AGH

Actuating, Sensing and Control
Mechatronic Systems

Combinational Circuits

Grzegorz Góra PhD

D3-4.02

ggora@agh.edu.pl

http://home.agh.edu.pl/~ggora/

Department of Robotics and Mechatronics

Faculty of Mechanical Engineering and Robotics

AGH University of Science and Technology

AGENDA

1. Conversion of DEC-BIN-HEX numbers
2. Binary numbers representations (NBC, BCD, Gray, S-M, U2, float, double)
3. Basic gates
4. Boolean algebra
5. Combinational circuit synthesis
   - writing equations from scratch;
   - simplifying equations using Boolean algebra laws;
   - simplifying equations using Karnaugh maps;
6. Hazards in digital circuits

DEC ⇔ BIN Conversion

DEC → BIN

The conversion of a number from the **decimal system (DEC)** to the **binary system (BIN)** consists of repeatedly dividing the number by 2 and recording the remainders. The result in the binary system is then formed from these remainders read in reverse order (from the last remainder to the first one).

Step	Operation	Quotient	Remainder
1.	173 : 2	86	1
2.	86 : 2	43	0
3.	43 : 2	21	1
4.	21 : 2	10	1
5.	10 : 2	5	0
6.	5 : 2	2	1
7.	2 : 2	1	0
8.	1 : 2	0	1

$$10101101_{BIN} = 173_{DEC}$$

BIN → DEC

The conversion of a number from the **binary system (BIN)** to the **decimal system (DEC)** consists of summing the values of successive bits multiplied by the corresponding powers of 2.

$$X_{DEC} = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$173_{DEC} = 10101101_{BIN}$$

DEC ⇔ HEX Conversion

DEC → HEX

The conversion of a number from the **decimal system (DEC)** to the **hexadecimal system (HEX)** consists of repeatedly dividing the number by 16 and recording the remainders. The result in the hexadecimal system is then formed from these remainders read in reverse order (from the last remainder to the first one).

Step	Operation	Quotient	Remainder
1.	12345678 : 16	771604	14 (E)
2.	771604 : 16	48225	4
3.	48225 : 16	3014	1
4.	3014 : 16	188	6
5.	188 : 16	11	12 (C)
6.	11 : 16	0	11 (B)

$$BC614E_{HEX} = 12345678_{DEC}$$

HEX → DEC

The conversion of a number from the **hexadecimal system (HEX)** to the **decimal system (DEC)** consists of summing the values of successive digits multiplied by the corresponding powers of 16.

$$X_{DEC} = B \cdot 16^5 + C \cdot 16^4 + 6 \cdot 16^3 + 1 \cdot 16^2 + 4 \cdot 16^1 + E \cdot 16^0$$

$$12345678_{DEC} = BC614E_{HEX}$$

BIN ⇔ HEX Conversion

BIN → HEX

The conversion of a number from the **binary system (BIN)** to the **hexadecimal system (HEX)** consists of dividing the binary number into groups of four bits, starting from the right side. Each such group is then converted into its corresponding digit in the hexadecimal system.

1111000011110111000

↓

0011 | 1100 | 0011 | 1011 |1000

↓

3 | C | 3 | B |8

↓

3C3B8

$1111000011110111000_{BIN} = 3C3B8_{HEX}$

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

HEX → BIN

The conversion of a number from the **hexadecimal system (HEX)** to the **binary system (BIN)** consists of replacing each hexadecimal digit with its equivalent four-bit binary representation. These groups of bits are then combined into a single sequence, forming the number in the binary system.

3C3B8

↓

3 | C | 3 | B |8

↓

0011 | 1100 | 0011 | 1011 |1000

↓

1111000011110111000

$3C3B8_{HEX} = 1111000011110111000_{BIN}$

NBC – Natural Binary Code

Natural Binary Code (*pl. NKB – Naturalny Kod Binarny*) – is the basic binary representation of **the integer number without sign (only positive values)**.

- The value of the n-bits number A represented by the bit sequence [a_{n-1} a_{n-2} … a_2 a_1 a_0], is:

$$A = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \ldots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

- The range of the values is: [0 … 2^n-1]
  - 4-bits 0 … 15
  - 8-bits 0 … 255
  - 16-bits 0 … 65 535
  - 32-bits 0 … 4 294 967 295
  - 64-bits 0 … 18 446 744 073 709 551 615

- NBC does not allow for write negative values of the numbers.

- In C/C++ language NBC is used to code variables:

unsigned char, unsigned short, unsigned int, unsigned long, etc.

BDC – Binary Coded Decimal

BCD 8421 (Binary Coded Decimal) – is a system of binary encodings of **integer numbers without sign**, where each digit is represented by a fixed (usually 4) number of bits.

• The number is divided into digits and then each digit is separately encoded into 4 bits.

• Only 10 out of 16 bit combinations are used.

• The numbers encoded in the BCD code are not natural binary numbers. The mathematical operations directly on the BCD code are possible, but they require corrections. Ordinary Arithmetic Logic Unit (ALU) will not return correct results.

• Applications:
  - in digital display controllers (eg. calculators, digital meters);
  - for storing and calculating directly in decimal numbers, without converting to binary code.

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
-	1010
-	1011
-	1100
-	1101
-	1110
-	1111

$$5827_{DEC} = \underset{5}{0101}\ \underset{8}{1000}\ \underset{2}{0010}\ \underset{7}{0111}\ _{BCD}$$

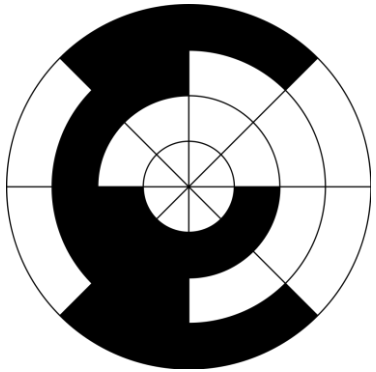
Gray Code

Gray Code (*pl. Kod Graya*) – is the binary numeral system to represent **integer numbers without sign**, such that two successive values differ in only one bit.

- The first and last words of this code also meet the condition of the one-bit difference.

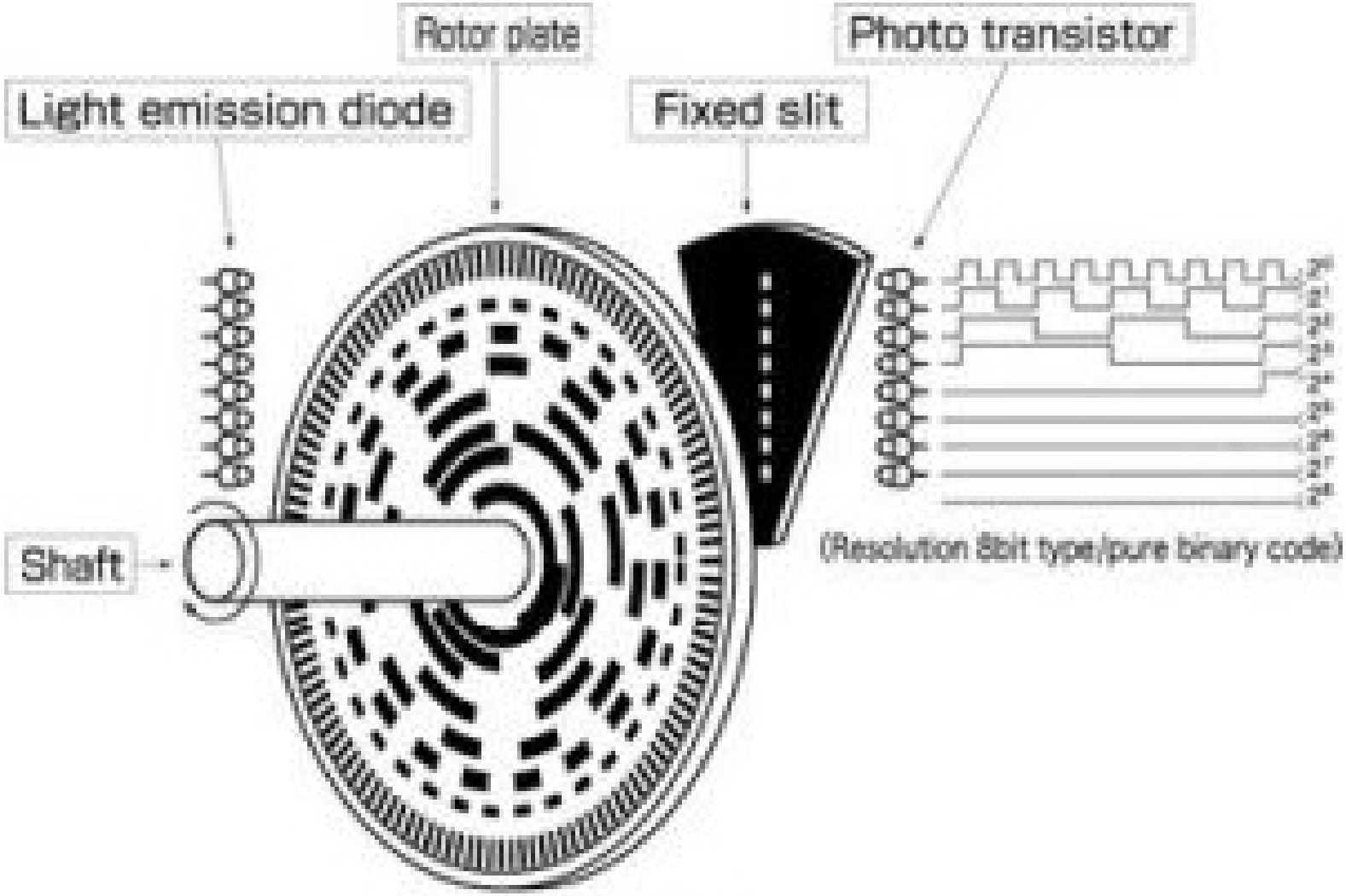
- The n-bits Gray code is recursively constructed by symmetrically reflecting the code for the (n-1) bits and completing the oldest bit with zeros (first half) and ones (second half).

- Applications (in order to avoid transients in digital electronics):
  - ADC converters;
  - rotary encoders (an example of a 3-bit encoder disc in the picture on the right);
  - ***<u>minimization of logical functions by using the Karnaugh maps.</u>***



3-bits	2-bits	1-
0	0	
0	0	
0	1	
0	1	
1	1	
1	1	

Gray Code
(Application of Gray code in the design of an absolute encoder disk)

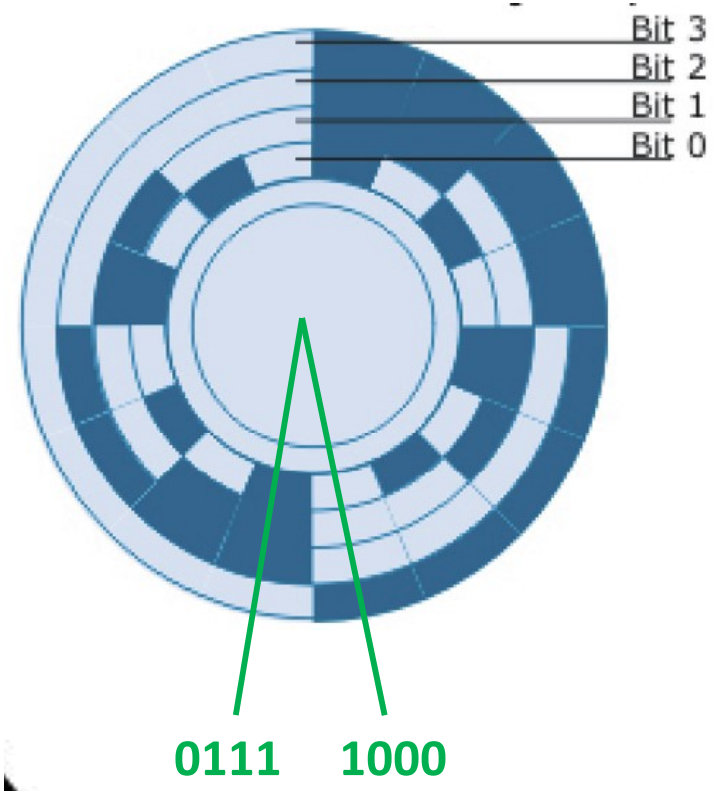


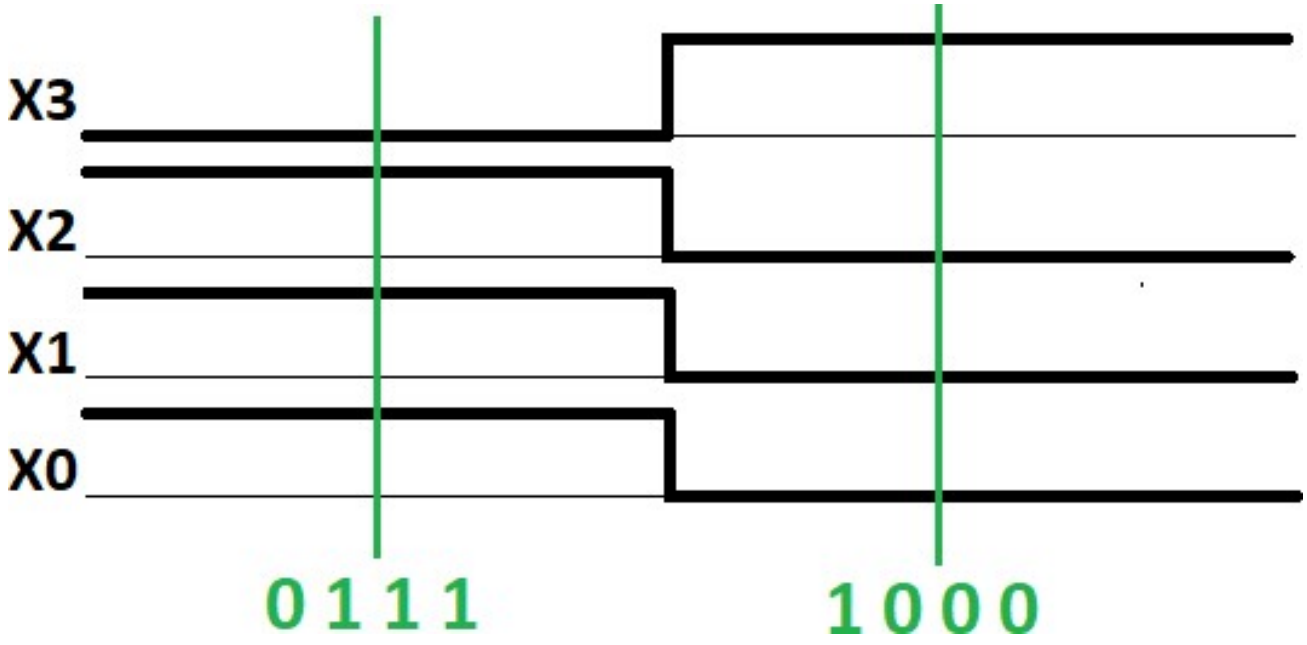
Absolute Encoder Simplified Structure

Gray Code
(Application of Gray code in the design of an absolute encoder disk)

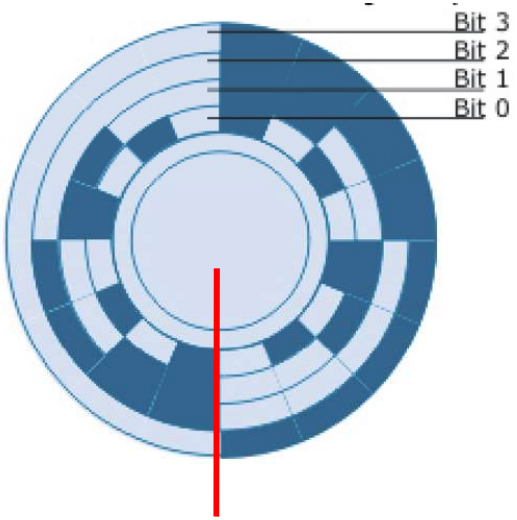
Why is it worth using Gray code in the construction of an encoder disc?
What happens in an encoder with a binary-coded disc
when the values of several bits change simultaneously?
Example of a 4-bit encoder (0..15):
Change 0111 ⇄ 1000

Binary-coded disc



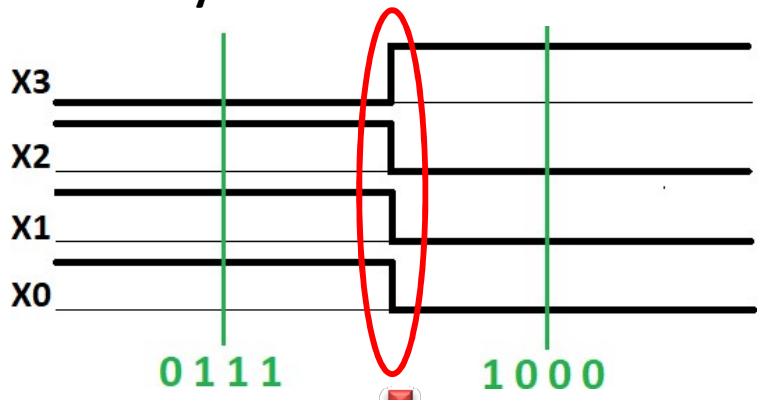


Gray Code
(Application of Gray code in the design of an absolute encoder disk)



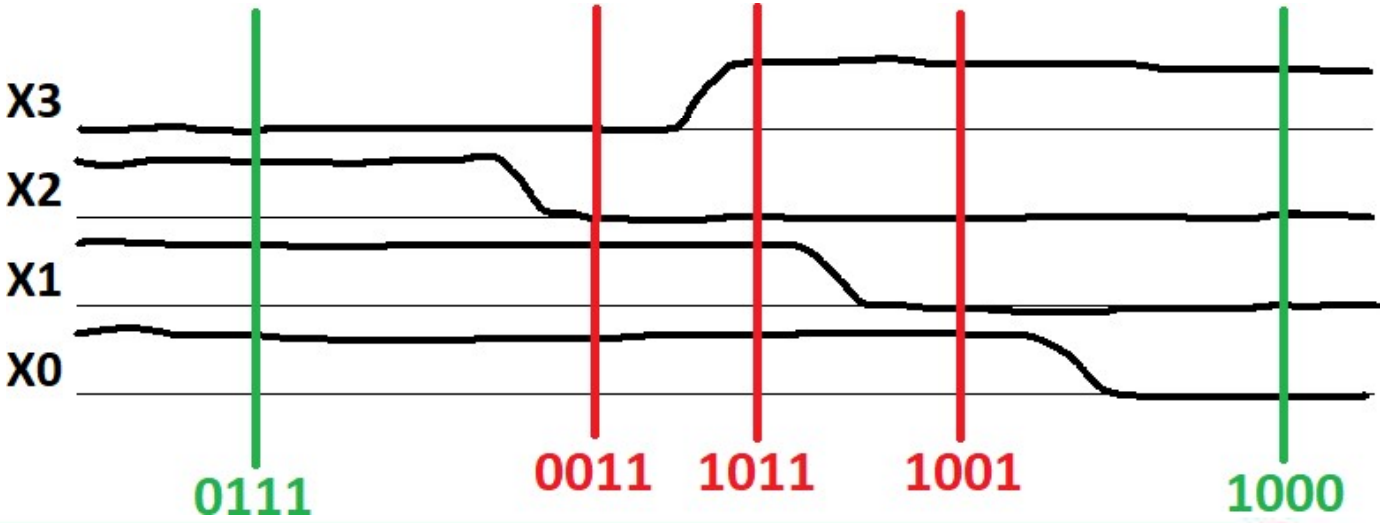
0111 → 0011 → 1011 → 1001 → 1000

Look closely…



Causes of issues:

- **inaccuracies (tolerance) in the manufacture of the disc;**
- **different delays in the signal acquisition path of each bit;**
- **limited rise/fall time of the signal edge.**



AGH

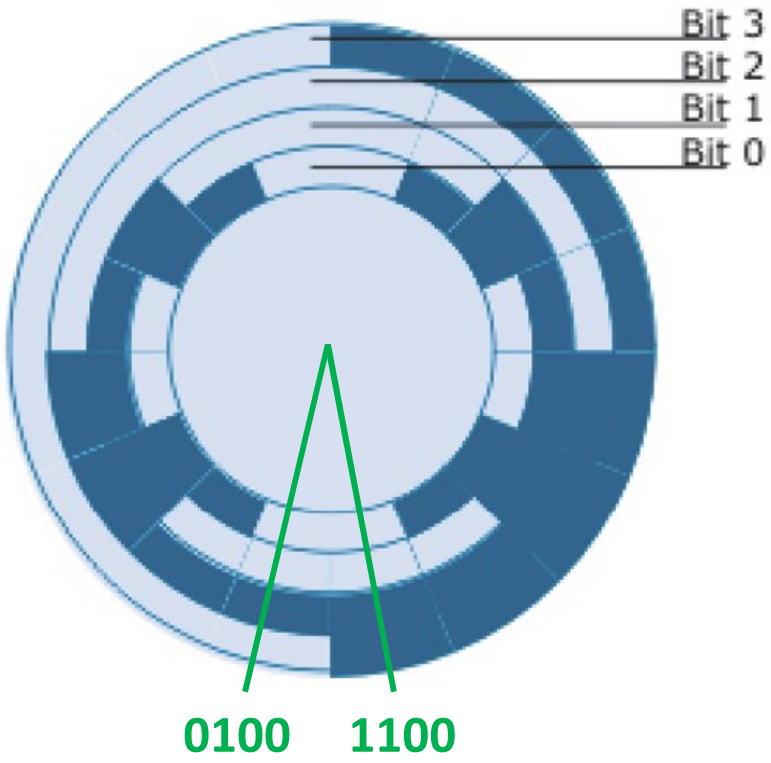
Gray Code

(Application of Gray code in the design of an absolute encoder disk)

SOLUTION: Gray-coded disc and translation Gray → NBC

In the case of a disk implemented with Gray code, only one bit changes at a time, so transitional states do not occur.

Gray-coded disc



Sign-Magnitude (S-M)

Sign-Magnitude (*pl. Z-M – Znak-Moduł*) – is the simplest way to represent **integer number with sign** (positives and negatives values).

- Sign-Magnitude:
  - one bit (usually the most significant bit) is the sign:
    0 – positive value, 1 – negative value;
  - rest of the bits represent the absolute value of the encoded number
    (in Natural Binary Code).

- The value of the n-bits number A represented by the bit sequence [a_{n-1} a_{n-2} … a_2 a_1 a_0], is:

$$A = (-1)^{a\_(n-1)} \cdot (a_{n-2} \cdot 2^{n-2} + \ldots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0).$$

- The range of the values is: $[-2^{n-1}+1 \ldots 2^{n-1}-1]$, eg. for 8-bits number <-127, 127>.

- The opposite value - negate the sign bit (the most significant bit).

- There are two ways to represent zero eg. for 8-bits number: 00000000 and 10000000 – both combination of bits represent zero **(+0 and -0)**.

- The Sign-Magnitude representation is the simplest way to encode an integer value with sign, but it cause problems when performing arithmetic operations. The sign bit has a completely different meaning from the other bits and does not participate directly in arithmetic operations. ***As a consequence, operations performed by the standard Arithmetic Logic Unit (ALU) on the numbers written in this code don't give correct results!!!***

U2 (Two's complement)

U2 (*Two's complement, pl.: Kod uzupełnień do 2*) – the most widely used system for encoding **signed integers**.

- The most significant bit have negative value and the rest of bits have positive value.
- The value of the n-bits number A represented by the bit sequence [a_{n-1} a_{n-2} … a_2 a_1 a_0], is:

$$A = \mathbf{(-1) \cdot a_{n-1} \cdot 2^{n-1}} + a_{n-2} \cdot 2^{n-2} + \ldots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

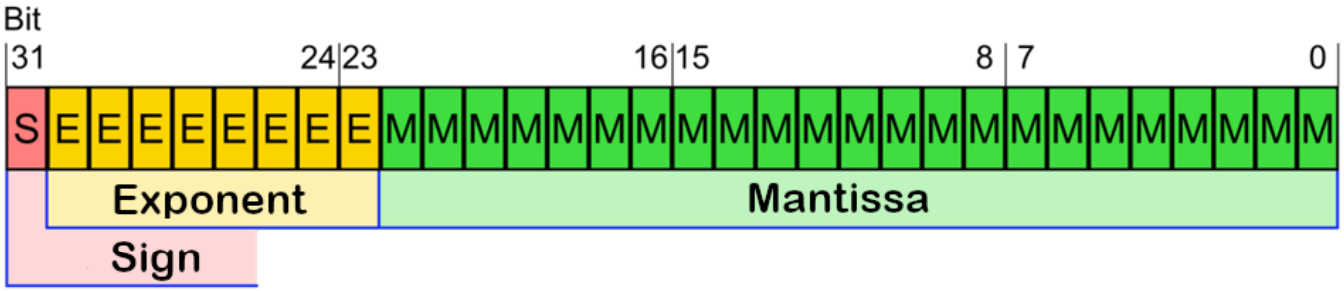
- The range of the values is: $[-2^{n-1} \ldots 2^{n-1}-1]$, eg. for 8-bits number <-128, 127>.
- For positive values U2 looks like Sign-Magnitude and NBC.
- The opposite value :
  - inverting all the bits;
  - adding 1.
- **The basic arithmetic operations performed by standard Arithmetic Logic Unit (ALU) on numbers written in the U2 code give correct results.**
- In C/C++ language U2 is used to code variables: ***char, short, int, long***, etc.

$0010\ 0011_{(U2)}$	= 35
~	
$1101\ 1100_{(U2)}$	
+1	
$1101\ 1101_{(U2)}$	= -35
~	
$0010\ 0010_{(U2)}$	
+1	
$0010\ 0011_{(U2)}$	= 35

Single-precision floating-point number (float)

Single-precision floating-point number (float) – a coding method that allows the representation of floating-point numbers.

- It allows the representation of positive and negative numbers, integers, fractions, as well as special values (e.g., infinity or NaN).
- 32-bit representation, consisting of: 1 sign bit, 8 exponent bits, and 23 mantissa bits.
- In C/C++, a single-precision floating-point number is represented using the **float** type.



$$M = 1.0 + \sum_{i=0}^{22} m_i \cdot 2^{-}$$

D – value of the number
S – sign bit
m_i – mantissa bit values [m_{22} m_{21} … m_2 m_1 m_0]
e_i – exponent bit values [e_7 e_6 e_5 e_4 e_3 e_2 e_1 e_0]

$$E = \sum_{i=0}^{7} e_i \cdot 2^i$$

Single-precision floating-point number (float)

Single-precision floating-point number (float) – a coding method that allows the representation of floating-point numbers.

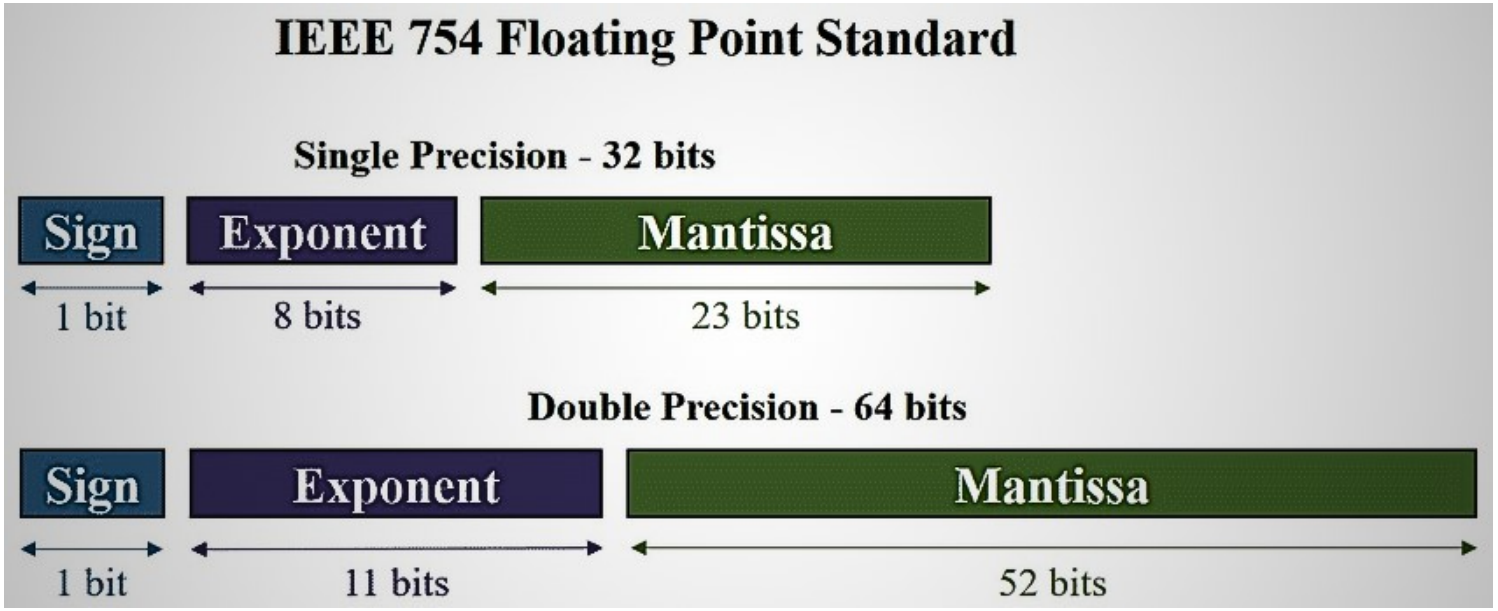
Sign	Exponent	Mantissa	Number type
Any	Different from ‘00000000’ and ‘11111111’	Any	Normalized number
Any	‘00000000’	Different from ‘00000000000000000000000’	Denormalized number
‘0’	‘00000000’	‘00000000000000000000000’	+0
‘1’	‘00000000’	‘00000000000000000000000’	-0
‘0’	‘11111111’	‘00000000000000000000000’	+ ∞
‘1’	‘11111111’	‘00000000000000000000000’	- ∞
Any	‘11111111’	Different from ‘00000000000000000000000’	NaN

Tab. Single precision floating point number encoding

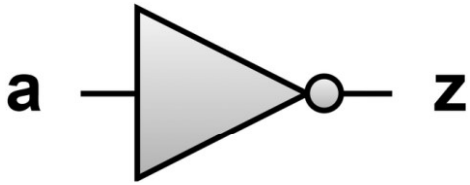
Double-precision floating-point number (double)

Double-precision floating-point number (double) – a coding method that allows the representation of floating-point numbers. Provides higher computational precision compared to single-precision floating-point numbers (float).

- Like float, it allows the representation of positive and negative numbers, integers, fractions, as well as special values (e.g., infinity or NaN).
- 64-bit representation, consisting of: 1 sign bit, 11 exponent bits, and 52 mantissa bits.
- In C/C++, a double-precision floating-point number is represented using the double type.



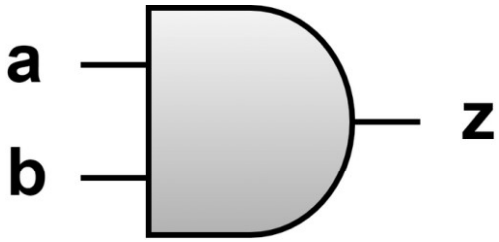
Gates: NOT



$$z = \overline{a}$$

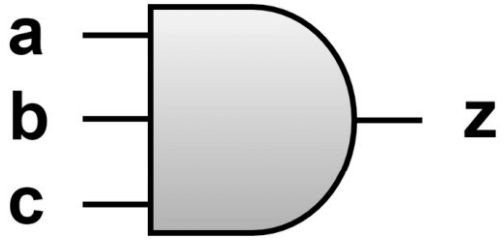
a	z
0	1
1	0

Gates: AND (logical conjunction)



$z = ab$

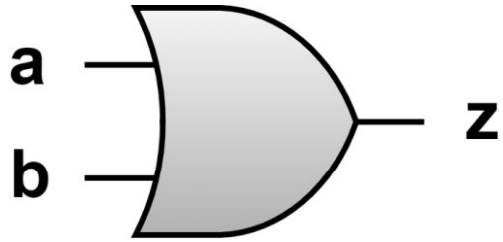
a	b	z
0	0	0
0	1	0
1	0	0
1	1	1



$z = abc$

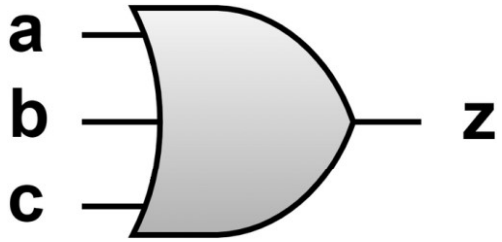
a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Gates: OR (logical disjunction)



$z = a+b$

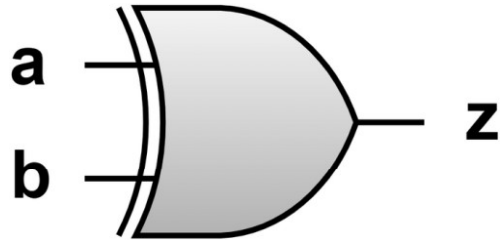
a	b	z
0	0	0
0	1	1
1	0	1
1	1	1



$z = a+b+c$

a	b	c	z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Gates: XOR (Exclusive OR)

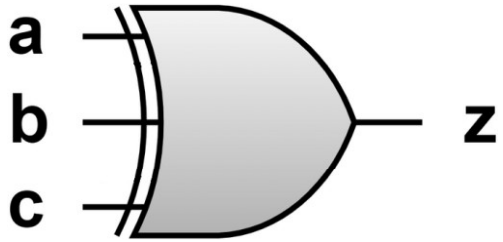


$$z = a \oplus b$$

$$z = \bar{a}b + a\bar{b}$$

a	b	z
0	0	0
0	1	1
1	0	1
1	1	0

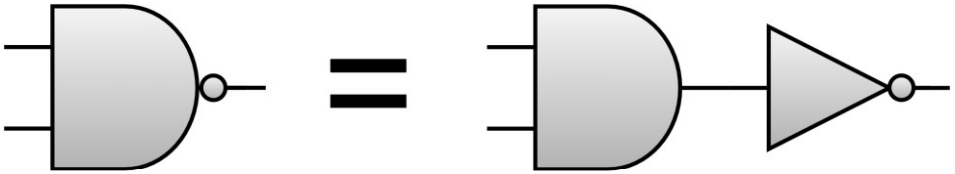
XOR can be viewed as addition modulo 2.

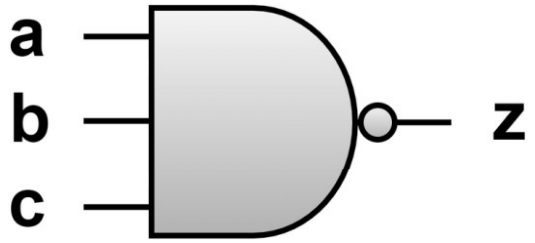


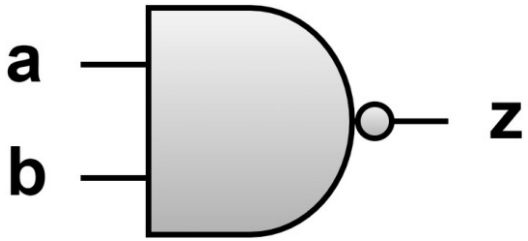
$$z = a \oplus b \oplus c$$

a	b	c	z
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

Gates: NAND







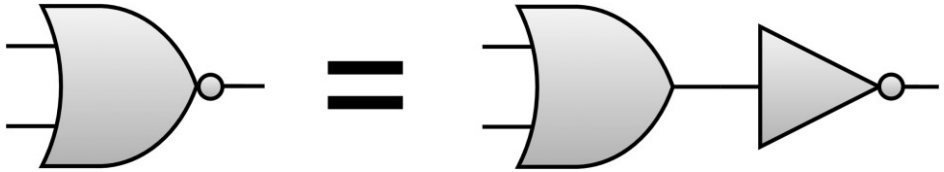
$$z = \overline{ab}$$

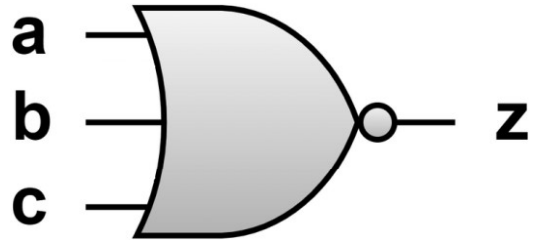
a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

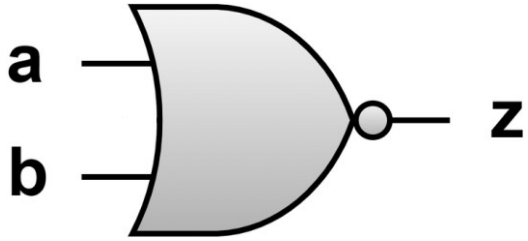
$$z = \overline{abc}$$

a	b	c	z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Gates: NOR







$$z = \overline{a+b}$$

a	b	z
0	0	1
0	1	0
1	0	0
1	1	0

$$z = \overline{a+b+c}$$

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

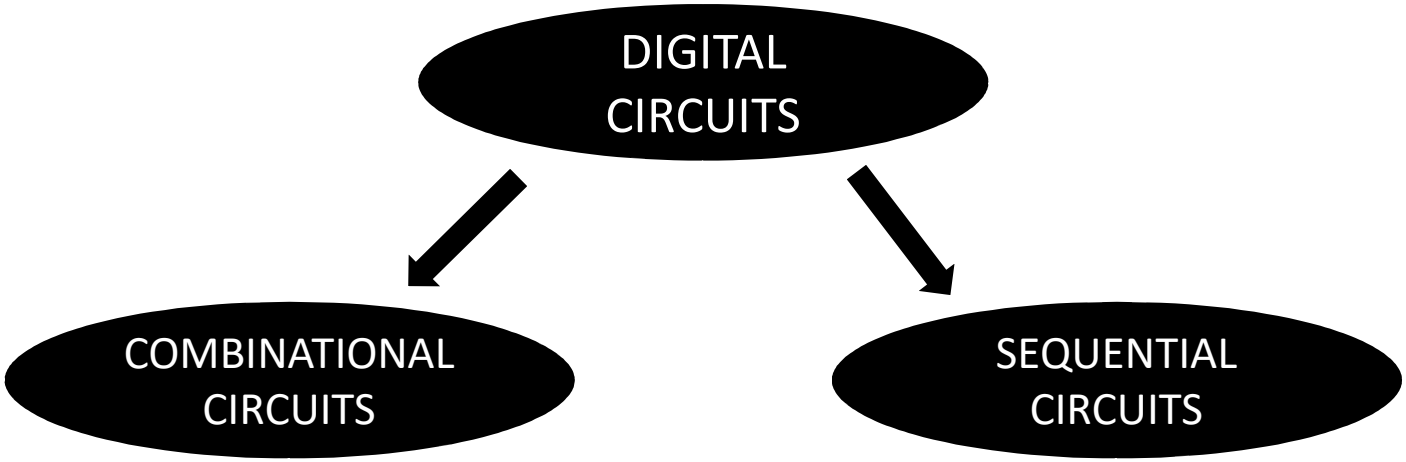
Boolean algebra

$$\overline{\overline{A}} = A$$

$$A + 0 = A \qquad A \cdot 0 = 0$$
$$A + 1 = 1 \qquad A \cdot 1 = A$$
$$A + A = A \qquad A \cdot A = A$$
$$A + \overline{A} = 1 \qquad A \cdot \overline{A} = 0$$

$A + B = B + A$	$A \cdot B = B \cdot A$	commutation rules
$A + (A \cdot B) = A$	$A \cdot (A + B) = A$	absorption rules
$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	association rules
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	distribution rules
$\overline{A + B} = \overline{A} \cdot \overline{B}$	$\overline{A \cdot B} = \overline{A} + \overline{B}$	DeMorgan's theorems
$A + \overline{A} \cdot B = A + B$	$A \cdot (\overline{A} + B) = A \cdot B$	elimination rules
$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$	$(A + B) \cdot (\overline{A} + C) \cdot (B + C) = (A + B) \cdot (\overline{A} + C)$	consensus rules
$A \cdot B + \overline{A} \cdot C = (A + C) \cdot (\overline{A} + B)$	$(A + B) \cdot (\overline{A} + C) = A \cdot C + \overline{A} \cdot B$	interchange rules

Digital circuits: combinational and sequential circuits



COMBINATIONAL CIRCUITS *– is a type of logic circuit whose* ***output depends only on the present value of its input signals****. Combinational cicruits are eg. adders, substractors,* multiplexers*, comparators, transcoders, etc.*

SEQUENTIAL CIRCUITS *– is a type of logic circuit whose* ***output depends on the present value of its input signals and on the sequence of past inputs (previous state of the circuit)****, eg. counters, state machines.*

Combinational circuit synthesis

EXAMPLE: ***Find Boolean algebra equation (simplified) and schematic for simplified elevator door controller.***

Simplified elevator door controller

The door will be closed when:

- the circuit of a photocell is closed (no one stands in the door), and
- the minimum time of door opening has passed (a clock – a separate device – is turned off), and
- a call button at another floor is pressed or
- a destination button inside the car of the elevator is pressed.

Combinational circuit synthesis: (1) writing equations from scratch

EXAMPLE: ***Find Boolean algebra equation (simplified) and schematic for simplified elevator door controller.***

The door “D” will be closed when
(the clock “C” is **NOT** switched on) **AND**
(the photocell “P” circuit is closed) **AND**
[(a call button “B” is pressed) **OR**
(a destination button “M” is pressed)]

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

Combinational circuit synthesis: (1) writing equations from scratch

EXAMPLE: ***Find Boolean algebra equation (simplified) and schematic for simplified elevator door controller.***

The door “D” will be closed when
(the clock “C” is **NOT** switched on) **AND**
(the photocell “P” circuit is closed) **AND**
[(a call button “B” is pressed) **OR**
(a destination button “M” is pressed)]

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

Writing equations from scratch can be used for simple examples (small number of variables/signals without complex relationships between them)

Combinational circuit synthesis based on the truth table

Idx.	Inputs			Output	Minterms:	Maxterms:
i	a	b	c	y		
0	0	0	0	1	$m_0 = \bar{a}\bar{b}\bar{c}$	$M_0 = a+b+c$
1	0	0	1	0	$m_1 = \bar{a}\bar{b}c$	$M_1 = a+b+\bar{c}$
2	0	1	0	1	$m_2 = \bar{a}b\bar{c}$	$M_2 = a+\bar{b}+c$
3	0	1	1	x	$m_3 = \bar{a}bc$	$M_3 = a+\bar{b}+\bar{c}$
4	1	0	0	1	$m_4 = a\bar{b}\bar{c}$	$M_4 = \bar{a}+b+c$
5	1	0	1	0	$m_5 = a\bar{b}c$	$M_5 = \bar{a}+b+\bar{c}$
6	1	1	0	1	$m_6 = ab\bar{c}$	$M_6 = \bar{a}+\bar{b}+c$
7	1	1	1	x	$m_7 = abc$	$M_7 = \bar{a}+\bar{b}+\bar{c}$

The canonical Sum of Products (SOP) form:

$$SOP = \sum_{i|y(i)=1} m_i$$

The canonical Product of Sums (POS) form:

$$POS = \prod_{i|y(i)=0} M_i$$

Don’t care condition (mapped as x) exists when certain combination of inputs never occur.

X’s may be considered as either 0s or 1s, whichever provides the greatest simplification.

Combinational circuit synthesis: (2) truth table + simplifying equations using Boolean algebra laws

C (clock)	P (photocell)	B (call button)	M (destination button)	D (door)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Simplified elevator door controller

The door will be closed when the circuit of a photocell is closed (no one stands in the door), and the minimum time of door opening has passed (a clock – a separate device – is turned off), and a call button at another floor is pressed or a destination button inside the car of the elevator is pressed.

The canonical Sum of Product (SOP) form:

$$D = \overline{C} \cdot P \cdot \overline{B} \cdot M + \overline{C} \cdot P \cdot B \cdot \overline{M} + \overline{C} \cdot P \cdot B \cdot M$$

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

Combinational circuit synthesis: (2) truth table + simplifying equations using Boolean algebra laws

C (clock)	P (photocell)	B (call button)	M (destination button)	D (door)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Simplified elevator door controller

The door will be closed when the circuit of a photocell is closed (no one stands in the door), and the minimum time of door opening has passed (a clock – a separate device – is turned off), and a call button at another floor is pressed or a destination button inside the car of the elevator is pressed.

$$D = \overline{C} \cdot P \cdot \overline{B} \cdot M + \overline{C} \cdot P \cdot B \cdot \overline{M} + \overline{C} \cdot P \cdot B \cdot M$$

$$= (\overline{C} \cdot P) \cdot (\overline{B} \cdot M + B \cdot \overline{M} + B \cdot M)$$

$$= (\overline{C} \cdot P) \cdot (\overline{B} \cdot M + B(\overline{M} + M))$$

$$= (\overline{C} \cdot P) \cdot (\overline{B} \cdot M + B)$$

$$= \overline{C} \cdot P \cdot (B + M)$$

Simplifying Boolean algebra equations is a complex and laborious process. The level of complexity increases rapidly as the number of variables increases.

Combinational circuit synthesis: (3) truth table + Karnaugh maps

C (clock)	P (photocell)	B (call button)	M (destination button)	D (door)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Step 1:

Transfer the information from truth table to Karnaugh maps.

BM / CP	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	0	0	0	0
10	0	0	0	0

Combinational circuit synthesis: (3) truth table + Karnaugh maps

BM / CP	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	0	0	0	0
10	0	0	0	0

Step 2:

Create group of 1s (implicants) or 0s (implicents).

- dimension of groups must be power of 2 (eg. 1x1, 1x2, 2x1, 1x4, 4x1, 2x4, 4x2, …);
- all 1s (or 0s) have to be included;
- one cell can belong to a few groups;
- the groups should be as large as possible (large number of cells in a group means a smaller number of variables to describe it - and consequently simplifying the equation).

Combinational circuit synthesis: (3) truth table + Karnaugh maps

BM / CP	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	0	0	0	0
10	0	0	0	0

Step 3:

Write equation in the form of:

SOP (for 1s)

or

POS (for 0s);

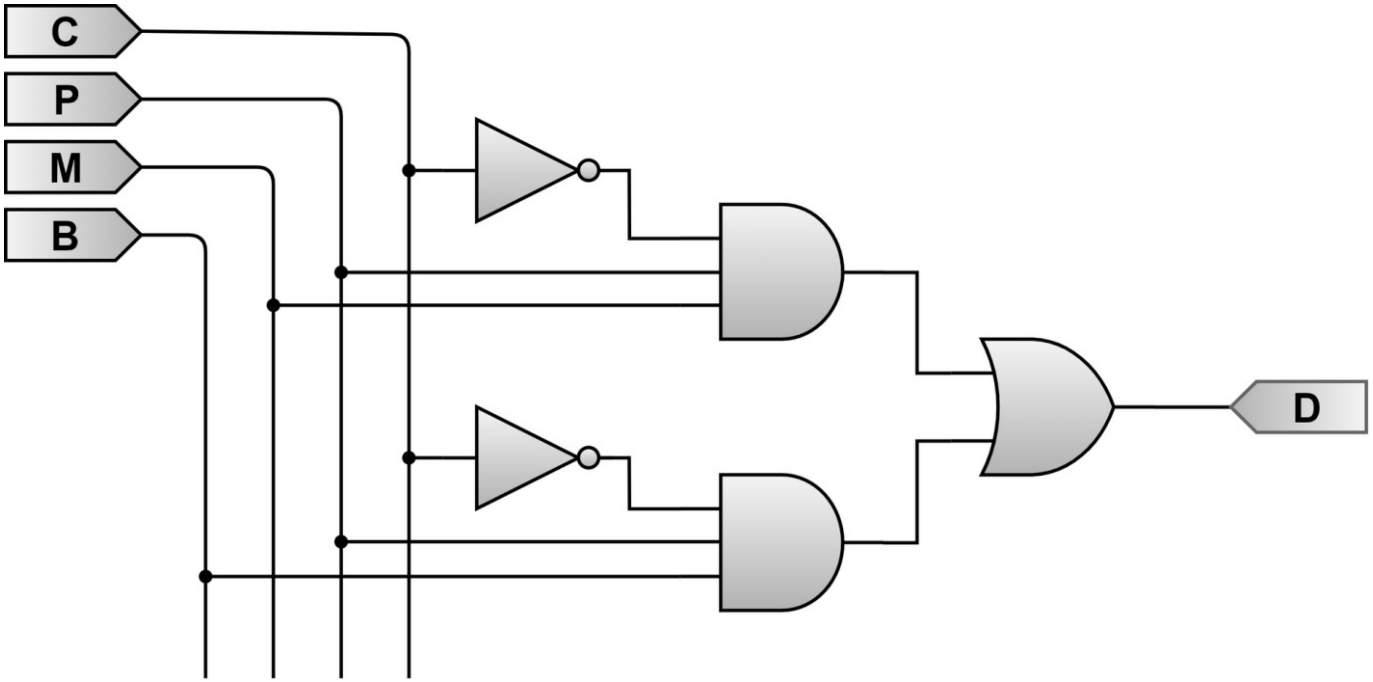
$$D = \overline{C} \cdot P \cdot M + \overline{C} \cdot P \cdot B$$

Combinational circuit synthesis: (3) truth table + Karnaugh maps

Step 4:

Draw diagram based on equation.

$$D = \overline{C} \cdot P \cdot M + \overline{C} \cdot P \cdot B$$



Combinational circuit synthesis: Karnaugh maps - summary

K-map is graphically method to simplify Boolean algebra expression. K-maps allows to get an immediate simplified form of the equation.

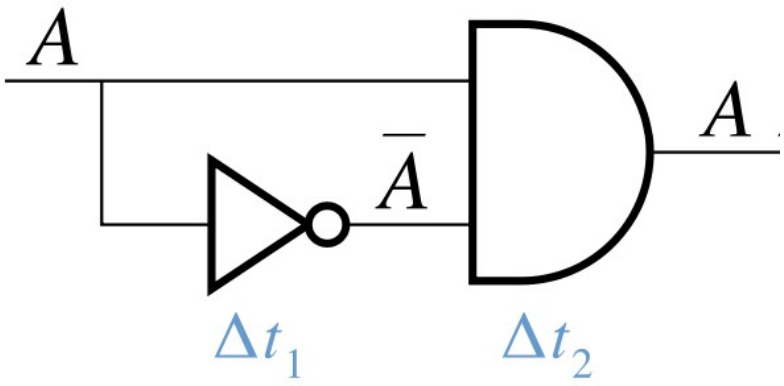
1. Transfer the information from truth table to Karnaugh maps.
2. Create group of 1s (implicants) or 0s (implicents):
   - dimension of groups must be power of 2 (1x1, 1x2, 2x1, 1x4, 4x1, 2x4, 4x2, …);
   - all 1s (or 0s) have to be included;
   - one cell can belong to a few groups;
   - the groups should be as large as possible (large number of cells in a group means a smaller number of variables to describe it - and consequently simplifying the equation).
3. Write equation in the form of SOP (for 1s) or POS (for 0s).
4. Draw diagram based on equation.

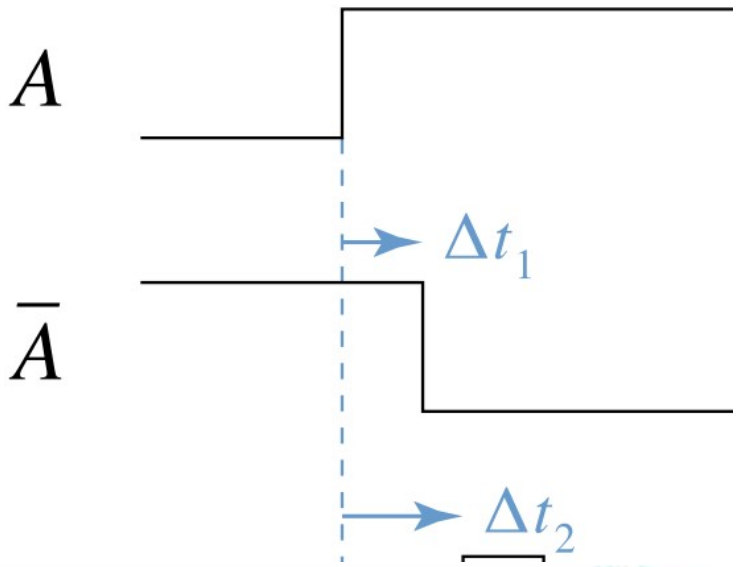
Hazards in digital circuits

Real gates are not able to change the state of the output immediately after the state changes at the input.

Propagation delay of a gate is the time delay between the input state and output change.

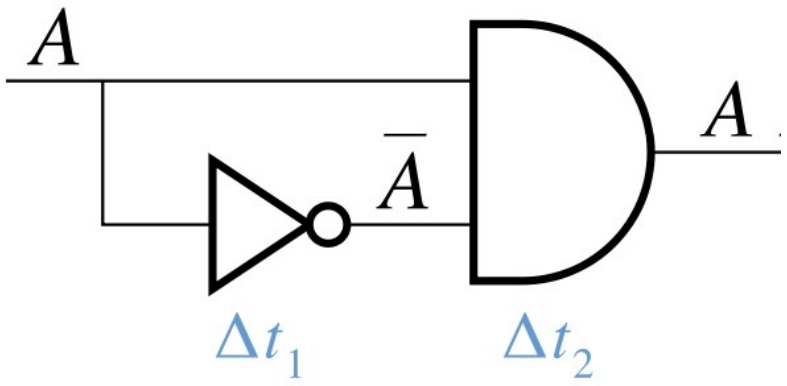
The manufacturers of the digital circuits provide information about the maximum guaranteed propagation time for their elements (the worst case propagation time of the gate).





Hazards in digital circuits

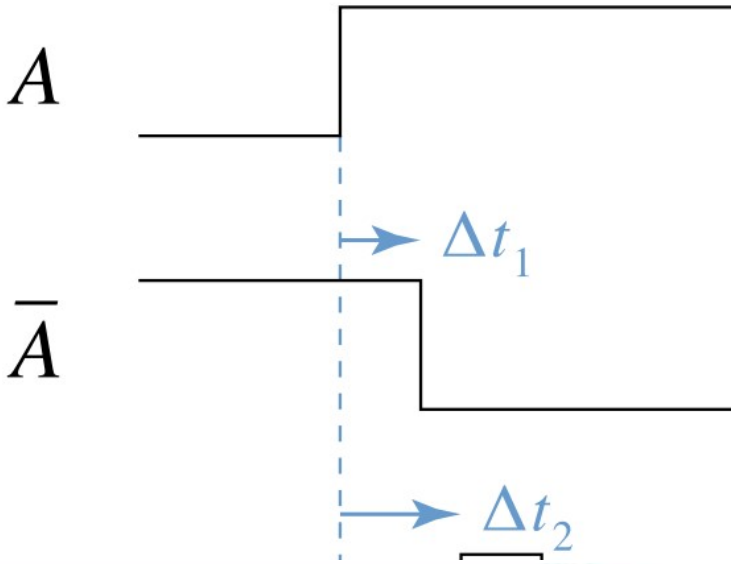
A hazard in a digital circuit causes a temporary fluctuation in the circuit's output. In other words, a hazard is a temporary disturbance in the ideal operation of the circuit. These disturbances occur when different paths from the input to the output have different delays. As a result, changes in input variables do not immediately affect the output but appear after a short delay caused by the circuit elements, such as logic gates.



There are two different kinds of hazards found in digital circuits:

static - static hazard takes place when change in an input causes the output to change momentarily before stabilizing to its correct value, the output changes only once before stabilizing;

dynamic - when the output changes several times then it should change from 1 to 0 or 0 to 1 only once.



Binary numeral system

There are 10 types of people in the world:

Those who understand binary system,

and those who don’t.

REFERENCE

[1] Barry Wilkinson: *Układy cyfrowe*; WKŁ

[2] Halina Kamionka-Mikuła, Henryk Małysiak, Bolesław Pochopień: Układy cyfrowe - teoria i przykłady; WPKJS;

[3] Wojciech Głocki: Układy cyfrowe; WSiP;

[4] Andrzej Skorupski: Podstawy techniki cyfrowej; WKŁ;

[5] prof. dr hab. in˙z. Joanna Józefowska: *Kodowanie informacji - Reprezentacja liczb*; Poznań; rok akademicki 2010/2011;

[6] dr hab. inż. Mikołaj Morzy: *Wykład - kodowanie liczb*;

[7] Politechnika Łódzka: *Kodowanie liczb całkowitych w systemach komputerowych*; http://neo.dmcs.p.lodz.pl/ak/kodowanie_liczb_calkowitych.pdf

[8] Understanding Floating-Point Arithmetic: How Computers Handle Decimals and Why It Matters - On-line 27.09.2025: https://medium.com/@yuvindur/understanding-floating-point-arithmetic-how-computers-handle-decimals-and-why-it-matters-1e32aec8d2ea

[9] Absolute Rotary Encoder - On-line 27.09.2025: https://www.omchsmps.com/absolute-rotary-encoder/