Experiment EL2

Design and implementation of digital logic circuits using NAND gates

Before you start to perform an experiment you are obliged to have mastered the following theoretical subjects:

1. Boolean algebra, principal laws and identities. [1-8]
2. Simplification of Boolean expressions using the Karnaugh map method. [1-8]
3. Symbols of gates performing basic logical functions AND, OR, NOT, NAND, NOR, EX-OR and EX-NOR [1-3,5-8].
4. Implementation of other types of logic gates as a network of just NAND or just NOR gates. [1-4,8]
5. Definitions of combinational and sequential logic circuits. [1,3,5-8]

Purpose

The purposes of this experiment are:

1. Comprehension of basic logical functions and gates.
2. The design, implementation, and testing of combinational logic circuit. The designed circuit should be as simple as possible for the stated problem.

Theoretical basis

Digital electronics uses predominantly Boolean algebra referred to variables taking only two possible values "1" and "0". There exist also three-state circuits, however, they will not be the subject of this exercise. The basic operators applicable to two-state variables are:

– conjunction, logical product, also known as AND operator: $Y = A \cdot B$,

– disjunction, logical sum, also known as OR operator: $Y = A + B$,

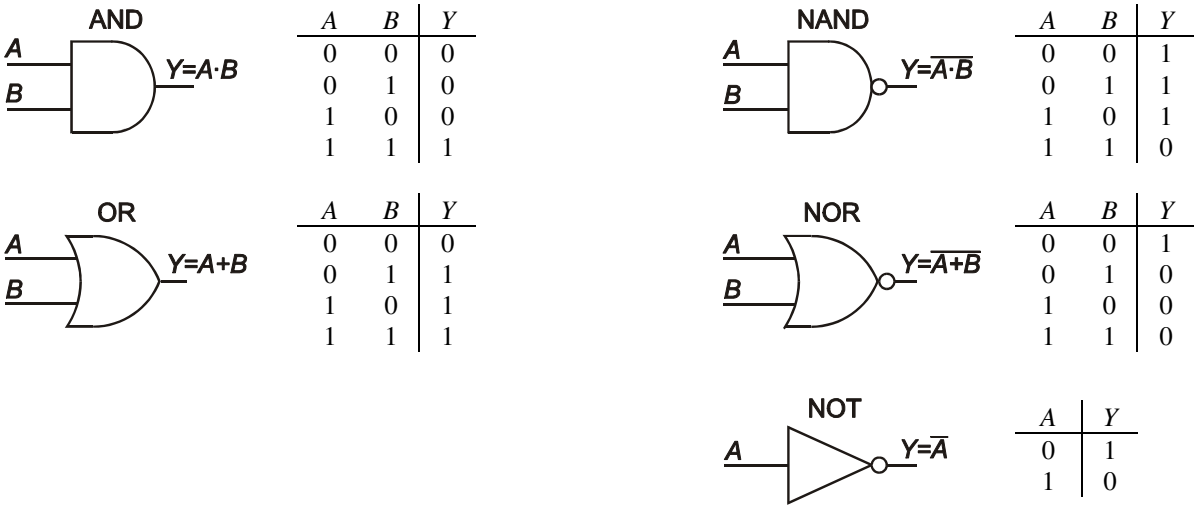
– logical negation, NOT operator: $Y = \overline{A}$.

In practice, it is very useful to introduce the composition of NOT and AND operators, as well as the composition of NOT and OR operators:

– negation of the conjunction, NAND: $Y = \overline{A \cdot B}$,

– negation of the disjunction, NOR: $Y = \overline{A + B}$.

The symbols of electric devices (called logic gates) which are performing the functions listed above are given in Fig. 1. The definitions of the AND, OR, NAND, and NOR operators can be easily extended to any number of arguments by employing the composition of two-argument operators, such as $A \cdot B \cdot C = (A \cdot B) \cdot C$.



A	B	Y (AND)
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Y (NAND)
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Y (OR)
0	0	0
0	1	1
1	0	1
1	1	1

A	B	Y (NOR)
0	0	1
0	1	0
1	0	0
1	1	0

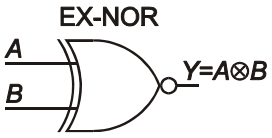
A	Y (NOT)
0	1
1	0

Fig. 1. The symbols of basic logic gates and their truth tables.

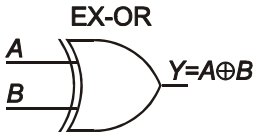
Moreover, two additional derived operators are very important in practice. The first is called exclusive-or (EX-OR) or parity (in Polish: operator ALBO, WYŁĄCZNE LUB). The second is defined as the negation NOT of EX-OR (EX-NOR) and is called logical biconditional, logical equality, or logical equivalence (in Polish: funkcja równoważności lub funkcja tożsamości).

– exclusive disjunction, EX-OR operator: $Y = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$,

– logical biconditional, EX-NOR operator: $Y = A \otimes B = A \cdot B + \overline{A} \cdot \overline{B}$.



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 2. The symbols of additional logic gates and their truth tables.

To design the logic circuits the following laws of Boolean algebra are commonly used: commutativity, associativity, distributivity, and De Morgan's laws. Note that distributivity of disjunction over conjunction and both De Morgan's laws do not have their counterparts in ordinary algebra of real numbers.

Property	For conjunction	For disjunction
Commutativity	$A \cdot B = B \cdot A$	$A + B = B + A$
Associativity	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
Distributivity	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + B \cdot C = (A + B) \cdot (A + C)$
De Morgan's laws	$\overline{A \cdot B \cdot \ldots} = \overline{A} + \overline{B} + \ldots$	$\overline{A + B + \ldots} = \overline{A} \cdot \overline{B} \cdot \ldots$
Basic identities	$A \cdot 0 = 0$ $A \cdot 1 = A$ $A \cdot A = A$ $A \cdot \overline{A} = 0$	$A + 1 = 1$ $A + 0 = A$ $A + A = A$ $A + \overline{A} = 1$
Additional identities	$A \cdot (A + B) = A$ $A + \overline{A} \cdot B = A + B$ $(A + B) \cdot (\overline{A} + B) = B$	$A + A \cdot B = A$ $A \cdot (\overline{A} + B) = A \cdot B$ $A \cdot B + \overline{A} \cdot B = B$

Table 1. Principal identities and laws of Boolean algebra.

Using De Morgan's laws it can be proved that only NAND and NOR logic gates are universal. All other types of Boolean operators (i.e., AND, OR, NOT, EX-OR, and EX-NOR) can be implemented as a suitable network of just NAND or just NOR gates. Reduction of a set of logical functors used to implement any Boolean function has many advantages and is often used in practice.

Minimization and synthesis of combinational logic circuits

Suppose that analysis of a certain control problem led to the following truth table, which describes the response W of a system for all possible logic states of the four inputs A, B, C, and D.

A	B	C	D	W
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	–

A	B	C	D	W
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	–

Table 2. A logical function written in the form of a truth table. Dash (–) are used for cases in which the logic state of output does not matter.

If one is given a truth table of a Boolean function, it is easy to write the function in a canonical sum form, i.e. as a sum of minterms, where each minterm is a logical product of all input variables in either literal or negated form and is related to one unique row, which has output $W = 1$ in the truth table

$$W = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + \overline{A}BC\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D}\,. \tag{1}$$

Simplification of the function $W = f(A, B, C, D)$ using only the principles of Boolean algebra may prove an arduous task and a final form will depend on the designer's intuition and luck. A much more effective approach employs the Karnaugh map, which taking advantage of human pattern-recognition. The Karnaugh map is a two-dimensional table of output states, where each grid box corresponds to one row in the truth table (see Fig. 3). The input states are given outside the table and are ordered according to the Gray code, in which any two successive values differ in exactly one bit. The size of the map reduces to 2×4 boxes for 3 input variables and to 2×2 boxes for 2 variables. Minimization of a function is realized by grouping neighboring "1s" into rectangular groups (Fig. 3.a), in which the number of grid boxes in the groups must be equal to a power of 2. Alternatively, the "0s" may be grouped (Fig. 3.b). The grid is toroidally connected, i.e. opposite edges are adjacent to each other. To obtain the simplest solution the minimal number of largest possible groups should be selected. All "1s" (or "0s") must be involved in at least one group. Selected "–" fields may be encircled together with "1s" (or "0s").

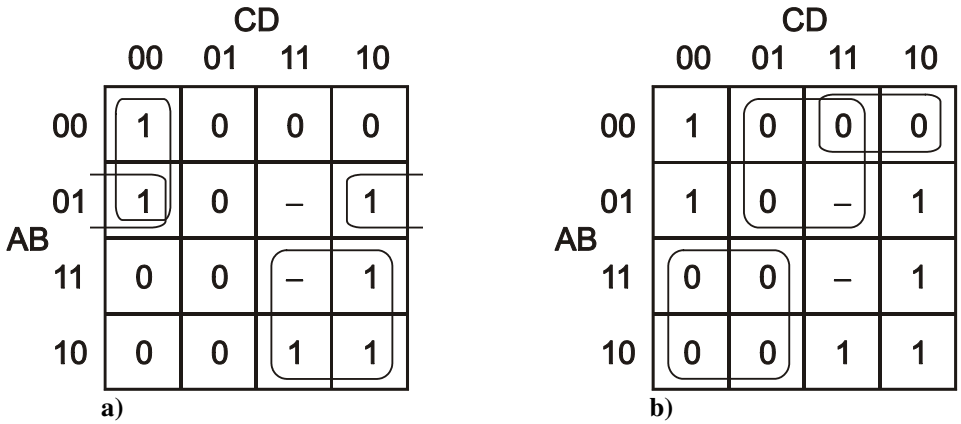


Fig. 3. The logical function given by Table 2 written in the form of the Karnaugh map. **a)** minimization by grouping "1s", **b)** minimization by grouping "0s".

The minimized function corresponding to the table shown in Fig. 3.a will be constructed as a sum of logical products, where each product must give the result 1 for all fields of a suitable group of "1s" and "–". For example, the encircling area that goes through the edges leads to the product $\overline{A}B\overline{D}$, where C is omitted here because it does not maintain the same state in the whole encircling area. The result for all groups is

$$W = \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{D} + AC\,. \tag{2}$$

When the "0s" are grouped in the Karnaugh table (see Fig. 3.b), we will create the minimized function as a product of sums, each of which gives the result 0 in one encircled area

$$W = (\overline{A} + C)(A + \overline{D})(A + B + \overline{C})\,. \tag{3}$$

Let assume, that the function must be implemented employing only the NOT and NAND gates. Thus, we have to transform all logical sums occurring in Eqs. (2) and (3) according to De Morgan's laws

$$\overline{X} + \overline{Y} + \ldots = \overline{XY\ldots}\,, \qquad X + Y + \ldots = \overline{\overline{X}\,\overline{Y}\ldots}\,. \tag{4}$$

Now Eqs. (2) and (3) will take the following forms, respectively,

$$W = \overline{\overline{(\overline{A}\,\overline{C}\,\overline{D})}\;\overline{(\overline{A}B\overline{D})}\;\overline{(AC)}}\,, \tag{5}$$

$$W = \overline{(A\overline{C})}\;\overline{(\overline{A}D)}\;\overline{(\overline{A}\,\overline{B}C)}\,. \tag{6}$$

Note, that all operations occurring in Eq. (5) can be implemented directly employing assumed available gates, while the outer logical product (AND) in Eq. (6) must be expressed as a negation (NOT) of negated product (NAND)

$$W = \overline{\overline{(A\overline{C})}\;\overline{(\overline{A}D)}\;\overline{(\overline{A}\,\overline{B}C)}}\,. \tag{7}$$

Description of experimental procedure

First, you will develop truth table and description of the operation of your logic circuit. Next, you will implement the circuit using available logic gates. Implementation and testing of the circuit are performing on the experimental setup composed of the following modules:
a) the block of logic state switches,
b) the table of some logic gates,
c) the block of logic probe testers.

The response of your circuit will be investigated by logic probe testers for all combinations of input states. Next, the experimental truth table will be compared with the theoretical one.

The block of logic state switches are composed of five switches, which allow to select 0 or 1 logic state at the sockets below the switches. When more than five independent inputs must be driven, please use also the sockets on the bottom of the front panel, which has fixed 0 or 1 state. Moreover, the block contain three generators of single square pulse. The generators are used only to investigate sequential logic circuits, which go beyond the scope of the present experiment.

The table of logic gates contains 4 NOT gates, 8 two-input NAND gates, and 8 three-input NAND gates. All other functors must be implemented as a network of available NAND and NOT gates.

The block of logic probe testers contains 10 independent testers. There are a one input socket for each channel with red and green LED lights, which indicate the high and low logic state, respectively. When the investigated state changes continually the both lights may be switched on. Oscillations should not appear in correct combinational circuits, so if it occurs check the network of connections. When the probe tester input is in the high-resistance state or the connected voltage does not correspond to any specific logic state, the both lights are switched off. In the case of connection made between the probe tester and any output of gate or logic state switch, such a situation indicates a failure of any wire or device, or a lack of power supply.

Experimental procedure

A. Verification of De Morgan's law

1. Use De Morgan's law related to logical disjunction (see Table 1) to draw a diagram of the circuit that implements two-input OR gate using available NAND and NOT gates.
2. Construct the circuit, wire the inputs to the switches of logic state, and wire the output to the logic probe tester.
3. Connect the power supply to all experimental modules. In order to do this connect the extreme left or extreme right module directly to the power supply. The other modules will be supplied through the connections of analogous +5V lines (on the top) and 0V lines (at the bottom) between the neighboring modules.
   WARNING:
   a) **all modules must be supplied from the power supply channel that provides constant output voltage +5 V (sockets on the right side of the power supply). Do not use the outputs allowing voltage adjustment,**
   b) **Do not connect +5V voltage output directly to the outputs of gates nor outputs in the block of logic state switches.**

   Ignoring these recommendations threatens to damage the devices.
4. After obtaining permission switch on the power supply and check out the status of the red LEDs on the +5V line. The LEDs related to the 0 and 1 logic states should be off when the appropriate input of logic probe tester is not connected.
5. Set up all possible combinations of input logic states and write down the truth table for the circuit. Compare your experimental truth table with the theoretical one shown in Fig. 1. In the case of any discrepancies check out the circuit again. If you can not resolve the problem consult with the laboratory staff.

B. Design and implementation of combinational logic circuit

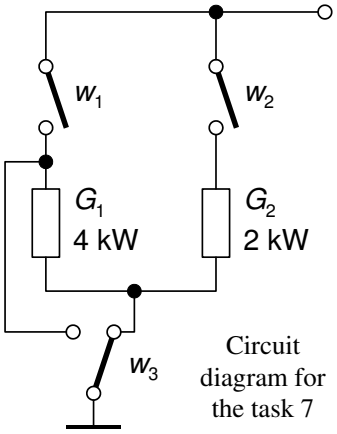
1. Consult the supervisor in order to select some tasks to solve. Some examples of tasks are collected in the next chapter.
2. Develop a theoretical truth table (see example show in Table 1). The cases when the state of output does not matter write as a dash.
3. Try to simplify your Boolean function using the Karnaugh map method. Consider a grouping of "1s" (see example shown in Fig. 3.a) as well as grouping of "0s" (Fig. 3.b).
4. Use the laws of Boolean algebra to transform simplified function. After transformation the function must be written employing only these elementary operators, which are directly implemented in available hardware (the table of NOT and two- and three-inputs NAND gates allow to use only $\overline{A}$, $\overline{A \cdot B}$, and $\overline{A \cdot B \cdot C}$ operators). If more than one output is necessary in the circuit, find common expressions in the Boolean functions related to the individual outputs. Try to simplify the circuit by implementing the common expressions only ones.
5. Switch on the power supply.
6. Construct the combinational circuit according to the functions derived in step 4. Do not assume, that not connected inputs of gates have any specific logic state. Use jumpers to connect redundant inputs to any other input of the same gate. Alternatively, wire redundant inputs to the sockets, which has fixed 0 or 1 logic state. The changes in connections may be made safely when the power is switched on. The devices are protected from wrong connections between some outputs of gates or outputs in the block of switches.
7. Write down experimental truth table for all outputs of the circuit.
8. Check out if the experimental truth tables is consistent with this developed in point 2. In the case of any discrepancies try to find their reasons. First, try to move plugs in sockets and look for any broken wire or jumper. Next, analyze again the transformations made in steps 3 and 4 and check out the connections made in point 6. Investigate again the circuit after correction and write down the new truth table. If the discrepancies between the assumptions and experimental results still exists, report the problem to the supervisor.
9. Switch off the power supply and draw the schematic diagram of the circuit.
10. Disconnect the circuit.

Examples of problems to solve by combinational logic circuit

The number of multiplication marks in brackets describes the difficulty of the task. The choice of the simplest of tasks may make it impossible to obtain the maximum assessment.

1. (*) Design and implement the circuit indicating divisibility of three-bits binary number by 3. Let note whether zero was considered as divisible by 3 or not.
2. (*) Design and implement the circuit working as a four-input NOR gate using only NOT and NAND gates with at most three inputs.
3. (*) Design and implement the driver controlling the line consisting of three LEDs, which are turned on sequentially. Number of currently shining LEDs is given by the two-bits binary value at the input of controller.
4. (**) Design and implement the driver, which allows to switch on and switch off the light in a large room using any single switch from among three switches mounted in three different places. The driver should change the state of its output after any change made on any single input, i.e. the driver should work as a three-bit even (or odd) parity generator. Tip: in case of a simpler system with only two switches the problem may be solved by a single EX-OR or EX-NOR gate shown in Fig. 2.
5. (**) Design and implement two-bit comparator of two binary values *BA* and *DC* (where *A*, *B*, *C*, *D* means the individual bits). The circuit should maintain the state 1 on the output for $DC \geq BA$ and state 0 for $DC < BA$.
6. (**) Design and implement the circuit indicating a failure of electric contact thermometer, which has four contacts *A*, *B*, *C*, *D*. When the temperature increases the contacts change its state from 0 to 1 in mentioned order. All non-sequential combinations, i.e. when any contact stay opened below the closed contact, should be interpreted as the failure of the thermometer.

7. (***) Design and implement the control system for a dryer equipped with two heaters G_1 and G_2 with a power 4 kW and 2 kW, respectively. The heaters can be connected in different ways by electromagnetic switches w_1, w_2, and w_3. The settings of switches shown in the diagram is related to the logical 0, while the position opposed consider as a logical 1. The temperature in a drying chamber is measured by an electric contact thermometer, which has four contacts A, B, C, D. When the temperature is increasing the contacts change its state from 0 to 1 in mentioned order. Let t_i denote the temperature of switching of the i-th contact. Mode of action of the control system will be as follows:
   $t < t_A$ – both heaters are connected in parallel,
   $t_A \le t < t_B$ – only the heater G_1 is switched on,
   $t_B \le t < t_C$ – only the heater G_2 is switched on,
   $t_C \le t < t_D$ – both heaters are connected in series,
   $t_D \le t$ – both heaters are switched off.



Circuit diagram for the task 7

8. (***) Design and implement the driver controlling the line consisting of four LEDs, which are turned on sequentially. Number of currently shining LEDs is given by the four-bit binary value at the input of controller. When this value is greater than 0100 (4 in decimal code) all LEDs should be lit up.
9. (****) Design and implement a 1-bit full adder. The circuit performs an addition operation on three binary digits A_i, B_i, and carry-in C_{i-1}. The latter input allows to take into account the carry-out of the previous adder. The adder produces a sum S_i and carry-out signal C_i for the next adder operating on more significant bits.

Report elaboration

Report has to be composed of:

1. Front page (by using a pattern).
2. Description of experiment purpose.
3. List of used instruments and devices (id number and type). For the table of logic gates provide full specification of available gates (types of gates, number of inputs, number of available gates).
4. Description of De Morgan's law verification.
5. Description of the problem, which should be solved by your combinational circuit.
6. Theoretical truth table for each output.
7. Minimization of logical function by the Karnaugh map method.
8. Transformation of obtained minimized functions to the form, which may be directly implemented using available gates.
9. Schematic diagrams of designed combinational circuits.
10. Experimental truth table.
11. Discussion and conclusions. Compare the results obtained with your theoretical assumptions. In the case of any discrepancy, describe your attempts to remove them, found errors and finally obtained result. Have you found the Karnaugh map method helpful for finding non-trivial simplification of your circuit?

References

[1] J. Kalisz, *Podstawy elektroniki cyfrowej*, WKiŁ, Warszawa 2002.
[2] P. Horowitz, W. Hill, *Sztuka elektroniki*, WKiŁ, Warszawa 2001,
[3] U. Tietze, Ch. Schenk, *Układy półprzewodnikowe*, WNT, Warszawa 1987.
[4] M. Molski, *Wstęp do techniki cyfrowej*, WKiŁ, Warszawa 1989.
[5] R. Ćwirko, M. Rusek, W. Marciniak, *Układy scalone w pytaniach i odpowiedziach*, WNT, Warszawa, 1987.
[6] W. Traczyk, *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*, WNT, Warszawa 1986.
[7] P. Misiurewicz, *Układy automatyki cyfrowej,* Wydawnictwa Szkolne i Pedagogiczne, Warszawa, 1984.
[8] A. Rusek, *Podstawy elektroniki*, część 2, Wydawnictwa Szkolne i Pedagogiczne, Warszawa, 1983.
[9] B. Zbierzchowski, T. Łuba, K. Jasiński, M. A. Markowski, *Synteza logiczna w układach programowalnych*, Wydawnictwa Politechniki Warszawskiej, Warszawa, 1992.