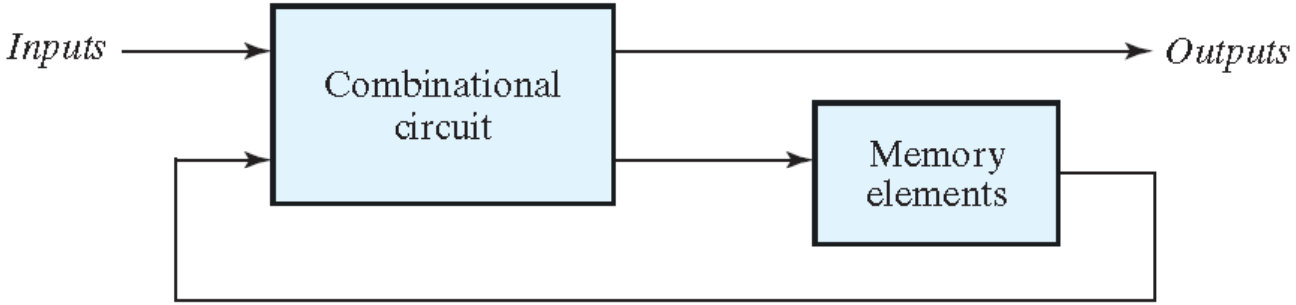
Chapter 5: Synchronous Sequential Logic

Introduction

- All digital systems contain *memory* components that can store information.
- Combinational circuits
  - contains no memory elements
  - the outputs depends on the inputs
- Sequential circuits, however, act as storage elements and have memory.
  - to store, retain, and then retrieve information when needed at a later time.
- Block diagram of a sequential circuit:



  - A combinational circuit with memory elements forming a feedback path.
  - The binary information stored in memory defines the *state*.
  - *Outputs* are determined by *Inputs* and present state.
  - Next state is also determined by *Inputs* and present state.

Synchronous vs. Asychronous

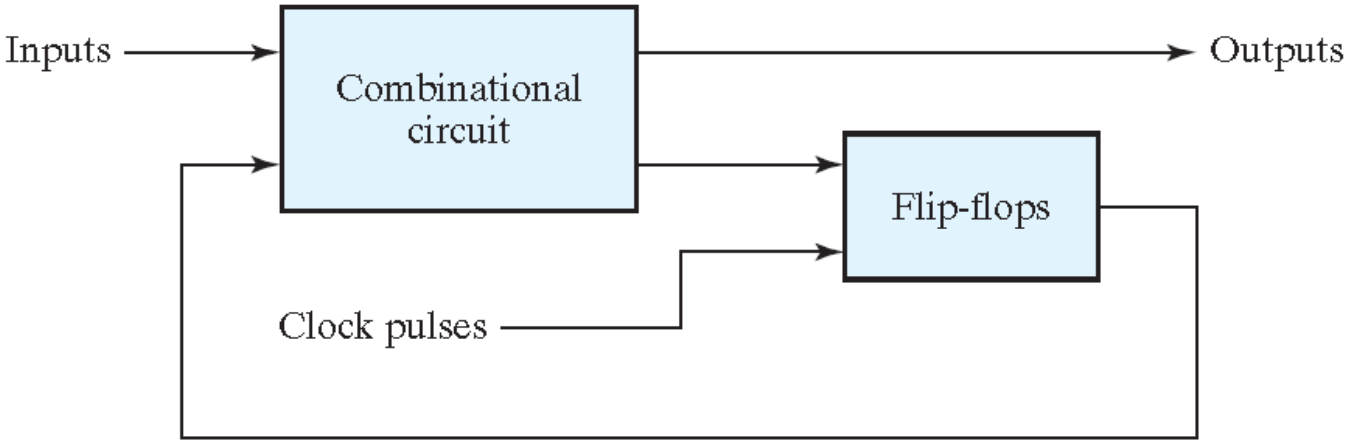
- There are two main types of sequential circuits: *synchronous* and *asynchronous*.
- The behavior of a *synchronous* sequential circuit can be defined from the knowledge of its signals at discrete instants of time.
- The behavior of an *asynchronous* sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change.
- The storage elements commonly used in asynchronous sequential circuits are time-delay devices. Thus, an asynchronous sequential circuit may be regarded as a combinational circuit with feedback (no actual storage elements used).
- Asynchronous sequential circuit may become unstable at times, imposing many difficulties on the designer, and will not be covered in this text.

Clocked Sequential Circuits

- Synchronous sequential circuits affect the storage elements at only discrete instants of time.
- A *clock generator* performs the synchronization, which provides a clock signal having a periodic train of clock pulses, commonly denoted as *clock* or *clk* .
- The storage elements are affected only with the arrival of each clock pulse.
- In practice, the clock pulses determine *when* computational activity will occur within the circuit, and other signals (external inputs and otherwise) determine *what* changes will take place affecting the storage elements and the outputs.
- Synchronous sequential circuits that use clock pulses to control storage elements are called *clocked sequential circuits* and are the type most frequently encountered in practice; also called *synchronous circuits* because the activity within the circuit and the resulting updating of stored values is synchronized to the occurrence of clock pulses.
- The design of synchronous circuits is feasible because they seldom manifest instability problems and their timing is easily broken down into independent discrete steps, each of which can be considered separately.

Filp-Flops as Storage Elements

- *Flip-flops* (*FF*), 1-bit memory, are used as the storage elements.
- A sequential circuit may use many flip-flops to store as many bits as necessary.
- The output of a flip-flop is either 0 or 1 (two *states*).
- The *outputs* (and next *states*) are combinational logic function of the inputs to the circuit and/or the values stored in the flip-flops.
- The new value is stored (updated) in flip-flop when the clock pulse occurs.



(a) Block diagram

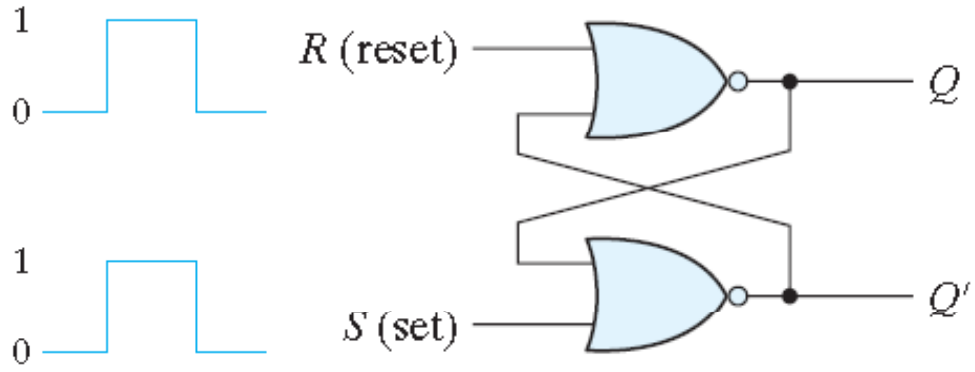
(b) Timing diagram of clock pulses

Clock Synchronization

- The next value of the flip-flop must have reached a stable value before the occurrence of the clock pulse, consequently, the combinational logic must respond to a change in the state of the flip-flop in time to be updated before the next clock pulse arrives.
- Propagation delays of the combinational logic determines the minimum interval between clock pulses to allow the circuit to operate correctly.
- The state of the flip-flops changes only by a clock pulse transition—for example, when the value of the clock signals changes from 0 to 1 (positive edge).
- If the clock pulse is not active, the input and output of the flip-flop is effectively isolated; flip-flop can be regarded as two *gates* controlled complementarily.
- Thus, the transition from one state to the next occurs only at predetermined intervals dictated by the clock pulses, that is so called synchronization.
- Storage elements that operate with *signal levels* (rather than signal transitions) are referred to as *latches*; those controlled by a *clock transition* are *flip-flops* .
- Latches are said to be *level sensitive* devices; flip-flops are *edge-sensitive* ones.
- For design simply and function correctly, use flip-flops as possible as you can.

5-3 Latches

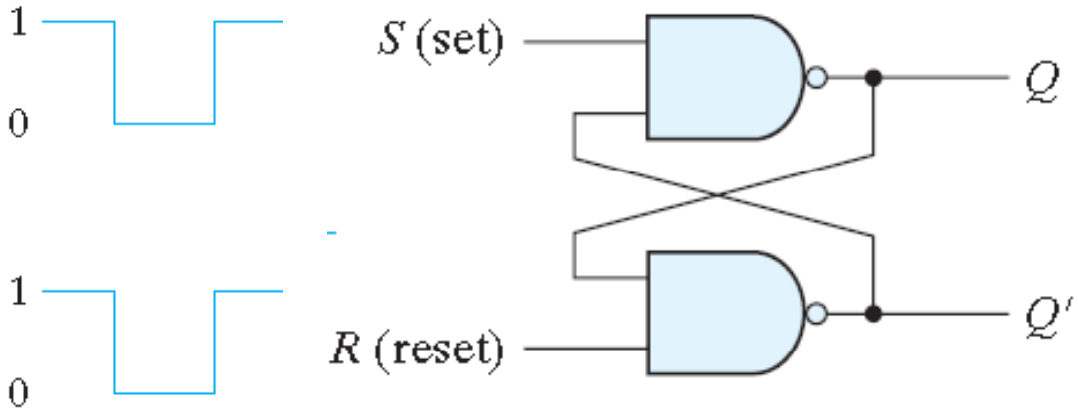
- The *SR* latch is a circuit with two cross-coupled NOR gates (or two cross-coupled NAND gates) and two inputs labeled with *S* for set and *R* for reset.
- When output *Q* = 1 and *Q'* = 0, the latch is said to be in the ***set*** state, and *Q* = 0 and *Q'* = 1, is in the ***reset*** state.
- Forbidden state:
  - Inputs *Q* and *Q'* are normally the complement of each other
  - When both inputs are 1 at the same time result both outputs equal to 0
  - If both inputs then change to 0 simultaneously, the device will enter an unpredictable or undefined state or a *metastable* state
  - The next state will depend on the order in which *S* and *R* return to 0.



S	*R*	*Q*	*Q'*	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

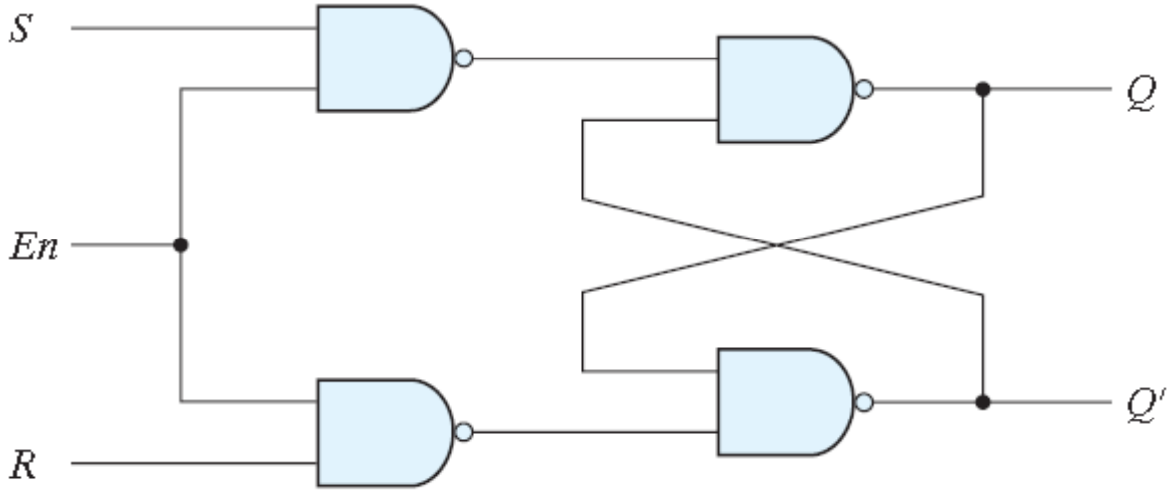
SR Latch with NAND Gates

- Both inputs are normally at 1.
- Input 0 to the *S* (*R*) causes *Q* (*Q'*) to be 1, putting the latch in the *set* (*reset*) state.
- The forbidden condition is both inputs being 0 at the same time.
- The NAND latch is *low* activated (*active low*).



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	(forbidden)

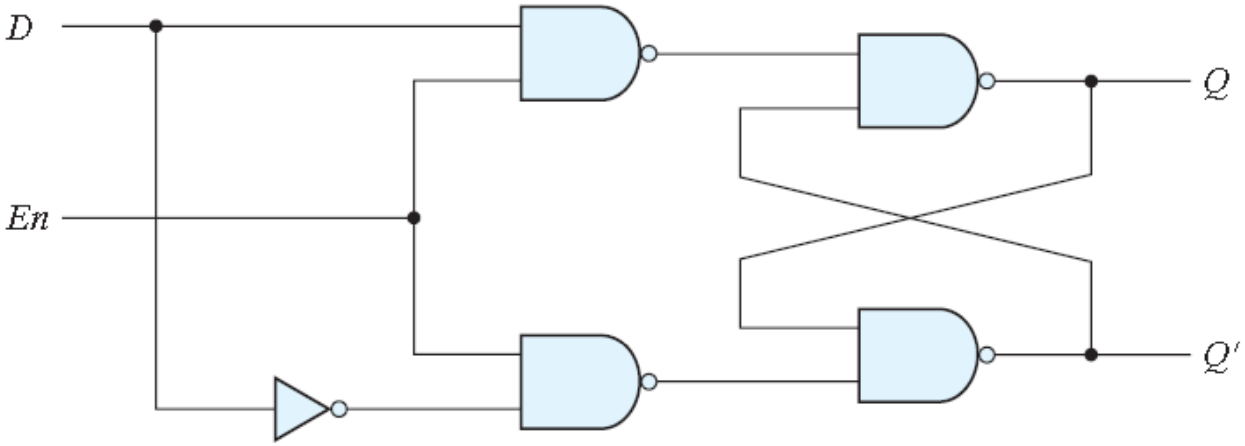
SR Latch with *Enable*



En	*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	Indeterminate

D Latch (Transparent Latch)

- Ensure *S* and *R* are never equal to 1 at the same time to eliminate the undesirable condition of the indeterminate state in the *SR* latch.
- As *En* is at 0, the cross-coupled *SR* latch has both inputs at 1 and the circuit cannot change state regardless of the value of *D* .
- *Transparent*: as *En* = 1, *Q* (*Q'*) follows the change of *D*
- When *En* transits from 1 to 0, the binary information at *D* at the transition time is retained (i.e., stored) at *Q* until *En* raises to 1 again.



En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

- Characteristic equation: $Q(t+1) = D$

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

Trigger

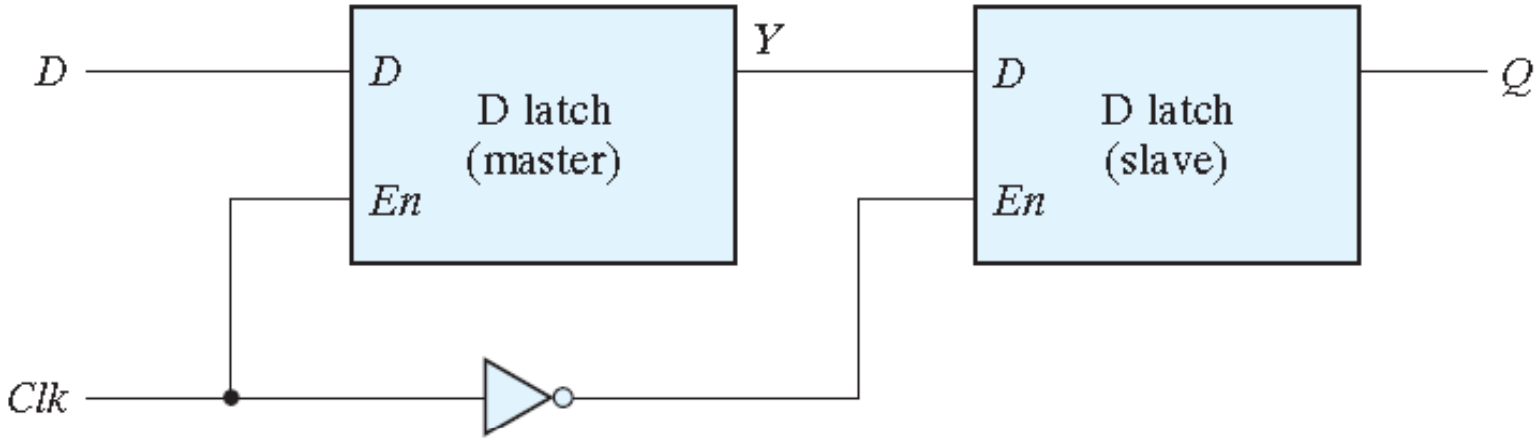
- Trigger: a latch or flip-flop is switched by a change of the control input
  - Level triggered – latches
  - Edge triggered – flip-flops

(a) Response to positive level (b) Positive-edge response (c) Negative-edge response

- The transparent latch (level sensitive) may fail due to a *race* condition.
  - The state of a latch changes as soon as the clock changes to 1, and the new state appears at the output while the clock is still active (due to transparent).
  - This output may race through the combinational circuit to the latch input.
  - If the clock is still active, the latch will respond to the new value again and a new output state may occur resulting an unpredictable situation.
  - To avoid such condition, the output of a latch cannot be applied directly or through combinational logic to the input of the same or another latch when all the latches are triggered by a common clock source.

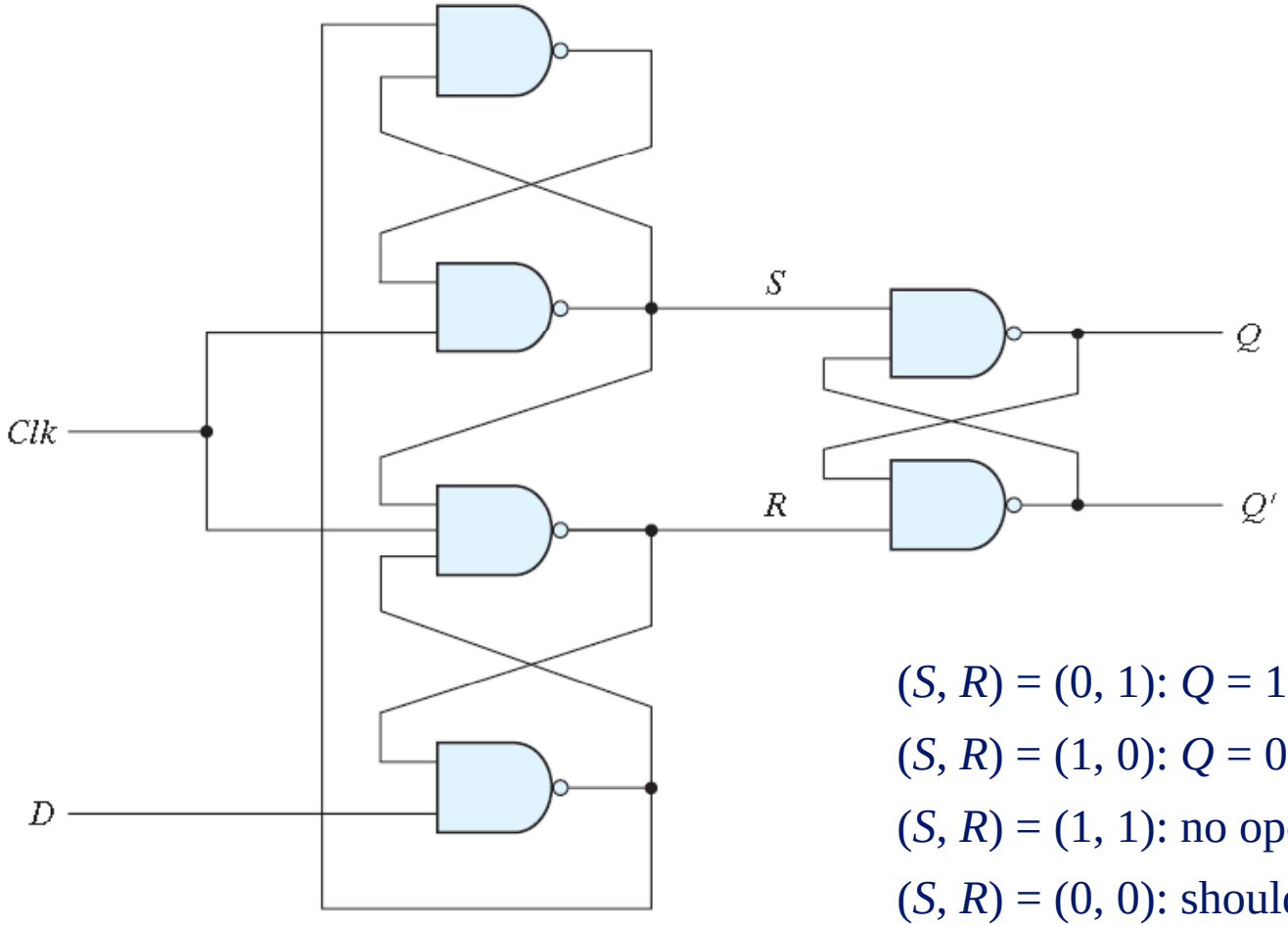
Master–slave *D* Flip-flop

- *D* flip-flop consists of two *D* latches and an inverter, the first latch is called the *master* and the second the *slave*.
- The two latches are controlled (enabled) complementarily; they flip and flop alternatively.
- Samples *D* and changes *Q* only at the *negative* edge of the clock (*Clk*), the transition of the clock from 1 to 0.
- Positive edge triggered *D* flip-flop can also be constructed by adding an inverter to the *Clk* input.



D-type Positive-edge-triggered Flip-flop

- Two latches respond to the *D* (data) and *Clk* (clock) inputs, and the third latch provides the outputs for the flip-flop.



(S, R) = (0, 1): Q = 1

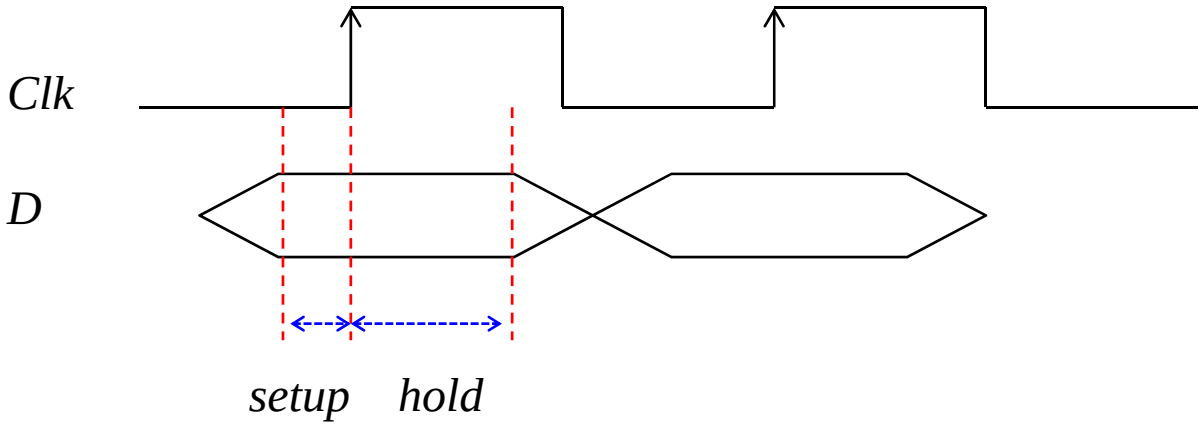
(S, R) = (1, 0): Q = 0

(S, R) = (1, 1): no operation

(S, R) = (0, 0): should be avoided

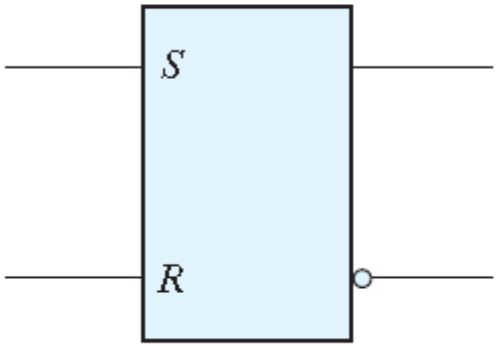
Timing Parameters

- ***Setup*** time
  - *D* input must be maintained at a constant value prior to the application of the positive-edge of *Clk* pulse (rise)
  - equal to the propagation delay through gates 4 and 1
  - data to the internal latches
- ***Hold*** time
  - *D* input must not changes after the application of the positive *Clk* pulse
  - equal to the propagation delay of gate 3
  - clock to the internal latch

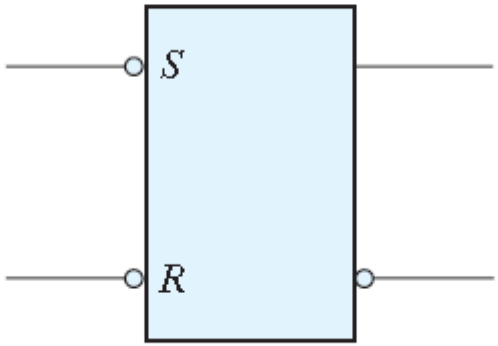


Graphic Symbols

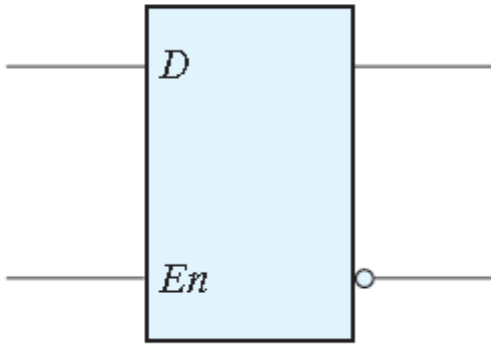
- Latch



SR

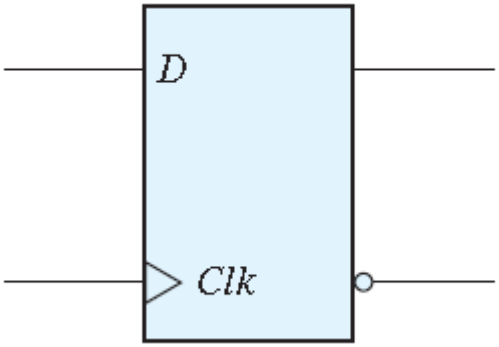


$\overline{S}\overline{R}$

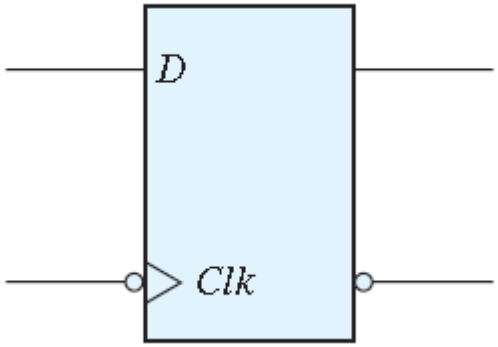


D

- Edge-triggered D flip-flop



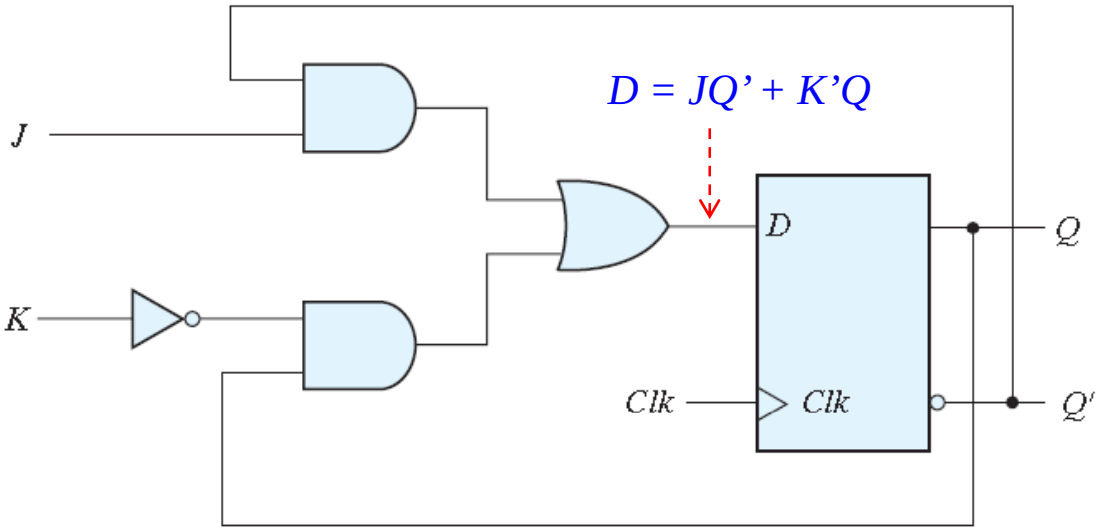
(a) Positive-edge



(a) Negative-edge

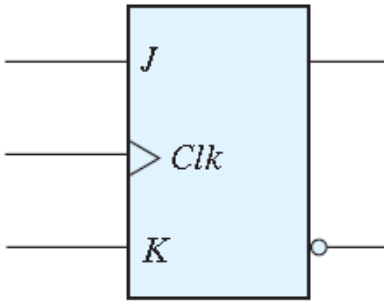
JK Flip-Flop

- Edge-triggered *D* flip-flop requires the smallest number of gates, and is the most economical and efficient flip-flop constructed in VLSI IC design.
- Other types of flip-flops can be constructed by *D* flip-flop and external logic.
- *JK* and *T* flip-flops are two other less used flip-flops.
- *JK* flip-flop:



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



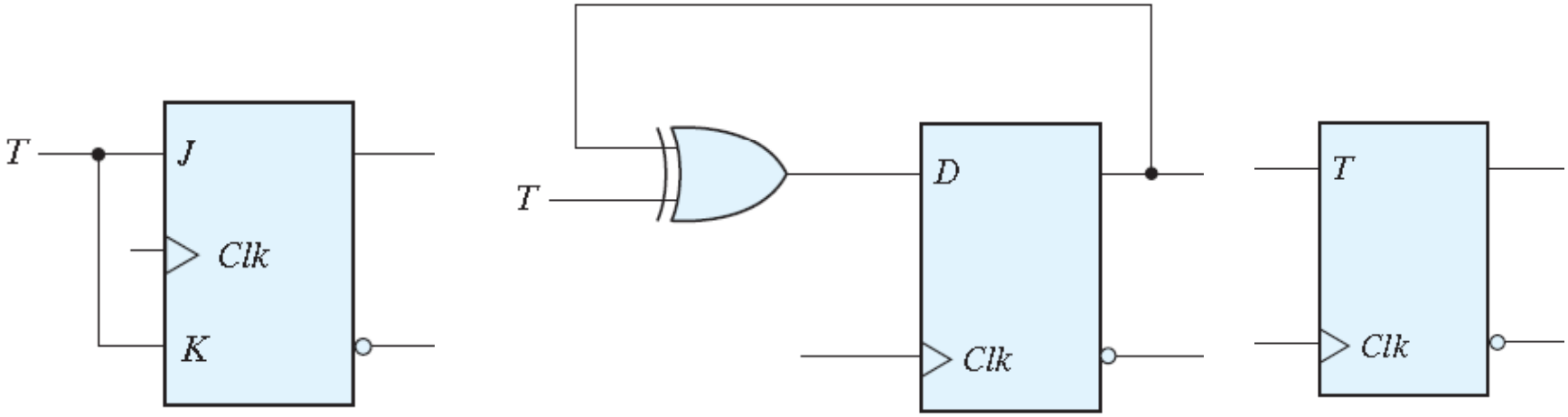
- Characteristic equation: $Q(t+1) = JQ' + K'Q$

T Flip-Flop

- *T* (toggle) flip-flop can be obtained from a *JK* flip-flop with *J* and *K* tied together.
- Can also be constructed with a *D* flip-flop and an exclusive-OR gate.

  $D = T \oplus Q = TQ' + T'Q$ ➔ Characteristic equation: $Q(t+1) = TQ' + T'Q$

- Useful for designing binary counters.

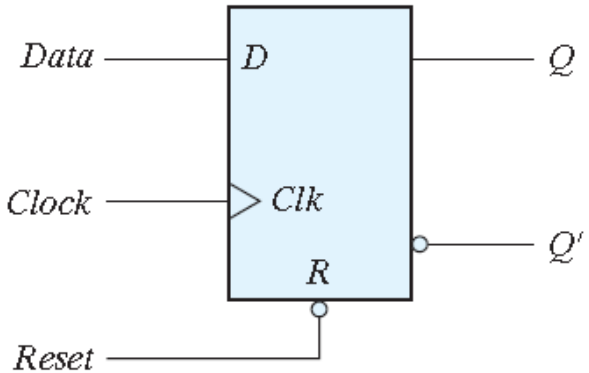


T	Q(t + 1)	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

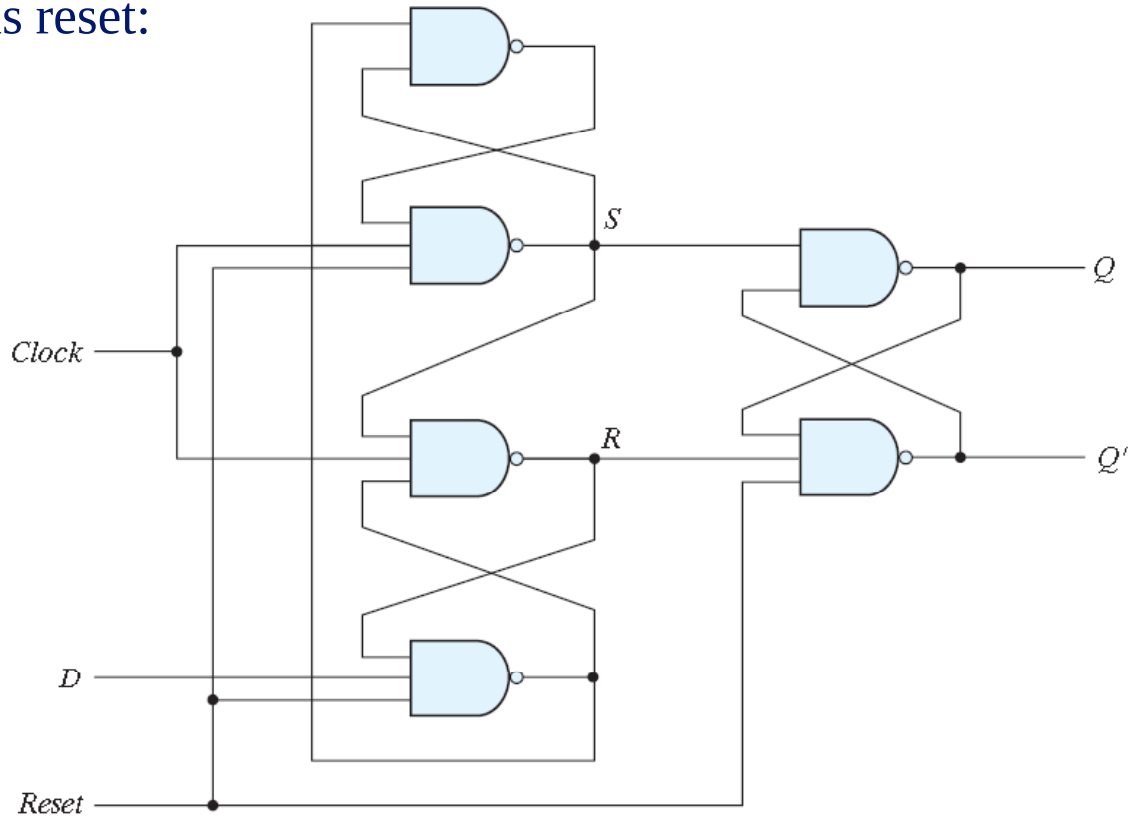
D	Q(t + 1)	
0	0	Reset
1	1	Set

Asynchronous Inputs

- The state of the flip-flops is unknown when power is turned on.
- Asynchronous inputs are used to force the flip-flop to a known starting state (*initialization*) independently of the clock.
- ***Preset*** or *direct set* sets the flip-flop to 1.
- ***Clear*** or *direct reset* clears the flip-flop to 0.
- *D* flip-flop with asynchronous reset:



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



5-5 Analysis of Clocked Sequential Ckts

- Analysis describes what a circuit will do under certain operating conditions.
- For clocked sequential circuits, the outputs and the next state are both a function of the inputs and the present state.
- The analysis of a sequential circuit consists of obtaining a ***state table*** or a ***state diagram*** for the time sequence of inputs, outputs, and internal states.
- Boolean expressions can also describe the behavior of the sequential circuit.
- A logic diagram is recognized as a clocked sequential circuit if it includes flip-flops with clock inputs.
- The flip-flops may be of any type, and the logic diagram may or may not include combinational logic gates.

State Equations

- The behavior of a clocked sequential circuit can be described algebraically by means of *state equations*; also called transition equations, specifies the next state as a function of the present state and inputs.
- Example: 0-detector

two *D* flip-flops *A* and *B*
an input *x* and an output *y*

State equations:

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

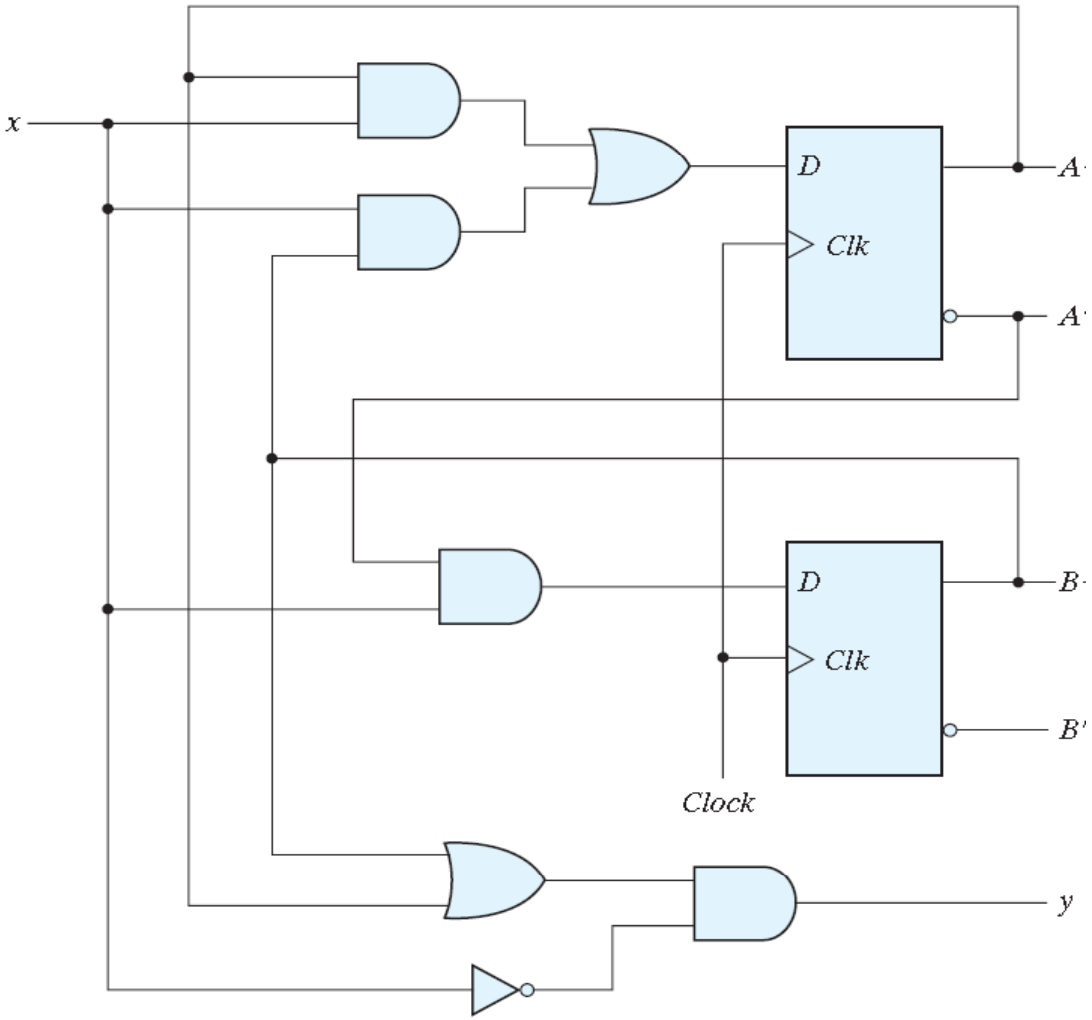
$B(t + 1) = A'(t)x(t)$

t: Present time

t+1: Next time

Output:

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



State Table

- *State table* (also called a *transition table*) enumerates the time sequence of inputs, outputs, and flip-flop states.
- The table consists of four labels: *present state*, *input*, *next state*, and *output*.
- List all possible binary combinations of present states and inputs.

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

similar to a truth table

State equations are derived as:

$A(t + 1) = Ax + Bx$

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

Output equation:

$y = (A + B)x'$

Also can be expressed with flip-flop input equations:

$D_A = Ax + Bx$

$D_B = A'x$

$y = (A + B)x'$

Second Form of State Table

- In general, a sequential circuit with *m* flip-flops and *n* inputs needs 2^{m+n} rows in the state table.
- A second from of state table uses only three labels: present state, next state, and output; and the input conditions are enumerated under the next-state and output sections.

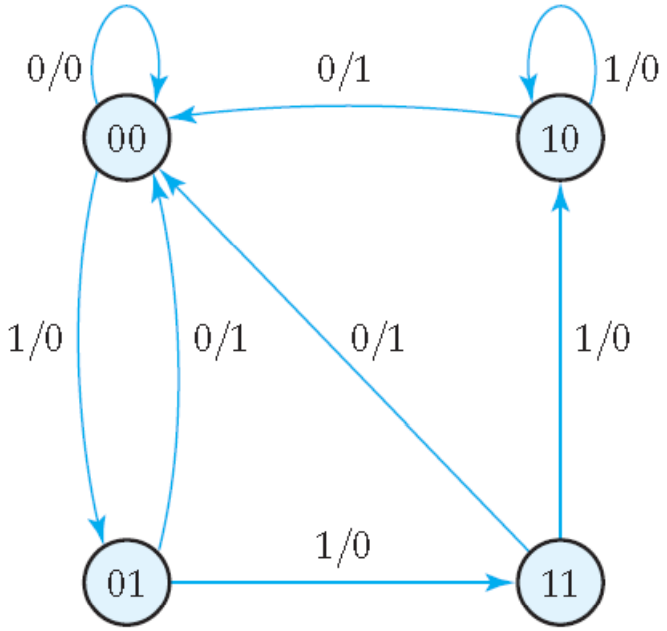
Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	*B*	*A*	*B*	*A*	*B*	*y*	*y*
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

similar to a K-map

State Diagram

- Graphical representation of a state table
- Well matched with the second form of state table
- Each circle represents an assigned state
- Directed lines, indicate a state transition, are labeled with *input*/*output*
- In this example, every circle (state) has two outgoing directed lines to other circles
- A directed line connecting a circle with itself indicates that no change of state occurs.

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



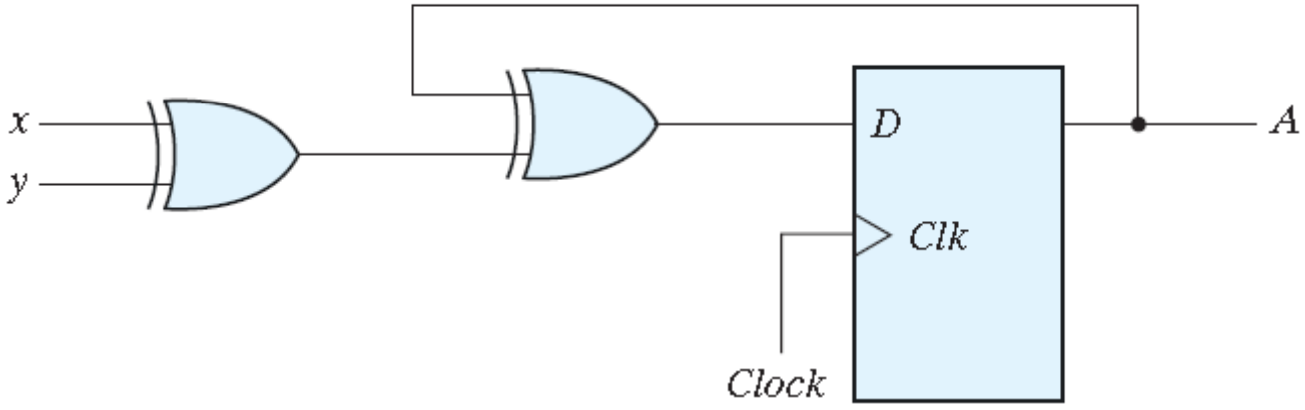
- The analysis steps are summarized as:

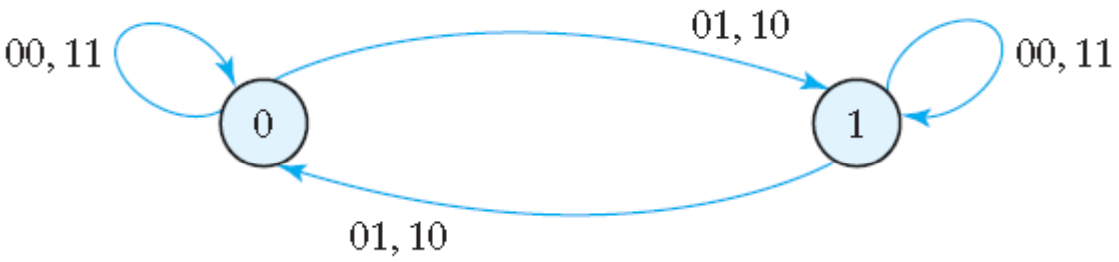
  circuit → equations → state table → state diagram

Analysis with *D* Flip-flops

- A sequential circuit with state equation: $D_A = A \oplus x \oplus y$

  D_A: *D* flip-flop with output *A*; *x* and *y*: inputs; and no output given.
- For a *D* flip-flop, the state equation is the same as the input equation.
- One flip-flop has two states.
- Two inputs have four possible combinations for each state.





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

Analysis with *JK* Flip-flops

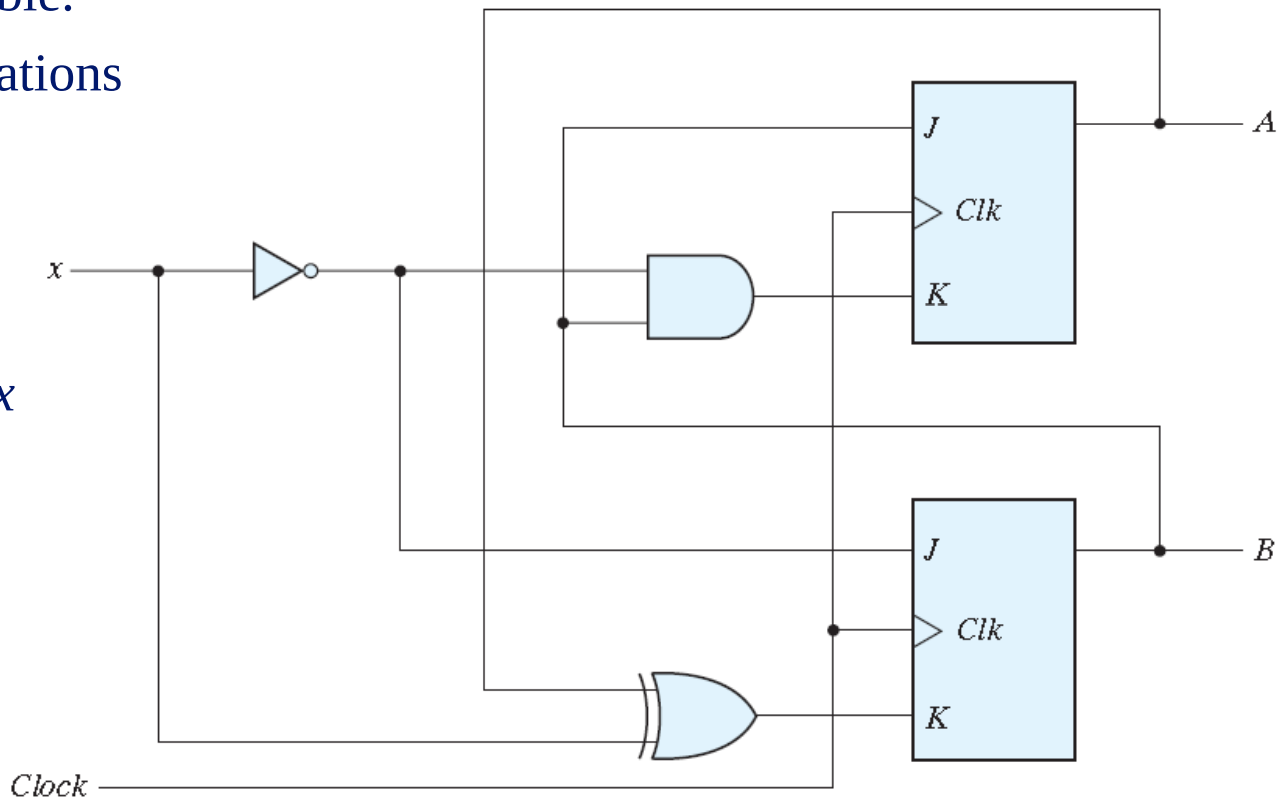
- The next-state values of *JK* or *T* flip-flops can be derived as follows:
  1. Determine the flip-flop input equations in terms of the present state and input variables.
  2. List the binary values of each input equation.
  3. Use the corresponding flip-flop characteristic table to determine the next-state values in the state table.
- The flip-flop input equations

$J_A = B$

$K_A = Bx'$

$J_B = x'$

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



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

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

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

- The above equations determine the flip-flop inputs to derive the next state

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

- Or, derive the state equations using characteristic eq.

Using Characteristic Equations

- Characteristic equation of *JK* FF: $Q(t+1) = JQ' + K'Q$
- So for the two *JK* FFs $A(t+1) = J_A A' + K_A' A, \quad B(t+1) = J_B B' + K_B' B$
- Substituting the values of J_A, K_A, J_B, and K_B

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

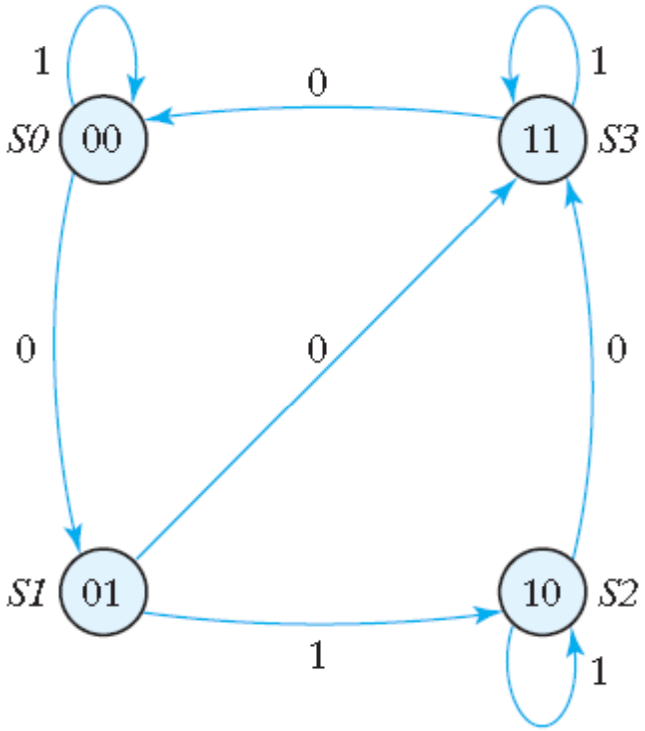
  $J_B = x', \quad K_B = A'x + Ax' = A \oplus x$

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

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

  The "Next state" can be derived from the above two equations.

- State diagram:



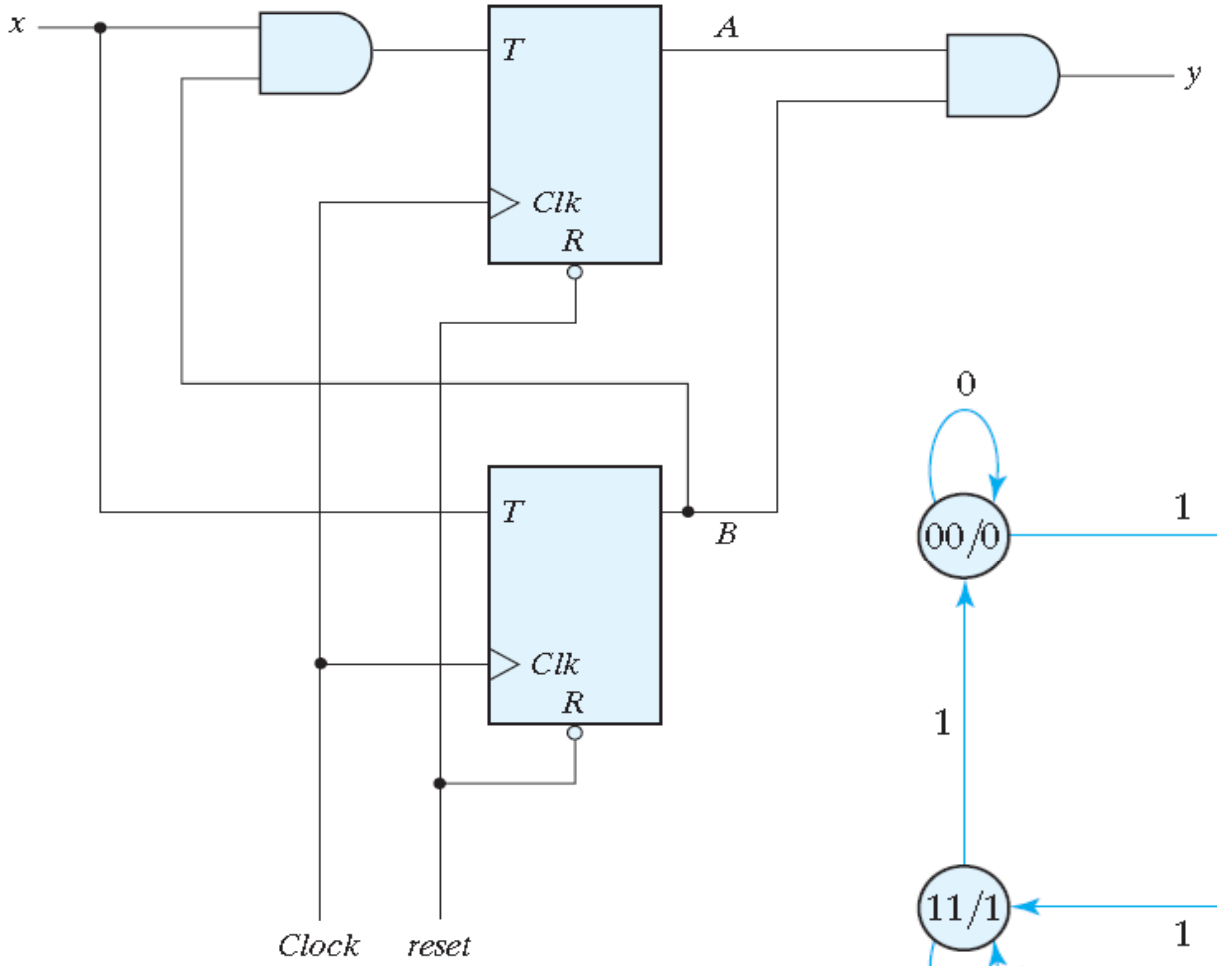
Analysis with *T* Flip-flops

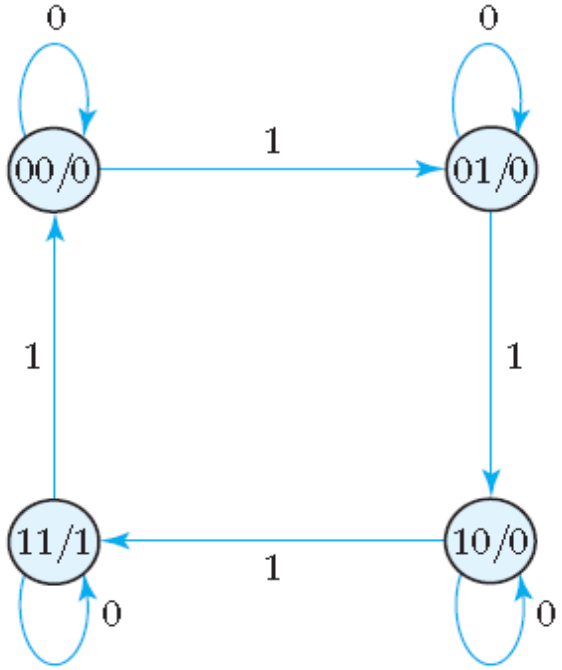
- Example: two *T* flip-flops *A* and *B*, one input *x*, and one output *y*
- Two input equations and an output equation:

$T_A = Bx$

$T_B = x$

$y = AB$





- Input equations and an output equation:

  $T_A = Bx, \quad T_B = x, \quad y = AB$

- Characteristic equation of T flip-flops: $Q(t + 1) = T \oplus Q = T'Q + TQ'$
- The values for the next state

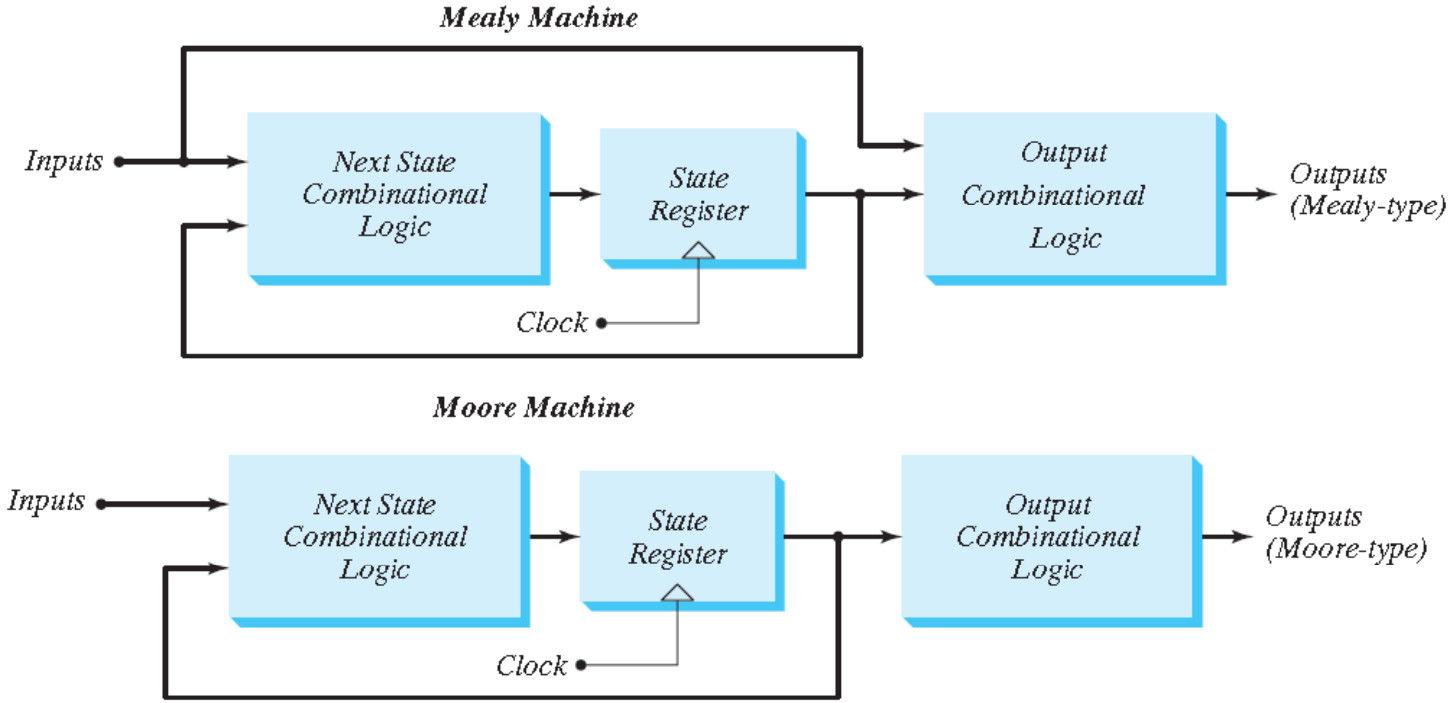
  $A(t + 1) = (Bx)'A + (Bx)A' = AB' + Ax' + A'Bx$

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

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

Finite State Machines (FSM)

- A sequential circuit has inputs, outputs, and internal states.
- Two commonly used finite state machine models of sequential circuits, the ***Mealy*** model and the ***Moore*** model, differing only in the way the output is generated.
- The outputs of *Moore* circuit are synchronized with the clock, depend only on flip-flop outputs that are synchronized with the clock.
- The output of the Mealy machine is the value that is present immediately before the active edge of the clock.

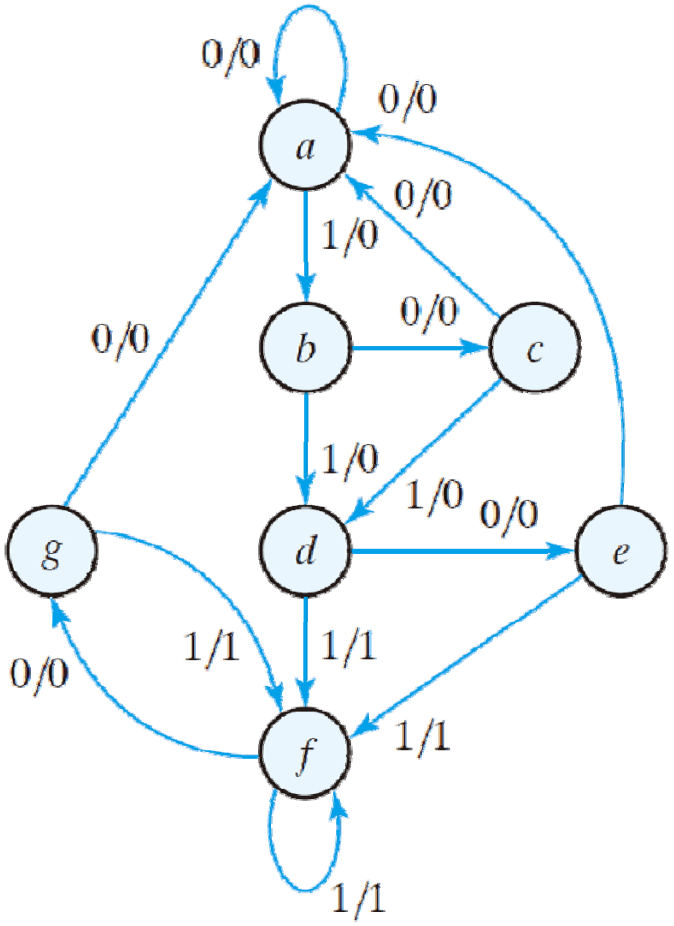


5-7 State Reduction and Assignment

- The *design* (*synthesis*) of a sequential circuit starts from a set of specifications and culminates in a logic diagram.
- Two sequential circuits may exhibit the same input–output behavior (function), but have a different number of internal states in their state diagram.
- The current section discusses certain properties of sequential circuits that may simplify a design by reducing the number of gates and flip-flops it uses.
- In general, reducing the number of flip-flops reduces the cost of a circuit.
- *State-reduction*, reducing the number of states in a state table, while keeping the external input–output requirements unchanged, can reduce the number of flip-flops used in a sequential circuit.
- Since m flip-flops produce 2^m states, a reduction in the number of states may (or may not) result in a reduction in the number of flip-flops.
- Reducing the number of flip-flops sometimes results the equivalent circuit with fewer flip-flops but more combinational gates to realize its next state and output logic.

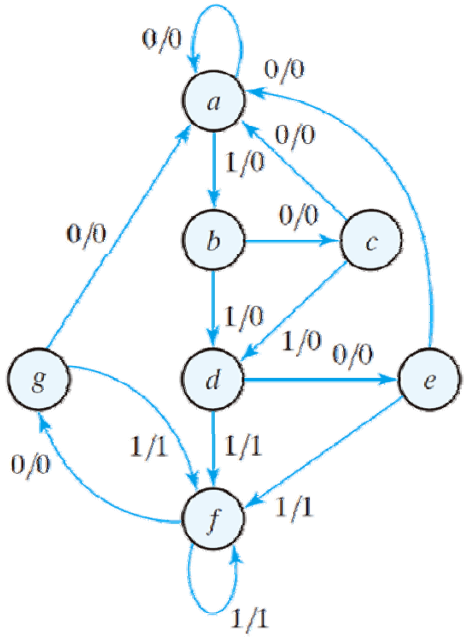
State Reduction Example

- Two circuits are ***equivalent*** if identical input sequences are applied to the two circuits and identical outputs occur for all input sequences, then one may be replaced by the other.
- State reduction reduces the number of states in a sequential circuit without altering the input–output relationships.
- Only the input-output sequences are important in this example.
- Consider the input sequence 01010110100 starting from the initial state ***a***.
- Complete the sequence to get the follows:



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 table is more convenient for state reduction than a diagram.
- State reduction algorithm: "*Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state.*"
- When two states are equivalent, one of them can be removed without altering the input–output relationships.
- Back to the example:
  - States ***e*** and ***g*** both go to states ***a*** and ***f*** and have outputs of 0 and 1 for $x = 0$ and $x = 1$, respectively.
  - States ***g*** and ***e*** are equivalent, and one of these states can be removed.
  - States ***f*** and ***d*** are also equivalent, so state ***f*** can be removed and replaced by ***d***.



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	

Original 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	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

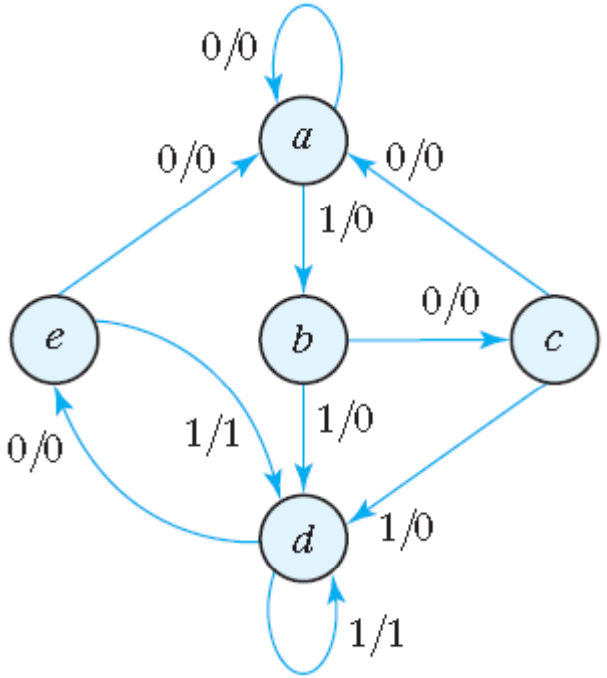
Reducing the State Table

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

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

Reduced State Diagram



State Assignment

- States must be assigned with unique coded binary values to implement the physical components.
- For a circuit with m states, the assigned codes must contain n bits, where $2^n \geqq m$.
- Unused states (codes) are treated as ***don't-care*** conditions during the design.
- Don't-care conditions usually help in obtaining a simpler circuit.
- The simplest way to code states is to use binary counting code or Gray code without guaranteeing a better result.
- One-hot assignment, uses one flip-flop per state, ensures only one bit is equal to 1 while all others are kept at 0, usually leads to simpler decoding logic for the next state and output, results a faster machines, and the silicon area required by the extra flip-flops can be offset by the area saved by using simpler decoding logic.

Binary Assignment

- A different assignment will result in a state table with different binary values for the states.
- The binary form of the state table is used to derive the next state and output -- forming combinational logic part of the sequential circuit.
- The complexity of the combinational circuit depends on the binary state assignment chosen.
- Sometimes, the name *transition table* is used for a state table with a binary assignment.

Reduced State Table with Binary Assignment 1

		Next State		Output	
Present State		$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	000	000	001	0	0
b	001	010	011	0	0
c	010	000	011	0	0
d	011	100	011	0	1
e	100	000	011	0	1

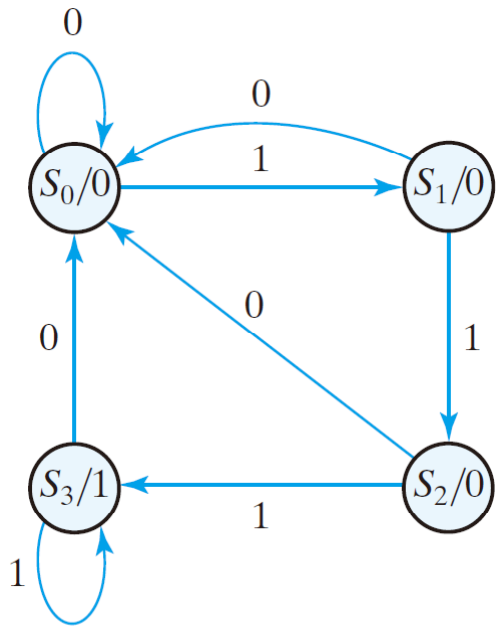
A great many possible binary assignments may exist.

5-8 Design Procedure

- A synchronous sequential circuit is made up of flip-flops and combinational gates.
- The design of the circuit consists of choosing the flip-flops and then finding a combinational gate structure that, together with the flip-flops, produces a circuit which fulfills the stated specifications.
- The design steps for synchronous sequential circuits can be summarized as:
  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.

Synthesis using *D* Flip-flops

- Example: detect a sequence of three or more consecutive 1's in a string of bits coming through an input line.



S_i: i consecutive 1's is detected

S_0: starting state

Moore FSM

- Assign binary codes to the states and list the state table.
- Two *D* FFs (A and B) represent the four states, and one input x and one output y.

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

- The characteristic equation of the *D* flip-flop is $Q(t + 1) = D_Q$
- The flip-flop input equations can be obtained directly from the next-state columns of *A* and *B* and expressed in sum-of-minterms form as

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

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

$y(A,B, x) = \sum(6, 7)$

- The Boolean equations are simplified by k-maps:

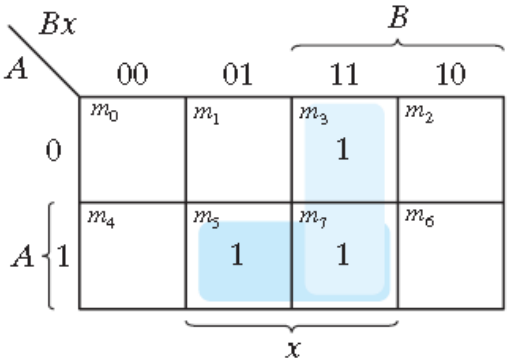
$D_A = Ax + Bx$

$D_B = Ax + B'x$

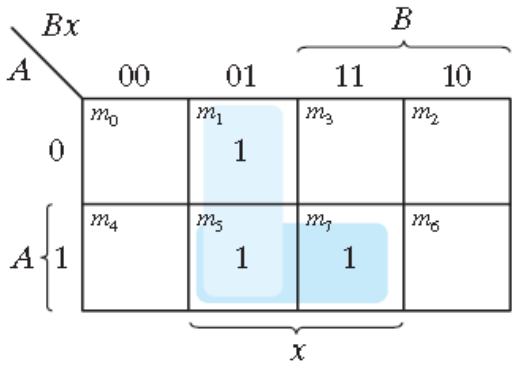
$y = AB$

State Table for Sequence Detector

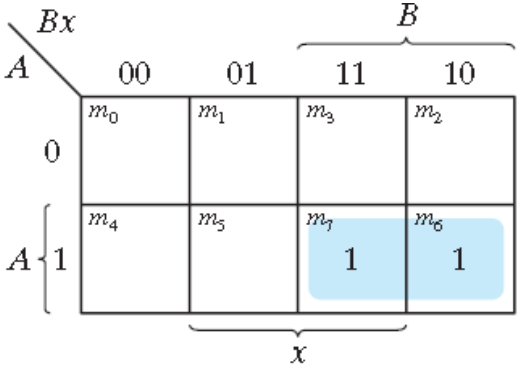
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1



$D_A = Ax + Bx$



$D_B = Ax + B'x$



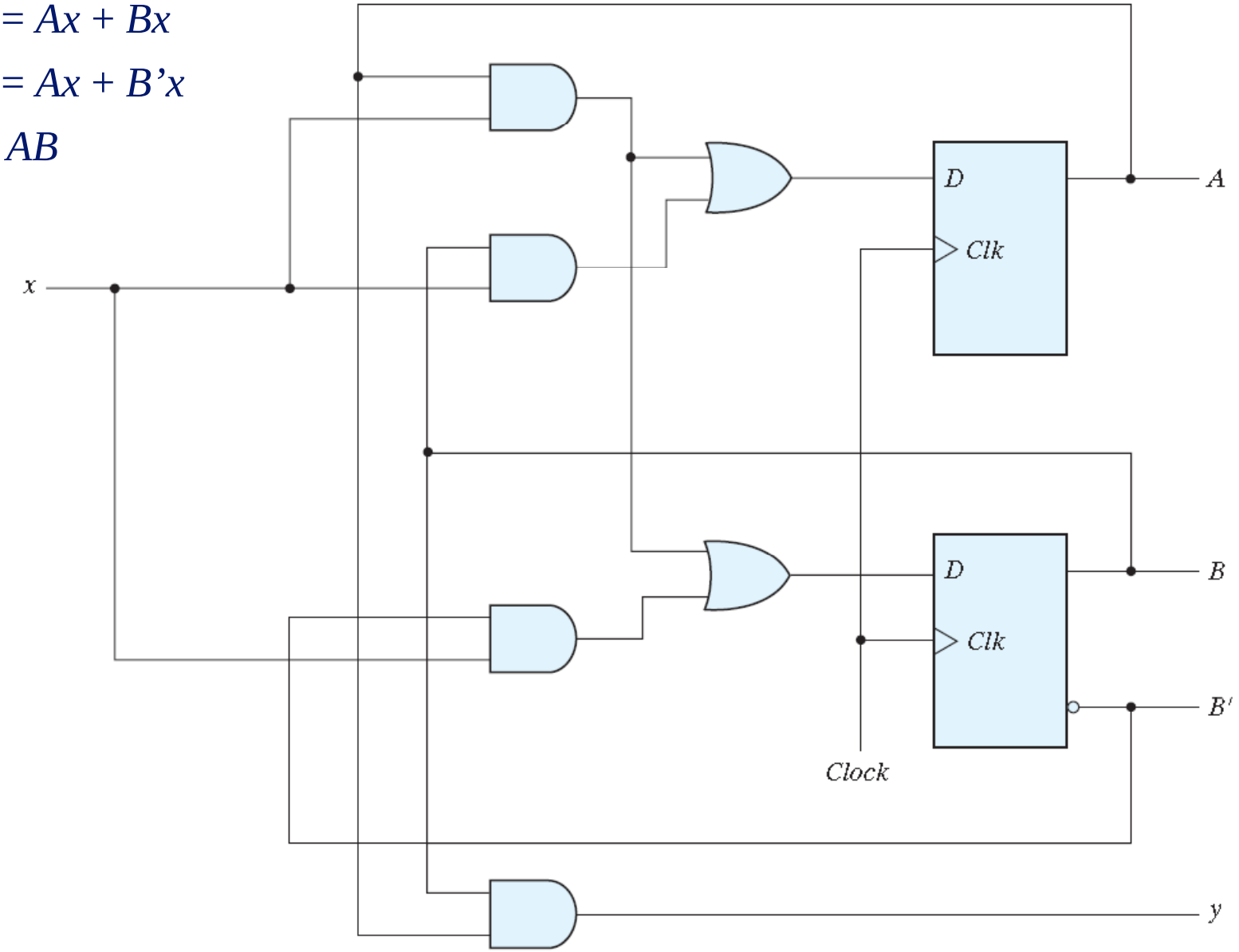
$y = AB$

Logic Diagram of the Sequence Detector

$D_A = Ax + Bx$

$D_B = Ax + B'x$

$y = AB$



Excitation Tables

- The advantage of designing with *D* FFs is that the Boolean equations describing the inputs to the flip-flops can be obtained directly from the state table, the input equations are obtained directly from the next state. This is not the case for the *JK* and *T* types of flip-flops.
- A state diagram $\Rightarrow$ flip-flop input functions
  - straightforward for *D* flip-flops
  - we need *excitation* tables for *JK* and *T* flip-flops
  - a table that lists the required inputs for a given change of state.

Flip-Flop 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

(a) *JK* Flip-Flop

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

(b) *T* Flip-Flop

Synthesis using *JK* Flip-flops

- The same example
- The state table and *JK* flip-flop inputs

Flip-Flop Excitation Tables

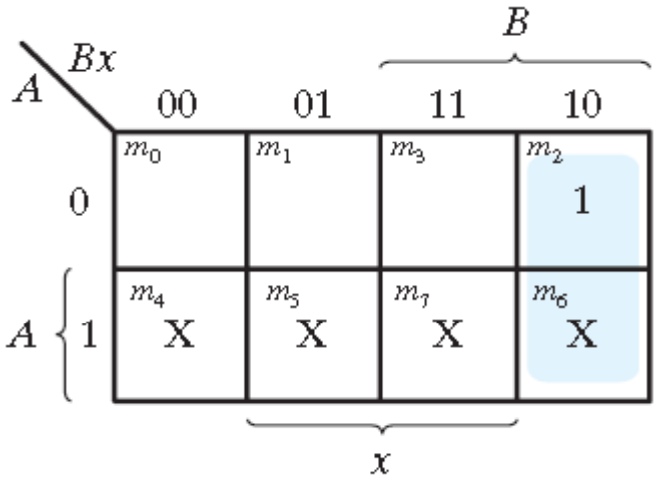
$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(a) *JK* Flip-Flop

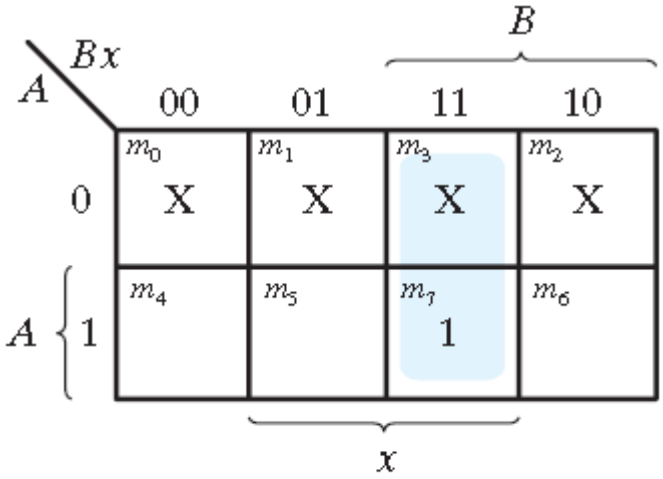
State Table and JK Flip-Flop Inputs

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

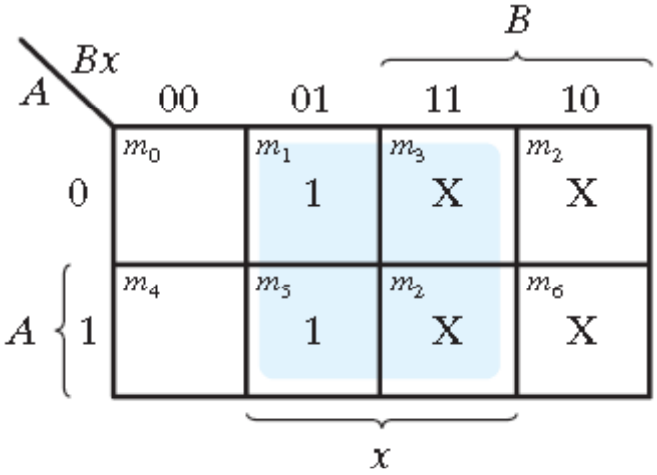
K-Maps for *JK* Input Equations



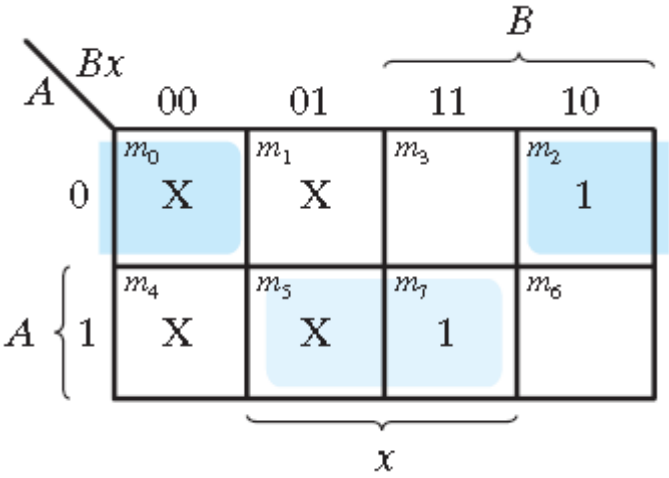
$J_A = Bx'$



$K_A = Bx$



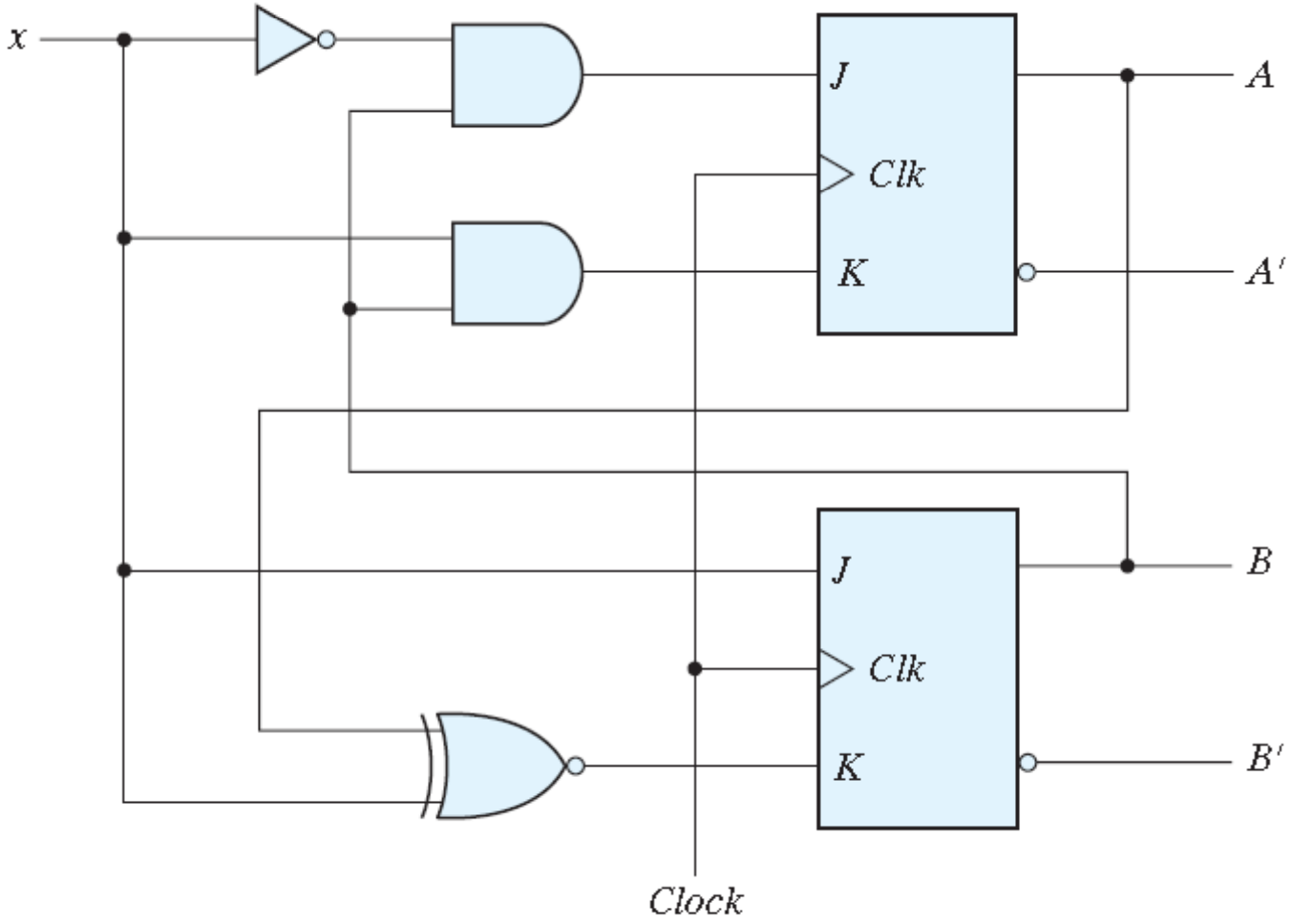
$J_B = x$



$K_B = (A \oplus x)'$

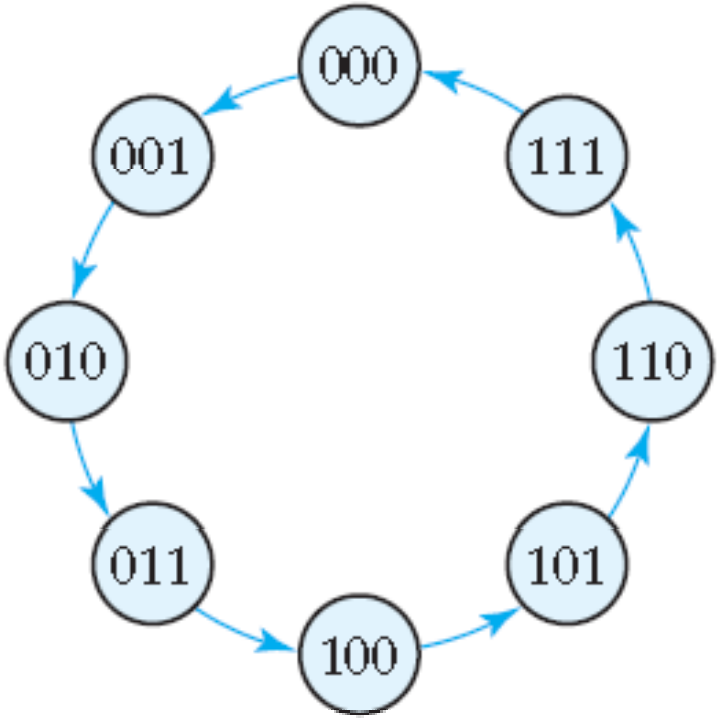
Logic Diagram with *JK* Flip-flops

$J_A = Bx'$ $K_A = Bx$ $J_B = x$ $K_B = (A \oplus x)'$



Synthesis using *T* Flip-flops

- Example: *n*-bit binary counter consists of *n* flip-flops that can count in binary from 0 to 2^{n-1}.
- The state diagram of a 3-bit counter is shown below, the input is the clock and the output is the state.



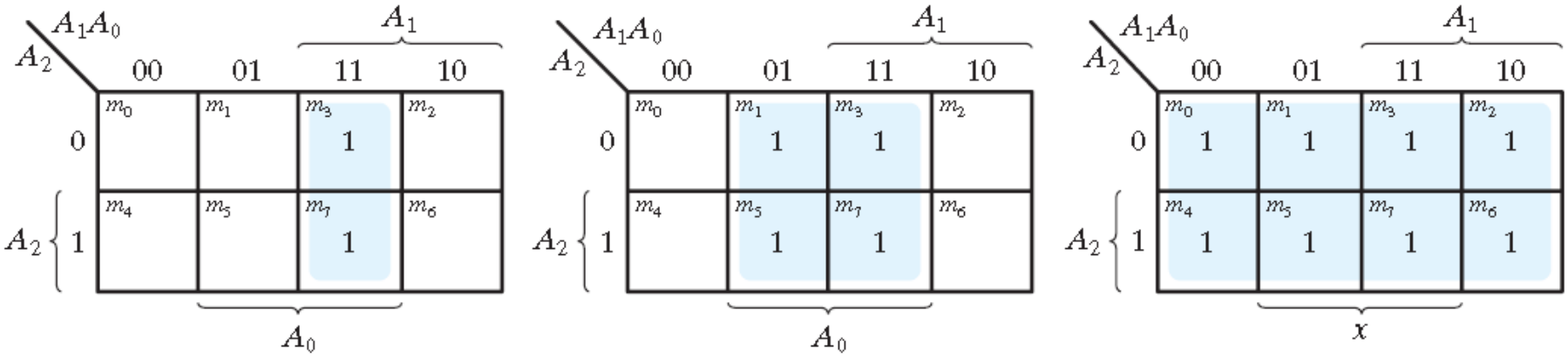
- Binary counters are constructed most efficiently with *T* flip-flops.
- Three flip-flops A_2, A_1, and A_0 are used.

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

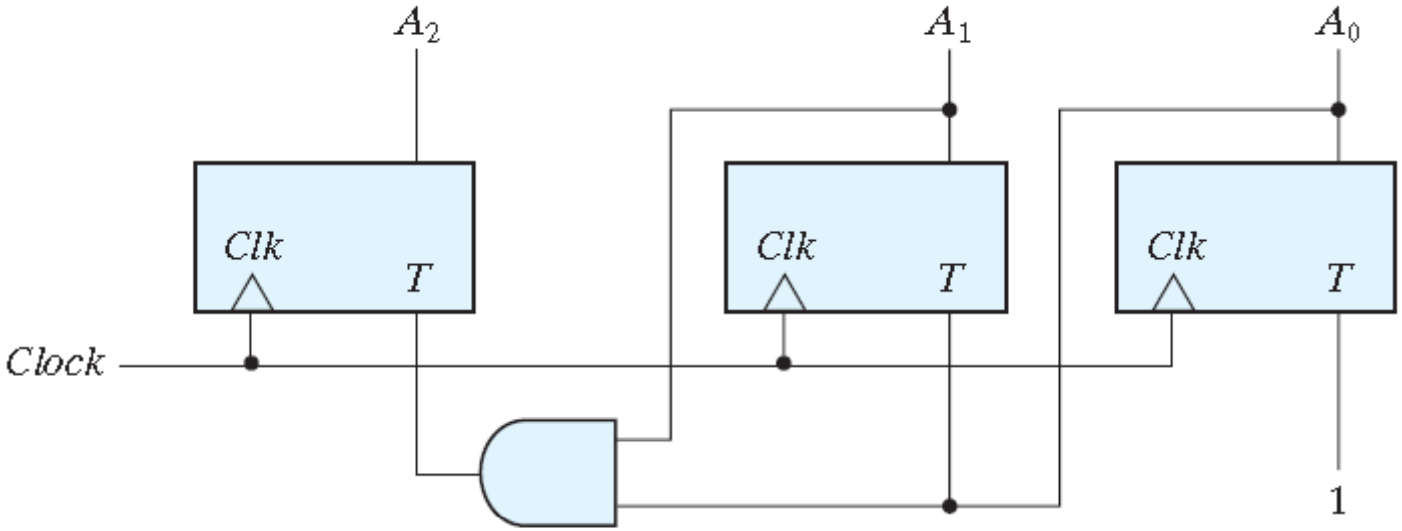
(b) *T* Flip-Flop

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



$T_{A2} = A_1A_0$ $T_{A1} = A_0$ $T_{A0} = 1$



Homework #5

- 5.6
- 5.8
- 5.10
- 5.12
- 5.16