

Digital Logic

Chapter 11

Dr. Ronald H.Y. Chung

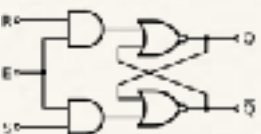


THE UNIVERSITY OF HONG KONG

DEPARTMENT OF
COMPUTER SCIENCE

Boolean Algebra

- ❖ Mathematical discipline used to design and analyze the behavior of the digital circuitry in digital computers and other digital systems
- ❖ Named after George Boole
 - ❖ English mathematician
 - ❖ Proposed basic principles of the algebra in 1854
- ❖ Claude Shannon suggested Boolean algebra could be used to solve problems in relay-switching circuit design
- ❖ Is a convenient tool:
 - ❖ Analysis
 - ❖ It is an economical way of describing the function of digital circuitry
 - ❖ Design
 - ❖ Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function



Boolean Variables and Operations

- ❖ Makes use of variables and operations
 - ❖ Are logical
 - ❖ A variable may take on the value 1 (TRUE) or 0 (FALSE)
 - ❖ Basic logical operations are OR, AND, and NOT
- ❖ OR
 - ❖ Yields true if either or both of its operands are true
- ❖ AND
 - ❖ Yields true (binary value 1) if and only if both of its operands are true
 - ❖ In the absence of parentheses the AND operation takes precedence over the OR operation
 - ❖ When no ambiguity will occur the AND operation is represented by simple concatenation instead of the dot operator
- ❖ NOT
 - ❖ Inverts the value of its operand



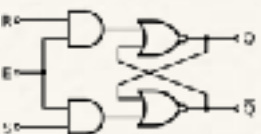
Boolean Operators

❖ Boolean Operators of Two Input Variables

P	Q	NOT P (\bar{P})	P AND Q ($P \cdot Q$)	P OR Q ($P + Q$)	P NAND Q ($\overline{P \cdot Q}$)	P NOR Q ($\overline{P + Q}$)	P XOR Q ($P \oplus Q$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

❖ Boolean Operators Extended to More than Two Inputs (A, B, ...)

Operation	Expression	Output = 1 if
AND	$A \cdot B \cdot \dots$	All of the set $\{A, B, \dots\}$ are 1.
OR	$A + B + \dots$	Any of the set $\{A, B, \dots\}$ are 1.
NAND	$\overline{A \cdot B \cdot \dots}$	Any of the set $\{A, B, \dots\}$ are 0.
NOR	$\overline{A + B + \dots}$	All of the set $\{A, B, \dots\}$ are 0.
XOR	$A \oplus B \oplus \dots$	The set $\{A, B, \dots\}$ contains an odd number of ones.



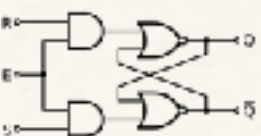
Basic Identities of Boolean Algebra

Basic Postulates



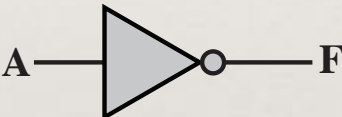


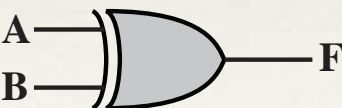
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \bar{A} = 0$	$A + \bar{A} = 1$	Inverse Elements

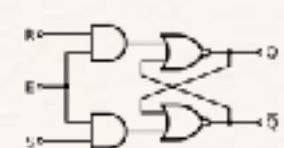
Other Identities

$0 \cdot A = 0$	$1 + A = 1$	Null Law
$A \cdot A = A$	$A + A = A$	Idempotent Law
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A} \cdot \bar{B}$	DeMorgan's Theorem



Basic Logic Gates

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																



Some Uses of NAND and NOR Gates

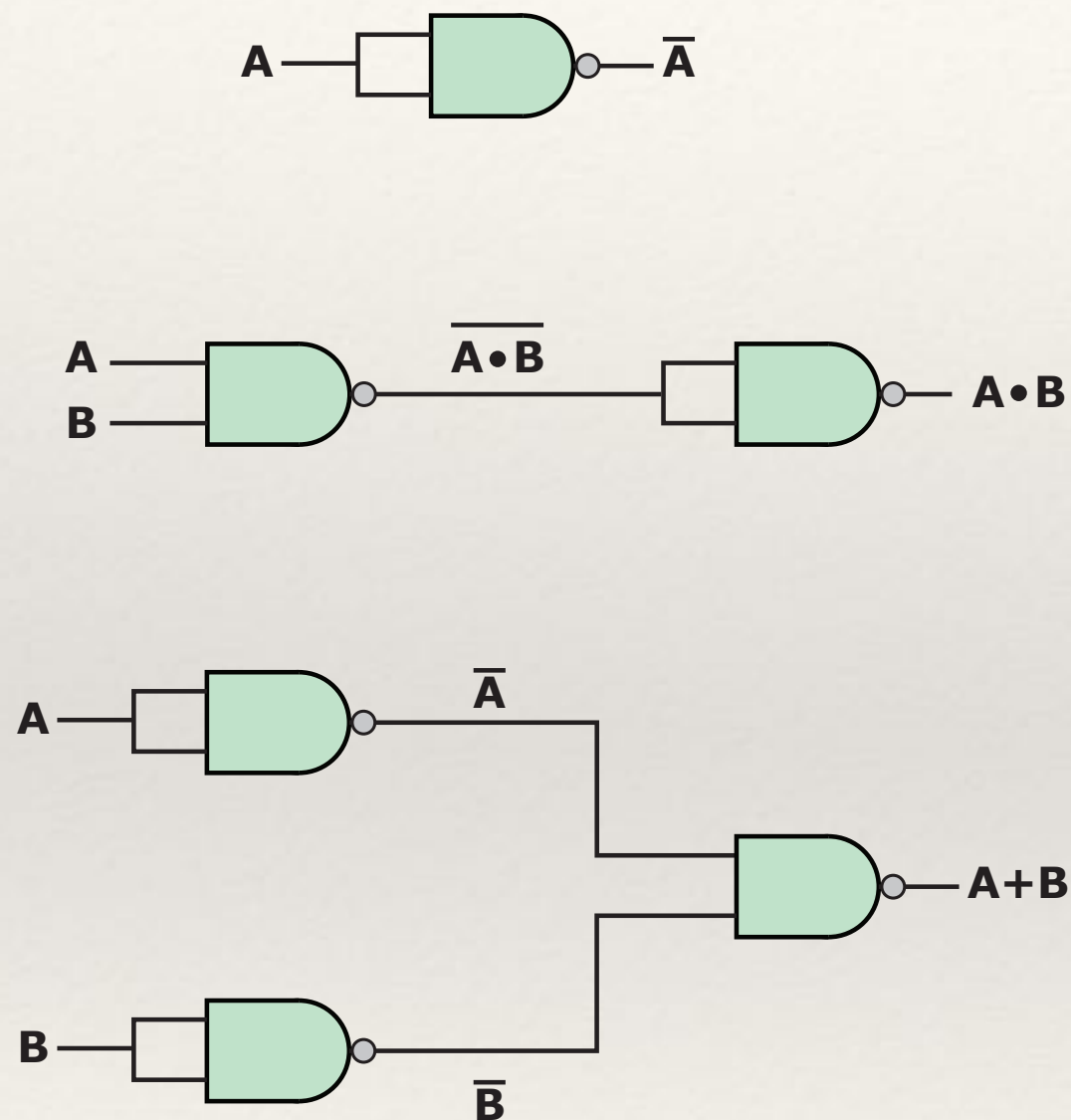


Figure 11.2 Some Uses of NAND Gates

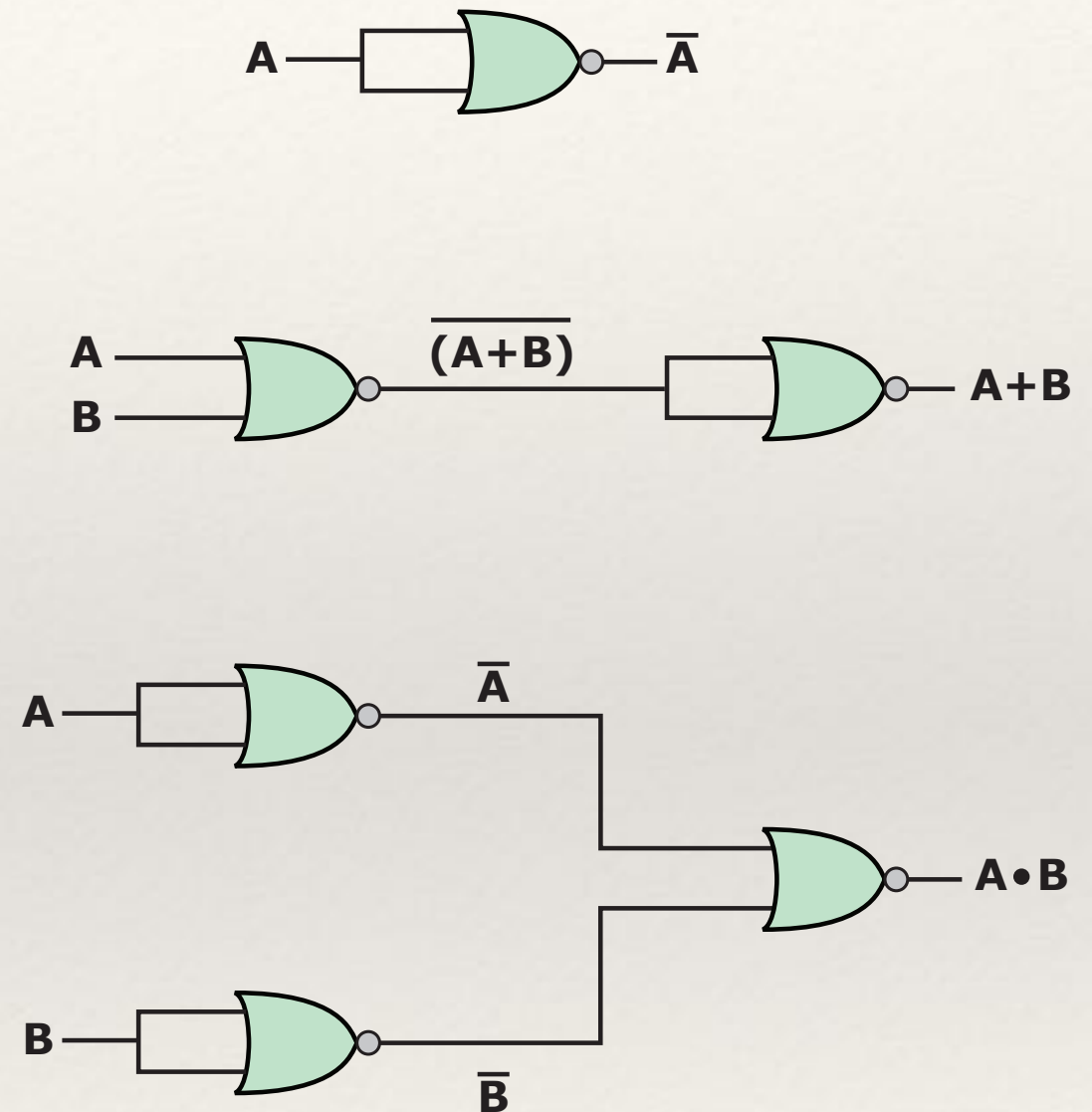
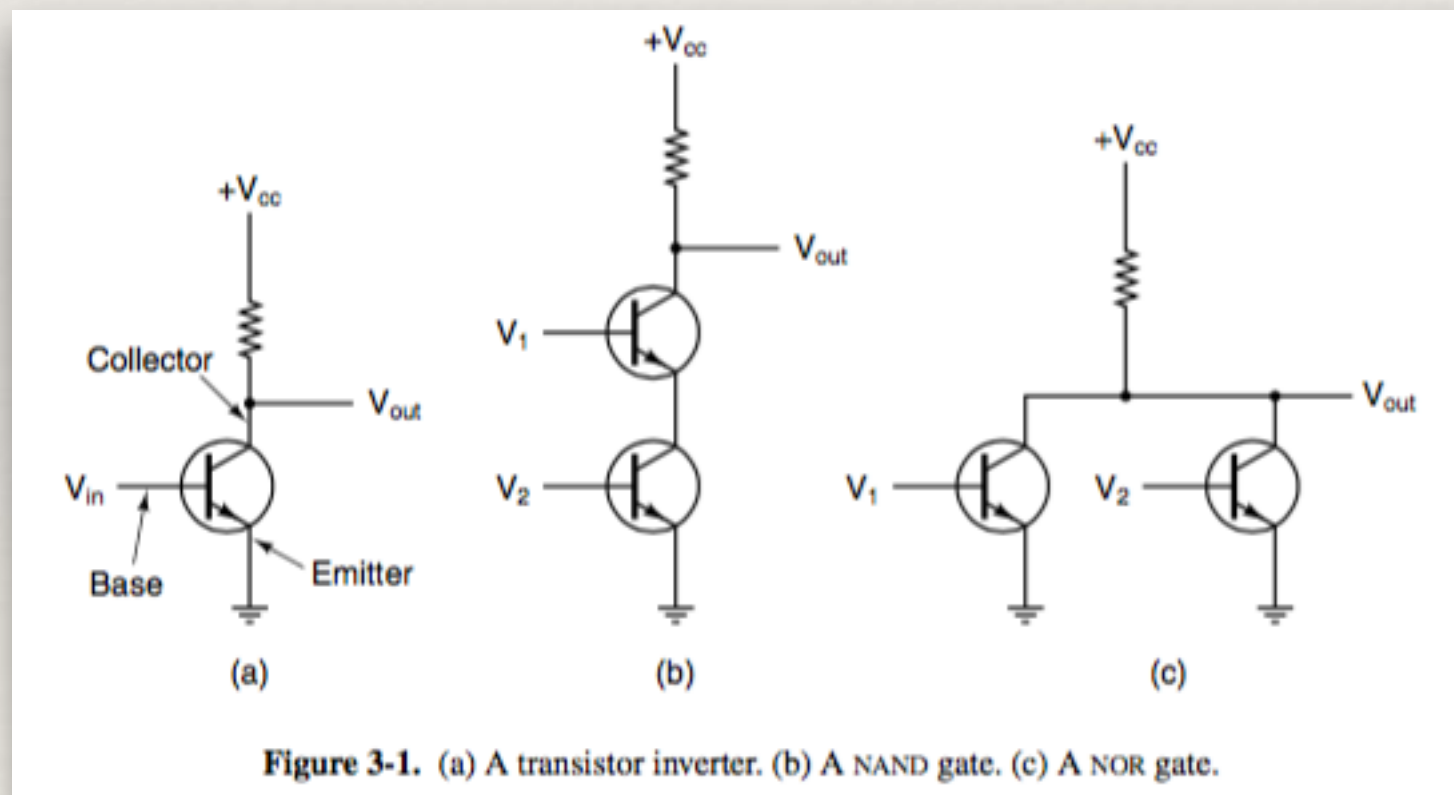


Figure 11.3 Some Uses of NOR Gates

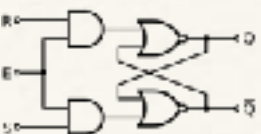
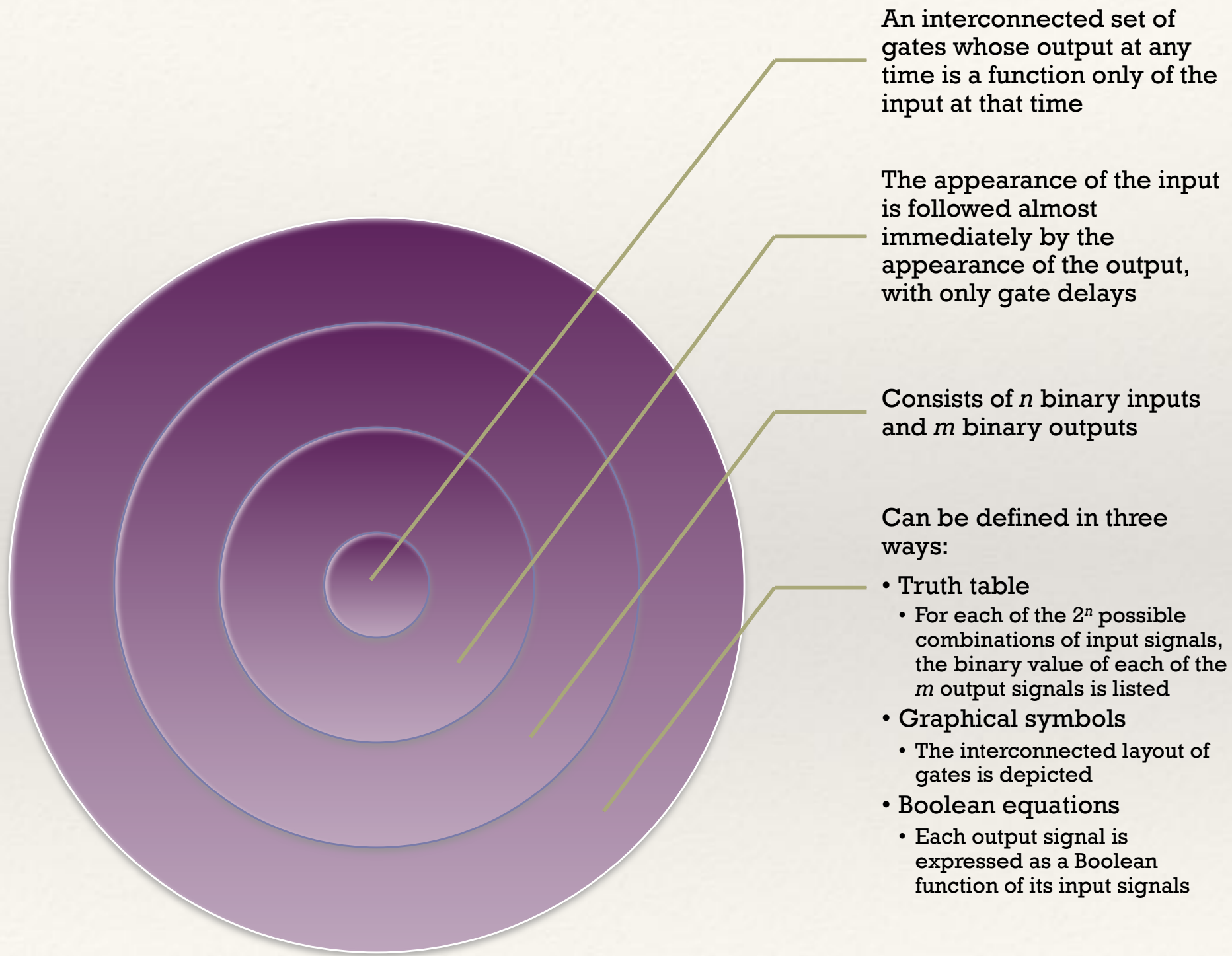
Logic Gates (Optional)

- ❖ Logic Gates are implemented using transistors
- ❖ Transistor can be treated as switches, which turns on/off according to the value of V_{in} . If V_{in} is 1, the transistor is *ON*, otherwise it is *OFF*
- ❖ V_{out} will be connected to the ground, when the transistor in (a), or both transistors in (b), or either transistor in (c) is/are turned on
- ❖ If V_{out} is connected to ground (which is 0), then V_{out} is 0. Otherwise V_{out} is 1



Source: Structured Computer Organization, Sixth Edition

Combinational Circuit

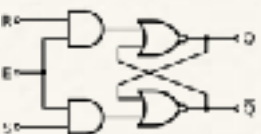


Implementation of Boolean Functions

- ❖ Any Boolean function can be implemented in electronic form as a network of gates
- ❖ Consider the following Boolean function

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

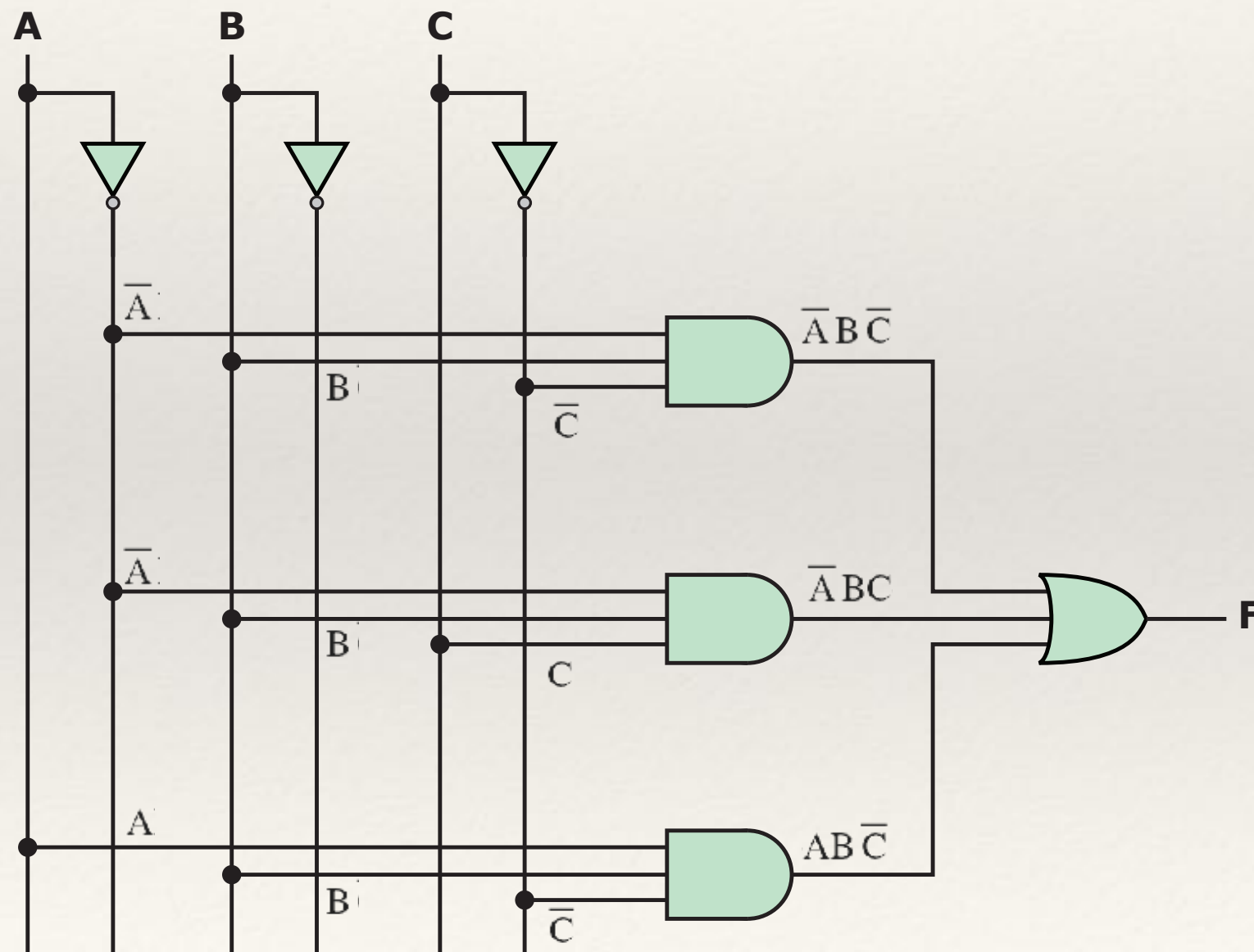
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Implementation of Boolean Functions (Cont'd)

- ❖ Sum of Products (SOP) implementation of

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$



Implementation of Boolean Functions (Cont'd)

- ❖ We can also say that the output is 1 if none of the input combinations that proceed 0 is true

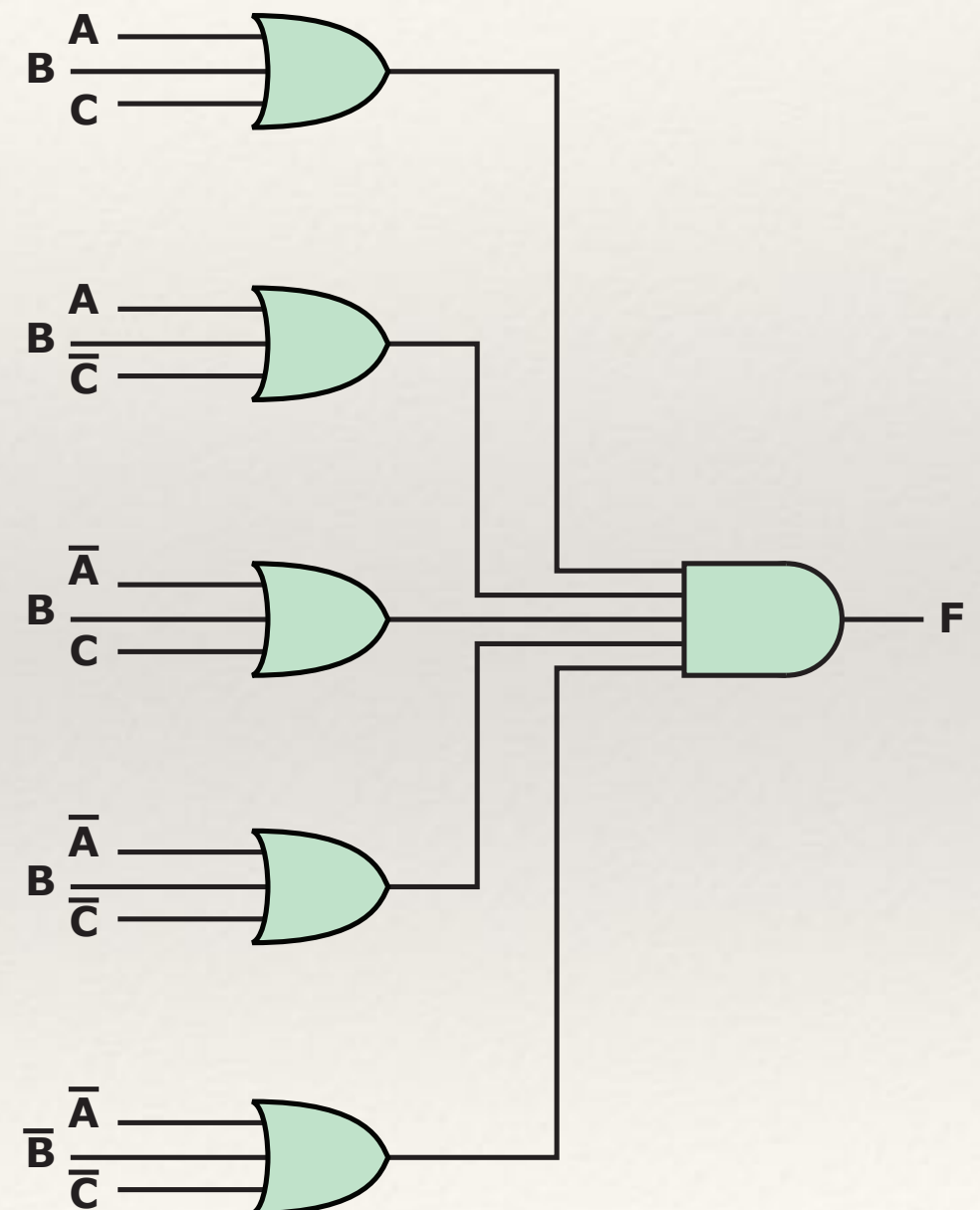
$$F = (\overline{A} \overline{B} \overline{C}) \cdot (\overline{A} \overline{B} C) \cdot (\overline{A} B \overline{C}) \cdot (\overline{A} B C) \cdot (ABC)$$

by DeMorgan's Theorem: $(\overline{A} \overline{B} \overline{C}) = A+B+C$

$$F = (A+B+C) \cdot (A+B+\overline{C}) \cdot (\overline{A}+B+C) \cdot (\overline{A}+B+\overline{C}) \cdot (\overline{A}+\overline{B}+\overline{C})$$

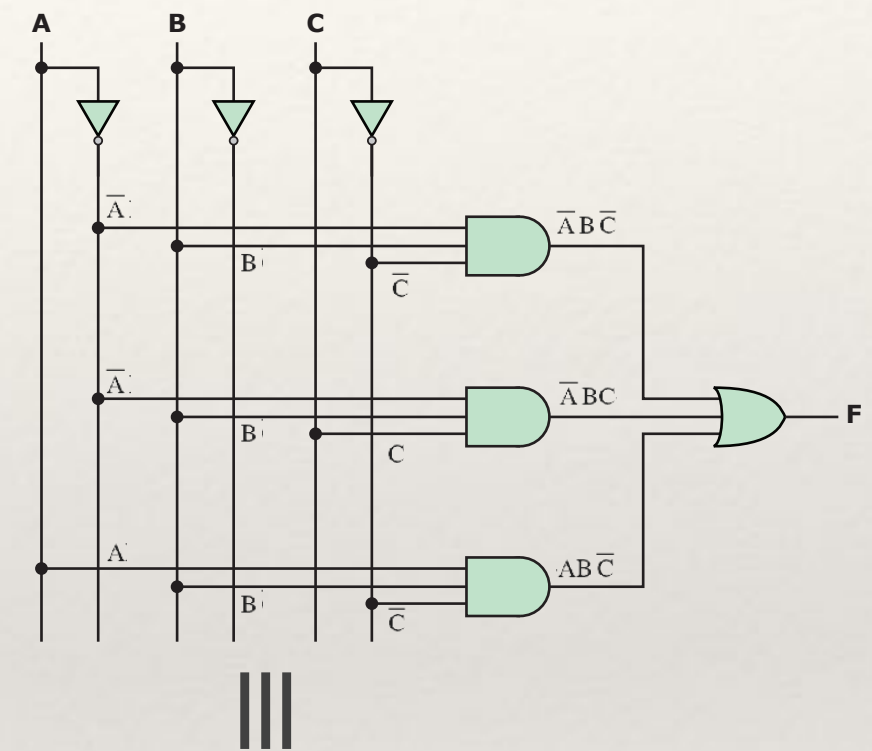
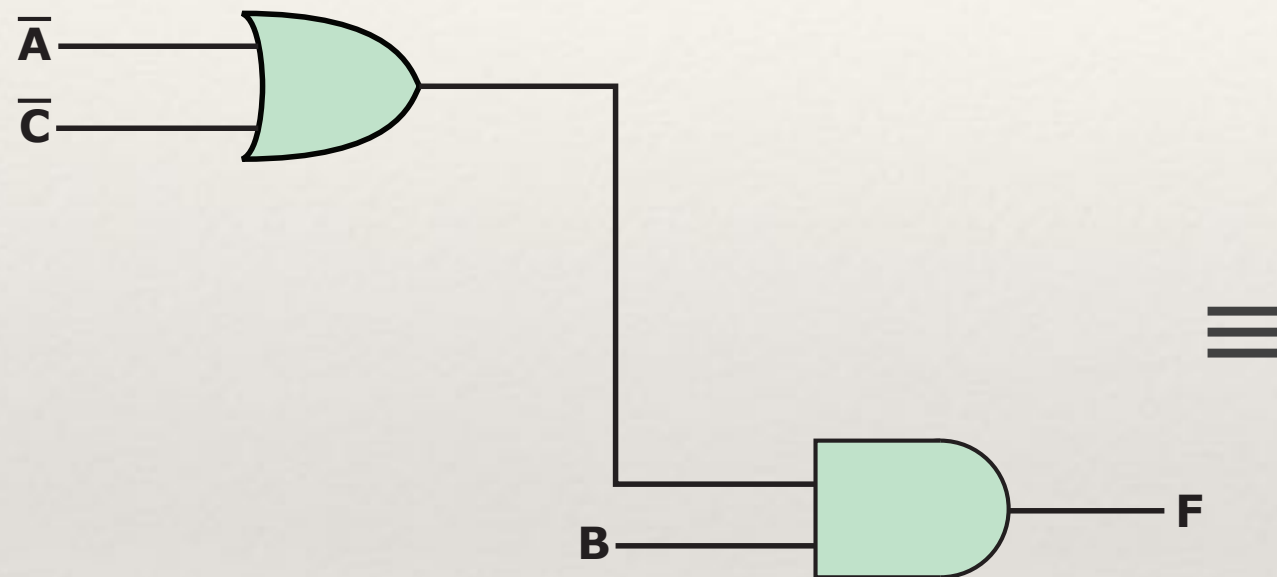
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Product of Sums (POS) Implementation

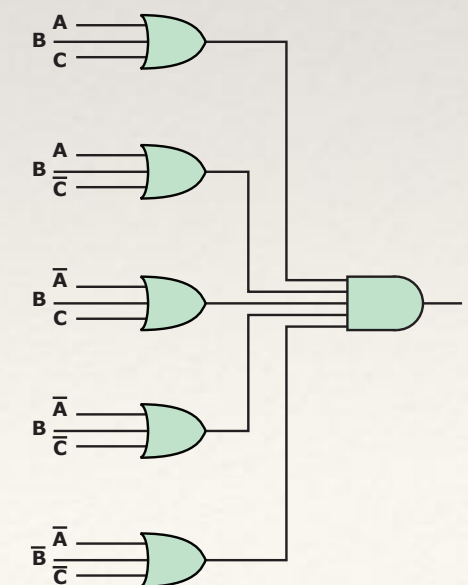


Simplified Implementation

- ❖ The following implementation (which is much more simpler) is equivalent to the previous two implementations
 - ❖ But how to do this simplification?



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Algebraic Simplification

❖ Recall the laws and identities

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

$$1 \cdot A = A$$

$$0 + A = A$$

$$A \cdot \bar{A} = 0$$

$$A + \bar{A} = 1$$

$$0 \cdot A = 0$$

$$1 + A = 1$$

$$A \cdot A = A$$

$$A + A = A$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$$A + (B + C) = (A + B) + C$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

❖ Consider the Boolean Function again

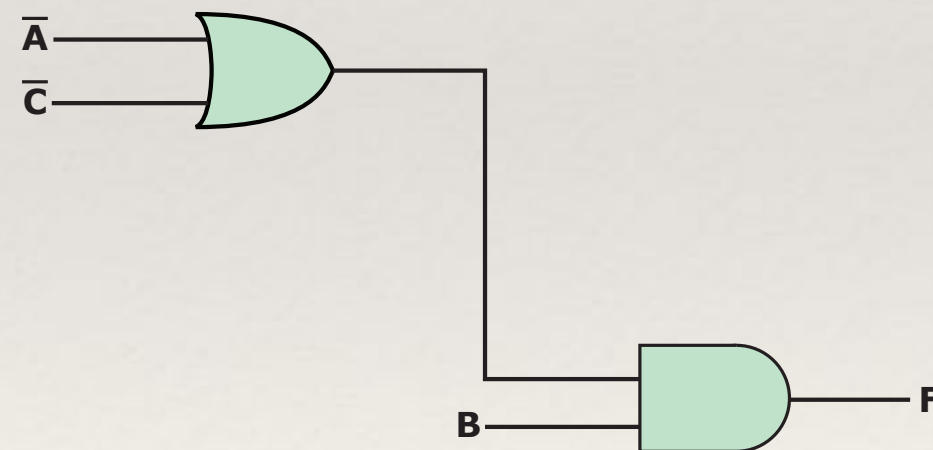
$$F = \bar{A} B \bar{C} + \bar{A} B C + A B \bar{C}$$

$$= \bar{A} B \bar{C} + \bar{A} B C + A B \bar{C} + \bar{A} B \bar{C}$$

$$= B \cdot (\bar{A} \bar{C} + \bar{A} C + A \bar{C} + \bar{A} \bar{C})$$

$$= B \cdot (\bar{A} \cdot (\bar{C} + C) + \bar{C} \cdot (\bar{A} + A))$$

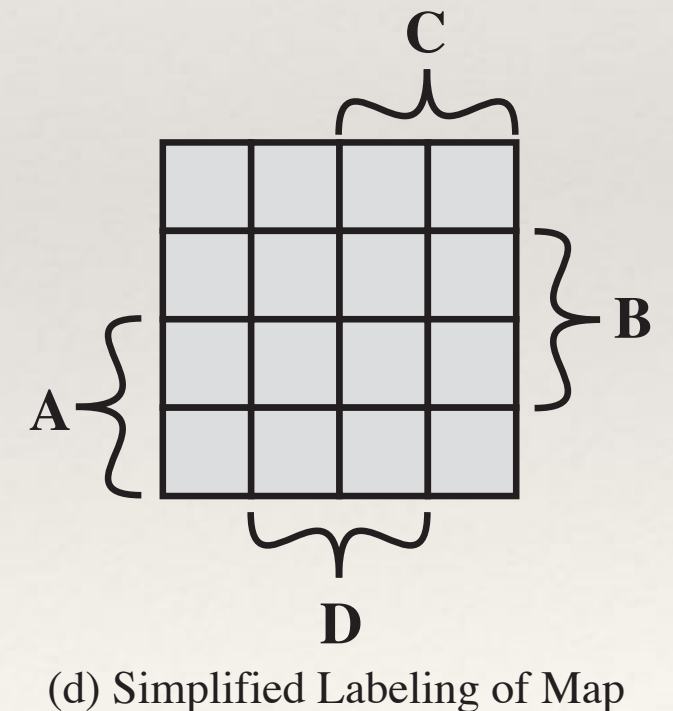
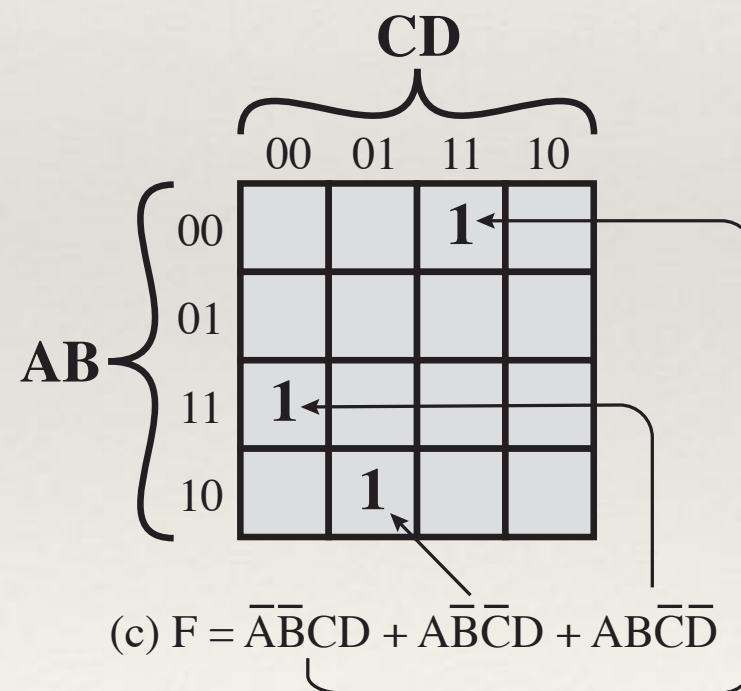
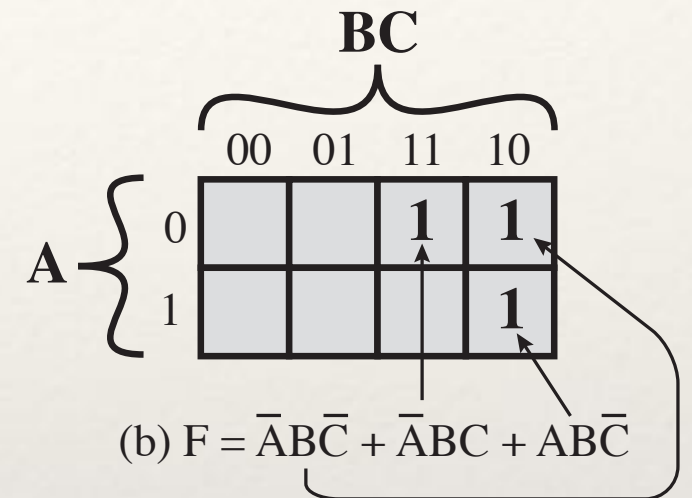
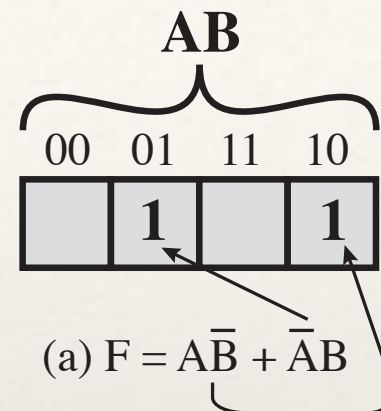
$$= B \cdot (\bar{A} + \bar{C})$$



❖ For more complex expressions, more systematic approach is needed

Karnaugh Map (K-Map)

- ❖ Karnaugh map is a convenient way of representing a Boolean function of a small number (up to four) of variables.
- ❖ Each square corresponds to a unique product in the sum-of-products form, with a 1 value corresponding to the variable and a 0 value corresponding to the NOT of that variable



Use of K-Map

- ❖ Any two squares that are adjacent differ in only one of the variables
- ❖ If two adjacent squares both have an entry of one
 - ❖ Then the corresponding product terms differ in only one variable
 - ❖ The two terms can be merged by eliminating that variable
- ❖ The concept of adjacency can be extended to include wrapping around the edge of the map
- ❖ We can group not just 2 squares but 2^n adjacent squares (i.e., 2, 4, 8, etc.)

		CD			
		00	01	11	10
AB	00				
	01		1	1	
	11				
	10				

(a)

		CD			
		00	01	11	10
AB	00		1		
	01				
	11				
	10		1		

(b)

		CD			
		00	01	11	10
AB	00				
	01	1			1
	11				
	10				

(c)

		CD			
		00	01	11	10
AB	00	1	1	1	1
	01				
	11				
	10				

(d)

		CD			
		00	01	11	10
AB	00				
	01	1	1		
	11	1	1		
	10				

(e)

		CD			
		00	01	11	10
AB	00				
	01	1			1
	11	1			1
	10				

(f)

		CD			
		00	01	11	10
AB	00	1	1	1	1
	01	1	1	1	1
	11				
	10				

(g)

		CD			
		00	01	11	10
AB	00	1			1
	01	1			1
	11	1			1
	10	1			1

(h)

		CD			
		00	01	11	10
AB	00			1	1
	01			1	1
	11			1	1
	10			1	1

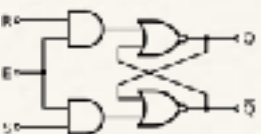
(i)



Use of K-Map (Cont'd)

❖ The rules for simplification:

1. Among the marked squares (squares with a 1), find those that belong to a unique largest block of 1, 2, 4, or 8 and circle those blocks.
2. Select additional blocks of marked squares that are as large as possible and as few in number as possible, but include every marked square at least once.
 - The results may not be unique in some cases.
3. Continue to draw loops around single marked squares, or pairs of adjacent marked squares, or groups of four, eight, and so on in such a way that every marked square belongs to at least one loop; then use as few of these blocks as possible to include all marked squares.



Overlapping Groups

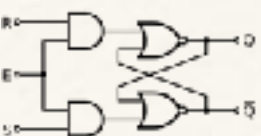
- ❖ If any isolated 1s remain after the groupings, then each of these is circled as a group of 1s
- ❖ Any group of 1s that is completely overlapped by other groups can be eliminated

		BC			
		00	01	11	10
A	0			1	1
	1				1

(a) $F = \bar{A}B + B\bar{C}$

		CD			
		00	01	11	10
AB	00				
	01		1		
	11		1	1	
	10			1	

(b) $F = \bar{B}\bar{C}D + ACD + ABD$



Another Example: Decimal Incrementer

❖ Decimal Incrementer

❖ 0->1, 1->2, 2->3, ... 8->9, 9->0

Input					Output				
Number	A	B	C	D	Number	W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Don't care con- dition	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d
	1	1	1	1		d	d	d	d

❖ Certain combinations of values of variables never occur, and therefore the corresponding output never occurs.

❖ These are referred to as 'Don't Care' state denoted as 'd', and they can be treated as a 1 or 0, whichever leads to simplest expression



Karnaugh Maps for the Incrementer

		CD			
		00	01	11	10
AB	00				
	01			1	
	11	d	d	d	d
	10	1		d	d

(a) $W = A\bar{D} + BCD$

		CD			
		00	01	11	10
AB	00			1	
	01	1	1		1
	11	d	d	d	d
	10			d	d

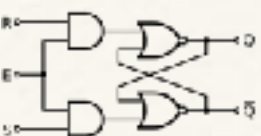
(b) $X = B\bar{D} + B\bar{C} + \bar{B}CD$

		CD			
		00	01	11	10
AB	00		1		1
	01		1		1
	11	d	d	d	d
	10			d	d

(c) $Y = \bar{A}\bar{C}D + \bar{A}C\bar{D}$

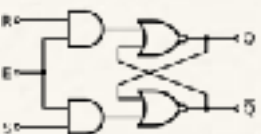
		CD			
		00	01	11	10
AB	00	1			1
	01	1			1
	11	d	d	d	d
	10	1		d	d

(d) $Z = \bar{D}$



Four or more variables?

- ❖ For more than four variables, the Karnaugh map method becomes increasingly cumbersome
- ❖ With five variables, two 4×4 maps are needed
 - ❖ One map stacked on top of another to explore adjacency
- ❖ With six variables, four 4×4 maps are needed in four dimensions
- ❖ That means some other approach is required



Quine-McCluskey Method

❖ Consider a boolean expression

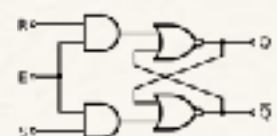
❖ e.g. $F = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD$

❖ Construct a table in which each row corresponds to one of the product terms of the expression. The terms are grouped according to the number of *uncomplemented* variables

❖ Then, find all pairs of terms that differ in only one variable

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	
4	$ABCD$	15	1	1	1	1	



Remark: Alternative expression for F
 $F = m_1 + m_5 + m_6 + m_7 + m_{11} + m_{12} + m_{13} + m_{15}$

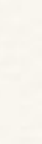
Quine-McCluskey Method (Cont'd)

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	✓
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	✓
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{C}D$	1,5	0	-	0	1	
2	$\bar{A}BD$	5,7	0	1	-	1	
2	$B\bar{C}D$	5,13	-	1	0	1	



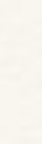
Quine-McCluskey Method (Cont'd)

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	✓
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	
3	$\bar{A}BCD$	7	0	1	1	1	✓
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	✓
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	✓
3	$\bar{A}BCD$	7	0	1	1	1	✓
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	✓

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
2	$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
2	$\bar{A}BC\bar{D}$	6	0	1	1	0	✓
2	$AB\bar{C}\bar{D}$	12	1	1	0	0	✓
3	$\bar{A}BCD$	7	0	1	1	1	✓
3	$A\bar{B}CD$	11	1	0	1	1	
3	$AB\bar{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\bar{A}\bar{C}D$	1,5	0	-	0	1	
2	$\bar{A}BD$	5,7	0	1	-	1	
2	$B\bar{C}D$	5,13	-	1	0	1	
2	$\bar{A}BC$	6,7	0	1	1	-	
2	$AB\bar{C}$	12,13	1	1	0	-	
3	BCD	7,15	-	1	1	1	



Quine-McCluskey Method (Cont'd)

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\overline{A}\overline{B}\overline{C}D$	1	0	0	0	1	✓
2	$\overline{A}B\overline{C}D$	5	0	1	0	1	✓
2	$\overline{A}BC\overline{D}$	6	0	1	1	0	✓
2	$AB\overline{C}\overline{D}$	12	1	1	0	0	✓
3	$\overline{A}BCD$	7	0	1	1	1	✓
3	$A\overline{B}CD$	11	1	0	1	1	✓
3	$AB\overline{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	✓

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\overline{A}\overline{B}\overline{C}D$	1	0	0	0	1	✓
2	$\overline{A}B\overline{C}D$	5	0	1	0	1	✓
2	$\overline{A}BC\overline{D}$	6	0	1	1	0	✓
2	$AB\overline{C}\overline{D}$	12	1	1	0	0	✓
3	$\overline{A}BCD$	7	0	1	1	1	✓
3	$A\overline{B}CD$	11	1	0	1	1	✓
3	$AB\overline{C}D$	13	1	1	0	1	✓
4	$ABCD$	15	1	1	1	1	✓

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\overline{A}\overline{C}D$	1,5	0	-	0	1	
2	$\overline{A}BD$	5,7	0	1	-	1	
2	$B\overline{C}D$	5,13	-	1	0	1	
2	$\overline{A}BC$	6,7	0	1	1	-	
2	$AB\overline{C}$	12,13	1	1	0	-	
3	BCD	7,15	-	1	1	1	
3	ACD	11,15	1	-	1	1	
3	ABD	13,15	1	1	-	1	



Quine-McCluskey Method (Cont'd)

- ❖ The new table is organized into groups in the same fashion as the first table, and it is then processed in the same manner as the first.
- ❖ In general, the process would proceed through successive tables until a table with no matches was produced

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\overline{A} \overline{C} D$	1,5	0	-	0	1	
2	$\overline{A} B D$	5,7	0	1	-	1	✓
2	$B \overline{C} D$	5,13	-	1	0	1	
2	$\overline{A} B C$	6,7	0	1	1	-	
2	$A B \overline{C}$	12,13	1	1	0	-	
3	$B C D$	7,15	-	1	1	1	
3	$A C D$	11,15	1	-	1	1	
3	$A B D$	13,15	1	1	-	1	✓

No. of "1"s	Product Term	Index	A	B	C	D	Done
2	$B D$	5,7,13,15	-	1	-	1	



Quine-McCluskey Method (Cont'd)

- ❖ Once the process just described is completed, we have eliminated many of the possible terms of the expression. Those terms that have not been eliminated are first identified

❖ i.e. $\overline{A}\overline{C}D$, $\overline{A}BC$, $AB\overline{C}$, ACD , BD

No. of "1"s	Product Term	Index	A	B	C	D	Done
1	$\overline{A}\overline{C}D$	1,5	0	-	0	1	
2	$\overline{A}BD$	5,7	0	1	-	1	✓
2	$B\overline{C}D$	5,13	-	1	0	1	✓
2	$\overline{A}BC$	6,7	0	1	1	-	
2	$AB\overline{C}$	12,13	1	1	0	-	
3	BCD	7,15	-	1	1	1	✓
3	ACD	11,15	1	-	1	1	
3	ABD	13,15	1	1	-	1	✓

No. of "1"s	Product Term	Index	A	B	C	D	Done
2	BD	5,7,13,15	-	1	-	1	
2	BD	5,13,7,15	-	1	-	1	



Last Stage of Quine-McCluskey Method

- ❖ Terms that have not been eliminated are used to construct a matrix
 - ❖ An X is placed at each intersection of a row and a column such that the row element is “compatible” with the column element
 - ❖ Next, *circle* each X that is *alone* in a column. Then place a *square* around each X in any row in which there is a circled X
 - ❖ If every column now has either a squared or a circled X, then we are done, and those row elements whose Xs have been marked constitute the minimal expression.
 - ❖ In cases in which some columns have neither a circle nor a square, keep adding row elements until all columns are covered

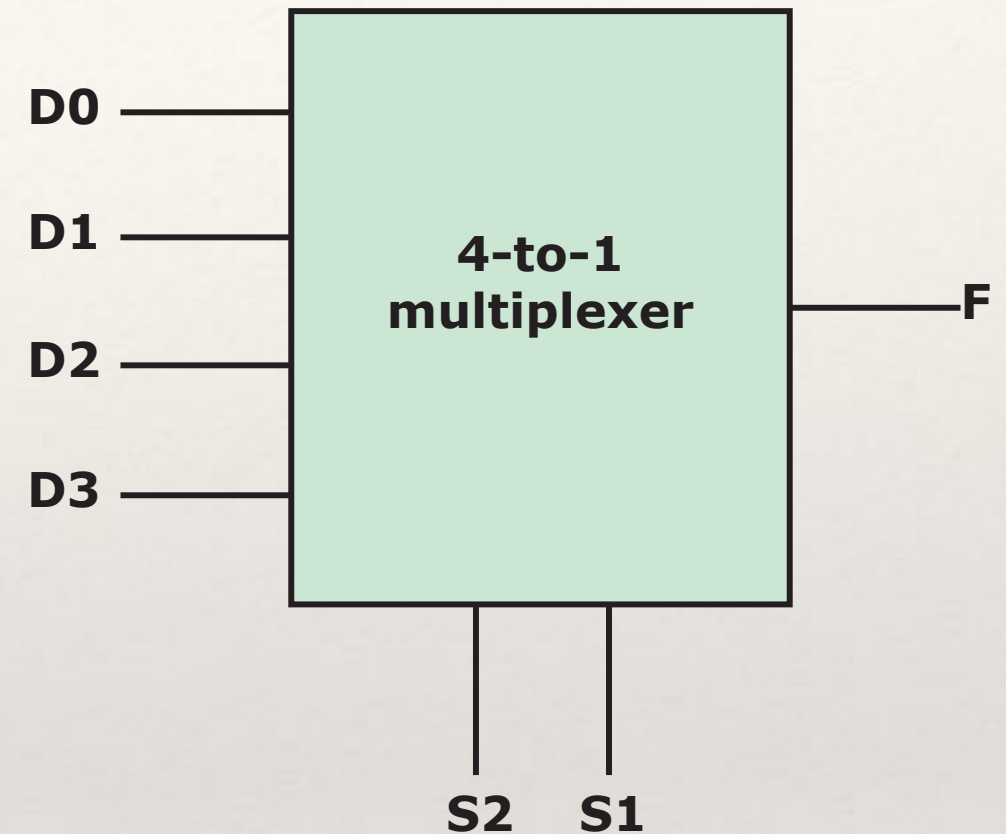
So, $F = \bar{A} \bar{C} D + \bar{A} B C + A B \bar{C} + A C D$

	$ABCD$	$AB\bar{C}D$	$AB\bar{C}\bar{D}$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$	$A\bar{B}C\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}\bar{C}\bar{D}$
BD	X	X			X		X	
$\bar{A}CD$							X	⊗
$\bar{A}BC$					X	⊗		
$AB\bar{C}$		X	⊗					
ACD	X			⊗				



Multiplexer

- ❖ The multiplexer connects multiple inputs to a single output
- ❖ At any time, one of the inputs is selected to be passed to the output
- ❖ Example
 - ❖ 4-to-1 multiplexer. One of these lines is selected to provide the output signal F. To select one of the four possible inputs, a 2-bit selection code is needed

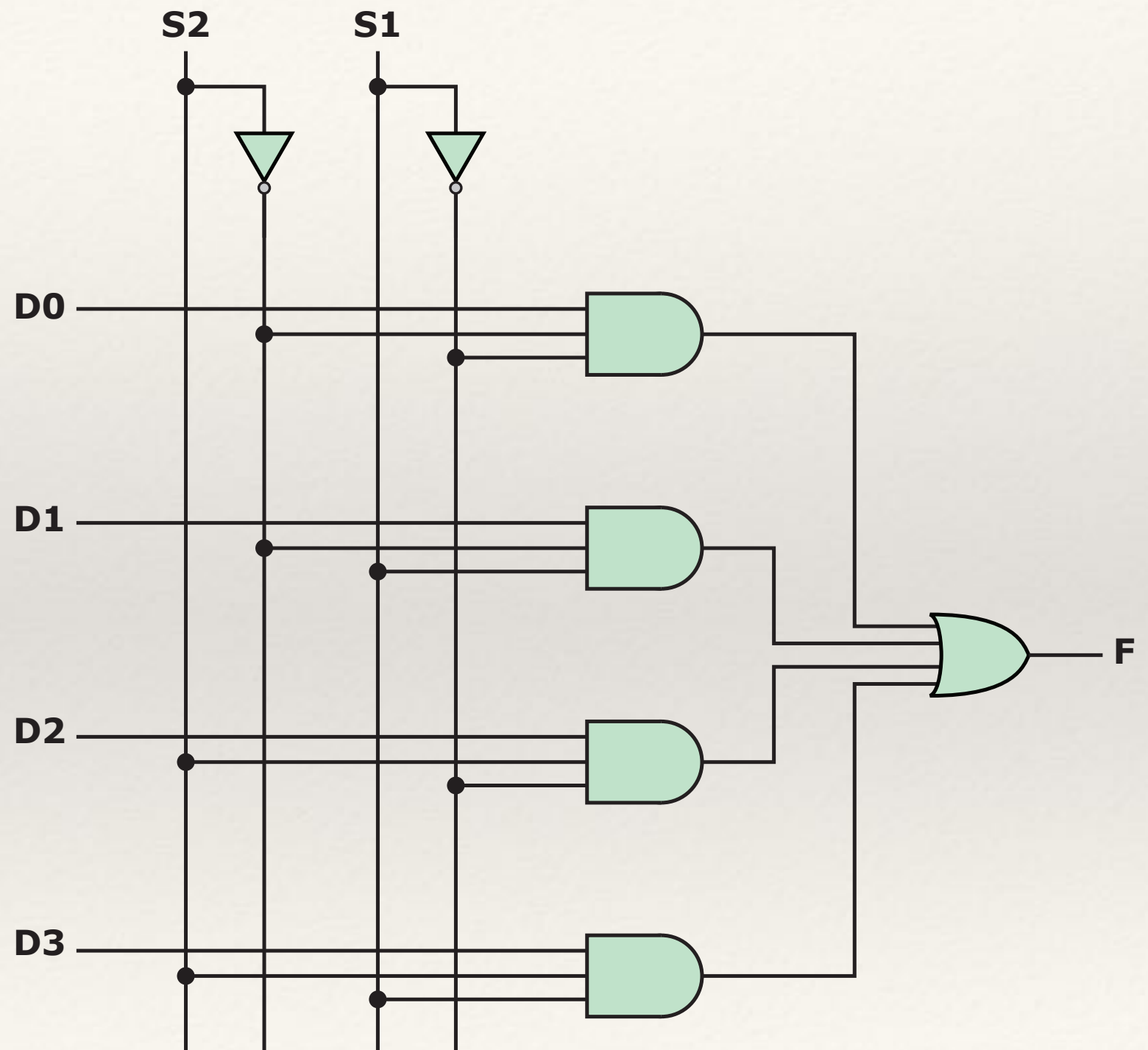


S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



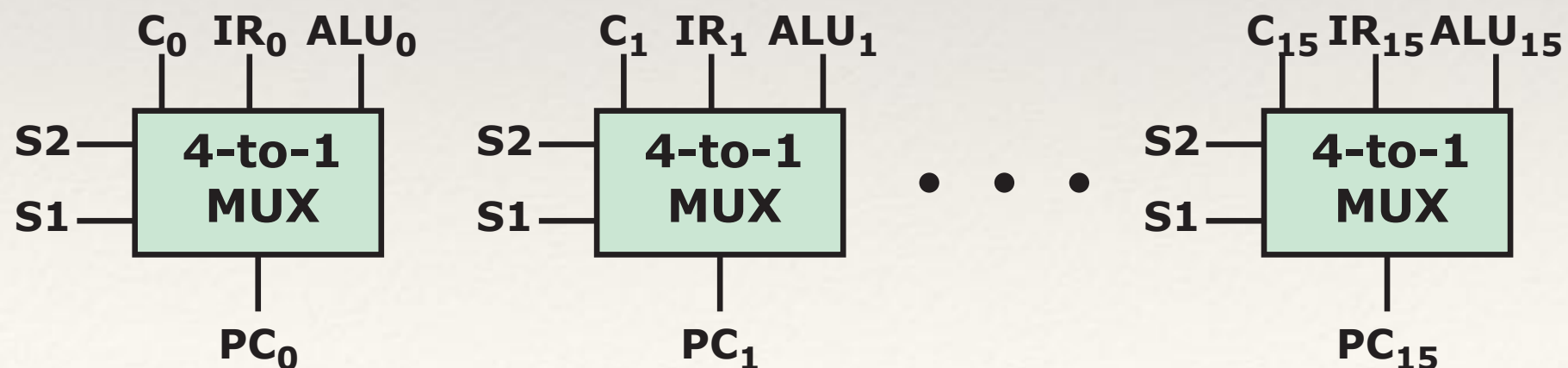
Example Multiplexer Implementation

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



Multiplexer Example Usage

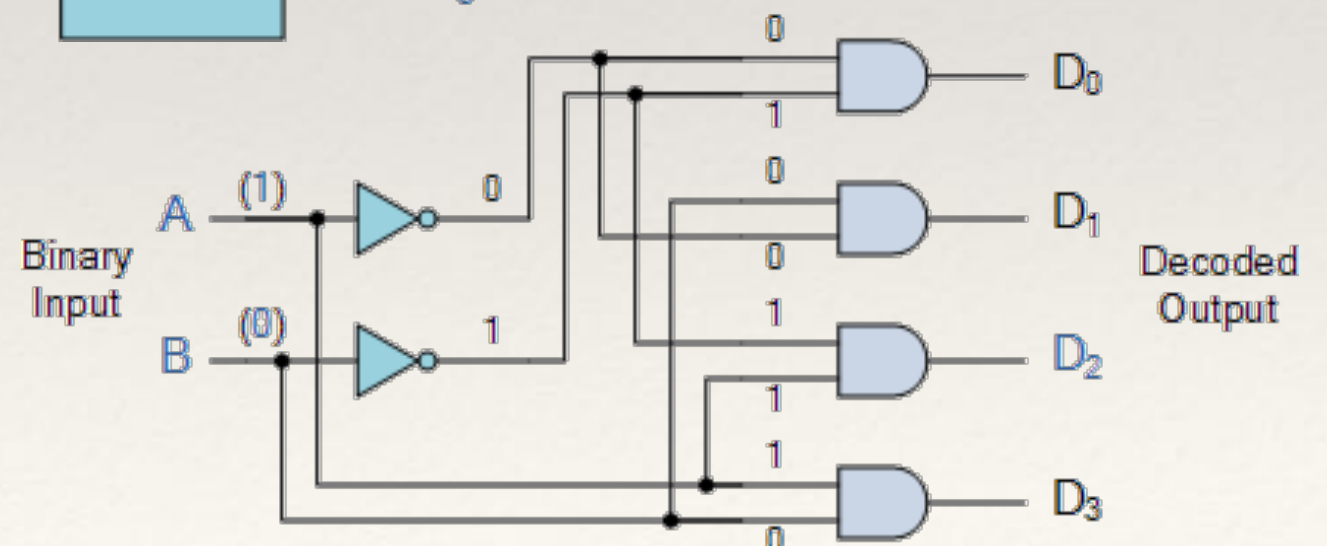
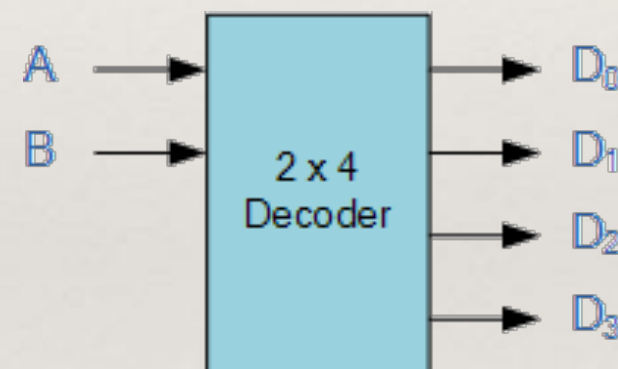
- ❖ Multiplexer Input to Program Counter
 - ❖ Program counter may come from one of several different sources:
 - ❖ A binary counter, if the PC is to be incremented for the next instruction
 - ❖ The instruction register, if a branch instruction using a direct address has just been executed
 - ❖ The output of the ALU, if the branch instruction specifies the address using a displacement mode



Decoder

- ❖ A decoder is a combinational circuit with a number of output lines, only one of which is asserted at any time
 - ❖ Which output line is asserted depends on the pattern of input lines
 - ❖ In general, a decoder has n inputs and 2^n outputs
- ❖ 2-to-4 decoder

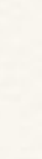
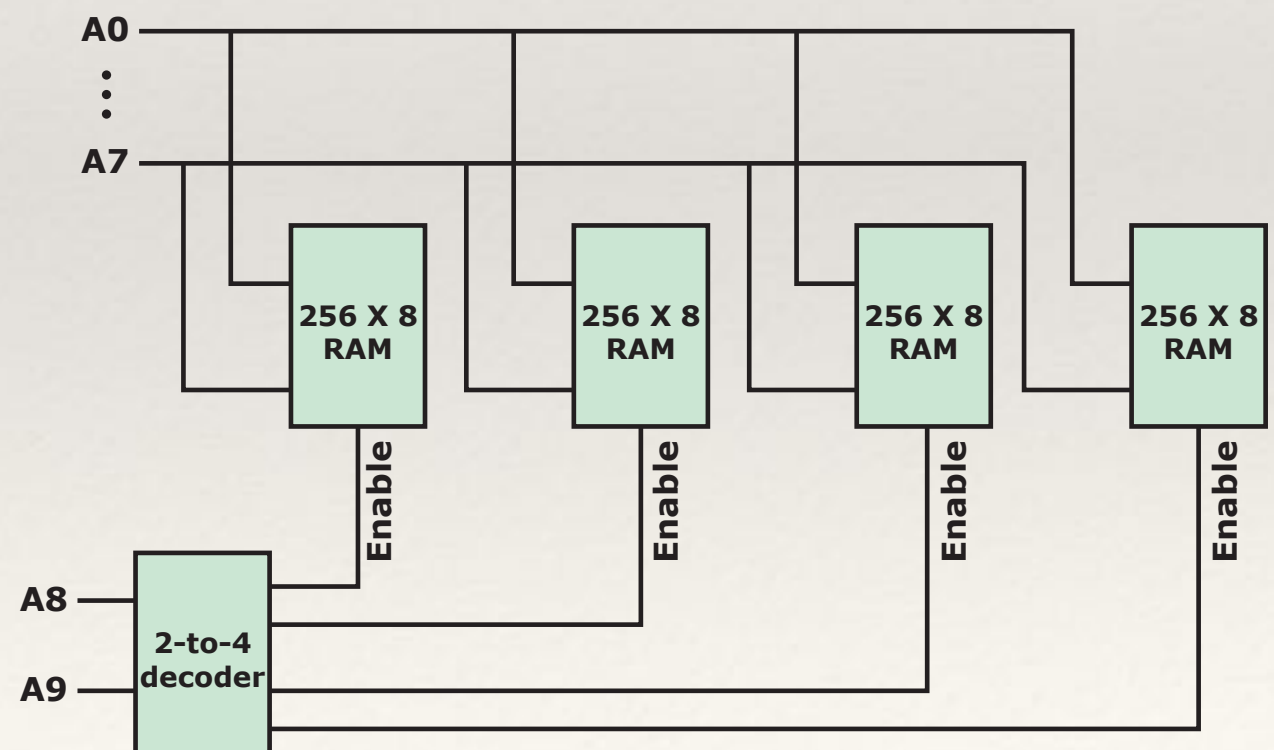
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Example Use of Decoders

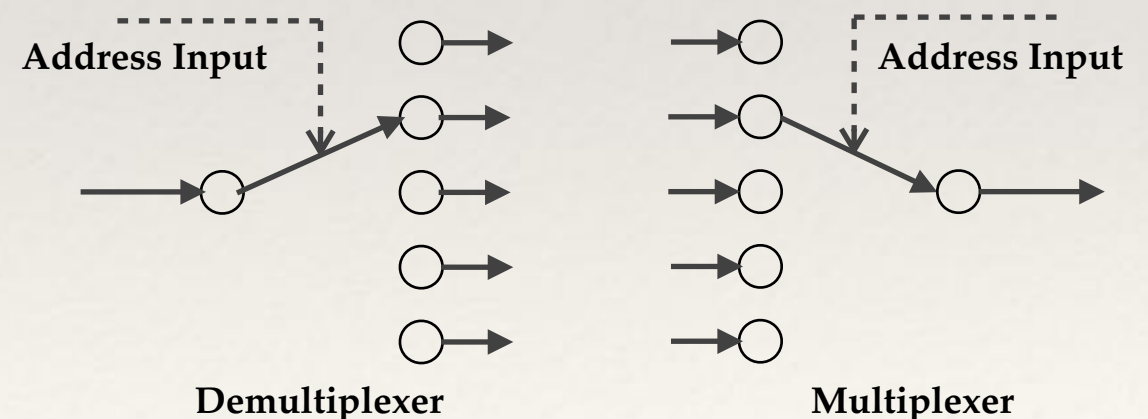
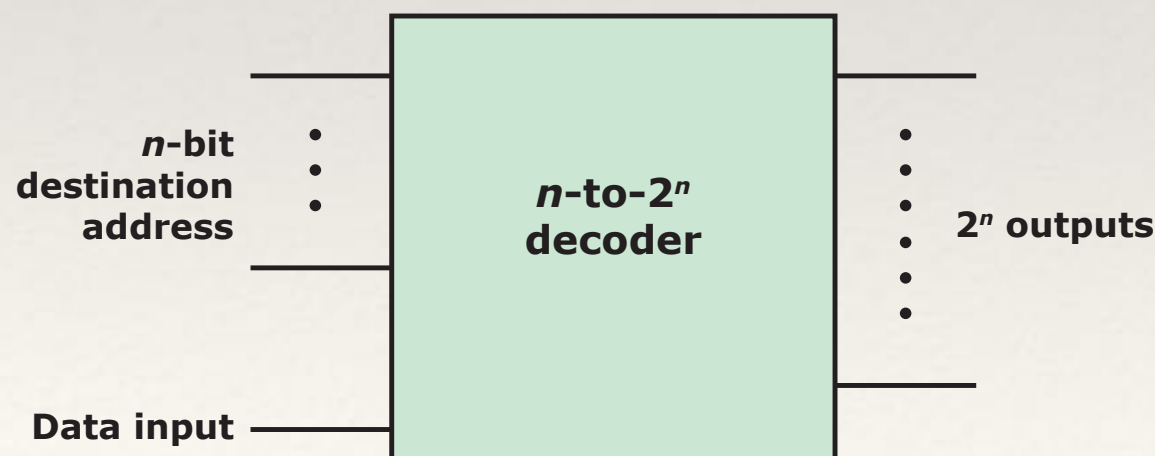
- ❖ Suppose we wish to construct a 1K-byte memory using four 256 × 8 bit RAM chips
- ❖ The higher-order 2 bits of the 10-bit address are used to select one of the four RAM chips
- ❖ 2-to-4 decoder is used whose output enables one of the four chips

Address	Chip
0000–00FF	0
0100–01FF	1
0200–02FF	2
0300–03FF	3



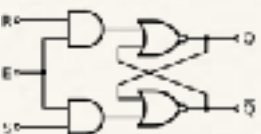
Demultiplexer

- ❖ The demultiplexer performs the inverse function of a multiplexer
- ❖ It can be constructed by a decoder, with an additional input line
 - ❖ n inputs are decoded to produce a single one of 2^n outputs. All of the 2^n output lines are ANDed with a data input line
 - ❖ The n inputs act as an address to select a particular output line, and the value on the data input line (0 or 1) is routed to that output line



Read-Only Memory (ROM)

- ❖ Memory that is implemented with combinational circuits
 - ❖ Combinational circuits are often referred to as “memoryless” circuits because their output depends only on their current input and no history of prior inputs is retained
- ❖ Memory unit that performs only the read operation
 - ❖ Binary information stored in a ROM is permanent and is created during the fabrication process
 - ❖ A given input to the ROM (address lines) always produces the same output (data lines)
 - ❖ Because the outputs are a function only of the present inputs, ROM is a combinational circuit
- ❖ A ROM can be implemented with a decoder and a set of OR gates.

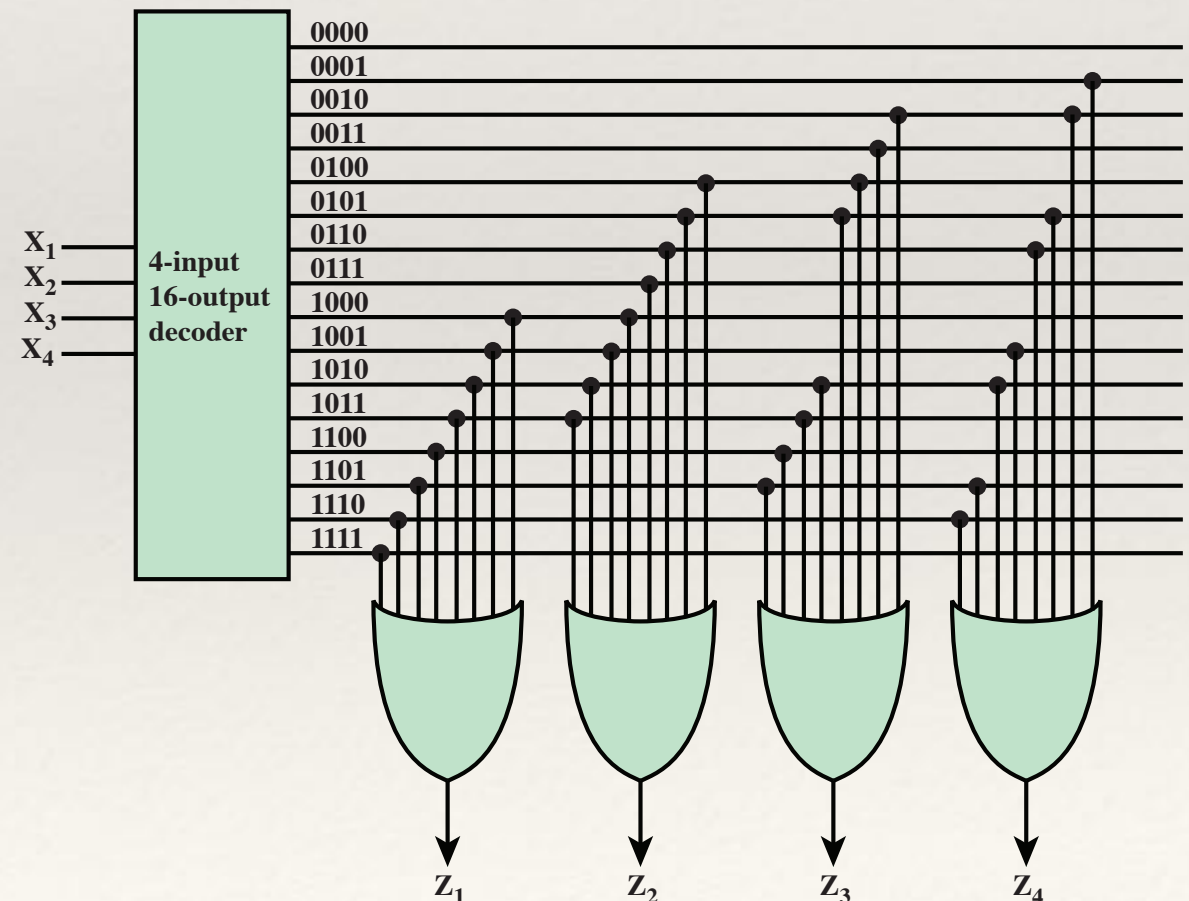


ROM Example

❖ 64 Bit ROM Example

- ❖ 16 words of 4 bits each
- ❖ Implemented using a 4-to-16 decoder and four OR gates
- ❖ Different desired results can be obtained by employing different interconnections between the decoder outputs and the OR gates

Input				Output			
X ₁	X ₂	X ₃	X ₄	Z ₁	Z ₂	Z ₃	Z ₄
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



Adders

❖ How to add two binary numbers?

❖ Single Binary Digit Addition

Case	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
C_{in}					1	1	1	1
A	0	0	1	1	0	0	1	1
B	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
Sum	0	1	1	1 0	1	1 0	1 0	1 1

(a) Single-Bit Addition

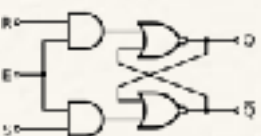
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

❖ Multiple-Digit Addition

Case	(5)	(8)	(7)	(6)	(4)
C_{in}					
A		1	1	0	1
B	+	1	0	1	1
Sum	1	C_{out} 1	C_{out} 0	C_{out} 0	C_{out} 0

(b) Addition with Carry Input

C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

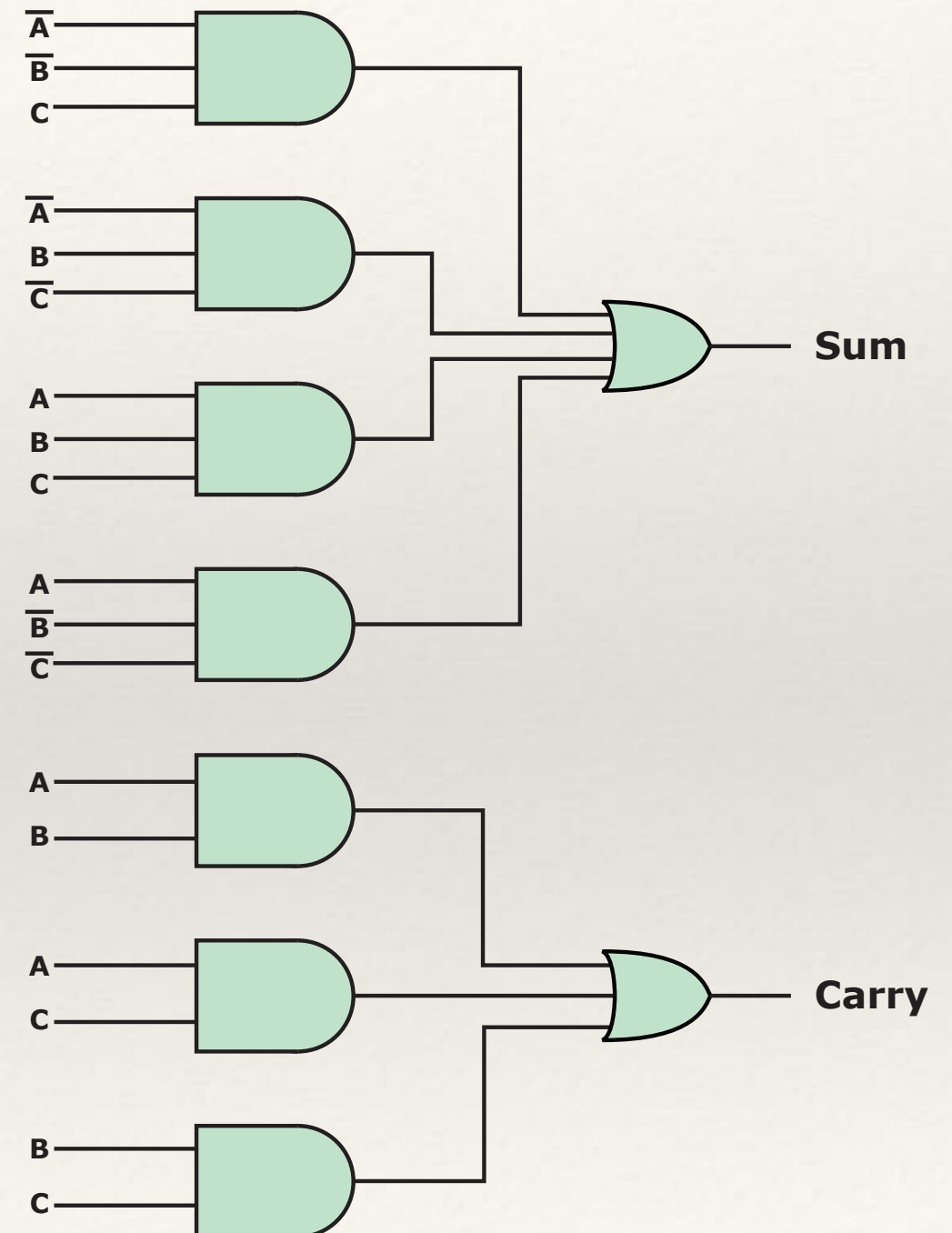


Implementation of An Adder with Carry Input

- ❖ Boolean function for Sum and Carry (C_{out})

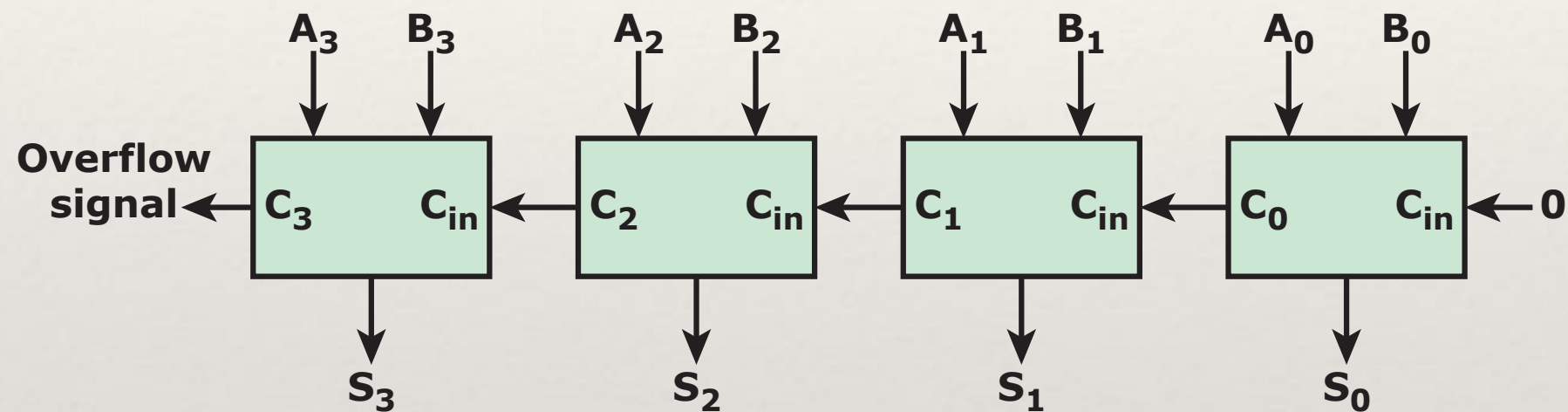
$$\begin{aligned}\text{Sum} &= \bar{A} \bar{B} C + \bar{A} B \bar{C} + A B C + A \bar{B} \bar{C} \\ \text{Carry} &= A B + A C + B C\end{aligned}$$

C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Multiple-Bit Adder

- ❖ Implement through the cascade of multiple single-bit adders
 - ❖ 4-bit adder example

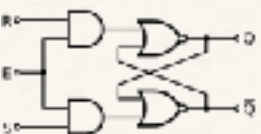


- ❖ Carry has to ripple through all the previous stages, causing delay
 - ❖ Carry look ahead might be employed to reduce such kind of delay

$$C_0 = A_0 B_0$$

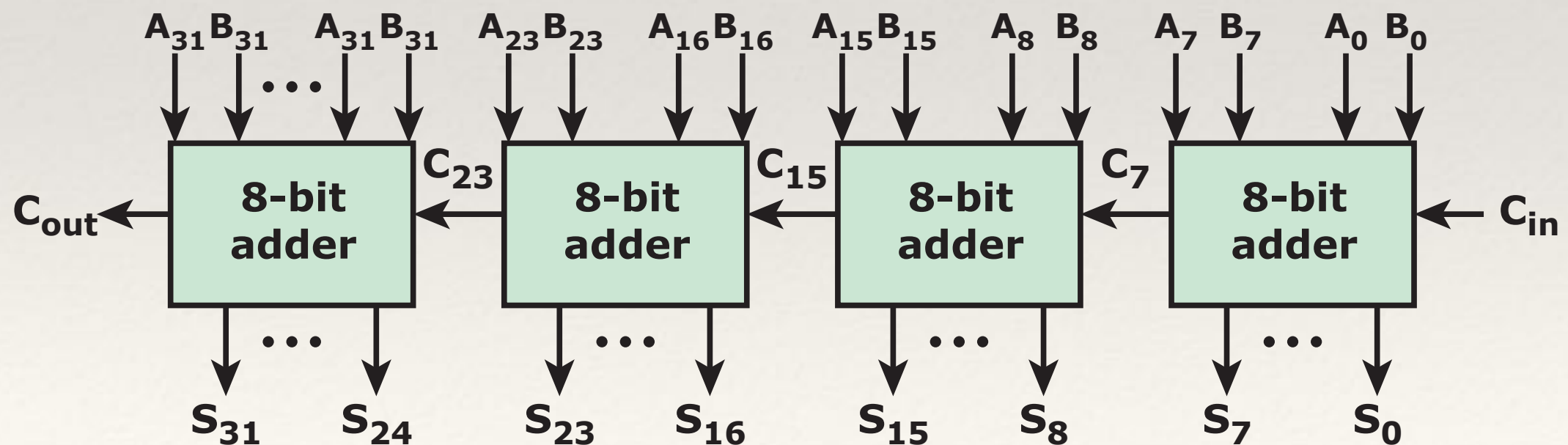
$$C_1 = A_1 B_1 + A_1 C_0 + B_1 C_0$$

$$C_2 = A_2 B_2 + A_2 A_1 B_1 + A_2 A_1 A_0 B_0 + A_2 B_1 A_0 B_0 + B_2 A_1 B_1 + B_2 A_1 A_0 B_0 + B_2 B_1 A_0 B_0$$

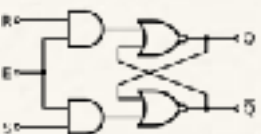
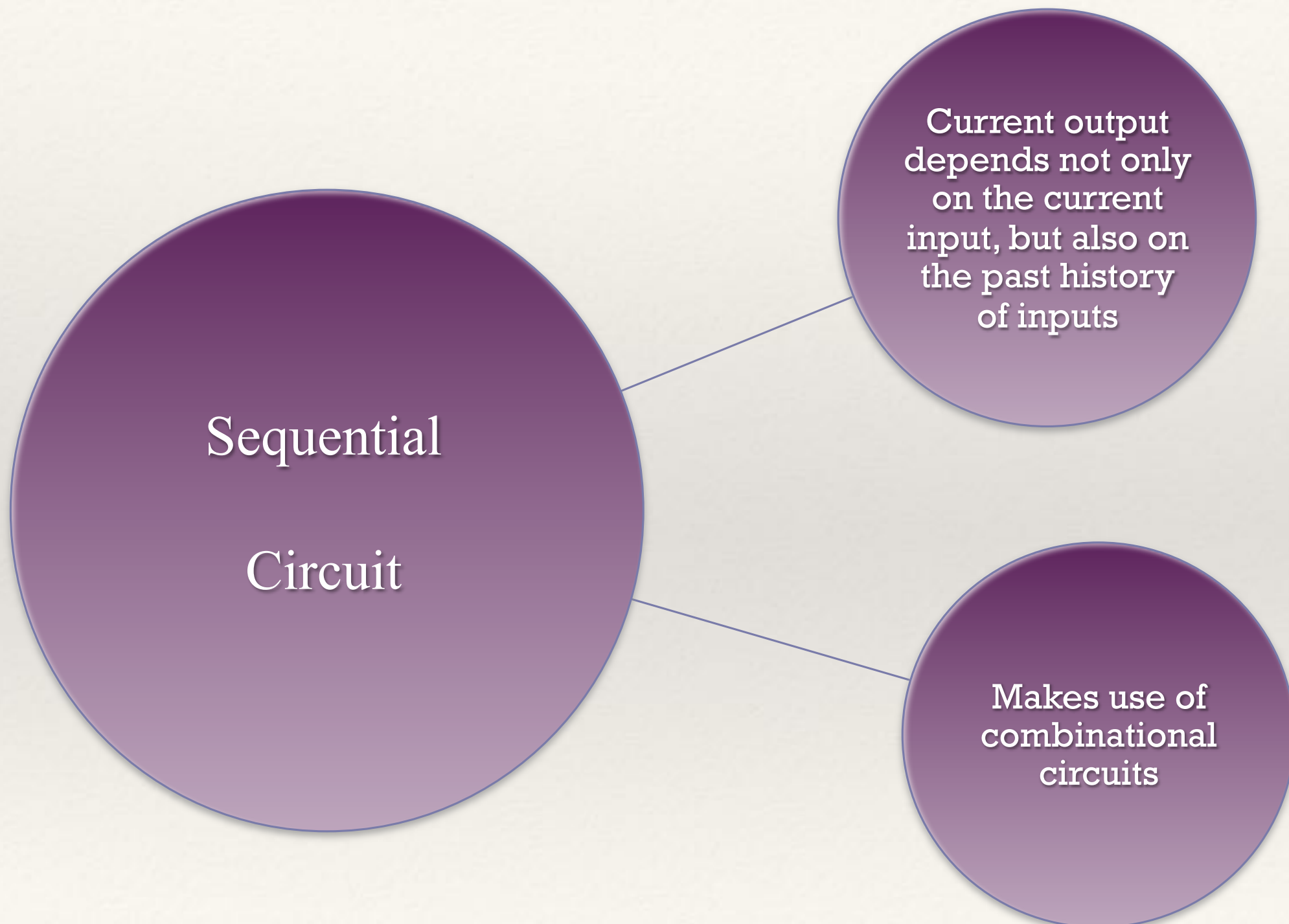


Multiple-Bit Adder (Cont'd)

- ❖ For long numbers, carry look ahead can become excessively complicated
 - ❖ So, fully carry lookahead is typically done only 4 to 8 bits at a time
- ❖ 32-bit adder can be constructed out of four 8-bit adders

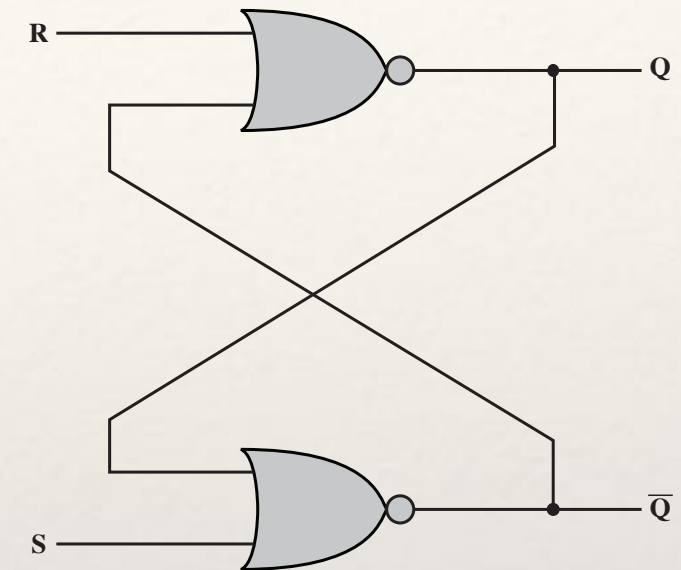


Sequential Circuit



Flip-Flops

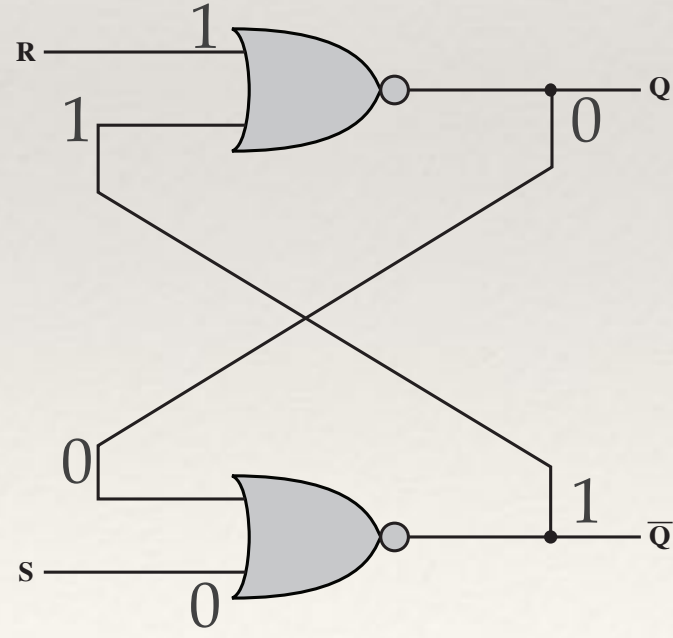
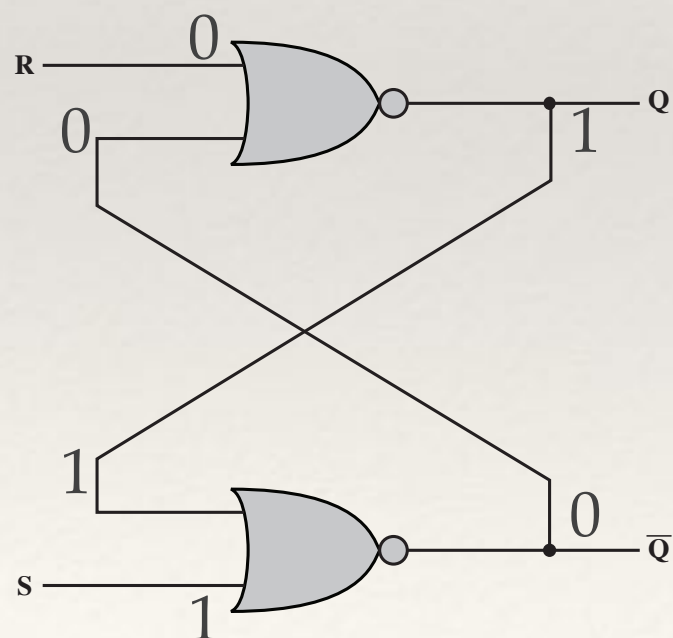
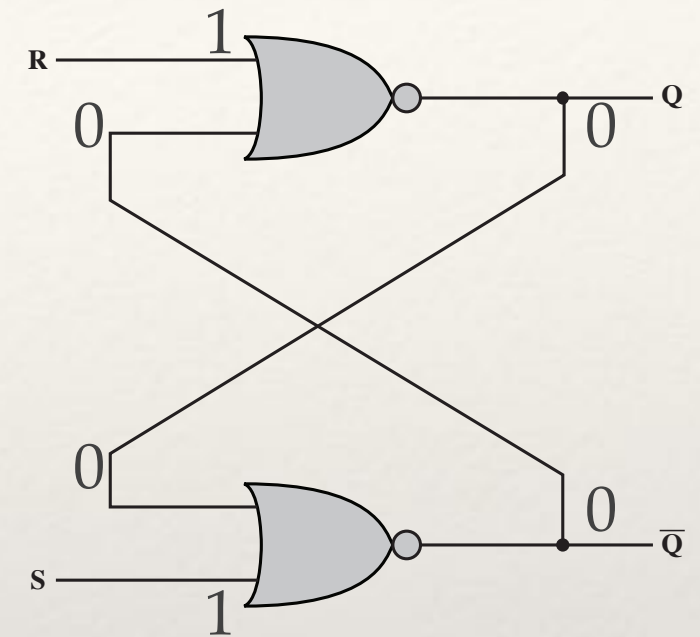
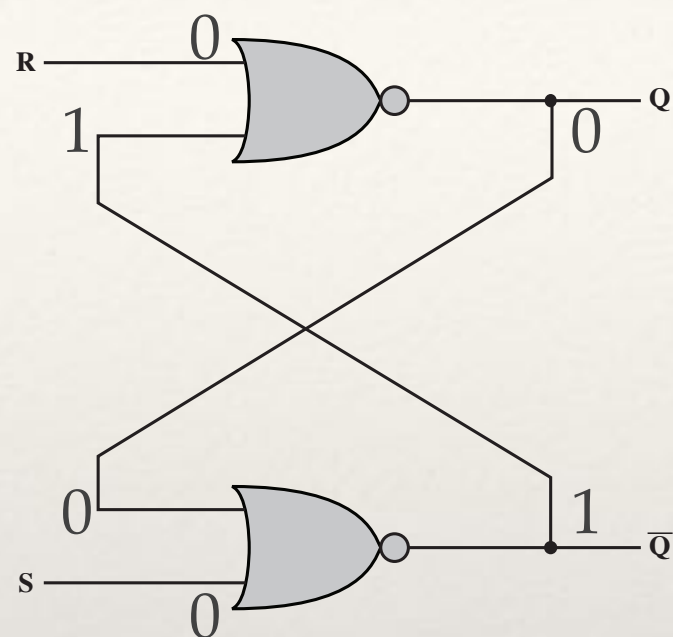
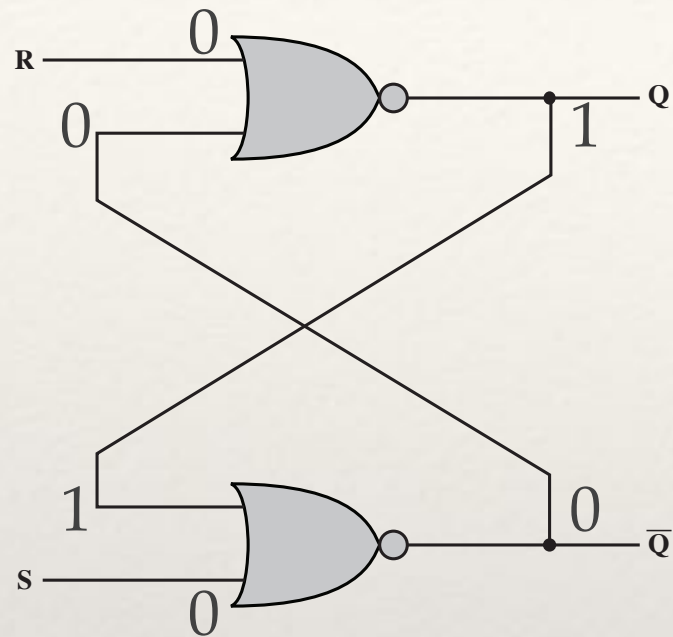
- ❖ Simplest form of sequential circuit
- ❖ There are a variety of flip-flops, all of which share two properties:
 1. The flip-flop is a bistable device. It exists in one of two states and, in the absence of input, remains in that state. Thus, the flip-flop can function as a 1-bit memory.
 2. The flip-flop has two outputs, which are always the complements of each other.



The S-R Latch Implemented with NOR Gates

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

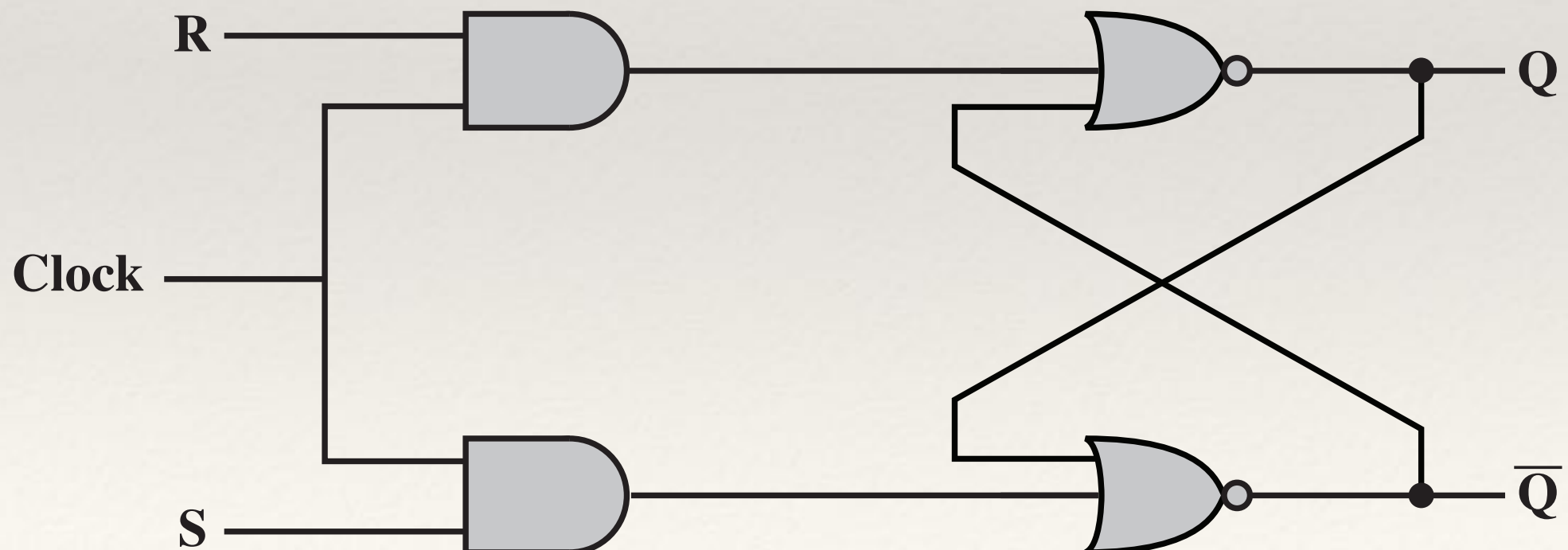
S-R Latch



Current Inputs SR	Current State Q_n	Next State Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

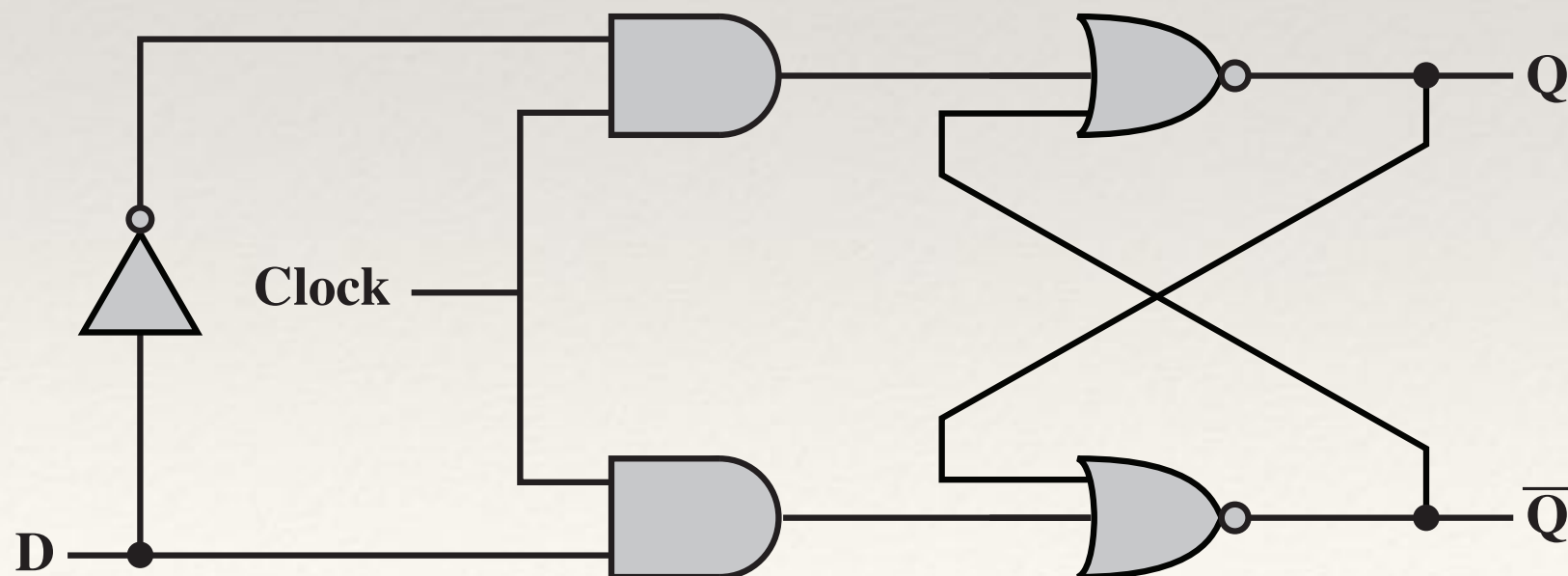
Clocked S-R Latch

- ❖ Events in the digital computer are synchronized to a clock pulse, so that changes occur only when a clock pulse occurs
- ❖ Clocked S-R flip-flop
 - ❖ The R and S inputs are passed to the NOR gates only during the clock pulse



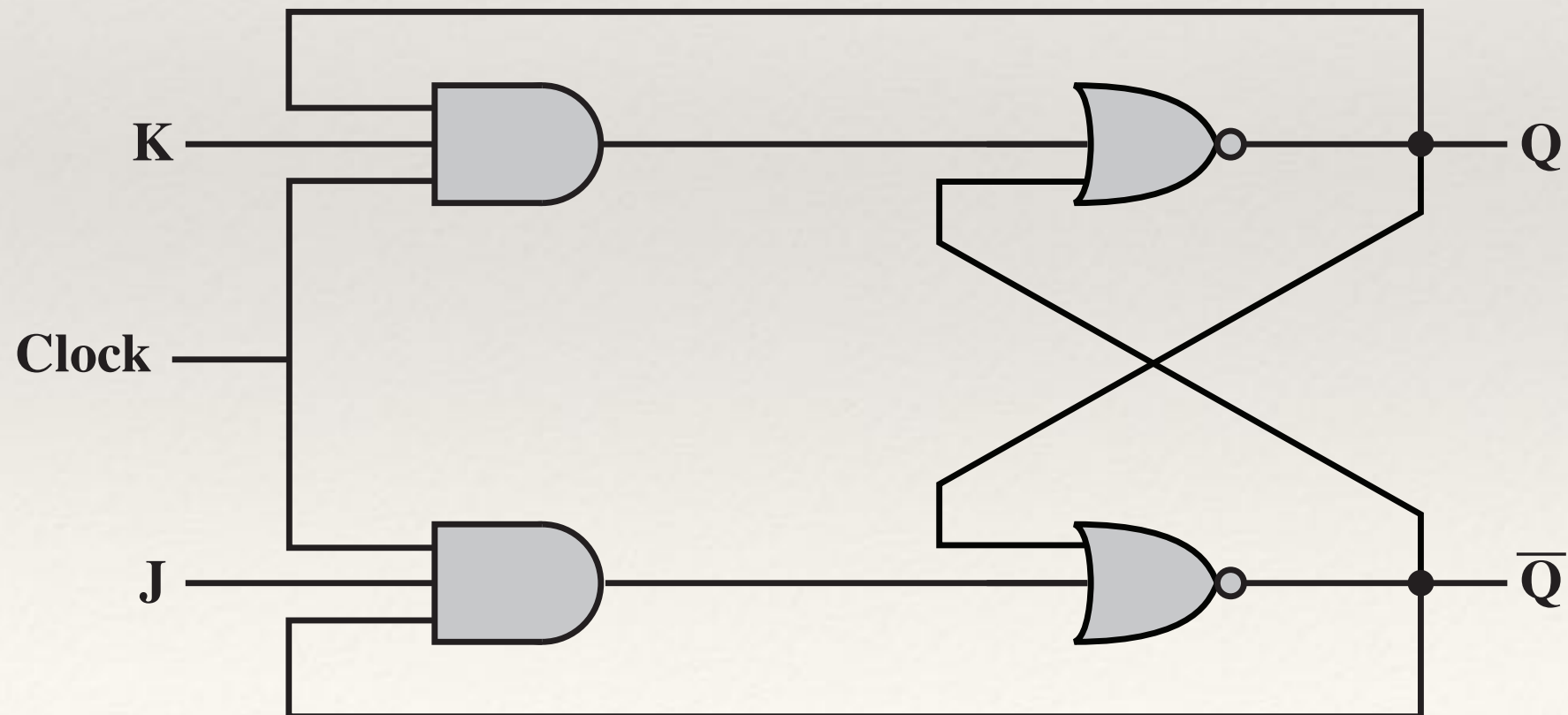
Clocked D Flip-Flop

- ❖ Avoid the problematic condition $R=1, S=1$
 - ❖ Just allow a single input, by using an inverter to ensure R and S are always opposite valued
- ❖ The D flip-flop is sometimes referred to as the data flip-flop
 - ❖ In effect, storage for one bit of data, it remembers and produces the last input
 - ❖ It is also referred to as the delay flip-flop, because it delays a 0 or 1 applied to its input for a single clock pulse.

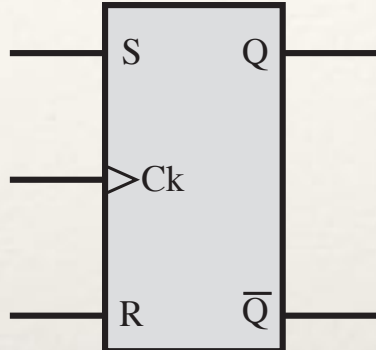
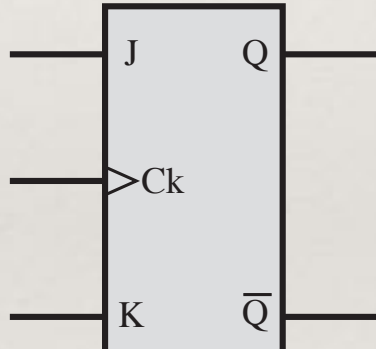
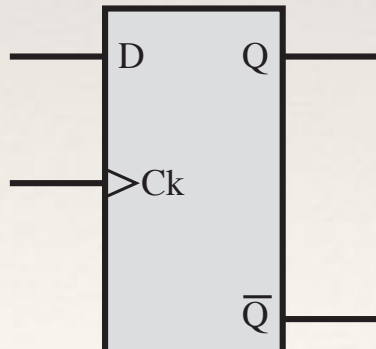


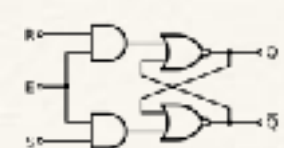
J-K Flip-Flop

- ❖ Like S-R Flip-Flop, it has two inputs, but J-K Flip-Flop is valid for all possible inputs
- ❖ In particular, when both inputs are 1, it toggles its output when the next clock arrives



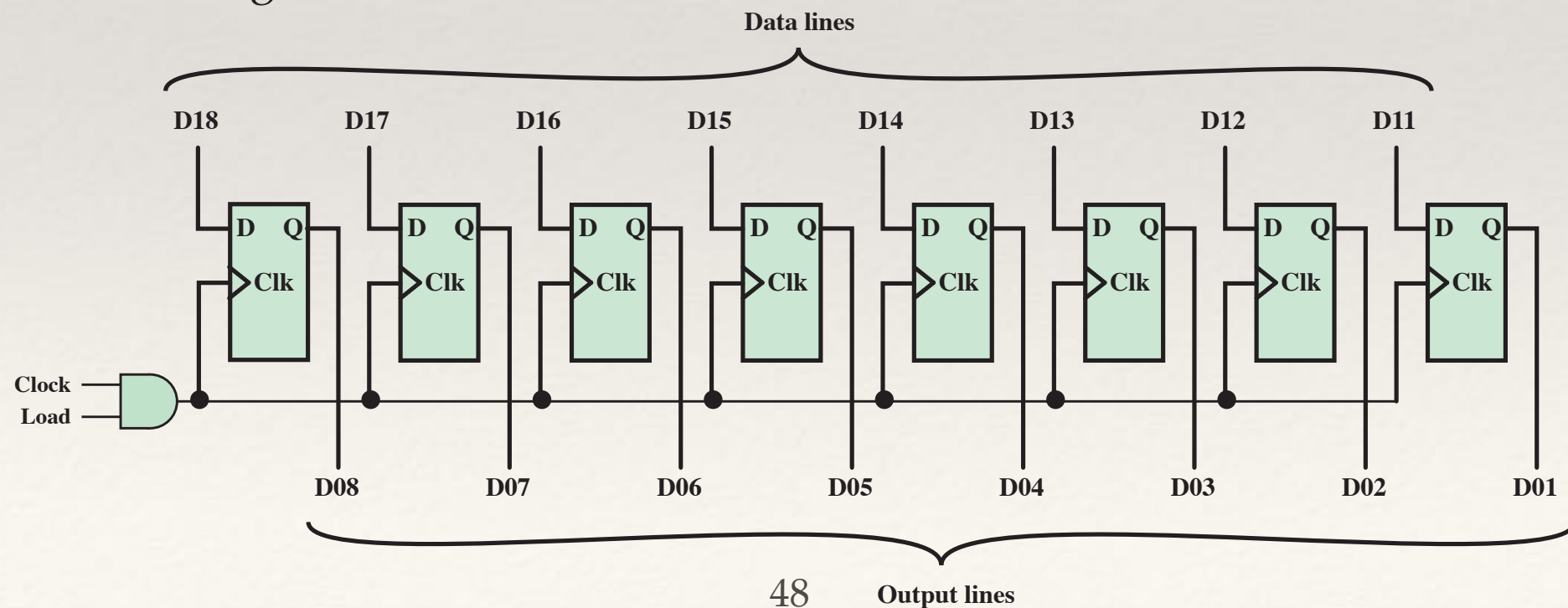
Flip-Flop Summary

Name	Graphical Symbol	Truth Table															
S-R		<table><tr><th>S</th><th>R</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>—</td></tr></table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	—
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	—															
J-K		<table><tr><th>J</th><th>K</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>$\overline{Q_n}$</td></tr></table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	$\overline{Q_n}$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	$\overline{Q_n}$															
D		<table><tr><th>D</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	D	Q_{n+1}	0	0	1	1									
D	Q_{n+1}																
0	0																
1	1																



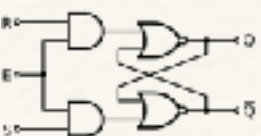
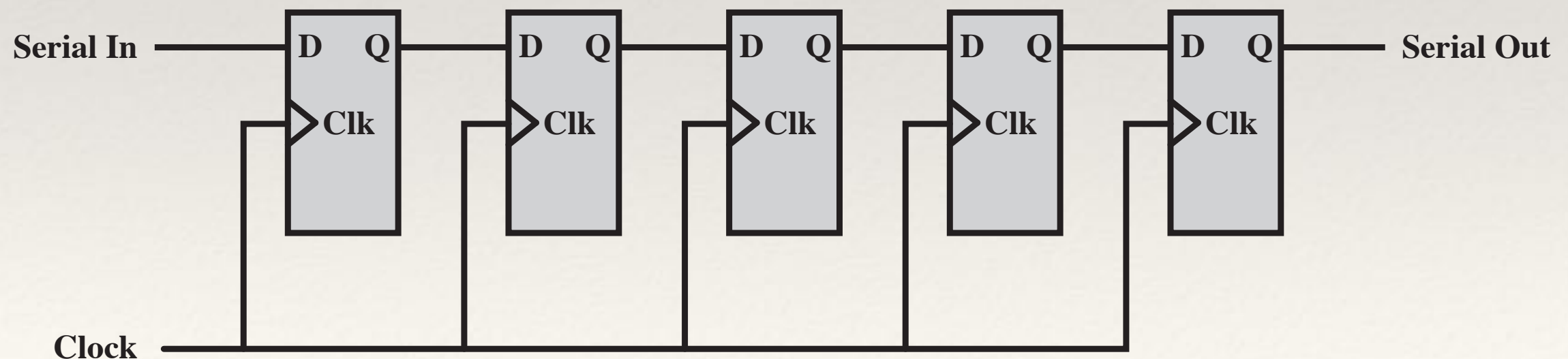
Parallel Register

- ❖ A parallel register consists of a set of 1-bit memories that can be read or written simultaneously
 - ❖ It is used to store data
- ❖ Example 8-bit register
 - ❖ A 'load' signal controls writing into the register from signal lines, D11 through D18
 - ❖ These lines might be the output of multiplexers, so that data from a variety of sources can be loaded into the register



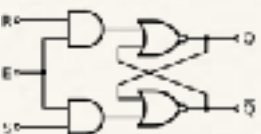
Shift Register

- ❖ A *shift register* accepts and / or transfers information serially
- ❖ Shift registers can be used to interface to serial I/O devices. In addition, they can be used within the ALU to perform logical shift and rotate functions.
- ❖ Example 5-bit shift register

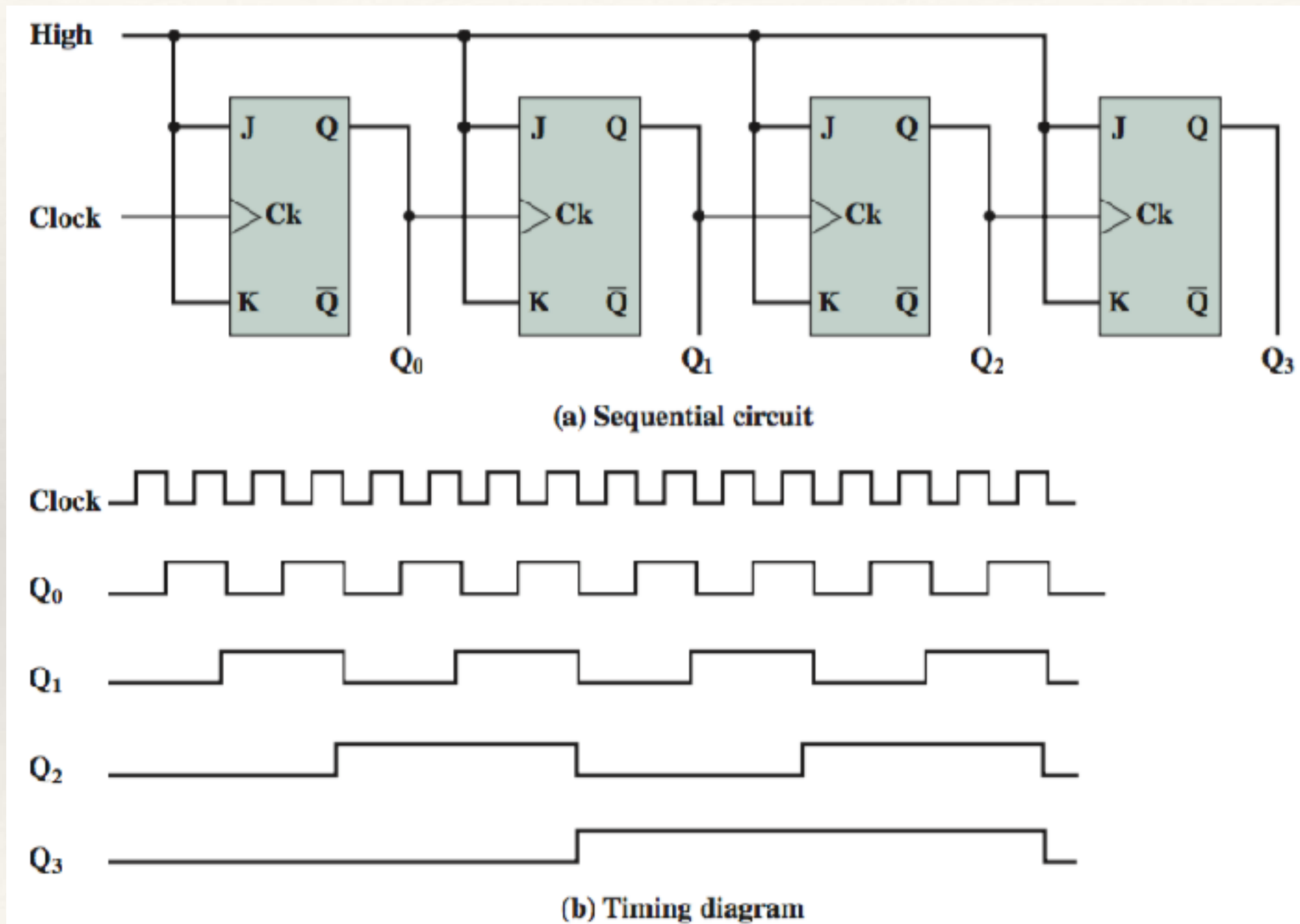


Counter Register

- ❖ A register whose value is easily incremented by 1 modulo the capacity of the register
- ❖ After the maximum value is achieved the next increment sets the counter value to 0
- ❖ An example of a counter in the CPU is the program counter
- ❖ Can be designated as:
 - ❖ Asynchronous
 - ❖ Relatively slow because the output of one flip-flop triggers a change in the status of the next flip-flop
 - ❖ Synchronous
 - ❖ All of the flip-flops change state at the same time
 - ❖ Because it is faster it is the kind used in CPUs

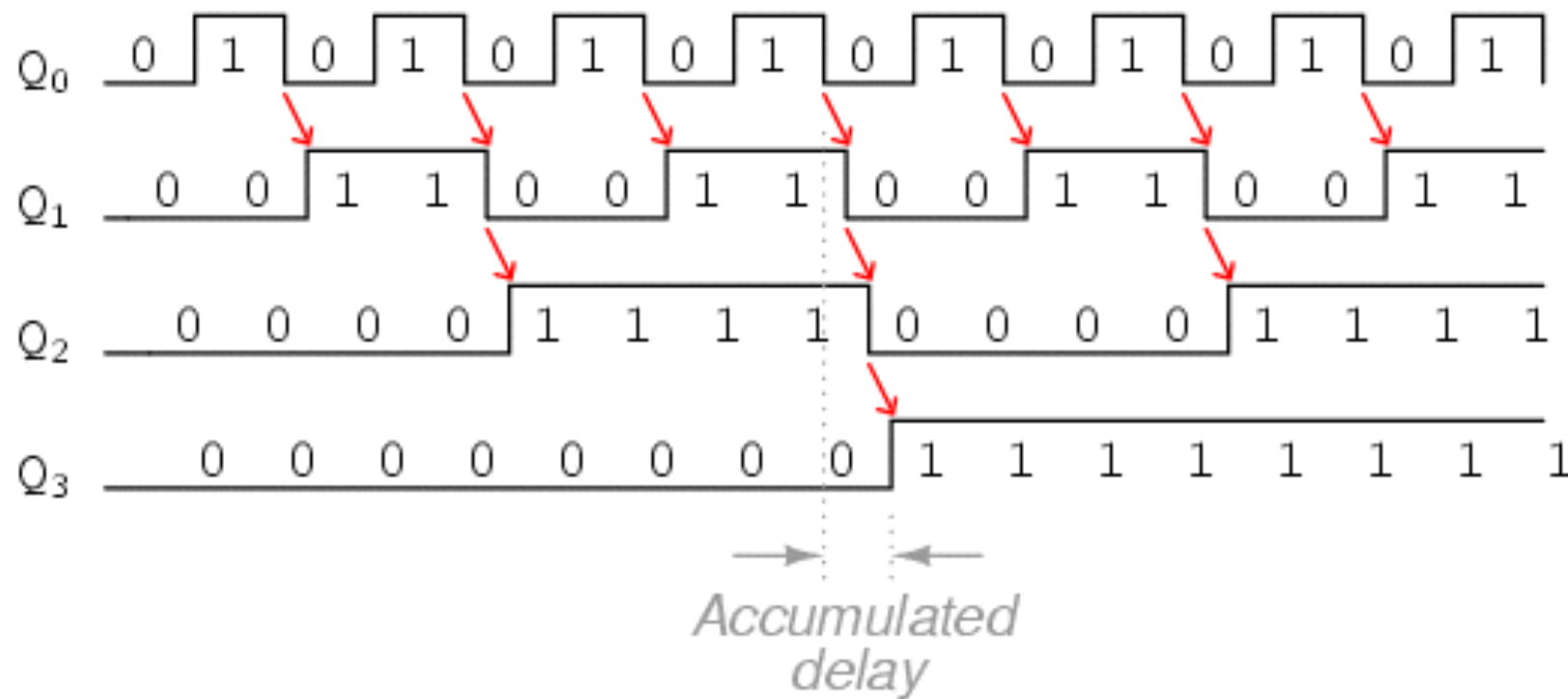


Asynchronous Counter Register

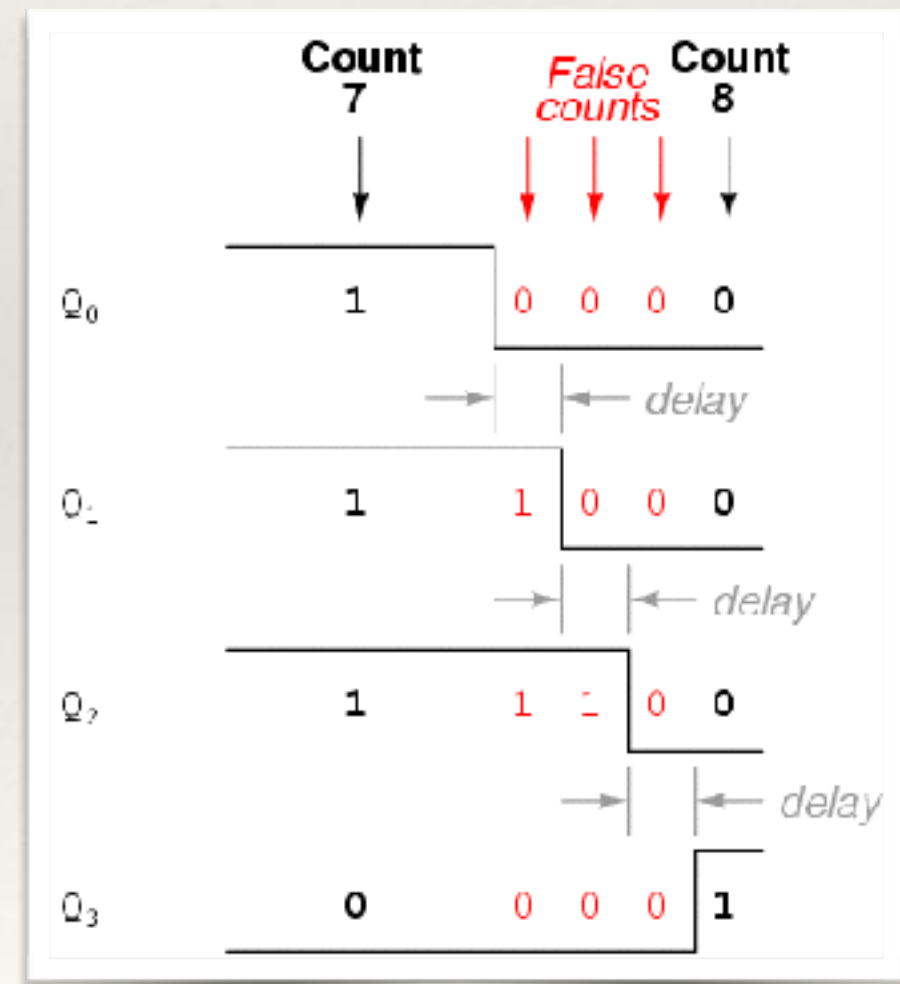


Timing Delay in Asynchronous Counter

Pulse diagram showing (exaggerated) propagation delays

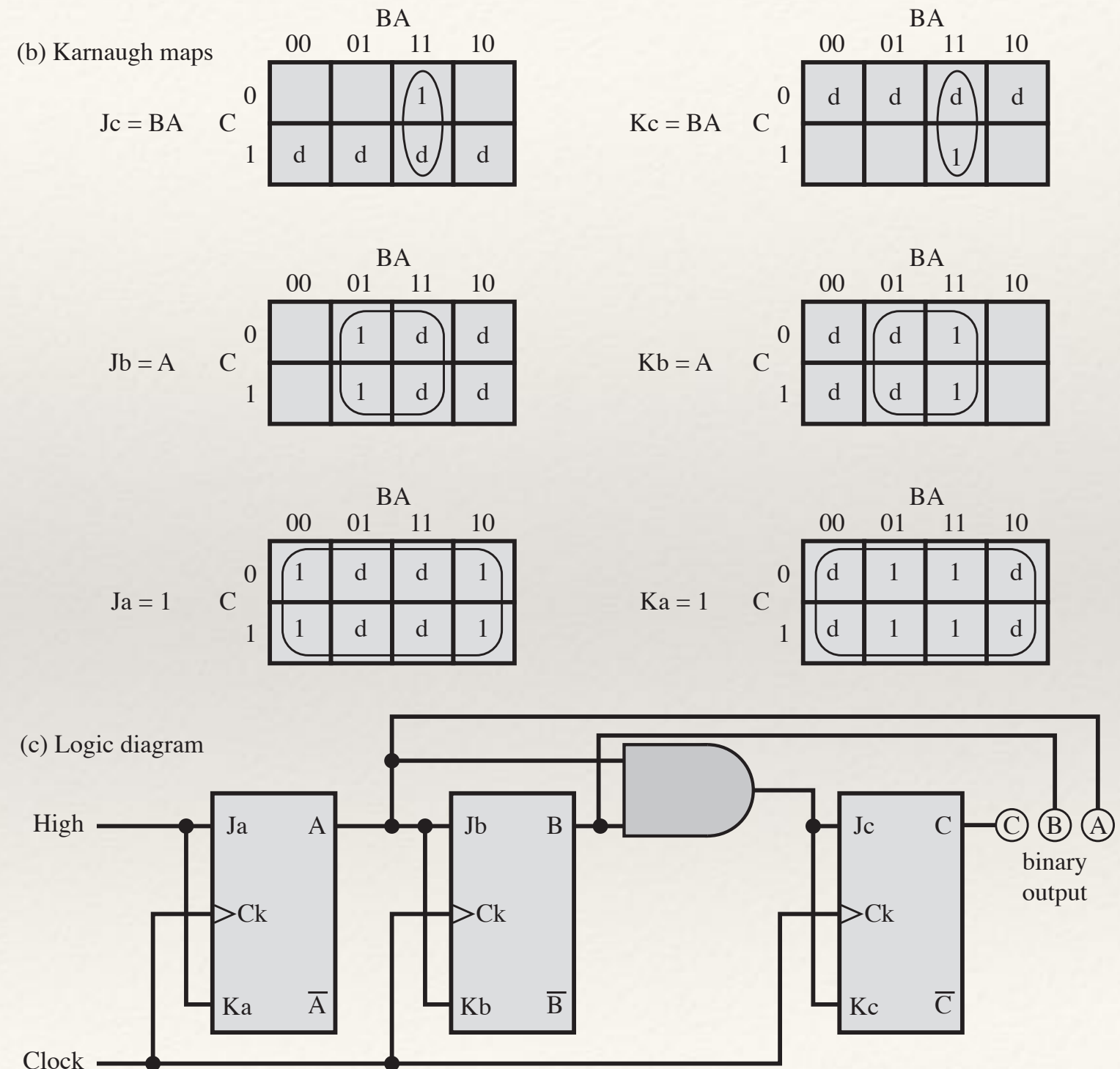


Source: <http://www.allaboutcircuits.com/textbook/digital/chpt-11/asynchronous-counters/>

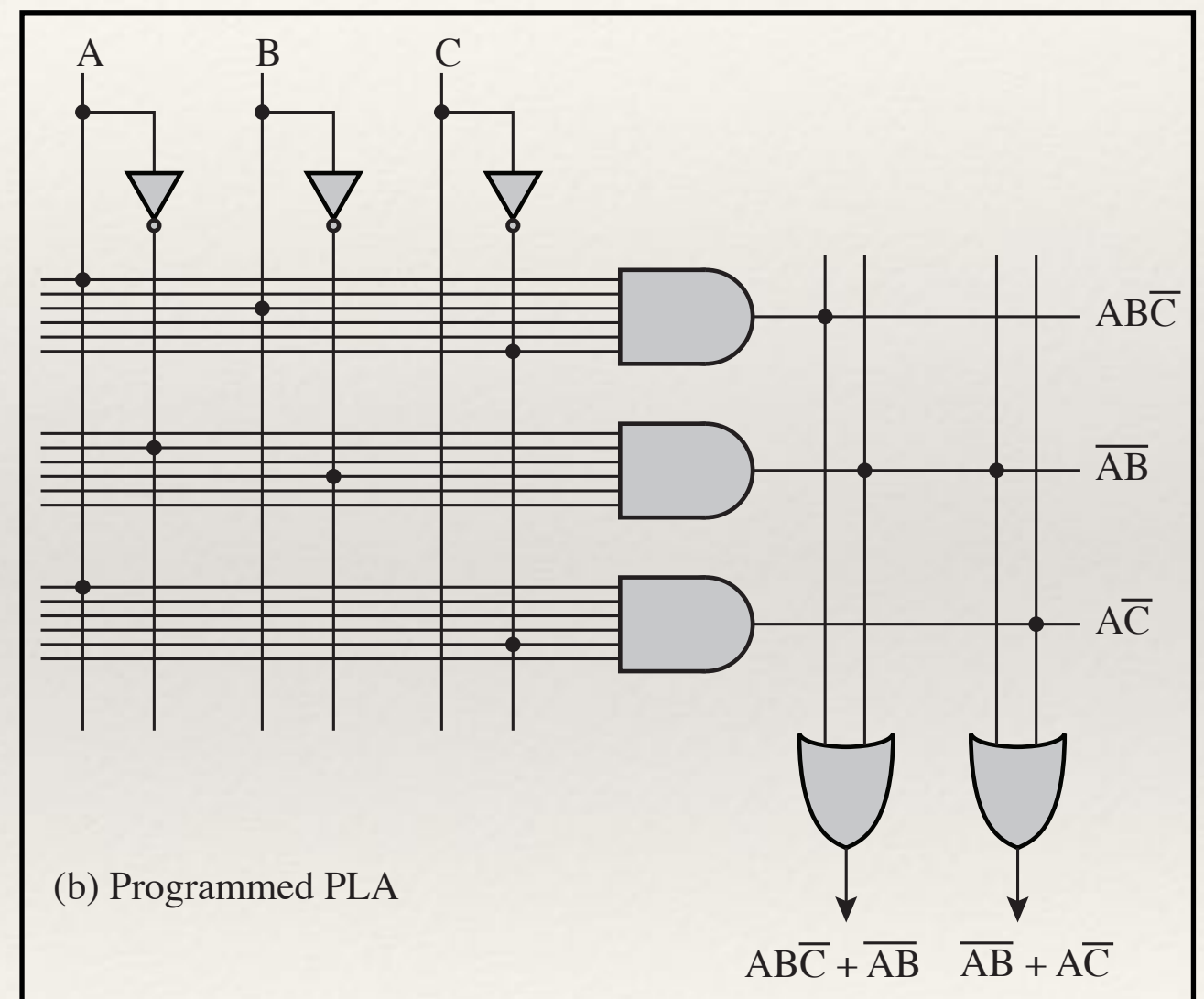
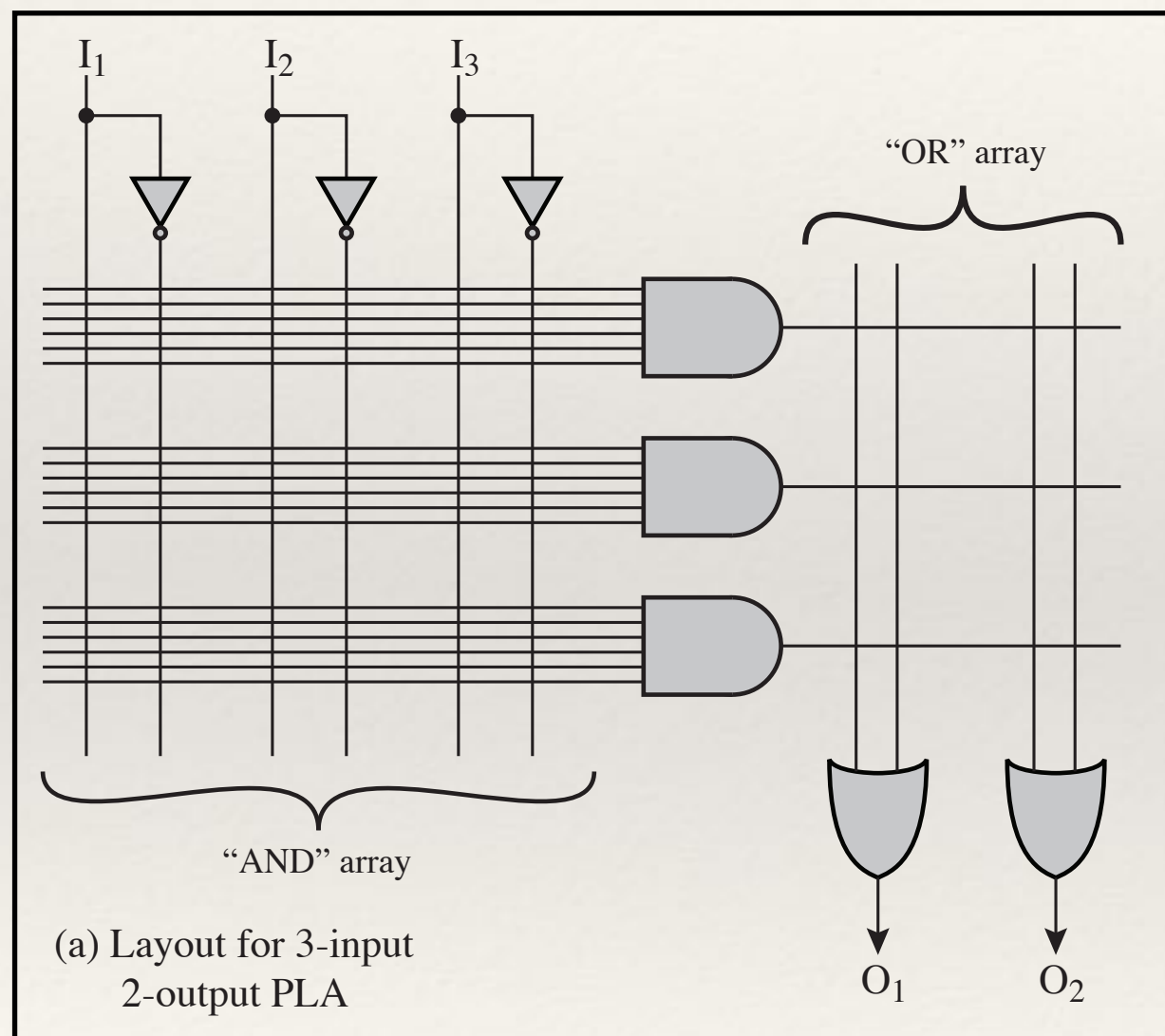


Design of Synchronous Counter

C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1



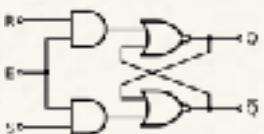
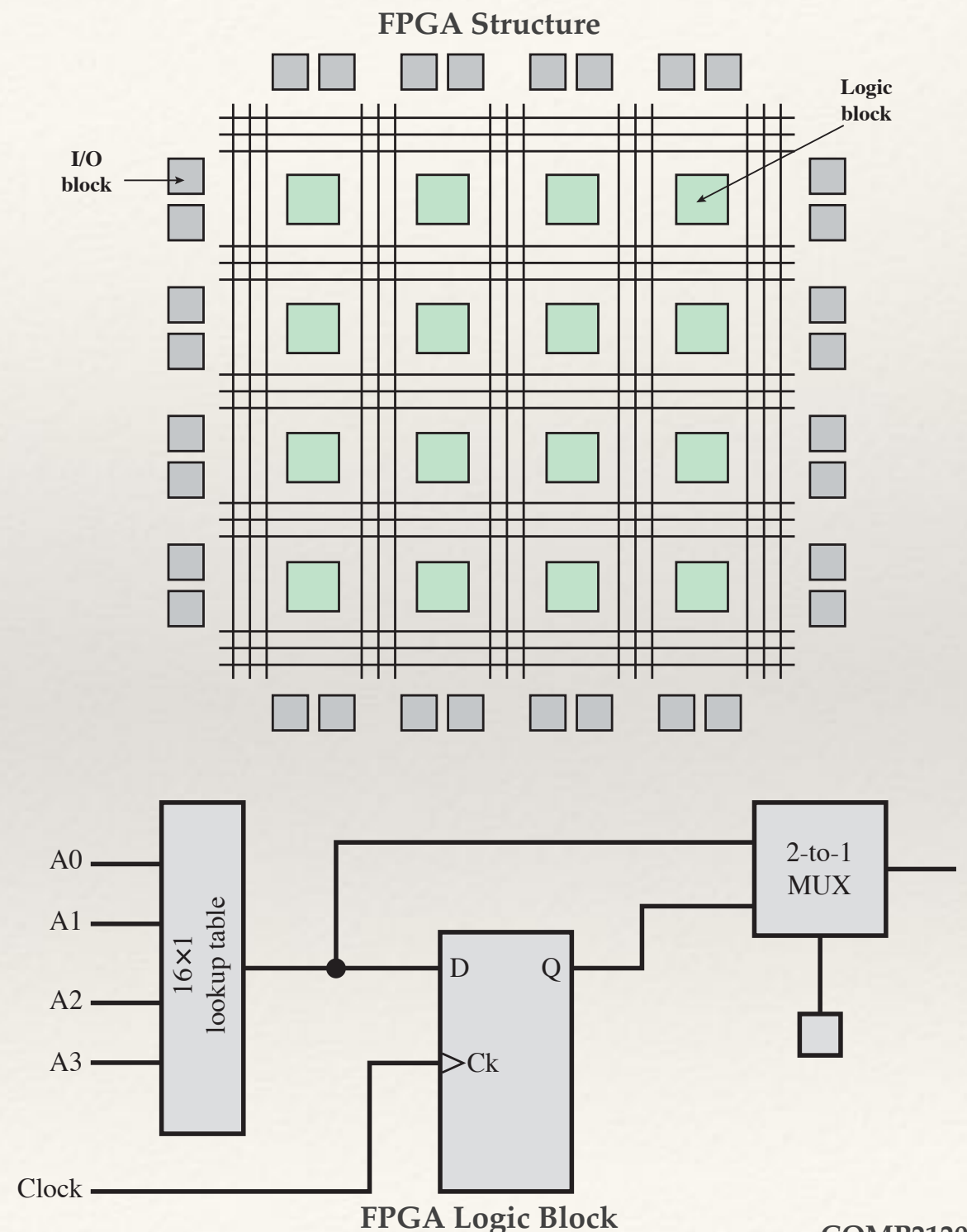
Programmable Logic Array



FPGA

❖ An FPGA consists of an array of uncommitted circuit elements, called logic blocks, and interconnect resources. An illustration of a typical FPGA architecture is shown in figure on the right. The key components of an FPGA are

- ❖ Logic block: The configurable logic blocks are where the computation of the user's circuit takes place.
- ❖ I/O block: The I/O blocks connect I/O pins to the circuitry on the chip.
- ❖ Interconnect: These are signal paths available for establishing connections among I/O blocks and logic blocks.



Chapter 11 - Summary

- ❖ Boolean Algebra
- ❖ Gates
- ❖ Combinational Circuits
 - ❖ Implementation of Boolean Functions
 - ❖ Multiplexers
 - ❖ Decoders
 - ❖ Read-Only-Memory
 - ❖ Adders
- ❖ Sequential Circuits
 - ❖ Flip-Flops
 - ❖ Registers
 - ❖ Counters
- ❖ Programmable Logic Devices
 - ❖ Programmable Logic Array
 - ❖ Field-Programmable Gate Array

