Logic and Computer Design Fundamentals

Sequential Circuit Design

Charles Kime & Thomas Kaminski

(Hyperlinks are active in View Show mode)

Overview

- Part 1 - Storage Elements
- Part 2 - Sequential Circuit Analysis
- Part 3- Sequential Circuit Design
  - Specification
  - Formulation
  - State Assignment
  - Flip-Flop Input and Output Equation Determination
  - Verification
- Part 4 – 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**

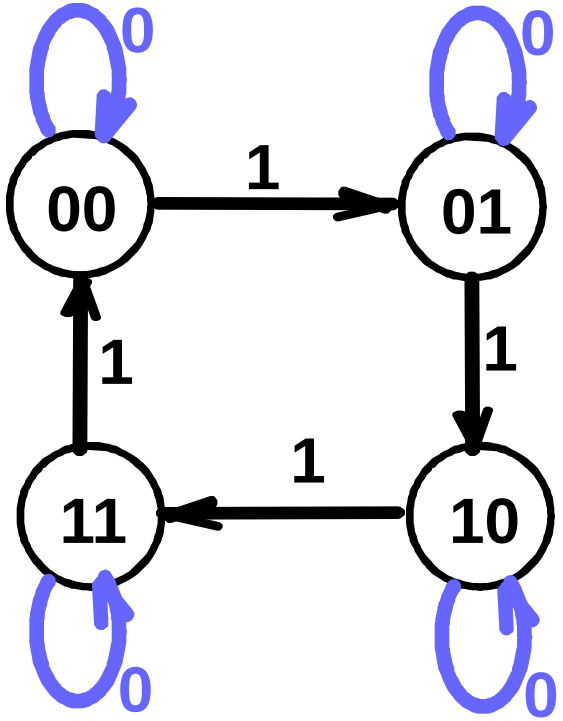
Example 1: Sequential Circuit Design 2-bit Counter using D-Flip Flops

Design a Counter using D Flip-Flop:

A	B	x	A	B	DA	DB
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0

Same

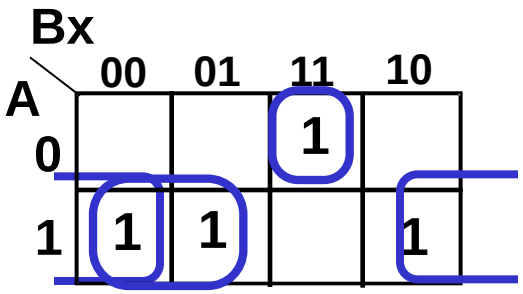
Find the optimized inputs of the two D flip-flops using K-map



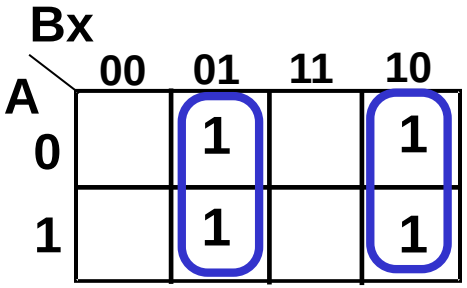
Excitation Table:

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

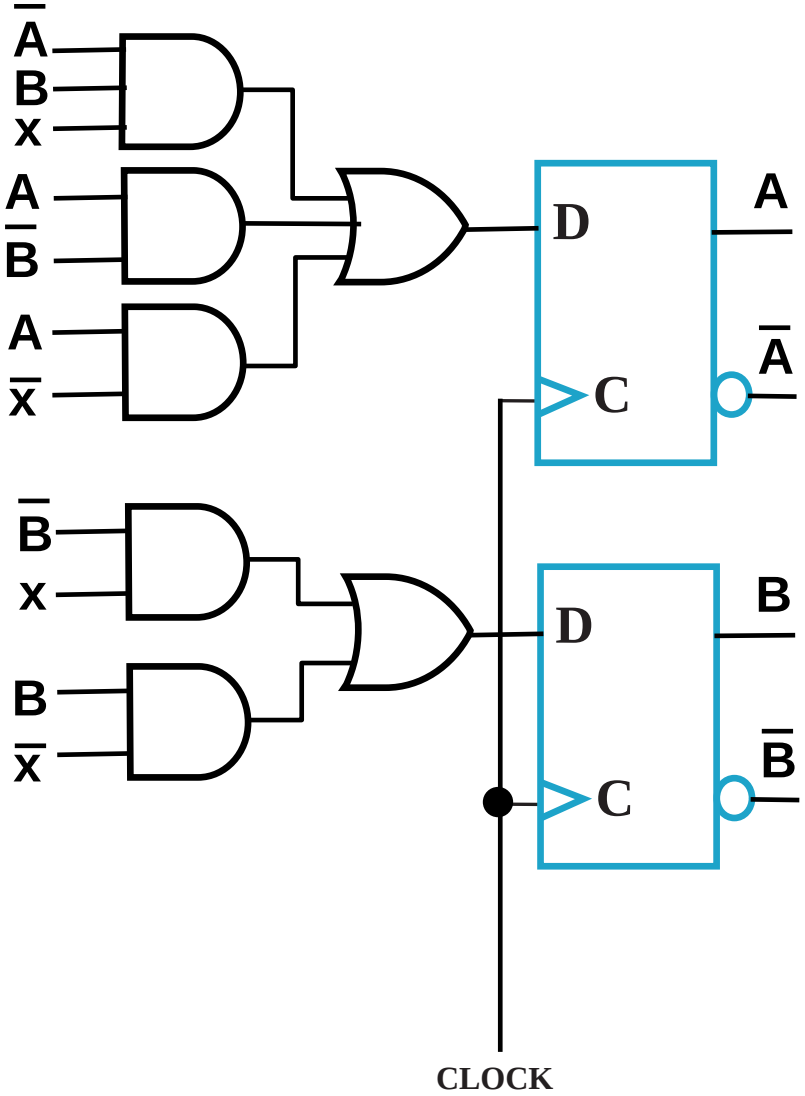
Example 1: Sequential Circuit Design 2-bit Counter using D-Flip Flops



$D_A = A'Bx + AB' + Ax'$



$D_B = B'x + Bx'$

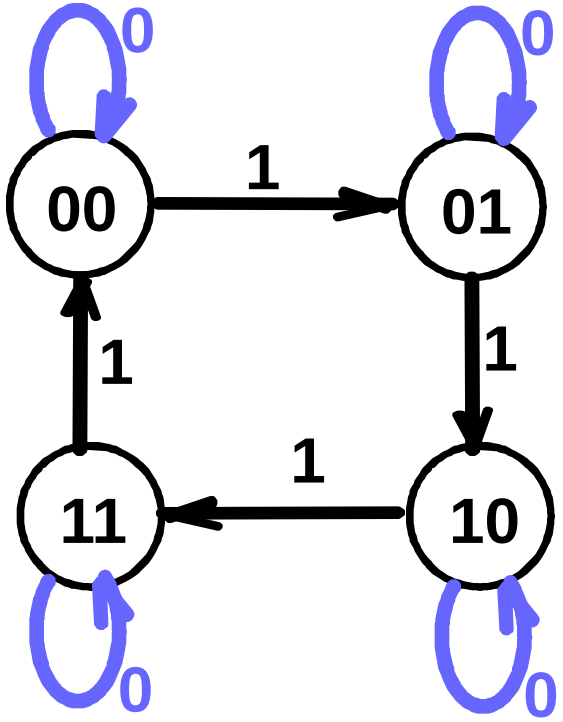


Example 2: Sequential Circuit Design 2-bit Counter using JK-Flip Flops

Design a Counter using JK Flip-Flop:

A	B	x	A	B	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	1	0	X	X	0
0	1	1	1	0	1	X	X	1
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

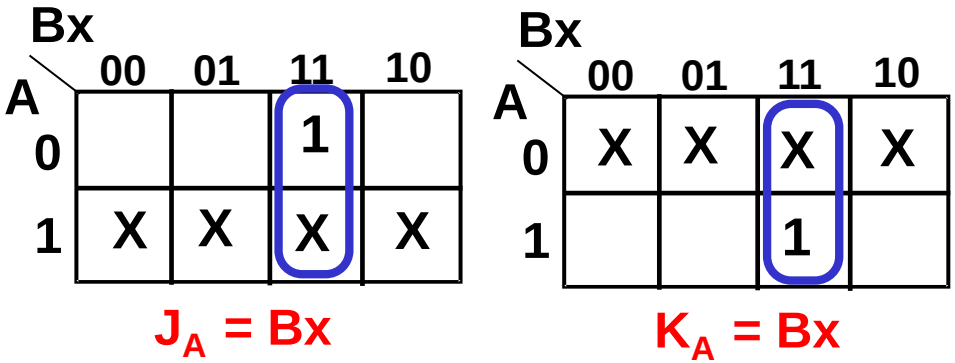
Find the optimized inputs of the two JK flip-flops using K-map



Excitation Table:

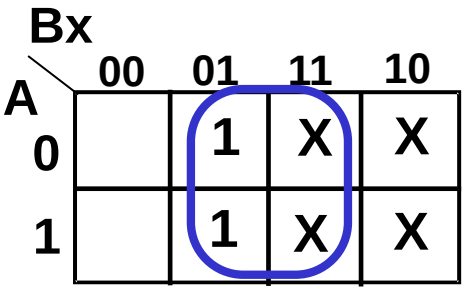
Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

Example 2: Sequential Circuit Design 2-bit Counter using JK-Flip Flops

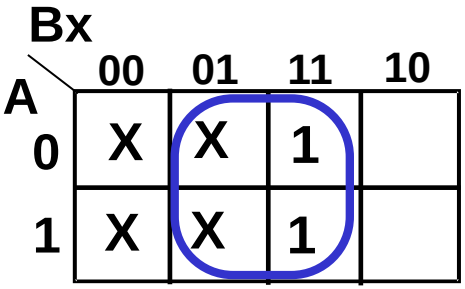


$J_A = Bx$

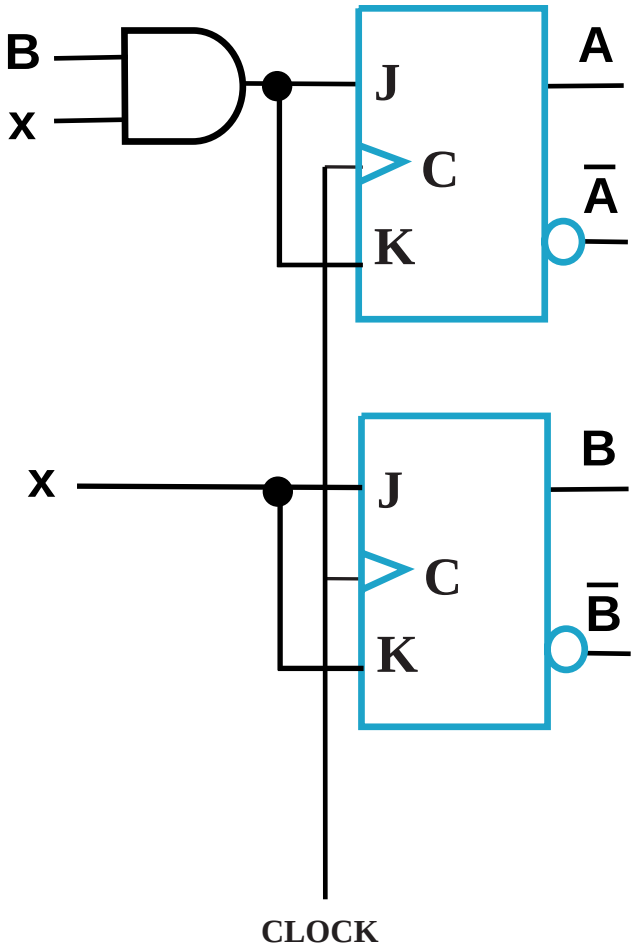
$K_A = Bx$



$J_B = x$



$K_B = x$



Example 3: Sequence Recognizer

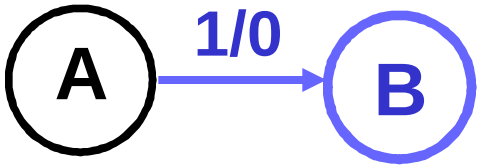
- **To develop a sequence recognizer state diagram:**
  - **Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically "reset" state).**
  - **Add a state that recognizes that the first symbol has occurred.**
  - **Add states that recognize each successive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

Example 3: Sequence Recognizer

- **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 3: Recognize 1101

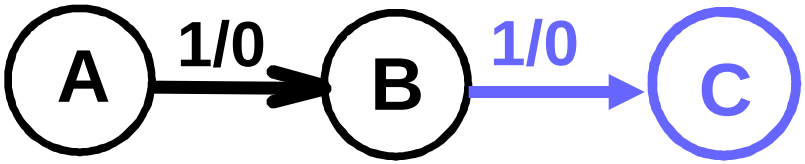
- **Define states for the sequence to be recognized:**
  - **assuming it starts with first symbol,**
  - **continues through each symbol in the sequence to be recognized, and**
  - **uses output 1 to mean the full sequence has occurred,**
  - **with output 0 otherwise.**
- **Starting in the initial state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



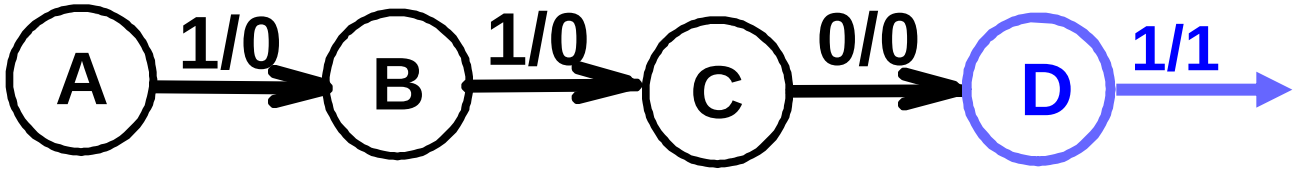
  - **State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.**

Example 3: Recognize 1101 (continued)

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

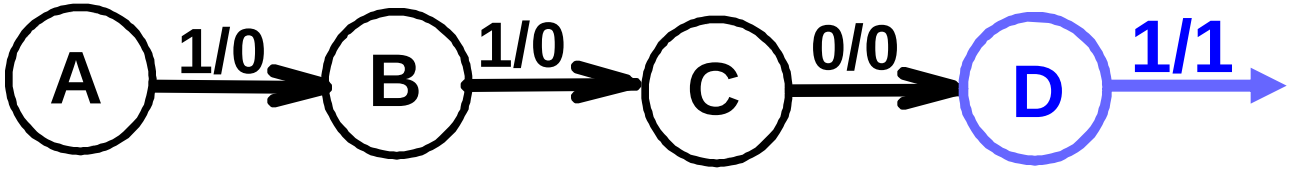


- **Finally, after 110 and a 1, we have:**

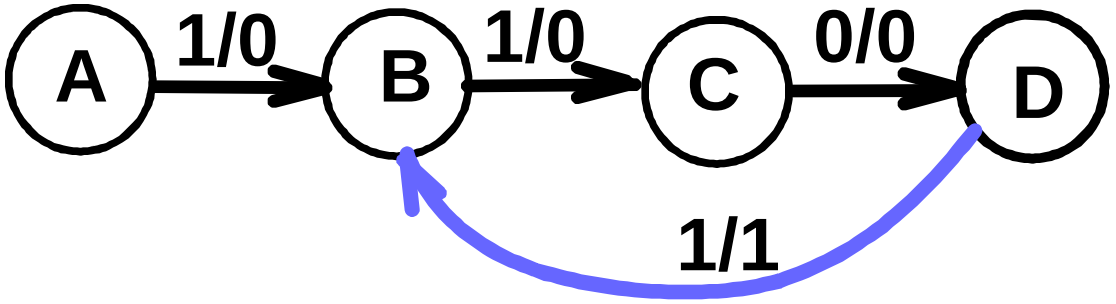


  - **Transition arcs are used to denote the output function (Mealy Model)**
  - **Output 1 on the arc from D means the sequence has been recognized**
  - **To what state should the arc from state D go? Remember: 1101<u>1101</u> ?**
  - **Note that D is the last state but the output 1 occurs for the input applied in D. This is the case when a *Mealy model* is assumed.**

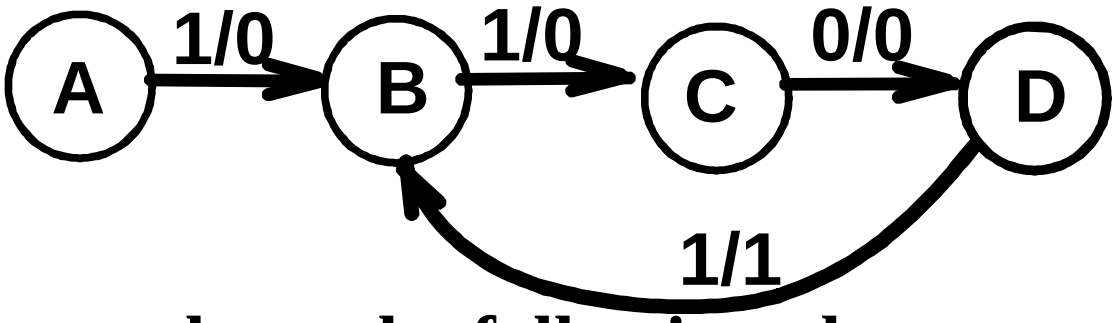
Example 3: Recognize 1101 (continued)



- **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:**



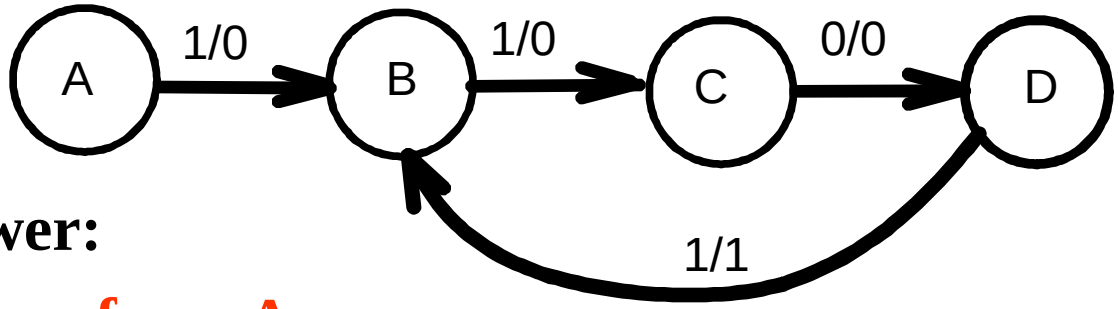
Example 3: Recognize 1101 (continued)



- **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 3: 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

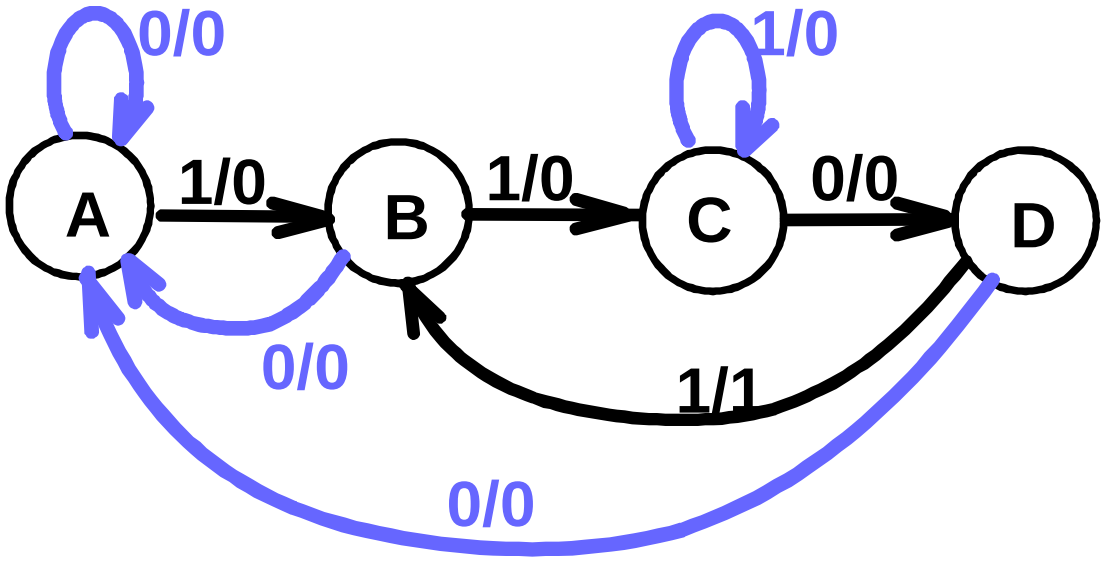
"0" arc from B

"1" arc from C

"0" arc from D.

Example 3: Recognize 1101 (continued)

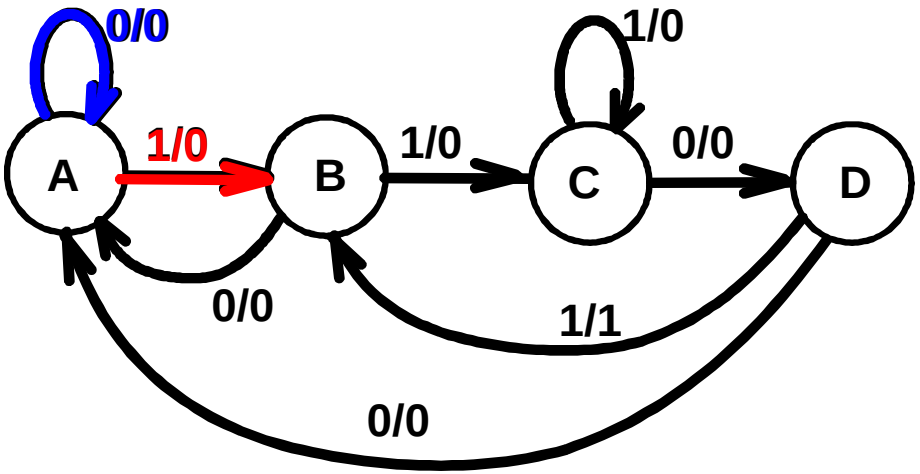
- **State transition arcs must represent the fact that an input subsequence has occurred. Thus we get:**



- **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

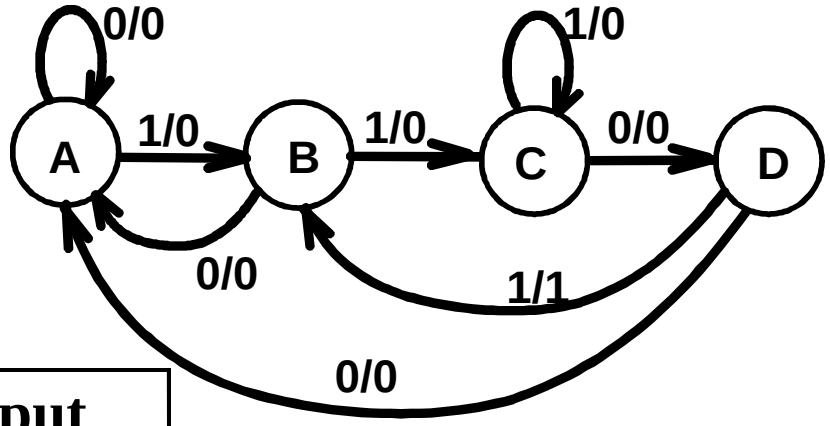
- From the State Diagram, we can fill in the State Table.
- There are 4 states, one input, and one output. We will choose the form with four rows, one for each current state.
- From State A, the 0 and 1 input transitions have been filled in along with the outputs.



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

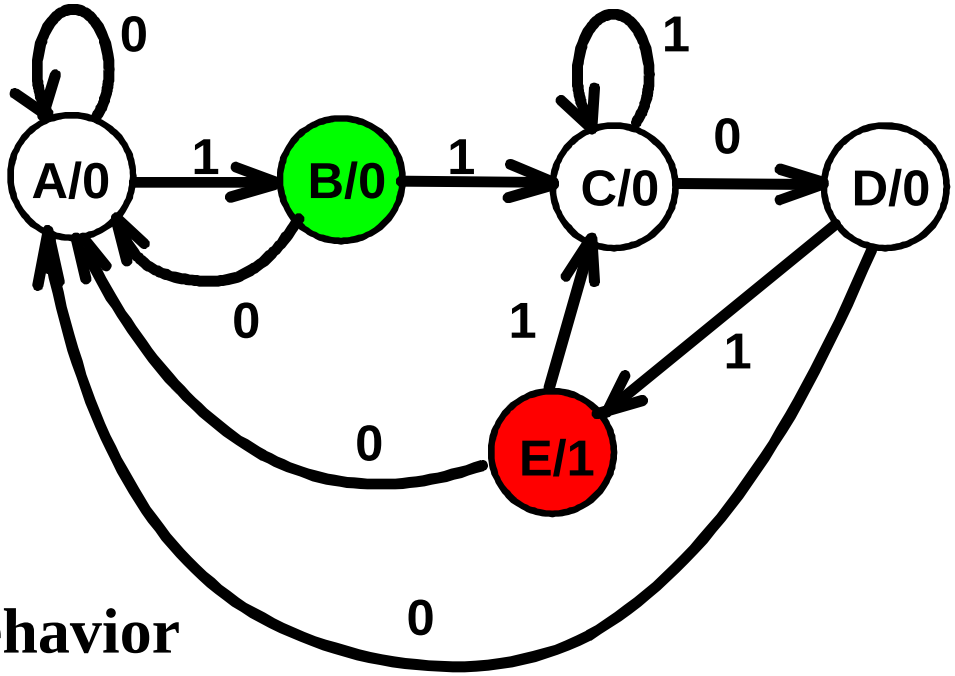
- What would the state diagram and state table look like for the Moore model?

Example 3:
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.

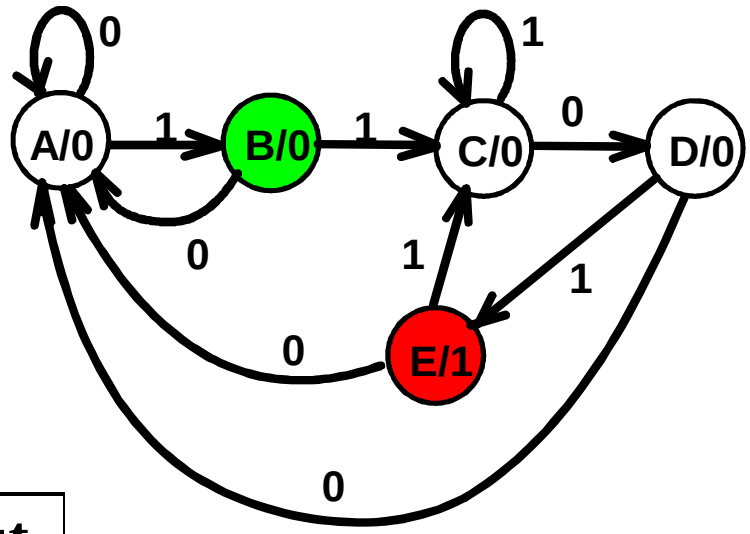
Example 3: Moore Model (continued)

- 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 3: Moore Model (continued)

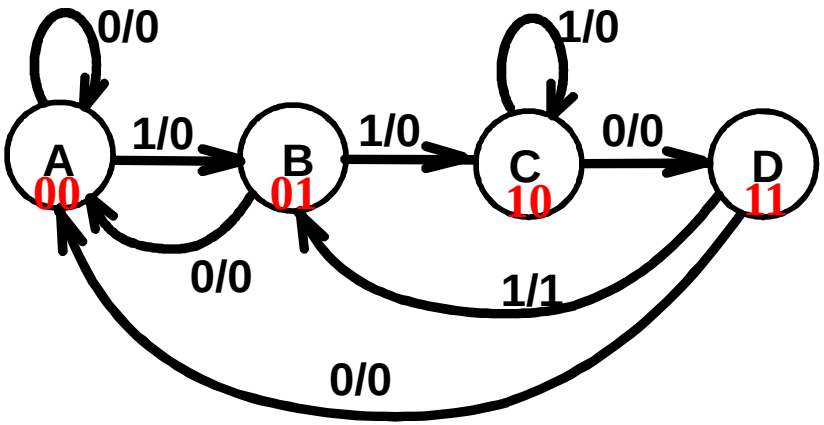
- The state table is shown below
- More state in the Moore model: "Moore is More."



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

State Assignment – Example 3 (continued)

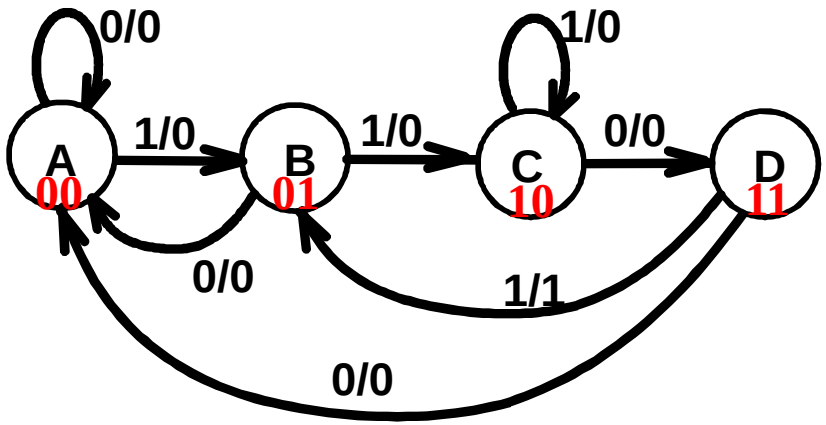
- **Counting Order Assignment: A = 0 0, B = 0 1, C = 1 0, D = 1 1**
- **The resulting coded state table:**



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

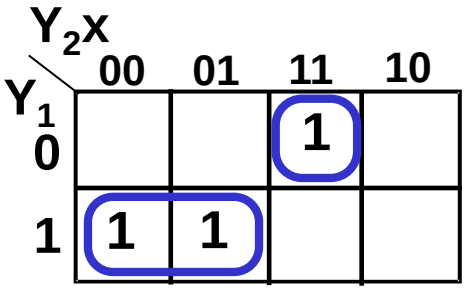
State Assignment – Example 3 (continued)

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

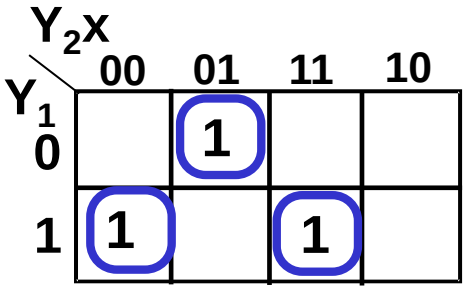


Present State	Next State	Output
$Y_1\ Y_2\ x$	$Y_1\ Y_2$	Z
0 0 0	0 0	0
0 0 1	0 1	0
0 1 0	0 0	0
0 1 1	1 0	0
1 0 0	1 1	0
1 0 1	1 0	0
1 1 0	0 0	0
1 1 1	0 1	1

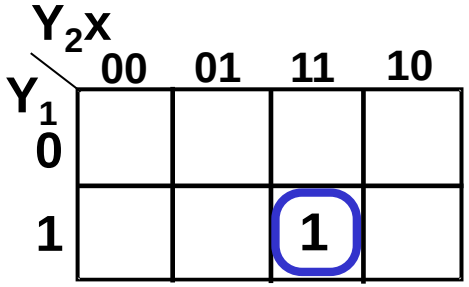
Optimization:
Example 3 - Counting Order Assignment



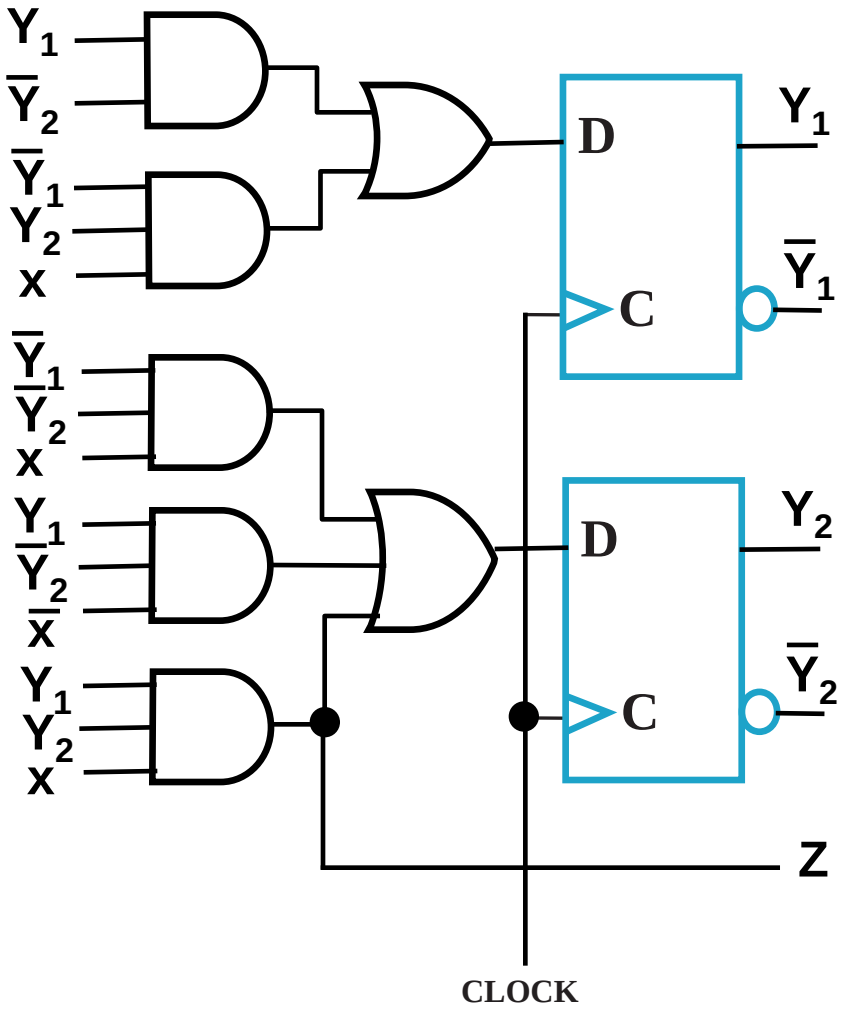
$D_{Y1} = Y_1Y_2' + Y_1'Y_2x$



$D_{Y1} = Y_1'Y_2'x + Y_1Y_2'x' + Y_1Y_2x$

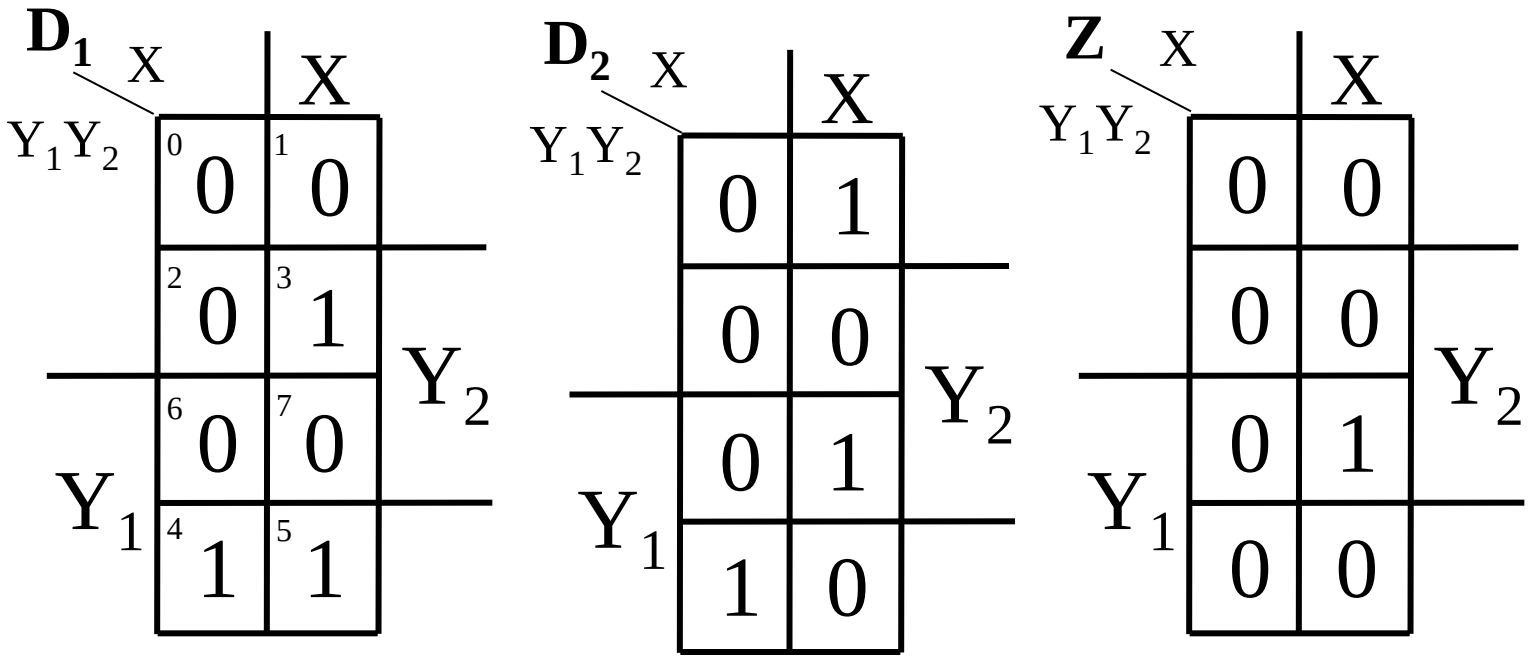


$Z = Y_1Y_2x$



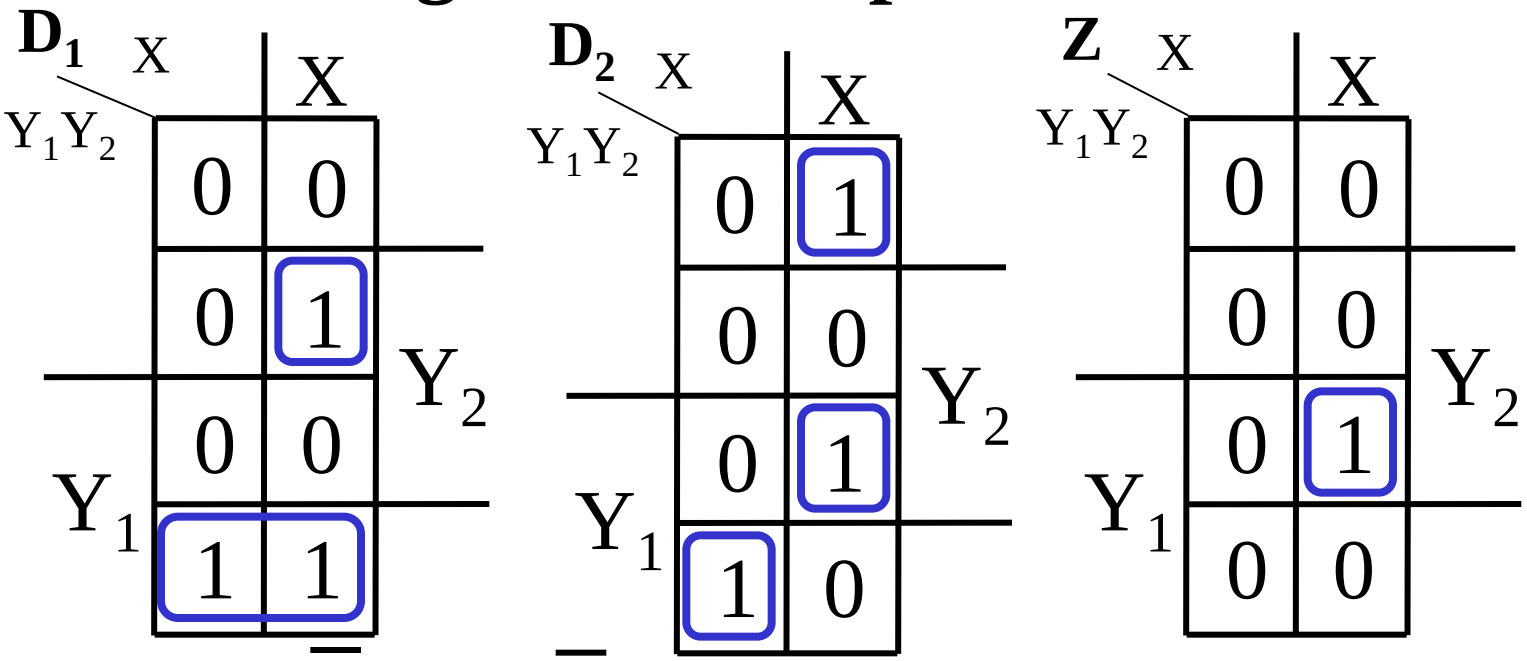
Find Flip-Flop Input and Output Equations: Example 3 – Counting Order Assignment

- 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 3 - Counting Order Assignment

- **Performing two-level optimization:**



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

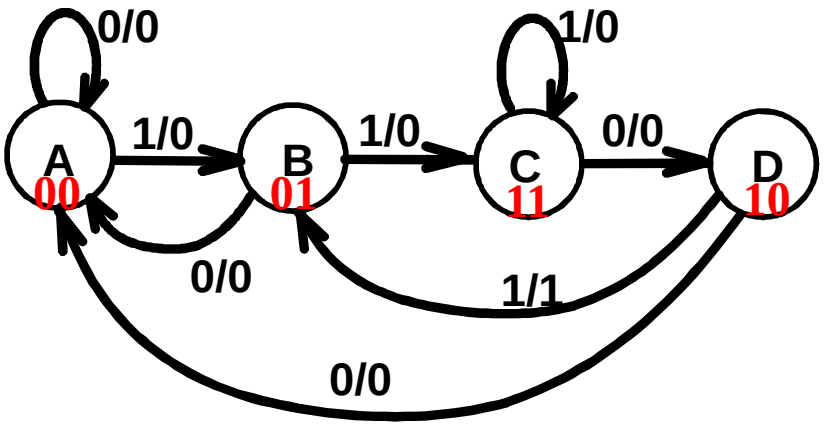
$$D_2 = \overline{X}Y_1\overline{Y}_2 + X\overline{Y}_1\overline{Y}_2 + XY_1Y_2$$

$$Z = XY_1Y_2$$

Gate Input Cost = 19

State Assignment – Example 3 (continued)

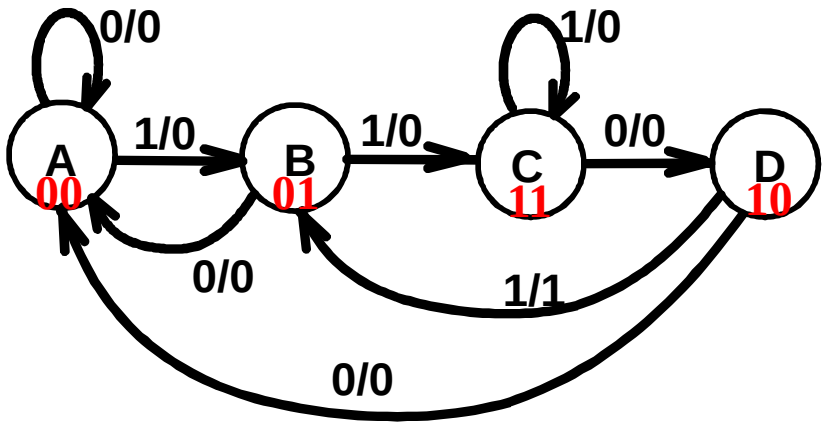
- **Gray Code Assignment: A = 0 0, B = 0 1, C = 1 1, D = 1 0**
- **The resulting coded state table:**



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

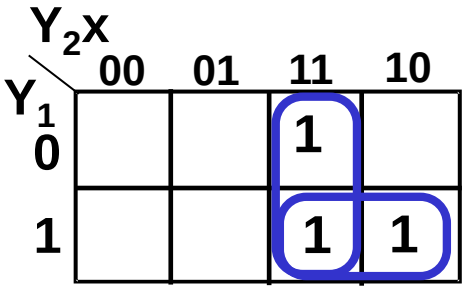
State Assignment – Example 3 (continued)

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

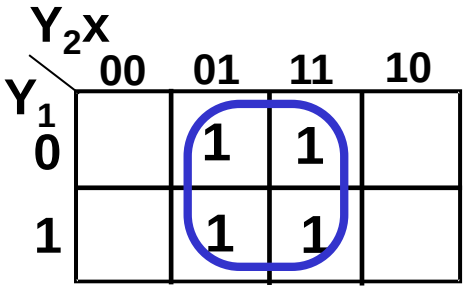


Present State	Next State	Output
$Y_1\ Y_2\ x$	$Y_1\ Y_2$	Z
0 0 0	0 0	0
0 0 1	0 1	0
0 1 0	0 0	0
0 1 1	1 1	0
1 0 0	0 0	0
1 0 1	0 1	1
1 1 0	1 0	0
1 1 1	1 1	0

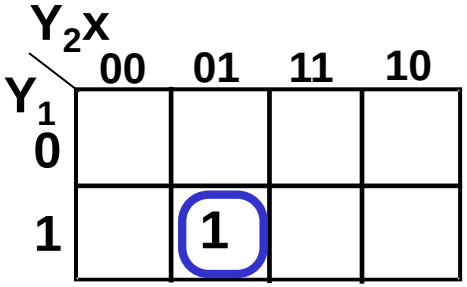
Optimization:
Example 3 - Gray Code Assignment



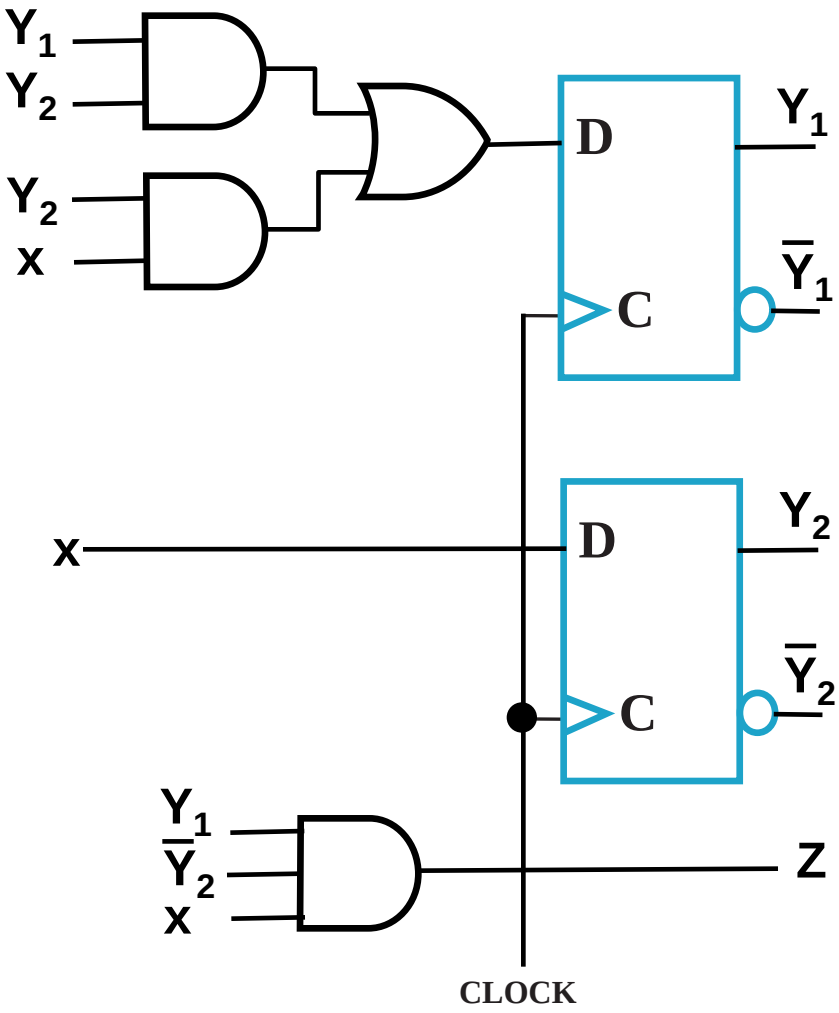
$D_{Y1} = Y_1Y_2 + Y_2x$



$D_{Y1} = x$

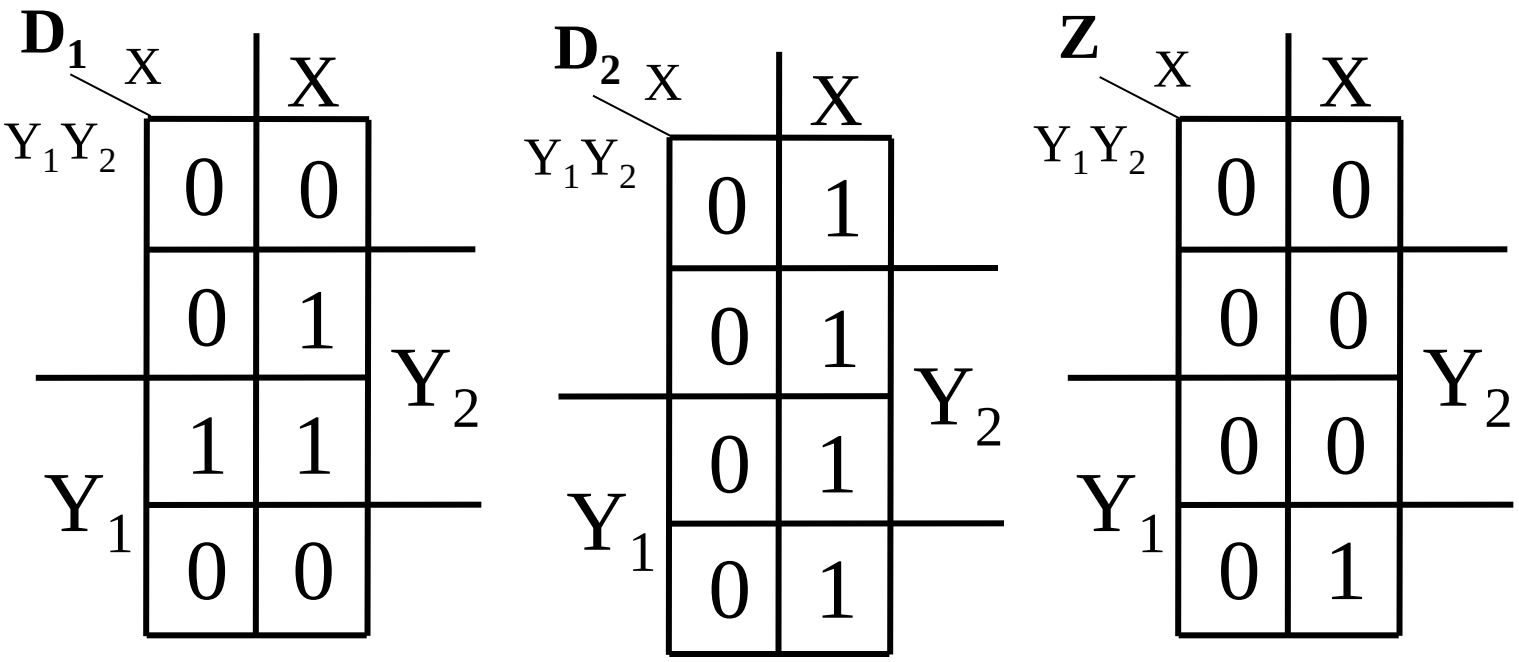


$Z = Y_1Y_2'x$



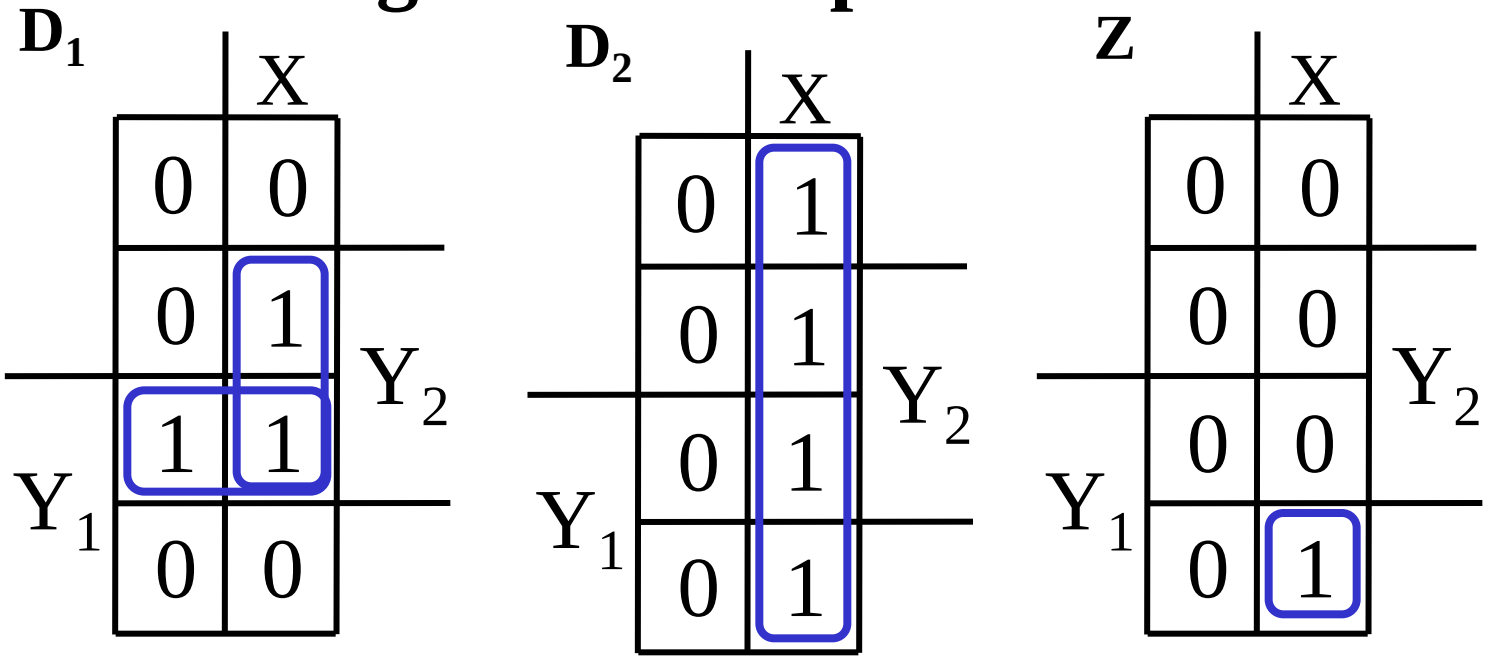
Find Flip-Flop Input and Output Equations: Example 3 – Gray Code Assignment

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



Optimization:
Example 3 – Gray Code Assignment

- **Performing two-level optimization:**



$D_1 = Y_1Y_2 + XY_2$

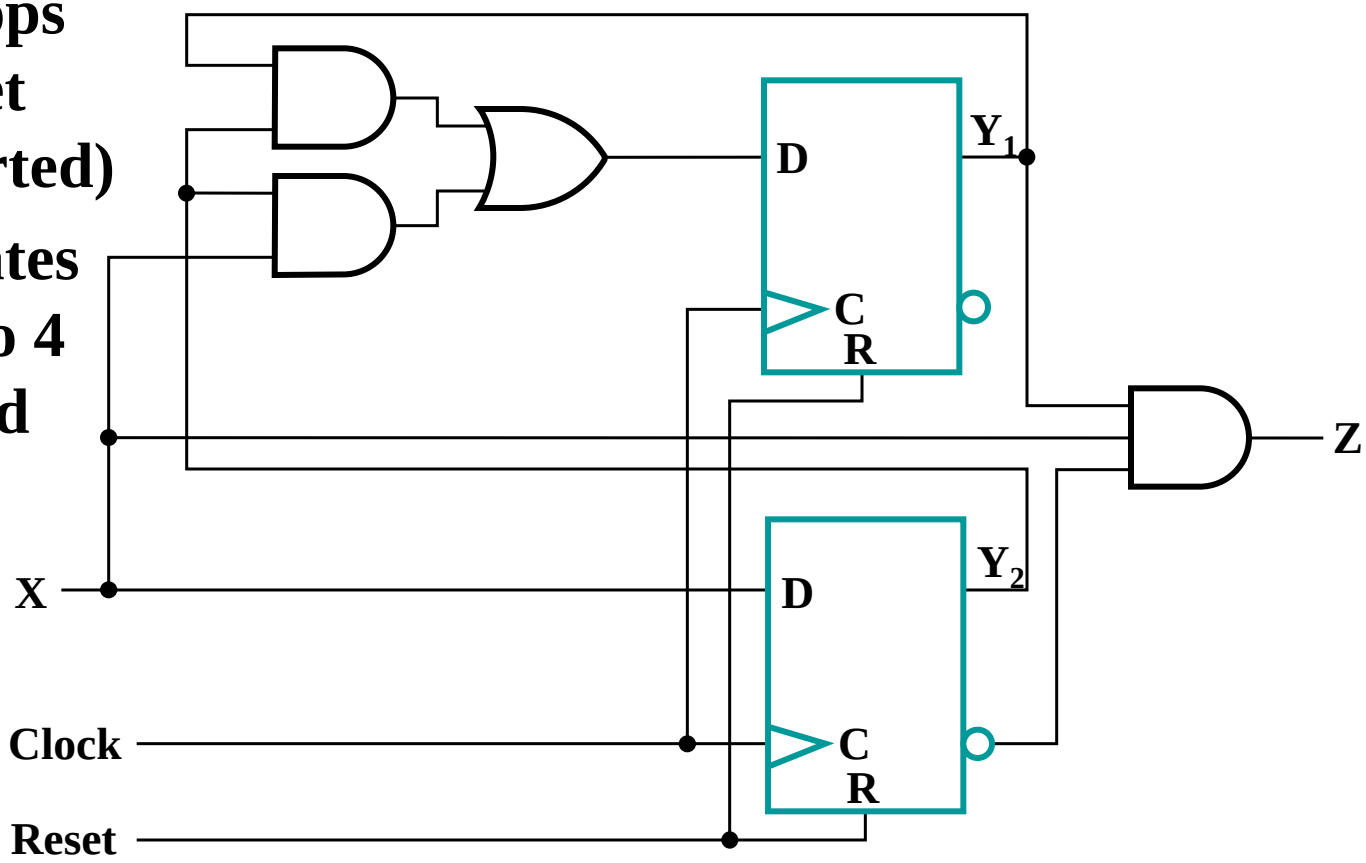
$D_2 = X$

$Z = XY_1\overline{Y}_2$

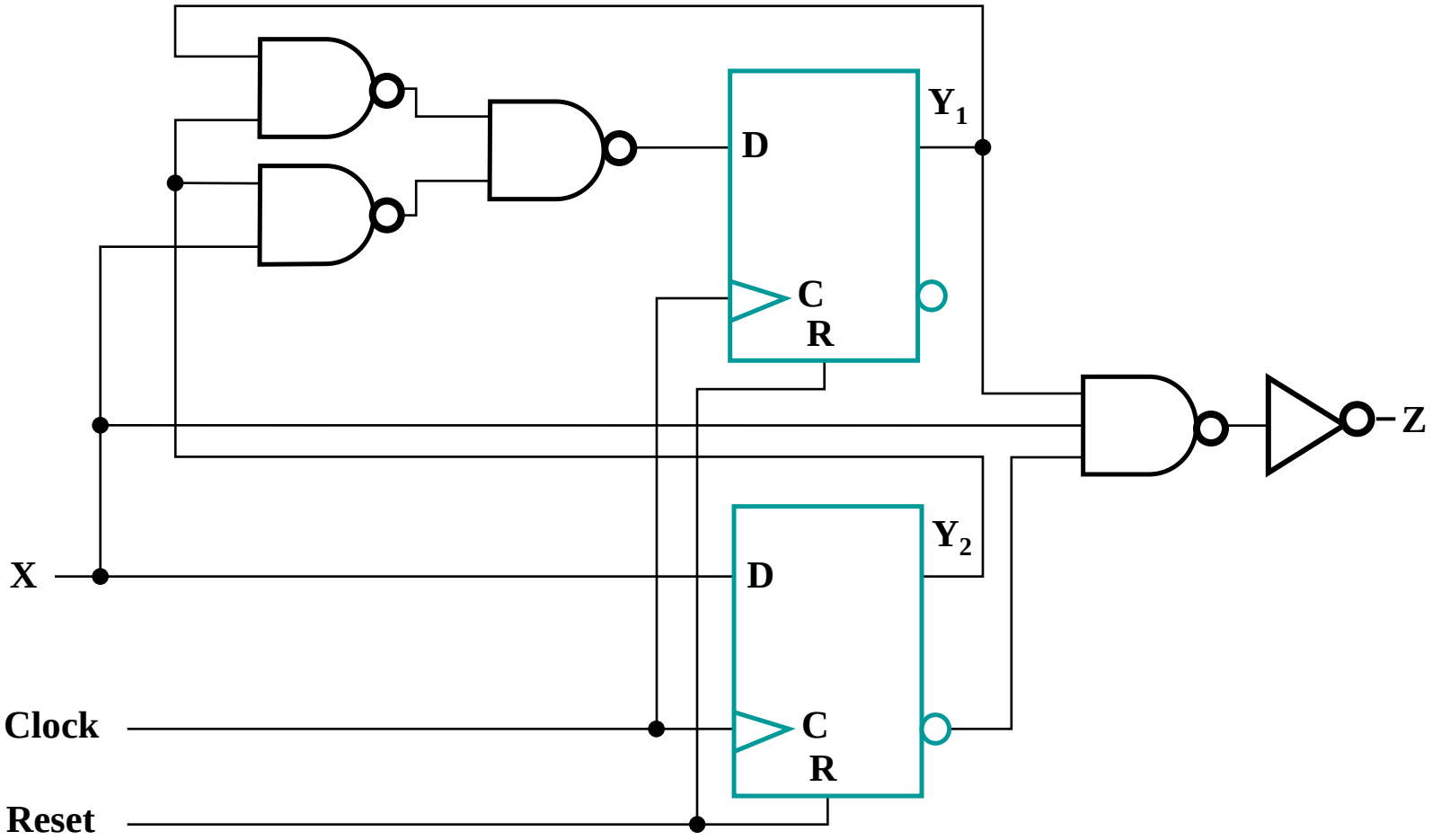
Gate Input Cost = 9

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



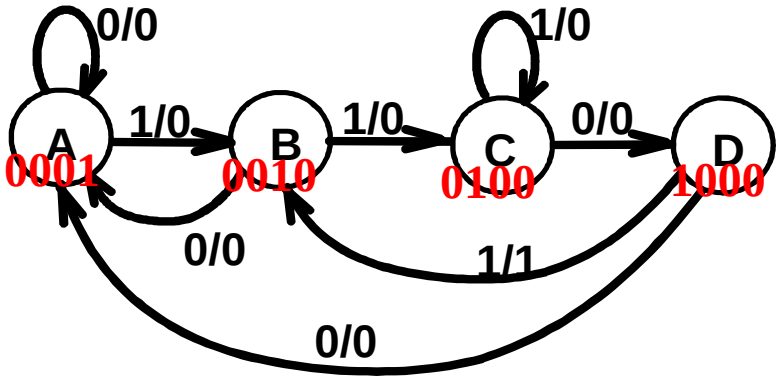
One Flip-flop per State (One-Hot) Assignment

- **Example codes for four states: (Y_3, Y_2, Y_1, Y_0) = 0001, 0010, 0100, and 1000.**
- **In equations, need to include only the variable that is 1 for the state, e. g., state with code 0001, is represented in equations by Y_0 instead of $\overline{Y_3}\,\overline{Y_2}\,\overline{Y_1}\,Y_0$ because all codes with 0 or two or more 1s have don't care next state values.**
- **Provides simplified analysis and design**
- **Combinational logic may be simpler, but flip-flop cost higher – may or may not be lower cost**

State Assignment – Example 3 - One Hot Assignment

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

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
$Y_3\ Y_2\ Y_1\ Y_0$	$Y_3\ Y_2\ Y_1\ Y_0$	$Y_3\ Y_2\ Y_1\ Y_0$	Z	Z
0 0 0 1	0 0 0 1	0 0 1 0	0	0
0 0 1 0	0 0 0 1	0 1 0 0	0	0
0 1 0 0	1 0 0 0	0 1 0 0	0	0
1 0 0 0	0 0 0 1	0 0 1 0	0	1



Optimization:
Example 3: One Hot Assignment

- Equations read from 1 next state variable entries in table:

$D_0 = \overline{X}(Y_0 + Y_1 + Y_3)$ or $\overline{X}\,\overline{Y}_2$ 2

$D_1 = X(Y_0 + Y_3)$ 4

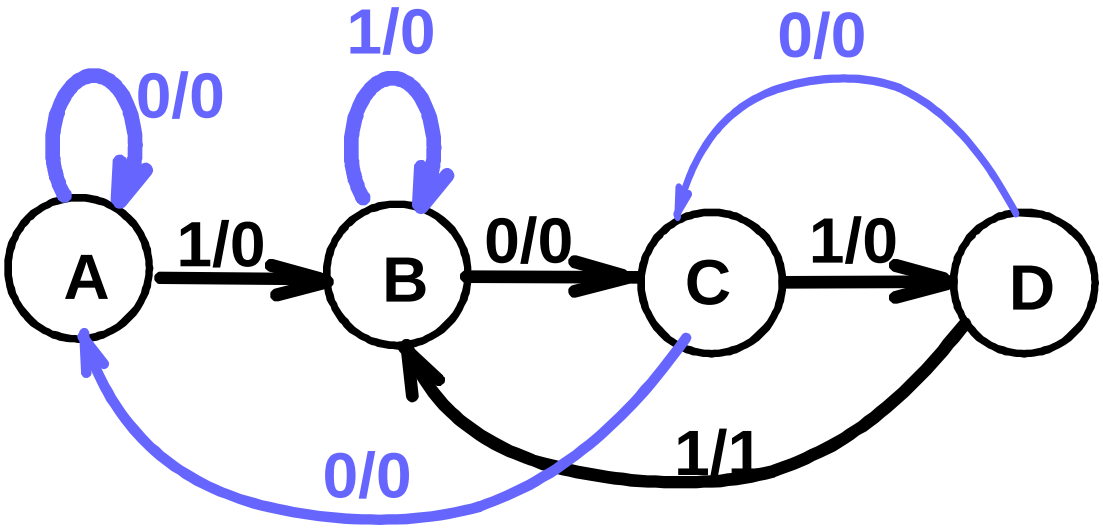
$D_2 = X(Y_1 + Y_2)$ or $X(\overline{Y_0 + Y_3})$ 4

$D_3 = \overline{X}\,Y_2$ 2

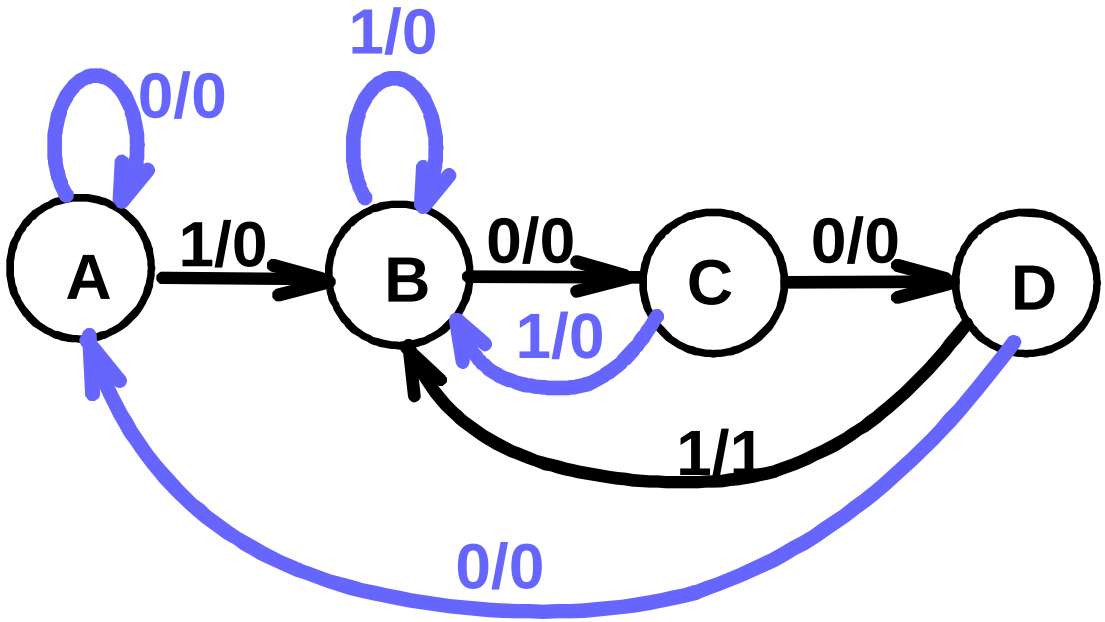
$Z = XY_3$ 2

- Gate Input Cost = 14
- Combinational cost intermediate plus cost of four more flip-flops needed.

Example 4:1011 Sequence Recognizer



Example 5: 1001 Sequence Recognizer

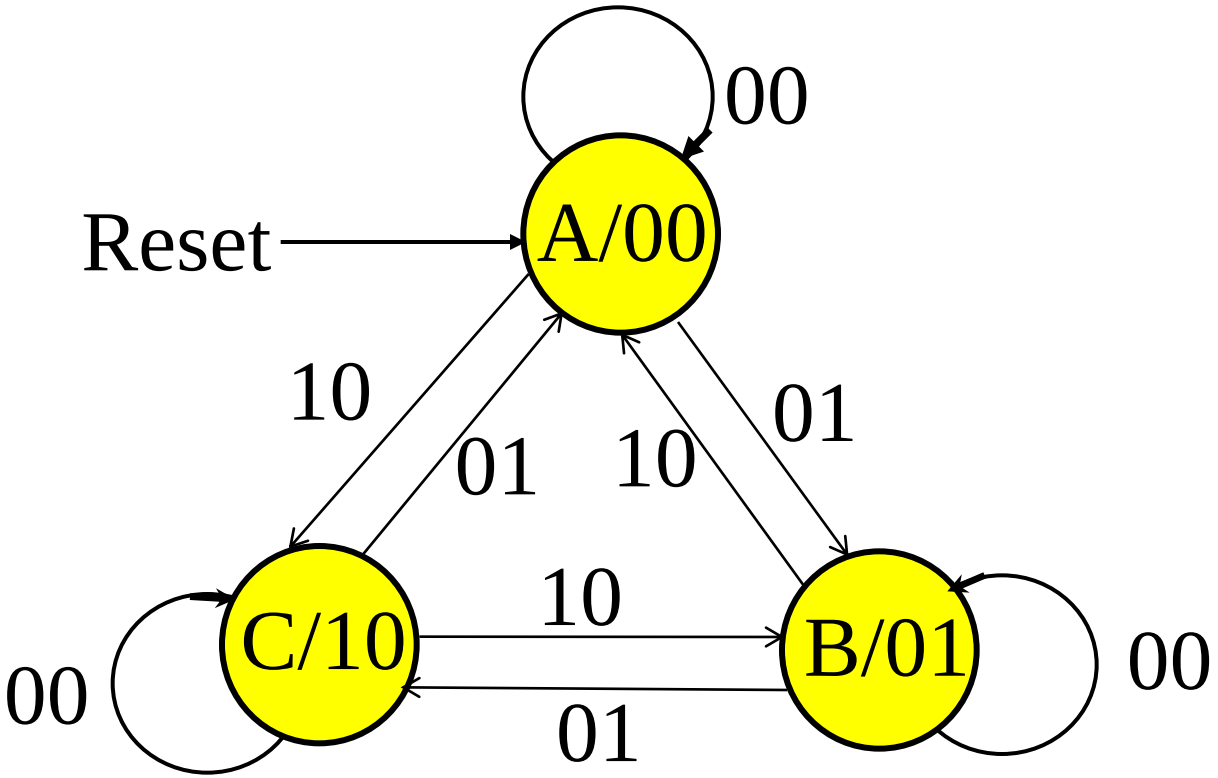


Example 6:Sequential Design

- **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 6 (continued)

- **The state diagram:**



Example 6 (continued)

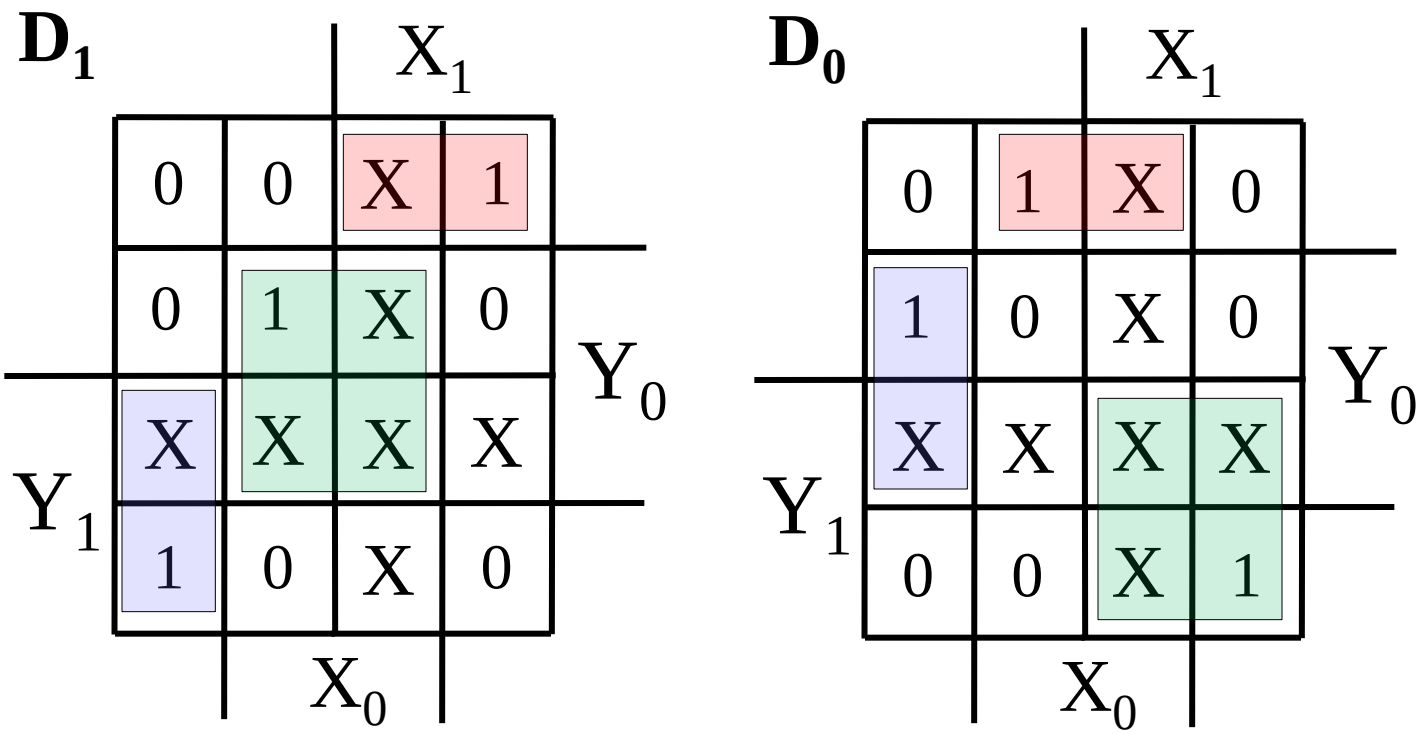
- **The state table**

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	01	X	10	00
B (01)	01	10	X	00	01
- (11)	X	X	X	X	11
C (10)	10	00	X	01	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**

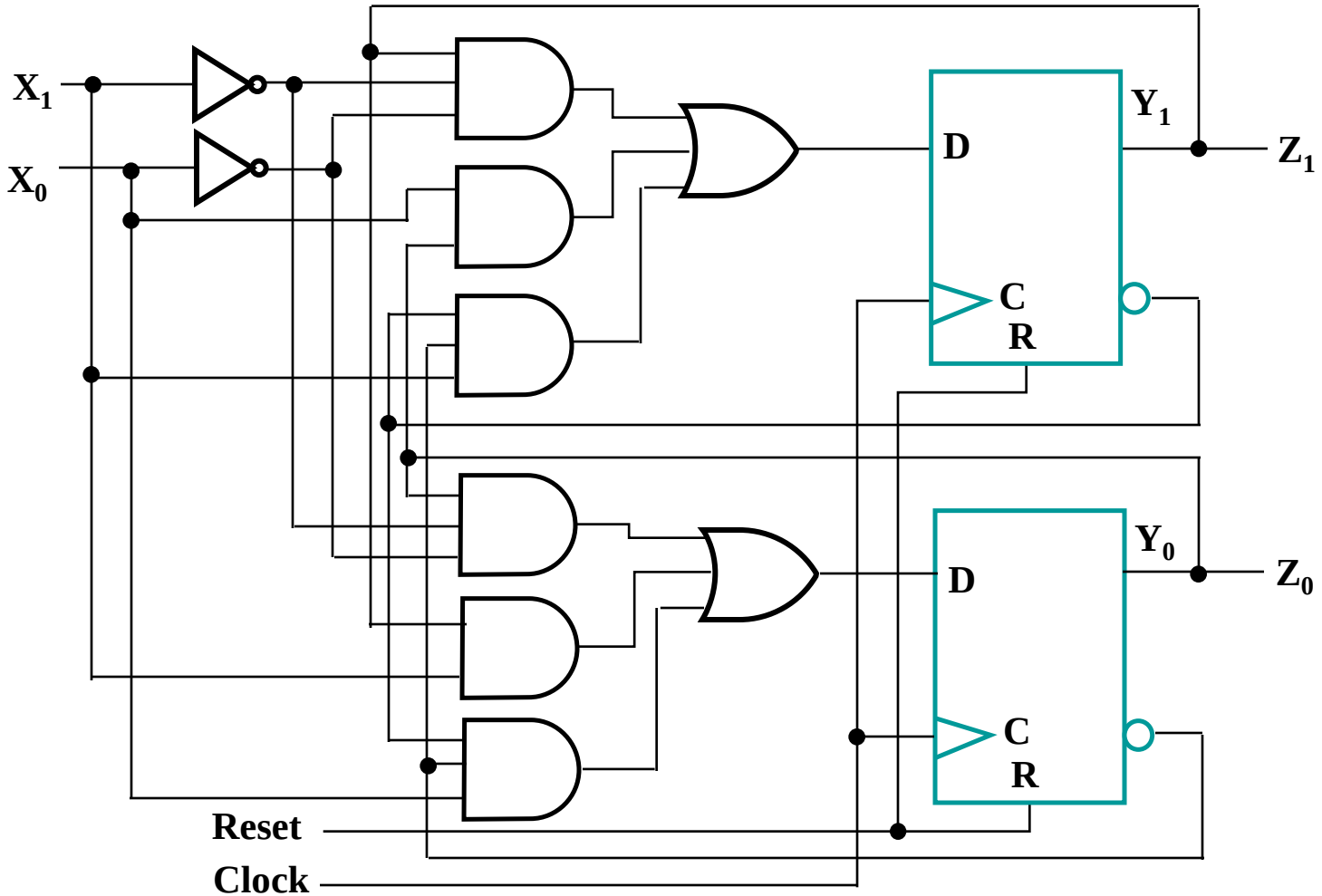
Example 6 (continued)

- **Optimized flip-flop input equations for D flip-flops**



- $D_1 = Y_0X_0 + Y_1X_1'X_0' + Y_1'Y_0'X_1$
- $D_0 = Y_1X_1 + Y_0X_1'X_0' + Y_1'Y_0'X_0$

Circuit - Final Result with AND, OR, NOT



Terms of Use

- **All (or portions) of this material © 2008 by Pearson Education, Inc.**
- **Permission is given to incorporate this material or adaptations thereof into classroom presentations and handouts to instructors in courses adopting the latest edition of Logic and Computer Design Fundamentals as the course textbook.**
- **These materials or adaptations thereof are not to be sold or otherwise offered for consideration.**
- **This Terms of Use slide or page is to be included within the original materials or any adaptations thereof.**