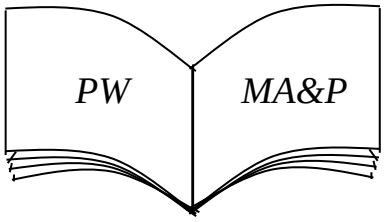
MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY
«KHARKIV POLYTECHNIC INSTITUTE»

METHODOLOGICAL INSTRUCTIONS
for practical work in the
«Microprocessor architecture and programming» academic discipline

for full-time and part-time students
majoring in «Computer Engineering»



Kharkiv 2024

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY
«KHARKIV POLYTECHNIC INSTITUTE»

METHODOLOGICAL INSTRUCTIONS
for practical work in the
«Microprocessor architecture and programming» academic discipline

for full-time and part-time students
majoring in «Computer Engineering»

Approved
by editorial and publishing
department of the NTU «KhPI»,
protocol № 3 dated 24.10.2024

Kharkiv
NTU «KhPI»
2024

Methodological instructions for practical work in the «Microprocessor architecture and programming» academic discipline for full-time and part-time students majoring in «Computer Engineering» / A. Podorozhniak, H. Heiko, S. Mezherytskii, O. Liubchenko. – Kharkiv : NTU «KhPI», 2024. – 90 p.

Authors: A. Podorozhniak, H. Heiko, S. Mezherytskii, O. Liubchenko

Reviewer: prof. V. Usik

Department of Computer Engineering and Programming

INTRODUCTION

The methodical instructions contain a methodology for performing practical classes, the purpose of which is to provide students with special skills in the basic basics of architecture, programming and modes of operation of microprocessors and microprocessor facilities and to familiarize them with the direct memory access controller, interrupt controller, system timer and real-time clock; study of the main modes of operation of a microprocessor, including various options for solving practical problems using a microprocessor and multitasking mode.

When conducting practical work, the following provisions should be followed:

1) practical classes are conducted face-to-face in the entire group; the volume of tasks is determined by the teacher;

2) each practical work requires independent preparation, including the study of theoretical material and implementation the necessary project work, theoretical analysis of the developed schemes, construction of time diagrams;

3) in the course of practical work, students must assemble the scheme on a universal circuit board or in the modeling program and perform its research;

4) a report is drawn up for each practical work, which includes: the topic and purpose of the work, an individual task, research results with explanations, conclusions on all experiments;

5) all developed circuits and research results created in simulation programs must be presented to the teacher to demonstrate the functionality of the circuit or device being modeled.

When submitting a practical work report, students should be ready to answer the control questions that are in methodical instructions and additional questions based on the material being studied.

Practical work 1

RESEARCH OF THE ORGANIZATION OF THE LEDS BLINKING PROCESS ON THE ATMEGA328 MICROPROCESSOR

The purpose of the work: Investigation of the organization of the process of flashing LEDs on the ATMega328 microprocessor on the Arduino platform. Obtaining practical skills in issuing a given signal to the pins of the ports of the ATMega328 microprocessor on the Arduino UNO R3 board.

Topics for preliminary study:

– theoretical information about the ATMega328 microprocessor;

– theoretical information about the Arduino platform;

– theoretical information about the Arduino IDE environment;

– theoretical information about the design of microprocessor devices in the PROTEUS VSM environment.

1. Introduction to the basic principles of connecting LEDs to microprocessors and color coding of resistors

A light-emitting diode (LED) is a semiconductor device that emits incoherent light when an electric current is passed through it. The work is based on the physical phenomenon of the occurrence of light radiation when an electric current passes through a *p-n* junction. The color of the glow (wavelength of the maximum emission spectrum) is determined by the type of semiconductor materials used that form *p-n* transition (Fig. 1.1).

Figure 1.1 – Light-emitting diodes

Advantages of LEDs:

1) LEDs do not have any glass bulbs and filaments, which ensures high mechanical strength and reliability (shock and vibration resistance);

2) the absence of heating and high voltage guarantees a high level of electrical and fire safety;

3) inertia-free makes LEDs indispensable when high speed is required;

4) miniaturization;

5) long service life;

6) high efficiency;

7) relatively low supply voltages and current consumption, low power consumption;

8) a large number of different colors of glow and directionality of radiation;

9) light adjustable intensity.

У світлодіодів є декілька основних параметрів:

LEDs have several basic parameters:

– type of hull;

– typical (operating) current and typical (operating) voltage;

– glow color (wavelength, nm);

– scattering angle.

In general, the type of body refers to the diameter and color of the bulb (lens). The LED must be powered with a current called typical. At the same time, a certain voltage drops on the LED. The color of the radiation is determined by both the semiconductor materials used and the alloying impurities.

The most important elements used in LEDs are aluminum (Al), gallium (Ga), indium (In), phosphorus (P), causing a glow in the range of red to yellow. Indium (In), gallium (Ga), nitrogen (N) are used to acquire a blue and green glow. In addition, if we add a phosphor to the crystal that causes the blue (blue) glow, we get a white color. The emission angle is also determined by the production characteristics of the materials, as well as the bulb (lens) of the LED.

Connection diagram and calculation of the required parameters

Since the LED is a semiconductor device, the polarity must be observed when

connected to an electrical circuit. The LED has two leads, one of which is the cathode ("minus") and the other is the anode ("plus"). To properly connect an LED in the simplest case, it is necessary to connect it through a current-limiting resistor (Fig. 1.2).

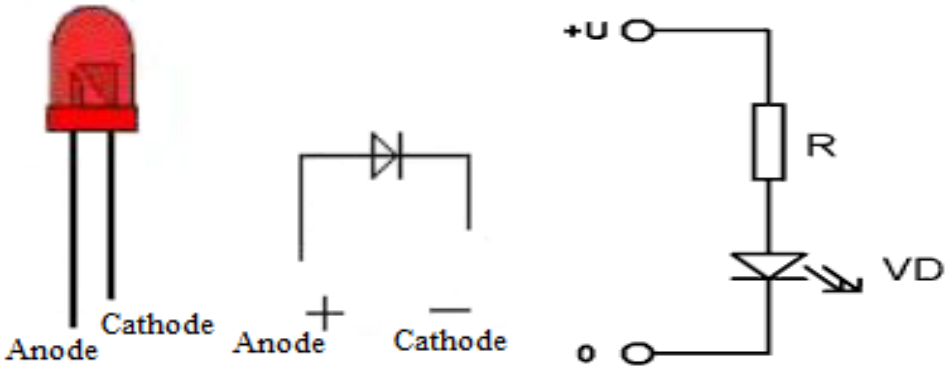


Figure 1.2 – LED wiring diagram

In our case, the calculation of the resistance of the current-limiting resistor is carried out according to the formulas:

$$R = U_{\text{current-limiting}} / I_{\text{LED}}$$

$$U_{\text{current-limiting}} = U_{\text{power}} - U_{\text{LED}}$$

In the case of using the ATMega328 microprocessor with a supply voltage of 5 V, we use $R = 200$ Ω.

Color-coded resistors

There are three strips on the left and one slightly offset on the right (Fig. 1.3). Each colored strip corresponds to a certain number (Table 1.1).

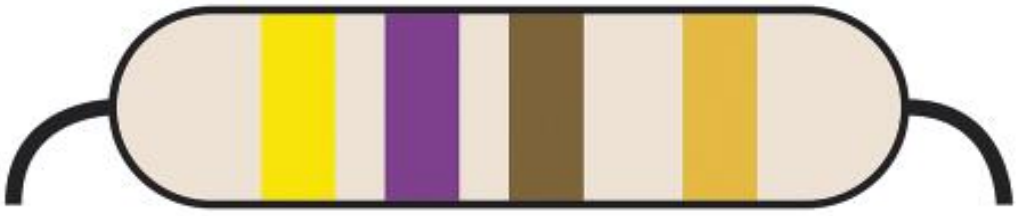


Figure 1.3 – Image of a 470 Ω resistor with color coding

Table 1.1 – Color numeric coding

Color	Digit	Color	Digit
Black	0	Green	5
Brown	1	Blue	6
Red	2	Purple	7
Orange	3	Grey	8
Yellow	4	White	9

If you look at the first two stripes (Fig. 1.3), this is the basic value, so we get the number "47". The third band is the multiplier, its numerical value is the number of zeros to be added to the number 47. Since in Fig. 1.3 the third band is brown, you need to add one "0", that is, the resistor value will be 470 Ω.

The fourth strip in Fig. 1.3 denotes the accuracy of the resistance of the resistor, the so-called tolerance, which indicates the discrepancy in the rating as a percentage. This is important when drawing up circuits in which resistance must be calibrated. The accuracy of the resistor rating is indicated as a percentage, and there are special color codes for the fourth band (Table 1.2).

Since in Fig. 1.3 The fourth band is gold, then the accuracy of the resistor will be ± 5%, that is, the true resistance value of the resistor will be between 447 Ω and 494 Ω.

Table 1.2 – Color coding of resistor rating accuracy

Color	Accuracy
Silver	± 10 %
Golden	± 5 %
Red	± 2 %
Brown	± 1 %

2. Introduction to the flashing process of the built-in LED on the Arduino UNO R3 board

Arduino is an electronic constructor and a convenient platform for the rapid

development of electronic devices for beginners and professionals. Advantages of the platform: easy to use, simplicity of the programming language, open architecture and program code. The device can be programmed via USB without the use of programmers.

Arduino-based devices can receive information about the environment through various sensors and can also control different actuators.

There are several versions of Arduino platforms. We will be looking at the Arduino UNO R3 platform (Fig. 1.4), which is based on the Atmel ATmega328 microcontroller. The microcontroller on the board is programmable using the Arduino C++ language in the Arduino development environment (based on the Processing environment). Arduino-based device designs can work on their own, or interact with software on the computer (e.g., Flash, Processing, MaxMSP).

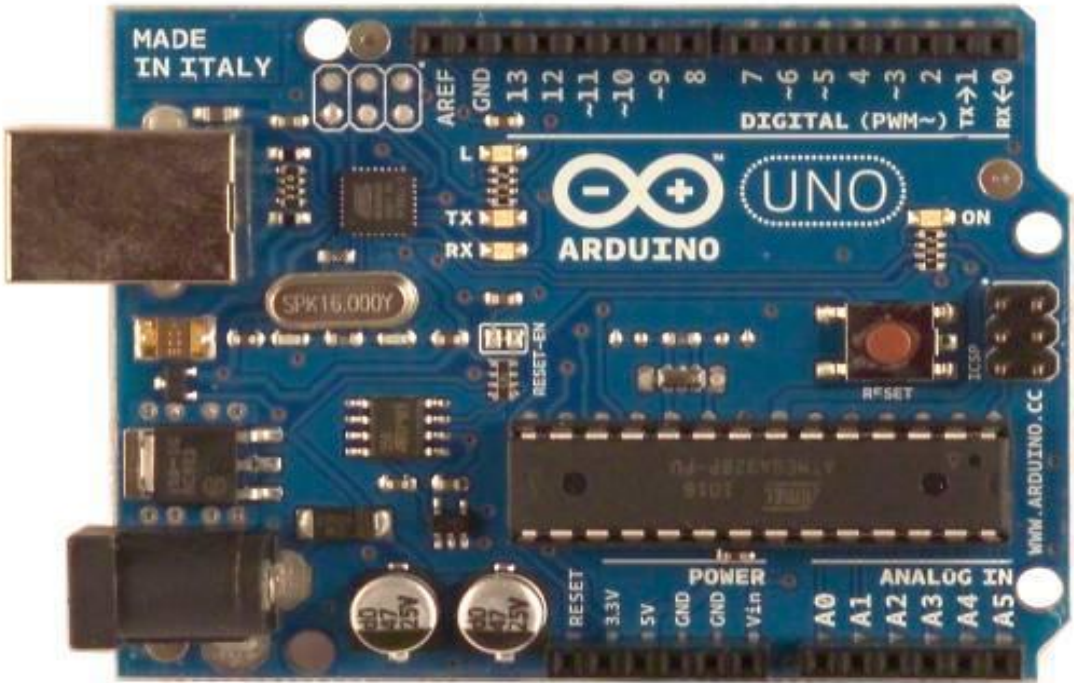


Figure 1.4 – Appearance of Arduino UNO R3

Boards can be collected by the user independently. The software is available for free download. Source schematic drawings (CAD files) are publicly available.

The platform has 14 digital inputs/outputs (6 of which can be used as pulse width modulation outputs), 6 analog inputs, a 16 MHz quartz oscillator, a USB connector, a power connector, an ICSP connector, and a reset button. Each of the UNO's 14 digital pins can be configured as input or output using the pinMode(), digitalWrite(), digitalRead()

functions. Each pin has a load resistor (disabled by default) 20-50 kΩ and can transmit currents up to 40 mA. To work, you need to connect the platform to a computer using a USB cable, or apply power using an AC/DC adapter or battery.

The platform can operate with external power supply from 6V to 20V. With a supply voltage below 7V, the 5V pin can output less than 5V, while the platform may be unstable. When using voltages above 12V, the voltage regulator may overheat and damage the board. Recommended range from 7 V to 12 V.

Some pins have special functions:

– VIN. The input is used to supply power from an external source (in the absence of 5 V from the USB connector or other regulated power supply). The supply voltage is supplied through this pin;

– 5V. Adjustable voltage source used to power the microcontroller and components on the board. Power can be supplied from the VIN terminal through a voltage regulator, or from a USB connector, or from another adjustable 5V voltage source;

– 3V3. The voltage on the 3.3 V pin is generated by a built-in regulator on the board. Maximum current consumption 50 mA;

– GND (Common Ground Output);

– serial bus: 0 (RX) and 1 (TX). The pins are used to receive (RX) and transmit (TX) TTL data. These pins are connected to the corresponding connectors on the ATmega8U2 USB-to-TTL serial bus chip.

– external interrupt: 2 and 3. These pins can be configured to call an interrupt either at a lower value, or at the front or back edge, or when the value changes. Detailed information can be found in the description of the attach Interrupt() function;

– PWM: 3, 5, 6, 9, 10, and 11. Any of the pins provides 8-bit PWM using the analogWrite() function;

– SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). With the help of these pins, SPI communication is carried out, for which the SPI library is used;

– LED: 13. Built-in LED connected to digital pin 13. If the value on the pin has a high potential, then the LED is on.

The UNO platform is equipped with 6 analog inputs (labeled as A0... A5), each with

a resolution of 10 bits (i.e. can take 1024 different values). By default, the pins have a measuring range of up to 5 V relative to the ground, but it is possible to change the upper limit using the AREF pin and the analogReference() function.

Some pins have additional functions:

– I2C: 4 (SDA) and 5 (SCL). The pins are used for I2C (TWI) communication, which is created using the Wire library.

An additional pair of platform pins:

– AREF. Reference voltage for analog inputs. Used with the analogReference() function;

– Reset. A low signal level on the pin reboots the microcontroller. It is usually used to connect the reboot button on the expansion board, which closes access to the button on the Arduino board.

Basic functions for working with LEDs

To work with an LED, you need to know and be able to use the following functions and constants:

– operator setup();

– operator loop();

– pinMode() function;

– digitalWrite() function;

– delay() function;

– constants OUTPUT, HIGH, LOW.

The following is the program code for the simplest example of flashing an LED built into an Arduino board that is connected to pin 13:

```
void setup()
{
pinMode(13, OUTPUT);
}
```

This function is performed at the beginning of the program (after the microcontroller starts). That is, each command that is between the curly braces of this function is sequentially

executed. At the end of each line, you must put the ending character of the command ";". The setup function contains the command pinMode(13, OUTPUT). This command configures the 13th port of the Arduino as the output. Port 13 is on the top pad of the Arduino ports. After the setup function, the loop function is executed.

```
void loop()
{
digitalWrite(13, HIGH); // Turn on the LED
delay(1000); // Wait a second
digitalWrite(13, LOW); // Turn off the LED
delay(1000); // Wait a second
}
```

Unlike setup, the loop function is constantly repeating – as soon as all the commands in parentheses are executed consecutively, the function starts again. The loop function for this example consists of four commands:

1) port 13 is supplied with voltage (5 V) – the LED turns on;

2) delay until the next command is executed 1000 ms (1 sec.);

3) port 13 is connected to the "ground" – the LED is turned off;

4) another delay of 1 second.

After all four commands have been executed, the first command is executed again (turning on the LED) and thus the flashing of the LED (circled with a red line in Fig. 1.5) continues as long as the Arduino is turned on, or until the RESET button is pressed.

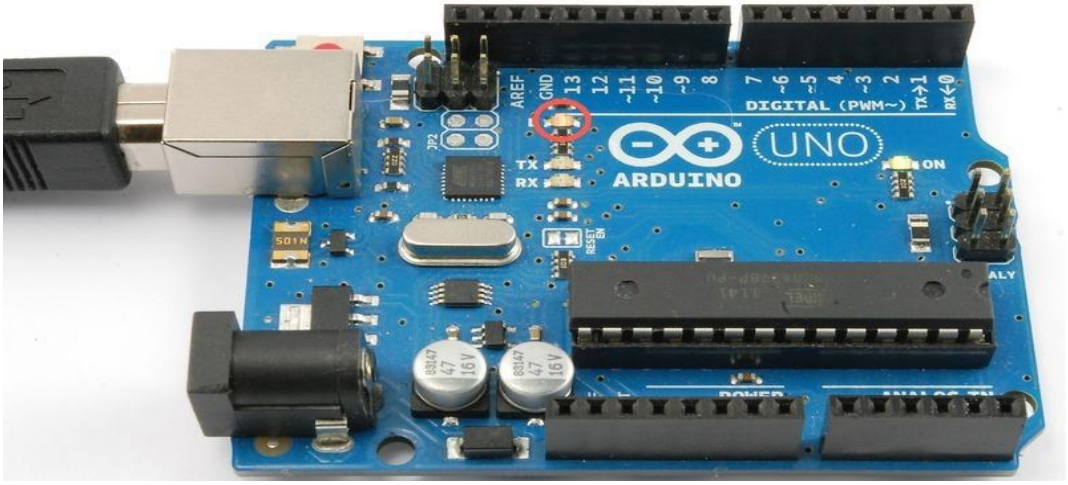


Figure 1.5 – LED flashing on Arduino UNO R3

3. Introduction to the process of flashing the LED located on the breadboard using the ATMega328 microprocessor

To ensure that the LED on the Arduino flashes and controls it, you will need:

– Arduino board;

– breadboard;

– two wires "daddy-daddy";

–Led;

– 200 Ω resistor.

The breadboard is a grid of sockets, which are usually connected as shown in Fig. 1.6.

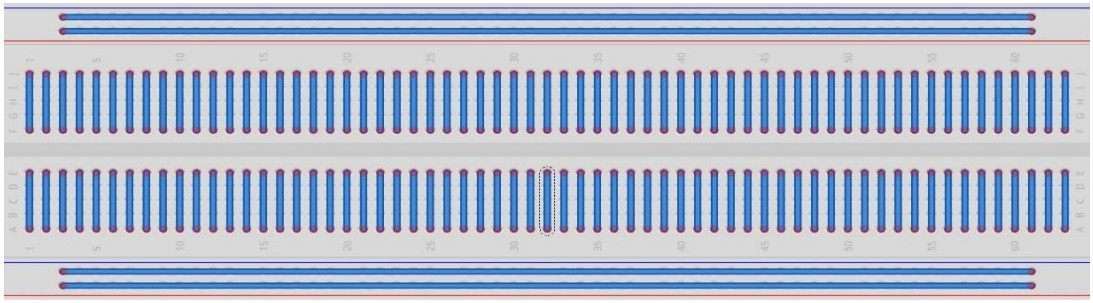


Figure 1.6 – Connections on the circuit board (breadboard)

The wiring diagram of the LED on a separate circuit board to ensure that the LED on the Arduino flashes is shown in Fig. 1.7.

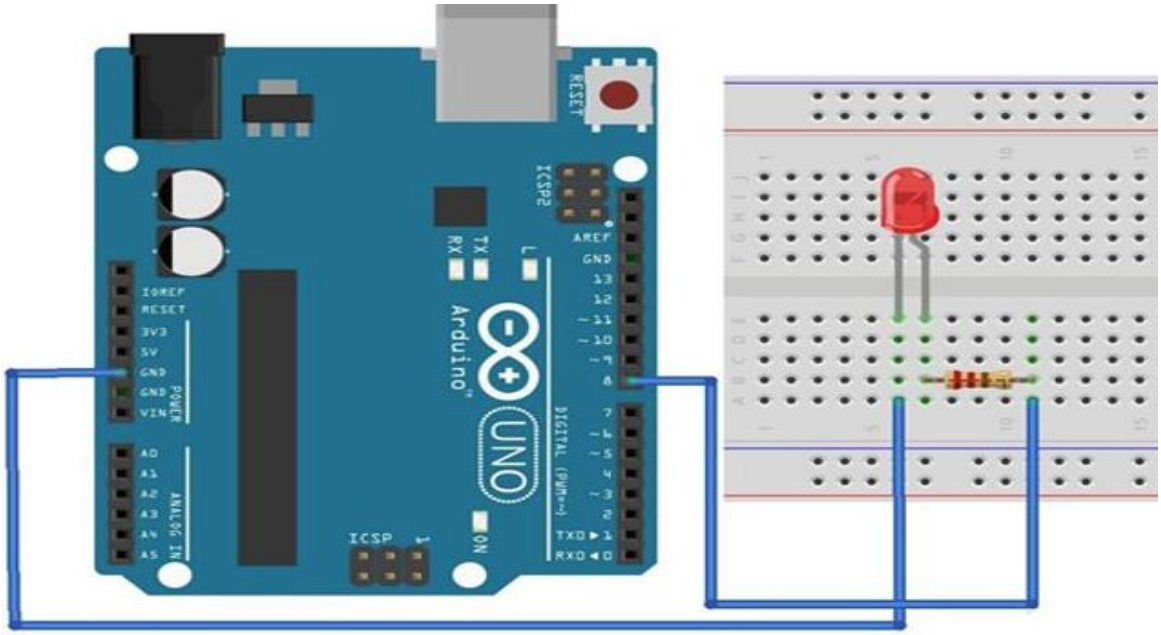


Figure 1.7 – LED wiring diagram

The following program can be used for this model to work:

```
int led = 8;
void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
```

Then it's the same with comments:

```
int led = 8; // declaring an integer variable containing the number of the port to which the second wire is connected
void setup () // Mandatory setup procedure, which starts at the beginning of the program; The announcement of procedures begins with the word void
{
pinMode (led, OUTPUT); // declaration of the port used, led – port number, second argument – type of port usage: INPUT or OUTPUT
}
void loop () // A mandatory loop procedure that runs cyclically after the setup procedure
{
digitalWrite (led, HIGH); // This command is used to turn the voltage on or off on the digital port; led is the port number, the second argument is on (HIGH) or off (LOW)
delay (1000); // This command is used to wait between actions, the argument being the timeout in milliseconds
digitalWrite (led, LOW);
delay (1000);
```

The assembly is shown in Fig. 1.8.

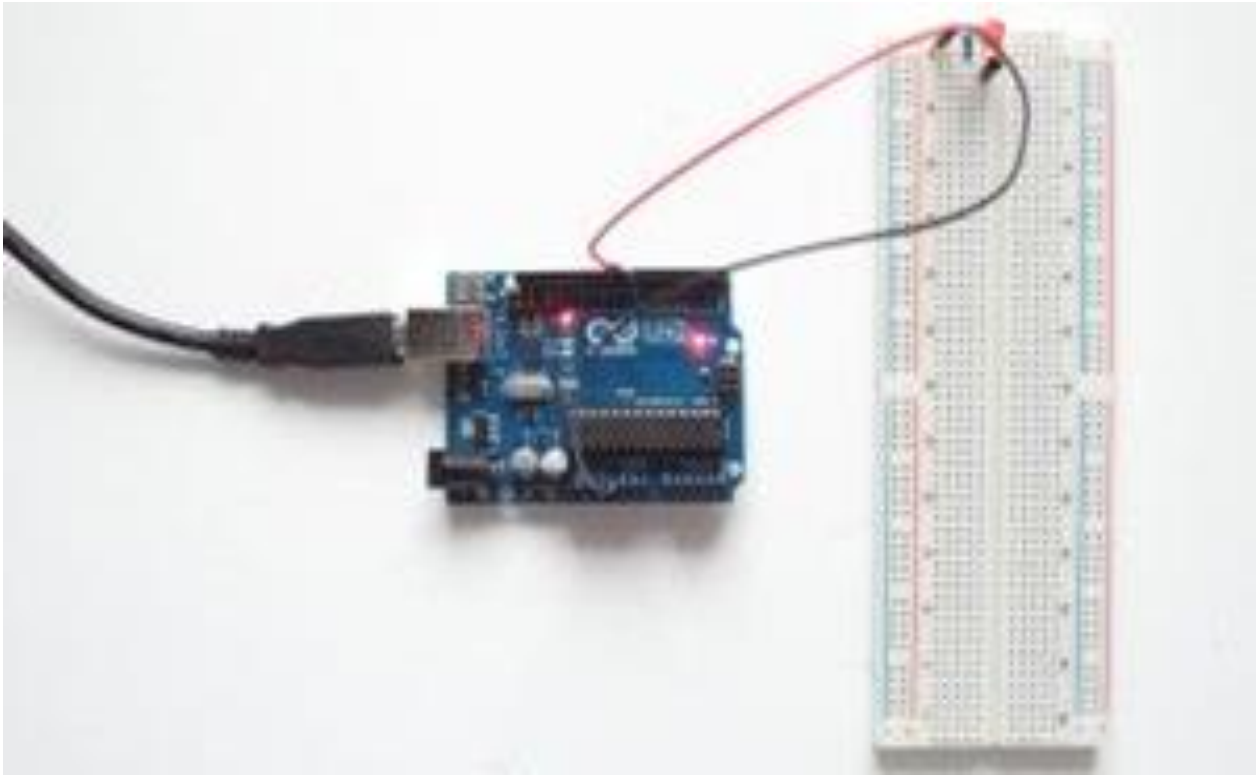


Figure 1.8 – Flashing of the LED located on the breadboard

4. Individual task

Ensure that two LEDs of the specified color flash on the specified ports with the specified burning and non-burning time of each LED. Variants of tasks are given in Table 1.3.

Table 1.3 – Task Options

No. var.	LED 1				LED 2			
	N Port	Color	LED on time, s	LED off time, s	N Port	Color	LED on time, s	LED off time, s
1	2	red	0,5	1	4	blue	1	0,8
2	3	yellow	0,2	0,2	5	red	0,8	1,6
3	4	blue	0,7	1	6	yellow	1,5	0,7
4	5	red	2	1	7	blue	0,5	0,5
5	6	yellow	1	0,5	8	red	0,3	0,3
6	7	blue	0,4	0,4	9	yellow	1	2
7	8	red	1	0,1	7	blue	2	0,4
8	9	yellow	1,5	0,5	6	red	0,5	0,8
9	2	blue	0,6	1,2	5	yellow	0,8	1,6
10	3	red	1,4	0,7	4	blue	0,5	1,8

Order of work

1. Familiarize yourself with the basic principles of connecting LEDs to a microprocessor and color coding of resistors.

2. Familiarize yourself with the process of flashing the built-in LED on the Arduino UNO R3 board.

3. Familiarize yourself with the flashing process of the LED located on the breadboard using the ATMega328 microprocessor.

4. Develop a flashing program with two LEDs according to the individual task.

5. Develop a model of the scheme according to the individual task in the Tinkercad environment and check its operation.

6. Develop a model of the scheme according to the individual task in the PROTEUS environment and check its operation.

7. Assemble the circuit on the circuit board according to the individual task, enter the program into the Arduino UNO R3, run and check its operation.

8. Check the correct functioning of the program in simulation environments and on a real diagram.

9. Draw up a work report.

Contents of the report

1. The topic of the practical work.

2. Purpose of the work.

3. Individual task.

4. Program and simplified block diagram of the algorithm for organizing the flashing process with a built-in LED on the Arduino UNO R3 board.

5. Program and simplified block diagram of the algorithm for organizing the process of flashing an LED located on the breadboard using the ATMega328 microprocessor.

6. The program and a simplified block diagram of the algorithm for organizing the process of flashing with two LEDs according to an individual task.

7. Working circuit model for flashing with two LEDs according to an individual task in the Tinkercad environment.

8. Working circuit model for flashing with two LEDs according to an individual task in the PROTEUS environment.

9. Photo of the working assembled circuit on the circuit board for flashing with two LEDs according to the individual task.

10. Conclusions.

Practical work 2

RESEARCH OF THE ORGANIZATION OF INFORMATION OUTPUT ON SEVEN-SEGMENT INDICATORS ON THE ATMEGA328 MICROPROCESSOR

The purpose of the work: To study the organization of the process of displaying information on seven-segment indicators on the ATMega328 microprocessor on the Arduino platform. Obtaining practical skills in issuing information to seven-segment indicators using the ATMega328 microprocessor on the Arduino UNO R3 board.

Topics for preliminary study:

– theoretical information about seven-segment LED indicators;

– theoretical information about dynamic indication.

1. Introduction to the basic principles of connecting LEDs to a microprocessor

The seven-segment LED indicator displays the symbol using seven LEDs – digit segments. The eighth LED illuminates the decimal point (Fig. 2.1). Segments are marked with Latin letters from "A" to "H".

The anodes or cathodes of each LED combine in the indicator and form a common wire. Therefore, there are indicators with a common anode (CA) (Fig. 2.3, *a*) and a common cathode (CC) (Fig. 2.3, *b*).

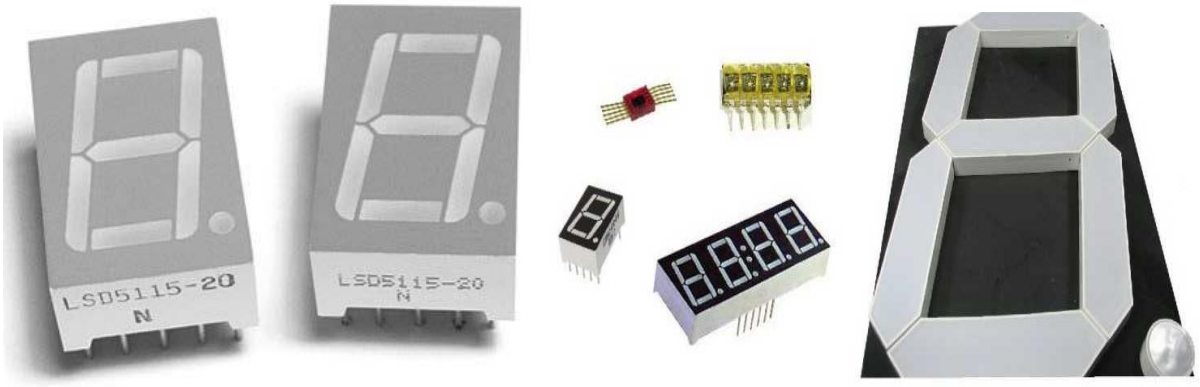


Figure 2.1 – Seven-segment LED indicators of various types

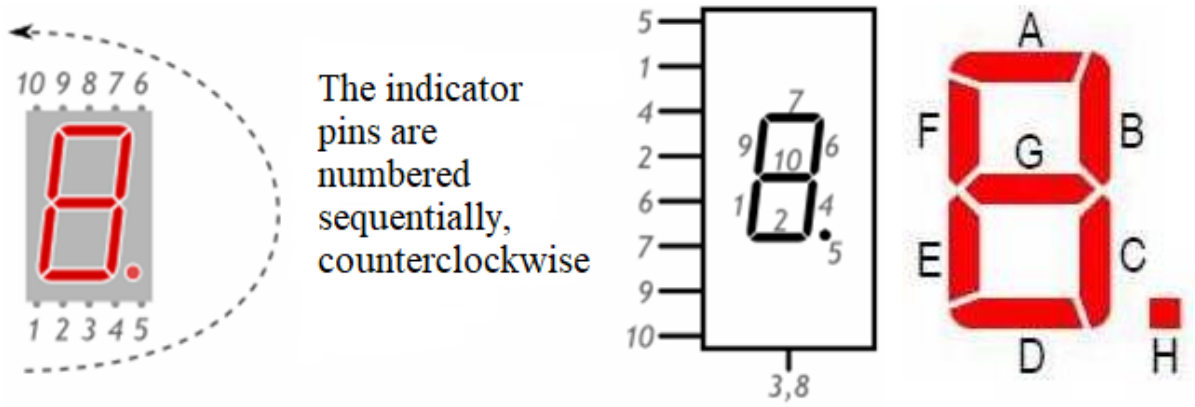
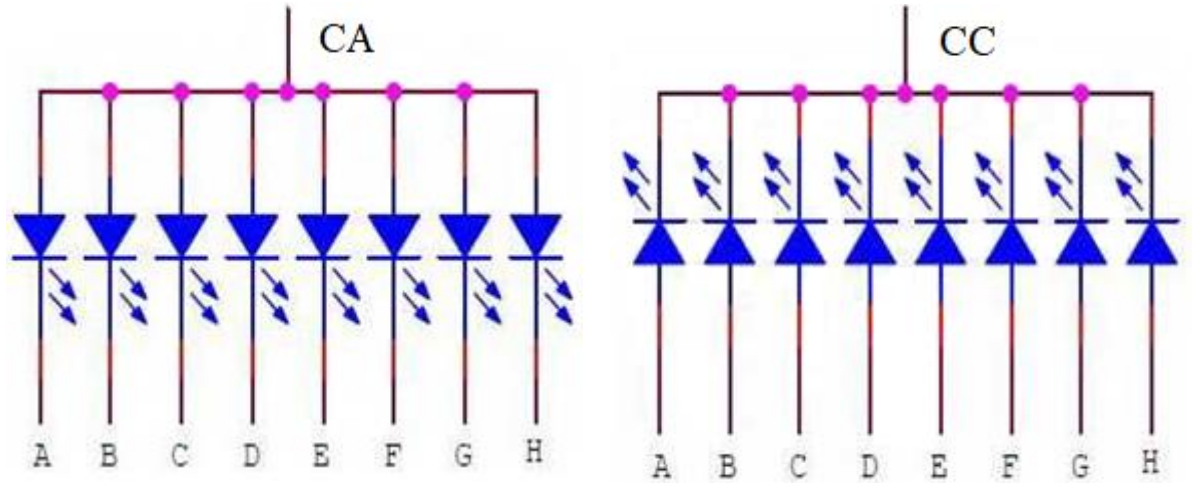


Figure 2.2 – Leg numbering and segment designations
in a seven-segment LED indicator

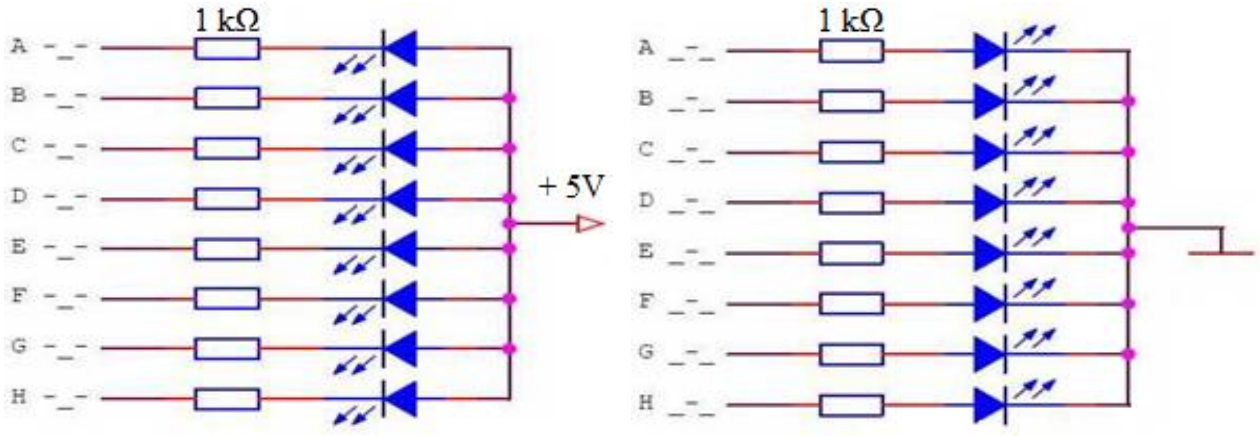


a) *b)*

Figure 2.3 – Schematic diagram of a seven-segment LED indicator
with common anode (*a*) and common cathode (*b*)

For static connections, the LED indicators must be connected to the microcontroller via resistors (200 Ω to 1.5 kΩ) that limit the current. Depending on the type of indicator (with a common cathode or a common anode), the polarity of the power supply and control signals changes (Fig. 2.4).

Eight pins are required to connect each seven-segment indicator to the microcontroller. If there are 3-4 indicators (bits), then the task becomes practically impossible, because there are not enough microcontroller pins.



а) *б)*

Figure 2.4 – Connection diagram of a seven-segment LED indicator with a common anode (*a*) and a common cathode (*b*)

When connecting a multi-digit seven-segment LED indicator to a microcontroller, a multiplexed mode is used, i.e. a dynamic indication mode.

The pins of the same name segments of each indicator are combined. The result is a matrix of LEDs connected between the segment terminals and the common indicator terminals. The scheme of multiplexed (dynamic) control of a two-digit indicator with a common anode is shown in Fig. 2.5.

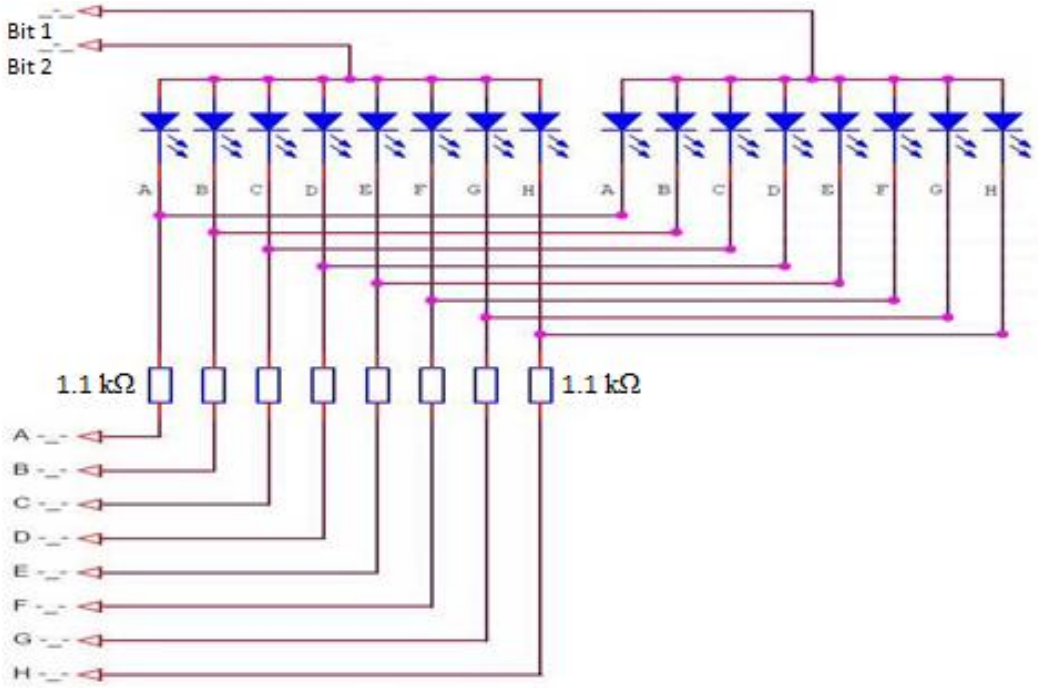


Figure 2.5 – Connection diagram of a two-digit indicator with common anode

The scheme of multiplexed (dynamic) control of a two-digit indicator with a common cathode is shown in Fig. 2.6.

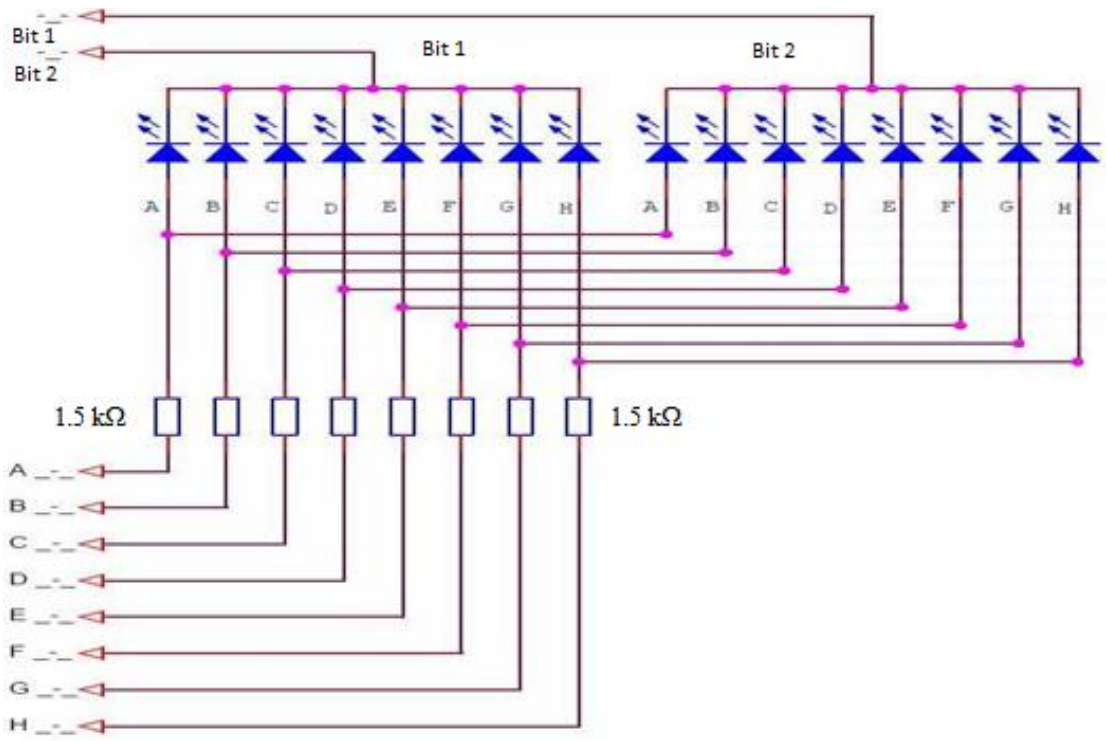


Figure 2.6 – Connection diagram of a two-digit indicator with common cathode

Relative to the circuit with a common anode, the polarity of all signals is reversed. Now a low level is applied to the common wire of the active discharge, and a high level is applied to the segments that should be lit. To connect a two-digit seven-segment indicator, 10 pins were enough instead of 18, and in the case of a four-digit indicator, there will be 12 pins instead of 36 pins.

2. Introduction to the process of displaying numbers and symbols on a seven-segment indicator using the ATMega328 microprocessor

To connect a single-digit LED indicator to an Arduino, 7 digital pins will be used (each pin of the A-G indicator is connected to the Arduino pins via a limiting resistor). In this case, a seven-segment indicator with a common cathode is used, the common wire is connected to the ground. The layout of the pins of the seven-segment indicator 5161AS is shown in Fig. 2.7. Fig. Figure 2.8 shows a diagram of connecting a single-digit seven-

segment indicator with a common cathode to an Arduino board.

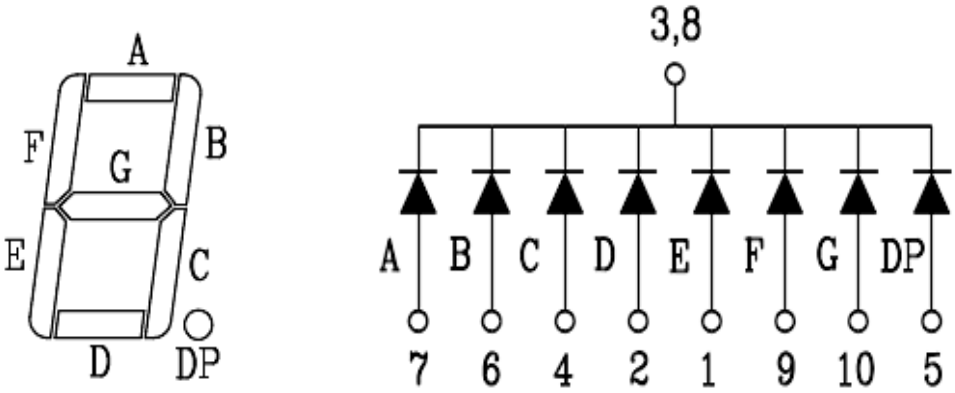


Figure 2.7 – Output diagram of the 5161AS seven-segment indicator

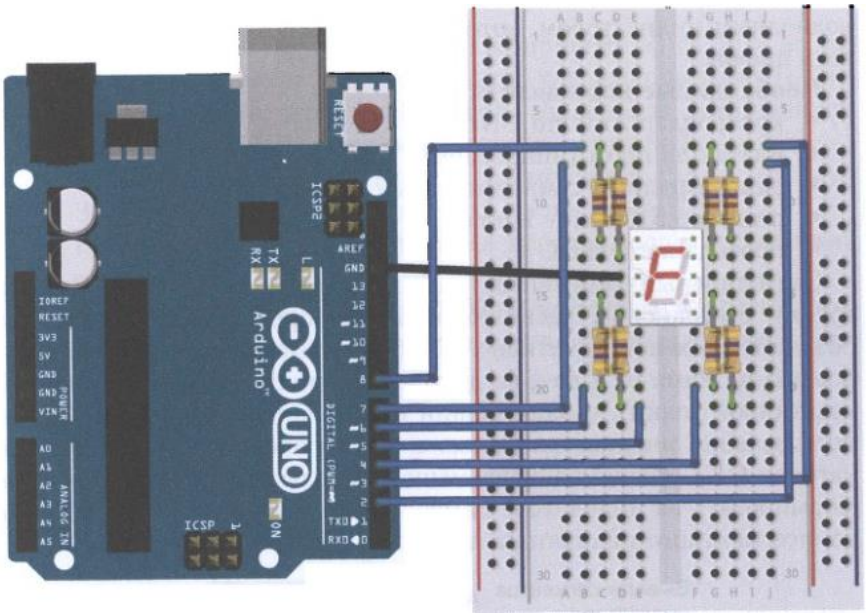


Figure 2.8 – Diagram of connecting the indicator to the Arduino board

On the seven-segment indicator in the cycle, we will display numbers from 0 to 9 with a pause of 1 second. Let's form an array of values for the digits 0... 9, where the highest bit of the byte corresponds to the label of segment A of the indicator, and the lowest bit corresponds to segment G:

byte numbers[10] = { B11111100, B01100000, B11011010, B11110010, B01100110, B10110110, B10111110, B11100000, B11111110, B11110110}; // variable to store the value of the current digit

The correspondence of the displayed sign to the port data is given in Table. 2.1.

Table 2.1 – Correspondence of the displayed sign to the data of the corresponding port

Sign	Common anode		Common cathode	
	Binary system	Decimal system	Binary system	Decimal system
«0»	00000011	3	11111100	252
«1»	10011111	159	01100000	96
«2»	00100101	37	11011010	218
«3»	00001101	13	11110010	242
«4»	10011001	153	01100110	102
«5»	01001001	73	10110110	182
«6»	01000001	65	10111110	190
«7»	00011111	31	11100000	224
«8»	00000001	1	11111110	254
«9»	00001001	9	11110110	246

To convert a digit to data to output a value to the Arduino pins, we will use the language's bit operations:

bitRead (x, n); // Retrieving the N bit value of byte X

Below is a program that outputs numbers from 0 to 9 on the indicator:

```
// List of pins for connection to the bits of the seven-segment indicator
int pins[7]={2,3,4,5,6,7,8};
// Values for the output of digits 0-9
byte numbers[10] = {B11111100, B01100000, B11011010, B11110010, B01100110,
B10110110, B10111110, B11100000, B11111110, B11100110};
// variable to store the value of the current digit
int number=0;
void setup()
  {
  // Configure Pins as Outputs
  for (int i=0;i<7;i++)
    pinMode(pins[i],OUTPUT);
  }
void loop()
  {
  showNumber(number);
  delay(1000);
  number=(number+1)%10;
  }
```

```
// Seven-segment display function
void showNumber(int num)
  {
  for (int i=0;i<7;i++)
    {
    if(bitRead(numbers[num],7-i)–HIGH) // Turn on Segment
      digitalWrite(pins[i],HIGH);
    else // Turn off Segment
      digitalWrite(pins[i],LOW);
    }
  }
```

Connection procedure:

1) connect the seven-segment indicator to the Arduino board according to Fig. 2.8;

2) load the above program into Arduino;

3) observe the display of numbers on the screen of a seven-segment indicator.

3. Introduction to the process of dynamic indication on the matrix of seven-segment indicators using the ATMega328 microprocessor on the Arduino UNO R3 board

Let's consider the process of dynamic indication on a four-digit matrix of seven-segment indicators with a common anode of 3641BS. The appearance of the indicator is shown in Fig. 2.9, the diagram is shown in Fig. 2.10, the location of the pins is in Fig. 2.11.

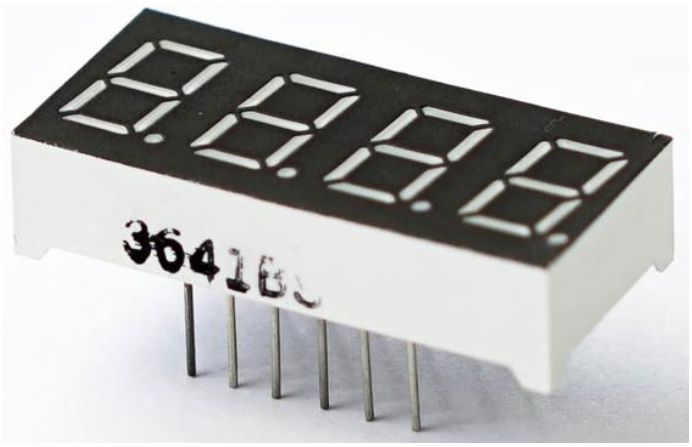


Figure 2.9 – Appearance of a four-digit matrix seven-segment indicators

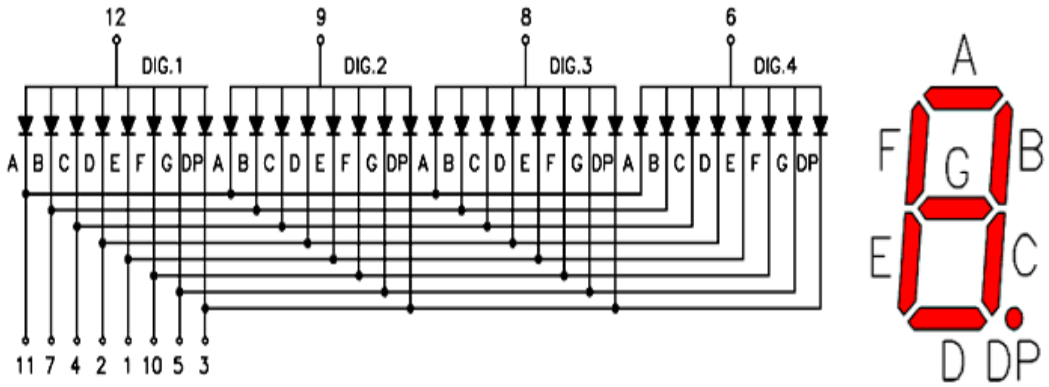
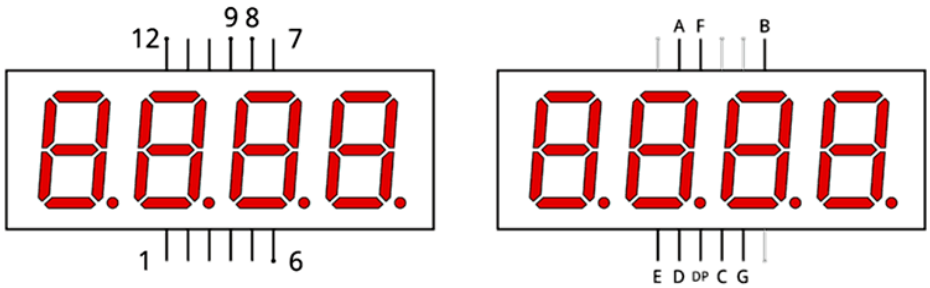


Figure 2.10 – Diagram of a four-digit matrix of seven-segment indicators



a) *b)*

Figure 2.11 – Placement of the pins of a four-digit matrix seven-segment indicators

Fig. 2.11 (*a*) – 12, 9, 8, 6 – these are the anode pins for each discharge, in Fig. 2.11 (*b*) – correspondence of the indicator pins to its segments.

By analogy with a seven-segment indicator, in order to display the necessary information, it is necessary to alternately apply voltage to the anodes, and the cathodes to form the necessary information – and so on for each discharge in turn. This should be done so quickly that it seems to a person that all four discharges are burning continuously at the same time. We assemble the diagram as shown in Fig. 2.12.

We check the performance of the scheme:

```
1   int anodPins[] = {A1, A2, A3, A4};
2   int segmentsPins[] = {5, 6, 7, 8, 9, 10, 11, 12};
3
4   void setup() {
5     for (int i = 0; i < 4; i++) {
6       pinMode(anodPins[i], OUTPUT);
7     }
```

```
8     for (int i = 0; i < 8; i++) {
9       pinMode(segmentsPins[i], OUTPUT);
10    }
11  }
12
13  int seg[] = {0, 1, 1, 0, 1, 1, 1, 0}; // Letter H
14
15  void loop() {
16     // display the letter H on all digits
17    for (int i = 0; i < 4; i++) {
18    for (int k = 0; k < 8; k++) {
19        digitalWrite(segmentsPins[k], ((seg[k] == 1) ? LOW : HIGH));
20     }
21     digitalWrite(anodPins[i], HIGH); // Energized the anode – all
22     // The lights are on
23     delay(1); // pause
24     digitalWrite(anodPins[i], LOW); // removed the voltage from the anode to
25     // Switching segments did not cause flickering
26    }
27  }
```

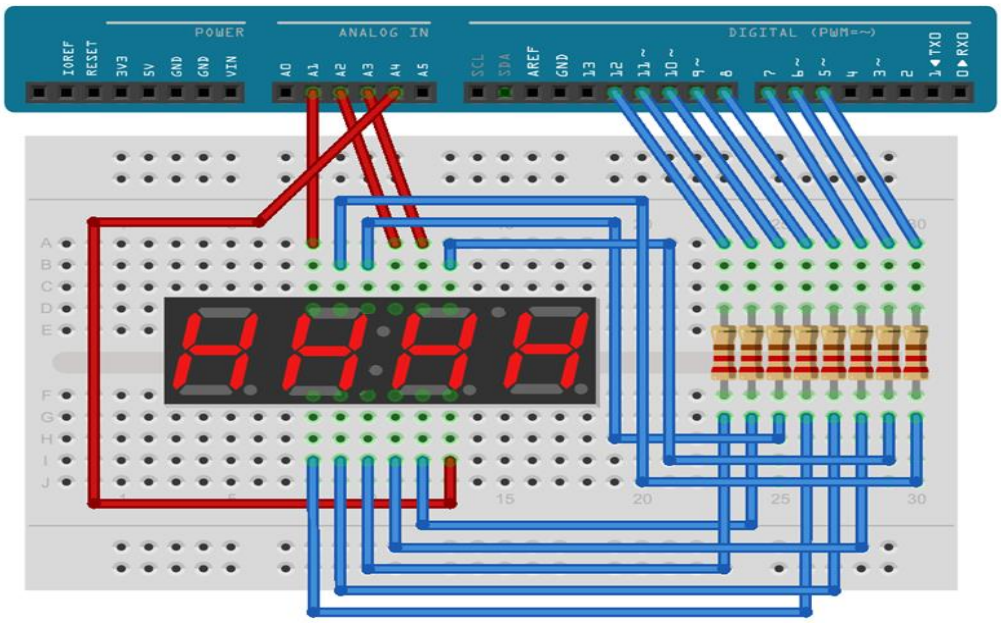


Figure 2.12 – Connection diagram of a four-digit matrix seven-segment indicators for Arduino UNO

Modifying the program code to display numbers:

```
1   int anodPins[] = {A1, A2, A3, A4}; // Set the pins for each digit
2   int segmentsPins[] = {5, 6, 7, 8, 9, 10, 11, 12}; // Set Pins for Everyone
```

```
3    // segment (from 7 + 1 (dot))
4   void setup() {
5     // all outputs are programmed as OUTPUT
6     for (int i = 0; i < 4; i++) {
7       pinMode(anodPins[i], OUTPUT);
8     }
9     for (int i = 0; i < 8; i++) {
10      pinMode(segmentsPins[i], OUTPUT);
11    }
12  }
13  //{A, B, C, D, E, F, G, DP} – Pinout segments
14  int seg[10][8] = {
15    {1, 1, 1, 1, 1, 1, 0, 0}, // digit 0
16    {0, 1, 1, 0, 0, 0, 0, 0}, // digit 1
17    {1, 1, 0, 1, 1, 0, 1, 0}, // digit 2
18    {1, 1, 1, 1, 0, 0, 1, 0}, // digit 3
19    {0, 1, 1, 0, 0, 1, 1, 0}, // digit 4
20    {1, 0, 1, 1, 0, 1, 1, 0}, // digit 5
21    {1, 0, 1, 1, 1, 1, 1, 0}, // digit 6
22    {1, 1, 1, 0, 0, 0, 0, 0}, // digit 7
23    {1, 1, 1, 1, 1, 1, 1, 0}, // digit 8
24    {1, 1, 1, 1, 0, 1, 1, 0}  // digit 9
25  };
26  int t = 0;
27  int digid = 0;
28  void loop() {
29    t += 1;
30    if (t > 9999) t = 0;
31    if ((t % 1000) == 0) {
32      digid = t / 1000; // Every second we display numbers in a row
33    }
34    for (int i = 0; i < 4; i++) { // each digit in turn
35    for (int k = 0; k < 8; k++) { // each segment in turn
36        digitalWrite(segmentsPins[k], ((seg[digid][k] == 1) ? LOW : HIGH));
37      }
38      digitalWrite(anodPins[i], HIGH);
39      delay(1);
40      digitalWrite(anodPins[i], LOW);
41    }
42  }
```

The connection diagram of four single-digit seven-segment indicators and the result of the program when displaying the number 2 is shown in Fig. 2.13.

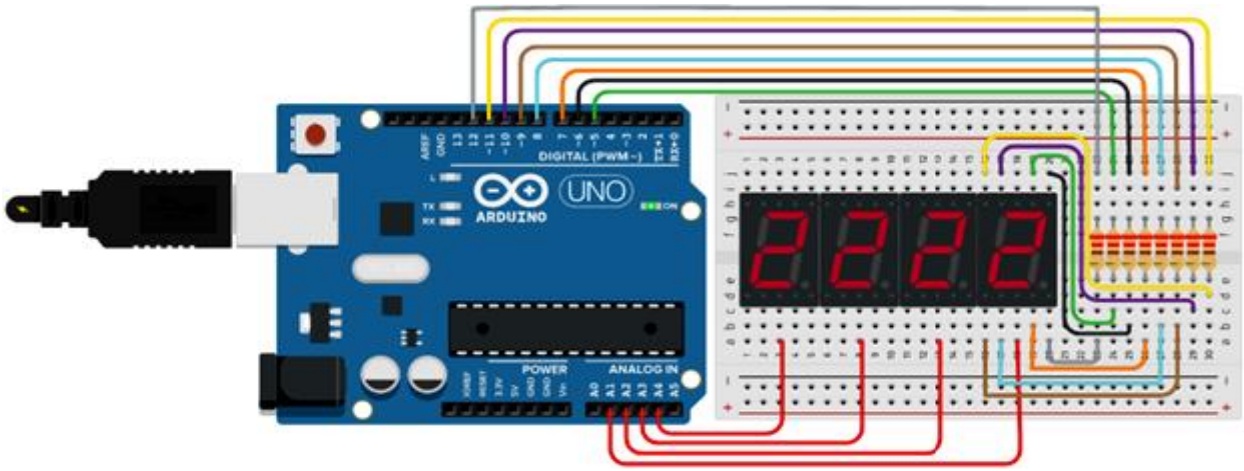


Figure 2.13 – Connection diagram of four seven-segment indicators for Arduino UNO

If you need to output different characters to different digits of the indicator, modify the program using the macro definition sizeofArray(a), which performs the operation sizeof array / sizeof array[0], which determines the number of elements in the array (sizeof(array) – returns the number of bytes occupied by the array; since each element can occupy more than 1 byte of space, you need to divide the result by the size of one element (sizeof (array [0])).

Program code for displaying different characters on different digit of the indicator:

```
#define sizeofArray(a) (sizeof a / sizeof a[0])

const int anodPins[] = {A1, A2, A3, A4}; // Set the pins for each digit
const int segmentPins[] = {5, 6, 7, 8, 9, 10, 11, 12}; // Set the pins for each segment
// 1234. – information displayed on a four-digit indicator
const int symbols[sizeofArray(anodPins)][sizeofArray(segmentPins)] =
{//a, b, c, d, e, f, g, dp – Indicator segments
  {1, 0, 0, 1, 1, 0, 0, 0}, // 1 digit: 4.
  {0, 0, 0, 0, 1, 1, 0, 1}, // 2 digit: 3
  {0, 0, 1, 0, 0, 1, 0, 1}, // 3 digit: 2
  {1, 0, 0, 1, 1, 1, 1, 1}  // 4 digit: 1
};
void setup()
{
 // Define the pins for each digit as outgoing
  for (int i = 0; i < sizeofArray(anodPins); i++)
    pinMode(anodPins[i], OUTPUT);
 // Define the pins for each segment as outgoing Pins
```

```
18    for (int i = 0; i < sizeofArray(segmentPins); i++)
19      pinMode(segmentPins[i], OUTPUT);
20  }
21  void setDisplay(const int symbol[])
22    // Set Pin values for each segment according to a defined string in the Symbol
array
23  {
24    for (int i = 0; i < sizeofArray(segmentPins); i++)
25      digitalWrite(segmentPins[i], symbol[i]);
26  }
27  void loop()
28  {
29    for (int i = 0; i < sizeofArray(anodPins); i++)
30    {
31      // Set the values on the pins for each segment according to the next line of the
symbol array
32      setDisplay(symbols[i]);
33      // Inclusion of each digit in turn
34      digitalWrite(anodPins[i], HIGH);
35      // Determination of the time of switching on each digit in turn
36      delay(5);
37      // Exclusion of each digit in turn
38      digitalWrite(anodPins[i], LOW);
39    }
40  }
```

The scheme and result of the program are shown in Fig.2.14.

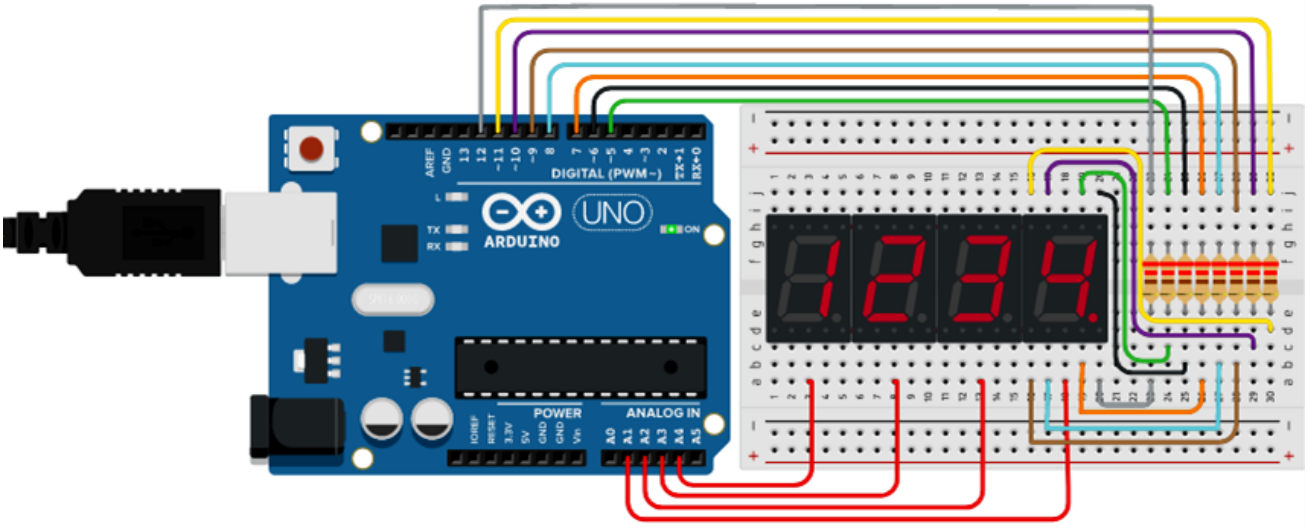


Figure 2.14 – Connection diagram of four seven-segment indicators to Arduino UNO and the result of the program

4. Individual task

4.1. Provide alternating output of the specified symbols on the seven-segment LED indicator. Variants of tasks are given in Table. 2.2.

4.2. Ensure the output of the specified symbols on a four-digit seven-segment LED indicator. Variants of tasks are given in Table. 2.2.

Table 2.2 – Task Options

Variant No.	Information to clause 4.1	Information to clause 4.2
1	0124Г0	1, 5, 2, F
2	1Г1904	2, 4, C, .F
3	РН1234	3, 6, F, A
4	0ГРС01	3, 4, 5, 6
5	НПГО14	B, C, D, 3
6	ГПН813	D, F, F, 1
7	ОНП145	C, F, 8, 9
8	ГРУ128	A, B, F, 6
9	ПОГ061	9, 7, B, A

Order of work

1. Familiarize yourself with the process of alternately displaying the specified symbols on a seven-segment LED indicator using the ATMega328 microprocessor.

2. Familiarize yourself with the process of displaying the specified symbols on a four-digit seven-segment LED indicator using the ATMega328 microprocessor.

3. To develop programs for alternating output of specified symbols on a seven-segment LED indicator and on a four-digit indicator according to an individual task.

4. Develop schematic models according to an individual task in the Tinkercad environment and check its operation.

5. Develop models of schemes according to an individual task in the PROTEUS environment and check its operation.

6. Assemble circuits according to an individual task, enter programs into Arduino UNO R3, run and check their operation.

7. Check the correct functioning of the program in simulation environments and on a real diagram.

8. Draw up a work report.

Contents of the report

1. The topic of the laboratory work.

2. Purpose of the work.

3. Individual task.

4. The program and a simplified block diagram of the algorithm for organizing the process of alternately displaying the specified symbols on a seven-segment LED indicator using the ATMega328 microprocessor according to an individual task.

5. The program and a simplified block diagram of the algorithm for organizing the process of displaying the specified symbols on a four-digit seven-segment LED indicator using the ATMega328 microprocessor according to an individual task.

6. Working models of schemes according to an individual task in the Tinkercad environment.

7. Working models of schemes according to an individual task in the PROTEUS environment.

8. Photos of working circuits assembled on the breadboard according to the individual task.

9. Conclusions.

Practical work 3

RESEARCH OF THE ORGANIZATION OF INFORMATION OUTPUT ON LCD INDICATORS ON THE ATMEGA328 MICROPROCESSOR

Purpose of the work: Study of the organization of the process of displaying information on LCD indicators on the ATMega328 microprocessor on the Arduino platform. Obtaining practical skills in displaying information on LCD indicators using the ATMega328 microprocessor on the Arduino UNO R3 board.

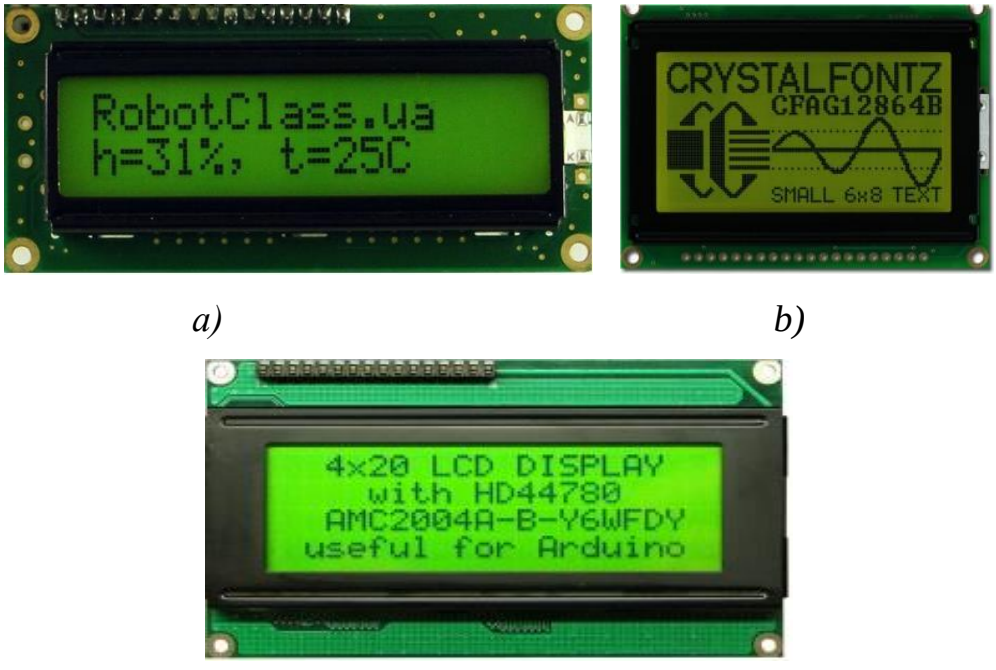
Topics for pre-study:

– theoretical information about LCD indicators;

– theoretical information about dynamic indication.

1. Acquaintance with the basic principles of connecting LCD indicators to a microprocessor

Electronic devices designed to visually display digital, alphanumeric or graphic information are called displays. Let's consider working with liquid crystal (LCD) displays (Fig. 3.1).



a) *b)*

c)

Figure 3.1 – Different types of liquid crystal displays

Liquid crystal indicators that display symbolic information, such as text and numbers, are the most inexpensive and easy to use of all LCDs and are called indicators. They come in a variety of sizes, measured by the number and length of the lines displayed. Some are backlit and allow you to choose the color of the characters and background. Any LCD indicators with HD44780 from Hitachi, KS0066 from Samsung, or compatible with them and a 5V backlight supply voltage work with the Arduino board. Such indicators have several modifications, which can also be with different backlighting (blue, green).

Text (line) displays differ from graphic displays only in the logical control of the dot matrix. In them, the points are pre-grouped into certain places for the display of signs, between which gaps are left. Such places form one, two, or four lines of 8, 12, 16, 20 or more characters each. Each place contains a separately controllable matrix (e.g., 5x8 points) that is designed to output a single character. The configuration of such a display is encoded in its name, where the numbers 1202, 1602, 1204 or similar indicate the number of lines (in examples 2 or 4) and characters in each line (12 and 16). There are also single-line displays of this type, but we will explore the more common two-line ones, namely the backlit LCD 1602, capable of displaying 2 lines of 16 characters each, shown in Fig. 3.2.

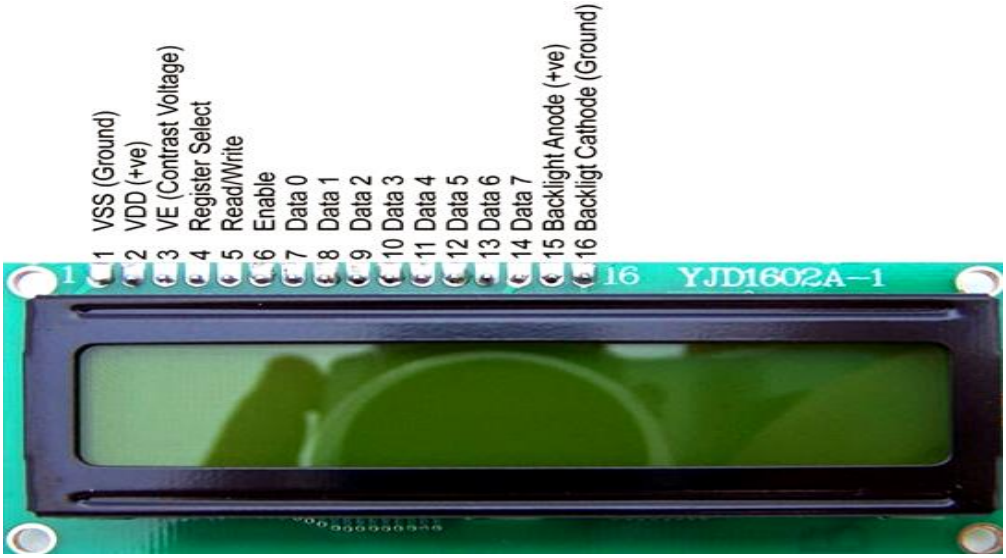


Figure 3.2 – Two-line LCD 1602 LCD

Each of the indicator pins has its own purpose:

1 – VSS (GND), "ground" ("minus power");

2 – VDD (Vcc), power supply +5 V;

3 – VE (V0), setting the contrast of the monitor (indicator);

4 – RS, register selection (commands, data);

5 – R/W, choice of data transmission direction (write or read data);

6 – Enable (EN, E), синхронізація;

7-14 – DB0–DB7, data bus;

15 – LED+, plus backlight (+5 V);

16 – LED–, minus backlight ("ground", "minus power").

The selected indicator has a display type with the ability to load characters, LED backlight, and is controlled by the HD44780 controller. The connection diagram of the LCD 1602 to the Arduino UNO is shown in Fig. 3.3.

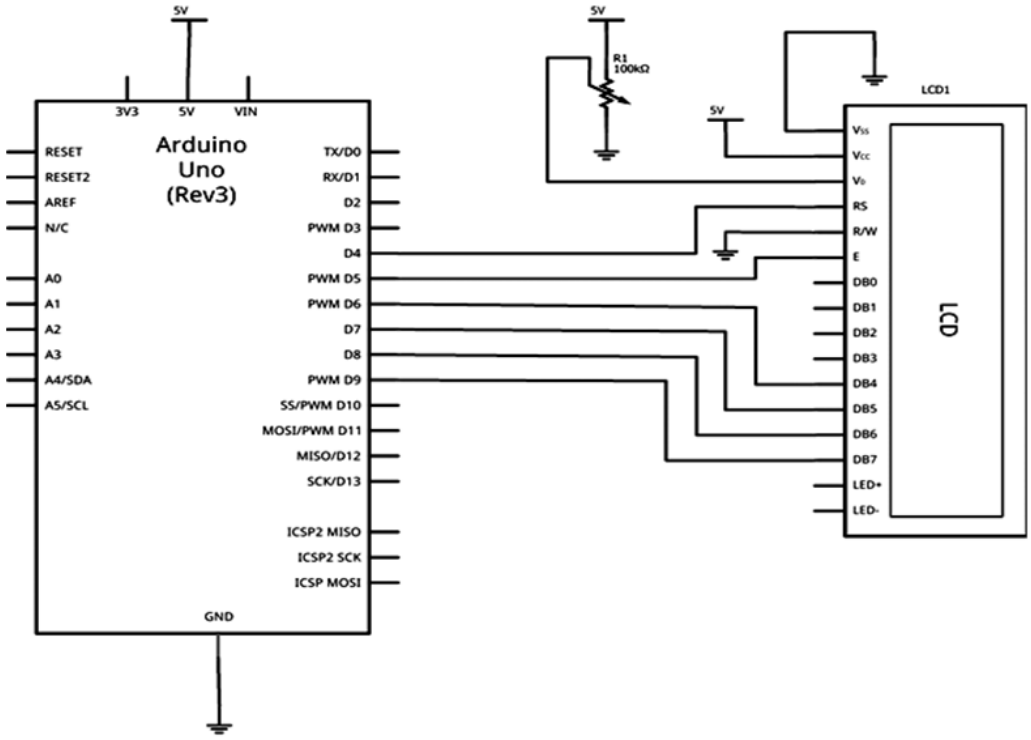


Figure 3.3 – Connection diagram of LCD 1602 to Arduino UNO

The VE (E), RS lines are connected to the Arduino UNO digital pins 5, 4, and the four DB4, DB5, DB6, DB7 data lines are connected to the controller's digital pins 6, 7, 8, 9. The "R/W" line is connected to the "ground" of the controller (because we only need the function of writing to the display memory).

Pinouts of the connections between the LCD 1602 and the Arduino UNO are shown

in Fig. 3.4, the connection diagram is shown in Fig. 3.5.

LCD 1602	1	2	4	6	11	12	13	14	15	16
Arduino UNO	GND	+5 V	4	5	6	7	8	9	+5 V	GND

Figure 3.4 – Pinout of connections between LCD 1602 and Arduino UNO

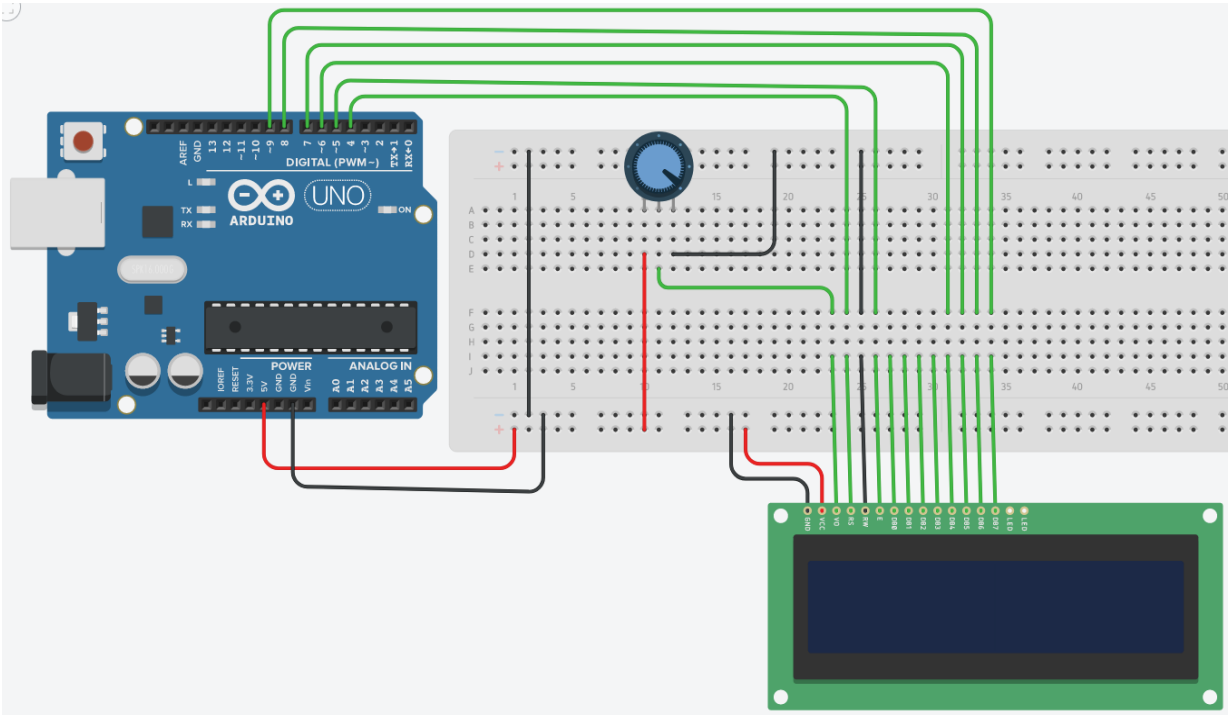


Figure 3.5 – LCD 1602 connection diagram to Arduino UNO

The LiquidCrystal.h library is used to display characters on the screen. After connecting the assembled circuit to a personal computer, it is necessary to adjust the contrast of the display. To do this, follow these steps:

– install the potentiometer on the breadboard and connect its three pins (Fig. 3.6);

– connect the first pin on the potentiometer to the "ground" ("common pin") on the breadboard;

– connect the middle pin of the potentiometer to the 3 pin on the display (it is marked as V0);

– connect the third pin on the potentiometer to the "plus" (+5 V) on the breadboard.

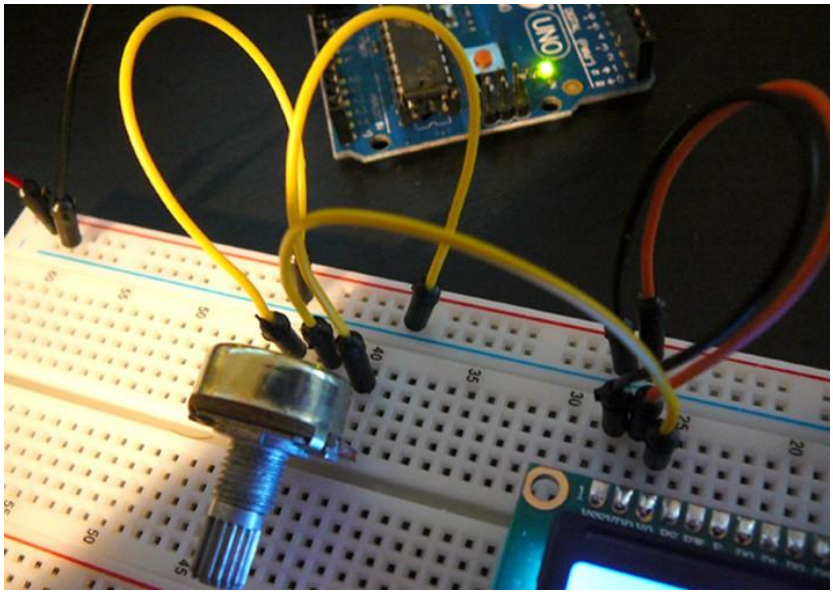


Figure 3.6 – Connecting the potentiometer R1 to adjust the contrast

After applying power to the board via a USB cable on the display, the first line should be filled with rectangles. If they are not visible, you need to turn the potentiometer knob slightly from left to right to adjust the contrast. A properly adjusted display looks something like the one shown in Fig. 3.7.

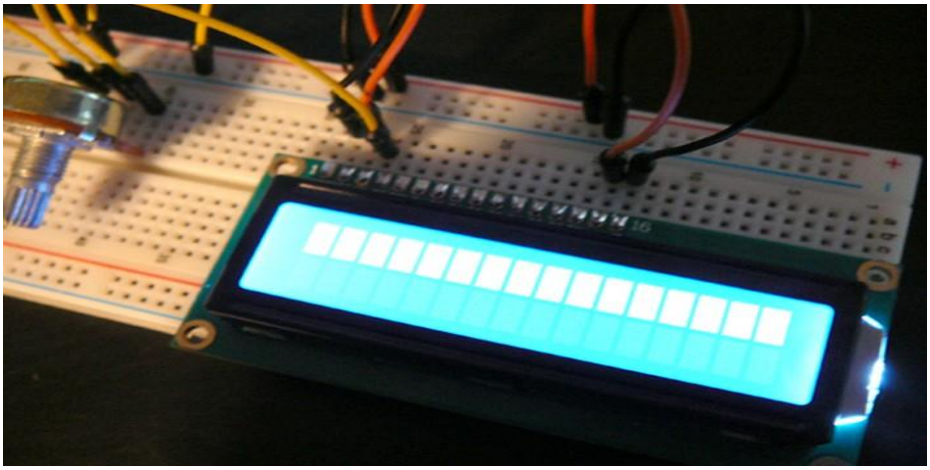


Figure 3.7 – Exterior of LCD 1602 with contrast adjustment

2. Acquaintance with the basic principles of connecting LCD indicators to microprocessors via the I2C interface

The disadvantage of the above diagram of connecting the liquid crystal indicator to the microcontroller (Fig. 3.5) is the need to use at least 6 terminals of the controller. If it is necessary to track data from several sensors and transmit control signals to several

executing devices, such a scheme imposes serious restrictions on the use of Arduino UNO or NANO boards in real projects.

But a liquid crystal monitor with support for the I2C interface (IIS, I2C, I^{2}C) is connected to the Arduino board using only four wires – two for data, two for power.

I²C (IIC, Inter-Integrated Circuit) – A serial asymmetric bus for communication between integrated circuits inside electronic devices. It uses two bidirectional communication lines (SDA and SCL) and is used to connect low-speed peripheral components to processors and microcontrollers (e.g. on motherboards, embedded systems, mobile phones). Developed by Philips in the early 1980s as a simple intercom bus to create control electronics.

I²C uses two bi-directional lines pulled up to the supply voltage and controlled via an open collector or open drain – a serial data line (SDA, Serial Data) and Serial Clock Line (SCL, Serial Clock). Standard voltages are +5 V or +3.3 V, but others are allowed.

For licensing reasons, this interface may also be referred to as a Two Wire Interface (TWI) or Two Wire Serial Interface (TWSI).

Classic addressing includes a 7-bit address space with 16 reserved addresses. This means that there are up to 112 free addresses for connecting peripherals per bus. The main mode of operation is 100 kbit/s; 10 kbit/s in low-speed mode. Note that the standard allows clock termination to work with slow devices.

After the revision of the 1992 standard, it became possible to connect even more devices on a single bus (due to the possibility of 10-bit addressing), as well as speeds of up to 400 kbit/s. The maximum allowable number of chips attached to a single bus is limited to a maximum bus capacity of 400 pF.

The simplest I2C circuit can contain one wired device (most often an Arduino microcontroller (ATMega 326) and several slaves (for example, an LCD display, various sensors and actuating devices). Each device has an address ranging from 7 to 127. There should not be two devices with the same address in the same scheme. Examples of microprocessor systems connected using the I2C interface are shown in Fig. 3.8.

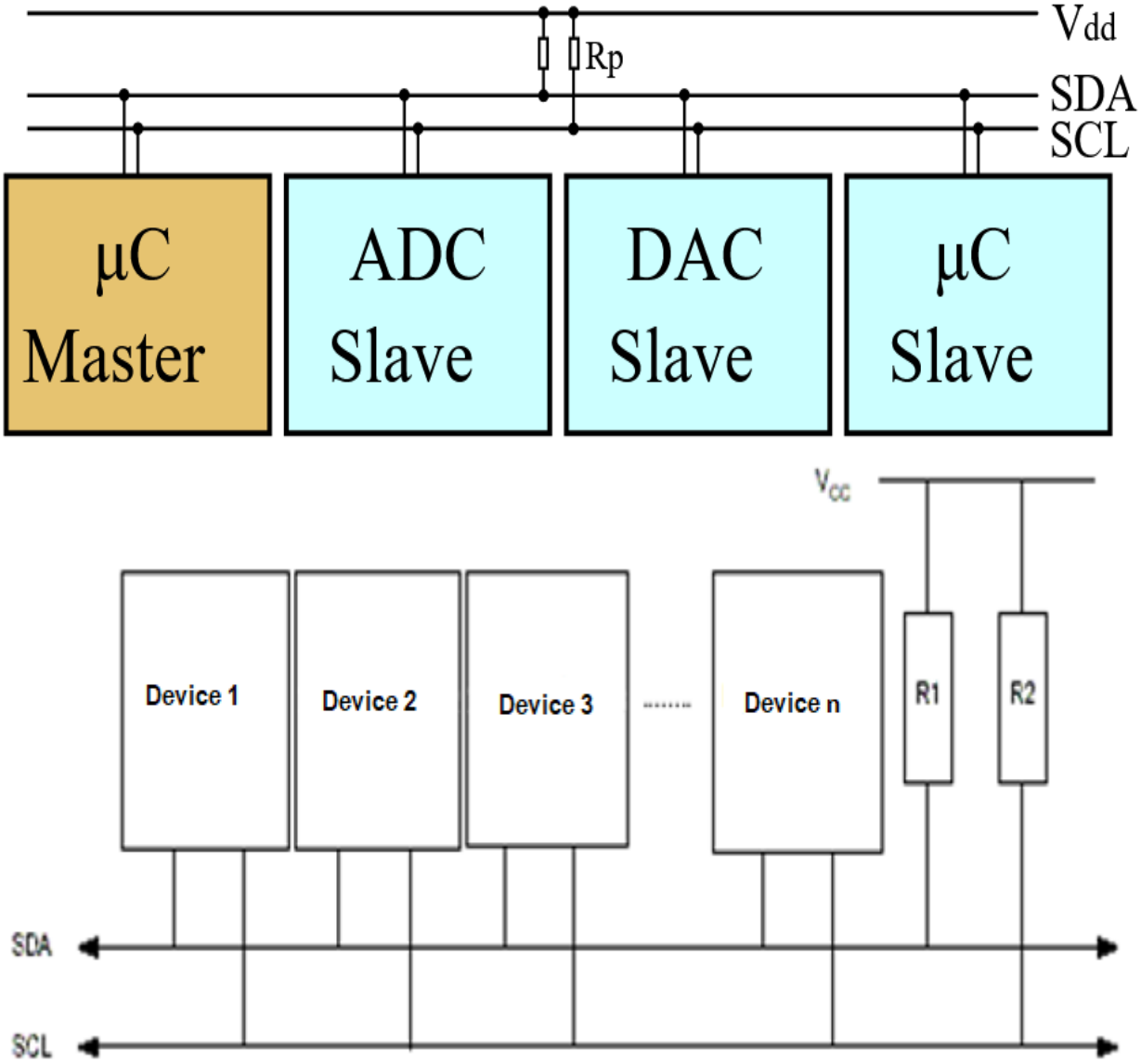


Figure 3.8 – Example of a microprocessor system with a single microcontroller (µC Master) and three slave devices: Device 1 – ADC (Analog-to-Digital Converter); Device 2 – DAC (Digital-to-Analog Converter); device 3 – the second µC Slave microcontroller, and pull-up resistors Rp

The Arduino board supports I^2C at the hardware level. Only two lines are required for operation – SDA (data line) and SCL (synchronization line). The connection diagrams of the LCD 1602 to the Arduino UNO board are shown in Fig. 3.9. To display symbols on the screen, the Wire.h (available in the standard Arduino IDE) and LiquidCrystal_I2C.h libraries are used.

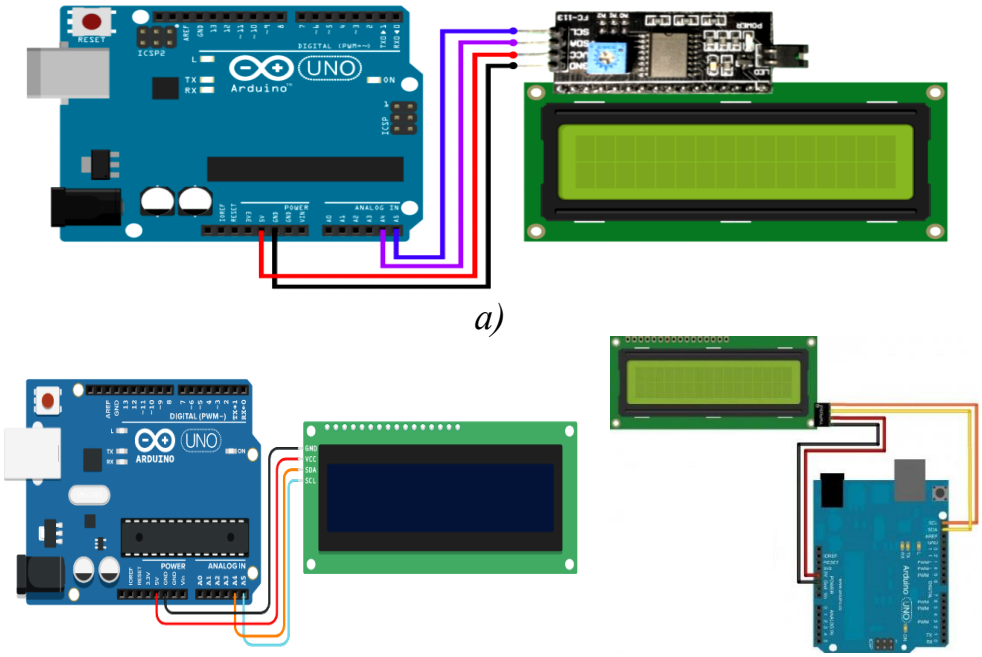


Figure 3.9 – Diagram of connection to Arduino UNO of LCD 1602 indicator with a separate I2C matching device (*a*) and a built-in I2C interface (*b*)

3. Introduction to the main operators of the LiquidCrystal Library

The LiquidCrystal library contains many examples that allow you to understand how it works. Let's take a quick look at the features of this library.

begin() function

Function syntax:

lcd.begin(cols, rows)

The function determines the dimension of the indicator (the number of characters in width and height). Function Options:

lcd – a variable of type LiquidCrystal;

cols – the number of characters per line;

rows – number of rows.

clear() function

Function syntax:

lcd.clear()

The function clears the LCD screen and has a cursor in the upper left corner.

Parameter:

lcd – is a variable of type LiquidCrystal.

home() function

Function syntax:

lcd.home()

The function has a cursor in the upper left corner of the LCD indicator. It is used to determine the starting position when sequential text is displayed on the indicator screen. To also clear the indicator screen, use the clear() function instead of this function.

Parameter:

lcd – is a variable of type LiquidCrystal.

setCursor() function

Function syntax:

lcd.setCursor(col, row)

The function positions the cursor of the liquid crystal indicator, i.e. sets the position in which the text will be displayed on its screen.

Parameters:

lcd – variable of type LiquidCrystal;

col – is the number of the place in the line (starting from 0 for the first place);

row – is the line number (starting from 0 for the first line).

write() function

Function syntax:

lcd.write(data)

The function writes the symbol to the liquid crystal indicator.

Parameters:

lcd – a variable of type LiquidCrystal;

data – a symbol that is written to the indicator.

print() function

The syntax of the print() function is:

led.print(data)

led.print(data, BASE)

The function prints text on the liquid crystal indicator. Options:

lcd – a variable of type LiquidCrystal;

data – data to be printed (type char, byte, int, long or string);

BASE (optional) is a number system in which numbers are printed: BIN for binary (number system 2), DEC for decimals (number system 10), OST for octal (number system 8), HEX for hexadecimal (number system 16).

cursor() function

Function syntax:

lcd.cursor()

The function displays a cursor on the screen of the liquid crystal indicator – underlining the place in the position in which the symbol will be written.

Parameter:

lcd – a variable of type LiquidCrystal.

noCursor() function

Function syntax:

lcd.noCursor()

The function hides the cursor from the LCD screen.

Parameter:

lcd – a variable of type LiquidCrystal.

blink() function

Function syntax:

lcd.blink()

The function displays a flashing cursor on the LCD screen.

If it is used in combination with the cursor() function, the result will depend on the specific indicator.

Parameter:

lcd – a variable of type LiquidCrystal.

noBlink() function

Function syntax:

lcd.noBlink ()

The function disables the flashing cursor on the LCD screen. Setting:

lcd – a variable of type LiquidCrystal.

display() function

Function syntax:

lcd.display()

The function turns on the liquid crystal indicator after it has been disabled by the noDispiay() function. This feature restores the text (and cursor) that was on the display.

Parameter:

lcd – a variable of type LiquidCrystal.

noDisplay() function

Function syntax:

lcd.noDisplay()

The function turns off the liquid crystal indicator without losing the information displayed on it.

Parameter:

lcd – a variable of type LiquidCrystal.

scrollDisplayLeft() function

Function syntax:

lcd.scrollDisplayLeft()

The function scrolls the information on the indicator screen (text and cursor) to one place on the left.

Parameter:

lcd – a variable of type LiquidCrystal.

scrollDisplayRight() function

Function syntax:

led.scrollDisplayRight()

The function scrolls the information on the indicator screen (text and cursor) to one

place on the right.

Parameter:

led – variable of type LiquidCrystal.

autoscroll() function

Function syntax:

lcd.autoscroll()

The function enables automatic scrolling of the LCD screen and forces the previous symbols to move the previous symbols to one digit each time a symbol is displayed on the indicator screen.

If the current direction of output of symbols from the left edge of the indicator to the right edge (default value) – the indicator screen scrolls to the left, if the current direction of display of symbols from the right edge of the indicator to the left – the indicator screen scrolls to the right. This has the effect of displaying each new symbol in the same place on the screen of the liquid crystal indicator.

Parameter:

led – variable of type LiquidCrystal.

noAutoscroll() function

Function syntax:

lcd.noAutoscroll()

The function disables the automatic scrolling of the LCD screen.

Parameter:

led – variable of type LiquidCrystal.

leftToRight() function

The syntax of the leftToRight() function is:

lcd.leftToRight()

The function sets the direction in which the symbols are displayed on the LCD screen from left to right (default value). This means that the following characters displayed on the indicator screen will go from left to right, but will not affect the previously displayed text.

Parameter:

led – variable of type LiquidCrystal.

rightToLeft() function

Function syntax:

lcd.rightToLeft()

The function sets the direction in which the symbols are displayed on the LCD screen from right to left (the default value is from left to right). This means that the following characters displayed on the indicator screen will go from right to left, but will not affect the previously displayed text.

Parameter:

led – variable of type LiquidCrystal.

createChar() function

Function syntax:

led.createChar(num, data)

The function creates a custom symbol (glyph) for use on the LCD display. Up to eight 5x8 pixel characters are supported (numbering from 0 to 7).

The creation of each character by the user is determined by an array of eight bytes – one byte for each line. The five least significant bits of each byte define the pixels in that string. To display a custom character on the screen, use the write() function with the character number as a parameter.

Parameters:

lcd – a variable of type LiquidCrystal;

num is the number of the character to be created (from 0 to 7);

data – data of character pixels.

4. Introduction to the process of displaying numbers and symbols on an LCD indicator using the ATMega328 microprocessor

Here is a program that displays a standard greeting on LCD 1602 using the ATMega328 microprocessor, connected according to the scheme shown in Fig. 3.5.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // to the pins RS, E, DB4, DB5, DB6, DB7
void setup()
{
 lcd.begin(16, 2);
 lcd.clear();
}
void loop()
{
 lcd.setCursor(5, 0);
 lcd.print("Hello");
 lcd.setCursor(6, 1);
 lcd.print("world!");
 delay(10000);
}
```

The result of the program is shown in Fig. 3.10.

Figure 3.10 – Output program output

standard greeting

Let's take a closer look at the work of this program. First of all, the program includes two initial lines. Their goal is to connect a library for working with liquid crystal indicators (it is automatically installed as part of the Arduino IDE) and tell it which pins on the Arduino board the LCD is connected to. This purpose is served by the next two lines before the void setup() function:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // to the pins RS, E, DB4, DB5, DB6, DB7
```

If you need to use other pins on the Arduino board, change the pin numbers in the second line.

Next, in the void setup() function, the library is told the size of the LCD screen in columns and rows. For example, in the above program, we report that the LCD screen has two lines of 16 characters each:

```
lcd.begin(16, 2);
```

After setting the LCD in the next line, the screen is cleaned:

```
lcd.clear();
```

The cursor is then set to the start position of the text output by calling the function:

```
lcd.setCursor(x, y);
```

Here, *x* is the column number in the row (from 0 to 15) and *y* is the row number (0 or 1). After that, the text is output by calling the lcd.print() function, for example, to print the word "text" you need to run the command:

```
lcd.print ("text")
```

BDisplay of variables and numbers on the LCD indicator

To display the contents of a variable on the LCD indicator, use the function call:

```
lcd.print(variable);
```

When displaying a variable of type float, you must specify the number of decimal places to be displayed. For example, lcd.print (pi, 3) will display the value of pi with three decimal places, as shown in Fig. 3.11:

```
float pi = 3.141592654;
lcd.print("pi: ");
lcd.print(pi, 3);
```

Figure 3.11 – The result of the output program

Pi value with three decimal places

The contents of the integer variable can be displayed on the LCD screen not only in decimal, but also in binary and hexadecimal systems, as shown in Fig. 3.12 and in the program:

```
int kk = 170;
lcd.setCursor(0, 0);
lcd.print("Binary:");
lcd.print(kk, BIN); // print the number 170 in binary
lcd.setCursor(0, 1);
lcd.print("Hexadecimal:");
lcd.print(kk, HEX); // print the number 170 in hexadecimal
```

Figure 3.12 – Result of the program

5. Introduction to the process of determining and displaying eigensymbols on the LCD indicator using the ATMega328 microprocessor

In addition to standard letters, numbers, and other characters from the standard set, you can also define your own characters programmatically. On the LCD 1602 screen, each character is displayed in a matrix containing eight rows of five dots (pixels). In Fig. 3.13 these matrices are shown in close-up.

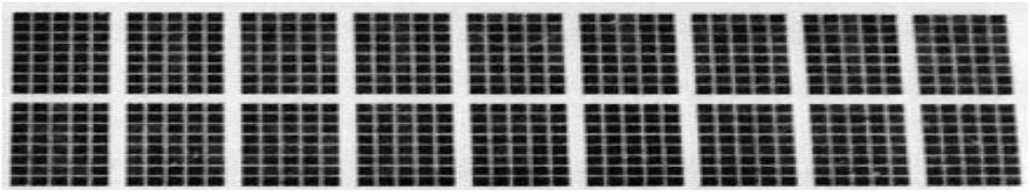


Figure 3.13 – Each symbol of LCD 1602 is composed of
of eight lines of five pixels

To display a custom symbol, it must first be defined as an array. For example, an emoji symbol is defined as follows:

```
byte a[8] =  {B00000,
              B01010,
              B01010,
              B00000,
              B00100,
              B10001,
              B01110,
              B00000};
```

Each digit in this array corresponds to a pixel: 0 is off, 1 is on. Array elements represent rows of pixels on the screen; The first element corresponds to the top line, the next element to the second line, and so on.

When planning to use your own symbols, you first need to draw them on lined paper (each painted square will correspond to a "1" in the array, and each empty square will correspond to a "0").

In the above example, the first element of the array is B00000, so all pixels in the top row will be disabled. In the second line (corresponding to element B01010), every second pixel will be included.

Next, we store the array defining the new symbol in the first of eight slots dedicated to non-standard characters in the void setup() function:

```
lcd.createChar(0, a); // Save array A[8] to slot 0
```

Finally, to display the symbol, add the following call to void loop():

lcd.write(byte (0));

Use the following code to display non-standard characters:

```
LiquidCrystal lcd (4, 5, 6, 7, 8, 9); // to the pins RS, E, DB4, DB5, DB6, DB7
byte a[8] = {B00000,
             B01010,
             B01010,
             B00000,
             B00100,
             B10001,
             B01110,
             B00000};
void setup ()
{
 lcd.begin(16, 2);
 lcd.createChar(0, a);
}
void loop()
{
 lcd.write(byte(0)); // Print a non-standard character from slot 0
// to the next cursor position
}
```

Fig. Figure 3.14 shows two rows of emoticons on the LCD screen.

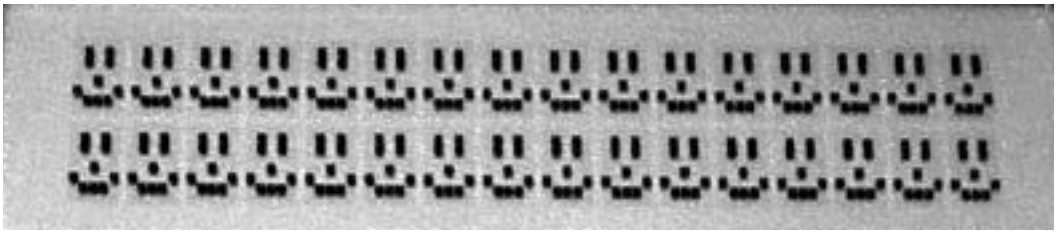


Figure 3.14 – The result of the program

prints a non-standard character

6. Individual tasks

1. Put the name of your own study group on the first line LCD 1602. Put your own surname and initials on the second line LCD 1602.

2. Display the value of *e* (the mathematical constant that is the basis of natural logarithms, e ≈ 2,7182818284) with the number of decimal places that coincides with the last digit of your variant.

3. Display the value of your variant in decimal, binary, and hexadecimal on the indicator screen.

4. Display your name on the indicator screen in handwritten form (using no more than eight non-standard characters).

Order of work

1. Familiarize yourself with the basic principles of connecting LCD indicators to microprocessors.

2. Familiarize yourself with the basic principles of connecting LCD indicators to a microprocessor via the I2C interface.

3. Familiarize yourself with the main operators of the LiquidCrystal library.

4. Familiarize yourself with the process of displaying numbers and symbols on the LCD indicator using the ATMega328 microprocessor.

5. Familiarize yourself with the process of detecting and displaying proper symbols on the LCD indicator using the ATMega328 microprocessor.

6. To develop models of circuits with the connection of LCD indicators to the microprocessor via a parallel interface and via the I^2C interface according to an individual task in the Tinkercad environment and check their operation.

7. To develop models of circuits with the connection of LCD indicators to the microprocessor via a parallel interface and the I2C interface according to an individual task in the PROTEUS environment and check its operation.

8. Assemble circuits according to an individual task, enter programs into Arduino UNO R3, run and check their operation.

9. Check the correct functioning of the program in simulation environments and on a real diagram.

10. Draw up a work report.

Contents of the report

1. The topic of the laboratory work.

2. Purpose of the work.

3. Individual task.

4. The program and a simplified block diagram of the algorithm for organizing the process of displaying the name of your own study group and your own surname and initials, the value of a variable with a given number of decimal places after the decimal point, the value of an integer (your own number of the variant) in the decimal, binary and hexadecimal systems, your own name using the developed non-standard symbols on LCD 1602 using an ATMega328 microprocessor connected via a parallel interface according to an individual task.

5. The program and a simplified block diagram of the algorithm for organizing the process of displaying the name of one's own study group and one's own surname and initials, the value of a variable with a given number of decimal places after the decimal point, the value of an integer (own number of the variant) in the decimal, binary and hexadecimal systems, a proper name using the developed non-standard symbols on LCD 1602 using an ATMega328 microprocessor connected via a serial interface I2C according to an individual task.

6. Working models of circuits with the connection of an LCD indicator to a microprocessor via a parallel interface and via an I2C interface according to an individual task in the Tinkercad environment.

7. Working models of circuits with the connection of LCD indicators to the microprocessor via a parallel interface and via the I2C interface according to an individual task in the PROTEUS environment.

8. Photos of working assembled circuits on the circuit board according to the individual task.

9. Conclusions.

Practical work 4

RESEARCH OF THE ORGANIZATION OF TIME CONTROL FUNCTIONS ON THE ATMEGA328 MICROPROCESSOR

Purpose of the work: Gaining practical skills in reading control signals from buttons, eliminating contact rattle and organizing several simultaneous timing control functions using the ATMega328 microprocessor on the Arduino UNO R3 board.

Topics for pre-study:

– theoretical information about time functions in Arduino;

– theoretical information about programming in Arduino.

1. Introduction to connecting the button to the ATMega328 microprocessor

The button (push-button switch) is the simplest and most accessible of all types of sensors (Fig. 4.1). By clicking on it, a signal is sent to the controller, which leads to the following actions: LEDs turn on, sounds are played, motors start.

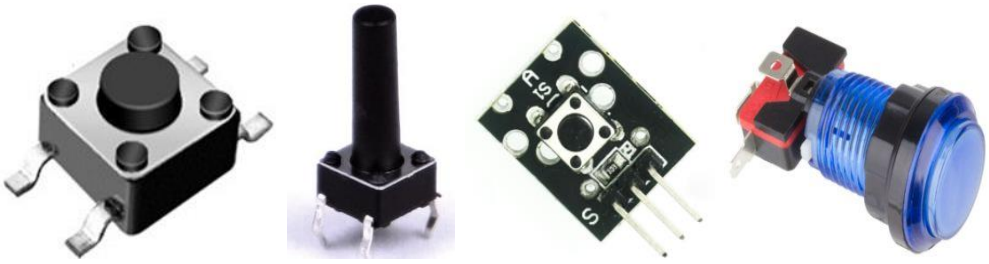
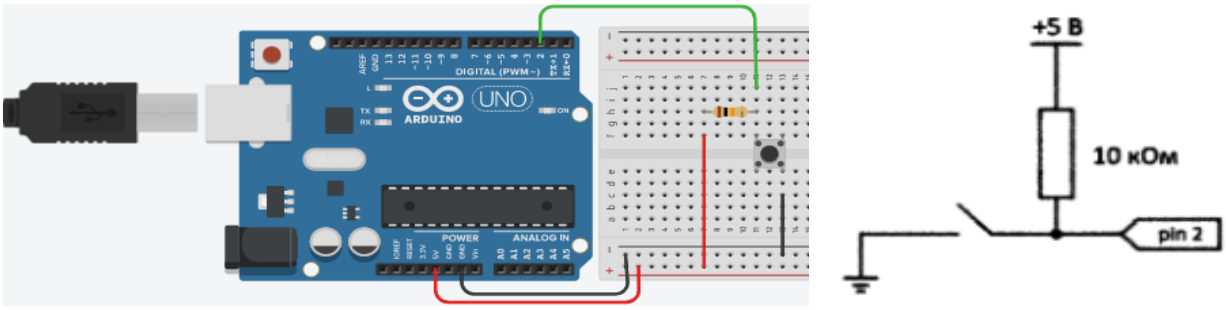


Figure 4.1 – Buttons (push-button switches) of different types

A button is a switch with two pairs of pins. The contacts in one pair are connected to each other, so it will not be possible to implement more than one switch in the circuit, but it is possible to control two parallel segments at the same time.

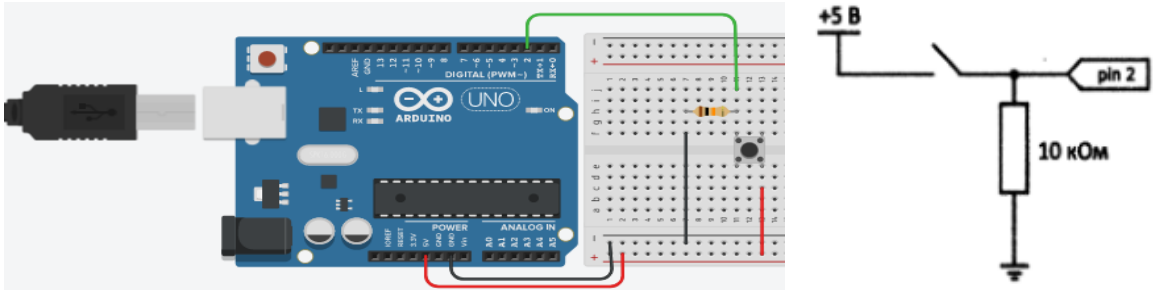
To connect a normally open button to the Arduino, it is possible to go the simplest way: connect one free conductor of the button to the power or "ground", and the other to the digital output (pin) of the Arduino. But this is wrong, because when the button is not closed, electromagnetic interference may appear on the digital output of the Arduino and because of this, false accidental alarms are possible.

To avoid interference, the digital output is usually connected via a sufficiently large resistor (5-10 kΩ) either to the power supply or to the "ground". In the first case, it is called a circuit with a pull-up resistor (Fig. 4.2), in the second – a circuit with a pull-down resistor (Fig. 4.3). Resistors in both circuits are used to set the "default value" on the input pin: in a circuit with a pulling resistor it is "1" (HIGH), in a circuit with a tightening resistor it is "0" (LOW)).



a) *b)*

Figure 4.2 – Wiring and schematic diagrams of the button connection to Arduino (circuit with pull-up resistor)



a) *b)*

Figure 4.3 – Wiring and schematic diagrams of the button connection to Arduino (circuit with a tightening resistor)

All pins inside the Arduino board are connected to the 5 V power bus through resistors of the order of 20-50 kΩ. These resistors can be programmatically connected to or disconnected from the pins. The software inclusion of resistors is carried out as follows:

pinMode (2, INPUT_PULLUP); // 20 kΩ internal pull-up resistor connected

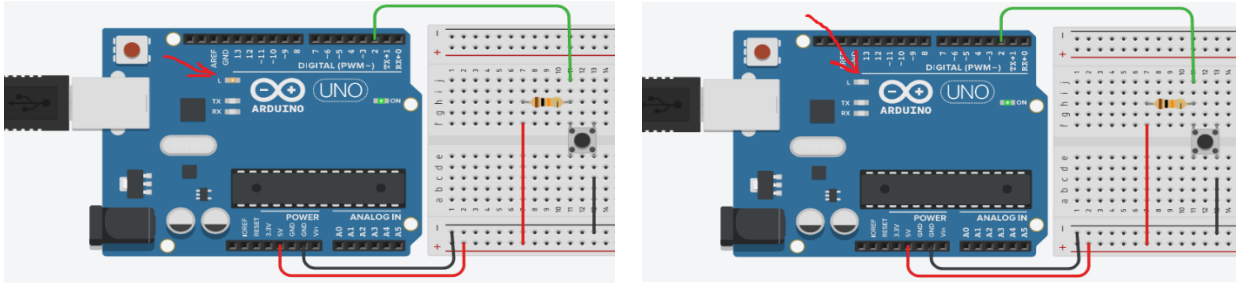
Let's write a program that sets the state of the LED connected to pin 13 (located on the Arduino board), depending on the state of the button (the button is pressed – the LED is

"on", the button is not pressed – the LED is "off").

```
const int LED = 13; // Pin 13 to connect the LED
const int BUTTON = 2; // Pin 2 to connect the button
boolean buttonState; // buttonState button status variable
void setup () {
// define the LED output as the output
pinMode (LED, OUTPUT);
// define the BUTTON pin as the input
pinMode (BUTTON, INPUT_PULLUP);
}
void loop () {
// read the BUTTON state of the input (button) and write it to buttonState
buttonState = digitalRead (BUTTON);
// Inversion of the buttonState variable for a circuit with a pull-up resistor
buttonState = ! buttonState;
// write the state from buttonState to the LED output (LED)
digitalWrite (LED, buttonState);
}
```

For a circuit with a pulling resistor, the state of the variable buttonState is inverted, because in this case, when the button is pressed, the signal state is low, and the LED lights up when high. For circuits with a tightening resistor, the state of the buttonState variable does not need to be inverted.

The result of the above program is the glow of the LED built into the Arduino UNO (connected to pin 13) when the button is pressed (contacts are closed) with a pull-up resistor (connected to +5 V), which is connected to pin 2 (Fig. 4.4, *a*). When the button is not pressed (the contacts are open), the built-in LED does not light up (Fig. 4.4, *b*). In both cases, the built-in LED is marked with a red arrow.



a) *b)*

Figure 4.4 – The result of the above program when the button (a) is pressed and when the button (b) is not pressed

2. Introduction to ways to eliminate button contact rattle

Let's consider the following task: we will switch the state of the LED (on/off) every time the button is pressed.

To do this, load the following program into the Arduino:

```
const int LED = 13; // Pin 13 to connect the LED
const int BUTTON = 2; // Pin 2 to connect the button
boolean buttonState; // buttonState button status variable
boolean buttonStatePrev = LOW; // Button status variable previous
boolean ledState = LOW; // LED status variable
void setup () {
// Starting a Serial Port
Serial.begin (9600);
// define the LED output as the output
pinMode (LED, OUTPUT);
// define the BUTTON pin as the input through the pull-up resistor
pinMode (BUTTON, INPUT_PULLUP);
// Initial state of the LED
digitalWrite (LED, ledState);
}
void loop () {
// read the status of the input BUTTON
buttonState = digitalRead (BUTTON);
// if the button is pressed (changing the status from LOW to HIGH)
if (buttonState == HIGH && buttonStatePrev = LOW) {
ledState = ! ledState;
// write the state from ledState to the output of the LED
digitalWrite (LED, ledState);
// Output the ledState status value to the serial port for tracking
// its value on a computer using a serial port monitor
Serial.println (ledState);
}
buttonStatePrev = buttonState;
}
```

Press the button and see that in the serial port, when the button is pressed once, there are several changes in its state (Fig. 4.5), and, accordingly, several LED switches.

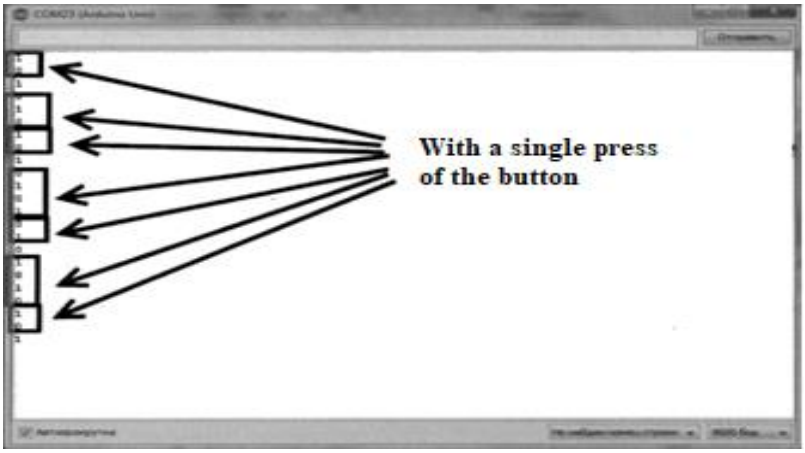
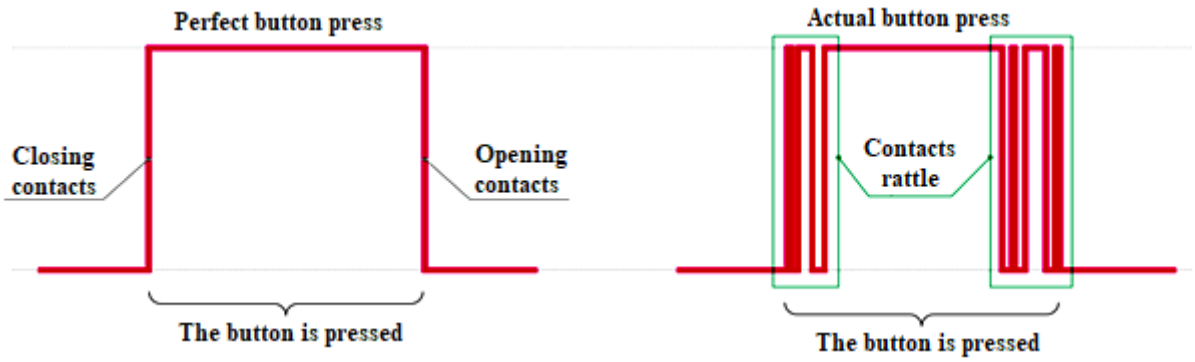


Figure 4.5 – Debugging data output to the serial port monitor

This is due to the fact that the buttons are mechanical devices with a spring contact system, subject to a phenomenon called rattle – in the process of pressing the button, the contact, especially at the beginning of pressing, closes and opens several times. That is, when you press a button, the signal doesn't just change from low to high – it changes value several times over the course of a few milliseconds before it is set to LOW. The graphs shown in Fig. Figure 4.6 illustrates the difference between the expected phenomenon and the real one.

The button is pressed for about tens to hundreds of milliseconds. It is advisable to immediately know the status of the button by reading the values from the contact input, as shown in Fig. 4.6 (a). However, the button actually moves up and down until the value is set as shown in Fig. 4.6 (b). Transients proceed very quickly, but when processing the signal from a button on an Arduino, we may encounter transient effects and must take them into account.



a) *b)*

Figure 4.6 – Illustration of the phenomenon of button contact rattle

The algorithm of the program, which eliminates the negative consequences of the phenomenon of rattling contacts, can be as follows:

1) save the previous state of the button and the current state of the button (when initializing – LOW);

2) read the current state of the button;

3) if the current state of the button is different from its previous state, wait for a certain period of time (for example, 5ms), because the button may have changed its state;

4) after a certain period of time (for example, 5 ms), read the state of the button and use it as the current one;

5) if the previous state of the button was LOW, and the current state was HIGH, switch the state of the LED;

6) set the previous state of the button for its current state;

7) return to step 2 of the given algorithm (if necessary, read the current state of the button contacts again).

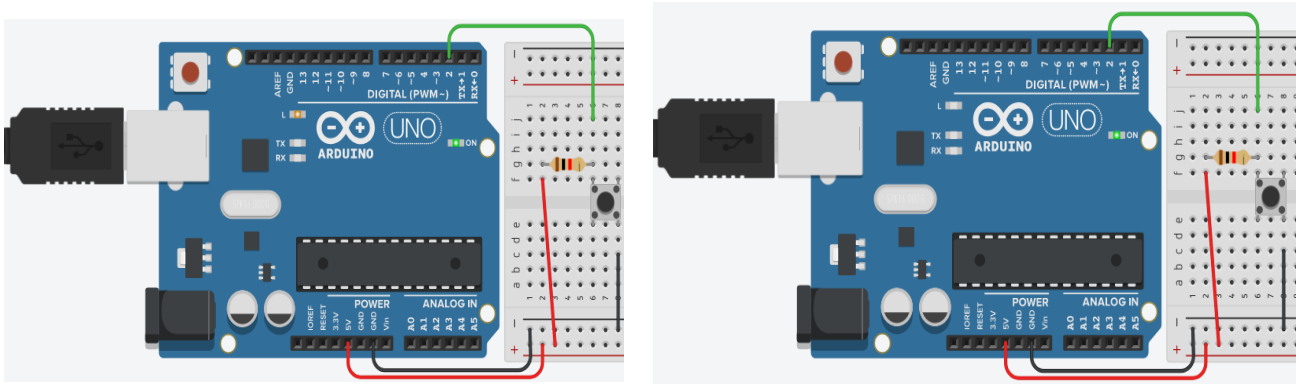
Software implementation of the algorithm:

```
const int LED = 13; // Pin 13 to connect the LED
const int BUTTON = 2; // Pin 2 to connect the button
boolean lastButton = LOW; // variable to save the previous state of the button
boolean currentButton = LOW; // variable to save the current state of the button
boolean ledOn = false; // Current LED Status (On/Off)
void setup () {
Serial.begin (9600); // Starting a Serial Port
pinMode (LED, OUTPUT); // Configure LED pin as output
pinMode (BUTTON, INPUT); // Configure button pin as input
}
void loop () {
currentButton = debounce (lastButton);
if (lastButton == LOW && currentButton == HIGH)
// якщо кнопку натиснули
{
ledOn = !ledOn; // Invert LED Status Value and Output Value
// ledOn state to the serial port to track its value
// on your computer using a serial port monitor
Serial.println (ledOn);
```

```
}
lastButton = currentButton;
digitalWrite (LED, ledOn); // Change LED Status Status
}
// The anti-aliasing function of the rattling effect of the contacts takes as
// argument previous state of the button, and returns its actual state
boolean debounce (boolean last) {
boolean current = digitalRead (BUTTON); // Read the status of the button
if (last != current) // If it have changed
{
delay (5);                          // Waiting for 5ms
current = digitalRead (BUTTON); // Reading the status of the button
return current;                     // Return the button state
}
}
```

When using the above program, a single press of the button leads to a one-time change in the state of the LED, an illustration of the operation of the program is shown in Fig. 4.7. When using the above program, a single press of the button leads to a one-time change in the state of the LED. Also, to combat the effect of rattling button contacts, it is possible to use the Bounce library.

Eliminating contact rattle using delays in the program is a very common method and does not require changing the circuit itself. But it is not always possible to use it – the software method cannot be used when using interrupts in the program, because the rattle will lead to multiple calls to functions and we will not be able to influence this process in the program.



a) *b)*

Figure 4.7 – The result of the above program after an odd press of the button (a) and after an even press of the button (b)

Hardware solutions are also used to eliminate the effect of rattling contacts, smoothing out the impulses coming from the button.

The hardware method of removing rattle is based on the use of smoothing filters. A smoothing filter is engaged in smoothing out signal bursts by adding elements to the circuit that have a kind of "inertia" in relation to electrical parameters such as current or voltage. The most common example of such "inertial" electronic components is the capacitor. The diagram of connecting the filter using a capacitor to eliminate rattle is shown in Fig. 4.8.

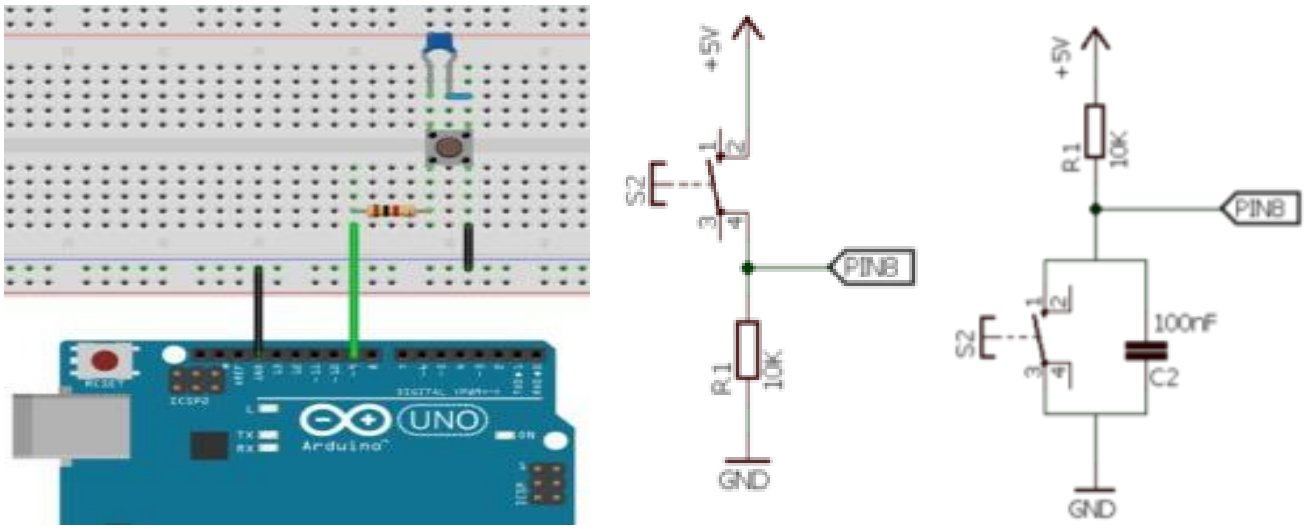


Figure 4.8 – Filter connection diagram using condenser to eliminate rattle

To obtain a rectangular waveform after the smoothing chain, a Schmitt trigger is used.

3. Introduction to the basic functions of time in Arduino

In Arduino, there are several different commands that are responsible for working with time and creating pauses (they differ in accuracy and have their own characteristics that should be taken into account when writing code):

– delay ();

– delayMicroseconds ();

– millis ();

– micros ().

The delay() function is essentially a delay that suspends the program for a number of milliseconds specified in parentheses. The maximum value can be 4294967295 ms, which is approximately equal to 50 days. The disadvantage of using this function is that it is impossible to use it when it is necessary to simultaneously process signals from several sensors and synchronously issue control signals to several executing devices, because at the time of its use, the execution of all these processes is suspended. During delay() execution, it is not possible to poll inputs, process data, or change the state of outputs. This feature utilizes the CPU to 100%. You can work around this restriction by using interrupts.

The delayMicroseconds() function is a complete analogue of delay() except that its units of measurement are microseconds instead of milliseconds. The maximum value of the function is 16383, which is equal to 16 ms.

The function micros() is a complete analogue of millis() except that its units of measurement are microseconds instead of milliseconds.

The millis() function returns the number of milliseconds that have elapsed since the Arduino program started. This number has an unsigned long format, which is used to store positive integers ranging from 0 to 4294967295 ($2^{32} – 1$) and takes up 32 bits (4 bytes) in memory.

If you try to perform mathematical operations between the value of the unsigned long format and values of another type (for example, int), an error will be generated. It is with the help of this function that we will organize the polling of sensors (buttons) and the issuance of control signals to control devices (LEDs).

4. Introduction to the organization of timing control functions on the ATMega328 microprocessor

Here is an example of replacing the delay() operator with millis(). For example, delay(1000), which provides a delay of 1 second. (1000 ms), can be replaced with the following code fragment:

```
unsigned long timing; // Variable to store the reference point
void setup () {
  Serial.begin (9600);
```

```
}
void loop () {
// At this point, the execution of the delay analogue () begins
// calculate the difference between the current moment and the previously saved point
// countdown, if the difference is greater than the required value, then execute the code,
// if not, we do nothing
  if (millis () – timing> 1000) {
// Instead of 1000, write the desired pause value
   timing = millis ();
   Serial.println ("1 second");
  }
}
```

An example of the organization of LED flashing using the millis() function:

```
const int ledPin = 13; // Pin number with LED connected
int ledState = LOW; // LED Status: On/Off
long previousMillis = 0; // Time when the LED status was updated
long interval = 1000; // Half flashing period (in ms)
void setup () {
  // Set the digital pin with the LED to the output mode
  pinMode (ledPin, OUTPUT);
}
void loop () {
  // Finding out if it's time to change the state of the LED
  unsigned long currentMillis = millis (); // Current time in milliseconds
  if (currentMillis – previousMillis> interval) {
    // save the last moment when the state of the LED changed
    previousMillis = currentMillis;
    // Change the state of the LED to the opposite
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;
    // Set the output voltage level to high or low
    // based on the value of the ledState variable
    digitalWrite (ledPin, ledState);
  }
}
```

An example of organizing LED flashing with different on and off times using the millis() function:

```
// These variables store a time pattern for flashing intervals
```

```
// and the current status of the LED
int ledPin = 13; // Number Pin with LED
int ledState = LOW; // LED Status
// the last point in time when the state of the LED changed
unsigned long previousMillis = 0;
long OnTime = 250; // LED on duration (in milliseconds)
long OffTime = 750; // LED off (in milliseconds)
void setup () {
  // Set the digital pin with the LED as the "output"
  pinMode (ledPin, OUTPUT);
}
void loop () {
  // Finding out if it's time to change the state of the LED
 unsigned long currentMillis = millis (); // Current time in milliseconds
  // if the LED is on and lights up more than it should
  if ((ledState == HIGH) && (currentMillis – previousMillis> = OnTime))
  {
    ledState = LOW; // Turn it off
    previousMillis = currentMillis; // Remembering a point in time
    digitalWrite (ledPin, ledState); // Implementing a new state
  }
  else if ((ledState == LOW) && (currentMillis – previousMillis> = OffTime))
  {
    ledState = HIGH; // Turn it off
    previousMillis = currentMillis; // Remembering a point in time
    digitalWrite (ledPin, ledState); // Implementing a new state
  }
}
```

An example of organizing flashing with two LEDs with different on and off times using the millis() function:

```
// These variables store a time pattern for flashing intervals
// and the current status of the LEDs
int ledPin1 = 12; // Number Pin with LED
int ledState1 = LOW; // LED Status
// the last point in time when the state of the LED changed
unsigned long previousMillis1 = 0;
long OnTime1 = 250; // LED on duration (in milliseconds)
long OffTime1 = 750; // LED off (in milliseconds)
int ledPin2 = 13; // Number Pin with LED
int ledState2 = LOW; // LED Status
// the last point in time when the state of the LED changed
unsigned long previousMillis2 = 0;
```

```
long OnTime2 = 330; // LED on duration (in milliseconds)
long OffTime2 = 400; // LED off (in milliseconds)
void setup () {
// Setting digital pins with LEDs as an "output"
pinMode (ledPin1, OUTPUT);
pinMode (ledPin2, OUTPUT);
}
void loop () {
// Finding out if it's time to change the state of the LED
unsigned long currentMillis = millis (); // Current time in milliseconds
// Control program for the first LED
if ((ledState1 == HIGH) && (currentMillis – previousMillis1> = OnTime1))
  {
   ledState1 = LOW; // Turn it off
   previousMillis1 = currentMillis; // Remembering a point in time
   digitalWrite (ledPin1, ledState1); // Implementing a new state
  }
else if ((ledState1 == LOW) && (currentMillis – previousMillis1> = OffTime1))
  {
   ledState1 = HIGH; // Turn it off
   previousMillis1 = currentMillis; // Remembering a point in time
   digitalWrite (ledPin1, ledState1); // Implementing a new state
  }
 // Control program for the second LED
if ((ledState2 == HIGH) && (currentMillis – previousMillis2> = OnTime2))
  {
   ledState2 = LOW; // Turn it off
   previousMillis2 = currentMillis; // Remembering a point in time
   digitalWrite (ledPin2, ledState2); // Implementing a new state
  }
else if ((ledState2 == LOW) && (currentMillis – previousMillis2> = OffTime2))
  {
   ledState2 = HIGH; // Turn it off
   previousMillis2 = currentMillis; // Remembering a point in time
   digitalWrite (ledPin2, ledState2); // Implementing a new state
  }
}
```

The Arduino programming language is a variation of C++ that supports object-oriented programming. By using the object-oriented properties of the language, we can put together all the state variables and functionality for a flashing LED into a C++ class. It's not difficult to do at all. After all, all the code is already there, we just need to repackage it into a classroom. We start by declaring a Flasher class:

```
class Flasher
{
  // variables – class members initialized at startup
  int ledPin; // Number Pin with LED
  long OnTime; // Switch-on time in milliseconds
  long OffTime; // Time when the LED is off
   // Current status
  int ledState; // On/off status
  unsigned long previousMillis; // Last moment of state change
};
```

Next, add a class constructor. The constructor has the same name as the class and is designed to initialize variables.

```
// The constructor creates an instance of the <em> Flasher </em class>
// and initializes the variables – members of the class and the state
public:
Flasher (int pin, long on, long off)
{
  ledPin = pin;
  pinMode (ledPin, OUTPUT);
   OnTime = on;
  OffTime = off;
  ledState = LOW;
  previousMillis = 0;
}
```

Let's take our loop and turn it into a function – a member of a class called Update(). It is identical to the original void loop () – only the name has been changed.

```
void Update ()
{
  // Finding out if it's time to change the state of the LED
unsigned long currentMillis = millis (); // Current time in milliseconds
if ((ledState == HIGH) && (currentMillis – previousMillis> = OnTime))
{
   ledState = LOW; // Turn it off
   previousMillis = currentMillis; // Remembering a point in time
   digitalWrite (ledPin, ledState); // Implementing a new state
}
else if ((ledState == LOW) && (currentMillis – previousMillis> = OffTime))
{
   ledState = HIGH; // Turn it off
```

```
    previousMillis = currentMillis; // Remembering a point in time
    digitalWrite (ledPin, ledState); // Implementing a new state
  }
  }
```

By recomposing the existing code into a Flasher class, we encapsulated all the variables (state) and got the functionality for flashing with an LED. Now, for each of the LEDs to flash, we create an instance of the Flasher class by calling the constructor and at each step of the loop we call Update() for each instance of the Flasher.

An example of a program that implements independent flashing of three LEDs:

```
class Flasher
{
  // variables – class members are initialized at startup
  int ledPin; // Number Pin with LED
  long OnTime; // Switch-on time in milliseconds
  long OffTime; // Time when the LED is off
  // Current status
  int ledState; // On/off status
  unsigned long previousMillis; // Last moment of state change
 // the constructor creates an instance of Flasher
 // and initializes variables – class members and state
  public:
  Flasher (int pin, long on, long off)
  {
   ledPin = pin;
   pinMode (ledPin, OUTPUT);
   OnTime = on;
   OffTime = off;
   ledState = LOW;
   previousMillis = 0;
  }
  void Update ()
  {
   // Finding out if it's time to change the state of the LED
   unsigned long currentMillis = millis (); // Current time in milliseconds
   if ((ledState == HIGH) && (currentMillis – previousMillis> = OnTime))
   {
    ledState = LOW; // Turn it off
    previousMillis = currentMillis; // Remembering a point in time
    digitalWrite (ledPin, ledState); // Implementing a new state
   }
   else if ((ledState == LOW) && (currentMillis – previousMillis> = OffTime))
```

```
    {
      ledState = HIGH; // Turn it off
      previousMillis = currentMillis; // Remembering a point in time
      digitalWrite (ledPin, ledState); // Implementing a new state
    }
   }
};
Flasher led1 (11, 100, 400);
Flasher led2 (12, 350, 350);
Flasher led3 (13, 300, 700);
void setup ()
{
}
void loop ()
{
  led1.Update ();
  led2.Update ();
  led3.Update ();
}
```

Thus, each additional LED requires only two lines of code.

5. Individual task

Ensure alternating flashing of the LEDs connected to the specified connectors (pins) each time the control buttons connected to the specified pins are pressed. Variants of tasks are given in Table. 4.1.

Table 4.1 – Task Options

№ var.	LED 1			LED 2		
	N control button pins	N pins for LED connecting	LED on/off time, s	N control button pins	N pins for LED connecting	LED on/off time, s
1	2	3	0,5/1	4	5	1/2
2	3	4	0,2/1,2	5	6	1,8/2,5
3	4	5	0,7/1,5	6	7	1,5/0,8
4	5	6	2/1	7	8	1,5/2,5
5	6	7	0,7/1,8	8	9	1,3/2,1
6	7	8	0,4/0,9	9	10	1/2
7	8	9	0,8/1,5	7	6	2/1
8	9	10	0,5/1,1	6	5	1,5/2,1
9	2	11	0,6/1,3	5	4	1,8/1
10	3	12	1,4/0,8	4	2	1,5/3,2

The operating mode of the LEDs is changed each time the corresponding control button is pressed. Ensure that the created project is protected from the effect of rattling contacts.

Order of work

1. Familiarize yourself with the connection of the button to the ATMega328 microprocessor.

2. Familiarize yourself with ways to eliminate contact rattle.

3. Familiarize yourself with the organization of time control functions on the ATMega328 microprocessor.

4. Develop a program for alternating flashing of LEDs connected to the specified ports each time the control button is pressed, connected to the specified pins, according to your version.

5. Develop a model of the scheme according to the individual task in the Tinkercad environment and check its operation.

6. Develop a model of the scheme according to the individual task in the PROTEUS environment and check its operation.

7. Assemble the circuit on the circuit board according to the individual task, enter the programs into the Arduino UNO R3, run and check its operation.

8. Check the correct functioning of the program in simulation environments and on a real diagram.

9. Draw up a work report.

Contents of the report

1. The topic of the laboratory work.

2. Purpose of the work.

3. Individual task.

4. Program and simplified block diagram of the algorithm for organizing the process of protection against contact rattling.

5. The program and a simplified block diagram of the algorithm for organizing the process of alternately turning on/off the LEDs connected to the specified pins each time you press the control buttons connected to the specified ports.

6. Working models of schemes according to an individual task in the Tinkercad environment.

7. Working models of schemes according to an individual task in the PROTEUS environment.

8. Photo of the working assembled circuit on the circuit board according to the individual task.

9. Conclusions.

Practical work 5

RESEARCH OF THE PRINCIPLES OF USING PULSE WIDTH MODULATORS ON THE ATMEGA328 MICROPROCESSOR

Purpose of the work: To get acquainted with the principles of using pulse width modulators and the features of the construction and varieties of servo motors, to study the operation of a servo drive controlled by the ATMega328 microprocessor on the Arduino platform. Gain practical skills in controlling servo drives with an ATMega328 microprocessor on an Arduino UNO R3 board.

Topics for pre-study:

– theoretical information about connecting the button to the ATMega328 microprocessor;

– theoretical information on the connection of the potentiometer to the ATMega328 microprocessor;

– theoretical information about programming in Arduino.

1. Pulse Width Modulation in Arduino

Pulse-width modulation (PWM) is divided into two types: analog and digital. Each of the types has its own advantages and can be implemented in different ways.

The principle of operation of an analog AI modulator is based on the comparison of two signals, the frequency of which differs by several orders of magnitude. The element of comparison is the op-amp (comparator). One of its inputs is supplied with a sawtooth voltage of high constant frequency, and the other is supplied with a low-frequency modulating voltage with variable amplitude. The comparator compares both values and generates rectangular pulses at the output, the duration of which is determined by the current value of the modulating signal.

PWM is also a way to control power under load by changing the duty cycle of pulses at a constant amplitude and pulse frequency.

There are two main areas of application of PWM:

1) in secondary power supplies, various power regulators, light source brightness

regulators, commutator motor speeds, etc. In these cases, the use of PWM can significantly increase the efficiency of the system and simplify its implementation;

2) to receive an analog signal using the digital output of the microcontroller. A kind of digital-to-analog converter (DAC) that is easy to implement and requires a minimum of external components (often one RC chain is enough).

The principle of PWM control is the change in the width of pulses at a constant amplitude and frequency of the signal (Fig. 5.1).

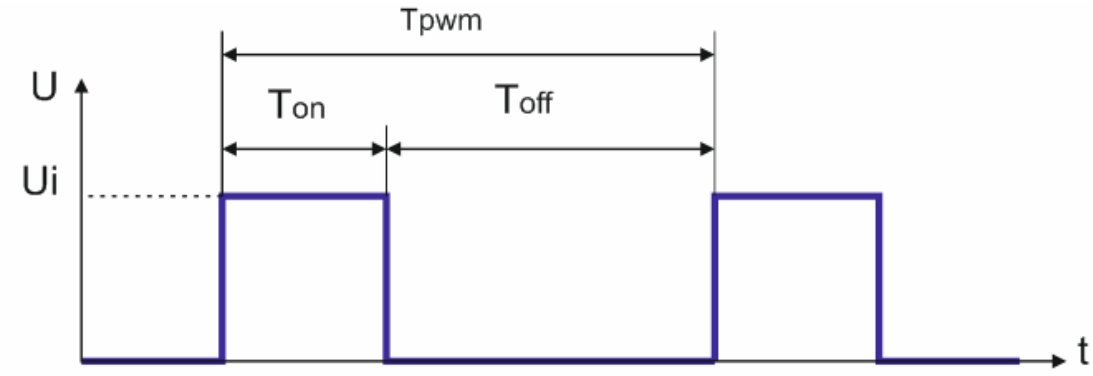


Figure 5.1 – Structure of the PWM signal

The main parameters of the PWM signal are:

U_i – pulse amplitude;

T_{on} – high (on) signal state time;

T_{off} – low (disconnected) signal state time;

T_{pwm} – PWM period time.

The power of the load is proportional to the ratio of the time of the signal on and off state.

This ratio determines the PWM fill factor:

$$K_w = T_{on}/T_{pwm}.$$

It shows how much of the period the signal is in the on state. It can vary from 0 (the signal is always off) to 1 (the signal is always on).

Most often, the percentage of occupancy is used, which ranges from 0 to 100% (Fig. 5.2).

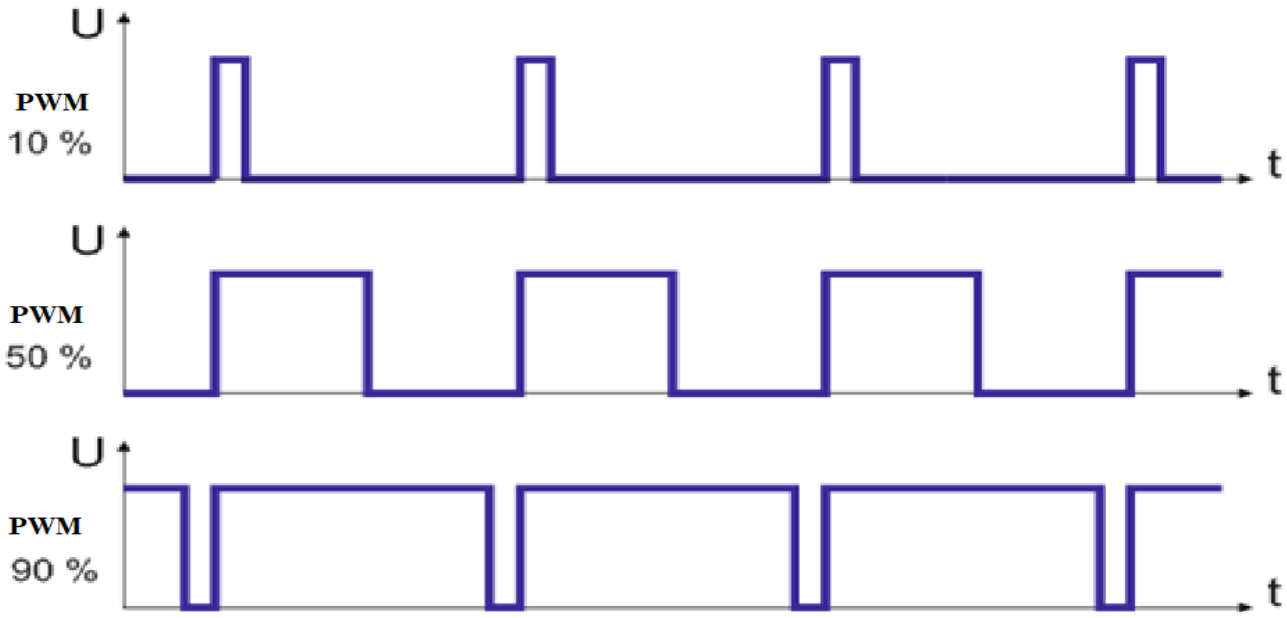


Figure 5.2 – Illustration of the percentage of filling

The average value of electrical power under load is strictly proportional to the fill factor. When they say that PWM is, for example, 10%, they mean the fill factor.

The active use of PWM-based controllers is due to their undeniable advantages:

– high signal conversion efficiency;

– stability of work;

– saving energy consumed by the load;

– low cost;

– high reliability of the entire device.

Arduino boards based on ATMega168/328 microcontrollers have 6 hardware AI modulators. PWM signals can be generated on pins 3, 5, 6, 9, 10, 11.

Hardware PWM is controlled using the system function analogWrite().

void analogWrite(pin, val)

The function switches the output to PWM mode and sets its fill factor. Before using analogWrite(), it is not necessary to call the pinMode() function to set the output to "output" mode.

Arguments to the analogWrite() operator:

pin – pin number for PWM signal generation;

val – PWM fill rate. Without additional settings, the val range from 0 to 255

corresponds to a fill factor from 0 to 100 %.

That is, it must be borne in mind that the bit depth of Arduino system PWM is 8 bits.

Example:

```
analogWrite(9, 25); // pin 9 PWM = 10 %
```

All three Arduino timers are used to generate PWM (Table 5.1).

Table 5.1 – Arduino timers used for PWM

Timer	Used for PWM generation on pins
Timer 0	pins 5 and 6
Timer 1	pins 9 and 10
Timer 2	Pins 3 and 11

By default, the Arduino system sets all PWM pins to the following parameters:

– frequency 976.56 Hz for Timer 0;

– frequency 488.28 Hz for Timer 1 and Timer 2;

– 8-bit resolution (0... 255).

If the timer is used for other purposes, such as interrupt control, the PWM parameters of the corresponding pins may not match the above values. Therefore, when using some libraries, such as MsTimer2, TimerOne, or the like, some pins cannot be used as PWM signal generators.

2. Introduction to the concept of servo

The structure of the servo motor is quite complex (Fig. 5.3). The servo has an internal feedback mechanism that allows the controller to accurately determine the position of the servo. This mechanism is used to ensure the precise movement of the servo to a predetermined position.

A gear reducer is located in the upper part of the device, which allows you to significantly increase the torque of a DC motor by reducing its rotational speed. Below is a potentiometer, the task of which is to determine at what angle to turn the output

shaft of the gearbox. Finally, in the depths of the case is a small control board, which makes the servo motor smart. This board constantly monitors the current state of the shaft and corrects it in case the shaft tries to move away from the set position.

Figure 5.3 – Appearance and internal structure of the servo motor

3. Creating a servo control project in a Proteus environment

We will design a servo drive in Proteus and develop a DC control drive using logic elements. Let's start by creating a simple DC motor that Proteus already has and is easy to use. To control the motor, we will use the direct method, that is, applying voltage to both sides of it. This type of motor needs different polarity at its two ends. If the polarity of the connection is straight, then the DC motor is moving in one direction, and changing the polarity will cause it to move in the opposite direction.

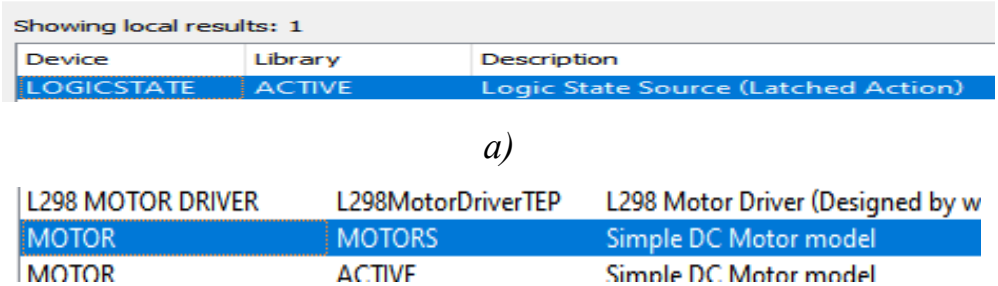
To implement this method:

– create a new project in the Proteus environment;

– add two LOGICSTATE elements (Fig. 5.4, a);

– add a servo motor MOTOR (Fig. 5.4, b);

– connect these elements as shown in Fig. 5.5.



а)

б)

Figure 5.4 – Adding elements

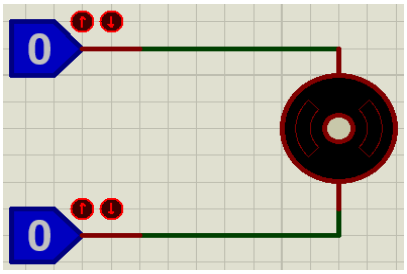


Figure 5.5 – Connecting elements

LOGICSTATE has two states: "1" and "0". When set to "0" means 0 V, and when set to "1" it means 5 V. To change the state, you need to click on the required Logic State.

The direction of the motor will depend on the logic of the Logic State. Thus, there will be only four states:

1. When both states are "0", the motor will not move and remain stationary.

2. When both states are "1", the motor will not move and remain stationary.

3. The motor will move clockwise when the upper state is "1" and the lower state is "0".

4. The motor will move counterclockwise when the upper state is "0" and the lower state is "1".

PWM servo control in a Proteus environment

PWM is a technique for controlling analog signals using digital signals. PWM is commonly used to control servos that provide precise positioning of objects, such as robot movement. Unlike a conventional DC motor, here the PWM level does not set the rotational speed, but the angle of rotation.

A servo controller usually works on the principle of PWM (pulse-width modulation, PWM – Pulse Width Modulation). This means that the servo receives the input signal in the form of pulses, and its position depends on the width of these pulses (the relative duration of the pulse over its period). To adjust the position of the servo, signals with a pulse width of 1 ms to 2 ms are usually used. The pulse width corresponds to the angular position of the servo, where 1 ms corresponds to full rotation in one direction, 1.5 ms to the neutral position, and 2 ms to full rotation in the other direction (Fig. 5.6).

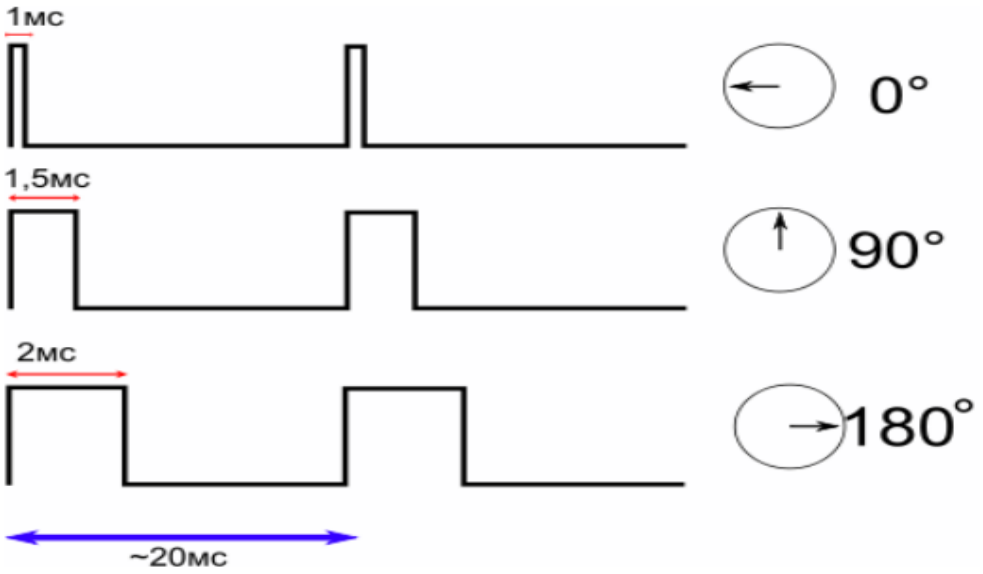


Figure 5.6 – Control signals

Let's create a new project in the Proteus environment and add three elements of MOTOR-PWMSERVO. Then connect the power supply to the servo motors and, by pressing twice on the POWER element, set it to +12 V. Connect everything as shown in Fig. 5.7.

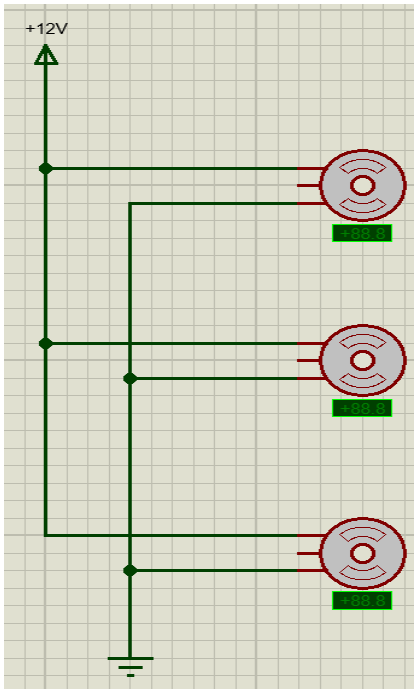


Figure 5.7 – Connection of MOTOR-PWMSERVO elements

To create signals of the desired frequency and duration, we can install elements that will generate three different signals for each of the servos. To do this, go to Generator

Mode in the left toolbar and select the PULSE component. Then add three instances of the PULSE component to the circuit and connect them as shown in Fig. 5.8.

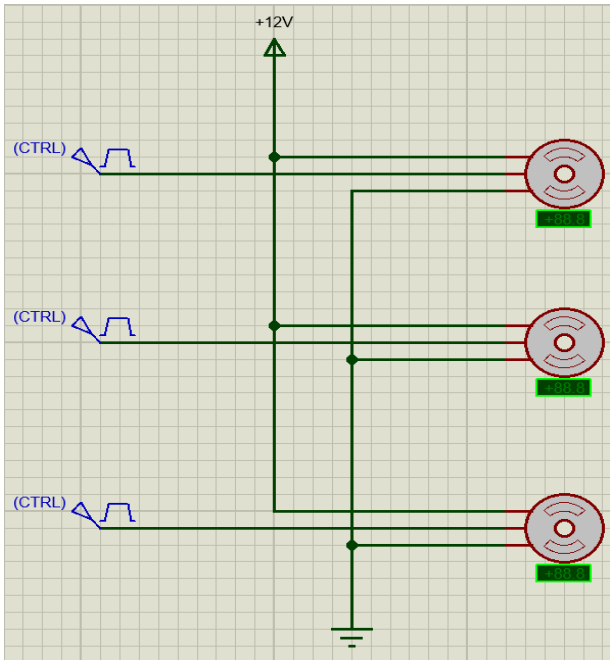


Figure 5.8 – Connecting the elements of the MOTOR-PWMSERVO

According to Fig. Figure 5.6 adjusts the pulse width of the three PULSE elements to 1 ms, 1.5 ms and 2 ms, respectively (Fig. 5.9).

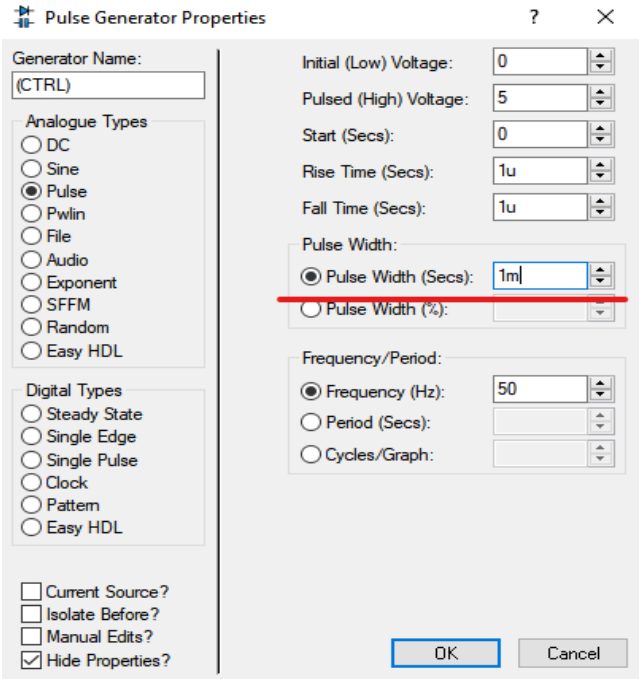


Figure 5.9 – Setting pulse widths for PULSE elements

Once we have everything connected and configured, we can run the simulation. The results of the simulation are shown in Fig. 5.10.

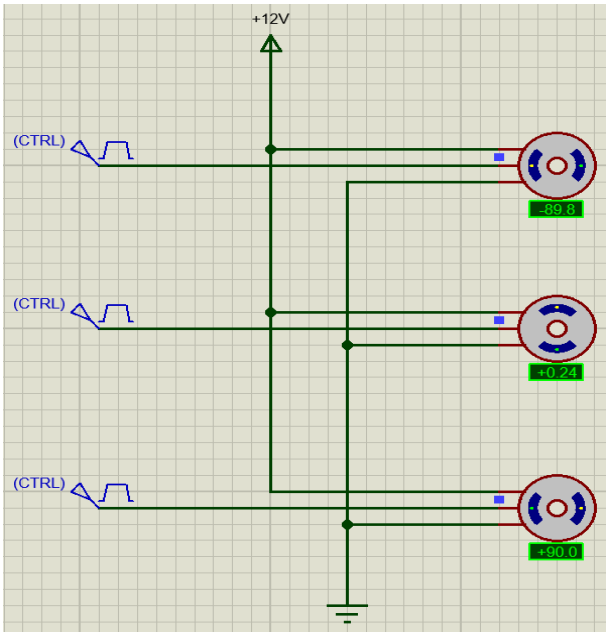


Figure 5.10 – Result of simulation of servo motors
in the Proteus environment

4. Creating a Servo Project Using Arduino IDE and Tinkercad

The wiring diagram of the servo motor to the Arduino board is shown in Fig. 5.11. As already mentioned, the servo motor has three drives: power, signal and ground.

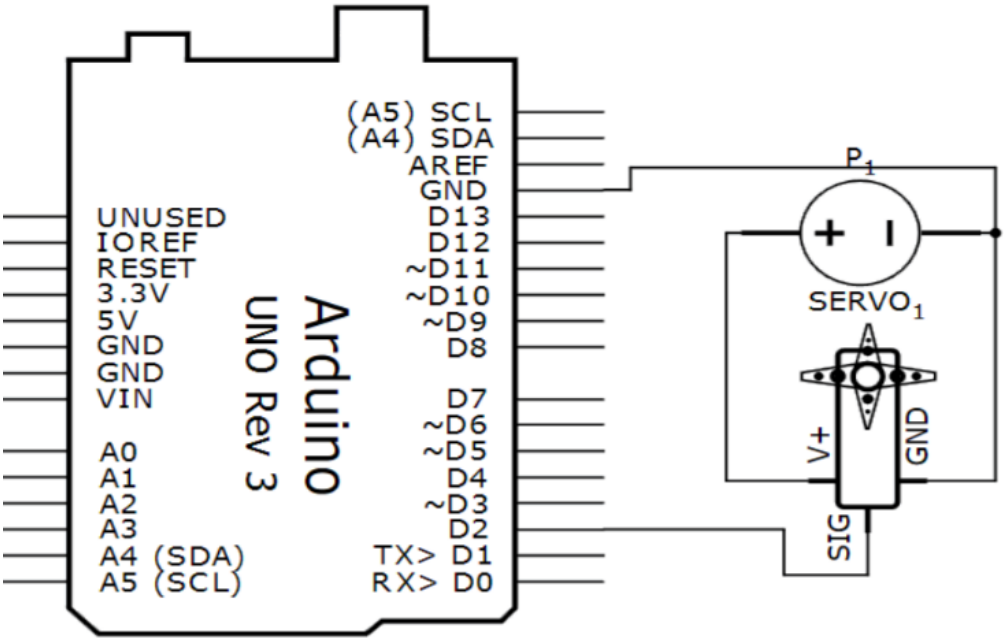


Figure 5.12 – An example of connecting a servo motor

The ATMega328 microcontroller, which is used on the Arduino UNO R3 board, uses PWM-enabled output pins, such as pins 9 and 10, to control the servo. To generate a PWM signal, the functions of the Servo.h library, which is available in the Arduino IDE, are used.

A servo control app might look like this:

```
#include <Servo.h>
Servo servo;
void setup() {
  servo.attach(9); // Install a pin to control the servo drive
}
void loop() {
  servo.write(90); // Set the angle of rotation of the servo drive to 90 degrees
  delay(1000); // Wait 1 second
  servo.write(0); // Set the angle of rotation of the servo drive to 0 degrees
  delay(1000); // Wait 1 second
}
```

An example of a circuit for controlling a servo in a Tinkercad environment might look like this (Fig. 5.12).

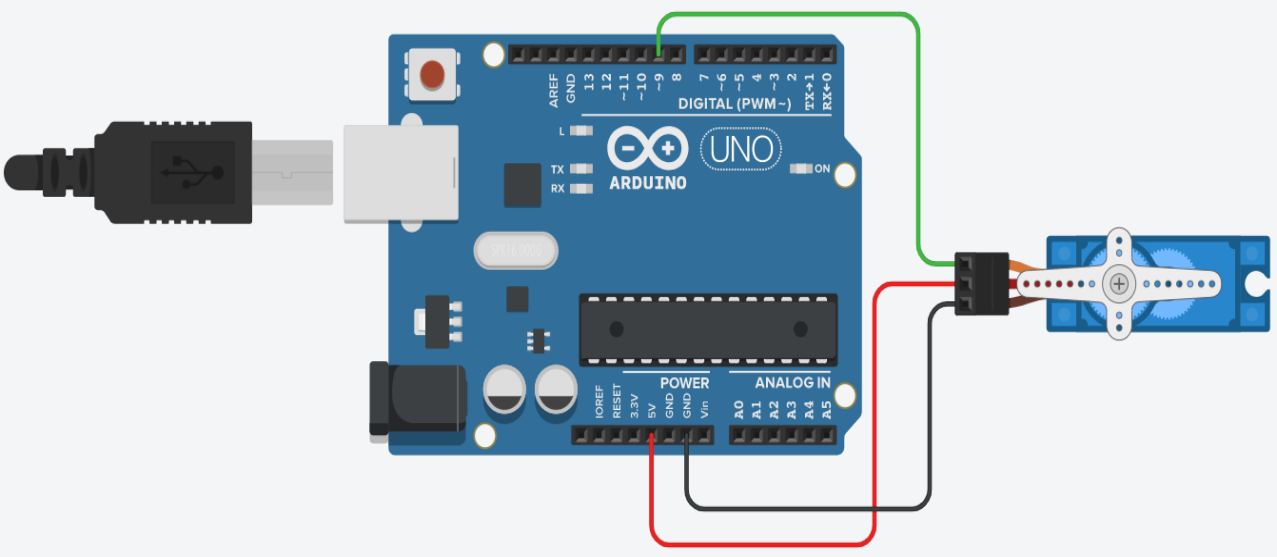


Figure 5.12 – Example of a servo control program

4. Individual task

Ensure the operation of the servo motor in accordance with the tasks given in Table 5.2.

Table 5.2 – Task Options

№ var.	Task
1	2
1	Create a project "Servo motor movement using a potentiometer". Connect the servo motor to the pin **3**. When the potentiometer is rotated, the servo motor should move from 0 to 180 degrees, depending on the position of the potentiometer.
2	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **11**. Connect button to pin **3**. When the button is pressed, the servo motor should move counterclockwise. When the button is pressed, the servo motor must stop.
3	Create a project "Servo motor movement with a button". Connect the servo motor to port **9**. Connect the button to the pin **2**. When the button is pressed, the servo motor should move clockwise. When the button is pressed, the servo motor must stop.
4	Create a project "Servo motor movement using a potentiometer". Connect the servo motor to the pin **5**. When the potentiometer is rotated, the servo motor should move from 0 to 180 degrees, depending on the position of the potentiometer.
5	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **6**. Connect button to pin **2**. When the button is pressed, the servo motor should rotate **15** degrees counterclockwise, when the button is pressed, return to its original position.
6	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **10**. Connect button to pin **3**. When the button is pressed, the servo motor should rotate **15** degrees clockwise, when the button is pressed, return to its original position.
7	Create a project "Servo motor movement using a potentiometer". Connect the servo motor to the pin **9**. When the potentiometer is rotated, the servo motor should move from 0 to 180 degrees, depending on the position of the potentiometer.
8	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **3**. Connect button to pin **2**. When the button is pressed, the servo motor should rotate **30** degrees counterclockwise, when the button is pressed, return to its original position.
9	Create a "Servo Motor Movement with Button" project. Connect the servo motor to port **6**. Connect button to pin **3**. When the button is pressed, the servo motor should rotate **30** degrees clockwise, when the button is pressed, return to its original position.
10	Create a "Servo Motor Movement with Button" project. Connect the servo motor to port **3**. Connect button to pin **2**. When the button is pressed, the servo motor should rotate **45** degrees counterclockwise, when the button is pressed, return to its original position.
11	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **6**. Connect button to pin **3**. When the button is pressed, the servo motor should rotate **45** degrees clockwise, when the button is pressed, return to its original position.

1	2
12	Create a project "Servo motor movement using a potentiometer". Connect the servo motor to port **5**. When the potentiometer is rotated, the servo motor should move from 0 to 180 degrees, depending on the position of the potentiometer.
13	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **6**. Connect button to pin **2**. When the button is pressed, the servo motor should rotate **75** degrees counterclockwise, when the button is pressed, return to its original position.
14	Create a project "Servo Motor Movement with Button". Connect the servo motor to port **10**. Connect button to pin **3**. When the button is pressed, the servo motor should rotate **75** degrees clockwise, when the button is pressed, return to its original position.

Order of work

1. Familiarize yourself with the basic principles of connecting servo motors to the ATMega328 microprocessor.

2. Become familiar with PWM servo control in a Proteus environment using the ATMega328 microprocessor.

3. Develop a program for turning the servo motor according to the individual task.

4. Develop a model of the circuit according to the individual task in the Tinkercad environment and check its operation.

5. Develop a model of the circuit according to the individual task in the Proteus environment and check its operation.

6. Assemble the circuit on the circuit board according to the individual task, enter the program into the Arduino UNO R3, run and check their operation.

7. Check the correct functioning of the program in simulation environments and on a real diagram.

8. Draw up a work report.

Contents of the report

1. The topic of the laboratory work.

2. Purpose of the work.

3. Individual task.

4. The program and a simplified block diagram of the algorithm for organizing the servo drive control process using the ATMega328 microprocessor according to an

individual task.

5. Screenshot and link to the working model of the scheme according to the individual task in the Tinkercad environment.

6. Working model of the scheme according to the individual task in the Proteus environment.

7. Photo of the working assembled circuit on the circuit board according to the individual task.

8. Conclusions.

Practical work 6

RESEARCH OF THE ORGANIZATION OF INFORMATION INPUT FROM SENSORS AND INDICATION OF RESULTS ON THE ATMEGA328 MICROPROCESSOR

Purpose of work: To get acquainted with the features of the structure and varieties of sensors, to study the operation of input and output elements on the ATMega328 microprocessor on the Arduino platform. Gain practical skills in working with sensors and results display elements on the ATMega328 microprocessor on the Arduino UNO R3 board.

Topics for preliminary study:

– theoretical information on connecting an LCD display to an ATMega328 microprocessor;

– theoretical information about dynamic indication;

– theoretical information about programming in Arduino.

1. Introduction to the concept of a sensor

Sensors for Arduinos are electronic devices that measure various parameters of the physical environment and convert them into signals that can be understood by an Arduino microcontroller. These sensors can measure various parameters such as temperature, humidity, pressure, light, motion, sound, and many others.

The sensors can be connected to the Arduino using various interfaces such as analog and digital ports, I2C, SPI, and UART bus interfaces. Some sensors may also use wireless communication protocols, such as Bluetooth and Wi-Fi.

One of the key aspects of sensors for Arduino is their accuracy and reliability. If the sensor is not accurate or reliable enough, then this can lead to incorrect measurements, which can lead to incorrect behavior of the system controlled by the Arduino. One of the most popular sensors for Arduino is the DS18B20 temperature sensor, which measures temperature with an accuracy of 0.50C. Another popular sensor is the DHT11 humidity sensor, which measures humidity and air temperature. The light sensor BH1750FVI is a

fairly accurate light sensor with an I2C interface (Fig. 6.1).

a) *b)* *c)*

Figure 6.1 – Sensors: temperature DS18B20 (a), humidity DHT11 (b), illumination BH1750FVI (c)

Overall, sensors for Arduinos are an important component of control and data acquisition systems. They allow you to measure various parameters of the physical environment and provide the necessary information for decision-making and systems management.

The DS18S20 temperature sensor is a single-wire digital thermometer that can measure temperatures ranging from -55°C to +125°C with an accuracy of ±0.5°C.

DS18B20 consists of a thermistor from which the temperature is read, as well as a logic interface that converts the signals from the thermistor into digital data transmitted via the 1-Wire bus. This means that the sensor has only one conductor for data and power transmission, which allows it to be quite easy to use and connect to microcontrollers. Fig. 6.2 shows how you can connect DS18B20 to an Arduino.

DS18B20 has a built-in memory for storing configuration data and the latest temperature measurement. To read data from the sensor, you need to send a request to receive data, after which the DS18B20 transmits digital data via the 1-Wire bus. Also, DS18B20 can be configured to operate in a feedback mode in which it continuously sends data to the microcontroller without prompting.

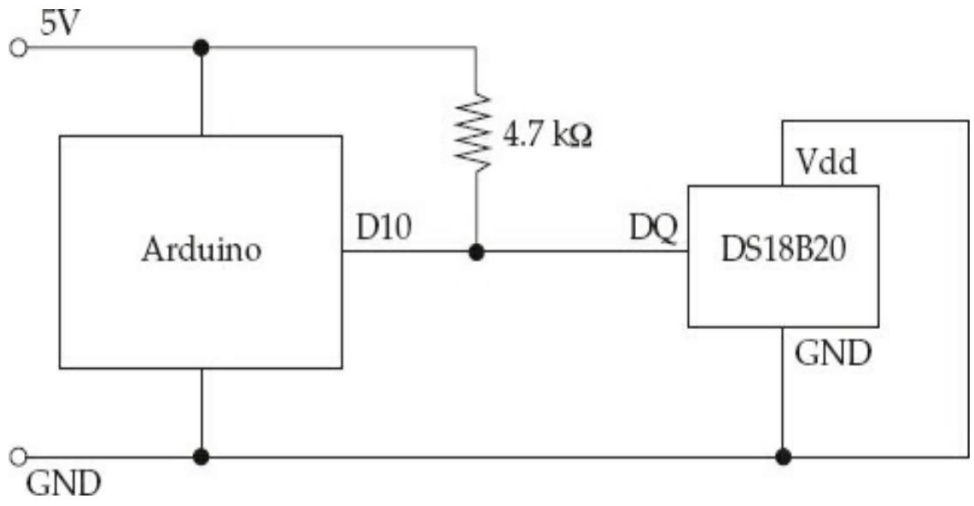


Figure 6.2 – Connecting a 1-Wire device to an Arduino

2. Creating a project with the TMP36 temperature sensor in TinkerCad and Proteus environments

TMP36 and DS18B20 are two different types of temperature sensors with their own characteristics. The main difference between these sensors lies in their design and interface with the microcontroller board. DS18B20 is a digital temperature sensor that works using a 1-wire interface, while TMP36 is an analog temperature sensor that measures voltages that vary with temperature.

TMP36 is an integrated circuit that is a temperature sensor with high accuracy, low power consumption, and ease of use. It has a built-in precision thermostat with an accuracy of ±1 °C, which allows you to measure temperatures in the range from -40 °C to +125 °C. TMP36 has three outputs: VCC, GND and VOUT. The VCC is connected to the power supply and the GND is connected to ground, the VOUT emits an analog voltage signal that varies with temperature (Fig. 6.3). With this signal, temperature can be measured by connecting the VOUT to the analog input of a microcontroller or ADC. TMP36 has a low temperature error rate and great resistance to changes, making it ideal for temperature measurement in various applications such as temperature control of critical systems, home appliances, thermometers, and other devices.

Figure 6.3 – TMP36 Temperature Sensor

Let's create a new project in the Proteus environment. Let's put together a diagram that will correspond to the following task:

– when the temperature is less than 0 ^{0}C, the blue LED (3 pins) lights up;

when the temperature is from 0 ^{0}C to 50 ^{0}C, the orange LED (4 pins) lights up;

– when the temperature is more than 50 ^{0C}, the red LED (5 pin) lights up.

For the diagram, it is necessary to add the elements that are shown in Fig. 6.4, then add Ground and Power. After that, we get a diagram as shown in Fig. 6.5.

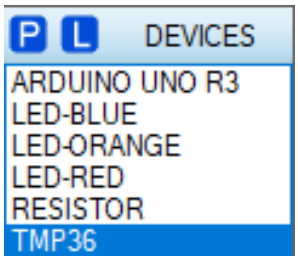


Figure 6.4 – Adding elements to the diagram

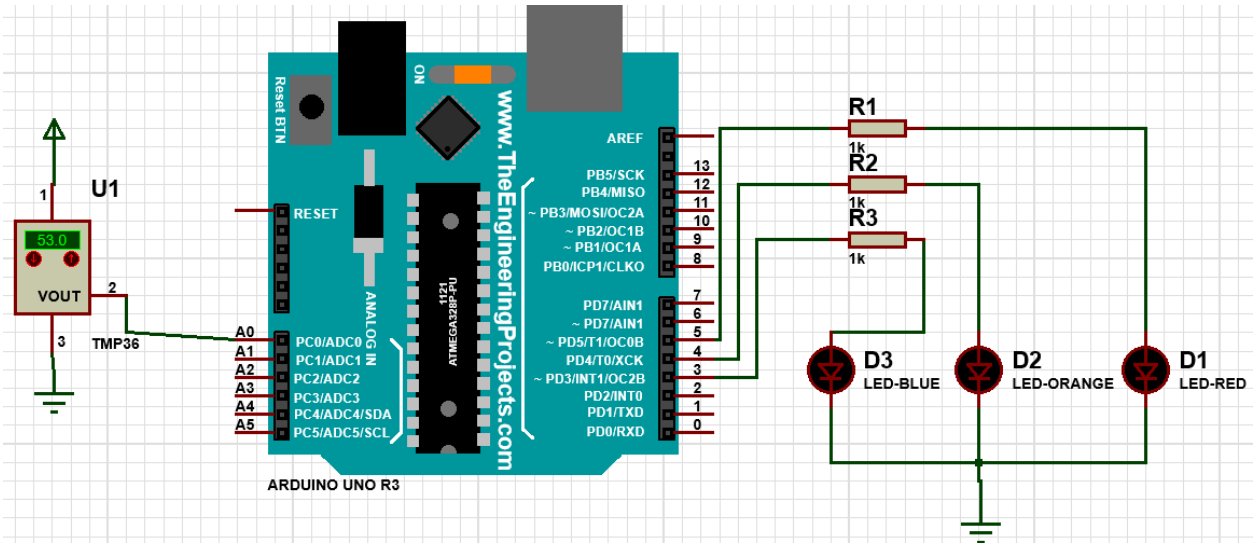


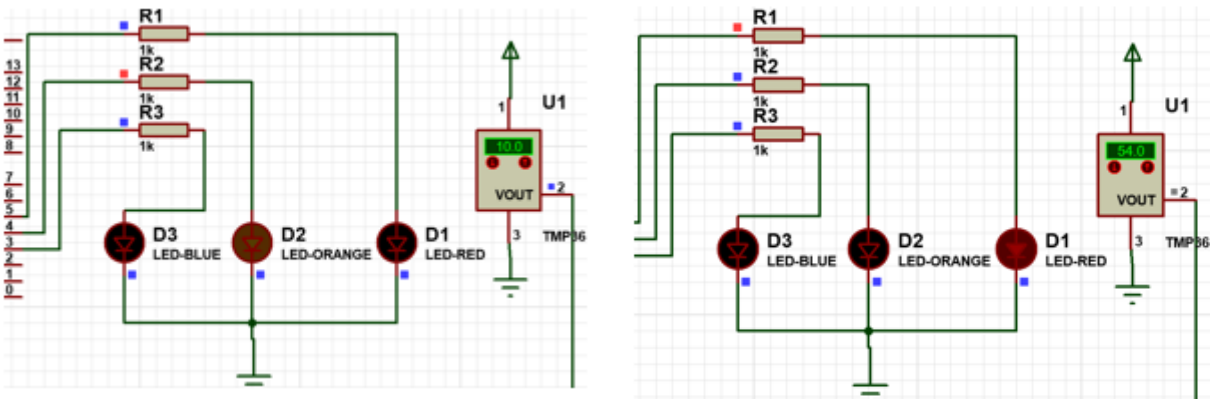
Figure 6.5 – Project diagram in the Proteus meter environment

on the TMP36 temperature sensor

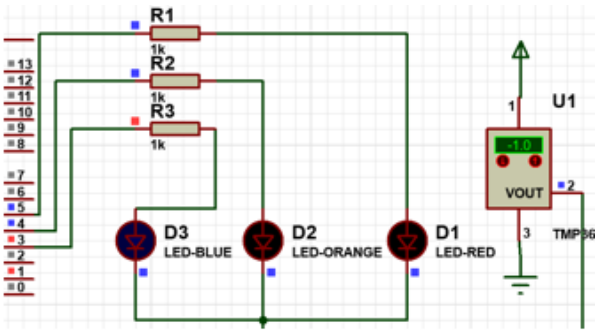
The following is the program code of the simplest example that satisfies the condition of the task:

```
void setup() {
  pinMode(A0, INPUT); // setting the pin A0 to input mode
  pinMode(3, OUTPUT); // Set the Pin 3 to output mode
  pinMode(4, OUTPUT); // Set the Pin 4 to output mode
  pinMode(5, OUTPUT); // Set the Pin 5o to output mode
  Serial.begin(9600); // Setting the data transfer rate
}
void loop() {
  int tmp = analogRead(A0); // Reading data from the sensor
  float voltage = (tmp * 5.0) / 1024; // Convert a 10-bit number to a voltage measurement
  float milliVolt = voltage * 1000; // This is multiplied by 1000 to convert it to millivolts
  float tmpCel = (milliVolt - 500) / 10; // for the TMP36 sensor. Range (-40 °C to +125 °C)
  float tmpFer = (((tmpCel * 9) / 5) + 32); // used to convert Celsius-> Fahrenheit
  digitalWrite(3, LOW); // setting the output pin 3 to the LOW level
  digitalWrite(4, LOW); // setting the output pin 4 to the LOW level
  digitalWrite(5, LOW); // setting the output pin 5 to the LOW level
  if (tmpCel <= 0) {
    digitalWrite(3, HIGH); // setting output pin 3 to HIGH level
    delay(1000); // 1 second delay
    digitalWrite(3, LOW); // setting the output pin 3 to the LOW level
    delay(1000); // 1 second delay
  } else if (tmpCel > 0 && tmpCel <= 50) {
    digitalWrite(4, HIGH); // setting the output pin 4 to the HIGH level
    delay(1000); // 1 second delay
    digitalWrite(4, LOW); // setting the output pin 4 to the LOW level
    delay(1000); // 1 second delay
  } else {
    digitalWrite(5, HIGH); // setting the output pin 5 to the HIGH level
    delay(1000); // 1 second delay
    digitalWrite(5, LOW); // setting the output pin 5 to the LOW level
    delay(1000); // 1 second delay
  }
}
```

Let's create a binary file and check the result as shown in Fig. 6.6.



а) *б)*



в)

Figure 6.6 – The result of the project in the Proteus environment for the meter on the temperature sensor TMP36

Let's repeat the same steps in the Tinkercad environment (Fig. 6.7 – 6.8).

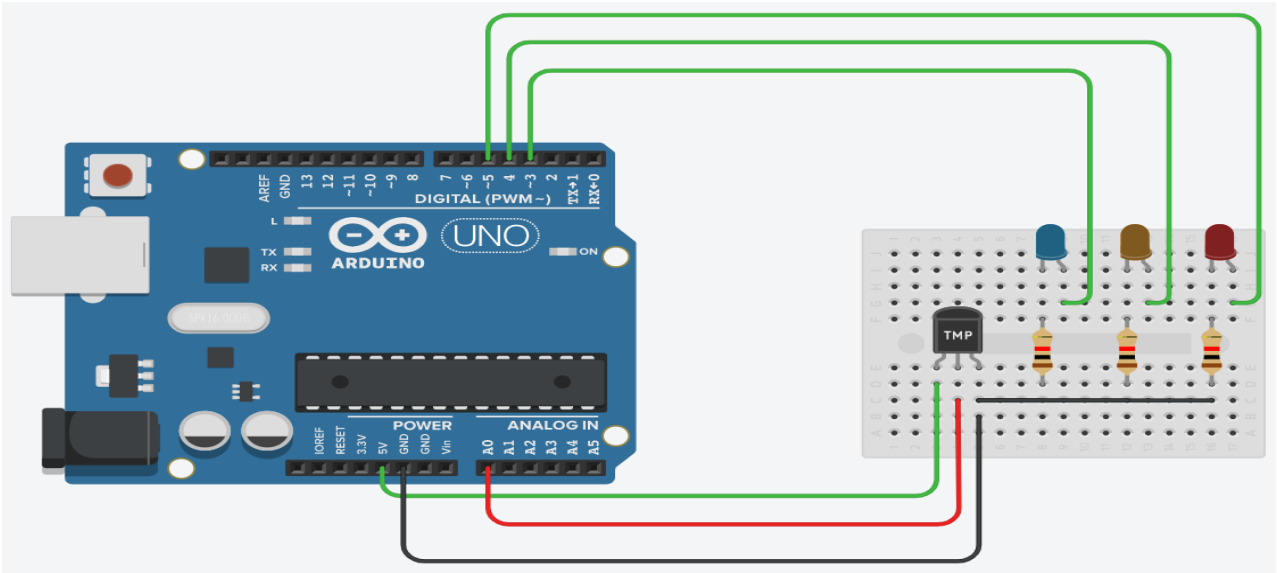


Figure 6.7 – Diagram of the meter design on the sensor TMP36 Temperature in TinkerCAD Environment

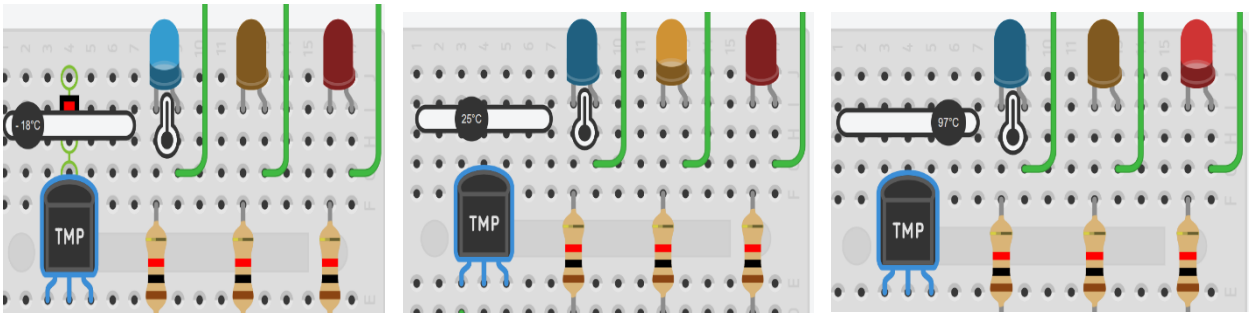


Figure 6.8 – The result of the project in the TinkerCAD environment for the meter on the temperature sensor TMP36

3. Connecting LCD indicators to a microprocessor

As noted in Case Study 3, liquid crystal displays that display symbolic information, such as text and numbers, are the most inexpensive and easy to use of all LCDs and are called indicators. The connection diagram of the LCD 1602 to the Arduino UNO is shown in Fig. 6.9.

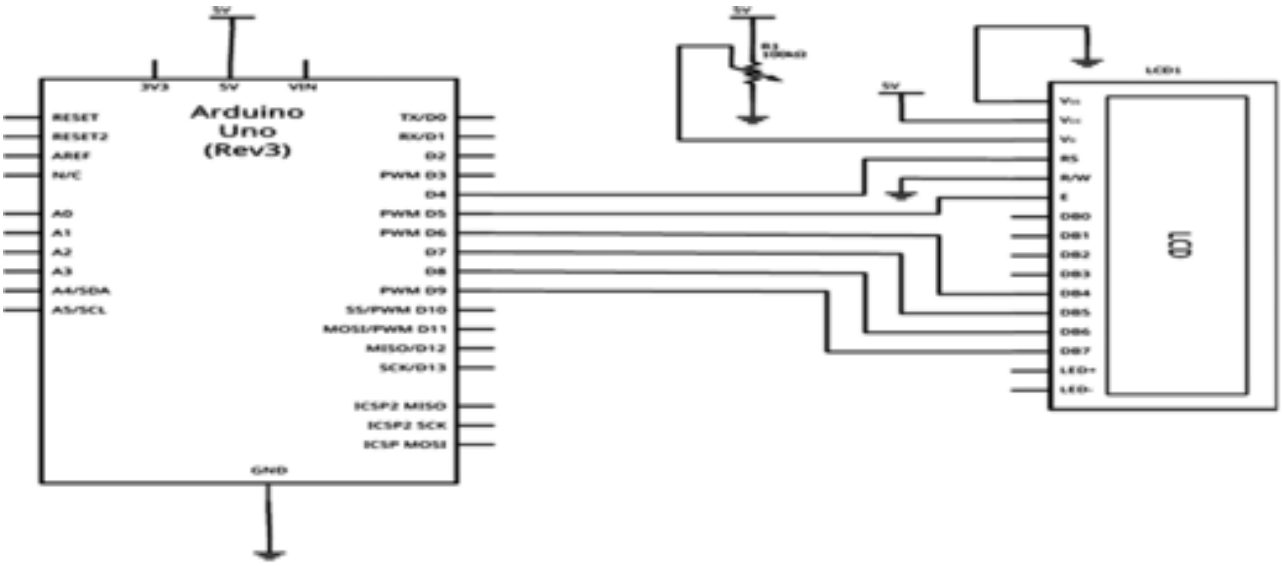


Figure 6.9 – Connection diagram of LCD 1602 to Arduino UNO

The connection diagram of the LCD 1602 to the Arduino UNO in the TinkerCAD environment is shown in Fig. 6.10.

The following is the program code for the simplest example of how LCD 1602 works:

```
#include <LiquidCrystal.h>
int seconds = 0;
```

```
LiquidCrystal lcd_1(12, 11, 5, 4, 3, 2);
void setup()
{
  lcd_1.begin(16, 2); // set up the number of columns and rows on the LCD.
  // print a message to the LCD.
  lcd_1.print("hello world!");
}
void loop()
{
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting
  // begins with 0):
  lcd_1.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd_1.print(seconds);
  delay(1000); // wait for 1000 millisecond(s)18b
  seconds += 1;
}
```

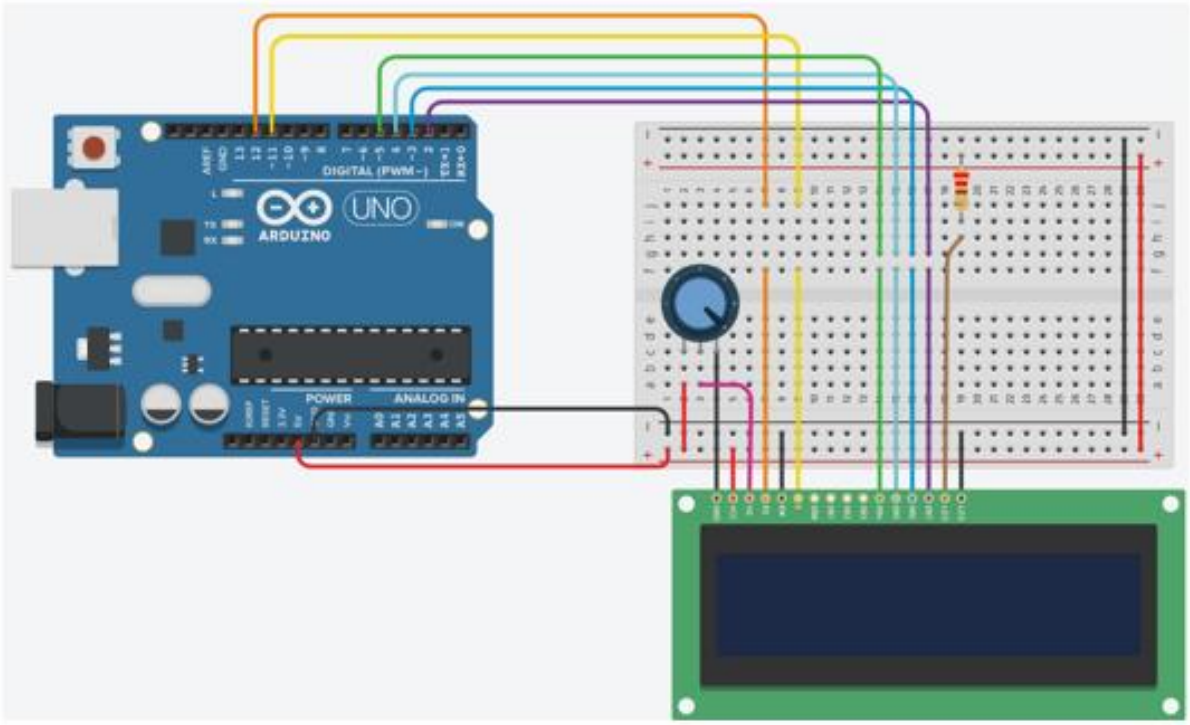


Figure 6.10 – Connection diagram of LCD 1602 to Arduino UNO in the TinkerCAD environment

4. Individual task

Ensure the operation of the sensor in accordance with the tasks given in Table. 6.1.

Table 6.1 – Task Options

№ var.	Task
1	Create Temperature Measurement projects in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to Pin **10** in Proteus CAD. Connect the temperature sensor **TMP36** to pin **A0** in Tinkercad. The measurement result should be displayed on the LCD in degrees Celsius.
2	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to the pin **9**. Connect the **TMP36** temperature sensor to **the A1** port in Tinkercad. The measurement result should be displayed on the LCD in degrees Fahrenheit.
3	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to the pin **8**. The measurement result should be displayed on the LCD in degrees Celsius.
4	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to the pin **7**. Connect the **TMP36** temperature sensor to **the A2** port in Tinkercad. The measurement result should be displayed on the LCD in Fahrenheit.
5	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to pin **6**. Connect the **TMP36** temperature sensor to **the A3** port in Tinkercad. The measurement result should be displayed on the LCD in degrees Celsius.
6	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to the pin **5**. Connect the **TMP36** temperature sensor to the **A4 pin** in Tinkercad. The measurement result should be displayed on the LCD in Fahrenheit.
7	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to pin **5**. Connect the **temperature sensor TMP36** to the **A5 pin** in Tinkercad. The measurement result should be displayed on the LCD in degrees Celsius.
8	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to the pin **4**. Connect the **temperature sensor TMP36** to pin **A0** in Tinkercad. The measurement result should be displayed on the LCD in Fahrenheit.
9	Create a Temperature Measurement project in Tinkercad and Proteus. Connect the **temperature sensor DS18B20** to pin **3**. Connect the **temperature sensor TMP36** to pin **A1** in Tinkercad. The measurement result should be displayed on the LCD in degrees Celsius.
10	Create a Temperature Measurement project. Connect the **temperature sensor DS18B20** to pin **2**. Connect the **temperature sensor TMP36** to Pin **A2** in Tinkercad. The measurement result should be displayed on the LCD in degrees Fahrenheit.

Order of work

1. Familiarize yourself with the basic principles of connecting sensors to the ATMega328 microprocessor.

2. Develop a program for entering information from sensors and displaying results.

3. Develop a sensor operation program according to the individual task.

4. Develop a model of the circuit according to the individual task in the Tinkercad environment and check its operation.

5. Develop a model of the circuit according to the individual task in the PROTEUS environment and check its operation.

6. Assemble the circuit on the circuit board according to the individual task, enter the program into the Arduino UNO R3, run and check its operation.

7. Check the correct functioning of the program in simulation environments and on a real scheme.

8. Draw up a work report.

Contents of the report

1. The topic of the laboratory work.

2. Purpose of the work.

3. Individual task.

4. Programs and simplified block diagrams of algorithms for organizing the process of entering information from sensors and indicating the results using the ATMega328 microprocessor and LCD indicator according to an individual task.

5. Screenshot and link to the working model of the scheme according to the individual task in the Tinkercad environment.

6. Working model of the scheme according to the individual task in the Proteus environment.

7. Photo of the working assembled circuit on the circuit board according to the individual task.

8. Conclusions.

References

1. Methodical instructions for performing laboratory work on the academic discipline "Microprocessor programming" for full-time and part-time students in the specialty "Computer Engineering" / A. O. Podorozhniak, S. G. Mezheritsky, G. V. Heiko. Kharkiv: NTU "KhPI". – 2020. – 36 p.

2. Methodical instructions for performing laboratory work on the discipline "Structure and functioning of microprocessors" for full-time and part-time students in the specialty "Computer Engineering" / A. O. Podorozhniak, S. G. Mezherytskyi, H. V. Heiko, V. V. Lymarenko. Kharkiv: NTU "KhPI". – 2020. – 56 p.

3. Methodical instructions for independent work of students in the discipline "Structure and functioning of microprocessors" for full-time and part-time students in the specialty "Computer Engineering" / A. O. Podorozhniak, H. V. Heiko. Kharkiv: NTU "KhPI". – 2020. – 20 p.

4. Methodical instructions for independent work of students in the discipline "Microprocessor programming" for full-time and part-time students in the specialty "Computer Engineering" / A. O. Podorozhniak, N. Y. Lyubchenko. Kharkiv: NTU "KhPI". – 2021. – 15 p.

5. Design and analysis of electrical circuits in the Proteus VSM software environment. Methodical instructions for independent work of students from the course "Design of microprocessor control systems of technological processes" / Medvid V.R., Pistsio V.P. – Ternopil: TNTU, 2018. – 26 p.

6. Methodical instructions for laboratory work on the discipline "Electronics and microprocessor systems" for applicants for higher education of the first (bachelor's) level in the specialty 015.10 "Professional education. Computer Technologies" full-time and part-time forms of education / Vasylets S.V., Vasylets K.S. – Rivne: NUWEE, 2019. – 134 p.

7. Microprocessor and Microcontroller Systems: Part 2. Design of microprocessor systems: Laboratory practicum. Helps. for students. educational program "Integrated Information Systems" specialty 126 "Information Systems and Technologies" / A.O. Novatsky. – Kyiv: KPI them. Igor Sikorsky, 2021. – 268 p.

8. Programming of microprocessors in protected mode: educational and methodological manual / I. S. Zykov, S. G. Mezheritsky, A. O. Podorozhniak, I. P. Khavina. – Kharkiv: DISA PLUS LLC, 2018. – 264 p.

9. Povoroznyuk A. I. Arkhitektura komputeriv [Computer architecture]. Methodical instructions for the implementation and design of a course project for full-time and part-time students in the direction of 123 "Computer Engineering" / A. I. Povoroznyuk, O. A. Povoroznyuk, G. E. Filatova. – Kharkiv: NTU "KhPI", 2022. – 42 p.

10. Rysovanyi, O. M. "Sistemne programming: textbook dlya studeniv napravlenu «Komputerna inzheneriya» vyshekh uchivachnykh zavleniya" [System programming: textbook for students of the direction "Computer Engineering" of higher educational institutions]. – Kharkiv: NTU "KhPI", 2010. 912 p.

11. Simon Monk. Programming Arduino. Getting Started with Sketches. – McGrow Hill, 2012. – 177 p.

12. Simon Monk. Programming Arduino. Next Steps. Going Further with Sketches. – McGrow Hill, 2014. – 212 p.

13. Proteus Design Suite. Getting Started Guide. – Labcenter Electronics Ltd. – 2020. – 191 p.

14. The Arduino Platform and C Programming. Introduction to Programming the Internet of Things (IOT). [Electronic resource] URL: https://www.coursera.org/learn/arduino-platform (last accessed 11.09.2024).

15. ATmega328p datasheet [Electronic resource] URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (last accessed 11.09.2024).

16. СТ3ВО-ХПІ-3.01-2021. Text documents in the field of educational process. General requirements for implementation. – Kharkiv: NTU "KhPI", 2021. – 47 p.

CONTENT

Educational edition

Methodological instructions
for practical works in the
«Microprocessor architecture and programming» academic discipline
for full-time and part-time students majoring in
«Computer Engineering»

Authors:
PODOROZHNIAK Andrii Oleksiiovych,
HEIKO Hennadii Viktorovych,
MEZHERYTSKYI Sergii Hennadiiovych,
LIUBCHENKO Oleksii Viktorovych

Responsible for the issue prof. Oleksandr ZAKOVOROTNYI
The work for the publication was recommended by prof. Mykola ZAPOLOVSKYI

In the author's edition

Plan for 2024, pos. 918.
Signed to print 24.10.2024. Format 60x84 1/16. Accounting and publishing sheet 2,8.

NTU "KhPI" Publishing Centre,
2, Kyrpychova str., Kharkiv, 61002
Certificate of the subject of publishing DK No. 5478 dated 21.08.2017.

Electronic edition