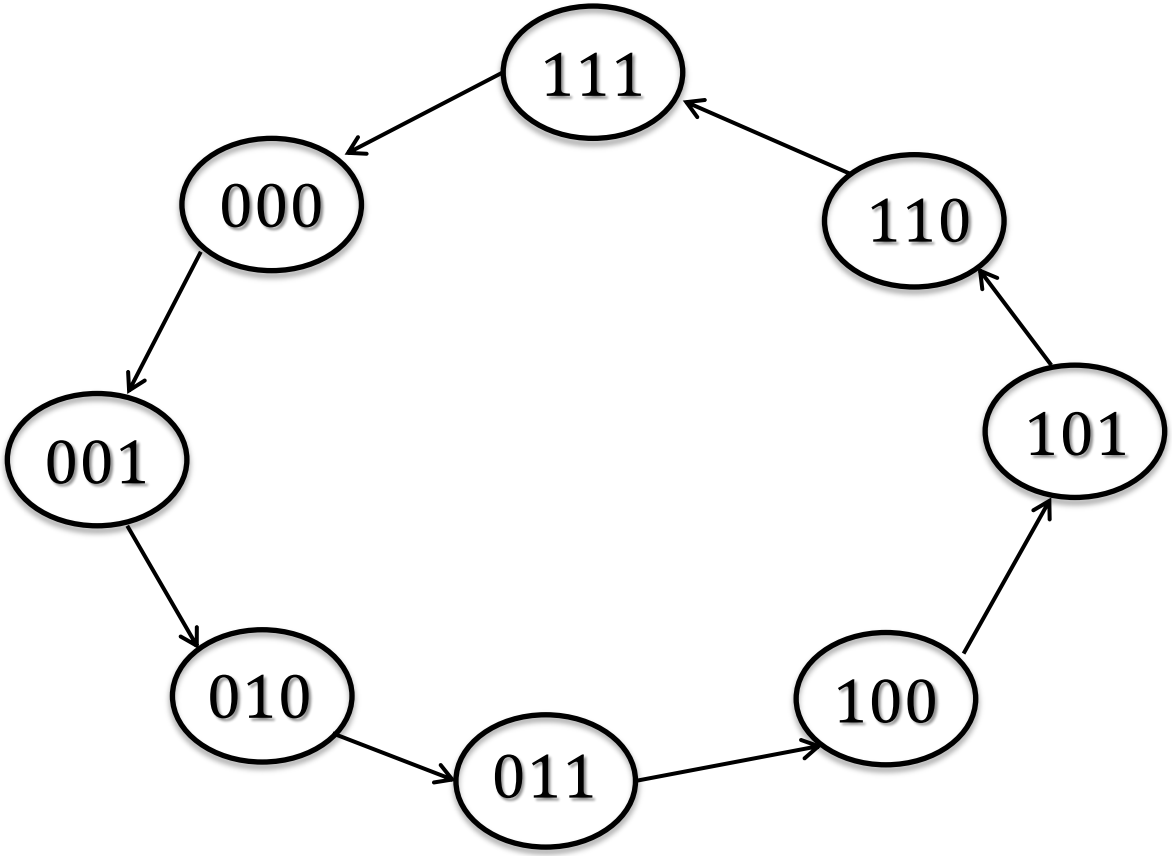
J_Y: $z\bar{x}$ K_Y: zx

- **Logic diagram**



Synthesis using T flip-flops

- **Design a three-bit counter consisting of three T flip-flops that can count in binary form from 0 to 7**
- **State diagram:**



Synthesis using T flip-flops

State Table:

Present state			Next state			Flip-Flop Inputs		
A	B	C	A(t+1)	B(t+1)	C(t+1)	T_A	T_B	T_C
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

T

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

$T_C = 1$

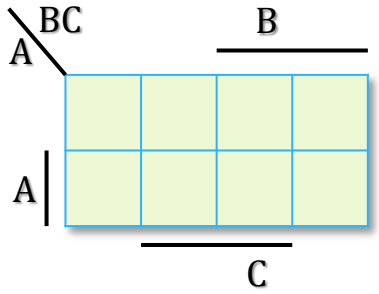
$T_B = C$

$T_A = BC$

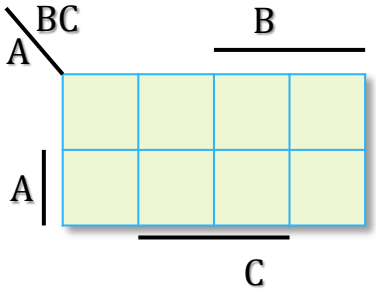
Synthesis using T flip-flops

Present state			Next state			Flip-Flop Inputs		
A	B	C	A(t+1)	B(t+1)	C(t+1)	T_A	T_B	T_C
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

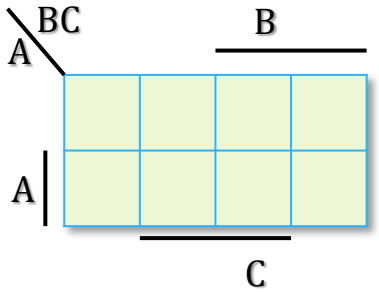
T_A:



T_B:



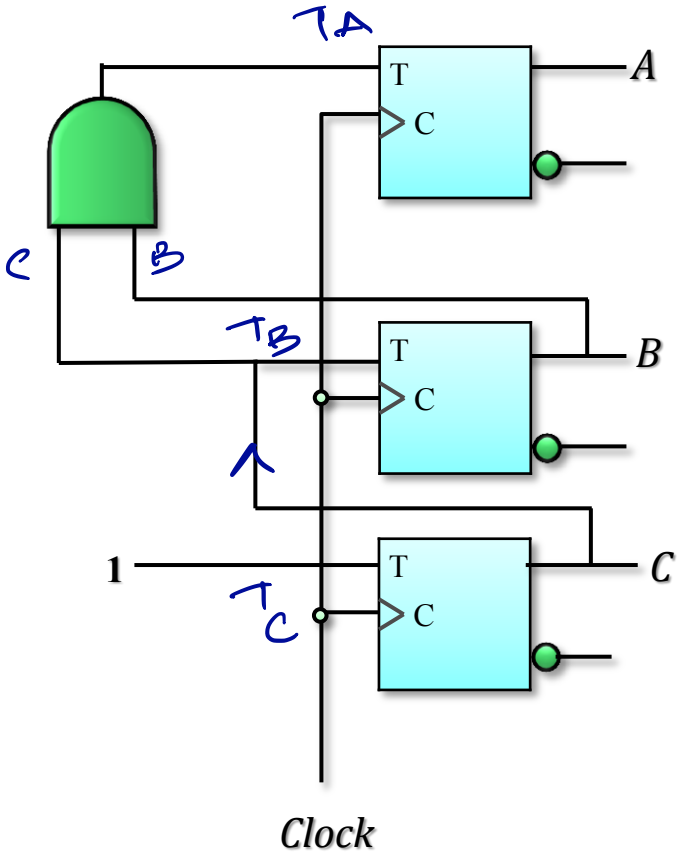
T_C:



Synthesis using T flip-flops

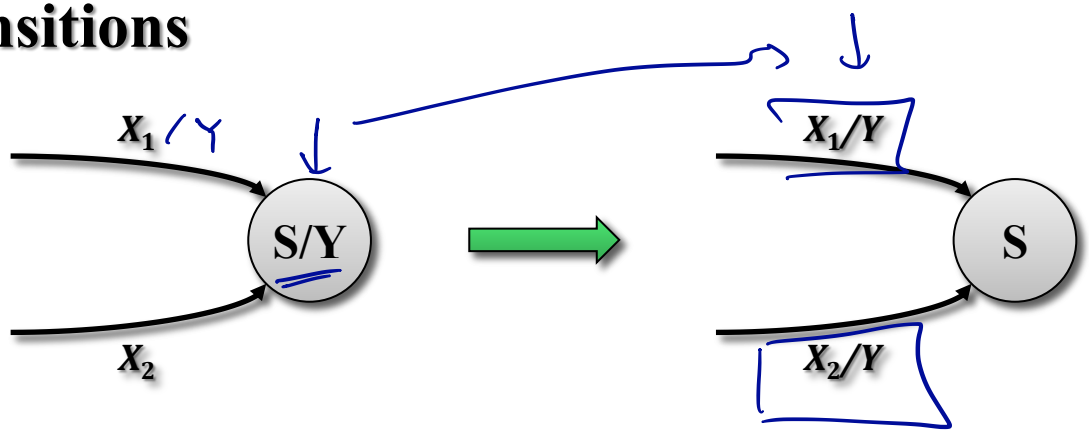
T_A: BC T_B: C T_C: 1

- **Logic diagram**



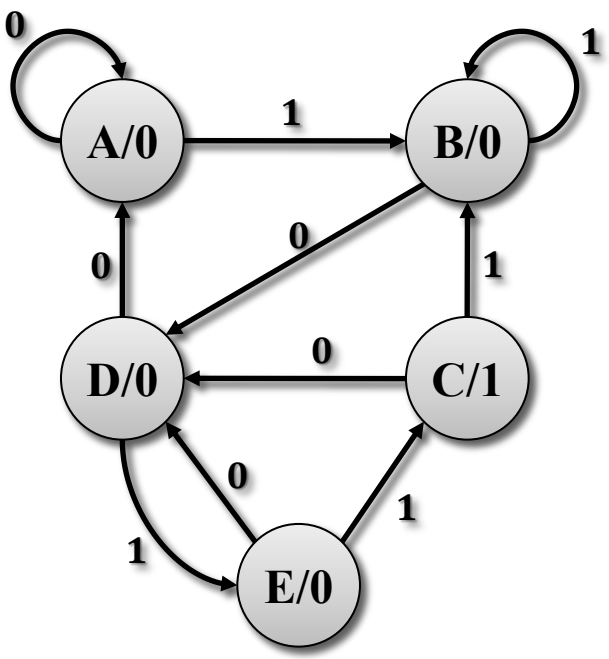
Moore to Mealy Conversion

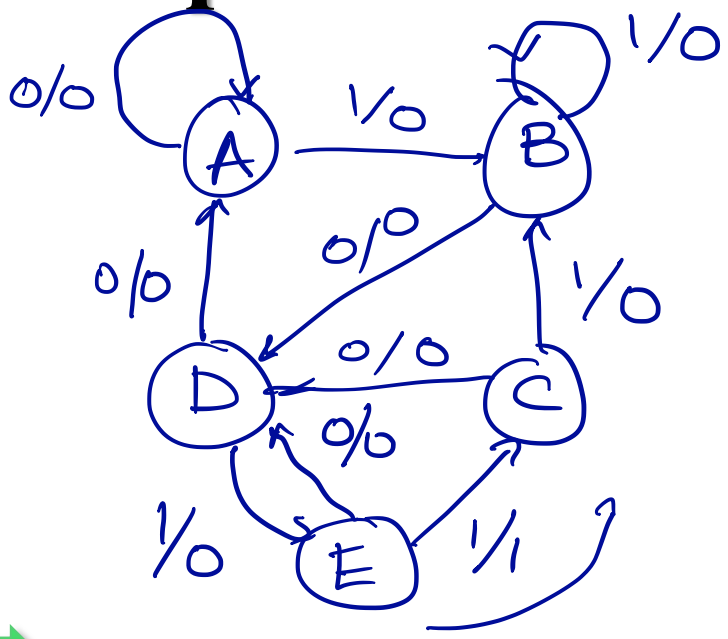
- To convert a Moore machine to a Mealy one:
- State Diagram – pull the outputs back in all incoming transitions



- State Table – duplicate the outputs of any state to all occurrences of that state as a next-state

Moore to Mealy Example



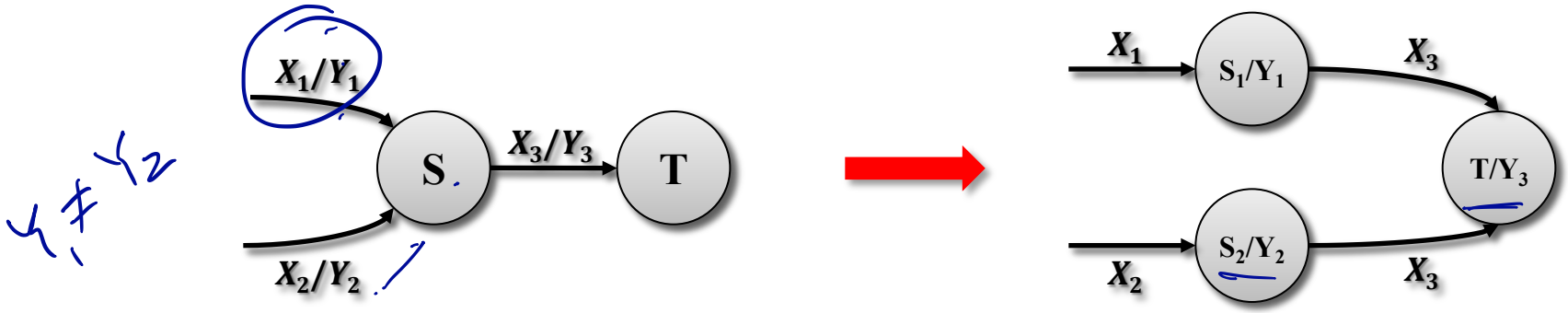


Current State	Next State X = 0	Next State X = 1	Output Z
A	A	B	0
B	D	B	0
C	D	B	1
D	A	E	0
E	D	C	0

Current State	Next State, Output Z X = 0		Next State, Output Z X = 1	
	NS	Z	NS	Z
A	A	0	B	0
B	D	0	B	0
C	D	0	B	0
D	A	0	E	0
E	D	0	C	1

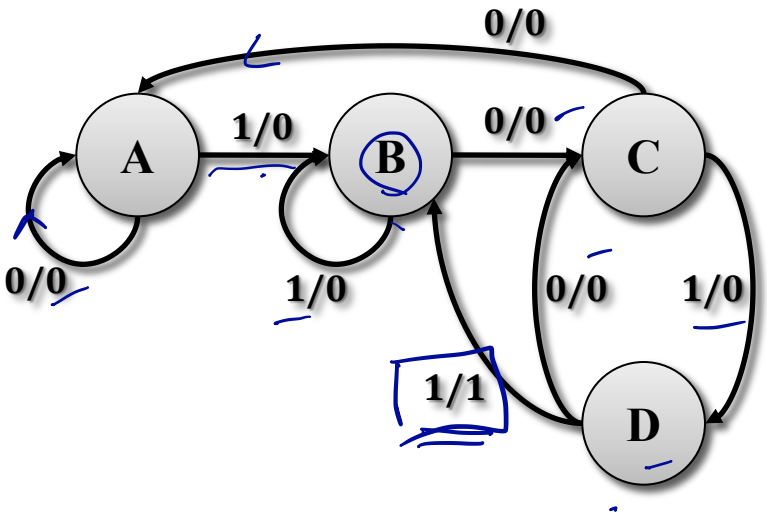
Mealy to Moore Conversion

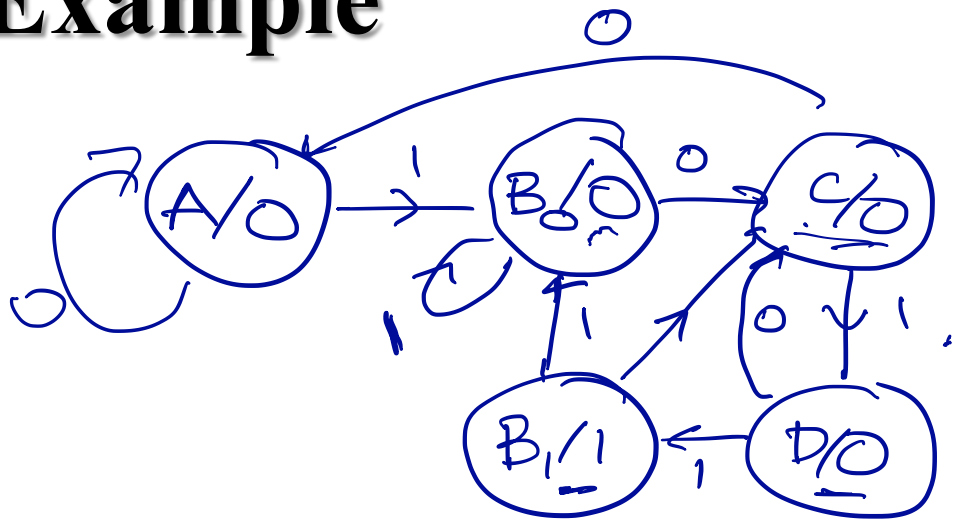
- To convert a Mealy machine to a Moore one:
- State Diagram – split each state to the number of different outputs coming in, and push the outputs into the state



- State Table – create new states corresponding to all state-output combinations and fill in a new table

Mealy to Moore Example





Current State	Next State, Output Z	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	B, 0
C	A, 0	D, 0
D	C, 0	B, 1

Current State	Next State		Output Z
	X = 0	X = 1	
A	A	B_0	0
B_0	C	B_0	0
B_1	C	B_0	1
C	A	D	0
D	C	B_1	0