18ECC103J
DIGITAL ELECTRONIC PRINCIPLES

Unit-1

Course Articulation Matrix

18ECC103J - DIGITAL ELECTRONIC PRINCIPLES	Program Outcomes (POs)												PSO		
	Graduate Attributes														
Course Learning Outcomes (CLOs)	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
Simplify Boolean expressions; carry out arithmetic operations with binary numbers; apply parity method for error detection and correction.	H	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Explain the operational characteristics / properties of digital ICs; implement gates as well as other types of IC devices using two major IC technologies, TTL and CMOS.	H	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Identify eight basic types of fixed-function combinational logic functions and demonstrate how the devices / circuits can be used in building complete digital systems such as computers.		M	H	-	H	-	-	-	-	-	-	-	-	-	-
Analyze and design Mealy and Moore models of sequential circuits using several types of flip-flops.		M	H	-	H	-	-	-	-	-	-	-	-	-	-
Implement multiple output combinational logic circuits using PLDs; Explain the operation of a CPLD and FPGA.	-	M	H	-	L	-	-	-	-	-	-	-	-	-	-
Solve specific design problem, which after completion will be verified using modern engineering tools such as PSPICE / Logisim	-	M	H	-	H	-	-	-	H	-	-	-	M	-	L

Program Outcomes (PO)

PO1- Engineering knowledge
Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an Engineering specialization to the solution of complex Engineering problems.

PO2- Problem analysis
Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3- Design/development of solutions
Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4-Conduct investigations of complex problems
Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5-Modern tool usage
Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6-The engineer and society
Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7-Environment and sustainability
Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8-Ethics
Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9-Individual and team work
Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10- Communication
Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11-Project management and finance
Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12- Life-long learning
Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Index

Unit-I: Binary Codes, Digital Arithmetic and Simplification of Boolean Functions

CLO1: Simplify Boolean expressions; carry out arithmetic operations with binary numbers; apply parity method for error detection and correction.

Learning Unit/ Module	Session	Description of Topic (Theory)	No. of Contact hours	CO	PO	BL	Reference
Unit-I: Binary Codes, Digital Arithmetic and Simplification of Boolean Functions	1.	Binary Codes, Digital Arithmetic and Simplification of Boolean Functions , Error detecting codes	1	1	1	1,2	[1] chapter 1
	1.	Error correcting code , Hamming Code, Arithmetic number representation , Binary arithmetic	2	1	1	1,2	[1] chapter 7
	1.	Hexadecimal arithmetic , BCD arithmetic simplification.	2	1	1	1,2	[1] chapter 2
	1.	Minimization of Boolean Functions: Algebraic simplification Problems on Algebraic simplification , Karnaugh map simplification	1	1	1	1,2	[1] chapter 3
	1.	Problems on Karnaugh map simplification	1	1	1	1,2	[1] chapter 3
	1.	Quine Mc Cluskey or Tabulation method,Problems on Quine McCluskey or Tabulation method.	2	1	1	1,2	[1] chapter 3

Reference: Morris Mano M, Michael D. Ciletti, Digital Design with an Introduction to the Verilog HDL, 5th ed., Pearson Education, 2014 .

Common Number Systems

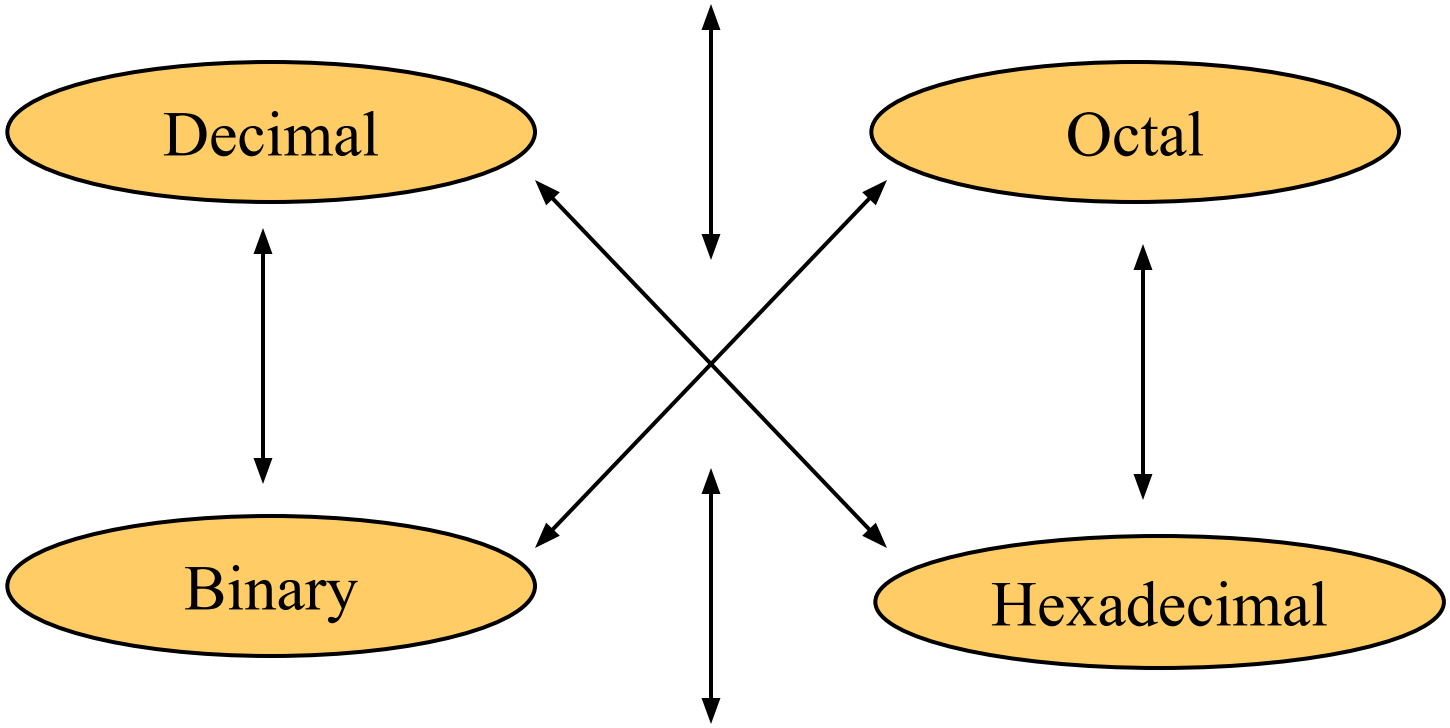
System	Base	Symbols
Decimal	10	0, 1, … 9
Binary	2	0, 1
Octal	8	0, 1, … 7
Hexa-decimal	16	0, 1, … 9, A, B, … F

Number System Quantities

Decimal	Binary	Octal	Hexa-decimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Decimal	Binary	Octal	Hexa-decimal
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17

Number System Conversion



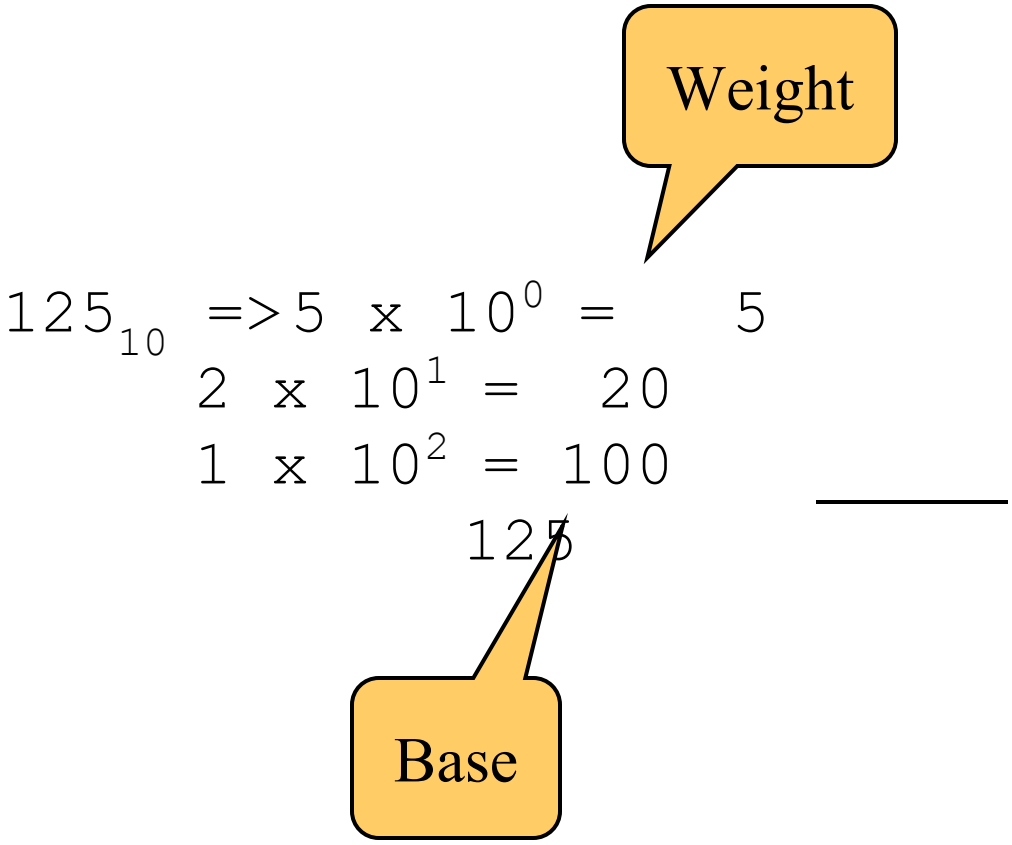
Example

Decimal – Binary- Octal- Hexa decimal

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base

Decimal to Decimal



Binary to Decimal

Technique

Multiply each bit by 2^n, where *n* is the "weight" of the bit

The weight is the position of the bit, starting from 0 on the right

Add the results

Example

Bit "0"

$$101011_2 \Rightarrow 1 \times 2^0 = 1$$
$$1 \times 2^1 = 2$$
$$0 \times 2^2 = 0$$
$$1 \times 2^3 = 8$$
$$0 \times 2^4 = 0$$
$$1 \times 2^5 = 32$$

$$43_{10}$$

Octal to Decimal

Technique

- Multiply each bit by 8^n, where *n* is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

Example

$$724_8 \Rightarrow 4 \times 8^0 = 4$$
$$2 \times 8^1 = 16$$
$$7 \times 8^2 = 448$$
$$468_{10}$$

Hexadecimal to Decimal

Technique

- Multiply each bit by 16^n, where *n* is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

Example

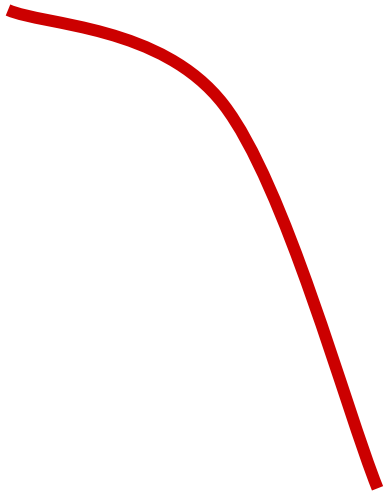
$$
\begin{aligned}
ABC_{16} \Rightarrow C \times 16^0 &= 12 \times 1 = 12 \\
B \times 16^1 &= 11 \times 16 = 176 \\
A \times 16^2 &= 10 \times 256 = 2560 \\
&\hline \\
&2748_{10}
\end{aligned}
$$

Decimal to Binary

Example

$125_{10} = ?_2$

2 | 125		
2 | 62	1	
2 | 31	0	
2 | 15	1	
2 | 7	1	
2 | 3	1	
2 | 1	1	
0	1	



$125_{10} = 1111101_2$

Technique

- Divide by two, keep track of the remainder
- First remainder is bit 0 (LSB, least-significant bit)
- Second remainder is bit 1
- Etc.

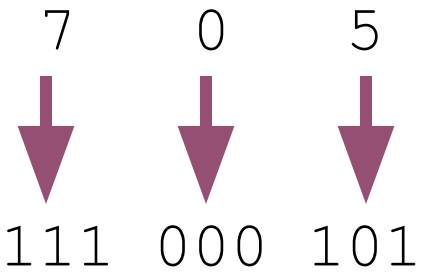
Octal to Binary

Technique

- Convert each octal digit to a 3-bit equivalent binary representation

Example

$705_8 = ?_2$



$705_8 = 111000101_2$

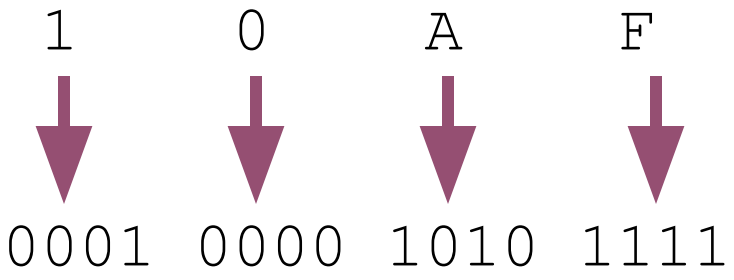
Hexadecimal to Binary

Technique

- Convert each hexadecimal digit to a 4-bit equivalent binary representation

Example

$10AF_{16} = ?_{2}$



$10AF_{16} = 0001000010101111_{2}$

Decimal to Octal

Technique

- Divide by 8
- Keep track of the remainder

Example

$1234_{10} = ?_{8}$

8 |	1234	
8 |	154	2
8 |	19	2
8 |	2	3
	0	2

$1234_{10} = 2322_{8}$

Decimal to Hexadecimal

Technique

- Divide by 16
- Keep track of the remainder

Example

$1234_{10} = ?_{16}$

16 | 1234		
16 | 77	2	
16 | 4	13	= D
0	4	

$1234_{10} = 4D2_{16}$

Binary to Octal

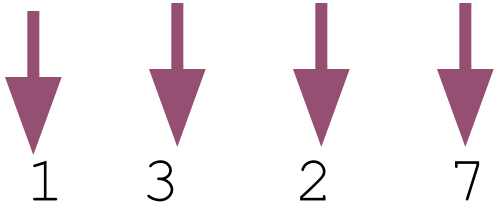
Technique

- Group bits in threes, starting on right
- Convert to octal digits

Example

$1011010111_2 = ?_8$

1 011 010 111



1 3 2 7

$1011010111_2 = 1327_8$

Binary to Hexadecimal

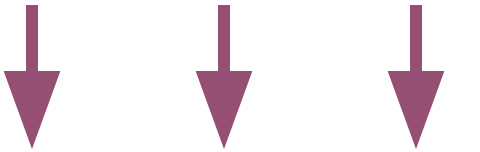
Technique

- Group bits in fours, starting on right
- Convert to hexadecimal digits

Example

$1010111011_2 = ?_{16}$

10 1011 1011



2 B B

$1010111011_2 = 2BB_{16}$

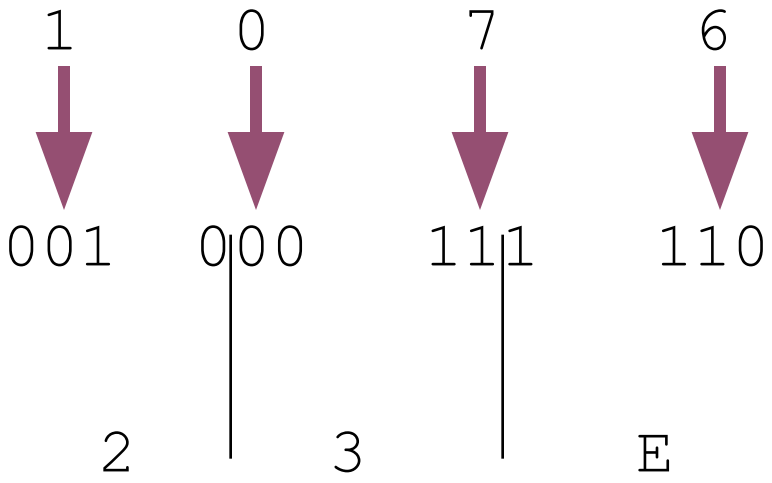
Octal to Hexadecimal

Technique

- Use binary as an intermediary

Example

$1076_8 = ?_{16}$



$1076_8 = 23E_{16}$

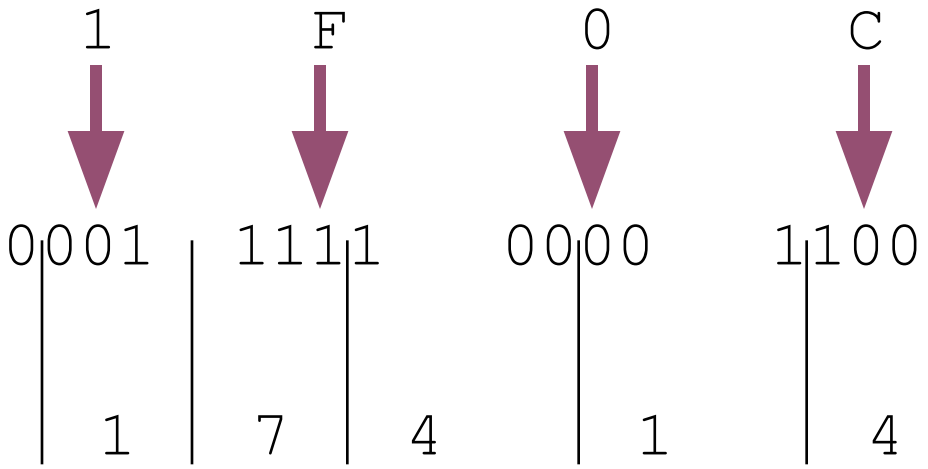
Hexadecimal to Octal

Technique

- Use binary as an intermediary

Example

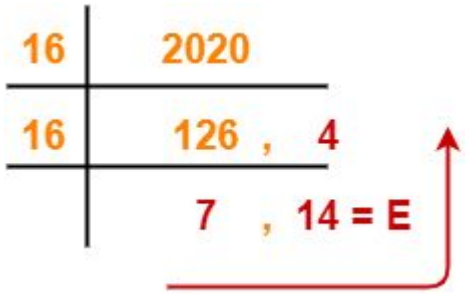
$1F0C_{16} = ?_{8}$



$1F0C_{16} = 17414_{8}$

Fractional bits

Convert the following numbers from base 10 to base 16-
$(2020.65625)_{10}$



Real Part-

- The real part is $(2020)_{10}$
- We convert the real part from base 10 to base 16 using division method same as above.

So, $(2020)_{10} = (7E4)_{16}$

For Fractional Part-

- The fractional part is $(0.65625)_{10}$
- We convert the fractional part from base 10 to base 16 using multiplication method.

	Real part	Fractional Part
0.65625 x 16	10 = A	0.5
0.5 x 16	8	0.0

From here, $(0.65625)_{10} = (0.A8)_{8}$

Combining the result of real and fractional parts, we have-
$(2020.65625)_{10} = (7E4.A8)_{16}$

Exercise

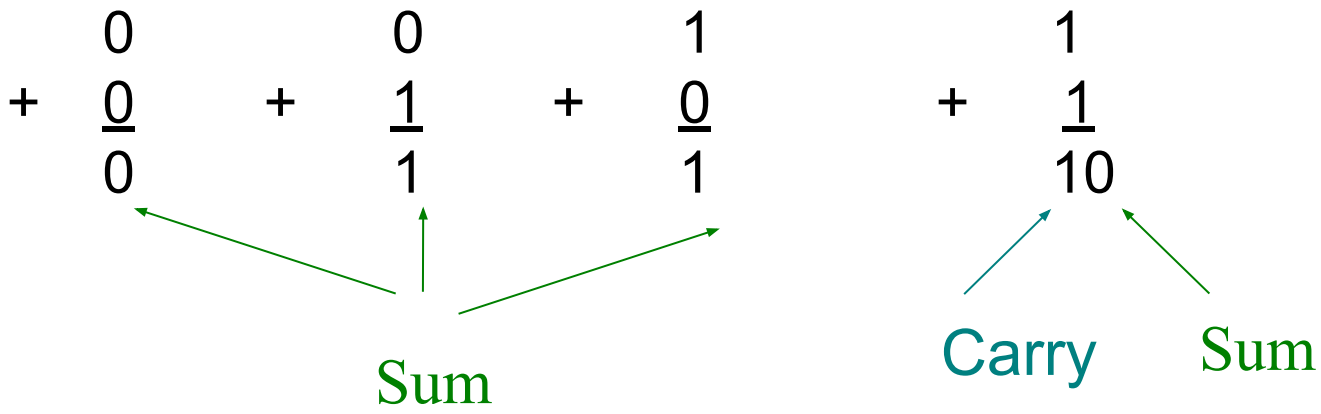
1. Convert $(1056)_{16}$ to $(?)_{8}$
2. Convert $(11672)_{8}$ to $(?)_{16}$
3. Convert the following numbers from base 10 to base 16- $(172.983)_{10}$
4. Convert the following numbers from base 10 to base 8-$(1032.6875)_{10}$
5. Convert the following numbers from base 10 to base 2-$(172.878)_{10}$

Answers

1. $(1056)_{16} = (10126)_{8}$
2. $(11672)_{8} = (13BA)_{16}$
3. $(172.983)_{10} = (AC.FBA5)_{16}$
4. $(1032.6875)_{10} = (2010.54)_{8}$
5. $(172.878)_{10} = (10101100.1110)_{2}$

Basic Binary Arithmetic

Binary Addition



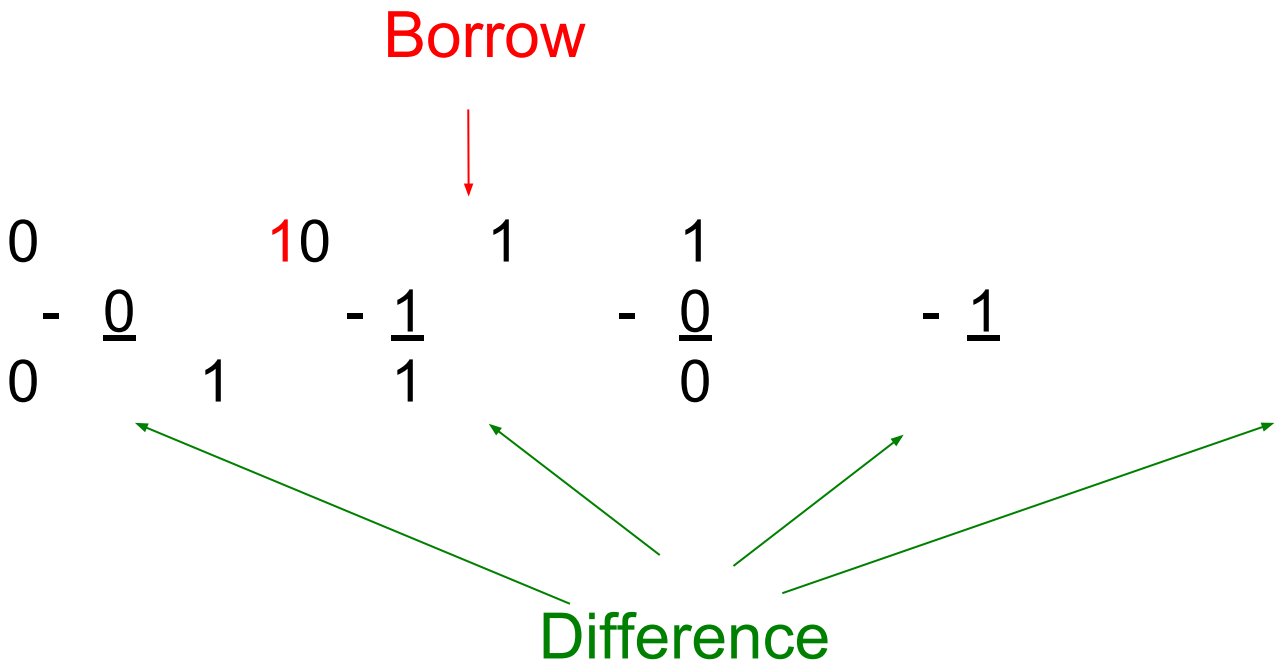
Examples:

```
 01011011
+01110010
 11001101
```

```
 10110101
+01101100
 ??
```

```
 00111100
+10101010
 ??
```

Binary Subtraction



Examples:

01110101	10110001	00111100
-00110010	-01101100	-10101100
01000011	??	??

Basic Binary Arithmetic

Single-bit Addition

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Single-bit Subtraction

x	y	Difference d
0	0	0
0	1	1
1	0	1
1	1	0

Subtraction

- ✔ Subtraction of a number from another can be accomplished by **adding the complement** of the subtrahend to the minuend.
- ✔ **1's complement method & 2's complement method**
- ✔ Subtraction of binary numbers using the **1's complement method** allows subtraction only by addition.
- ✔ The 1's complement of a binary number can be obtained by changing all 1s to 0s and all 0s and 1s.
- ✔ The 2's complement of a binary number can be obtained by adding 1 to its 1's complement.

1’s complement method

- To subtract a smaller number from a larger number, the

  1’s complement method is as follows

1. Determine the 1’s complement of the smaller number.

2. Add this to the larger number

3. Remove the carry and add it to the result. This carry is called **end-around-carry**

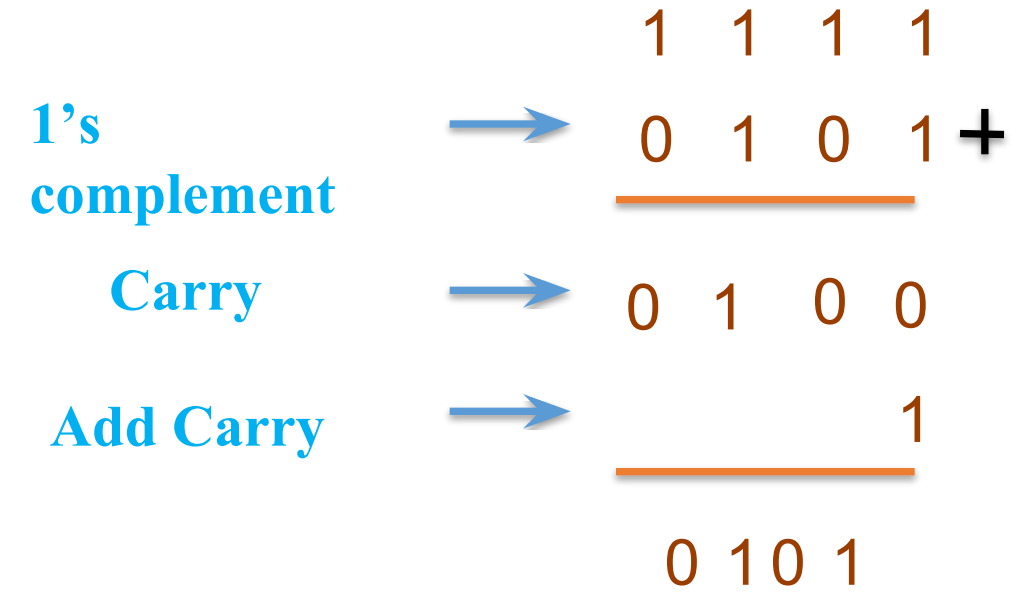
Example of 1's complement

Subtract $(1010)_2$ from $(1111)_2$

Direct Subtraction

```
  1  1  1  1
- 1  0  1  0
-------------
  0  1  0  1
-------------
```

1's complement method



Subtraction Steps

Subtraction of a large number a smaller one by the 1's complement method involves the following steps

1. **Determine the 1's complement of a large number**
2. **Add this to the smaller number**
3. **The answer is the 1's complement of the result and is opposite in sign. There is no carry.**

2's Complement Subtraction

Steps

Subtraction of a smaller number from a larger one by the 2's complement method involves the following steps

1. Determine the 2's complement of the smaller number
2. Add this to the larger number
3. Omit the carry (there is always a carry in this case)

Example of 2's complement

Subtract $(1010)_2$ from $(1111)_2$

Direct Subtraction

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ -\ 1\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1 \end{array}$$

2's complement Carry

2's complement method

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 0\ 1\ 0\ 1 \end{array}$$

1 ← Omit carry

$$\begin{array}{r} 0101 \\ \quad\ 1 \\ \hline 01\ 10 \end{array}$$

Subtraction Steps

- The carry is discarded. Thus answer is **$(0101)_2$**

- The 2's complement method for subtraction of a large number from a smaller one is as follows.

1. **Determine the 2's complement of the larger number**
2. **Add the 2's complement to the smaller number**
3. **There is no carry . The result is in 2's complement form is negative**
4. **To get an answer in true form, take the 2's complement and change the sign.**

Comparison between 1's and 2's complements

1's Complement	2's Complement
It can be easily obtained using an Inverter	It has to be arrived at by first obtaining the 1's complement and then adding one (1) to it
It requires two operations	Only one arithmetic operation is required
It is often used in logical manipulations for inversion operation	It is used only for arithmetic applications

Binary Division

Division

The division process is like as division of decimal numbers.

```
101) 1001 (1
     - 101
     -----
       100
```

Quotient 1 and remainder 100.

Hexadecimal Arithmetic

1. Hexadecimal Numbers Addition

•Write two hexadecimal numbers one after another in two different lines
•Begin adding from the rightmost digits.
•If the digit is in the form of an alphabet then convert it to the respective decimal number to make the process easy
•Add those digits and convert the sum to the hexadecimal
•If you got the carry, then represent it on the top of the first number next digit and result on the bottom of the second number added digit.
•Continue the process until you left nothing on the left side.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	7	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

2. Hexadecimal Numbers Subtraction

•Write two hexadecimal numbers in different lines
•Subtraction starts from the rightmost digits of the numbers.
•Convert the alphabets into decimals and subtract two digits and again convert the difference value as hexadecimal.
•In case the first number digit is smaller than the second number digit, then barrow from the left side digit.
•The borrowed value is always 16 as its base is 16. Then add borrowed value and first number digit and subtract.
•Don't forget to mention the borrowed value on the top of the first number digit.
•After borrowing, the left side digit decreased by 1.
•Repeat the process till you have nothing remaining on the left side.

1. 926 + 1A2 = AC8

2. 938 – 1A2 = 796.

3. Given expression is $(1AB2)_{16} + (2198)_{16}$
From the table,
2 + 8 = A
B + 9 = 4 and 1 is carry
1 + A + 1 = C
1 + 2 = 3
Therefore, $(1AB2)_{16} + (2198)_{16} = 3C4A$

4. Find subtraction of $(B84F)_{16}$ and $(A53)_{16}$.

Solution:

F means 15. F – 3 = 15 – 3 = 12 = C

4 + 16 = 20 – 5 = 15 = F

8 – 1 = 7

7 + 16 = 23 – A = 23 – 10 = 13 = D

8 – 1 = 7

23

10 ~~7~~ 20

~~B 8 4~~ F

(-) 0 A 5 3

= 7 D F C

So, $(B84F)_{16} – (A53)_{16} = (7DFC)_{16}$

5. Find the addition, subtraction of $(AB53)_{16}$, $(155)_{16}$

Solution:

The addition of numbers is $(AB53)_{16} + (155)_{16}$

3 + 5 = 8

5 + 5 = 10 = A

B + 1 = 11 + 1 = 12 = C

A + 0 = 10 + 0 = 10 = A

So, $(AB53)_{16} + (155)_{16} = (ACA8)_{16}$

Subtraction of numbers is $(AB53)_{16} – (155)_{16}$

(3 + 16) – 5 = 19 – 5 = 14 = E

(5 – 1) – 5 = 4 – 5

(4 + 16) – 5 = 20 – 5 = 15 = F

(B – 1) – 1 = (11 – 1) – 1 = 10 – 1 = 9

A – 0 = A

A B 5 3

(-) 0 1 5 5

= A 9 F E

So, $(AB53)_{16} – (155)_{16} = (A9FE)_{16}$

6. Calculate $(9AB)_{16} + (12C)_{16}$
7. Compute $(CB5)_{16} – (223)_{16}$

Solution:

(i) $(9AB)_{16} + (12C)_{16}$

B + C = 11 + 12 = 23 = 7 and 1 is carry

1 + A + 2 = 3 + 10 = 13 = D

9 + 1 = 10 = A

So, $(9AB)_{16} + (12C)_{16} = (AD7)_{16}$

(ii) $(CB5)_{16} – (223)_{16}$

5 – 3 = 2

B – 2 = 11 – 2 = 9

C – 2 = 12 – 2 = A

So, $(CB5)_{16} – (223)_{16} = (A92)_{16}$

Problems

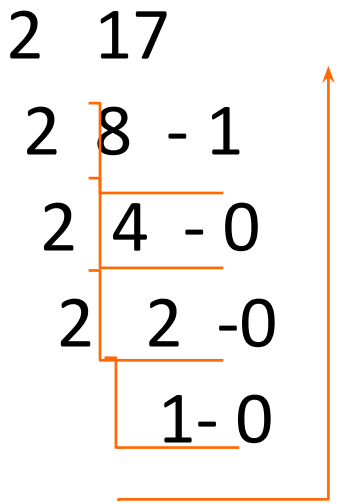
1. Convert the following binary numbers to their decimal representations: (a) 11 (b) 1101 (c) 111011 (d) 0101
2. Convert the following hexadecimal numbers to their decimal representations: (a) 11 (b) A1 (c) CEF (d) BA9
3. Convert the following decimal numbers to their hexadecimal and binary representations: (a) 11 (b) 4000 (c) 42 (d) 4095
4. Do the binary arithmetic: (a) 10110 + 01101 (b) 11001 + 00101 (c) 10110 - 01101 (d) 11111 - 01011
5. Do the hexadecimal arithmetic: (a) 82CD + 1982 (b) E2C + A31 (c) FB28 - 3254 (d) E2C - A31

Solutions

1. (a) 3 (b) 13 (c) 59 (d) 5
2. (a) 17 (b) 161 (c) 3311 (d) 2985
3. (a) B (hex) and 1011 (binary) (b) FA0 (hex) and 1111 1010 0000 (binary) (c) 2A (hex) and 10 1010 (binary) (d) FFF (hex) and 1111 1111 1111 (binary)
4. (a) 100011 (b) 11110 (c) 1001 (d) 10100
5. (a) 9C4F (b) 185D (c) C8D4 (d) 3FB

BCD- Binary Coded Decimal

If decimal converted binary (eg) 17



0001 0001

BCD 1 7

0001 0111

BCD addition

In BCD addition of two numbers involve following rules:-

1. Maximum value of the sum for two digits = 9 (max digit 1) + 9 (max digit 2) + 1 (previous addition carry) = 19
2. If sum of two BCD digits is less than or equal to 9 (1001) without carry then the result is a correct BCD number.
3. If sum of two BCD digits is greater than or equal to 10 (1010) the result is in-correct BCD number. Perform steps 4 for correct BCD sum.
4. Add 6 (0110) to the result.

```
    1 0 0 1
    0 1 0 0   (+)
  ---------
    1 1 0 1
    0 1 1 0   (+)
  ---------
0 0 0 1 0 0 1 1
  1       3
```

19 - 0 0 0 1 1 0 0 1

14 - 0 0 0 1 0 1 0 0

33- 0 0 1 0 1 1 0 1

0 0 1 0 1 1 0 1

+6 0 1 1 0

0 0 1 1 0 0 1 1

3 3

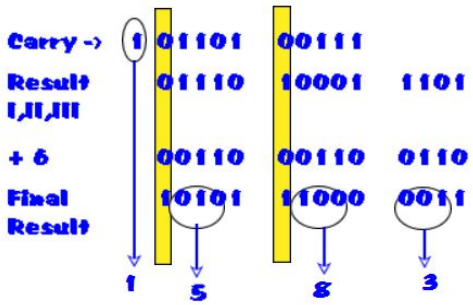
Examples:- BCD Addition

Add 599 and 984 using BCD numbers.

BCD	1	2	3
599	0101	1001	1001
+984	1001	1000	0100
SUM	1110	10001	1101

Binary Sum 1, 2 and 3 are greater than 1010.
So, from Step 3 and 4 add '6' to the sum.

carry 1	carry 1	carry 1	
Result	1110	10001	1101
+6	0110	0110	0110
End carry 1	0101(5)	1000 (8)	0011 (3)



LOGIC GATES

Logic Gates

- The building blocks used to create digital circuits are **logic gates**
- There are three elementary logic gates and a range of other simple gates
- Each gate has its own **logic symbol** which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a **truth table** or using **Boolean notation**

Gates

BASIC

AND

OR

NOT

UNIVERSAL

NAND

NOR

DERIVED

EX-OR

EX-NOR

- **AND gate**

(a) Circuit symbol

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

(b) Truth table

$C = A \cdot B$

(c) Boolean expression

OR gate

(a) Circuit symbol

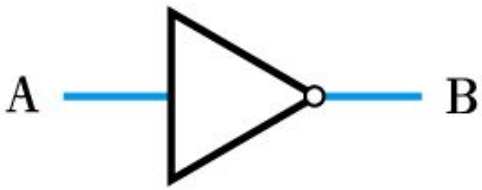
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$C = A + B$

(c) Boolean expression

NOT gate (or inverter)



(a) Circuit symbol

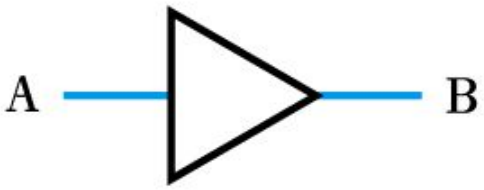
A	B
0	1
1	0

(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

A logic buffer gate



(a) Circuit symbol

A	B
0	0
1	1

(b) Truth table

$B = A$

(c) Boolean expression

NAND gate

(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

NOR gate

(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

Exclusive OR gate (Ex-OR)

(a) Circuit symbol

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

(b) Truth table

$C = A \oplus B$

(c) Boolean expression

Exclusive NOR gate (Ex-NOR)

(a) Circuit symbol

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

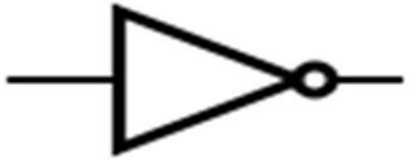
(b) Truth table

$$C = \overline{A \oplus B}$$

(c) Boolean expression

LOGIC GATES

Digital Logic Gate Symbols

Gate	Symbol	Notation	Truth Table
AND		$A \cdot B$	INPUT / OUTPUT A B | A AND B 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$A + B$	INPUT / OUTPUT A B | A OR B 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$\overline{A}$	INPUT / OUTPUT A | NOT A 0 | 1 1 | 0
NAND		$\overline{A \cdot B}$	INPUT / OUTPUT A B | A NAND B 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$\overline{A + B}$	INPUT / OUTPUT A B | A NOR B 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$A \oplus B$	INPUT / OUTPUT A B | A XOR B 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Minimization of Boolean Functions: Algebraic simplification

Boolean Operations and Expressions

Addition	Multiplication
0 + 0 = 0	0 * 0 = 0
0 + 1 = 1	0 * 1 = 0
1 + 0 = 1	1 * 0 = 0
1 + 1 = 1	1 * 1 = 1

Laws and Rules of Boolean Algebra

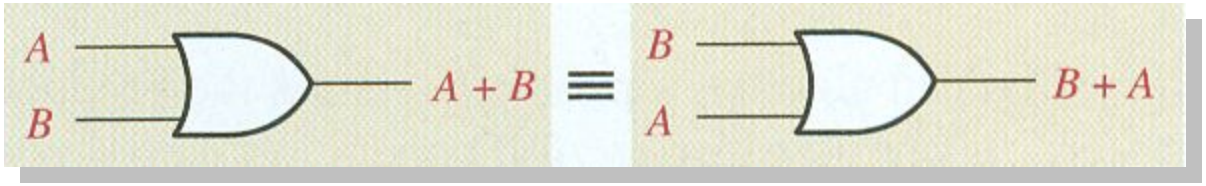
Laws Boolean Algebra

- Commutative Laws
- Associative Laws
- Distributive Law

Laws of Boolean Algebra

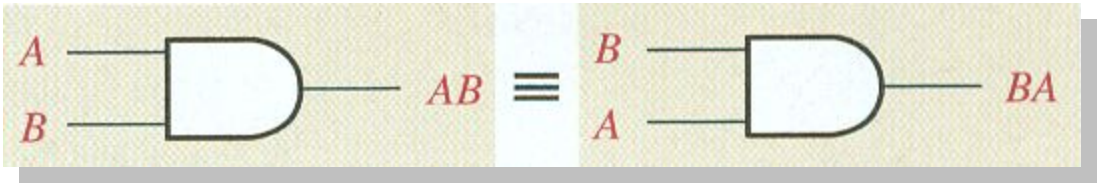
Commutative Law of Addition:

A + B = B + A



Commutative Law of Multiplication:

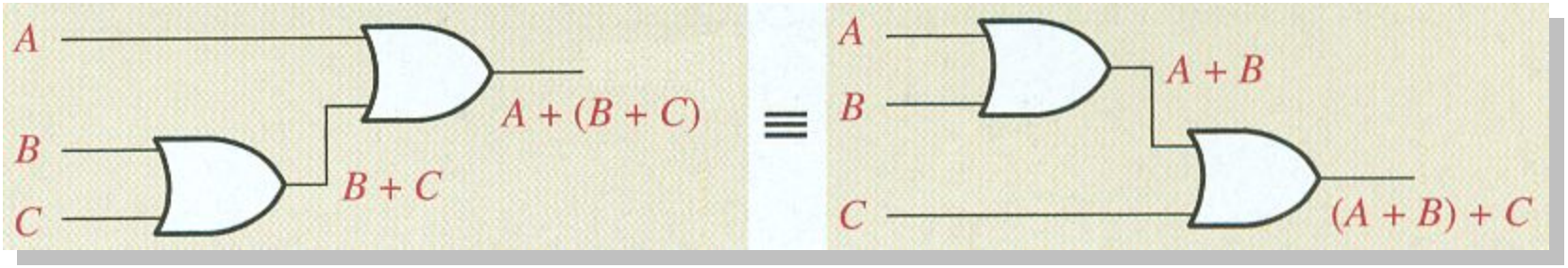
A * B = B * A



Laws of Boolean Algebra

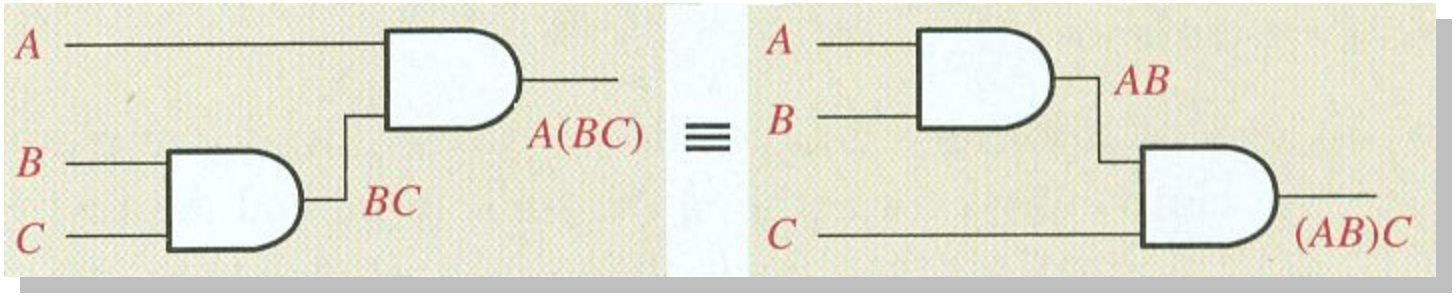
Associative Law of Addition:

A + (B + C) = (A + B) + C



Associative Law of Multiplication:

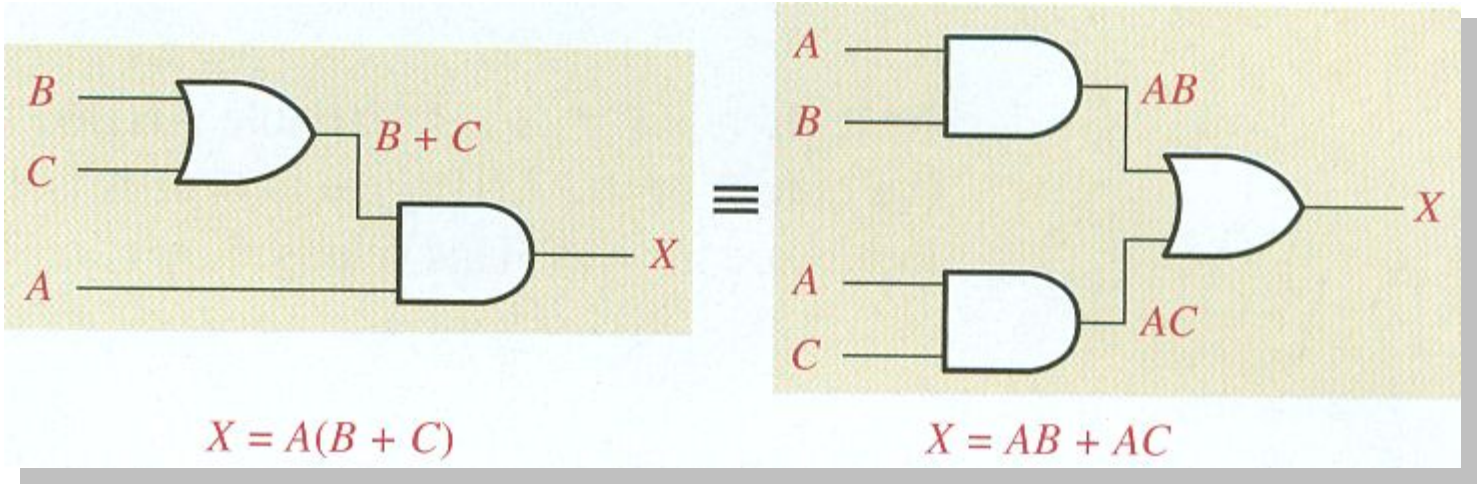
A * (B * C) = (A * B) * C



Laws of Boolean Algebra

Distributive Law:

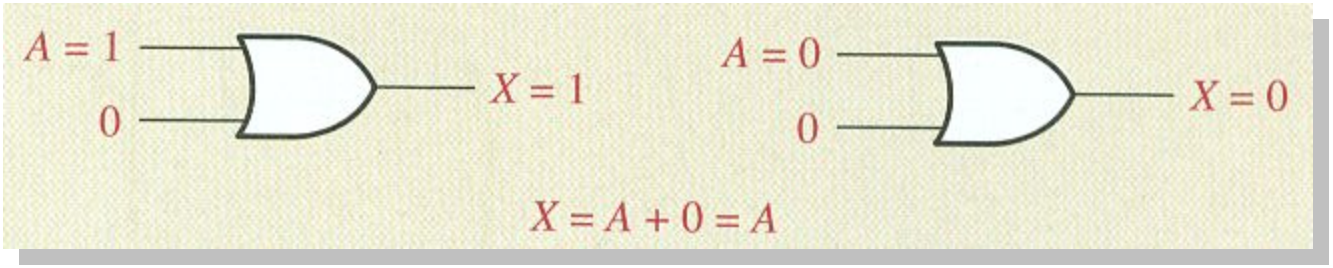
A(B + C) = AB + AC



Rules of Boolean Algebra

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

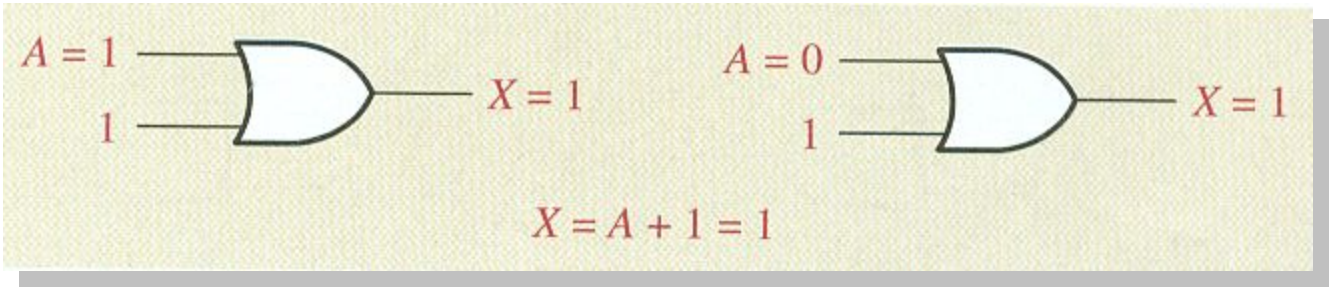
Rule 1



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

Rule 2

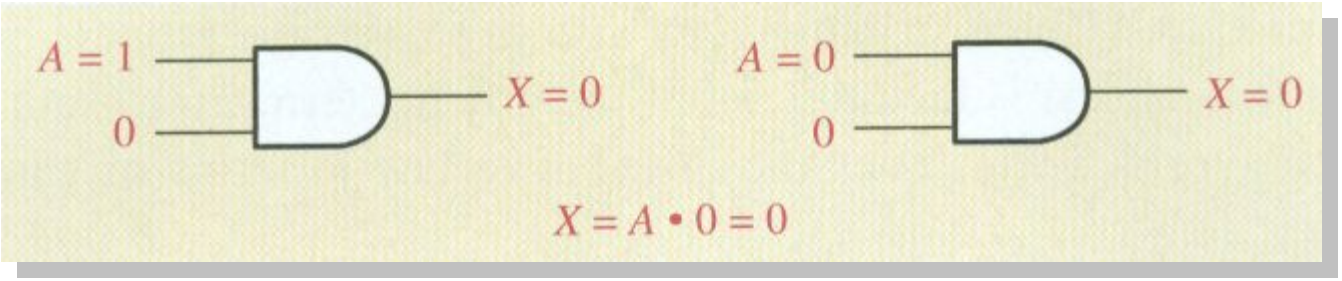


A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

Rules of Boolean Algebra

Rule 3

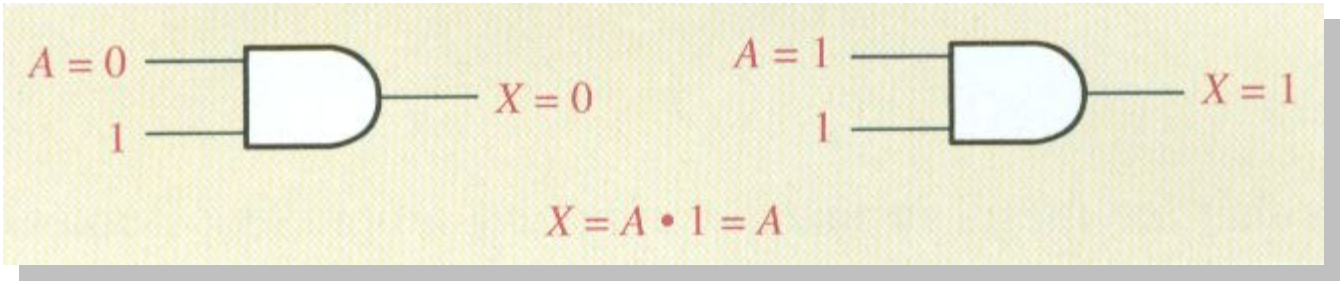


A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

Rules of Boolean Algebra

Rule 4

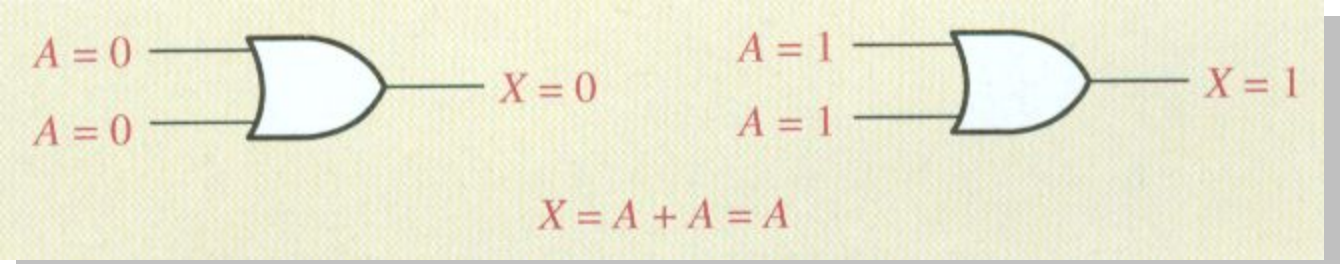


A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

Rules of Boolean Algebra

Rule 5

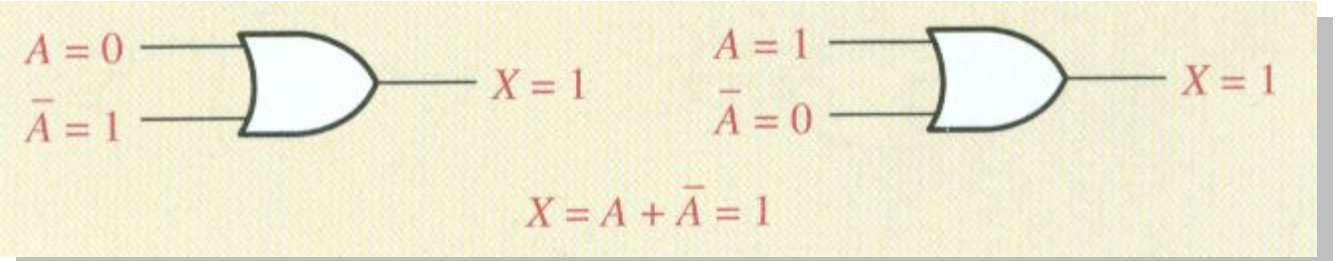


A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

Rules of Boolean Algebra

Rule 6

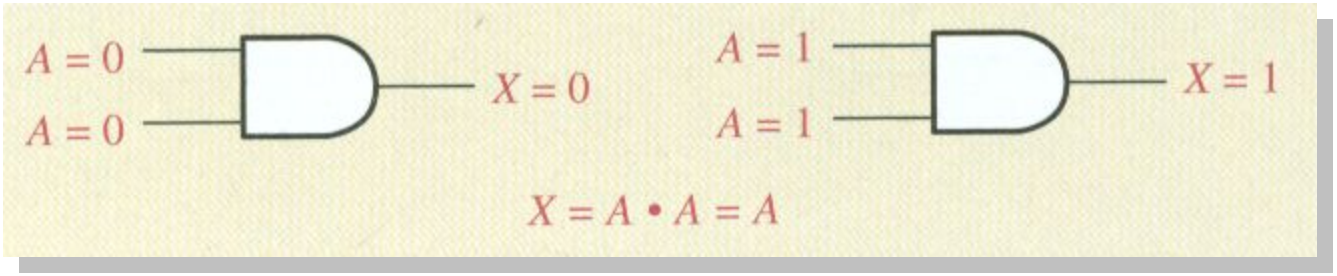


A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

Rules of Boolean Algebra

Rule 7

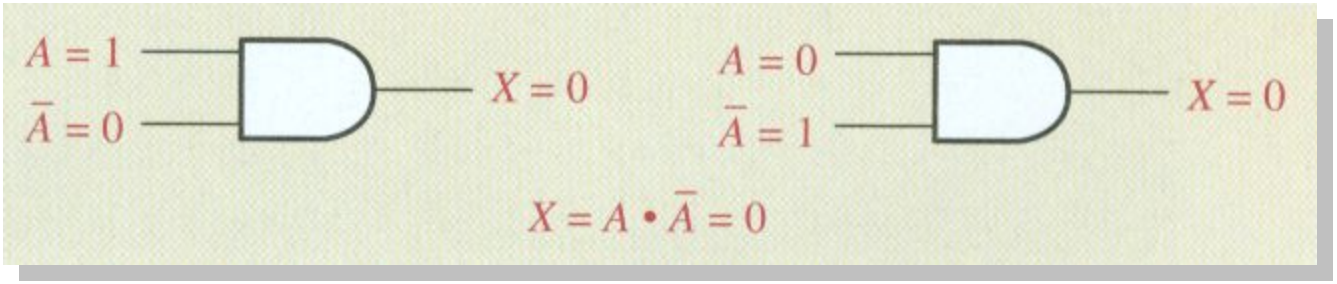


A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

Rules of Boolean Algebra

Rule 8

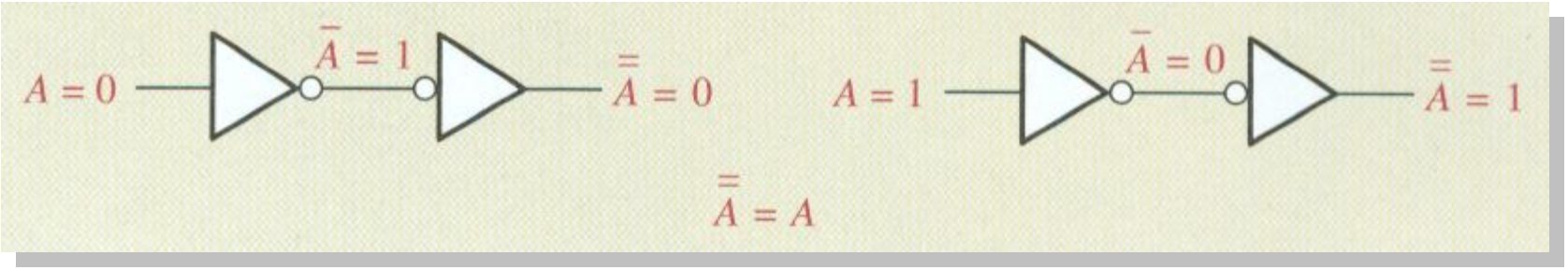


A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

Rules of Boolean Algebra

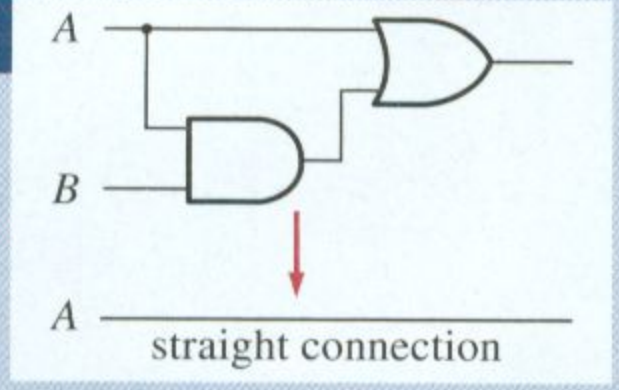
Rule 9



Rule 10: A + AB = A

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

Rules of Boolean Algebra

Rule 11: $A + \overline{A}B = A + B$

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

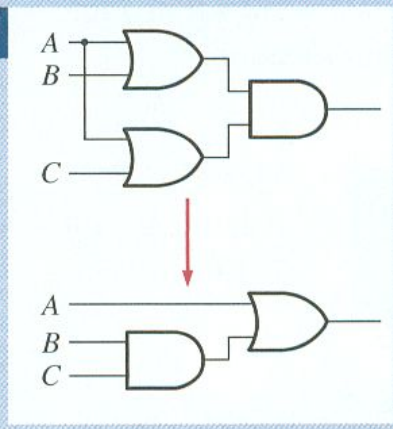
OR Truth Table

Rules of Boolean Algebra

Rule 12: (A + B)(A + C) = A + BC

A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR Truth Table

DeMorgan's Theorem

DeMorgan's Theorems

- Theorem 1

$$\overline{XY} = \overline{X} + \overline{Y} \qquad \overline{\overline{XY}} = \overline{\overline{X}} + \overline{\overline{Y}}$$

- Theorem 2

$$\overline{X + Y} = \overline{X}\,\overline{Y}$$

Remember:

"Break the bar, change the sign"

DE MORGANS

$$\overline{AB} = \bar{A} + \bar{B}$$

BUBBLE PUSHING THEOREM

$$\overline{A + B} = \bar{A} \,.\, \bar{B}$$

Standard Forms of Boolean Expressions

Standard Forms of Boolean Expressions

- ✔ The sum-of-product (SOP) form

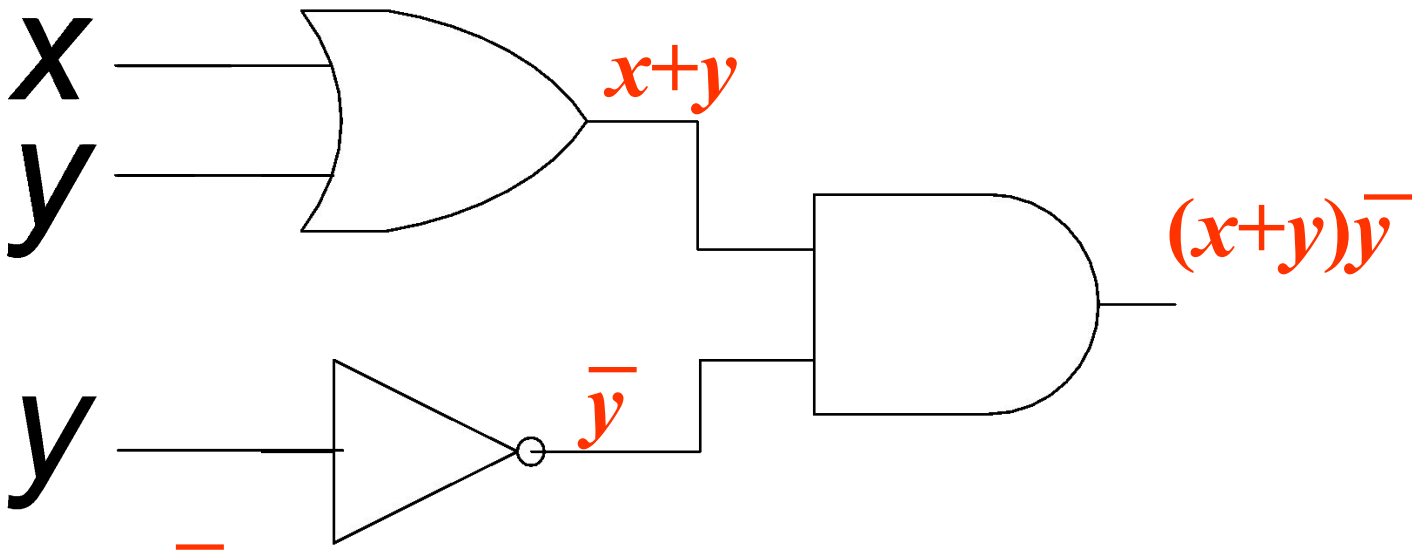
  Example: X = AB + CD + EF

- ✔ The product of sum (POS) form

  Example: X = (A + B)(C + D)(E + F)

Converting between Circuits and Equations

Find the output of the following circuit



Answer: $(x+y)\overline{y}$

Recollect the Boolean Rules

COMMUTATIVE

A+B=B+A A.B=B.A

ASSOCIATIVE PROPERTY

A+(B+C)=(A+B)+C
A.(B.C)=(A.B).C

DISTRIBUTIVE

A+BC= (A+B)(A+C)
A.(B+C)=(A.C)+(A.B)

ADDITION

1) A+0=A
2) A+1=1
3) A+A=A
4) $A+\bar{A}=1$

MULTIPLICATION

5) A.0=0
6) A.1=A
7) A.A=A
8) $A.\bar{A}=0$
9) $\bar{\bar{A}}=A$

Solve

i) A+AB = A

A+AB= A(1+B)

= A(1)

= A

ii) $A+ \bar{A}B=A+B$

$A+ \bar{A}B= (A+ \bar{A}).(A+B)$

$= \quad (1) .(A+B)$

$= \quad A+B$

iii) $AB+\bar{A}C + BC = AB + \bar{A}C$

$AB+ BC+ \bar{A}C = \quad AB+ \bar{A}C +BC(A+ \bar{A})$

$= \quad AB+ \bar{A}C +ABC + \bar{A}BC$

$= \quad AB(1+C)+ \bar{A}C\ (B+1)$

$= \quad AB + \bar{A}C$ (CONSENSUS THEOREM)

(iv) $A+AB+A\bar{B}C = A+A\bar{B}C$ *(A+AB=A)*

$= A(1+\bar{B}C)$

$= A(1)$

$= A$

(v) $A\bar{B}C(BD+CED)+A\bar{C}$

$=A\bar{B}CBD+A\bar{B}CCED+A\bar{C}$

$= 0 + A\bar{B}CED + A\bar{C}$

$= A(\bar{B}CED+\bar{C})$ *$(A+\bar{A}B=A+B)$*

$= A(\bar{B}ED+\bar{C})$

(vi) $\overline{\overline{(a+\bar{c})}.b\bar{d}}.\overline{(a+\bar{c}).b\bar{d}}=\mathrm{F}$

$$\mathrm{F}=\overline{\overline{\overline{(a+\bar{c})}+\overline{b\bar{d}}}.\overline{(a+\bar{c})}+\overline{b\bar{d}}}$$

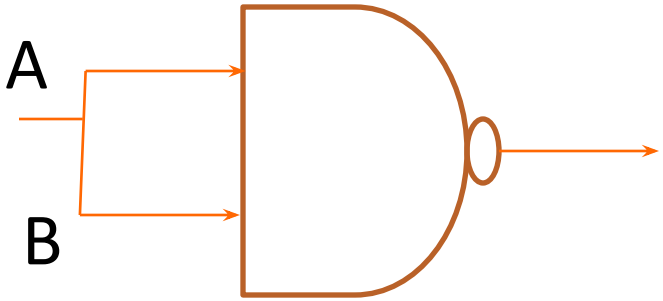
$$=\overline{(a+\bar{c})+\overline{b\bar{d}}.\overline{(a+\bar{c})}+\overline{b\bar{d}}}$$

$$=\overline{(a+\bar{c}).\overline{(a+\bar{c})}+\overline{b\bar{d}}} \qquad (a+b)(a+c)=a+bc$$

$$=\overline{\overline{b\bar{d}}}$$

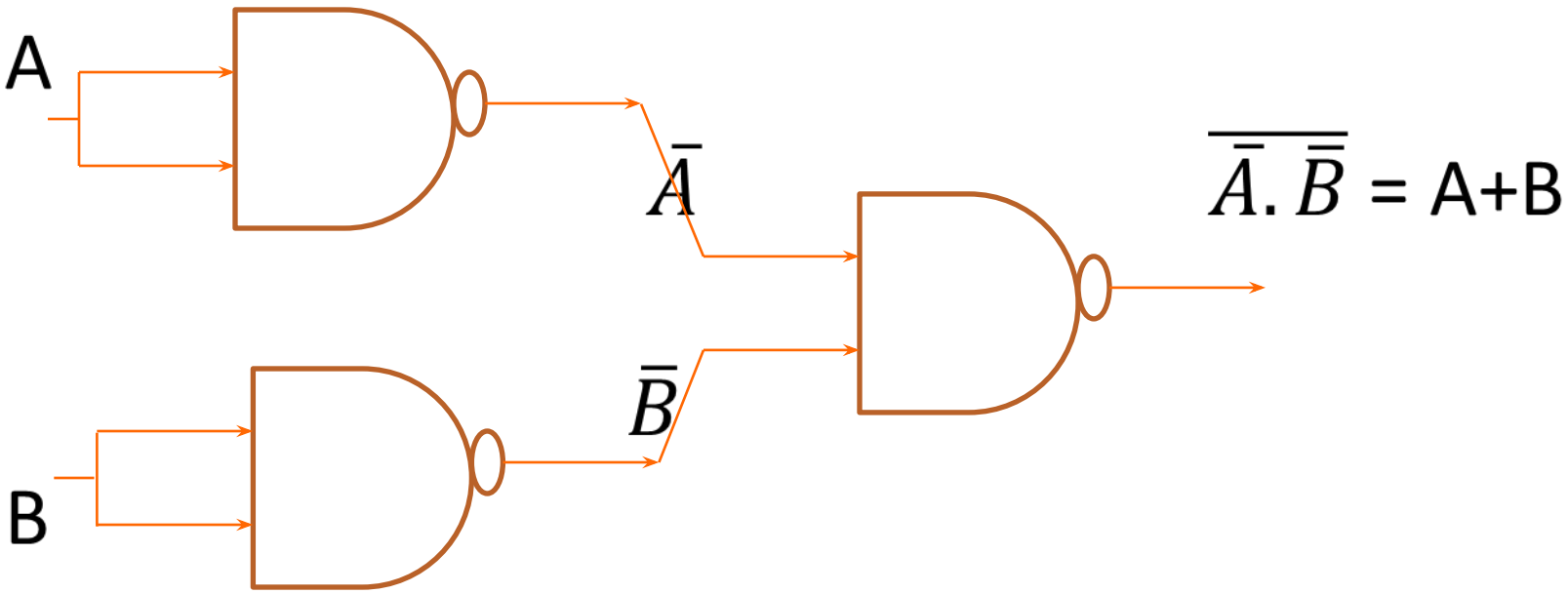
$$=b\bar{d}$$

IMPLEMENT INVERTER USING NAND GATE



A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

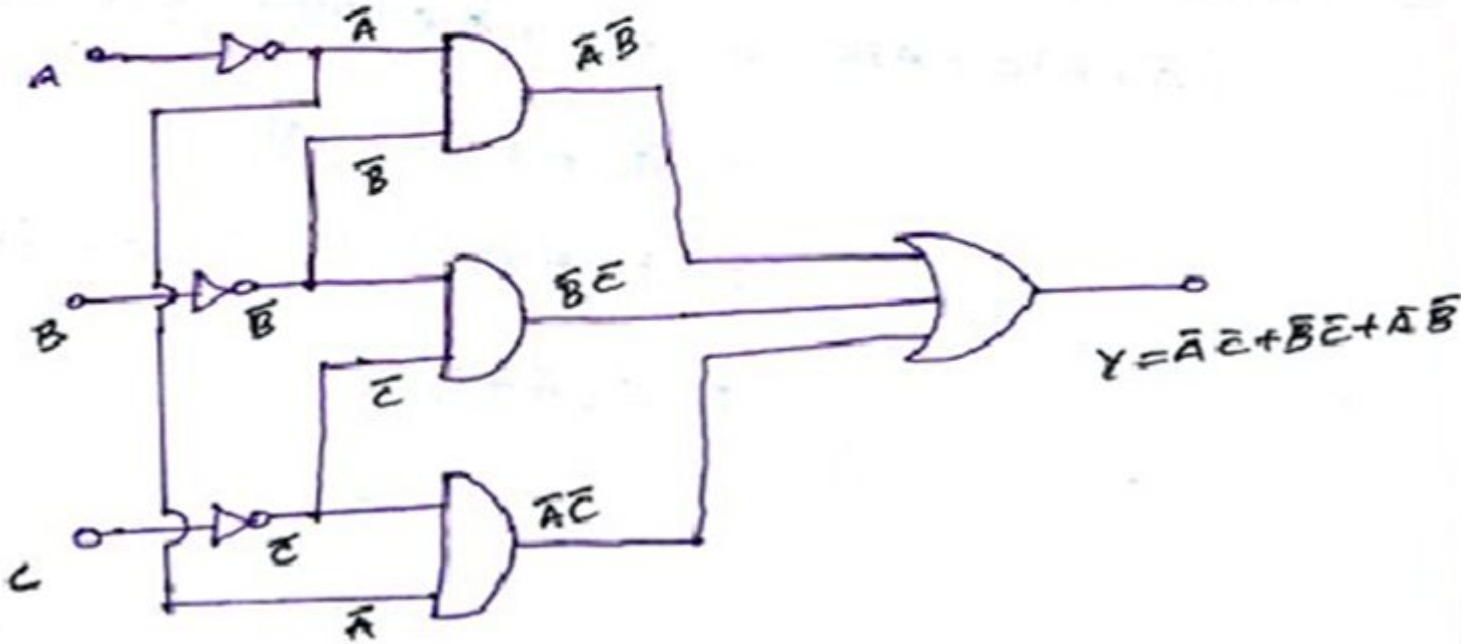
OR gate using NAND gate



Example

Implementation of Boolean Expression using Logic Gates

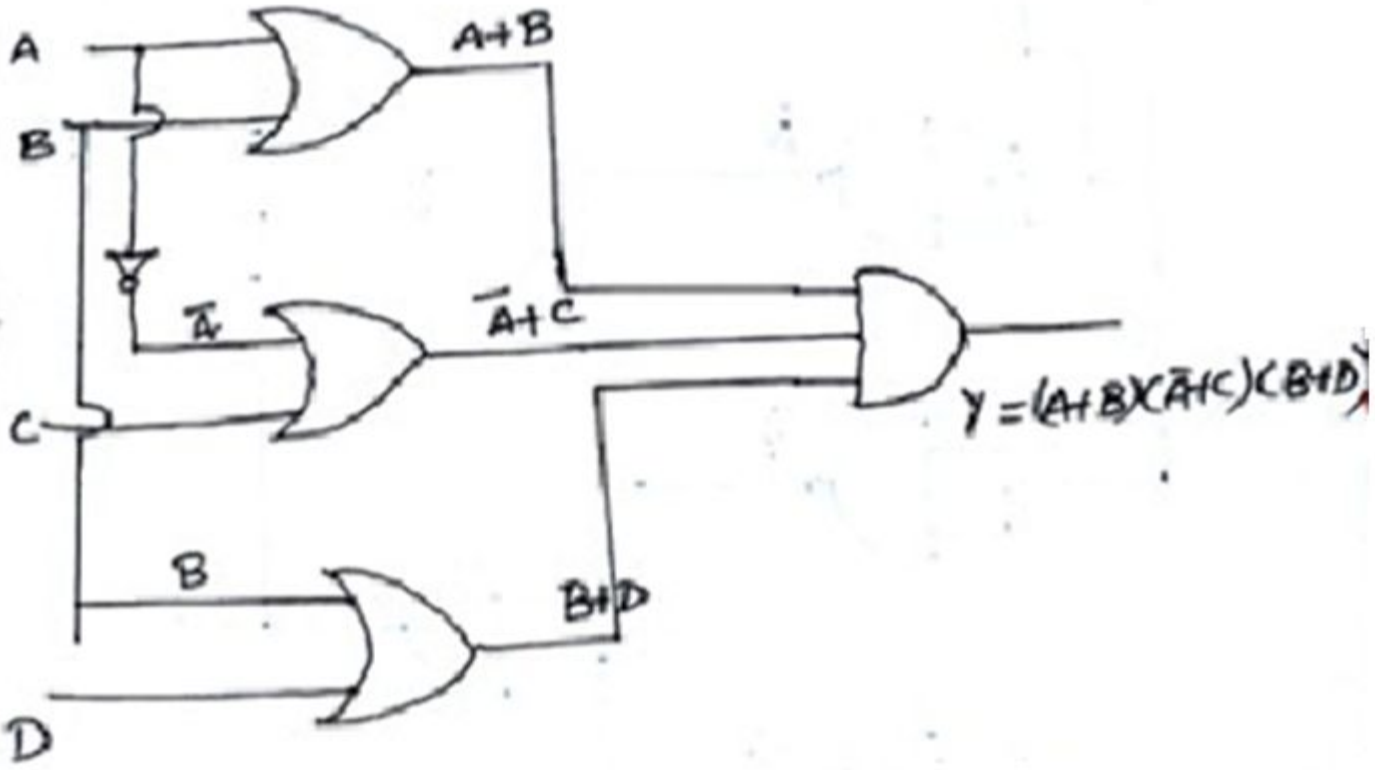
Realise the logic expression $Y = \bar{B}\bar{C} + \bar{A}\bar{C} + \bar{A}\bar{B}$ using basic gates.



Example

Realise the logic expression

$Y = (A+B)(\bar{A}+C)(B+D)$ using basic gates.



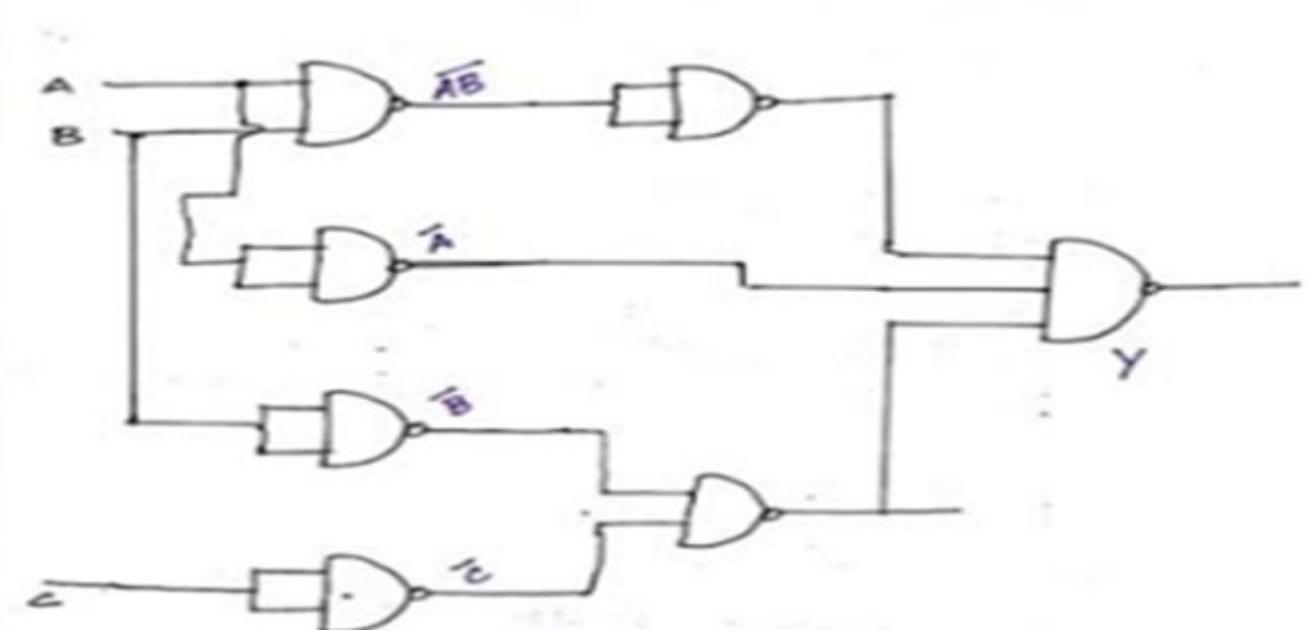
Example

Implement $Y = \overline{AB} + A + \overline{(B+C)}$ using NAND gates only.

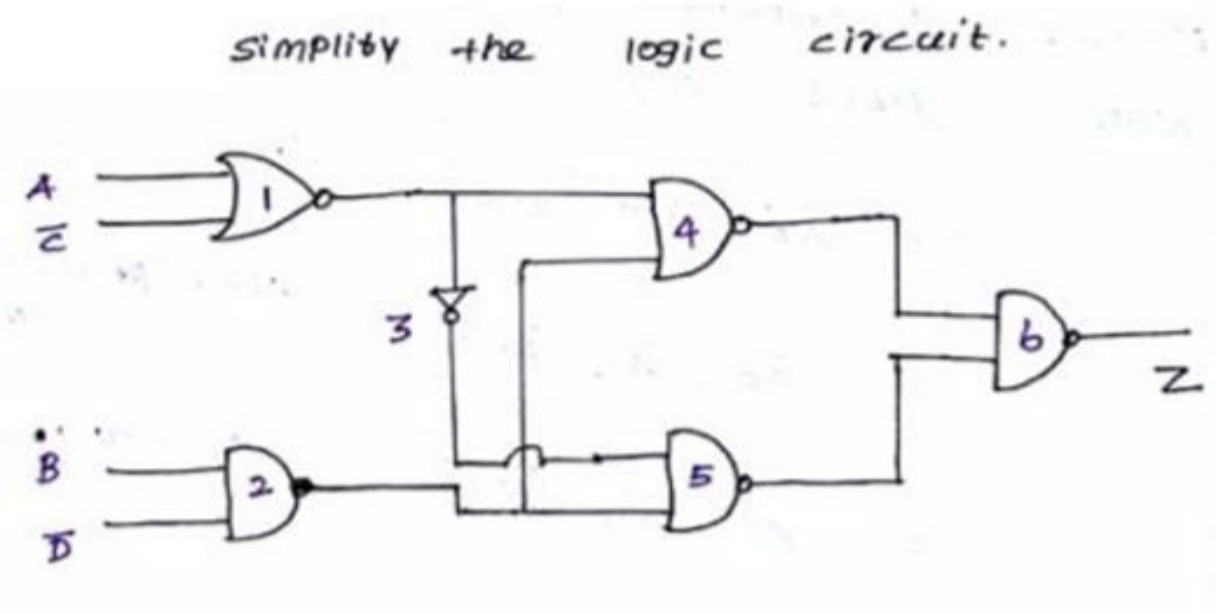
$$Y = \overline{\overline{AB} + A + \overline{(B+C)}}$$ (Take double complement)

$$= \overline{\overline{\overline{AB}} \cdot \overline{A} \cdot \overline{\overline{B+C}}}$$ (Apply Demorgan's Law)

$$= \overline{\overline{\overline{AB}} \cdot \overline{A} \cdot \overline{B \cdot C}}$$ (De Morgan's Law)



Example



Karnaugh Maps
(K – MAPS)

K-Map

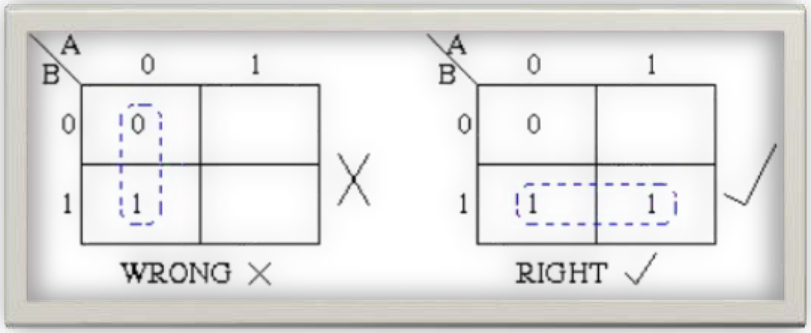
- Karnaugh maps (K-maps) are *graphical* representations of Boolean functions.
- One map cell corresponds to a row in the truth table.
- Also, one map cell corresponds to a minterm or a maxterm in the Boolean expression
- Multiple-cell areas of the map correspond to standard terms.
- A K-map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.
- It's similar to truth table; instead of being organized (i/p and o/p) into columns and rows, the K-map is an array of cells in which each cell represents a binary value of the input variables.

- The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells.

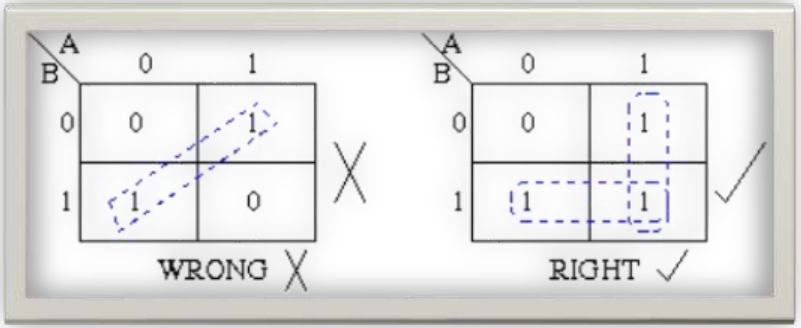
Grouping

Rules of grouping -

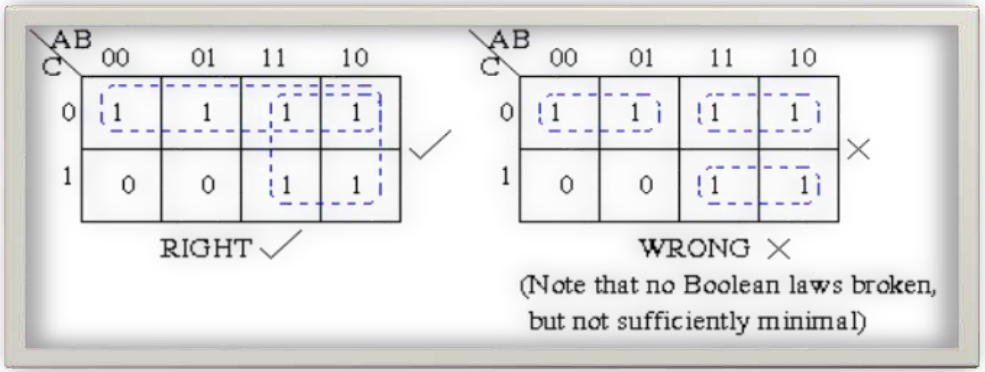
1’s & 0’s can not be grouped



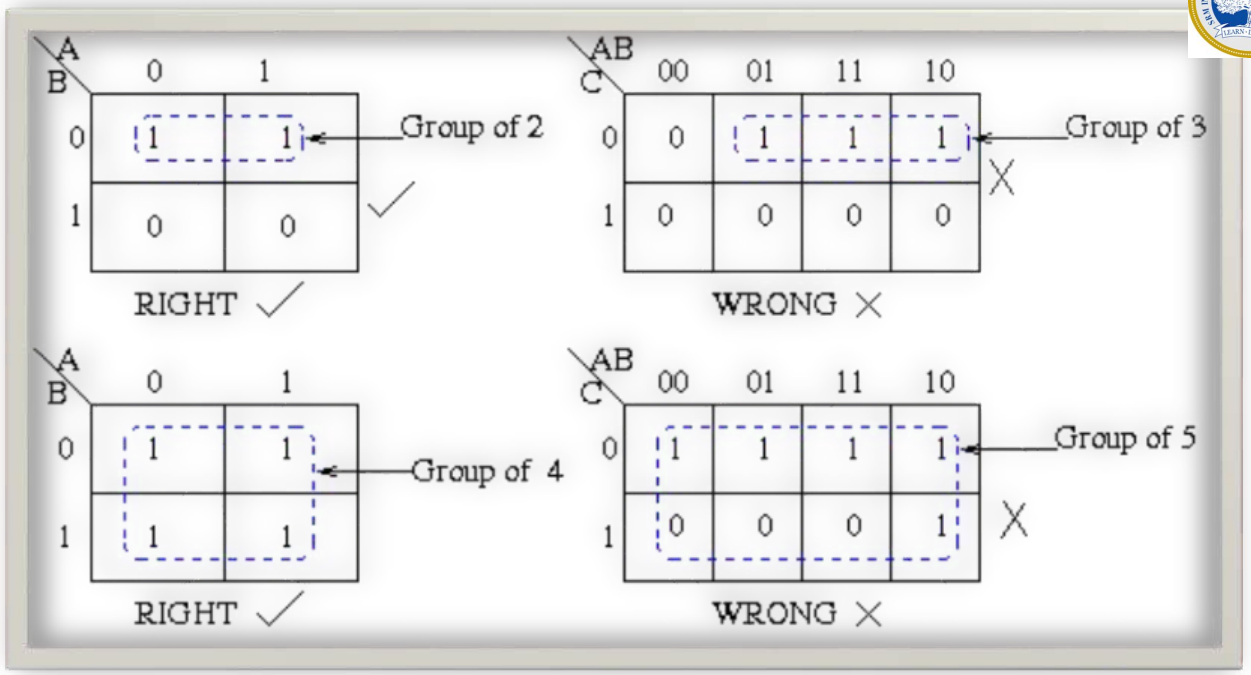
diagonal 1’s can not be grouped



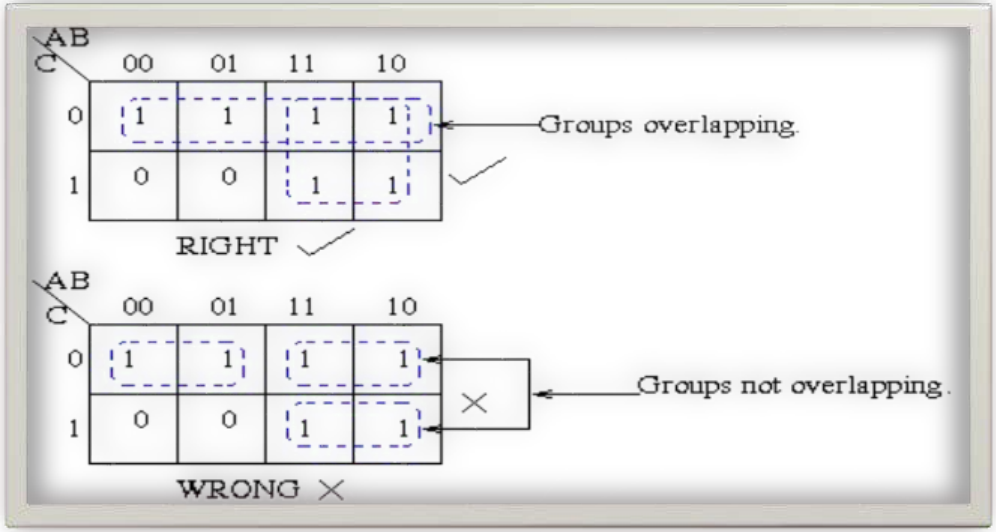
Minimum Groups should be formed

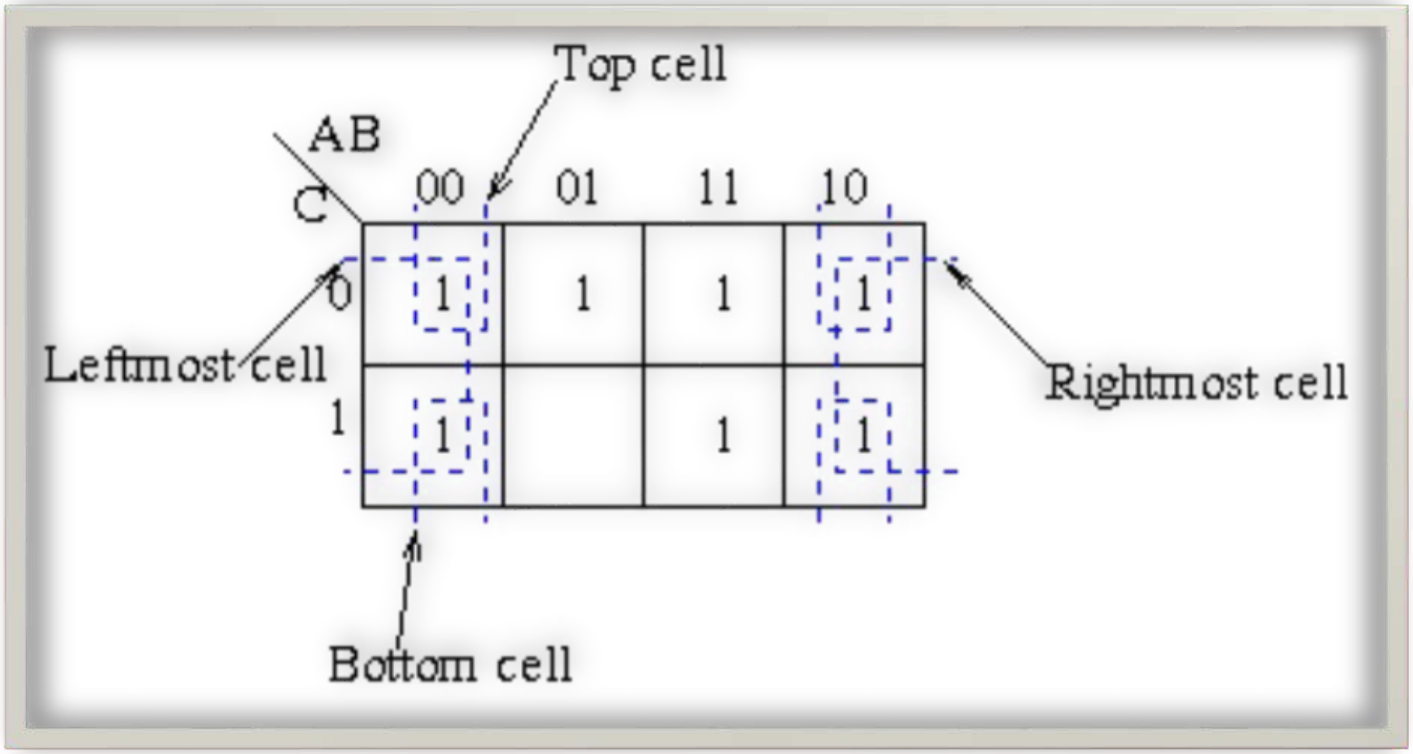


Elements in a group should be 2^n

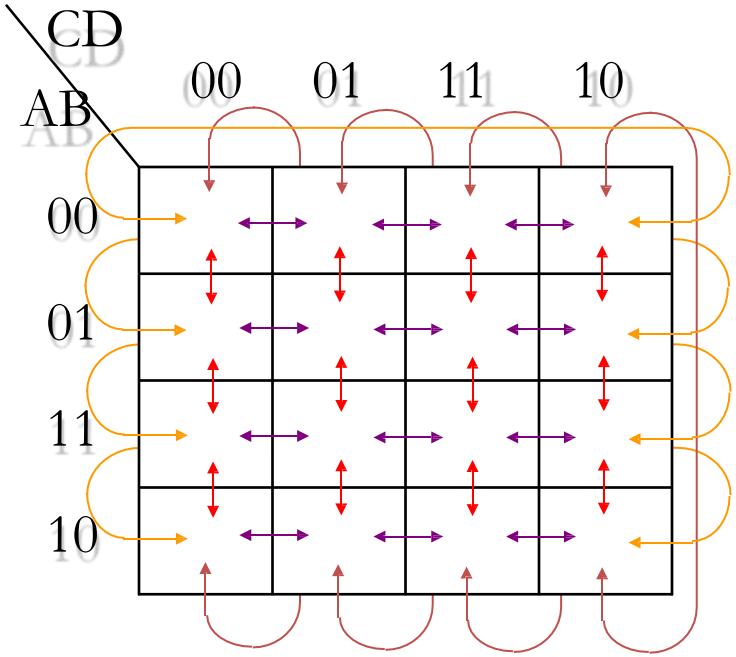


For above rule group Overlapping is applicable




Top cell
AB
C
00
01
11
10
0
1
1
1
1
1
1
1
1
Leftmost cell
Rightmost cell
Bottom cell

Cell Adjacency

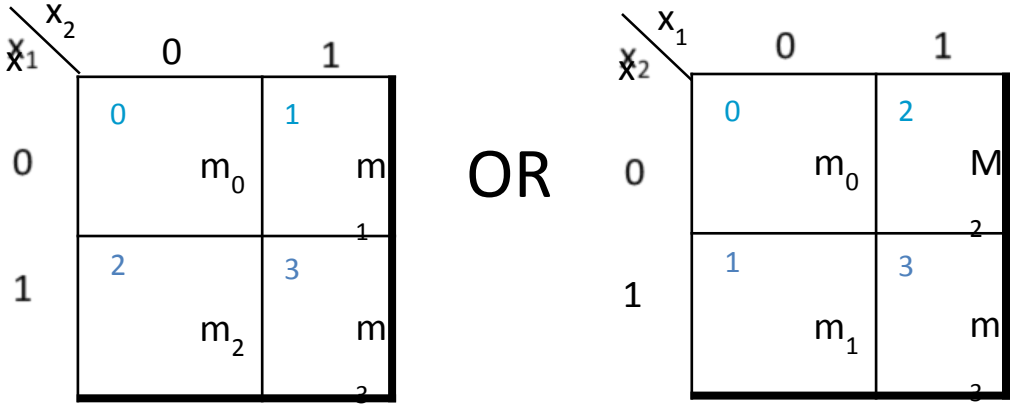


Two-Variable Map

- Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.
- Example:

  $m_0 (=x_1'x_2')$ is adjacent to $m_1 (=x_1'x_2)$ and $m_2 (=x_1x_2')$ but NOT $m_3 (=x_1x_2)$

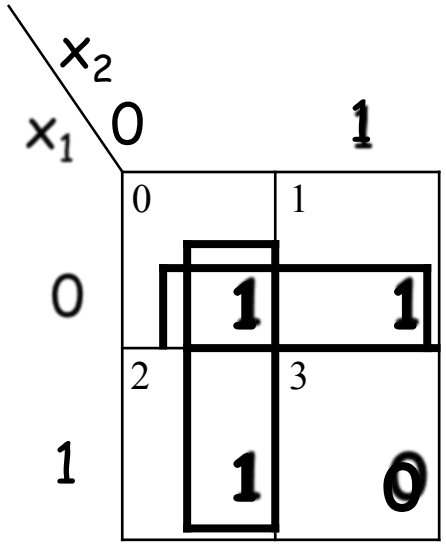
Two-Variable Map



- ordering of variables is IMPORTANT for $f(x_1,x_2)$, x_1 is the row, x_2 is the column.
- Cell 0 represents $x_1'x_2'$; Cell 1 represents $x_1'x_2$; etc. If a minterm is present in the function, then a 1 is placed in the corresponding cell.

2-Variable Map -- Example

- $f(x_1,x_2) = x_1'x_2' + x_1'x_2 + x_1x_2'$
  $= m_0 + m_1 + m_2$
  $= x_1' + x_2'$
- 1s placed in K-map for specified minterms m_0, m_1, m_2
- Grouping of 1s allows simplification
- What (simpler) function is represented by each dashed rectangle?
  - $x_1' = m_0 + m_1$
  - $x_2' = m_0 + m_2$
- Here m_0 covered twice



3 Variable K-Map

- There are 8 cells as shown:

AB \ C	0	1
00	$\overline{A}\overline{B}\overline{C}$	$\overline{A}\overline{B}C$
01	$\overline{A}B\overline{C}$	$\overline{A}BC$
11	$AB\overline{C}$	ABC
10	$A\overline{B}\overline{C}$	$A\overline{B}C$

Example : 3 var. k-map

Minimize the given equation using K-map

F = $\bar{A}\bar{B}\bar{C}+\bar{A}B\bar{C}+A\bar{B}C+ABC$

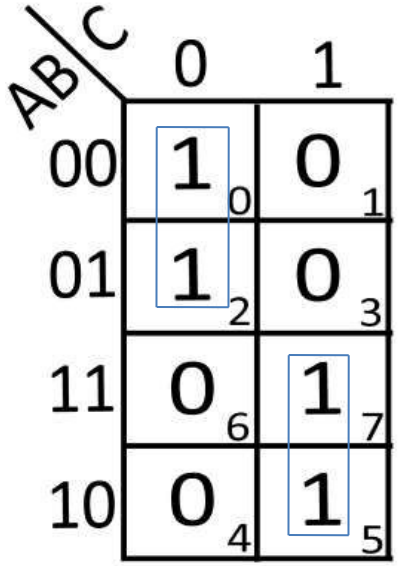
$\bar{A}\bar{B}\bar{C}$ = 000 = 0

$\bar{A}$BC = 010 = 2

A$\bar{B}$C = 101 = 5

ABC = 111 = 7

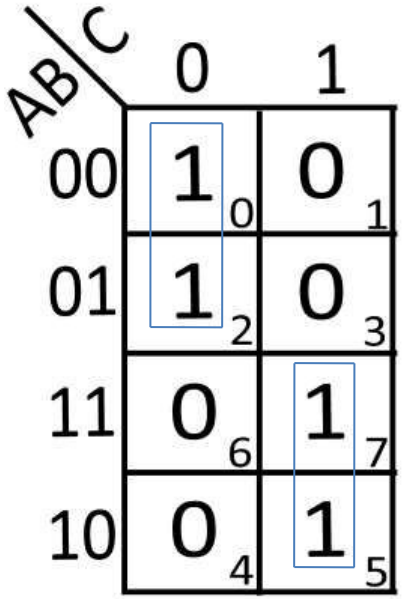
Using this fill the k-map



- Grouping – here 2 groups of 2 1's Is possible

- For upper group A and C are constants and B is varying. Neglect B.A and C both are 0. Hence output of this group is $\bar{A}\bar{C}$
- For upper group A and C are constants and B is varying. Neglect B.A and C both are 1. Hence output of this group is AC
- Thus output Y is given by ,

$$Y=AC+\bar{A}\bar{C}$$
$$=A\odot C$$

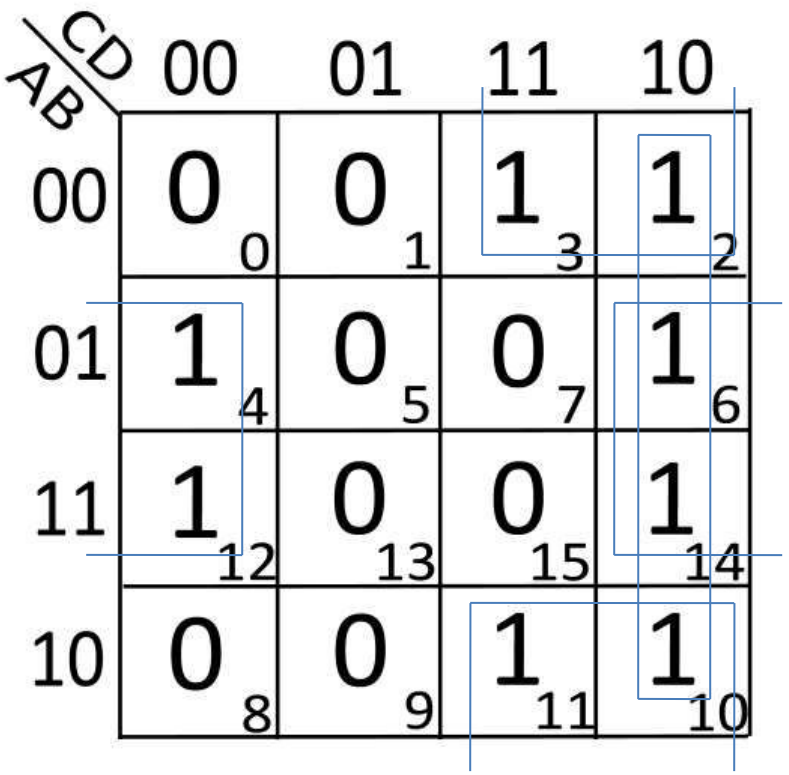


4-Variable K-Map

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

AB \ CD	00	01	11	10
00	$\overline{A}\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}D$	$\overline{A}\overline{B}CD$	$\overline{A}\overline{B}C\overline{D}$
01	$\overline{A}B\overline{C}\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$
11	$AB\overline{C}\overline{D}$	$AB\overline{C}D$	$ABCD$	$ABC\overline{D}$
10	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$	$A\overline{B}CD$	$A\overline{B}C\overline{D}$

Solve the given k-map



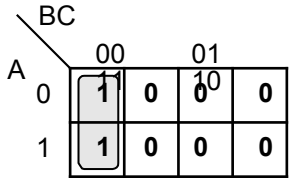
- Step I -grouping
- Step II -output of each group
- Step III -final output

 Here answer is ,

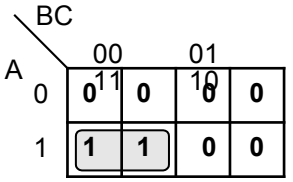
$Y=C\overline{D}+B\overline{C}+B\overline{D}$

Three-Variable K-Maps

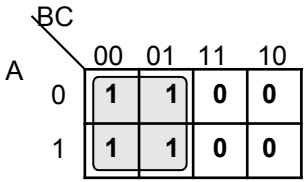
$f = \sum(0,4) = \overline{B}\,\overline{C}$



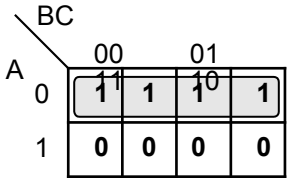
$f = \sum(4,5) = A\,\overline{B}$



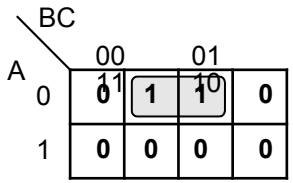
$f = \sum(0,1,4,5) = \overline{B}$



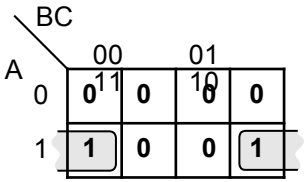
$f = \sum(0,1,2,3) = \overline{A}$



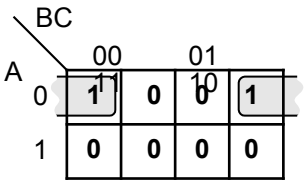
$f = \sum(0,4) = \overline{A}\,\overline{C}$



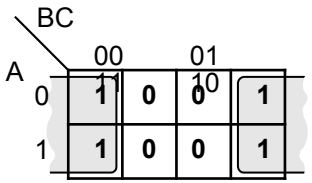
$f = \sum(4,6) = A\,\overline{C}$



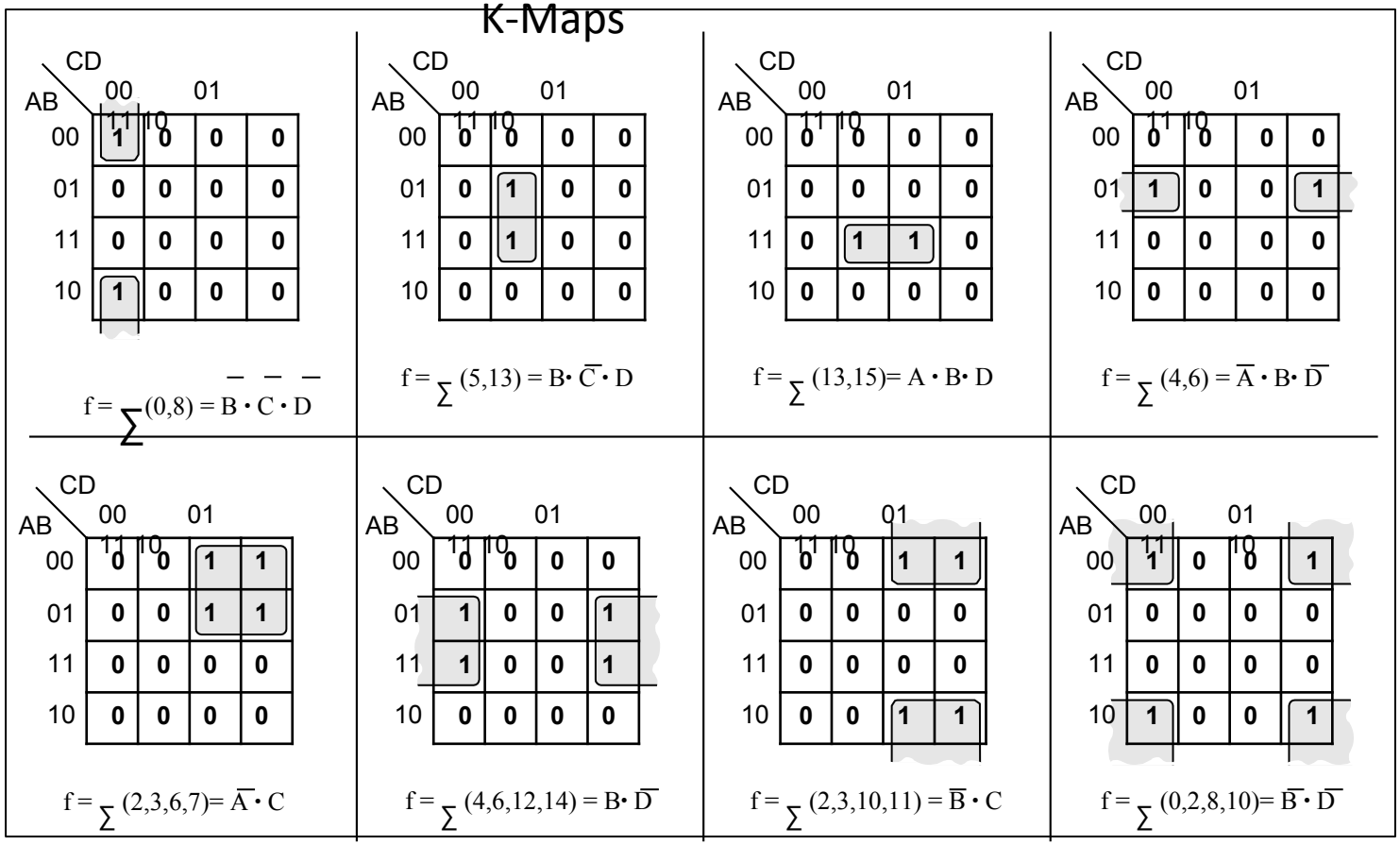
$f = \sum(0,2) = \overline{A}\,\overline{C}$



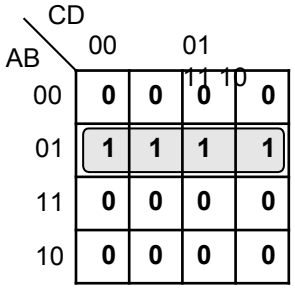
$f = \sum(0,2,4,6) = \overline{C}$



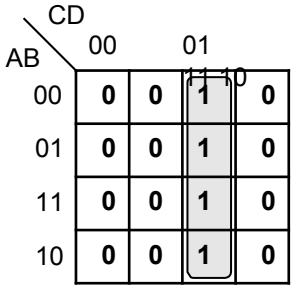
Four-Variable K-Maps



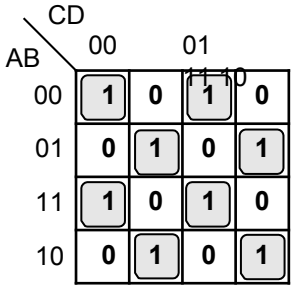
Four-Variable K-Maps



$f = \sum (4, 5, 6, 7) = \overline{A} \cdot B$

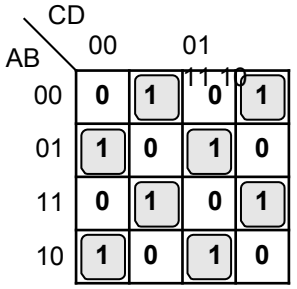


$f = \sum (3, 7, 11, 15) = C \cdot D$



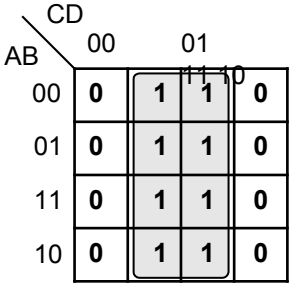
$f = \sum (0, 3, 5, 6, 9, 10, 12, 15)$

$f = A \otimes B \otimes C \otimes D$



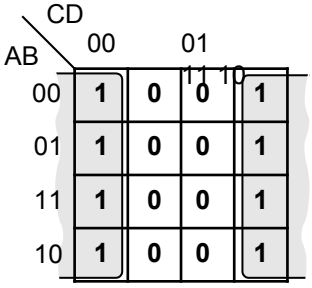
$f = \sum (1, 2, 4, 7, 8, 11, 13, 14)$

$f = A \oplus B \oplus C \oplus D$



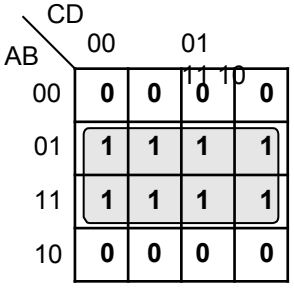
$f = \sum (1, 3, 5, 7, 9, 11, 13, 15)$

$f = D$



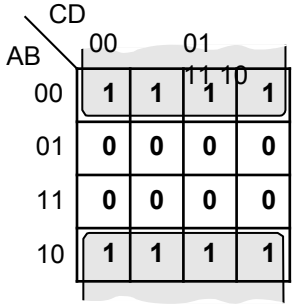
$f = \sum (0,2,4,6,8,10,12,14)$

$f = \overline{D}$



$f = \sum (4,5,6,7,12,13,14,15)$

$f = B$



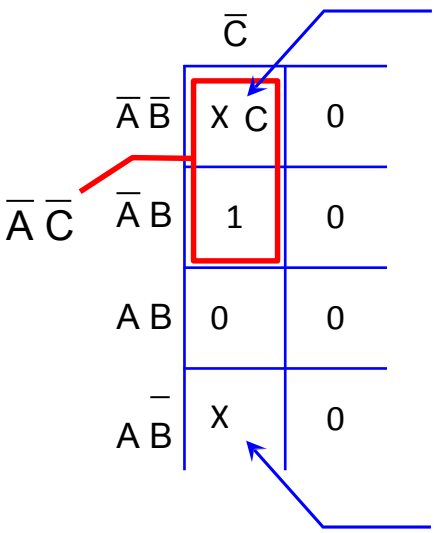
$f = \sum (0,1,2,3,8,9,10,11)$

$f = B$

Don't Care Conditions

- A *don't care* condition, marked by (X) in the truth table, indicates a condition where the design doesn't care if the output is a (0) or a (1).
- A *don't care* condition can be treated as a (0) or a (1) in a K-Map.
- Treating a *don't care* as a (0) means that you do not need to group it.
- Treating a *don't care* as a (1) allows you to make a grouping larger, resulting in a simpler term in the SOP equation.

Some You Group, Some You Don't



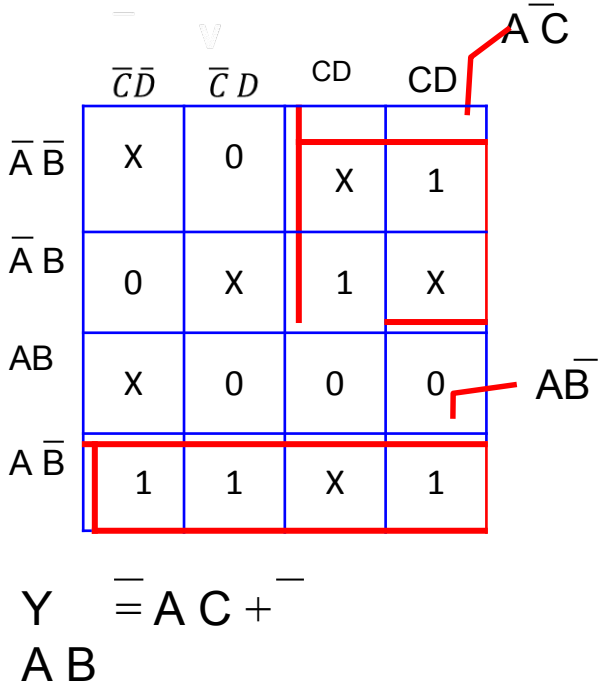
This *don't care* condition was treated as a (1). This allowed the grouping of a single one to become a grouping of two, resulting in a simpler term.

There was no advantage in treating this *don't care* condition as a (1), thus it was treated as a (0) and not grouped.

Example

Solution:

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

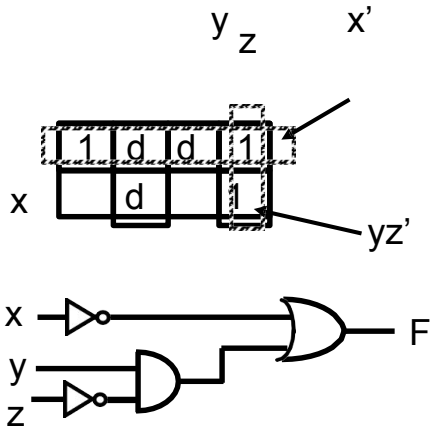


IMPLEMENTATION OF K-MAPS with don't care

- In some logic circuits, the output responses for some input conditions are don't care whether they are 1 or 0.

- In K-maps, don't-care conditions are represented by d's in the corresponding cells.

- Don't-care conditions are useful in minimizing the logic functions using K-map.
  - Can be considered either 1 or 0
  - Thus increases the chances of merging cells into the larger cells
    --> Reduce the number of variables in the product terms



K-Map SOP Minimization

- The K-Map is used for simplifying Boolean expressions to their minimal form.
- A minimized SOP expression contains the fewest possible terms with fewest possible variables per term.
- Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression.

Mapping a Standard SOP Expression

- For an SOP expression in standard form:
  - A 1 is placed on the K-map for each product term in the expression.
  - Each 1 is placed in a cell corresponding to the value of a product term.

variable map.

AB \ C	0	1
00	$\overline{A}\overline{B}\overline{C}$	$\overline{A}\overline{B}C$
01	$\overline{A}B\overline{C}$	$\overline{A}BC$
11	$AB\overline{C}$	ABC
10	$A\overline{B}\overline{C}$	$A\overline{B}C$

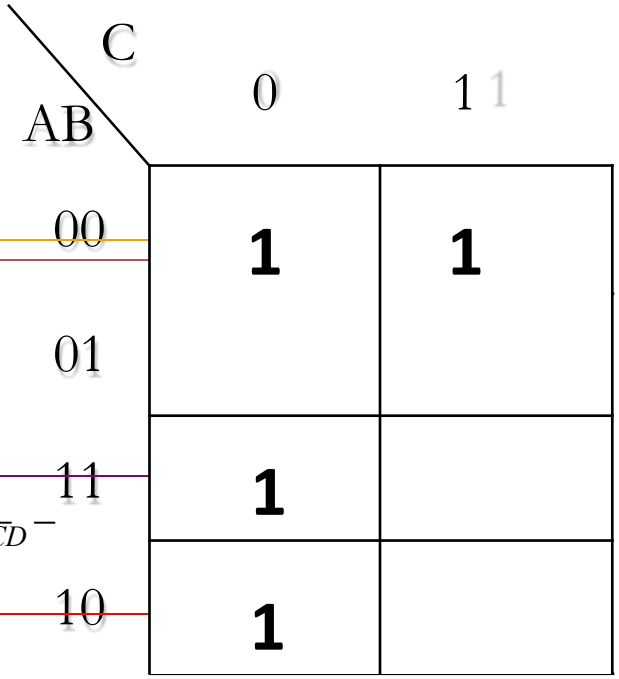
- Example: for the product term , a 1 goes in the 101 cell on a 3-

Mapping a Standard SOP Expression

The expression:

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

000 001 110 100



Practice:

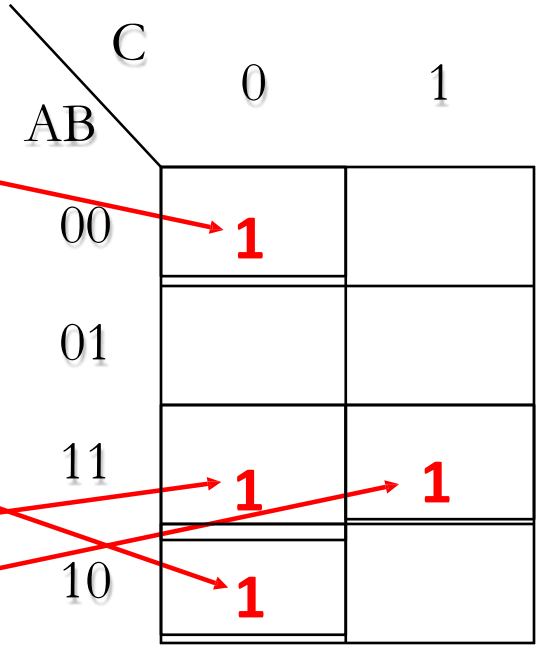
$ABC + ABC + ABC + ABC$

$\overline{A}BC + A\overline{B}C + AB\overline{C}$

$\overline{A}\overline{B}CD + \overline{A}BC\overline{D} + ABC\overline{D} + ABCD + ABC\overline{D} + \overline{A}\overline{B}\overline{C}D + AB\overline{C}D$

Mapping Directly from a Table

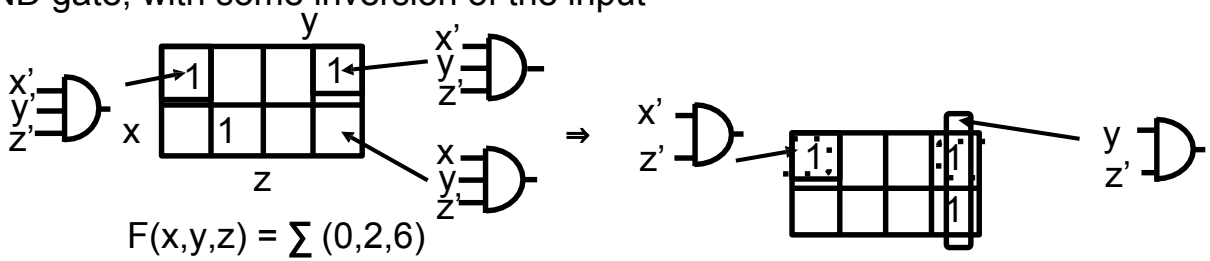
I/P			O/P
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

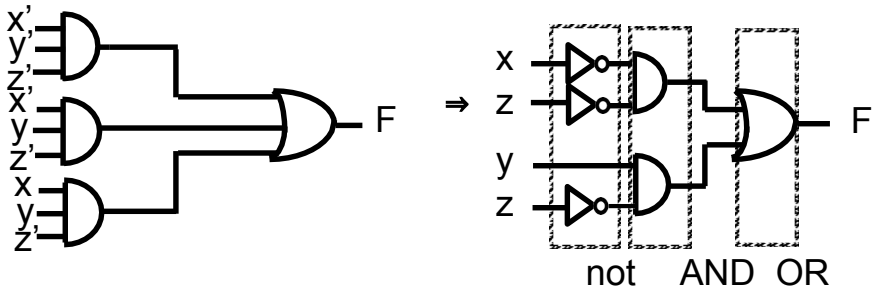


IMPLEMENTATION OF K-MAPS Sum- of- Products Form(SOP)

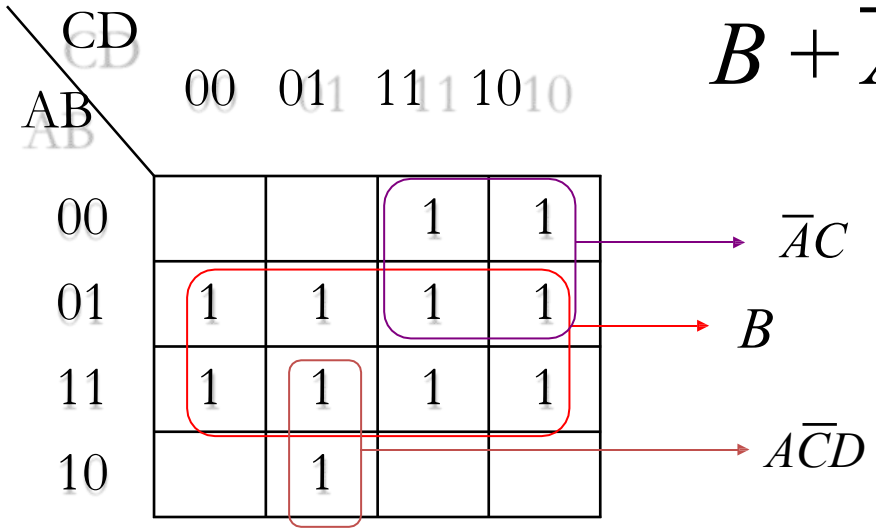
- Logic function represented by a Karnaugh map can be implemented in the form of not-AND-OR

- A cell or a collection of the adjacent 1-cells can be realized by an AND gate, with some inversion of the input variables.





Determining the Minimum SOP Expression from the Map



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

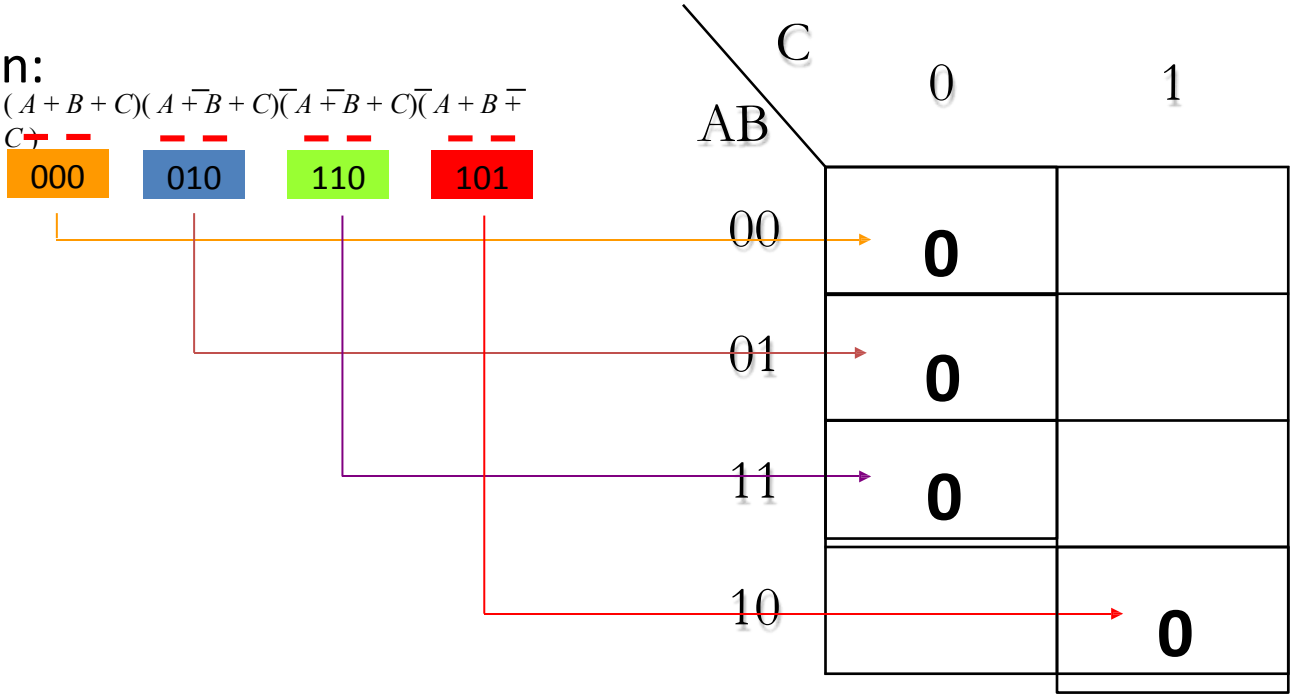
K-Map POS Minimization

- The approaches are much the same (as SOP) except that with POS expression, 0s representing the standard sum terms are placed on the K-map instead of 1s.

Mapping a Standard POS Expression

The expression:

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

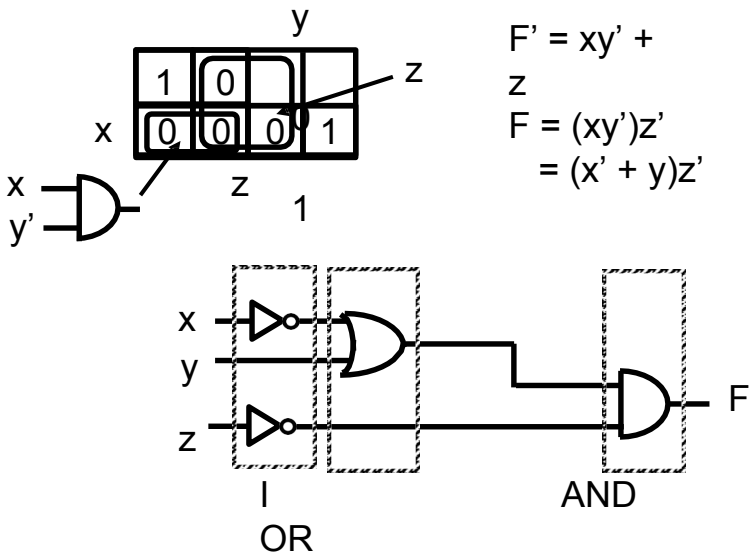


IMPLEMENTATIONOFK-MAPS Product-of- Sums Form(POS)

- Logic function represented by a Karnaugh map can be implemented in the form of I-OR-AND

- If we implement a Karnaugh map using 0-cells, the complement of F, i.e., F', can be obtained. Thus, by complementing F' using DeMorgan's theorem F can be obtained

F(x,y,z) = (0,2,6)



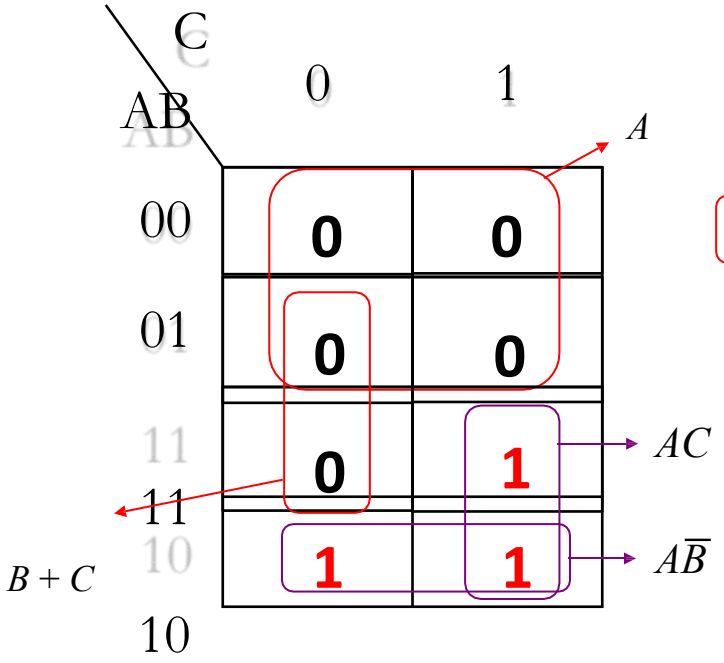
F' = xy' + z

F = (xy')z'

= (x' + y)z'

K-map Simplification of POS Expression

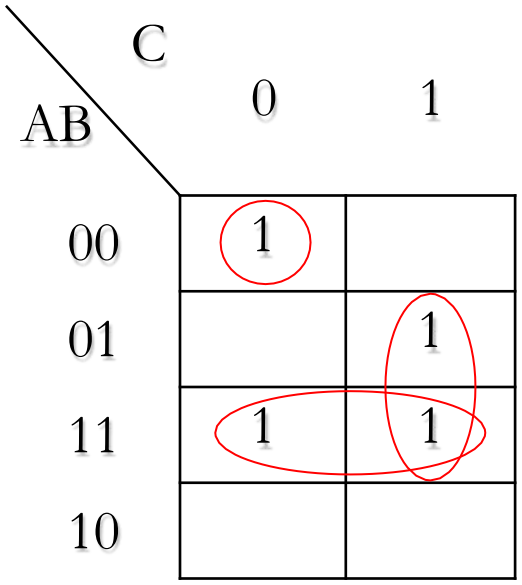
$$(A+B+C)(A+B+\overline{C})(A+\overline{B}+C)(\overline{A}+\overline{B}+\overline{C})(\overline{A}+B+C)$$



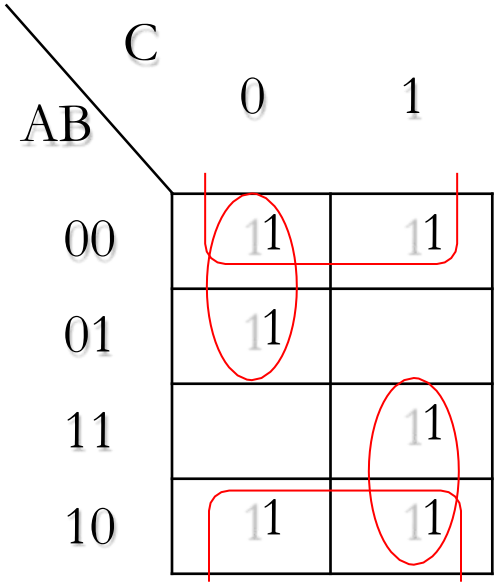
$A(\overline{B}+C)$

$\overline{A}B + AC$

Determining the Minimum SOP Expression from the Map



$$AB + BC + \overline{A}\overline{B}\overline{C}$$



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

Design of combinational digital circuits

- Steps to design a combinational digital circuit:

  - From the problem statement derive the truth table
  - From the truth table derive the unsimplified logic expression
  - Simplify the logic expression
  - From the simplified expression draw the logic circuit

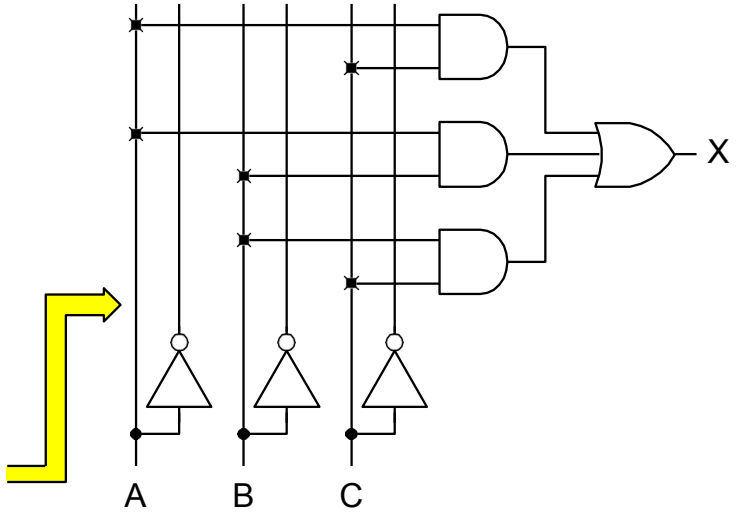
Example: Design a 3-input (A,B,C) digital circuit that will give at its (X) output logic 1 only if the binary number formed at the input has more ones than zeros.

	Inputs ABC	Output X
0	000	0
1	001	0
2	010	0
3	011	1
4	100	0
5	101	1
6	110	1
7	111	1

$X = \sum(3, 5, 6, 7)$

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

$X = AC + AB + BC$



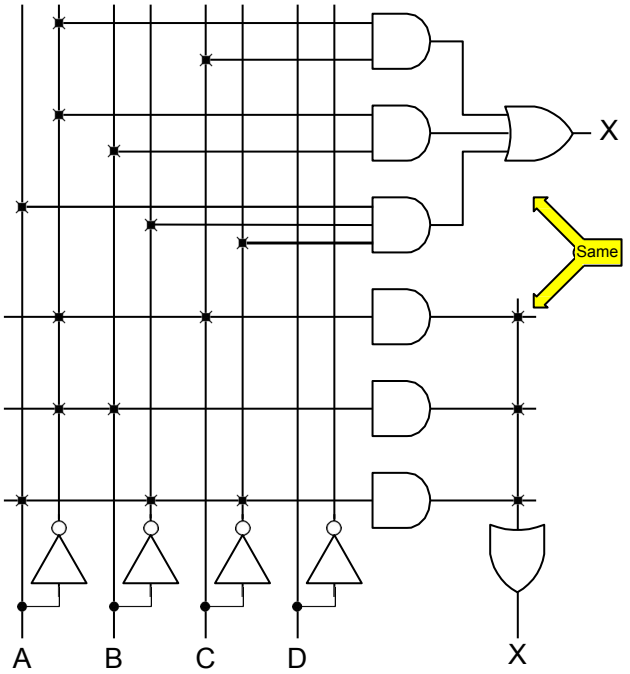
Example: Design a 4-input (A,B,C,D) digital circuit that will give at its output (X) a logic 1 only if the binary number formed at the input is between 2 and 9 (including).

	Inputs ABCD	Output X
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	1
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	1
10	1 0 1 0	0

$X = \sum(2,3,4,5,6,7,8,9)$

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

$X = \overline{A}C + \overline{A}B + A\overline{B}\,\overline{C}$



Quine McClusky method (Tabular Method)

EXAMPLE:

Simplify the Boolean Expression using Quine McClusky method (Tabular Method)

$$F(A, B, C, D) = \sum m(0,1,3,7,8,9,11,15)$$

Convert Decimal Numbers To Binary Numbers

Table 1

DECIMAL NUMBER	EQUIVALENT BINARY NUMBER	MINTERMS
0	0000	m0
1	0001	m1
3	0011	m3
7	0111	m7
8	1000	m8
9	1001	m9
11	1011	m11
15	1111	m15

STEP: 1

Arrange all Minterms according to number of 1 as shown in table 2

TABLE : 2

Group	Minterm No.	IN BINARY			
		A	B	C	D
0	0	0	0	0	0
1	1	0	0	0	1
	8	1	0	0	0
2	3	0	0	1	1
	9	1	0	0	1
3	7	0	1	1	1
	11	1	0	1	1
4	15	1	1	1	1

STEP: 2

Compare each minterm in group 'n' with each minterm in group (n+1) and identify the match pairs. A match pair is a pair of minterms which differ only in one variable. For the variables differ place (-) dash, as shown in Table 3

TABLE : 3

Group	Minterm No.	IN BINARY			
		A	B	C	D
0	(0,1)	0	0	0	-
	(0,8)	-	0	0	0
1	(1,3)	0	0	-	1
	(1,9)	-	0	0	1
	(8,9)	1	0	0	-
2	(3,7)	0	-	1	1
	(3,11)	-	0	1	1
	(9,11)	1	0	-	1
3	(7,15)	-	1	1	1
	(11,15)	1	-	1	1

STEP 3:

Now compare all the pairs of mintermsof table 3 with those in the adjacent groups. As shown in table 4

TABLE : 2

Group	Minterm No.	IN BINARY			
		A	B	C	D
0	0	0	0	0	0
1	1	0	0	0	1
	8	1	0	0	0
2	3	0	0	1	1
	9	1	0	0	1
3	7	0	1	1	1
	11	1	0	1	1
4	15	1	1	1	1

TABLE : 3

Group	Minterm No.	IN BINARY			
		A	B	C	D
0	(0,1)	0	0	0	-
	(0,8)	-	0	0	0
1	(1,3)	0	0	-	1
	(1,9)	-	0	0	1
	(8,9)	1	0	0	-
2	(3,7)	0	-	1	1
	(3,11)	-	0	1	1
	(9,11)	1	0	-	1
3	(7,15)	-	1	1	1
	(11,15)	1	-	1	1

TABLE : 4

Group	Minter m No.	IN BINARY			
		A	B	C	D
1	0,1,8,9	-	0	0	-
	0,8,1,9	-	0	0	-
2	1,3,9,11	-	0	-	1
	~~1,9,3,11~~	~~-~~	~~0~~	~~-~~	~~1~~
3	3,7,11,15	-	-	1	1
	~~3,11,7,15~~	~~-~~	~~-~~	~~1~~	~~1~~

TABLE: 5

GROUP	MINTERMS	BINARY REPRESENTATION				
		A	B	C	D	
1	m_0-m_1-m_8-m_9	-	0	0	-	$\overline{B}\,\overline{C}$
	m_0-m_8-m_1-m_9	-	0	0	-	
2	m_1-m_3-m_9-m_{11}	-	0	-	1	$B'D$
	m_1-m_9-m_3-m_{11}	-	0	-	1	
3	m_3-m_7-m_{11}-m_{15}	-	-	1	1	CD
	m_3-m_{11}-m_7-m_{15}	-	-	1	1	

STEP: 4

Repeat the procedure for grouping. If can group the Quads of minterms in the adjacent groups of table 4 to obtain groups of eight minterms. There are no such matching.

Now prepare Prime Implicant Table as shown in Table 5

TABLE: 6

PI	Minterms group & Boolean representation	GIVEN MINTERMS							
		0	1	3	7	8	9	11	15
√	(0,1,8,9) $B'C'$	X	X			X	X		
	(1,3,9,11) BD		X	X			X	X	
√	(3,7,11,15) CD			X	X			X	X
		√			√	√			√

From table 6 Essential Prime Implicants are $B\overline{C}$ and CD

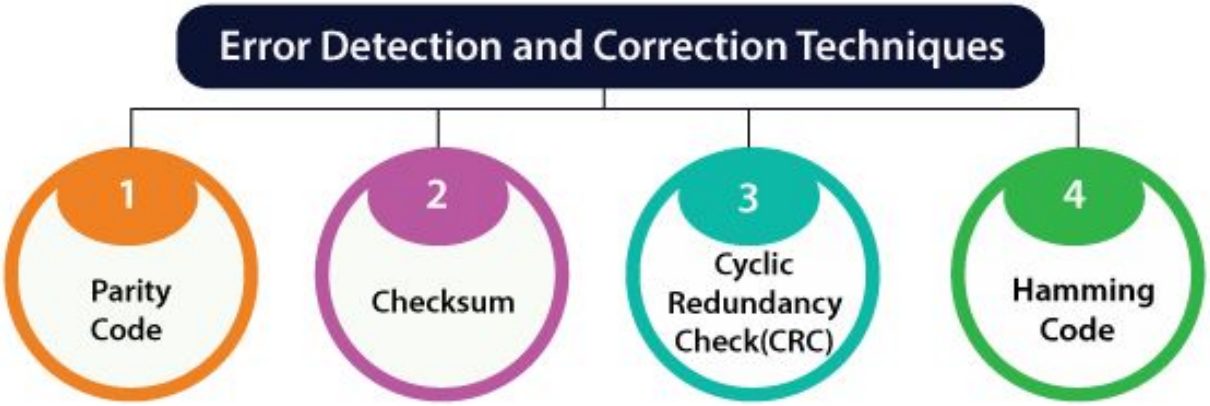
Required Output

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

Error Detecting and Correcting codes

Introduction

- Error detection and correction code plays an important role in the transmission of data from one source to another. The noise also gets added into the data when it transmits from one system to another, which causes errors in the received binary data at other systems. The bits of the data may change(either 0 to 1 or 1 to 0) during transmission.
- It is impossible to avoid the interference of noise, but it is possible to get back the original data. For this purpose, we first need to detect either an error **z** is present or not using error detection codes. If the error is present in the code, then we will correct it with the help of error correction codes.



Introduction

- The dynamic physical interaction of the electrical signals affecting the data path of a memory unit may cause occasional errors in storing and retrieving the binary information. The reliability of a memory unit may be improved by employing error-detecting and error-correcting codes. The most common error detection scheme is the parity bit.
- A parity bit is generated and stored along with the data word in memory. The parity of the word is checked after reading it from memory. The data word is accepted if the parity of the bits read out is correct. If the parity checked results in an inversion, an error is detected, but it cannot be corrected.
- An error-correcting code generates multiple parity check bits that are stored with the data word in memory. Each check bit is a parity over a group of bits in the data word. When the word is read back from memory, the associated parity bits are also read from memory and compared with a new set of check bits generated from the data that have been read.
- If the check bits are correct, no error has occurred.
- A single error occurs when a bit changes in value from 1 to 0 or from 0 to 1 during the write or read operation. If the specific bit in error is identified, then the error can be corrected by complementing the erroneous bit.

Parity bit

- It is used for the purpose of detecting errors during Tx of the binary information.
- A parity bit is an extra bit included with a binary massage to make the number of 1’s either odd or even.
- 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.

Parity generator and Checker

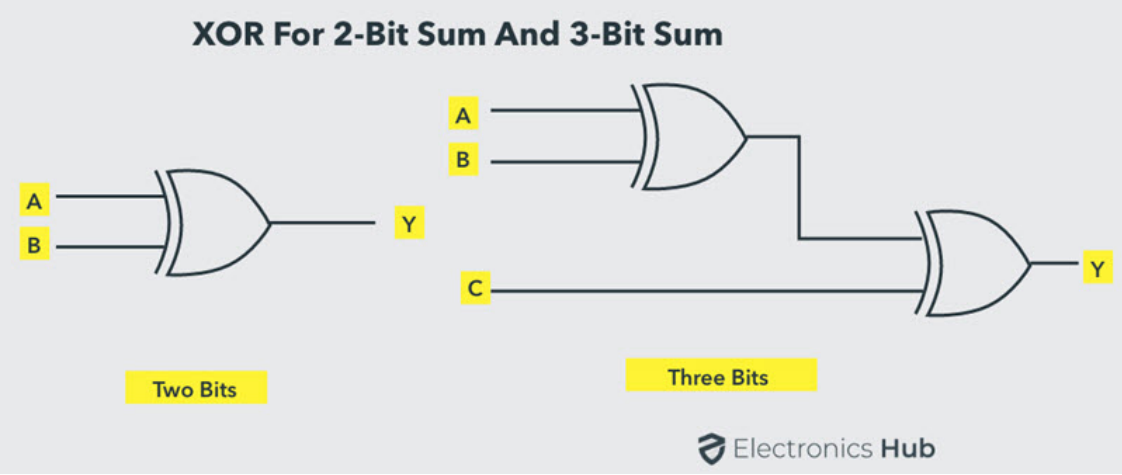
- A Parity Generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called Parity Checker. A combined circuit or device of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data.

EVEN & ODD Parity

- In even parity , the added parity bit will make the total no of 1's even amount. Use XOR
- In odd parity , the added parity bit will make the total no of 1's odd amount. Use XNOR

Even Parity and Odd Parity

- The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1s an even number, whereas in odd parity, the added parity bit will make the total number of 1s an odd number.
- The basic principle involved in the implementation of parity circuits is that sum of odd number of 1s is always 1 and sum of even number of 1s is always 0. Such error detecting and correction can be implemented by using Ex-OR gates (since Ex-OR gate produce zero output when there are even number of inputs).
- To produce two bits sum, one Ex-OR gate is sufficient whereas for adding three bits, two Ex-OR gates are required as shown in below figure.



Courtesy: https://www.electronicshub.org/parity-generator-and-parity-check/

Parity Generator

- It is combinational circuit that accepts an n-1 bit data and generates the additional bit tha to be transmitted with the bit stream. This additional or extra bit is called as a Parity Bit.
- In even parity bit scheme, the parity bit is '0' if there are even number of 1s in the data stream and the parity bit is '1' if there are odd number of 1s in the data stream.
- In odd parity bit scheme, the parity bit is '1' if there are even number of 1s in the data stream and the parity bit is '0' if there are odd number of 1s in the data stream.

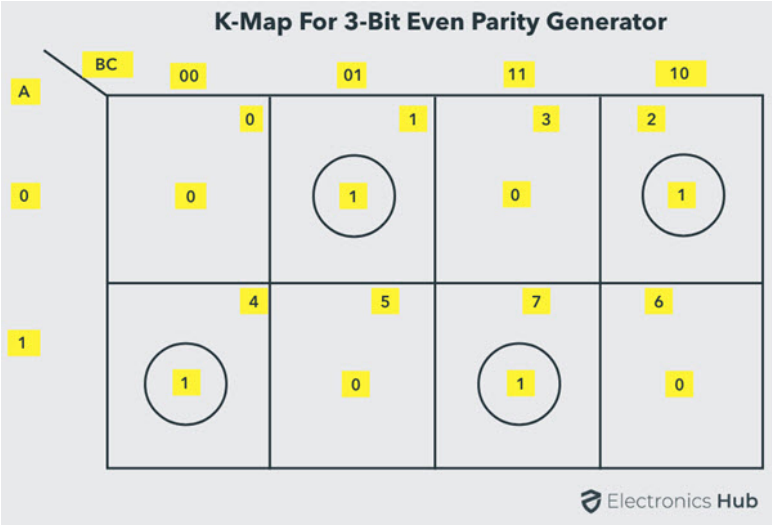
Even Parity Generator

- 3-bit message is to be transmitted with an even parity bit. Let the three inputs A, B and C are applied to the circuit and output bit is the parity bit P. The total number of 1s must be even, to generate the even parity bit P.
- The figure below shows the truth table of even parity generator in which 1 is placed as parity bit in order to make all 1s as even when the number of 1s in the truth table is odd.

3-bit message			Even parity bit generator (P)
A	B	C	Y
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

Courtesy:
https://www.electronicshub.org/parity-generator-and-parity-check/

The K-map simplification for 3-bit message even parity generator is



From the above truth table, the simplified expression of the parity bit can be written as

$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

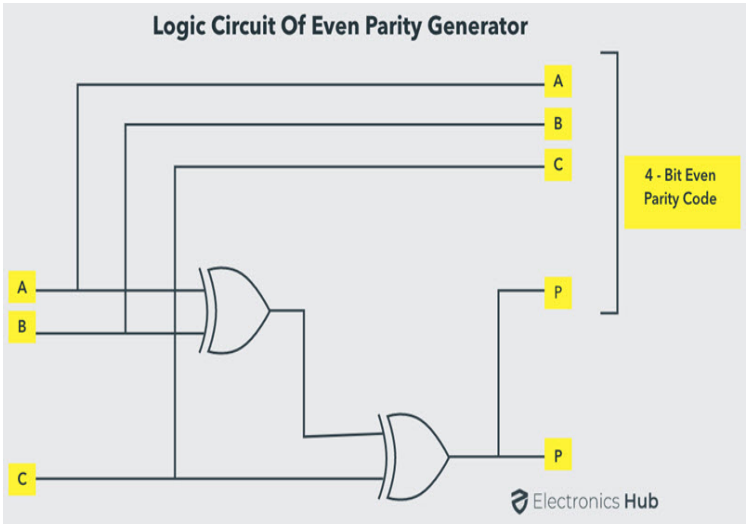
$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

$$= \bar{A}\,(B \oplus C) + A\,(\overline{B \oplus C})$$

$$P = A \oplus B \oplus C$$

The above expression can be implemented by using two Ex-OR gates. The logic diagram of even parity generator with two Ex – OR gates is shown below. The three bit message along with the parity generated by this circuit which is transmitted to the receiving end where parity checker circuit checks whether any error is present or not.

To generate the even parity bit for a 4-bit data, three Ex-OR gates are required to add the 4-bits and their sum will be the parity bit.

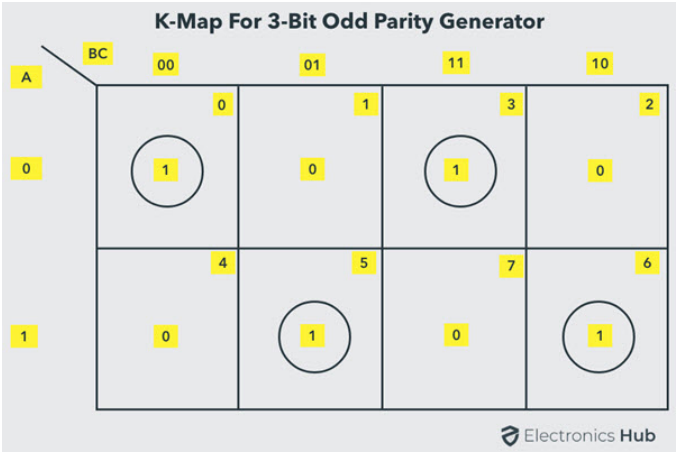


Courtesy: https://www.electronicshub.org/parity-generator-and-parity-check/

Odd Parity Generator

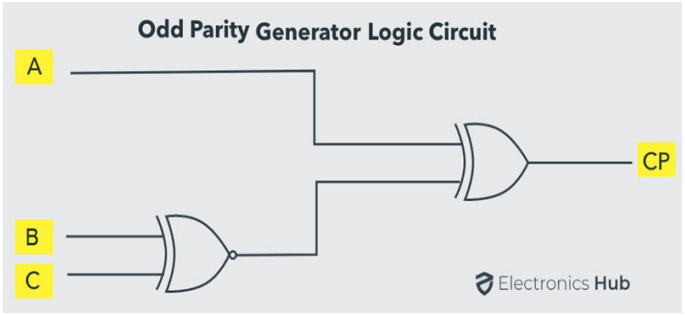
- Let us consider that the 3-bit data is to be transmitted with an odd parity bit. The three inputs are A, B and C and P is the output parity bit. The total number of bits must be odd in order to generate the odd parity bit.
- In the given truth table below, 1 is placed in the parity bit in order to make the total number of bits odd when the total number of 1s in the truth table is even.

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
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	0



The output parity bit expression for this generator circuit is obtained as

$$P = A \oplus (B \oplus C)$$



Courtesy: https://www.electronicshub.org/parity-generator-and-parity-check/

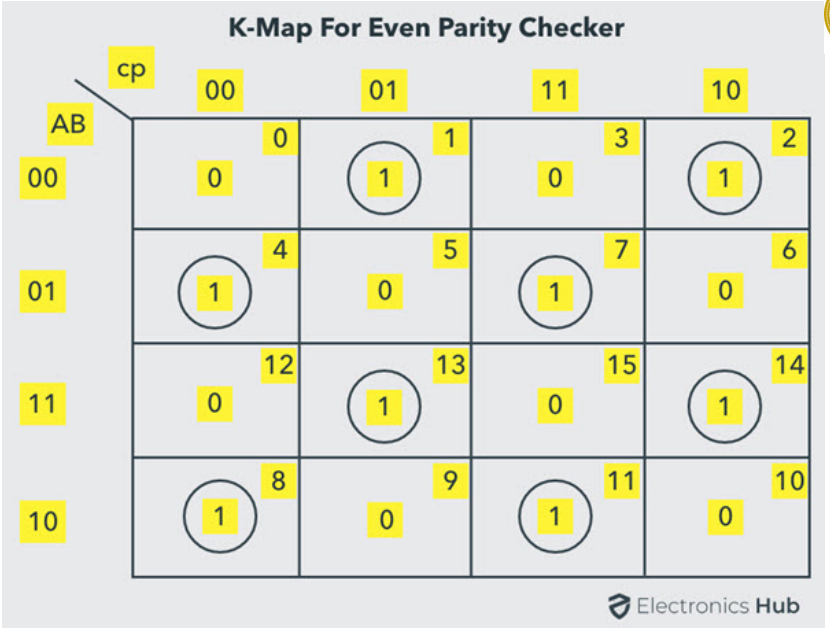
Parity Check

- It is a logic circuit that checks for possible errors in the transmission. This circuit can be an even parity checker or odd parity checker depending on the type of parity generated at the transmission end. When this circuit is used as even parity checker, the number of input bits must always be even.

Even Parity Checker

- Consider that three input message along with even parity bit is generated at the transmitting end. These 4 bits are applied as input to the parity checker circuit, which checks the possibility of error on the data.
- Since the data is transmitted with even parity, four bits received at circuit must have an even number of 1s.
- If any error occurs, the received message consists of odd number of 1s. The output of the parity checker is denoted by PEC (Parity Error Check).
- The below table shows the truth table for the Even Parity Checker in which PEC = 1 if the error occurs, i.e., the four bits received have odd number of 1s and PEC = 0 if no error occurs, i.e., if the 4-bit message has even number of 1s.
-

4-bit received message				Parity error check C_p
A	B	C	P	
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

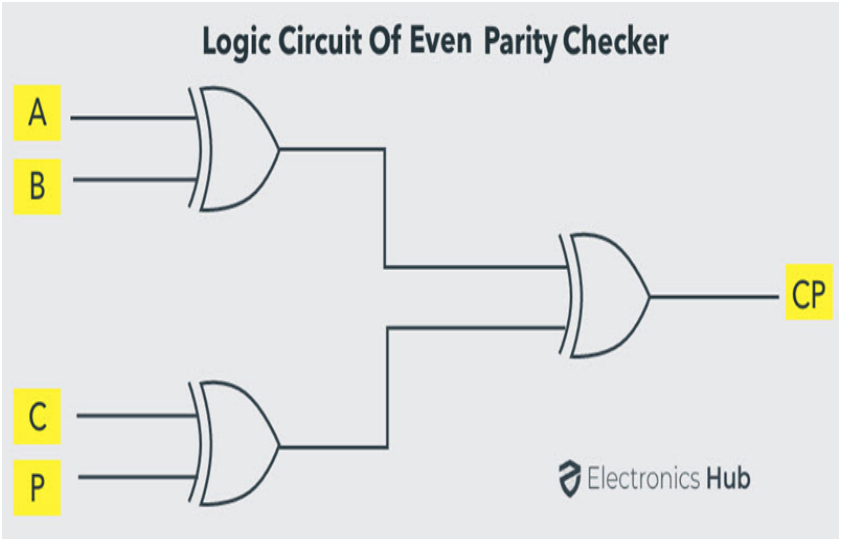


$$PEC = \overline{A}\,\overline{B}\,(\overline{C}P + C\overline{P}) + \overline{A}B(\overline{C}\,\overline{P} + CP) + AB(\overline{C}P + C\overline{P}) + A\overline{B}\,(\overline{C}\,\overline{P} + CP)$$

$$= \overline{A}\,\overline{B}\,(C \oplus P) + \overline{A}B(\overline{C \oplus P}) + AB(C \oplus P) + A\overline{B}\,(\overline{C \oplus P})$$

$$= (\overline{A}\,\overline{B} + AB)(C \oplus P) + (\overline{A}B + A\overline{B})(\overline{C \oplus P})$$

$$= (A \oplus B) \oplus (C \oplus P)$$

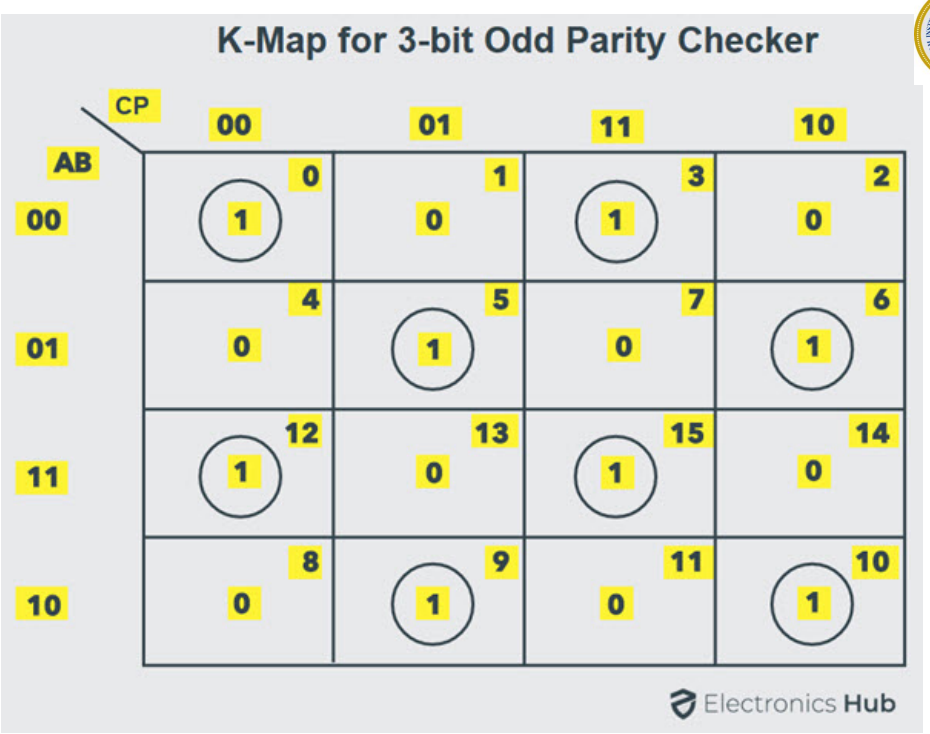


Courtesy: https://www.electronicshub.org/parity-generator-and-parity-check/

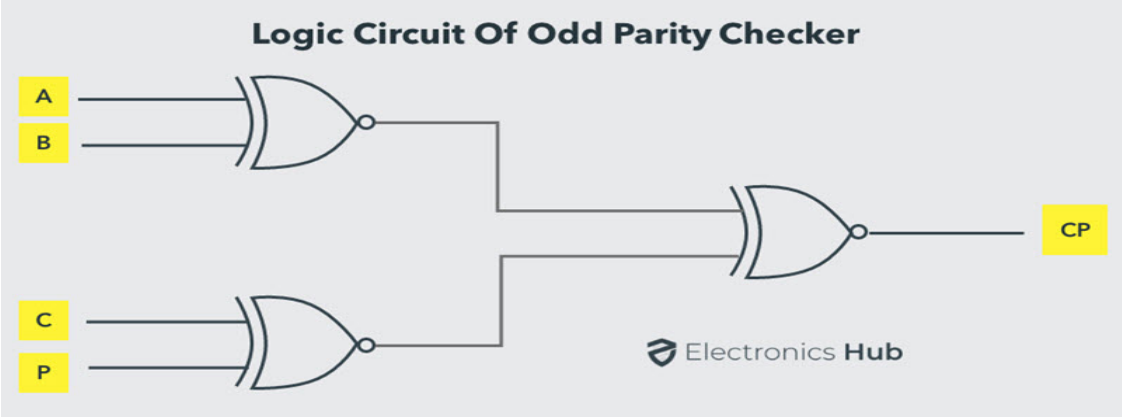
Odd Parity Checker

- Consider that a three bit message along with odd parity bit is transmitted at the transmitting end. Odd parity checker circuit receives these 4 bits and checks whether any error are present in the data.
- If the total number of 1s in the data is odd, then it indicates no error, whereas if the total number of 1s is even then it indicates the error since the data is transmitted with odd parity at transmitting end.
- The below figure shows the truth table for odd parity generator where PEC =1 if the 4-bit message received consists of even number of 1s (hence the error occurred) and PEC= 0 if the message contains odd number of 1s (that means no error).

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR P)



Courtesy: https://www.electronicshub.org/parity-generator-and-parity-check/

Hamming Code

Hamming Code

Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.

In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

Encoding a message by Hamming Code

The procedure used by the sender to encode the message encompasses the following steps −

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Calculating the values of each redundant bit.

Once the redundant bits are embedded within the message, this is sent to the user.

Step 1 − Calculation of the number of redundant bits.

If the message contains mmnumber of data bits, rrnumber of redundant bits are added to it so that mr is able to indicate at least (m + r+ 1) different states. Here, (m + r) indicates location of an error in each of (m + r) bit positions and one additional state indicates no error. Since, rr bits can indicate $2^r r$ states, $2^r r$ must be at least equal to (m + r + 1). Thus the following equation should hold $2^r \geq m+r+1$

Step 2 − Positioning the redundant bits.

The r redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc. They are referred in the rest of this text as r_1 (at position 1), r_2 (at position 2), r_3 (at position 4), r_4 (at position 8) and so on.

Step 3 − Calculating the values of each redundant bit.

The redundant bits are parity bits. A parity bit is an extra bit that makes the number of 1s either even or odd. The two types of parity are −

- **Even Parity** − Here the total number of bits in the message is made even.
- **Odd Parity** − Here the total number of bits in the message is made odd.

Each redundant bit, r_i, is calculated as the parity, generally even parity, based upon its bit position. It covers all bit positions whose binary representation includes a 1 in the i^{th} position except the position of r_i. Thus −

- r_1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)
- r_2 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)
- r_3 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)

Decoding a message in Hamming Code

Once the receiver gets an incoming message, it performs recalculations to detect errors and correct them. The steps for recalculation are −

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Parity checking.
- **Step 4** − Error detection and correction

Step 1 − Calculation of the number of redundant bits

Using the same formula as in encoding, the number of redundant bits are ascertained.

$2^r \geq m + r + 1$ where *m* is the number of data bits and *r* is the number of redundant bits.

Step 2 − Positioning the redundant bits

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

Step 3 − Parity checking

Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of c_1,c_2 ,c_3 ,c_4 etc. Thus

c_1 = parity(1, 3, 5, 7, 9, 11 and so on)

c_2 = parity(2, 3, 6, 7, 10, 11 and so on)

c_3 = parity(4-7, 12-15, 20-23 and so on)

Step 4 − Error detection and correction

The decimal equivalent of the parity bits binary values is calculated. If it is 0, there is no error. Otherwise, the decimal value gives the bit position which has error. For example, if $c_1c_2c_3c_4$ = 1001, it implies that the data bit at position 9, decimal equivalent of 1001, has error. The bit is flipped to get the correct message.

Application of Hamming code

Here are some common applications of using Hemming code:

- Satellites
- Computer Memory
- Modems
- PlasmaCAM
- Open connectors
- Shielding wire
- Embedded Processor

Advantages of Hamming code

- Hamming code method is effective on networks where the data streams are given for the single-bit errors.
- Hamming code not only provides the detection of a bit error but also helps you to indent bit containing error so that it can be corrected.
- The ease of use of hamming codes makes it best them suitable for use in computer memory and single-error correction.

Disadvantages of Hamming code

- Single-bit error detection and correction code. However, if multiple bits are founded error, then the outcome may result in another bit which should be correct to be changed. This can cause the data to be further errored.
- Hamming code algorithm can solve only single bits issues.

Hamming Code

- Hamming code not only provide the detection of a bit error, but also identifies which bit is in error so that it can be corrected.

Number of Parity Bits

$$2^r \geq m + r + 1$$

m -> information bit

r -> parity bit

Case:1

m=4, let r=2 then $2^2 = 4$, 4+2+1=7, 4 ≠7 Not satisfied

Case:1

m=4, let r=3 then $2^3 = 8$, 4+3+1=8, 8 =7 satisfied

Problem-1

Encode the binary word 1011 into seven bit even parity hamming code.

$2^r \geq m + r + 1$

So m= 4 , let r = 3

$2^3 \geq 4 + 3 + 1$

$8 \geq 8$

$$p1 = m3 \oplus m5 \oplus m7 = 1$$

$$p2 = m3 \oplus m6 \oplus m7 = 0$$

$$p4 = m5 \oplus m6 \oplus m7 = 0$$

Bit desinition	m7	m6	m5	p4	m3	p2	p1
Bit location	7	6	5	4	3	2	1
Binary code	111	110	101	100	011	010	001
Information bits	1	0	1		1		
Paity Bit				**0**		**0**	**1**
Coded message	**1**	**0**	**1**	**0**	**1**	**0**	**1**

Problem-2

Encode the binary word 10111 into seven bit odd parity hamming code.

$2^r \geq m + r + 1$

So m= 5 , let r = 4

$2^4 \geq 5 + 4 + 1$

$16 \geq 10$

$$p1 = \overline{m3 \oplus m5 \oplus m7 \oplus m9} = 0$$

$$p2 = \overline{m3 \oplus m6 \oplus m7} = 1$$

$$p4 = \overline{m5 \oplus m6 \oplus m7} = 1$$

$$p8 = \overline{m9} = 0$$

Bit desinition	m9	p8	m7	m6	m5	p4	m3	p2	p1
Bit location	9	8	7	6	5	4	3	2	1
Binary code	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		0	1	1		1		
Paity Bit		**0**				**1**		**1**	**0**
Coded message	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**

Problem-3

Assume that even parity hamming code 0100011 is received. Find error in the message and correct

Bit desinition	m7	m6	m5	p4	m3	p2	p1
Bit location	7	6	5	4	3	2	1
Binary code ??	111	110	101	100	011	010	001
Information bits	0	1	0	0	0	1	1
Correct information	**0**	**1**	**1**	**0**	**0**	**0**	**1**

$c1 = p1 \oplus m3 \oplus m5 \oplus m7 = 1$

$c2 = p2 \oplus m3 \oplus m6 \oplus m7 = 0$

$c4 = p4 \oplus m5 \oplus m6 \oplus m7 = 1$

101 = 5

Invert the bit in 5th location

Example problem 4

Encode a binary word 11001 into the even parity hamming code.

Given, number of data bits, n =5.

To find the number of redundant bits,

Let us try P=4.

The equation is satisfied and so 4 redundant bits are selected.

So, total code bit = n+P = 9

The redundant bits are placed at bit positions 1, 2, 4 and 8.

Construct the bit location table.

Bit Location	9	8	7	6	5	4	3	2	1
Bit designation	D_5	**P_4**	D_4	D_3	D_2	**P_3**	D_1	**P_2**	**P_1**
Binary representation	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		1	0	0		1		
Parity bits		1				1		0	1

To determine the parity bits

For P1: Bit locations 3, 5, 7 and 9 have three 1s. To have even parity, P1 must be 1.

For P2: Bit locations 3, 6, 7 have two 1s. To have even parity, P2 must be 0.

For P3: Bit locations 5, 6, 7 have one 1s. To have even parity, P3 must be 1.

For P4: Bit locations 8, 9 have one 1s. To have even parity, P2 must be 1.

Thus the encoded 9-bit hamming code is 111001101.

Example problem 5

Let us assume the even parity hamming code from the above example (111001101) is transmitted and the received code is (110001101). Now from the received code, let us detect and correct the error.

To detect the error, let us construct the bit location table.

Bit Location	9	8	7	6	5	4	3	2	1
Bit designation	D5	**P4**	D4	D3	D2	**P3**	D1	**P2**	**P1**
Binary representation	1001	1000	0111	0110	0101	0100	0011	0010	0001
Received code	1	1	0	0	0	1	1	0	1

Checking the parity bits

For P1 : Check the locations 1, 3, 5, 7, 9. There is three 1s in this group, which is wrong for even parity. Hence the bit value for P1 is 1.

For P2 : Check the locations 2, 3, 6, 7. There is one 1 in this group, which is wrong for even parity. Hence the bit value for P2 is 1.

For P3 : Check the locations 3, 5, 6, 7. There is one 1 in this group, which is wrong for even parity. Hence the bit value for P3 is 1.

For P4 : Check the locations 8, 9. There are two 1s in this group, which is correct for even parity. Hence the bit value for P4 is 0.

The resultant binary word is 0111. It corresponds to the bit location 7 in the above table. The error is detected in the data bit D4. The error is 0 and it should be changed to 1. **Thus the corrected code is 111001101**.


THANK-YOU