Fundamentals of Electronics

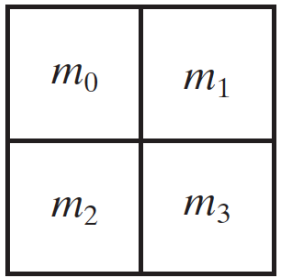
ECE 101

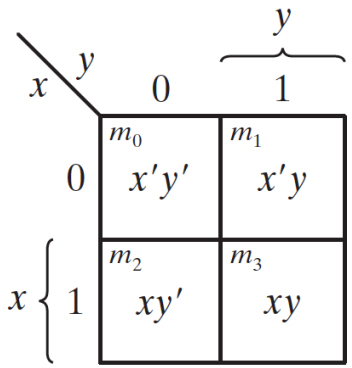
Gate Level Minimization

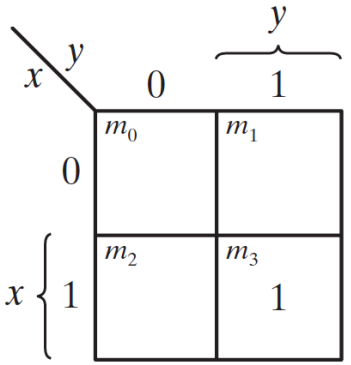
Finding an **optimal gate-level implementation** of the Boolean functions **describing a digital circuit**.

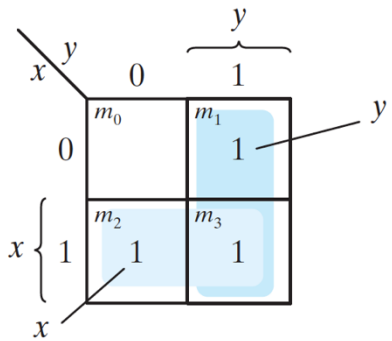
Karnaugh map or K-map

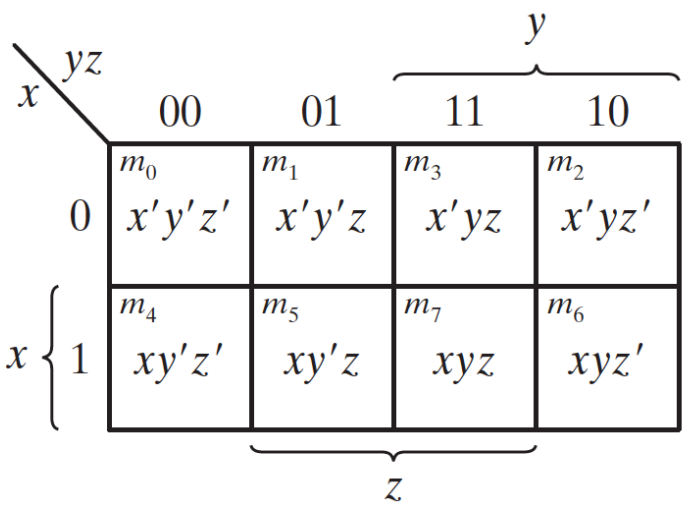
Pictorial form of the Truth Table



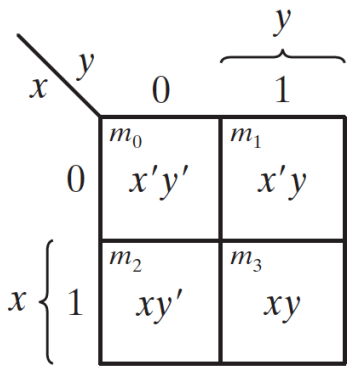


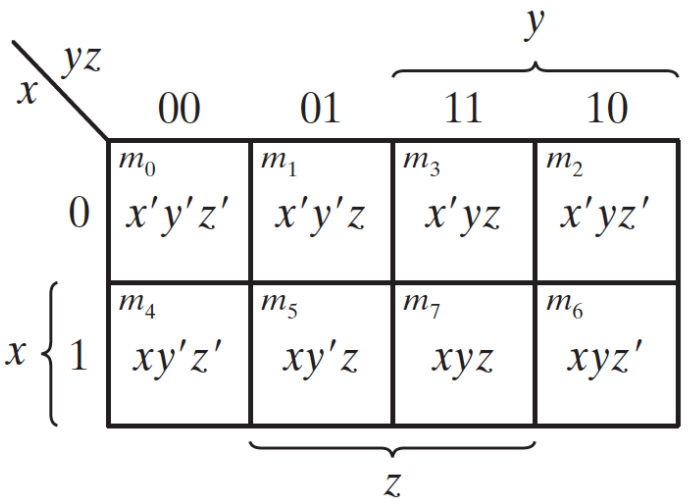






K-map





Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other – which of the terms are adjacent here?

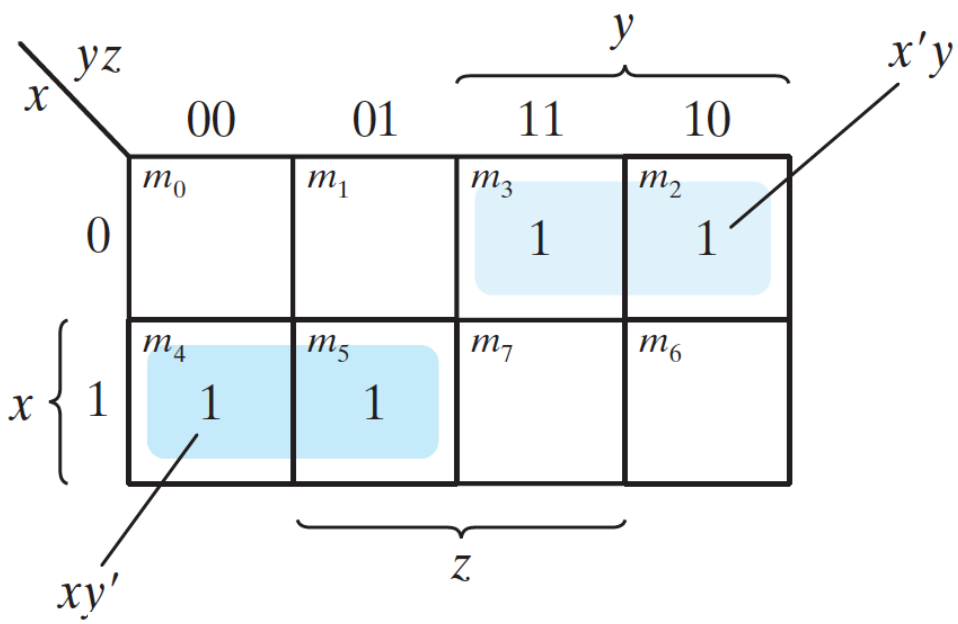
$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Example:

Simplify this: $F(x, y, z) = \Sigma(2, 3, 4, 5)$

Example:

$$F(x, y, z) = \Sigma(2, 3, 4, 5)$$



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

One square represents one minterm, giving a term with three literals.

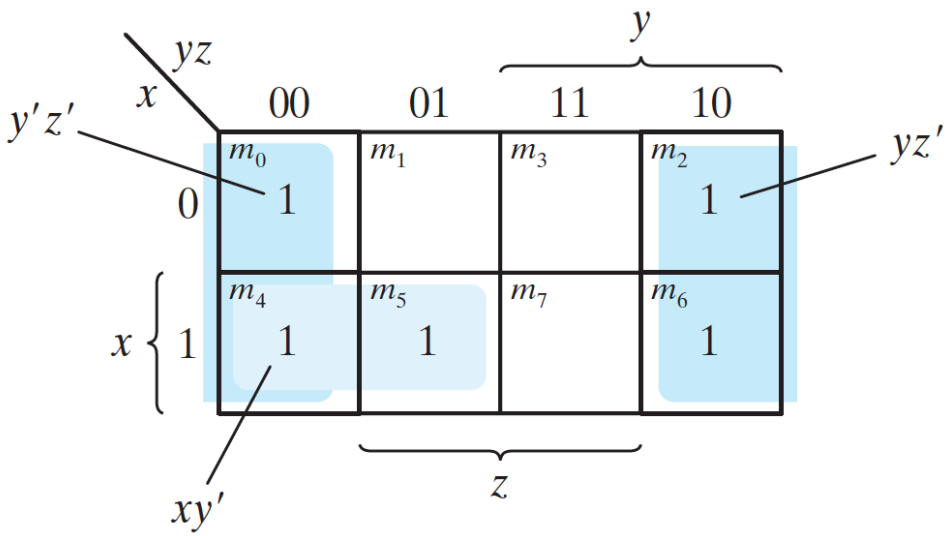
Two adjacent squares represent a term with two literals.

Four adjacent squares represent a term with one literal.

Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

Example: Simplify the following Boolean function

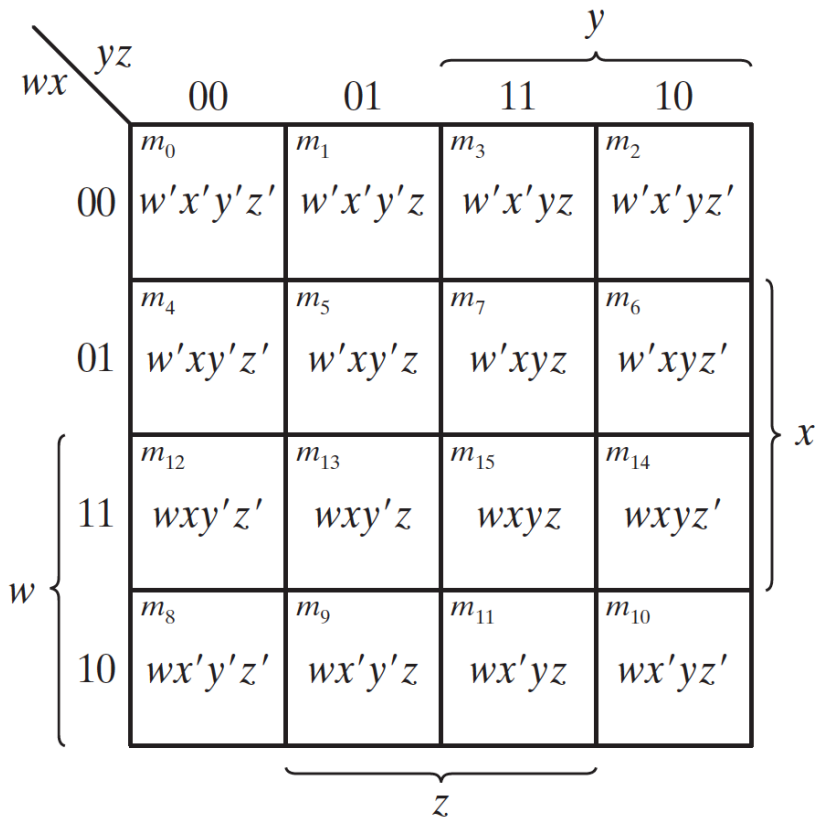
$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$



$$F = z' + xy'$$

4 variable K-map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



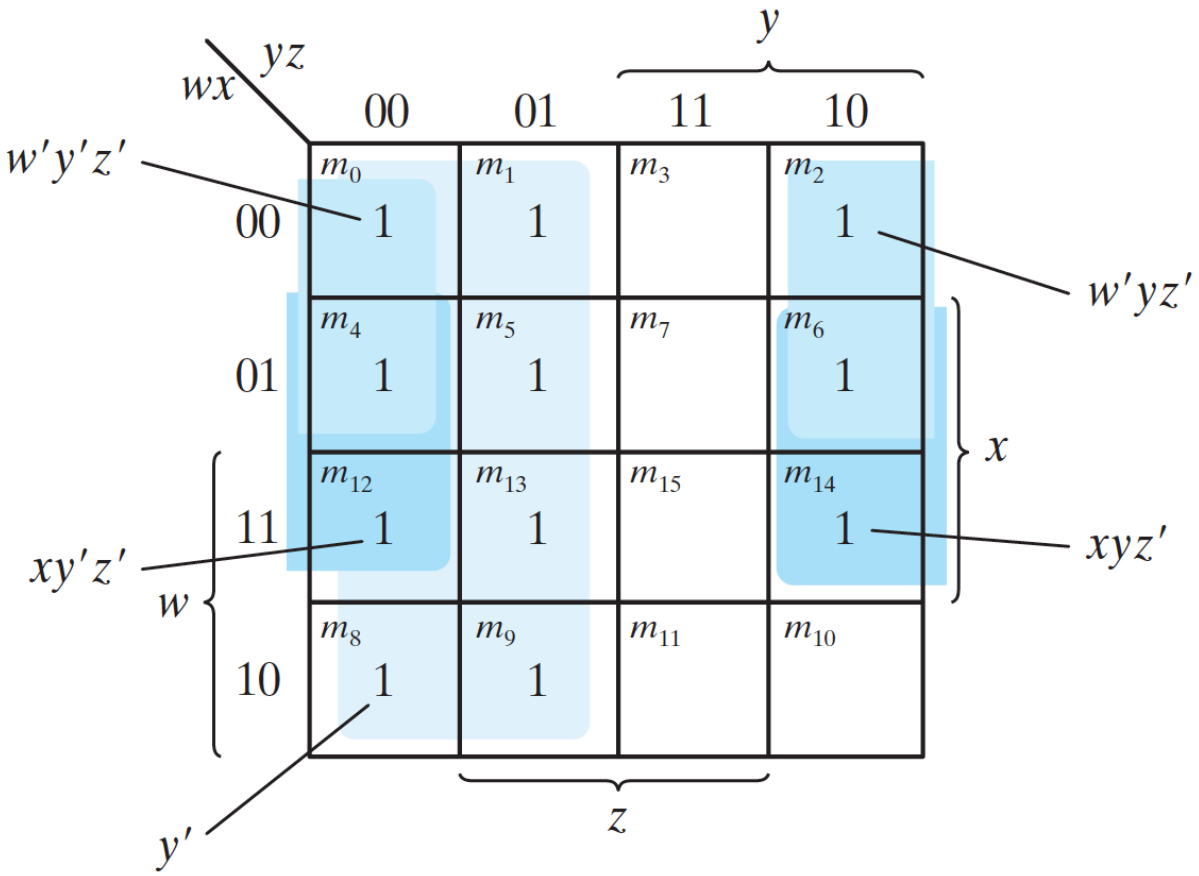
One square represents one minterm, giving a term with four literals.
Two adjacent squares represent a term with three literals.
Four adjacent squares represent a term with two literals.
Eight adjacent squares represent a term with one literal.
Sixteen adjacent squares produce a function that is always equal to 1.

Example: Simplify the following Boolean function

$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Example: Simplify the following Boolean function

$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

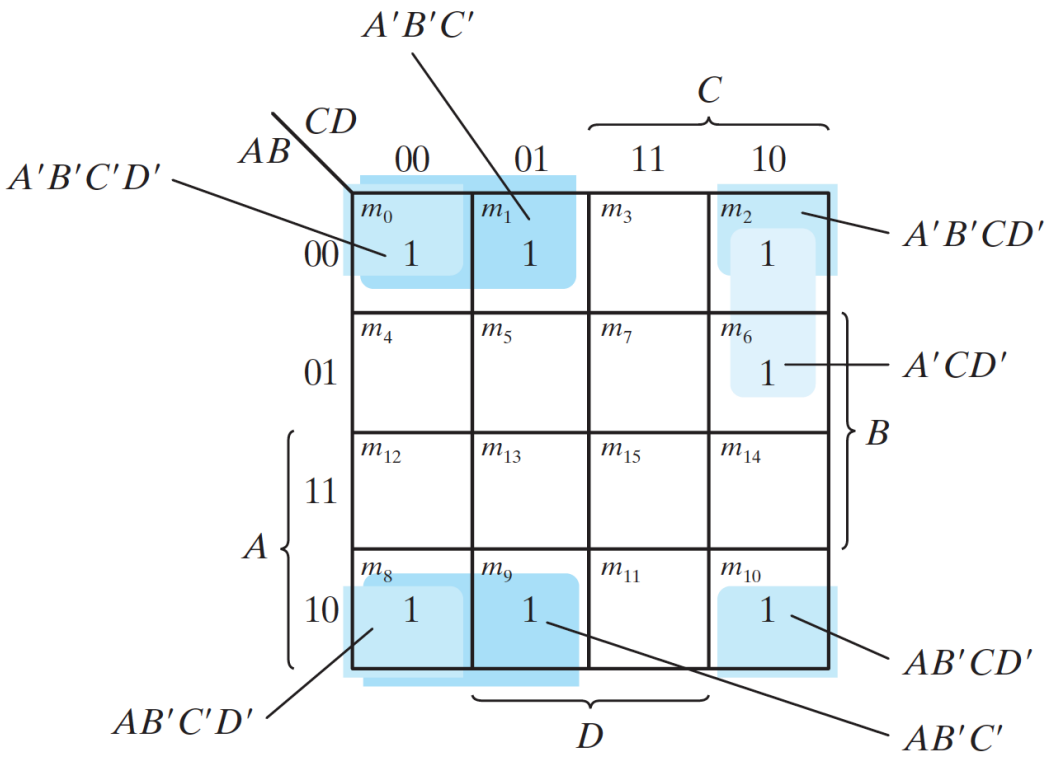


$$w'y'z' + w'yz' = w'z'$$
$$xy'z' + xyz' = xz'$$

$$F = y' + w'z' + xz'$$

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.**

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

Prime Implicant

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

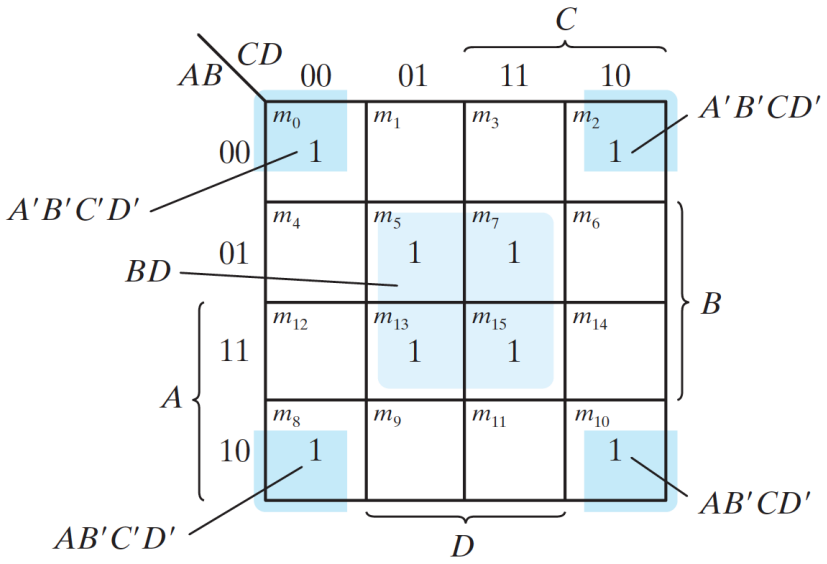
- A single 1 on a map represents a prime implicant if it is not adjacent to any other 1's.
- Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares.
- Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent squares,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

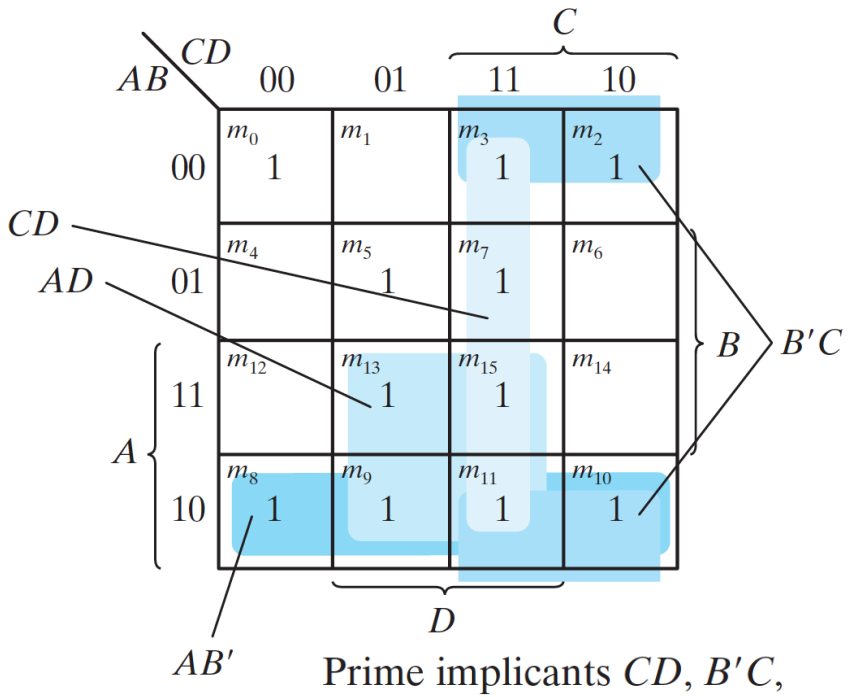
The prime implicant is essential if it is the only prime implicant that covers the minterm

- 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.

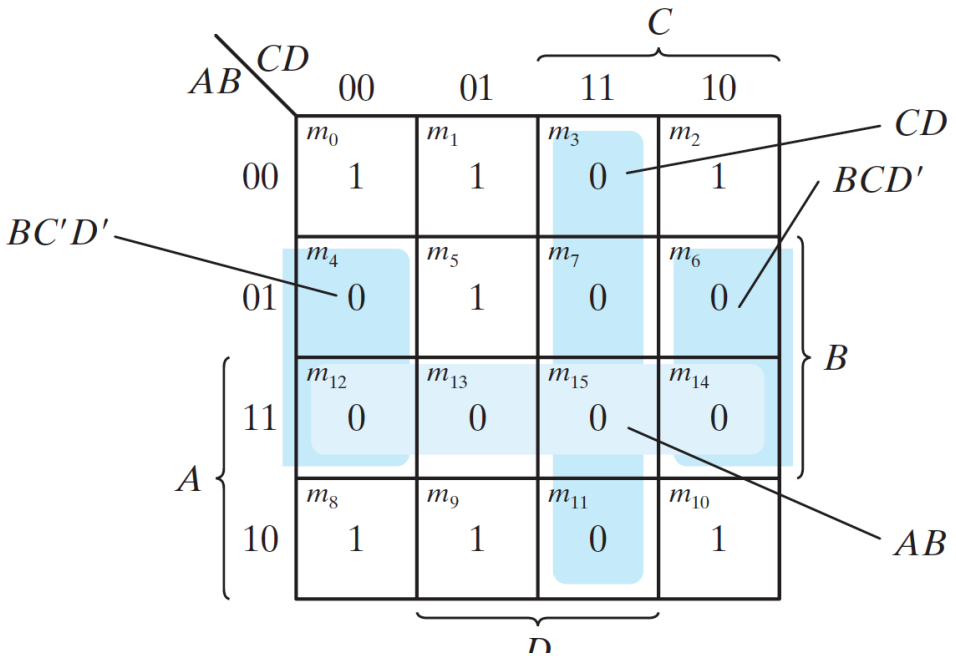
PRODUCT-OF-SUMS SIMPLIFICATION

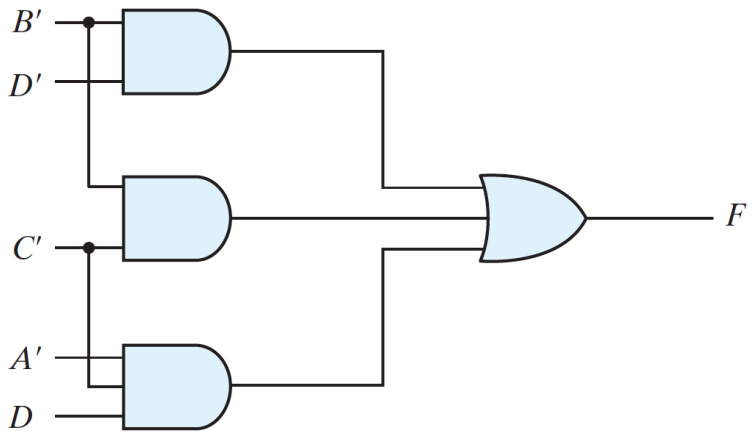
$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

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

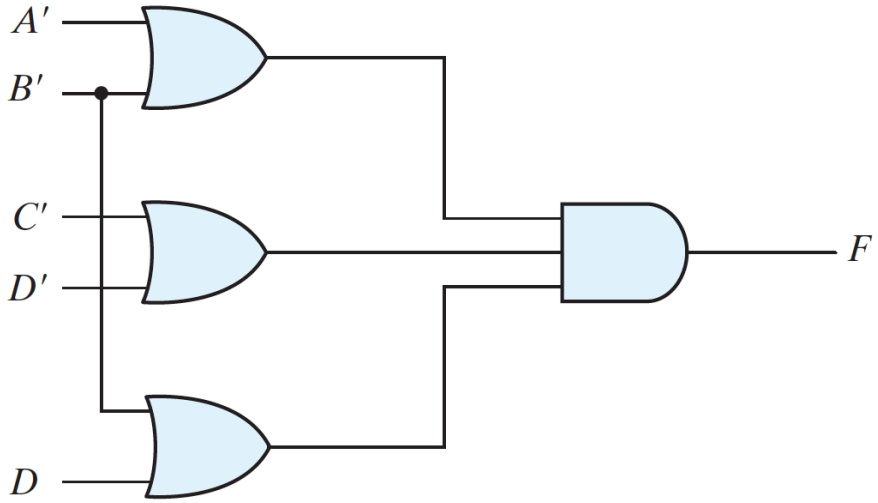
$$F' = AB + CD + BD'$$

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





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



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

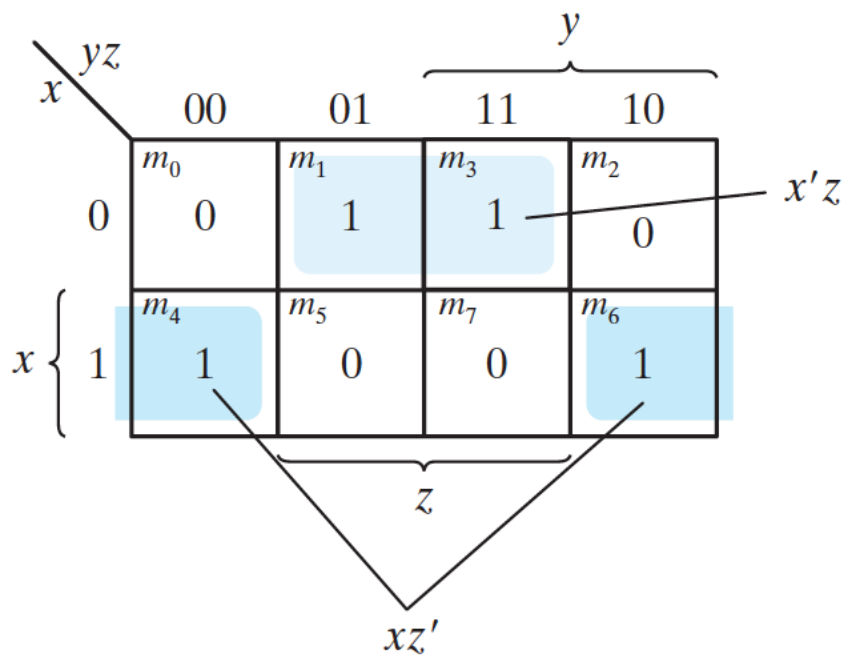
K-map for PRODUCT-OF-SUMS representation

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

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

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

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



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

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

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.

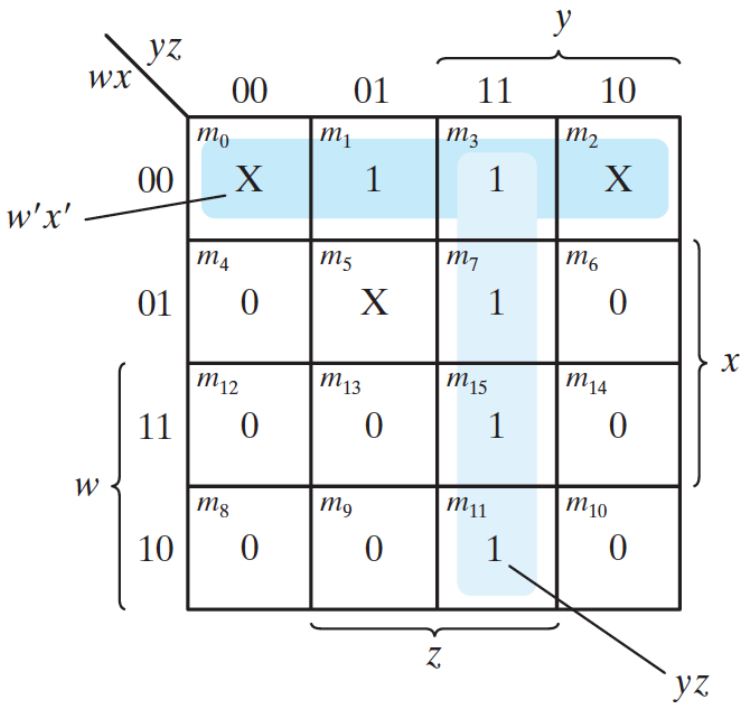
Generally, these minterms are marked by ‘X’ in the K-map.

In choosing adjacent

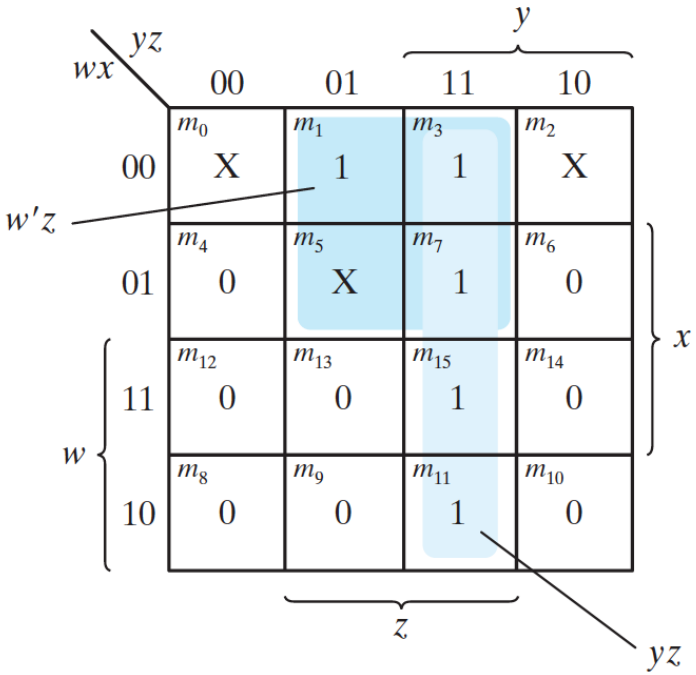
Example:

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

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

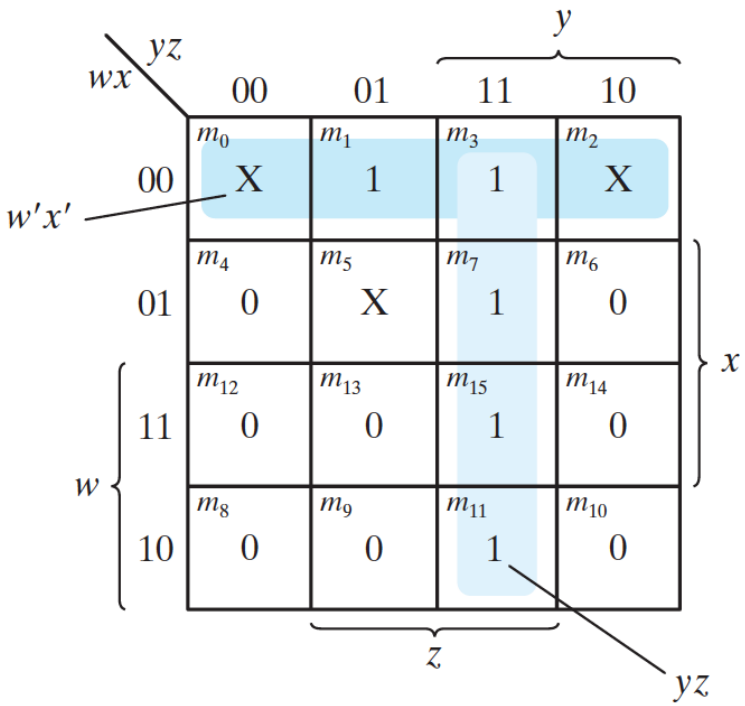
Decimal	Excess-3
–3	0000
–2	0001
–1	0010
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100
10	1101
11	1110
12	1111

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

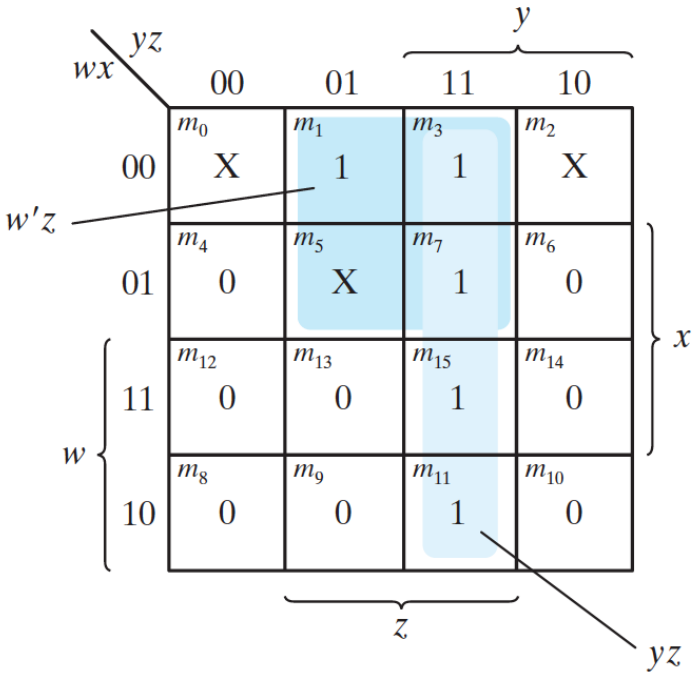
Example:

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

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



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



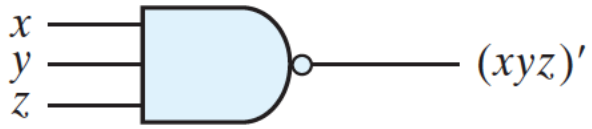
$$F = yz + w'z$$

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

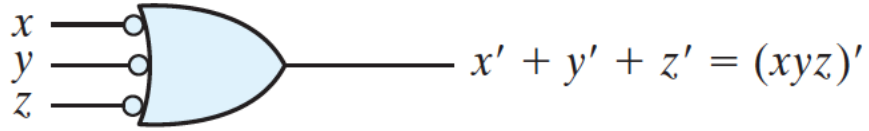
$$F(w, x, y, z) = yz + w'z = \Sigma(1, 3, 5, 7, 11, 15)$$

NAND Implementation

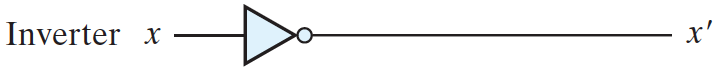
NAND and NOR gates are universal and relatively easier to implement

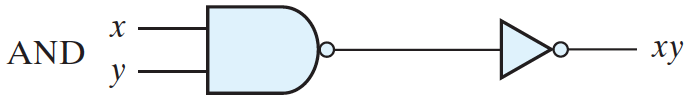


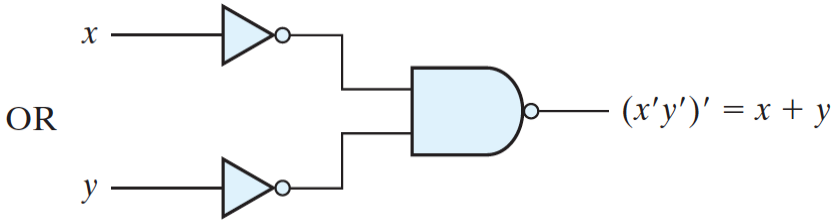
AND-invert



Invert-OR



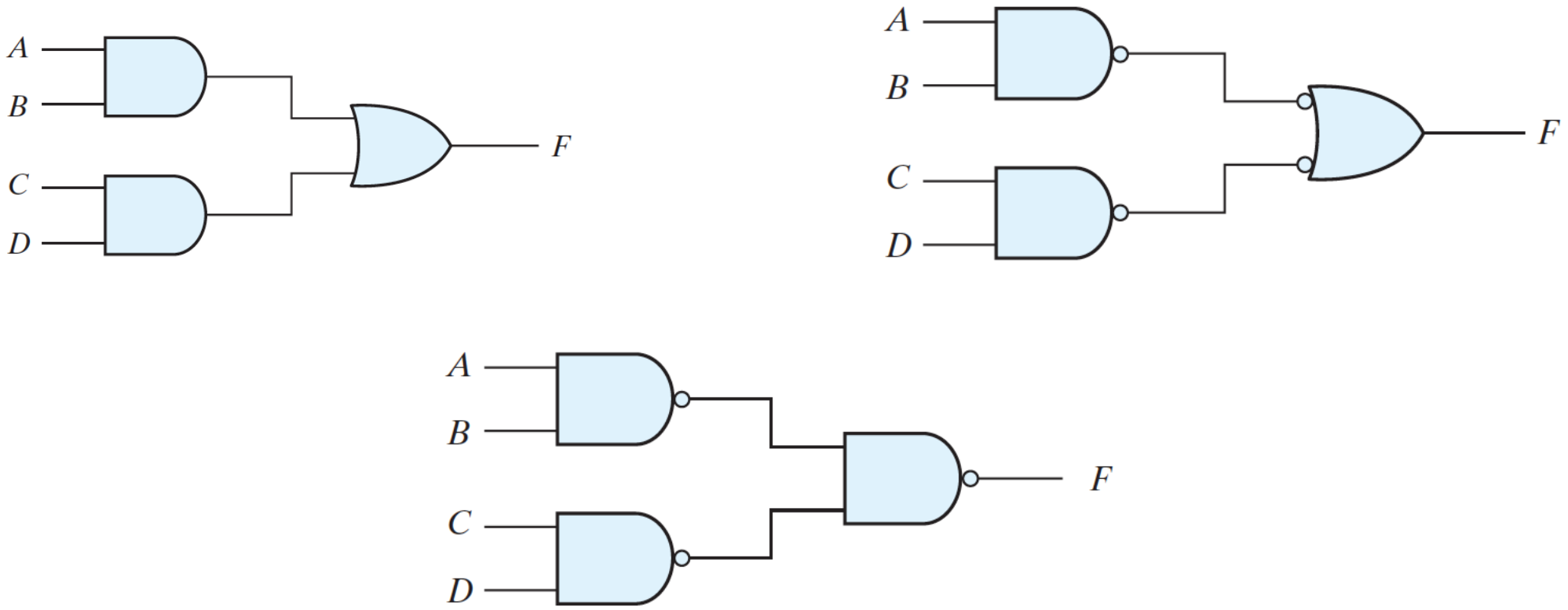




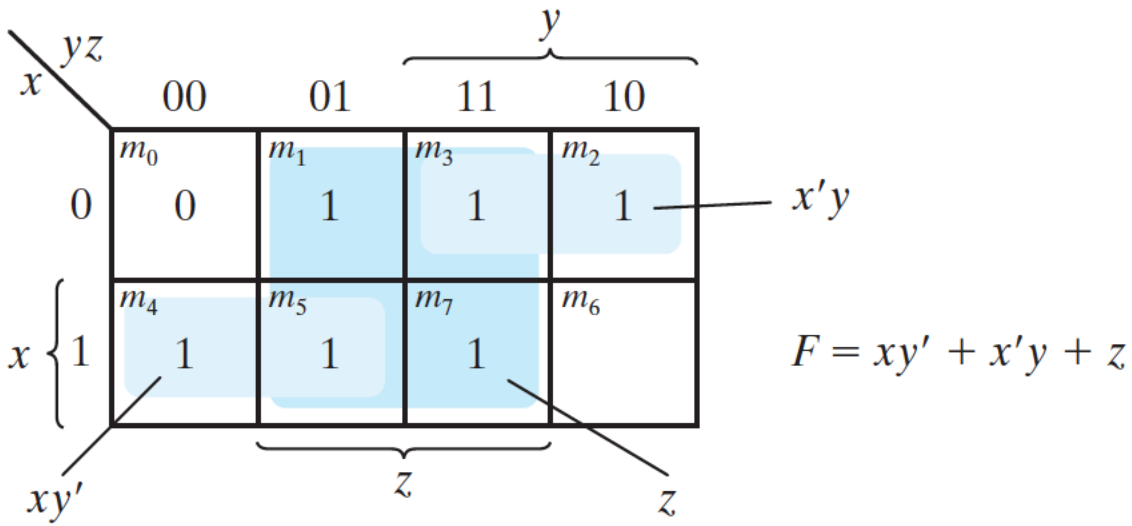
NAND Implementation: Two level implementation

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

$$F = AB + CD$$



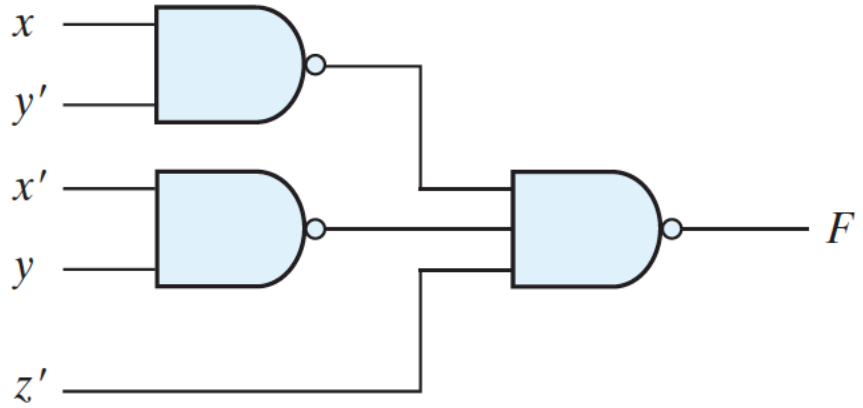
Example: Implement the following Boolean function with NAND gates:



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

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

- **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.**



Thank you