Development of Complex Curricula for Molecular Bionics and Infobionics Programs within a consortial* framework**

Consortium leader

PETER PAZMANY CATHOLIC UNIVERSITY

Consortium members

SEMMELWEIS UNIVERSITY, DIALOG CAMPUS PUBLISHER

The Project has been realised with the support of the European Union and has been co-financed by the European Social Fund ***

**Molekuláris bionika és Infobionika Szakok tananyagának komplex fejlesztése konzorciumi keretben

***A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

ELECTRICAL MEASUREMENTS

(Elektronikai alapmérések)

Logic systems, binary system and basic operations

(Logikai rendszerek, bináris számábrázolás és alapműveletek)

Dr. Cserey György

Information

- The set of characters of information representation (conventional signs, symbology), and font context, rules are called code. The conventional rule of transformation is called encoding.
- Deciphering and transformation to greater equipment need is called decoding.
- The unit of information is „bit”, meaning „1”-> yes „0”-> No
- The byte content of its 8 bit series can be
  - A number -> 0 … 255
  - A character -> ‘0’…’9’ ‘A’…’Z’ …
  - An analog value (ADC DAC) (sampling, quantization)

Numeral systems

- Binary, decimal, hexadecimal (0000 – 1111, 0 – 16, 0 – F)
- The procedure when a unit of information is assigned to a group of bits is called encoding.
- The rule that transforms into binary numerical representation is called binary code.
- The description of the rule that transforms letters, numbers and other characters is called alphanumeric code.
- Information can only be processed and transmitted in an encoded form, otherwise neither the computer, nor people would understand it.
- Information, in digital technology, is usually in binary code.

ASCII code table

ASCII code words	Upper three bits							
Lower four bits	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	´	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	}
1101	CR	GS	-	=	M	]	m	|
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Codes

Bin/BCD		Gray	Bin/BCD		Gray
• 0000	—	0000	• 0101	—	0111
• 0001	—	0001	• 0110	—	0101
• 0010	—	0011	• 0111	—	0100
• 0011	—	0010	• 1000	—	1100
• 0100	—	0110	• 1001	—	1101

2659 -> 0010 0110 0101 1001 more complicated calculation, but easier representation

The speciality of the Gray code is that only one number changes every time, so it is great to characterize rotations and displacements.

Unsigned code

- Depending of the number of the bits, how many states can be coded by a word?
- The possible states of a word is 2^n, where n is the number of bits.
- For the differentiation of X states, we need $n = \log_2 X$ bits.
- The more detailed we want to show the information, the more bits we need. Of course there is not only binary code information, but the quantity of a differently coded information can also be characterized by bits.
- Binary code
  - 1 byte 0 … 255 1111 1111
  - 2 byte 0 … 65 535 1111 1111 1111 1111
  - 4 byte 0 … 4 294 967 295

$$v = \sum_i c_i 2^i$$

Hexadecimal representation

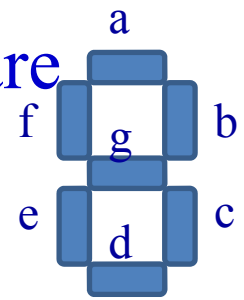
- Each and every hexadecimal number can be substituted by a tetrad. From a hexadecimal number we can get a binary number if we formally write the tetrads corresponding to each and every hexadecimal number in a sequence.
  - 1 byte 0 … 0FFH 0 … 0xff
  - 2 byte 0 … 0FFFFH 0 … 0xffff
  - 3 byte 0 … 0FFFFFFH 0 … 0xffffff
  - 4 byte 0 … 0FFFFFFFFH 0 … 0xffffffff

Hex	Tetrád
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

$$v = \sum_i k_i 16^i$$

BCD representation

- Decimal numbers are often represented on a 7 segment display. The seven segments are basically seven two-state structural elements, so in theory it could represent 27 different figures. But only 10 are used, because 10 types of decimal numbers exist.



- 1 byte 0 … 99H 0 … 0x99
  - 2 byte 0 … 9999H 0 … 0x9999
  - 3 byte 0 … 999999H 0 … 0x999999
  - 4 byte 0 … 99999999H 0 … 0x99999999

$$v = \sum_i k_i 10^i$$

Binary signed numbers

- We are only dealing with operations on integers, and within that, only addition. This is because with addition and modifications, all mathematical operations can be done.
- The usual sign of negative numbers is the negative sign character. But when we use binary numbers, we do not want to use sign characters, because this would mean that we are not using only two (0 and 1) characters any more. We would give up the greatest advantage of binary representation. Because of this, when we use binary numbers, we represent the sign of the number with the binary numbers as well, in the following way: if the first bit of the number is 1, then it is negative, if it is 0, then we are talking about a positive number. This way, the signed numbers only consist of two characters as well: 0 and 1.

Signed (negative) numbers I.

Complement code representation

- Let us code the sign of the number using 1 bit (if the value is 1, then it is a negative number, if it is 0, then it is positive)
- Maximum represented value in both ranges : 2^(N-1)-1,where N is the number of bits ➔we have +0 and -0 this is not very good

Example: Decimal +29647

Binary representation: 111 0011 1100 1111 Signed representation: 0111 0011 1100 1111

Decimal -5748

Binary representation : 1 0110 0111 0100 Signed representation: 1001 0110 0111 0100

Signed (negative) numbers II.

2's complement code representation

The bit with the highest digit is the sign

- If == 1 ➔ the number is negative If == 0 ➔ the number is positive

Binary code

- 1 byte -128 … 127
- 2 byte -32 768 … 32 767
- 3 byte -8 388 608 … 8 388 608
- 4 byte -2 147 483 648 … 2 147 483 647
- 0111 1111 1111 1111 = 32767 the biggest positive number represented on two bytes.

Signed (negative) numbers II.

- In the case of a positive number, binary numbers fill the bit positions in a way that the imaginary binary point follows the bit positions kept for representation.
- Addition and subtraction with 2's complement code
  - Operands should be added
  - The transmission created in the addition is negligible
  - If the sign bit of the result is 0, the result is positive, and its value gives the absolute value of the actual amount.
  - If the sign bit of the result is 1, the result is negative, and its value gives the 2's complement of the absolute value of the actual amount.

Signed (negative) numbers II.

- In the case of a negative number,we represent the absolute value the same way, then we invert every bit piece by piece, then we add 1 (binary) to the result.
- 1000 0000 0000 0000 = -32768
  the biggest negative number representable on two bytes.
- The advantage of representing signed integer numbers in a 2's complement code is that subtraction can be traced back to the addition made together with the sign bit, and this way the sign bit should not be treated separately in the operations.

Signed (negative) numbers II.

Example:

- 1111 1010 1010 0010 2's complement
- Inverting, because it is negative 0000 0101 0101 1101
- We add 1: 0000 0101 0101 1110
- We get: -1374

Floating-point number representation

Basic idea: A = M x 10^k, where M is the coefficient and K is the exponent or characteristic

Normal form, where M contains only a fraction.

As the basis of representation, instead of 10 we rather use 2 or 16.

Simple binary code

- In this code, the quantities to be represented are represented with simple binary numbers.

Two bits

Dec	A	B
0	0	0
1	0	1
2	1	0
3	1	1

three bits

Dec	A	B	C
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

four bits

Dec	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Priority code

- The only important thing about the priority code is the first 1, when we are proceeding from the biggest priority to the lower priority bits. We can encode 2n-1 bits to a log2n bit number. In this example, we are encoding from 6 to 3:

K0	K1	K2	K3	K4	K5	A2	A1	A0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1
0	0	0	0	1	X	0	1	0
0	0	0	1	X	X	0	1	1
0	0	1	X	X	X	1	0	0
0	1	X	X	X	X	1	0	1
1	X	X	X	X	X	1	1	0

Error detecting and correcting codes

- Information transmission is done without errors if the received data equals the sent message. In practice, this does not always happen. Sometimes one or more bits change from 1 to 0 or from 0 to 1. To disclose and eliminate the problems, we need to expand the code carrying the pure information with a bit or bits that increase the length of the code words so that with their help we can control and correct the code words, without changing their information content. Then we can say that we increase the redundancy of the code.

Error detecting and correcting codes

- A code word containing redundancy therefore contains the following bits:
- - Bits containing useful information
- - Bits containing not useful information, or redundancy bits
- The code systems containing redundancy can be classified into two groups: ED, Error Detecting, and EC, Error Correcting codes.
- The principle of the parity code is that we complete the code word of a given code so that in the completed code word, the number of 1s is even or odd.

Error detecting codes, parity

- The complementary bit is called parity bit.
- The even parity code (even parity) is the completed code where the number of 1s in the completed code word is even.
- The odd parity code (even parity) is the completed code where the number of 1s in the completed code word is odd.
- The advantage of parity creation is that every code can have an error detecting ability.
- Its disadvantages:
  - If we are detecting the problem, we cannot correct the error.
  - If more than one bits are out of order, parity control might not detect it, because there might be two (or an even number of) bits changing their values.

Error detecting codes, parity

Even					Odd				
A	B	C	D	P	A	B	C	D	P
0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	1	0
0	0	1	0	1	0	0	1	0	0
0	0	1	1	0	0	0	1	1	1
0	1	0	0	1	0	1	0	0	0
0	1	0	1	0	0	1	0	1	1
0	1	1	0	0	0	1	1	0	1
0	1	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0	0	0
1	0	0	1	0	1	0	0	1	1
1	0	1	0	0	1	0	1	0	1
1	0	1	1	1	1	0	1	1	0
1	1	0	0	0	1	1	0	0	1
1	1	0	1	1	1	1	0	1	0
1	1	1	0	1	1	1	1	0	0
1	1	1	1	0	1	1	1	1	1

Error correcting codes

- The supplementary bits are useful in correcting errors, too. For the correction of errors, one bit per code word is not enough. In case of a matrix transmission, data are transmitted in blocks or packages. We are only capable of correcting errors during matrix transmission if data are supplemented with error indicating codes vertically and horizontally. This way, we will know which word has the error and which bit is bad within that word. Since the error of a bit means that its value changed from 0 to 1 or the other way round, its correction means that its value should be changed back.
- In our example, we use even parity bits vertically and horizontally in an 8 byte matrix.

Error correction during matrix transmission

Sent matrix:

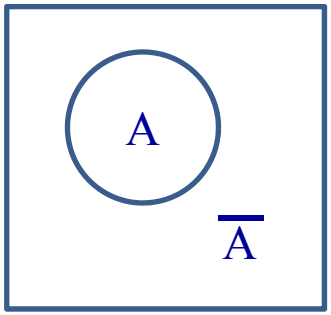
A7	A6	A5	A4	A3	A2	A1	A0	P
0	1	1	1	0	0	1	0	0
0	1	0	0	1	0	1	0	1
1	1	0	0	0	1	1	0	0
0	0	1	1	0	1	0	0	1
1	0	1	0	1	1	1	1	0
1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	0	0	1
1	1	0	1	0	0	1	1	1
0	1	1	0	1	1	1	1	0

Erroneous received matrix:

A7	A6	A5	A4	A3	A2	A1	A0	P
0	1	1	1	0	0	1	0	0
0	1	0	0	1	0	1	0	1
1	1	0	0	0	1	1	0	0
0	0	1	1	0	1	0	0	1
1	0	1	0	0	1	1	1	1
1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	0	0	1
1	1	0	1	0	0	1	1	1
0	1	1	0	0	1	1	1	0

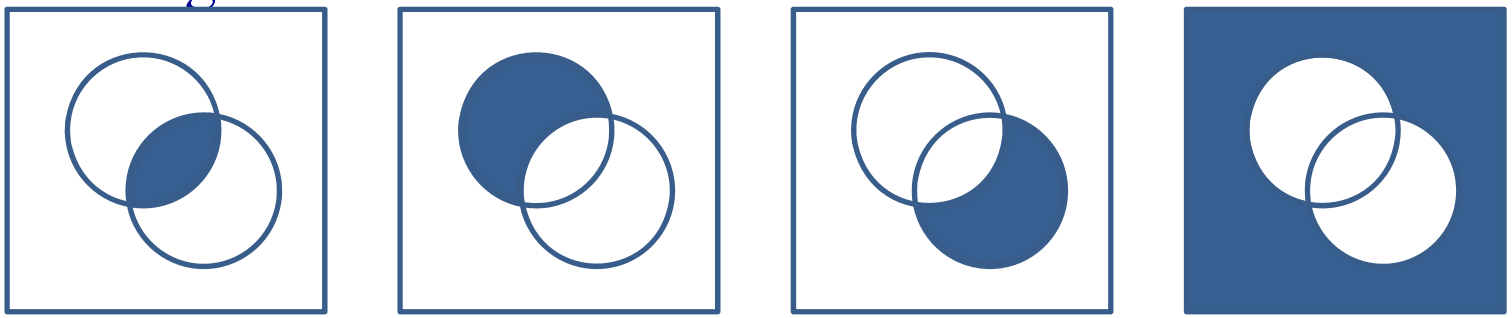
The logic of sets

- Set means the collection of different things having common characteristics.
- The elements of the set are the things belonging to the set.
- The complement set of a set is a set $\overline{A}$ whose elements does not share the characteristics of set A.
- The empty set is also called zero set or 0 set. Its symbol is 0.
- A subset is the group of the elements of a set that are characterized by further characteristics.
- The complement of the empty set is the universial set in the Boolean algebra. The symbol of the universial set is 1. The complement of the universial set is the 0 set.



The logic of sets

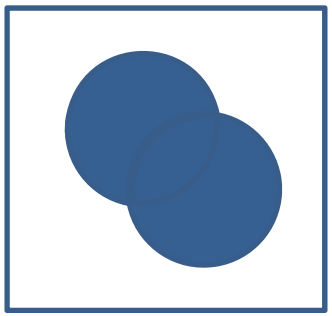
- The common part of two sets (A és B), their logical product, intersection or conjunction are the elements that are simultaneously the elements of A and B. Logical product is written in the Boolean algebra in the following way: C = AB.
- The subsets AB, $\overline{A}$'B, A$\overline{B}$' and $\overline{A}$'$\overline{B}$' are also called minterms. Minterms are therefore logical products in all the combinations occuring in words.



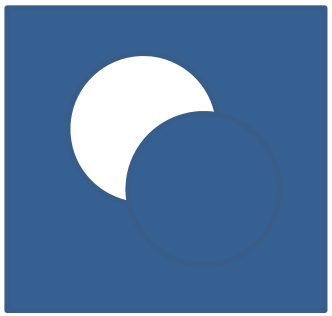
The logic of sets

The elements that belong to set A or set B or both of them constitute the logical sum, or union of the two sets.

The subsets $A + B$, $\overline{A} + B$, $A + \overline{B}$ and $\overline{A} + \overline{B}$ are also called minterms. Minterms are therefore logical products in all the combinations occuring in words.



$A+B$



$\overline{A}+B$

$A+\overline{B}$

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

Bivalent logic and binary numbers

- To characterize the two states, we can use numbers 0 and 1. Is the set is empty, it is 0, if it is not empty, it is 1.

Boolean functions

- Boolean functions are dependent logical variable(s) assigned to independent logical variables. It is similar with numbers, but here the values are logical values. So we examine how one or more logical variable depends on the independent logical variables. E.g. $X = A\overline{B}C$ means that X is true if A is true and B is false and C is true. Similarly, equation $Y = A\overline{B}C\overline{D} + ABC\overline{D} + AB\overline{C}D + \overline{A}B\overline{C}D$ is a Boolean function as well, which describes when Y will be true.

Predicate calculus

- Making a logical decision is also called predicate calculus. Predicate calculus is similar to the spoken language.
- X is true, that water is flowing, if tap A is open AND tap B is NOT open, AND security valve C is NOT open.
  Equation: $X = A\overline{B}\overline{C}$
- Lamp L is on, if eastern switch A is in state 1 AND western witch B is also in state 1 OR eastern switch A is NOT in state 1 AND western witch B is also NOT in state 1. This is a so called alternative switching, L is on if the two switches are in the same state.
  Equation: $L = AB + \overline{A}\overline{B}$

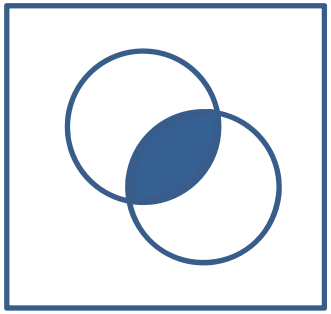
Truth table

- We often make simple, bivalent decisions with simple, two possible conditions. This can be represented in a table. In this table, we simply write down under what conditions the decision will be true and false. This table is called truth table.
- The truth table of logical NOT: $X=\overline{A}$

A	X
False	1
true	0

Truth table of logical AND

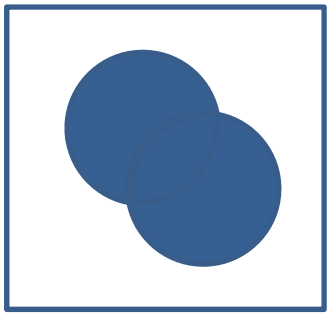
Y=AB					
Spoken			Sign		
A	B	Y	A	B	Y
False	False	False	0	0	0
False	True	False	0	1	0
True	False	False	1	0	0
True	True	True	1	1	1



AB

Truth table of logical OR

Z=A+B					
Spoken			Sign		
A	B	Z	A	B	Z
False	False	False	0	0	0
False	True	True	0	1	1
True	False	True	1	0	1
True	True	True	1	1	1



A+B

Identites and theorems of the Boolean algebra

$A + A = A$. If we add a set to itself, we get the original set.

$A + \overline{A} = 1$. This follows from the interpretation of the complement set.

$AA = A$. If we multiply a set with itself, we get the original set.

$A\overline{A} = 0$. A set and its complement set does not have a common part.

$\overline{\overline{A}} = A$ Double negation is a predicate.

$A + 0 = A$ If we add the 0 set to any kind of A, we get A.

$A + 1 = 1$ Because A is the subset of the universal set.

$A1 = 1A = A$

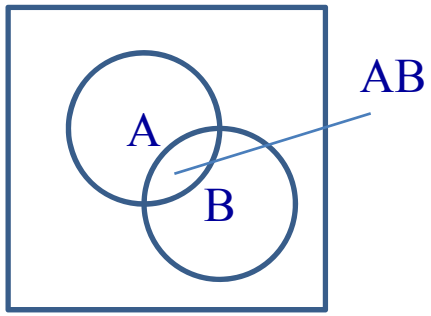
$A0 = 0A = 0$ Because A and the 0 set does not have a common element.

Commutativity, associativity, distributivity

- Commutativity (interchangeability)
  - A + B = B + A
  - AB = BA
- Associativity (parentheses can be rearranged)
  - A logical product's parentheses can be rearranged.
  - ABC = A(BC) = (AB)C = B(AC), stb.,
  - A logical sum's parentheses can be rearranged.
  - A + B +C = A +(B +C) = (A +B) +C = B +(A +C), stb.
- Distributivity
  - A(B +C) = AB + AC.

Absorption law

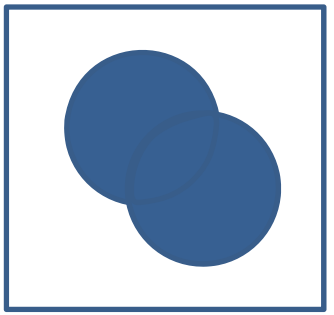
- A + AB = A
  - Absorption does not resemble the behaviour of numbers!
- A(A +B) = A
- Using distribitivity we can prove that they are equals:
  - A(A + B) = AA + AB = A + AB. So we only have to justify A + AB = A. This can be seen from the Venn diagram.
- Further absorption laws:
  - A + AB = A + B
  - A(A +B) = AB



De Morgan's laws

$\overline{A+B}=\bar{A}\bar{B}$, or, for more variables:

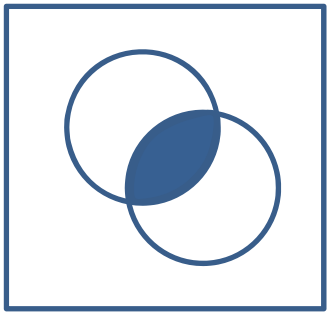
$\overline{A+B+...+N}=\bar{A}\bar{B}...\bar{N}$



A+B

$\overline{AB}=\bar{A}+\bar{B}$, or, for more variables:

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



AB

Proof of logical predicates

- Proof with truth table:
  - If we examine all the possible cases of a predicate with a table, the truth table, and it justifies our predicate, then the predicate is proved.
- Let us prove the first absorption law, namely that A + AB = A. Here A and B can both have two values. Let us examine all the possible cases :

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

Proof of logical predicates

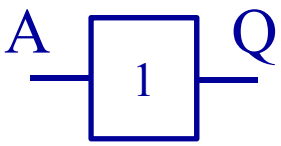
- Proof with algebraic method:
  - This means a transformation by which we transform, by using the identities and theorems of the Boole algebra, the original predicate to a form which we want to prove. To do this, we need to expand and group with great ingenuity. It is a long process. If we succeed, we proved our predicate. If we get a result that contradicts our predicate, then we proved that the original predicate is false.

Gate arrays

- Gate arrays are the basic elements of digital circuits. They are logical machines having inputs and outputs. Inputs are the independent logical variables, while the outputs are logical variables depending on the logical state of the inputs.
- Logical gate arrays are often simply called gates.
- All the in- and outputs of the gates can be of two kinds of voltage, either having a higher voltage (symbol: H) or a lower voltage (symbol: L).
- If logical 1 corresponds to voltage level H, then it is called positive logic. If logical 1 corresponds to voltage level L, then it is called negative logic.

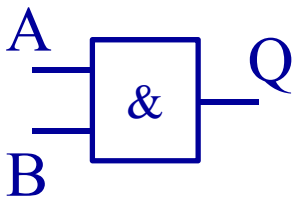
Gate arrays

Gate NOT



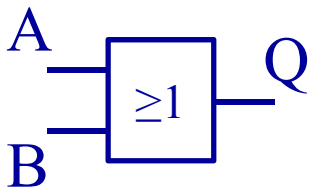
A	Q
0	1
1	0

Gate AND



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

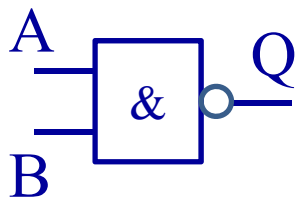
Gate OR



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

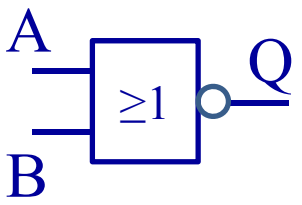
Gate arrays

Gate NAND



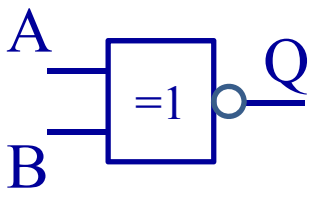
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Gate NOR



A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Gate XOR



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Defining logical equations

In the truth table, we list the possible states of the independent variables according to their binary codes, and we examine the values of the depennding variables according to these. A Boolean function with n independent variable can take states.

The listing can be of two kinds: it might concern logical products, or minterms, or logical sums, or maxterms.

A minterm is a logical AND in which all variables occur only once in a negated or non-negated form.

A maxterm is a logical OR in which all variables occur only once in a negated or non-negated form.

Regular (normal) form

- Disjunctive normal form: A function consisting of the OR connection of minterms. It is also customary to call it minterm form, the sum of products, or sum-product, or the sum of minterms.
- Conjunctive normal form: A function consisting of the AND connection of maxterms. It is also customary to call it maxterm form, or the product of sums, or product-sum, or the product of maxterms.
- These are also called minterm and maxterm form.

Karnaugh table

- The Boolean functions having few, maximum 4 variables are represented by Karnaugh tables.
- The function is represented on a table of cells. Each and every cell represents a regular term (minterm or maxterm). The cells are placed side by side in a way that the neighbouring cells' terms should be different from each other in only one logical value. This should be true vertically and horizontally as well.
- The Boolean functions are written in the Karnaugh table so that those having the term 1 has a 1 in its cell, while the other cells are left empty.