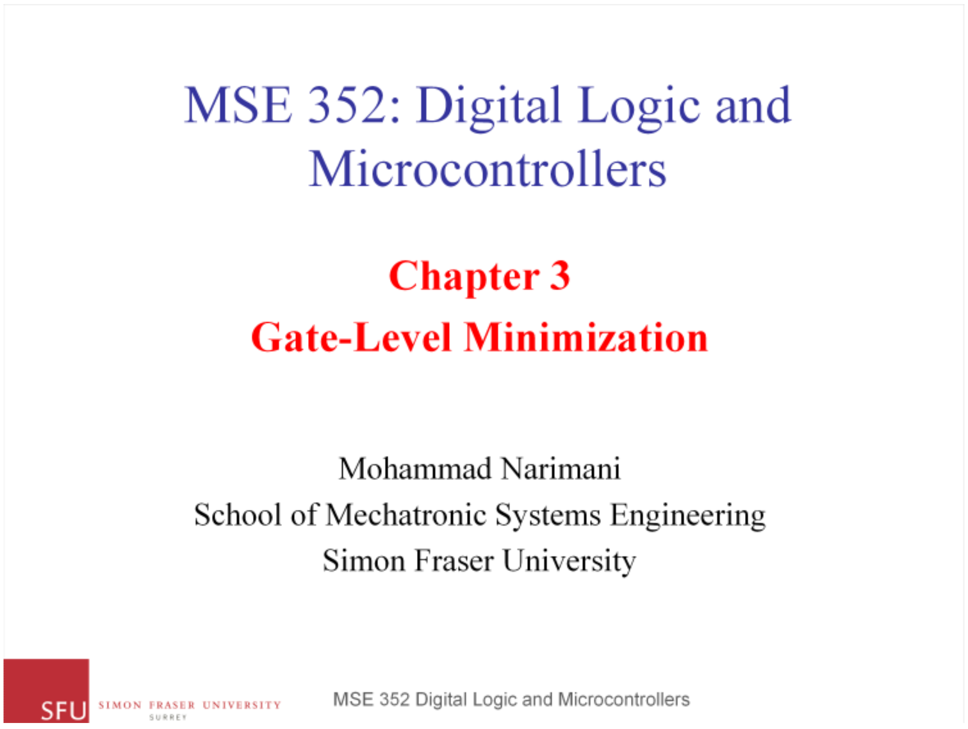
Chapter 3

September 18, 2018 5:45 PM



MSE 352: Digital Logic and Microcontrollers

Chapter 3

Gate-Level Minimization

Mohammad Narimani
School of Mechatronic Systems Engineering
Simon Fraser University

SFU SIMON FRASER UNIVERSITY SURREY

MSE 352 Digital Logic and Microcontrollers

Karnaugh map (K-map)

September 18, 2018 5:48 PM

- **What is Gate-Level Minimization?**
  - The simplest algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term.
  - The simplest expression is not unique.

- **Why Gate-Level Minimization?**
  - The number of terms in a Boolean function is the representation of the number of required gates.
  - The number of literals in each term is the representation of the number of required inputs in a gate.

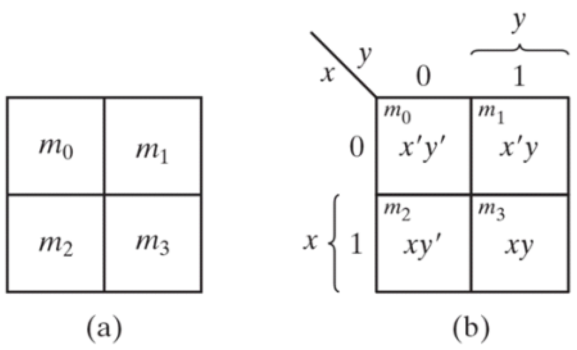
    Therefore, with Gate-level minimization one can make sure that an algebraic expression can be implemented with lowest complexity and cost.

- **What is Karnaugh Map (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.
  - A Boolean function can be expressed as a sum of minterms, recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.

- **Why Karnaugh Map?**
  - Boolean expressions may be simplified by algebraic means as discussed in Chapter 2. However, this procedure of minimization is awkward because **it lacks specific rules to predict each succeeding step** in the manipulative process.
  - The **Karnaugh map (K-map)** method presented here provides a **simple, straightforward procedure** for minimizing Boolean functions. This method may be regarded as a pictorial form of a truth table.

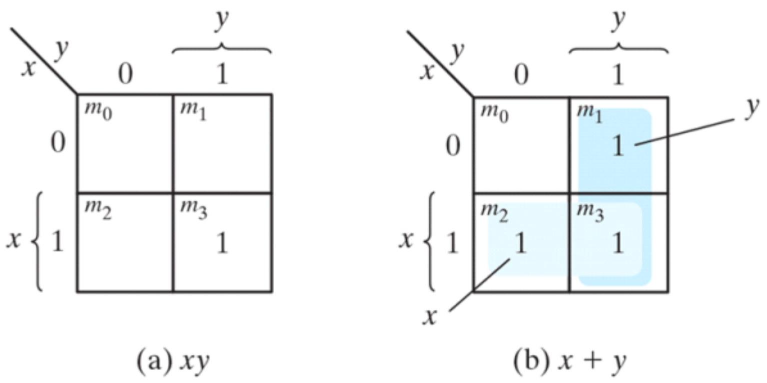
Two-variable K-map

September 18, 2018 6:35 PM



Two-variable K-map

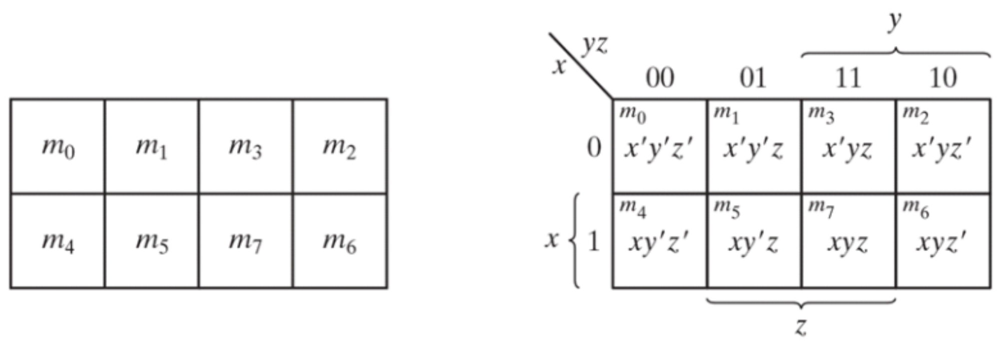
Representation of functions in the K-map:



Three-variable K-map

September 18, 2018 6:41 PM

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



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

(b)

Three-variable K-map

- Any two adjacent squares in the map differ by only one variable

$$m_5 + m_7 =$$

Example: Simplify the Boolean function

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

Three-variable K-map - Cont'd

September 18, 2018 7:00 PM

Example: Simplify the Boolean function

$$F(x, y, z) = \sum(3,4,6,7)$$

Example: Simplify the Boolean function

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

Three-variable K-map - Cont'd

September 18, 2018 7:05 PM

Example: For the Boolean function

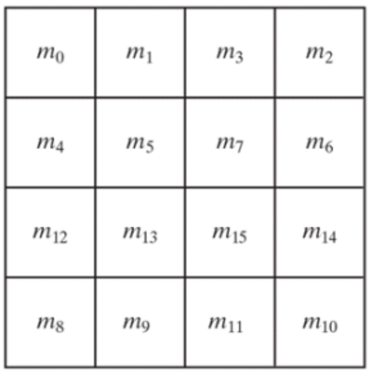
$$F(A,B,C) = A'C + A'B + AB'C + BC$$

(a) Express this function as a sum of minterms.

(b) Find the minimal sum-of-products expression.

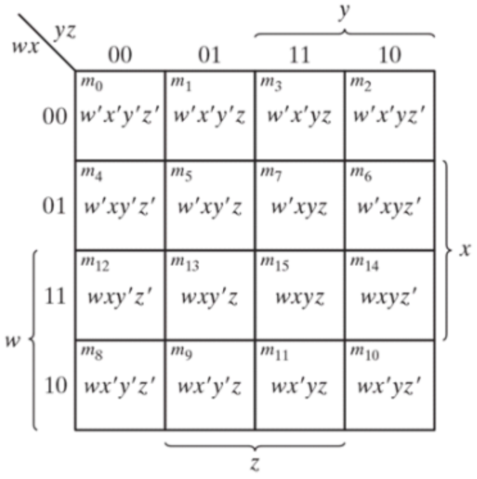
Four-variable K-map

September 20, 2018 9:46 AM



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}

(a)



$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

(b)

Example: Simplify the Boolean function

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

Four-variable K-map - Cont'd

September 20, 2018 10:16 AM

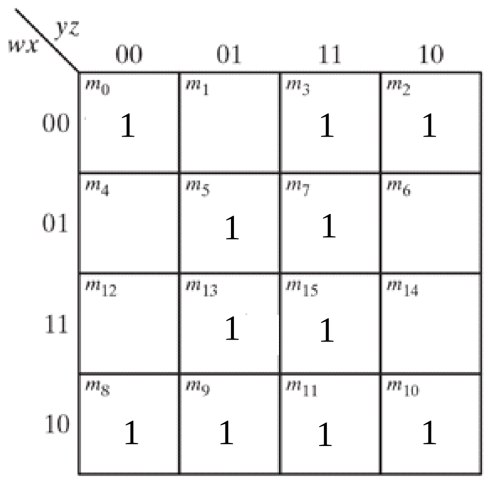
Example: Simplify the Boolean function

$$F\ (A, B, C, D) = A'B'C' + B'CD' + ABCD' + AB'C'$$

Prime Implicant

September 20, 2018 10:23 AM

When K-map is used for function simplification using adjacent squares in a map, we must ensure that the number of terms in the expression is minimized, that is there are no redundant terms (i.e., minterms already covered by other terms).



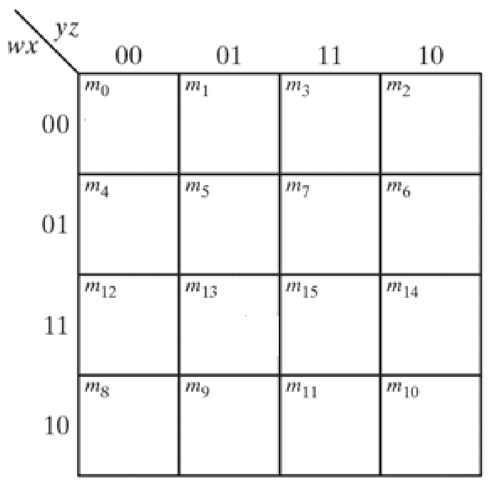
Prime Implicant and Essential Prime Implicant:

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

If a cell is covered by only one prime implicant, that prime implicant is said to be **essential**.

Consider the above map where its Boolean function is:

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



POS simplification using K-map

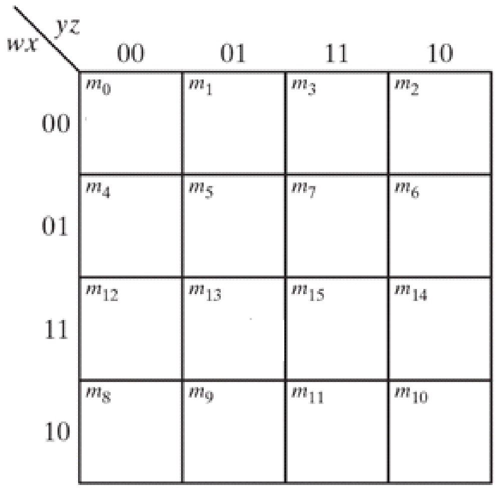
September 20, 2018 11:07 AM

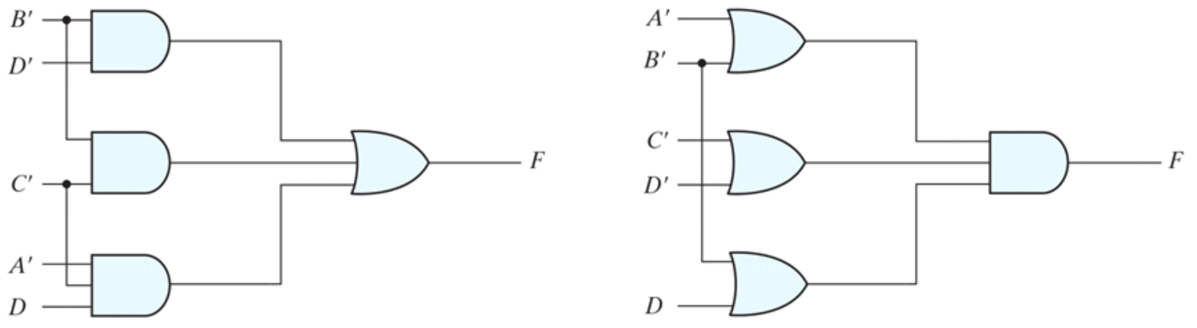
Simplify the following Boolean function into:

(a) sum-of-products form

(b) product-of-sums form

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





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

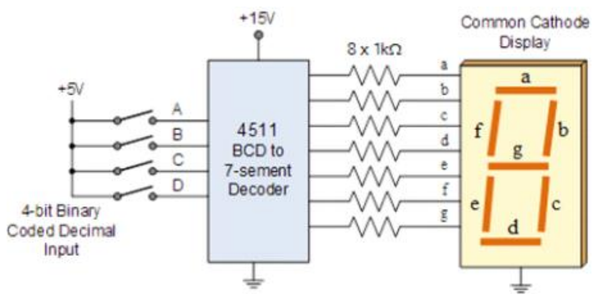
Gate implementations of the function of the above Example in SOP and POS

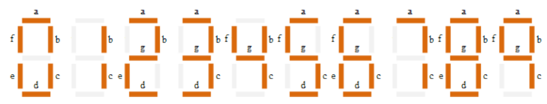
Don't Care Conditions

September 20, 2018 11:16 AM

In some applications the Boolean function to be simplified is not specified for certain combinations of the variables.

Example: A binary to BCD converter circuit





- There are six combinations (minterms) that are not used when convert the four-bit binary code to the decimal digits.
- A **don't-care minterm** (condition) is a combination of variables whose logical value is not specified.
- To distinguish the don't-care condition an X is used inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to the function for the particular minterm.

Example: Simplify the Boolean function

$F\ (w,x,y,z) = \sum(1,3,7,11,15)$
$d\ (w,x,y,z)\ = \sum(0,2,5)$

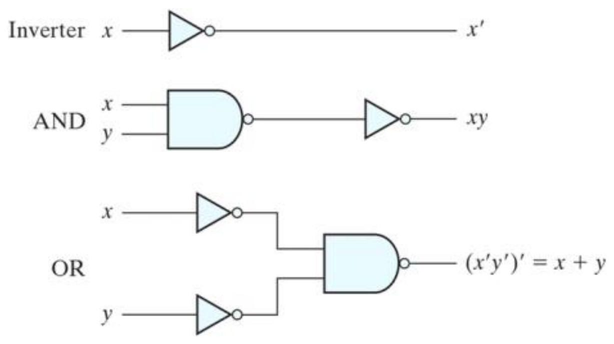
NAND and NOR Implementation

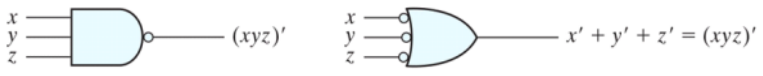
September 20, 2018 6:37 PM

- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.
- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.

NAND Circuits:

The NAND gate is said to be a **universal gate** because any logic circuit can be implemented with it.



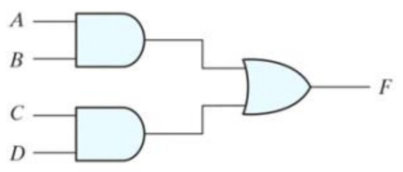


(a) AND-invert (b) Invert-OR

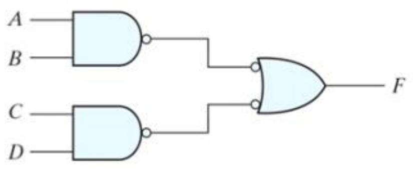
Two-Level Implementation with NAND gates:

Consider function $F = AB + CD$

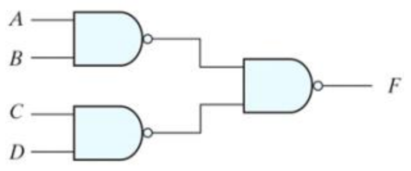
This function can be implemented by NAND gates as follows:



(a)



(b)



(c)

NAND and NOR Implementation - Cont'd

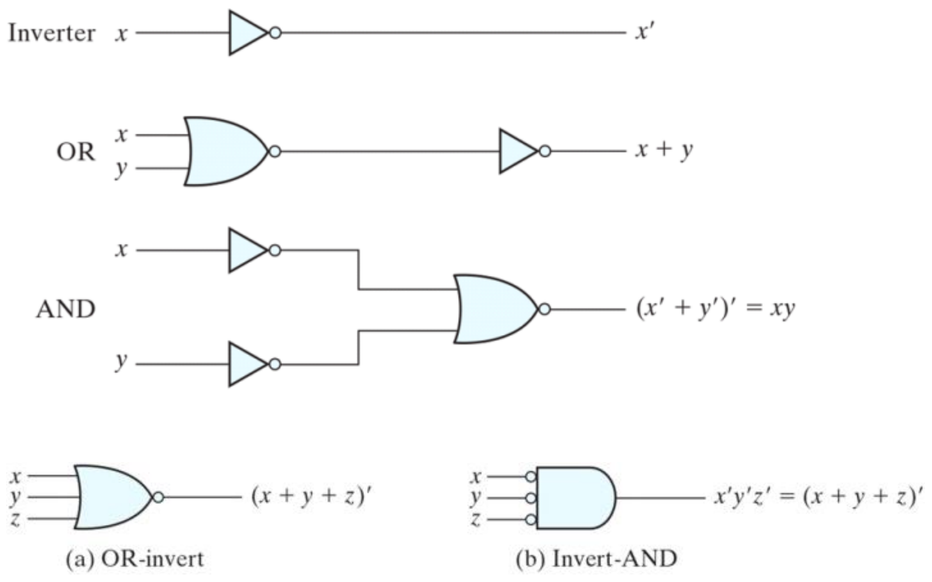
September 20, 2018 6:55 PM

Example: Implement the following Boolean function with NAND gates:

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

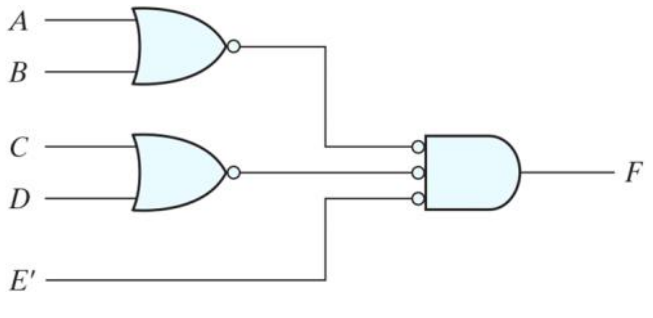
NOR Implementation:

The NOR operation is the dual of the NAND operation.



The following figure shows the NOR implementation of a function expressed as a product of sums:

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



EXCLUSIVE-OR Function

September 20, 2018 7:08 PM

Exclusive-OR (XOR): denoted by the symbol $\oplus$, is a logical operation that performs the following Boolean operation:

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

The XOR is equal to 1 if only x is equal to 1 or if only y is equal to 1 (i.e., x and y differ in value), but not when both are equal to 1 or when both are equal to 0.

Exclusive-NOR (XNOR): denoted by the symbol $\odot$, also known as equivalence, performs the following Boolean operation:

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

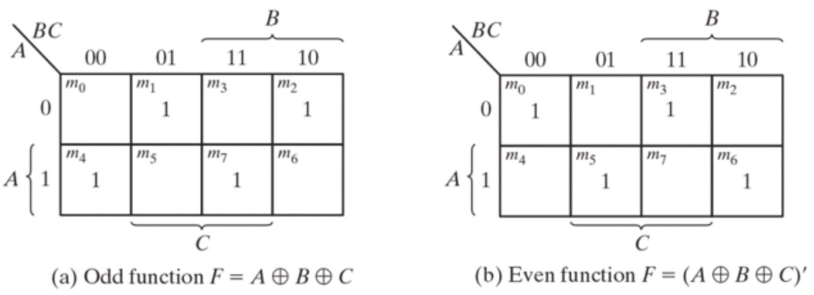
The XNOR is equal to 1 if both x and y are equal to 1 or if both are equal to 0.

Properties of XOR:

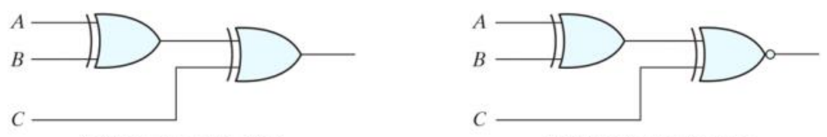
$$x \oplus y = y \oplus x$$

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$$

K-map for a three-variable exclusive-OR function:

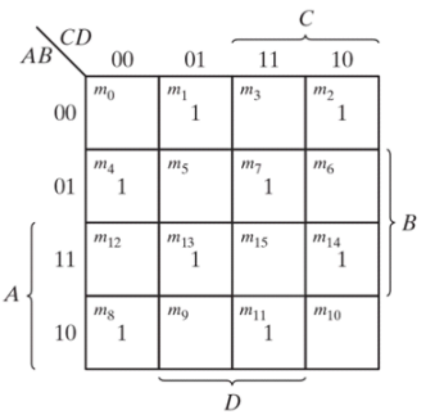


(a) Odd function $F = A \oplus B \oplus C$ (b) Even function $F = (A \oplus B \oplus C)'$



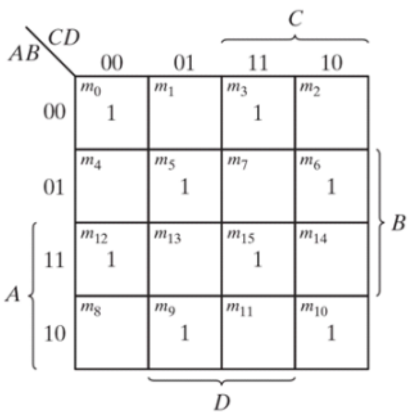
(a) 3-input odd function (b) 3-input even function

K-map for a four-variable exclusive-OR function:



CD
AB
C
00 01 11 10
B
A
D
m_0 m_1 1 m_3 m_2 1
m_4 1 m_5 m_7 1 m_6
m_{12} m_{13} 1 m_{15} m_{14} 1
m_8 1 m_9 m_{11} 1 m_{10}

(a) Odd function $F = A \oplus B \oplus C \oplus D$



CD
AB
C
00 01 11 10
B
A
D
m_0 1 m_1 m_3 1 m_2
m_4 m_5 1 m_7 m_6 1
m_{12} 1 m_{13} m_{15} 1 m_{14}
m_8 m_9 1 m_{11} m_{10} 1

(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

The following identities apply to the XOR operation:

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

Parity Generation and Checking

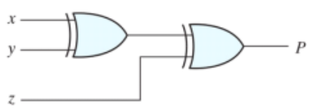
September 20, 2018 10:50 PM

Recall:
Parity Generator: A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The circuit that generates the parity bit in the transmitter is called a parity generator.

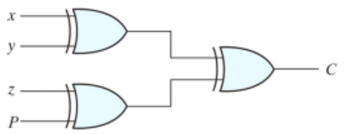
Parity Checker: The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that checks the parity in the receiver is called a parity checker.

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



(b) 4-bit even-parity checker

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

$$C = x \oplus y \oplus z \oplus P$$

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