Computer Logic Design Fundamentals

Chapter 4 – Sequential Circuits

Part 2 – Sequential Circuit Design

Prof. Yueming Wang
ymingwang@zju.edu.cn
College of Computer Science and Technology,
Zhejiang University

Overview

- **Part 1 - Storage Elements and Sequential Circuit Analysis**
- **Part 2- Sequential Circuit Design**
  - **Specification**
  - **Formulation**
  - **State Assignment**
  - **Flip-Flop Input and Output Equation Determination**
  - **Verification**
- **Part 3 – State Machine Design**

The Design Procedure

- **Specification**
- **Formulation - Obtain a state diagram or state table**
- **State Assignment - Assign binary codes to the states**
- **Flip-Flop Input Equation Determination - Select flip-flop types and derive flip-flop equations from next state entries in the table**
- **Output Equation Determination - Derive output equations from output entries in the table**
- **Optimization - Optimize the equations**
- **Technology Mapping - Find circuit from equations and map to flip-flops and gate technology**
- **Verification - Verify correctness of final design**

Specification

- **Component Forms of Specification**
  - **Written description**
  - **Mathematical description**
  - **Hardware description language***
  - **Tabular description***
  - **Equation description***
  - **Diagram describing operation (not just structure)***
- **Relation to Formulation**
  - **If a specification is rigorous at the binary level (marked with * above), then all or part of formulation may be completed**

Formulation: Finding a State Diagram

- **A state is an abstraction of the history of the past applied inputs to the circuit (including power-up reset or system reset).**
  - **The interpretation of "past inputs" is tied to the synchronous operation of the circuit. E. g., an input value (other than an asynchronous reset) is measured only during the setup-hold time interval for an edge-triggered flip-flop.**
- **Examples:**
  - **State A represents the fact that a 1 input has occurred among the past inputs.**
  - **State B represents the fact that a 0 followed by a 1 have occurred as the most recent past two inputs.**

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.**

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 Procedure

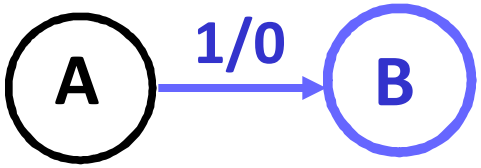
- **To develop a sequence recognizer state diagram:**
  - **Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically "reset" state).**
  - **Add a state that recognizes that the first symbol has occurred.**
  - **Add states that recognize each successive 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."*.**

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., 1101101 or 1101101.**
- **And, the 1 in the middle, 1101101, is in both subsequences.**
- **The sequence 1101 must be recognized each time it occurs in the input sequence.**

Example: Recognize 1101

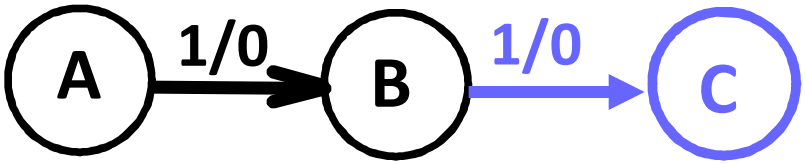
- **Define states for the sequence to be recognized:**
  - **assuming it starts with first symbol,**
  - **continues through each symbol in the sequence to be recognized, and**
  - **uses output 1 to mean the full sequence has occurred,**
  - **with output 0 otherwise.**
- **Starting in the initial state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



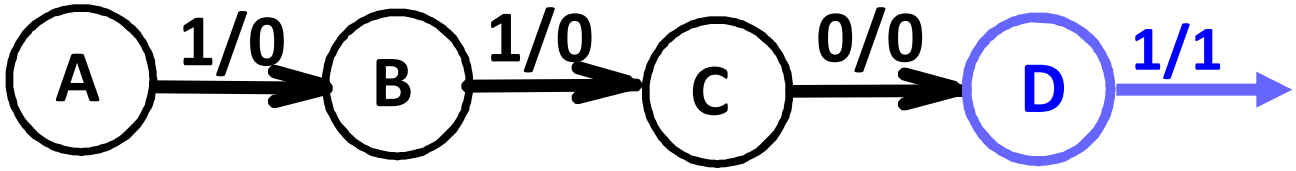
  - **State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.**

Example: Recognize 1101 (continued)

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

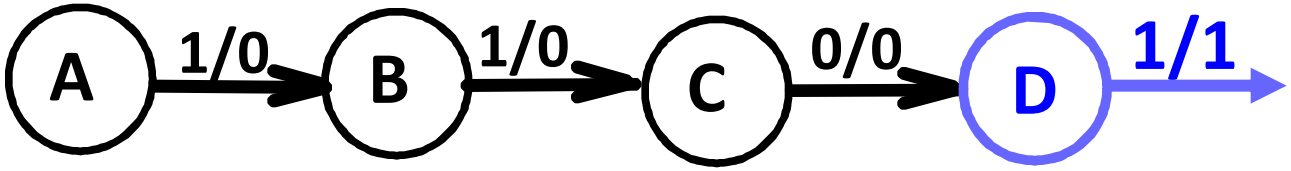


- **Finally, after 110 and a 1, we have:**

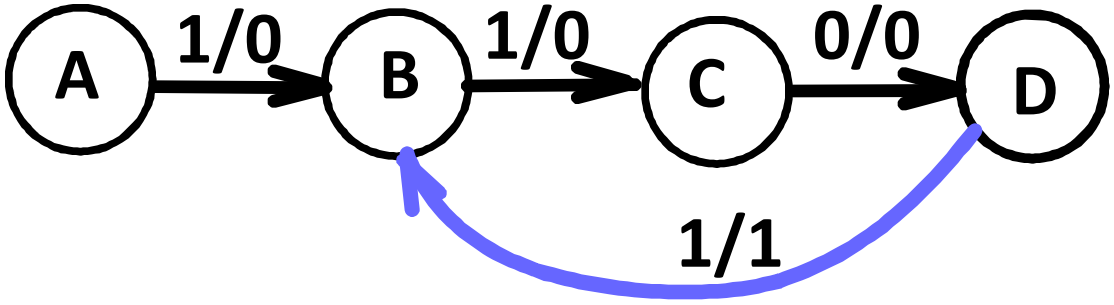


  - **Transition arcs are used to denote the output function (Mealy Model)**
  - **Output 1 on the arc from D means the sequence has been recognized**
  - **To what state should the arc from state D go? Remember: 1101101 ?**
  - **Note that D is the last state but the output 1 occurs for the input applied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

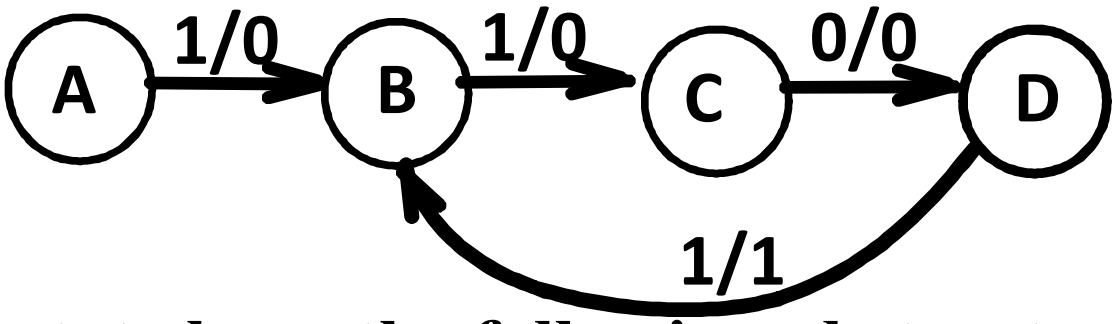
Example: Recognize 1101 (continued)



- **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:**



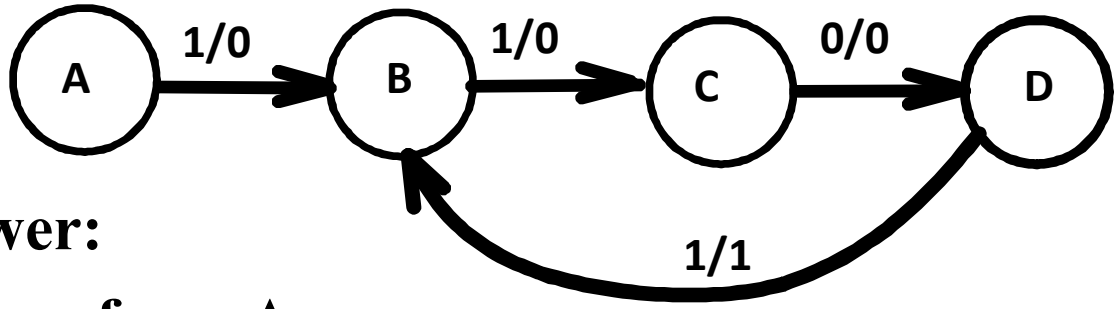
Example: Recognize 1101 (continued)



- **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?**



- **Answer:**

"0" arc from A

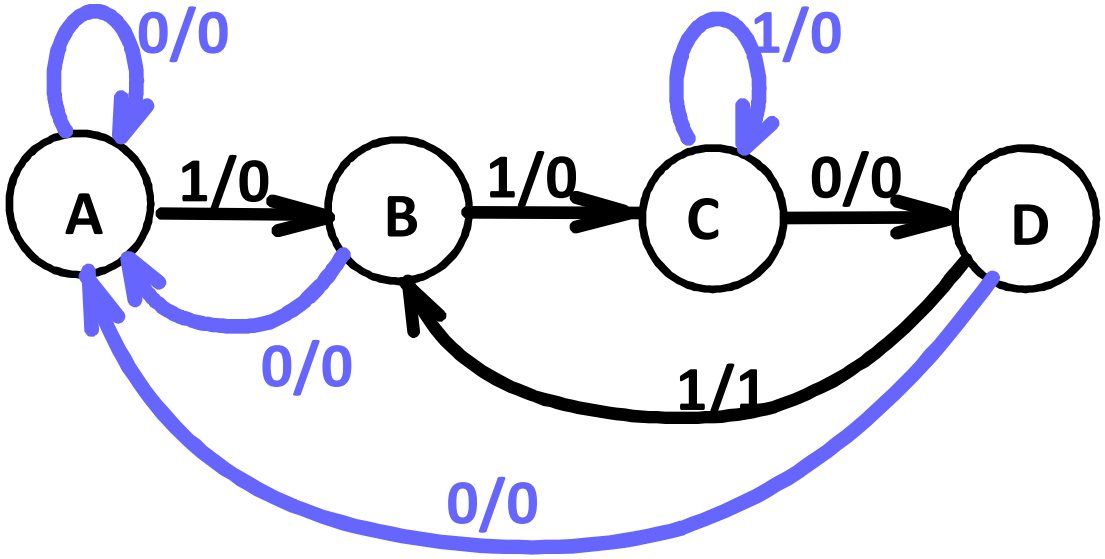
"0" arc from B

"1" arc from C

"0" arc from D.

Example: Recognize 1101 (continued)

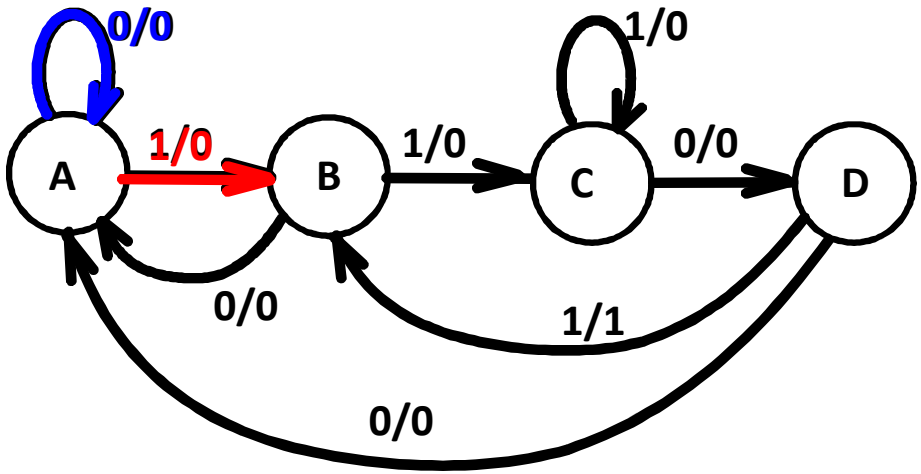
- **State transition arcs must represent the fact that an input subsequence has occurred. Thus we get:**



- **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

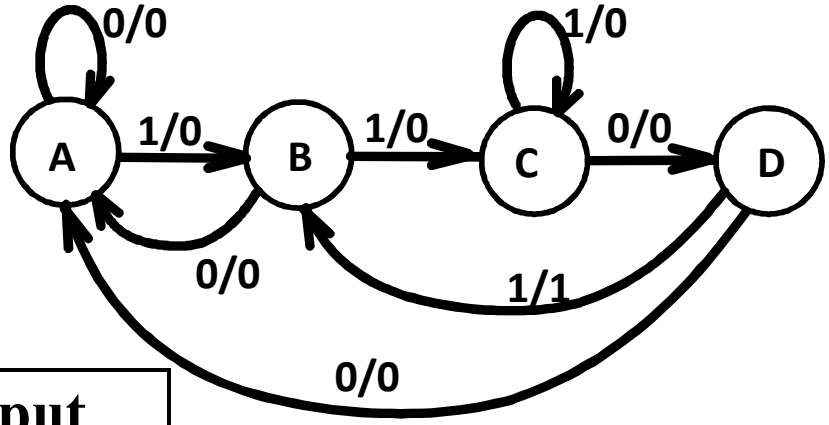
- From the State Diagram, we can fill in the State Table.
- There are 4 states, one input, and one output. We will choose the form with four rows, one for each current state.
- From State A, the 0 and 1 input transitions have been filled in along with the outputs.



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

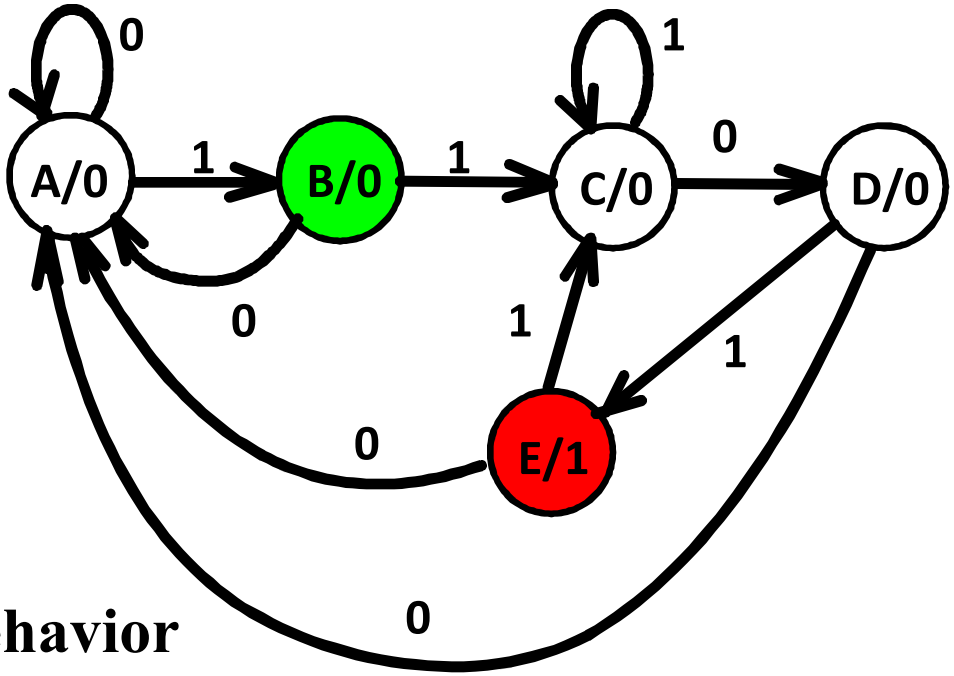
- What would the state diagram and state table look like for the Moore model?

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.**

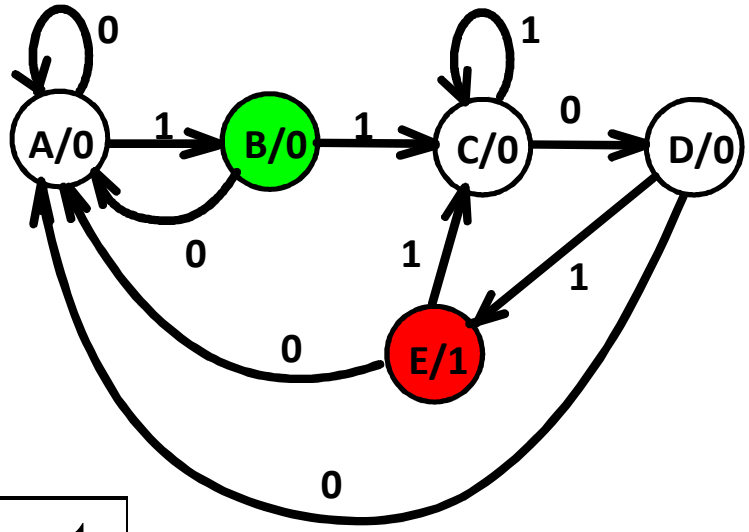
Example: Moore Model (continued)

- We mark outputs on states 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
- Memory aid re more state in the Moore model: "Moore is More."



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

状态表的简化

- 一般情况下，原始状态图和原始状态表中存在着多余的状态。状态个数越多，电路中所需的触发器的数目也越多，制造成本就越高。为降低制造成本，需要去掉多余的状态。
- 所谓状态简化，就是要获得一个最小化的状态表。这个表不仅能正确地反映设计的全部要求，而且状态的数目最少。

状态等效的含义

- 完全确定状态表： 状态表中的次态和输出都有确定的状态和确定的输出值。
- 等效状态：设状态S1和S2是完全确定状态表中的两个状态,如果对于所有可能的输入序列，分别从状态S1和状态S2出发，所得到的输出响应序列完全相同，则状态S1和S2是等效的，记作(S1, S2), 或者说，状态S1和S2是等效对。等效状态可以合并。这里“所有可能的输入序列”是指长度和结构是任意的，它包含无穷多位，且有无穷多种组合。

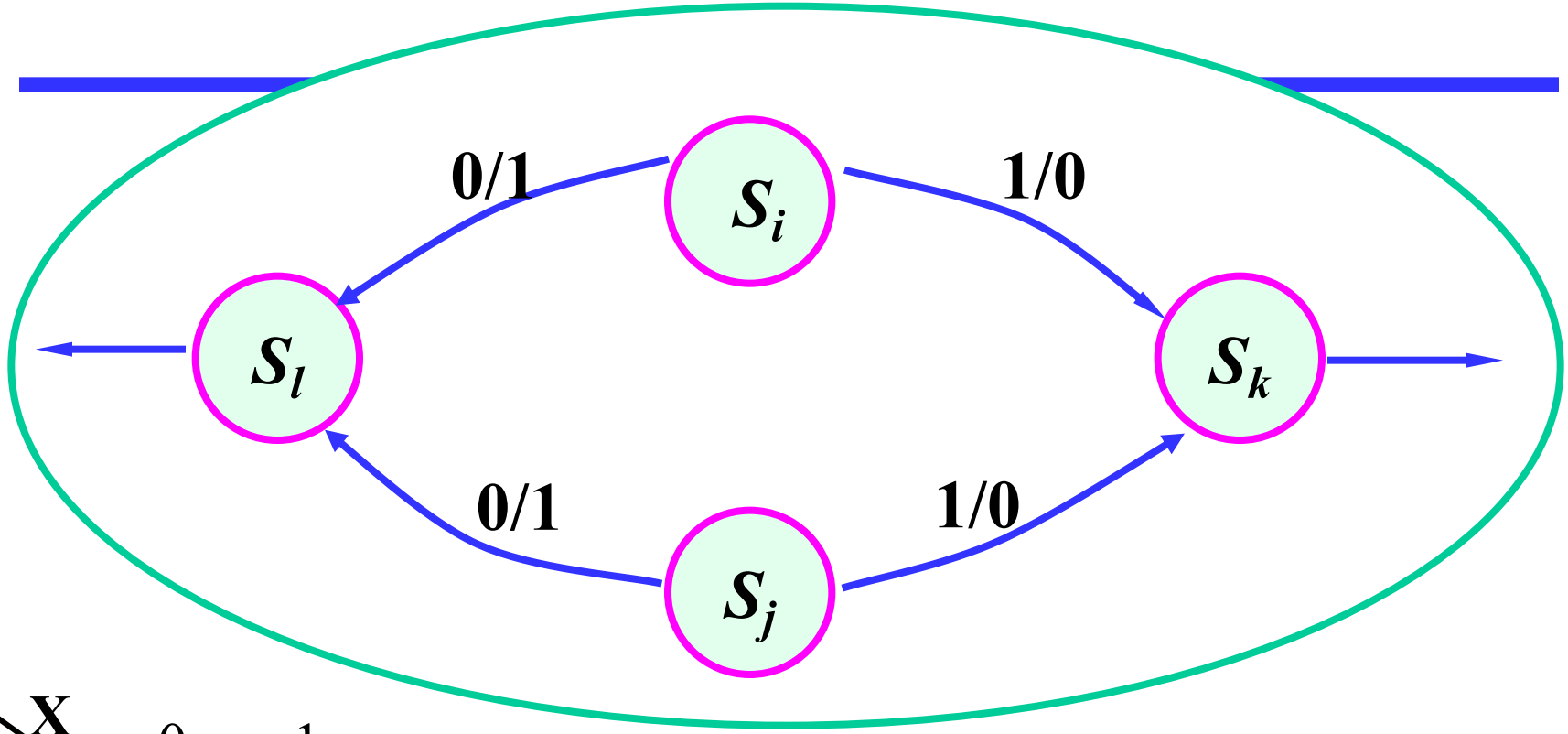
“状态等效”判别

- “状态等效”的三种情况：

在所有输入情况下，

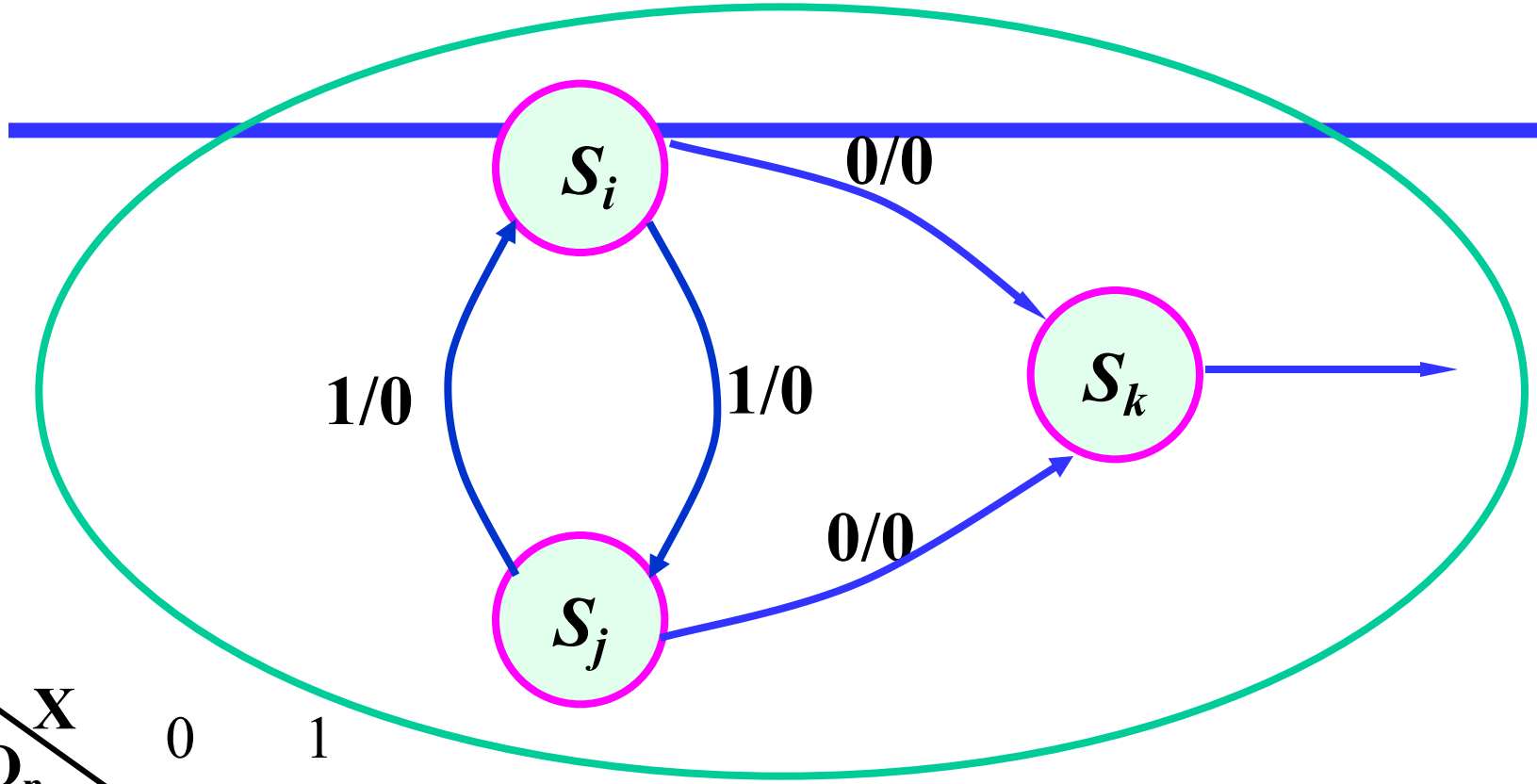
首先　　输出相同

同时　　或次态相同
　　或次态交错
　　或次态循环



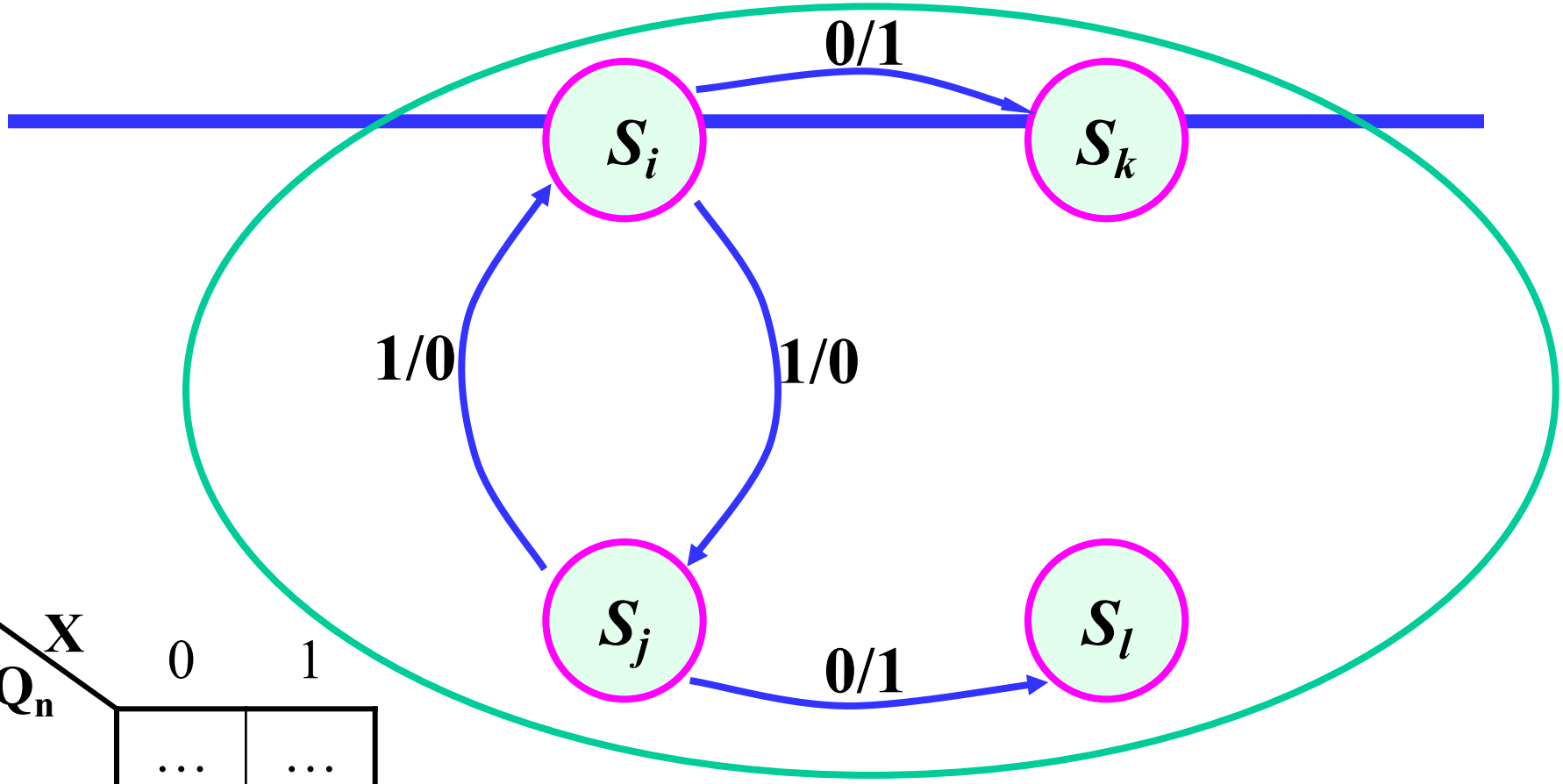
Q_n \ X	0	1
	…	…
S_i	S_l/1	S_k/0
S_j	S_l/1	S_k/0
	…	…

次态相同



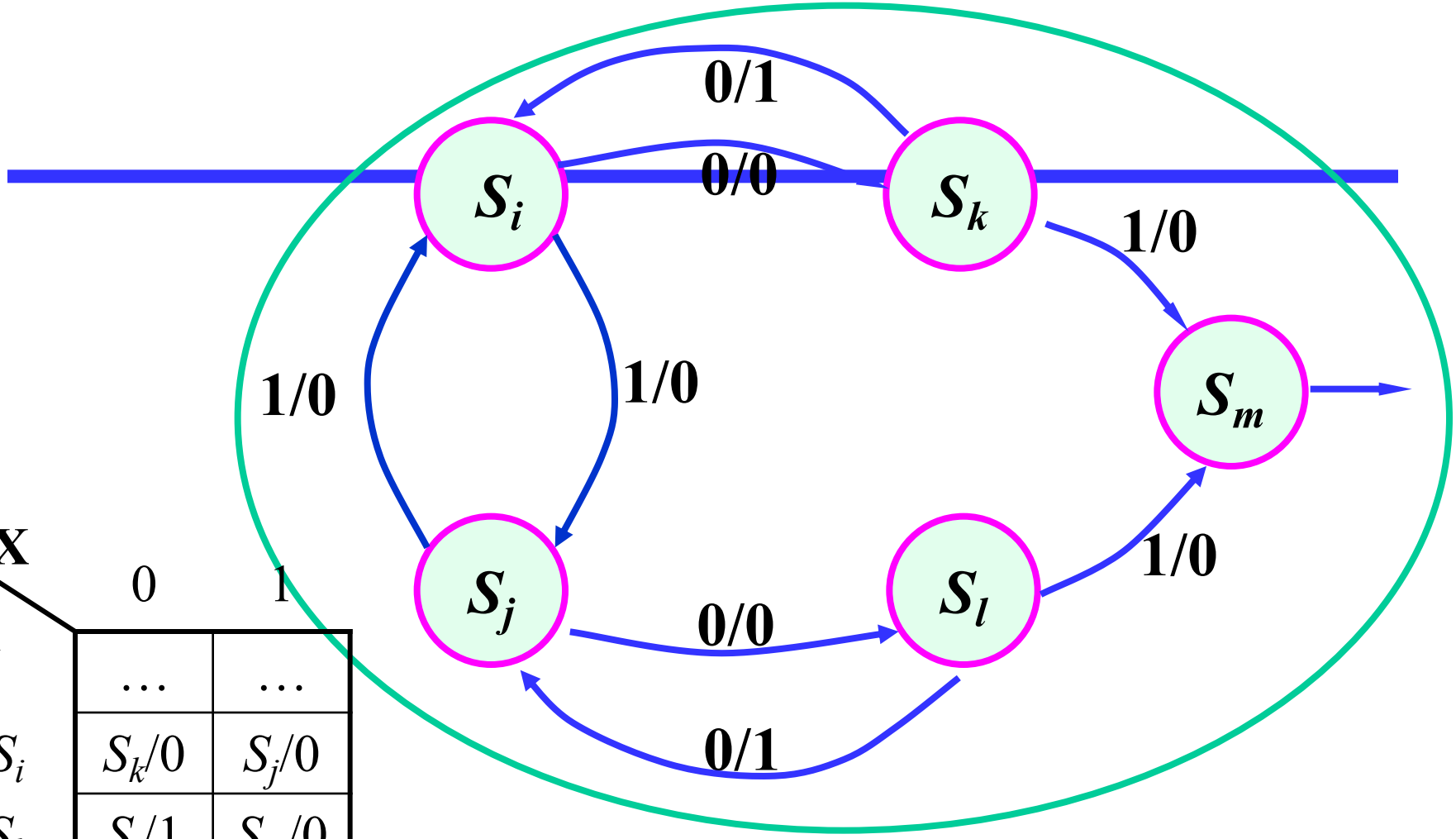
Q_n \ X	0	1
	…	…
S_i	S_k/0	S_j/0
S_j	S_k/0	S_i/0
	…	…

次态相同或交错



Q_n \ X	0	1
	…	…
S_i	S_k/1	S_j/0
S_j	S_l/1	S_i/0
	…	…

次态交错或等效(S_k,S_l等效)



Q_n \ X	0	1
	…	…
S_i	$S_k/0$	$S_j/0$
S_k	$S_i/1$	$S_m/0$
S_l	$S_j/1$	$S_m/0$
S_j	$S_l/0$	$S_i/0$
	…	…

次态交错或相同或循环

1、观察法化简

- 例：简化下表所示的状态表

现态	次态/输出	
	X=0	X=1
A	A/0	B/0
B	A/0	C/0
C	A/0	D/1
D	A/0	D/1

1、观察法化简（续）

- A和B，C和D的输出完全相等；
- C和D在输入的各种取值组合下，次态相同，因此C和D等效；
- A和B在X=1时的次态不满足四条件之一，因此A和B不等效；
- 最大等效类为{A}，{B}，{C,D},分别用A', B', C'表示；

1、观察法化简（续）

- 最小化状态表为

现态	次态/输出	
	X=0	X=1
A'	A'/0	B'/0
B'	A'/0	C'/0
C'	A'/0	C'/1

2、隐含表法化简

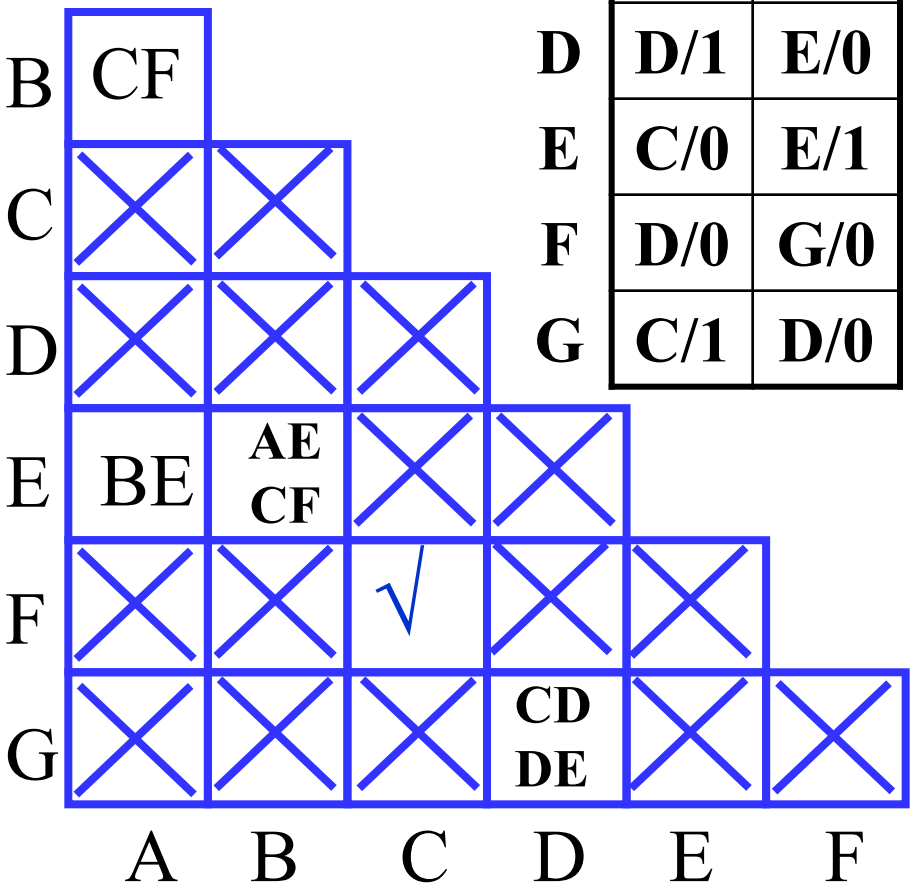
- 例：简化下表所示的状态表

现态	次态/输出	
	X=0	X=1
A	C/0	B/1
B	F/0	A/1
C	D/0	G/0
D	D/1	E/0
E	C/0	E/1
F	D/0	G/0
G	C/1	D/0

2、隐含表法化简（续）

Q_n \ X	0	1
A	C/0	B/1
B	F/0	A/1
C	D/0	G/0
D	D/1	E/0
E	C/0	E/1
F	D/0	G/0
G	C/1	D/0

- 作隐含表
- 顺序比较，寻找等效状态对
  - 状态对等效，打“√”；
  - 状态对不等效，打“×”；
  - 状态对是否等效需进一步检查，则标记次态对。



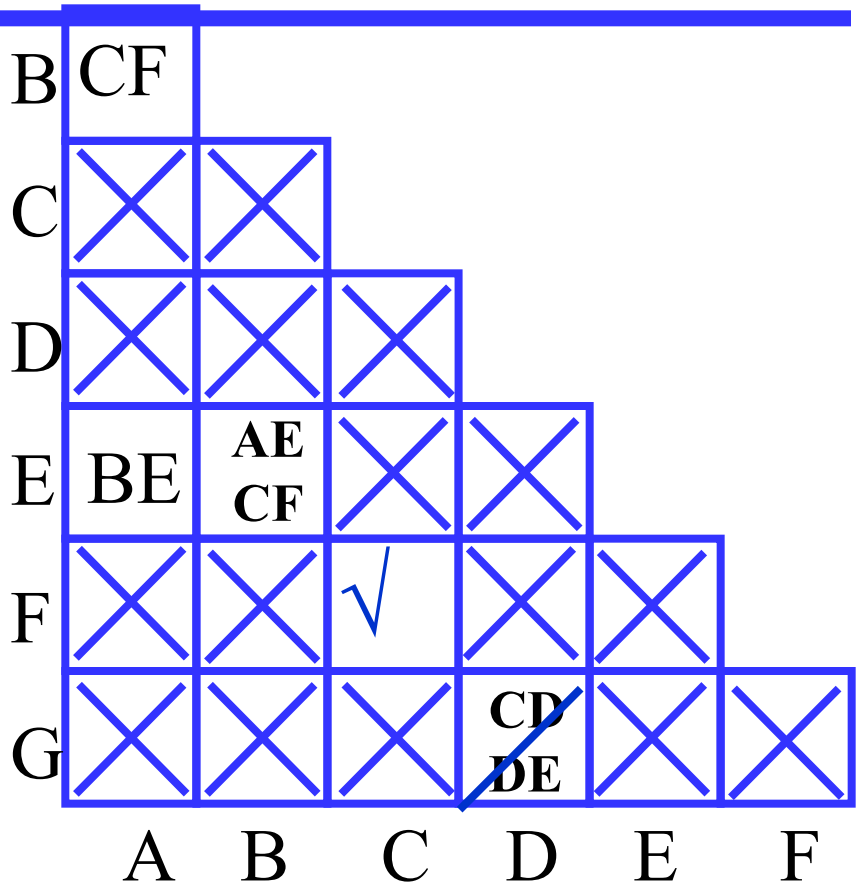
2、隐含表法化简（续）

- 进行关连比较，
  确定等效状态对

由于CD、DE不等效，所以DG不等效，画斜线标志

AB→CF

AE→BE→CF



处于循环链中的每一个状态对都是等效状态对。

2、隐含表法化简（续）

- 确定最大等效类，作最小化状态表
  - 四个等效对 **(A,B), (A,E), (B,E), (C,F)**
  - 最大等效类**(A,B,E)**
  - 四个状态 **(A,B,E), (C,F), (D), (G)**
  - 令以上四个状态依次为**a, b, c, d**

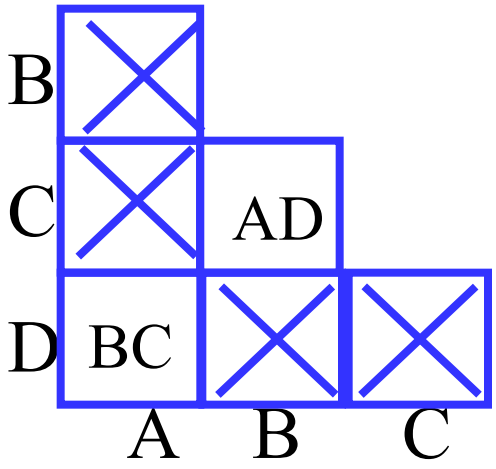
2、隐含表法化简（续）

- 最小化状态表：

现态	次态/输出	
	x=0	x=1
a	b/0	a/1
b	c/0	d/0
c	c/1	a/0
d	b/1	c/0

例：下表状态化简

- (AD)、(BC)输出相同，
  次态循环，故状态A、D等效，
  状态B、C等效。



Q \ X	0	1
A	A/0	B/0
B	A/1	C/0
C	D/1	C/0
D	A/0	C/0

Q \ X	0	1
A	A/0	B/0
B	A/1	B/0

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)

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

Present State $Y_1\ Y_2$	Next State $D_1\ D_2$ x = 0	Next State $D_1\ D_2$ x = 1	Output x = 0	Output 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

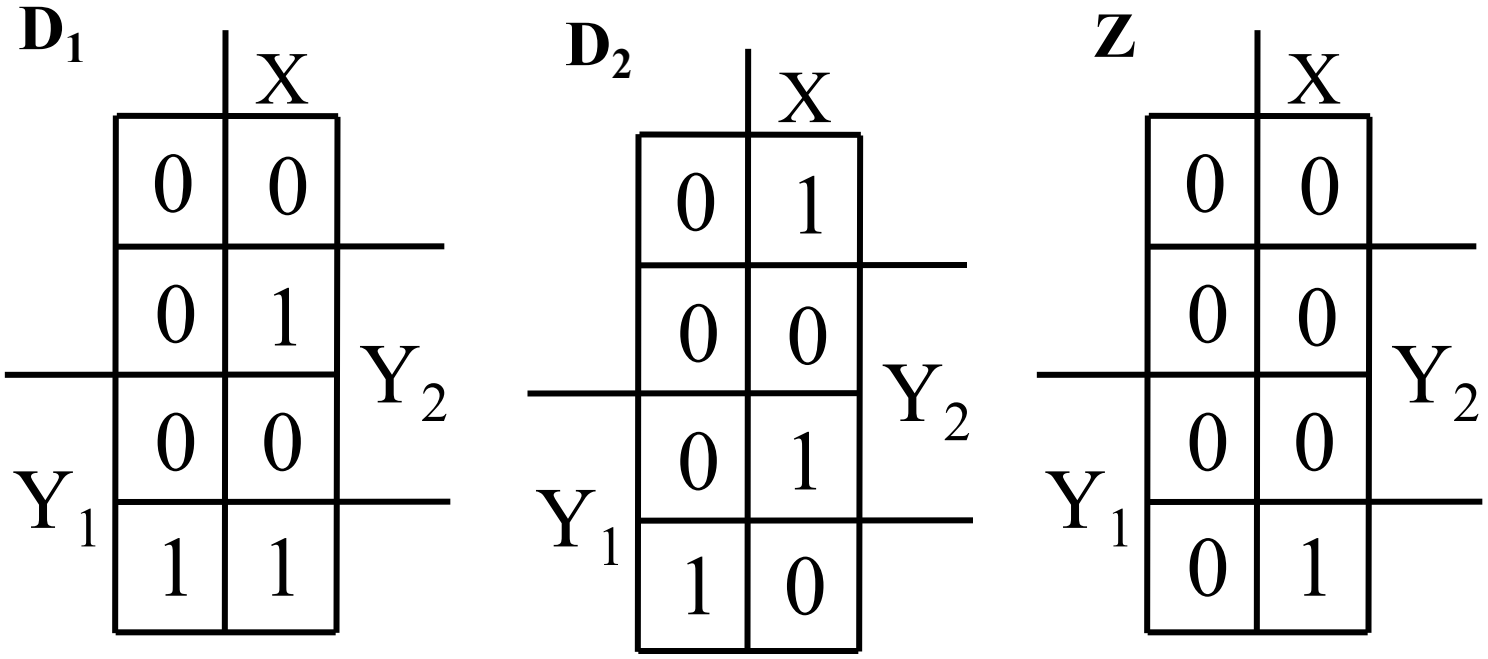
State Assignment – Example 2 (continued)

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

Present State $Y_1\ Y_2$	Next State $x=0$ $D_1\ D_2$	Next State $x=1$ $D_1\ D_2$	Output $x=0$	Output $x=1$
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

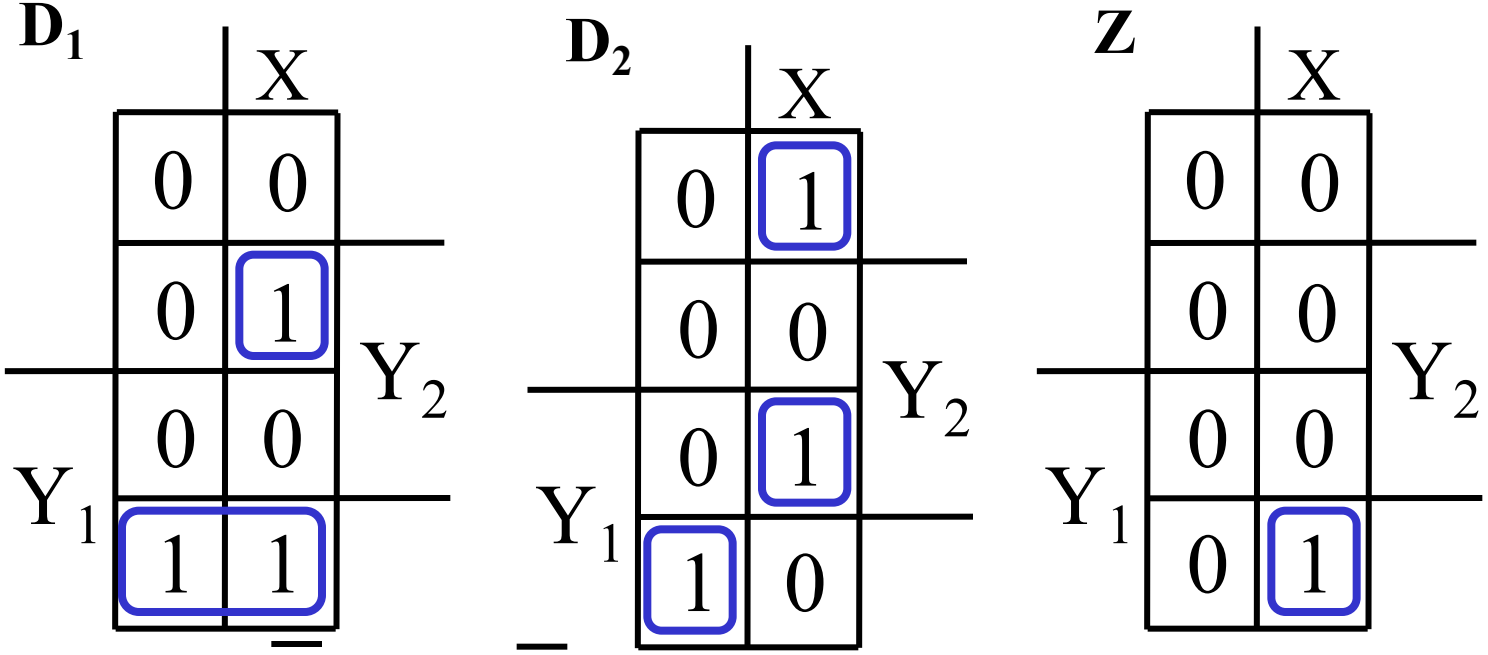
Find Flip-Flop Input and Output Equations: Example 2 – Counting Order Assignment

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



Optimization: Example 2: Counting Order Assignment

- **Performing two-level optimization:**



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

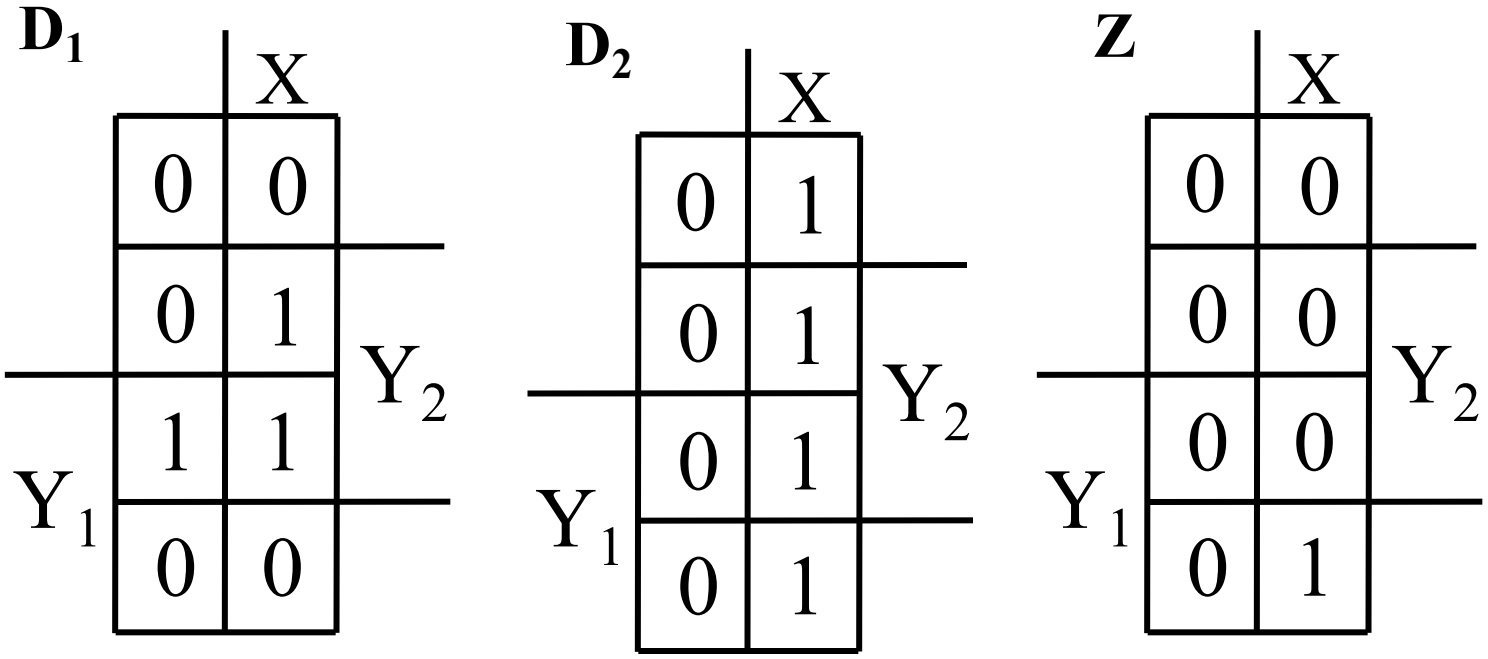
$$D_2 = \overline{X}Y_1\overline{Y}_2 + X\overline{Y}_1\overline{Y}_2 + XY_1Y_2$$

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

Gate Input Cost = 22

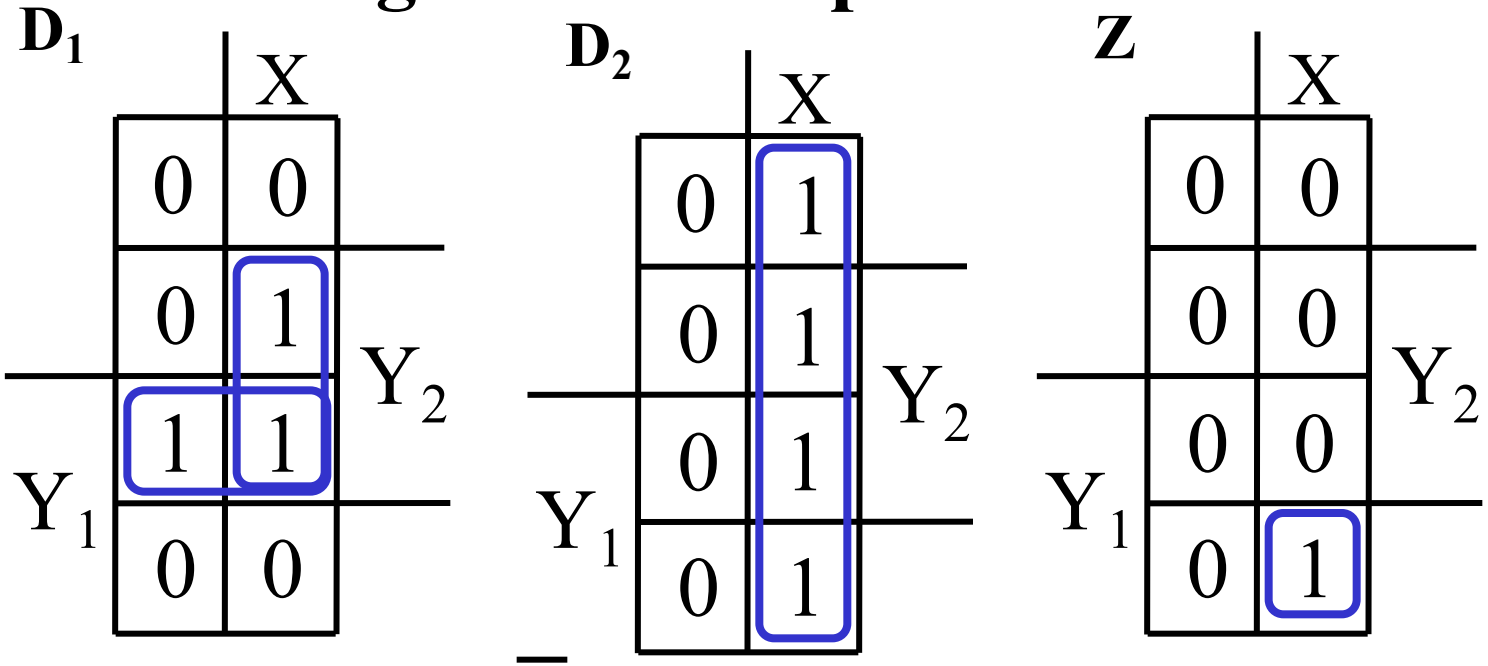
Find Flip-Flop Input and Output Equations: Example 2 – Gray Code Assignment

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



Optimization: Example 2: Assignment 2

- **Performing two-level optimization:**



$D_1 = Y_1Y_2 + X\bar{Y}_2$ **Gate Input Cost = 9**

$D_2 = X$ **Select this state assignment to**

$Z = XY_1\bar{Y}_2$ **complete design in slide**

状态分配

- 通常情况下，状态分配的方案不一样，所得到的输出函数和激励函数的表达式也不同，由此而设计出来的电路复杂度也不同。状态分配的任务是：
  - 决定编码的长度
  - 寻找一种最佳的或接近最佳的状态分配方案
- 然而，当n较大时，要真正找到最佳的分配方案是十分困难的，况且分配方案的好坏还与所采用的触发器的类型有关。因此，实际应用时都是采用工程近似的方法，依据以下四条件原则来进行状态分配。

状态分配的基本原则

- 在相同输入条件下具有相同次态的现态，应尽可能分配相邻的二进制代码
- 在相邻输入条件，同一现态的次态应尽可能分配相邻的二进制代码
- 输出完全相同的现态应尽可能分配相邻的二进制代码
- 最小化状态表中出现次数最多的状态或初始状态应分配逻辑0

状态分配的基本原则（续）

- 一般情况下，第一条原则较为重要，需优先考虑，其次要考虑由前三条原则得到的应分配相邻代码的状态对出现的次数，次数多的状态对应优先分配相邻的二进制代码。

例：对下表所示的状态表进行状态分配

现态	次态/输出	
	X=0	X=1
A	C/0	D/0
B	C/0	A/0
C	B/0	D/0
D	A/1	B/1

(1)在相同输入条件下具有相同次态的现态，应尽可能分配相邻的二进制代码：A和B，A和C 应相邻

(2)在相邻输入条件，同一现态的次态应尽可能分配相邻的二进制代码： C和D，C和A，B和D，A和B应相邻；

(3) 输出完全相同的现态应尽可能分配相邻的二进制代码： A，B，C 三者应相邻，即A和B，A和C，B和C应相邻；

(4) 最小化状态表中出现次数最多的状态或初始状态应分配逻辑0：A分配为逻辑0

例：对下表所示的状态表进行状态分配（续）

- 确定n=2，2个触发器可表示4个状态
- 确定分配
  - 由规则(1) 得 A和B，A和C 应相邻；
  - 由规则(2)得 C和D，A和C，B和D，A和B应相邻；
  - 由规则(3)得 A，B，C 三者应相邻，即A和B，A和C，B和C应相邻；
  - 由规则(4)得A分配为逻辑0

例：对下表所示的状态表进行状态分配（续）

Y_2 \ Y_1	0	1
0	A	B
1	C	D

	Y_2	Y_1
A	0	0
B	0	1
C	1	0
D	1	1

- (1)A和B，A和C 应相邻
- (2)C和D，A和C，B和D，A和B应相邻；
- (3) A，B，C 三者应相邻，即A和B，A和C，B和C应相邻；
- (4) A分配为逻辑0

例：对下表所示的状态表进行状态分配（续）

- 最后我们可以得到二进制状态表

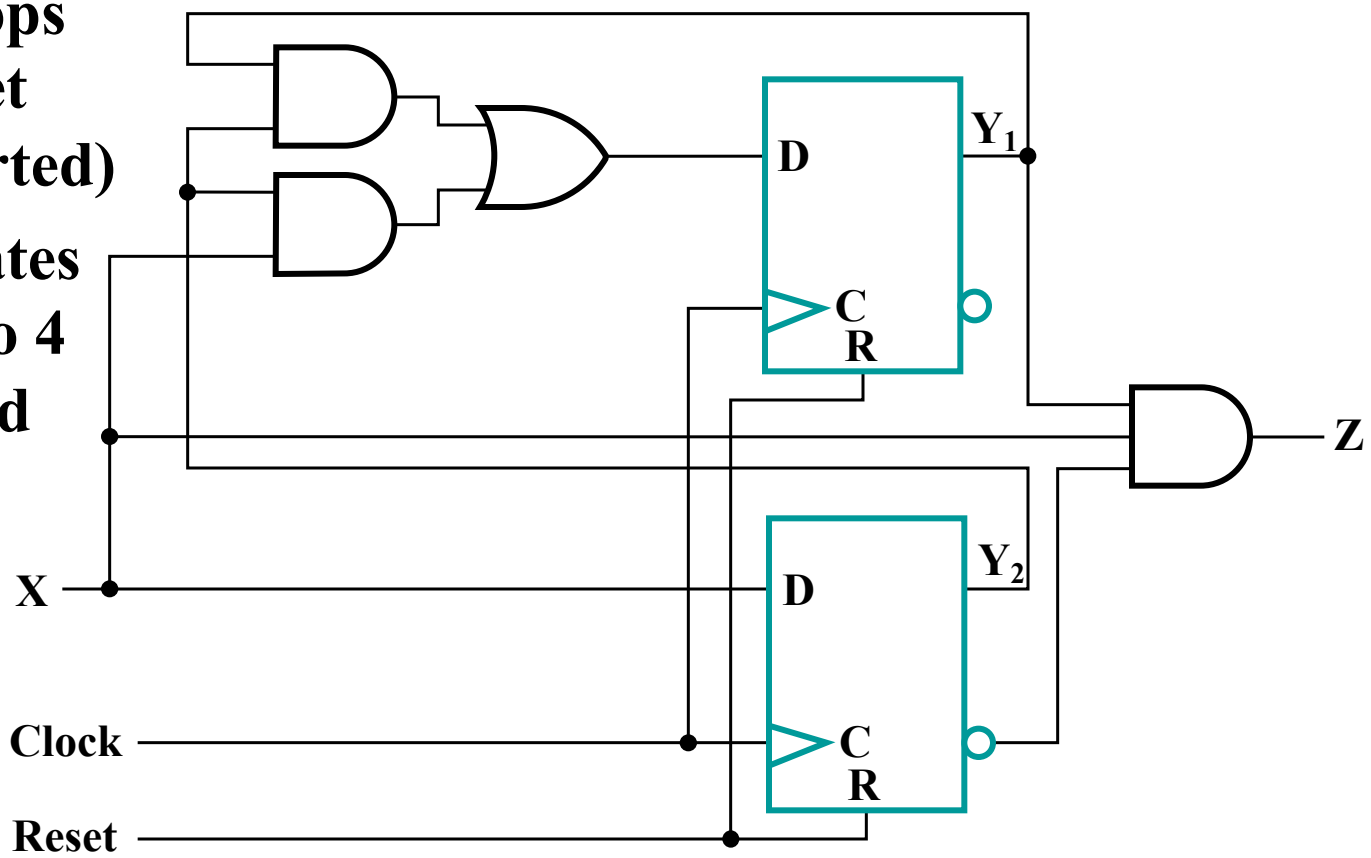
现态	次态/输出	
	X=0	*X*=1
A	C/0	D/0
B	C/0	A/0
C	B/0	D/0
D	A/1	B/1

现态		次态$y_2^{(t+1)}y_1^{(t+1)}$/输出	
y_2	y_1	*X*=0	*X*=1
0	0	10/0	11/0
0	1	10/0	00/0
1	1	00/1	01/1
1	0	01/0	11/0

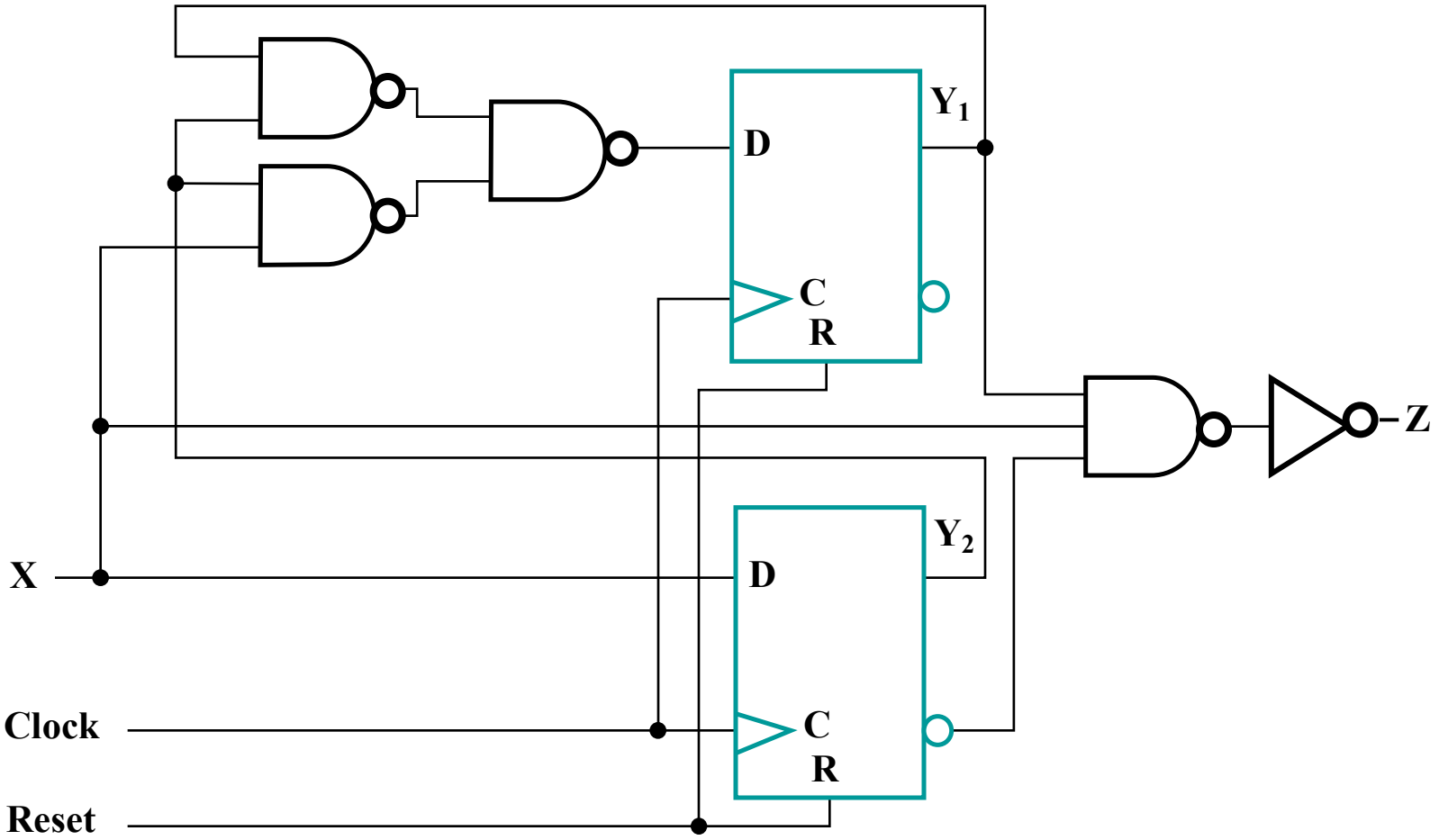
- 有时满足分配原则的分配方案不唯一，这时可任选一种

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

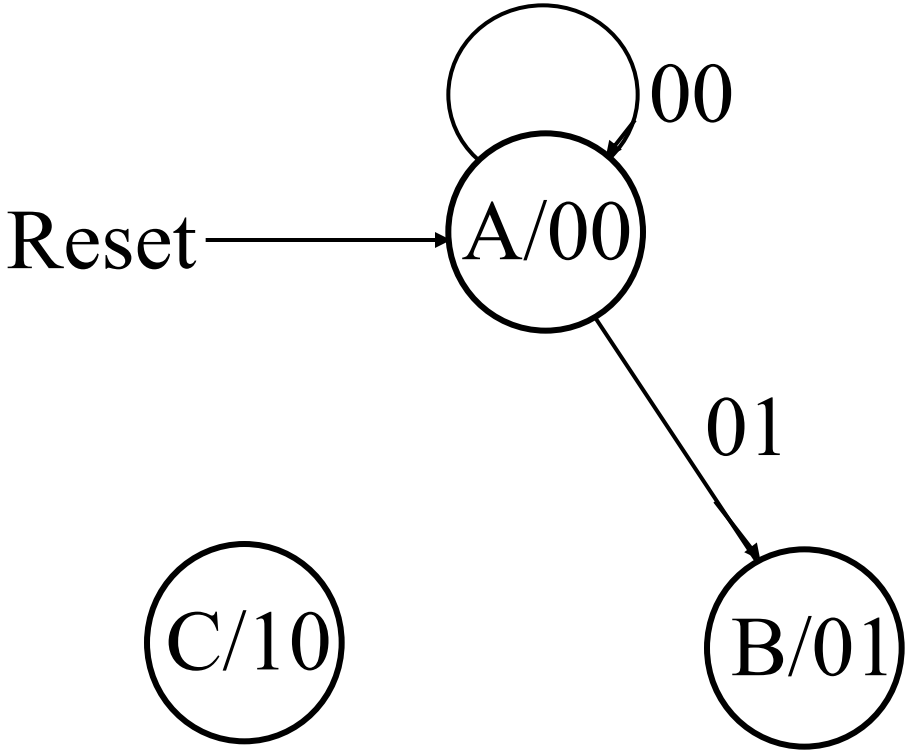


Sequential Design: Example 3

- **Design a sequential modulo 3 accumulator for 2-bit operands**
- **Definitions:**
  - **Modulo *n* adder - an adder that gives the result of the addition as the remainder of the sum divided by *n***
    - **Example: 2 + 2 modulo 3 = remainder of 4/3 = 1**
  - **Accumulator - a circuit that "accumulates" the sum of its input operands over time - it adds each input operand to the stored sum, which is initially 0.**
- **Stored sum: (Y_1,Y_0), Input: (X_1,X_0), Output: (Z_1,Z_0)**

Example 3 (continued)

- **Complete the state diagram:**



Example 3 (continued)

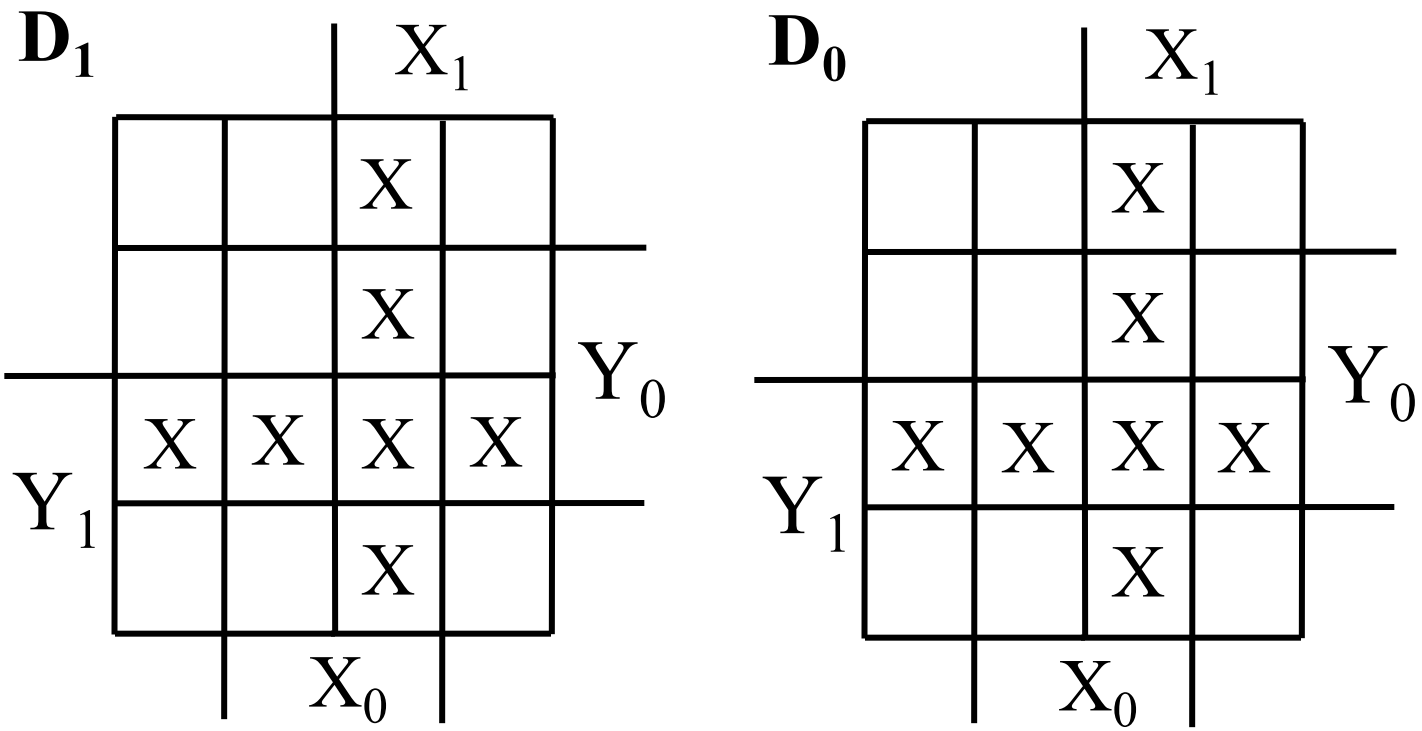
- **Complete the state table**

Y_1Y_0 \ X_1X_0	00	01	11	10	Z_1Z_0
	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	
A (00)	00		X		00
B (01)			X		01
- (11)	X	X	X	X	11
C (10)			X		10

- **State Assignment: $(Y_1,Y_0) = (Z_1,Z_0)$**
- **Codes are in gray code order to ease use of K-maps in the next step**

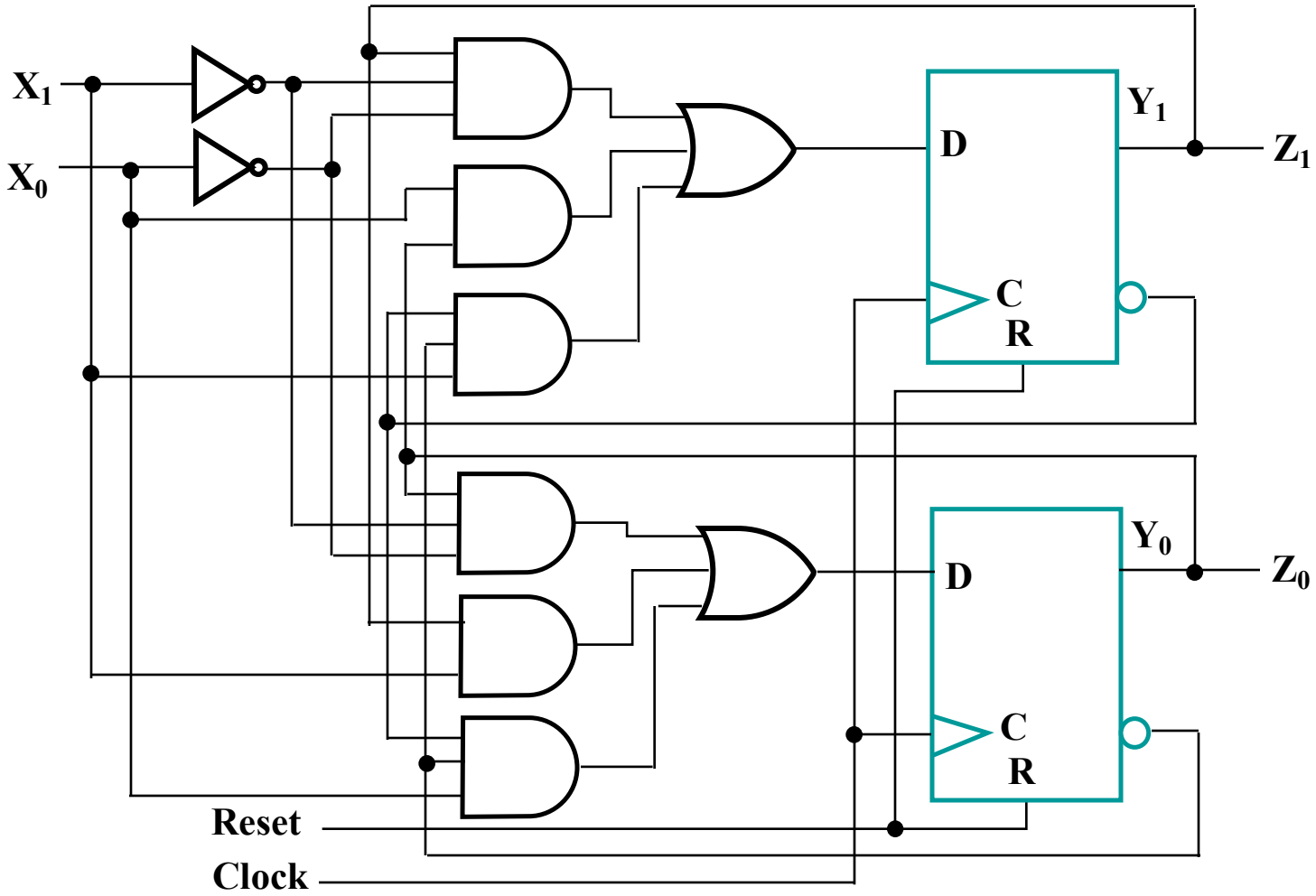
Example 3 (continued)

- **Find optimized flip-flop input equations for D flip-flops**



- **D_1 =**
- **D_0 =**

Circuit - Final Result with AND, OR, NOT



Other Flip-Flop Types

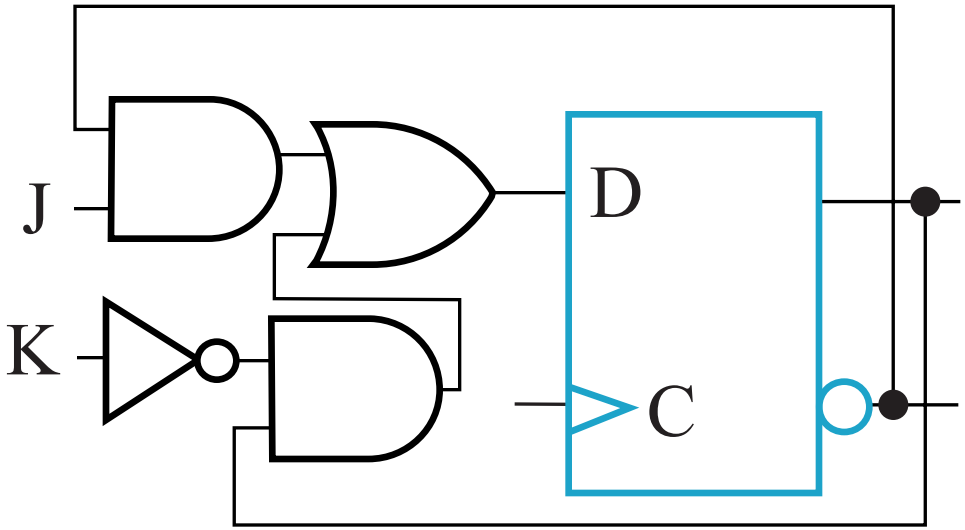
- **J-K and T flip-flops**
  - **Behavior**
  - **Implementation**
- **Basic descriptors for understanding and using different flip-flop types**
  - **Characteristic tables**
  - **Characteristic equations**
  - **Excitation tables**
- **For actual use, see Reading Supplement - Design and Analysis Using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

J-K Flip-flop

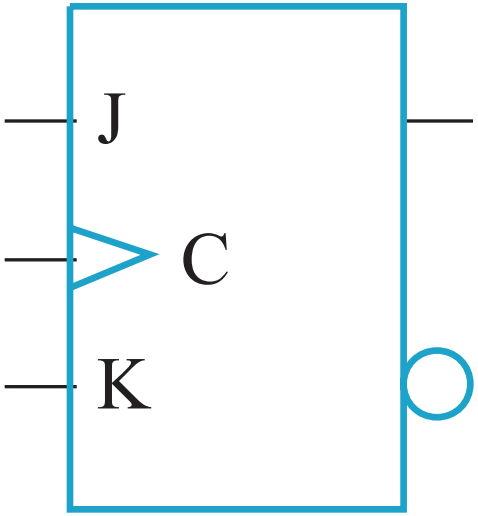
- **Behavior**
  - **Same as S-R flip-flop with J analogous to S and K analogous to R**
  - **<u>Except</u> that J = K = 1 is allowed, and**
  - **For J = K = 1, the flip-flop changes to the *opposite state***
  - **As a master-slave, has same "1s catching" behavior as S-R flip-flop**
  - **If the master 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**

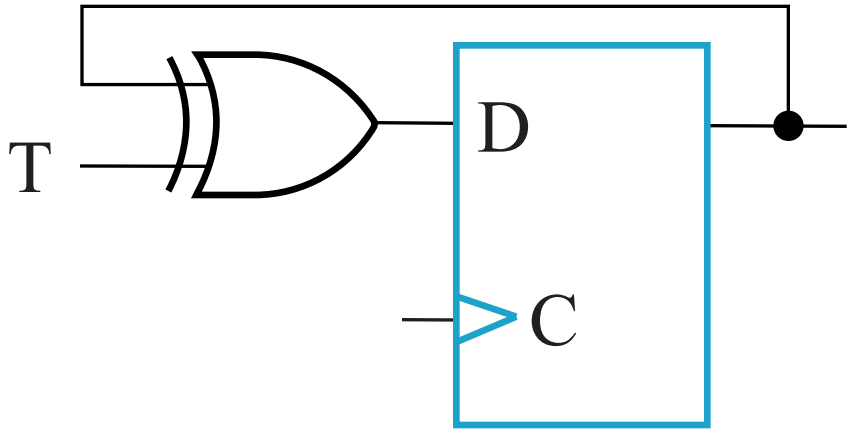


T Flip-flop

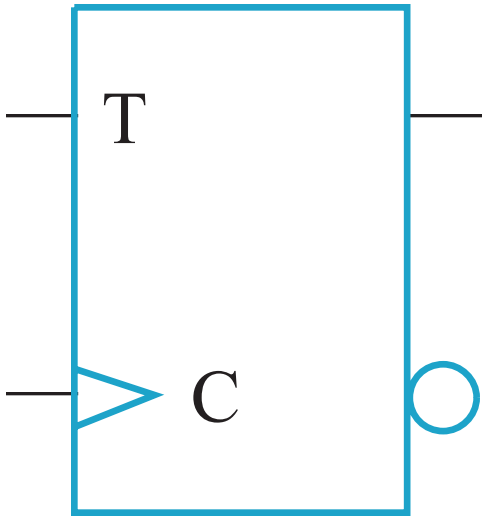
- **Behavior**
  - **Has a single input T**
    - **For T = 0, no change to state**
    - **For T = 1, changes to opposite state**
- **Same as a J-K flip-flop with J = K = T**
- **As a master-slave, has same "1s catching" behavior as J-K flip-flop**
- **Cannot be initialized to a known state using the 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	$\overline{Q}(t)$	Complement

- **Characteristic Equation**

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

- **Excitation Table**

Q(t+1)	T	Operation
$Q(t)$	0	No change
$\overline{Q}(t)$	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**

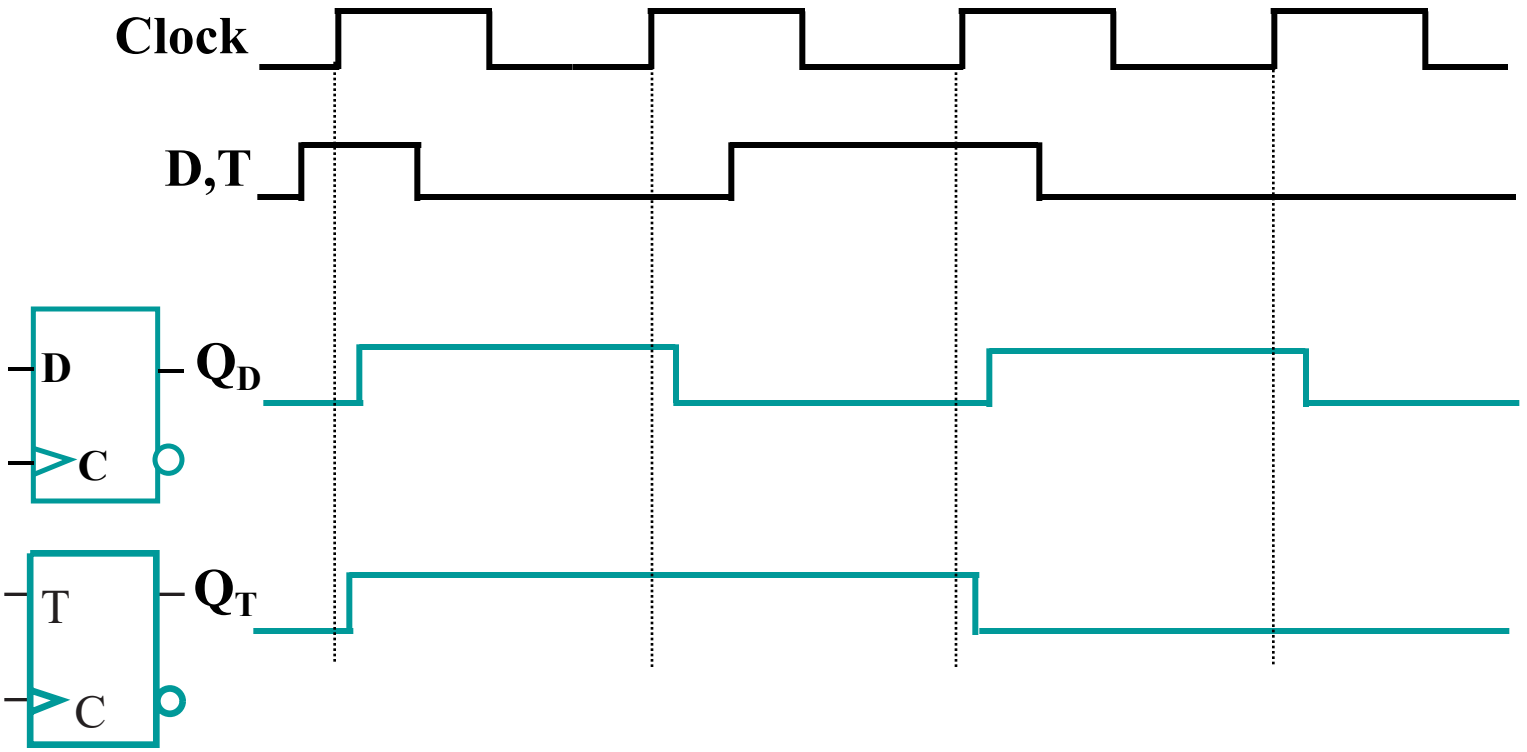
$$Q(t+1) = J\,\overline{Q} + \overline{K}\,Q$$

- **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

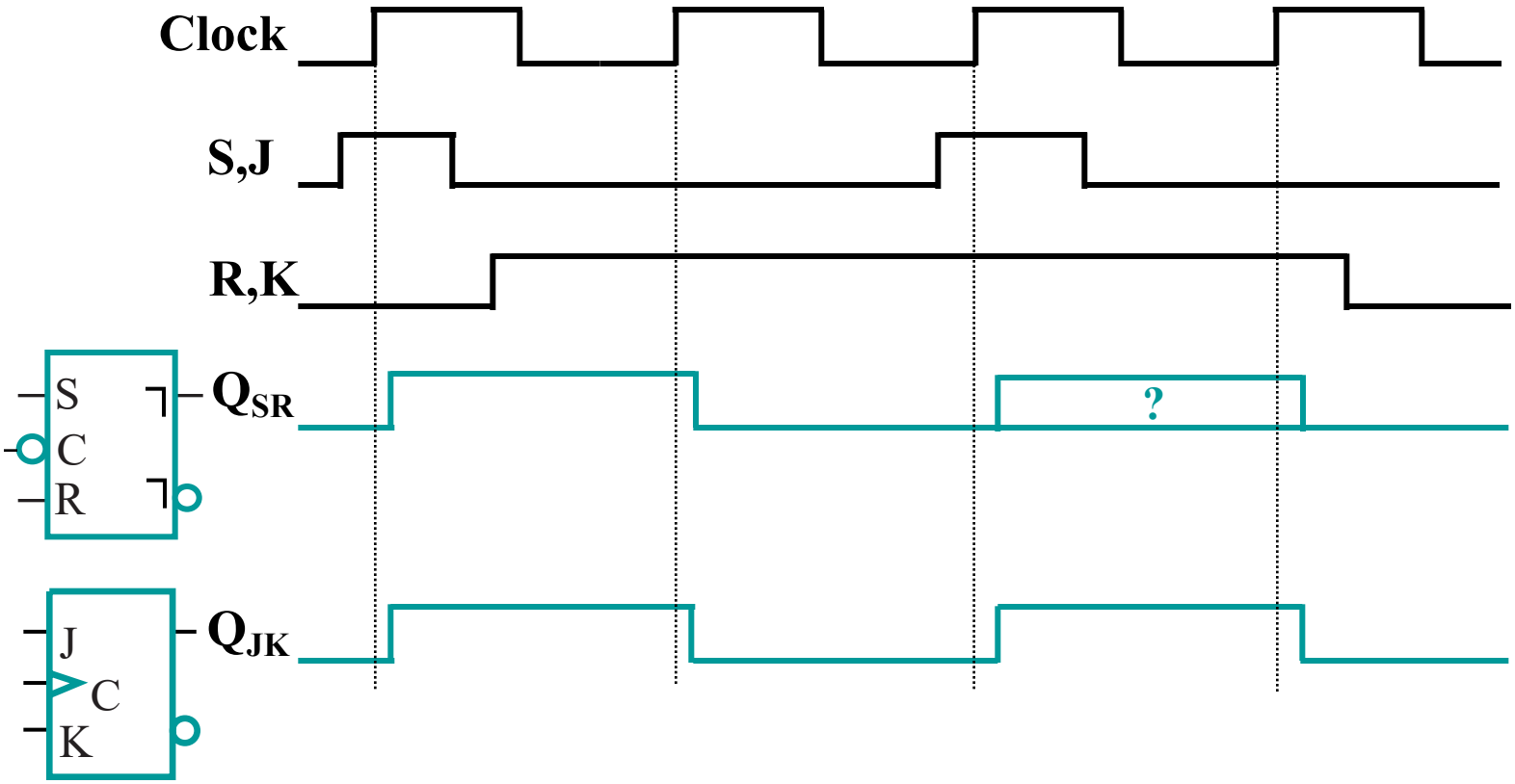
Flip-flop Behavior Example

- **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:



Assignments

- **4-21; 4-22; 4-25; 4-29**