Ollscoil na Gaillimhe
University of Galway

CT101 Computing Systems

Dr. Bharathi Raja Chakravarthi
Lecturer-above-the-bar
Email: bharathi.raja@universityofgalway.ie

University ofGalway.ie

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Recap

Gate-Level Minimization

- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.
- This task is well understood but is difficult to execute by manual methods when the logic has more than a few inputs.
- Computer-based logic synthesis tools can minimize a large set of Boolean equations efficiently and quickly.

K-map

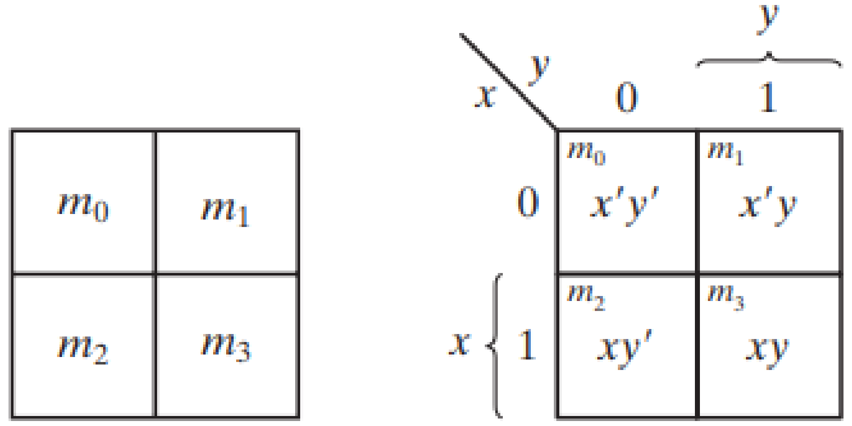
- A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.

- The map presents a visual diagram of all possible ways a function may be expressed in standard form.

- The simplified expressions produced by the map are always in one of the two standard forms:
    - sum of products or
    - products of sums

Two-Variable K-Map

- The two-variable map is shown here.
- Four minterms for two variables.



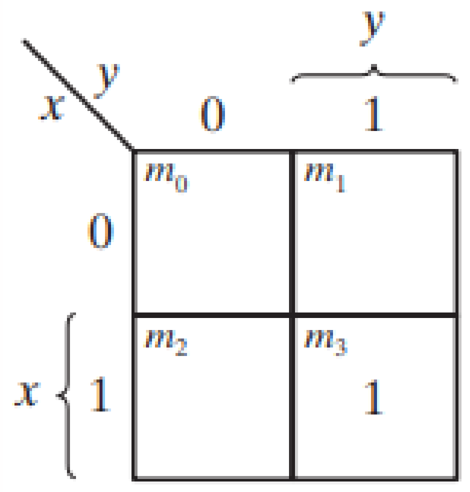
(a) (b)

- (b) shows the relationship between the squares and the two variables x and y .
- **0** and **1** designate the values of variables.
- Variable x appears primed in row 0 and unprimed in row 1.
- Similarly, y appears primed in column 0 and unprimed in column 1.

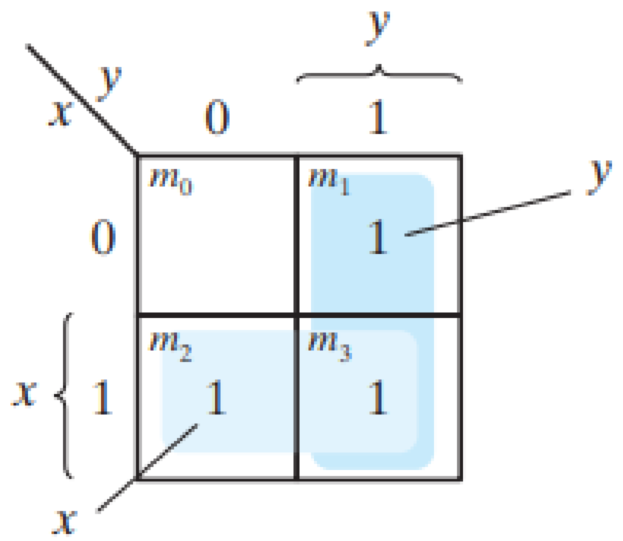
Two-Variable K-Map

- The two-variable map becomes another useful way to represent any one of the 16 Boolean functions of two variables.

- Example, the function xy is shown Fig. (a).



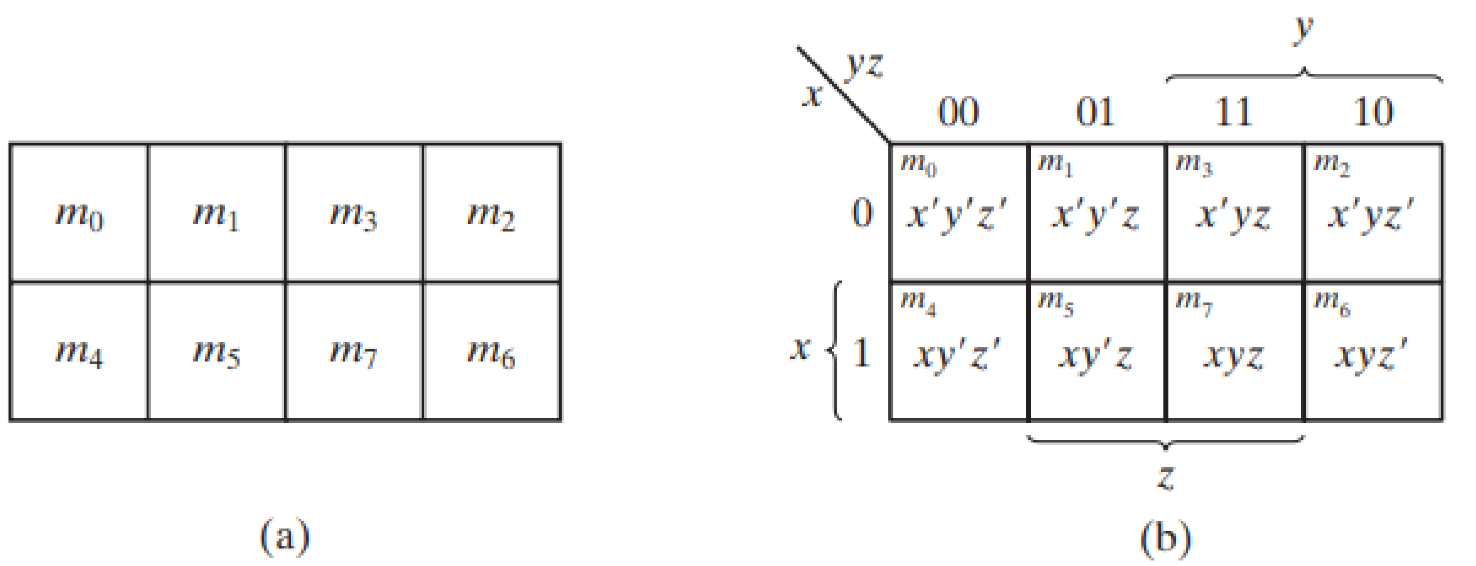
(a) xy



(b) $x + y$

Three-Variable K-Map

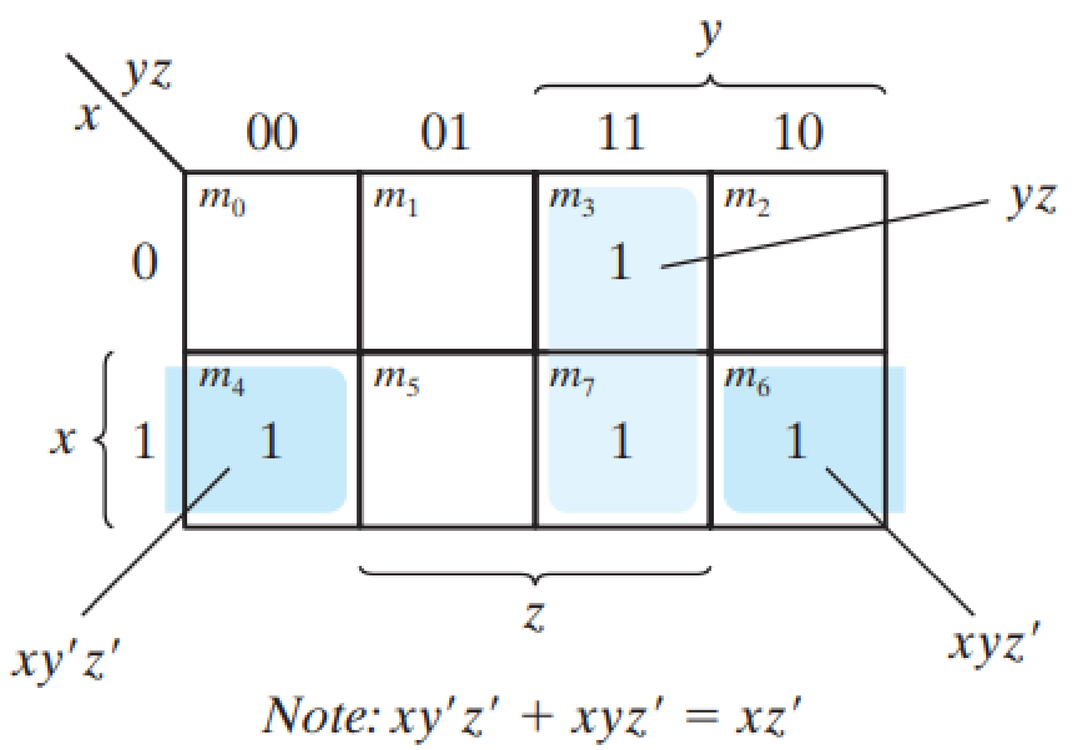
- A three-variable K-map is shown here.



- Eight minterms for three binary variables; the map consists of eight squares.

- Only one bit changes in value from one adjacent column to the next.

Examples



Simplify the Boolean function **F(x, y, z) = ∑(3, 4, 6, 7)**

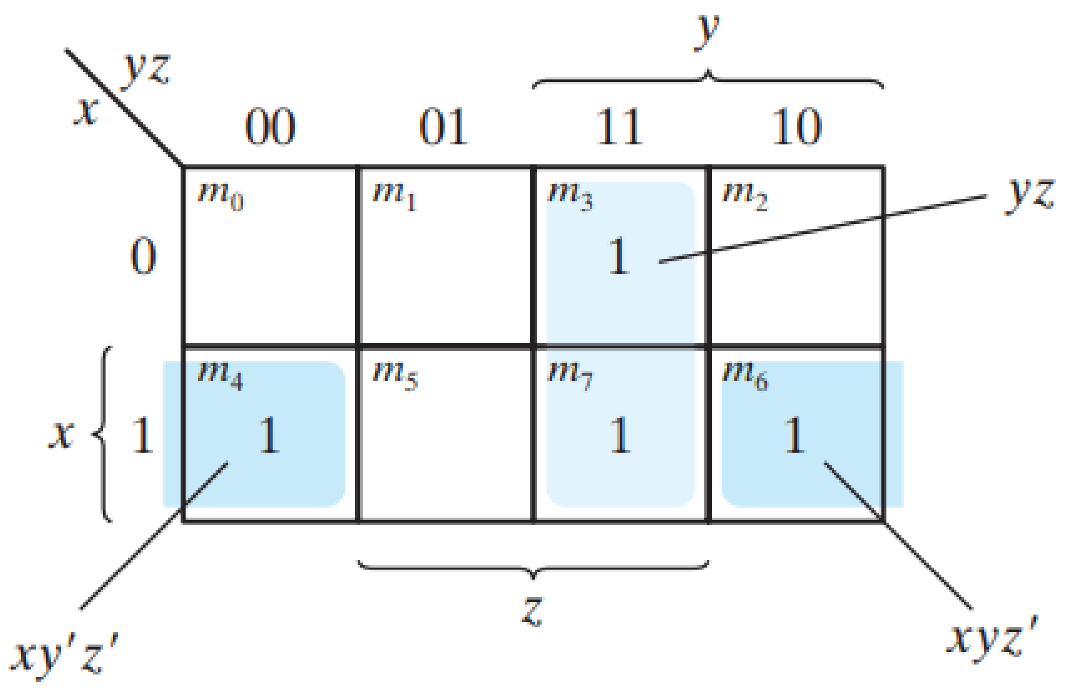
1. Mark squares with 1s in the Karnaugh Map (minterms 011, 100, 110, 111).
2. Two adjacent squares are combined in the third column to give a two-literal term **yz**.
3. The remaining two squares with 1's are also adjacent by the new definition.
4. These two squares, when combined, give the two-literal term **xz**.
5. Results the simplified function:

F = yz + xz'

Examples

6. Simplification Principle:
   - Four adjacent squares in a three-variable map represent the logical sum of four minterms, resulting in an expression with only one literal.
     - Example: The sum of adjacent minterms 0, 2, 4, and 6 simplifies to **z'**.
     - Consider the logical sum of the four adjacent minterms 0, 2, 4, and 6.
     - Simplified to the single literal term **z'**:

$$\begin{aligned} m_0 + m_2 + m_4 + m_6 &= x'y'z' + x'yz' + xy'z' + xyz' \\ &= x'z'(y' + y) + xz'(y' + y) \\ &= x'z' + xz' = z'(x' + x) = z' \end{aligned}$$

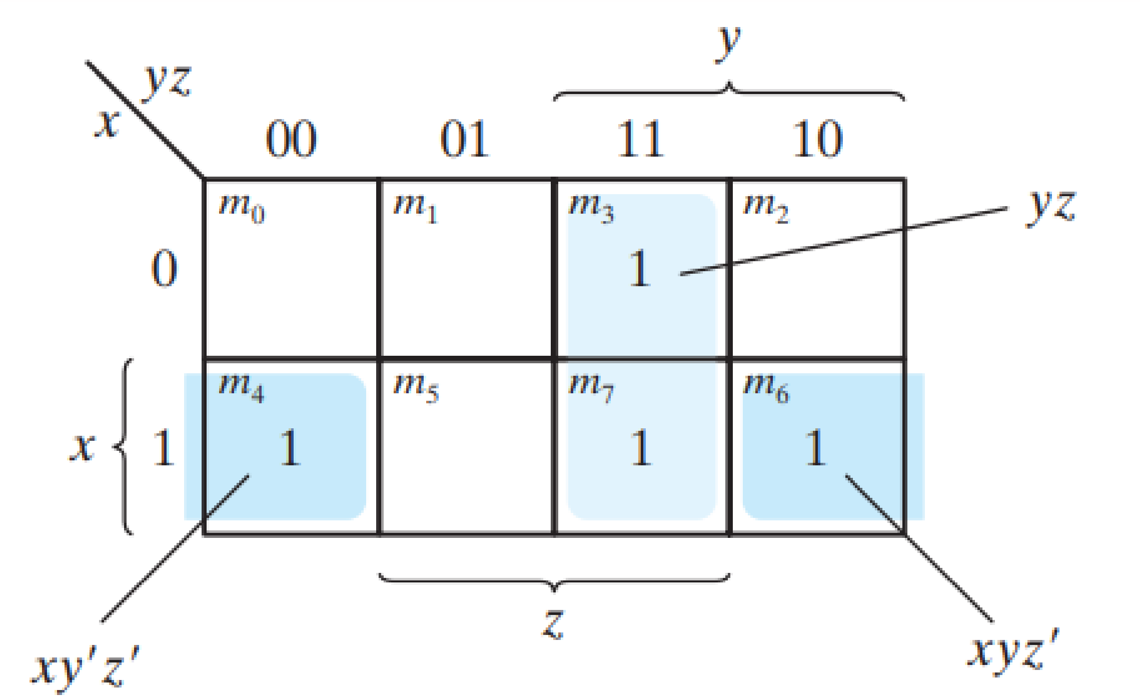


Note: $xy'z' + xyz' = xz'$

Adjacent Squares and Literal Reduction

The number of Adjacent Squares and Literal Reduction:

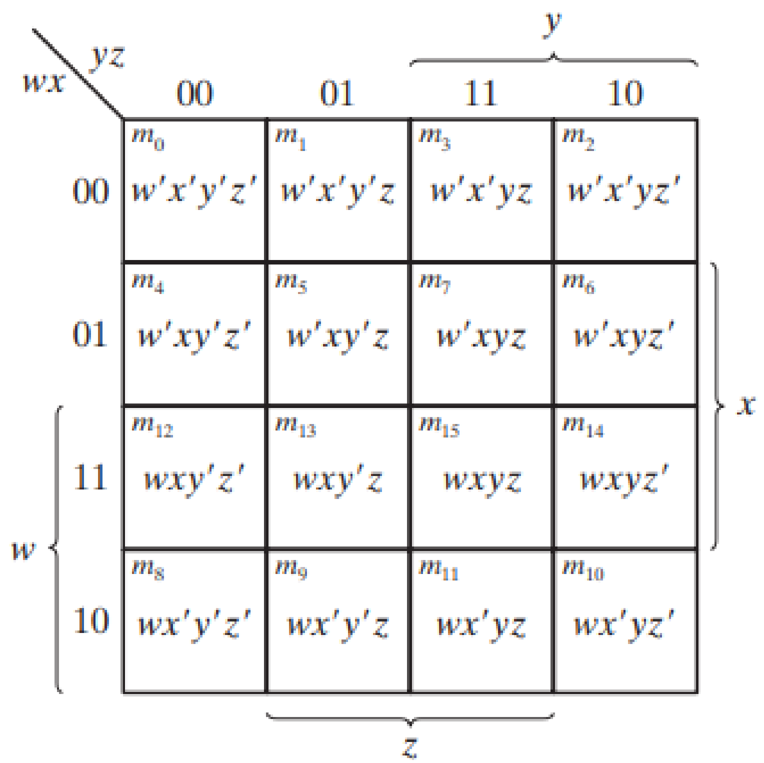
- The number of adjacent squares combined should always be a power of two (e.g., 1, 2, 4, 8).
- One square represents a term with three literals.
- Two adjacent squares represent a term with two literals.
- Four adjacent squares represent a term with one literal.
- Eight adjacent squares encompass the entire map and result in a function always equal to 1.



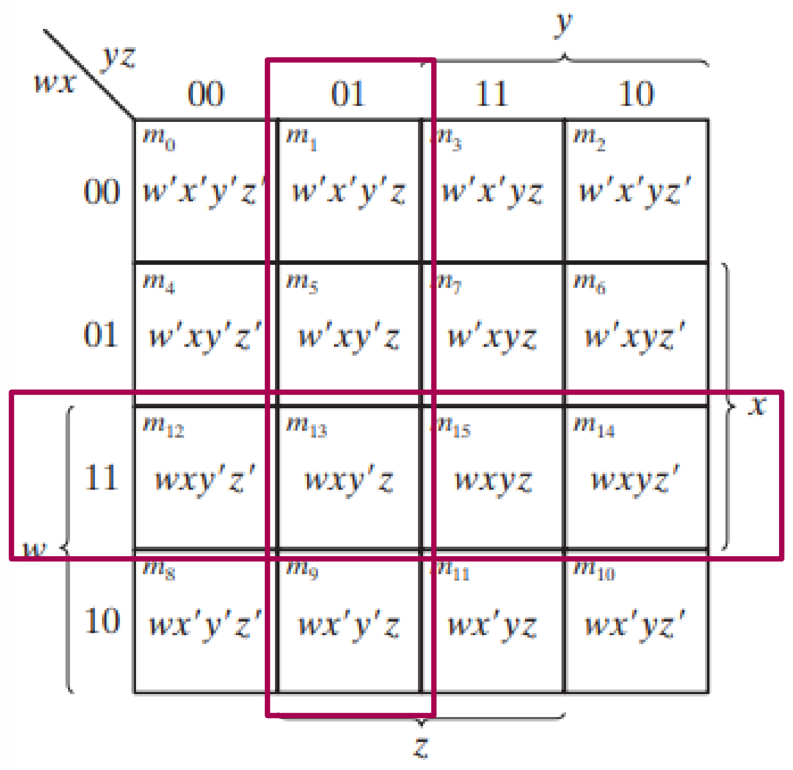
Note: $xy'z' + xyz' = xz'$

Four-Variable K-Map

- Rows and columns are numbered in a Gray code sequence, with only one digit changing between adjacent rows or columns.



- The minterm for each square is obtained by concatenating the row and column numbers.
- For example, the third row (11) and the second column (01) when concatenated yield the binary number 1101, equivalent to decimal 13.
- Thus, the square in the third row and second column represents minterm $\mathbf{m_{13}}$.



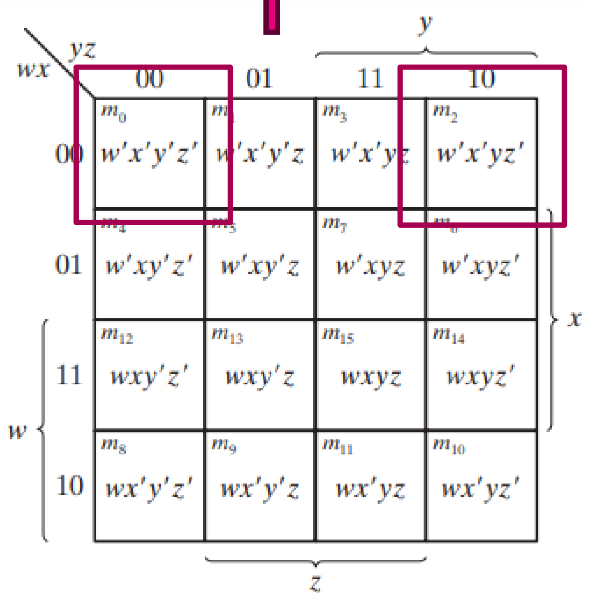
Four-Variable K-Map

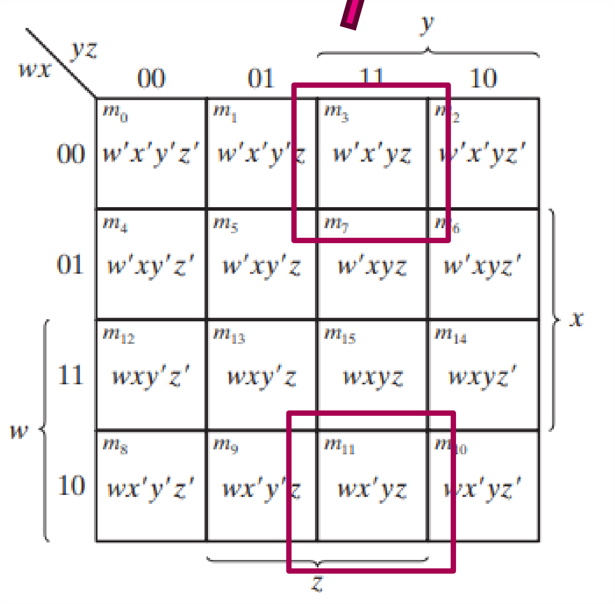
- Minimization of four-variable Boolean functions is similar to the method used for three-variable functions.

- Adjacent squares are defined as those next to each other.

- The map is considered to be on a surface where the top and bottom edges, as well as the right and left edges, touch each other to form adjacent squares.

- For example, m0 and m2 are adjacent squares, and so are m3 and m11.





Four-Variable K-Map

The combination of adjacent squares during simplification can be determined from inspection of the four-variable map:

- One square represents a minterm, resulting in 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 yield a function that is always equal to 1.

Prime Implicants

Choosing adjacent squares in a map:

1. Ensure all the minterms of the function are covered when we combine the squares,

2. Ensure the number of terms in the expression is minimized

3. Ensure that, there are no redundant terms (i.e., minterms already covered by other terms).

Prime Implicants

- A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.
- If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.
- The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.

Prime Implicants

- A single 1 on the map represents a prime implicant if it's not adjacent to any other 1’s.
- Two adjacent 1's form a prime implicant, provided 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.

Prime Implicants

Essential Prime Implicants:

- To find essential prime implicants, examine each square marked with a 1 and check how many prime implicants cover it.
- If a square is covered by only one prime implicant, that prime implicant is considered essential.

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Gate-Level Minimization

University
of Galway.ie

Product-of-Sum Simplification

- **Sum-of-Products Form:**
  - The minimized Boolean functions obtained from Karnaugh maps were expressed in the sum-of-products form.

- **Obtaining Product-of-Sums Form:**
  - To derive a minimized function in product-of-sum form, consider the basic properties of Boolean functions.
  - The 1's in the map represent the minterms of the function, while minterms not included in the standard sum-of-products form denote the complement of the function.

Product-of-Sum Simplification

Complement Representation:

- The complement of a function is represented in the map by squares not marked with 1’s.

- By marking the empty squares with 0's and combining them into valid adjacent squares, a simplified sum-of-products expression of the complement of the function (F') is obtained.

Product-of-Sum Simplification

DeMorgan's Theorem:

- The complement of F' gives back the original function F, thanks to **DeMorgan's theorem**.

- This process ensures that the derived function is automatically in product-of-sums form.

Example Illustration:

- The best way to understand this is through practical examples.

Product-of-Sum Simplification

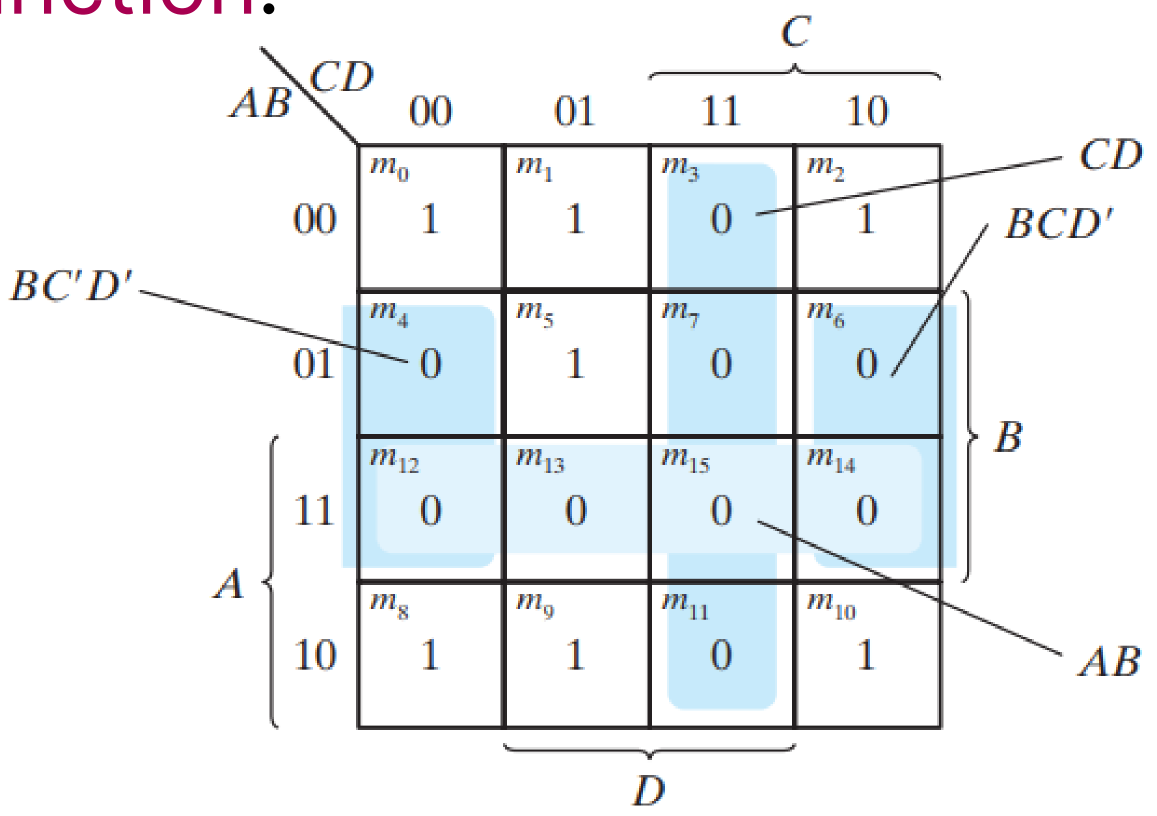
Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

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

1. **Identify Minterms:** In the map, 1's represent included minterms, and 0's represent excluded minterms.

2. **Sum-of-Products Form:** Combine 1's to obtain the simplified function:

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



Note: $BC'D' + BCD' = BD'$

Product-of-Sum Simplification

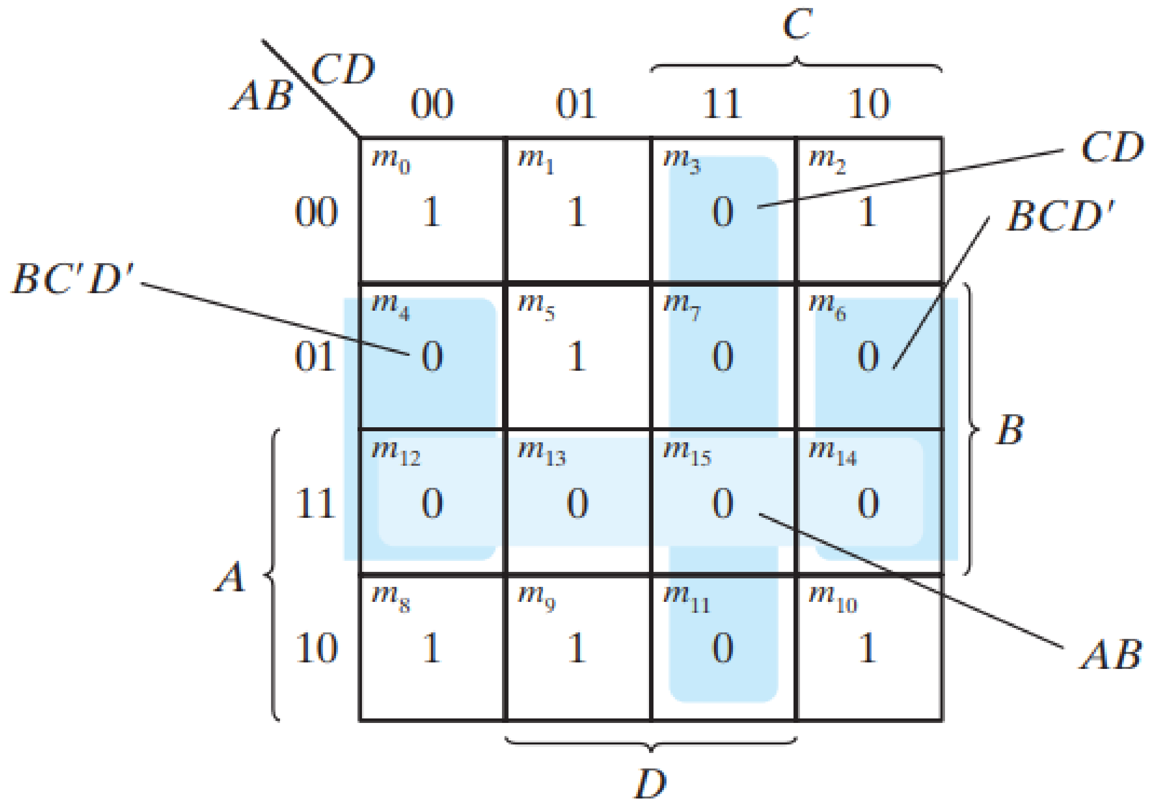
(b) Product-of-Sums Form:

- **Identify Missing Minterms:** Combine squares marked with 0's to obtain the complemented function:

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

- **Apply DeMorgan's Theorem:** The product-of-sums form is obtained by taking the dual and complementing each literal:

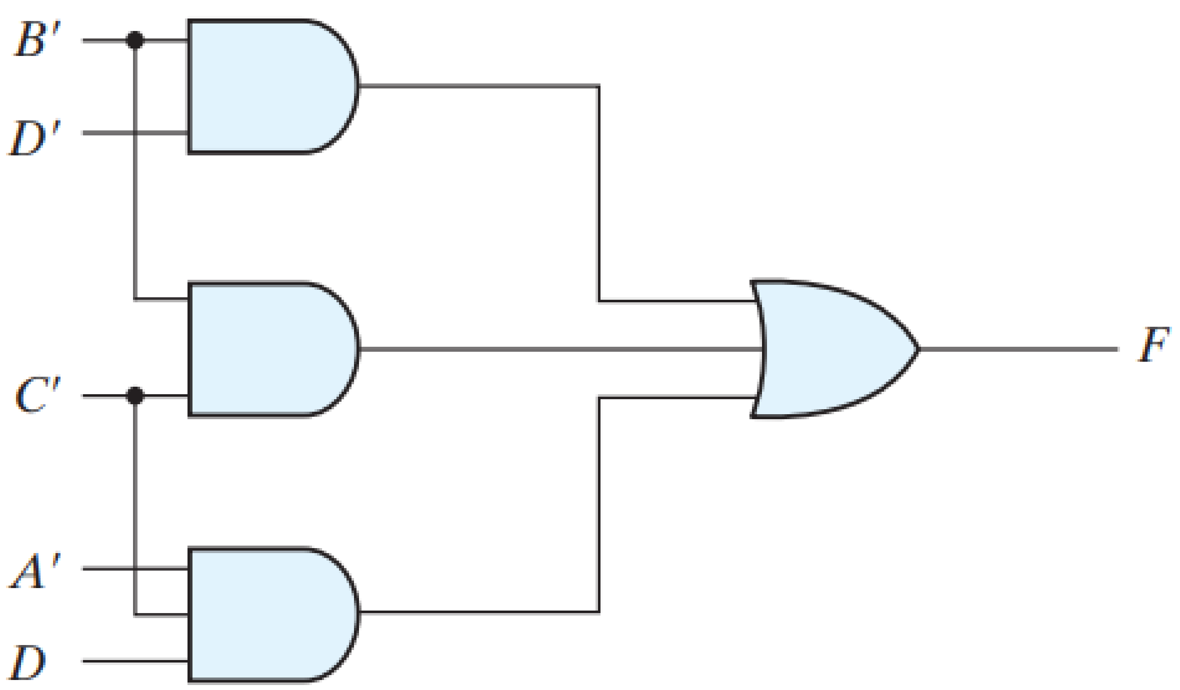
$$F = (A' + B')(C' + D')(B' + D)$$



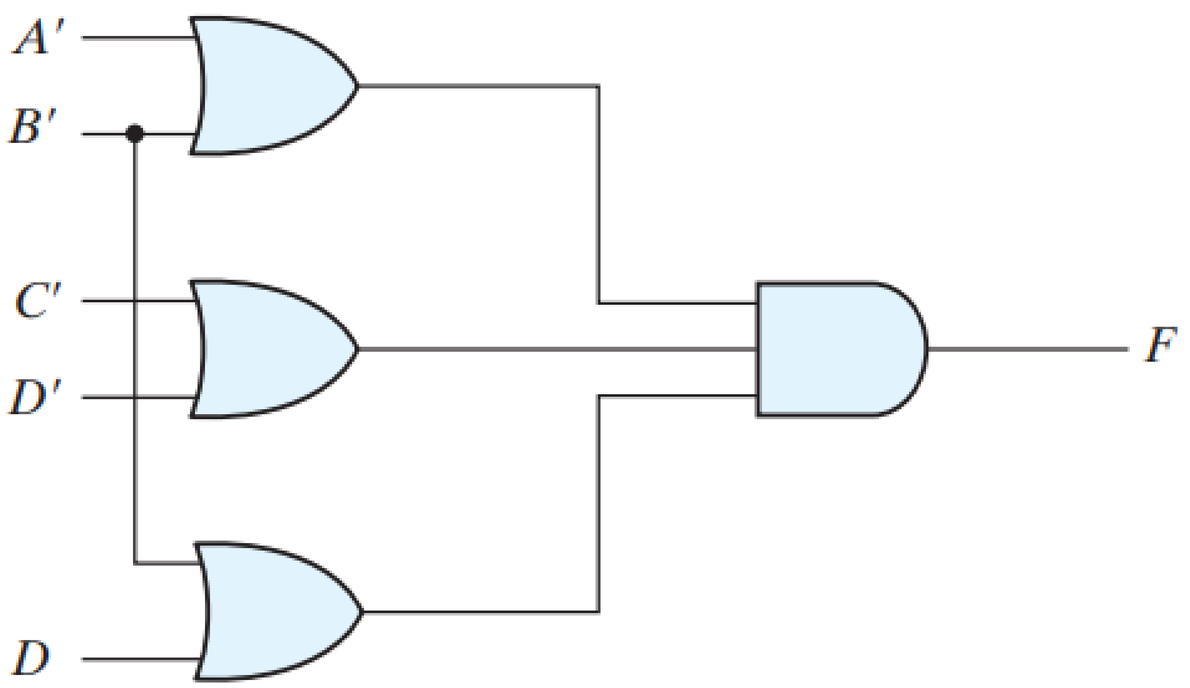
Product-of-Sum Simplification

Gate-Level Implementation:

- **Sum-of-Products Implementation (a):** Implemented with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to a single OR gate.
- **Product-of-Sums Implementation (b):** Implemented with a group of OR gates, one for each OR term. The outputs of the OR gates are connected to the inputs of a single AND gate.



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



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

Don’t Care Conditions

- In Boolean functions, the logical sum of minterms specifies when the function equals 1, while it equals 0 for all other minterms.
- In practical applications, some variable combinations might not be specified or needed.
- These unspecified conditions are known as "don't-care" conditions.
- They allow for further simplification of Boolean expressions by optimizing the function, and we often don't need to know the specific output values for these conditions.
- These situations are common in real-world applications and are referred to as incompletely specified functions.
- Don't-care conditions can be used on a map to simplify Boolean expressions even further.

Don’t Care Conditions

- They allow for further simplification of Boolean expressions by optimizing the function, and we often don't need to know the specific output values for these conditions.

- These situations are common in real-world applications and are referred to as incompletely specified functions.

- Don't-care conditions can be used on a map to simplify Boolean expressions even further.

Don’t Care Conditions

- **Don't-Care Minterms:** These minterms represent combinations of variables where the logical value is unspecified.

- These minterms are not marked with a 1 in the map because that would imply the function is always 1 for such combinations.

- Likewise, marking them with 0 would imply the function is always 0 for those combinations.

Don't Care Conditions

- **Representation:**
  - An X is used to distinguish the don't-care condition from 1's and 0's.
  - An X is used inside a square in the map to indicate that we don't care whether the value of 0 or 1 is assigned to F for the particular minterm, distinguishing it from 1's and 0's.
- **Flexibility in Simplification:**
  - When simplifying the function using a map, we have the flexibility to treat don't-care minterms as either 0 or 1.
  - This choice depends on which combination results in the simplest expression.

Example

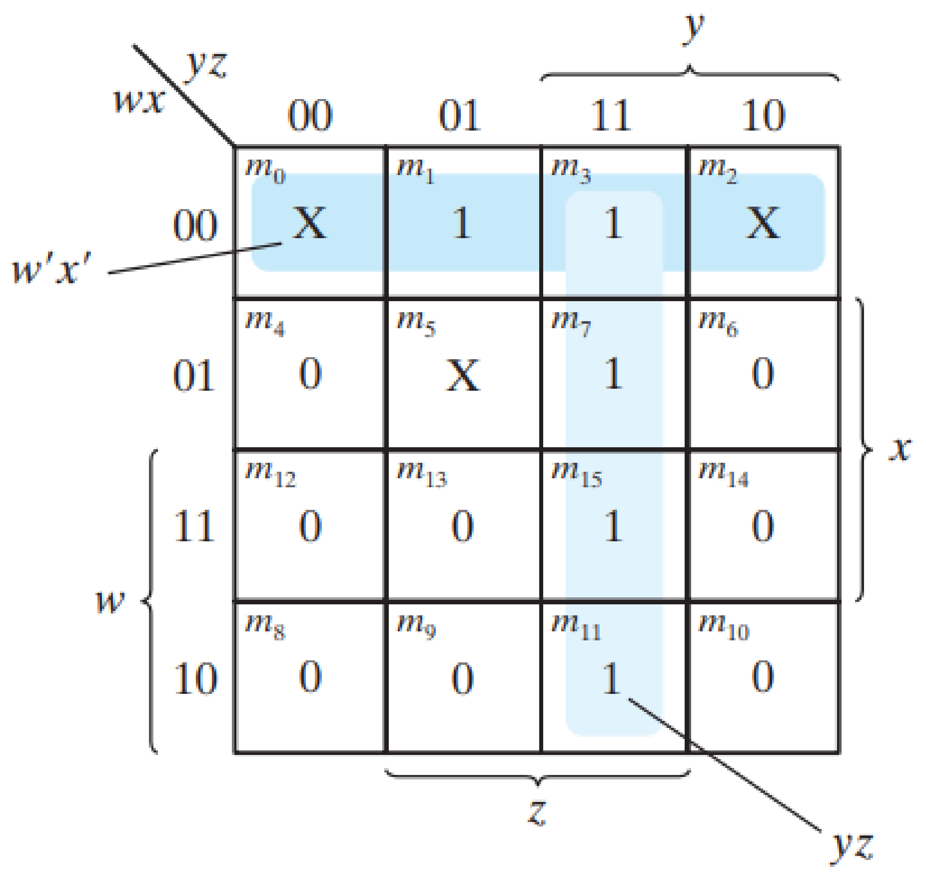
Simplify the Boolean function

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

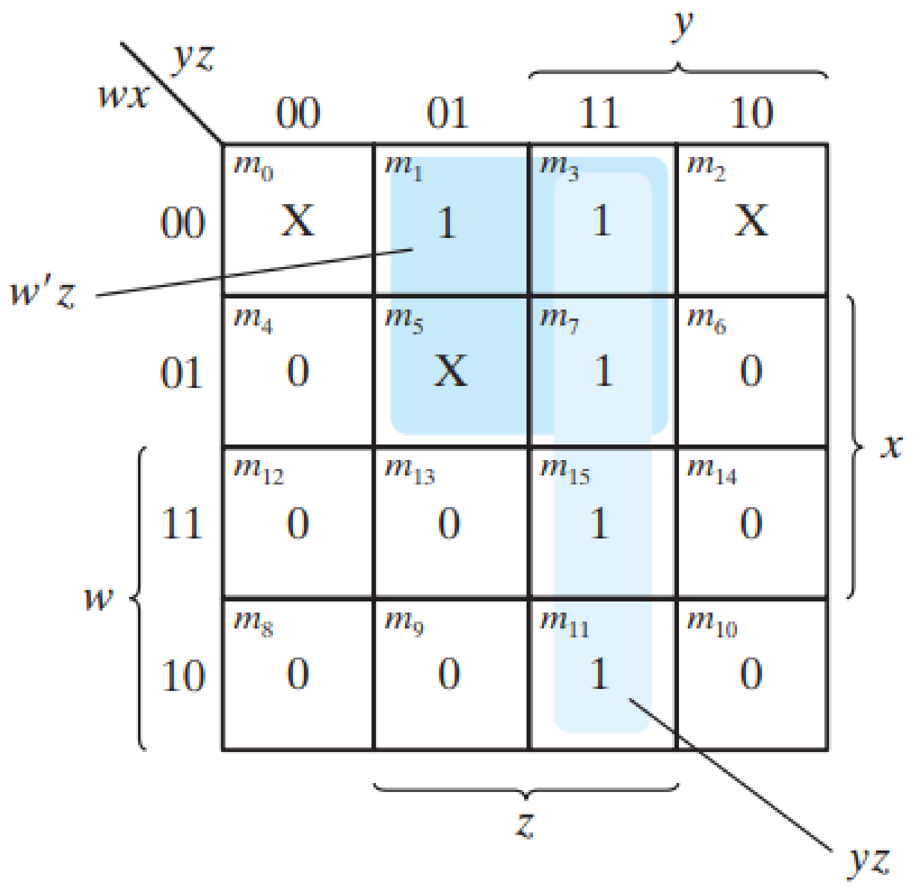
which has the don't-care conditions

$d(w, x, y, z) = \Sigma(0, 2, 5)$

- **Map Simplification:**
    - Minterms of F are marked with 1's.
    - Don't-care minterms of d are marked with X's.
    - Remaining squares are filled with 0's.



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

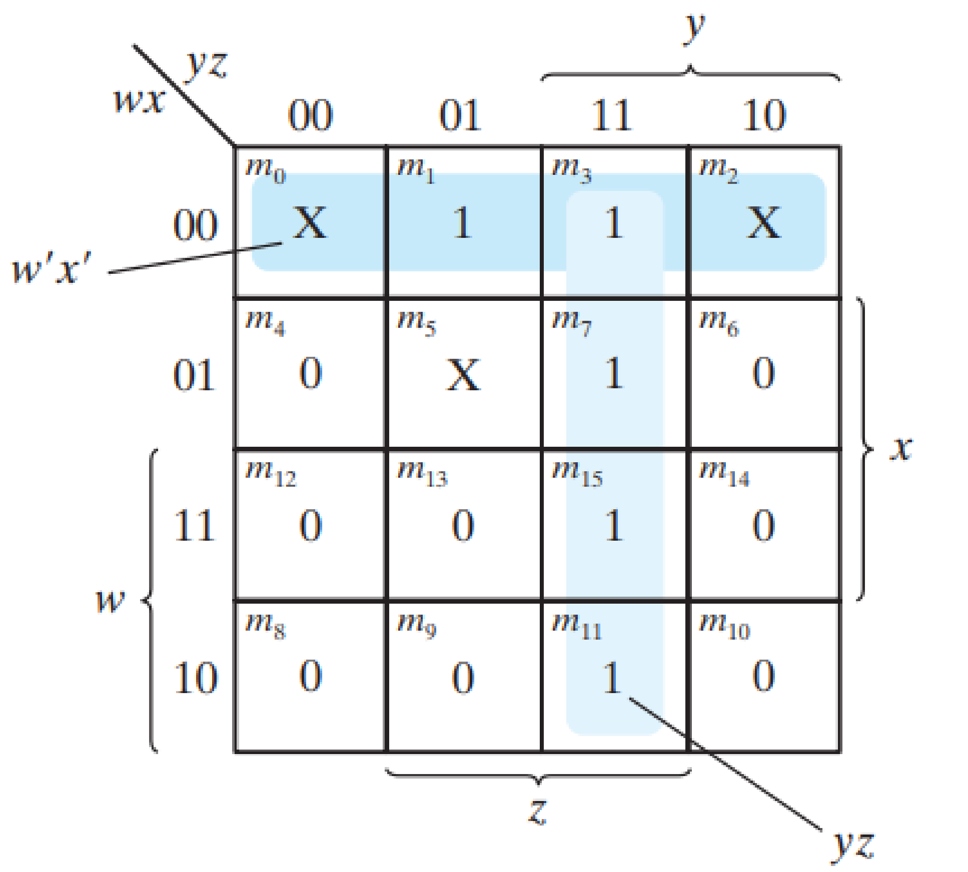
Example

- **Sum-of-Products Form:** To simplify F in sum-of-products form, all five 1's in the map must be included.

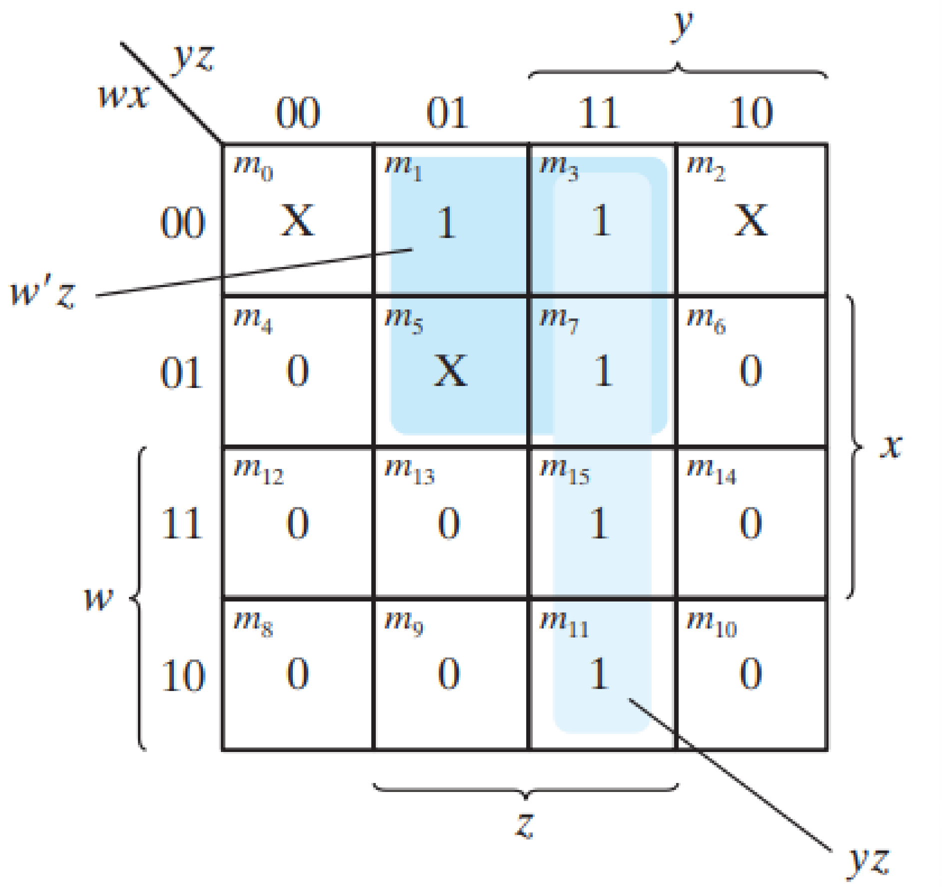
- **Term YZ:** The term **yz** covers the four minterms in the third column.

- **Remaining Minterms:** Minterm m_1 can be combined with m_3 to give the three-literal term **w'x'z**.

- **Using Don't-Care Minterms:** By including one or two adjacent **X's**, we can combine four adjacent squares to give a two-literal term.



(a) $F = yz + w'x'$

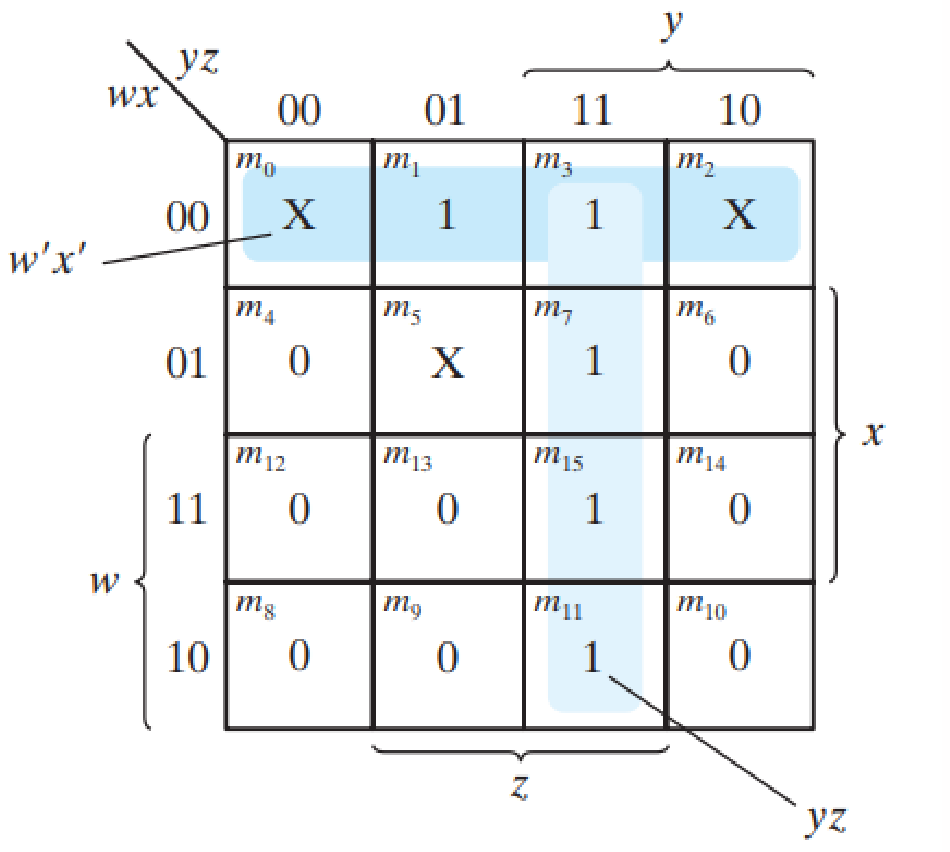


(b) $F = yz + w'z$

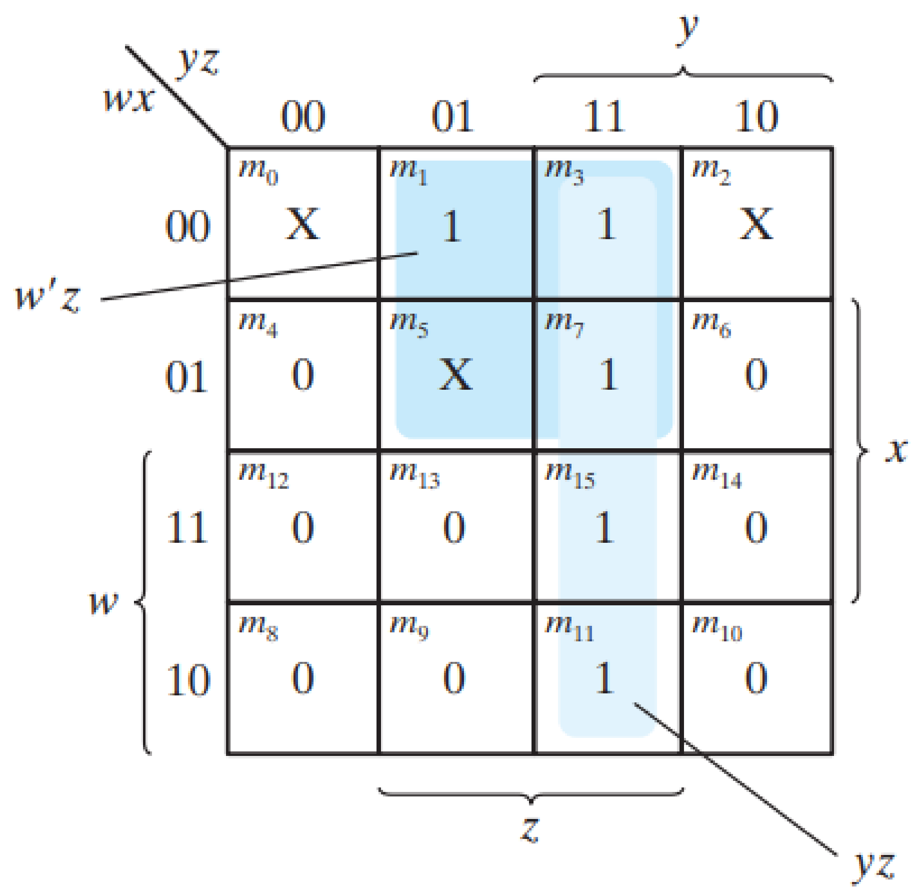
Example

- **Two Simplified Functions:**
  - In Fig. (a), don't-care minterms 0 and 2 are included with the 1's, resulting in the simplified function **F = yz + w'x'.**
  - In Fig. (b), don't-care minterm 5 is included with the 1's, and the simplified function is now **F = yz + w'z**.

- **Either Expression Works:** Either one of the preceding two expressions satisfies the conditions stated for this example.



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

Don’t Care Conditions

- In the previous example, the don't-care minterms in the map are initially marked with X's, and they can be considered as either 0 or 1.

- **Choice in Simplification:** The choice between 0 and 1 for the don't-care minterms depends on the way the incompletely specified function is simplified.

- Resulting Simplified Functions:

  - The first expression is **$F(w, x, y, z) = yz + w'x' = \sum(0, 1, 2, 3, 7, 11, 15)$**.

  - The second expression is **$F(w, x, y, z) = yz + w'z = \sum(1, 3, 5, 7, 11, 15)$**.

Don’t Care Conditions

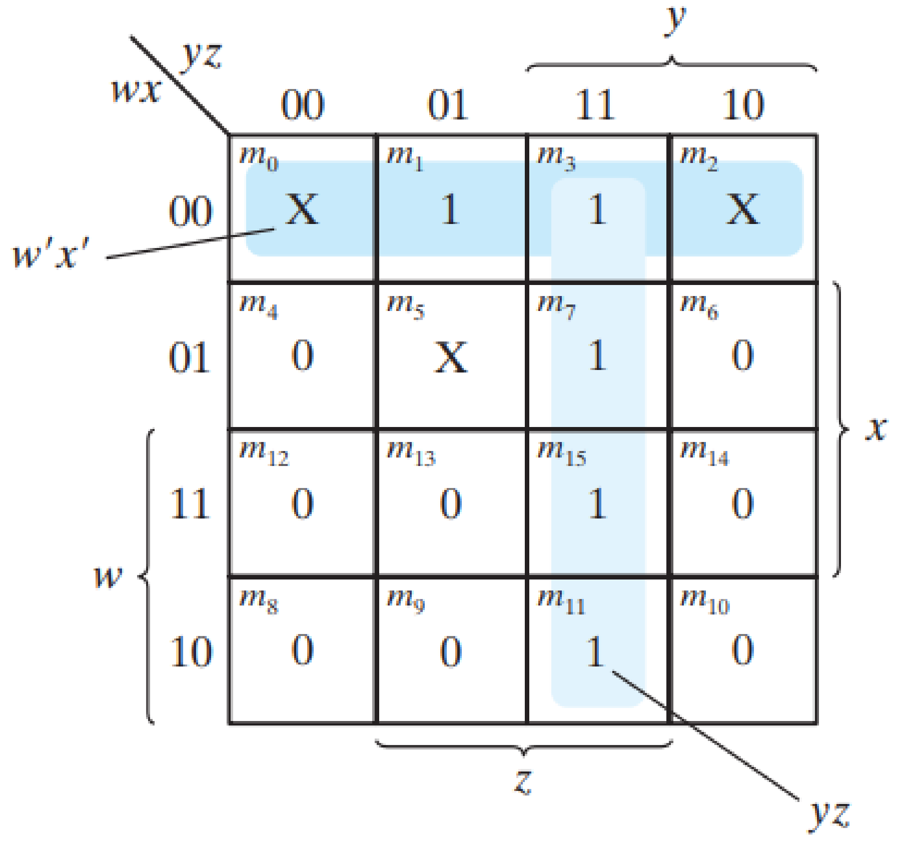
- Both expressions include minterms 1, 3, 7, 11, and 15, making **F equal to 1**.
- Don't-care minterms (0, 2, and 5) are handled differently in each expression.
- The first expression includes 0 and 2 with 1's and assigns 0 to minterm 5.
- The second expression includes minterm 5 with 1's and assigns 0 to 0 and 2.

Don't Care Conditions

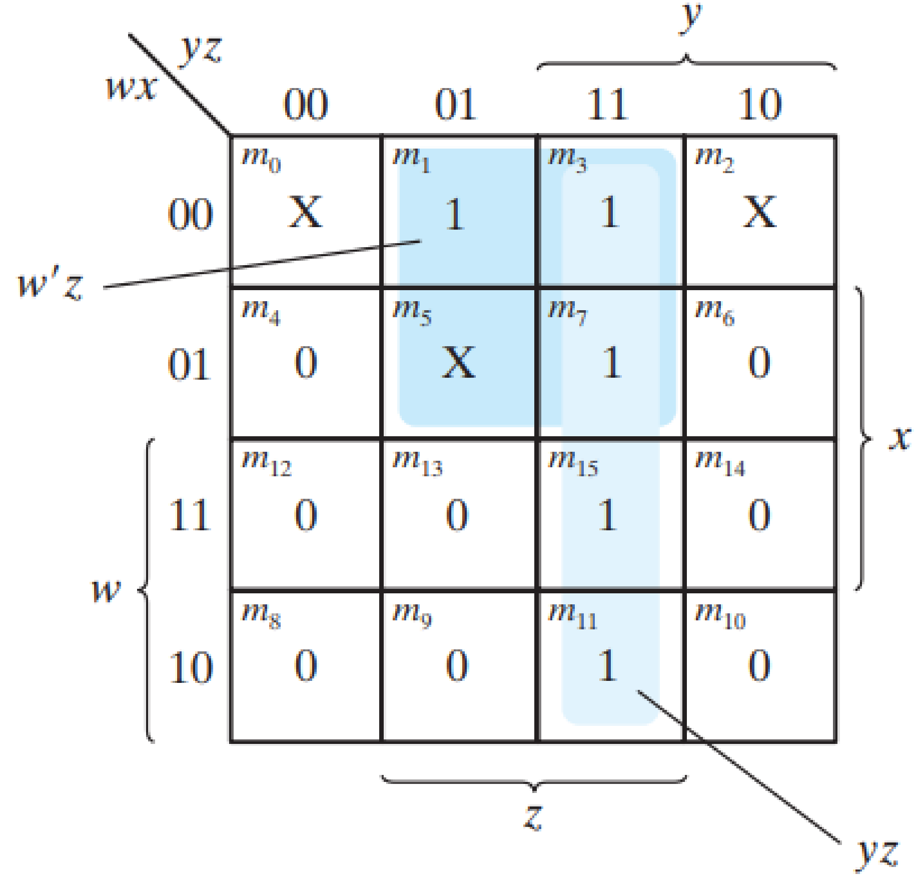
- These two expressions represent different functions, not algebraically equal.
- Both cover specified minterms but handle don't-care minterms differently.
- In the context of the incompletely specified function, either expression is acceptable, differing only in the value of F for don't-care minterms.

Don’t Care Conditions

- Simplifying the function from Figure in **product-of-sums form**:
  - Combining the 0's by including don't-care minterms 0 and 2 with them.
  - This results in the simplified complemented function: **F' = z' + wy’**.
  - Taking the complement of **F'** gives the simplified expression in product-of-sums form: **F = z(w' + y) = ∑(1, 3, 5, 7, 11, 15).**
  - This expression includes minterms 0 and 2 with the 0's and 5 with the 1's.



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

References

- Computer Organization and Architecture Designing for Performance Tenth Edition by William Stallings
- Digital Design With an Introduction to the Verilog HDL FIFTH EDITION by M Morris, M. and Michael, D., 2013.

Ollscoil na Gaillimhe
University of Galway

Thank *you*