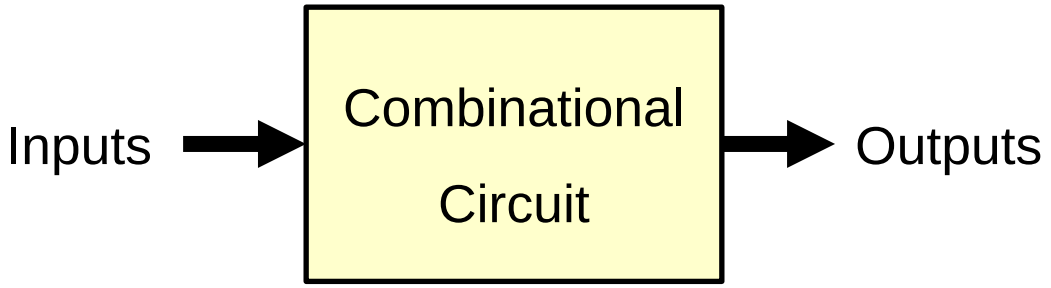
Synchronous Sequential Logic

Combinational versus Sequential

- Two classes of digital circuits
  - Combinational Circuits
  - Sequential Circuits
- Combinational Circuit
  - Outputs = F(Inputs)
  - Function of Inputs only
  - NO internal memory



- Sequential Circuit
  - Outputs is a function of Inputs and internal Memory
  - There is an internal memory that stores the state of the circuit
  - Time is very important: memory changes with time

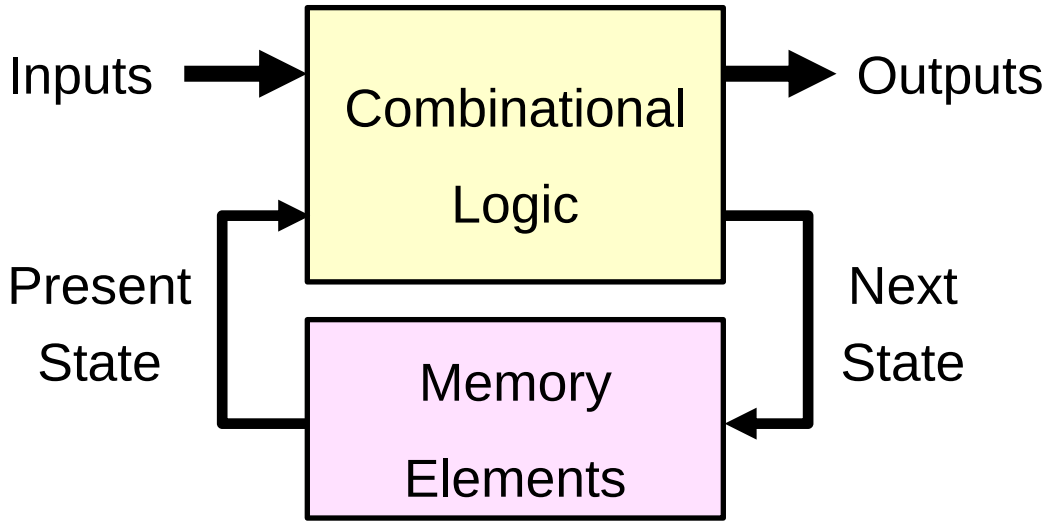
Introduction to Sequential Circuits

A Sequential circuit consists of:

1. Memory elements:
   - ✧ **Latches** or **Flip-Flops**
   - ✧ Store the **Present State**
2. Combinational Logic
   - ✧ Computes the **Outputs** of the circuit

     Outputs depend on Inputs and Current State
   - ✧ Computes the **Next State** of the circuit

     Next State also depends on the Inputs and the Present State



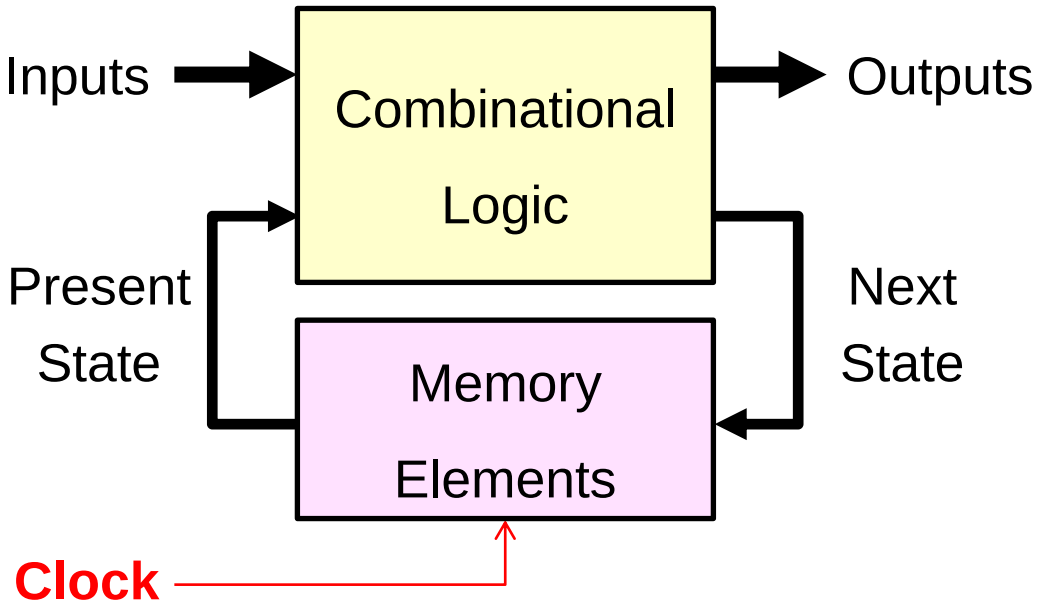
Two Types of Sequential Circuits

1. **Synchronous** Sequential Circuit
   - ✧ Uses a clock signal as an additional input
   - ✧ Changes in the memory elements are controlled by the clock
   - ✧ Changes happen at discrete instances of time
2. **Asynchronous** Sequential Circuit
   - ✧ No clock signal
   - ✧ Changes in the memory elements can happen at any instance of time

❖ Our focus will be on Synchronous Sequential Circuits

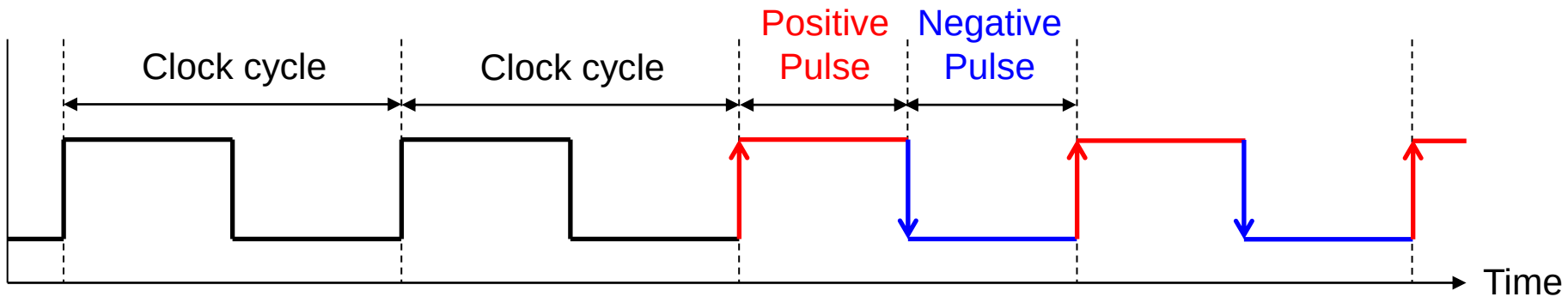
- ✧ Easier to design and analyze than asynchronous sequential circuits

Synchronous Sequential Circuits



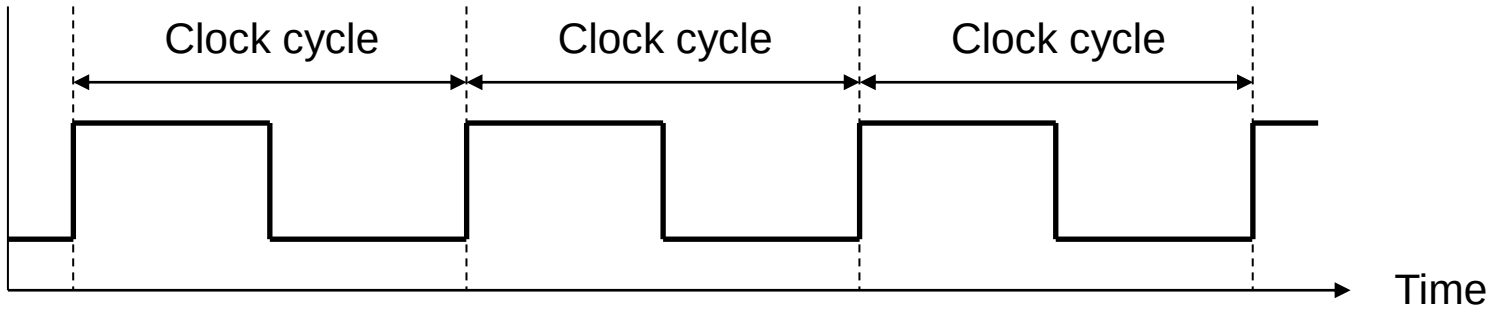
- Synchronous sequential circuits use a **clock signal**
- The clock signal is an input to the memory elements
- The clock determines **when** the memory should be updated
- The **present state** = output value of memory (stored)
- The **next state** = input value to memory (not stored yet)

The Clock



- Clock is a periodic signal = Train of pulses (1's and 0's)
- The same clock cycle repeats indefinitely over time
- **Positive Pulse**: when the **level** of the clock is **1**
- **Negative Pulse**: when the **level** of the clock is **0**
- **Rising Edge**: when the clock goes **from 0 to 1**
- **Falling Edge**: when the clock goes **from 1 down to 0**

Clock Cycle versus Clock Frequency

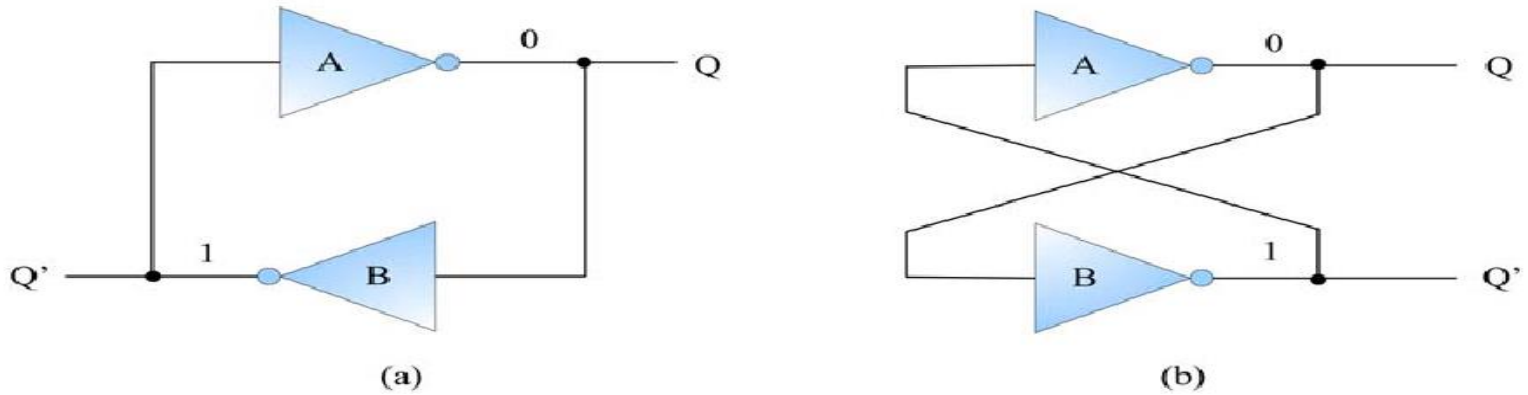


- Clock cycle (or period) is a time duration
  - Measured in seconds, milli-, micro-, nano-, or pico-seconds
  - 1 ms = 10^{-3} sec, 1 µs = 10^{-6} sec, 1 ns = 10^{-9} sec, 1 ps = 10^{-12} sec
- Clock frequency = number of cycles per second (Hertz)
  - 1 Hz = 1 cycle/sec, 1 KHz = 10^{3} Hz, 1 MHz = 10^{6} Hz, 1 GHz = 10^{9} Hz
- Clock frequency = 1 / Clock Cycle
  - Example: Given the clock cycle = 0.5 ns = 0.5 ×10^{-9} sec
  - Then, the clock frequency = 1/(0.5×10^{-9}) = 2×10^{9} Hz = 2 GHz

Memory Elements

- Memory can store and maintain binary state (0's or 1's)
  - Until directed by an input signal to change state
- Main difference between memory elements
  - Number of inputs they have
  - How the inputs affect the binary state
- Two main types:
  - Latches are level-sensitive (the level of the clock)
  - Flip-Flops are edge-sensitive (sensitive to the edge of the clock)
- Flip-Flips are used in synchronous sequential circuits
- Flip-Flops are built with latches

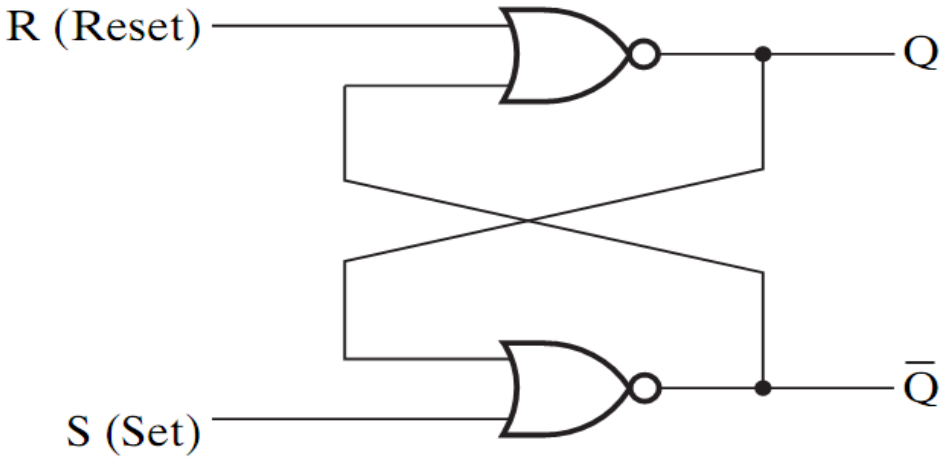
Memory Elements - Latches



- A basic memory element, as shown in Figure (a), is the **latch.**
- A latch is a circuit capable of storing one bit of information.
- The latch circuit consists of two inverters; with the output of one connected to the input of the other.
- The latch circuit has two outputs, one for the stored value (**Q) and one for its** complement (**Q').**
- Figure (b) shows the same latch circuit re-drawn to illustrate the two complementary outputs.
- The problem with the latch formed by **NOT gates is that we can't change the** stored value. For example, if the output of inverter B has logic 1, then it will be latched forever; and there is no way to change this value.

SR Latch

- An **SR Latch** can be built using two NOR gates
- Two inputs: S (Set) and R (Reset)
- Two outputs: Q and $\overline{Q}$



(a) Logic diagram

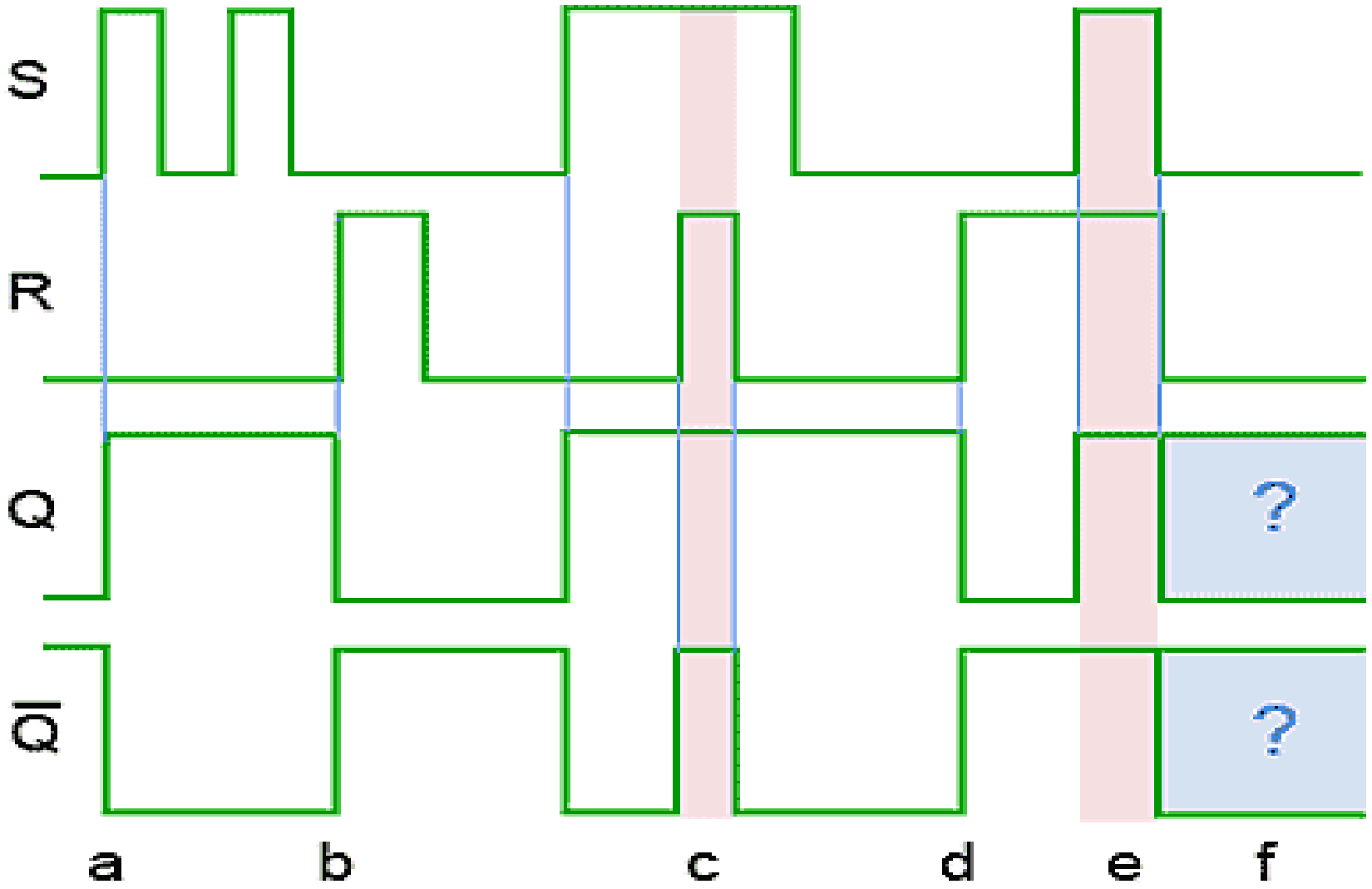
S	R	Q	Q̄	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

(b) Function table

SR Latch Operation

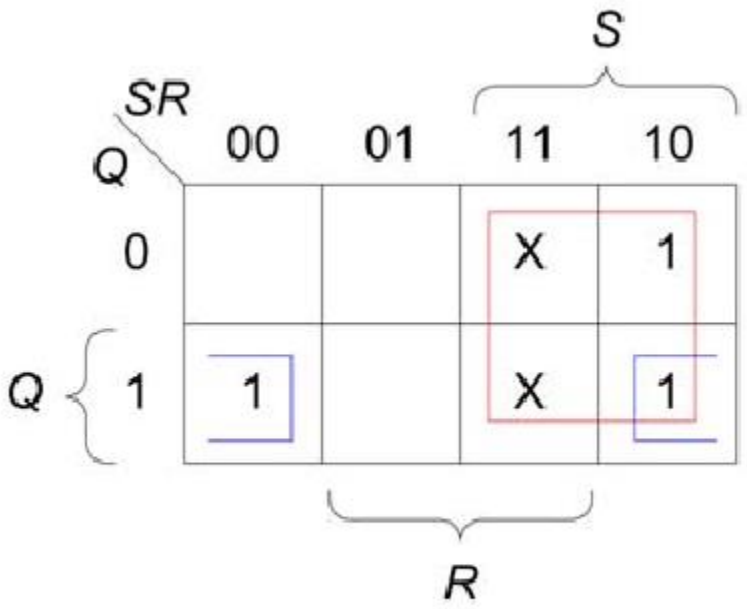
- If $S = 1$ and $R = 0$ then **Set** ($Q = 1,\ \overline{Q} = 0$)
- If $S = 0$ and $R = 1$ then **Reset** ($Q = 0,\ \overline{Q} = 1$)
- When $S = R = 0$, Q and $\overline{Q}$ are unchanged
- The latch stores its outputs Q and $\overline{Q}$ as long as $S = R = 0$
- When $S = R = 1$, Q and $\overline{Q}$ are undefined (should never be used)

SR Latch Timing Diagram



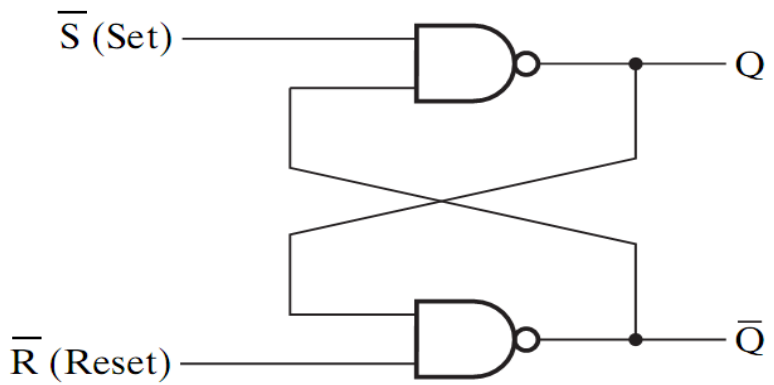
Characteristic Equation of the SR Latch

Q(t)	S	R	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Indeterminate



$Q(t+1) = S + R'Q$ $SR = 0$

$\overline{S}$ $\overline{R}$ Latch with NAND Gates



(a) Logic diagram

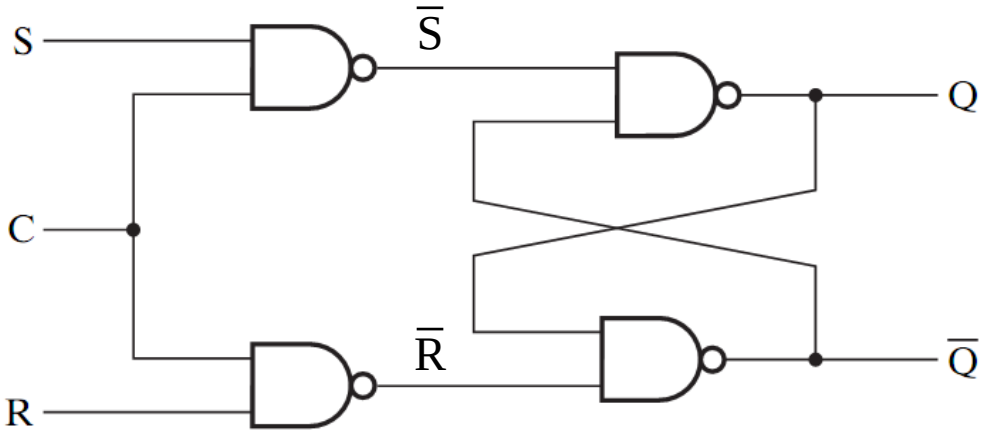
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	1	1	0	Set state
1	1	1	0	
1	0	0	1	Reset state
1	1	0	1	
0	0	1	1	Undefined

(b) Function table

Known as the $\overline{S}$ $\overline{R}$ Latch

- If $\bar{S} = 0$ and $\bar{R} = 1$ then **Set** ($Q = 1$, $\overline{Q} = 0$)
- If $\bar{S} = 1$ and $\bar{R} = 0$ then **Reset** ($Q = 0$, $\overline{Q} = 1$)
- When $\bar{S} = \bar{R} = 1$, Q and $\overline{Q}$ are unchanged (remain the same)
- The latch stores its outputs Q and $\overline{Q}$ as long as $\bar{S} = \bar{R} = 1$
- When $\bar{S} = \bar{R} = 0$, Q and $\overline{Q}$ are undefined (should never be used)

SR Latch with a Clock Input



(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

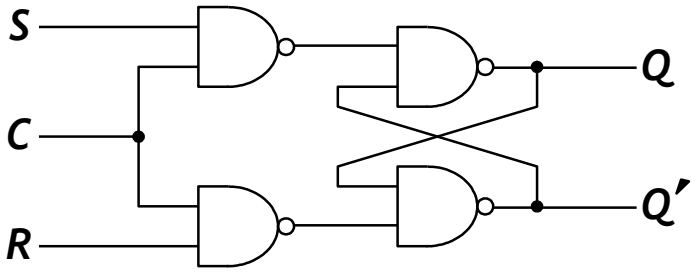
(b) Function table

- An additional Clock input signal **C** is used
- Clock controls **when** the state of the latch can be changed
- When **C=0**, the S and R inputs have no effect on the latch

  The latch will remain in the same state, regardless of S and R
- When **C=1**, then normal SR latch operation

SR Latch with a Clock Input

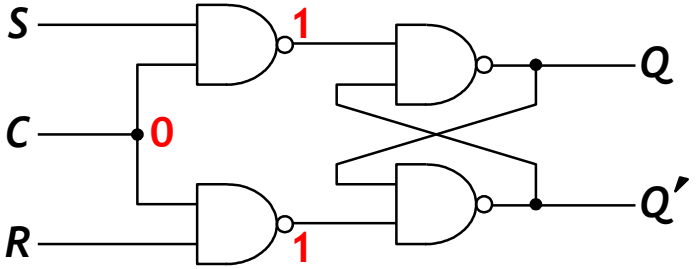
Logic Diagram



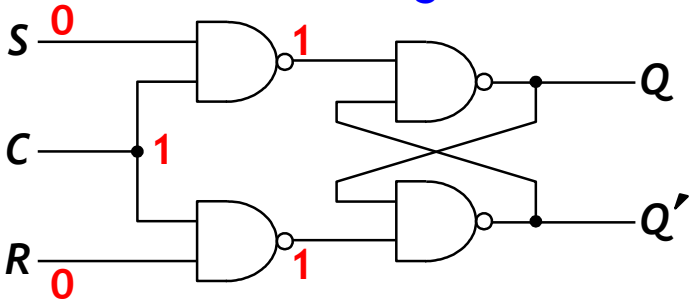
Function Table

C	S	R	Next State
0	X	X	No Change
1	0	0	No Change
1	0	1	Q=0; Reset
1	1	0	Q=1; Set
1	1	1	Indeterminate

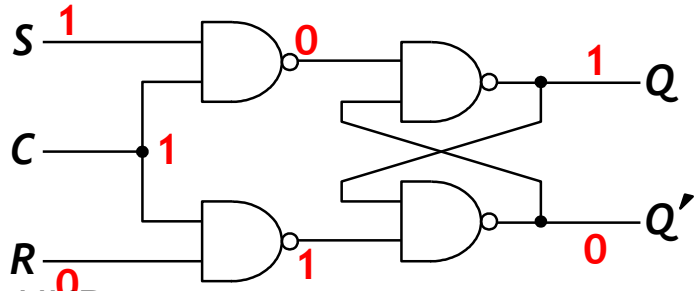
No Change



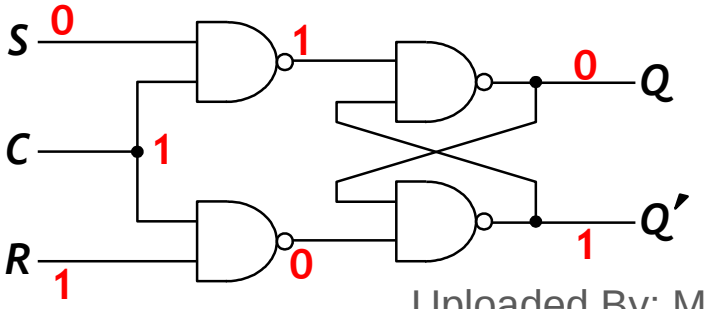
No Change



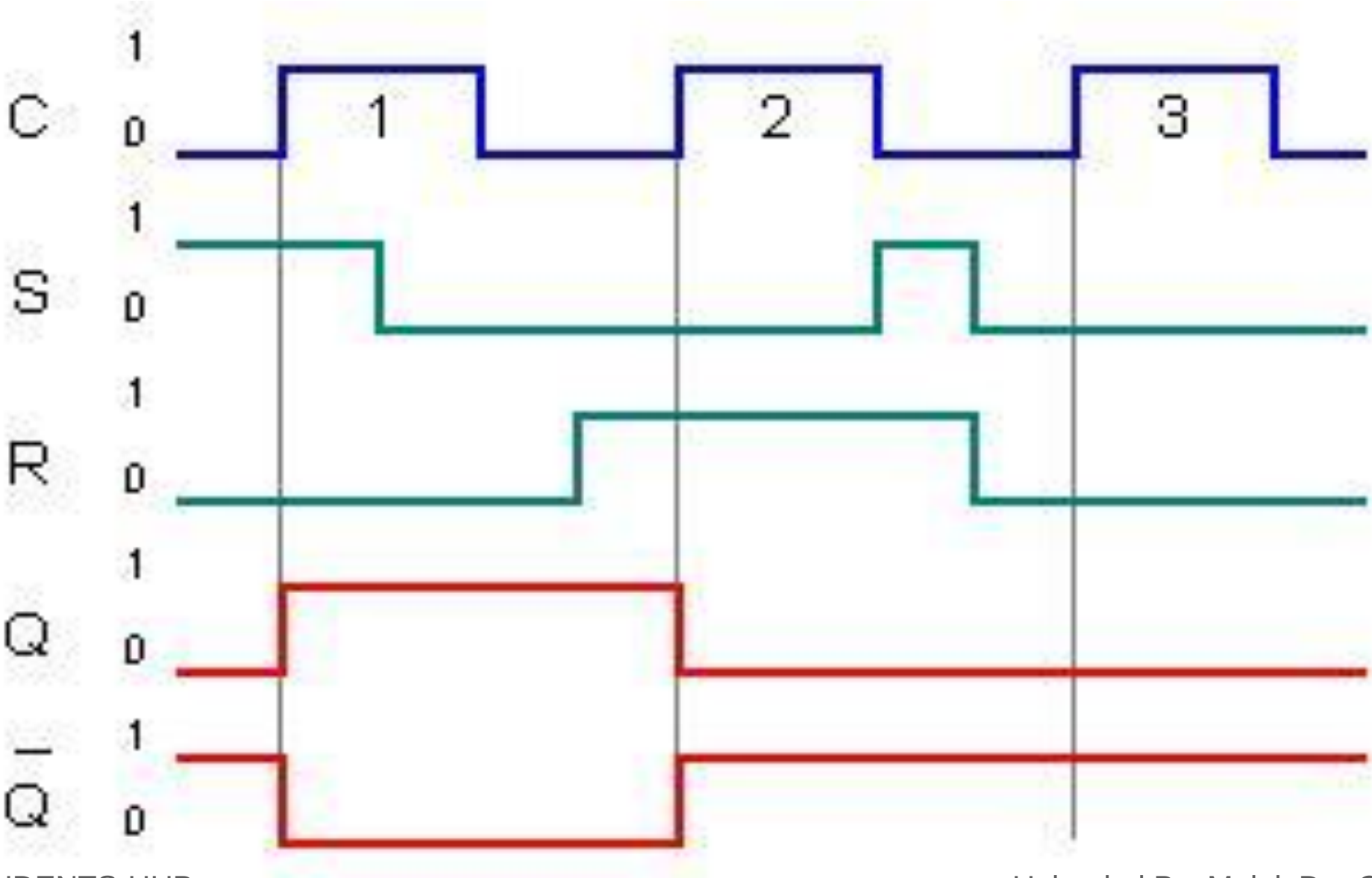
Set



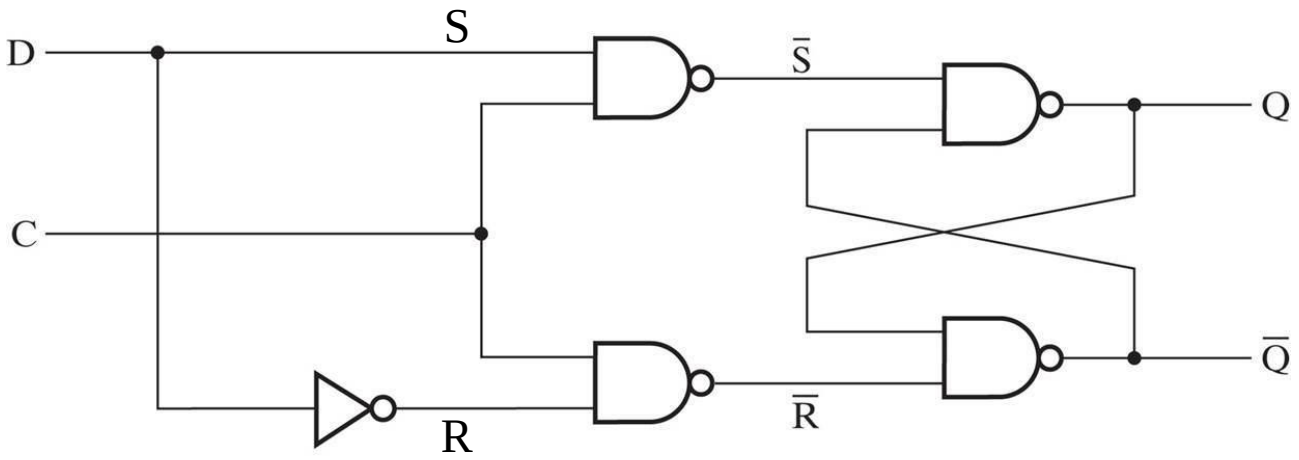
Reset



SR Latch with a Clock Input Timing Diagram



D-Latch with a Clock Input



(a) Logic diagram

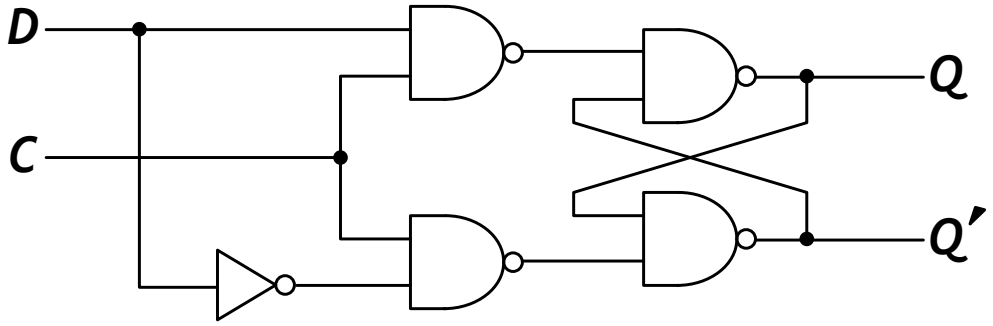
C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

(b) Function table

- **Elimination the undesirable condition of the indeterminate state in SR latch**
  - Only one data input D
  - An inverter is added: $S = D$ and $R = \overline{D}$
  - S and R can never be 11 simultaneously ➔ No undefined state
  - When $C = 0$, Q remains the same (No change in state)
  - When $C = 1$, $Q = D$ and $\overline{Q} = \overline{D}$

D-Latch with a Clock Input

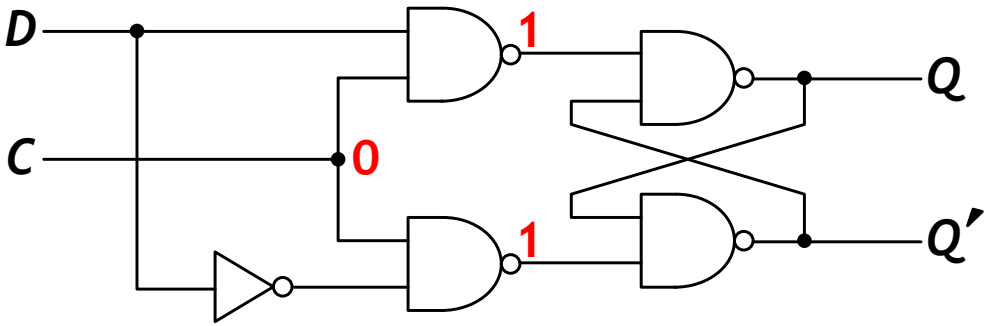
Logic Diagram



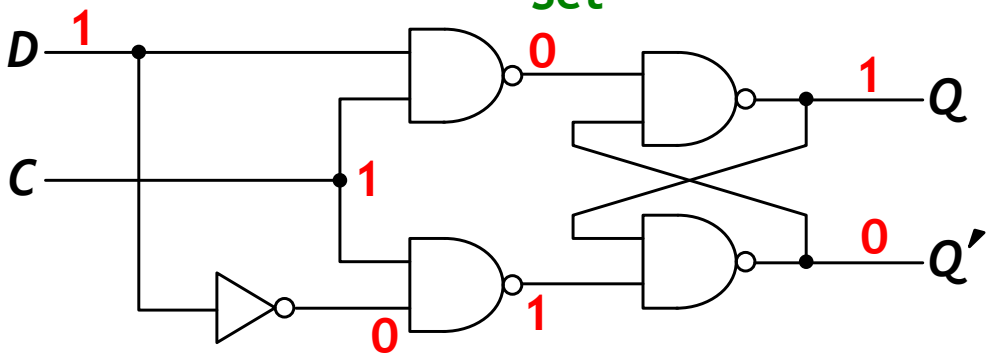
Function Table

C	D	Next State
0	X	No Change
1	0	Q=0; Reset
1	1	Q=1; Set

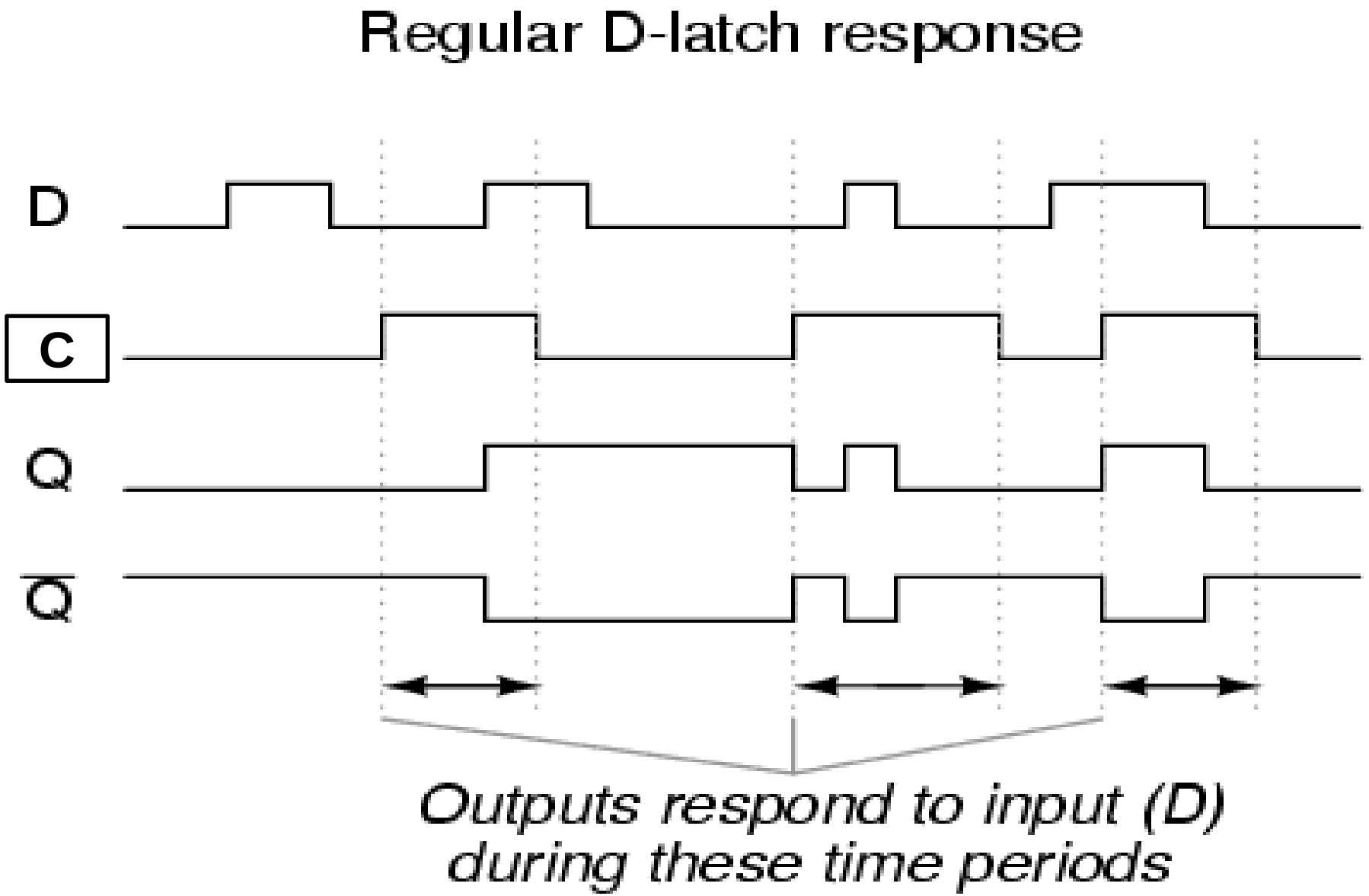
No Change



Set

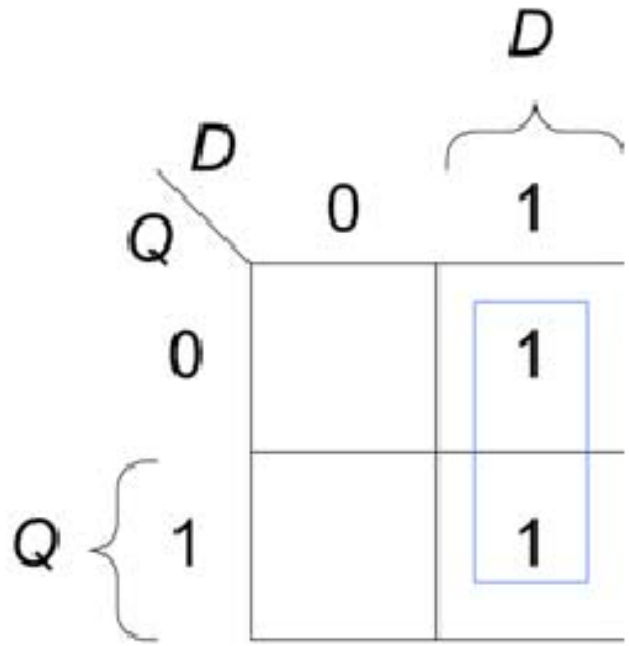


D-Latch with a Clock Input Timing Diagram



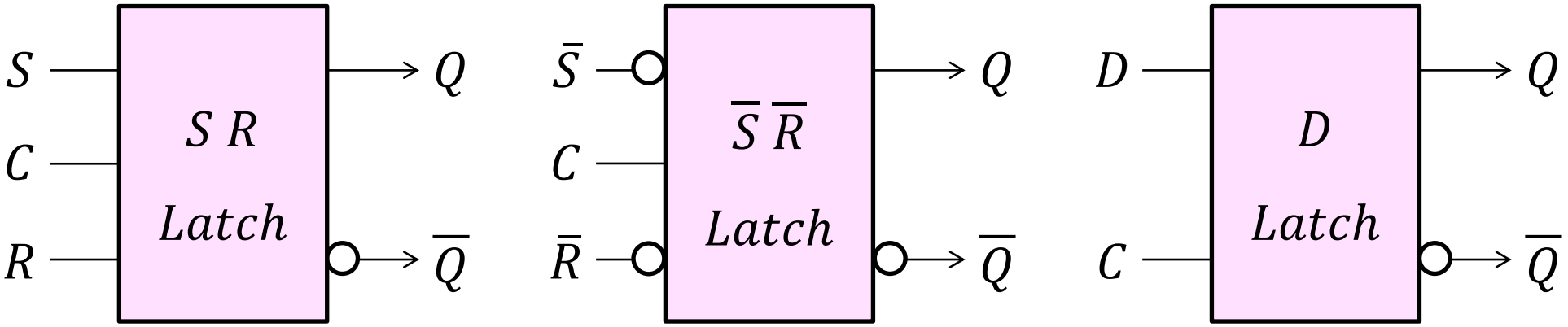
Characteristic Equation of the D-Latch

Q(t)	D	Q(t + 1)
0	0	0
0	1	1
1	0	0
1	1	1



$Q(t+1) = D$

Graphic Symbols for Latches



- A bubble appears at the complemented output $\overline{Q}$

  Indicates that $\overline{Q}$ is the complement of Q

- A bubble also appears at the inputs of an $\overline{S}\,\overline{R}$ latch

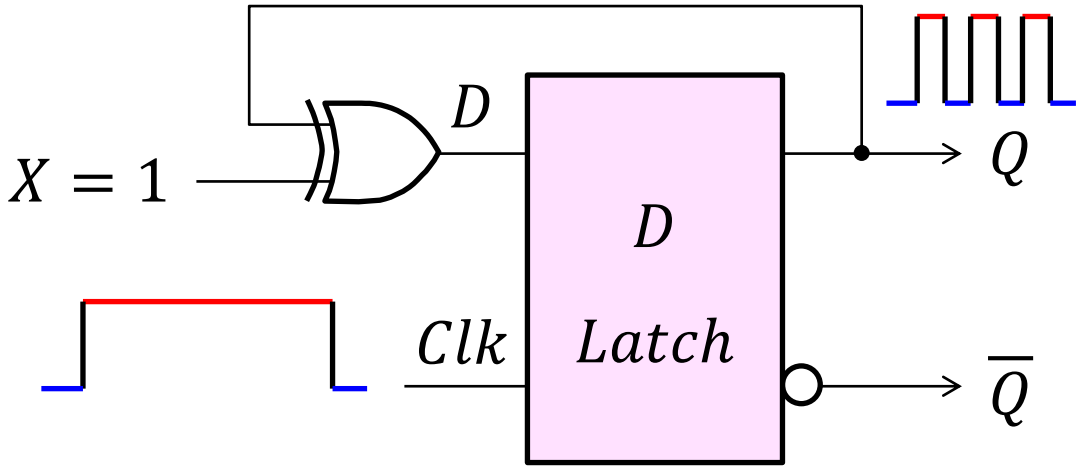
  Indicates that **logic-0** is used (not logic-1) to set (or reset) the latch (as in the NAND latch implementation)

Problem with Latches

- A latch is **level-sensitive** (sensitive to the level of the clock)
- As long as the clock signal is **high** …

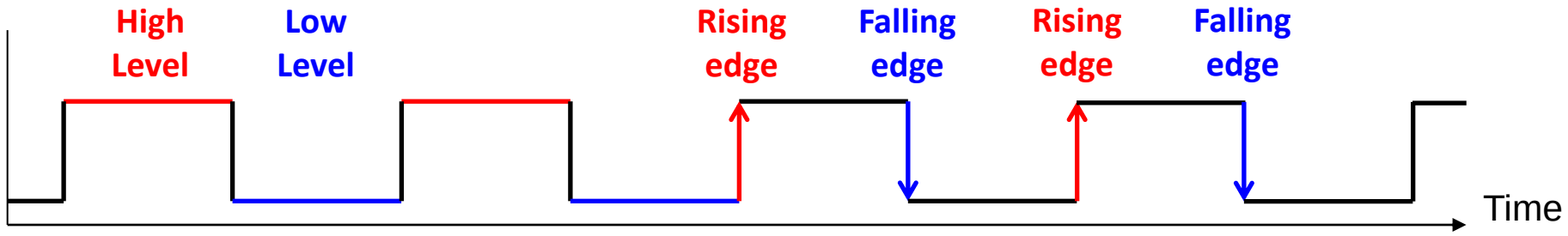
  Any change in the value of input D appears in the output Q
- Output Q keeps changing its value during a clock cycle
- Final value of output Q is uncertain

Due to this uncertainty, latches are NOT used as memory elements in synchronous circuits



Flip-Flops

- A **Flip-Flop** is a better memory element for synchronous circuits
- Solves the problem of latches in synchronous sequential circuits
- A **latch** is sensitive to the **level** of the clock
- However, a **flip-flop** is sensitive to the **edge** of the clock
- A flip-flop is called an **edge-triggered** memory element
- It changes it output value at the **edge** of the clock

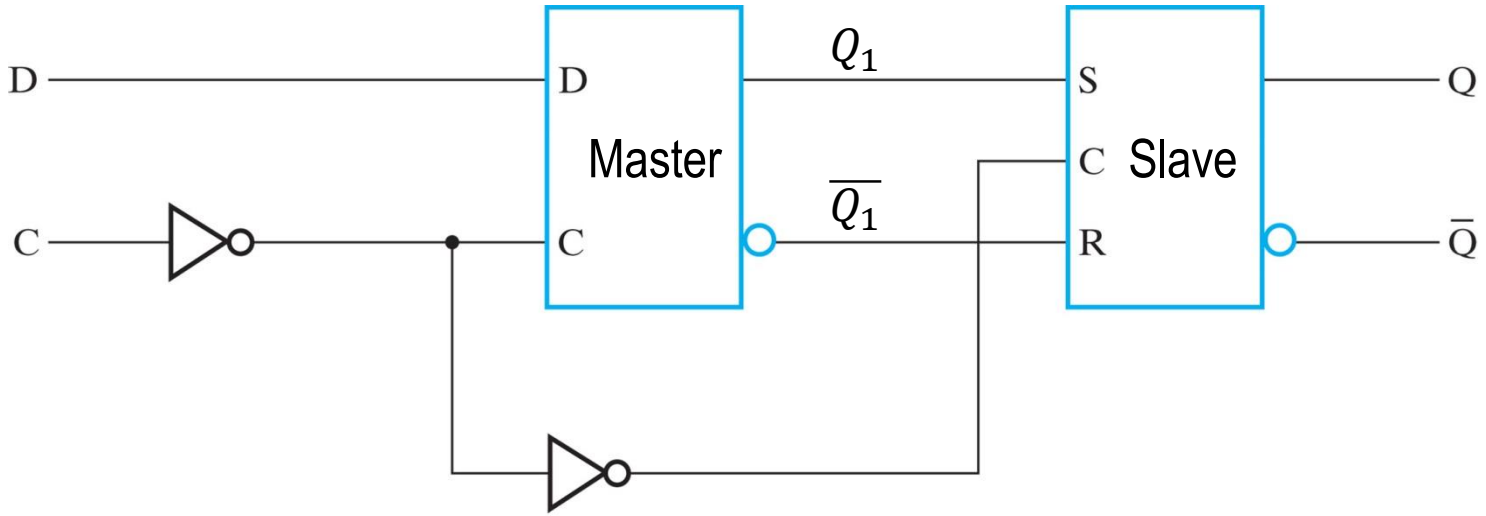


Positive Edge-Triggered D Flip-Flop

- Built using two latches in a **master-slave** configuration
- A master latch (D-type) receives external inputs
- A slave latch (SR-type) receives inputs from the master latch
- Only one latch is enabled at any given time

When **C=0**, the master is enabled and the D input is latched (slave disabled)

When **C=1**, the slave is enabled to generate the outputs (master is disabled)



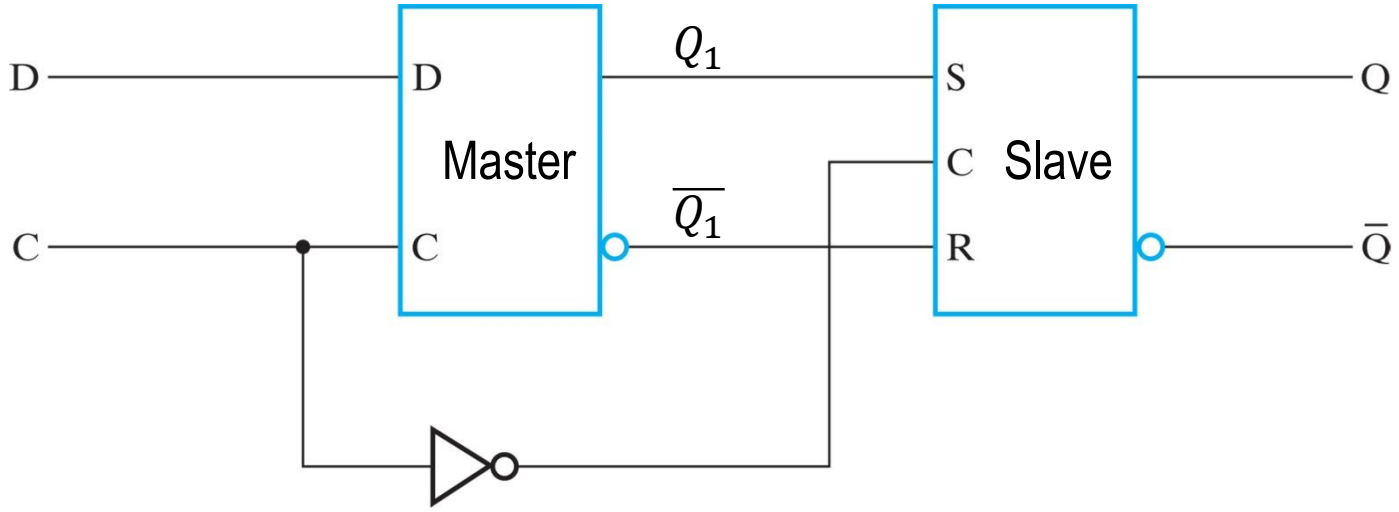
Outputs change when C changes **from 0 to 1**

Negative Edge-Triggered D Flip-Flop

- Similar to positive edge-triggered flip-flop
- The first inverter at the Master C input is removed
- Only one latch is enabled at any given time

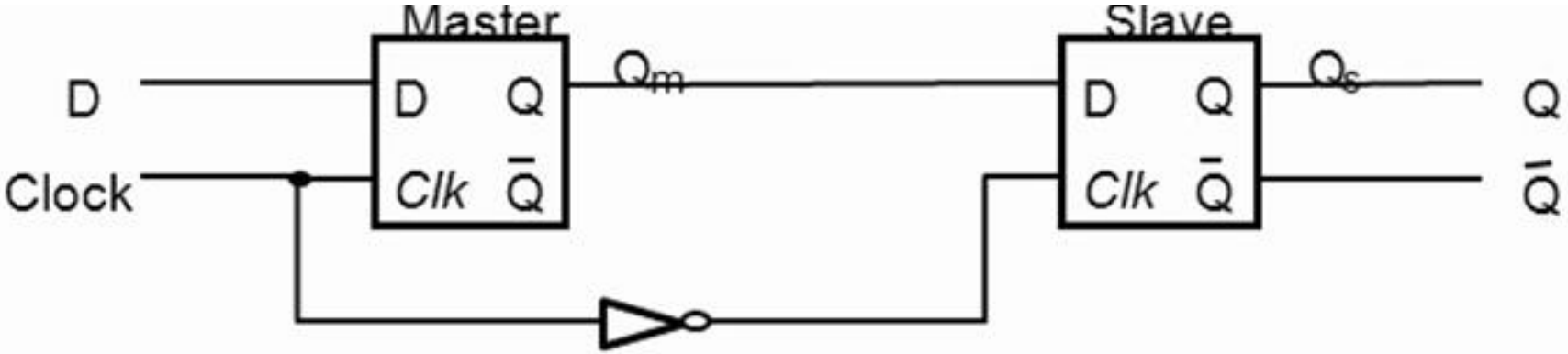
  When **C=1**, the master is enabled and the D input is latched (slave disabled)

  When **C=0**, the slave is enabled to generate the outputs (master is disabled)



Outputs change when C changes **from 1 to 0**

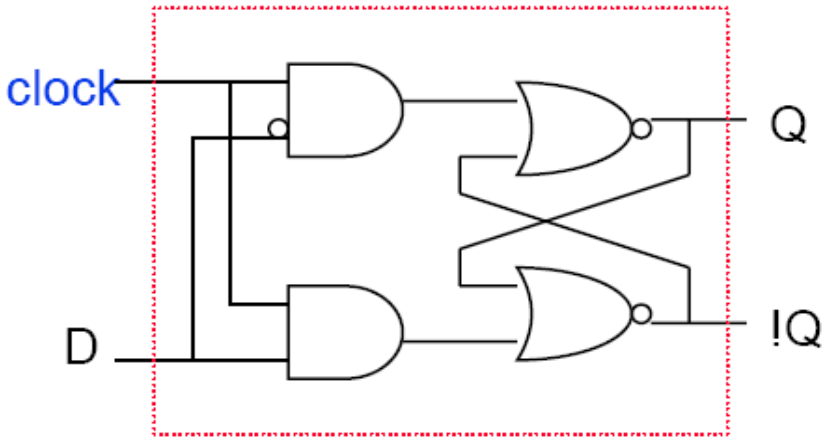
Negative-Edge D Flip-Flop Timing Diagram

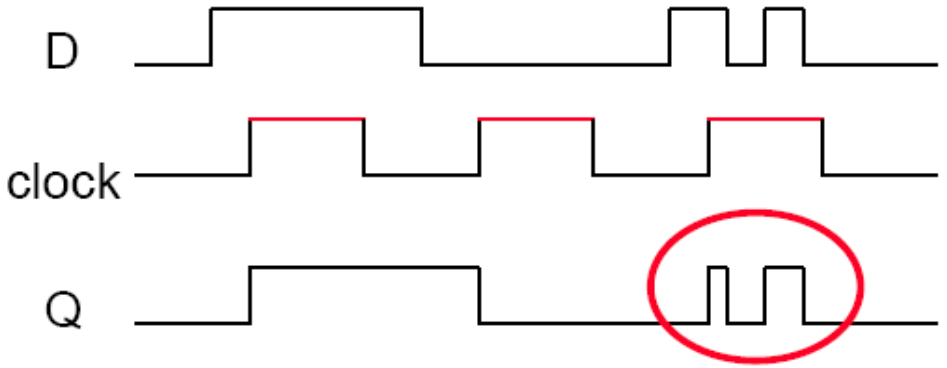


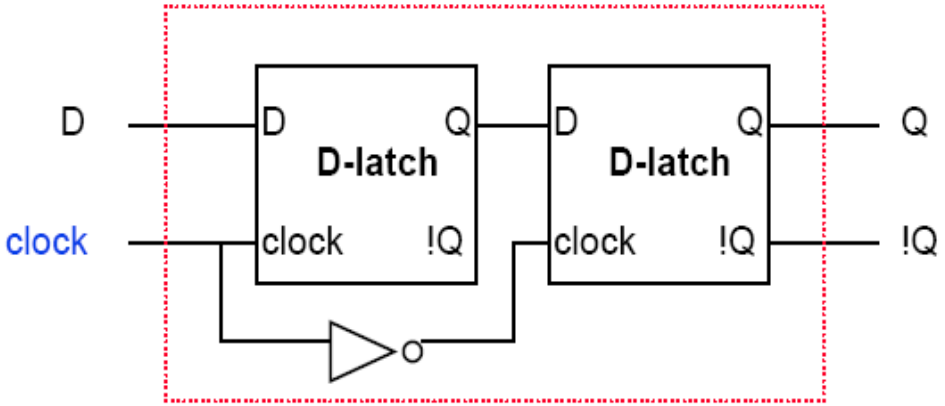
(a) Circuit

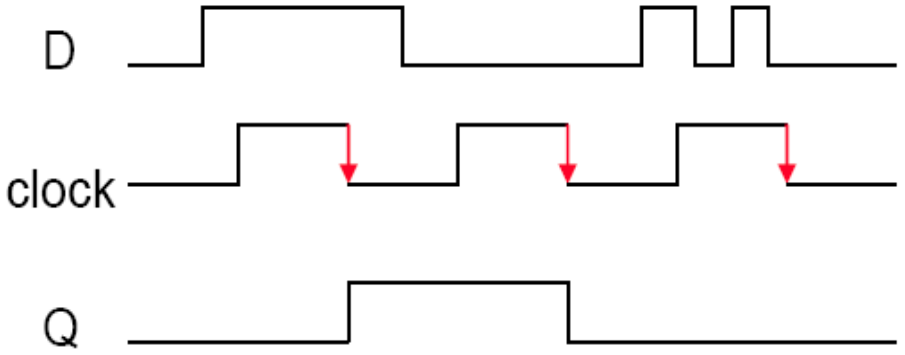
(b) Timing diagram

D-Latch vr. Edge-Triggered D Flip-Flop

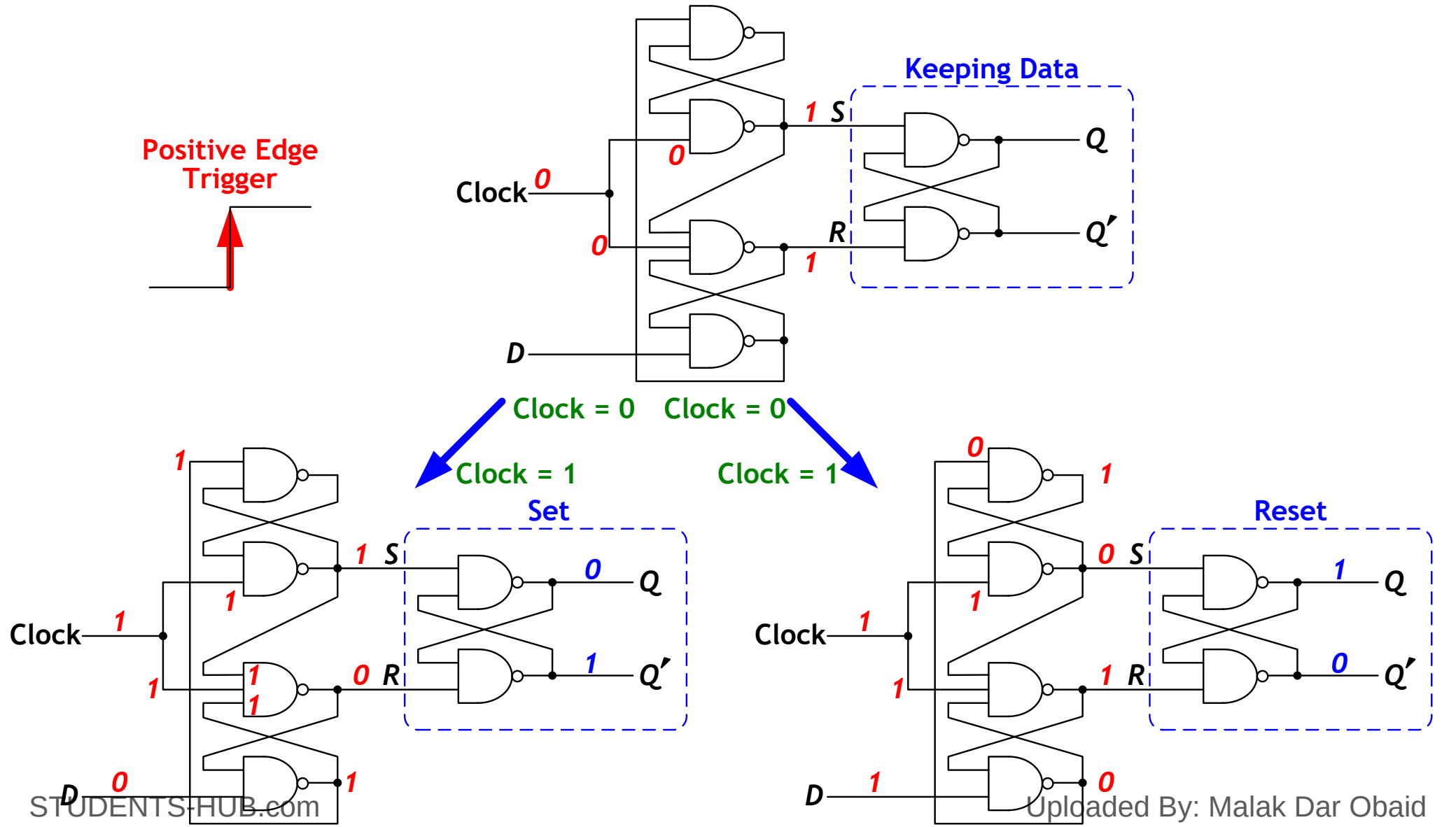




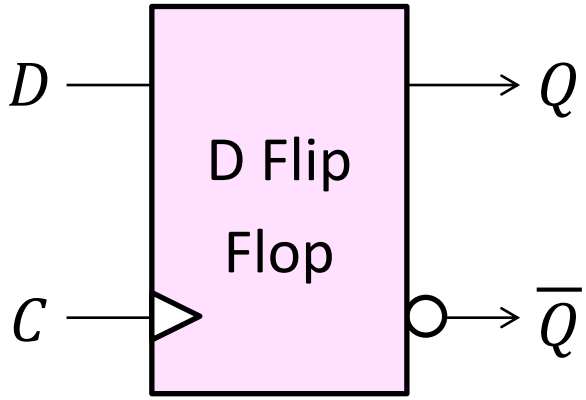


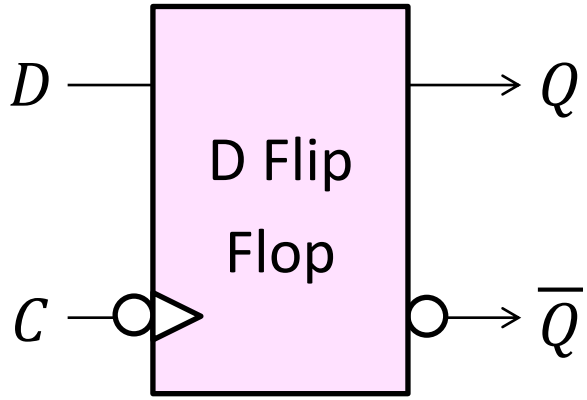


Positive Edge-Triggered D Flip-Flop Another Construction



Graphic Symbols for Flip-Flops

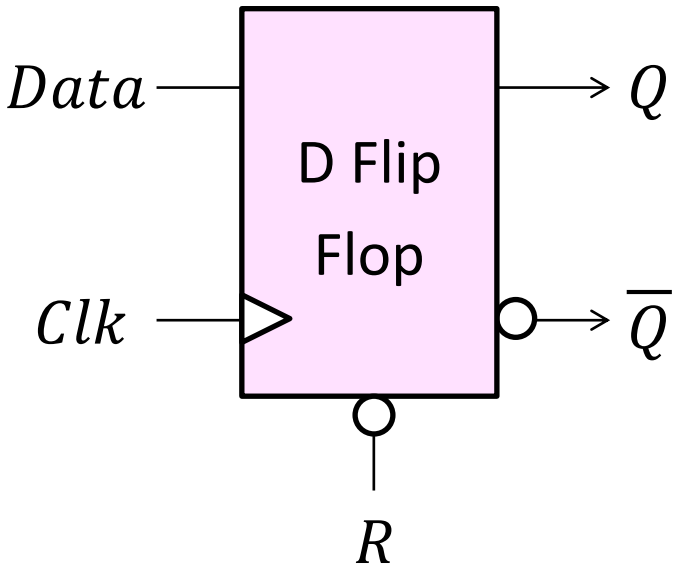




- A Flip-Flop has a similar symbol to a Latch
- The difference is the arrowhead at the clock input *C*
- The arrowhead indicates sensitivity to the edge of the clock
- A bubble at the *C* input indicates negative edge-triggered FF

D Flip-Flop with Asynchronous Reset

- When Flip-Flops are powered, their initial state is unknown
- Some flip-flops have an **Asynchronous Reset** input *R*
- Resets the state (to logic value **0**), independent of the clock
- This is required to initialize a circuit before operation
- If the *R* input is inverted (bubble) then *R* = 0 resets the flip-flop

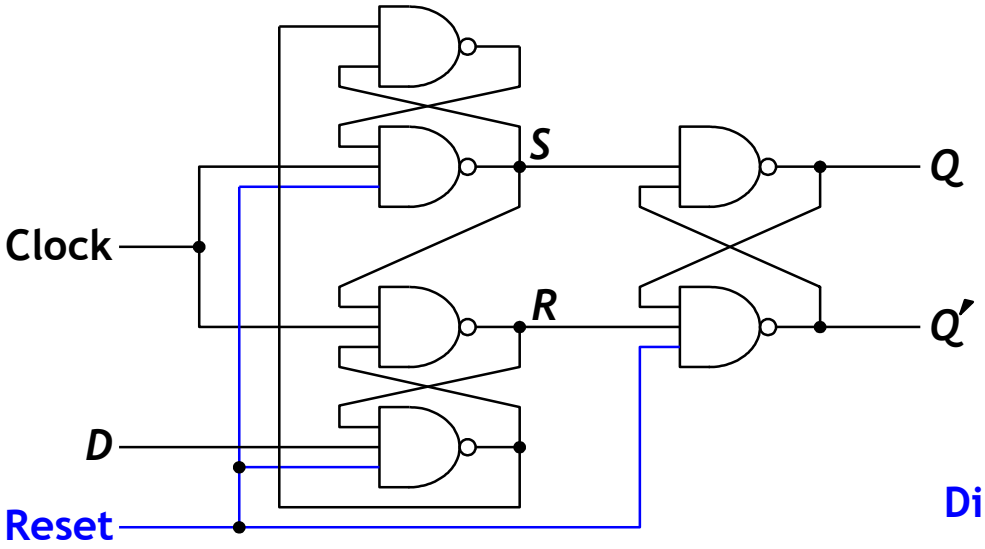


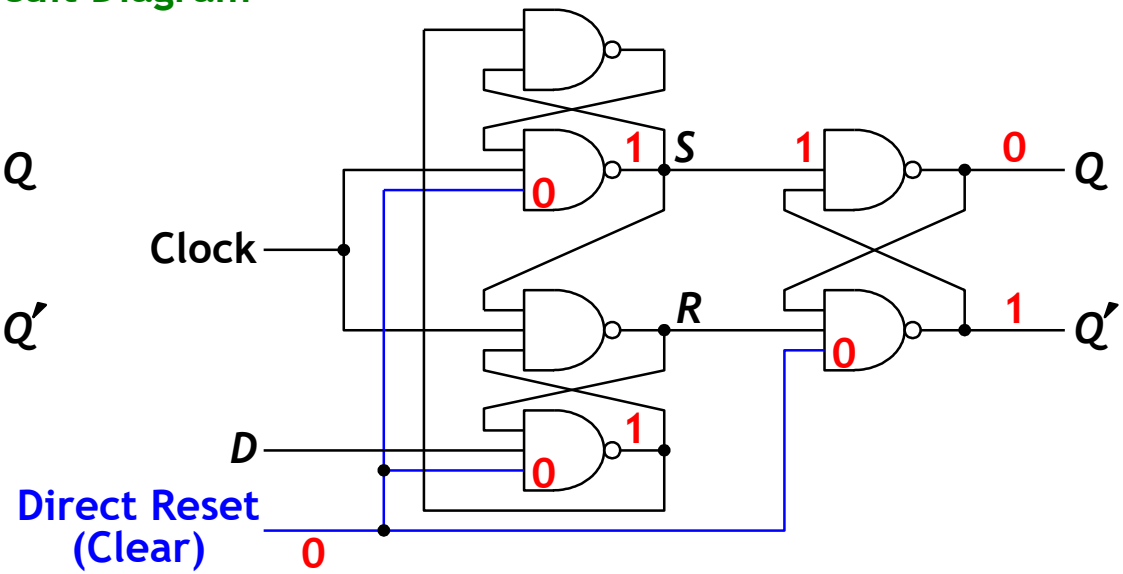
Inputs			Outputs	
R	$Data$	Clk	Q	$\overline{Q}$
0	X	X	0	1
1	0	↑	0	1
1	1	↑	1	0

Function Table

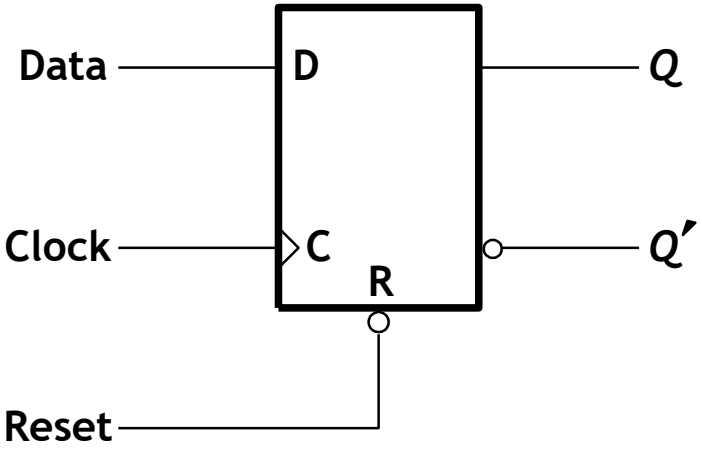
D Flip-Flop with Asynchronous Reset

Circuit Diagram





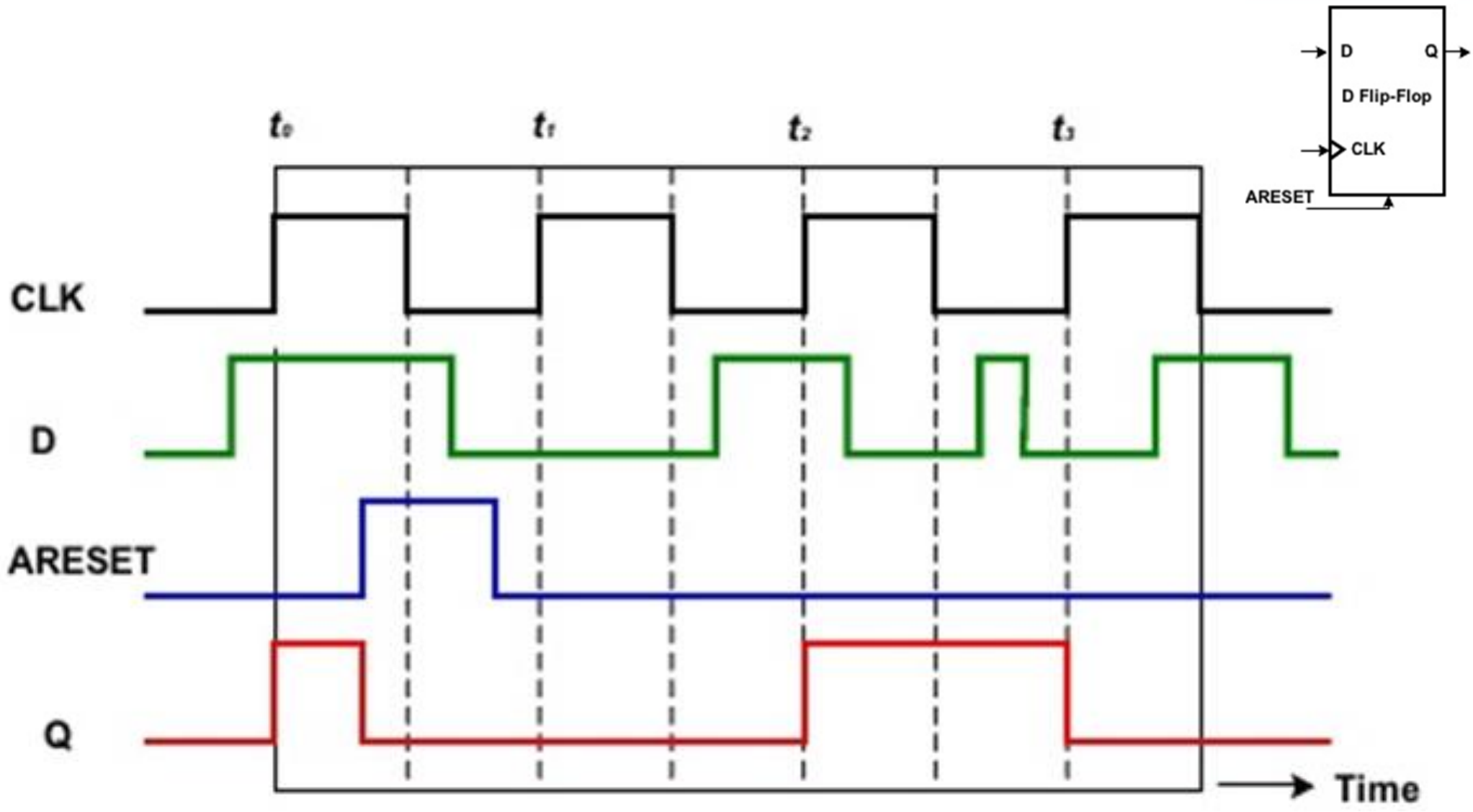
Graphic Symbol



Function Table

R	C	D	Q	Q′
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

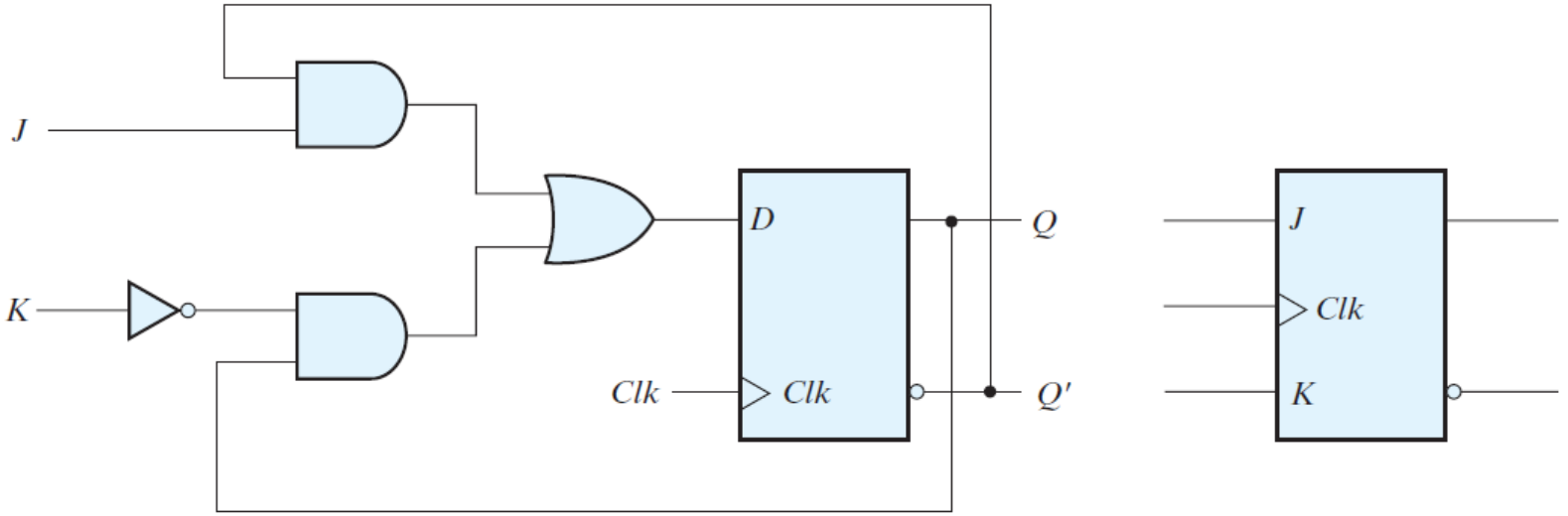
D Flip-Flop with Asynchronous Reset



JK Flip-Flop

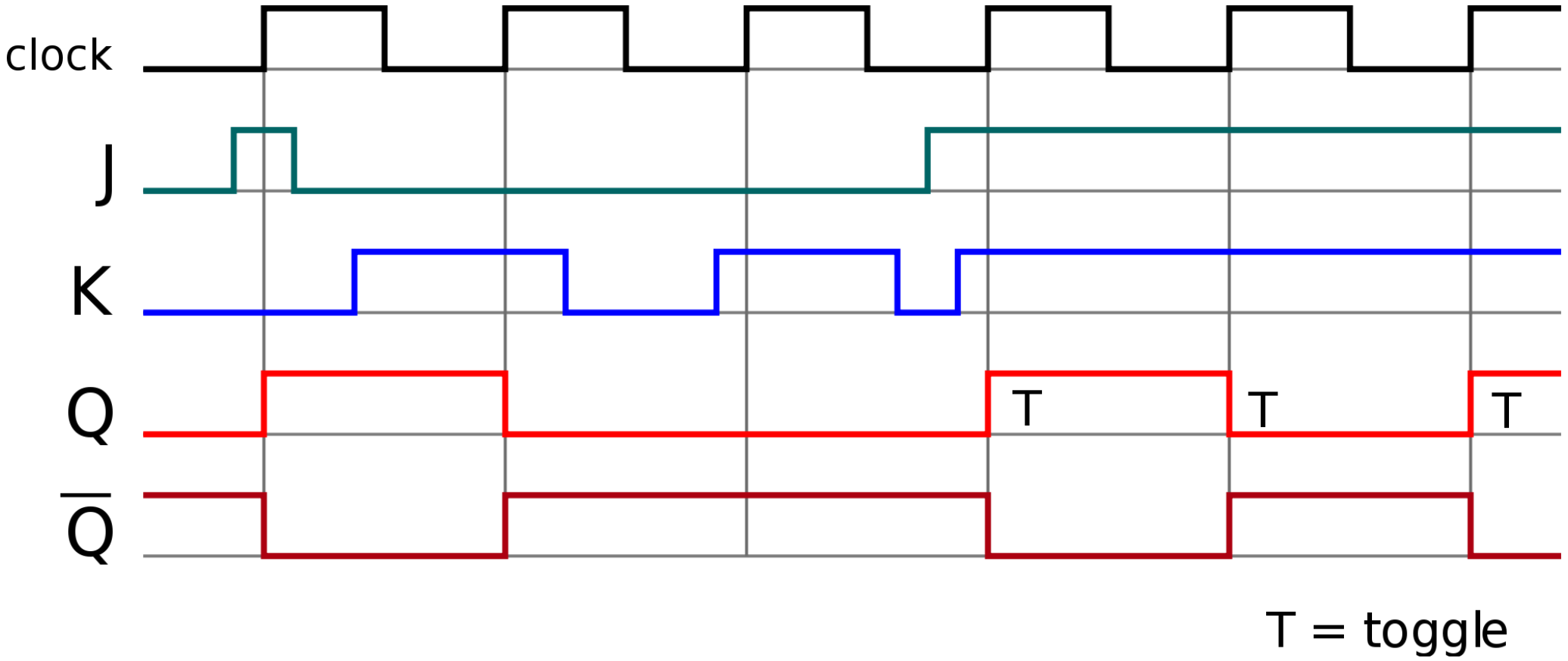
- The D Flip-Flop is the most commonly used type
- The JK is another type of Flip-Flop with inputs: J, K, and Clk
- When JK = 10 ➔ Set, When JK = 01 ➔ Reset
- When JK = 00 ➔ No change, When JK = 11 ➔ Invert outputs
- JK can be implemented using D FF

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q}_t$



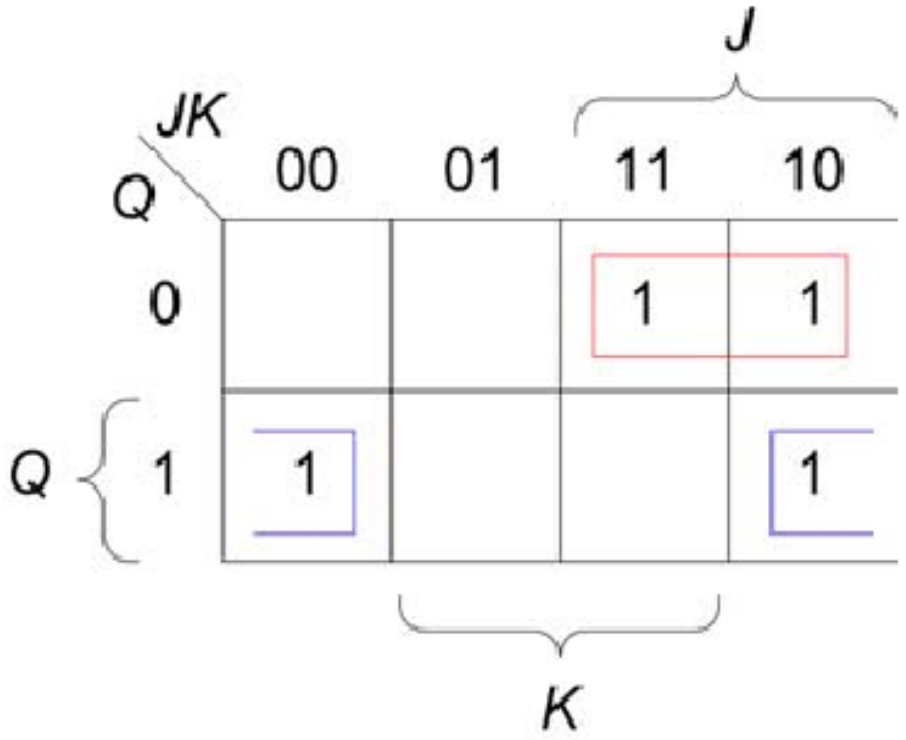
(a) Circuit diagram (b) Graphic symbol

JK Flip-Flop Timing Diagram



Characteristic Equation of the JK Flip-Flop

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

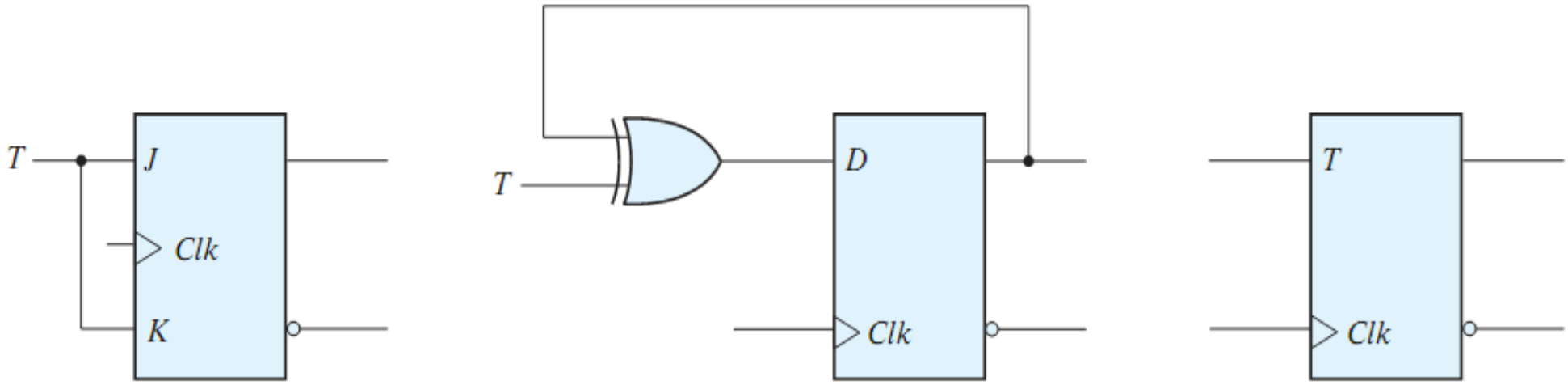


$$Q(t+1) = JQ' + K'Q$$

T Flip-Flop

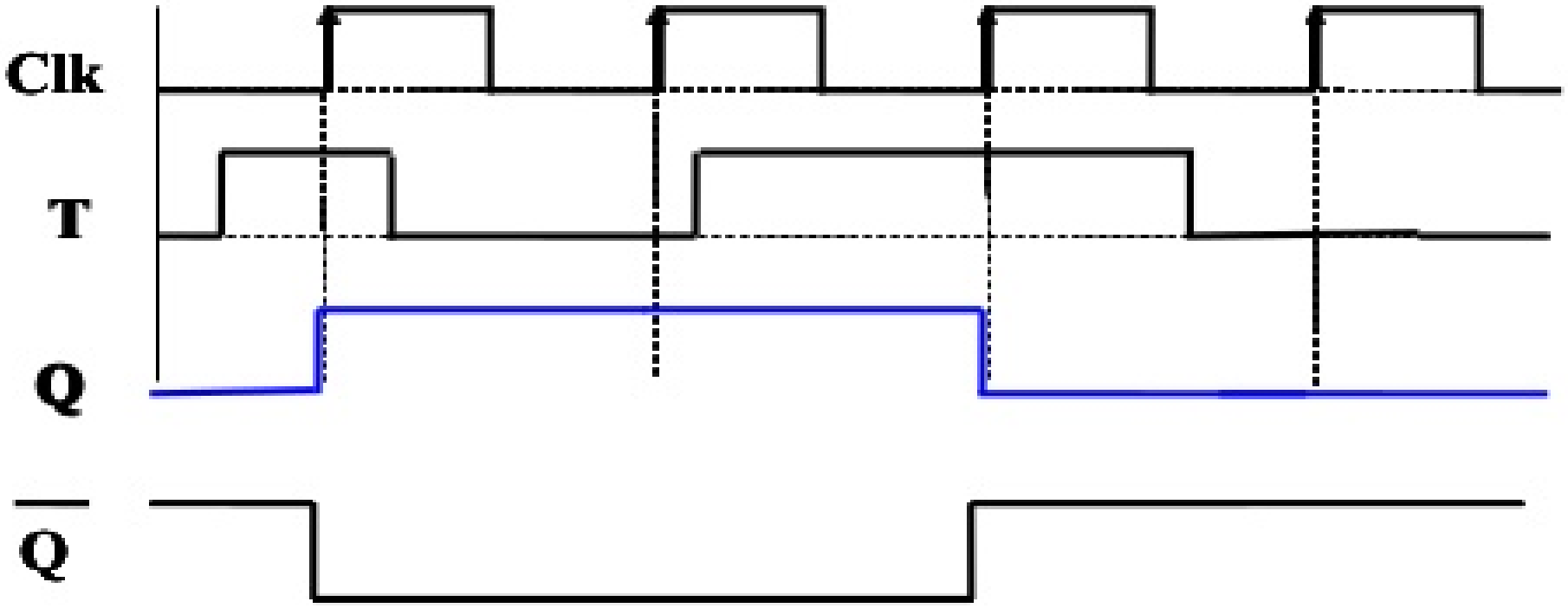
- The T (Toggle) flip-flop has inputs: T and Clk
- When T = 0 ➔ No change, When T = 1 ➔ Invert outputs
- The T flip-flop can be implemented using a JK flip-flop
- It can also be implemented using a D flip-flop and a XOR gate

T	Q_{t+1}
0	Q_t
1	$\overline{Q}_t$



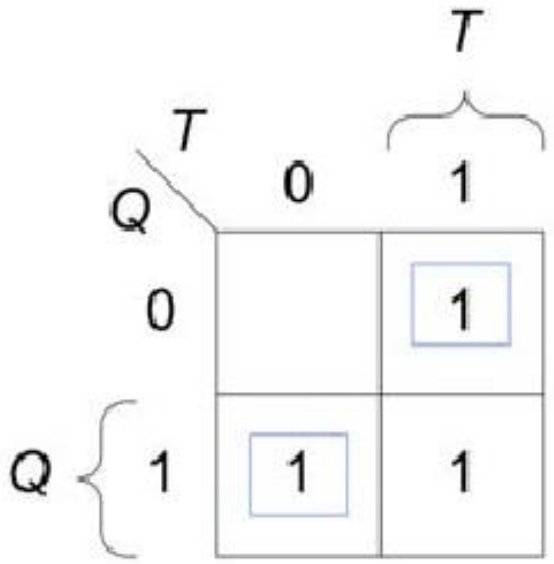
(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

T Flip-Flop Timing Diagram



Characteristic Equation of the T- Flip Flop

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



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

Flip-Flop Characteristic Table

- Defines the operation of a flip-flop in a tabular form
- Next state is defined in terms of the current state and the inputs

  $Q(t)$ refers to current state **before** the clock edge arrives

  $Q(t+1)$ refers to next state **after** the clock edge arrives

D Flip-Flop	
D	$Q(t+1)$
0	0 Reset
1	1 Set

JK Flip-Flop	
J K	$Q(t+1)$
0 0	Q(t) No change
0 1	0 Reset
1 0	1 Set
1 1	Q'(t) Complement

T Flip-Flop	
T	$Q(t+1)$
0	Q(t) No change
1	Q'(t) Complement

Flip-Flop Characteristic Equation

- The characteristic equation defines the operation of a flip-flop
- For D Flip-Flop: $Q(t+1) = D$
- For JK Flip-Flop: $Q(t+1) = J\,Q'(t) + K'\,Q(t)$
- For T Flip-Flop: $Q(t+1) = T \oplus Q(t)$
- Clearly, the D Flip-Flop is the simplest among the three

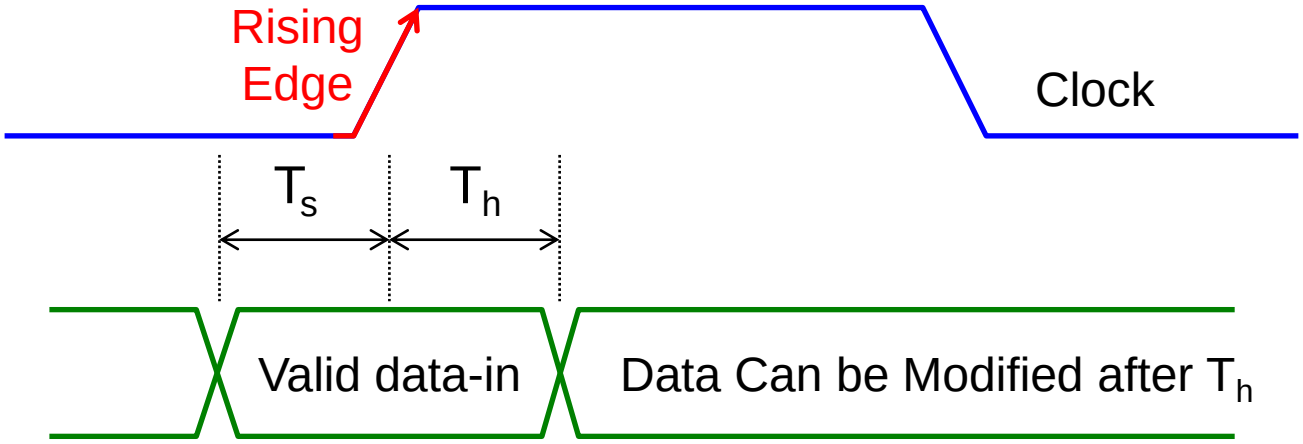
D Flip-Flop	
D	$Q(t+1)$
0	0 Reset
1	1 Set

JK Flip-Flop	
J K	$Q(t+1)$
0 0	Q(t) No change
0 1	0 Reset
1 0	1 Set
1 1	Q'(t) Complement

T Flip-Flop	
T	$Q(t+1)$
0	Q(t) No change
1	Q'(t) Complement

Timing Considerations for Flip-Flops

- **Setup Time (T_s):** Time duration for which the data input must be valid and stable before the arrival of the clock edge.
- **Hold Time (T_h):** Time duration for which the data input must not be changed after the clock transition occurs.
- T_s and T_h must be ensured for the proper operation of flip-flops



Analysis of Clocked Sequential Circuits

- Analysis is describing what a given circuit will do
- The output of a clocked sequential circuit is determined by
  1. Inputs
  2. State of the Flip-Flops

- **Analysis Procedure:**
  1. Obtain the equations at the inputs of the Flip-Flops
  2. Obtain the output equations
  3. Fill the state table for all possible input and state values
  4. Draw the state diagram

Analysis Example

- Is this a clocked sequential circuit?

  YES!

- What type of Memory?

  D Flip-Flops

- How many state variables?

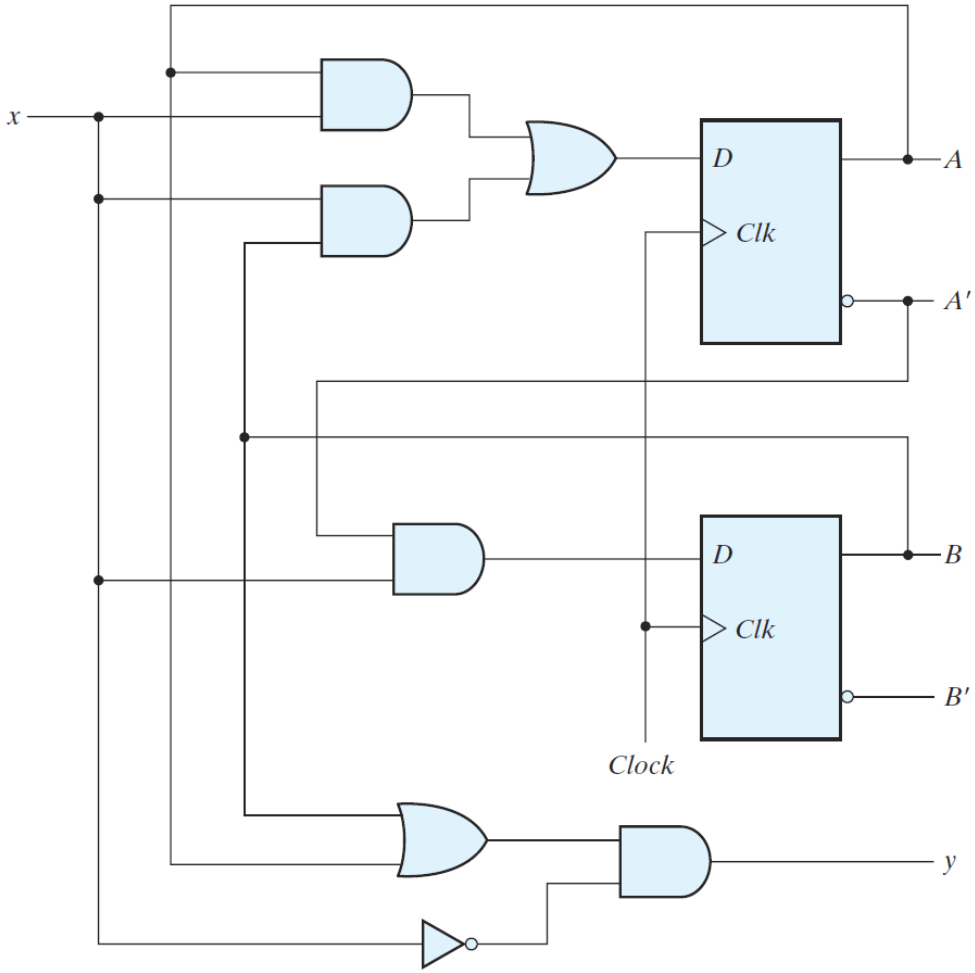
  Two state variables: A and B

- What are the Inputs?

  One Input: x

- What are the Outputs?

  One Output: y



Flip-Flop Input Equations

- What are the equations on the D inputs of the flip-flops?

$$D_A = A\,x + B\,x$$

$$D_B = A'\,x$$

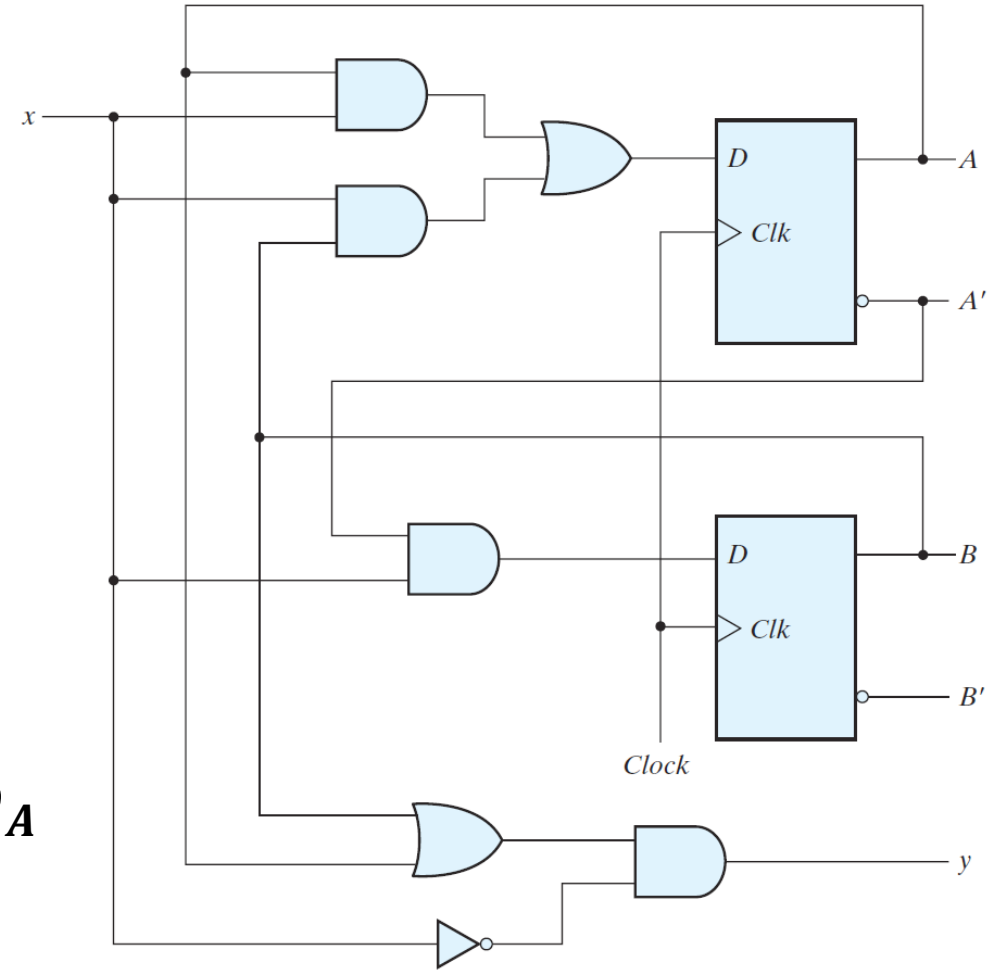
- A and B are the **current state**

$$A(t) = A, \quad B(t) = B$$

- D_A and D_B are the **next state**

$$A(t+1) = D_A, \quad B(t+1) = D_B$$

- The values of A and B will be D_A and D_B at the next clock edge



Next State and Output Equations

- The next state equations define the **next state**

  At the **inputs** of the Flip-Flops

- Next state equations?

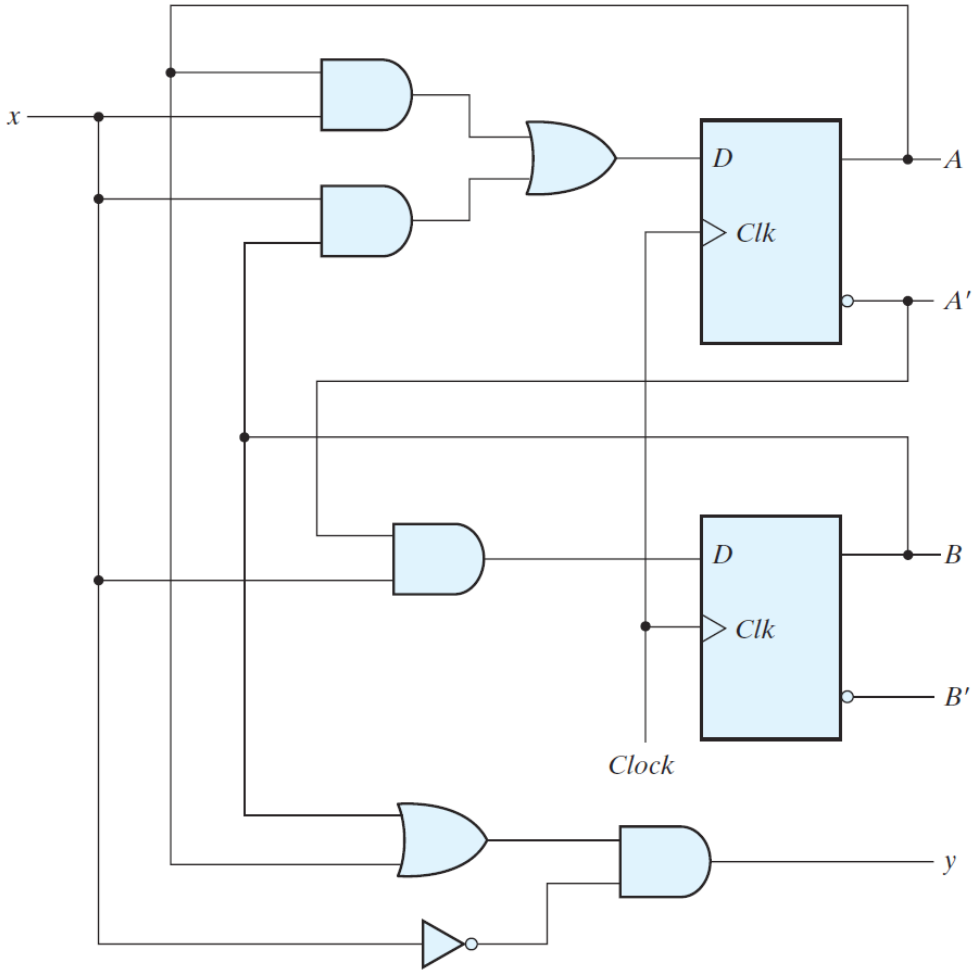
$$A(t+1) = D_A = A\,x + B\,x$$

$$B(t+1) = D_B = A'\,x$$

- There is only one output y

- What is the output equation?

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



State Table

- State table shows the Next State and Output in a tabular form
- Next State Equations: $A(t+1) = A\,x + B\,x$ and $B(t+1) = A'\,x$
- Output Equation: $y = (A + B)\,x'$

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

Another form of the state table

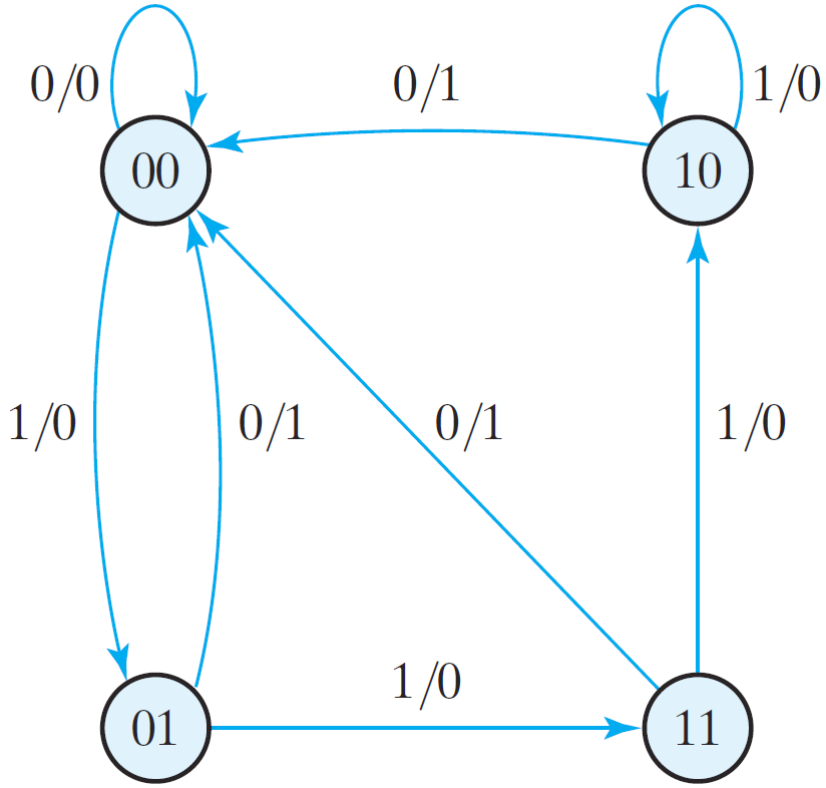
Present State		Next State				Output	
		x = 0		x = 1		x = 0	x = 1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

State Diagram

- State diagram is a graphical representation of a state table
- The circles are the states
- Two state variable ➔ Four states (ALL values of A and B)
- Arcs are the state transitions

  Labeled with: Input x / Output y

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



Combinational versus Sequential Analysis

Analysis of Combinational Circuits

- Obtain the Boolean Equations
- Fill the Truth Table

Output is a function of input only

Analysis of Sequential Circuits

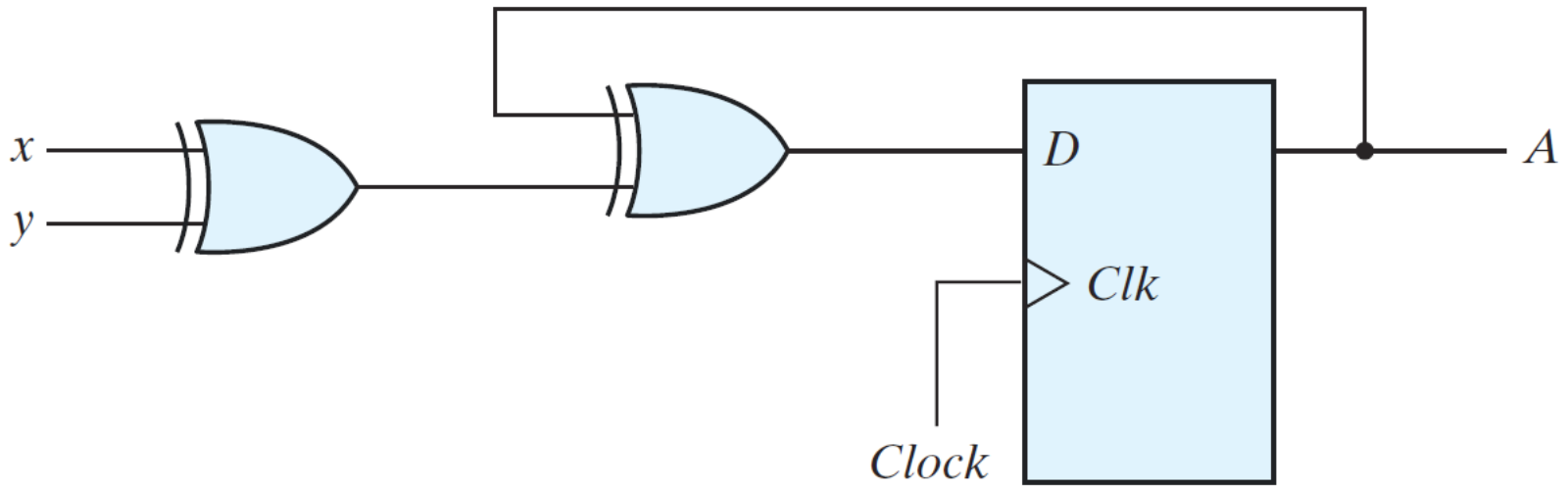
- Obtain the Next State Equations
- Obtain the Output Equations
- Fill the State Table
- Draw the State Diagram

Next state is a function of input and current state

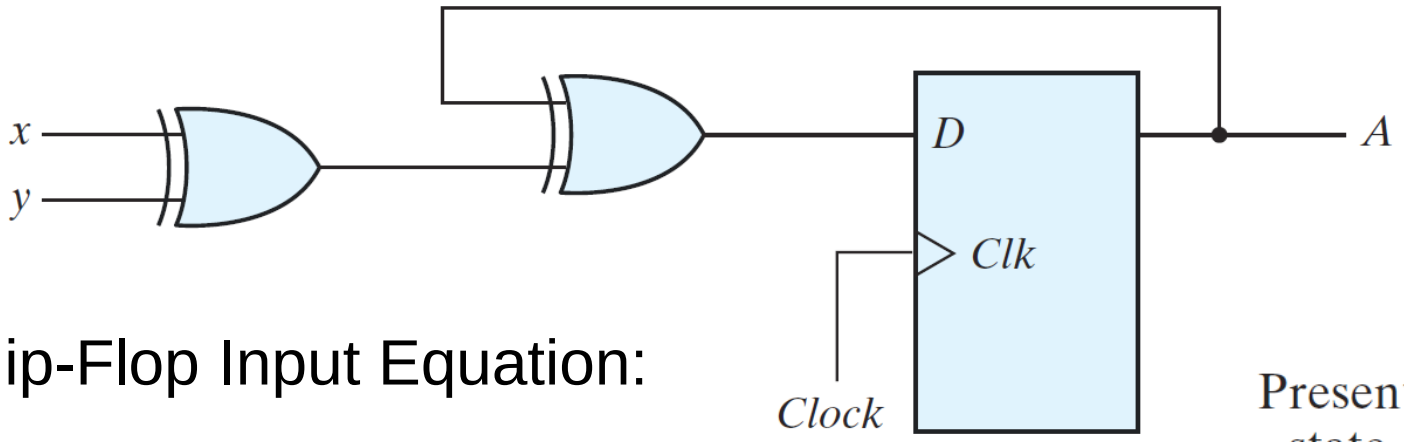
Output is a function of input and current state

Example with Output = Current State

- Analyze the sequential circuit shown below
- Two inputs: x and y
- One state variable A
- No separate output ➔ Output = current state A
- Obtain the next state equation, state table, and state diagram



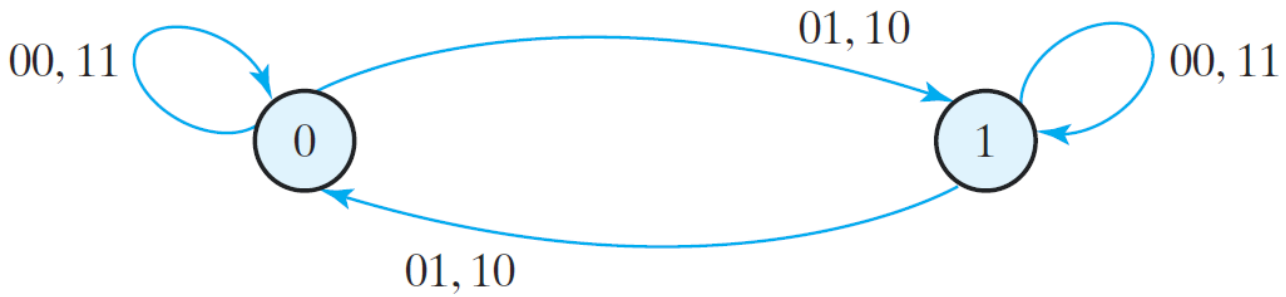
Example with Output = Current State



- Flip-Flop Input Equation:

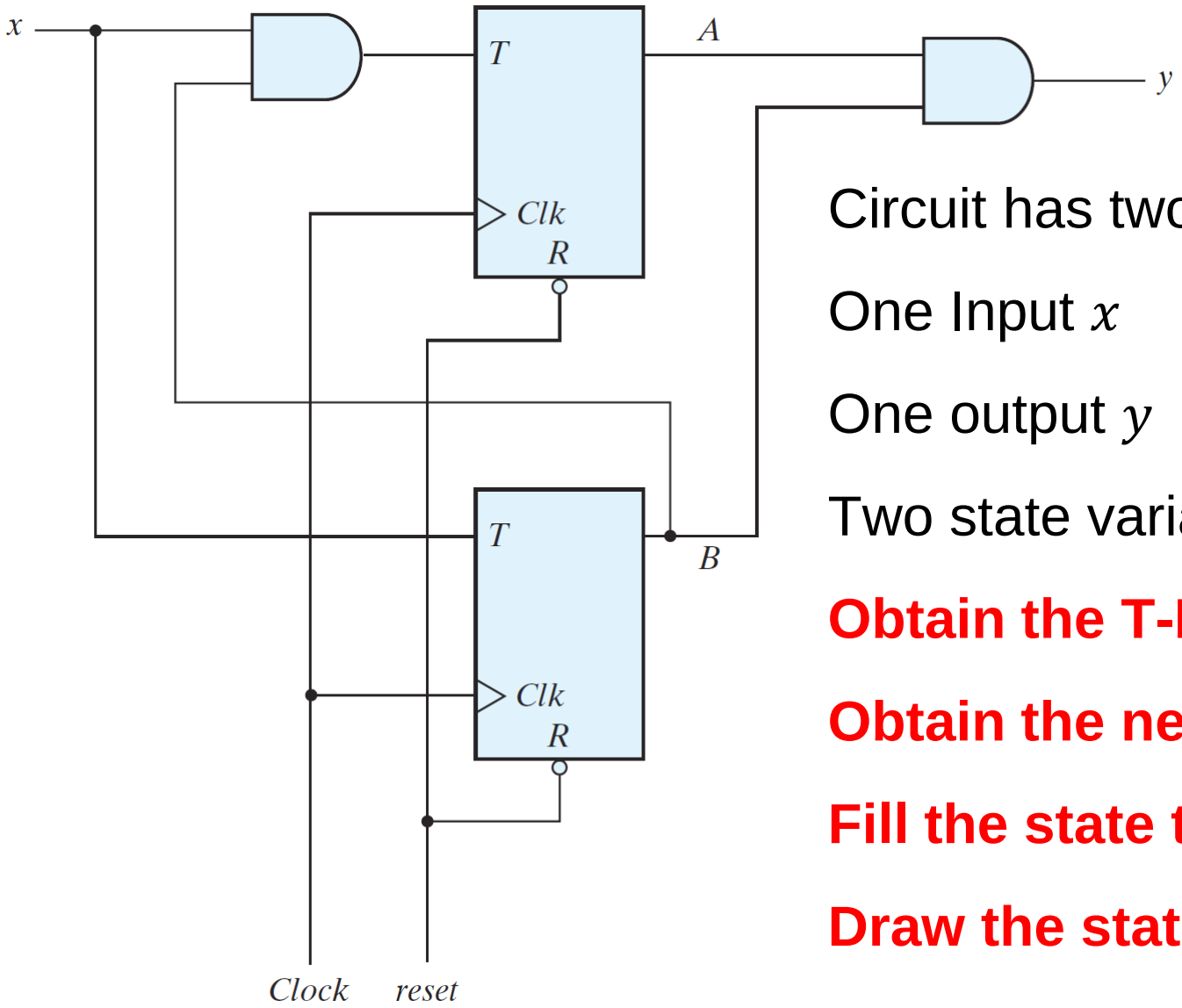
$$D_A = A \oplus x \oplus y$$

- Next State Equation: $A(t+1) = A \oplus x \oplus y$



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

Sequential Circuit with T Flip-Flops



Circuit has two T Flip-Flops

One Input x

One output y

Two state variables: A and B

Obtain the T-FF input equations

Obtain the next state equations

Fill the state table

Draw the state diagram

Recall: Flip-Flop Characteristic Equation

- For D Flip-Flop: $Q(t+1) = D$
- For T Flip-Flop: $Q(t+1) = T \oplus Q(t)$
- For JK Flip-Flop: $Q(t+1) = J\,Q'(t) + K'\,Q(t)$

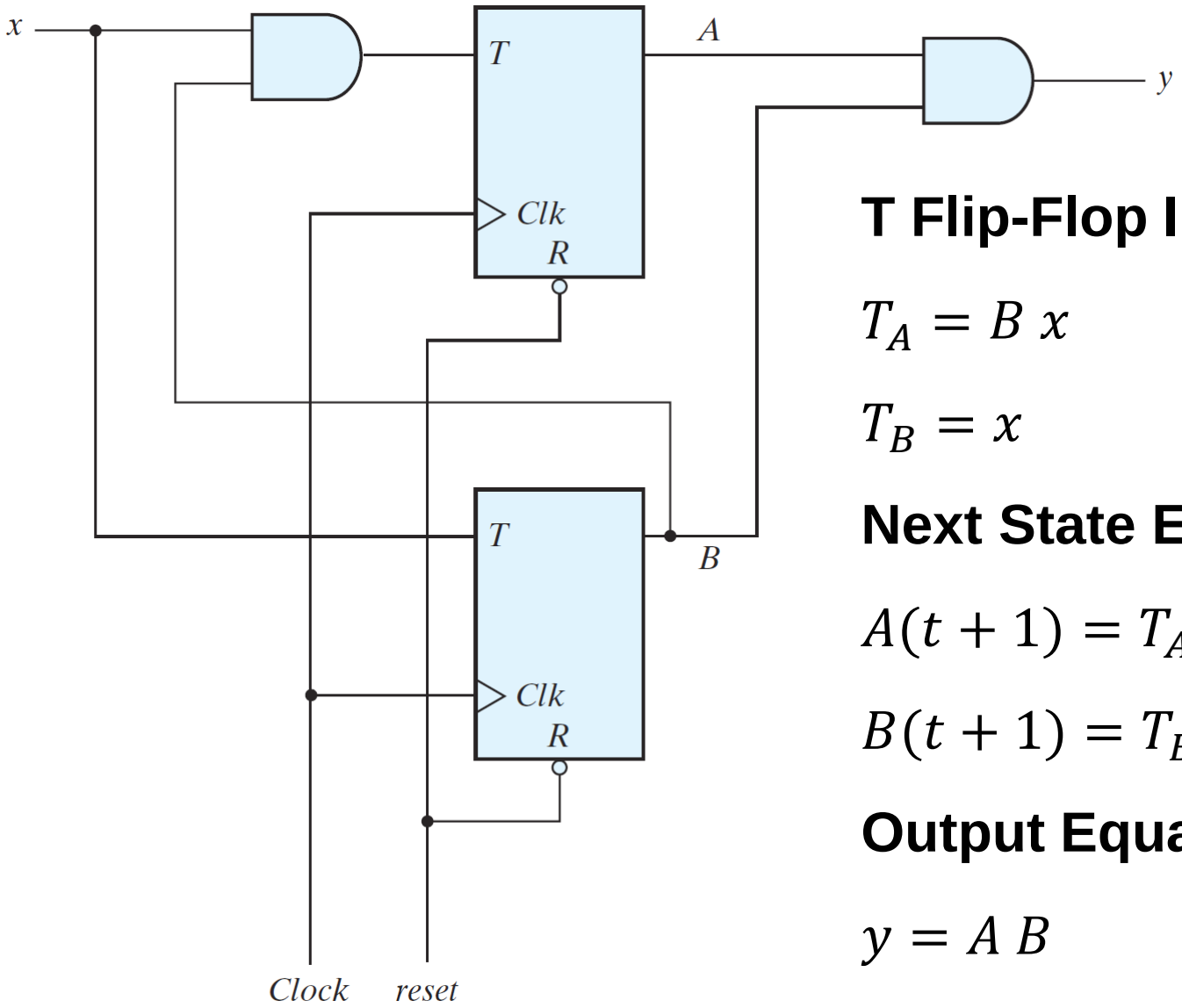
These equations define the Next State

D Flip-Flop	
D	$Q(t+1)$
0	0 Reset
1	1 Set

T Flip-Flop	
T	$Q(t+1)$
0	Q(t) No change
1	Q'(t) Complement

JK Flip-Flop	
J K	$Q(t+1)$
0 0	Q(t) No change
0 1	0 Reset
1 0	1 Set
1 1	Q'(t) Complement

Sequential Circuit with T Flip-Flops



T Flip-Flop Input Equations:

$$T_A = B\,x$$

$$T_B = x$$

Next State Equations:

$$A(t+1) = T_A \oplus A = (B\,x) \oplus A$$

$$B(t+1) = T_B \oplus B = x \oplus B$$

Output Equation:

$$y = A\,B$$

From Next State Equations to State Table

T Flip-Flop Input Equations:

$T_A = B\,x$

$T_B = x$

Next State Equations:

$A(t+1) = (B\,x) \oplus A$

$B(t+1) = x \oplus B$

Output Equation:

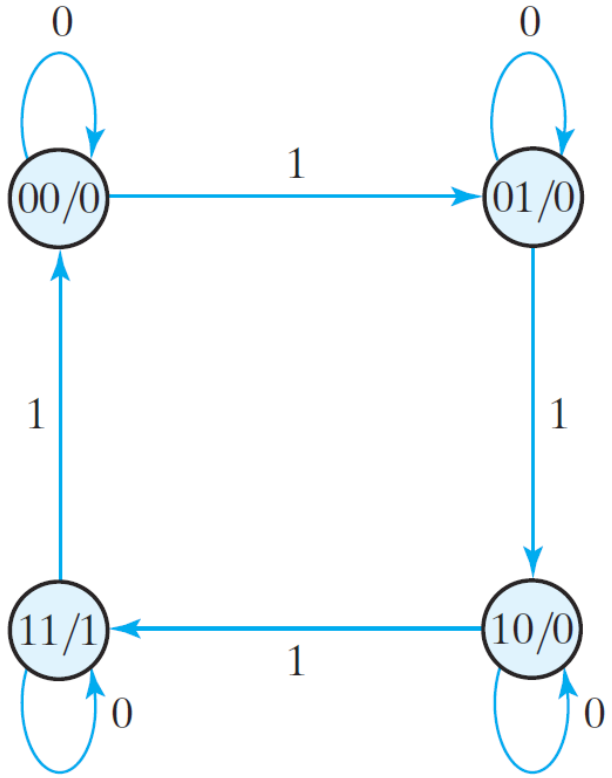
$y = A\,B$

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

Notice that the output is a function of the present state only.
It does **NOT** depend on the input x

From State Table to State Diagram

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



- Four States: $AB = 00,\ 01,\ 10,\ 11$ (drawn as circles)
- Output Equation: $y = A\,B$ (does not depend on input x)
- Output y is shown inside the state circle (AB/y)

Sequential Circuit with a JK Flip-Flops

One Input x and two state variables: A and B (outputs of Flip-Flops)

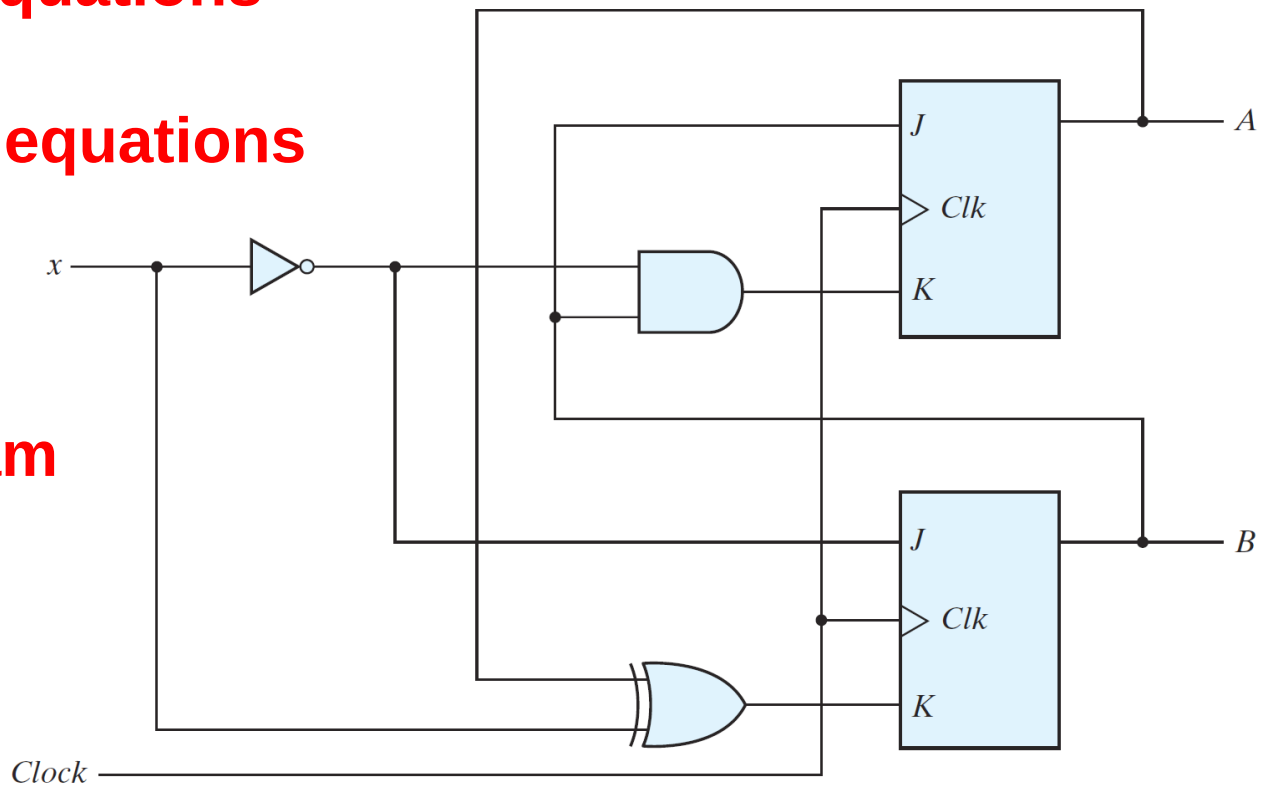
No separate output ➔ Output = Current state $A\ B$

Obtain the JK input equations

Obtain the next state equations

Fill the state table

Draw the state diagram

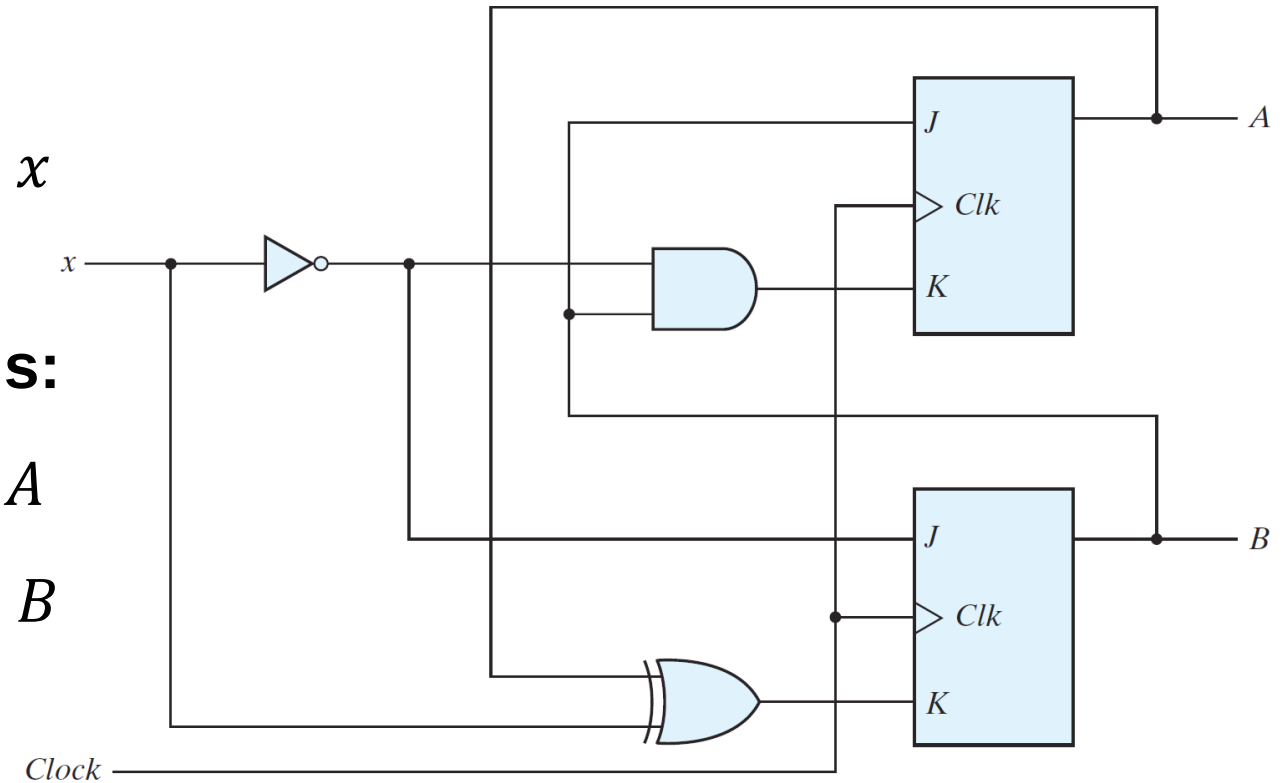


JK Input and Next State Equations

JK Flip-Flop Input Equations:

$J_A = B$ and $K_A = B\,x'$

$J_B = x'$ and $K_B = A \oplus x$



Next State Equations:

$A(t+1) = J_A\,A' + K_A'\,A$

$B(t+1) = J_B\,B' + K_B'\,B$

Substituting:

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

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

From JK Input Equations to State Table

JK Input Equations: $J_A = B$, $K_A = B\,x'$, $J_B = x'$ and $K_B = A \oplus x$

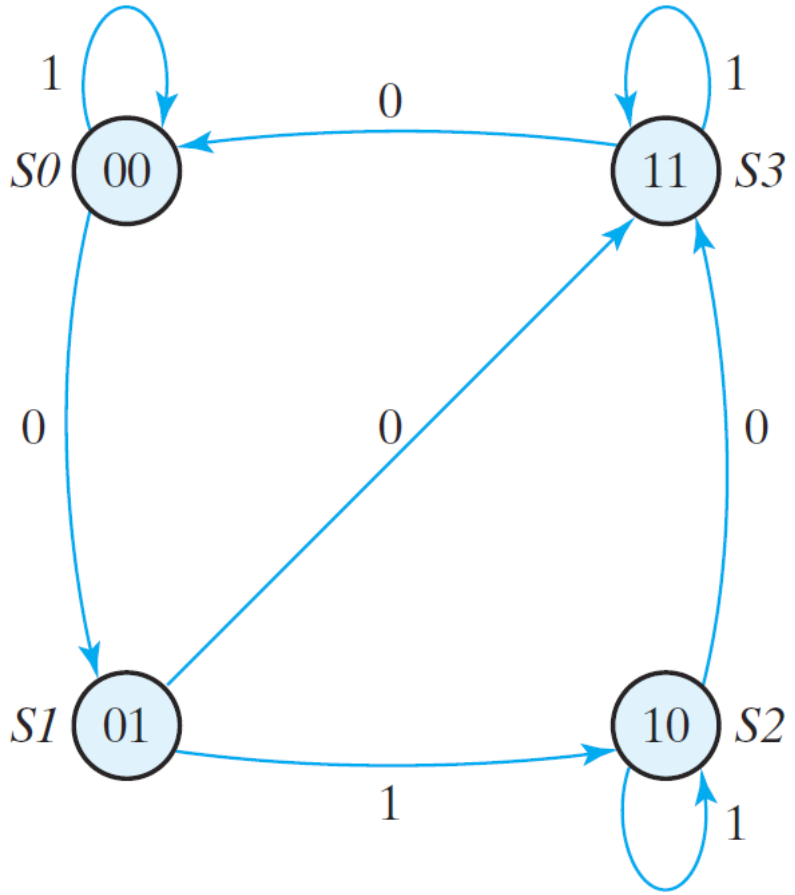
Present State		Input	Next State		Flip-Flop Inputs			
A	**B**	**x**	**A**	**B**	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

From State Table to State Diagram

Four states: $A\ B = 00, 01, 10, and\ 11$ (drawn as circles)

Arcs show the input value x on the state transition

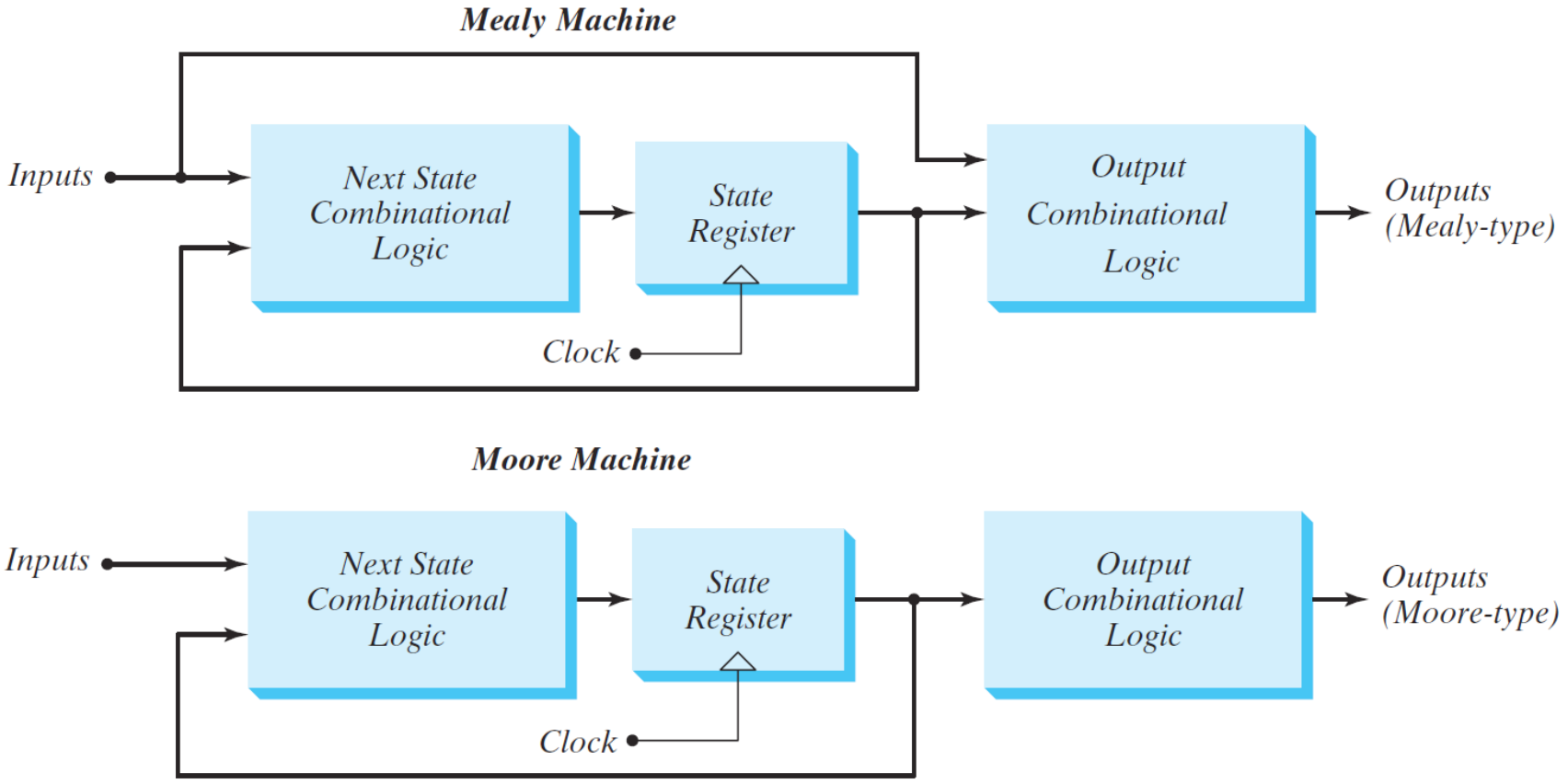
Present State		Input	Next State	
A	**B**	**x**	**A**	**B**
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1



Mealy versus Moore Sequential Circuits

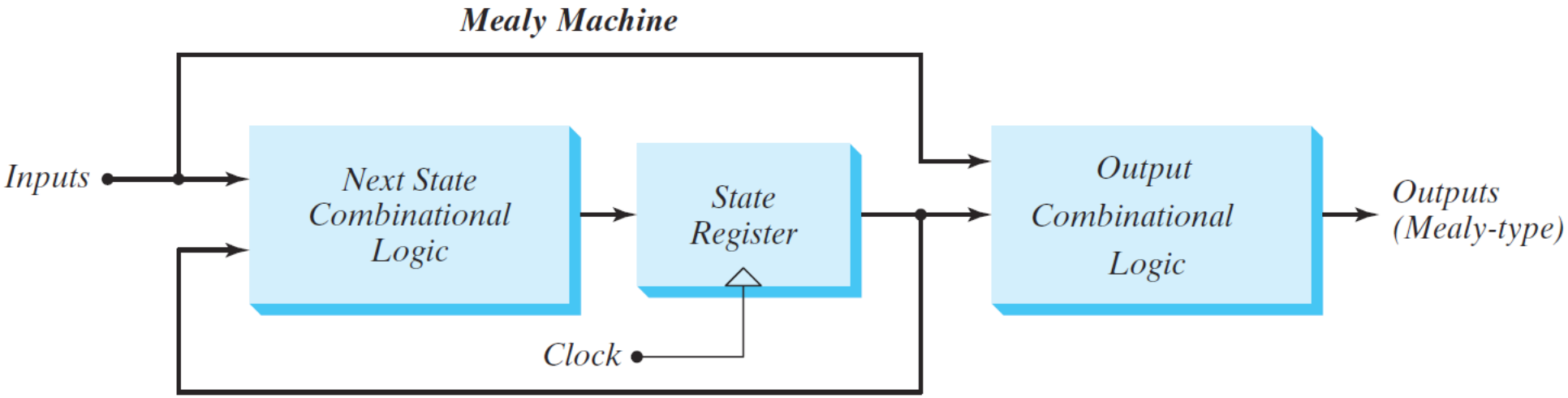
There are two ways to design a clocked sequential circuit:

1. **Mealy Machine:** Outputs depend on present state and inputs
2. **Moore Machine:** Outputs depend on present state only



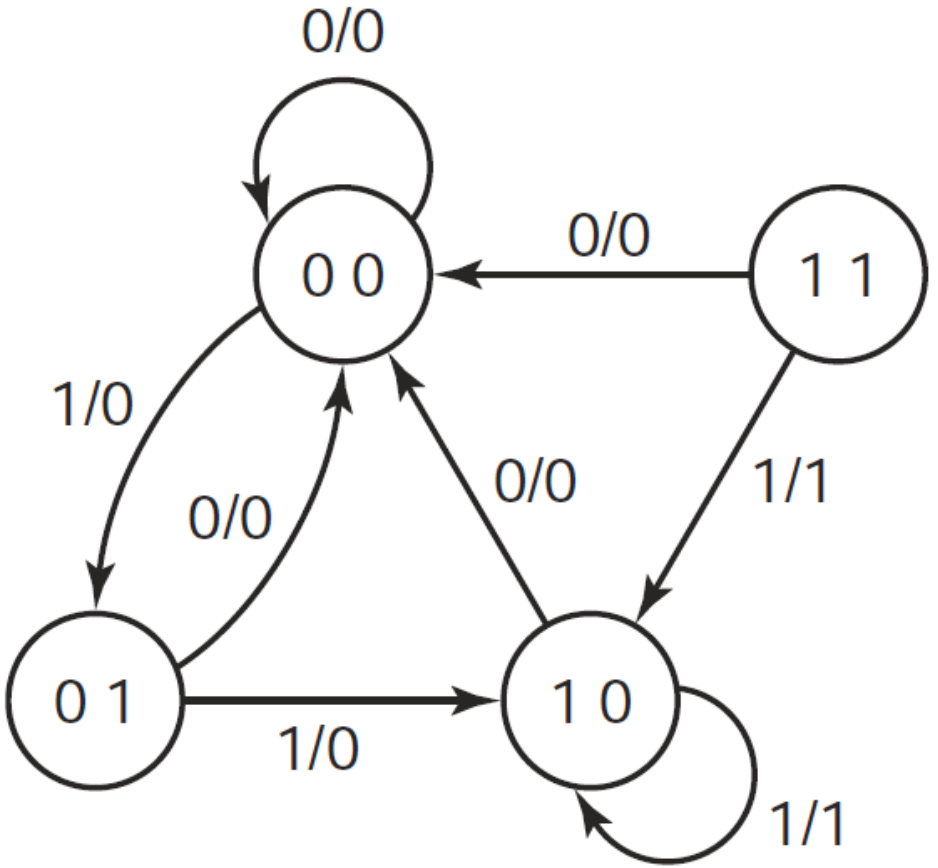
Mealy Machine

- The outputs are a function of the present state and Inputs
- The outputs are **NOT** synchronized with the clock
- The outputs may change if inputs change during the clock cycle
- The outputs may have momentary false values (called glitches)
- The correct outputs are present just before the edge of the clock

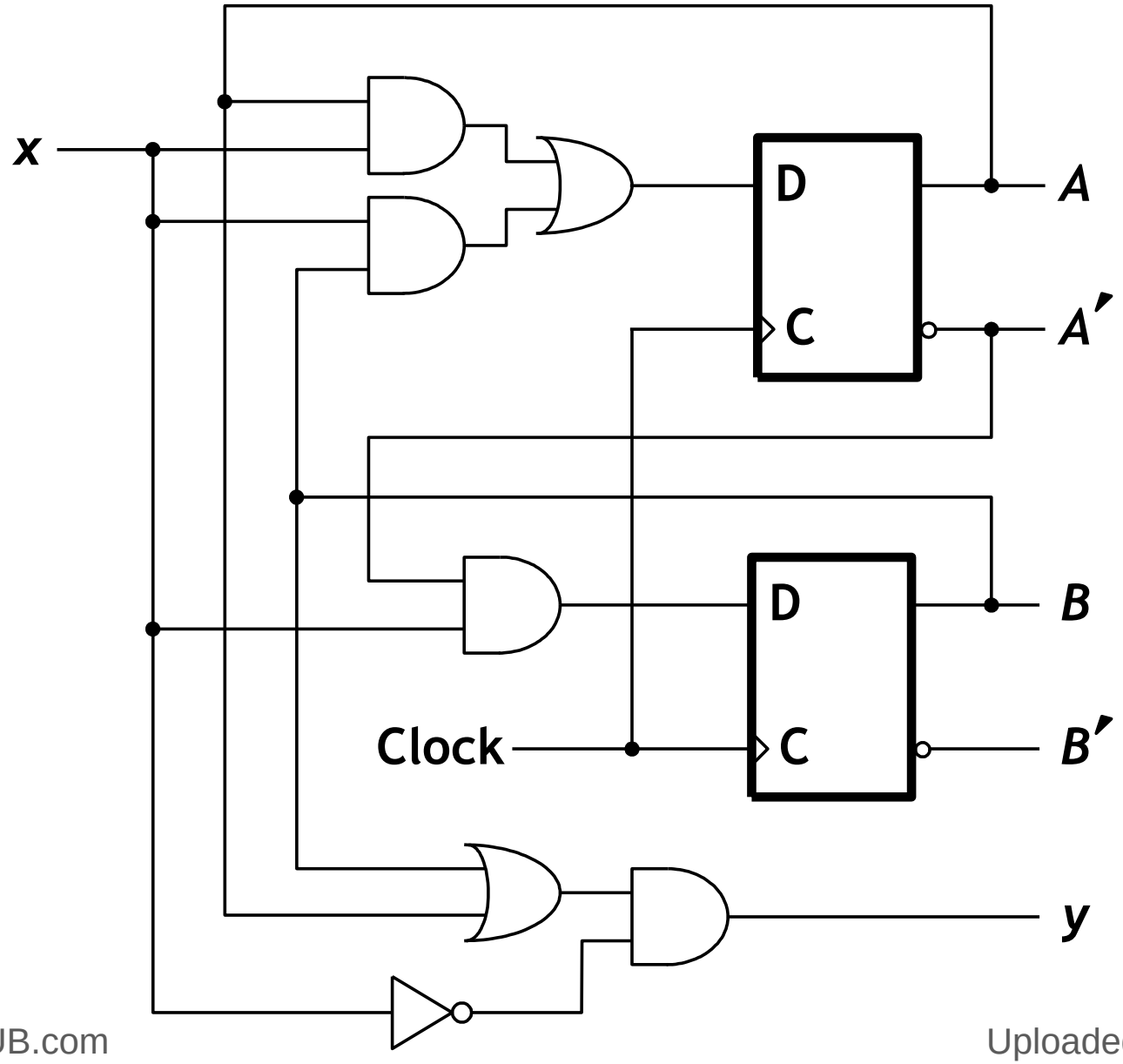


Mealy State Diagram

- An example of a Mealy state diagram is shown on the right
- Each arc is labeled with: **Input / Output**
- The output is shown on the arcs of the state diagram
- The output depends on the current state and input
- Notice that State 11 cannot be reached from the other states

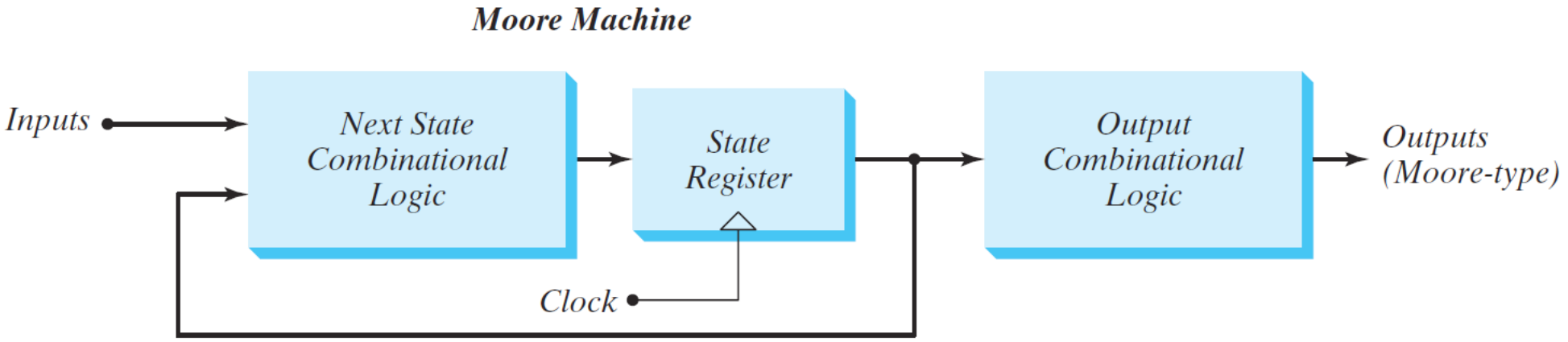


Example of Mealy Model



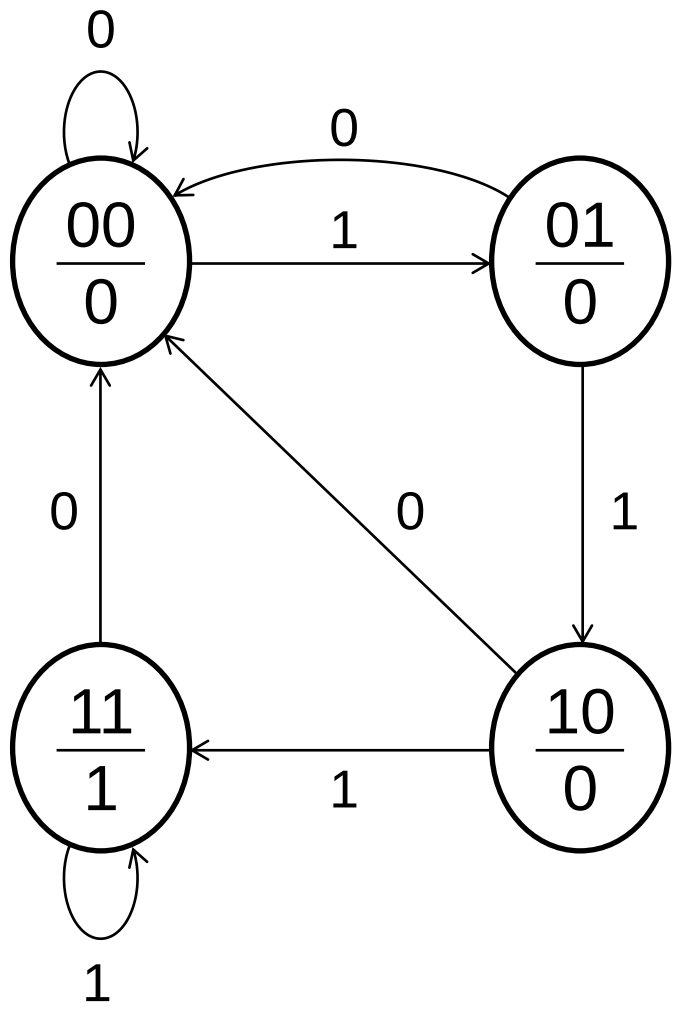
Moore Machine

- The outputs are a function of the Flip-Flop outputs only
- The outputs depend on the current state only
- The outputs are synchronized with the clock
- Glitches cannot appear in the outputs (even if inputs change)
- A given design might mix between Mealy and Moore



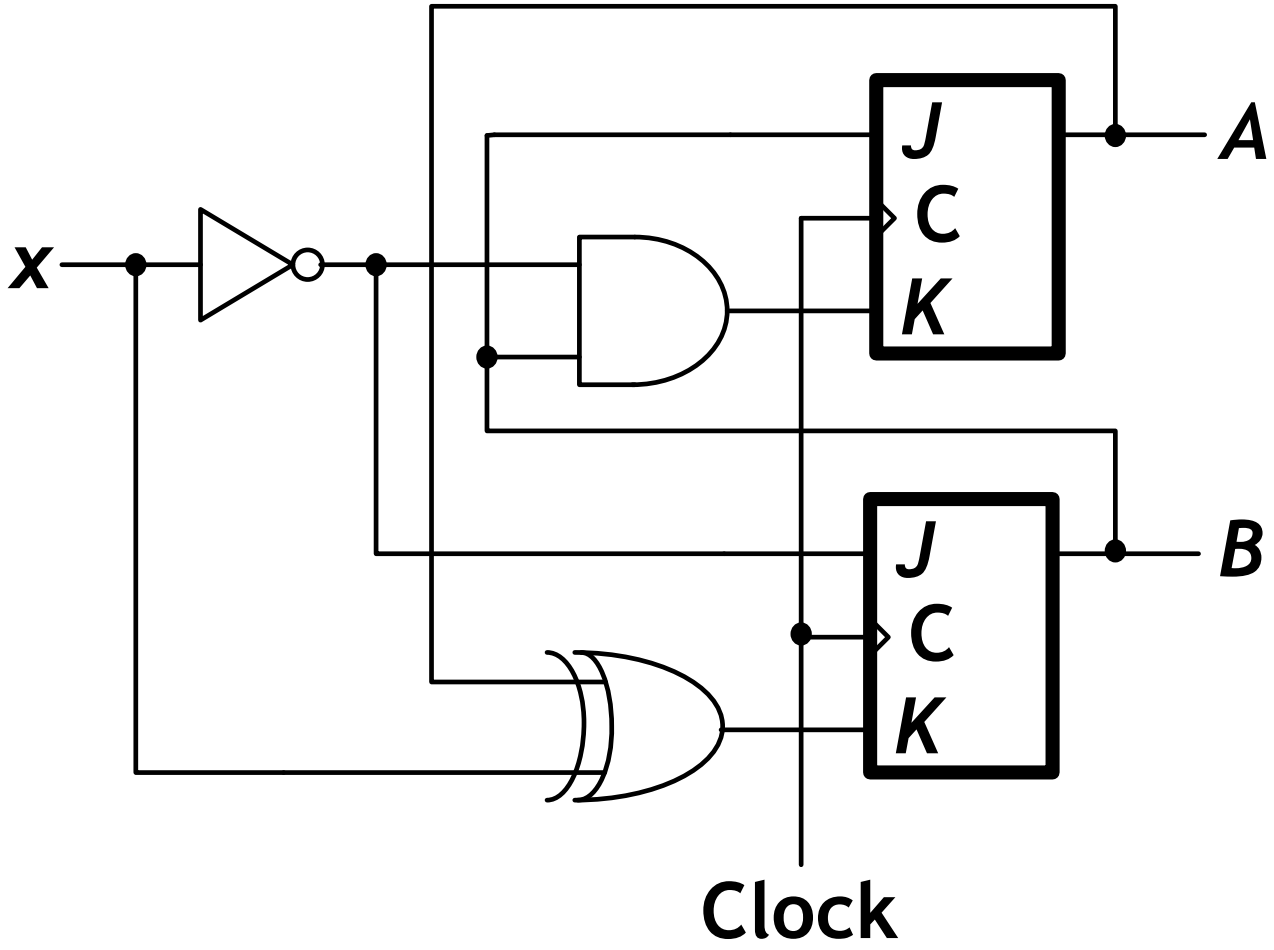
Moore State Diagram

- An example of a Moore state diagram is shown on the right
- Arcs are labeled with input only
- The output is shown inside the state: (State / Output)
- The output depends on the current state only



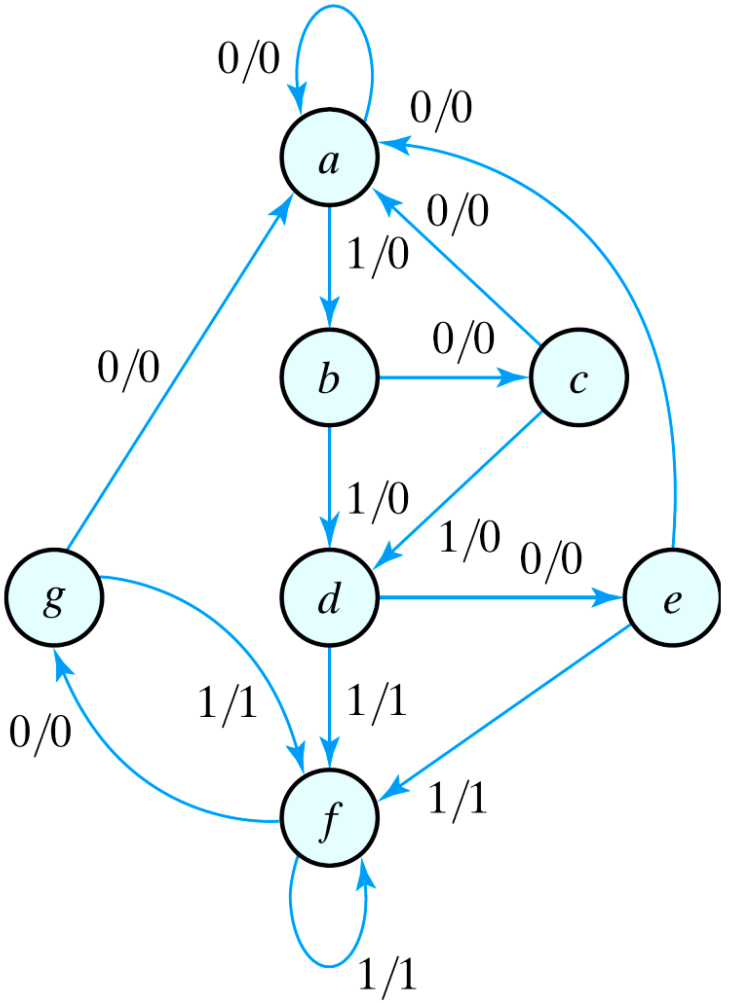
Example of Moore Model

Sequential Circuit with *JK* Flip-Flop



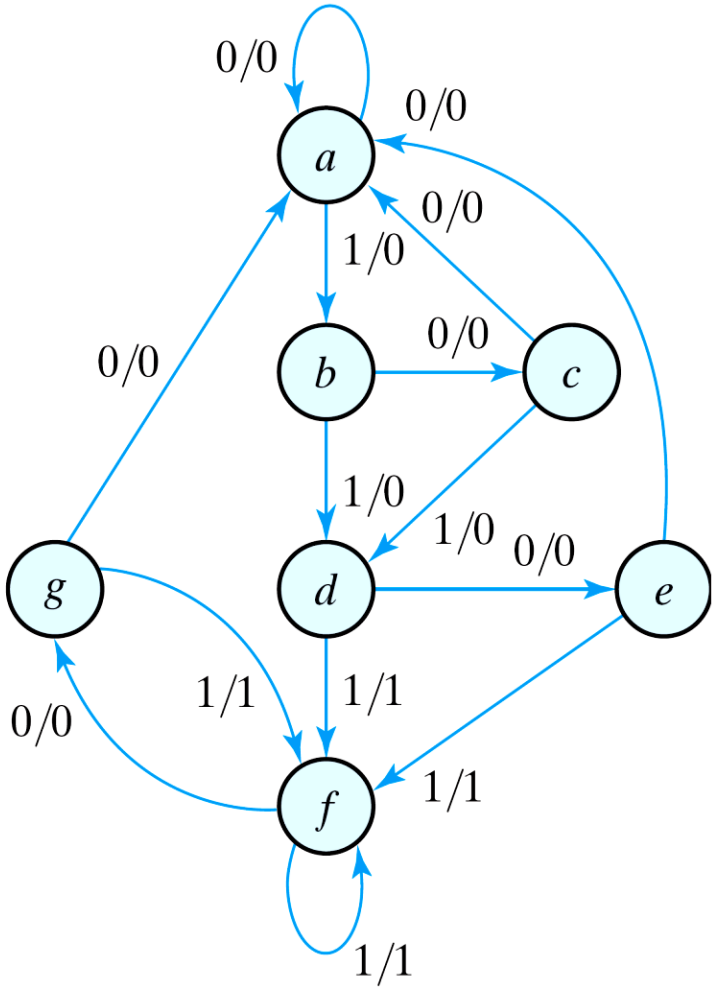
State Reduction and Assignment

- Design starts with state table or diagram
- State reduction aims at exhibiting the same input-output behavior but with a lower number of internal states
- State Reduction
  - Reductions on the number of flip-flops and the number of gates.
  - A reduction in the number of states may result in a reduction in the number of flip-flops.
  - May lead to use more gates



State Reduction

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	



- Only the input-output sequences are important.
- Two circuits are equivalent
  - Have identical outputs for all input sequences;
  - The number of states is not important.

Equivalent states

- Two states are said to be equivalent
  - For each member of the set of inputs, they give exactly the same output and send the circuit to the same state or to an equivalent state.
  - One of them can be removed.

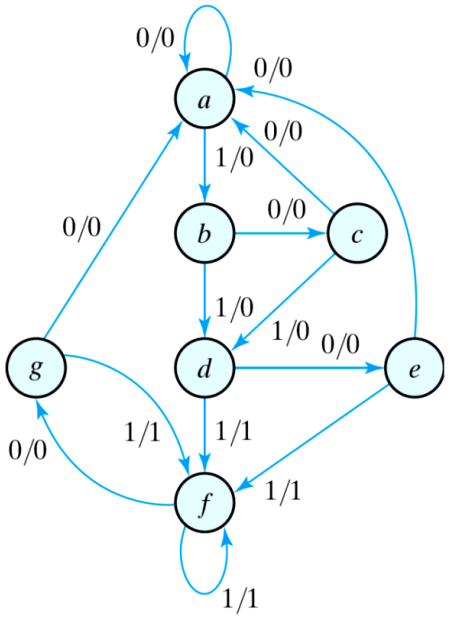


Table 5.6
State Table

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

Equivalent states

Remove state

e = g

Reducing the state table

- $e = g$ (remove g);
- $d = f$ (remove f);

Table 5.7
Reducing the State Table

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

State Reduction

- The checking of each pair of states for possible equivalence can be done systematically
- The unused states are treated as don't-care condition ⇒ fewer combinational gates.

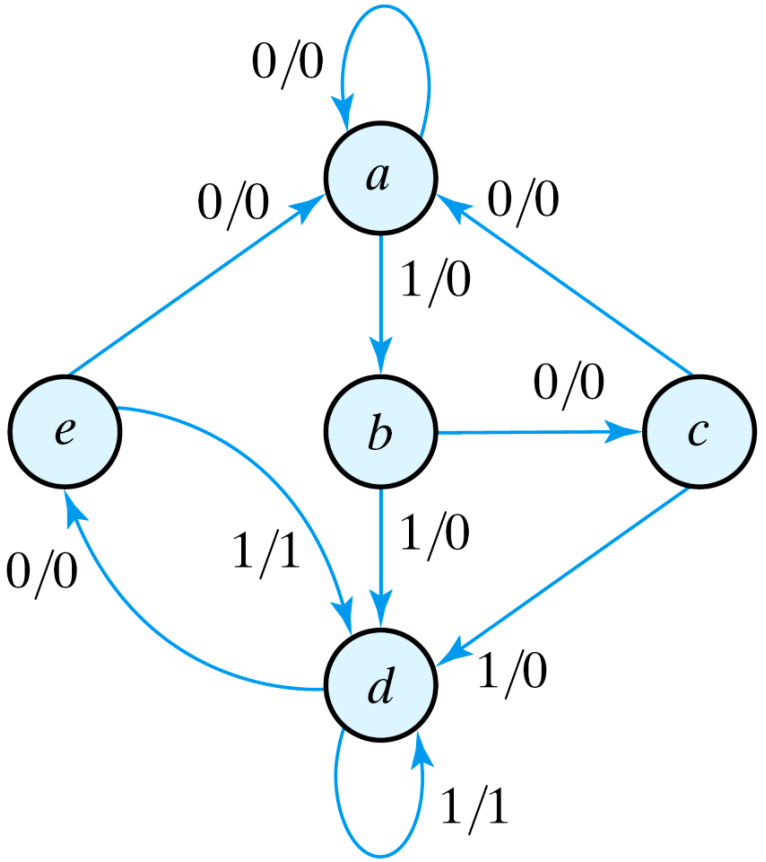


Table 5.8
Reduced State Table

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

State Assignment

- To minimize the cost of the combinational circuits.
- Three possible binary state assignments. (m states need n-bits, where $2^n > m$)

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

What code assignment would you choose? Why?

State Assignment

- Any binary number assignment is satisfactory as long as each state is assigned a unique number.
- Use binary assignment 1.

Table 5.10
Reduced State Table with Binary Assignment 1

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

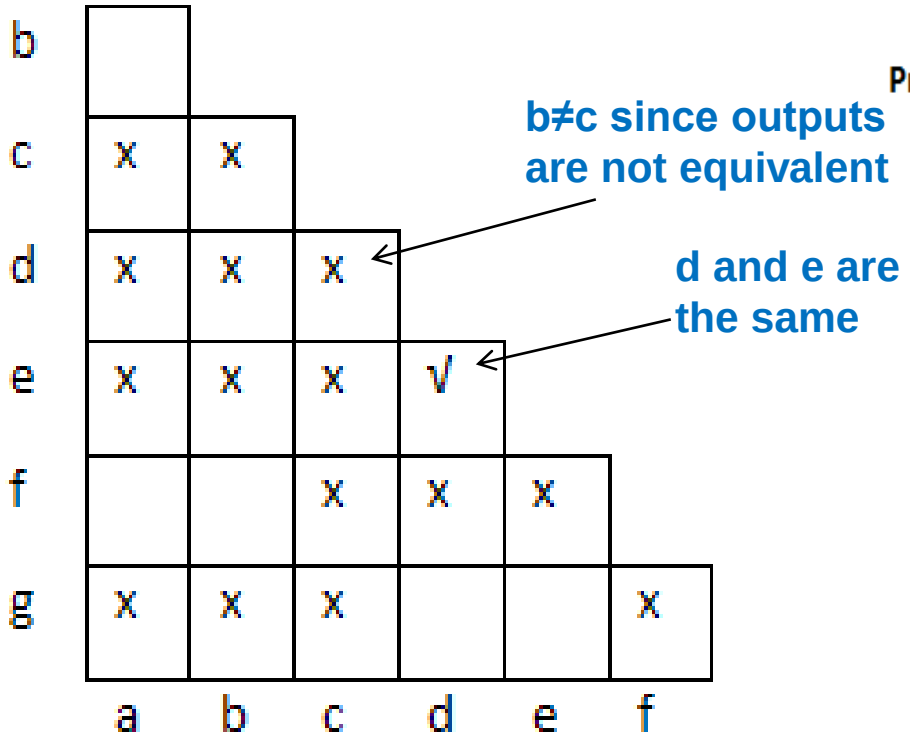
Implication Chart

- To check possible equivalent states in table with large number of states.
- Example

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

Implication Chart

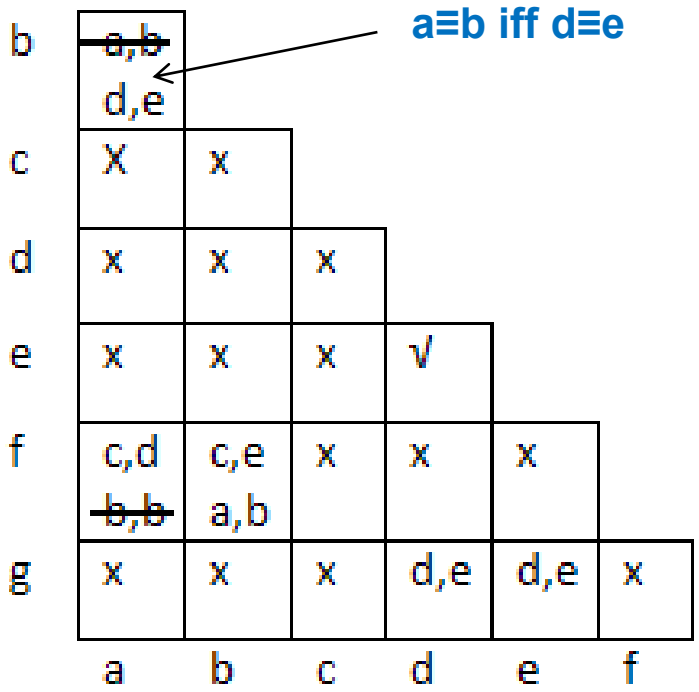
- Step1: draw the implication chart and place (X) in any square of a pair of states whose outputs are not equivalent.
  - Place (√) for equivalent states (same outputs, same next state).



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

Implication Chart

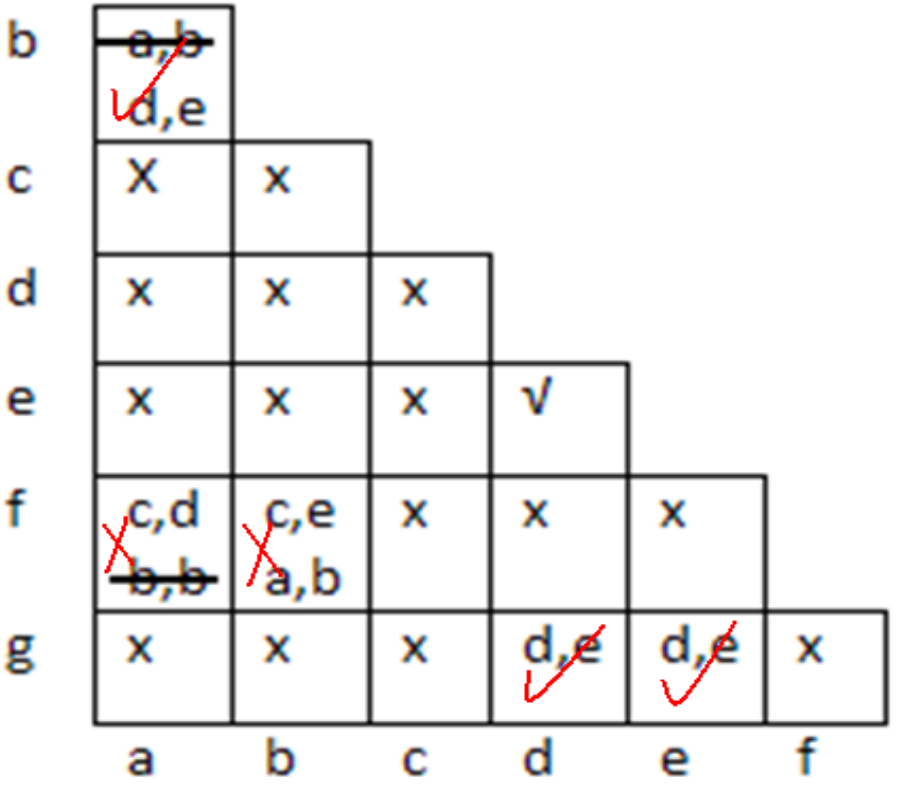
- Step2: for remaining squares, enter the implied states



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

Implication Chart

- Step3: Place (√) for equivalent states and (X) for not equivalent states



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

Implication Chart

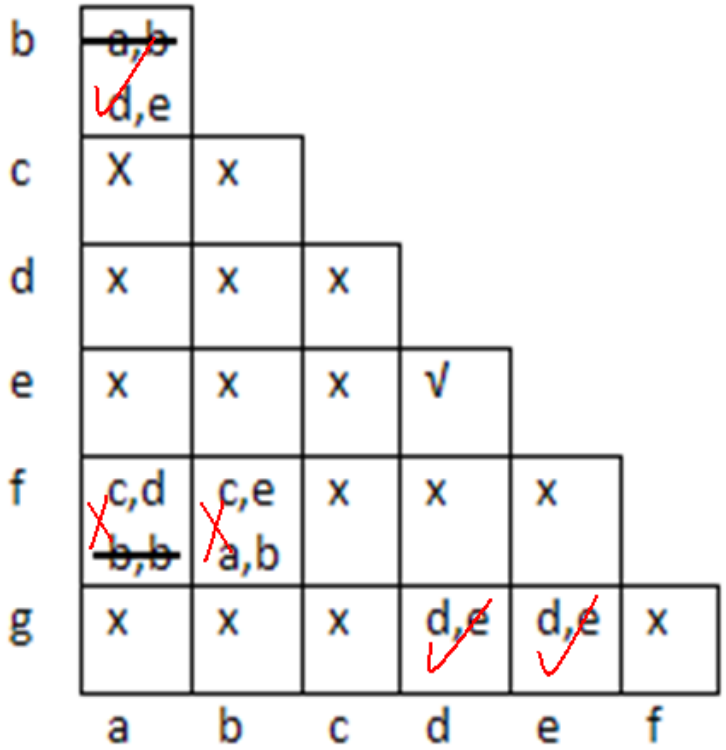
- Step4: list equivalent states from squares with (√)

(a, b),(d, e), (d, g),(e, g)

d d d

- Step5: combine pairs of states into large group

(a, b), (d, e, g)



Implication Chart

- Step6: the final states are the equivalent states and all remaining states in state table:

(a,b) ⟹ a

(c)

(d,e,g) ⟹ d

(f)

Implication Chart

- The table can be reduced from seven states into four states:

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

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

Design of Sequential Logic

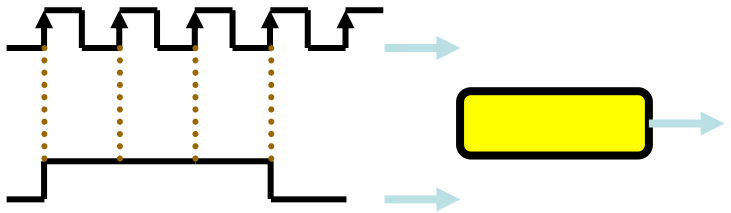
❖ Design Procedure

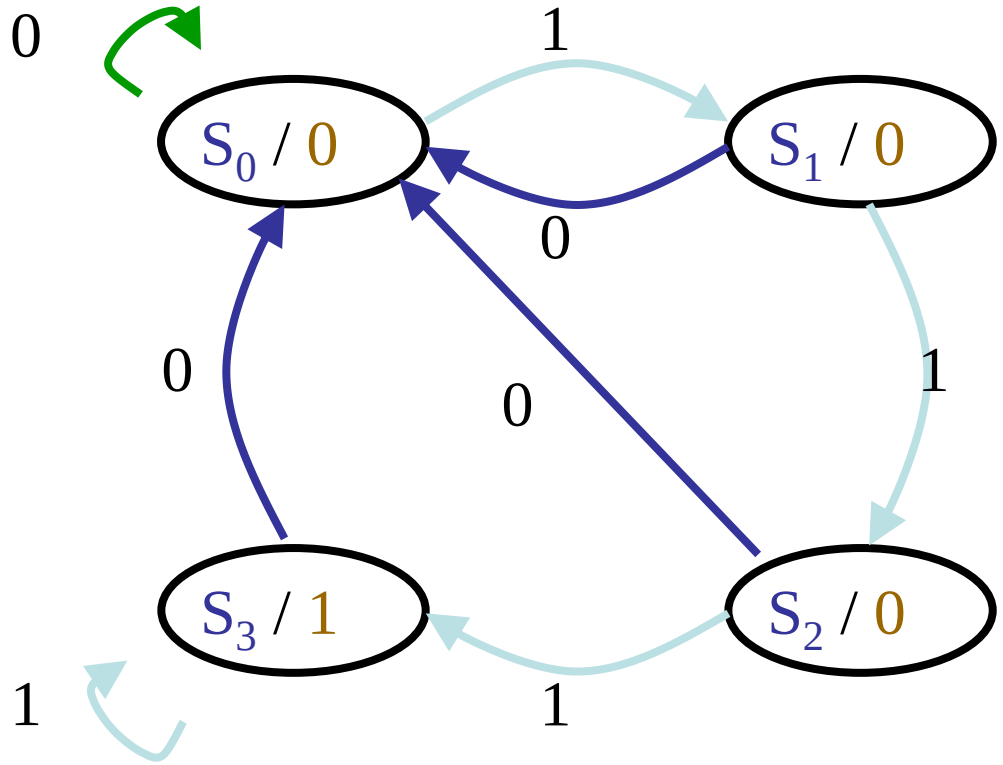
1. From the word description and specifications of the desired operation, derive a state diagram for the circuit.
2. Reduce the number of states if necessary
3. Assign binary values to the states
4. Obtain the binary-coded state table
5. Choose the type of flip-flops to be used
6. Derive the simplified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

Design of Clocked Sequential Circuits

❖ *Example*:

Detect 3 or more consecutive 1's



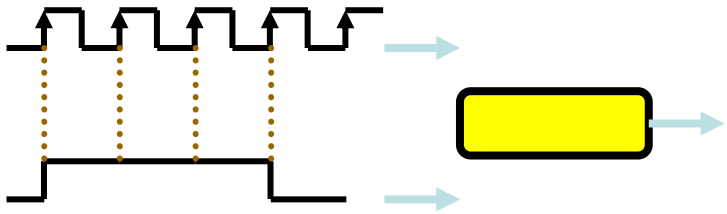


State	*A* *B*
S_0	0 0
S_1	0 1
S_2	1 0
S_3	1 1

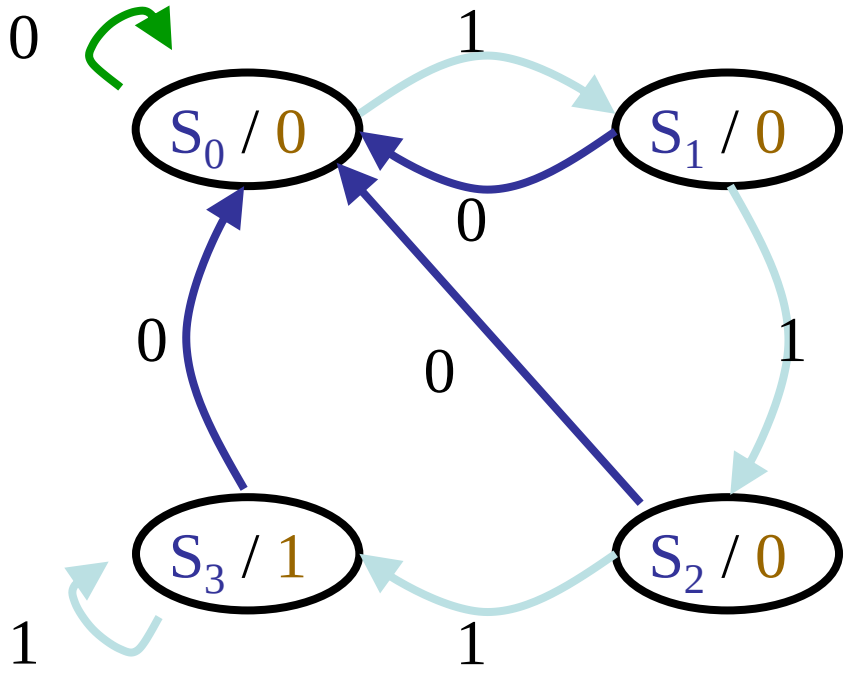
Design of Clocked Sequential Circuits

❖ *Example*:

Detect 3 or more consecutive 1's



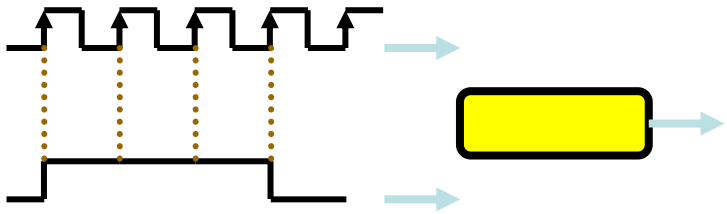
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1



Design of Clocked Sequential Circuits

❖ *Example*:

Detect 3 or more consecutive 1's



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

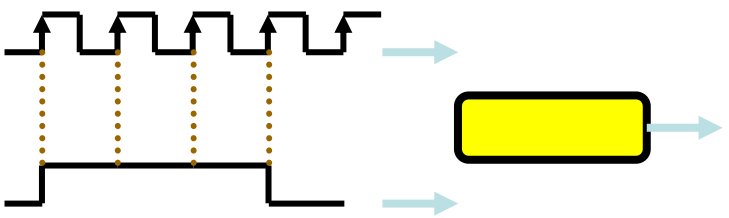
Synthesis using *D* Flip-Flops

$$A(t+1) = D_A(A, B, x)$$
$$= \sum(3, 5, 7)$$
$$B(t+1) = D_B(A, B, x)$$
$$= \sum(1, 5, 7)$$
$$y(A, B, x) = \sum(6, 7)$$

Design of Clocked Sequential Circuits with D F.F.

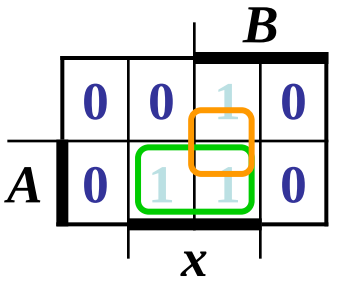
❖ *Example*:

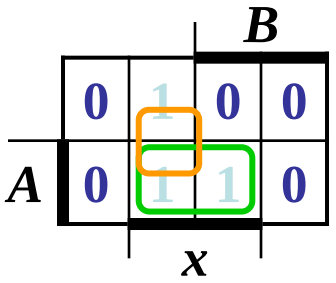
Detect 3 or more consecutive 1's

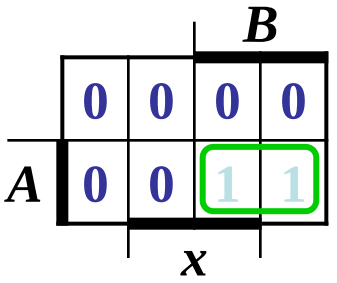


Synthesis using *D* Flip-Flops

$$D_A(A, B, x) = \sum(3, 5, 7)$$
$$= A\,x + B\,x$$
$$D_B(A, B, x) = \sum(1, 5, 7)$$
$$= A\,x + B'x$$
$$y(A, B, x) = \sum(6, 7)$$
$$= A\,B$$



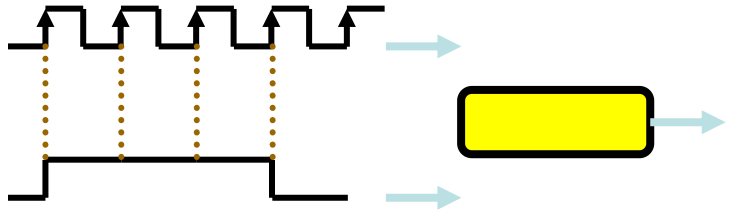




Design of Clocked Sequential Circuits with D F.F.

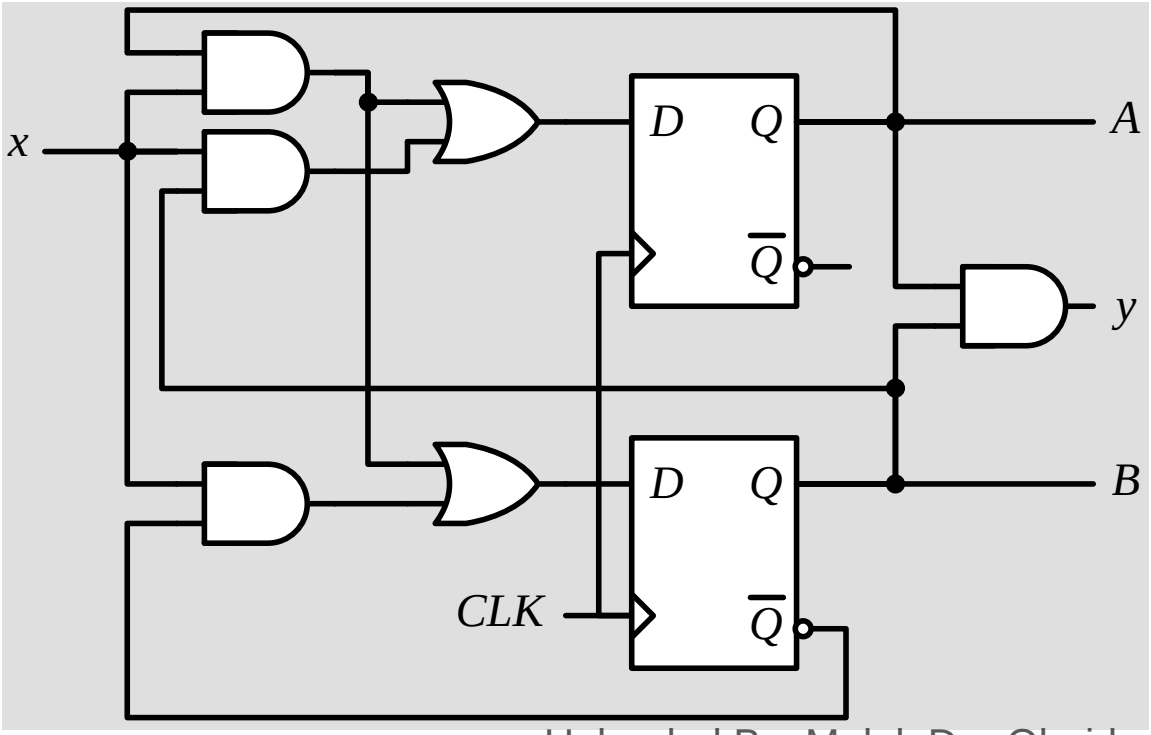
❖ *Example*:

Detect 3 or more consecutive 1's



Synthesis using *D* Flip-Flops

$$D_A = A\,x + B\,x$$
$$D_B = A\,x + B'x$$
$$y = A\,B$$



Flip-Flop Excitation Tables

Present State	Next State	F.F. Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Present State	Next State	F.F. Input	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

0 0 (No change)
0 1 (Reset)

1 0 (Set)
1 1 (Toggle)

0 1 (Reset)
1 1 (Toggle)

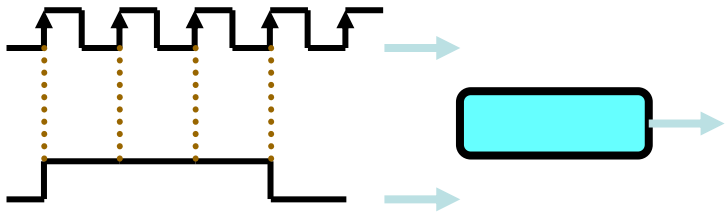
0 0 (No change)
1 0 (Set)

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

Design of Clocked Sequential Circuits with JK F.F.

❖ *Example*:

Detect 3 or more consecutive 1's



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	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	0	0	x	1	0	x
1	0	1	1	1	x	0	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	1	x	0	x	0

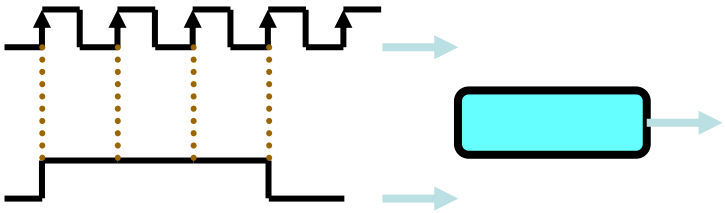
Synthesis using *JK* F.F.

$$J_A(A, B, x) = \sum(3)$$
$$d_{JA}(A, B, x) = \sum(4,5,6,7)$$
$$K_A(A, B, x) = \sum(4, 6)$$
$$d_{KA}(A, B, x) = \sum(0,1,2,3)$$
$$J_B(A, B, x) = \sum(1, 5)$$
$$d_{JB}(A, B, x) = \sum(2,3,6,7)$$
$$K_B(A, B, x) = \sum(2, 3, 6)$$
$$d_{KB}(A, B, x) = \sum(0,1,4,5)$$

Design of Clocked Sequential Circuits with *JK* F.F.

❖ *Example*:

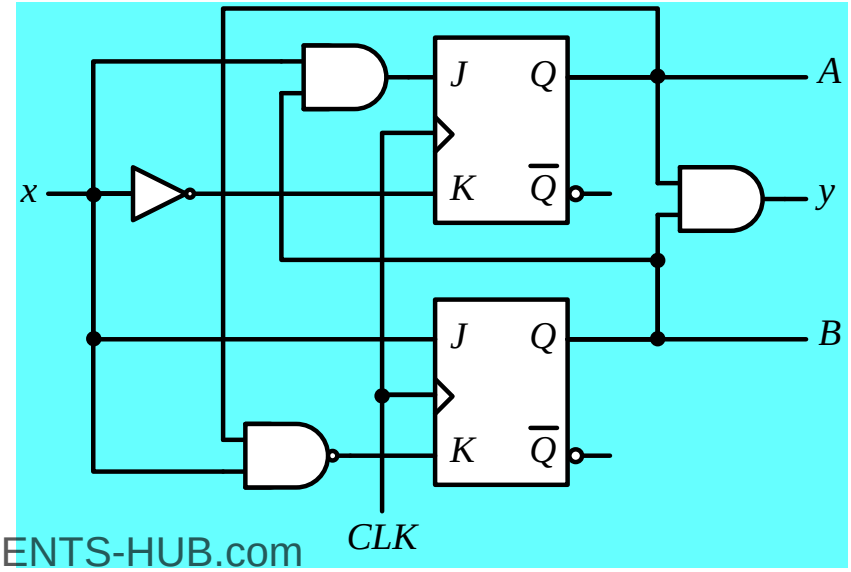
Detect 3 or more consecutive 1's

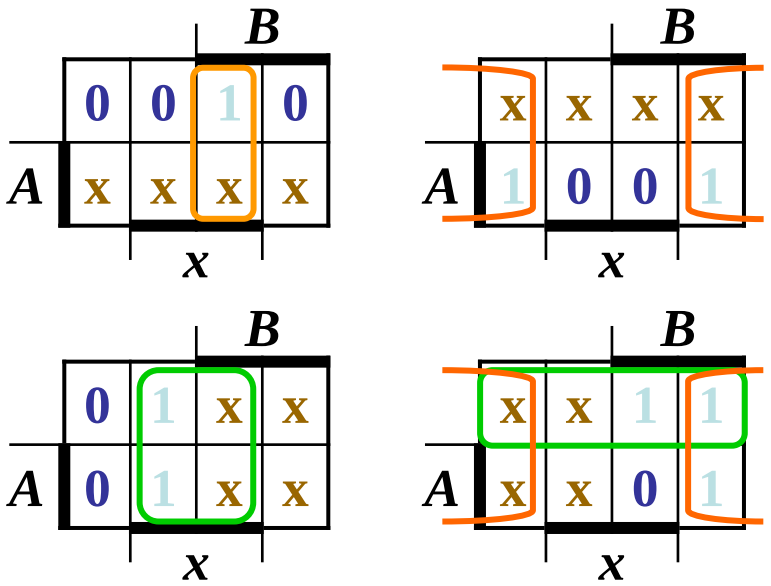


Synthesis using *JK* Flip-Flops

$J_A = B\,x$ $\quad K_A = x'$

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

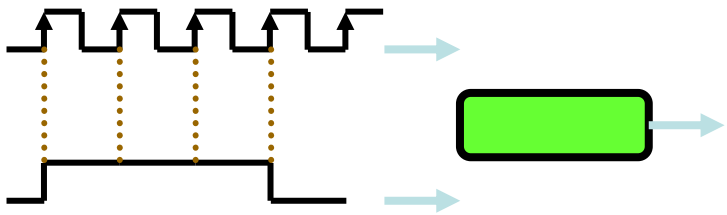




Design of Clocked Sequential Circuits with T F.F.

❖ *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		F.F. Input	
A	*B*	*x*	*A*	*B*	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	1	1	0	0

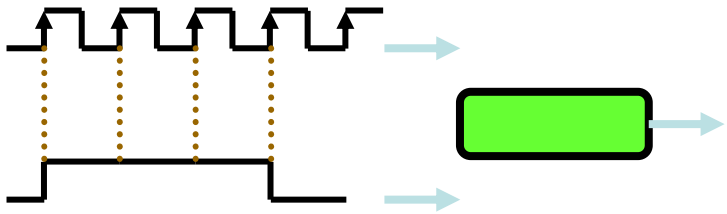
Synthesis using *T* Flip-Flops

$$T_A(A, B, x) = \sum(3, 4, 6)$$
$$T_B(A, B, x) = \sum(1, 2, 3, 5, 6)$$

Design of Clocked Sequential Circuits with T F.F.

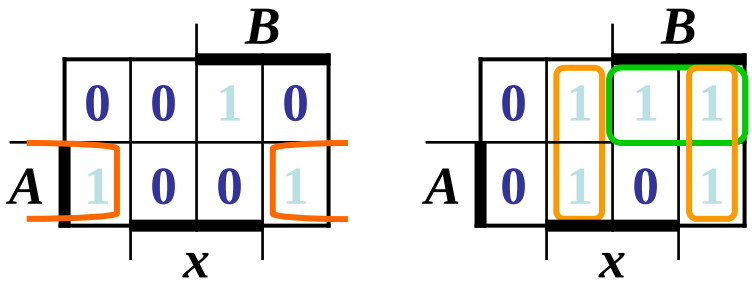
❖ *Example*:

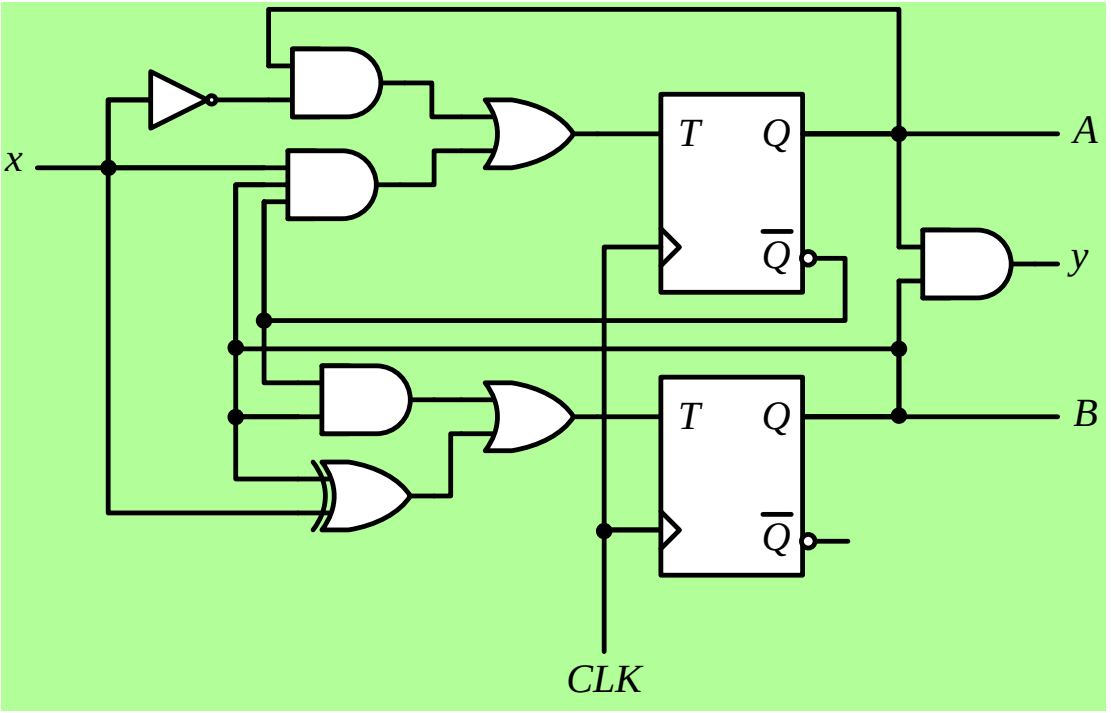
Detect 3 or more consecutive 1's



Synthesis using *T* Flip-Flops

$$T_A = A\,x' + A'B\,x$$
$$T_B = A'B + B \oplus x$$





Design of a Binary Counter

Problem Specification:

- Design a circuit that counts up from 0 to 7 then back to 0

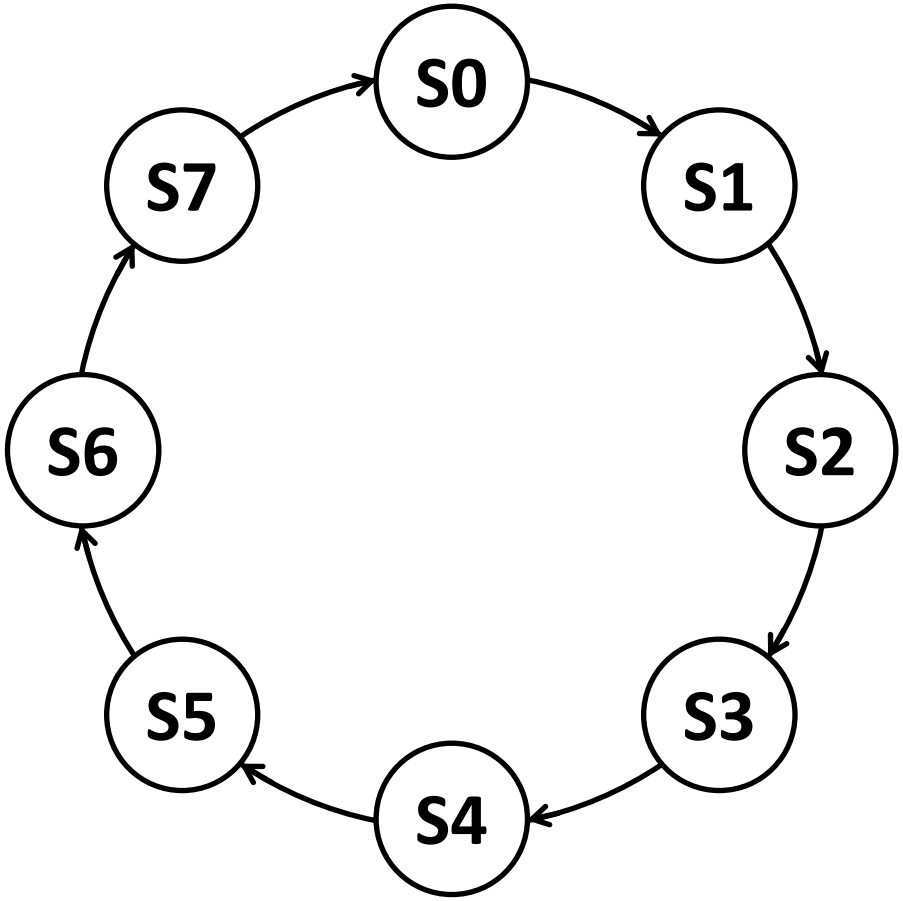
  000 → 001 → 010 → 011 → 100 → 101 → 110 → 111 → 000

  When reaching 7, the counter goes back to 0 then goes up again

- There is no input to the circuit
- The counter is incremented each cycle
- The output of the circuit is the present state (count value)
- The circuit should be designed using D-type Flip-Flops

Designing the State Diagram

- Eight states are needed to store the count values 0 to 7
- No input, state transition happens at the edge of each cycle



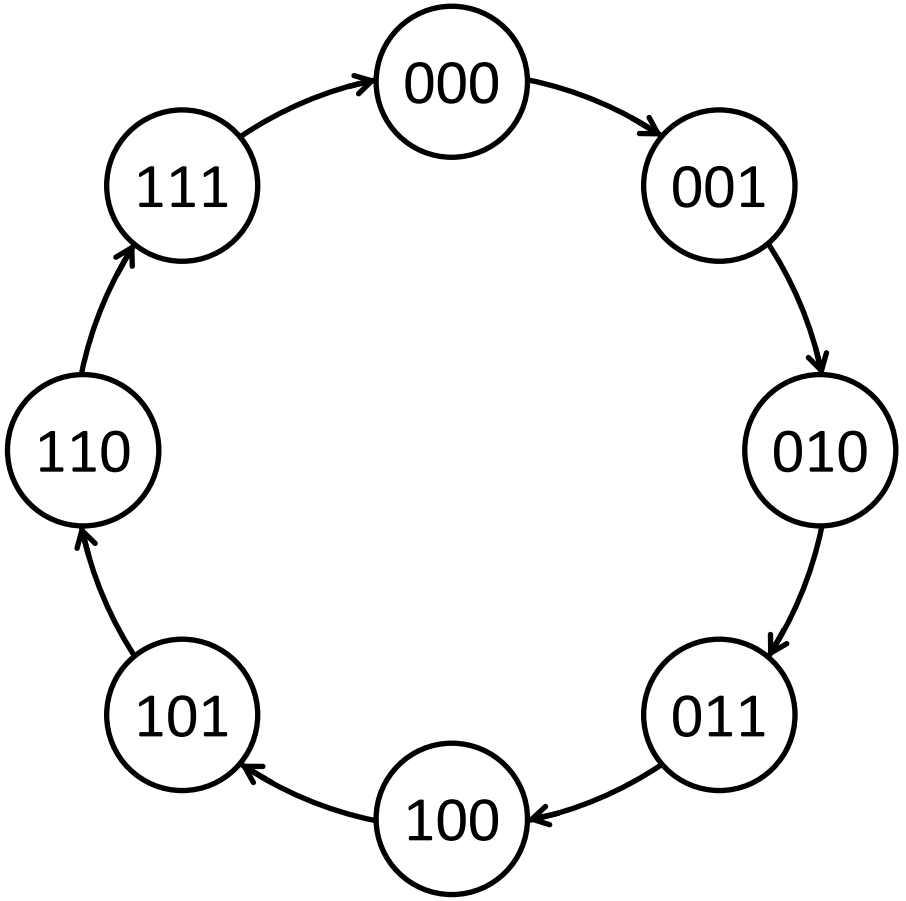
Three Flip-Flops are required for the eight states

Each state is assigned a unique binary count value

State Table

Only two columns: Present State and Next State

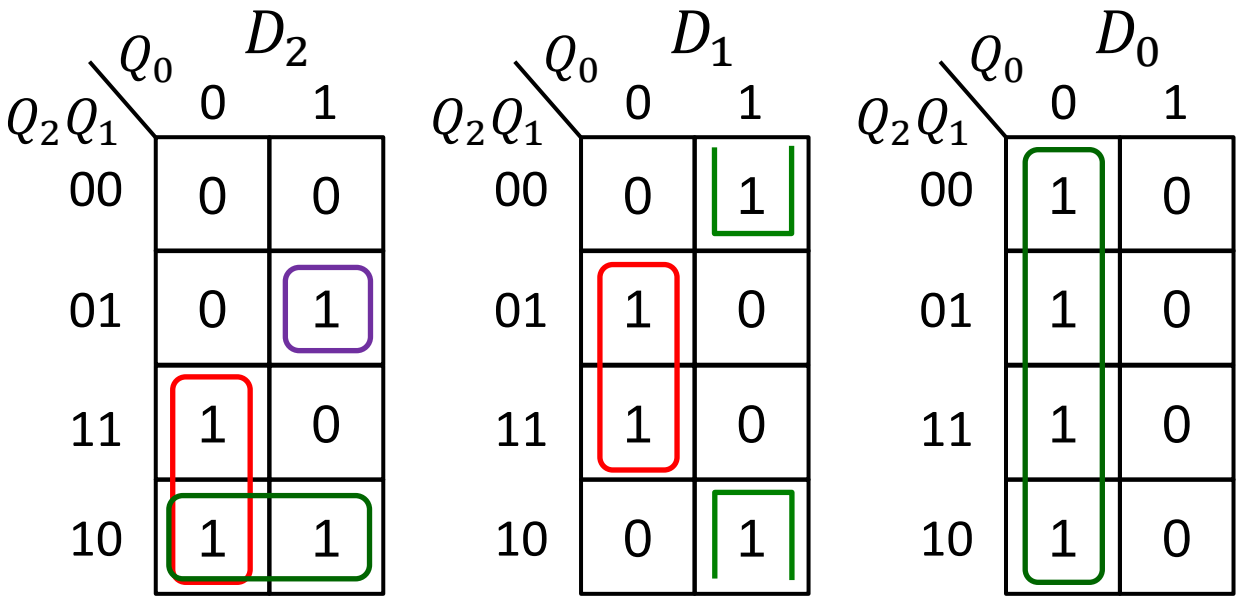
State changes each cycle



Present State Q_2 Q_1 Q_0	Next State D_2 D_1 D_0
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0
1 1 0	1 1 1
1 1 1	0 0 0

Deriving the Next State Equations

Present State Q_2 Q_1 Q_0	Next State D_2 D_1 D_0
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0
1 1 0	1 1 1
1 1 1	0 0 0



$$D_2 = Q_2Q_1' + Q_2Q_0' + Q_2'Q_1Q_0$$

$$D_2 = Q_2(Q_1' + Q_0') + Q_2'Q_1Q_0$$

$$D_2 = Q_2(Q_1Q_0)' + Q_2'(Q_1Q_0) = Q_2 \oplus (Q_1Q_0)$$

$$D_1 = Q_1Q_0' + Q_1'Q_0 = Q_1 \oplus Q_0$$

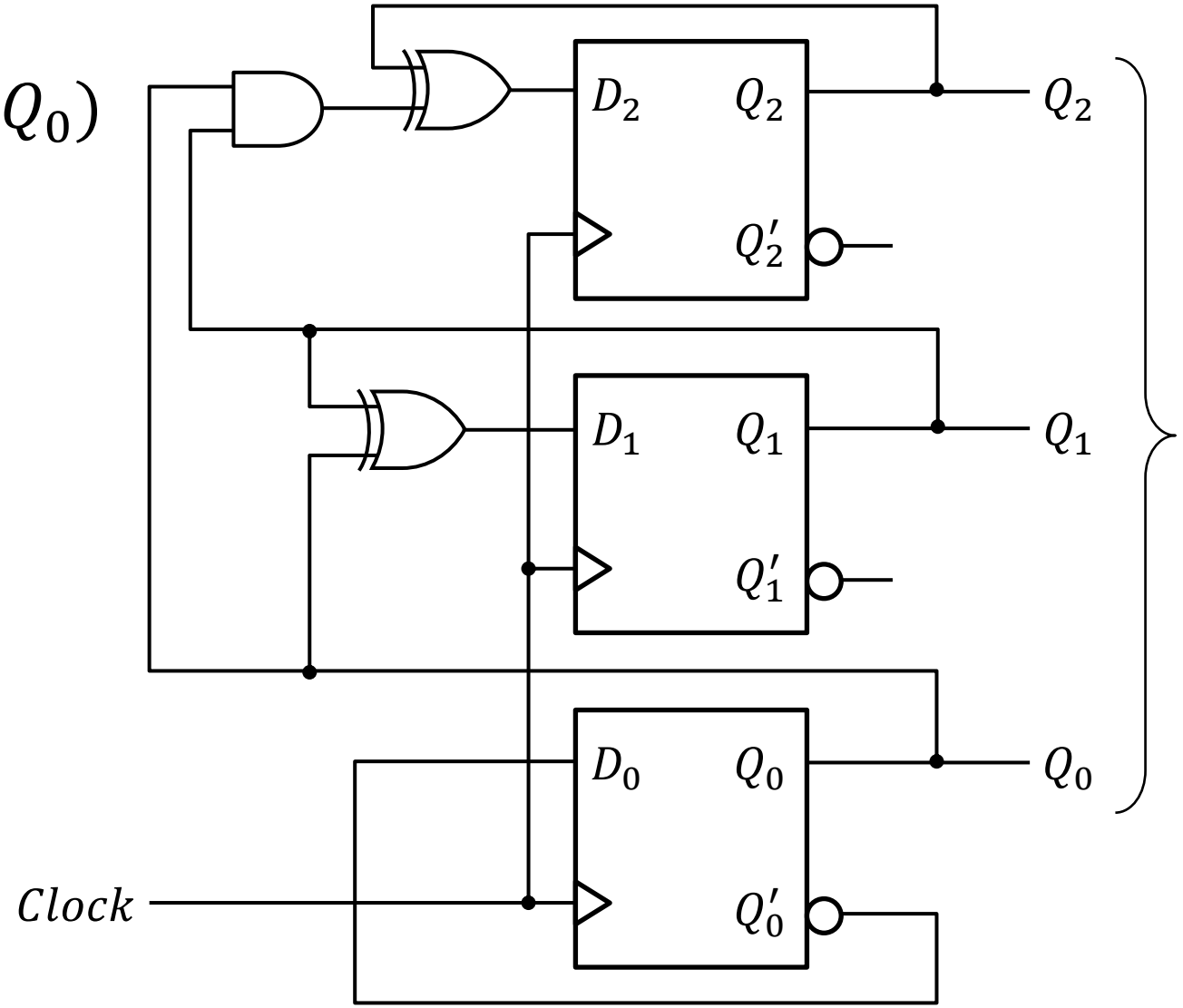
$$D_0 = Q_0'$$

3-Bit Counter Circuit Diagram

$D_2 = Q_2 \oplus (Q_1 Q_0)$

$D_1 = Q_1 \oplus Q_0$

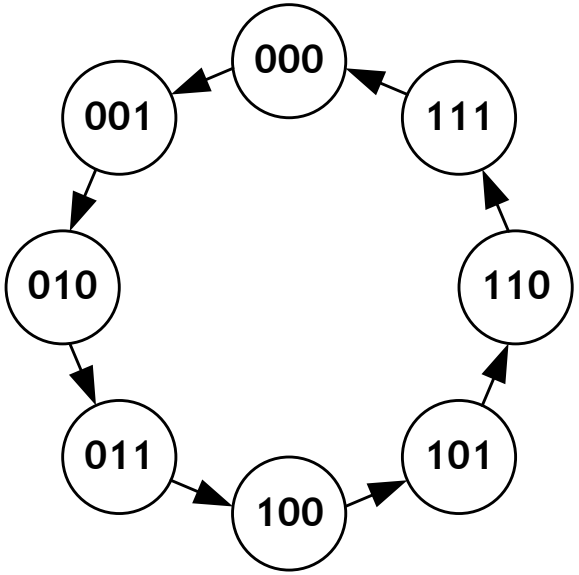
$D_0 = Q_0'$



Design Example: 3-bit Binary Counter Using T FFs.

❖ State Diagram and State Table of 3-bit Binary Counter

State Diagram



Excitation Table

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

State Table

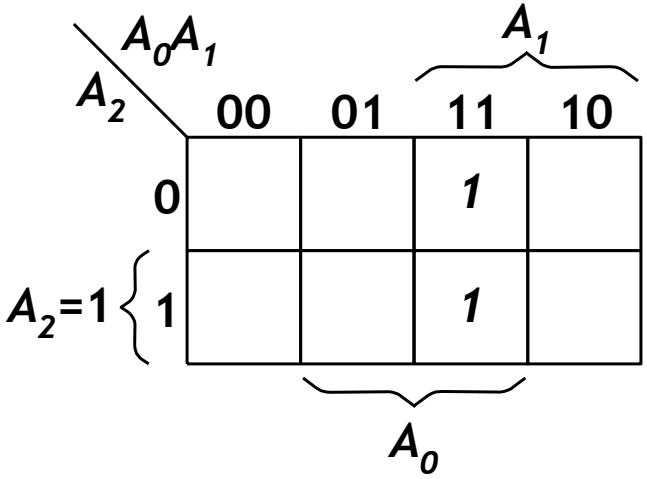
Present State $A_2\ A_1\ A_0$	Next State $A_2\ A_1\ A_0$	Flip-Flop Inputs $T_{A2}\ T_{A1}\ T_{A0}$
0 0 0	0 0 1	0 0 1
0 0 1	0 1 0	0 1 1
0 1 0	0 1 1	0 0 1
0 1 1	1 0 0	1 1 1
1 0 0	1 0 1	0 0 1
1 0 1	1 1 0	0 1 1
1 1 0	1 1 1	0 0 1
1 1 1	0 0 0	1 1 1

Refer to T-FF Excitation Table

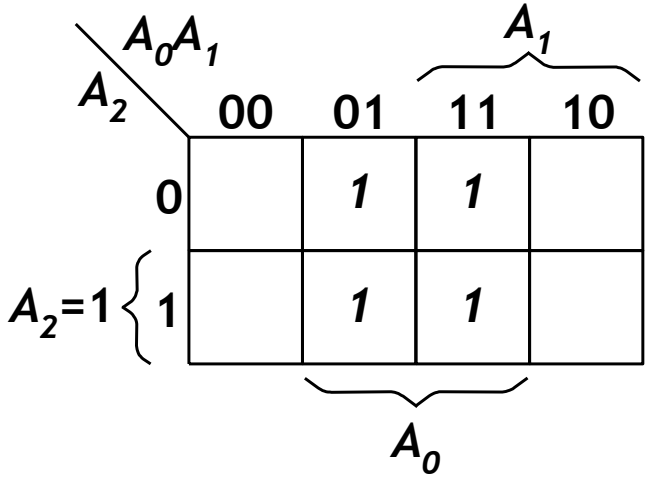
Design Example: 3-bit Binary Counter Using T FFs.

❖ K-Map Logic Simplification for 3-bit Binary Counter

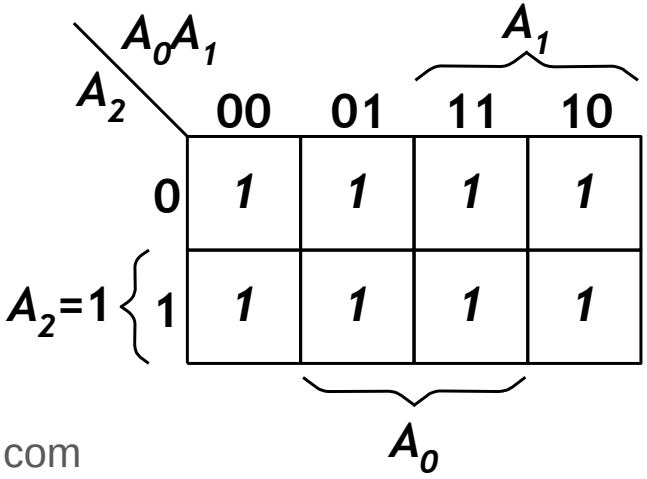
$T_{A2} = A_1A_0$



$T_{A1} = A_0$

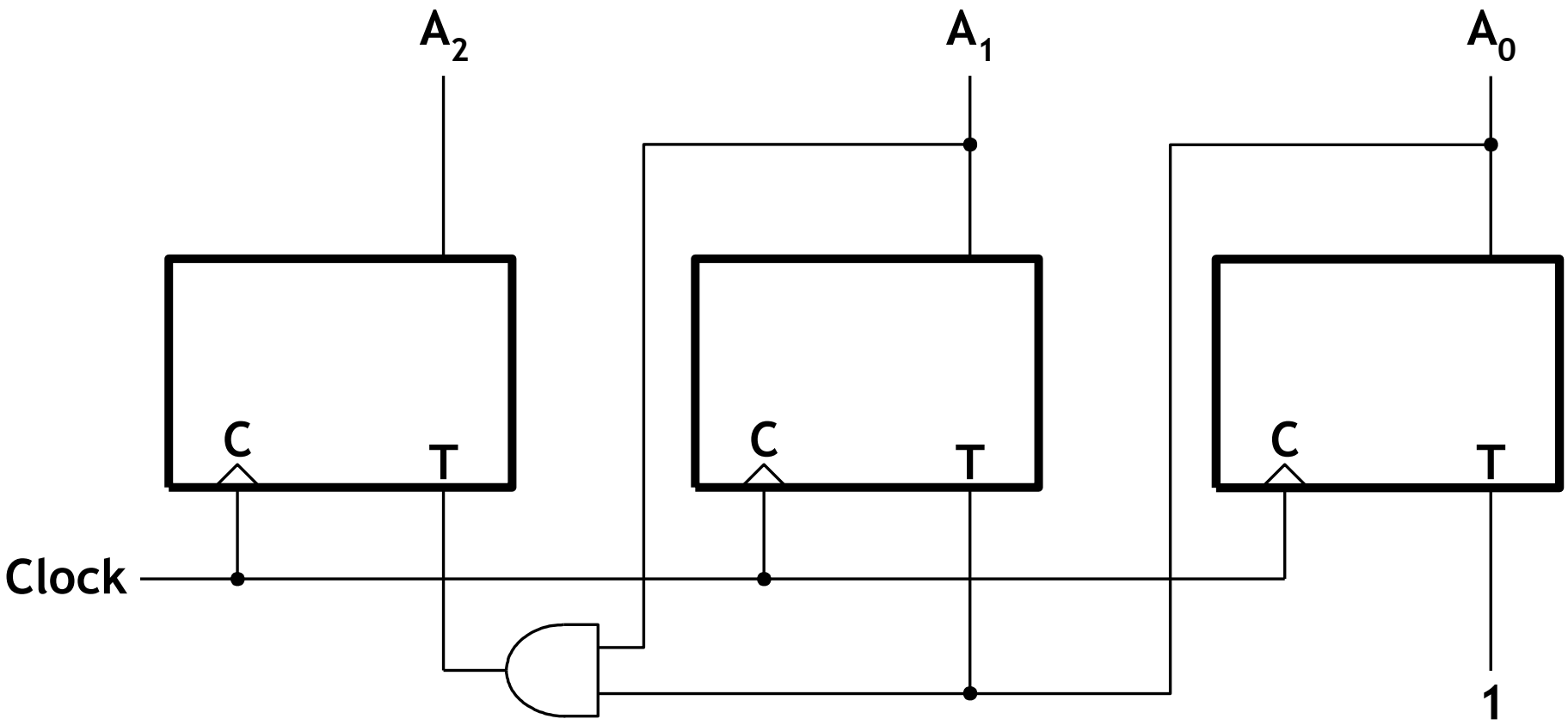


$T_{A0} = 1$



Design Example: 3-bit Binary Counter Using T FFs.

- Draw the 3-bit Binary Counter Circuits with T FFs

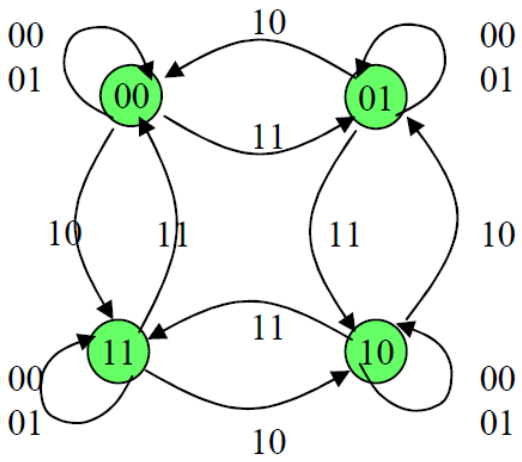


Up/Down Counter with Enable

- **Problem**: Design a synchronous up-down T flip-flop 2-bit binary counter with a select input line S and a count enable En input. When S = 0, the counter counts down; and when S = 1, the counter counts up. When En = 1, the counter is in normal up- or down- counting; and En = 0 for disabling both counts.
- **Solution**: Required mode of operation:

Inputs		Operation
En	S	
0	x	Hold status
1	0	Count Down
1	1	Count Up

State Diagram/Table for 2-bit Up-Down Binary Counter



Arc Label: EnS

Present State			Inputs		Next State			T flip-flops	
No	Q1	Q0	En	S	**No**	Q1	Q0	T_{Q1}	T_{Q0}
0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	1	**0**	0	0	0	0
0	0	0	1	0	**3**	1	1	1	1
0	0	0	1	1	**1**	0	1	0	1
1	0	1	0	0	**1**	0	1	0	0
1	0	1	0	1	**1**	0	1	0	0
1	0	1	1	0	**0**	0	0	0	1
1	0	1	1	1	**2**	1	0	1	1
2	1	0	0	0	**2**	1	0	0	0
2	1	0	0	1	**2**	1	0	0	0
2	1	0	1	0	**1**	0	1	1	1
2	1	0	1	1	**3**	1	1	0	1
3	1	1	0	0	**3**	1	1	0	0
3	1	1	0	1	**3**	1	1	0	0
3	1	1	1	0	**2**	1	0	0	1
3	1	1	1	1	**0**	0	0	1	1

T Flip-Flop

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

Input Equations for 2-bit Up-Down Binary Counter

EnS / Q_1Q_0	00	01	11	10
00	0	0	0	1
01	0	0	1	0
11	0	0	1	0
10	0	0	0	1

$T_{Q1} = Q_0EnS + Q_0'EnS'$

EnS / Q_1Q_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

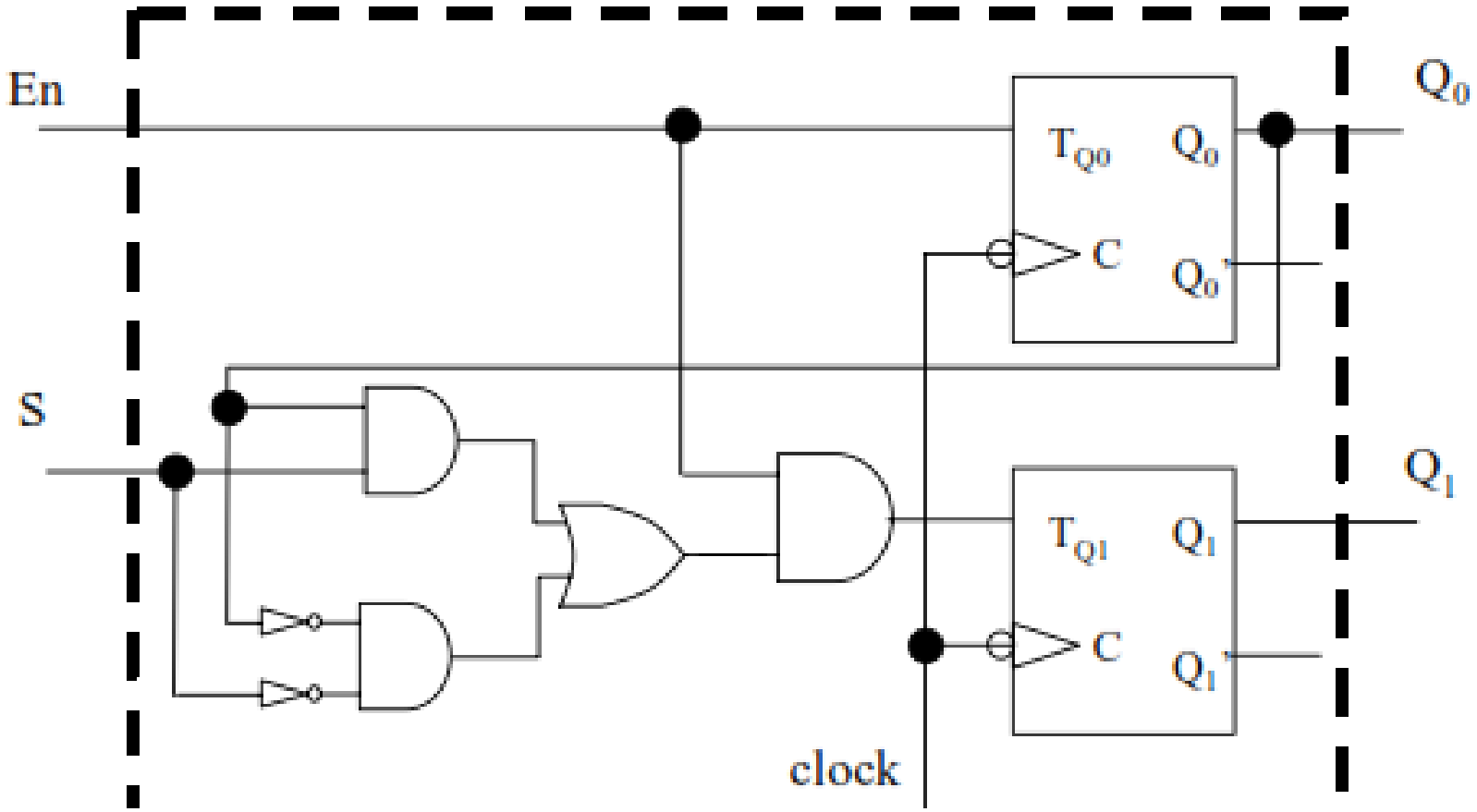
$T_{Q0} = En$

- The carry out signals:
  - CO_{up} and CO_{down}

$CO_{up} = Q_0Q_1EnS$ ➔ counter reached 11 and it is counting up

$CO_{down} = Q_0'Q_1'EnS'$ ➔ counter reached 00 and it is counting down

Circuit for 2-bit Up-Down Binary Counter



Dealing with Unused States

- An n-bit counter has 2n states, but there are occasions when we wish to use less than the total number of states available.
- The unused states may be treated as “don’t care” conditions (or assigned to specific next states).
- Because outside interference may land the counter in these states, we must ensure that the counter can find its way back to a valid state.

Dealing with Unused States

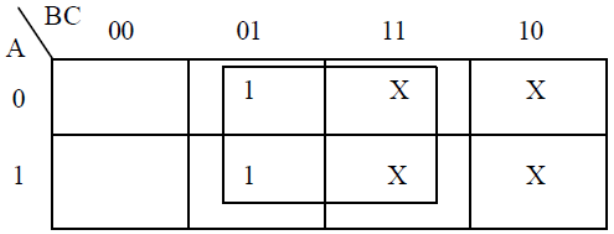
- Self-correcting counter
  - Ensure that when a counter enter one of its unused states, it eventually goes into one of the valid states after one or more clock pulses so it can resume normal operation.
  - Analyze the counter to determine the next state from an unused state after it is designed
  - If the unused states are assigned specific next states, this ensures that the circuit is self correcting by design
  - An alternative design could use additional logic to direct every unused state to a specific next state.
- Design your counters to be self-starting
  - Draw all states in the state diagram
  - Fill in the entire state-transition table
  - May limit your ability to exploit don't cares
    - Choose startup transitions that minimize the logic

Counters with unused states

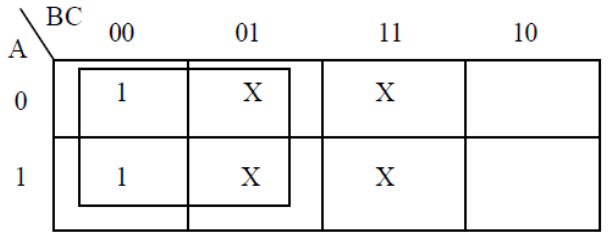
State Table for Counter

Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

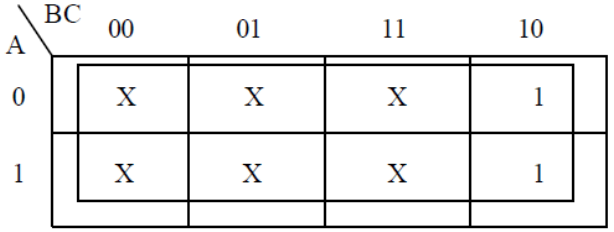
K-Maps for JK Flip Flop Inputs



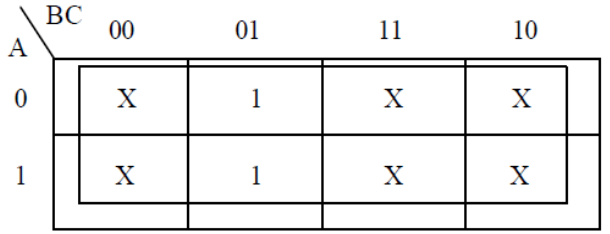
$J_B = C$



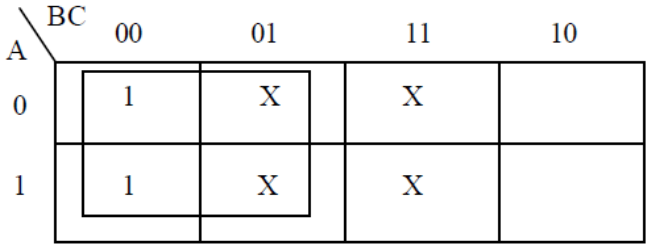
$J_C = B'$



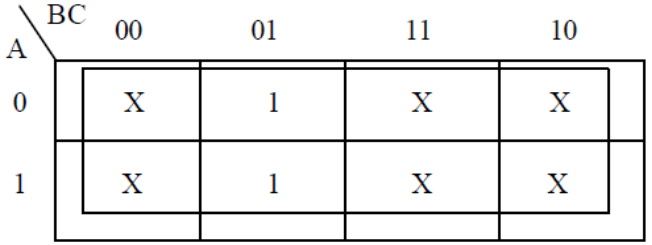
$K_B = 1$



$K_C = 1$

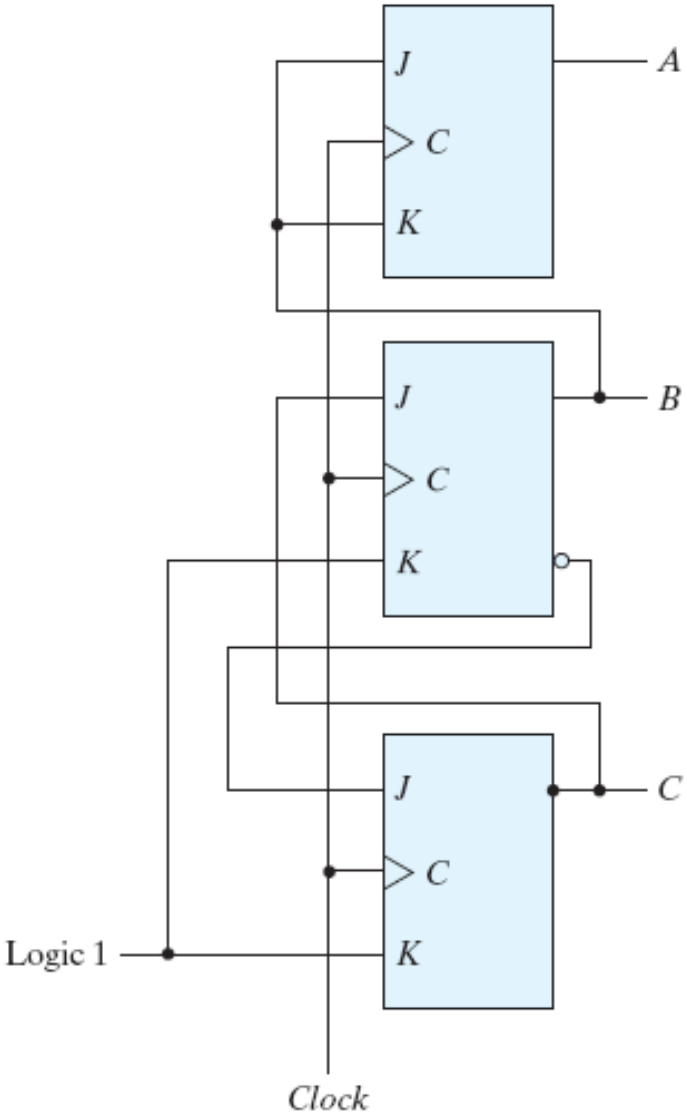


$J_C = B'$



$K_C = 1$

Counter with unused states

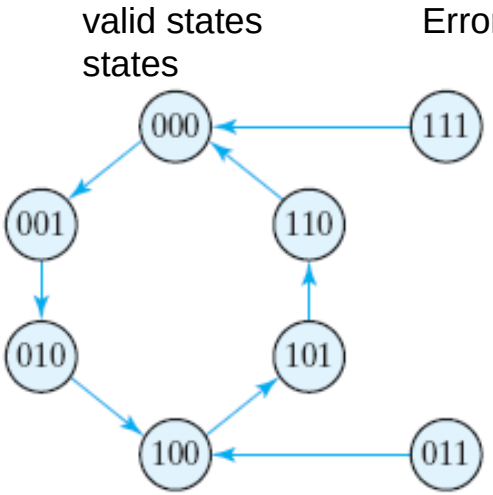


(a) Logic diagram

State Table for Counter

Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

$$J_A = B \qquad K_A = B$$
$$J_B = C \qquad K_B = 1$$
$$J_C = B' \qquad K_C = 1$$

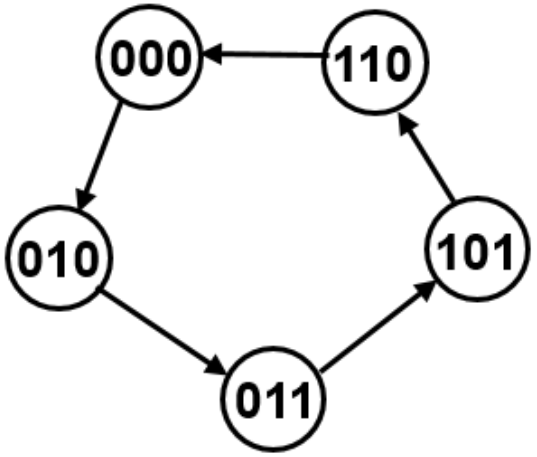


(b) State diagram

Example: 5-state counter

- Counter repeats 5 states in sequence
  - Sequence is 000, 010, 011, 101, 110, 000

Step 1: State diagram



Step 2: State transition table

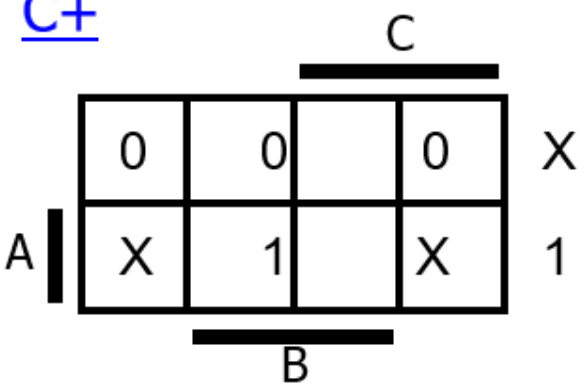
Assume D flip-flops

Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X	X	X

Example: 5-state counter

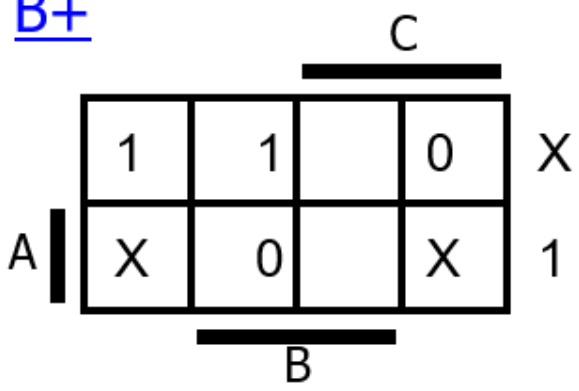
Step 3: Encode next state functions

C+



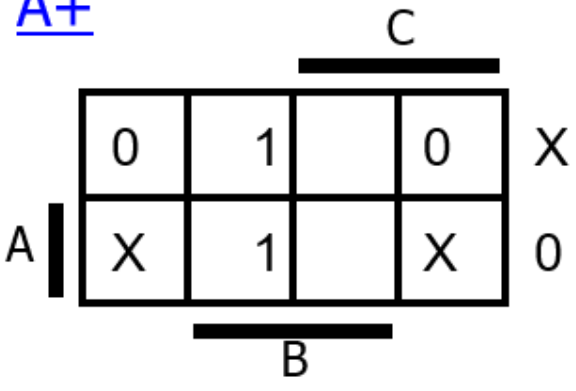
C+ = A

B+



B+ = B' + A'C'

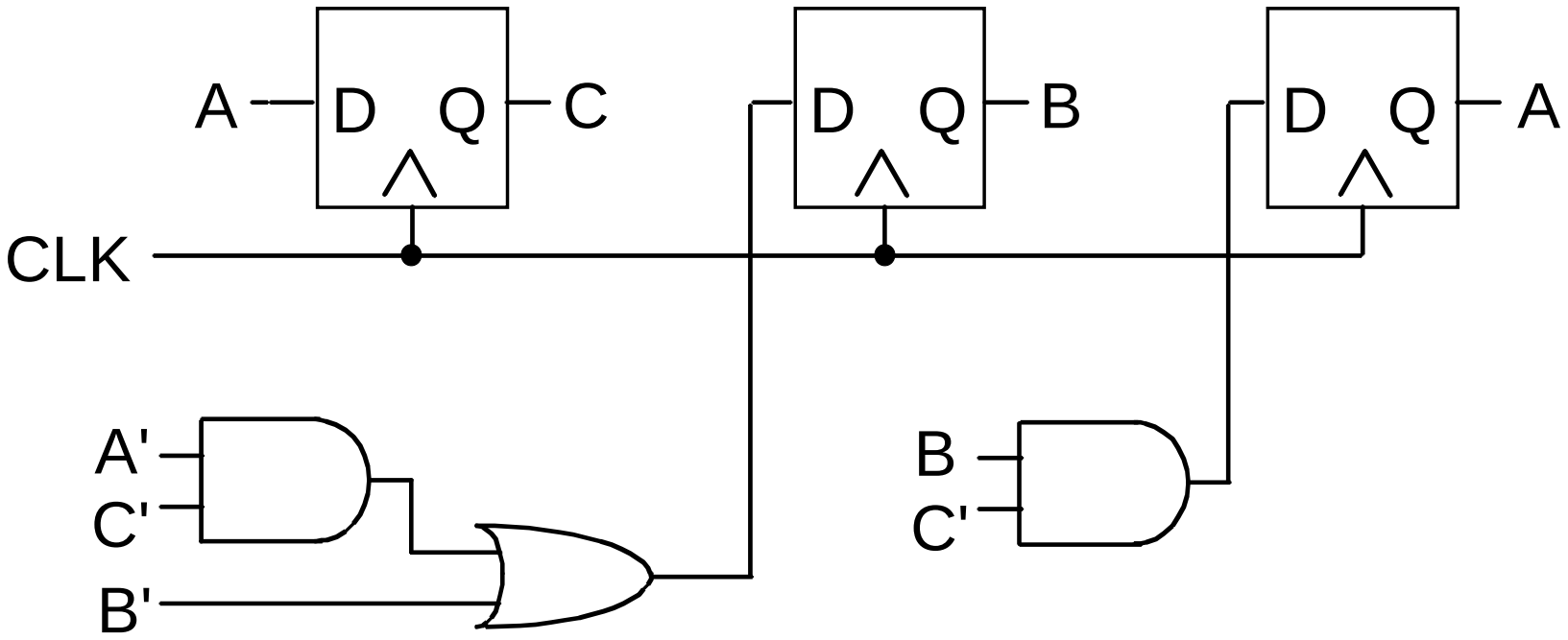
A+



A+ = BC'

Example: 5-state counter

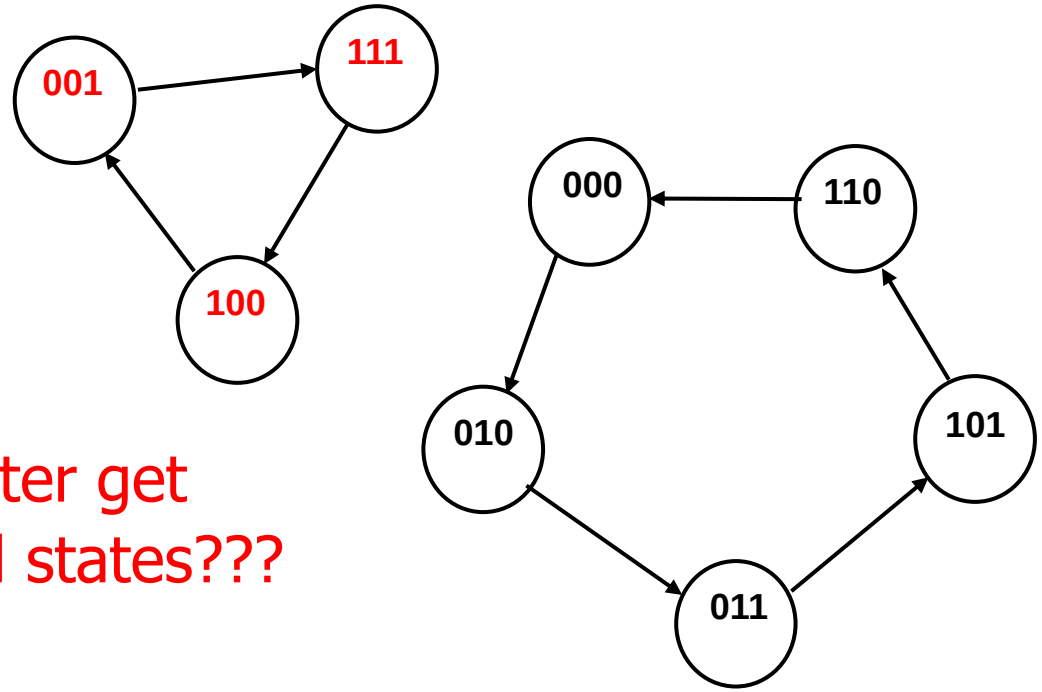
Step 4: Implement the design



Recall that a D flip flop also produces Q' so A', B', and C' would all be available without any extra inverters

Is our design robust?

- What if the counter starts in a 111 state?



Does our counter get stuck in invalid states???

5-state counter

- Back-annotate our design to check it

Fill in state transition table

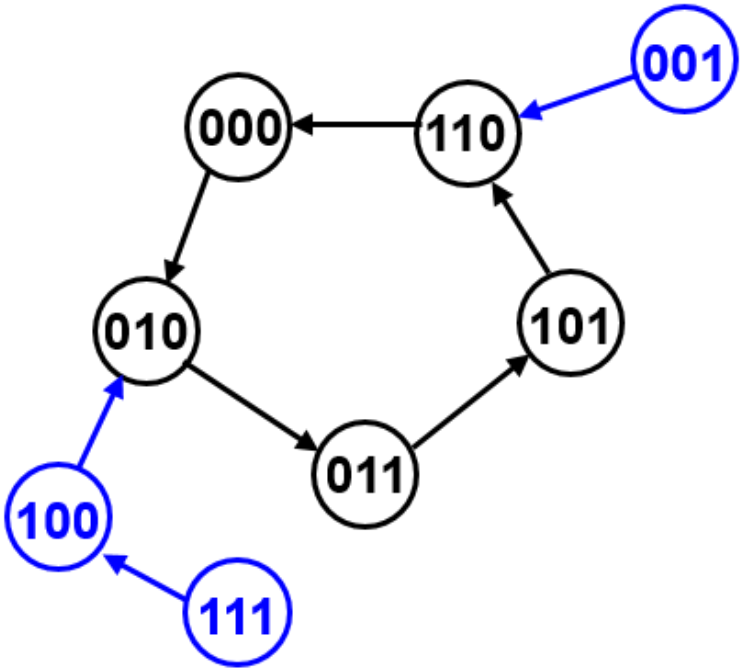
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	1	0	0

A+ = BC'

B+ = B' + A'C'

C+ = A

Draw state diagram



The proper methodology is to ***design*** your counter to be self-starting