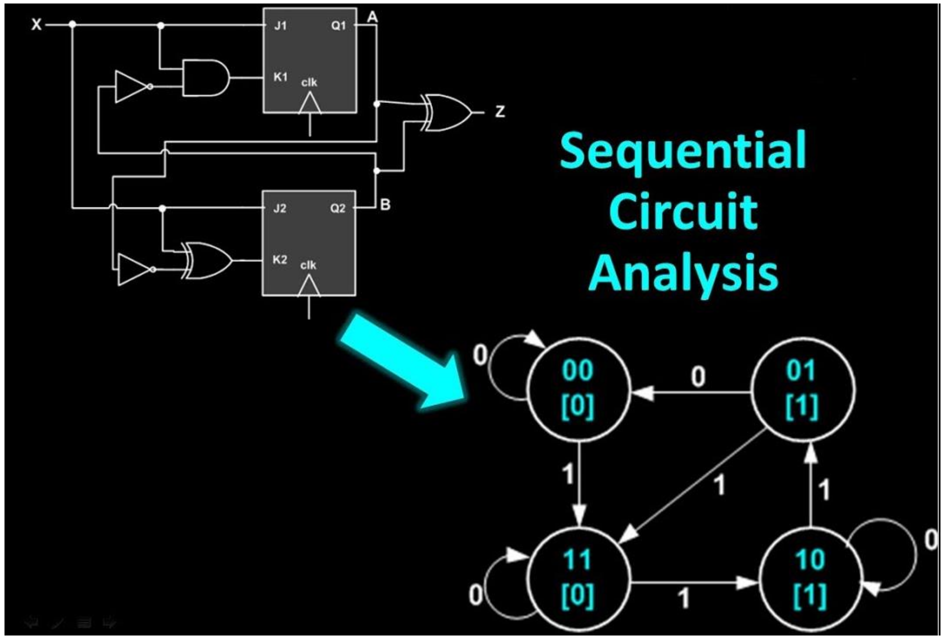
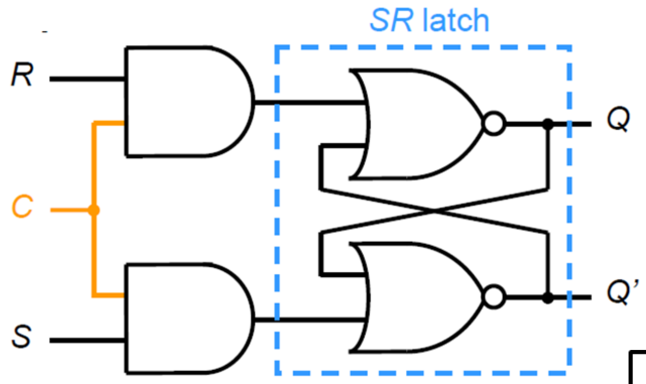
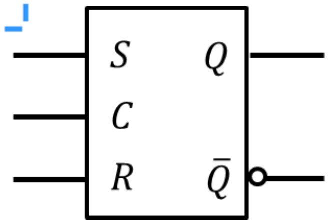
EE2000 Logic Circuit Design

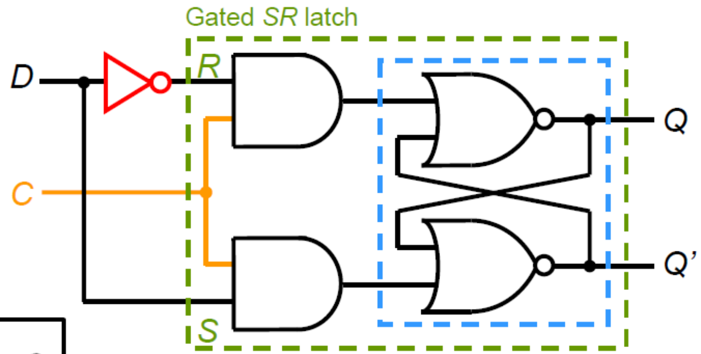
Lecture 9 – Sequential Logic Circuit Design

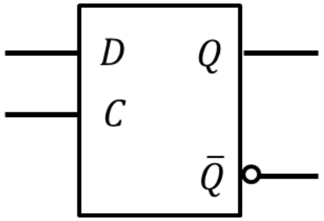


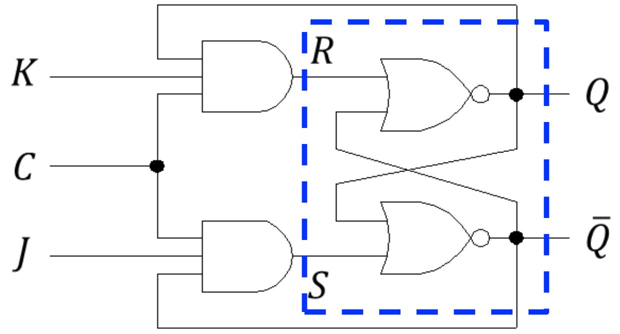
Flip-Flops

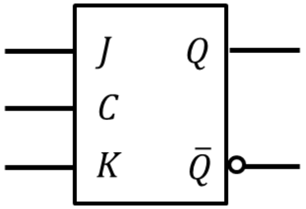

SR latch
R
C
S
Q
Q'

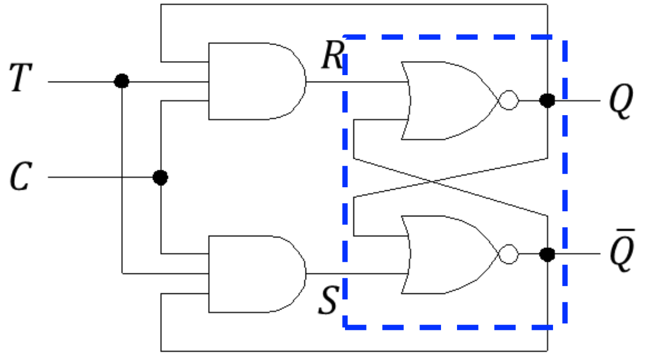

S
C
R
Q
$\bar{Q}$

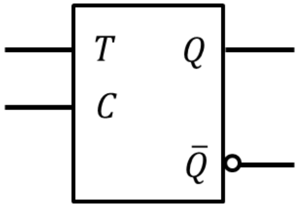

Gated SR latch
D
C
R
S
Q
Q'


D
C
Q
$\bar{Q}$


K
C
J
R
S
Q
$\bar{Q}$


J
C
K
Q
$\bar{Q}$


T
C
R
S
Q
$\bar{Q}$


T
C
Q
$\bar{Q}$

State Transition Tables

S	R	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	0	Q_t	$\overline{Q_t}$	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	1	1	Undefined

J	K	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	0	Q_t	$\overline{Q_t}$	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	$\overline{Q_t}$	Q_t	Toggle

D	Q_{t+1}	$\overline{Q_{t+1}}$	State
1	1	0	Set
0	0	1	Reset

T	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	Q_t	$\overline{Q_t}$	Hold
1	$\overline{Q_t}$	Q_t	Toggle

What will you learn?

9.1 What are Finite State Machines

- Concept of States
- Mealy *vs* Moore machines
- State table and diagram

9.2 Learn to analyze a Sequential Circuit

9.3 Learn to design a Sequential Circuit

- Design example
- State minimization

Sequential and Combinational Circuits

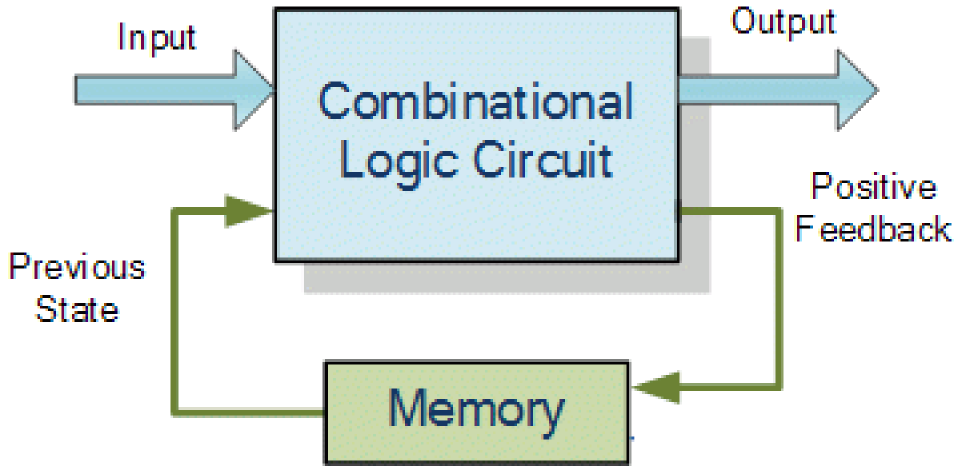
Combinational logic circuit

Output depends only on the inputs (As discussed in previous lectures)

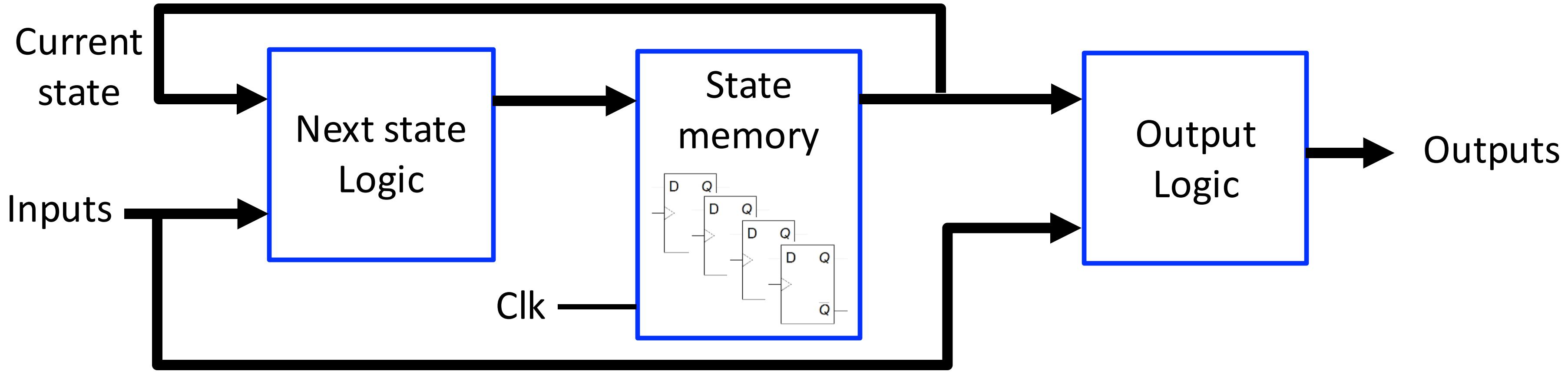
Sequential logic circuit

Output depends on present input + past history

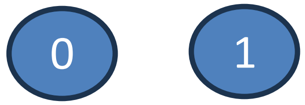
Memory circuit (to store previous STATE information) is required



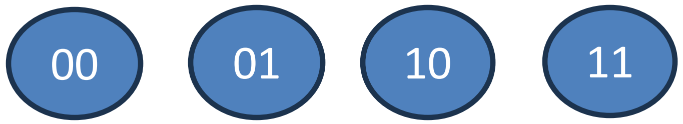
9.1 Finite State Machines - Concept



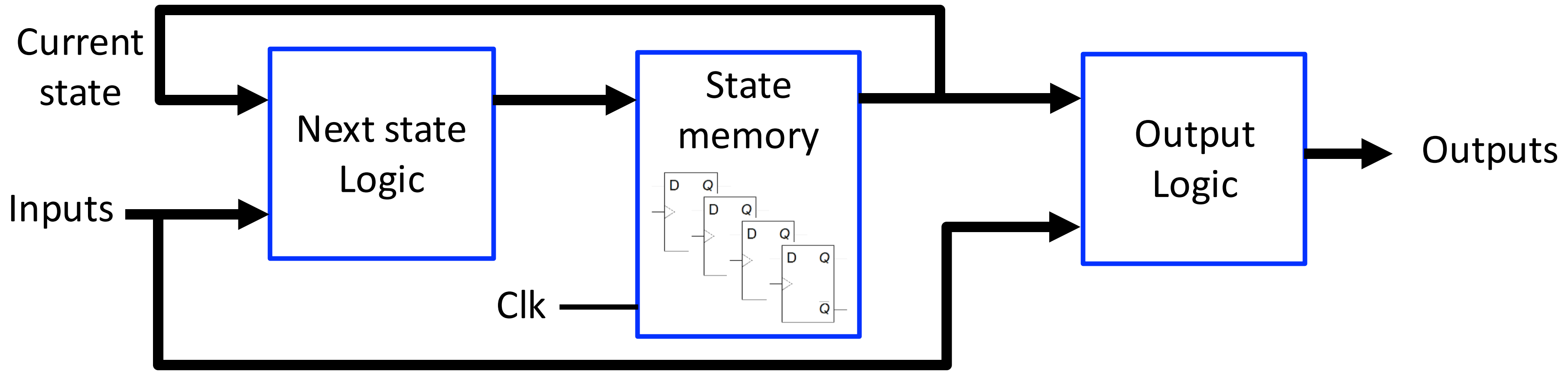
- A generic model/tool used in sequential circuit design
- State: Status of all memory units in the circuit. (*n* flip-flops → 2^n states)
  - 1 Flip-Flop: 0 or 1 (Two states)



  - 2 Flip-Flops: 0 & 0, 0 & 1, 1 & 0 or 1 & 1 (Four states)

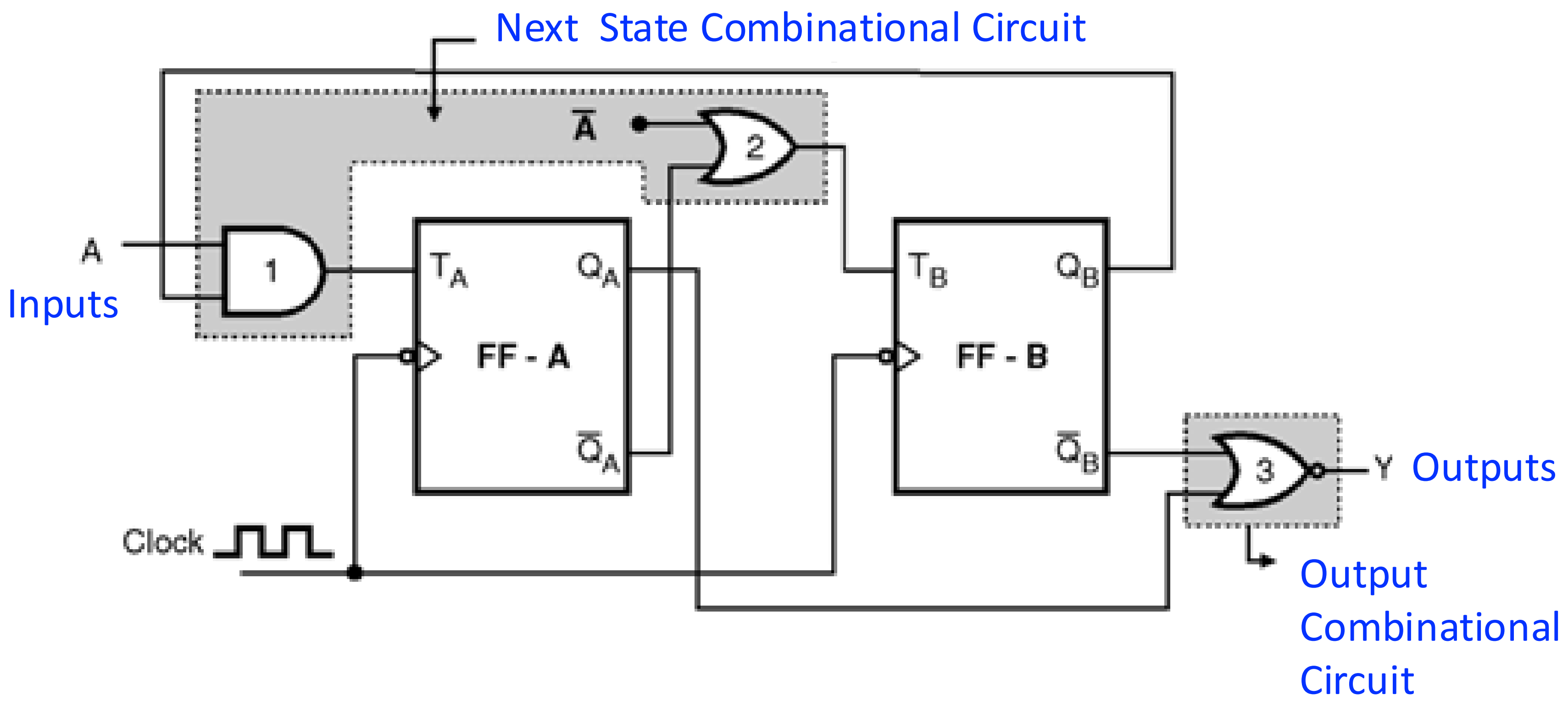


9.1 Finite State Machines - Concept



- Next State Logic: A combinational logic function to determine the next state of the system
- Output Logic: A combinational logic function to produce the outputs

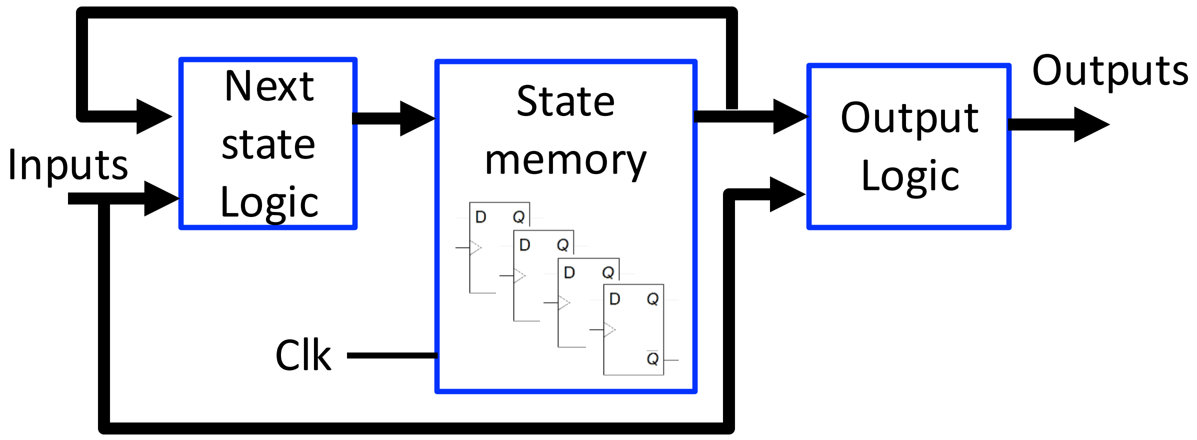
Example



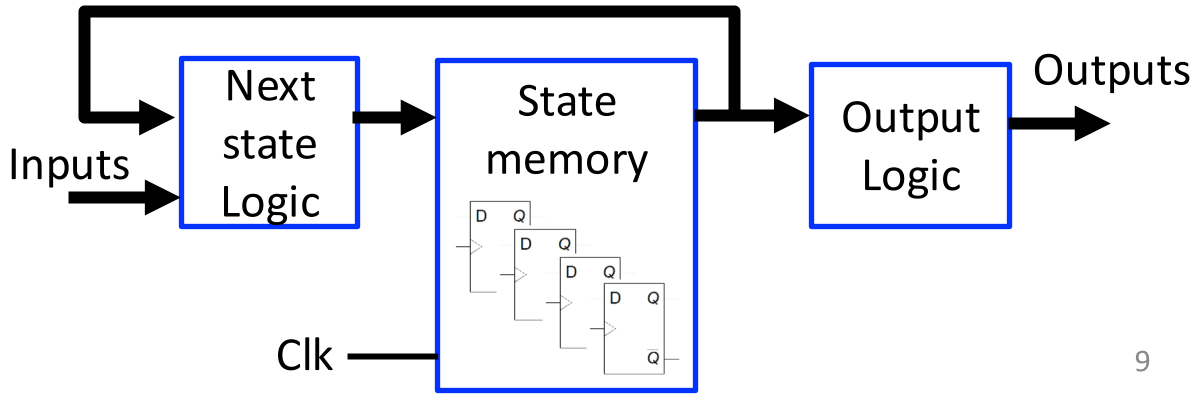
Next State Combinational Circuit
$\bar{A}$
2
A
1
Inputs
T_A
Q_A
FF - A
$\bar{Q}_A$
T_B
Q_B
FF - B
$\bar{Q}_B$
3
Y
Outputs
Clock
Output Combinational Circuit

Mealy and Moore Machines

Mealy machine Output depends on the current state and input

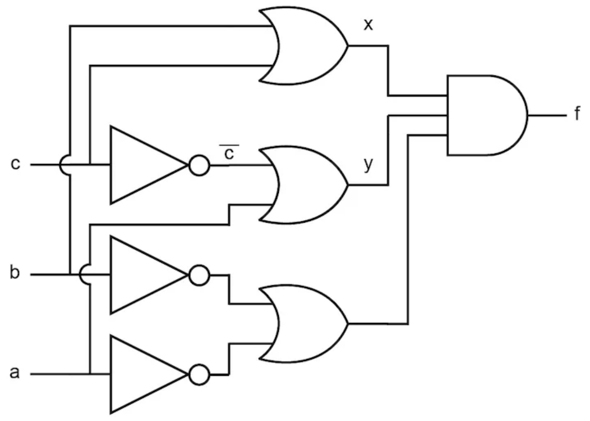


Moore machine Output depends only on the current state

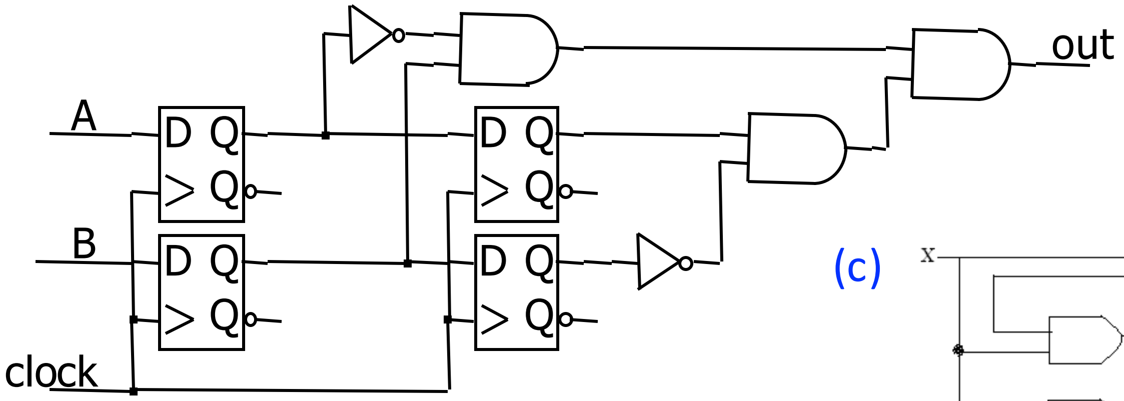


Examples (Mealy or Moore?)

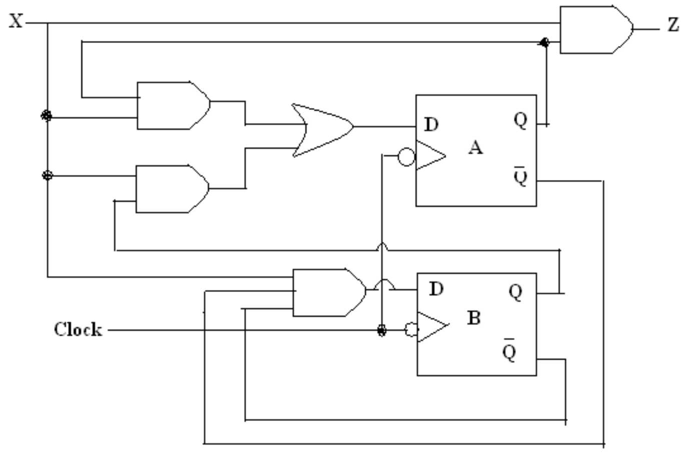
(a)



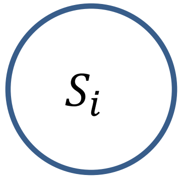
(b)



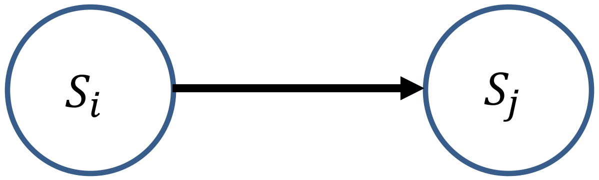
(c)



State Diagram

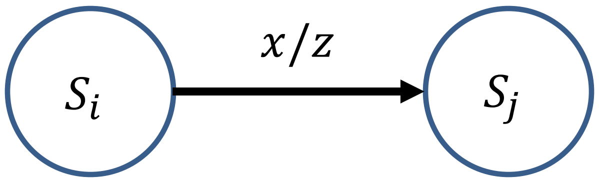


represents a state



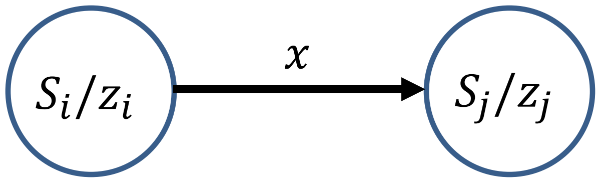
represents a transition from state S_i to S_j

Mealy machine



x is input
z is output
(output depends on input)

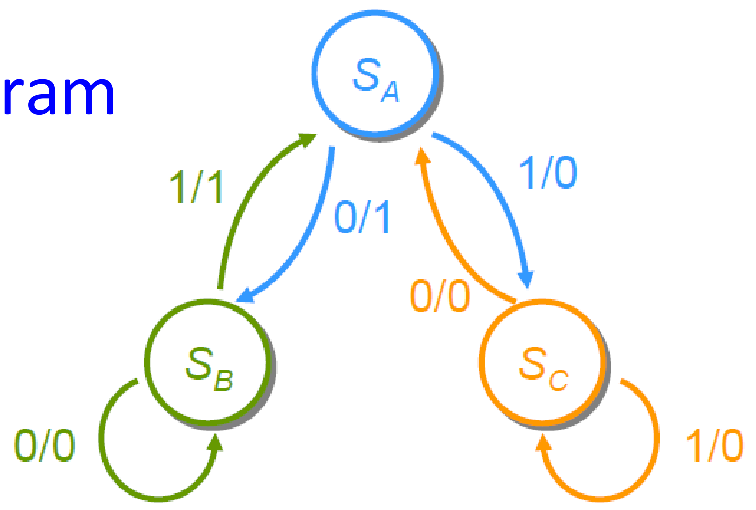
Moore machine



x is input
$z_i$is output
Output is independent on input

Example (Mealy Machine)

State diagram



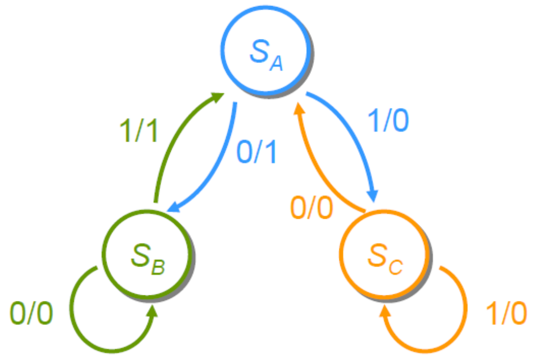
State table

Present State	**Next State**		**Output *Z***	
	Input *X* = 0	**Input *X* = 1**	**Input *X* = 0**	**Input *X* = 1**
S_A	S_B	S_C	1	0
S_B	S_B	S_A	0	1
S_C	S_A	S_C	0	0

Present State	**Input *X***	
	0	**1**
S_A	$S_B/1$	$S_C/0$
S_B	$S_B/0$	$S_A/1$
S_C	$S_A/0$	$S_C/0$

Example (Mealy Machine)

Given the initial state is S_A, work out how will the circuit behave with an input sequence of 011010.

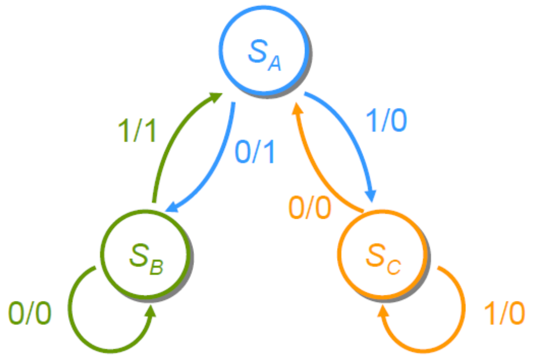


Present State	Next Stage		Output Z	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
S_A	S_B	S_C	1	0
S_B	S_B	S_A	0	1
S_C	S_A	S_C	0	0

Time	0	1	2	3	4	5
Present State (PS)	S_A					
Input X	0	1	1	0	1	0
Output Z						
Next State (NS)						

Example (Mealy Machine)

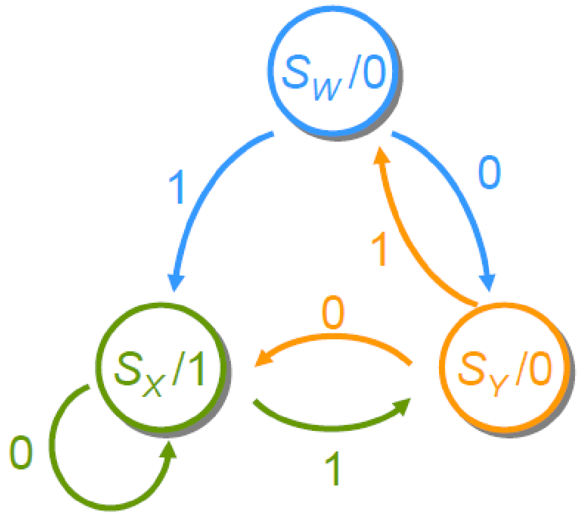
Given the initial state is S_A, work out how will the circuit behave with an input sequence of 011010.



Present State	Next Stage		Output Z	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
S_A	S_B	S_C	1	0
S_B	S_B	S_A	0	1
S_C	S_A	S_C	0	0

Time	0	1	2	3	4	5
Present State (PS)	S_A	S_B	S_A	S_C	S_A	S_C
Input X	0	1	1	0	1	0
Output Z	1	1	0	0	0	0
Next State (NS)	S_B	S_A	S_C	S_A	S_C	S_A

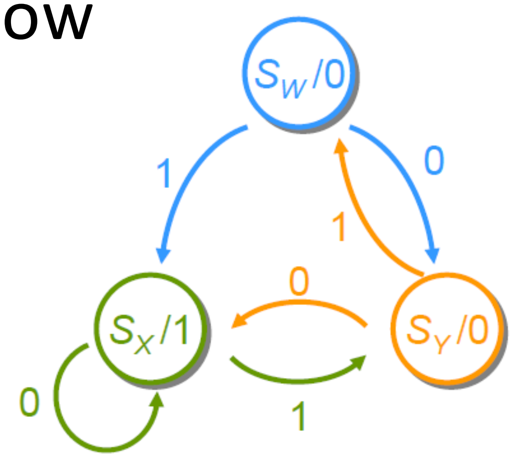
Example (Moore Machine)



Present State	Present Output Z	Input X	
		0	**1**
S_W	0	S_Y	S_X
S_X	1	S_X	S_Y
S_Y	0	S_X	S_W

Example (Moore Machine)

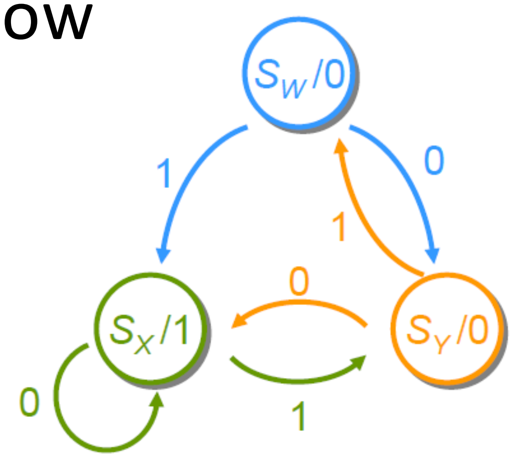
Given the initial state is S_W, work out how will the circuit behave with an input sequence of 011010.



Time	0	1	2	3	4	5
Present State (PS)	S_W					
Input X	0	1	1	0	1	0
Output Z						
Next State (NS)						

Example (Moore Machine)

Given the initial state is S_W, work out how will the circuit behave with an input sequence of 011010.



Time	0	1	2	3	4	5
Present State (PS)	S_W	S_Y	S_W	S_X	S_X	S_Y
Input X	0	1	1	0	1	0
Output Z	0	0	0	1	1	0
Next State (NS)	S_Y	S_W	S_X	S_X	S_Y	S_X

Tables (Example of SR FF)

Truth table

S	R	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

Outputs based on inputs

Excitation table

$S_t \quad S_{t+1}$	S	R
$S_0 \rightarrow S_0$	0	X
$S_0 \rightarrow S_1$	1	0
$S_1 \rightarrow S_0$	0	1
$S_1 \rightarrow S_1$	X	0

State transition based on inputs

S_0: $Q = 0$

S_1: $Q = 1$

State table

Present State	**Present Output** Q_t	**Input** $S\ R$			
		0 0	**0 1**	**1 0**	**1 1**
S_0	0	S_0	S_0	S_1	x
S_1	1	S_1	S_0	S_1	x

States, Input/Transition, Outputs

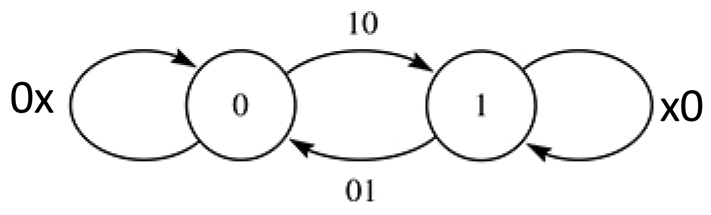
Example (SR-FF)

S	R	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	0	Q_t	$\overline{Q_t}$	Hold
0	1	**0**	**1**	Reset
1	0	**1**	**0**	Set
1	1	**0**	**0**	Undefined

State table

Present State	**Present Output Q_t**	**Input S R**			
		0 0	**0 1**	**1 0**	**1 1**
0	0	**0**	**0**	**1**	x
1	1	**1**	**0**	**1**	x

State diagram



Characteristic equation

$$Q_t^* = S + R'Q_t$$

Q_t^* Q_t \ $S\,R$	00	01	11	10
0	0	0	x	1
1	1	0	x	1

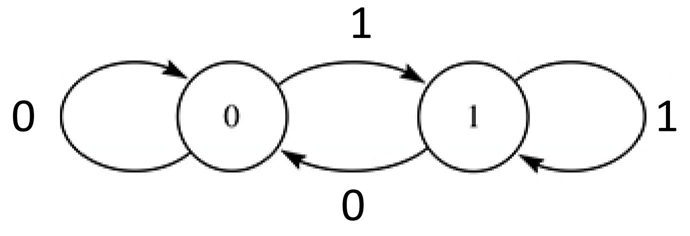
Other FFs

D	Q_{t+1}	$\overline{Q_{t+1}}$	State
1	1	0	Set
0	0	1	Reset

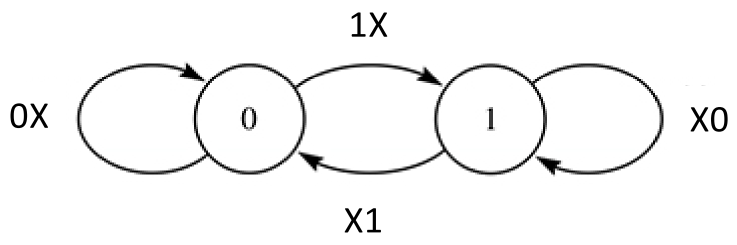
J	K	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	0	Q_t	$\overline{Q_t}$	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	$\overline{Q_t}$	Q_t	Toggle

T	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	Q_t	$\overline{Q_t}$	Hold
1	$\overline{Q_t}$	Q_t	Toggle

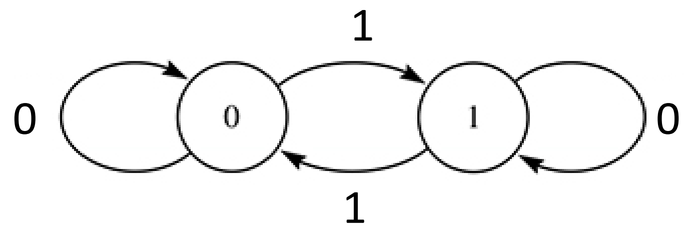
State diagram



D-FF



JK-FF



T-FF

Exercise (Other FFs)

State diagram

Characteristic equation

$$\begin{tikzcd}
0 \arrow[loop left, "0"] \arrow[r, bend left, "1"] & 1 \arrow[loop right, "1"] \arrow[l, bend left, "0"]
\end{tikzcd}$$

D-FF

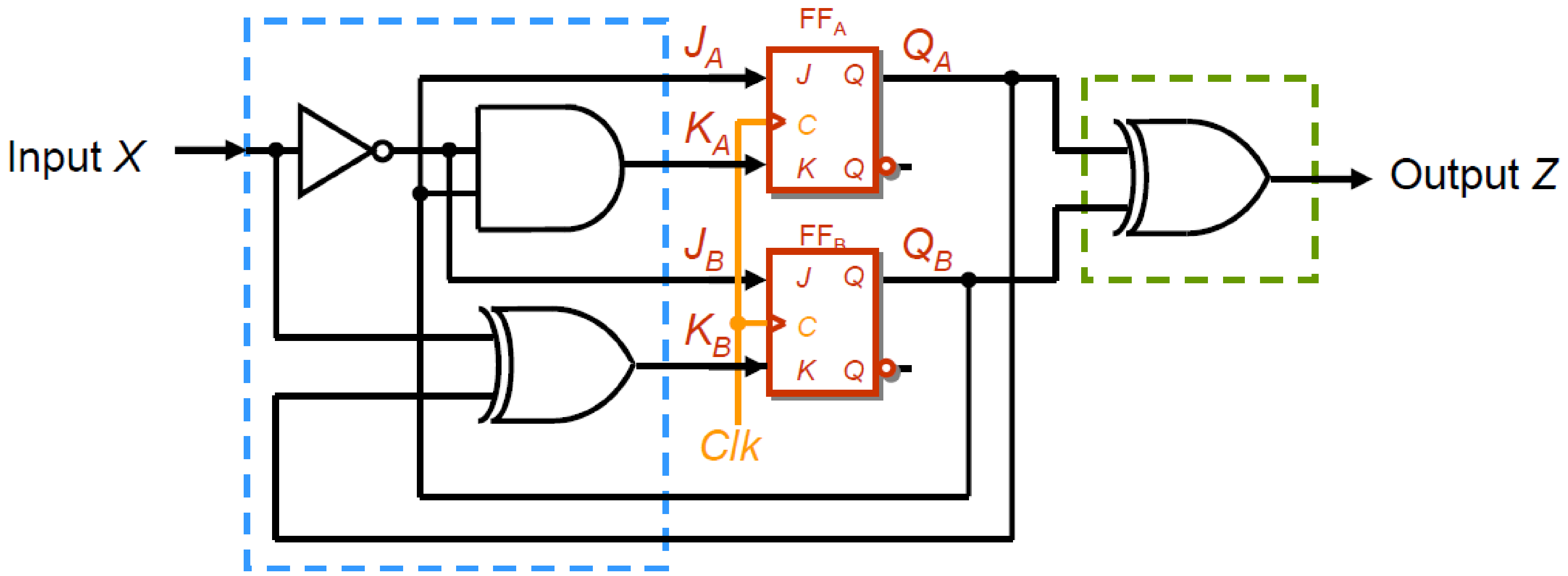
$$\begin{tikzcd}
0 \arrow[loop left, "\begin{matrix}00\\01\end{matrix}"] \arrow[r, bend left, "\begin{matrix}11\\10\end{matrix}"] & 1 \arrow[loop right, "\begin{matrix}00\\10\end{matrix}"] \arrow[l, bend left, "\begin{matrix}01\\11\end{matrix}"]
\end{tikzcd}$$

JK-FF

$$\begin{tikzcd}
0 \arrow[loop left, "0"] \arrow[r, bend left, "1"] & 1 \arrow[loop right, "0"] \arrow[l, bend left, "1"]
\end{tikzcd}$$

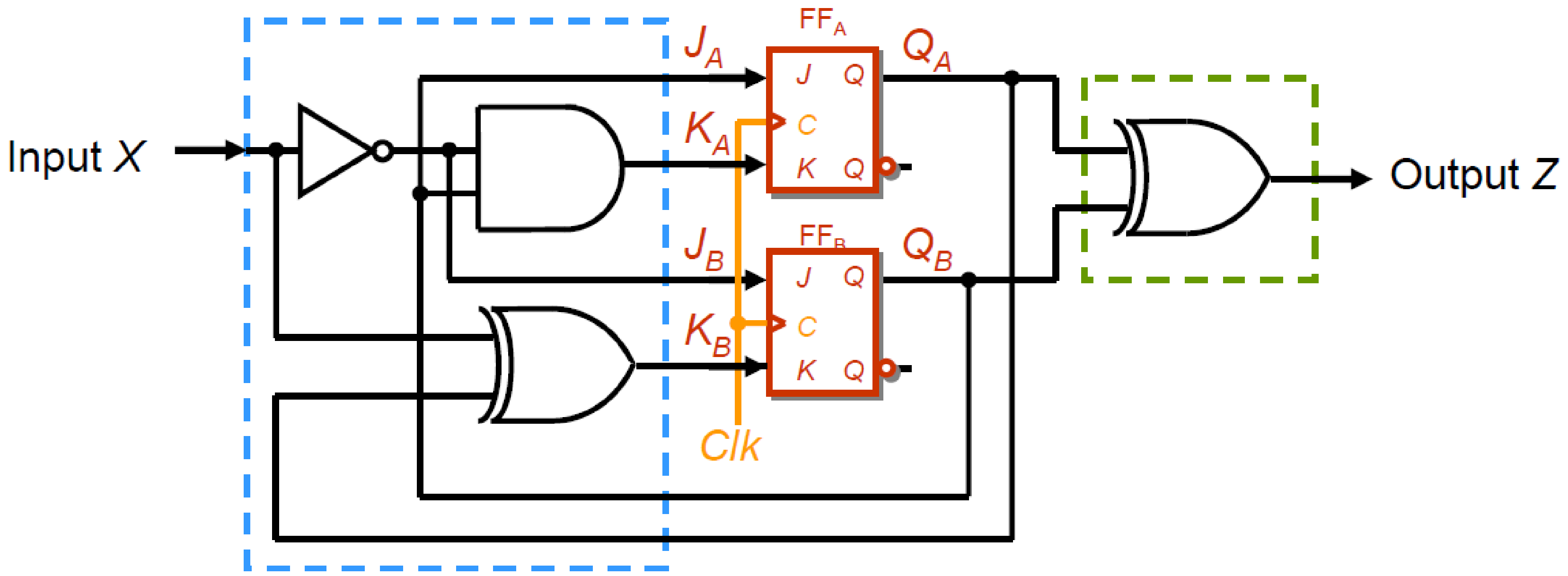
T-FF

9.2 Sequential Circuit Analyzer



Given a sequential circuit, analyze its behavior by producing the state diagram and state table.

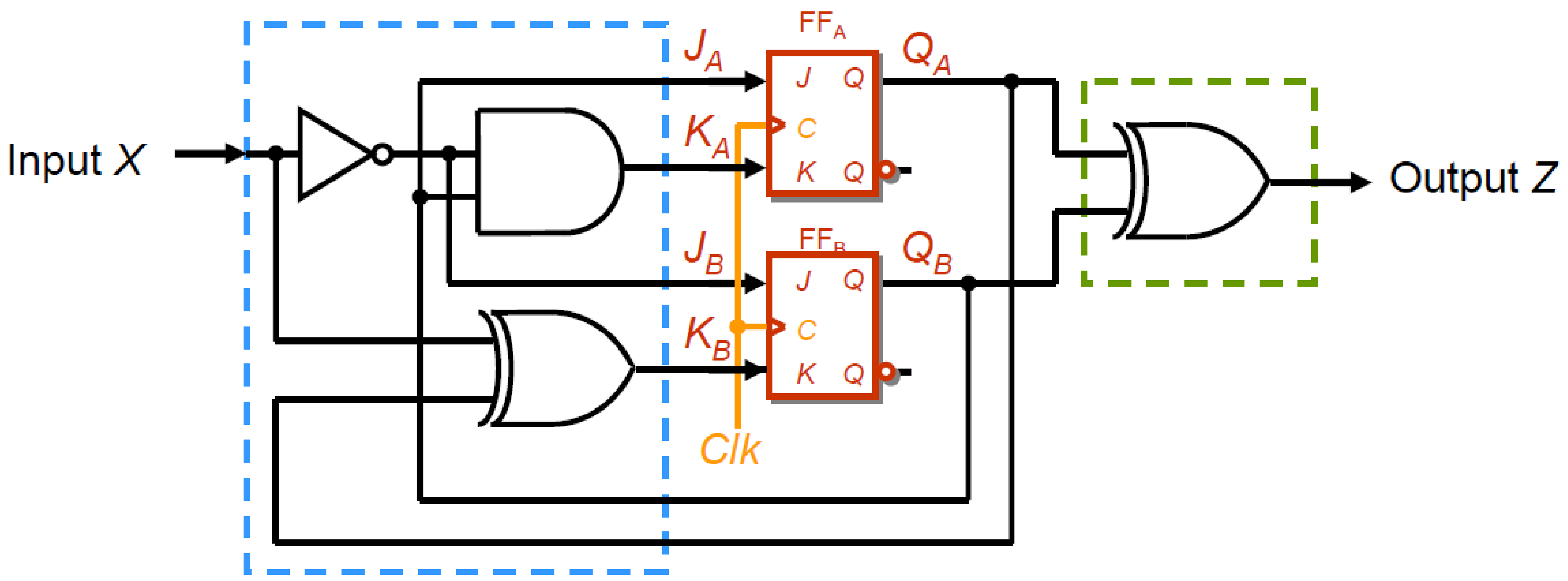
Example



STEP 1: Which state machine? Mealy or Moore?

STEP 2: Input(s)? Output(s)? No of FF(s)? How many states?

Example



STEP 3: Determine the flip-flop input function

$$J_A = Q_B,\ K_A = X'Q_B$$
$$J_B = X',\ K_B = X \oplus Q_A$$

STEP 4: Determine the output function

$$Z = Q_A \oplus Q_B$$

Example

$Z = Q_A \oplus Q_B$

STEP 5: Fill in the Analysis Table

Present State (PS)	Input	Present Output	Flip-Flops' Excitations				Next State (NS)
State $(Q_A Q_B)$	X	Z	J_A	K_A	J_B	K_B	$Q_A^* Q_B^*$
$(0\ \ 0)$	0						
	1						
$(0\ \ 1)$	0						
	1						
$(1\ \ 0)$	0						
	1						
$(1\ \ 1)$	0						
	1						

Example

STEP 6: Work out the State Table

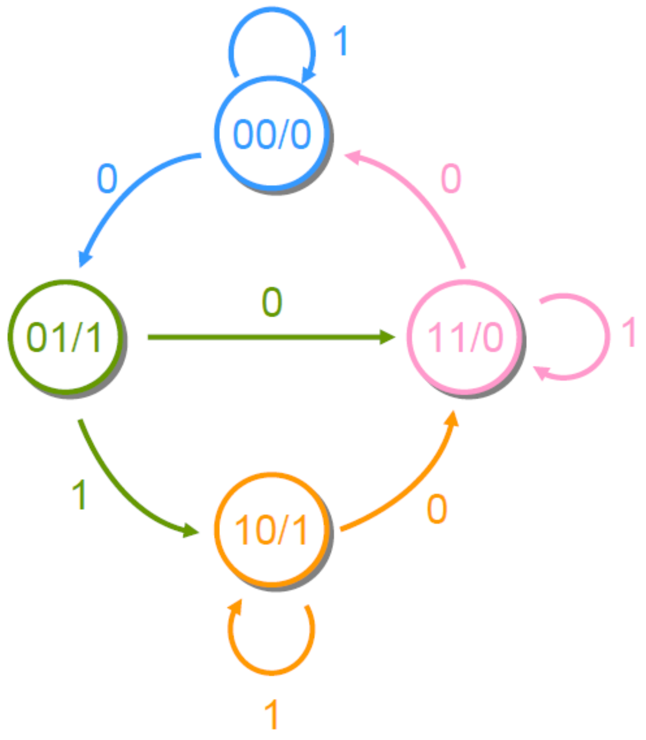
Present State (PS) State ($Q_A Q_B$)	Input X	Present Output Z	Next State (NS) $Q_A^* Q_B^*$
$(0\ \ 0)$	0	0	0 1
	1	0	0 0
$(0\ \ 1)$	0	1	1 1
	1	1	1 0
$(1\ \ 0)$	0	1	1 1
	1	1	1 0
$(1\ \ 1)$	0	0	0 0
	1	0	1 1

Present State	Present Output Z	Input X	
		0	1
$(0\ \ 0)$	0	$(0\ \ 1)$	$(0\ \ 0)$
$(0\ \ 1)$	1	$(1\ \ 1)$	$(1\ \ 0)$
$(1\ \ 0)$	1	$(1\ \ 1)$	$(1\ \ 0)$
$(1\ \ 1)$	0	$(0\ \ 0)$	$(1\ \ 1)$

Example

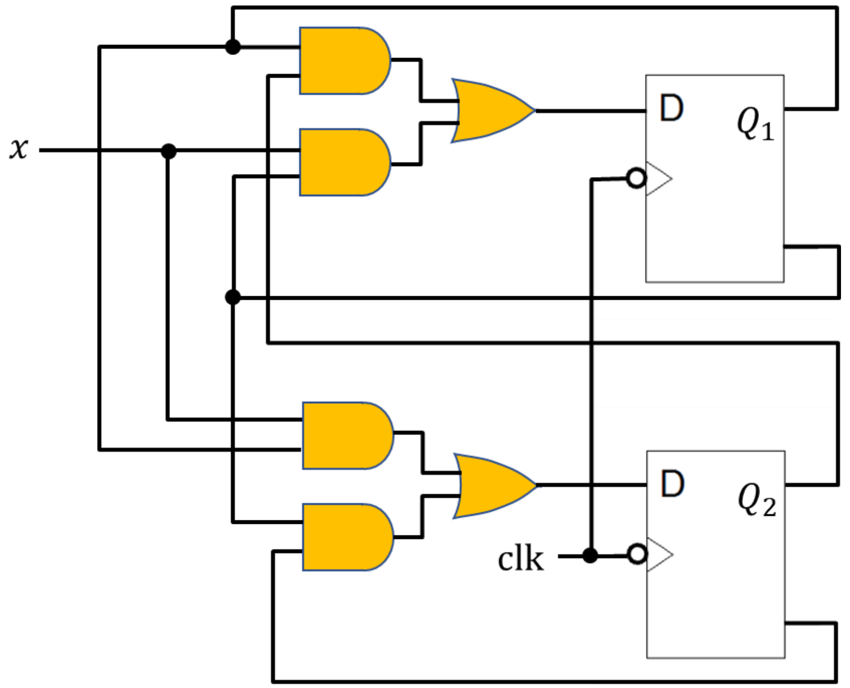
STEP 7: Work out the State Diagram

Present State	Present Output *Z*	Input *X*	
		0	**1**
(0 0)	0	(0 1)	(0 0)
(0 1)	1	(1 1)	(1 0)
(1 0)	1	(1 1)	(1 0)
(1 1)	0	(0 0)	(1 1)



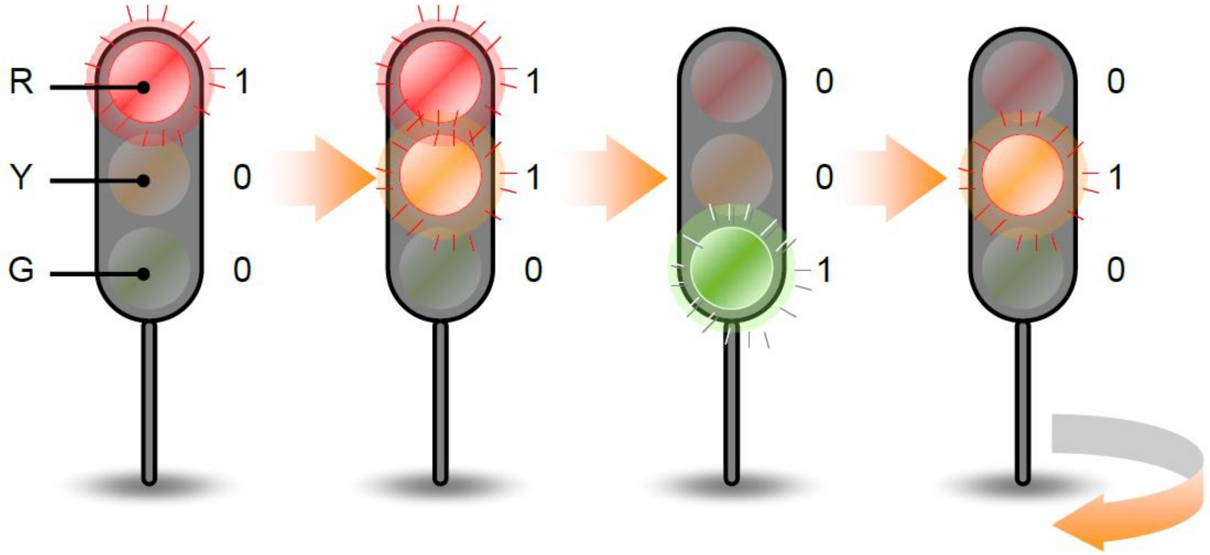
Exercise

Work out the State Table and State Diagram of the following circuit



Present State Q_1 Q_2	Input X 0	1
$(0\ \ 0)$		
$(0\ \ 1)$		
$(1\ \ 0)$		
$(1\ \ 1)$		

9.3 Sequential Circuit Design

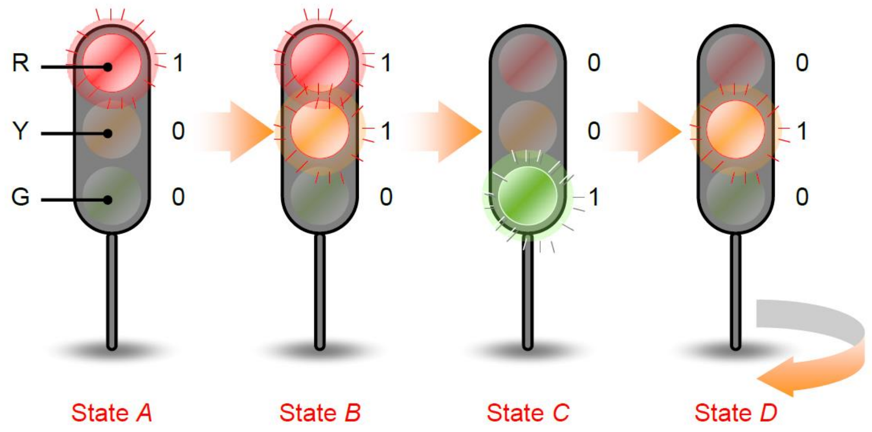


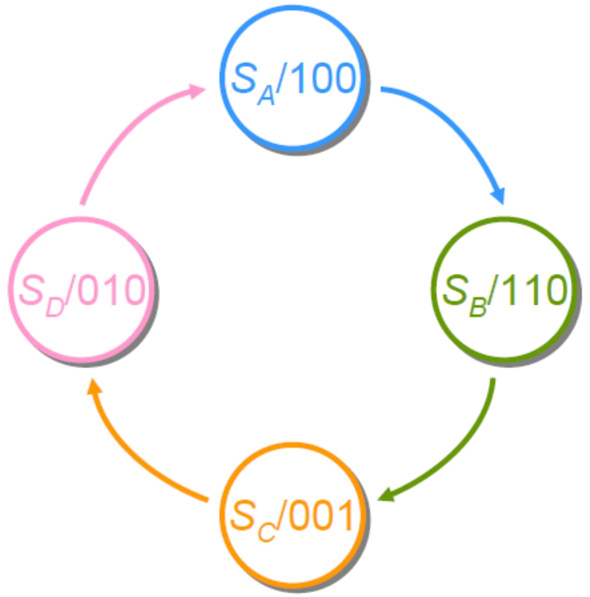
(a) How many outputs?

(b) How many states?

(c) Mealy or Moore machine?

Traffic Light Circuit





PS	NS	Outputs *R Y G*
S_A	S_B	1 0 0
S_B	S_C	1 1 0
S_C	S_D	0 0 1
S_D	S_A	0 1 0

Example (Sequence Detector)

Design a Moore machine to detect the sequence "111" (Overlapping)

In other words,

Design a system with one input x and one output z such that z = 1 if x has been 1 for at least three consecutive clock times.

x	0	1	1	0	1	1	1	0	1	1	1	1	1	0
z	0	0	0	0	0	0	0	1	0	0	0	1	1	1

Question: How about no overlapping is allowed?

Example (Sequence Detector)

x	0	1	1	0	1	1	1	0	1	1	1	1	1	0
z	0	0	0	0	0	0	0	1	0	0	0	1	1	1

STEP 1: Determine what needs to be stored in memory and how to store them.

A: Input is ‘0’

B: one ‘1’ is detected

C: two ‘1’s are detected

D: three ‘1’s are detected and output 1

Example (Sequence Detector)

x	1	1	1	1	1	1	0
z	0	0	0	1	0	0	1

STEP 2: Work out the State Diagram

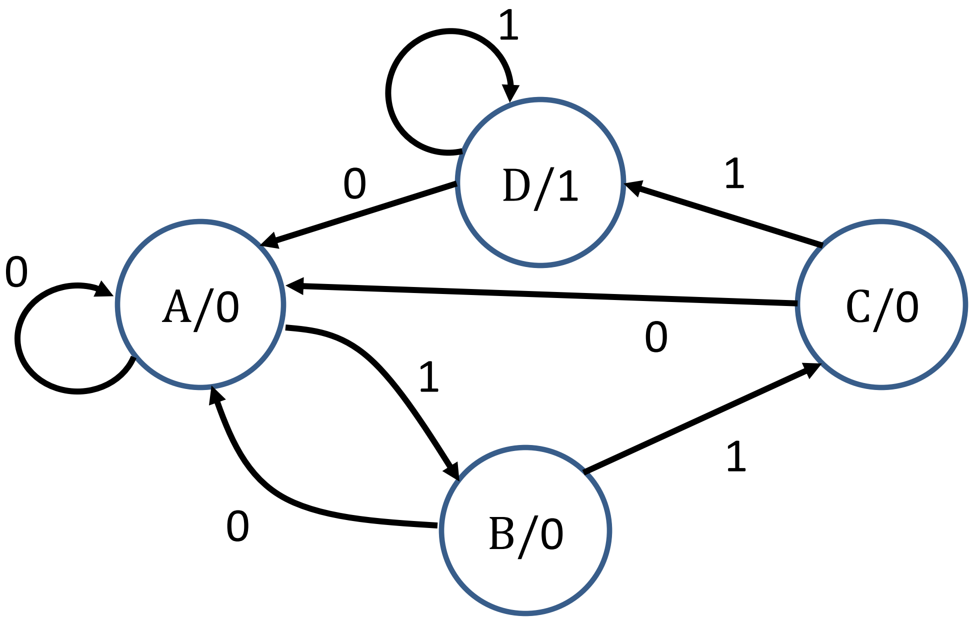
A: Input is ‘0’

B: one ‘1’ is detected

C: two ‘1’s are detected

D: three ‘1’s are detected and output 1

Question: How can we change the diagram if no overlapping is allowed?

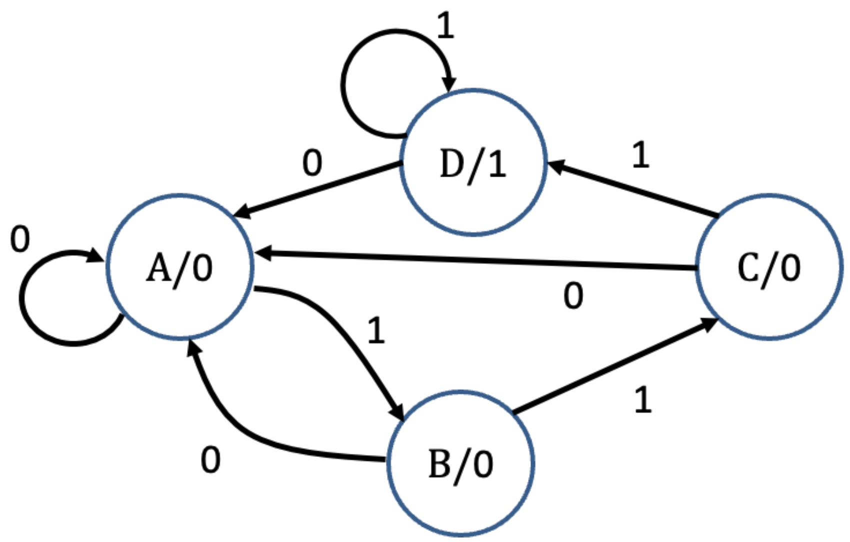


$$\begin{tikzcd}
 & D/1 \arrow[loop above, "1"] \arrow[dl, "0"'] & \\
A/0 \arrow[loop left, "0"] \arrow[dr, bend left, "1"] & & C/0 \arrow[ul, "1"'] \arrow[ll, "0"] \\
 & B/0 \arrow[ul, bend left, "0"] \arrow[ur, "1"'] &
\end{tikzcd}$$

Example (Sequence Detector)

STEP 3: Work out analysis table with assigned FFs

4 states → 2 FFs (We use D-FFs in this example)



Present State (Q_1Q_2)	Input X	Present Output Z	Next State Q_1^*	Next State Q_2^*
A (0 0)	0	0	0	0
A (0 0)	1	0	0	1
B (0 1)	0	0	0	0
B (0 1)	1	0	1	0
C (1 0)	0	0	0	0
C (1 0)	1	0	1	1
D (1 1)	0	1	0	0
D (1 1)	1	1	1	1

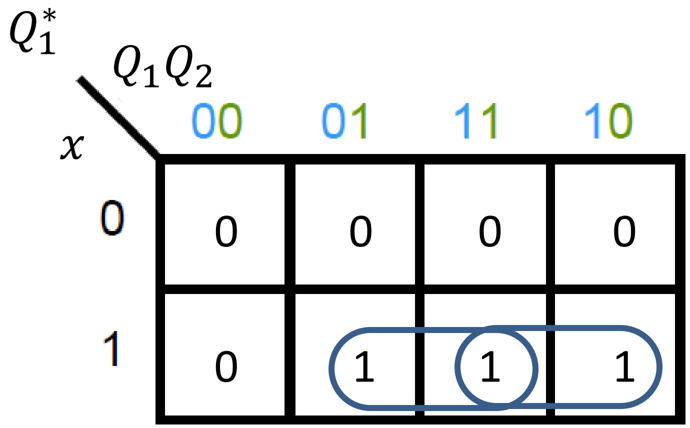
Assign State A:

$Q_1 \rightarrow 0$ and $Q_2 \rightarrow 0$ etc

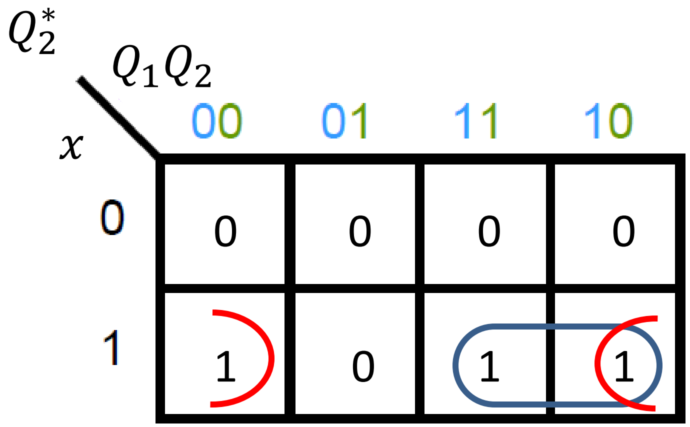
Example (Sequence Detector)

STEP 4: Work out D_1 and D_2

Present State (Q_1Q_2)	**Input *X***	**Present Output *Z***	**Next State**	
			Q_1^*	Q_2^*
A (0 0)	0	0	0	0
A (0 0)	1	0	0	1
B (0 1)	0	0	0	0
B (0 1)	1	0	1	0
C (1 0)	0	0	0	0
C (1 0)	1	0	1	1
D (1 1)	0	1	0	0
D (1 1)	1	1	1	1



$$D_1 = Q_1^* = x(Q_1+Q_2)$$



$$D_2 = Q_2^* = x(Q_1+Q_2')$$

Example (Sequence Detector)

STEP 5: Work out z

Present State (Q_1Q_2)	Input X	Present Output Z	Next State	
			Q_1^*	Q_2^*
A (0 0)	0	0	0	0
A (0 0)	1	0	0	1
B (0 1)	0	0	0	0
B (0 1)	1	0	1	0
C (1 0)	0	0	0	0
C (1 0)	1	0	1	1
D (1 1)	0	1	0	0
D (1 1)	1	1	1	1

z

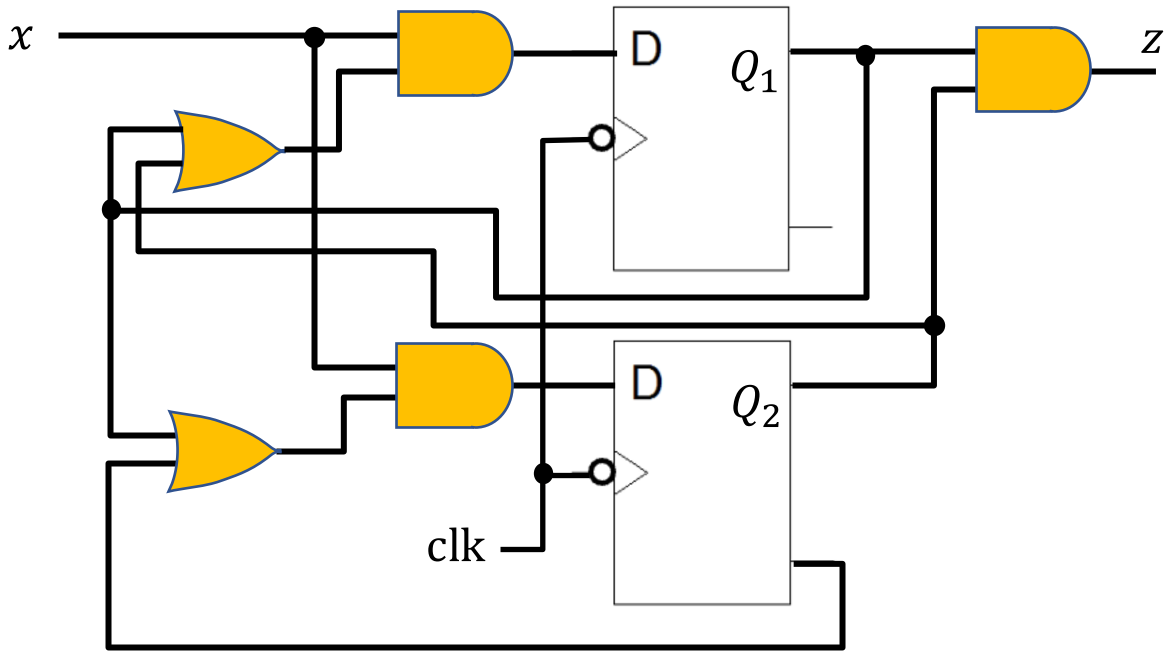
$x \backslash Q_1Q_2$	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$$z = Q_1Q_2$$

Example (Sequence Detector)

STEP 6: Draw the sequential logic circuits

$D_1 = x(Q_1 + Q_2)$ $D_2 = x(Q_1 + Q_2')$ $z = Q_1 Q_2$



Exercise (Sequence Detector)

Design a Mealy machine to detect the sequence “111” (Overlapping)

In other words,

Design a system with one input x and one output z such that z = 1 if x has been 1 for at least three consecutive clock times.

x	0	1	1	0	1	1	1	0	1	1	1	1	1	0
z	0	0	0	0	0	0	1	0	0	0	1	1	1	0

Exercise (Sequence Detector)

x	0	1	1	0	1	1	1	0	1	1	1	1	1	0
z	0	0	0	0	0	0	1	0	0	0	1	1	1	0

STEP 1: Determine what needs to be stored in memory and how to store them.

Exercise (Sequence Detector)

STEP 2: Work out the State Diagram

Exercise (Sequence Detector)

STEP 3: Work out the analysis table with assigned FFs

3 states → 2 FFs (We use D-FFs in this example)

Present State ($Q_1 Q_2$)	**Input *X***	**Next stage**		**Output *Z***
		Q_1^*	Q_2^*	
A (0 0)	0			
A (0 0)	1			
B (0 1)	0			
B (0 1)	1			
(1 0)	X			
C (1 1)	0			
C (1 1)	1			

Assign State A:

Q_1 → 0 and Q_2 → 0 etc

Exercise (Sequence Detector)

STEP 4: Work out D_1 and D_2

Present State (Q_1Q_2)	**Input** X	**Next State**		**Output** Z
		Q_1^*	Q_2^*	
A (0 0)	0			
A (0 0)	1			
B (0 1)	0			
B (0 1)	1			
(1 0)	x			
C (1 1)	0			
C (1 1)	1			

Q_1^*

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$D_1 = Q_1^* =$

Q_2^*

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$D_2 = Q_2^* =$

Exercise (Sequence Detector)

STEP 5: Work out z

Present State (Q_1Q_2)	**Input X**	**Next State**		**Output Z**
		Q_1^*	Q_2^*	
A (0 0)	0			
A (0 0)	1			
B (0 1)	0			
B (0 1)	1			
(1 0)	x			
C (1 1)	0			
C (1 1)	1			

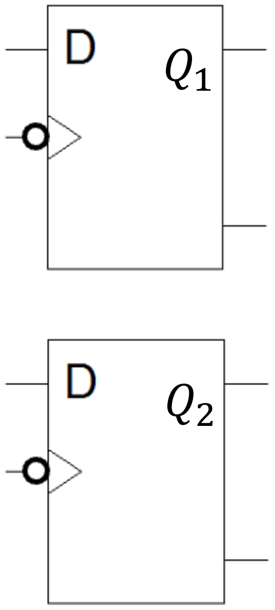
z

x \ Q_1Q_2	00	01	11	10
0				
1				

$z =$

Exercise (Sequence Detector)

STEP 6: Draw the sequential logic circuits



Exercise (Sequence Detector)

Use T FFs to design a Mealy machine to detect the sequence “111” (Overlapping)

Present State (Q_1Q_2)	**Input *X***	**Next State** Q_1^*	Q_2^*	**Flip-Flops** T_1	T_2	**Output *Z***
A (0 0)	0	0	0			0
A (0 0)	1	0	1			0
B (0 1)	0	0	0			0
B (0 1)	1	1	1			0
(1 0)	x	x	x			x
C (1 1)	0	0	0			0
C (1 1)	1	1	1			1

T	Q_{t+1}	$\overline{Q_{t+1}}$	State
0	Q_t	$\overline{Q_t}$	Hold
1	$\overline{Q_t}$	Q_t	Toggle

Exercise (Sequence Detector)

Present State	**Input X**	**Flip-Flops**	
(Q_1Q_2)		**T_1**	**T_2**
A (0 0)	0		
A (0 0)	1		
B (0 1)	0		
B (0 1)	1		
(1 0)	x		
C (1 1)	0		
C (1 1)	1		

T_1

x \ Q_1Q_2	00	01	11	10
0				
1				

$T_1 =$

T_2

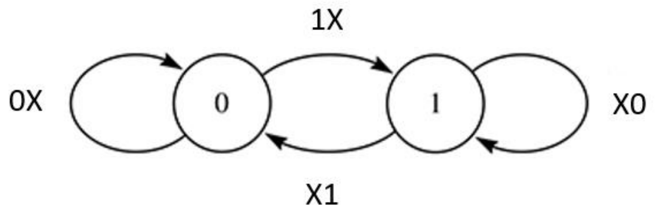
x \ Q_1Q_2	00	01	11	10
0				
1				

$T_2 =$

Exercise (Sequence Detector)

Use JK FFs to design a Mealy machine to detect the sequence “111” (Overlapping)

Present State (Q_1Q_2)	Input X	Next State Q_1^*	Q_2^*	Flip-Flops J_1	K_1	J_2	K_2	Output Z
A (0 0)	0	0	0					0
A (0 0)	1	0	1					0
B (0 1)	0	0	0					0
B (0 1)	1	1	1					0
(1 0)	x	x	x					x
C (1 1)	0	0	0					0
C (1 1)	1	1	1					1



Present State (Q_1Q_2)	Input X	Flip-Flops			
		J_1	K_1	J_2	K_2
A (0 0)	0				
A (0 0)	1				
B (0 1)	0				
B (0 1)	1				
(1 0)	x				
C (1 1)	0				
C (1 1)	1				

J_1

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$J_1 =$

K_1

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$K_1 =$

Present State (Q_1Q_2)	Input X	Flip-Flops			
		J_1	K_1	J_2	K_2
A (0 0)	0				
A (0 0)	1				
B (0 1)	0				
B (0 1)	1				
(1 0)	x				
C (1 1)	0				
C (1 1)	1				

J_2

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$J_2 =$

K_2

$x \backslash Q_1Q_2$	00	01	11	10
0				
1				

$K_2 =$

Mealy *vs* Moore Machines

Mealy machine

Present State	**Input *X***	
	0	**1**
A	A/0	B/0
B	A/0	C/0
C	A/0	C/1

Moore machine

Present State	**Input *X***		**Present Output *Z***
	0	**1**	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

Moore machine

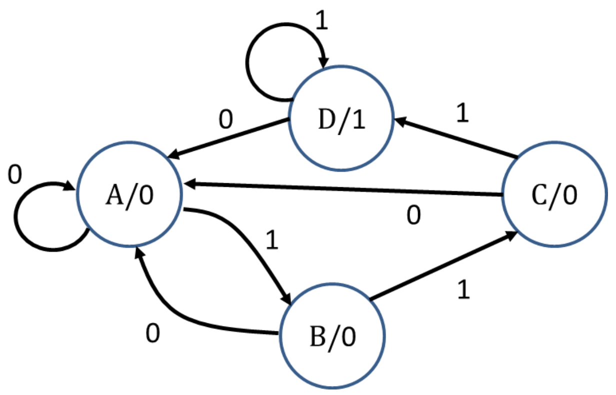
- Typically more states, more complex logic circuits

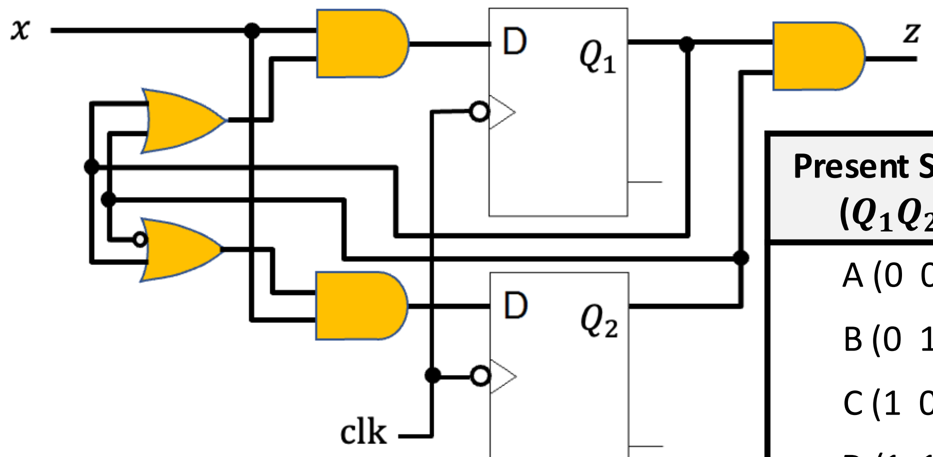
Mealy machine

+ Typically fewer states, simpler logic circuits

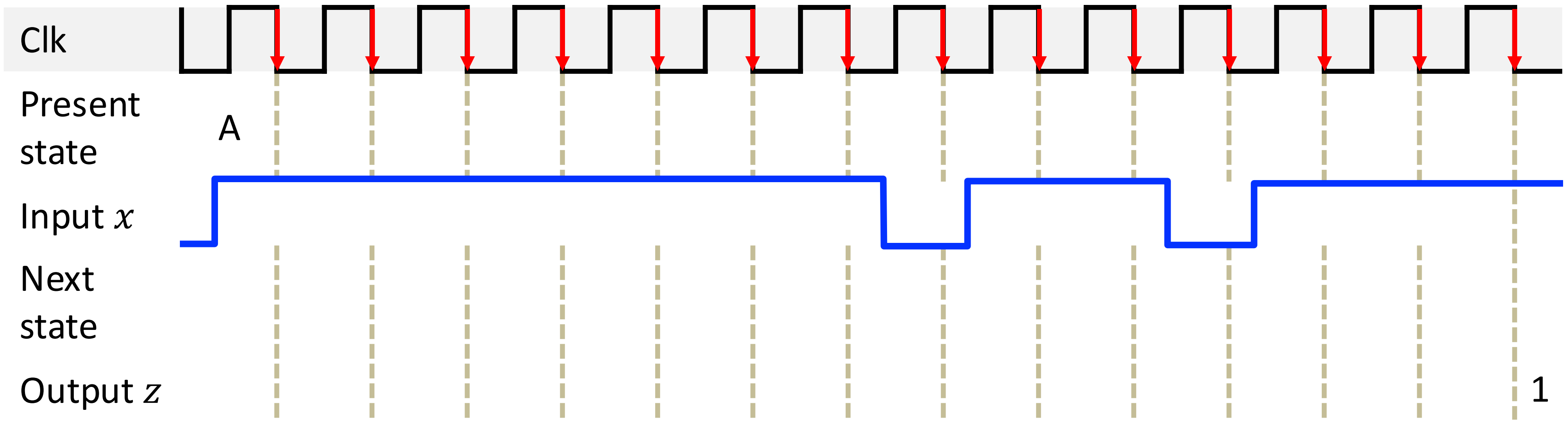
Example (Timing Diagram)

$z = Q_1 Q_2$



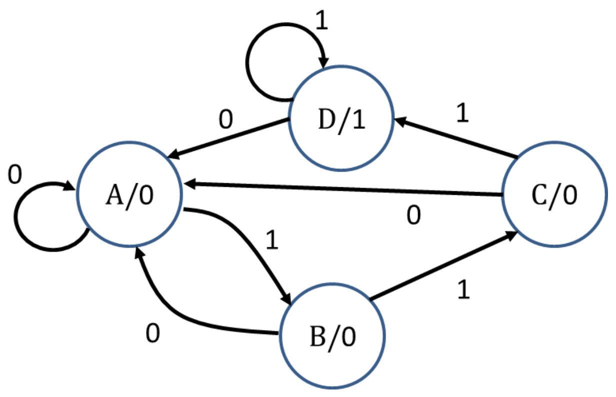


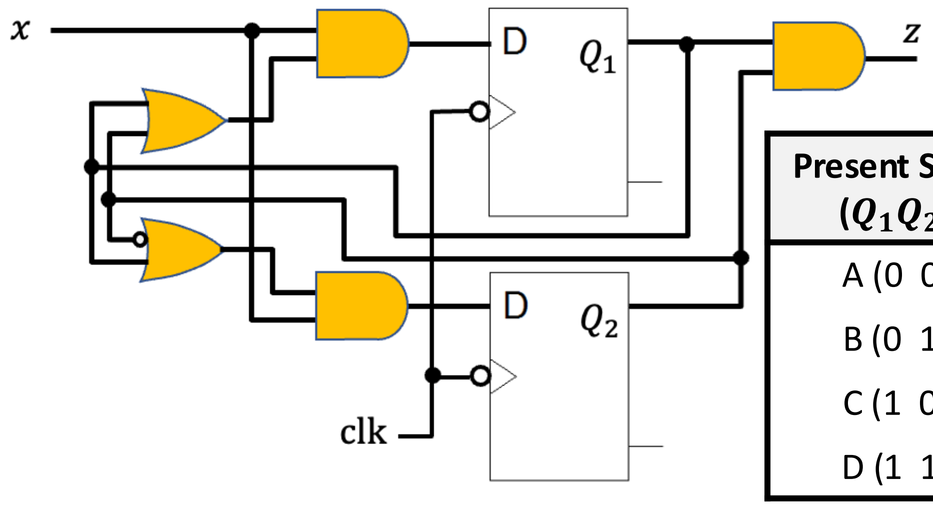
Present State $(Q_1 Q_2)$
A (0 0)
B (0 1)
C (1 0)
D (1 1)



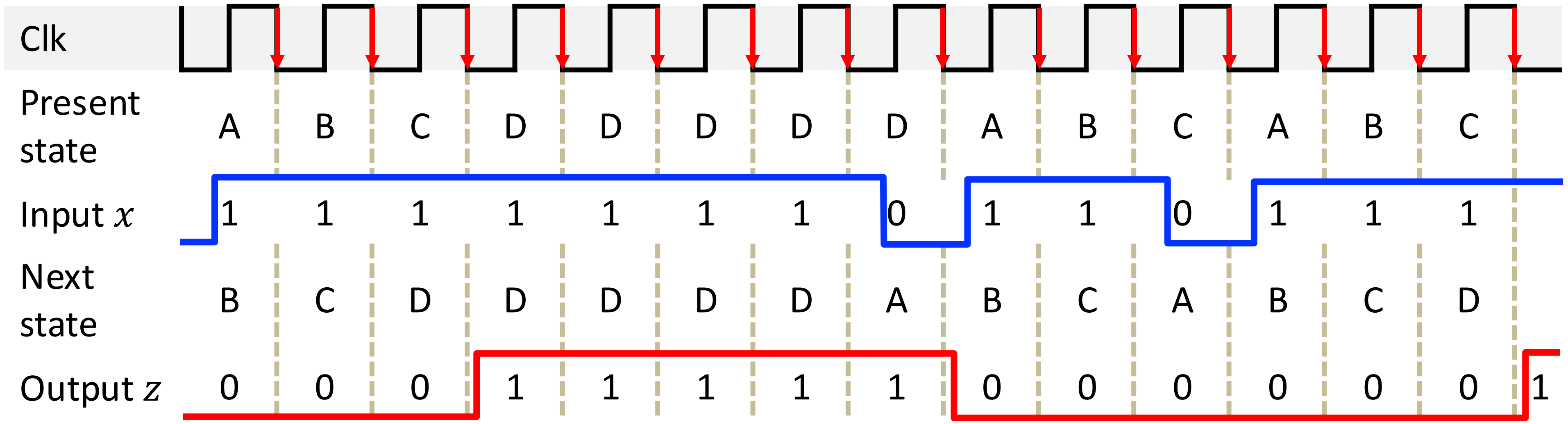
Example (Timing Diagram)

$$z = Q_1 Q_2$$



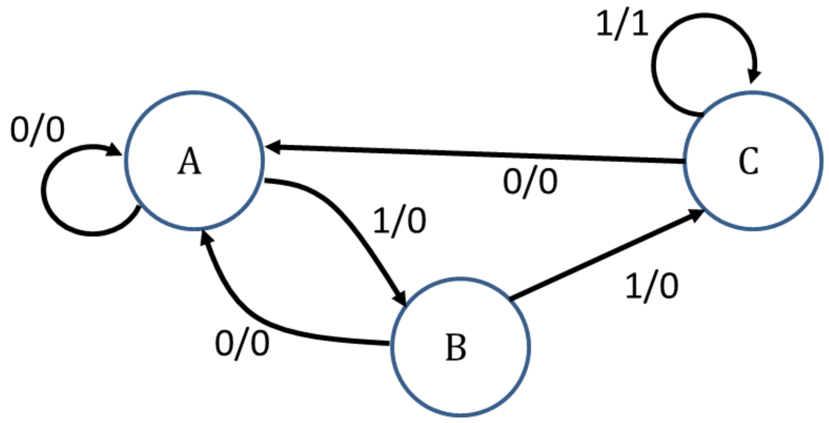


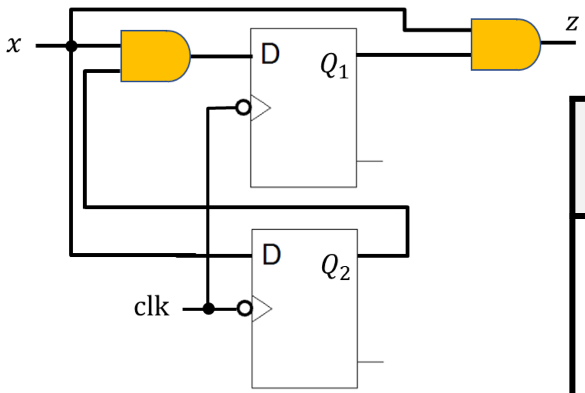
Present State $(Q_1 Q_2)$
A (0 0)
B (0 1)
C (1 0)
D (1 1)



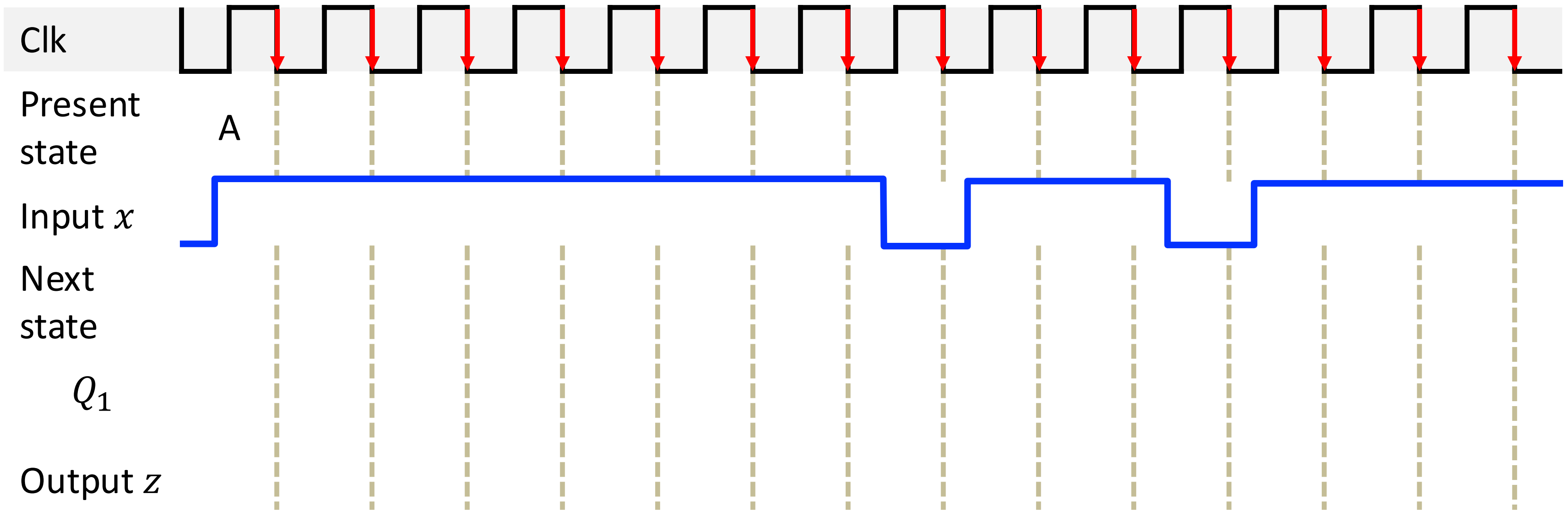
Exercise (Timing Diagram)

$z = xQ_1$

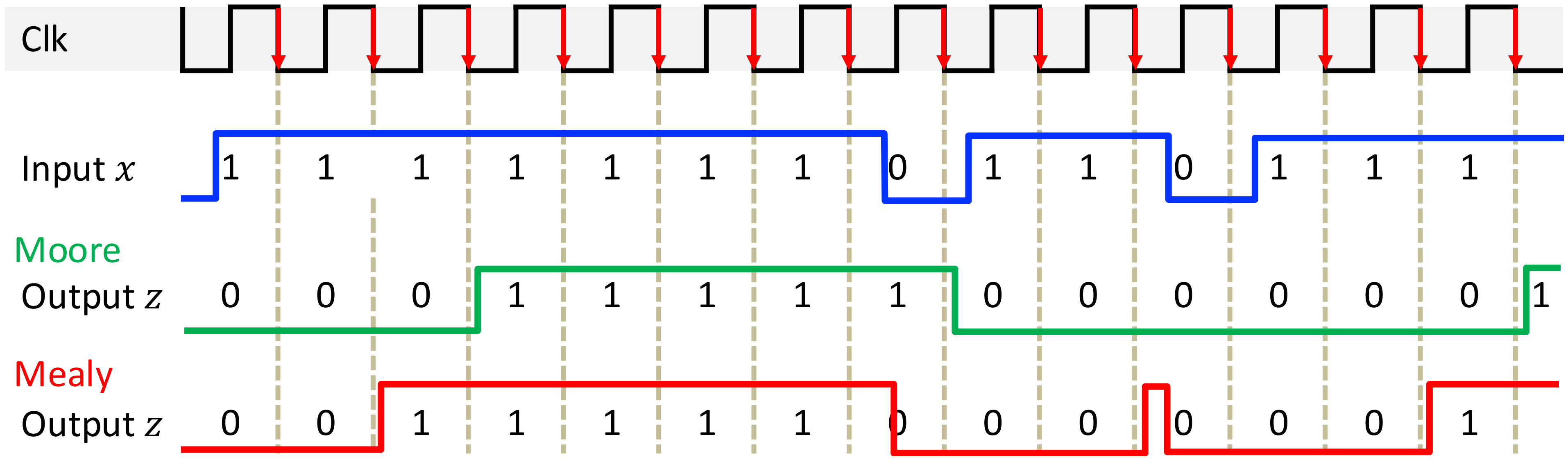




Present State (Q_1Q_2)
A (0 0)
B (0 1)
C (1 1)



Mealy *vs* Moore Machines



Observation

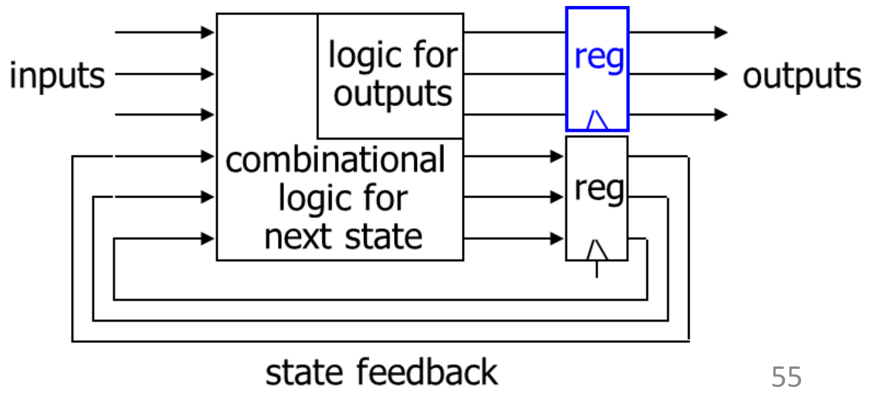
- Moore: Output changes occur only after clk edge
- Mealy: Output changes occur whenever input changes
- Mealy: Faster response but glitch might occurs

Mealy *vs* Moore Machines

Mealy machine	Moore machine
Output depends on inputs and present state	Output depends only on present state
Typically fewer states, simpler logic circuits	Typically more states, more complex logic circuits
React faster to inputs	React one clock cycle later
Asynchronous	Synchronous
Glitches might present	No glitch

Better solution?

Synchronous Mealy machine



Exercise (Sequence Detector)

Design a Moore machine to detect the sequence “11” or “000” (Overlapping)

In other words,

Design a system with one input x and one output z such that z = 1 if x has been 1 for at least two consecutive clock times or 0 for at least three consecutive clock times.

x	0	0	0	0	1	0	1	1	0	0	1	1	1	0
z	?	?	?	1	1	0	0	0	1	0	0	0	1	1

Exercise (Sequence Detector)

x	0	0	0	0	1	0	1	1	0	0	1	1	1	0
z	?	?	?	1	1	0	0	0	1	0	0	0	1	1

(Hint: 5 states)

Exercise (Sequence Detector)

Design a Mealy machine to detect the sequence “00110” (No overlapping)

In other words,

Design a system with one input x and one output z such that z = 1 if x has been 0 for two consecutive clock times, follows by two 1’s and then a 0.

x	0	0	1	1	0	0	1	1	0	0	1	1	0	1
z	0	0	0	0	1	0	0	0	0	0	0	0	1	0

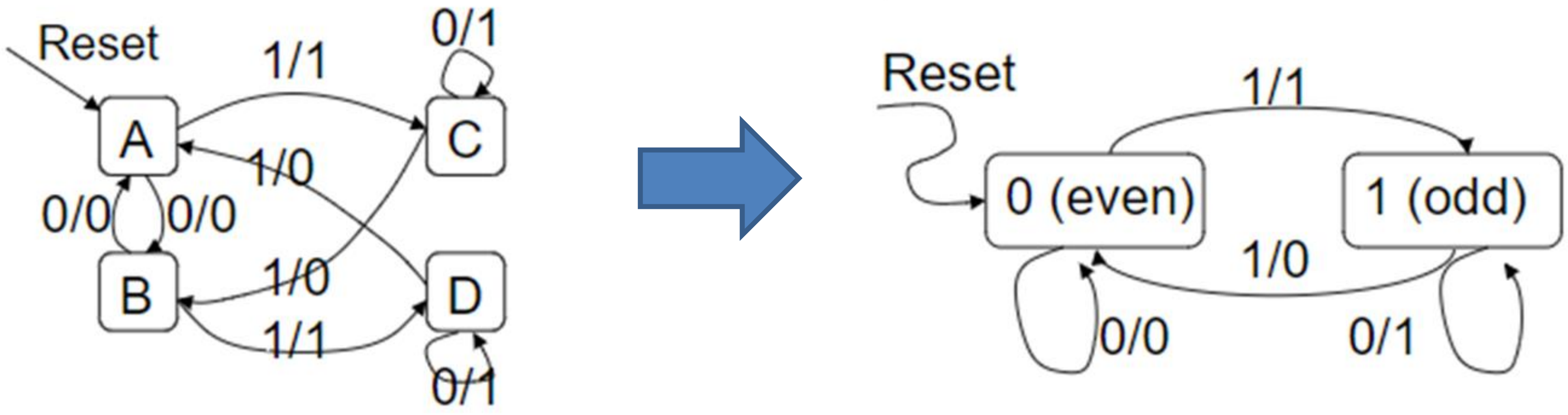
Exercise (Sequence Detector)

x	0	0	1	1	0	0	1	1	0	0	1	1	0	1
z	0	0	0	0	1	0	0	0	0	0	0	0	1	0

(Hint: 5 states)

State Minimization

- No of FFs ∝ No of states
- Combinational logic complexity ∝ No of states
- More FFs, more complex logic circuits → higher COST!
- Solution: Aims to remove redundant states



State Minimization

- Direct observation

Identify same output combinations and same state

	0	1
A	A/0	E/1
B	E/1	C/0
C	A/1	D/1
D	F/0	G/1
E	B/1	C/0
F	F/0	E/1
G	A/1	D/1

	0	1
A	A/0	E/1
B	E/1	C/0
C	A/1	D/1
D	F/0	G/1
E	B/1	C/0
F	F/0	E/1
G	A/1	D/1

	0	1
A	A/0	E/1
B	E/1	C/0
C	A/1	D/1
D	F/0	G/1
E	B/1	C/0
F	F/0	E/1
G	A/1	D/1

	0	1
A	A/0	E/1
B	E/1	C/0
C	A/1	D/1
D	F/0	G/1
E	B/1	C/0
F	F/0	E/1
G	A/1	D/1

	0	1
A=F	A/0	B/1
B=E	B/1	C/0
C=G	A/1	D/1
D	A/0	C/1

Partitioning Method

	0	1
A	A/0	E/1
B	E/1	C/0
C	A/1	D/1
D	F/0	G/1
E	B/1	C/0
F	F/0	E/1
G	A/1	D/1

- Separate states with different outputs to different partitions

$$P_0 = (\mathrm{A\ B\ C\ D\ E\ F\ G})$$

- A, D and F have outputs (0 1); B and E have outputs (1 0); C and G have outputs (1 1)

$$P_1 = (\mathrm{A\ D\ F})(\mathrm{B\ E})(\mathrm{C\ G})$$

Partitioning Method

P		0	1
1	A	A/0	E/1
	D	F/0	G/1
	F	F/0	E/1
2	B	E/1	C/0
	E	B/1	C/0
3	C	A/1	D/1
	G	A/1	D/1

$$P_1 = (\mathrm{A\ D\ F})(\mathrm{B\ E})(\mathrm{C\ G})$$

- Next check the next state of each state in each partition
  - A(0) → A (P1)   A(1) → E (P2)
  - D(0) → F (P1)   D(1) → G (P3)
  - F(0) → F (P1)   F(1) → E (P2)

  ∴ A and F same partitions; D is different

$$P_2 = (\mathrm{A\ F})(\mathrm{D})(\mathrm{B\ E})(\mathrm{C\ G})$$

Partitioning Method

P		0	1
1	A	A/0	E/1
	F	F/0	E/1
2	B	E/1	C/0
	E	B/1	C/0
3	C	A/1	D/1
	G	A/1	D/1
4	D	F/0	G/1

$$P_2 = (\mathrm{A\ F})(\mathrm{D})(\mathrm{B\ E})(\mathrm{C\ G})$$

This is the final with no more changes!

	I	J
A=F	A/0	B/1
B=E	B/1	C/0
C=G	A/1	D/1
D	A/0	C/1

Exercise

Present State	Input *X* 0	Input *X* 1	Present Output *Z*
A	I	C	0
B	B	I	0
C	C	G	0
D	I	C	1
E	D	E	1
F	I	C	1
G	E	F	1
H	H	A	0
I	A	C	0

Reduce the state table using partitioning method

$$P_0 = (\text{A B C D E F G H I})$$

Group states based on same output

Exercise

Present State	Input X		Present Output Z
	0	1	

Reduce the state table using partitioning method

$$P_1 =$$

Exercise

Present State	Input X		Present Output Z
	0	1	

Reduce the state table using partitioning method

$P_2 =$

Exercise

Present State	Input X		Present Output Z
	0	1	

Reduce the state table using partitioning method

$P_3 =$

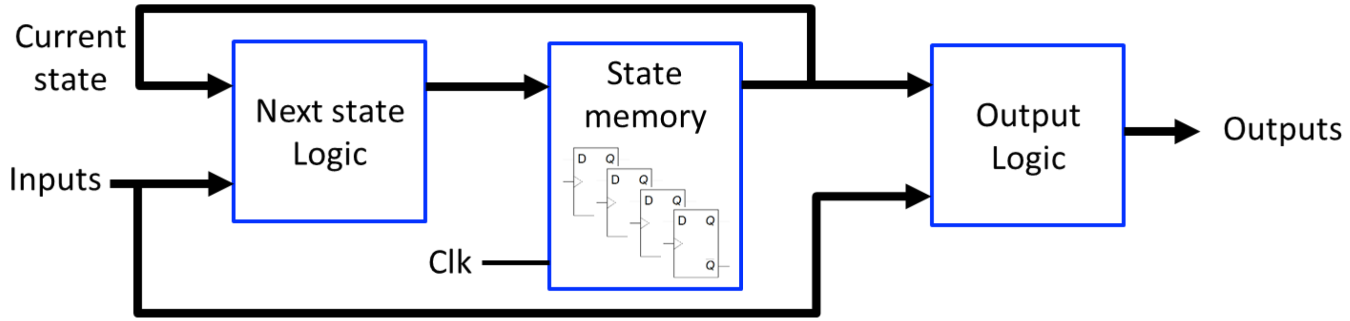
Exercise

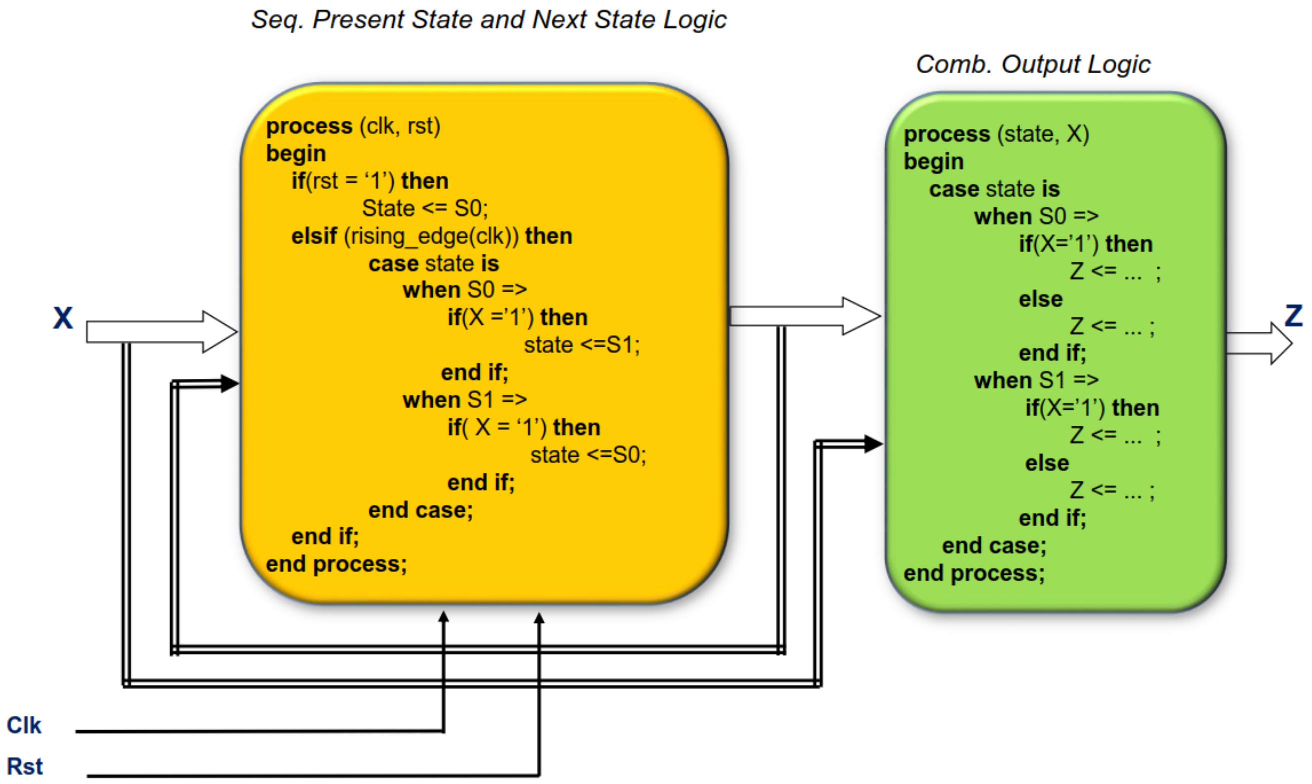
Present State	Input *X*		Present Output *Z*
	0	1	

Reduce the state table using partitioning method

$P_3 =$

VHDL for Finite State Machine

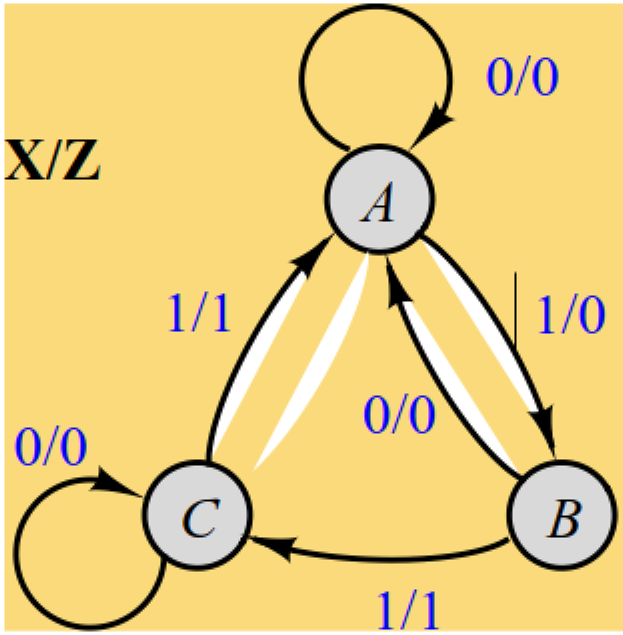




```
process (clk, rst)
begin
   if(rst = '1') then
           State <= S0;
   elsif (rising_edge(clk)) then
           case state is
              when S0 =>
                  if(X ='1') then
                         state <=S1;
                 end if;
              when S1 =>
                  if( X = '1') then
                          state <=S0;
                  end if;
           end case;
   end if;
end process;
```

```
process (state, X)
begin
   case state is
       when S0 =>
           if(X='1') then
               Z <= ... ;
           else
               Z <= ... ;
           end if;
       when S1 =>
           if(X='1') then
               Z <= ... ;
           else
               Z <= ... ;
           end if;
   end case;
end process;
```

Example

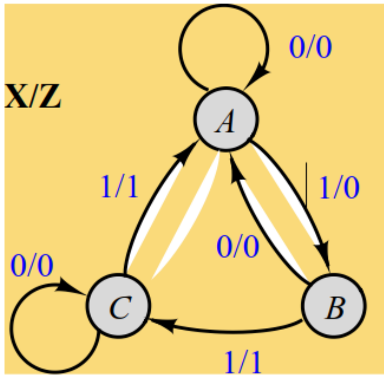


Present state	Input x 0	1
A	A/0	B/0
B	A/0	C/1
C	C/0	A/1

Next state/output

```
entity seqckt is
port (     x: in std_logic; -- FSM input
           z: out std_logic; -- FSM output
         clk: in std_logic ); -- clock
end seqckt;
```

Output Logic



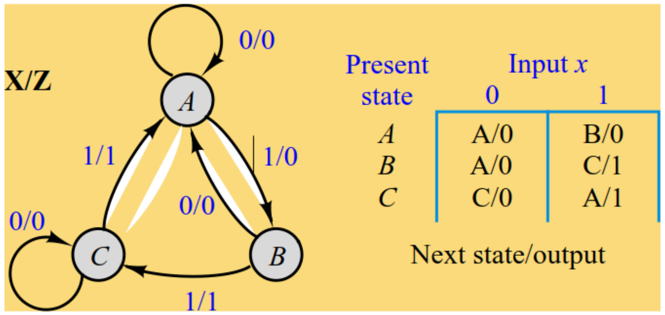
Present state	Input x 0	Input x 1
A	A/0	B/0
B	A/0	C/1
C	C/0	A/1

Next state/output

```
architecture behave of seqckt is

  type states is (A,B,C);  -- symbolic state names (enumerate)
  signal state: states;    --state variable

Begin

-- Output Logic

z <= ‘1’ when ((state = B) and (x = ‘1’))   --all conditions
            or ((state = C) and (x = ‘1’))  --for which z = 1
         else ‘0’;                          --otherwise z = 0
```

Next State Logic

```
process (clk) – Present & next states logic
  begin
    if rising_edge(clk) then -- clock edge
      case state is
        when A => if (x = '0') then
                        state <= A;
                    else
                        state <= B;   -- x = '1'
                    end if;
        when B => if (x='0') then
                        state <= A;
                    else
                        state <= C;   -- x = '1'
                    end if;
        when C => if (x='0') then
                        state <= C;
                    else
                        state <= A;   -- x = '1'
                    end if;
      end case;
    end if;
end process;
end behave;
```



Present state	Input x 0	Input x 1
A	A/0	B/0
B	A/0	C/1
C	C/0	A/1

Next state/output