Chapter 5

Synchronous Sequential Logic

- A digital system is composed of
  - registers
  - data processing units
- Binary information is transferred from one set of registers into another
- Transfer can be done directly or via a processing unit to perform an operation

Registers

- A register is a group of *Binary cells*
- a register with *n* cells can store *n* bits

Example:

- Register with 8 cells can store 8 bits and can be in one of 2^8 possible states

0	0	0	0	0	0	1	1

The content is a function of the interpretation given to the information stored in to it. Here it's the positive integer (+3)

Registers

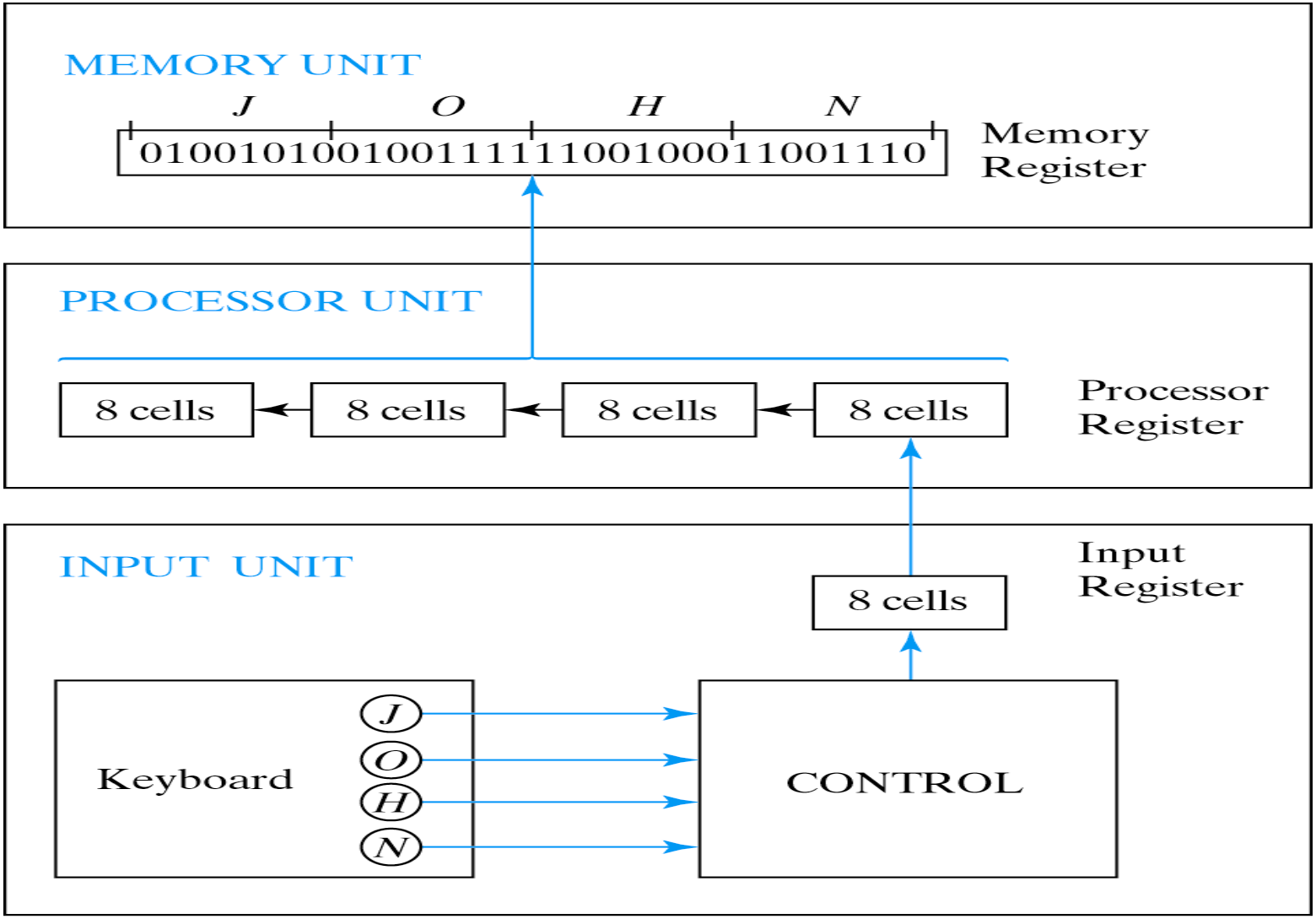


Fig. 1-1 Transfer of information with registers

Registers

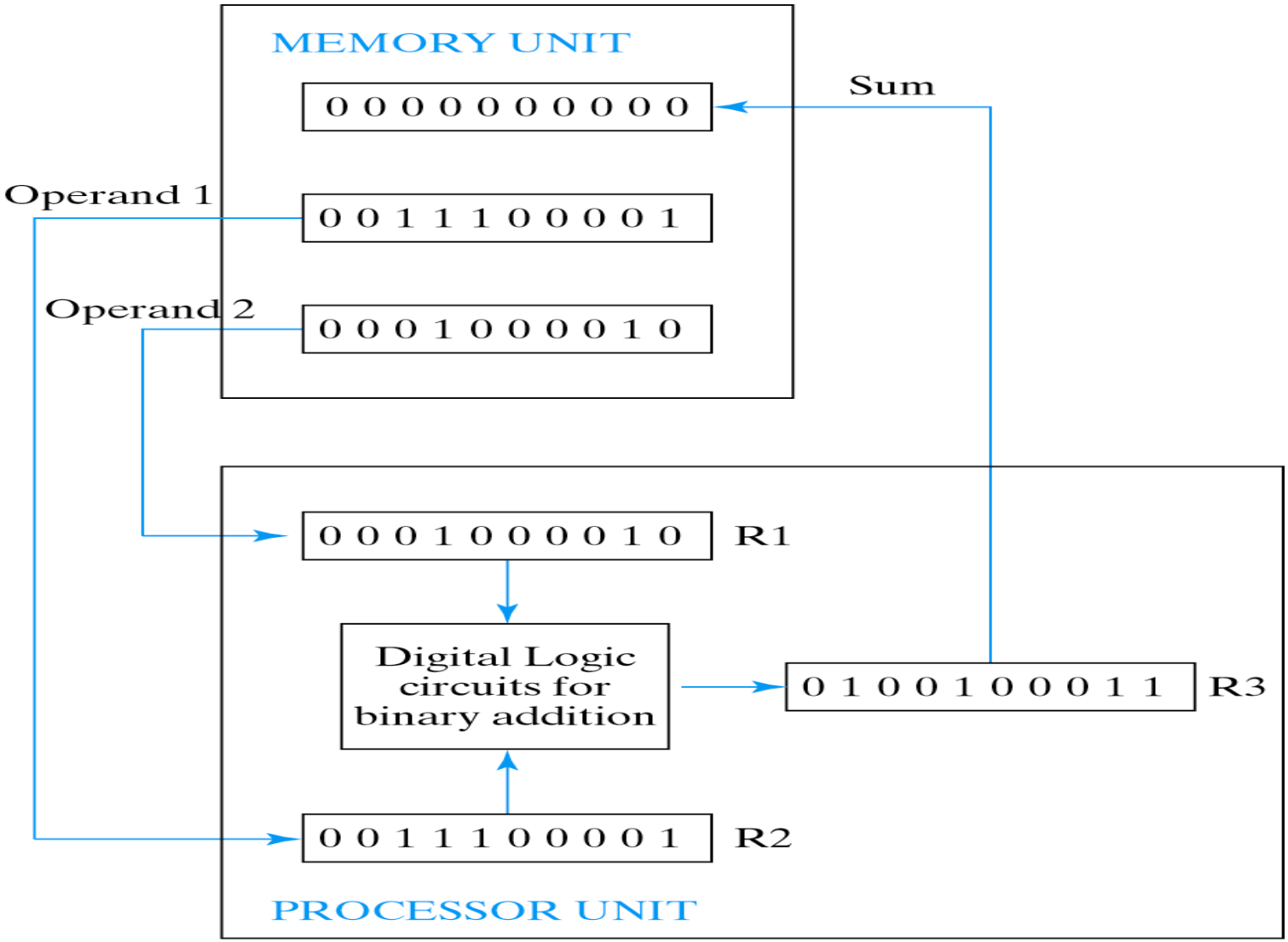


Fig. 1-2 Example of binary information processing

Sequential Circuits

→outputs logical functions of inputs and previous history of circuit (memory)

→ after changed inputs, new outputs appear in the next clock cycle

→ feedback loops

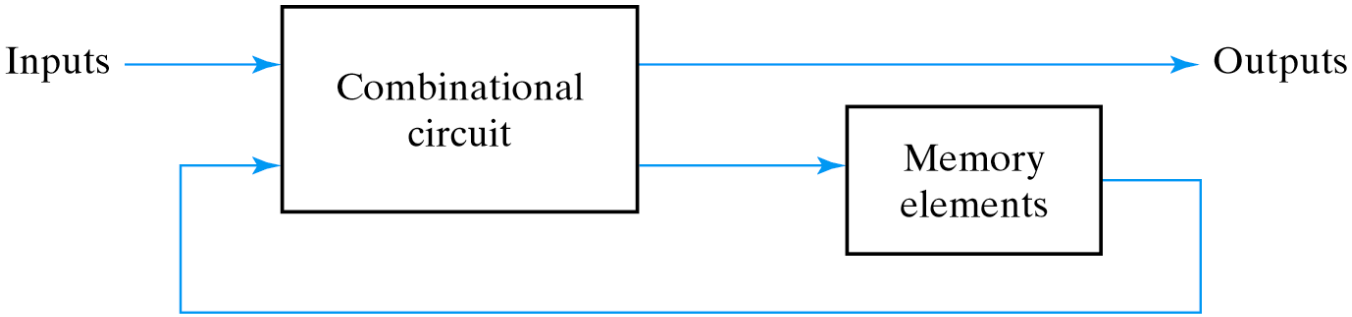
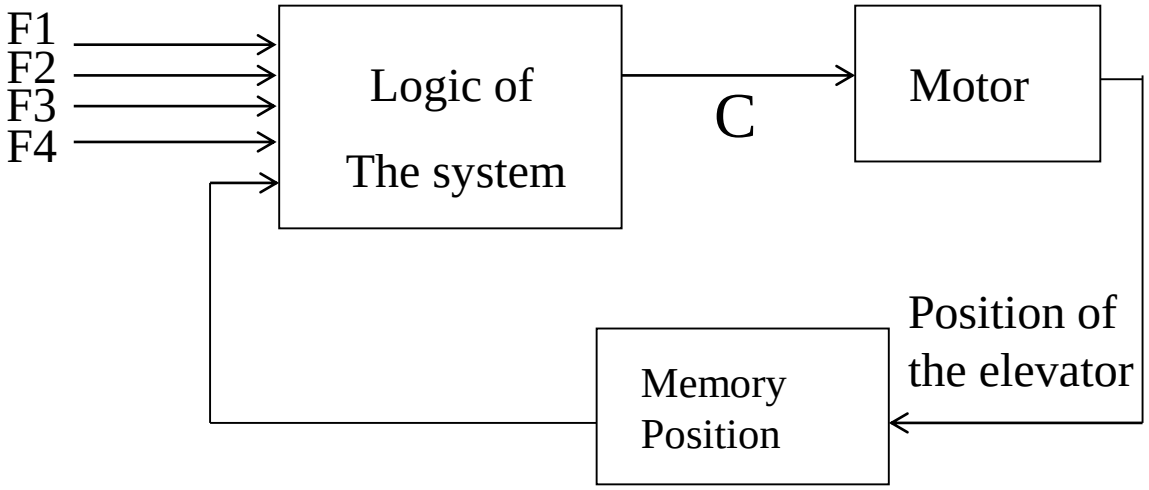


Fig. 5-1 Block Diagram of Sequential Circuit

Sequential logic circuits -Example

-- Elevator Control --

- The buttons are the input variables of the system
- The position is also an input variable which represents the current state of the elevator.
- in the figure the command control 'C' depends on the button selection AND and the position of the elevator (I.e. the previous state of the system: before the activation of the buttons)



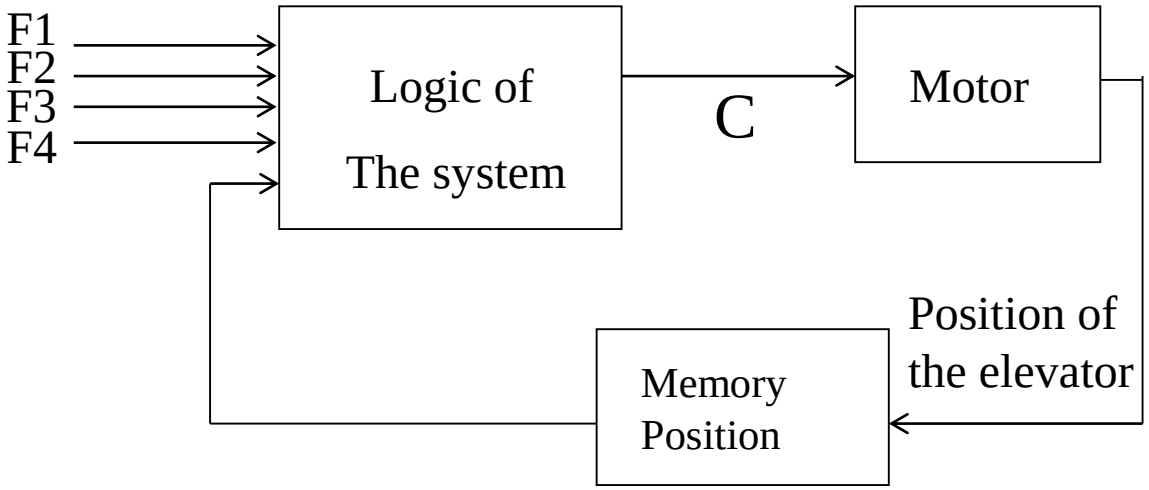
Sequential logic circuits –Example (2)

-- Elevator Control --

This example represents a **Sequential** logic system That is the output C depends on combination of

1 – **the input variables**

2 – **AND the previous states** (internal states).



Sequential circuits: time and memory

→ ***In contrast to the combinational circuits*** *two new parameters are to be considered*

1 - ***Time***

We have to consider the propagation delay of the circuits

2 – ***Memory Element***

We have to keep the previous results and impose a predetermined order.

→ Generally speaking, the function performed by the memory element is to store information in a binary form.

Sequential logic circuits : Flip-flop

- The memory cell that stores one "bit" of information is termed Flip-flop.
- A set of Flip-flop can be used to store several bits (one each) in a REGISTER.
- two different modes:

1- Asynchronous

2- Synchronous

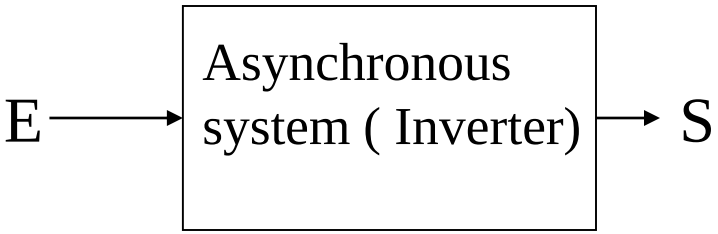
Sequential logic circuits

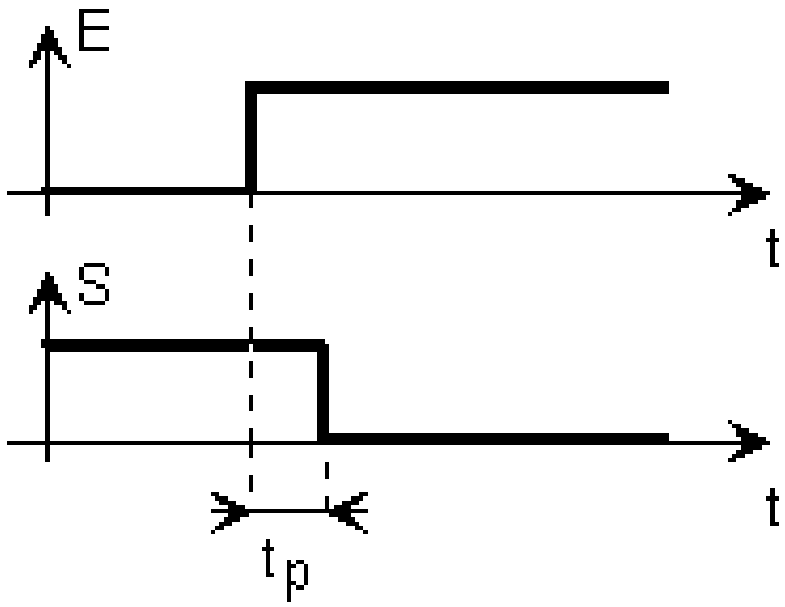
1- Asynchronous

• The output reacts "immediately" to the changes of the input variables

→ Propagation delay of the gates are to be considered

--- > t_p

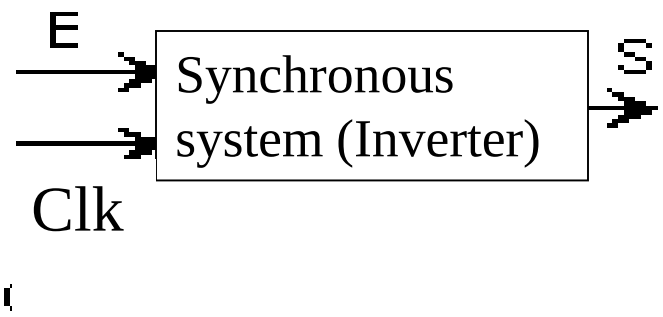


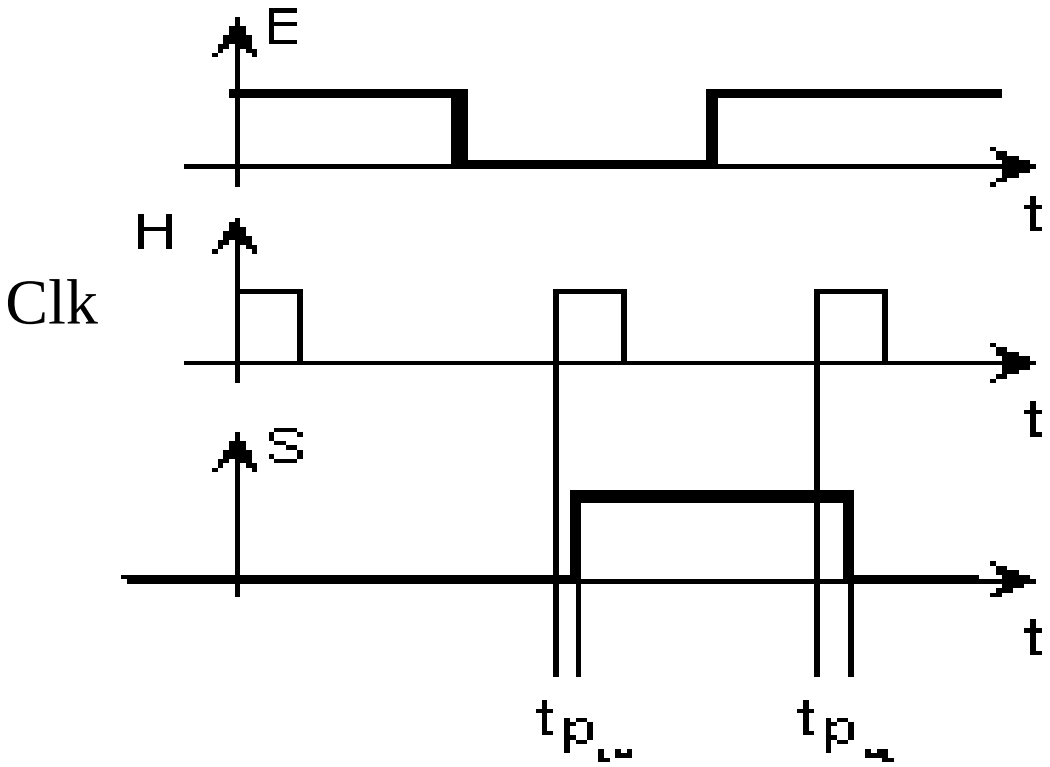


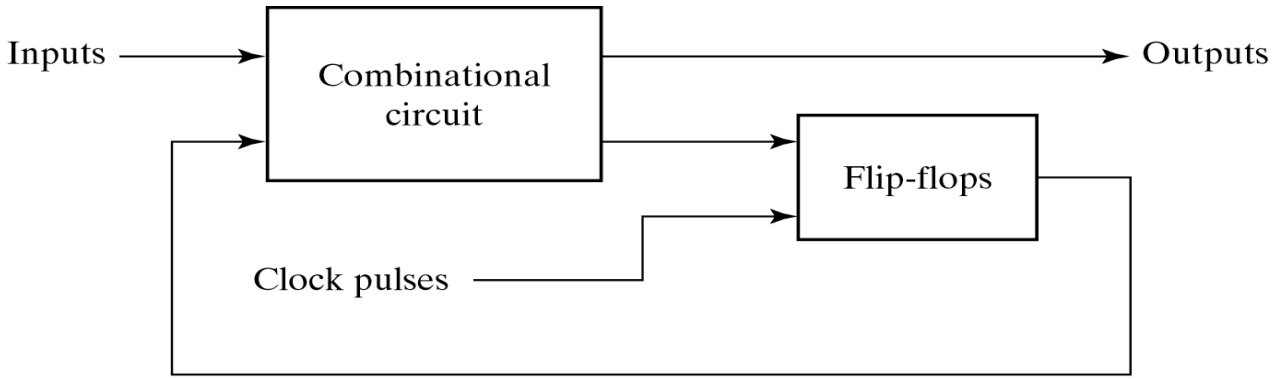
Sequential logic circuits

2- Synchronous

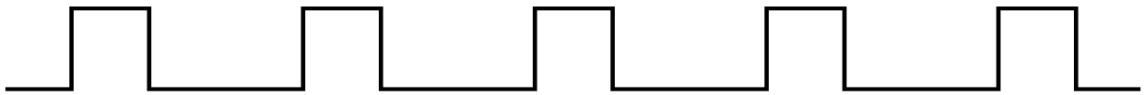
• In this mode of operation, changes of the input variables will be become **EFFECTIVE** when the clock input is active.







(a) Block diagram



(b) Timing diagram of clock pulses

Fig. 5-2 Synchronous Clocked Sequential Circuit

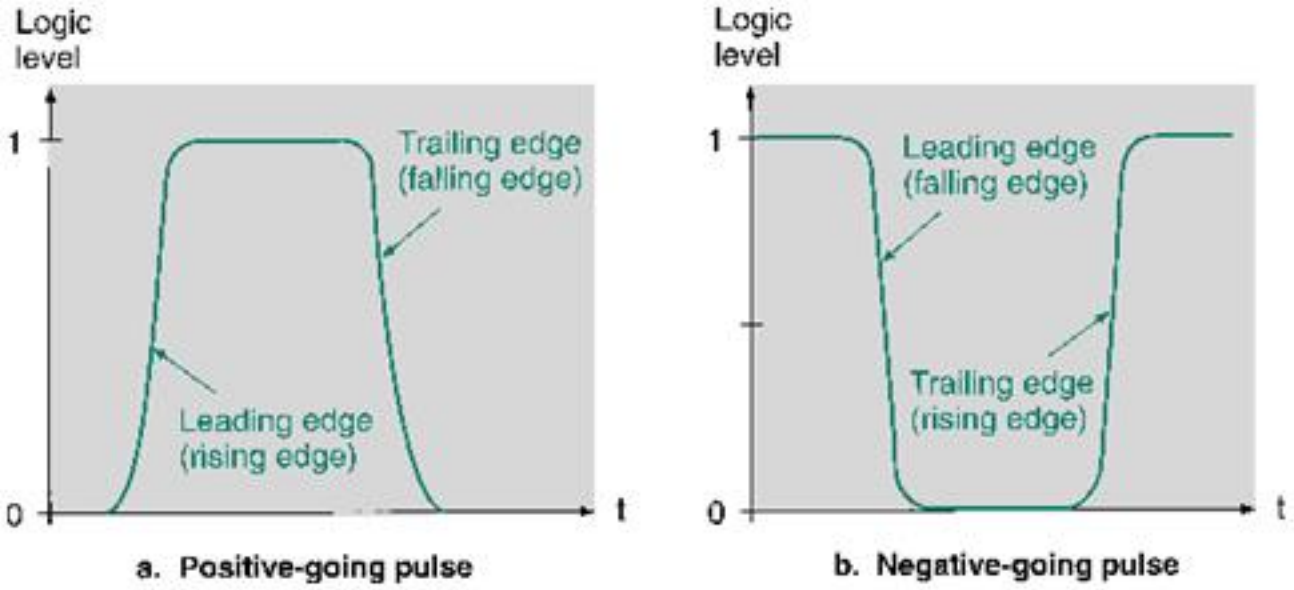
Clock signal : definitions

•Clock is used in synchronous logic circuits to trigger the circuit (Flip-flop) allowing it to switch its state.

• Clock signal can trigger a flip-flop in three ways:

- During Positive level
- During Positive transition (or positive edge)
- During Negative transition (or negative edge)

Timing Diagram –Input and output signals



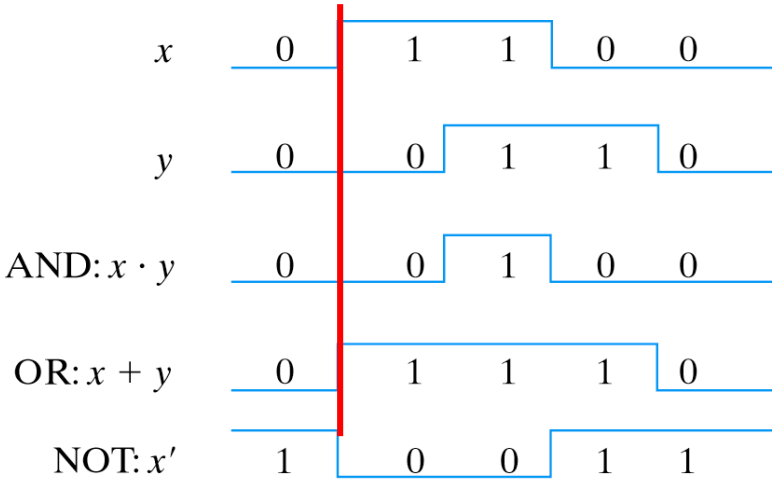
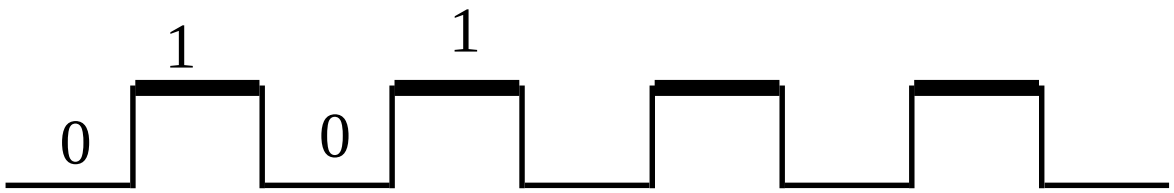


Fig. 1-5 Input-output signals for gates

Clock signal : definitions

1- Positive level

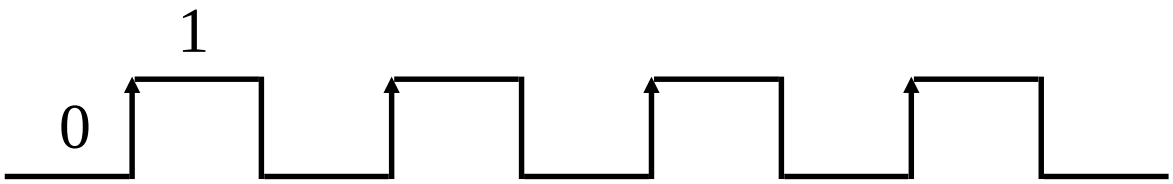
- The flip-flop is triggered while the clock pulse is at logic ‘1’
- the logic ‘0’ is therefore inactive state where changes to the flip-flop (state) are not allowed.

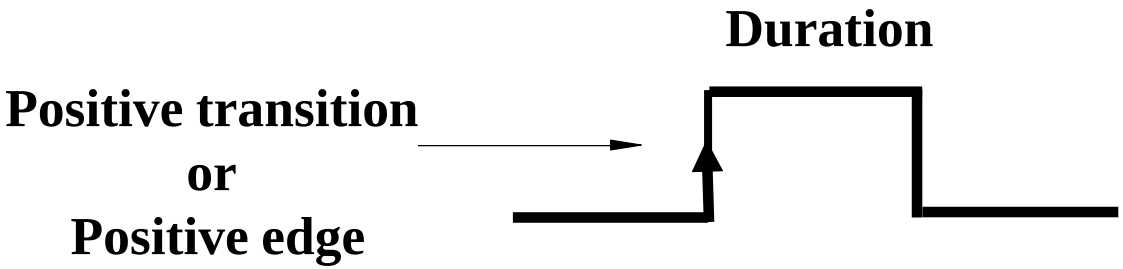


Clock signal : definitions (2)

2- Positive transition (or positive edge)

The flip flop is triggered only during the signal transition from 0 to 1 , this transition is defines as *positive edge*

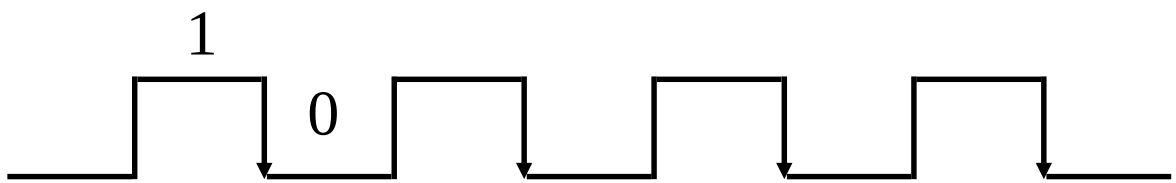


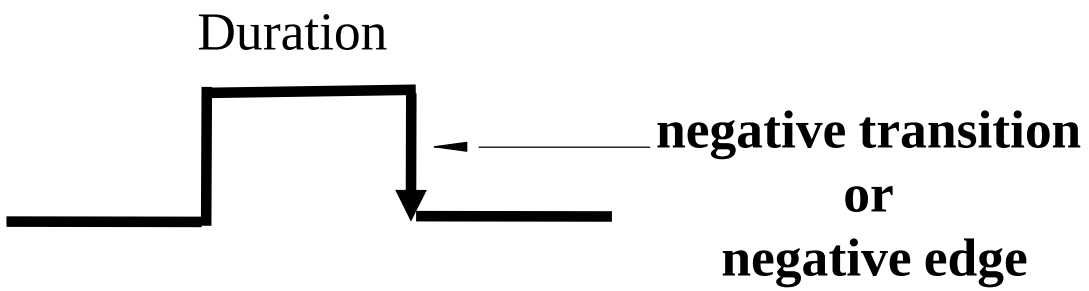


Clock signal : definitions (4)

3- negative transition (or negative edge)

The flip flop is triggered only during the signal transition from 1 to 0 , this transition is defines as *negative edge*





Memory Element (Latch)

- A flip-flop can maintain a binary state indefinitely until directed by an input signal to switch states
- The most basic types of flip-flops operate with signal levels and are referred to as Latches.
- Latches are basic circuits from which all flip-flops are constructed

Memory Element (SR Latch)

• Flip-flop (Latch) SR has two input variables that are defined as S and R

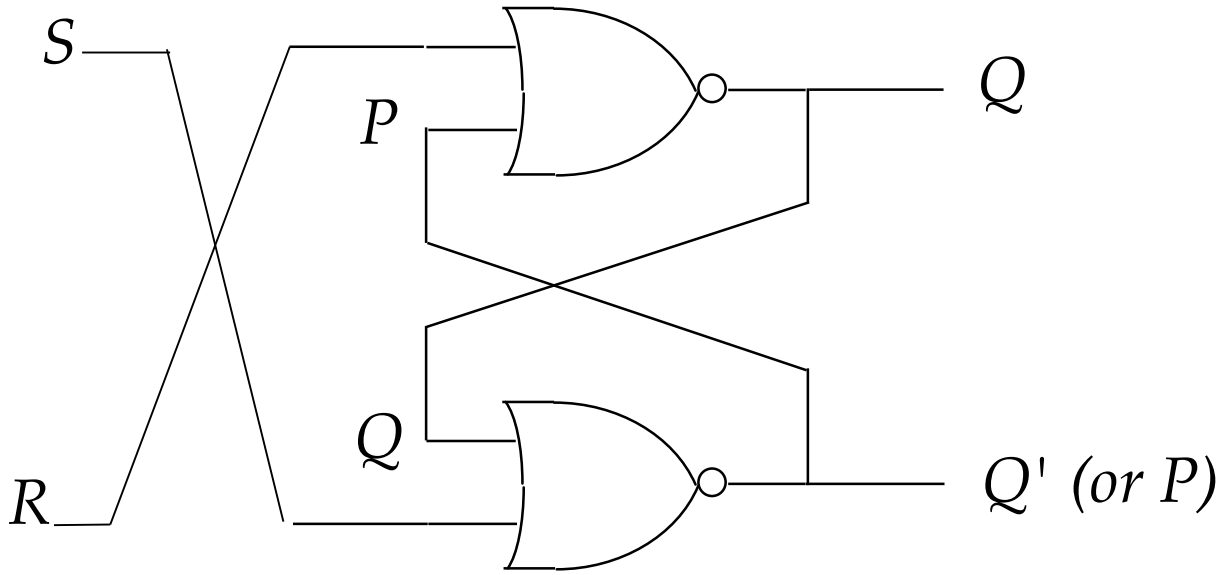
- S for SET
- R for RESET

•It also has two outputs Q (normal state) and Q' (complement state)

Symbol

Memory Element (SR Latch)

1- NOR Implementation



Q = (R+P)'= R'P'

P = (S+Q)'= S'Q'

Memory Element (SR Latch)

Symbol

Function table for RS with NOR gates

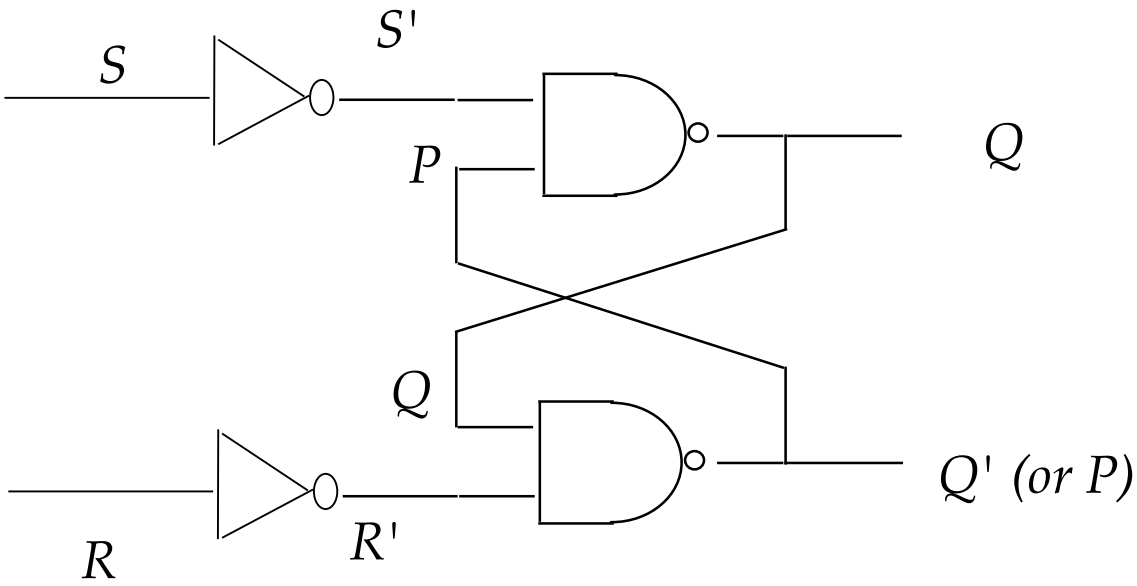
S	R	Q	Q'	
1	0	1	0	Set to "1"
0	1	0	1	Set to "0"
0	0	1	0	unchanged (after SR = 10)
0	0	0	1	unchanged (after SR = 01)
1	1	0	0	forbidden!

Memory Element (SR Latch)

2- Inverter and NAND Implementation

Q = (S'.P)' = S + P'

P = (R'Q)' = R+Q'



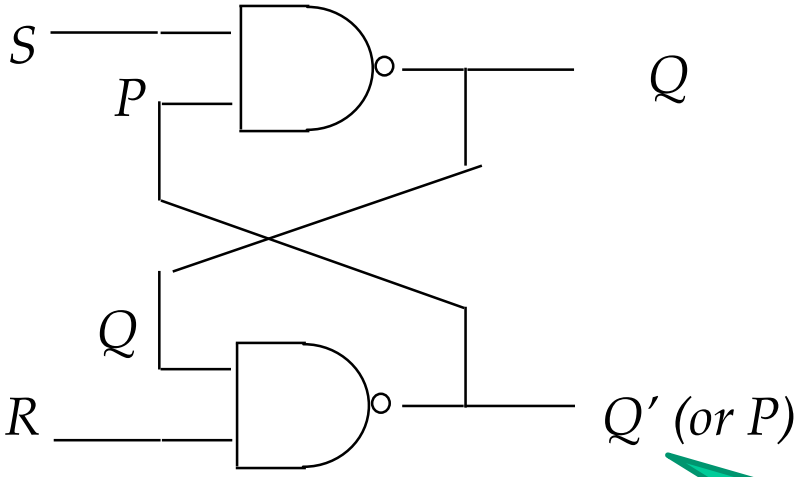
Memory Element (SR Latch)

Function table for RS with Inverter and NAND gates

S	R	Q	Q'	
1	0	1	0	Set to "1"
0	1	0	1	Set to "0"
0	0	1	0	unchanged (after SR = 10)
0	0	0	1	unchanged (after SR = 01)
1	1	1	1	forbidden!

Memory Element (SR Latch)

3- NAND Implementation also called S'R' Latch



Q = S'+P'

P = R'+Q'

Active Low Inputs

Function table for RS with NAND gates:

S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

R

set to « 0 »

unchanged (after SR = 10)

set to « 1 »

unchanged (after SR = 01)

Not allowed! (forbidden)

Memory Element (Latch)

- A flip-flop can maintain a binary state indefinitely until directed by an input signal to switch states
- The most basic types of flip-flops operate with signal levels and are referred to as Latches.
- Latches are basic circuits from which all flip-flops are constructed

Synchronous Sequential Circuits

• If we add to the memory element a **synchronization signal** (i.e.clock impulse or control input) we would obtain a synchronous SR latch (i.e. a Flip-Flop)

• We will examine the following synchronous flip-flops

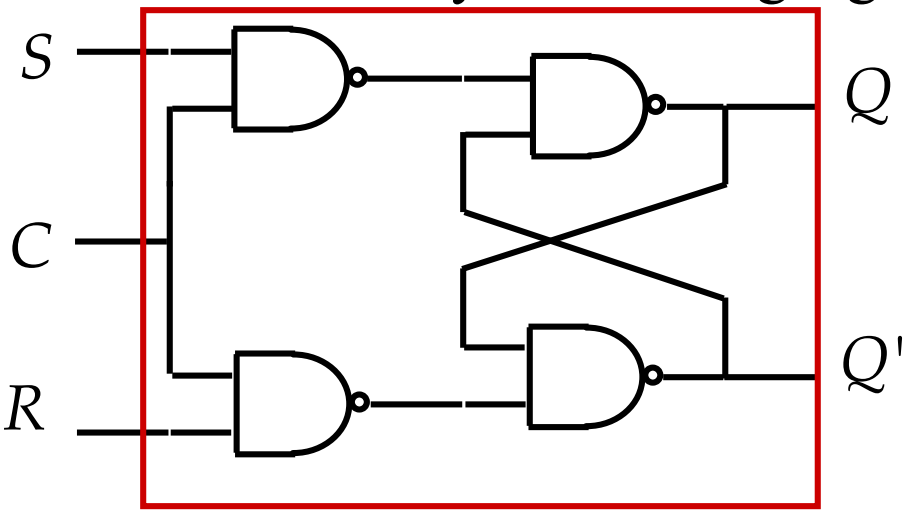
- SR
- D and T
- JK

SR Latch with Control Input

- The time when a latch is allowed to change state is regulated.
- Change of state is regulated by a control signal called ENABLE.
- **Circuit is a NAND latch controlled by steering gates.**

Latch with Control Input: SR Latch

- Used in two principal ways.
  - as an ON/OFF signal.
  - as a synchronizing signal.



C	S	R	Qt_{+1}	$Q't_{+1}$	Function
1	0	0	Qt	Q't	No change
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	1	1	Forbidden
0	X	X	Qt	Q't	Inhibited.

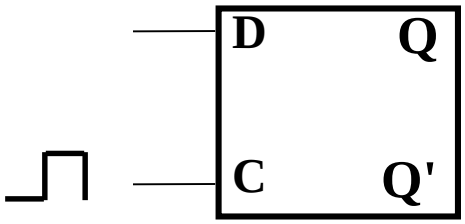
- Sometimes it's useful to avoid latch changes

  When C = 0 → disables all latch state changes
- Control signal *enables* state change when C = 1
- Right side of circuit same as ordinary S-R latch with NAND.

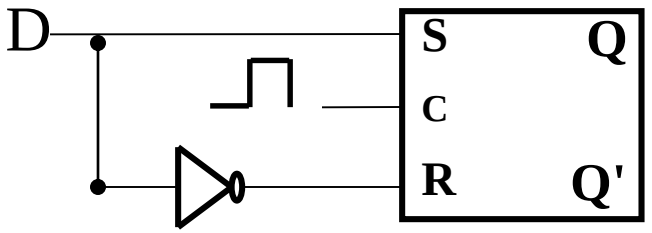
Latches with Control Input: D latch

- D for Data. Data is transferred to the Latch

Symbol



Implementation with SR latch



When C is high, D passes from input to output (Q)

D	C	Q	Q'
0	1	0	1
1	1	1	0
X	0	Q_n	Q_n'

Edge Sensitive Flip-Flop

- Latches respond to trigger levels on control inputs

  Example: If C = 1, input is reflected at output

- Flip flips store data on a positive or negative edge.

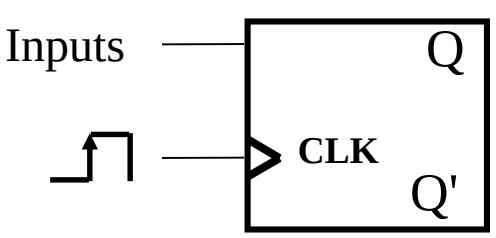
  Example: control input transitions from 0 to 1, data input available at output

- Data remains stable in the flip flop until next positive /negative edge.

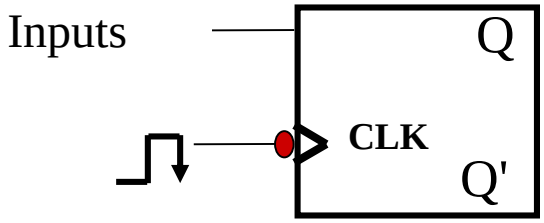
- Different types of flip flops are used for specific functions

Edge Sensitive Flip-Flop :definitions

- The sequential circuit output changes when its CLOCK input detects an edge. Edge-sensitive instead of level-sensitive.
- Symbol is a triangle on the CLK (clock) input of a flip-flop.
- Flip flop can be triggered on positive or negative edge
- Bubble before *Clock (CLK)* input indicates negative edge trigger

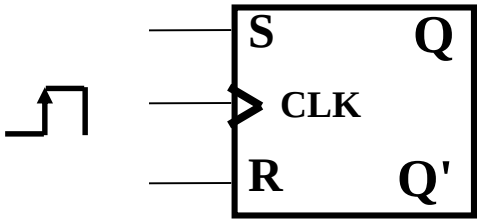


Symbol with positive edge

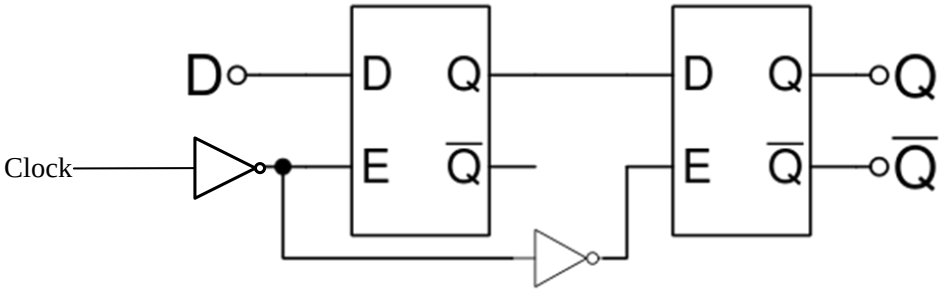


Symbol with negative edge

Edge Sensitive SR



Symbol



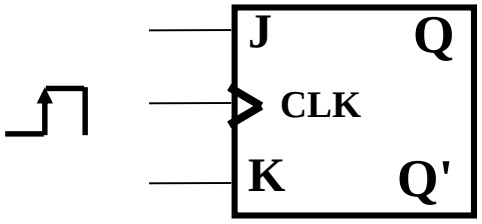
Characteristic Table

S	R	C	Q(n+1)	Comment
0	0	↑	Q(n)	No Change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	Forbidden
x	x	x	Q(n)	No Change

- Q_n = state *before* positive edge
- Q_{n+1} = state *after* positive edge

JK Flip flop

Symbol



Characteristic Table

J	K	C	Q(n+1)	Function
0	0	↑	Q(n)	No Change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q'(n)	Toggle
x	x	x	Q(n)	No Change

Same as SR except for

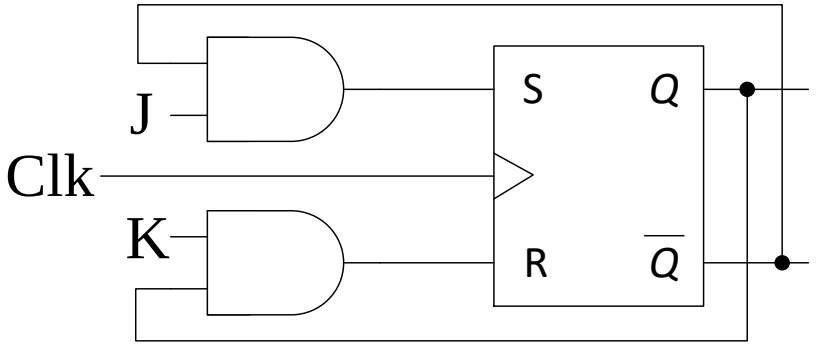
K=J= 1 the JK flip flop will

Output the opposite state of Q_n.

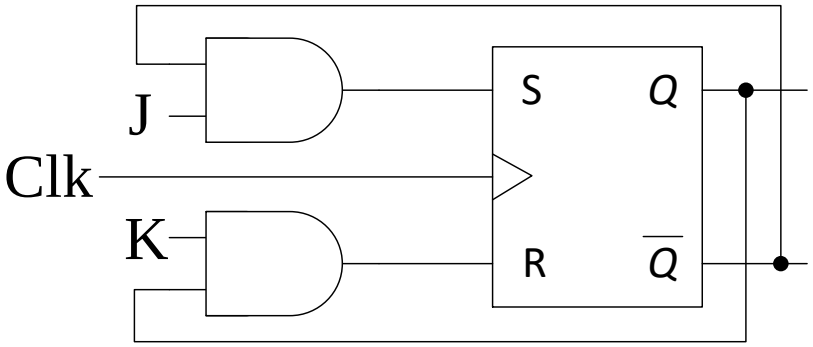
- Q_n = state *before* positive edge
- Q_{n+1} = state *after* positive edge

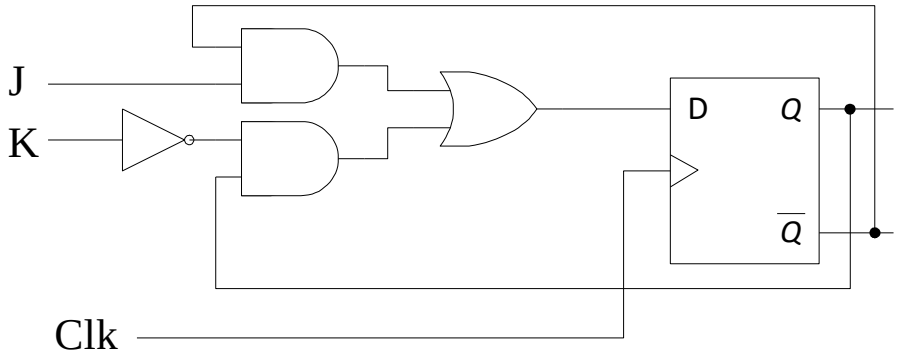
If $Q_n = 1$ then $Q_{n+1} = 0$

If $Q_n = 0$ then $Q_{n+1} = 1$



J-K Flip-Flop Implementations





$$S = JQ'$$

$$R = KQ$$

$$D = JQ' + K'Q$$

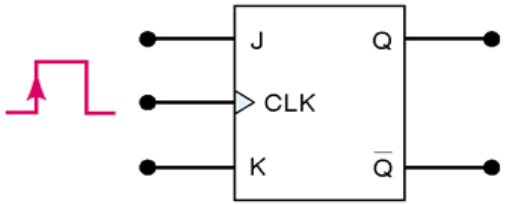
J	K	S	R	Q(n+1)	Comment
0	0	0	0	Q(n)	
0	1	0	Q	0	If Q 1, reset to 0.
1	0	Q'	0	1	If Q 0, set to 1.
1	1	Q'	Q	Q'(n)	S,R = 0,1 (Q=1), Reset S,R = 1,0 (Q=0), Set

J	K	D	Q(n+1)	Comment
0	0	Q	Q(n)	
0	1	0	0	0 + 0
1	0	1	1	Q'+Q
1	1	Q'	Q'(n)	

Characteristic Table

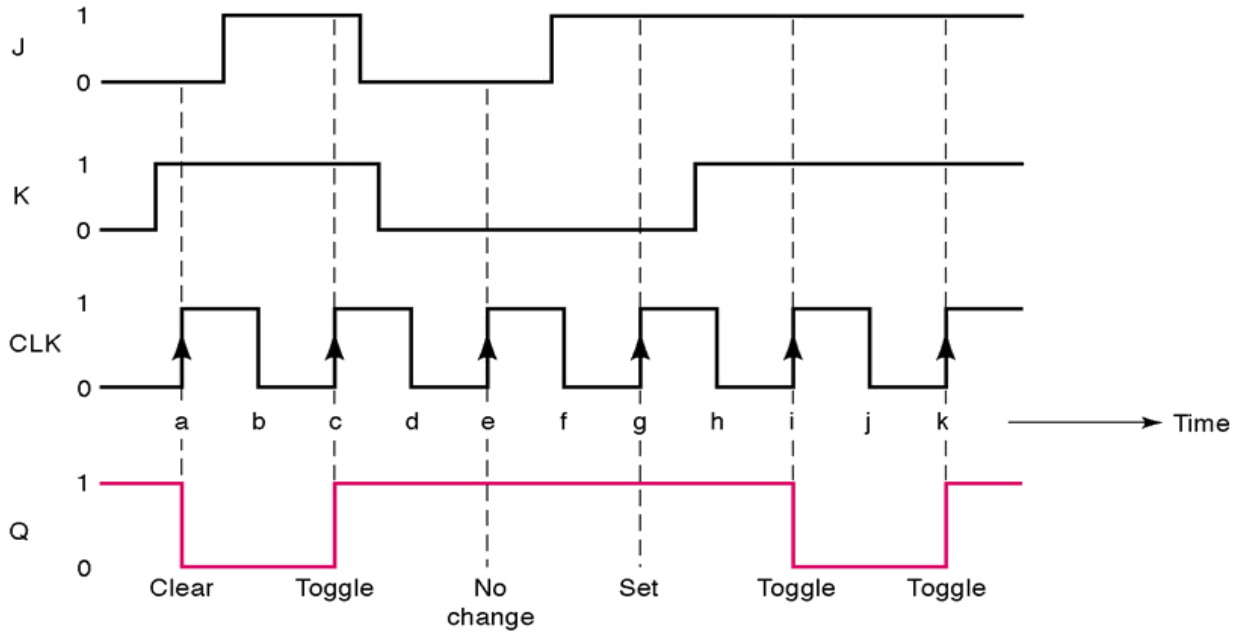
JK Flip flop

- Two data inputs, J and K
- J -> set, K -> reset, if J=K=1 then toggle (complement) output



J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (toggles)

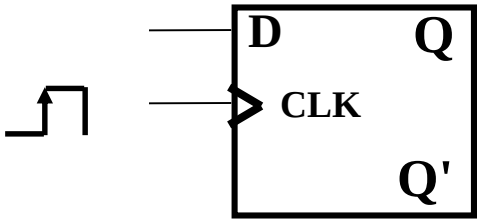
(a)



(b)

D Flip flop

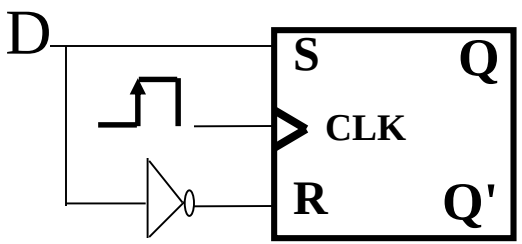
- Output changes only on the clock transition



Symbol

D	Q_{n+1}
0	0
1	1

D	CLK	Q	Q'
0	↑	0	1
1	↑	1	0
X	0	Q_0	Q_0'



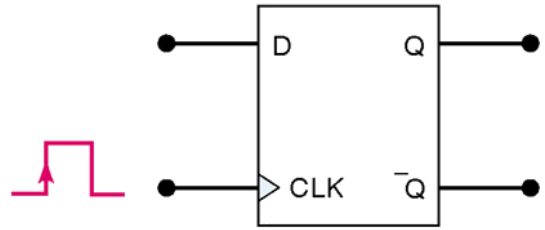
D Flip flop can be implemented using SR

S = D

R =D'

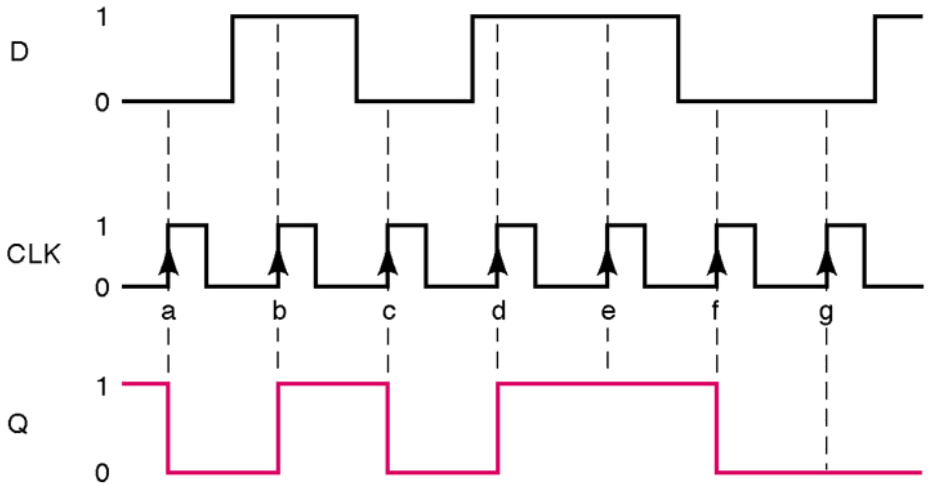
D Flip flop

- Stores a value on the positive edge of *Clock*
- Input changes at other times have no effect on output



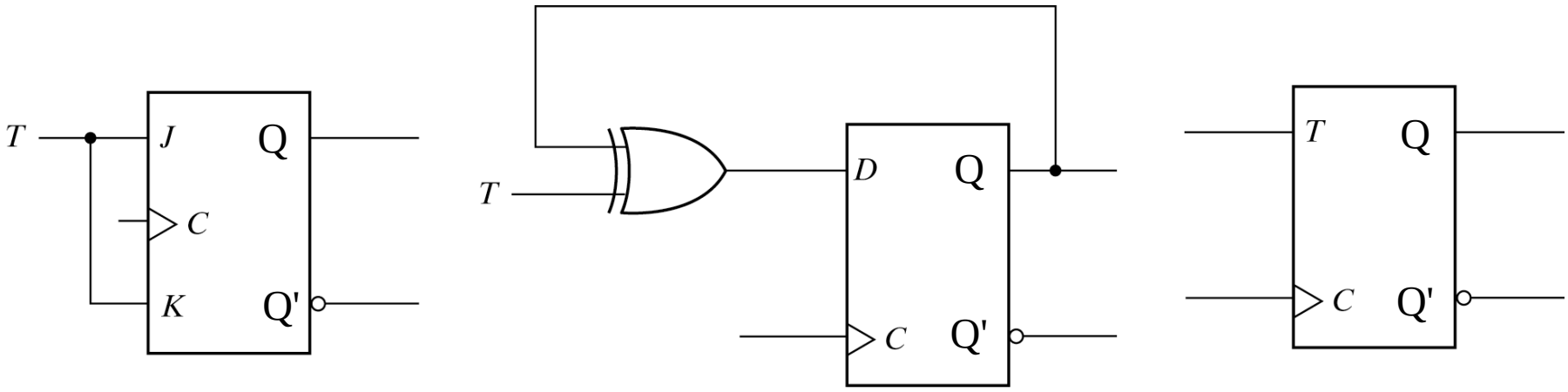
D	CLK	Q
0	↑	0
1	↑	1

(a)



(b)

T Flip-Flop



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

Fig. 5-13 T Flip-Flop

°Can be Created from JK or D flip flop

$$D = T \oplus Q$$
$$= TQ' + T'Q$$

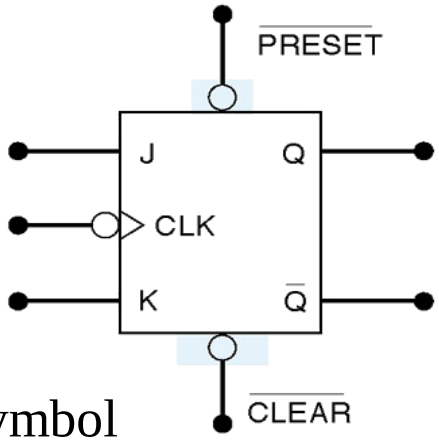
T	C	Q(t+1)	
0	↑	Q(t)	No change_
1	↑	Q'(t)	Complement(Toggle)

Asynchronous Inputs

- So far we have only considered **synchronous inputs** (e.g. D, S, R, J, K) → Their Effects on the output are synchronized with the *CLK* input.

- **Flip flops have asynchronous inputs.** They operate independently of the synchronous inputs and clock

  - used to Set the Flip flop to 1 or 0 state *at any time*.



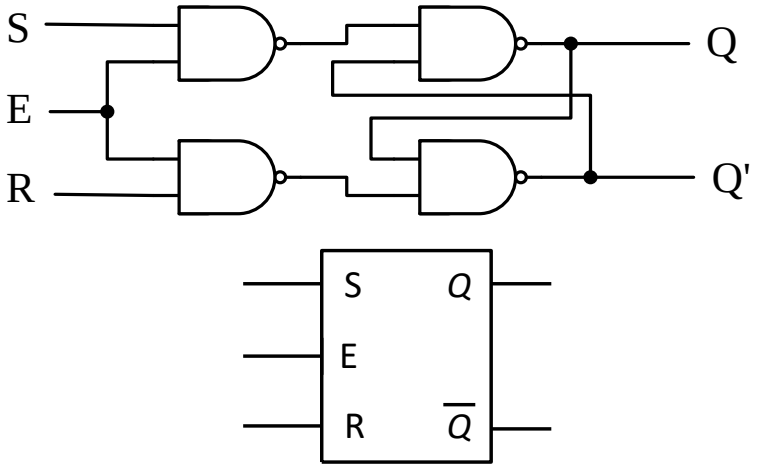
Symbol

PRESET	CLEAR	FF response
1	1	Clocked operation*
0	1	Q = 1 (regardless of CLK)
1	0	Q = 0 (regardless of CLK)
0	0	Not used

*Q will respond to J, K, and CLK

Latch (Controlled) Summary

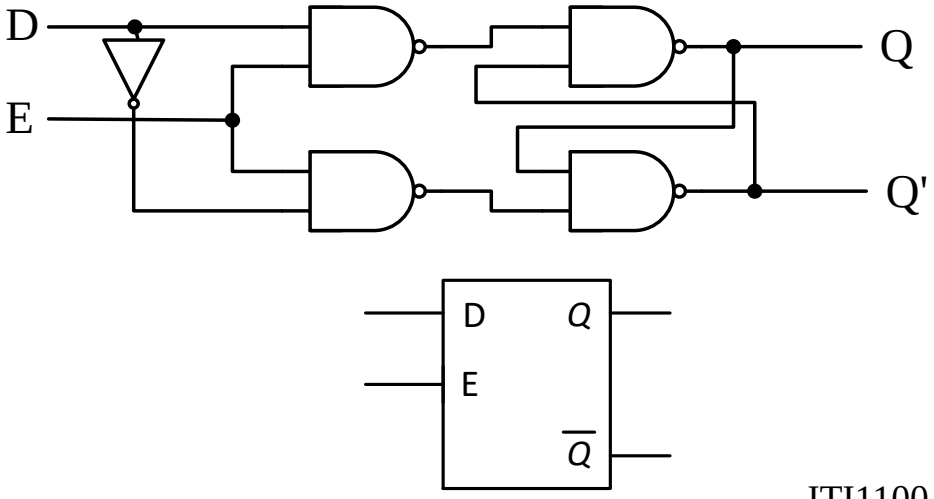
SR Latch



Function Table

S	R	E	Q(n+1)	Q'(n+1)	Comment
0	0	1	Q(n)	Q'(n)	No change
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	?	?	Forbidden
x	x	0	Q(n)	Q'(n)	No change

D Latch



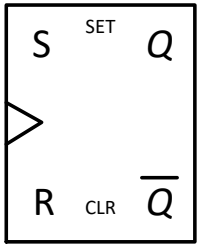
Function Table

D	E	Q(n+1)	Q'(n+1)	Comment
0	1	0	1	
1	1	1	0	
x	0	Q(n)	Q'(n)	No change

Flip-Flop Summary – Symbols/Function Tables

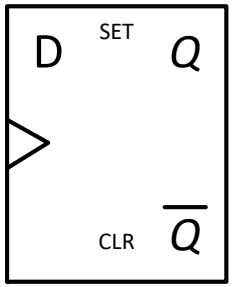
- Qn = state *before* positive edge
- Qn +1 = state *after* positive edge

SR Flip-Flop



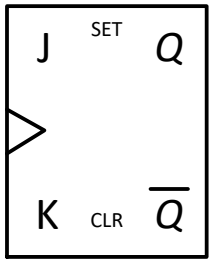
S	R	C	Q(n+1)	Q'(n+1)	Comment
0	0	↑	Q(n)	Q'(n)	No Change
0	1	↑	0	1	Reset
1	0	↑	1	0	Set
1	1	↑	?	?	Forbidden
x	x	0/1	Q(n)	Q'(n)	No Change

D Flip-Flop



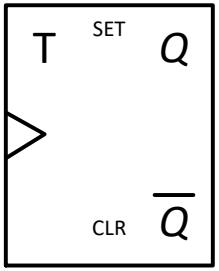
D	Clock	Q(n+1)	Q'(n+1)	Comment
0	↑	0	1	
1	↑	1	0	
x	0/1	Q(n)	Q'(n)	No Change

JK Flip-Flop



J	K	C	Q(n+1)	Comment
0	0	↑	Q(n)	No Change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q'(n)	Toggle
x	x	0/1	Q(n)	No Change

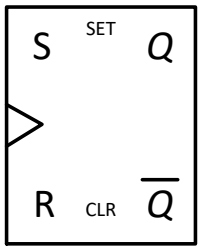
T Flip-Flop



T	C	Q(n+1)	Q'(n+1)	Comment
0	↑	Q(n)	Q'(n)	
1	↑	Q'(n)	Q(n)	Complement (toggle)
x	0/1	Q(n)	Q'(n)	No change in output

Flip-Flop Summary – Characteristic Table/Function

SR Flip-Flop

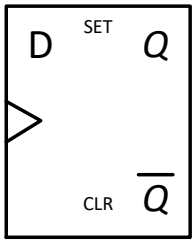


$$Q(t+1) = S + (R'Q)$$

S	R	Q(t+1)	Comment
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Undef.	Forbidden

D Flip-Flop

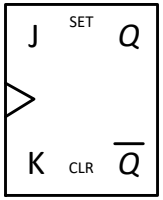
- Qt = state *before* positive edge
- Q(t +1) = state *after* positive edge



$$Q(t+1) = D$$

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

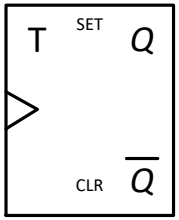
JK Flip-Flop



$$Q(t+1) = JQ' + K'Q$$

J	K	Q(t+1)	Comment
0	0	Q(t)	No Change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

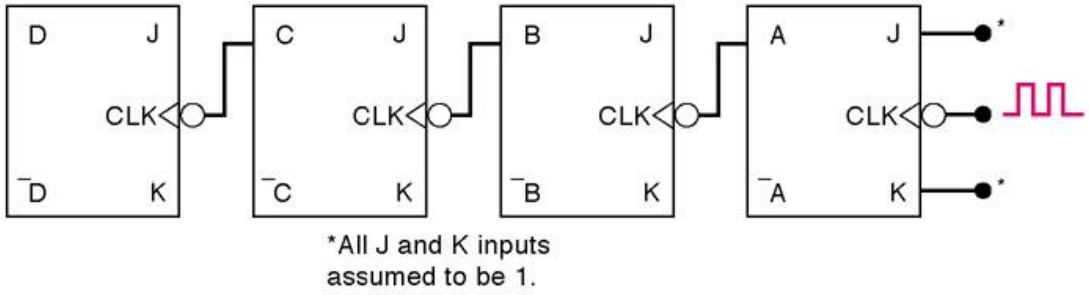
T Flip-Flop

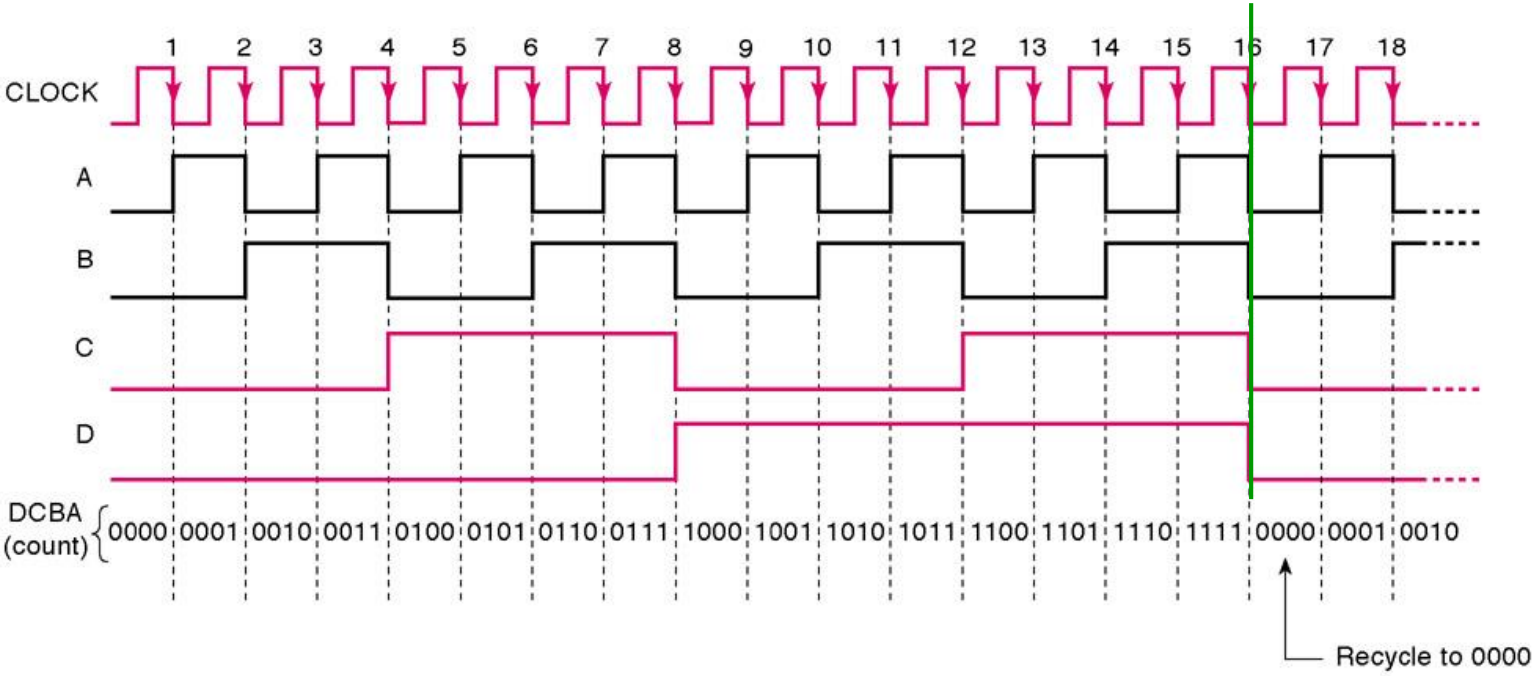


$$Q(t+1) = TQ' + T'Q = T \oplus Q$$

T	Q(t+1)	Comment
0	Q(t)	No change
1	Q'(t)	Complement (toggle)

Example: an application of Flip flops





Analysis of synchronous sequential circuits

→Analysis describes what a given circuit will do under certain conditions

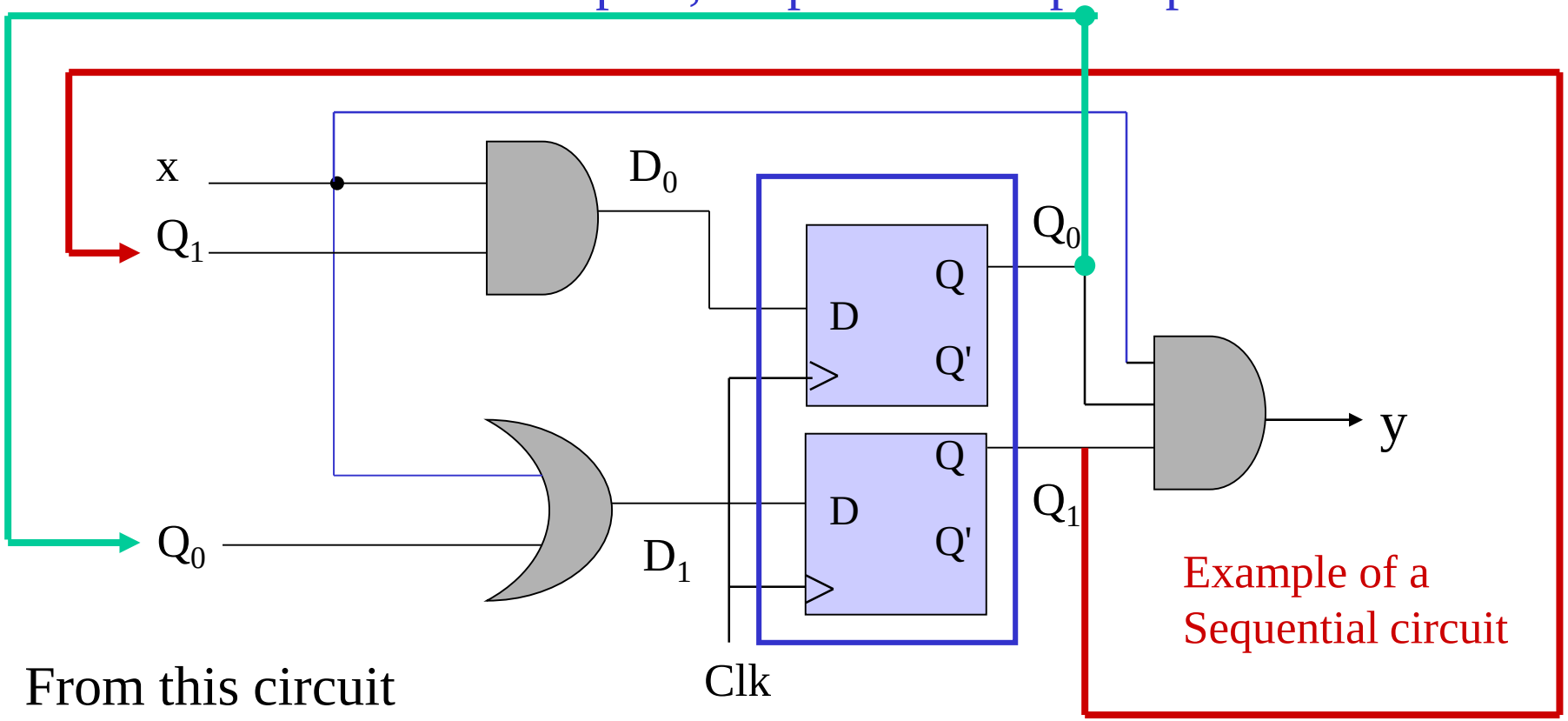
→Behavior of synchronous sequential circuit can be determined from

- **Its inputs,**

- **Its outputs and its Flip Flop state**

Flip Flop State

- Behavior of synchronous sequential circuit can be determined from inputs, outputs and Flip Flop state



From this circuit we can derive:

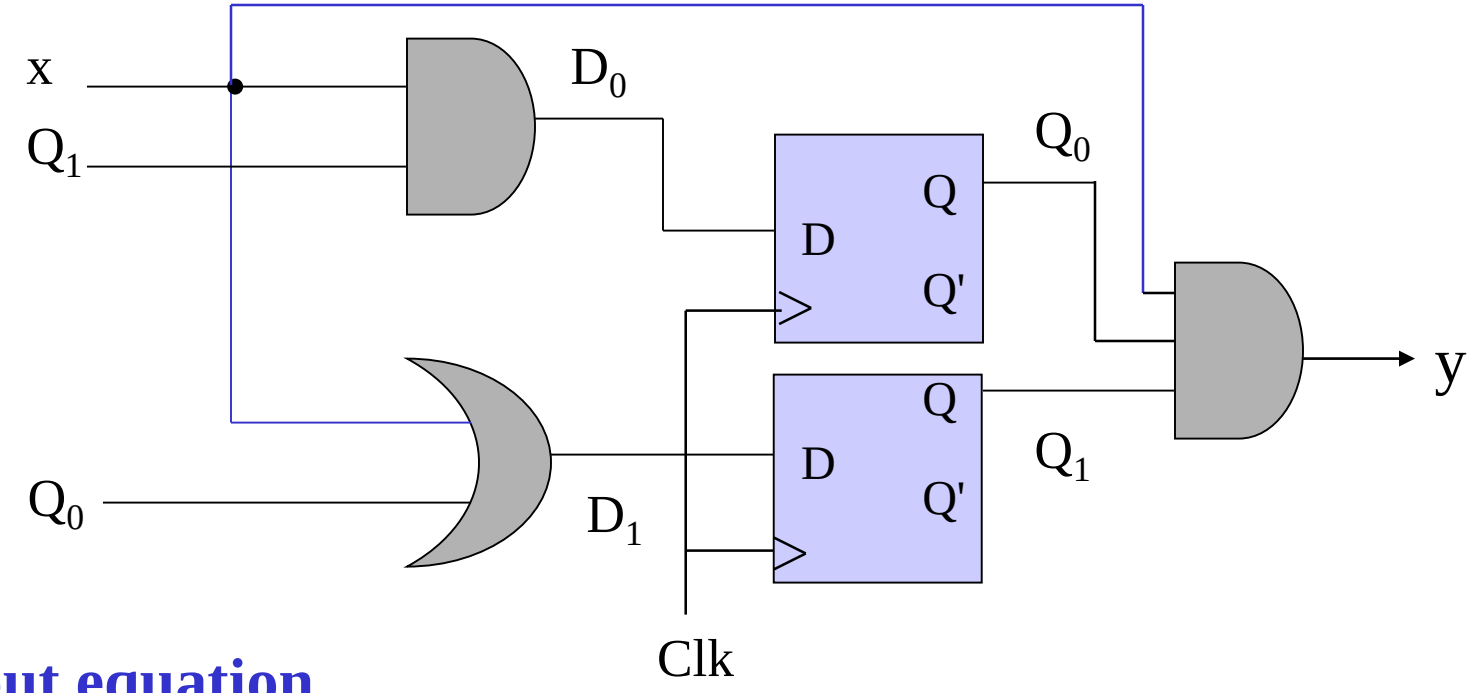
$y(t) = x(t)Q_1(t)Q_0(t)$

$Q_0(t+1) = D_0(t) = x(t)Q_1(t)$

$Q_1(t+1) = D_1(t) = x(t) + Q_0(t)$

Output and State Equations

- **Next state dependent on previous state.**



Output equation

$y(t) = x(t)Q_1(t)Q_0(t)$

State equations

$Q_0(t+1) = D_0(t) = x(t)Q_1(t)$

$Q_1(t+1) = D_1(t) = x(t) + Q_0(t)$

State Table

- Sequence of outputs, inputs, and flip flop states are listed in a table called "State Table"
- Present state indicates current value of flip flops
- Next state indicates state after next clock edge
- Output is output value on current clock edge

State Table

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
0 0	00	10	0	0
0 1	10	10	0	0
1 0	00	11	0	0
1 1	10	11	0	1

$Q_1(t)\ Q_0(t)$ $Q_1(t+1)\ Q_0(t+1)$

$y(t) = x(t)Q_1(t)Q_0(t)$

$Q0(t+1) = D_0(t) = x(t)Q_1(t)$

$Q1(t+1) = D_1(t) = x(t) + Q_0(t)$

State Table – 2 forms

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$Q_1(t)/ Q_0(t)$	$Q_1(t+1)/Q_0(t+1)$	$Q_1(t+1)/ Q_0(t+1)$	y	y
00	00	10	0	0
01	10	10	0	0
10	00	11	0	0
11	10	11	0	1

Present State		Input	Next State		Output
$Q_1(t)$	$Q_0(t)$	x	$Q_1(t+1)$	$Q_0(t+1)$	y
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	1	1	1

State Table

- All possible input combinations are listed
- All possible state combinations are listed
- Separate columns for each output value.
- Easier to use a symbol for each state.

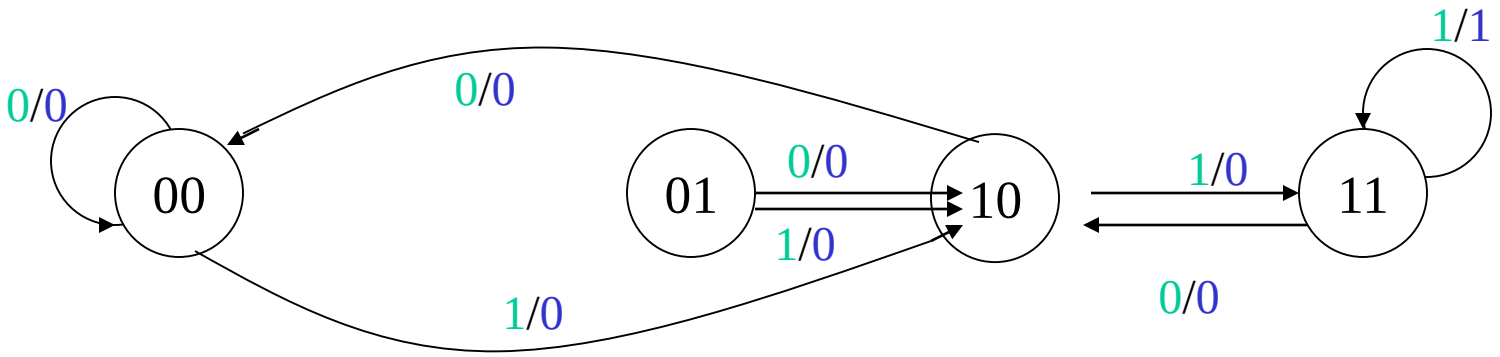
Let:
$s_0 = 00$
$s_1 = 01$
$s_2 = 10$
$s_3 = 11$

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
s_0	s_0	s_2	0	0
s_1	s_2	s_2	0	0
s_2	s_0	s_3	0	0
s_3	s_2	s_3	0	1

State Diagram

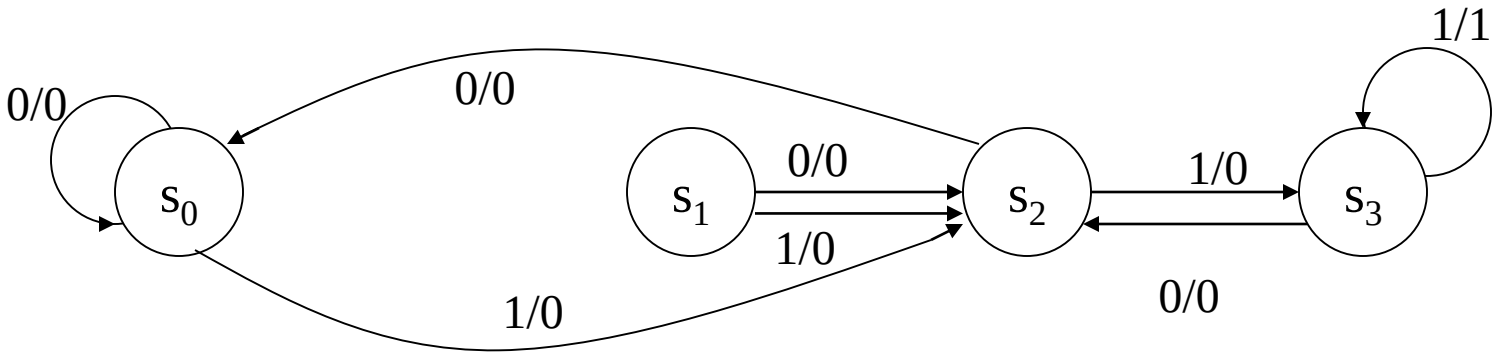
- Circles indicate current state
- Arrows point to next state
- For x/y, x is input and y is output

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
0 0	00	10	0	0
0 1	10	10	0	0
1 0	00	11	0	0
1 1	10	11	0	1



State Diagram

- **Each state has two arrows leaving**
  - **One for x = 0 and one for x = 1**
- **Unlimited arrows can enter a state**
- **Use of state names in this example**
  - **Easier to identify**



Analysis of synchronous sequential circuits

→Analysis describes what a given circuit will do under certain conditions

→Behavior of synchronous sequential circuit can be determined from

- Its inputs, its outputs and its Flip Flop state

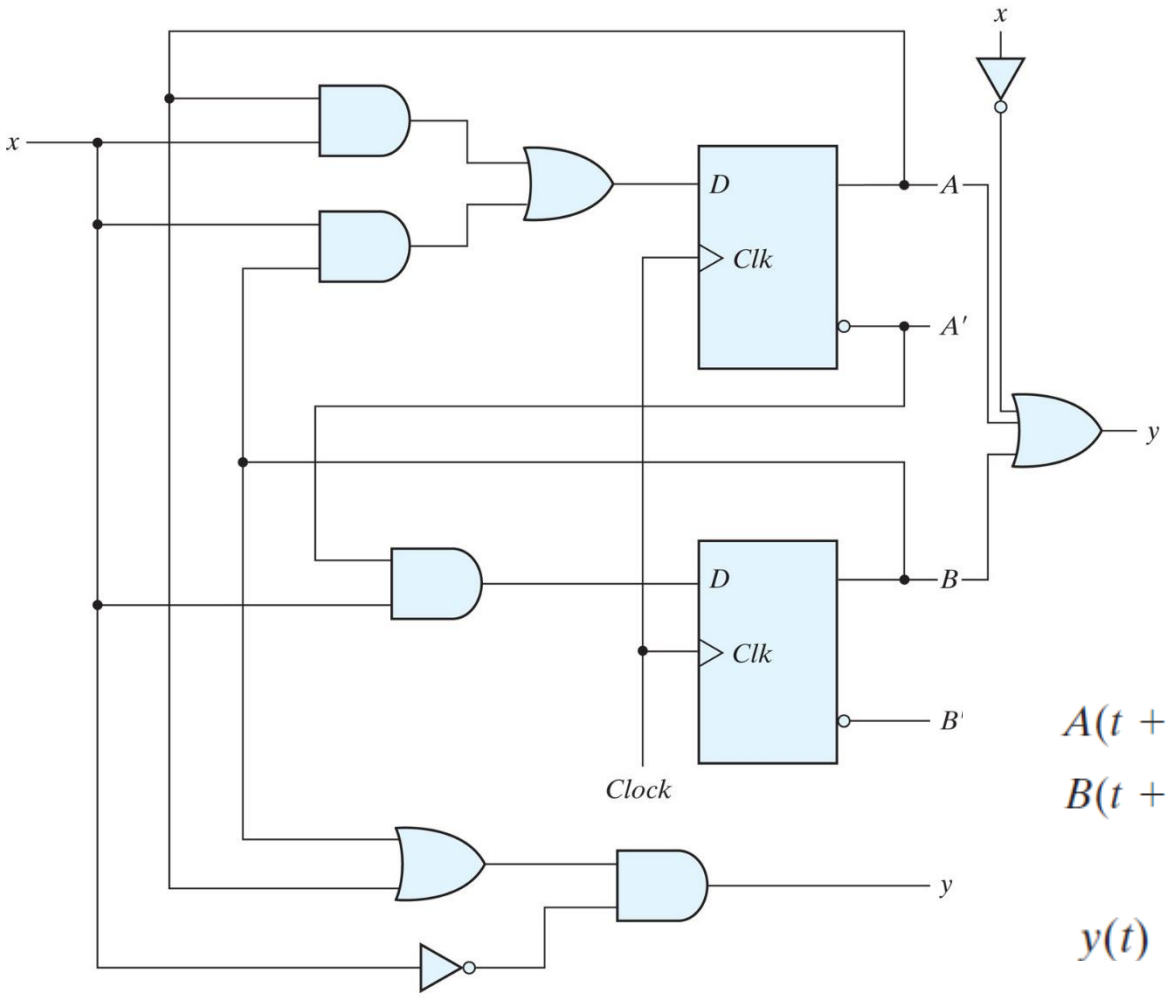
→Analysis Steps

- Circuit Diagram →Equations →State Table → State Diagram

- Identify Equations:

  - F-F input eqn: F-F inputs = F1(inputs, present state)
  - Output eqn: Outputs = F2(inputs, present state)
  - State eqn: Next state = F3(inputs, present state)

Example: Zero Detector



$$A(t+1) = A(t)x(t) + B(t)x(t)$$

$$B(t+1) = A'(t)x(t)$$

$$y(t) = [A(t) + B(t)]x'(t)$$

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

Zero Detector: State Table

Table 5.2
State Table for the Circuit of Fig. 5.15

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

Zero Detector: Second Form of the State Table

Table 5.3
Second Form of the State Table

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

Zero Detector: State Diagram

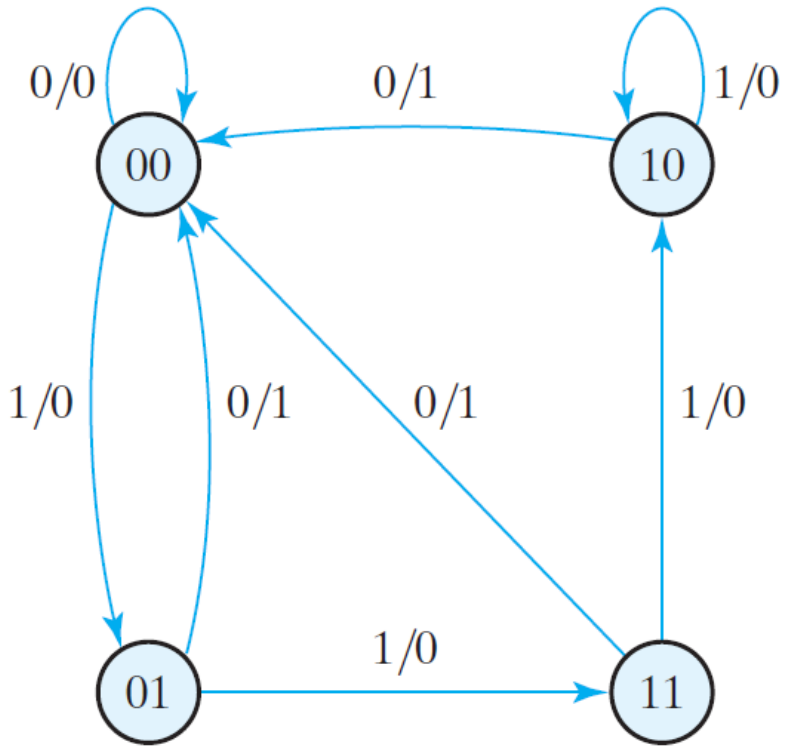
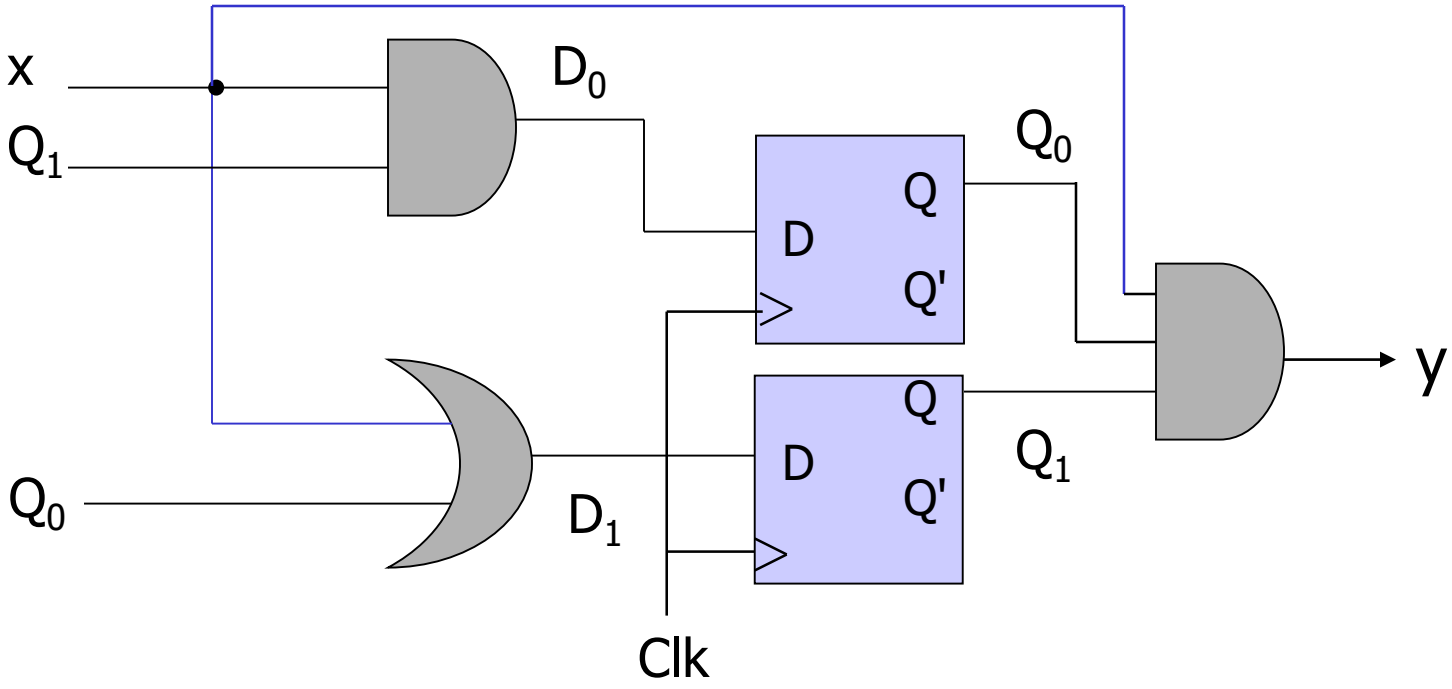


FIGURE 5.16
State diagram of the circuit of Fig. 5.15

Flip Flop Input Equations

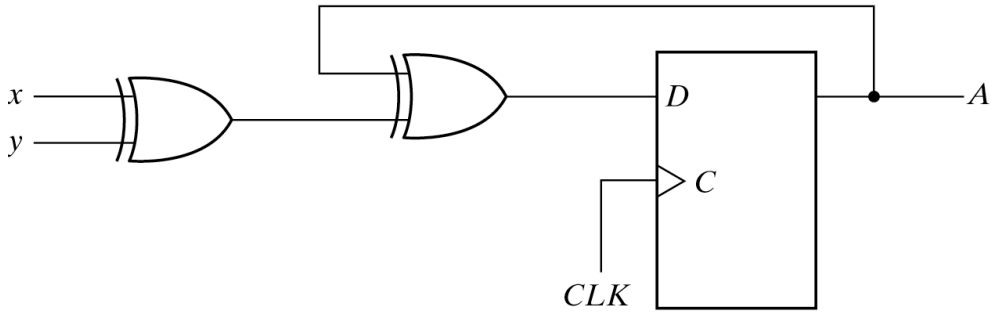
- **Boolean expressions which indicate the input to the flip flops.**



$$D_{Q0} = xQ_1$$
$$D_{Q1} = x + Q_0$$

Analysis with D Flip-Flops

- **Identify flip flop input equations D_A = A⊕x ⊕y**
- **Identify output equation (no output in this example)**
- **Identify state equation A(t+1)=D_A= A⊕x ⊕y**

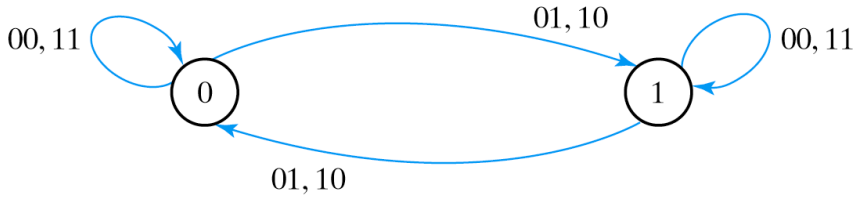


(a) Circuit diagram

Present state	Inputs		Next state
A	*x*	*y*	*A*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(b) State table

Note: this example has no output



(c) State diagram

Fig. 5-17 Sequential Circuit with *D* Flip-Flop

Example: JK Flip-Flops

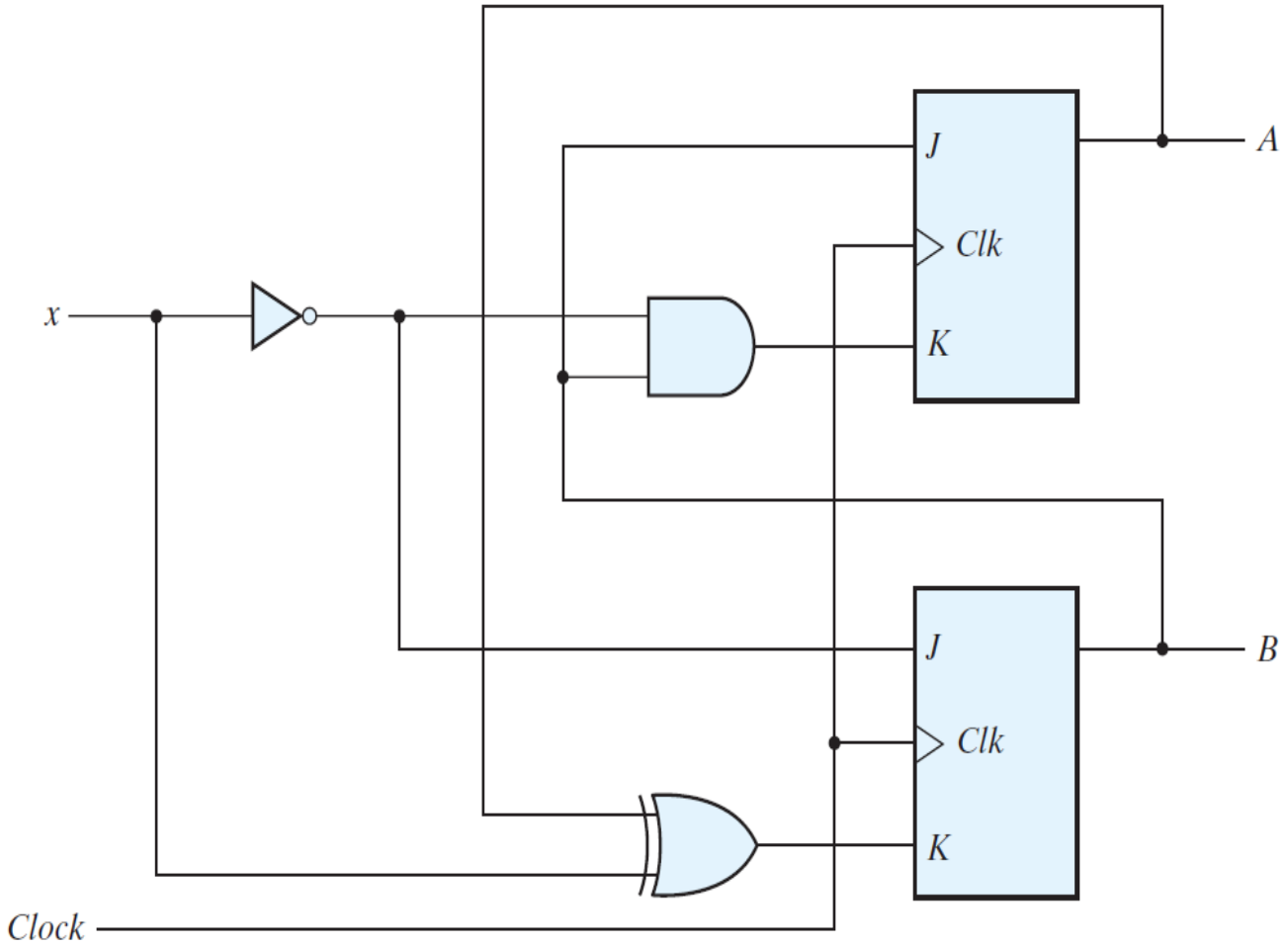


FIGURE 5.18
Sequential circuit with *JK* flip-flop

Example: JK Flip-Flops

1. Determine the flip-flop input equations in terms of the present state and input variables.
2. List the binary values of each input equation.
3. Use the corresponding flip-flop characteristic table to determine the next-state values in the state table.

The next-state values can also be obtained by evaluating the state equations from the characteristic equation. This is done by using the following procedure:

1. Determine the flip-flop input equations in terms of the present state and input variables.
2. Substitute the input equations into the flip-flop characteristic equation to obtain the state equations.
3. Use the corresponding state equations to determine the next-state values in the state table.

$J_A = B \quad K_A = Bx'$

$J_B = x' \quad K_B = A'x + Ax' = A \oplus x$

$A(t+1) = J_A A' + K'_A A$

$B(t+1) = J_B A' + K'_B B$

$$A(t+1) = BA' + (Bx')'A = A'B + AB' + Ax$$

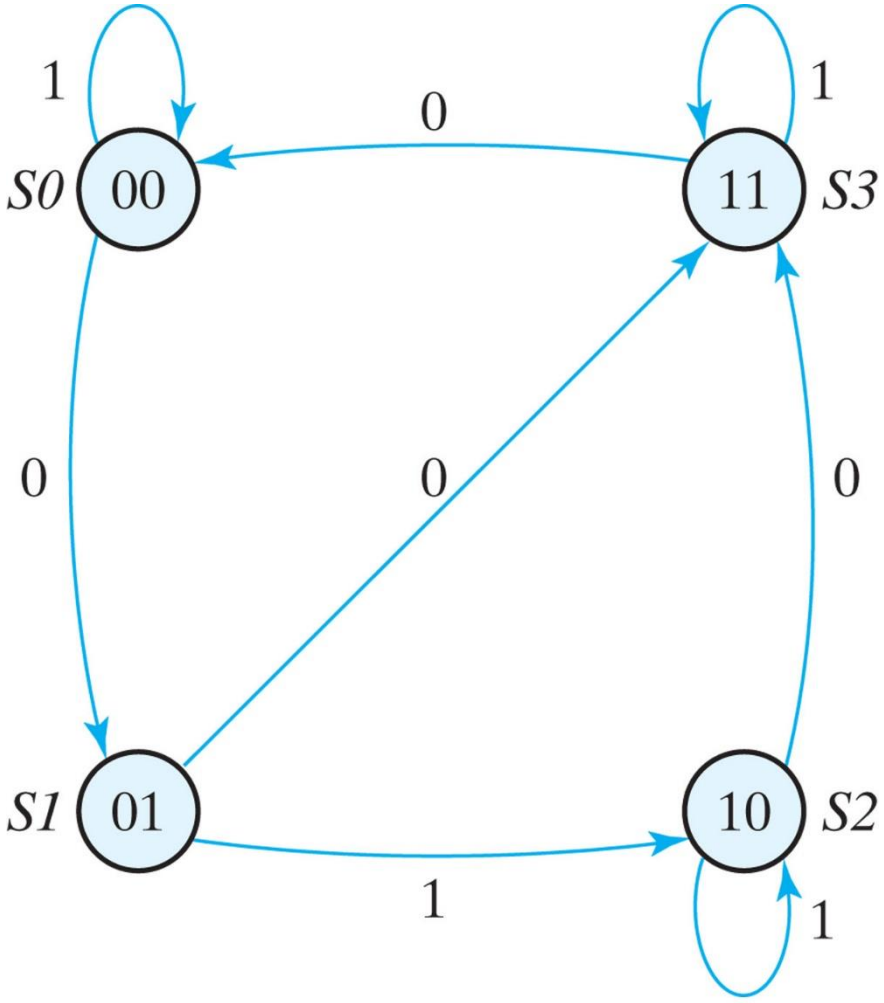
$$B(t+1) = x'B' + (A \oplus x)'B = B'x' + ABx + A'Bx'$$

Example: JK Flip-Flops

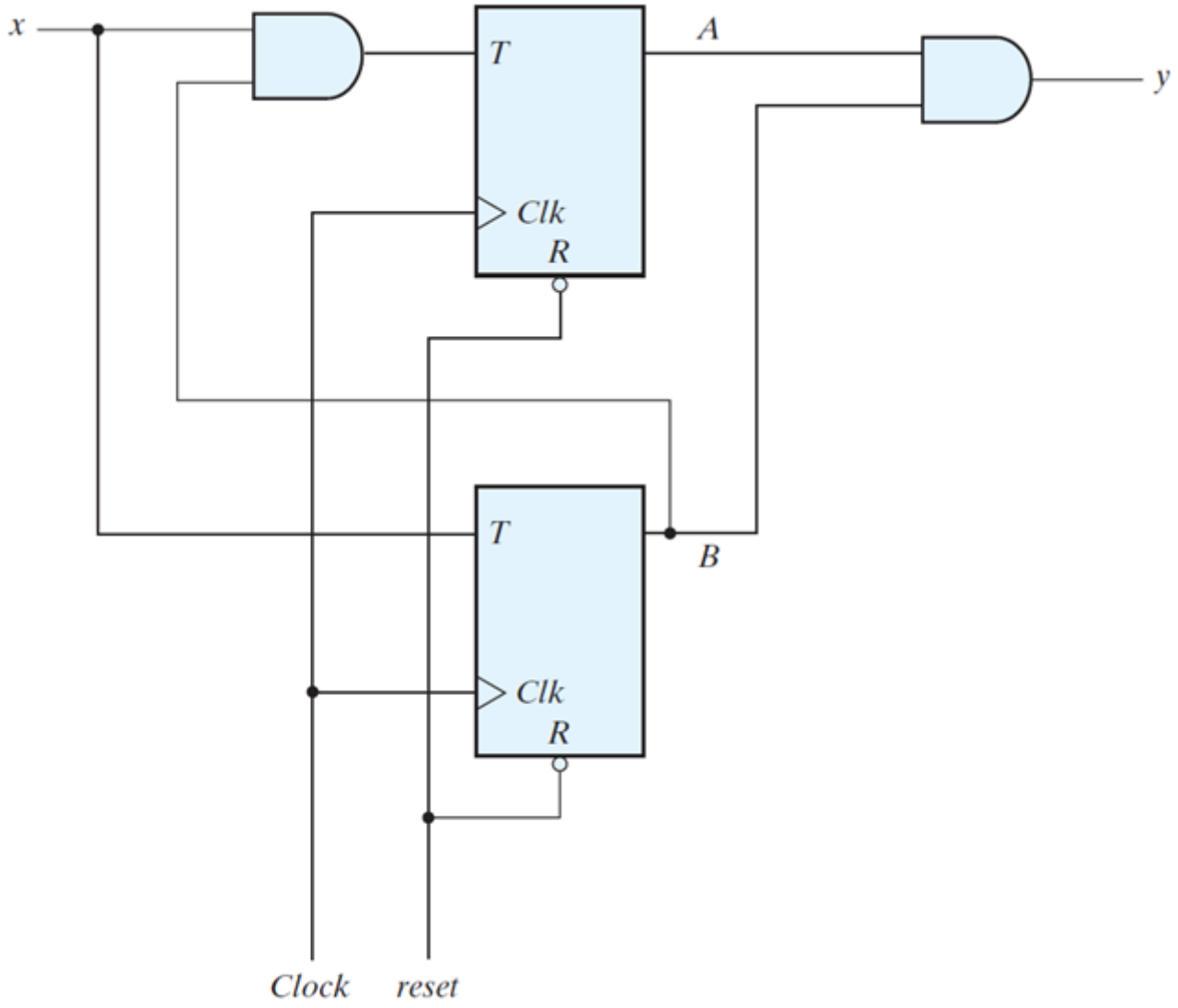
Table 5.4
State Table for Sequential Circuit with JK Flip-Flops

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
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

Example: JK Flip-Flops



Example: T Flip-Flops



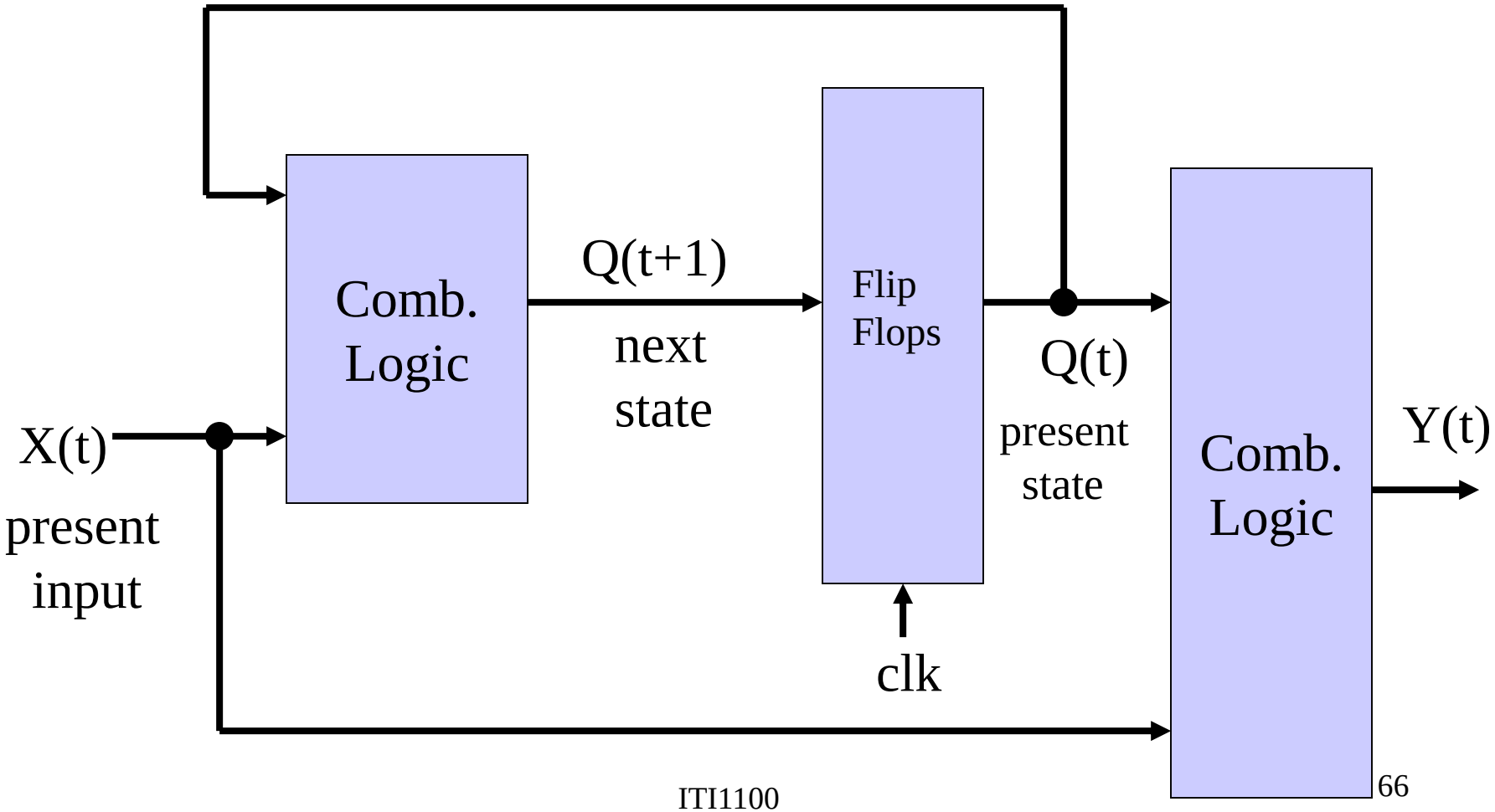
(a) Circuit diagram

Mealy and Moore Models

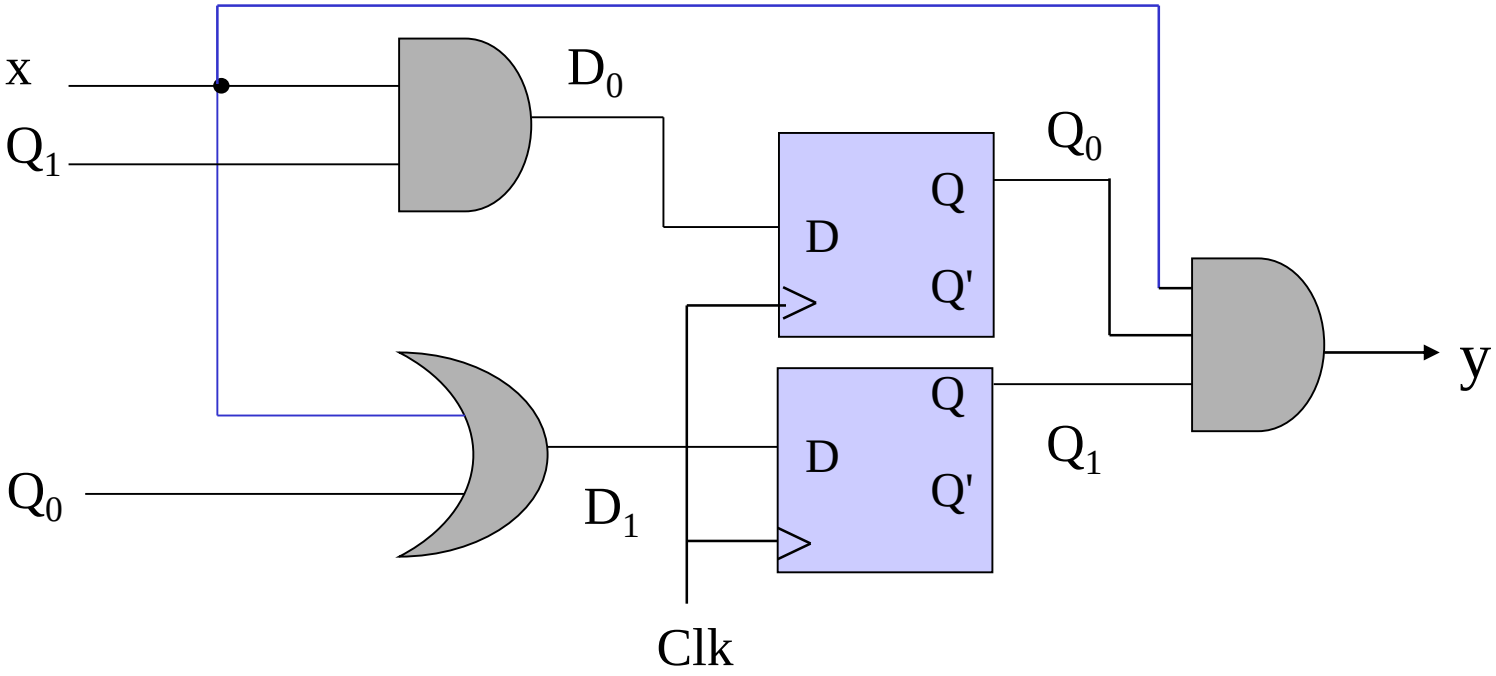
- A general model of a Sequential Circuit has
  - Inputs
  - Outputs
  - Internal States
- Circuits with different states know as Finite State Machines (FSMs)
- There are two "standard models"
  - Mealy Model, known as Mealy Finite State Machine (or Mealy machine)
  - Moore Model known as Moore Finite State Machine (or Moore Machine)

Mealy Machine

- Output based on state and present input

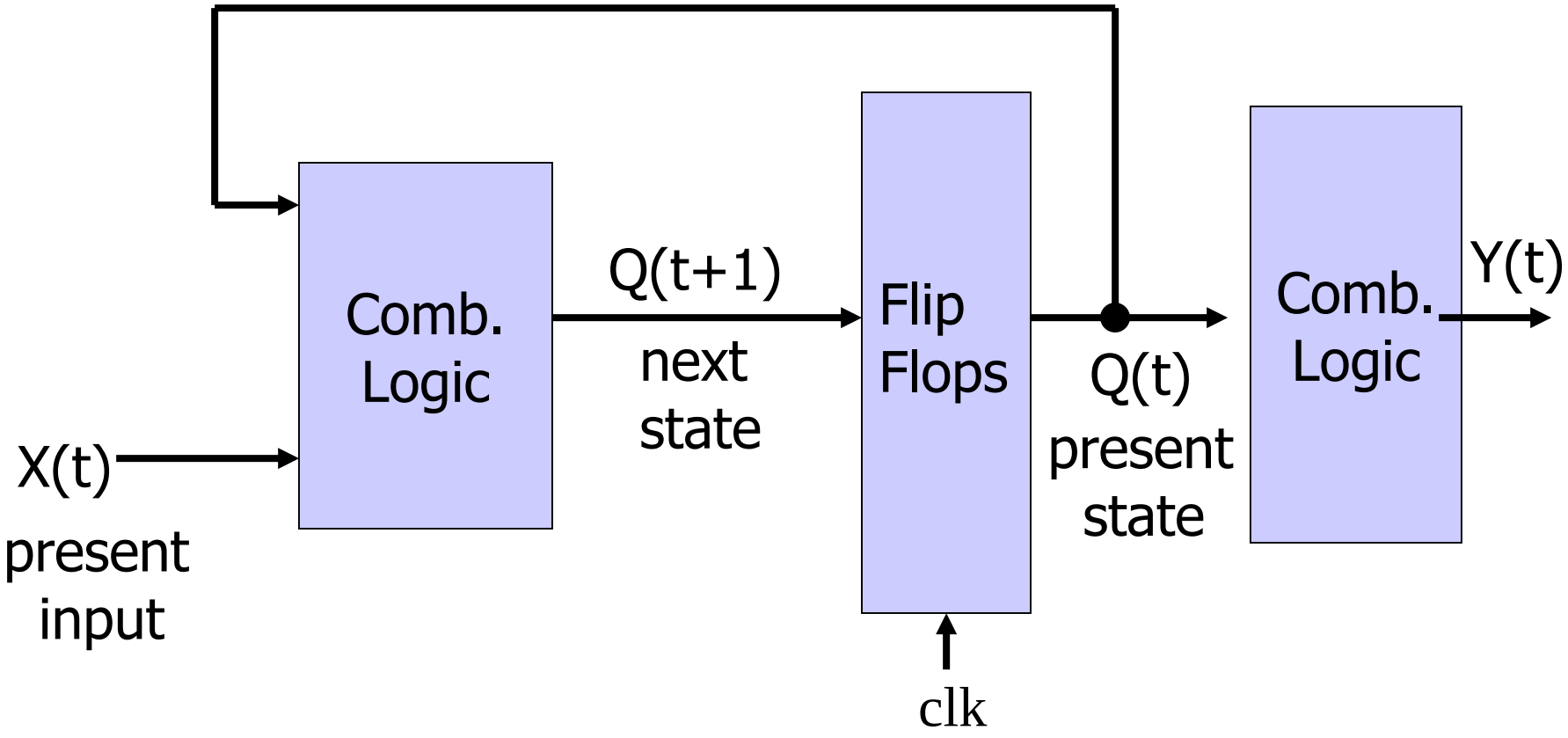


Example of Mealy Machine



Moore Machine

- Output based on state only



Example of Moore Machine

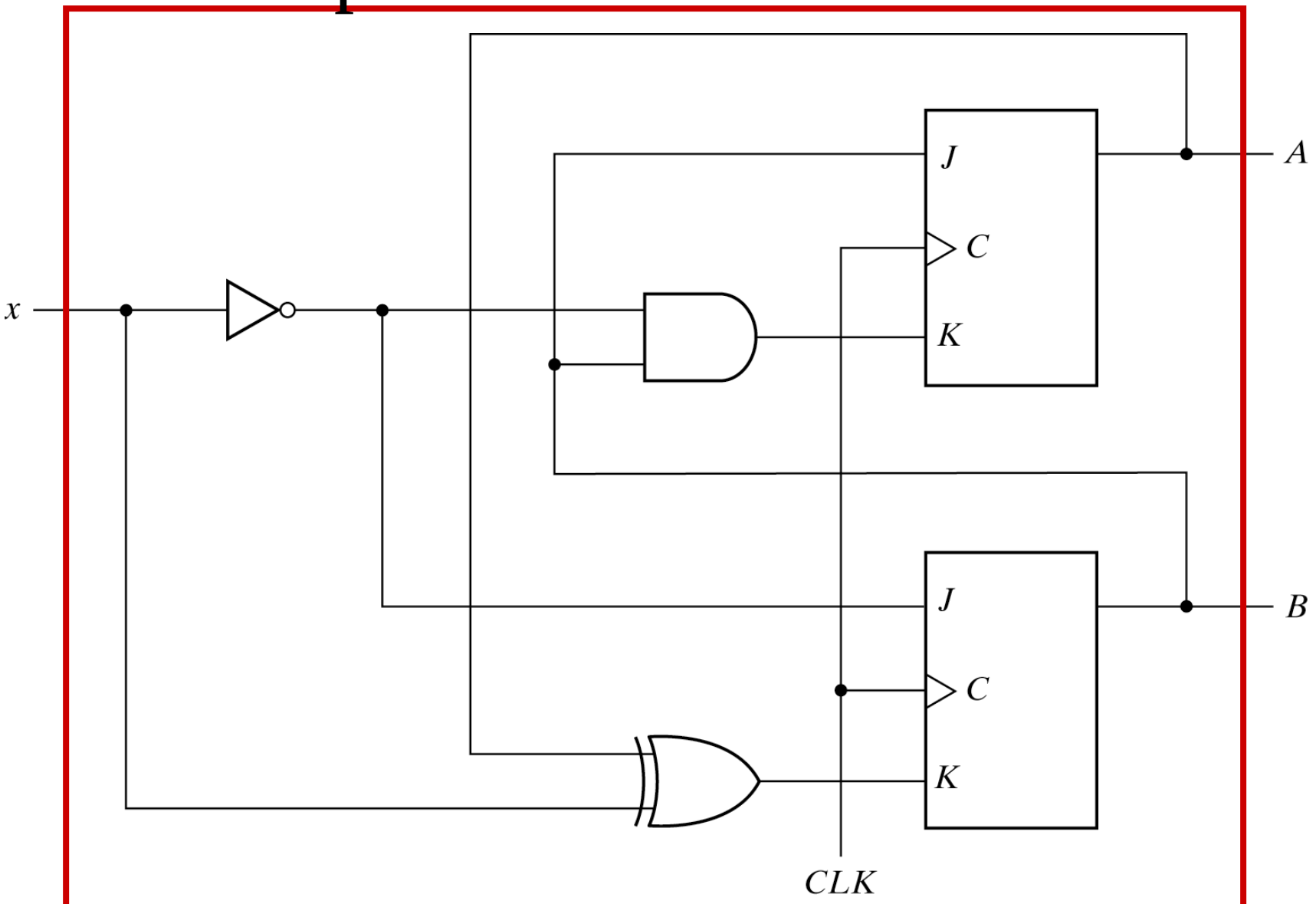


Fig. 5-18 Sequential Circuit with *JK* Flip-Flop

Example 1 D FF Analysis

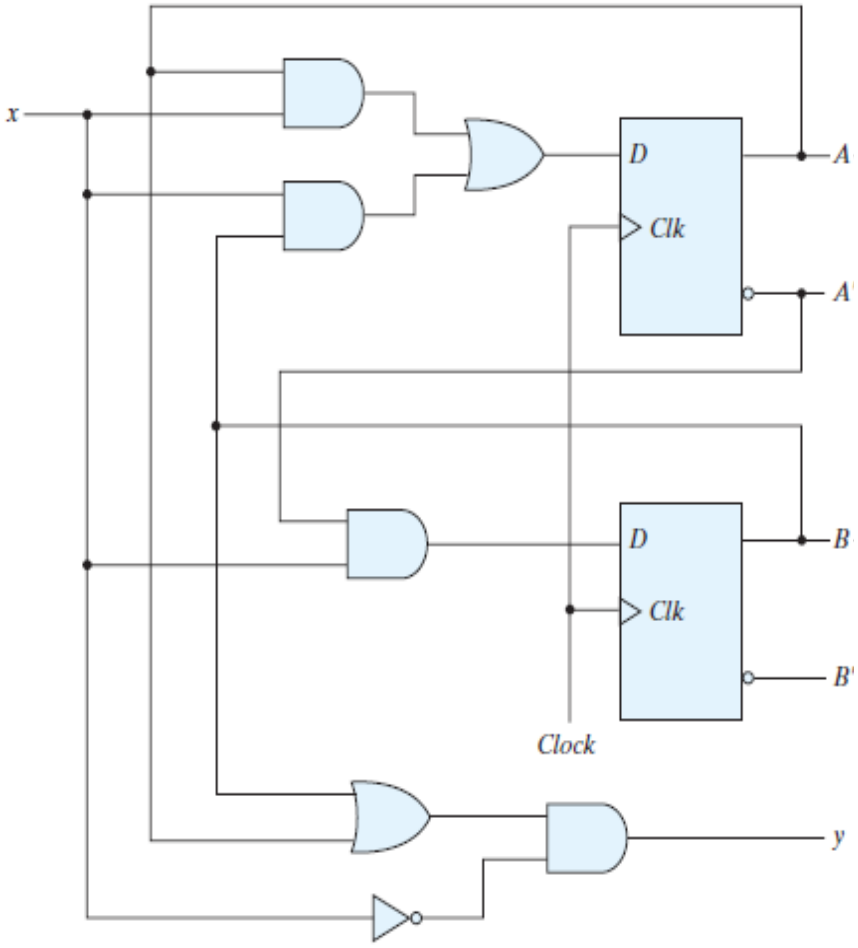
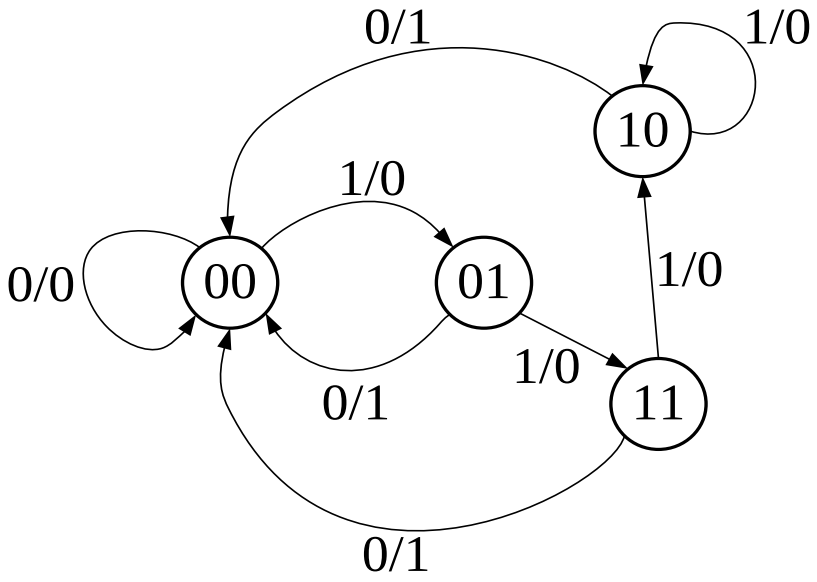


FIGURE 5.15
Example of sequential circuit



Zero Detector

Mealy FSM
(Outputs indicated on transition arrows)

Example 2 JK FF Analysis

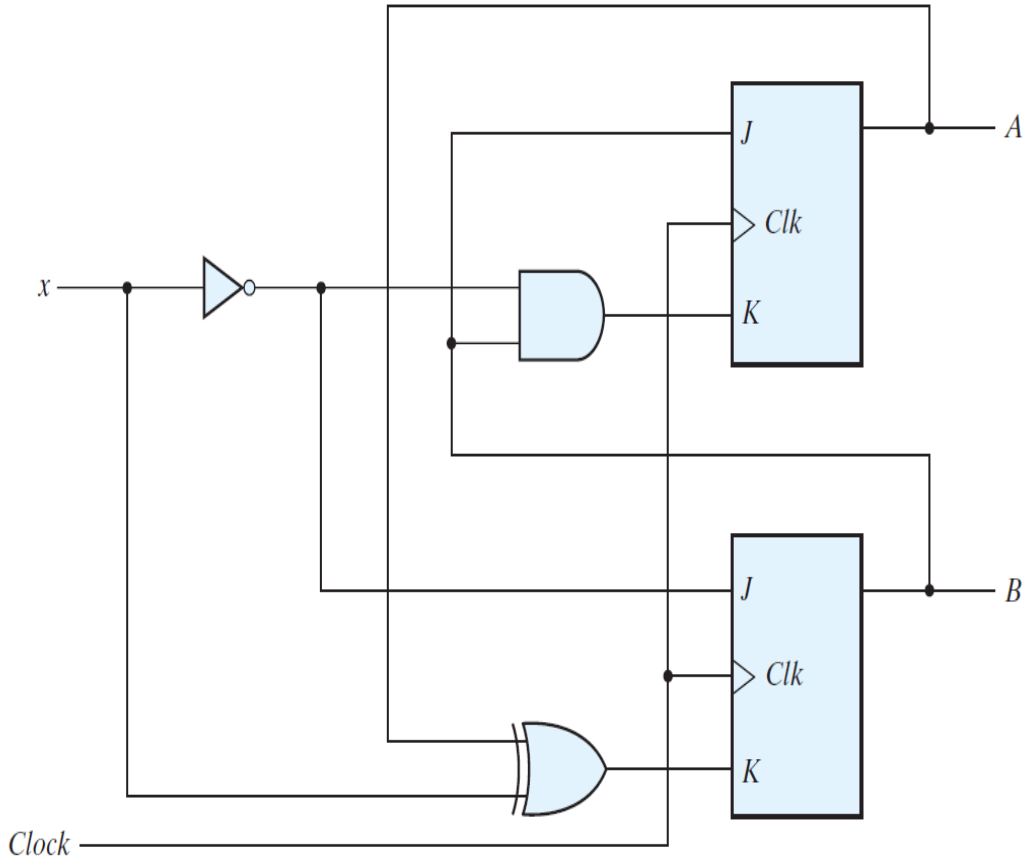
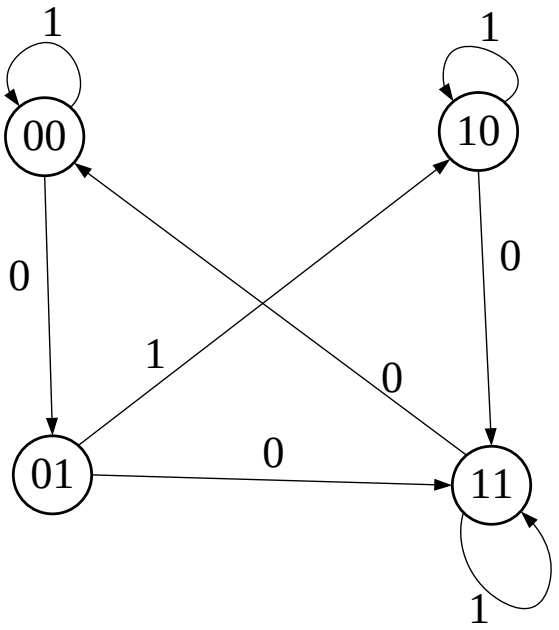
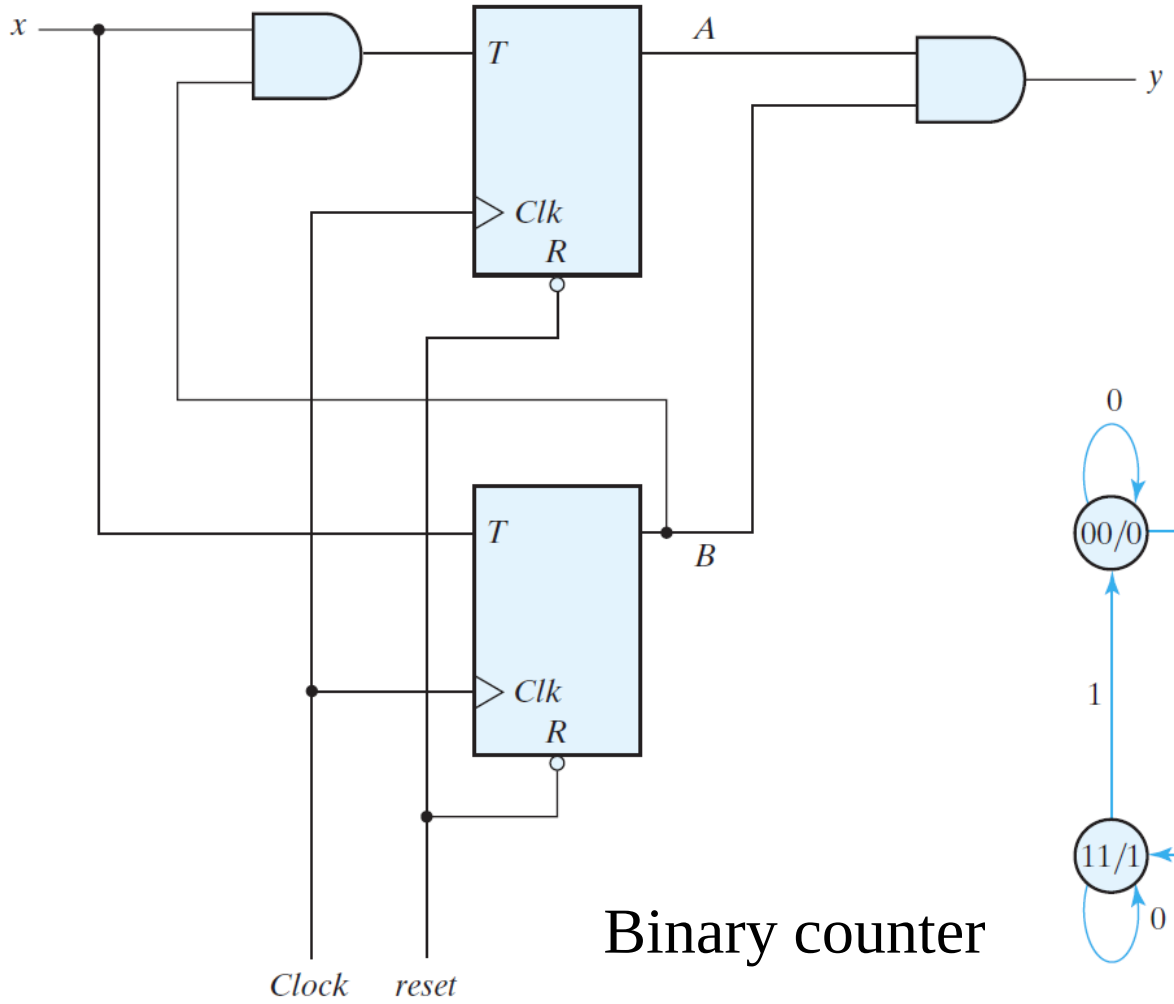


FIGURE 5.18
Sequential circuit with *JK* flip-flop



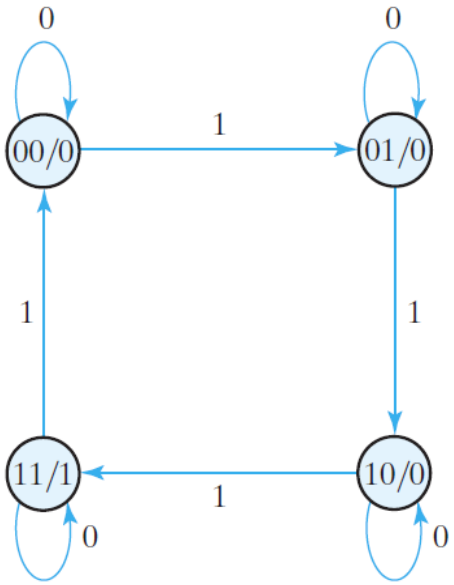
Moore FSM
(Outputs reflected by FF states)

Example 3 T FF Analysis



(a) Circuit diagram

Moore FSM
(Outputs reflected by FF states, within state bubbles)



(b) State diagram

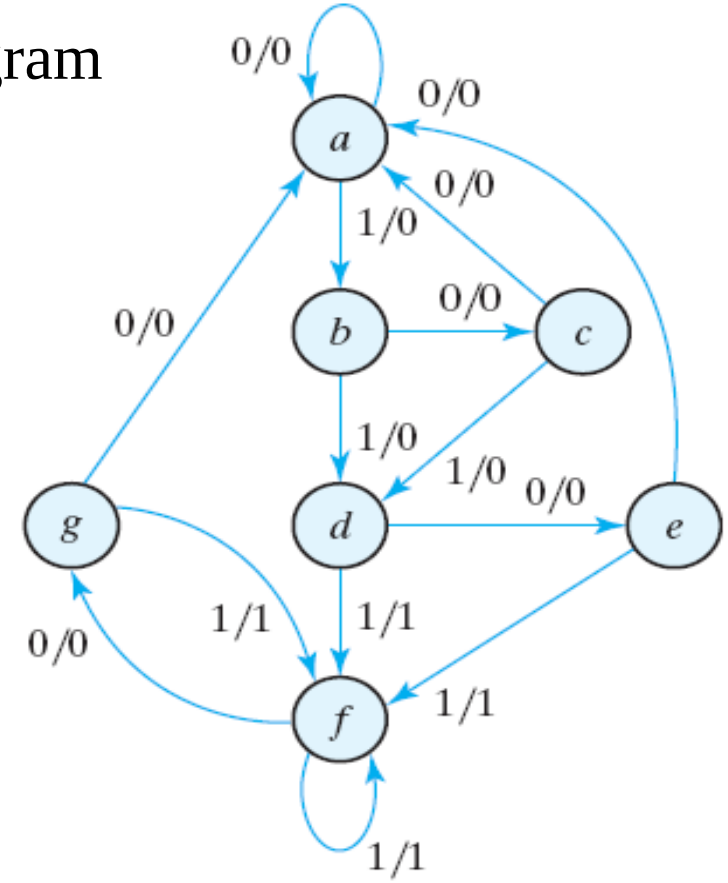
Binary counter

Design of Sequential Circuits

- So far, we have examined the analysis of sequential circuits
  - Circuit Diagram →Equations →State Table → State diagram
- Now we want to design sequential circuits
  - State Diagram → State Reduction → State Assignment → State Table → Select type of FF's →Derive: FF input Equations, State equations; Output equations (minimized) → Circuit Diagram
  - Note: FF Input Equations/Output Equations lead to combinational circuit design.

State Reduction (1)

- Consider a the following state diagram



- The following is the output sequence generated by the FSM for the given input sequence:

state	*a*	*a*	*b*	*c*	*d*	*e*	*f*	*f*	*g*	*f*	*g*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

State Reduction (2)

- Possible that an FSM with fewer states can generate same input/output sequences
  - Nice to have method to reduce # of states in FSM
- An approach:
  - Generate a state table from the given state diagram
  - Apply the following algorithm:
    - 2 states are equivalent if
      - They give the same output for each member of the set of inputs
      - They send the circuit to the same or to an equivalent state
    - When 2 states are equivalent, one can be removed without altering the input/output relationship:
      - Remove the row(s) in the state table where present state equals the removed state
      - Where ever the removed state occurs in the Next state column(s), it is replaced by its equivalent state.

State Reduction (3)

- Obtaining the State table from State diagram of the previous slide

State e and g are equivalent→ have the same next states and same output when x=0, x=1

State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Reducing the State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

Remove row with present state g and replace it by e in remaining next states

State Reduction (4)

Reducing the State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

f and d are equivalent

Reduced State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
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

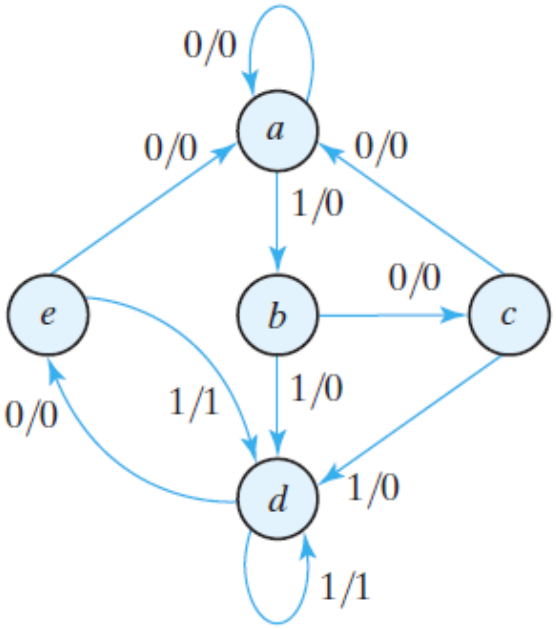
111100

State Reduction (4)

- New state machine contains only 5 states and produces same input/output sequences
  - May result in less equipment, but May not lead to savings in Flip-flops
  - In practice, state reduction may be skipped.

Reduced State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
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	*a*	*a*	*b*	*c*	*d*	*e*	*d*	*d*	*e*	*d*	*e*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

State Assignment (1)

- To assign coded binary values to the states: for circuit with *m* states, codes must contain *n bits* where $2^n \geq m$
  - E.g., with 7 states, require 3 bits, one state is unused
  - E.g., with 5 states, require 3 bits, three states are unused
- Unused states can be treated as don't care conditions
  - Can simplify combinational circuits
- Coding states
  - Use counting order: fives states use 000 to 100 (101, 110, 111 are unused)
  - Use Gray code: Only one bit changes from one number to next
  - One-hot assignment: as many bits as states and only 1 bit set in the state number

State Assignment (2)

Table 5.9
Three Possible Binary State Assignments

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	110	10000

- One-Hot
  - Usually leads to simpler decoding logic for next state and output
  - FSM can be faster than that with sequential binary encoding
  - Silicon area required for extra flip-flops can be offset by area saved in simpler decoding logic. No guarantee in this trade-off, must be evaluated for given design

Design Procedure

- Design of systems that input flip flops and combinational logic
- Specifications start with a word description
- Create a state table to indicate next states
- Convert next states and outputs to output and flip flop input equations
  - Reduce logic expressions using truth tables
- Draw resulting circuits.

Design Procedure

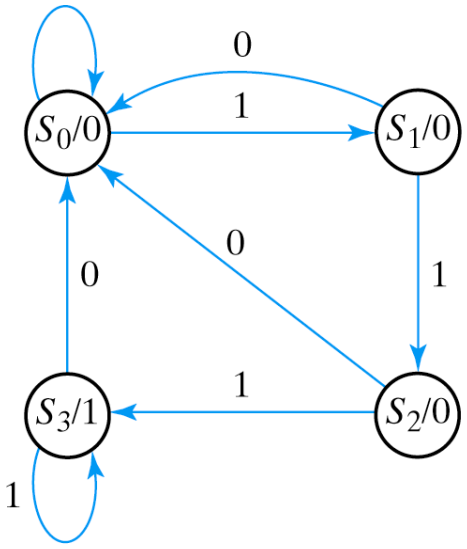
1. From the word description and specifications of the desired operation, derive a state diagram for the circuit.
2. Reduce the number of states if necessary.
3. Assign binary values to the states.
4. Obtain the binary-coded state table.
5. Choose the type of flip-flops to be used.
6. Derive the simplified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

Designing Finite State Machines

- **Specify the problem with words**
  - ***(e.g. Design a circuit that detects three consecutive 1 inputs)***
- **Derive a state diagram**
- **Assign binary values to states**
- **Develop a state table**
- **Choose flip flop type**
- **Use K-maps to simplify expressions**
  - **Flip flop input equations and output equations**
- **Create appropriate logic diagram**
  - **Should include combinational logic and flip flops**

Example: Detect 3 or More Consecutive 1 Inputs

State Diagram



- State S_0 : zero 1s detected
- State S_1 : one 1 detected
- State S_2 : two 1s detected
- State S_3 : three or more 1s detected

Fig. 5-24 State Diagram for Sequence Detector

- Moore circuit with output 0 at S0, S1, S2, and 1 at S3.
- Each state has 2 output arrows
- Two bits needed to encode state

State Table for Sequence Detector

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	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

- **Sequence of outputs, inputs, and flip flop states enumerated in state table**
- **Present state indicates current value of flip flops**
- **Next state indicates state after next rising clock edge**
- **Output is output value on current clock edge**

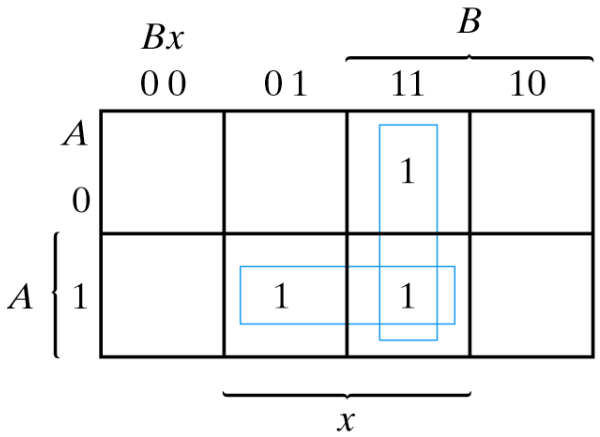
- **$S_0 = 00$**
- **$S_1 = 01$**
- **$S_2 = 10$**
- **$S_3 = 11$**

Design with D FF's

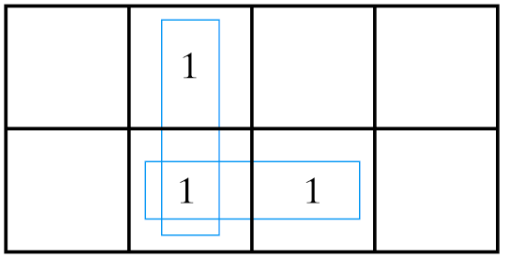
- Word to State Diagram (Done)
- Derive the State Table from State Diagram (Done)
- Choose type of Flip-Flop: D
  - Two D flip-flops to represent 4 states
  - Outputs of FF: A and B
- Develop the FF Input equations and output equations
- Derive Circuit Diagram

Finding FF Input Equation and Output Equation

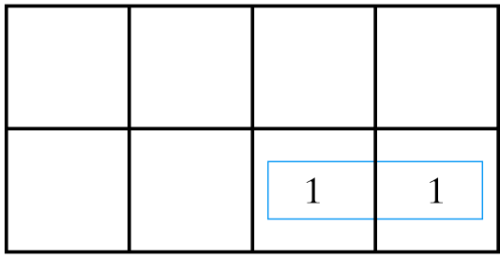
- FF D input = "next state", $Q(t+1) = D_Q$
- From state table, find FF input equation of D_Q with regard to present state and inputs
- Find output equation of y with regard to present state and inputs
- Create K-map directly from state table (3 K-maps for $D_A, D_B, and\ y$)
- Minimize K-maps to find SOP representations



$D_A = Ax + Bx$

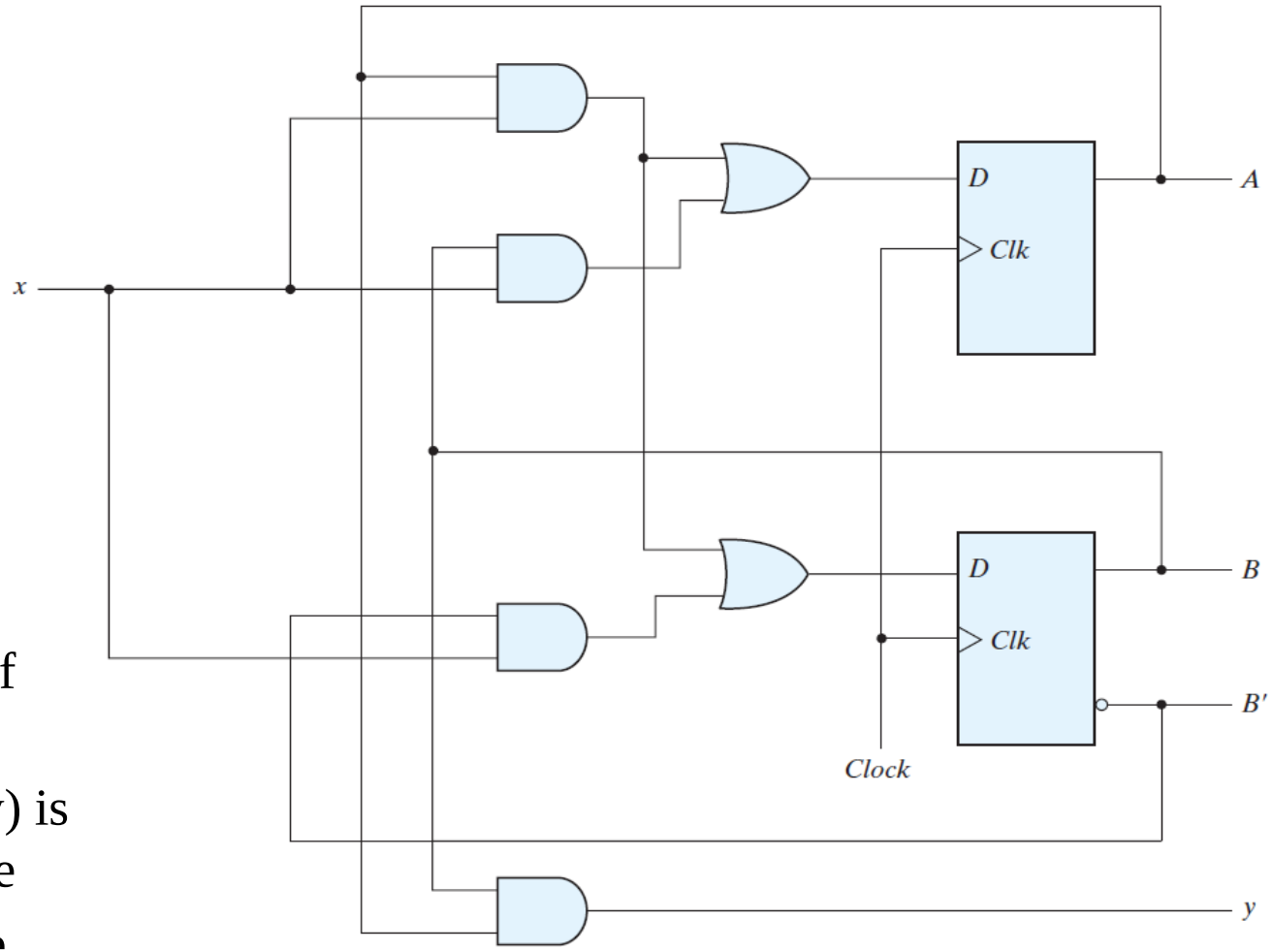


$D_B = Ax + B'x$



$y = AB$

Circuit Diagram – Sequence Decoder



- Note location of state flip flops
- Output value (y) is function of state
- This is a Moore machine.

FIGURE 5.29
Logic diagram of a Moore-type sequence detector

Excitation Tables (JK and T FF's)

- Input FF equations derived indirectly from state table
  - For D FF, input equations obtained directly from next state
  - Not so for JK and T FF's – need excitation tables.
- Excitation table: list required inputs for a given state change.

Characteristic Tables

JK Flip Flop

J	K	Q(t+1)	Comment
0	0	Q(t)	No Change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

T Flip Flop

T	Q(t+1)	Comment
0	Q(t)	No change
1	Q'(t)	Complement (toggle)

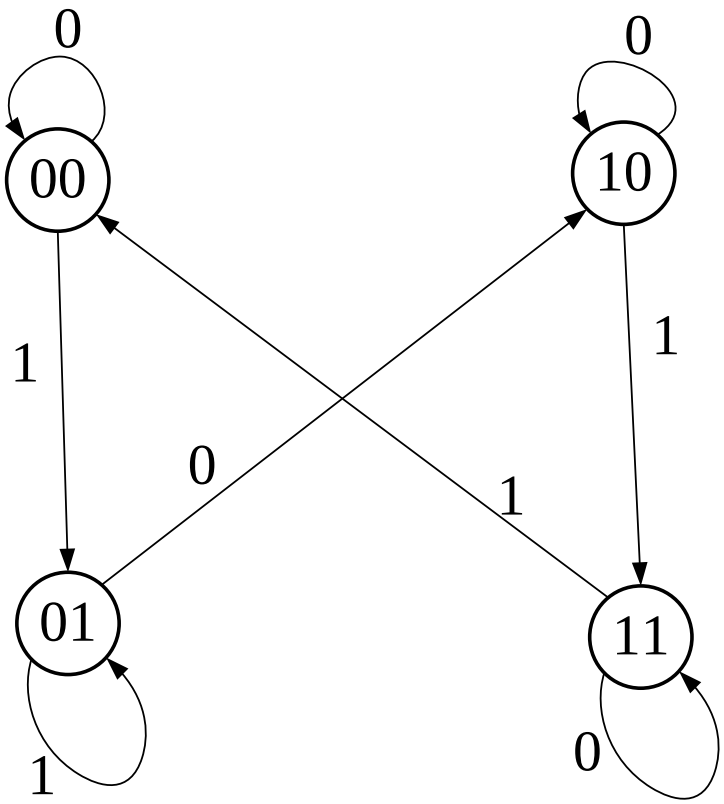
Excitation Tables

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

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

Design with JK FF's

- Design the circuit for the following state diagram
  - State table, FF Input equations, Circuit



Design with JK FF's – State Table

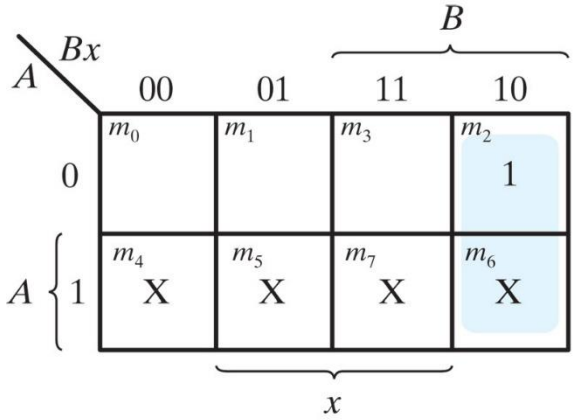
Add FF inputs in the state table to get FF input equations

Table 5.13
State Table and JK Flip-Flop Inputs

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
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

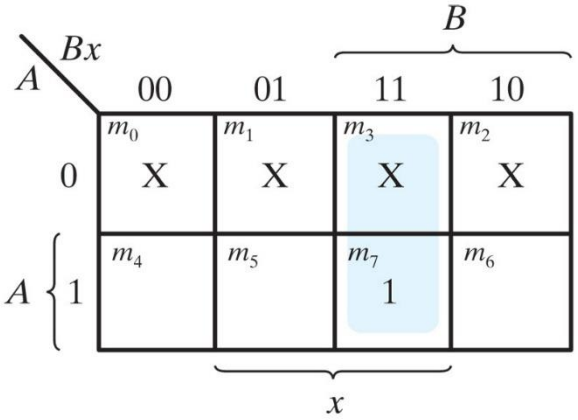
Design with JK FF's – FF Input Eqn's

$J_A = Bx'$

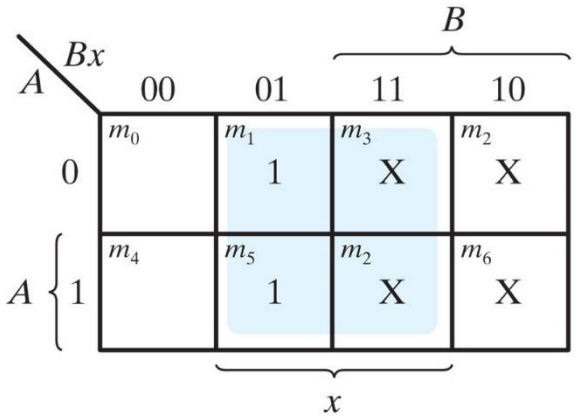


$J_A = Bx'$

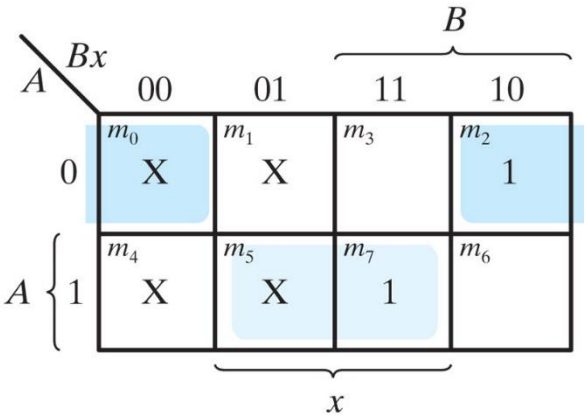
$K_A = Bx$



$K_A = Bx$

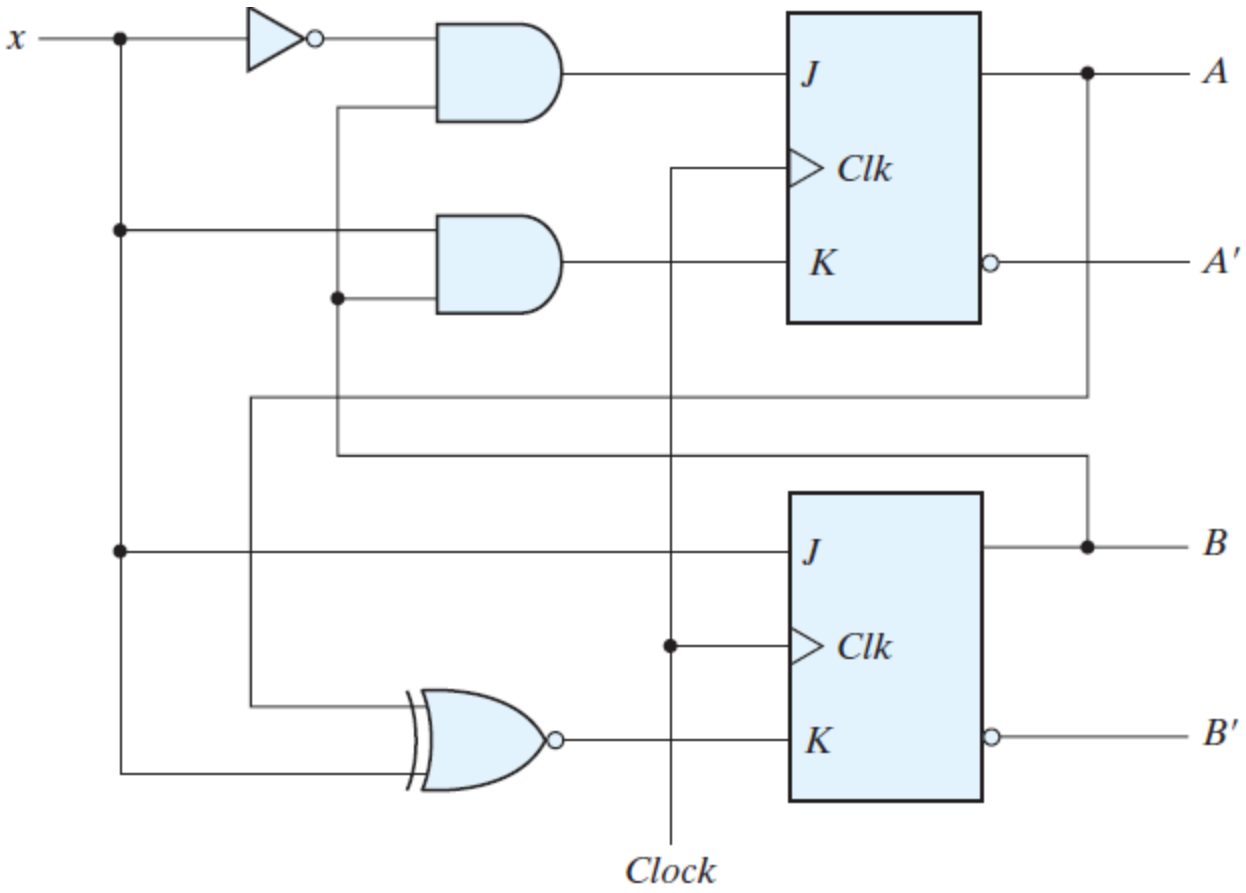


$J_B = x$



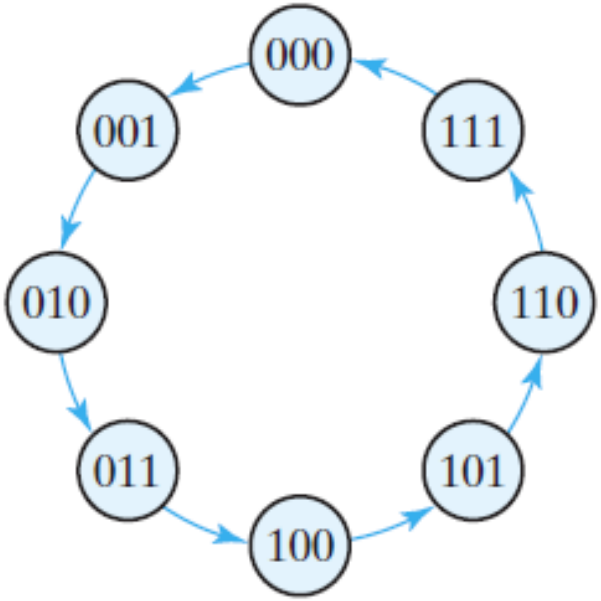
$K_B = (A \oplus x)'$

Design with JK FF's – Circuit



Design with T FF's

- Design the circuit for the following state diagram, a 3-bit binary counter:
  - State Diagram: no input/output, triggered by a clock edge
  - 3 T flip-flops : FF outputs map binary count
  - State table, FF Input equations, Output equations (no output), Circuit



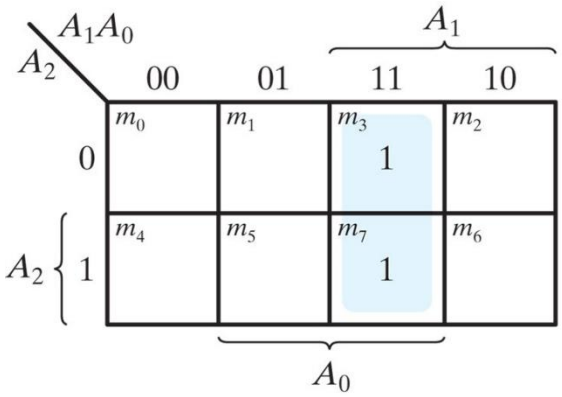
Design with T FF's – State Table

Add FF inputs in the state table to get FF input equations

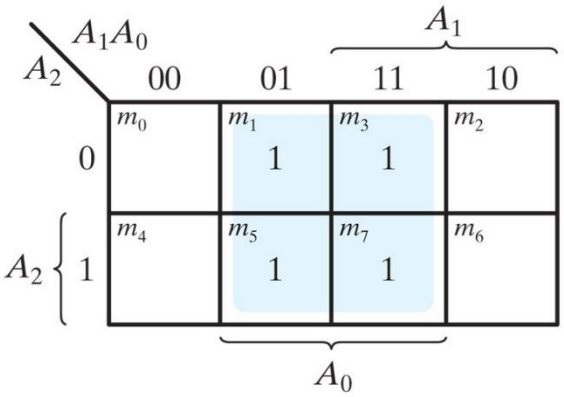
Table 5.14
State Table for Three-Bit Counter

Present State			Next State			Flip-Flop Inputs		
A_2	A_1	A_0	A_2	A_1	A_0	T_{A2}	T_{A1}	T_{A0}
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	1	1
1	1	1	0	0	0	1	1	1

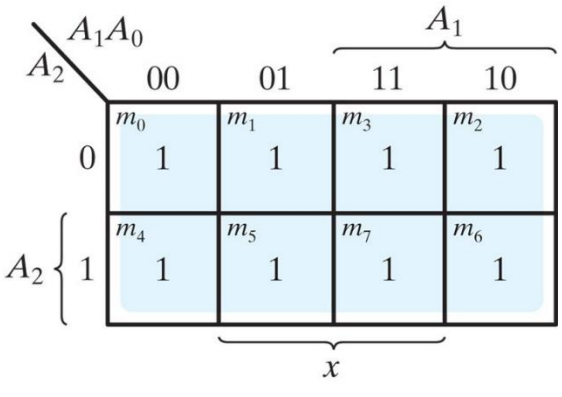
Design with T FF's – FF Input Equations



$T_{A2} = A_1A_0$

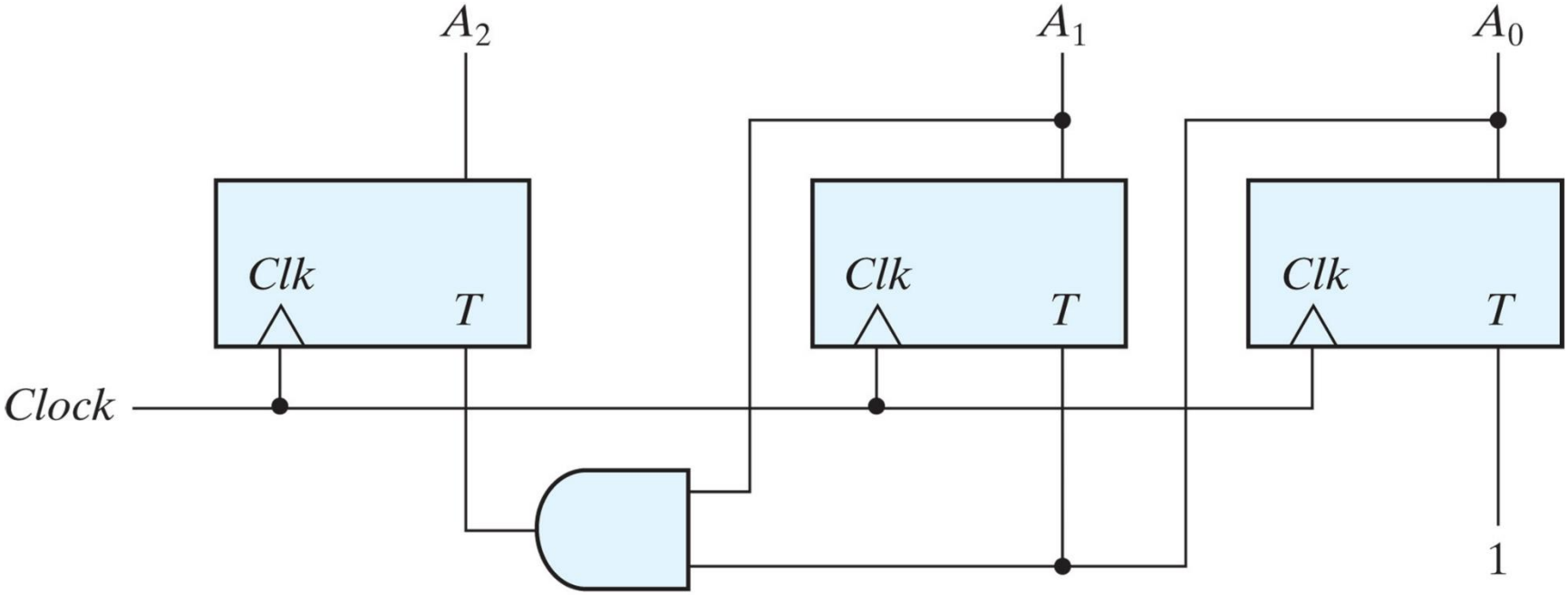


$T_{A1} = A_0$



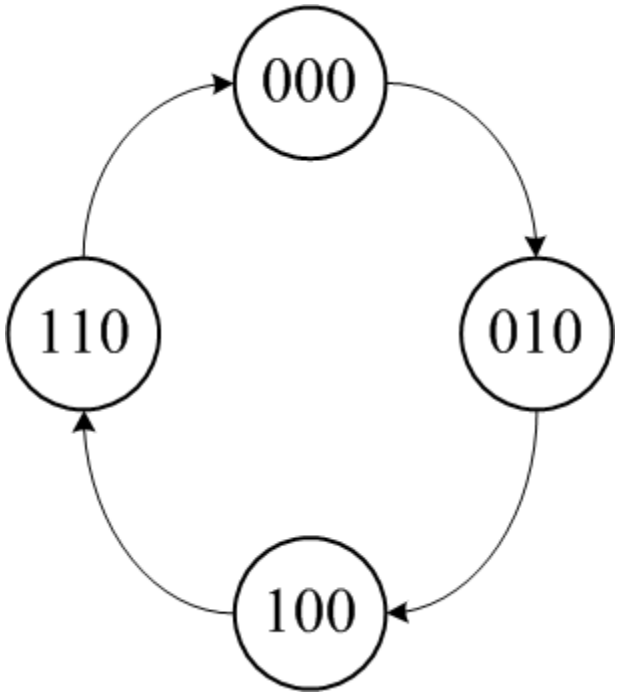
$T_{A0} = 1$

Design with T FF's – Circuit



Design with JK FF's – Even Counter

- Design the circuit for the following state diagram, a 3-bit even counter
  - State table, FF Input equations, Circuit
  - What is interesting about the following state diagram?



Design with JK FF's – Even Counter – State Table

Present State	Next State	JK FF Inputs					
ABC	ABC	J_A	K_A	J_B	K_B	J_C	K_C
000	010	0	x	1	x	0	x
001	xxx	x	x	x	x	x	x
010	100	1	x	x	1	0	x
011	xxx	x	x	x	x	x	x
100	110	x	0	1	x	0	x
101	xxx	x	x	x	x	x	x
110	000	x	1	x	1	0	x
111	xxx	x	x	x	x	x	x

Design with JK FF's – Even Counter – FF Input Eqn's

$J_A = B$

A \ Bx	00	01	11	10
0	m_0	m_1 x	m_3 x	m_2 1
1	m_4 x	m_5 x	m_7 x	m_6 x

$K_A = B$

A \ Bx	00	01	11	10
0	m_0 x	m_1 x	m_3 x	m_2 x
1	m_4	m_5 x	m_7 x	m_6 1

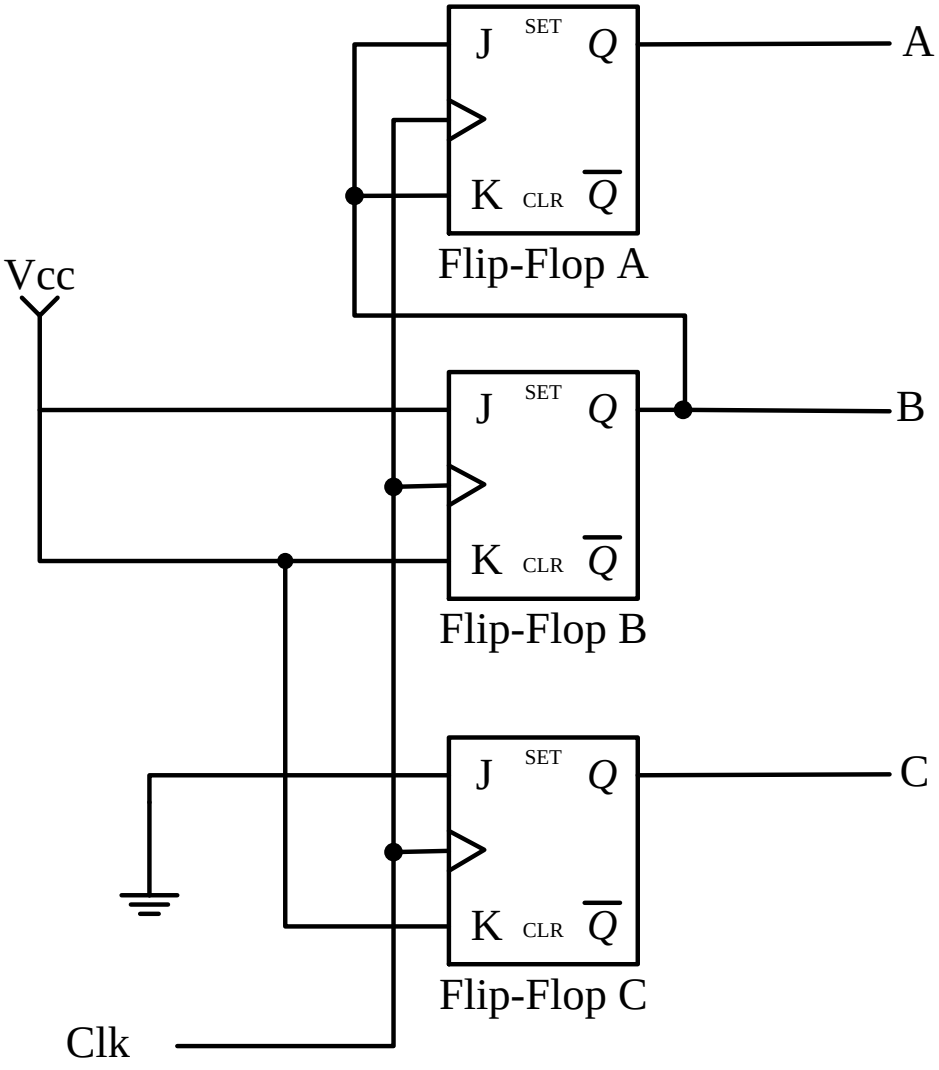
$J_B = 1$

$K_B = 1$

$J_c = 0$

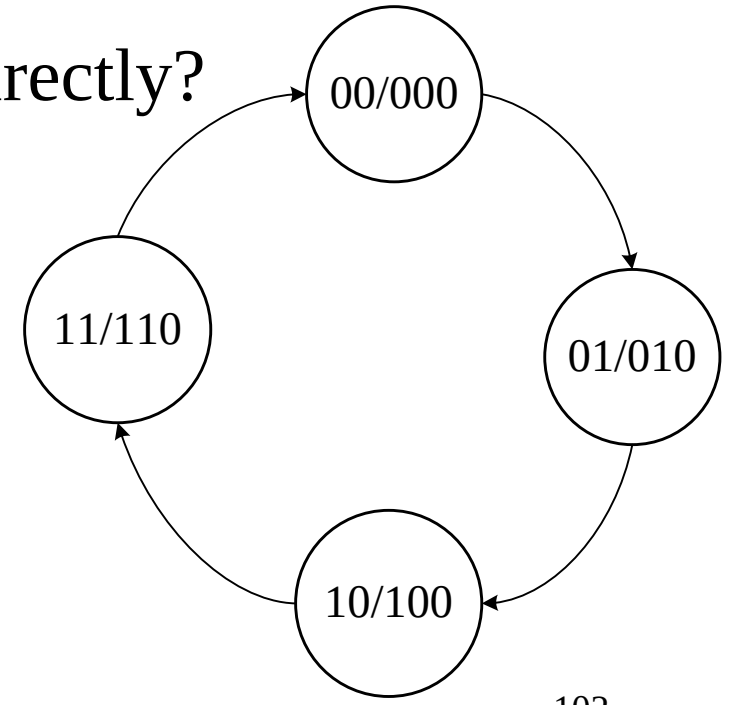
$K_c = 1$

Design with JK FF's – Even Counter – Circuit



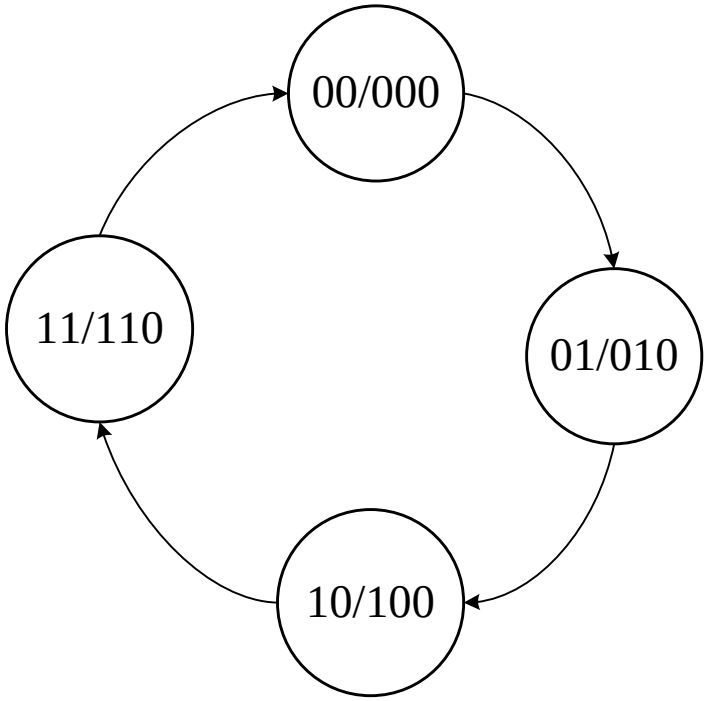
Design with JK FF's – Even Counter - Alternate

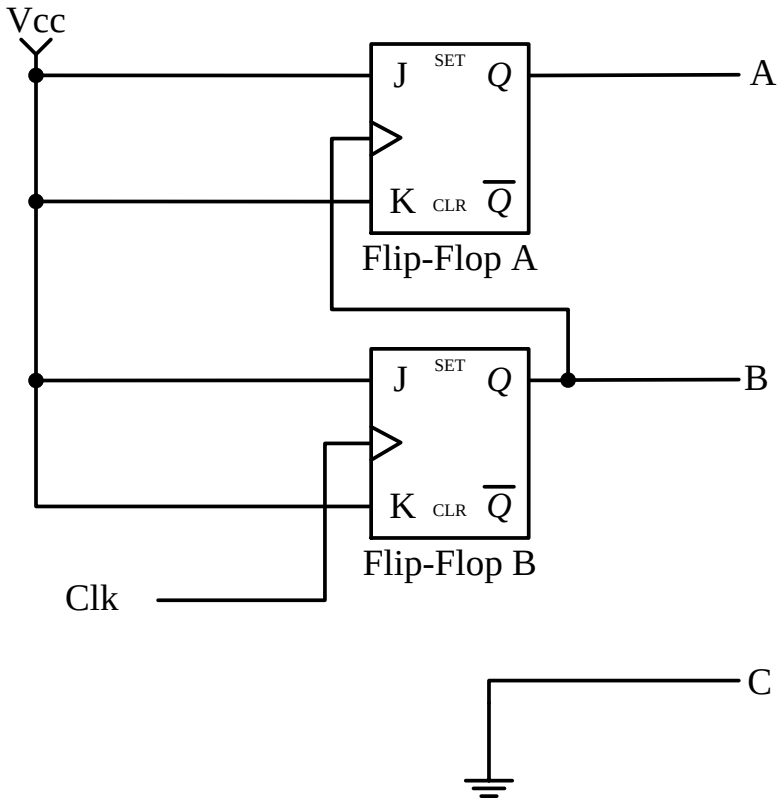
- Design the circuit for the following state diagram, a 3-bit event counter
  - State table, FF Input equations, Circuit
  - What is interesting about the following state diagram.
  - Can you deduce the circuit directly?
  - What would our design give us?



Design with JK FF's – Even Counter (alt) - Circuit

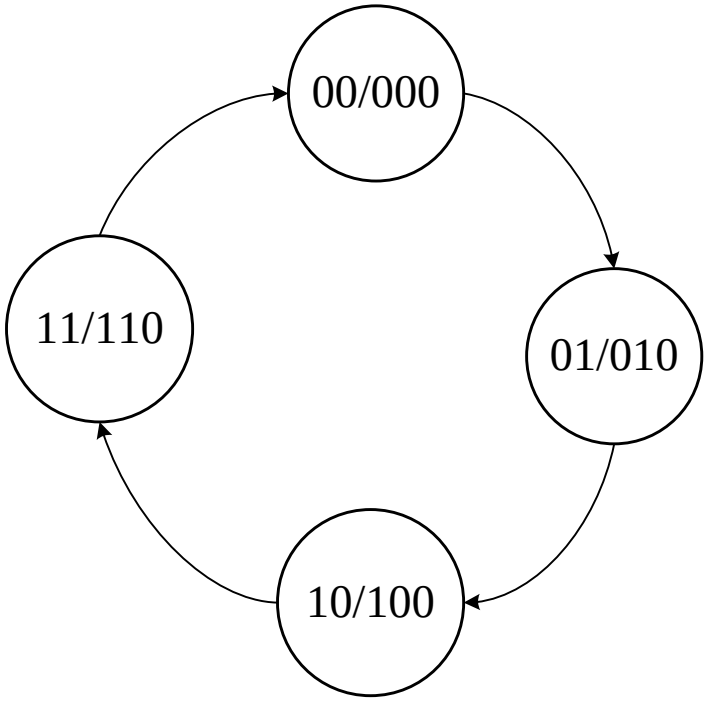
- Circuit deduced directly from the state diagram

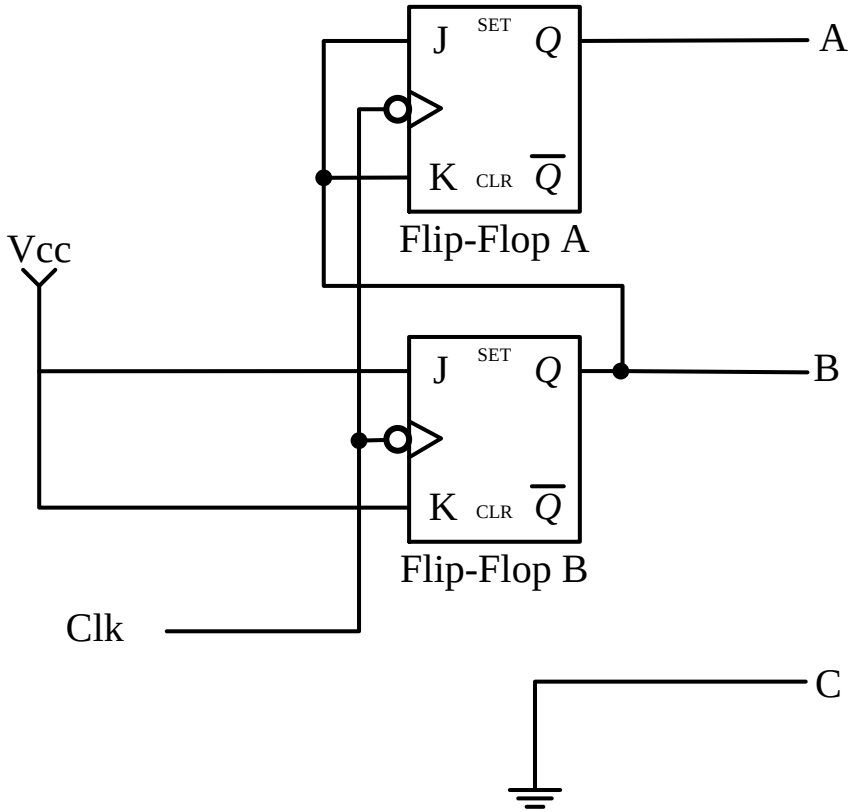




Design with JK FF's – Even Counter (alt) - Circuit

- Modified original circuit.
- Check with design.





END of Chapter 5