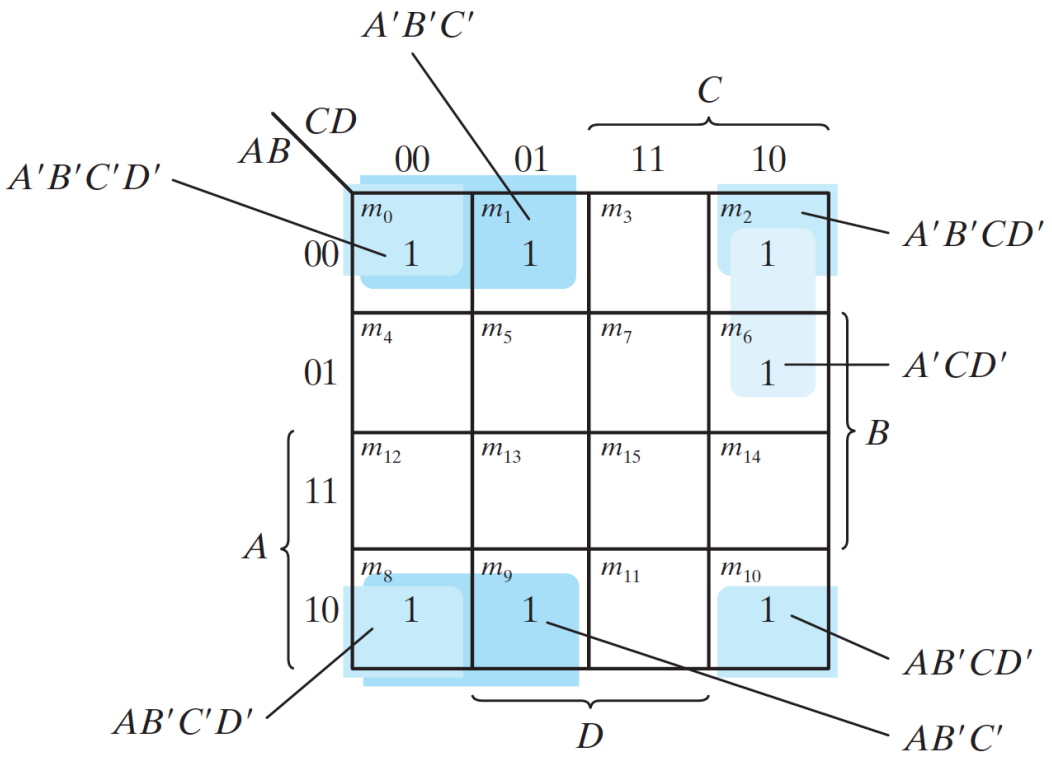
Fundamentals of Electronics

ECE 101

Exercise: Simplify the following Boolean function

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$



$$A'B'C'D' + A'B'CD' = A'B'D'$$
$$AB'C'D' + AB'CD' = AB'D'$$
$$A'B'D' + AB'D' = B'D'$$
$$A'B'C' + AB'C' = B'C'$$

$$F = B'D' + B'C' + A'CD'$$

Prime Implicant

A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.

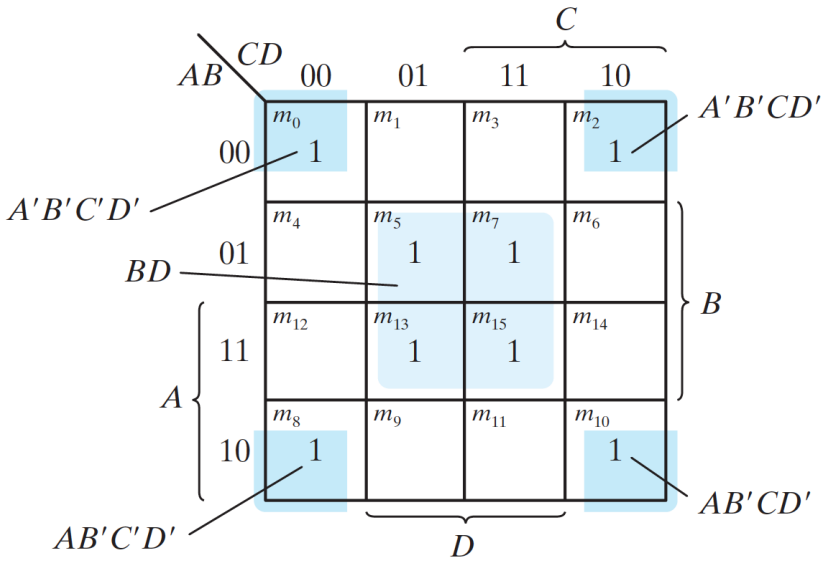
- A single 1 on a map represents a prime implicant if it is not adjacent to any other 1's.
- Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares.
- Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent squares, and so on.

The **essential prime implicants** are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it.

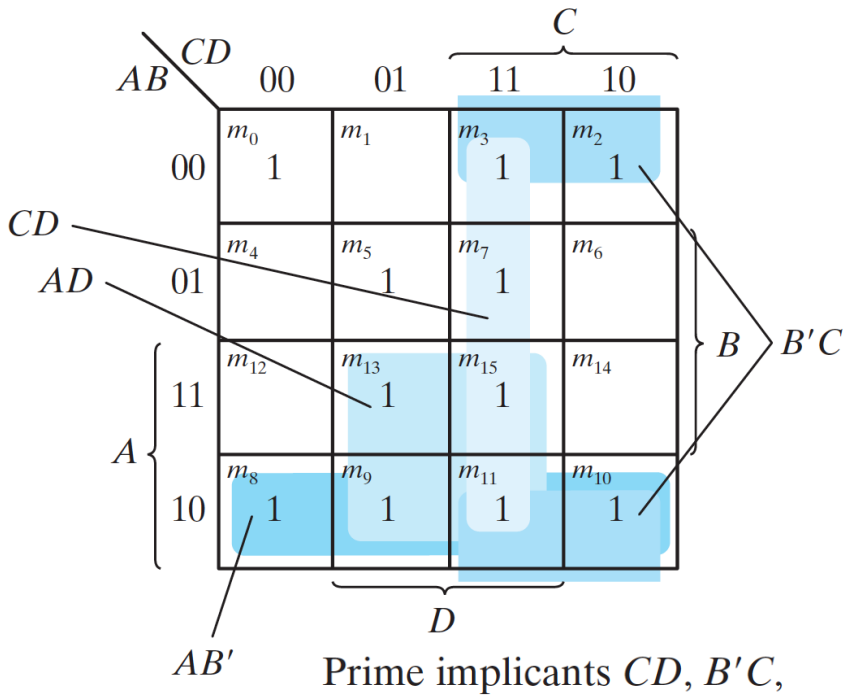
The prime implicant is essential if it is the only prime implicant that covers the minterm

- The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants.
- The simplified expression is obtained from the **logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants.**
- Occasionally, there may be more than one way of combining squares, and each combination may produce an equally simplified expression.

Example: $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$



Essential prime implicants BD and $B'D'$



Prime implicants CD, $B'C$, AD, and AB'

The simplified expression is obtained from the logical sum of the two essential prime implicants and any two prime implicants

$$F = BD + B'D' + CD + AD$$
$$= BD + B'D' + CD + AB'$$
$$= BD + B'D' + B'C + AD$$
$$= BD + B'D' + B'C + AB'$$

Logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants.

Prime Implicant

A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.

The **essential prime implicants** are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it.

The prime implicant is essential if it is the only prime implicant that covers the minterm

- The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants.
- The simplified expression is obtained from the **logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants.**
- Occasionally, there may be more than one way of combining squares, and each combination may produce an equally simplified expression.

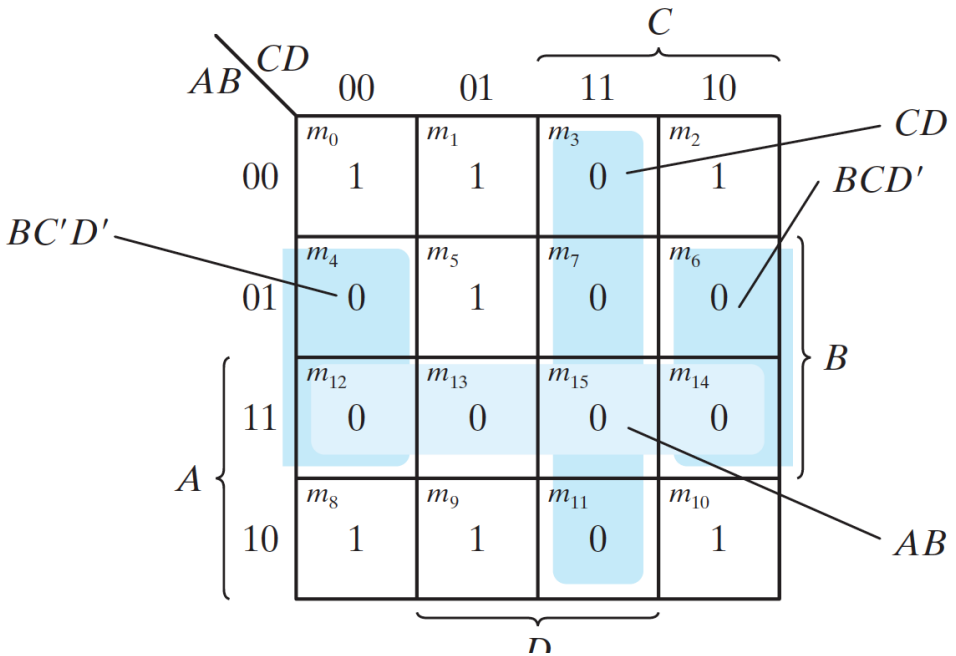
PRODUCT-OF-SUMS SIMPLIFICATION

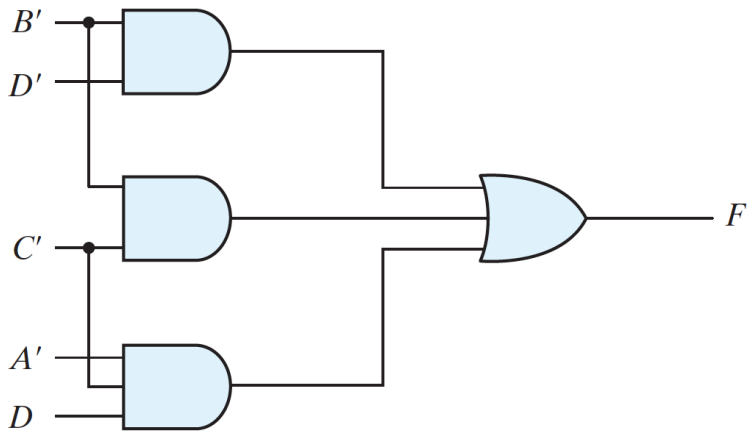
$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

$$F = B'D' + B'C' + A'C'D$$

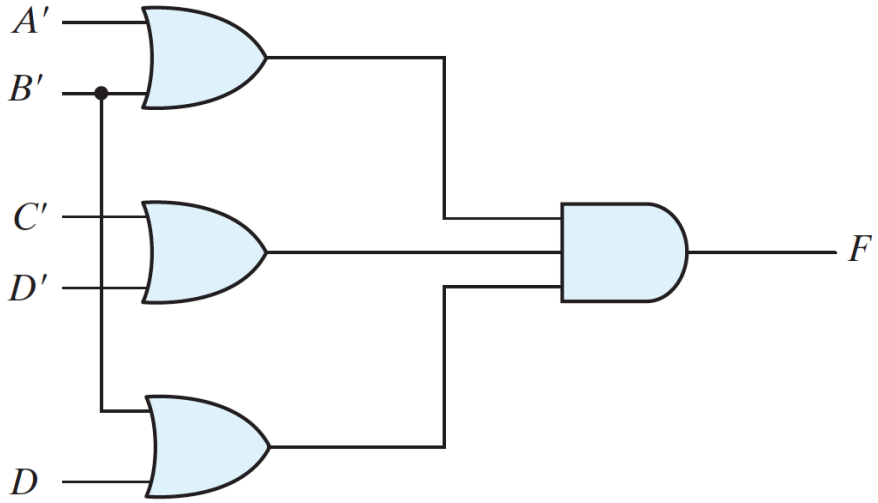
$$F' = AB + CD + BD'$$

$$F = (A' + B')(C' + D')(B' + D)$$





(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

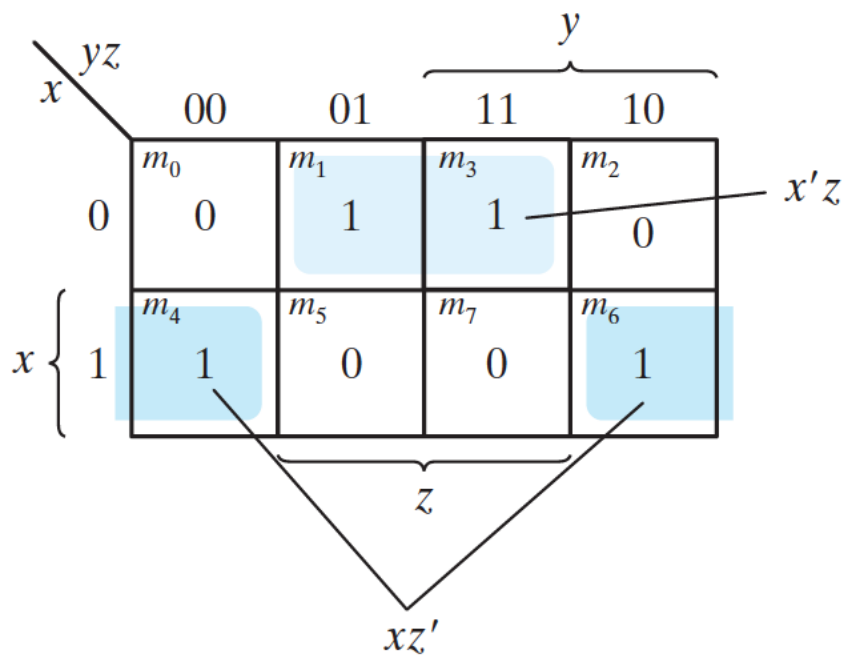
K-map for PRODUCT-OF-SUMS representation

$$F(x, y, z) = \Pi(0, 2, 5, 7)$$

$$F = x'z + xz'$$

$$F' = xz + x'z'$$

$$F = (x' + z')(x + z)$$



$$F = (A' + B' + C')(B + D)$$

$$F' = ABC + B'D'$$

To build a K-map of F from POS, mark 0's in the squares representing the minterms of F.

Don’t care conditions

Sometimes, functions have unspecified outputs for some input combinations, such functions are called **incompletely specified functions**.

In such case, we generally **don’t care** what value is assumed by the function for the unspecified minterms.

They are called as **don’t care conditions**.

These **don’t-care conditions** can be used on a map to provide further simplification of the Boolean expression.

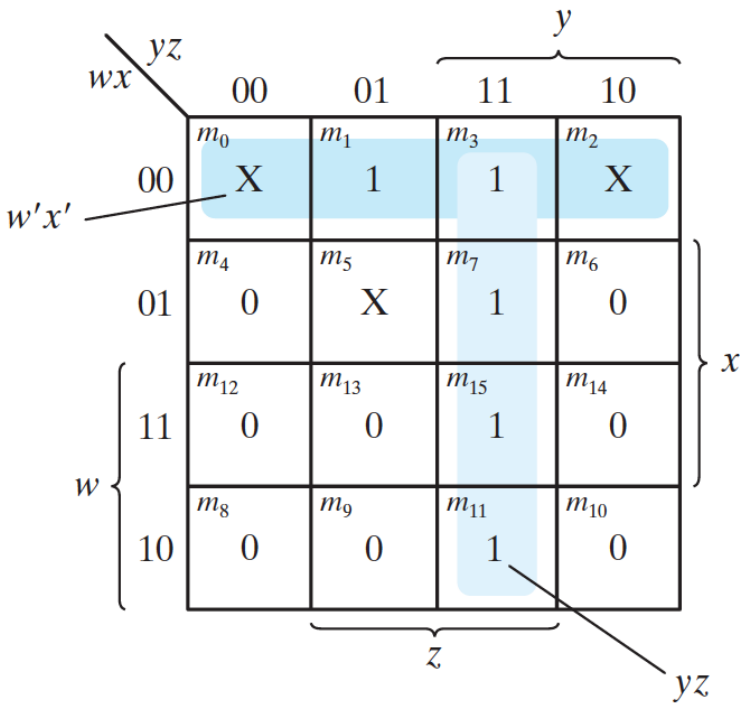
Generally, these minterms are marked by ‘X’ in the K-map.

In choosing adjacent

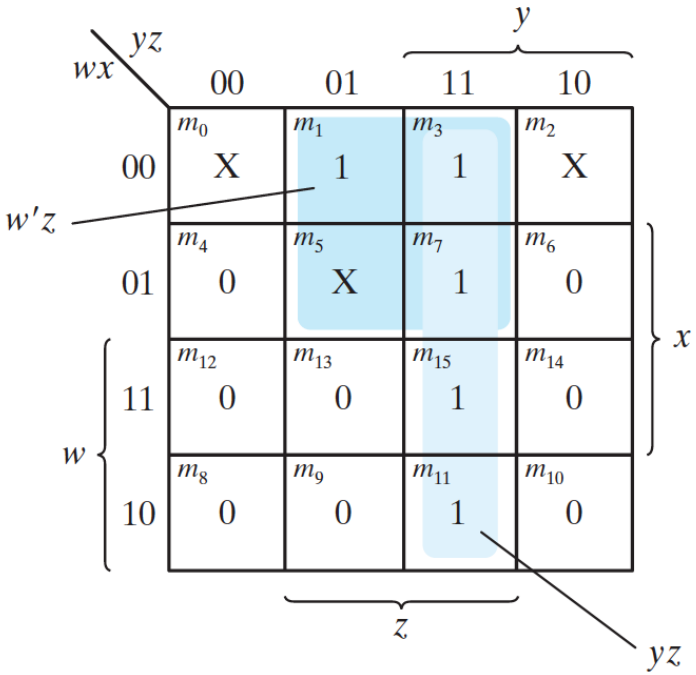
Example:

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$



$$F = yz + w'x'$$



$$F = yz + w'z$$

$$F(w, x, y, z) = yz + w'x' = \Sigma(0, 1, 2, 3, 7, 11, 15)$$

$$F(w, x, y, z) = yz + w'z = \Sigma(1, 3, 5, 7, 11, 15)$$

Don’t care conditions

Sometimes, functions have unspecified outputs for some input combinations, such functions are called **incompletely specified functions**.

In such case, we generally **don’t care** what value is assumed by the function for the unspecified minterms.

They are called as **don’t care conditions**.

These **don’t-care conditions** can be used on a map to provide further simplification of the Boolean expression.

Generally these minterms are marked by ‘X’ in the K-map.

In choosing adjacent

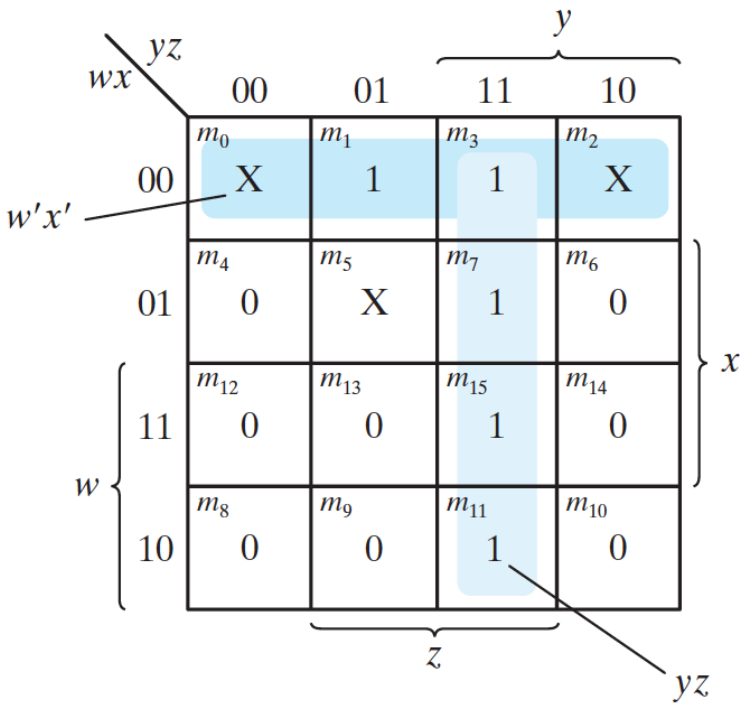
Decimal	Excess-3
–3	0000
–2	0001
–1	0010
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100
10	1101
11	1110
12	1111

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

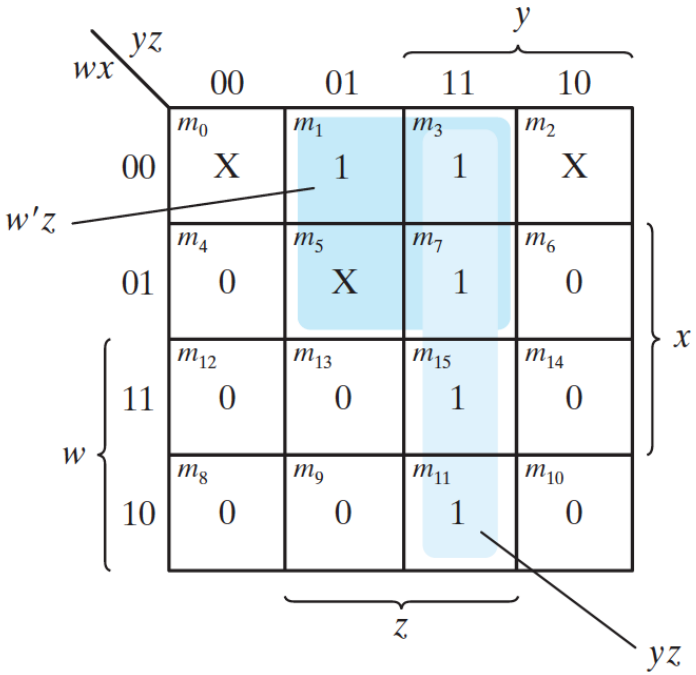
Example:

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$



$$F = yz + w'x'$$



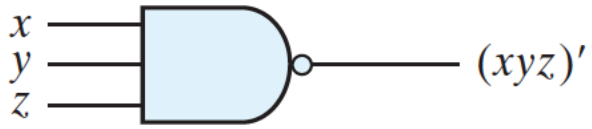
$$F = yz + w'z$$

$$F(w, x, y, z) = yz + w'x' = \Sigma(0, 1, 2, 3, 7, 11, 15)$$

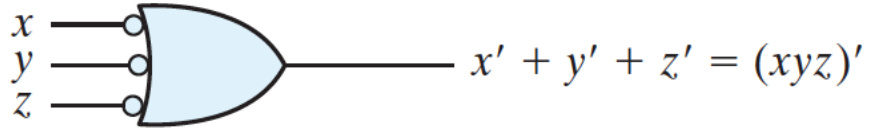
$$F(w, x, y, z) = yz + w'z = \Sigma(1, 3, 5, 7, 11, 15)$$

NAND Implementation

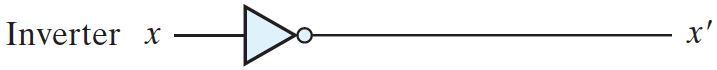
NAND and NOR gates are universal and relatively easier to implement

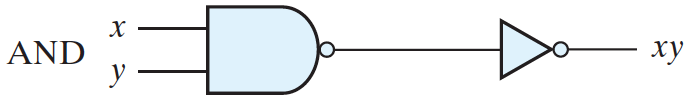


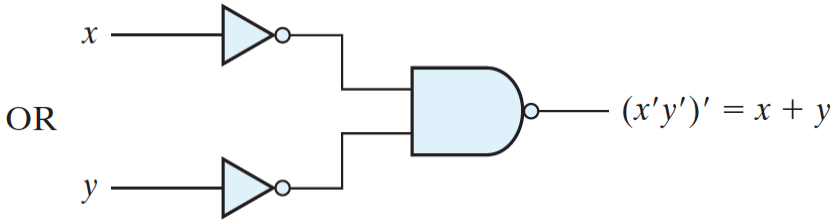
AND-invert



Invert-OR



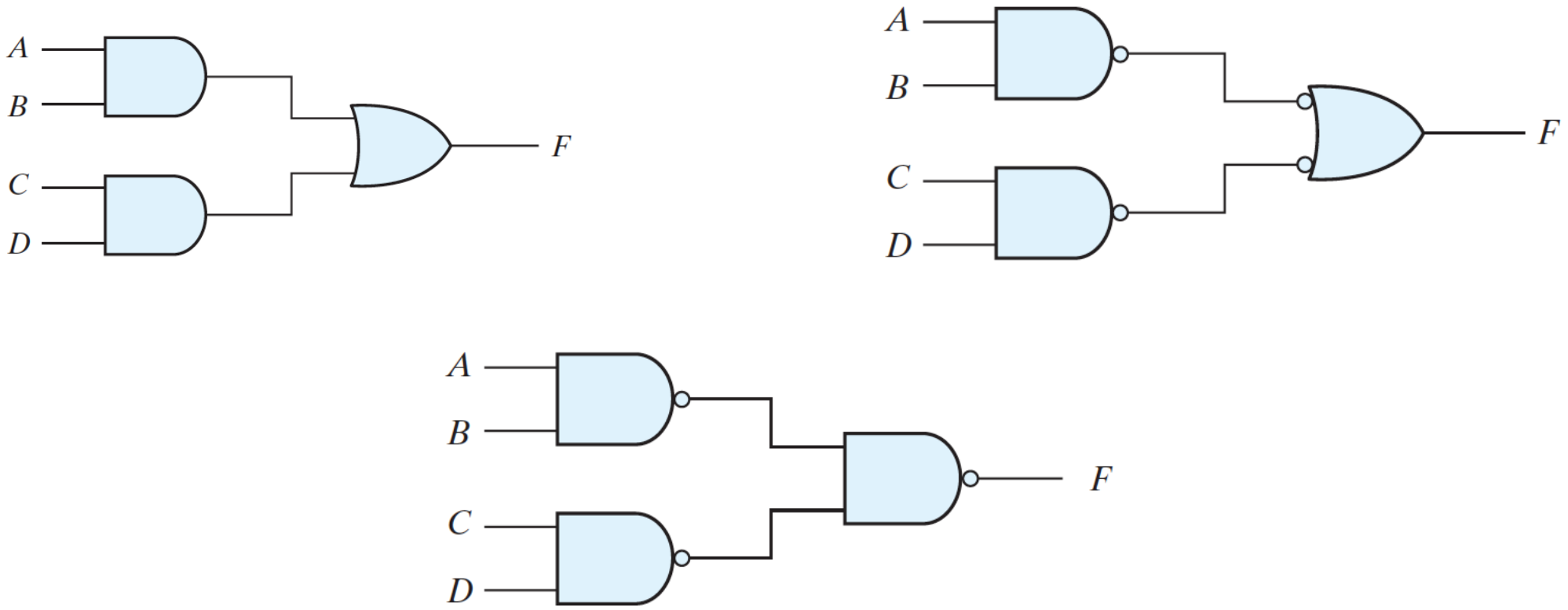




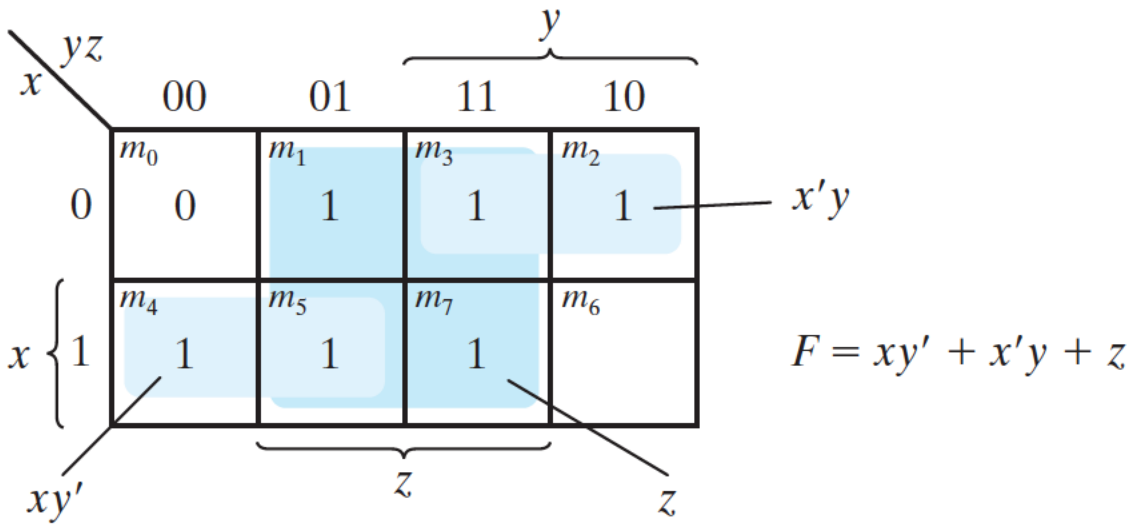
NAND Implementation: Two level implementation

The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form

$$F = AB + CD$$



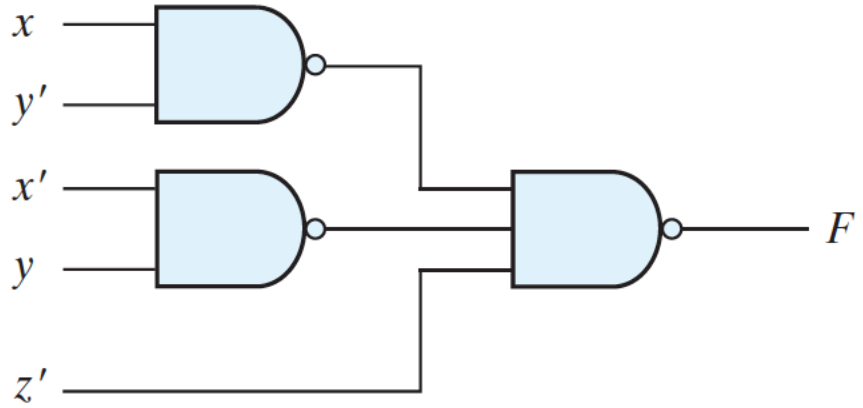
Example: Implement the following Boolean function with NAND gates:



$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$

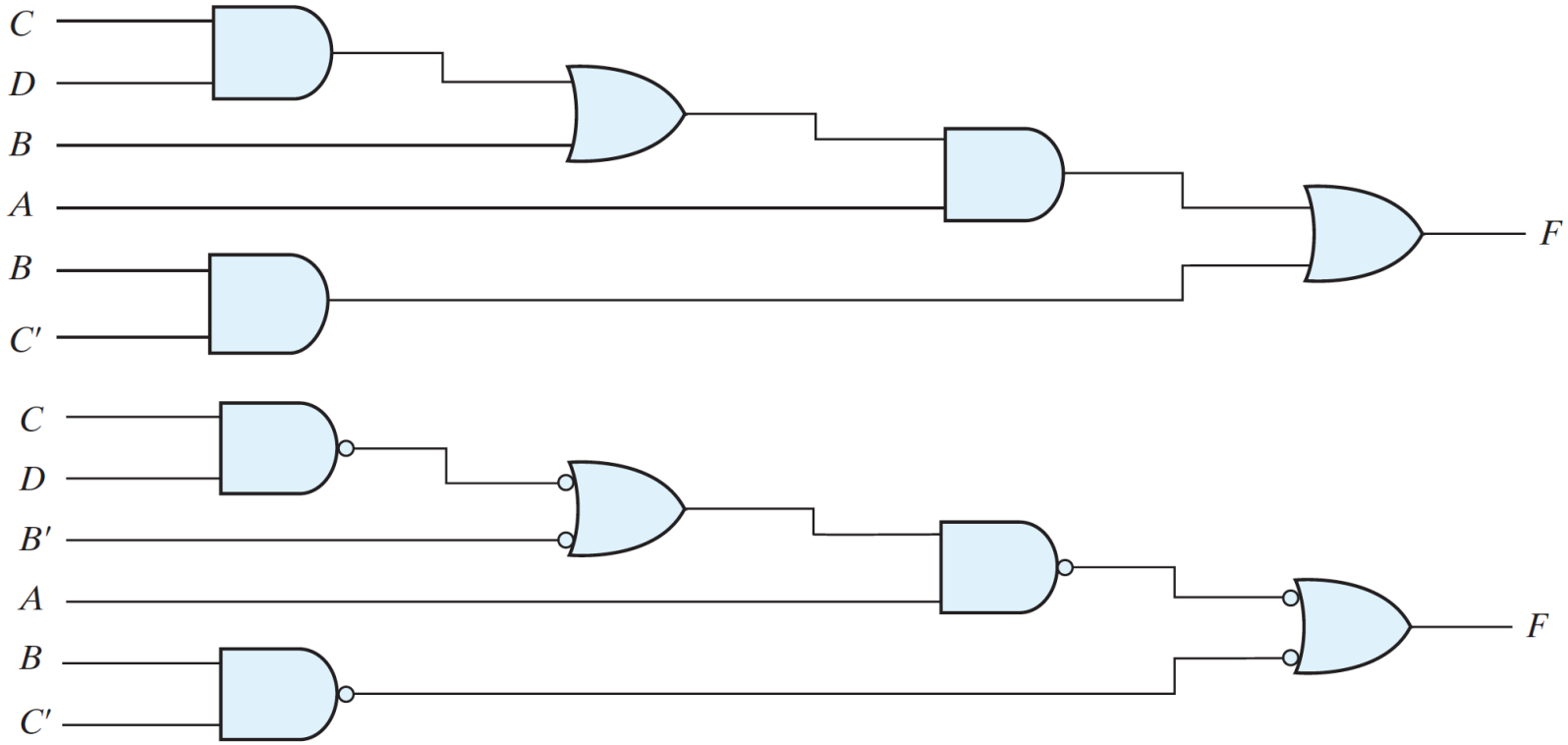
$$F = xy' + x'y + z$$

- **Simplify the function and express it in sum-of-products form.**
- **Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of first-level gates.**
- **Draw a single gate using the AND-invert or the invert-OR graphic symbol in the second level, with inputs coming from outputs of first-level gates.**
- **A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate.**



Multilevel NAND Implementation

$$F = A\,(CD + B) + BC'$$



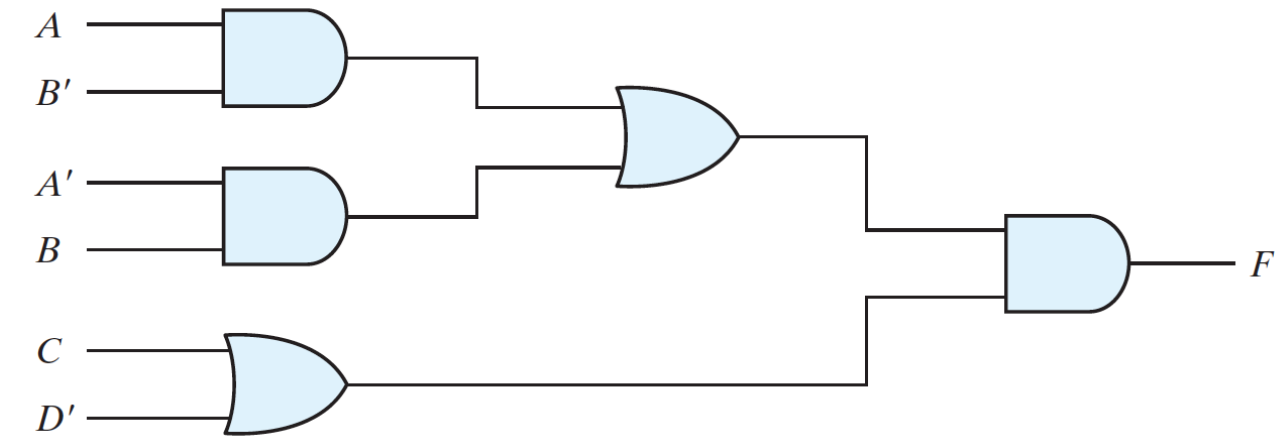
Convert all AND gates to NAND gates with AND-invert graphic symbols.

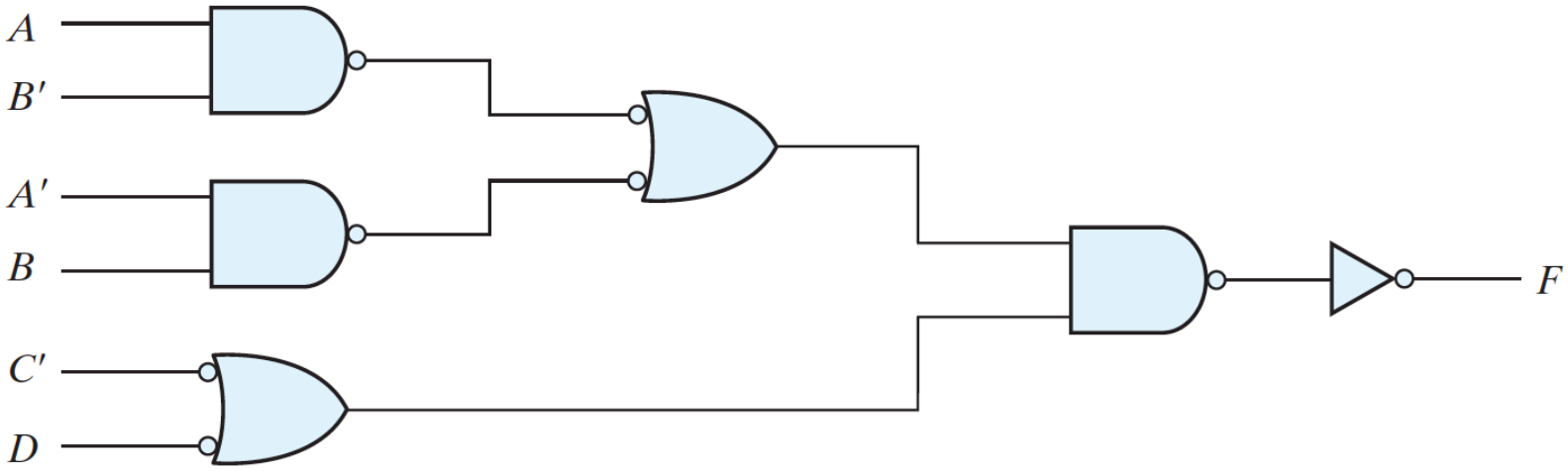
Convert all OR gates to NAND gates with invert-OR graphic symbols.

Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

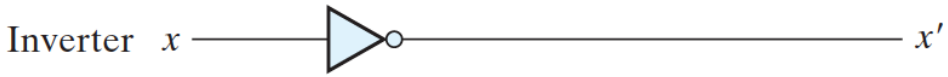
Multilevel NAND Implementation

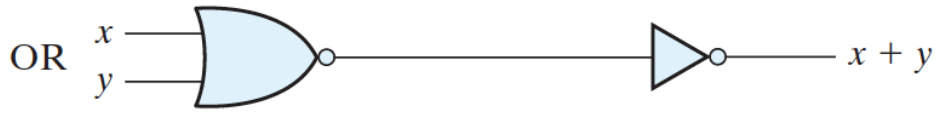
$$F = (AB' + A'B)(C + D')$$

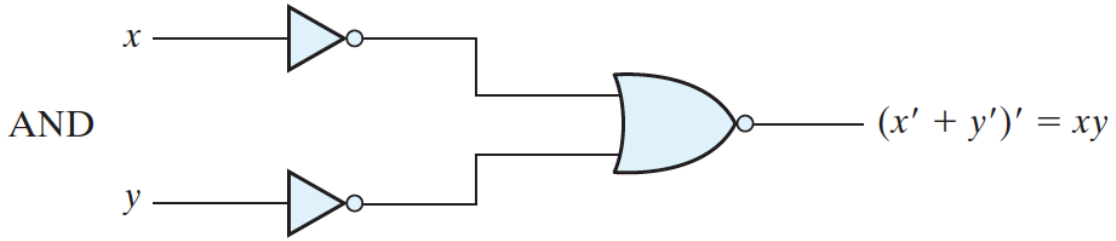


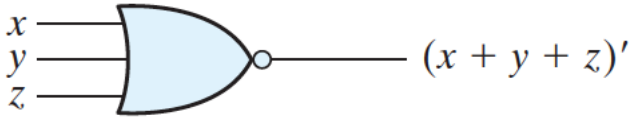


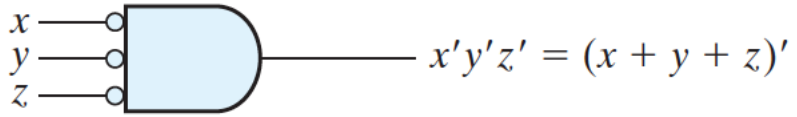
NOR Implementation





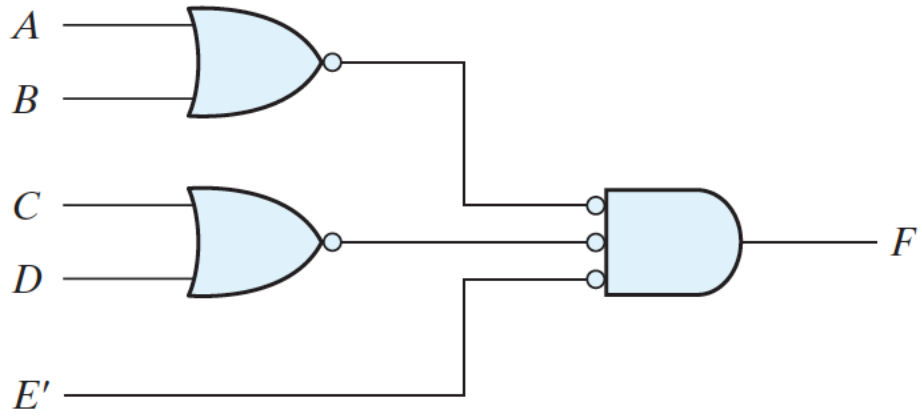






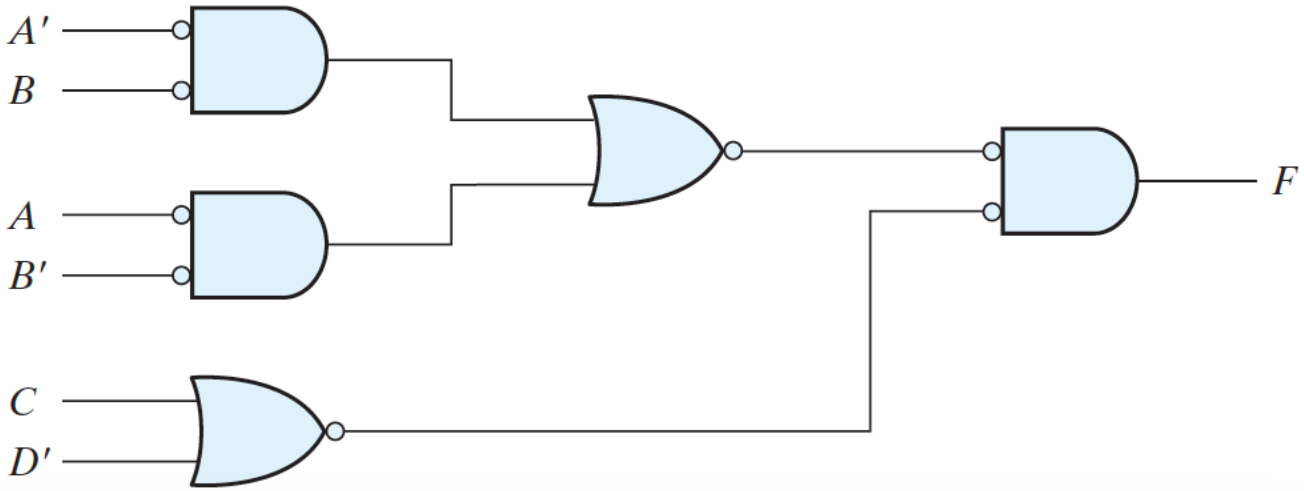
Example:

$$F = (A + B)(C + D)E$$



NOR Implementation

$$F = (AB' + A'B)(C + D')$$



XOR Function

$$x \oplus y = xy' + x'y$$

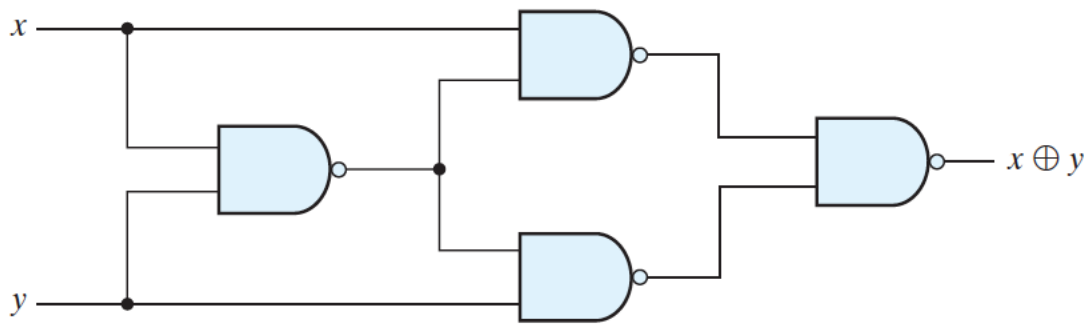
$$(x \oplus y)' = xy + x'y'$$

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

XOR is an Odd Function

i.e. an odd number of variables be equal to 1 to make function equal to 1.



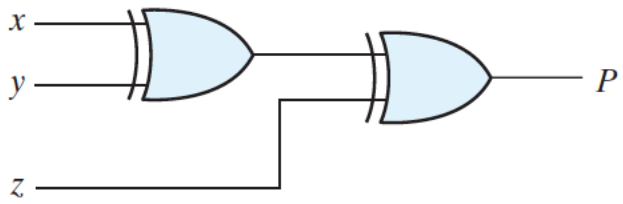
(b) Exclusive-OR with NAND gates

Parity Generation and Checking

Exclusive-OR functions are very useful in systems requiring error detection and correction codes

Even-Parity-Generator Truth Table

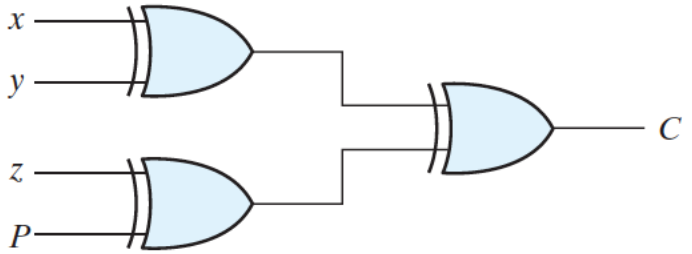
Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



(a) 3-bit even parity generator

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



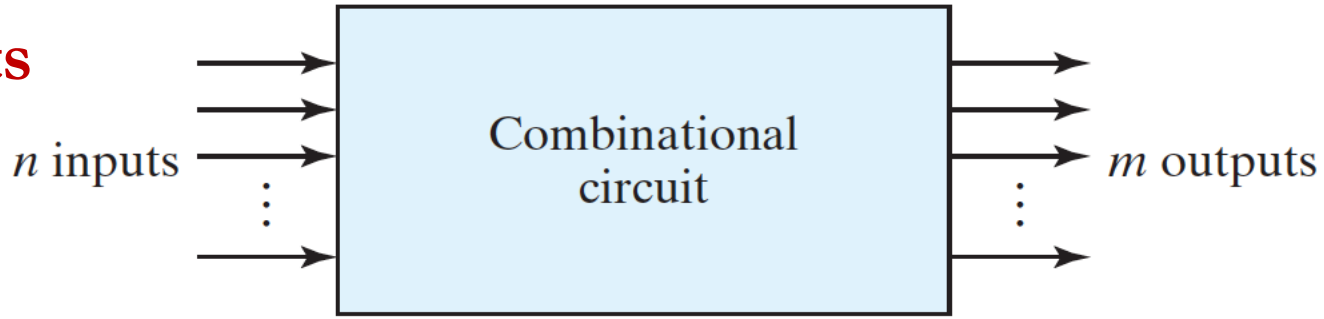
(b) 4-bit even parity checker

Combinational and Sequential Circuits

- **Combinational circuit:** Logic gates whose outputs at any time are determined from only the present combination of inputs
  - A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.

- **Sequential circuit:** Contain storage elements in addition to logic gates.
  - Their outputs are a function of the inputs and the state of the storage elements.
  - The state of the storage elements is a function of previous inputs
  - The outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs

Combinational Circuits



- A combinational circuit can be described by **m** Boolean functions
  - One for each output variable.
- Each output function is expressed in terms of the **n** input variables.
- **The aim here is:**
  - Analyze the behavior of a given logic circuit
  - Synthesize a circuit that will have a given behavior

- There are several combinational circuits that are employed extensively in the design of digital systems. These circuits are available in integrated circuits and are classified as standard components.

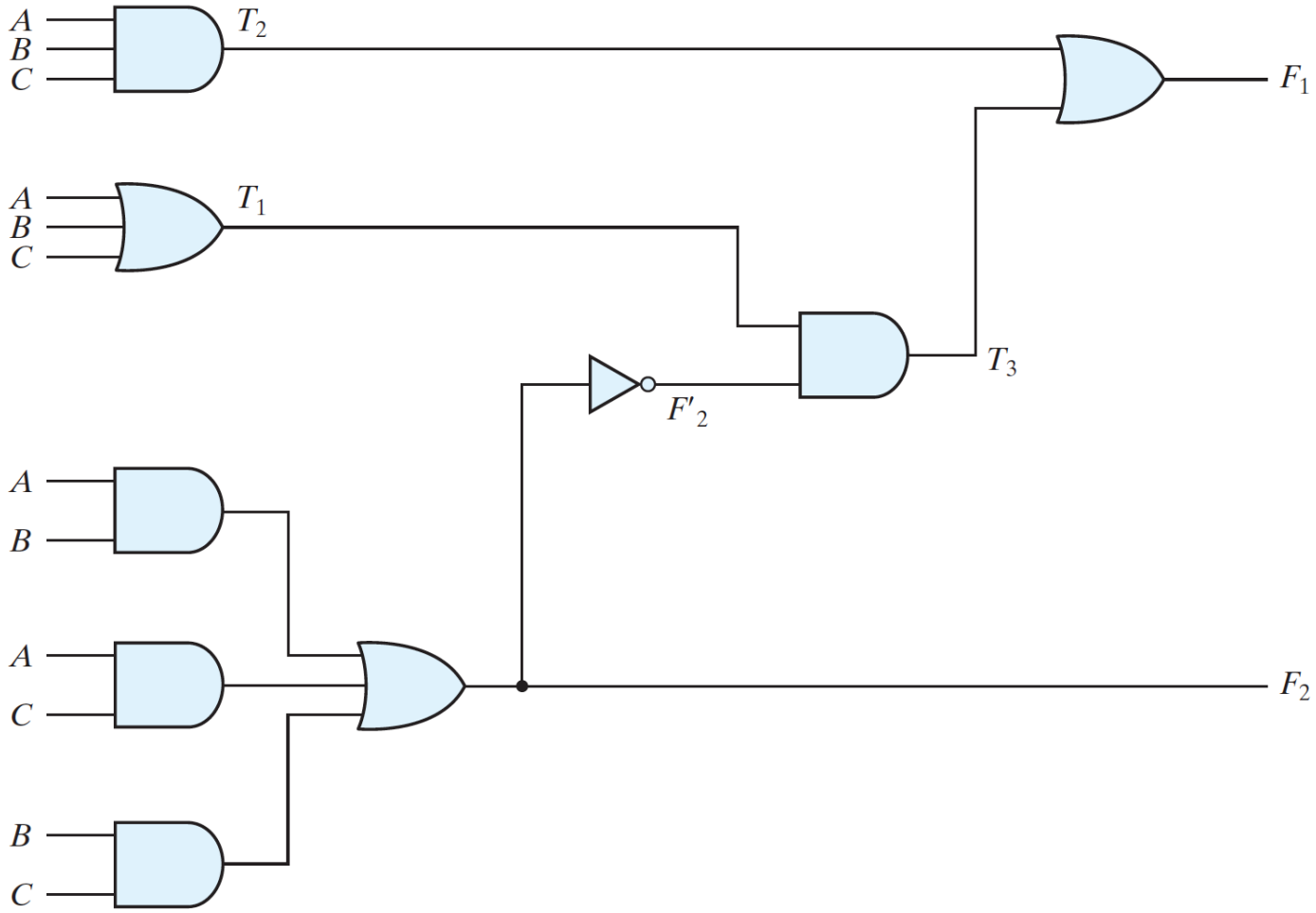
Combinational Circuits

- **Examples are:** Adders, subtractors, comparators, decoders, encoders, and multiplexers.
- These components are available in integrated circuits as medium-scale integration (MSI) circuits.
- **The analysis of a combinational circuit requires that we determine the function that the circuit implements.**
- This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or, possibly, an explanation of the circuit operation.

- The diagram of a combinational circuit has logic gates with no feedback paths or memory elements .

- A feedback path is a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate.
  - Feedback paths in a digital circuit define a sequential circuit and must be analyzed by special methods

Analysis



$$
\begin{aligned}
F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\
&= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\
&= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\
&= A'BC' + A'B'C + AB'C' + ABC
\end{aligned}
$$

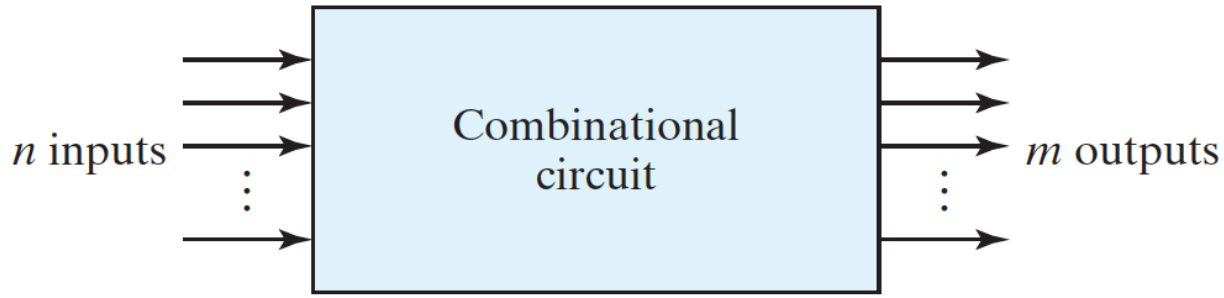
Example:

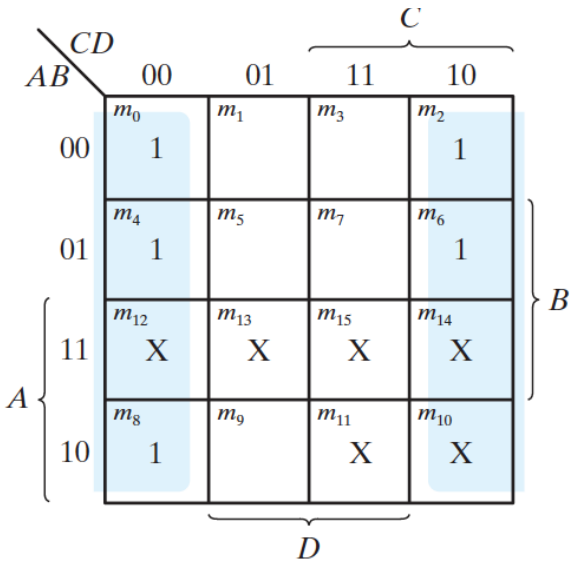
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

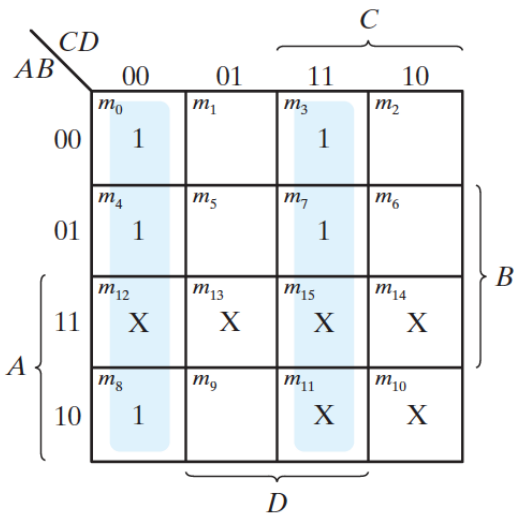
How to create this circuit?

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

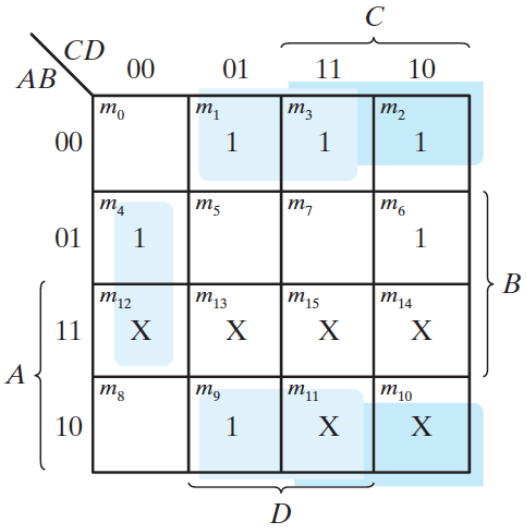




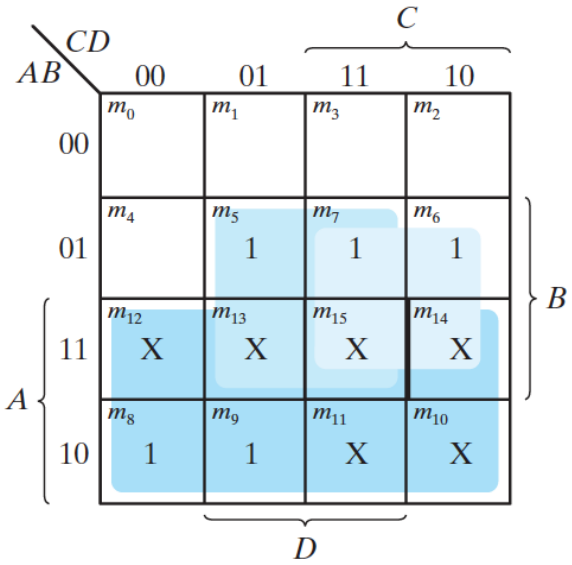
$z = D'$



$y = CD + C'D'$



$x = B'C + B'D + BC'D'$

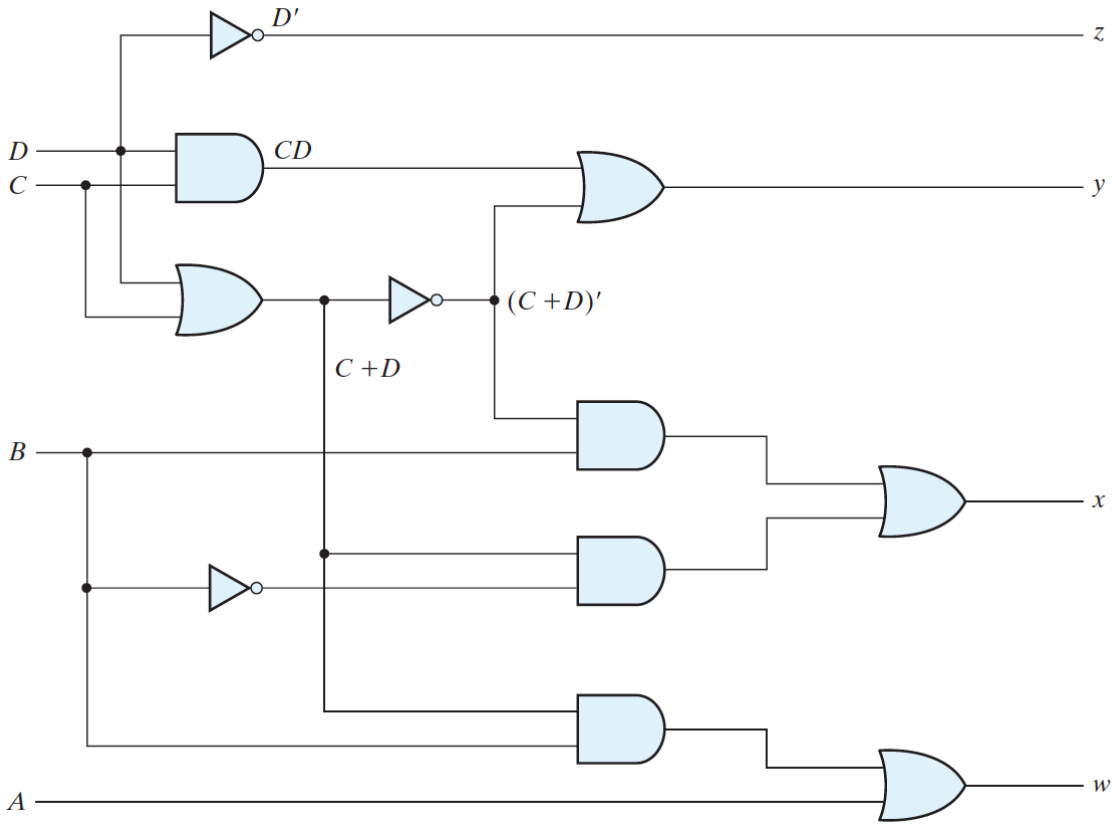


$w = A + BC + BD$

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

Circuit Implementation



Binary Adder and Subtractor

Half Adder

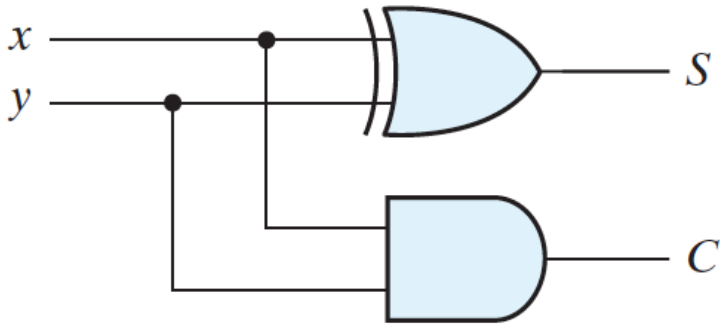
Needs two binary inputs and two binary outputs

A combinational circuit that performs the addition of two bits is called a half adder .

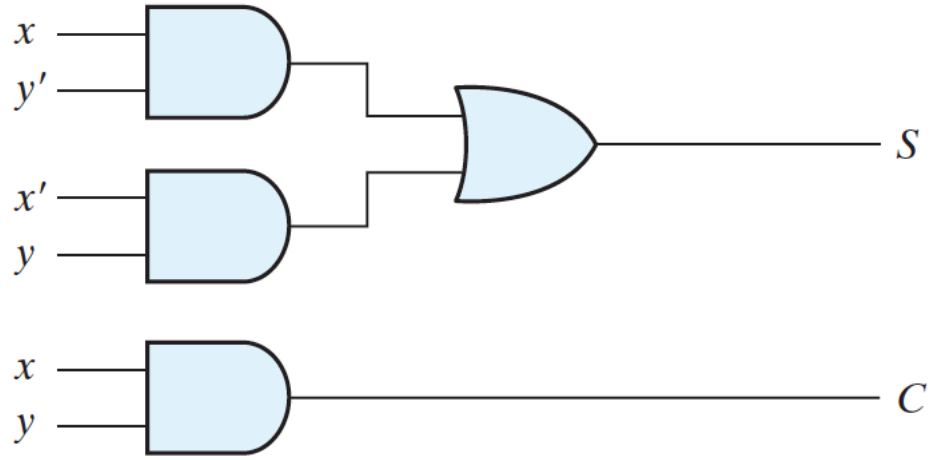
One that performs the addition of three bits (two significant bits and a previous carry) is a full adder .

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy'$$
$$C = xy$$



(b) $S = x \oplus y$
$C = xy$



(a) $S = xy' + x'y$
$C = xy$

Thank you