Introduction to Digital Logic

Assist. Prof. Hamza Osman ILHAN

hoilhan@yildiz.edu.tr

Course Outline

1. Digital Computers, Number Systems, Arithmetic Operations, Decimal, Alphanumeric, and Gray Codes
2. Binary Logic, Gates, Boolean Algebra, Standard Forms
3. Circuit Optimization, Two-Level Optimization, Map Manipulation, Multi-Level Circuit Optimization
4. Additional Gates and Circuits, Other Gate Types, Exclusive-OR Operator and Gates, High-Impedance Outputs
5. Implementation Technology and Logic Design, Design Concepts and Automation, The Design Space, Design Procedure, The major design steps
6. Programmable Implementation Technologies: Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic,Technology mapping to programmable logic devices
7. Combinational Functions and Circuits
8. Arithmetic Functions and Circuits
9. Sequential Circuits Storage Elements and Sequential Circuit Analysis
10. Sequential Circuits, Sequential Circuit Design State Diagrams, State Tables
11. Counters, register cells, buses, & serial operations
12. Sequencing and Control, Datapath and Control, Algorithmic State Machines (ASM)
13. Memory Basics

Formulation: Finding a State Diagram

- In specifying a circuit, we use states to remember meaningful properties of past input sequences that are essential to predicting future output values.
- A sequence recognizer is a sequential circuit that produces a distinct output value whenever a prescribed pattern of input symbols occur in sequence, i.e, recognizes an input sequence occurence.
- We will develop a procedure specific to sequence recognizers to convert a problem statement into a state diagram.
- Next, the state diagram, will be converted to a state table from which the circuit will be designed.

Sequence Recognizer Procedure

- To develop a sequence recognizer state diagram:
  - Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically "reset" state).
  - Add a state that recognizes that the first symbol has occurred.
  - Add states that recognize each successive symbol occurring.
  - The final state represents the input sequence (possibly less the final input value) occurence.
  - Add state transition arcs which specify what happens when a symbol *not* in the proper sequence has occurred.
  - Add other arcs on non-sequence inputs which transition to states that represent the input subsequence that has occurred.
- The last step is required because the circuit must recognize the input sequence *regardless of where it occurs within the overall sequence applied since "reset."*.

State Assignment

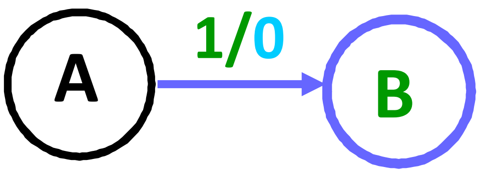
- Each of the m states must be assigned a unique code
- Minimum number of bits required is n such that
  $$n \geq \lceil \log_2 m \rceil$$
  where $\lceil x \rceil$ is the smallest integer $\geq x$
- There are useful state assignments that use more than the minimum number of bits
- There are 2^n - m unused states

Sequence Recognizer Example

- Example: Recognize the sequence 1101
  - Note that the sequence 1111101 contains 1101 and "11" is a proper sub-sequence of the sequence.
- Thus, the sequential machine must remember that the first two one's have occurred as it receives another symbol.
- Also, the sequence 1101101 contains 1101 as both an initial subsequence and a final subsequence with some overlap, i. e., <u>1101</u>101 or 110<u>1101</u>.
- And, the 1 in the middle, 110<u>1</u>101, is in both subsequences.
- The sequence 1101 must be recognized each time it occurs in the input sequence.

Example: Recognize 1101

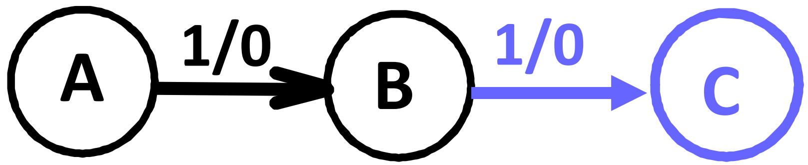
- Define states for the sequence to be recognized:
  - assuming it starts with first symbol,
  - continues through each symbol in the sequence to be recognized, and
  - uses output 1 to mean the full sequence has occurred,
  - with output 0 otherwise.
- Starting in the initial state (Arbitrarily named "A"):
  - Add a state that recognizes the first "1."



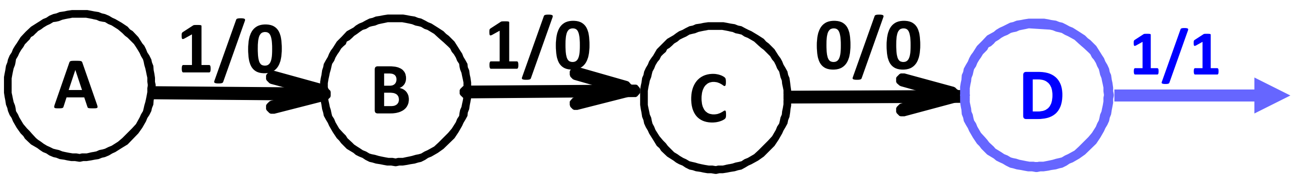
  - State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred.
  - The output symbol "0" means that the full recognized sequence has not yet occurred.

Example: Recognize 1101 (continued)

- After one more 1, we have:
  - C is the state obtained when the input sequence has two "1"s.

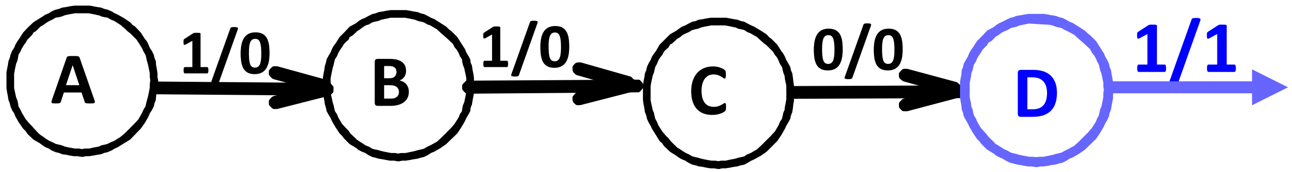


- Finally, after 110 and a 1, we have:

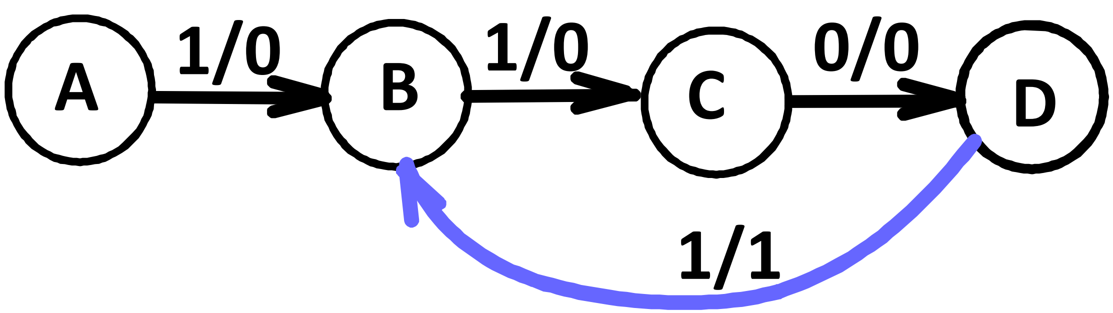


  - Transition arcs are used to denote the output function (Mealy Model)
  - Output 1 on the arc from D means the sequence has been recognized
  - To what state should the arc from state D go? Remember: 1101101 ?
  - Note that D is the last state but the output 1 occurs for the input applied in D. This is the case when a *Mealy model* is assumed.

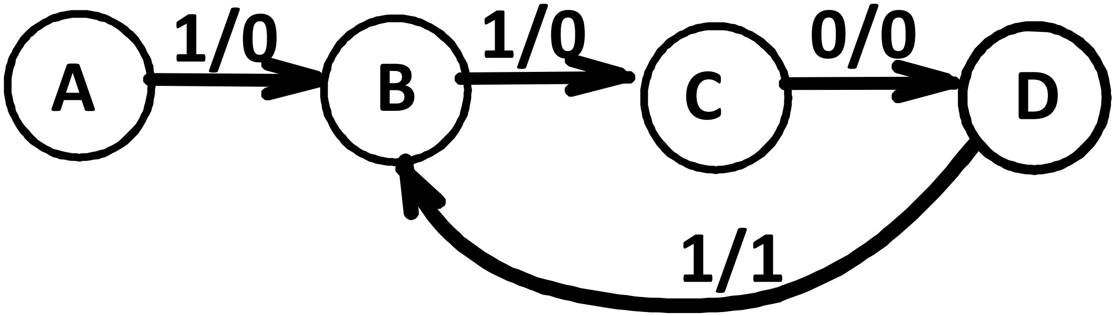
Example: Recognize 1101 (continued)



- Clearly the final 1 in the recognized sequence 1101 is a sub-sequence of 1101. It follows a 0 which is not a sub-sequence of 1101. Thus it should represent *the same state reached from the initial state after a first 1 is observed*. We obtain:

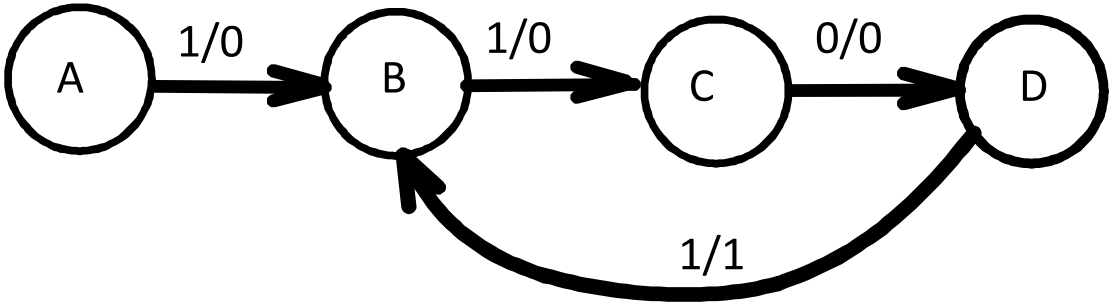


Example: Recognize 1101 (continued)



- The state have the following abstract meanings:
  - A: No proper sub-sequence of the sequence has occurred.
  - B: The sub-sequence 1 has occurred.
  - C: The sub-sequence 11 has occurred.
  - D: The sub-sequence 110 has occurred.
  - The 1/1 on the arc from D to B means that the last 1 has occurred and thus, the sequence is recognized.

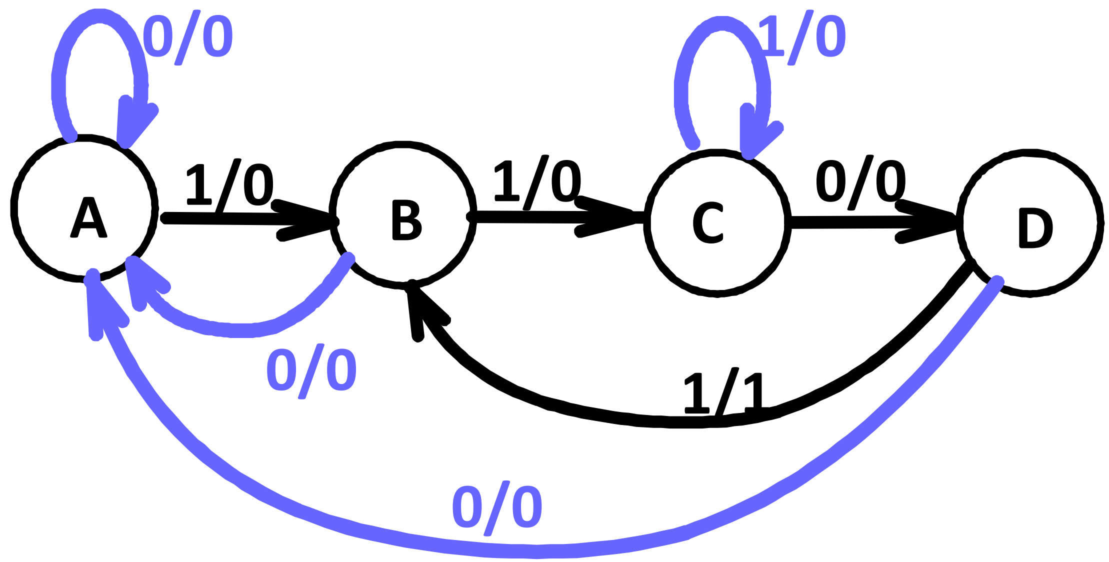
Example: Recognize 1101 (continued)



- The other arcs are added to each state for inputs not yet listed. Which arcs are missing?
  - "0" arc from A
  - "0" arc from B
  - "1" arc from C
  - "0" arc from D.

Example: Recognize 1101 (continued)

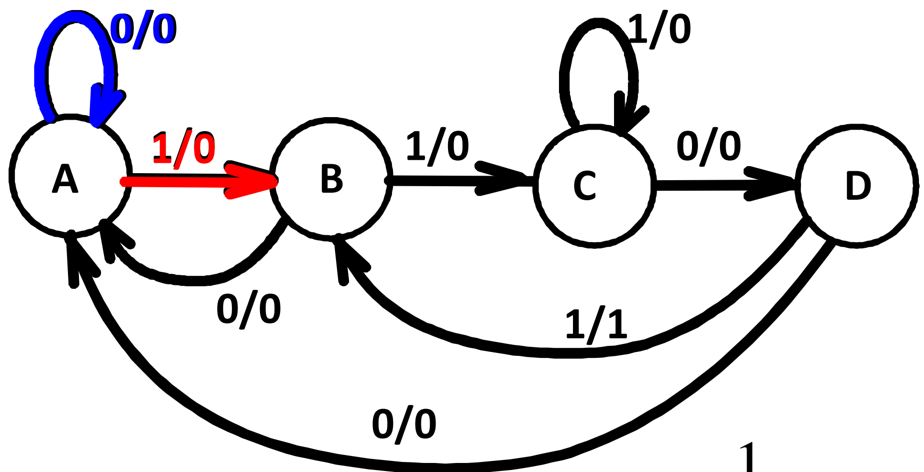
- State transition arcs must represent the fact that an input subsequence has occurred. Thus we get:



- Note that the 1 arc from state C to state C implies that State C means *two or more 1's have occurred*.

Formulation: Find State Table

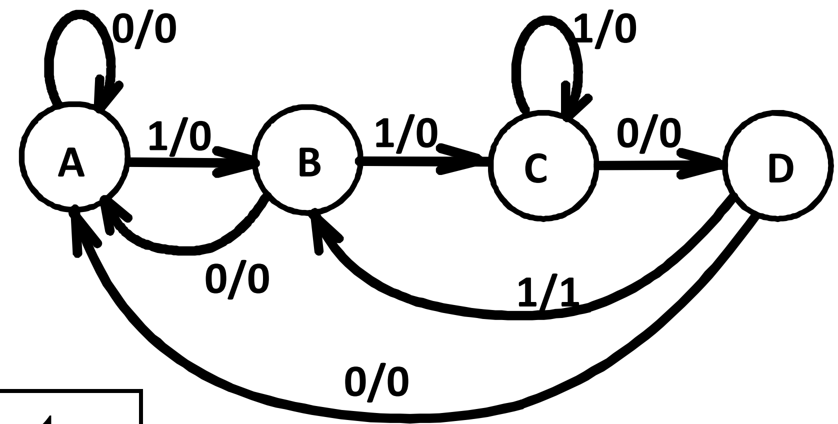
- From the State Diagram, we can fill in the State Table.
- There are 4 states, one input, and one output. We will choose the form with four rows, one for each current state.
- From State A, the 0 and 1 input transitions have been filled in along with the outputs.



Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B				
C				
D				

Formulation: Find State Table

- From the state diagram, we complete the state table.



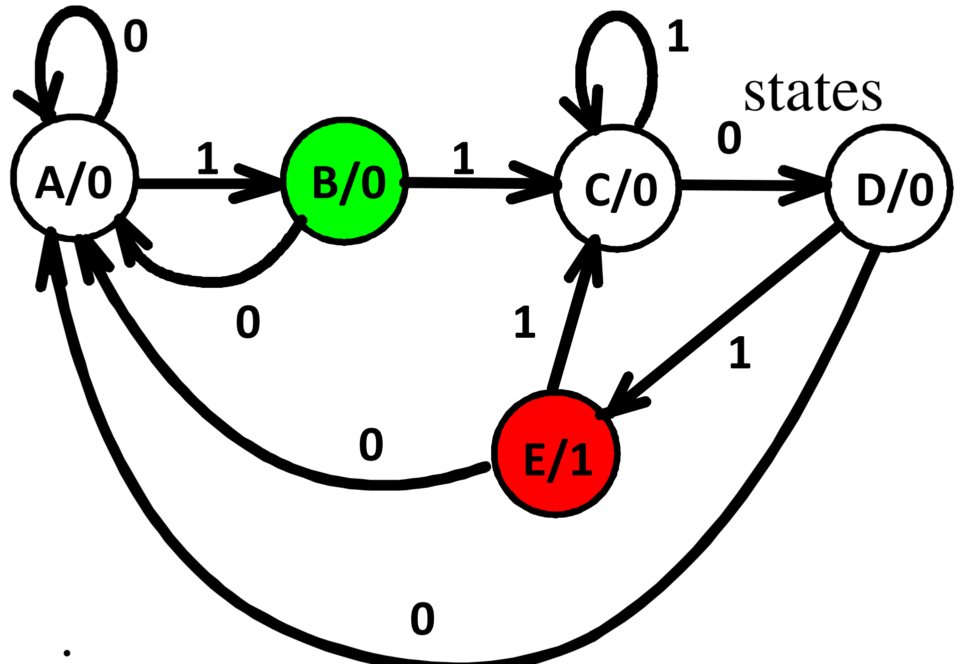
Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- What would the state diagram and state table look like for the Moore model?

Example: Moore Model for Sequence 1101

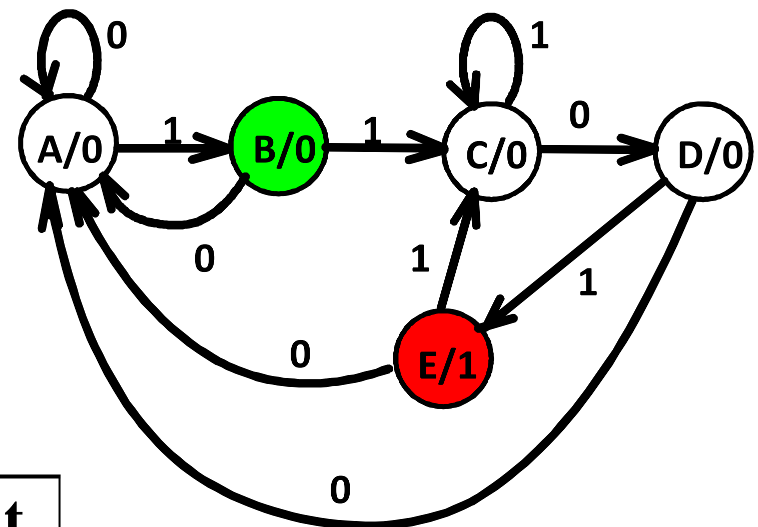
- For the Moore Model, outputs are associated with states.
- We need to add a state "E" with output value 1 for the final 1 in the recognized input sequence.
  - This new state E, though similar to B, would generate an output of 1 and thus be different from B.
- The Moore model for a sequence recognizer usually has *more states* than the Mealy model.

Example: Moore Model (continued)



- We mark outputs on for Moore model
- Arcs now show only state transitions
- Add a new state E to produce the output 1
- Note that the new state, E produces the same behavior in the future as state B. But it gives a different output at the present time. Thus these states do represent a *different abstraction* of the input history.

Example: Moore Model (continued)



- The state table is shown below

Present State	Next State x=0	Next State x=1	Output y
A	A	B	0
B	A	C	0
C	D	C	0
D	A	E	0
E	A	C	1

State Assignment – Example 1

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

- How may assignments of codes with a minimum number of bits?
  - Two

    A = 0, B = 1 or A = 1, B = 0
- Does it make a difference?
  - Only in variable inversion, so small, if any.

State Assignment – Example 2

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- How may assignments of codes with a minimum number of bits?

  $4 \times 3 \times 2 \times 1 = 24$

- Does code assignment make a difference in cost?

State Assignment – Example 2 (continued)

- Assignment 1: A = 0 0, B = 0 1, C = 1 0, D = 1 1
- The resulting coded state table:

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

State Assignment – Example 2 (continued)

- Assignment 2: A = 0 0, B = 0 1, C = 1 1, D = 1 0
- The resulting coded state table:

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

Find Flip-Flop Input and Output Equations: Example 2 - Assignment 1

- **Assume D flip-flops**
- **Interchange the bottom two rows of the state table, to obtain K-maps for D_1, D_2, and Z:**

D_1

		X	
	0	0	
	0	1	Y_2
Y_1	0	0	Y_2
Y_1	1	1	

D_2

		X	
	0	1	
	0	0	Y_2
Y_1	0	1	Y_2
Y_1	1	0	

Z

		X	
	0	0	
	0	0	Y_2
Y_1	0	0	Y_2
Y_1	0	1	

Optimization: Example 2: Assignment 1

- Performing two-level optimization:

D_1

Y_1Y_2	$\bar{X}$	X
00	0	0
01	0	1
11	0	0
10	1	1

D_2

Y_1Y_2	$\bar{X}$	X
00	0	1
01	0	0
11	0	1
10	1	0

Z

Y_1Y_2	$\bar{X}$	X
00	0	0
01	0	0
11	0	0
10	0	1

$$D_1 = Y_1\bar{Y}_2 + X\bar{Y}_1Y_2$$

$$D_2 = \bar{X}Y_1\bar{Y}_2 + X\bar{Y}_1\bar{Y}_2 + XY_1Y_2$$

$$Z = XY_1\bar{Y}_2$$

Gate Input Cost = 22

Find Flip-Flop Input and Output Equations: Example 2 - Assignment 2

- Assume D flip-flops
- Obtain K-maps for D_1, D_2, and Z:

D_1

Y_1Y_2 \ X	0	1
00	0	0
01	0	1
11	1	1
10	0	0

D_2

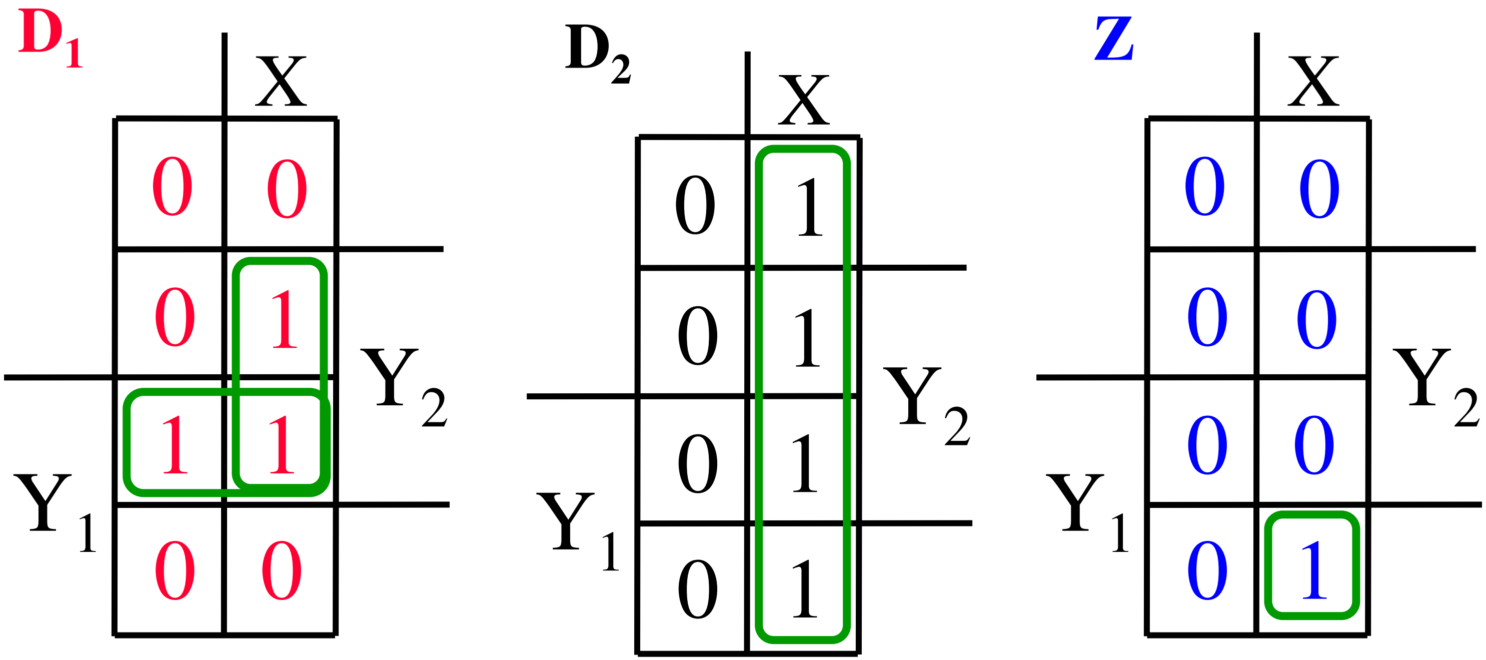
Y_1Y_2 \ X	0	1
00	0	1
01	0	1
11	0	1
10	0	1

Z

Y_1Y_2 \ X	0	1
00	0	0
01	0	0
11	0	0
10	0	1

Optimization: Example 2: Assignment 2

- Performing two-level optimization:



$D_1 = Y_1Y_2 + XY_2$ Gate Input Cost = 9

$D_2 = X$

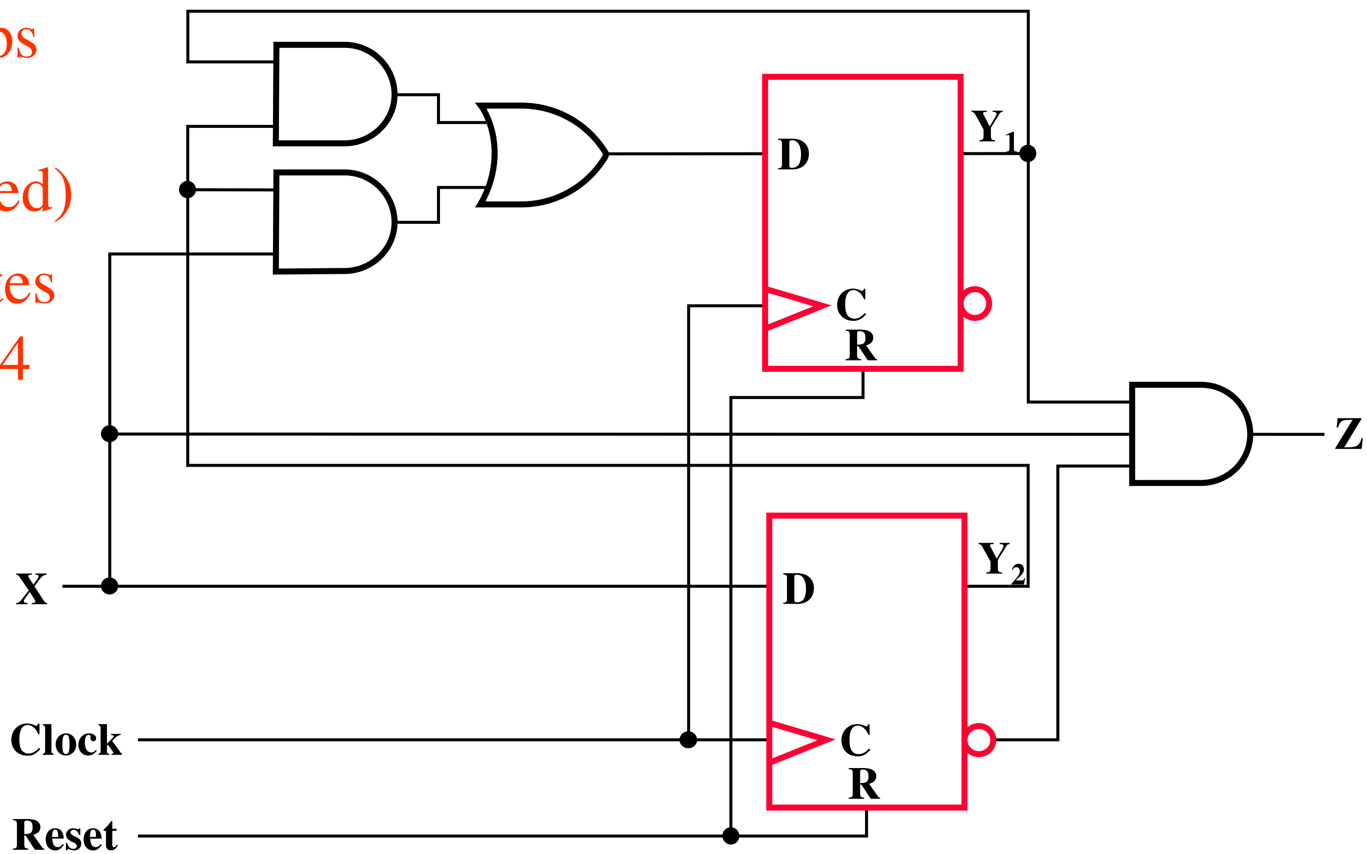
$Z = XY_1\overline{Y}_2$

Select this state assignment for completion of the design

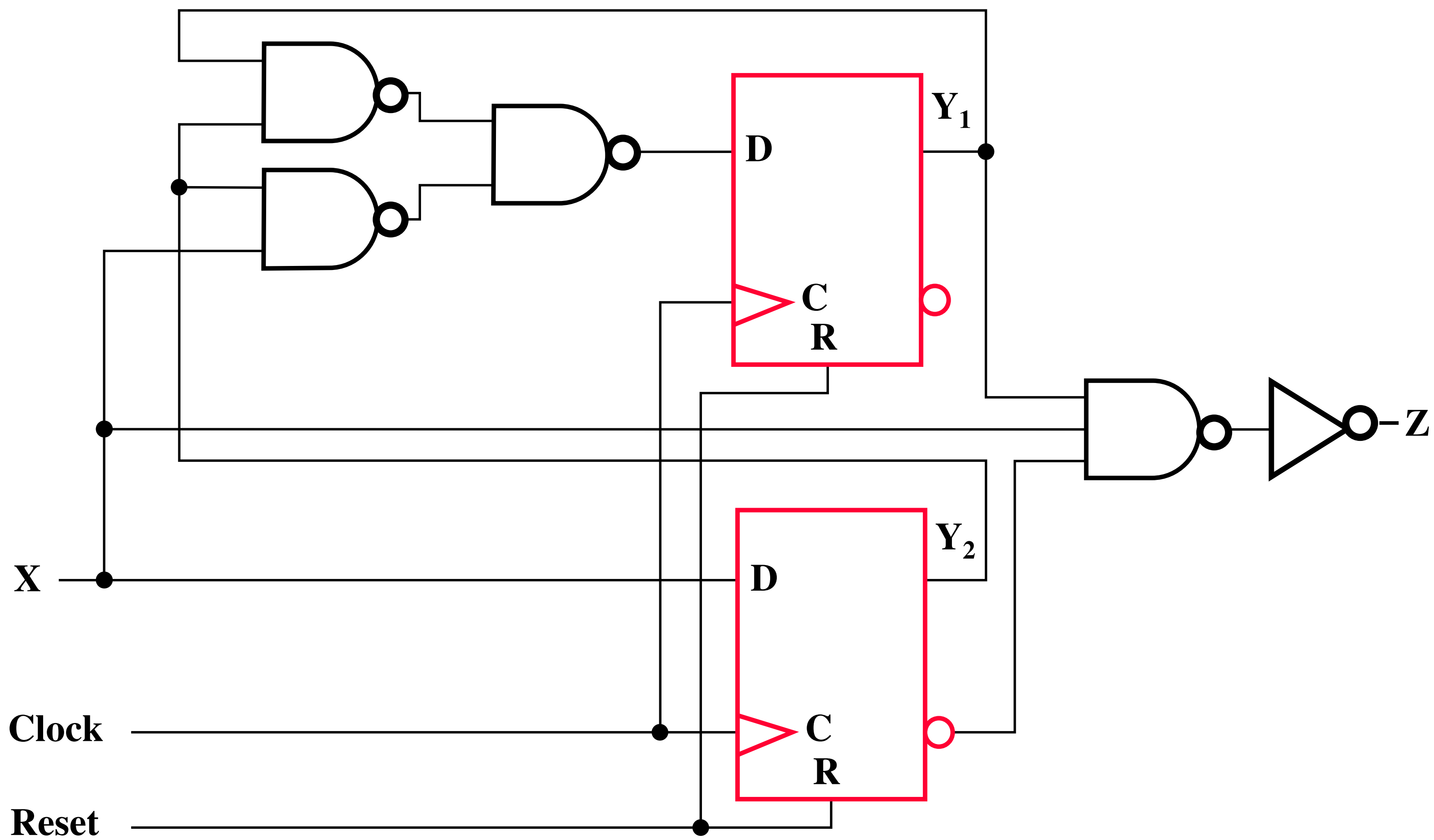
Map Technology

- Library:
  - D Flip-flops with Reset (not inverted)
  - NAND gates with up to 4 inputs and inverters

- **Initial Circuit:**



Mapped Circuit - Final Result



State Assignment (continued)

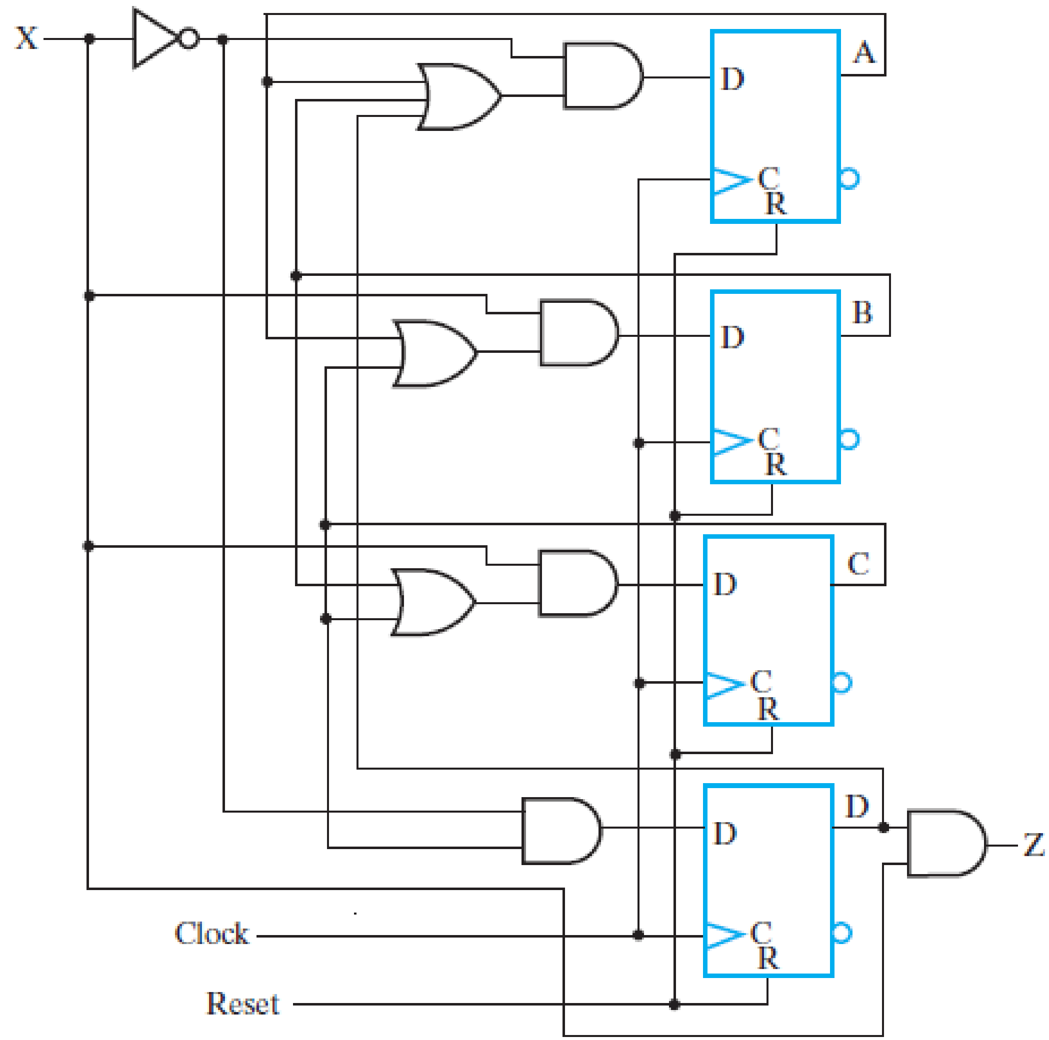
- One-Hot Assignment : A =1000, B = 0100, C = 0010, D = 0001 The resulting coded state table:

Table 5-3 with Names Replaced by a 4-Bit One-Hot Code

Present State	Next State		Output *Z*	
ABCD	X = 0	X = 1	X = 0	X = 1
1000	1000	0100	0	0
0100	1000	0010	0	0
0010	0001	0010	0	0
0001	1000	0100	0	1

Implementation

$$A(t+1) = D_A = A\overline{X} + B\overline{X} + D\overline{X}$$
$$= (A + B + D)\overline{X}$$
$$B(t+1) = D_B = AX + DX$$
$$= (A + D)X$$
$$C(t+1) = D_C = BX + CX$$
$$= (B + C)X$$
$$D(t+1) = D_D = C\overline{X}$$
$$Z = DX$$



Other Flip-Flop Types

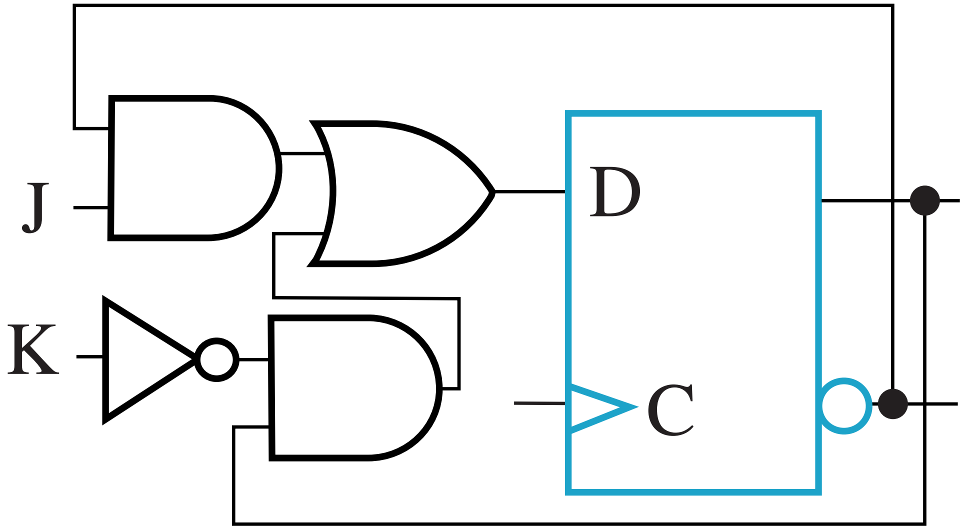
- J-K and T flip-flops
  - Behavior
  - Implementation
- Basic descriptors for understanding and using different flip-flop types
  - Characteristic tables
  - Characteristic equations
  - Excitation tables
- For actual use, see Reading Supplement - Design and Analysis Using J-K and T Flip-Flops

J-K Flip-flop

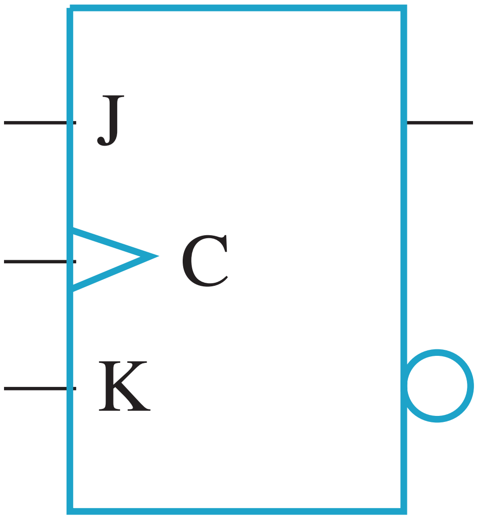
- Behavior
  - Same as S-R flip-flop with J analogous to S and K analogous to R
  - Except that J = K = 1 is allowed, and
  - For J = K = 1, the flip-flop changes to the *opposite state*
  - As a master-slave, has same "1s catching" behavior as S-R flip-flop
  - If the master changes to the wrong state, that state will be passed to the slave
    - E.g., if master falsely set by J = 1, K = 1 cannot reset it during the current clock cycle

J-K Flip-flop (continued)

- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop



- **Symbol**

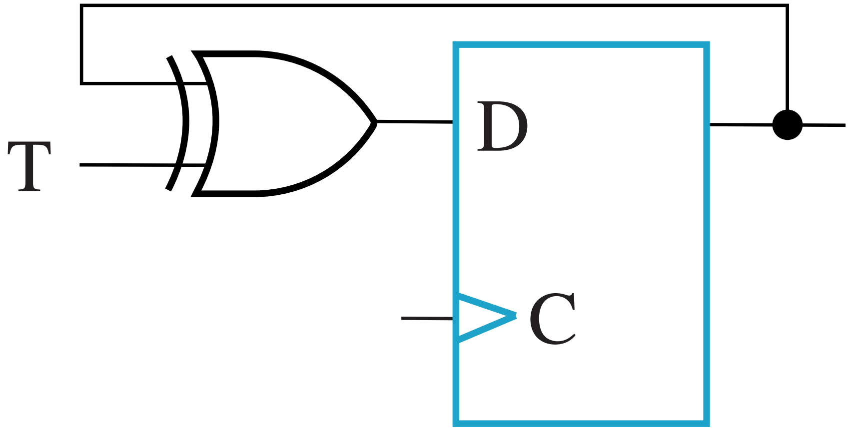


T Flip-flop

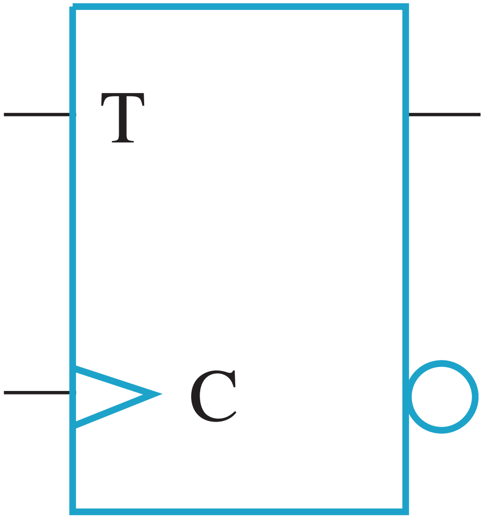
- Behavior
  - Has a single input T
    - For T = 0, no change to state
    - For T = 1, changes to opposite state
- Same as a J-K flip-flop with J = K = T
- As a master-slave, has same "1s catching" behavior as J-K flip-flop
- Cannot be initialized to a known state using the T input
  - Reset (asynchronous or synchronous) essential

T Flip-flop (continued)

- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop



- **Symbol**



Basic Flip-Flop Descriptors

- Used in analysis
  - *Characteristic table* - defines the next state of the flip-flop in terms of flip-flop inputs and current state
  - *Characteristic equation* - defines the next state of the flip-flop as a Boolean function of the flip-flop inputs and the current state
- Used in design
  - *Excitation table* - defines the flip-flop input variable values as function of the current state and next state

D Flip-Flop Descriptors

- Characteristic Table

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

- Characteristic Equation

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

- Excitation Table

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

T Flip-Flop Descriptors

- Characteristic Table

T	Q(t+1)	Operation
0	$Q(t)$	No change
1	$\bar{Q}(t)$	Complement

- Characteristic Equation

$$Q(t+1) = T \oplus Q$$

- Excitation Table

Q(t+1)	T	Operation
$Q(t)$	0	No change
$\bar{Q}(t)$	1	Complement

S-R Flip-Flop Descriptors

- Characteristic Table

S	R	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

- Characteristic Equation

$$Q(t+1) = S + \overline{R}\,Q,\ S \cdot R = 0$$

- Excitation Table

Q(t)	Q(t+1)	S	R	Operation
0	0	0	X	No change
0	1	1	0	Set
1	0	0	1	Reset
1	1	X	0	No change

J-K Flip-Flop Descriptors

- Characteristic Table

J	K	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement

- Characteristic Equation

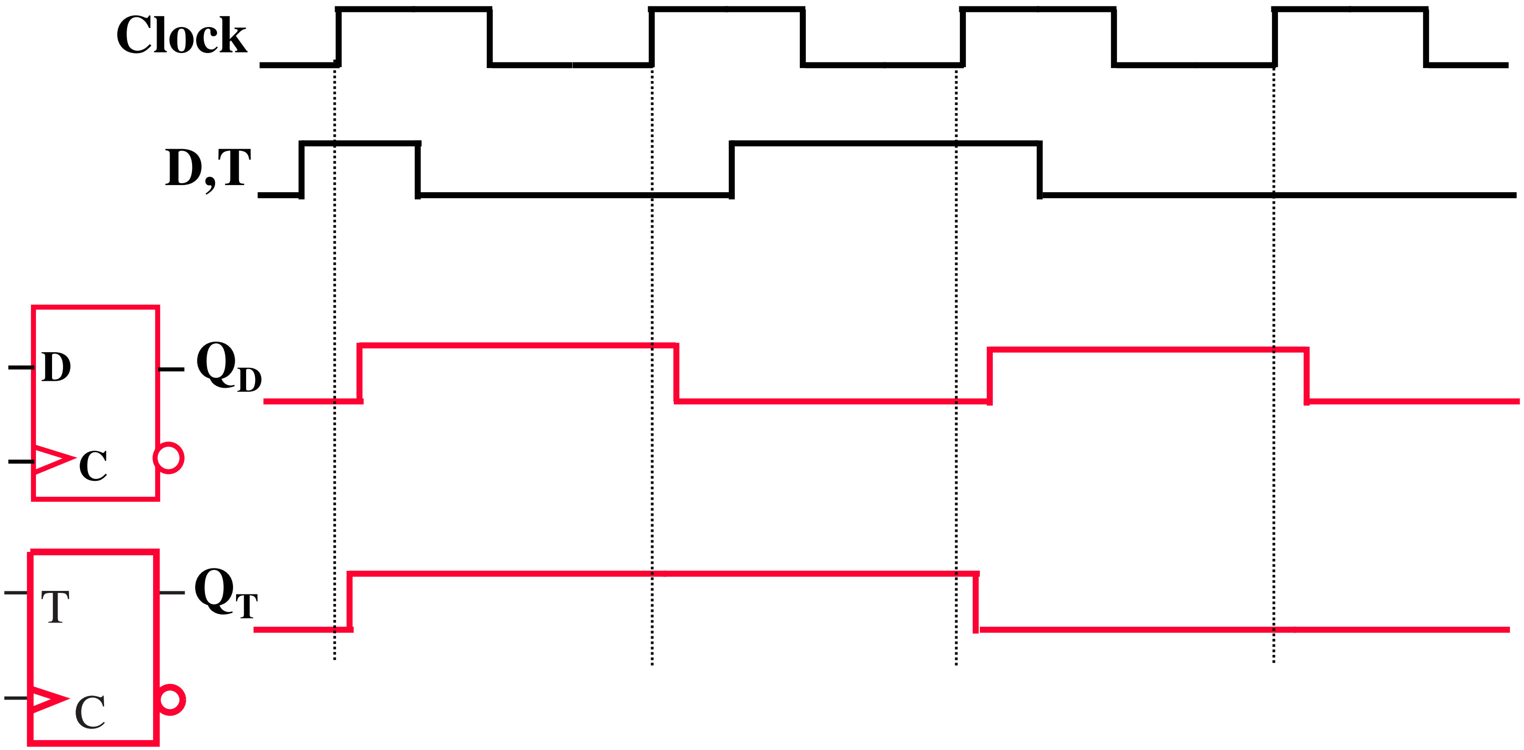
$$Q(t+1) = J\,\overline{Q} + \overline{K}\,Q$$

- Excitation Table

Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

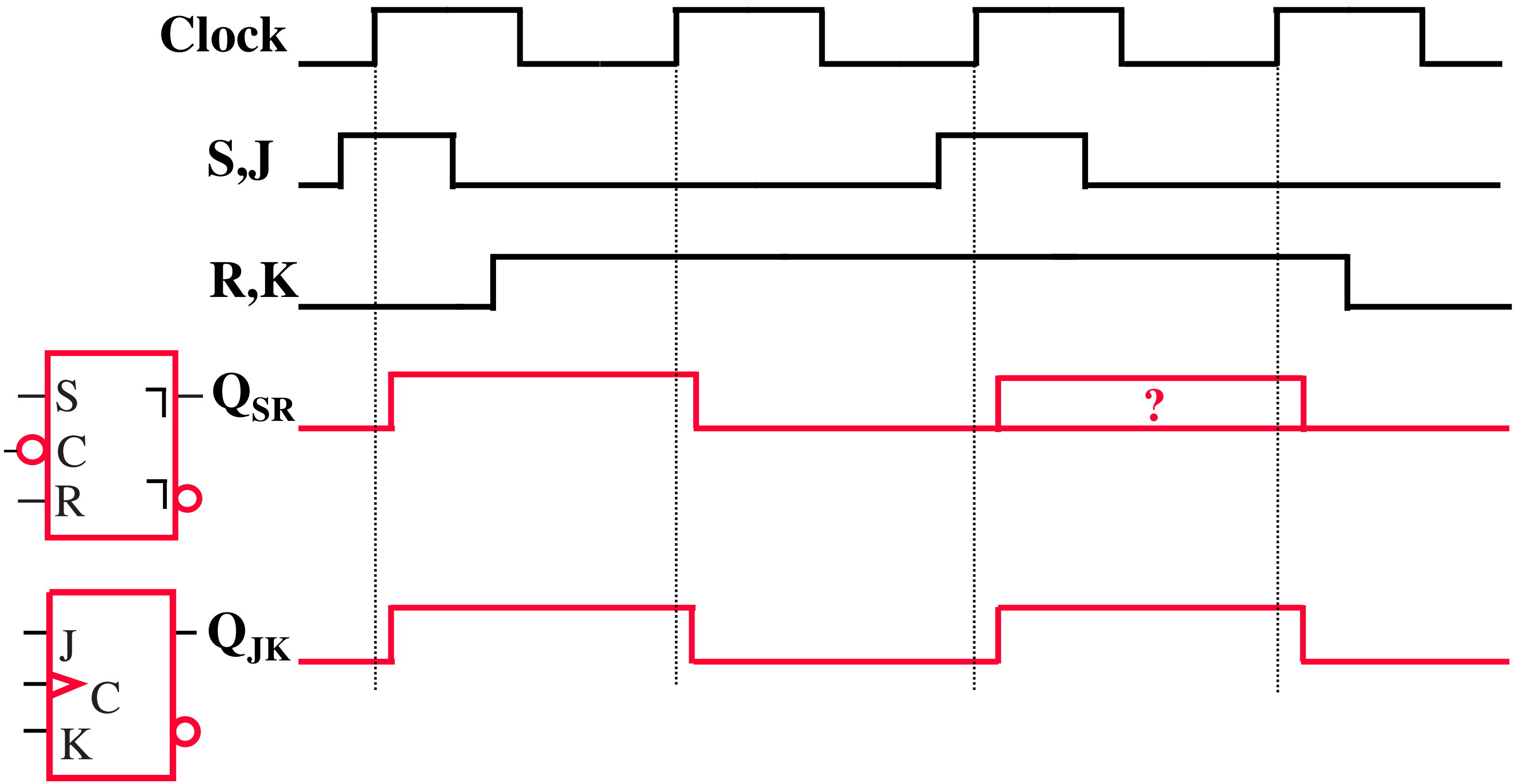
Flip-flop Behavior Example

- Use the characteristic tables to find the output waveforms for the flip-flops shown:

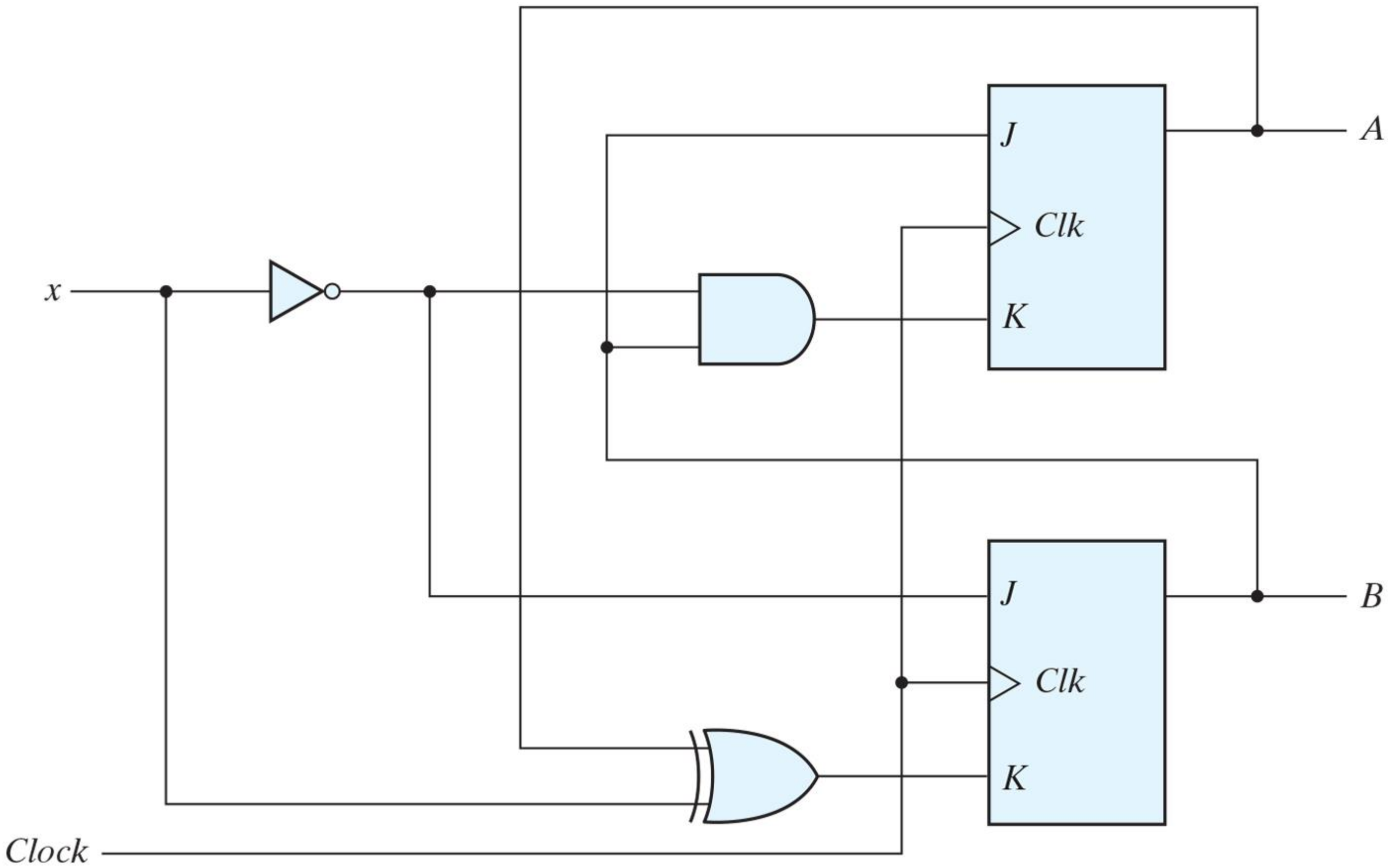


Flip-Flop Behavior Example (continued)

- Use the characteristic tables to find the output waveforms for the flip-flops shown:

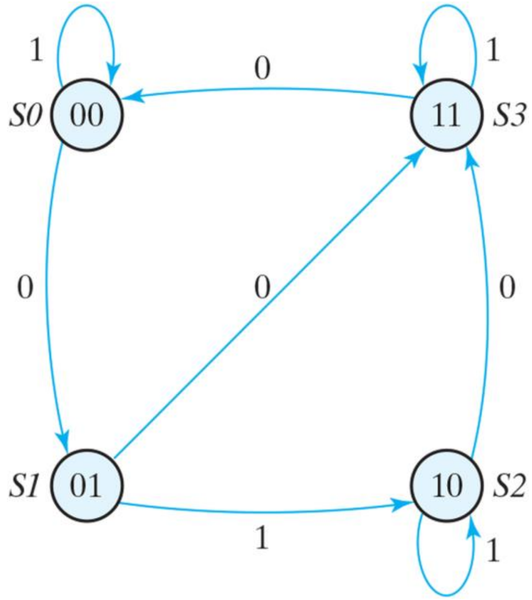


J-K Flip-flop (example)

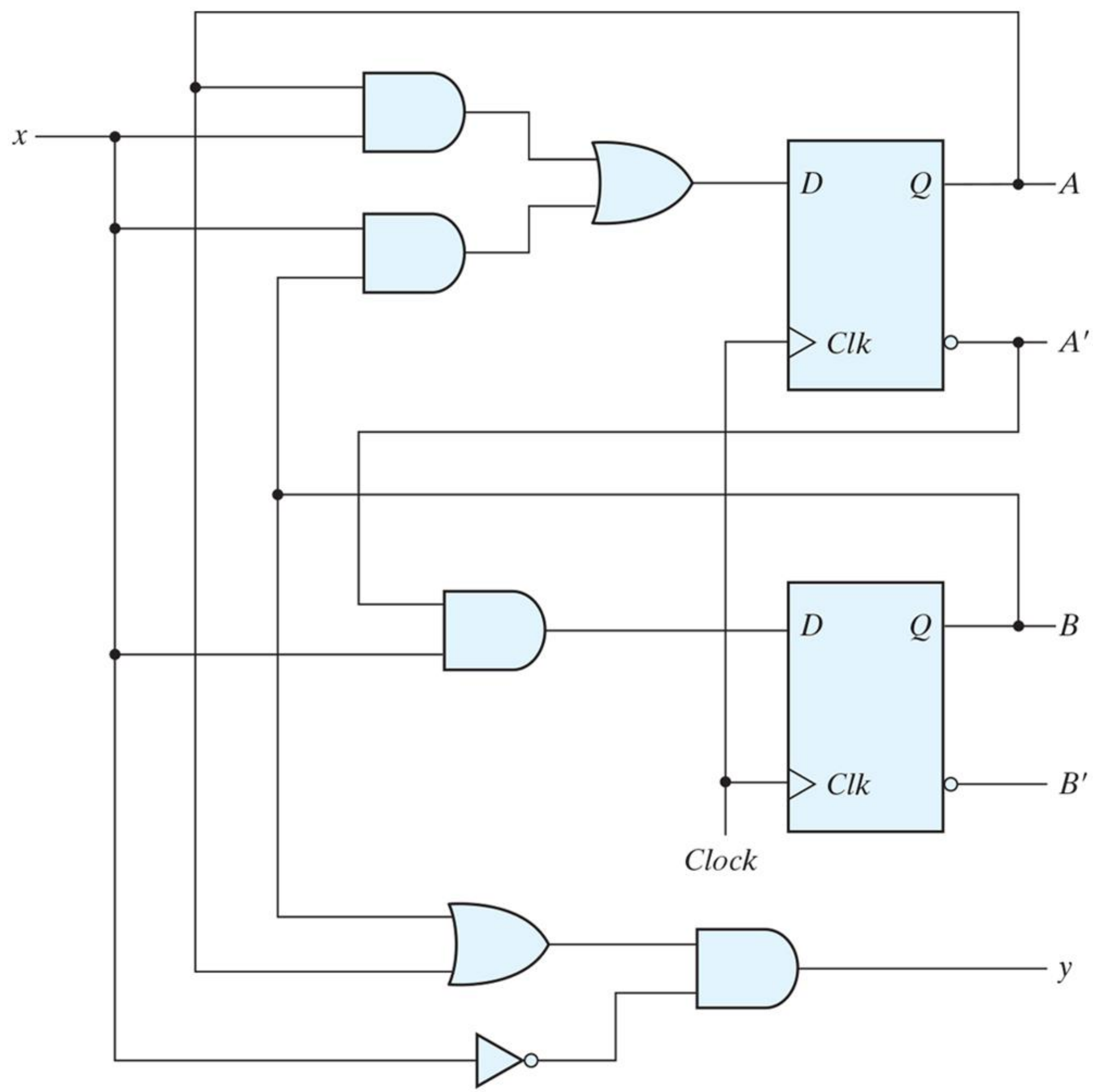


J-K Flip-flop (example)

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

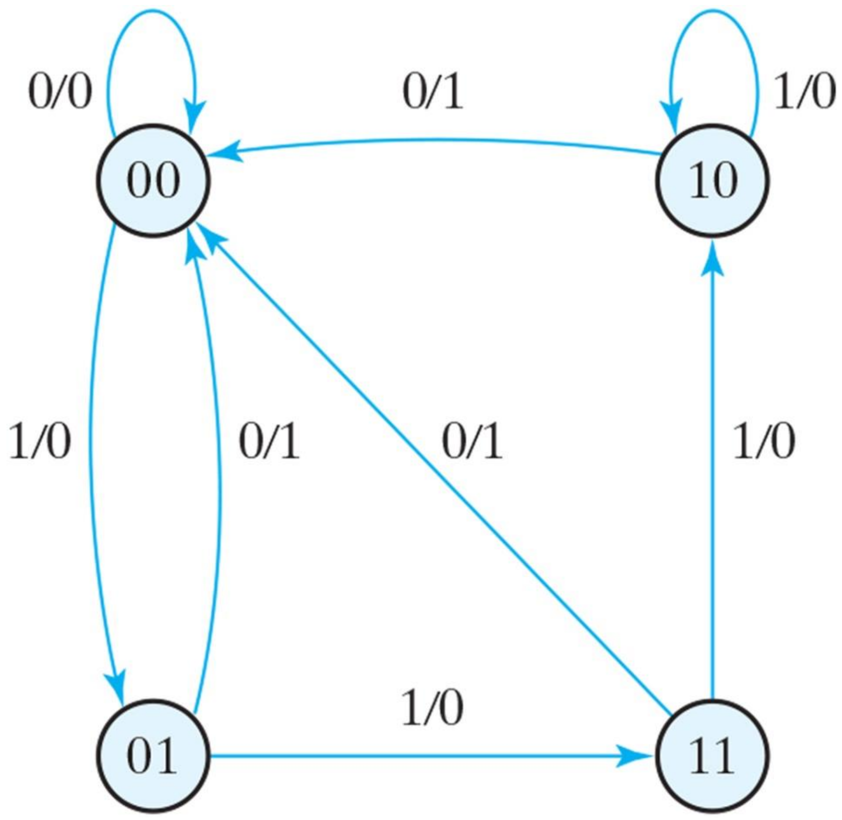


D Flip-flop (example)

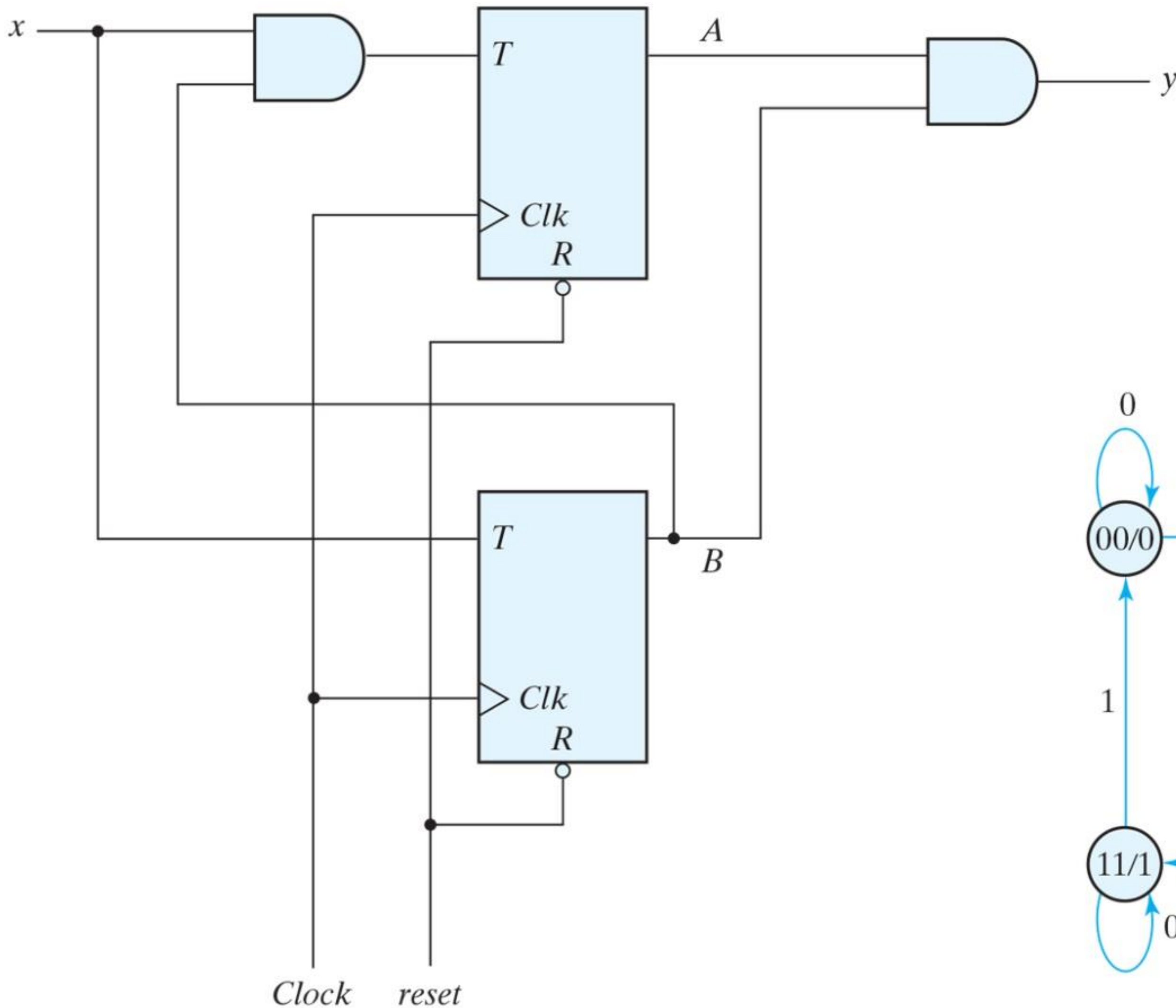


D Flip-flop (example)

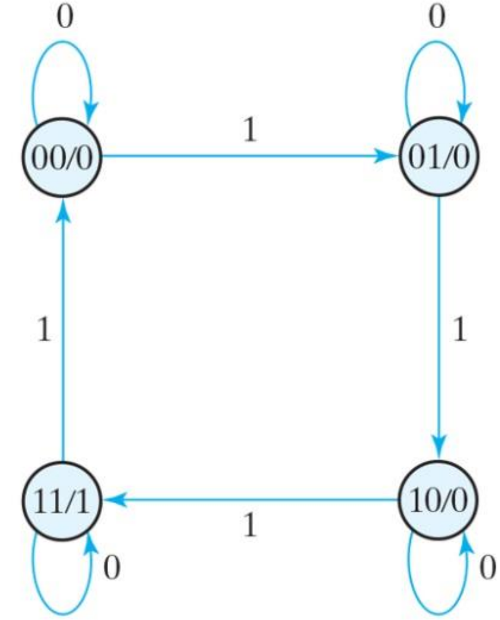
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



T Flip-flop (example)



(a) Circuit diagram

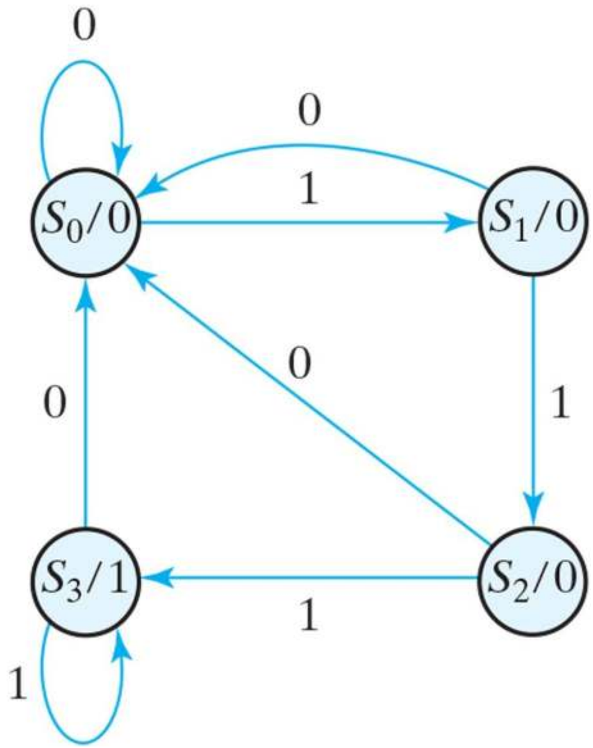


(b) State diagram

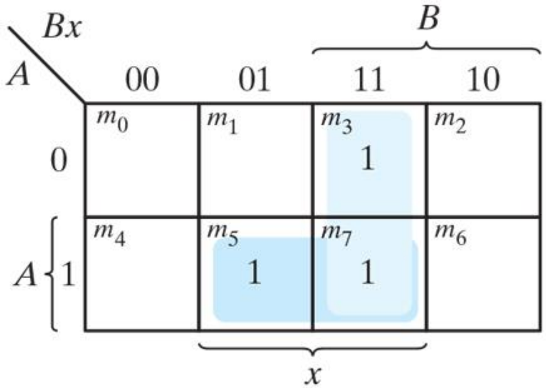
T Flip-flop (example)

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

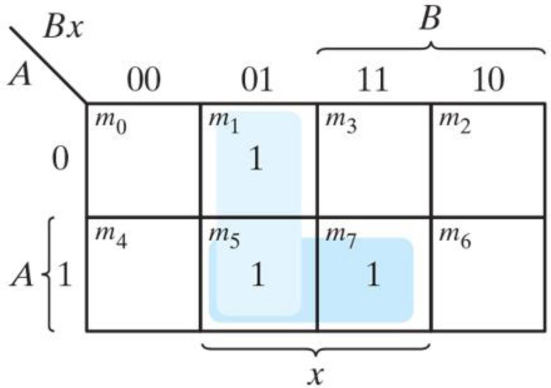
Moore Example



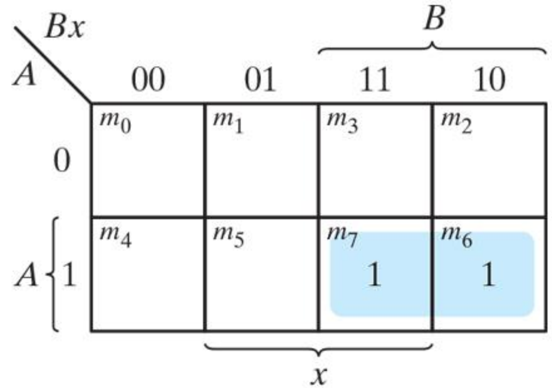
Present State		Input	Next State		Output
A	**B**	**x**	**A**	**B**	**y**
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1



$D_A = Ax + Bx$

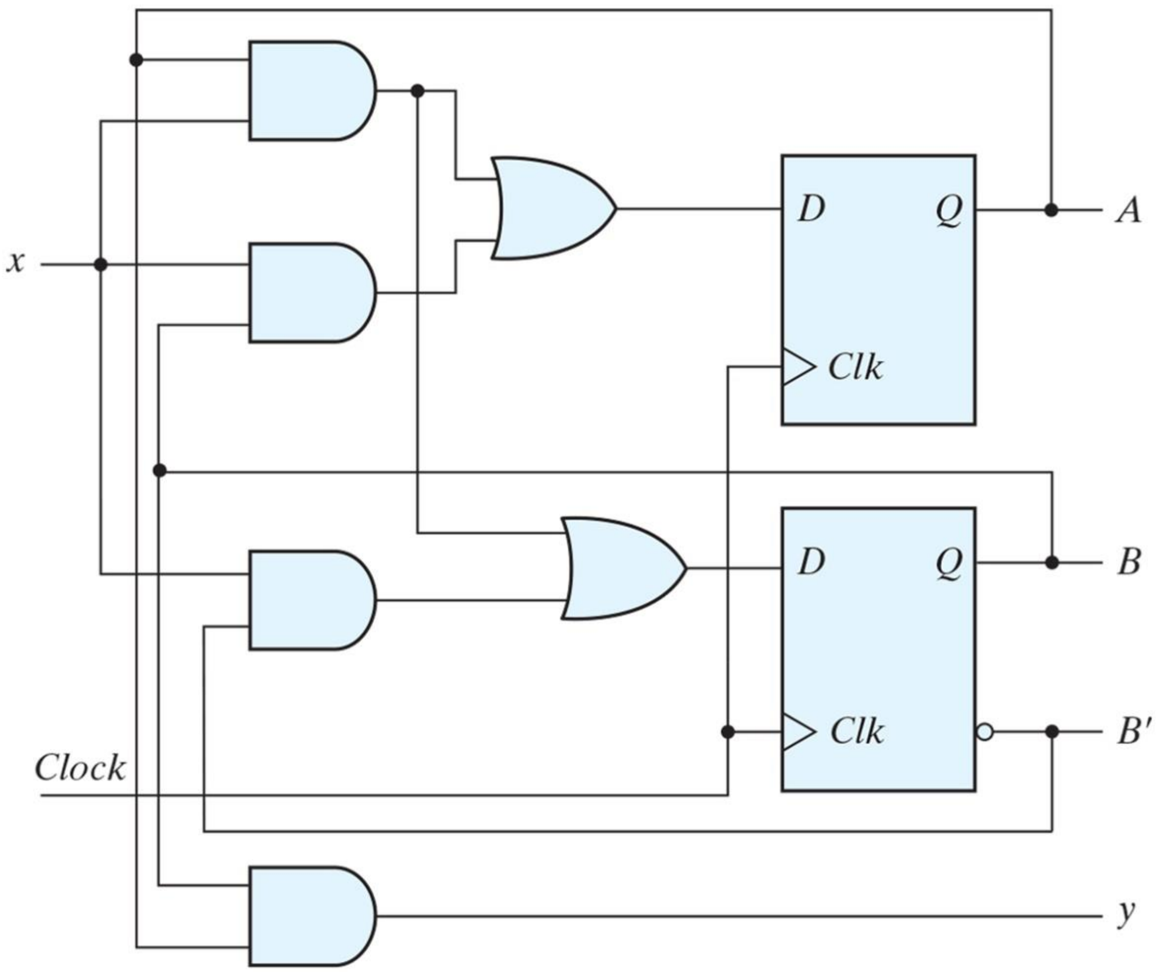


$D_B = Ax + B'x$



$y = AB$

Moore Example



Moore Example

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

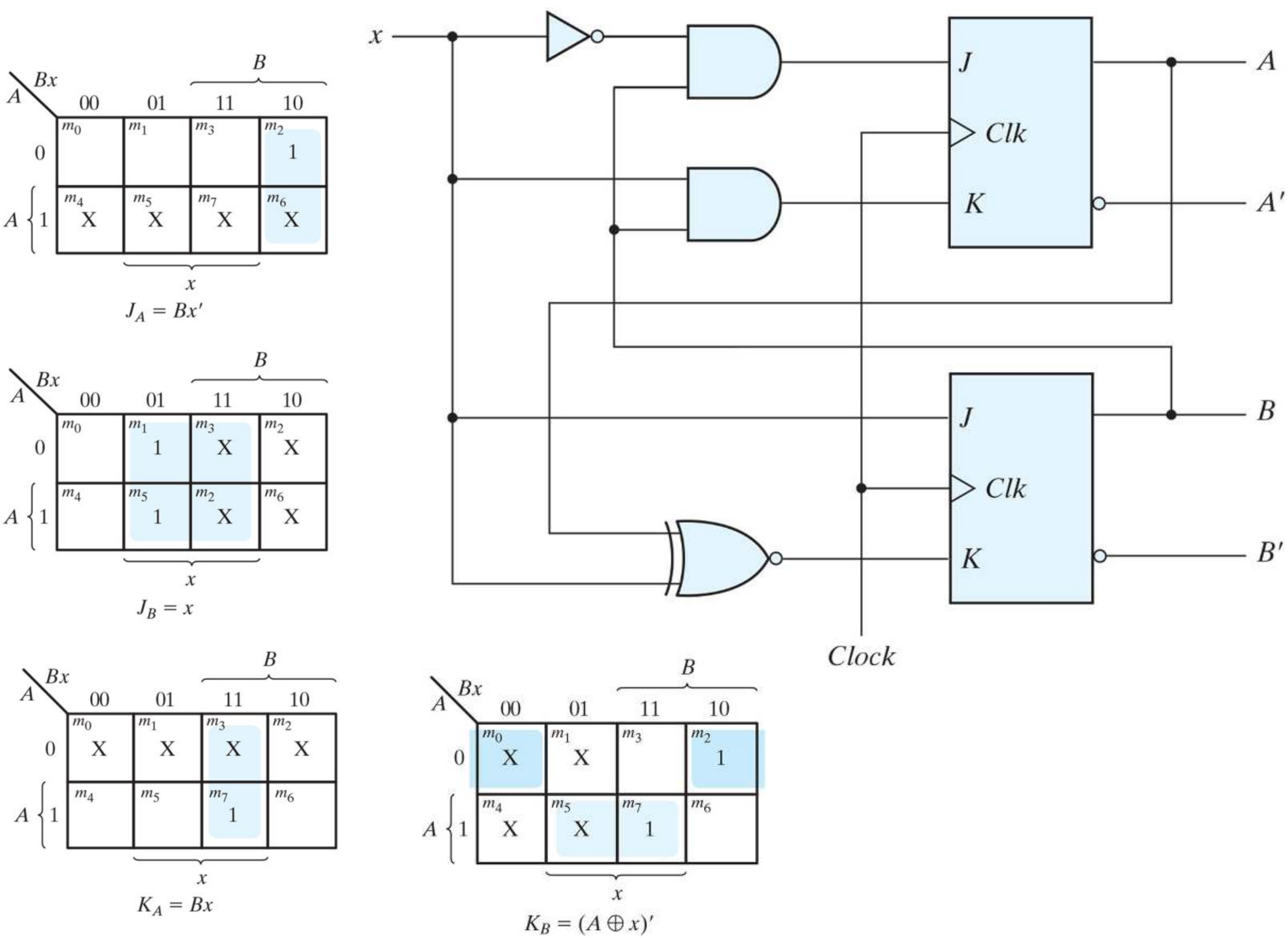
(a) *JK* Flip-Flop

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

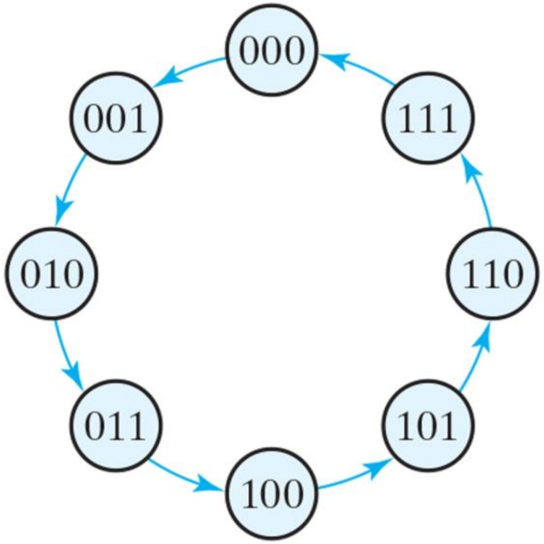
(b) *T* Flip-Flop

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

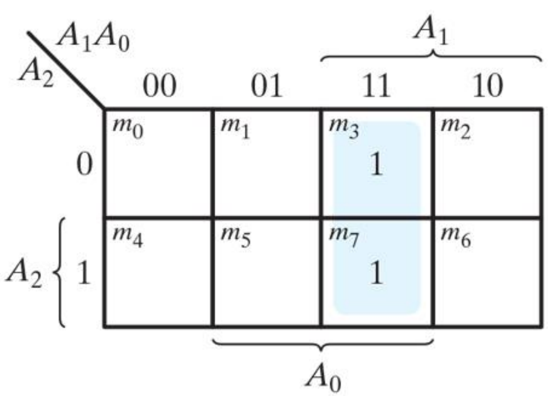
Moore Example



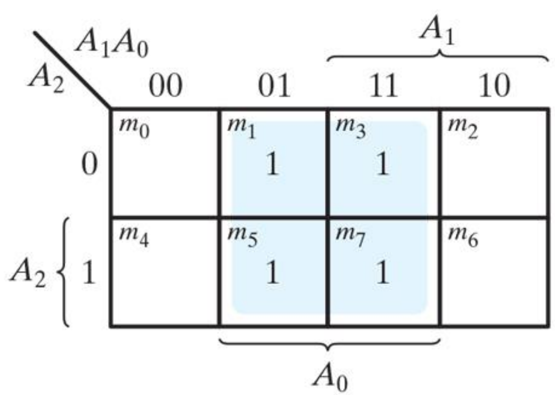
Example



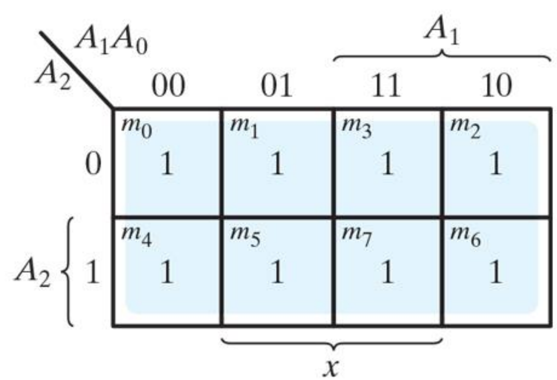
Present State			Next State			Flip-Flop Inputs		
A_2	A_1	A_0	A_2	A_1	A_0	T_{A2}	T_{A1}	T_{A0}
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



$T_{A2} = A_1A_0$



$T_{A1} = A_0$



$T_{A0} = 1$

Example