

Fundamentals of Digital Logic
By Motegaonkar V. M.

NUMBER SYSTEM AND CODES

Contents

1.1 Number system types

1.2 Conversion from one number system to another number system

1.3 Binary arithmetic

1.4 Complementation method

1.5 Codes

1.6 Error detecting and correcting code

1.1 Number system types

What is Number system?

It is a system in which various symbols are used to represent and track the count of the things.

E.g. to count days between consecutive rainfalls.

There are two types of number system

1. Non-positional number system
2. Positional number system

Non-positional number system

In this system same type of symbols are used to represent any number, and the position of symbol does not have any value.

E.g.: 1 - I

2 - II

3- III

I

II

II

Positional number system

In this system different symbols are used to represent a number and place of the symbol has value.

E.g.: 2 , 145, 1E

25 and 52

- Binary number system
- Decimal number system
- Octal number system
- Hexadecimal number system

Binary number system

The number system with base two is known as binary number system. In this number system only two symbols are used to represent any number .and these symbols(0,1) are known as bits.

E.g. $(110)_2$

$$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$1 \times 4 + 1 \times 2 + 0$$

$$4 + 2 + 0$$

$$6$$

Decimal number system

The number system with base ten is known as decimal number system. In this number system ten symbols are used to represent any number .and these symbols(0,1,2,3,4,5,6,7,8,9) are known as digits.

E.g. $(2461)_{10}$

$$2 \times 10^3 + 4 \times 10^2 + 6 \times 10^1 + 1 \times 10^0$$

$$2 \times 1000 + 4 \times 100 + 6 \times 10 + 1 \times 1$$

$$2000 + 400 + 60 + 1$$

$$2461$$

Octal number system

The number system with base eight is known as octal number system. In this number system eight symbols are used to represent any number .and these symbols are (0,1,2,3,4,5,6,7)

E.g. :- $(436)_8$

Hexadecimal number system

The number system with base sixteen is known as hexadecimal number system. In this number system 16 symbols are used to represent any number .and these symbols are digit from 0 to 9 and letters from A to F, these letters are used to represent numbers from 10 to 15

Digits- 0 to 9 (0,1,2,3,4,5,6,7,8,9) = 10 Digits

Letters- A to F A-10, B-11, C-12, D-13, E-14 and F - 15 = 6 Letters

Total symbols = Digits + letter
= 10 + 6 = 16

E.g. $(9A)_{16}$

1.2 Conversion from one number system to another number system


Conversion from Decimal to Binary with Steps

1. Write the given decimal number
2. If the given decimal number is less than 2 the binary number is the same.
3. If the decimal number is greater than 1 then divide the number by 2 .
4. Note the remainder we get after division
5. Repeat step 3 and 4 with the quotient till it is less than 2
6. Now, write the remainders in reverse order(bottom to top)
7. The resultant is the equivalent binary number to the given decimal number.

Decimal to Binary Conversion

Example: 156_{10}


2	156	
2	78	0
2	39	0
2	19	1
2	9	1
2	4	1
2	2	0
2	1	0
	0	1



$$156_{10} = 10011100_2$$

Decimal number : 17

2	17	1
2	8	0
2	4	0
2	2	0
	1	



Binary number: 10001

Conversion from Decimal to Octal with Steps

1. Write the given decimal number
2. If the given decimal number is less than 8 the octal number is the same.
3. If the decimal number is greater than 7 then divide the number by 8.
4. Note the remainder we get after division
5. Repeat step 3 and 4 with the quotient till it is less than 8
6. Now, write the remainders in reverse order(bottom to top)
7. The resultant is the equivalent octal number to the given decimal number.

Decimal to Octal Conversion

Result

Decimal Number is : **(12345)₁₀**

8	12345	1	LSB
8	1543	7	
8	192	0	
8	24	0	
	3	3	MSB

Octal Number is
(30071)₈

Conversion from Decimal to Hexadecimal with Steps

1. Write the given decimal number
2. If the given decimal number is less than 16 the hexadecimal number is the same.
3. If the decimal number is greater than 15 then divide the number by 16
4. Note the remainder we get after division
5. Repeat step 3 and 4 with the quotient till it is less than 16
6. Now write the remainders in reverse order(bottom to top)
7. The resultant is the equivalent octal number to the given decimal number.

Decimal to Hexadecimal Conversion

Result

Example 1

Decimal Number is : $(12345)_{10}$

Hexadecimal Number is
 $(3039)_{16}$

16	12345
16	771
16	48
8	3

9	LSB
3	
0	
3	MSB

1.3 Binary Arithmetic

Binary Addition Rules

$1 + 1 = 0$ with carry 1

$1 + 0 = 1$

$0 + 1 = 1$

$0 + 0 = 0$

Examples

- $10001 + 11101 = 101110$:

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

- $101101 + 11001 = 1000110$:

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

- $1011001 + 111010 = 10010011$:

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

- $1110 + 1111 = 11101$:

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

- $10111 + 110101 = 1001100$:

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

- $11011 + 1001010 = 1100101$:

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

Binary Subtraction Rules

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

$$0 - 1 = 1 \text{ with borrow } 1$$

Examples

- $1011011 - 10010 = 1001001$:

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

- $1010110 - 101010 = 101100$:

$$\begin{array}{r} 0\quad\quad 0 \\ \cancel{1}\ 10\ \cancel{1}\ 10\ 1\ 1\ 0 \\ -\quad\quad 1\ 0\ 1\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

- $1000101 - 101100 = 11001$:

$$\begin{array}{r} 0\ 1\ 1 \\ \cancel{1}\ \cancel{1}0\ \cancel{1}0\ 10\ 1\ 0\ 1 \\ -\quad\quad 1\ 0\ 1\ 1\ 0\ 0 \\ \hline 1\ 1\ 0\ 0\ 1 \end{array}$$

- $100010110 - 1111010 = 10011100$:

$$\begin{array}{r} 0\ 1\ 1\ 1\ 10 \\ \cancel{1}\ \cancel{1}0\ \cancel{1}0\ \cancel{1}0\ \cancel{1}\ 10\ 1\ 1\ 0 \\ -\quad\quad 1\ 1\ 1\ 1\ 0\ 1\ 0 \\ \hline 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \end{array}$$

- $101101 - 100111 = 110$:

$$\begin{array}{r} 0\ 10 \\ 1\ 0\ \cancel{1}\ \cancel{1}\ 10\ 1 \\ -\quad 1\ 0\ 0\ 1\ 1\ 1 \\ \hline 1\ 1\ 0 \end{array}$$

- $1110110 - 1010111 = 11111$:

$$\begin{array}{r} 0\ 10\ 1\ 10\ 10 \\ 1\ \cancel{1}\ \cancel{1}\ \cancel{1}0\ \cancel{1}\ \cancel{1}\ 10 \\ -\quad 1\ 0\ 1\ 0\ 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1 \end{array}$$

Binary Multiplication Rules

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

$$0 \times 0 = 0$$

Example

$$\begin{array}{r} 101001 \\ \times 110 \\ \hline 000000 \\ 101001 \\ + 101001 \\ \hline 11110110 \end{array}$$

Binary division rules

$$1 \div 1 = 1$$

$$1 \div 0 = 0$$

$$0 \div 1 = \text{undefined}$$

$$0 \div 0 = \text{undefined}$$

Example

$$\begin{array}{r} 10 \overline{) 1111100} \quad (111110) \\ \underline{(-) 10} \\ 11 \\ \underline{(-) 10} \\ 11 \\ \underline{(-) 10} \\ 11 \\ \underline{(-) 10} \\ 10 \\ \underline{10} \\ 00 \\ \underline{00} \\ 00 \\ \underline{00} \end{array}$$

1.4 Complementation method

1's Complement

In a given binary number if 1 is replaced by zero and zero is replaced by 1 then we get 1's complement of that number.

E.g. :

(10011)

1's complement is (01100)

2's complement

If we add 1 in 1's complement then we get 2's complement ie 1's complement plus 1 is known as 2's complement.

E.g. (10110)

Step 1:

Take 1's complement of given number 01001

Step 2:

2's complement = 1's complement + 1
= (01001 + 1)
= (01010)

1.5 Codes

BCD Code

The binary coded decimal (BCD) is a type of binary code used to represent a given decimal number in an equivalent binary form using (8421). It is a 4 bit code therefor 16 different symbols are represented in BCD code.

BCD code is also known as 8421 code.

In BCD code numbers from 0 to 9 and alphabets from A to Z are as given below.

Numbers from 0 to 9

DECIMAL	BCD
	8 4 2 1
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Alphabets from A to Z

Character	BCD Code		Character	BCD Code	
	Zone	Digit		Zone	Digit
A	11	0001	N	10	0101
B	11	0010	O	10	0110
C	11	0011	P	10	0111
D	11	0100	Q	10	1000
E	11	0101	R	10	1001
F	11	0110	S	01	0010
G	11	0111	T	01	0011
H	11	1000	U	01	0100
I	11	1001	V	01	0101
J	10	0001	W	01	0110
K	10	0010	X	01	0111
L	10	0011	Y	01	1000
M	10	0100	Z	01	1001

Excess-3 code

The excess-3 code (or XS3) is a non-weighted code used to express decimal numbers. It is a self-complementary binary coded decimal

It is written using BCD code as, excess -3 code is equal to BCD +3.

Decimal number from 0 to 9 in excess 3 code

Decimal Digit	BCD Code	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Gray code

It is a non-weighted code, and it is written using binary code

The gray code using binary is as given below..

Conversion of Binary to Gray Code

Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111

1.6 Error detecting and correcting code

Parity bit

A parity bit is a check bit, which is added to a block of data for error detection purposes. The value of the parity bit is assigned either 0 or 1 that makes the number of 1s in the data block either even or odd depending upon the type of parity. Parity check is suitable for single bit error detection only.

The two types of parity checking are

- **Even Parity**
- **Odd Parity**

Error detection by parity bit

Sender's End: While creating a frame, the sender counts the number of 1s in it and adds the parity bit in following way

- **In case of even parity:** If number of 1s is even, parity bit value is 0. If number of 1s is odd, parity bit value is 1.
- **In case of odd parity:** If number of 1s is odd, parity bit value is 0. If number of 1s is even, parity bit value is 1.

Receiver's End: On receiving a frame, the receiver counts the number of 1s in it. In case of even parity check, if the count of 1s is even, the frame is accepted, otherwise it is rejected. In case of odd parity check, if the count of 1s is odd, the frame is accepted, otherwise it is rejected.

Example

1. Set even parity for data bits (1100101)

Data bits	1100101
-----------	---------

Data with even parity	11001010
-----------------------	----------

2. Set even parity for data bits (1110100)

Data bits	1110100
-----------	---------

Data with o parity	11101001
--------------------	----------

Hamming Code

Hamming code was developed by R.W. hamming in the year 1951. This code is used to detect and correct error in digital data. Hence it is also known as error detecting and correcting code.

The data format for hamming code is : D7 D6 D5 P4 D3 P2 P1

Parity bits are :

P1 (P1, D3, D5, D7)

P2 (P2, D3, D6, D7)

P4 (P4, D5, D6, D7)

Example

Detect and correct the error for receiving data bit (1011011) using hamming code, use even parity.

First, we need to detect whether there are any errors in this received hamming code.

Step 1: For checking parity bit P1, use check one and skip one method, which means, starting from P1 and then skip P2, take D3 then skip P4 then take D5, and then skip D6 and take D7, this way we will have the following bits,

Data format is -----

D7 D6 D5 P4 D3 P2 P1

1 0 1 1 0 1 1

P1 (P1,D3,D5,D7)

Put the value of parity bit and data bit.(1 ,0 ,1, 1)

As we can observe the total number of bits are odd so we will write the value of parity bit as **P1 = 1**. This means there is an **error**.

Step 2: Check for P2 but while checking for P2, But remember since we are checking for P2, so we have to start our count from P2

P2 (P2,D3,D6,D7)

(1 0 0 1)

Number of 1's are two (even) therefore parity bit $P2 = 0$

As we can observe that the number of 1's are even, then we will write the value of **P2 = 0**. This means there is **no error**.

Step 3: Check for P4 but while checking for P4, we will start from P4 which will give us the following data bits.(P1 & P2 should not be considered)therefore P4 is,

P4 (P4, D5, D6,D7)

P4 (1, 1, 0, 1)

Number of 1's are odd therefore parity bit **P4 = 1**

Write the value of parity bit P4, P2, P1

P4	P2	P1
1	0	1

Now we have to determine the decimal value of this error word 101 which is 5

There is error in 5th bit. Hence correct word using Hamming code is (1001011).

Fundamentals of Digital Logic
By Moteagaonkar V. M.

LOGIC GATES AND LOGIC EQUATION SIMPLIFICATION WITH K-MAP

Contents

2.1 Introduction to signals

2.2 Logic Gates

2.3 De-Morgan's theorems

2.4 Standard Representation of logical functions in SOP & POS form

2.5 Simplification of logical function using K-map

2.1 Introduction to signals

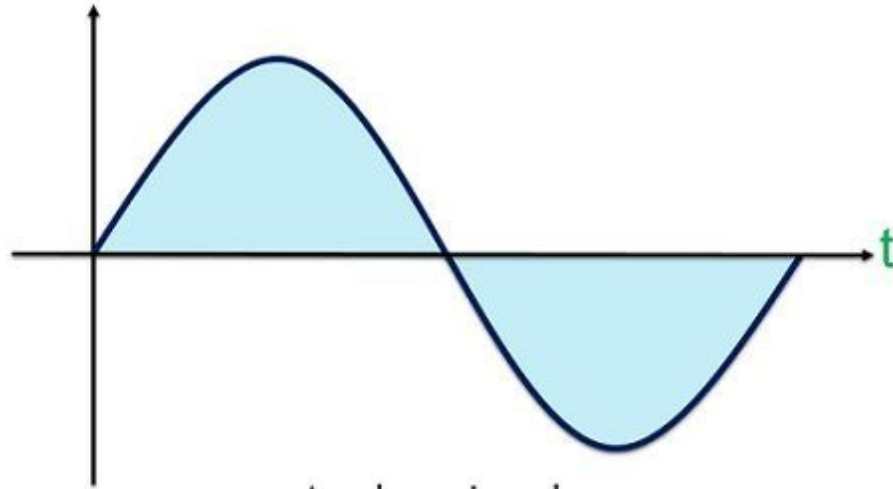
What is signal?

A signal is an electromagnetic or electrical current that is used for carrying data from one system or network to another. There are two types of signals:

1. Analog signal
2. Digital signal

Analog signal

It is a continuous signal. Analog signal uses a continuous range of values that help you to represent information. Its representation looks like a wave.

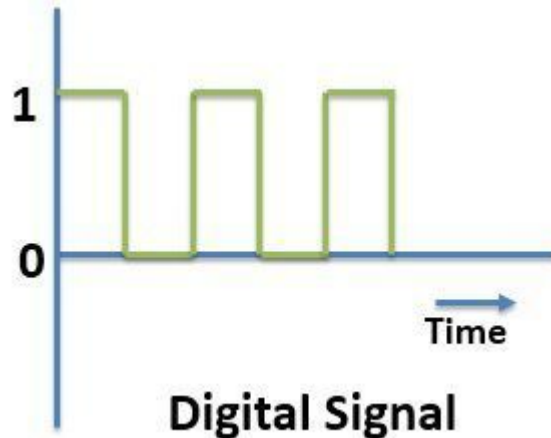


Analog signal

Digital signal

It is a signal that is being used to represent data as a sequence of discrete values; at any given time it can only take on, at most, one of a finite number of values.

Digital signal has only two states 0 and 1 state. '0' is known as OFF or low state and '1' is known as ON state.



2.2 Logic Gates

Logic gates

It is an electronic or logic circuit having one or more then one inputs and only one output. Logic gates are the basic building blocks of any digital circuit.

There are three types of logic gates:

1. Basic gate
2. Universal gate
3. Special purpose gate

Basic gates

It is an electronic circuit which is used to perform basic operation is known as basic gate. There are three types of basic gates:

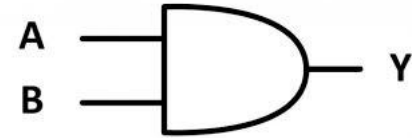
1. AND gate
2. OR gate
3. NOT gate

AND gate

It is a basic gate which is used to perform AND (*) operation.

An AND gate has two or more inputs and a single output. In this gate, the output is 1 when all the inputs are 1. In other words, the output is high when all the inputs are high. The most commonly used symbol for an AND gate is as shown above with inputs A and B , output Y.

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



$$Y = A.B$$

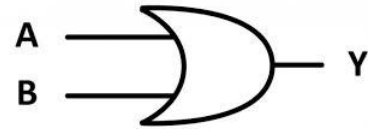
OR gate

It is a basic gate which is used to perform OR(+) operation.

The OR gate is a digital logic gate with minimum 2 inputs and one output. The output of the OR gate is true only when one or more inputs are true.

If all the inputs of the gate are false, only then the output of the is false. The symbol and truth table of an OR gate with two inputs is shown below

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



$$Y = A+B$$

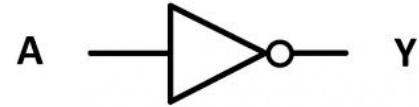
NOT Gate

It is a basic gate which is used to perform NOT(complementation) operation.

The NOT gate is a digital logic gate with one input and one output that operates an inverter operation of the input. The output of the NOT gate is the reverse of the input. When the input of the NOT gate is true then the output will be false and vice versa.

By using this gate, we can implement NOR and NAND gates

Input	Output
A	Y
0	1
1	0



$$Y = \overline{A}$$

Universal gates

NAND and NOR are known as universal gate because using either NAND or NOR we can implement all the three basic gates. There are two special purpose gates:

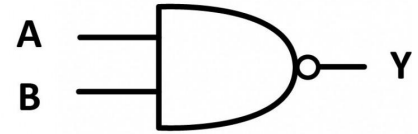
1. NAND Gate
2. NOR Gate

NAND Gate

It is a universal gate that is used to perform NAND (AND +NOT) operation.

The output of NAND gate is 1 when any one input or more than one input are 0.

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



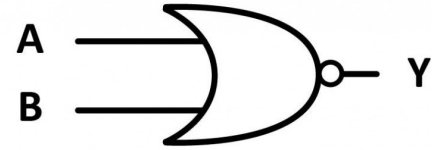
$$Y = \overline{A.B}$$

NOR Gate

It is a universal gate that is used to perform NOR (OR +NOT) operation.

The output of NOR gate is 1 when all the inputs are 0.

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



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

Special purpose gate

X-OR and X-NOR are known as special purpose gate, because it is used in digital circuits to reduce hardware part of some digital circuits. There are two special types of special purpose gate

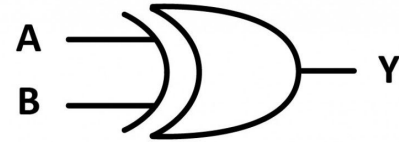
1. X-OR Gate
2. X-NOR Gate,

X-OR gate

It is a special purpose gate which is used to perform X-OR operation.

XOR gate (sometimes EOR, or EXOR and pronounced as Exclusive OR) is a digital logic gate that gives a true (1 or HIGH) output when the number of true inputs is odd. An XOR gate implements an exclusive or; that is, a true output results if one, and only one, of the inputs to the gate is true.

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



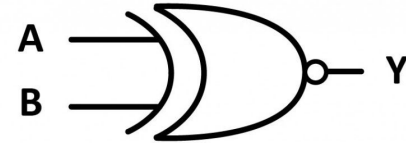
$$Y = A \oplus B$$

X-NOR Gate

It is a special purpose gate which is used to perform X-NOR operation.

is a digital logic gate, that performs X-NOR operation ,that is the logical complement of the Exclusive OR (XOR) gate.

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

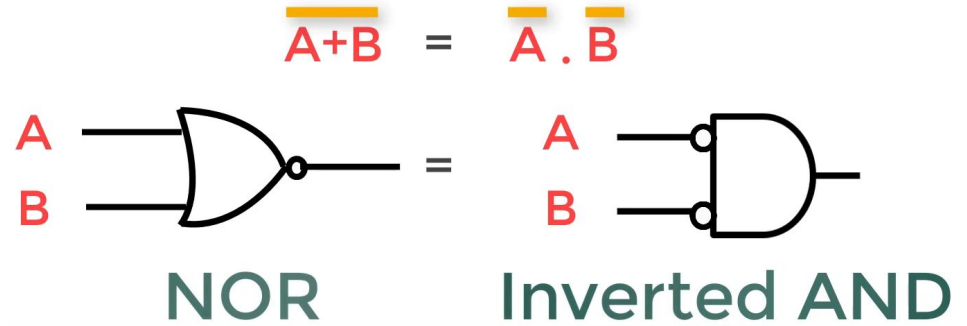


$$Y = \overline{A \oplus B}$$

2.3 De-Morgan's theorems

De-Morgan's first theorem

It states that the complement of sum is equal to the product of their individual complements.

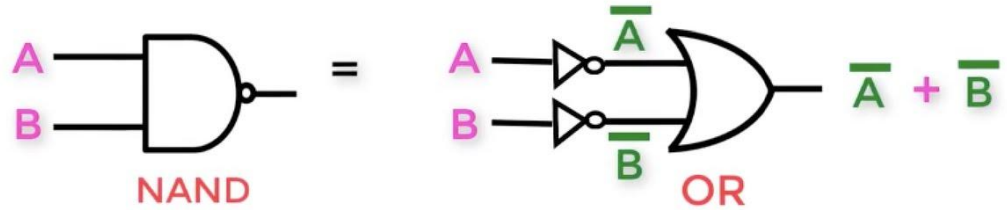


A	B	$\overline{A+B}$	\overline{A}	\overline{B}	$\overline{A} \cdot \overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

De-Morgan's second theorem

It states that the complement of product is equal to the sum of their individual complements.

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



A	B	$\overline{A \cdot B}$	\overline{A}	\overline{B}	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0