MEC 3102 – PRODUCTION ENGINEERING I AND ELECTRICITY & ELECTRONICS II

MR. Boyd Munkombwe

Engineering Annex Board Room

Cell: (+260)-968315273/50435239/72860920

Email: boyd.munkombwe@unza.zm

MEng in Power Electronics and Motor Control (Southeast; China, 2020)

BEng in Electrical Power System and Machines (UNZA; January, 2017)

LECTURE 2.3

KARNAUGH MAPS

Introduction

- ✓Here, we will discuss the Karnaugh Map technique other than the application of laws and theorems of Boolean algebra discussed in the preceding lectures for minimizing a given complex Boolean expression.
- ✓The primary objective of all simplification procedures is to obtain an expression that has the minimum number of terms.
- ✓Obtaining an expression with the minimum number of literals is usually the secondary objective.
- ✓If there is more than one possible solution with the same number of terms, the one having the minimum number of literals is the choice.

Logic circuit Analysis

- ✓The analysis of a logic ckts consists in writing a logic statement expressing the overall operation performed in the operation.
- ✓This can be done in a straight forward manner, by starting at the input and tracing through the circuit, noting function realized at each output.

- ✓The resulting expression can be simplified or written in an alternative form using Boolean algebra. A truth table can then be constructed.

Logic Circuit Synthesis

- ✓One fascinating aspects of digital electronics is the construction of circuits that can perform simple mental processes at superhuman speeds.
- ✓A typical digital computer can perform thousands of additions of 10-numbers per second.
- ✓The logic designer starts with a logical statement or truth table, converts the logic function into a convenient form, and then realizes the desired function by means of standard or special logic elements.

Simplification Techniques

- ✓Before we move on to discuss the Karnaugh map technique, it would be relevant briefly to describe sum-of-products and product-of-sums Boolean expressions.
- ✓The given Boolean expression will be in either of the two forms, and the objective is to find a minimized expression in the same or the other form.

Sum-of-Products Boolean Expressions

- ✓A sum-of-products (SOP) expression contains the sum of different terms, with each term being either a single literal or product of more than one literal.
- ✓It can be obtained from the truth table directly by considering those input combinations that produce a logic '1' at the output. Each such input combination produces a term.
- ✓Different terms are given by the product of the corresponding literals. The sum of all terms gives the expression.

✓ Example 1

Consider the truth table in table 4.2. Obtain the Boolean expression and show the circuit realization. Hence, use the Boolean laws and theorems to minimize this expression and show the circuit realization after simplification.

Table 4.2 Truth table.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

✓ Solution

- ✓ Considering the first term, the output is '1' when $A = 0$, $B = 1$ and $C = 1$.
- ✓ This is only possible when $\overline{A}$, B and C are ANDed.
- ✓ Also, for the second term, the output is '1' only when A, $\overline{B}$ and C are ANDed.

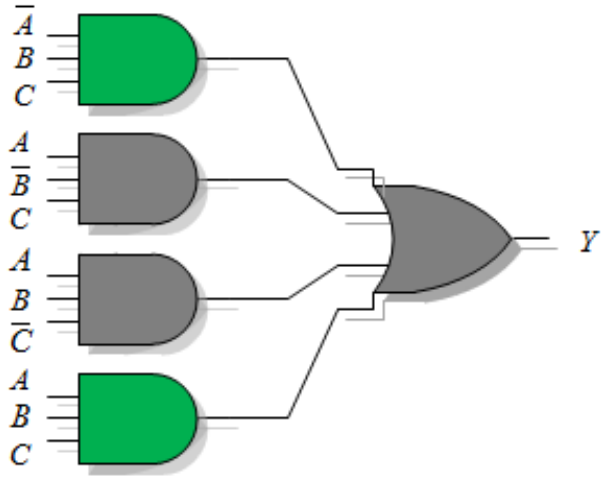
- ✓ Other terms can be explained similarly.

✓ Example 1 Solution

- The Boolean expression is thus,

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

- Assuming that the complement of each variable is available, as is true in most computers, the straight forward ckt realization is as shown in Figure 4.11 (a).



(a) Circuit realisation.

Figure 4.11

✓ Example 1 Solution

- To simplify the Boolean expression, we first expand it, i.e.,

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

since $ABC + ABC = ABC$, by the **Idempotent theorem**.

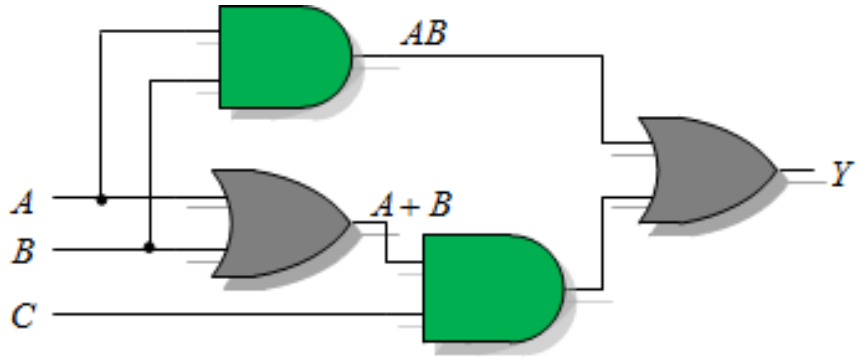
- By the **distributive theorem**, we factor the above expression to yield

$$Y = C\left(\overline{A}B + A\overline{B} + AB\right) + AB\left(\overline{C} + C\right)$$

- Recall that $\overline{C} + C = 1$, and $\overline{A}B + A\overline{B} + AB = A + B$, the function becomes

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

- This function requires only four logic elements as shown in Figure 12 (b).



(b) Ckt after Boolean simplification.

Figure 4.12

Simplification Techniques

Product-of-Sums Boolean Expressions

- ✓A product-of-sums (POS) expression contains the product of different terms, with each term being either a single literal or a sum of more than one literal.
- ✓It can be obtained from the truth table by considering those input combinations that produce a logic ‘0’ at the output. Each such input combination gives a term , and the product of all such terms gives the expression.
- ✓Different terms are obtained by taking the sum of the corresponding literals.
- ✓Here, ‘0’ and ‘1’ respectively mean the uncomplemented and complemented variables, unlike sum-of-products expressions where ‘0’ and ‘1’ respectively mean complemented and uncomplemented variables.
- ✓To illustrate this, consider the truth table in Table 1 of example 1.

✓ Example 2

- Consider the truth table in table 1. Obtain the product-of-sums Boolean expression. Hence, show that it equals its sum-of-products dual.

Solution

- Each term in this case is a sum of literals implemented using an OR operation.
- Now, an OR gate produces a logic '0' only when all its inputs are in the logic '0' state.
- Thus, the first term corresponding to the first row of the table will be $A+B+C$. All the other terms are obtained in a similar manner.
- Therefore, the expression is $Y=(A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+C)$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

✓ Example 2 Solution

- Transforming the given product-of-sums expression into an equivalent sum-of-products is a straightforward process.
- We simply find the dual of the expression.
- **Dual of a Boolean Expression**
- The dual of a Boolean expression is obtained by replacing all ‘ $\cdot$ ’ operations with ‘+’ operations, all ‘+’ operations with ‘ $\cdot$ ’ operations, all 0s with 1s and all 1s with 0s and leaving all literals unchanged.
- Thus, dual of $Y = (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (\overline{A} + B + C)$ is

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

Expanded Forms of Boolean Expressions

- ✓Are useful not only in analyzing Boolean expressions but in the application of minimizing techniques such as the Karnaugh mapping method for simplifying Boolean expressions.
- ✓The expanded form, sum-of-products (SOP) or product-of-sums (POS), is obtained by including all possible combinations of missing variables.
- ✓To illustrate this, consider the following sum-of-products expression:

$$A \cdot \overline{B} + B \cdot \overline{C} + A \cdot B \cdot \overline{C} + \overline{A} \cdot C$$

- ✓ It is a three-variable expression. Expanded versions of different miniterms can be written as follows:
  - ❑ $A \cdot \overline{B} = A \cdot \overline{B} \cdot (C + \overline{C}) = A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C}$.
  - ❑ $B \cdot \overline{C} = B \cdot \overline{C} \cdot (A + \overline{A}) = B \cdot \overline{C} \cdot A + B \cdot \overline{C} \cdot \overline{A}$.
  - ❑ $A \cdot B \cdot \overline{C}$ is a complete term and has no missing variable .
  - ❑ $\overline{A} \cdot C = \overline{A} \cdot C \cdot (B + \overline{B}) = \overline{A} \cdot C \cdot B + \overline{A} \cdot C \cdot \overline{B}$

✓The expanded sum-of-products expression is therefore given by

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

✓ As another illustration, consider the product-of-sums expression

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

✓ It is four-variable expression. $\overline{A}+B$ in this case expands to

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

✓ The expanded product-of-sums expression is therefore given by

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

Canonical Form of Boolean Expressions

- ✓An expanded form of Boolean expression, where each term contains all Boolean variables in their true or complemented form, is also known as the **canonical form** of the expression
- ✓For instance, $f(A,B,C) = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot\overline{B}\cdot C + A\cdot B\cdot C$ is a Boolean function of three variables expressed in canonical form.
- ✓This function after simplification reduces to $f(A,B,C) = \overline{A}\cdot\overline{B} + A\cdot B\cdot C$ and loses its canonical form.

Σ and $\prod$ Nomenclature

- ✓Let us consider the Boolean function: $f(A,B,C) = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot\overline{B}\cdot C + A\cdot B\cdot C$
  - Using the Σ notation it is $f(A,B,C) = \sum 0,1,7$.
- ✓Similarly, for the function: $f(A,B,C) = (\overline{A}+\overline{B}+\overline{C})\cdot(\overline{A}+\overline{B}+C)\cdot(A+B+C)$
- ✓Since the sum terms denote binary numbers, 111, 110, and 000, this yields
  - Using the $\prod$ notation it is $f(A,B,C) = \prod 0,6,7.$

Karnaugh Map

- ✓A Karnaugh map is a graphical representation of the logic system. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions.
- ✓Drawing a Karnaugh map from the truth table involves an additional step of writing the minterm or maxterm expression depending upon whether it is desired to have a minimized sum-of-products or a minimized product-of-sums expression.

Construction of a Karnaugh Map

- ✓An *n*-variable Karnaugh map has 2^n squares, and each possible input is allocated a square.
- ✓In the case of a minterm Karnaugh map, '1' is placed in all those squares for which the output is '1', and '0' is placed in all those squares for which the output is '0'. 0s are omitted for simplicity.
- ✓An 'X' is placed in squares corresponding to 'don't cares' conditions.

- ✓In the case of a maxterm Karnaugh map, a ‘1’ is placed in all those squares for which the output is ‘0’, and a ‘0’ is placed for input entries corresponding to a ‘1’ output. Again, 0s are omitted for simplicity, and an ‘X’ is placed in squares corresponding to ‘don’t care’ conditions.
- ✓It is worth noting that, extreme rows and extreme columns are considered adjacent.
- ✓The minterms are ordered according to Gray code, i.e., only one variable changes between adjacent squares.
- ✓The commonly used designation styles for two-, three- and four-variable minterm Karnaugh maps are given in Figure 4.12.
- ✓ Having drawn the Karnaugh map, the next step is to form groups as per the following guidelines:

1. Each square containing a ‘1’ must be considered at least once, although it can be considered as often as desired.
2. The objective is to account for all marked squares in the minimum number of groups.

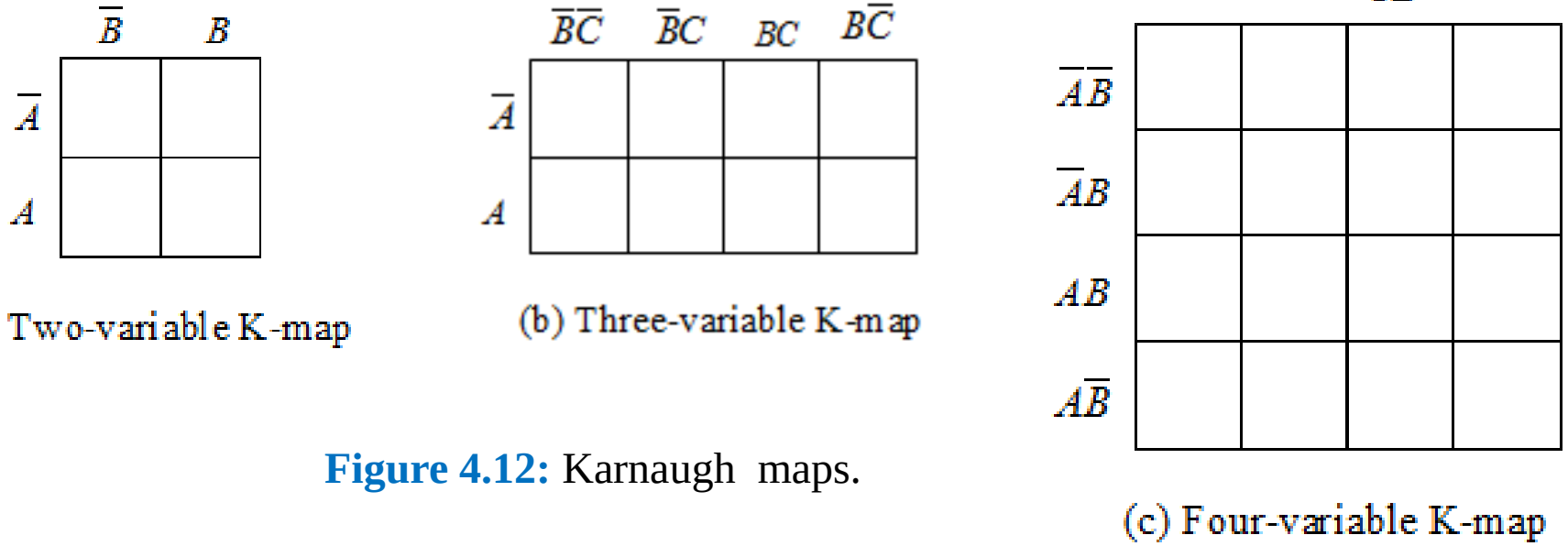


Figure 4.12: Karnaugh maps.

3. The number of squares in a group must always be a power of 2, i.e., groups can have 1, 2, 4, 8, 16, … squares.
4. Each group should be as large as possible, which means that a square should not be accounted for by itself if it can be accounted for by a group of two squares and so forth.
5. 'Don't care' entries can be used in accounting for all of 1-squares to make optimum groups. Not all 'don't cares' need to be accounted for though.

- ✓ Having accounted for groups with all 1s, the minimum 'SOP' or 'POS' expressions can be written directly from the Karnaugh map.

- ✓ **Example 3**

- ❑ Given in table 4.3 is the truth table of the Boolean function of a two-input OR gate. Write the minterm and maxterm Boolean expressions, hence create the minterm Karnaugh map and maxterm Karnaugh map.

Table 4.3: Truth table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

✓ **Solution**

	$\overline{B}$	B
$\overline{A}$		1
A	1	1

Sum-of-products K-map

	$\overline{B}$	B
$\overline{A}$		
A		1

Product-of-sums K-map

$Y = \overline{A} \cdot B + A \cdot \overline{B} + A \cdot B$, (minterm or SOP)

$Y = A + B$, (maxterm or POS)

Karnaugh Map Method

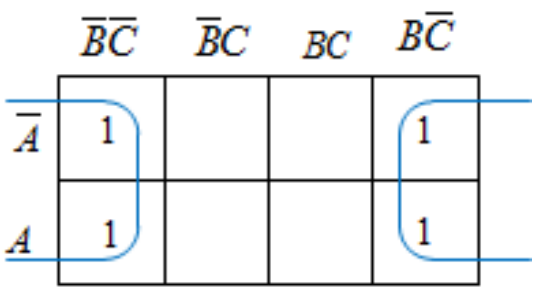
- ✓ **Example 4**
- ✓ Table 4.4 is the truth table of the three-variable Boolean function, with minterm and maxterm repectively given by $Y = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot B\cdot\overline{C} + A\cdot\overline{B}\cdot\overline{C} + A\cdot B\cdot\overline{C}$

  and $Y = (\overline{A}+\overline{B}+\overline{C})\cdot(\overline{A}+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(A+B+\overline{C})$
- ❑ Create the minterm Karnaugh map and maxterm Karnaugh map,

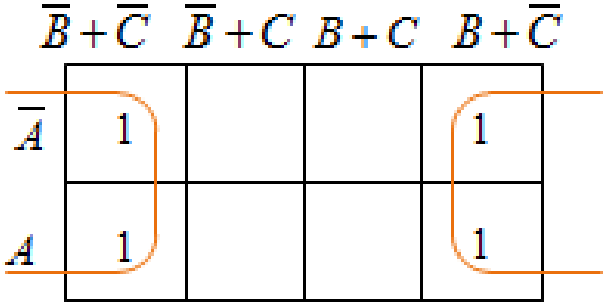
Table 4.4: truth table.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- ✓ **Solution**



Sum-of-products K-map



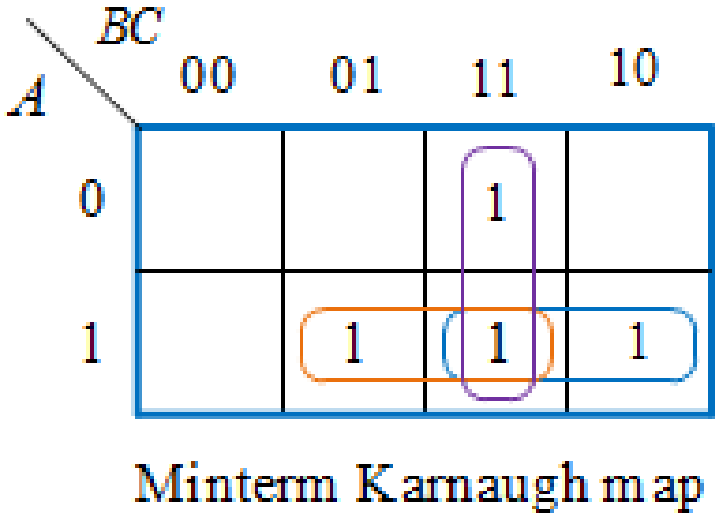
Product-of-sums K-map

- ✓ Thus, the simplified SOP and POS expressions are both given by $Y = \overline{C}$.

✓ Example 5

- The three-variable Boolean function $Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$ was found for the truth table of example 1. Simplify this expression using the Karnaugh map method.
- **Solution**
- Cover all 1s with maximum grouping.
- The simplified Boolean equation is one that sums all the terms corresponding to each of the group:

$$Y = AB + AC + BC$$

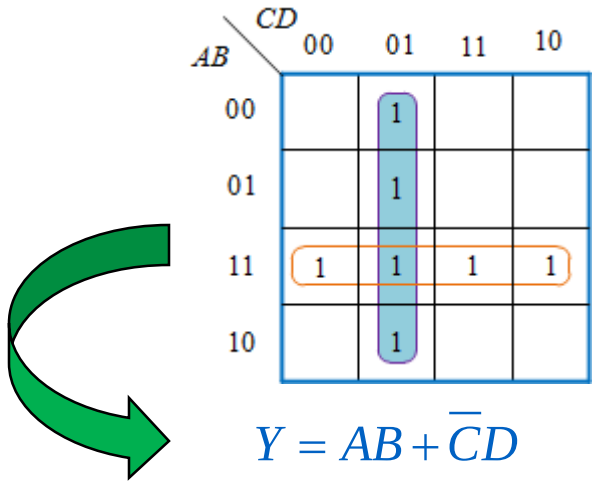


Minterm Karnaugh map

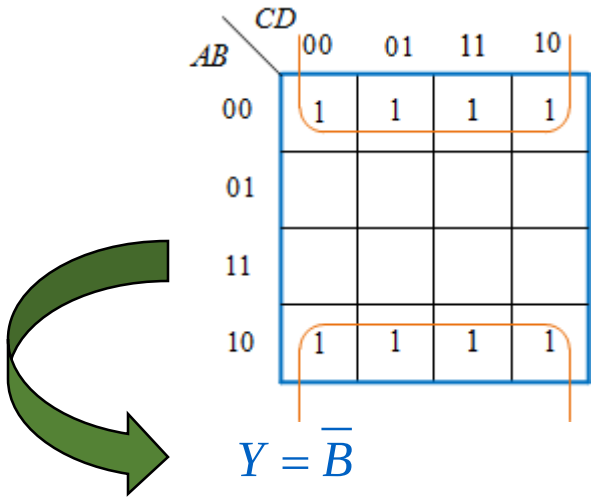
- There is no simpler expression for this function.
- Using DeMorgan's law, the simplified expression can be converted to a NANDed product of NANDs, i.e.,

$$Y = \overline{\overline{AB} \cdot \overline{AC} \cdot \overline{BC}}$$

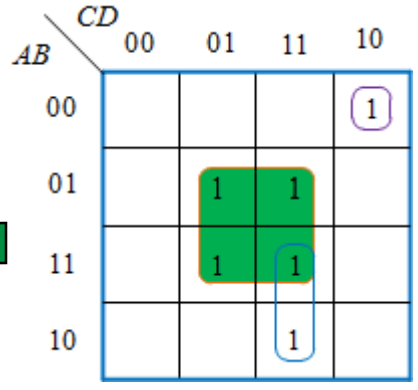
❑Further Examples of grouping



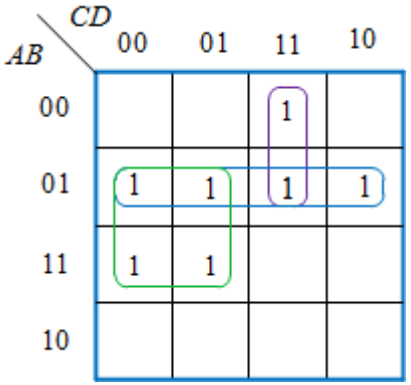
$$Y = AB + \overline{C}D$$



$$Y = \overline{B}$$



$$Y = \overline{A}\overline{B}C\overline{D} + BD + ACD$$



$$Y = \overline{A}B + B\overline{C} + \overline{A}CD$$

✓ Example 6

- The respective four variable minterm and maxterm Boolean expresisons are

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}D + ABCD$$

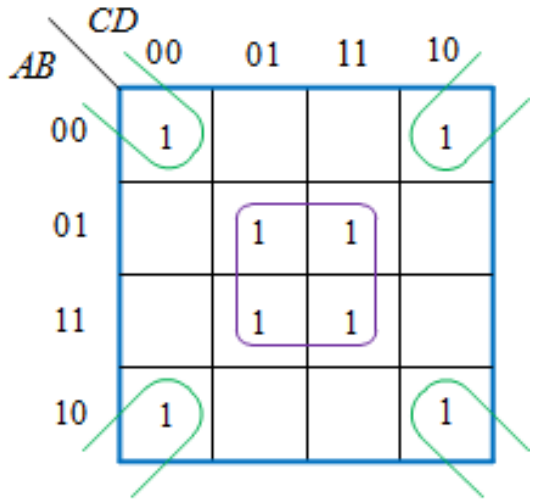
and

$$Y = \left(A + B + C + \bar{D}\right)\cdot\left(A + B + \bar{C} + \bar{D}\right)\cdot\left(A + \bar{B} + C + D\right)\cdot\left(A + \bar{B} + C + \bar{D}\right)$$
$$\cdot\left(A + \bar{B} + \bar{C} + \bar{D}\right)\cdot\left(A + \bar{B} + \bar{C} + D\right)\cdot\left(\bar{A} + \bar{B} + C + \bar{D}\right)\cdot\left(\bar{A} + \bar{B} + \bar{C} + \bar{D}\right)$$
$$\cdot\left(\bar{A} + B + C + \bar{D}\right)\cdot\left(\bar{A} + B + \bar{C} + \bar{D}\right)$$

- Draw the respective minterm and maxterm Karnaugh maps. Hence deduce the minimized expressions from the Karnaugh maps in the two cases.
- **Solution**
- The respective minterm and maxterm Karnaugh maps together with their corresponding simplified expressions are as shown below.

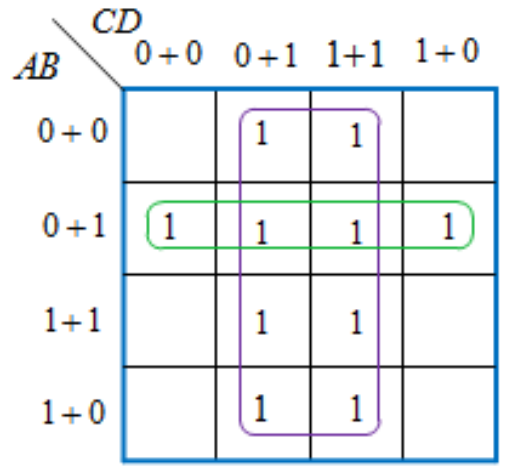
✓ Solution to Example 6

- The respective four variable minterm and maxterm Boolean expresisons are



Sum-of-products K-map

$$Y = \bar{B}\bar{D} + BD$$

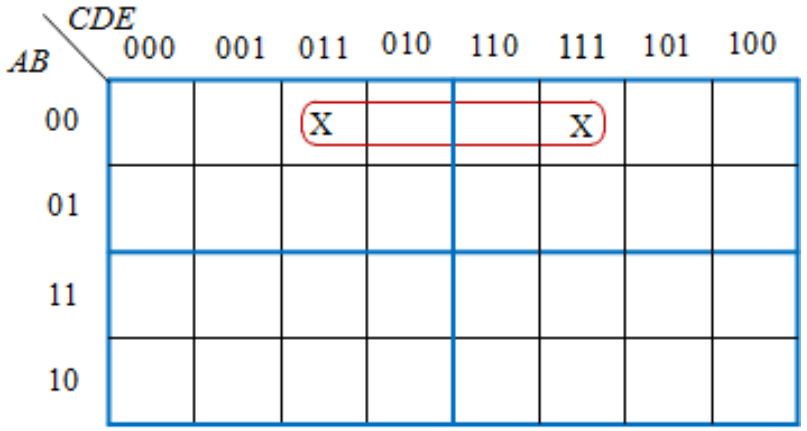


Product-of-sums K-map

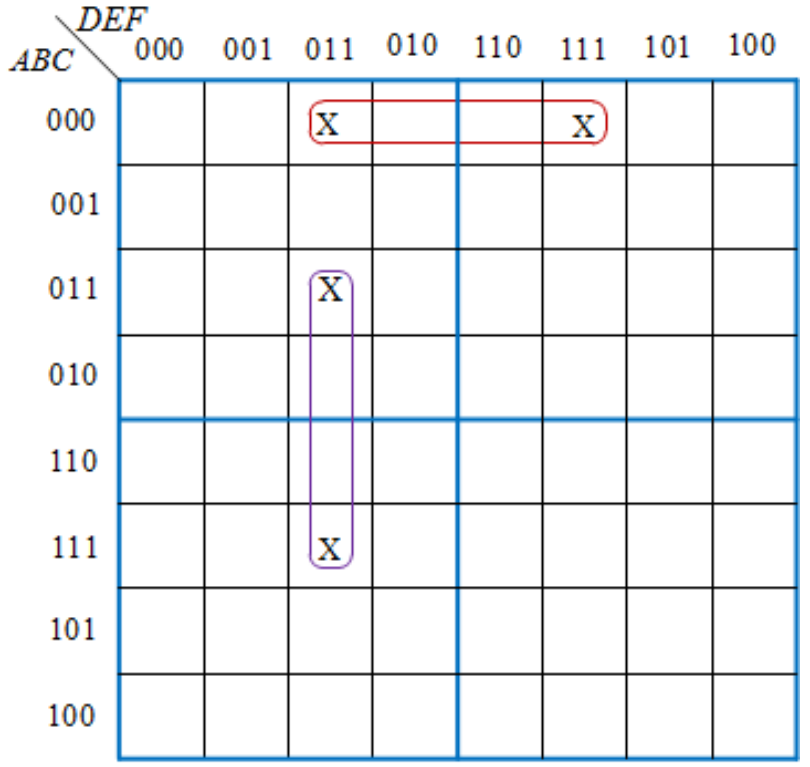
$$Y = \bar{D} \cdot \left(A + \bar{B} \right)$$

K-Map for Boolean Expressions with a Large Number of Variables

- The construction of K-maps for a large number of variables is a complex and cumbersome exercise, although manageable up to six variables.
- Five- and six-variable representation of Karnaugh maps are shown in Figure 4.13(a) and (b).



(a) Five-variable K-map



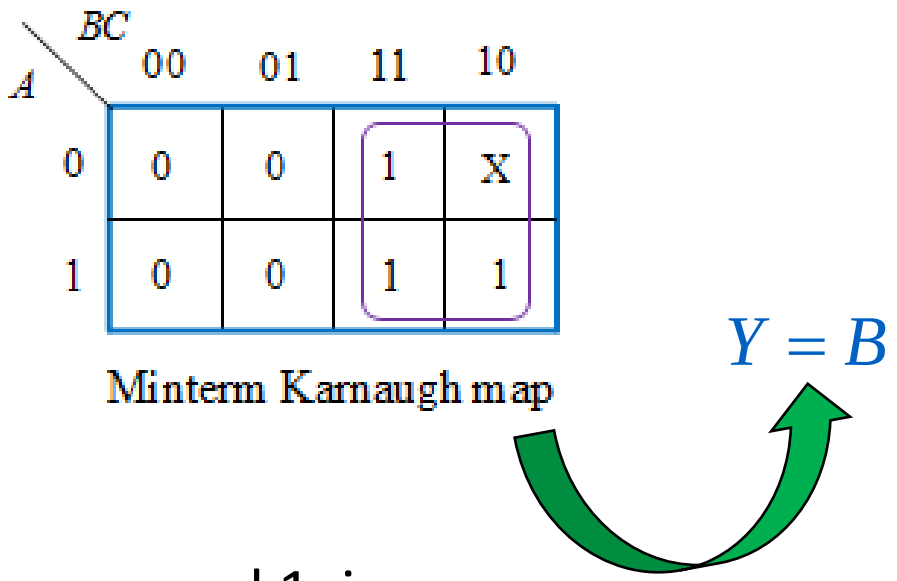
(b) Six-variable K-map

Figure 4.13: Karnaugh maps.

- It is worth noting that while forming groups in Karnaugh maps involving more than four variables is that terms equidistant from the central horizontal and central vertical lines are considered adjacent.
- These lines are shown thicker in Figures 3(a) and (b). Squares marked 'X' in the figures above are adjacent and therefore can be grouped.
- In general, an *n*-variable Boolean expression can be represented by 2^{n-4} four-variable maps.
- In such multiple maps, groups are made as before, except that, in addition to adjacencies discussed earlier, corresponding squares in two adjacent maps are also considered adjacent and can therefore be grouped.

Don’t Care Conditions (Optional)

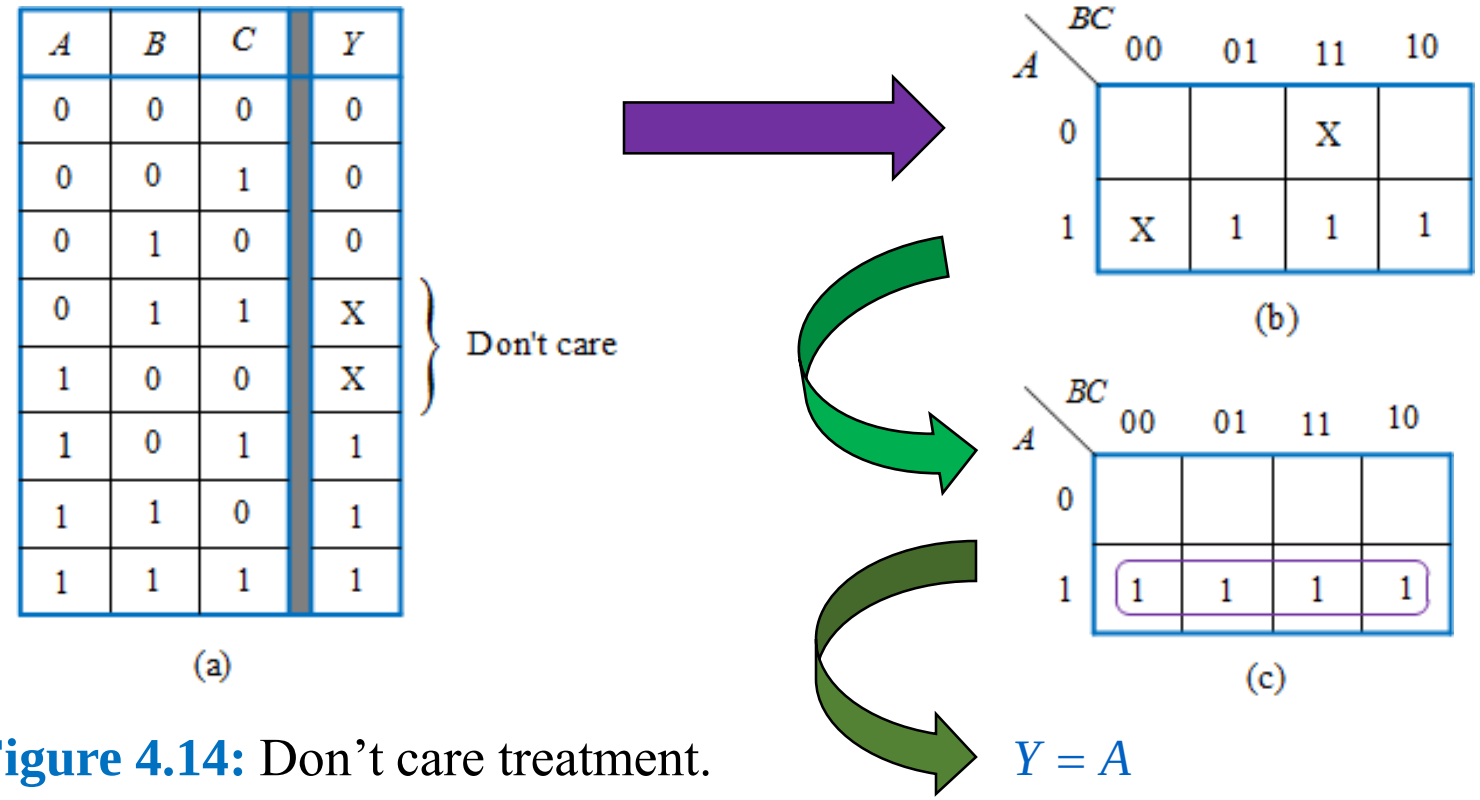
- In certain cases some of the minterms may never occur or it may not matter what happens if they do.
- In such cases we fill in the Karnaugh map with X, meaning don’t care.
- When minimizing an X can be 0 or 1 – whatever helps best with the minimization.
- **For example**



- Simplifies to *B* if X is assumed 1, i.e.,

More "Don't Care" Examples

- Don't care conditions should be changed to either 0 or 1 to produce K-map looping that yields the simplest expression.



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	X
1	0	0	X
1	0	1	1
1	1	0	1
1	1	1	1

Don't care

(a)

A \ BC	00	01	11	10
0			X	
1	X	1	1	1

(b)

A \ BC	00	01	11	10
0				
1	1	1	1	1

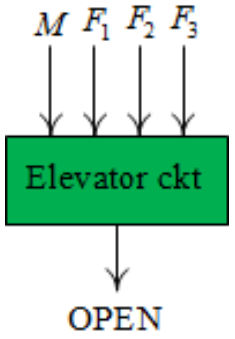
(c)

$Y = A$

Figure 4.14: Don't care treatment.

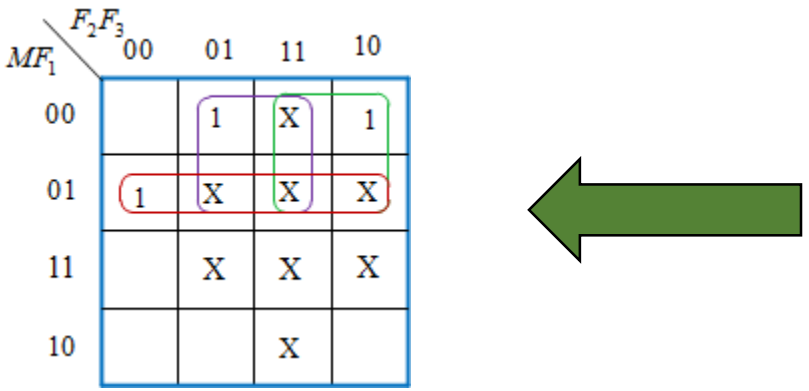
More “Don’t Care” Examples

- Elevator circuit.



Truth table

M	F_1	F_2	F_3	OPEN
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



$$OPEN = \overline{M}F_1 + \overline{M}F_2 + \overline{M}F_3$$

✓ Example 7

- Minimize the Boolean function

$$f(A,B,C)=\sum 0,1,3,5+\sum_{d} 2,7$$

using the mapping method in both minimized sum-of-products and product-of-sums forms. Note that *d* denotes the don't cares conditions

- **Solution**
- $f(A,B,C)=\sum 0,1,3,5+\sum_{d} 2,7=\prod 4,6+\prod_{d} 2,7$.
- From given Boolean functions above, we can write SOP and POS Boolean expressions as follows:

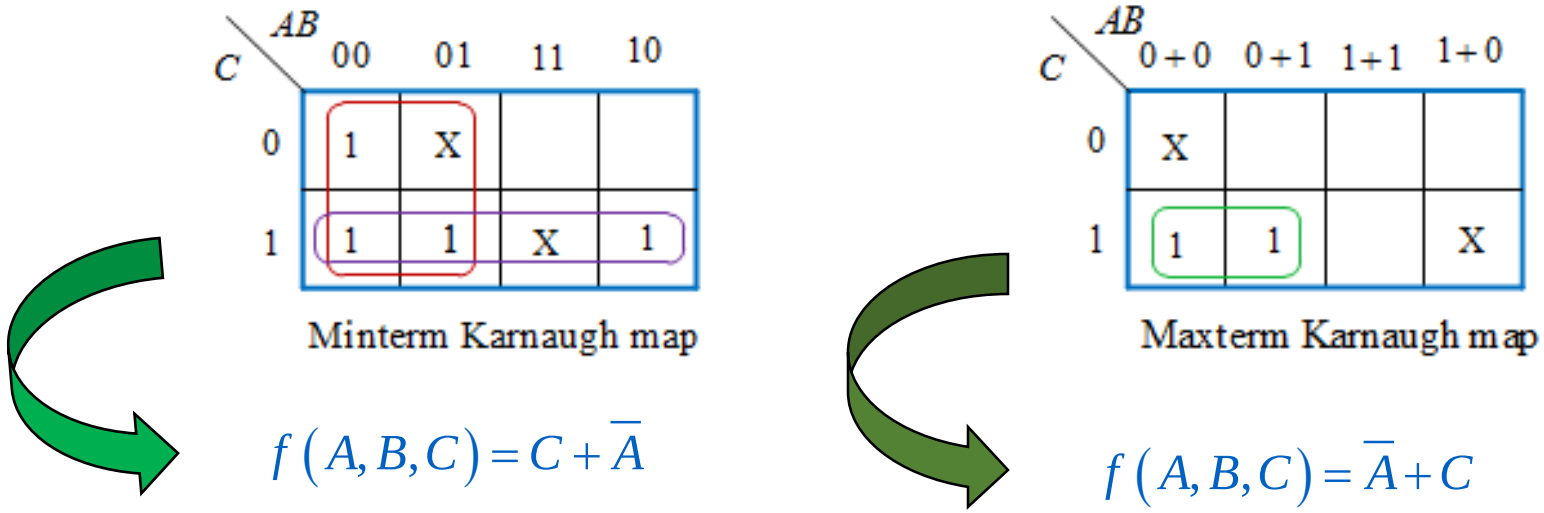
$$f(A,B,C)=\overline{A}\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot\overline{B}\cdot C+\overline{A}\cdot B\cdot C+A\cdot\overline{B}\cdot C$$

and

$$f(A,B,C)=\left(\overline{A}+B+C\right)\cdot\left(\overline{A}+\overline{B}+C\right)$$

✓ Solution to Example 7

- The ‘don’t care’ input combinations for the SOP Boolean expression are $\overline{A}\cdot B\cdot\overline{C}$ and $A\cdot B\cdot C$
- The ‘don’t care’ input combinations for the POS Boolean expression are $\left(A+\overline{B}+C\right)\cdot\left(\overline{A}+\overline{B}+\overline{C}\right)$.
- The Karnaugh maps are as shown below.



COMBINATIONAL LOGIC

- A combinational circuit is one where the output at any time depends only on the present combination of inputs at that point of time with total disregard to the past state of the inputs.
- The logic gate is the most basic building block of combinational logic.
  - ✓The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions.

The different steps involved in the design of a combinational logic circuit are as follows:

1. Statement of the problem.
2. Identification of input and output variables.
3. Expressing the relationship between the input and output variables.
4. Construction of a truth table to meet input–output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. Minimization of Boolean expressions.
7. Implementation of minimized Boolean expressions.

- There are various simplification techniques available for minimizing Boolean expressions, which have been discussed earlier. These include the use of theorems and identities, Karnaugh mapping e.t.c.
- The following guidelines should be followed while choosing the preferred form for hardware implementation:
    1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
    2. There should be a minimum number of interconnections, and the propagation time should be the shortest.
    3. Limitation on the driving capability of the gates should not be ignored.

Arithmetic Circuits

- Lets combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers.
- Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction

Binary Adder

- Recall binary addition:
  1. 0 + 0 = 0,
  2. 0 + 1 = 1,
  3. 1 + 0 = 1,
  4. 1 + 1 = 10 (This should be read as “Equal to zero, carry one”)
  5. 1 + 1 + 1 = 11 (This should be read as “Equal to one, carry one”)

➢ Half adder: is a logic circuit that adds two bits. From the rules of binary addition, there will always be a SUM and a CARRY digit. Below is a truth table obtained when adding two numbers A and B:

The half-adder equations are

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

$$\text{CARRY } C = A.B$$

A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

How to implement the half-adder circuit

- ✓ Look at the results in the CARRY column and you see A AND B.
- ✓ Look at the results in the SUM column and you see A XOR B.

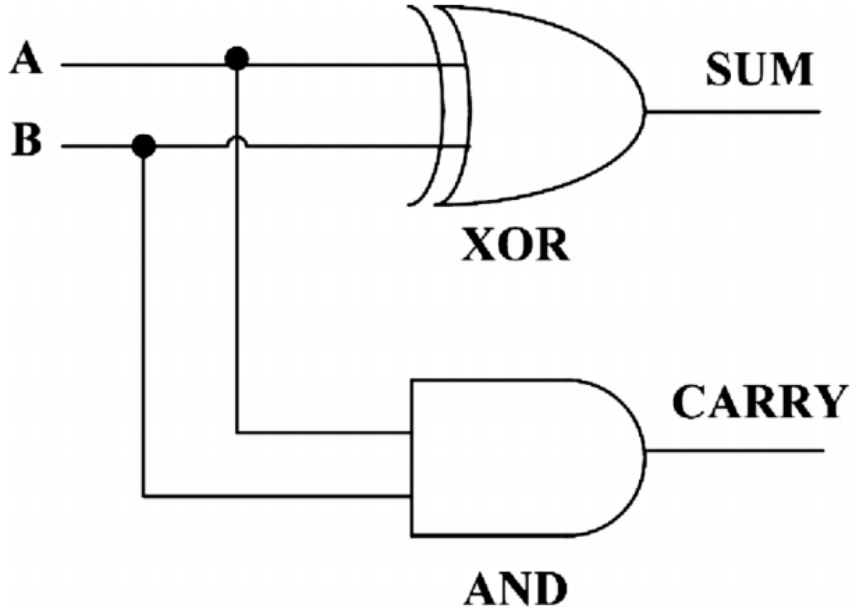


Figure 4.15: Implementation of a half-adder

Full Adder

A full adder is a circuit that adds 3 bits. It has two outputs, Sum (S) and Carry (C_o): It is possible to have variants in logic implementation but basically it is a combination of two halfadders.

A	B	C_I	C_O	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table for a full adder

The full-adder equations are

$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(\overline{\overline{A}.B + A.\overline{B}})$$

$$C_{out} = A.B + C_{in}.(\overline{A}.B + A.\overline{B})$$

A 1011
+B + 1110
11001

C 1110
A 01011
B 01110
Sum 11001
Carry 01110

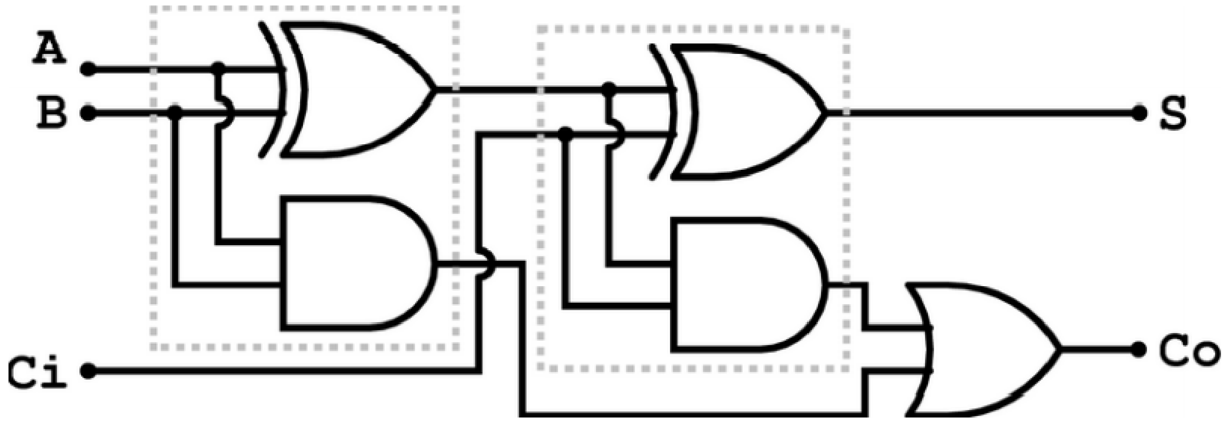


Figure 4.16: Implentation of a full adder

Binary Subtractor

- Half-subtractor: is a combinational circuit that can be used to subtract one binary digit from another to produce a **DIFFERENCE** output and a **BORROW** output.
  - ✓The **BORROW** output here specifies whether a '**1**' has been borrowed to perform the subtraction.

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth table of a Half-subtractor

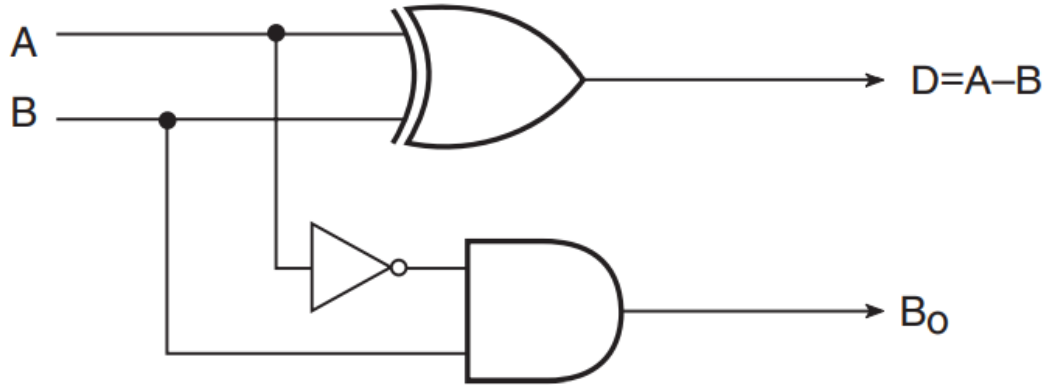


Figure 4.17: Implementation of a Half-subtractor

- The equations used for the half-subtractore is

$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

- **Full-subtractor**: performs subtraction operation on 2 bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.
  - ✓As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}.
  - ✓There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o.
- The equations for the full-subtractor are given as

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$

Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth table for a full-subtractor

- Hence the implementation of a full-subtractor is as follows:

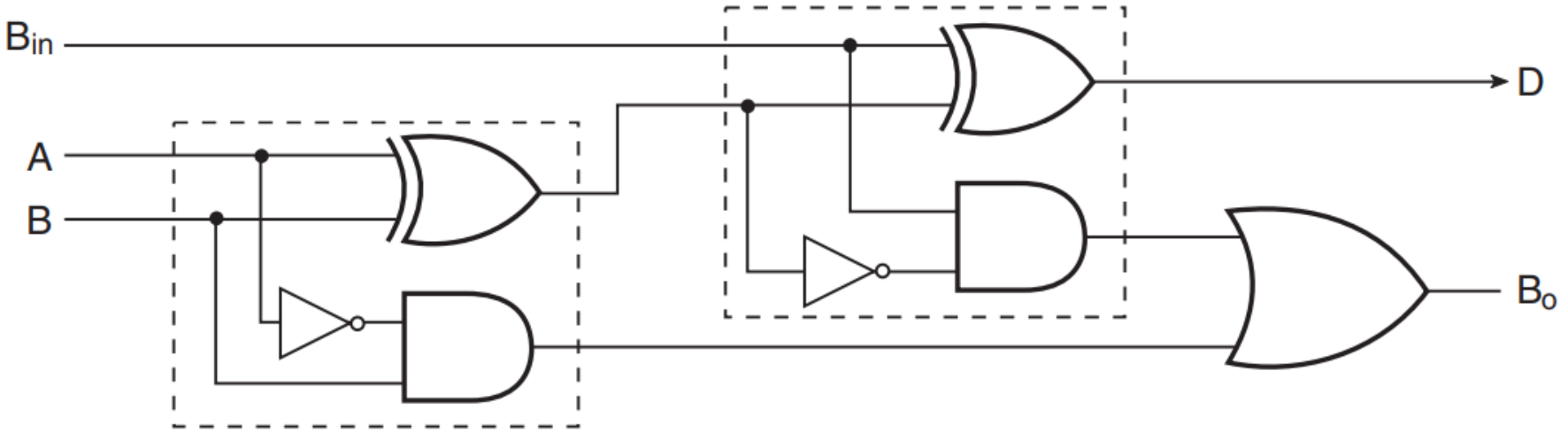


Figure 4.18: Implementation of a full-subtractor

Controlled Inverter

- A controlled inverter is needed when an adder is to be used as a subtractor. As outlined earlier, subtraction is nothing but addition of the 2's complement of the subtrahend to the minuend.
- Thus, the first step towards practical implementation of a subtractor is to determine the 2's complement of the subtrahend.
  - ✓A controlled inverter is used to find 1's complement.
- A one-bit controlled inverter is nothing but a two-input EX-OR gate with one of its inputs treated as a control input.
  - ✓When the control input is LOW, the input bit is passed as such to the output. (Recall the truth table of an EX-OR gate.)
  - ✓When the control input is HIGH, the input bit gets complemented at the output. Figure 4.20 shows an eight-bit controlled inverter of this type.

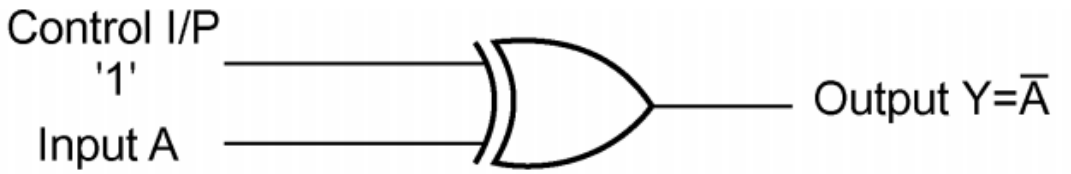


Figure 4.19: One-bit controlled inverter

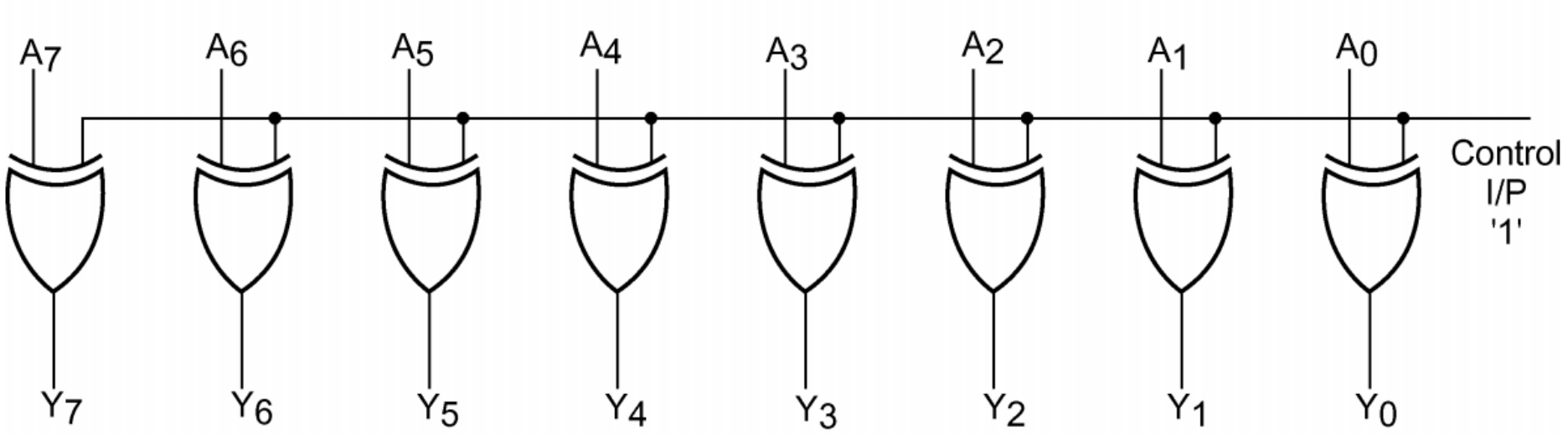


Figure 4.20: Eight-bit controlled inverter

Adder–Subtractor

- Subtraction of two binary numbers can be accomplished by adding 2's complement of the subtrahend to the minuend and disregarding the final carry, if any.
  - ✓ If the MSB bit in the result of addition is 0, then the result of addition is correct.
  - ✓ Otherwise, the answer is negative.

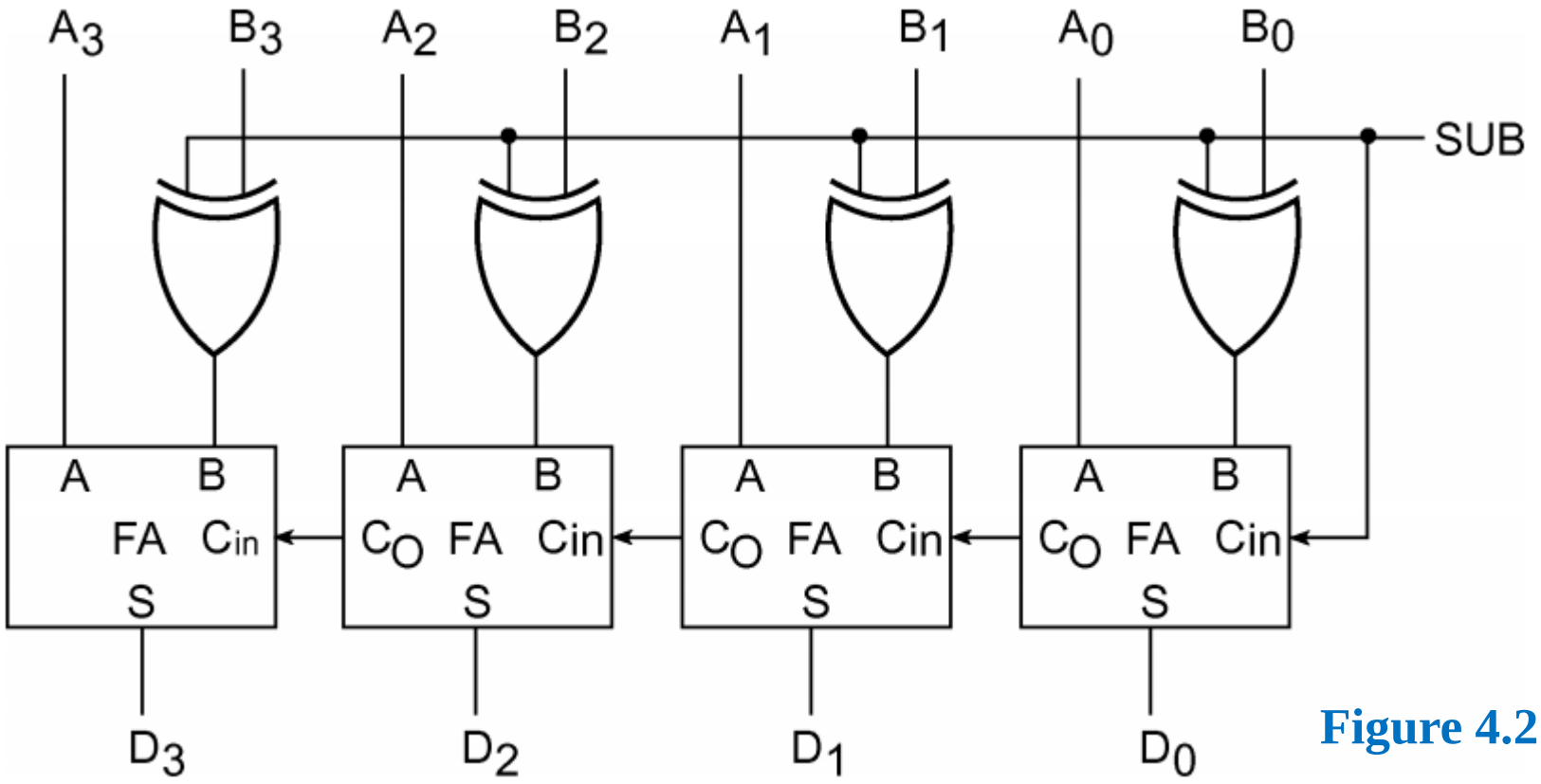


Figure 4.21: Four-bit adder-subtractor

END OF LECTURE!