TME EDUCATION
www.TMEeducation.com

Documentation of the

Arduino based TME Educational Board

TME-EDU-ARD-2 January 2019

T M E ®
Electronic Components
WWW.TME.EU

www.TMEeducation.com

Contents

Introduction

The **TME-EDU-ARD-2** educational board for learning of Arduino programming has been prepared for the needs of the **TME Education** programme. More information on the subject can be found at:

- https://www.tmeeducation.com
- https://fb.com/tmeeducation

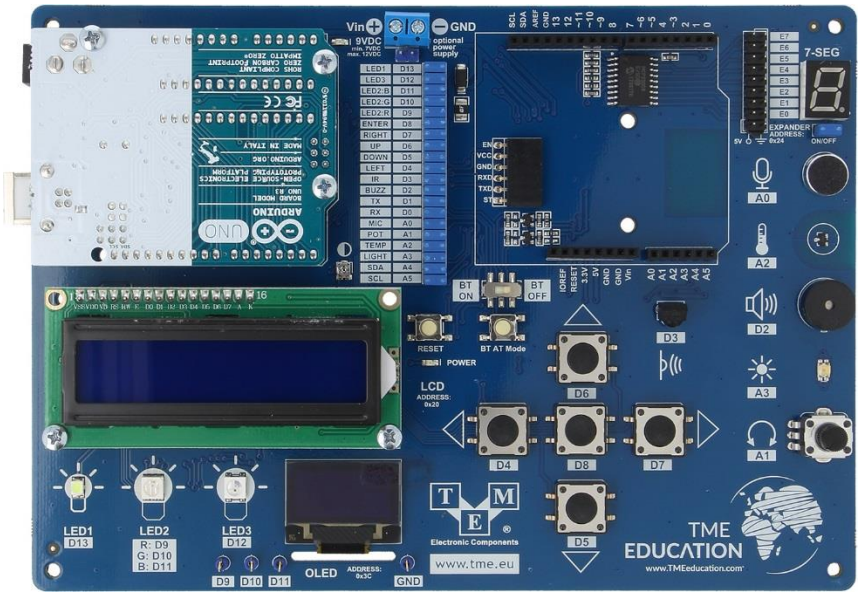
This documentation describes the use of peripherals included in the **TME-EDU-ARD-2** educational board, **which works with the popular Arduino UNO controller**. The board has over 20 peripherals using all available outputs of the controller. The elements are connected, however, it is still possible to change these connections using the jumpers (more on this later).

What is Arduino?

Arduino platform is one of the most popular solutions dedicated to beginners to electronics in the world. It integrates popular AVR microcontrollers and dedicated, beginners-friendly programming language (based on C / C++) into a coherent and easy-to-use tool with an infinite number of applications.

Basic kit description

There is a place for the Arduino UNO controller in the top left-hand corner of the board. In the bottom left-hand corner there is a section with displays (LCD and OLED) and with LEDs. To the right of the display there are 5 large monostable buttons.



On the right-hand side there is an IR receiver, a potentiometer, light sensor, buzzer, temperature sensor, microphone and a 7-segment display with a pin expander. In the upper, middle part of the board, you can find a connector for external power, a set of jumpers to reconfigure the connections and place for a Bluetooth module and any additional shield with other accessories.

List of available peripherals

The following peripherals are connected to the Arduino UNO:

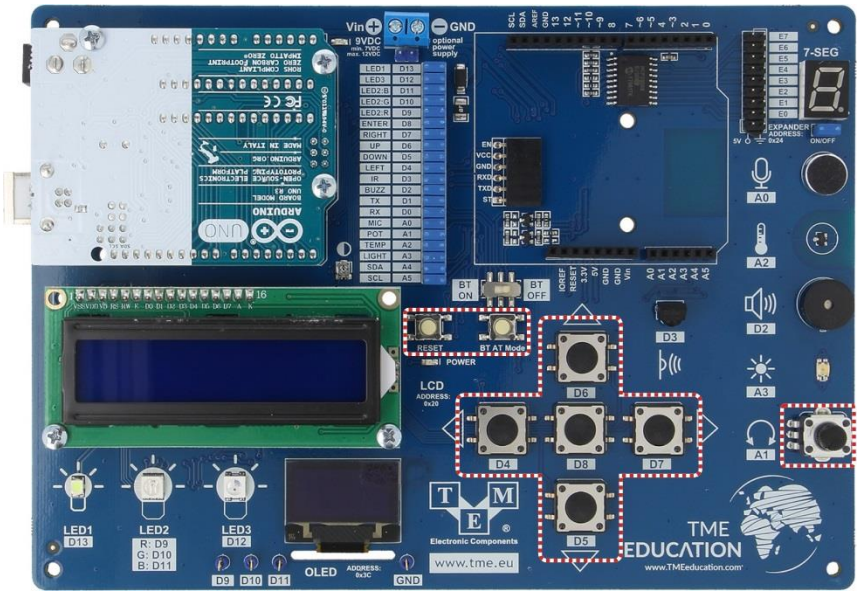
- Simple, single-colour LED (LED1 - D13).
- Simple, three-colour (RGB) LED (LED2 - D9, D10, D11).
- Digitally controlled, three-colour (RGB) LED WS2812 (LED3 - D12), to which another 4, identical LEDs located on the bottom-side of the board are connected in series.
- Keyboard with 5 monostable buttons arranged in the forward (D6), backward (D5), left (D4), right (D7) and centre (D8).
- Analog sensors:
  - Potentiometer (A1),
  - Light sensor (A3),
  - Thermometer (A2),
  - Microphone (A0).
- IR receiver (D3).
- Buzzer with generator (D2).
- Text LCD display 2x16 characters, connected via the I2C expander (address: 0x20).
- OLED graphic display with I2C interface (address: 0x3C).
- 7-segment display connected via the I2C expander (address: 0x24).

Additionally, the board has:

- LED (POWER) indicating correct powering of the system.
- A small monostable button (RESET) that allows you to reset the Arduino controller.
- A small monostable button (BT AT Mode) to enter into AT commands mode of the BT module.
- A sliding two-position switch (BT ON / BT OFF) to disconnect the optional BT module.
- 4 measuring points in the form of meshes (connected to D9, D10, D11, GND).

Safety information

When operating the device, avoid direct contact with the PCB (touching electronic components and paths), because in extreme situations it may damage the board. The exception are 7 buttons and a potentiometer, which can be used in programmes loaded onto the board.

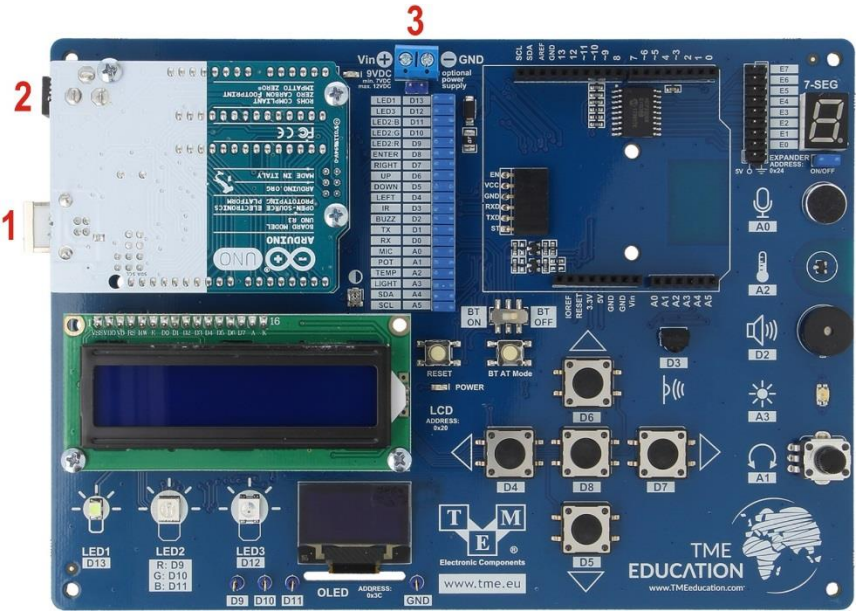


Any changes to the configuration of jumpers
and connecting additional modules should be performed
when the board is **disconnected** from power source!

Before turning on the board, it is also worth making sure that the plate was not accidentally placed on metal objects (e.g. on a screwdriver), which could close the signals flowing under the plate.

Power supply

The educational board can be powered in one of the three ways:



- USB connector (1) placed on the Arduino board (power supply from the computer's USB port or from the 5V adapter allowing to connect a cable with a USB type B connector).
- External power connector (2) on the Arduino board (from 7V to 12V).
- External power connector (3) placed on the expansion board (from 7V to 12V).

> In the vast majority of cases, it's best to power the educational board with a USB cable that is used for Arduino programming.

Necessary software

For programming the board you need the Arduino IDE environment, which can be downloaded for free from the project website, i.e.: https://arduino.cc. From the page navigation menu, select the "*SOFTWARE*" tab. On the newly opened page, go to the "*Download the Arduino IDE*" section and select the appropriate version for the given system from the list on the right.

Then you will be asked for voluntary financial support for the project. At this stage, you can donate or download the software for free by selecting the "*JUST DOWNLOAD*" button. At this point, the installer will start downloading.

Programming the educational board

Programmes written for Arduino are called sketches. All programmes should include the *setup* and *loop* functions. Instructions included in the *setup* function are performed only once (when turning Arduino on). All settings and instructions to be performed at the beginning of the device operation should be placed here (immediately after switching on).

The *loop* function acts as an infinite loop. Instructions contained inside it will be performed all the time. This loop starts after the instructions inside the *setup* function are completed.

A blank sketch, which should be the starting point for each programme:

```
// the setup function runs once when you press reset or power the board
void setup() {

}

// the loop function runs over and over again forever
void loop() {

}
```

Comments

It's possible to add comments inside the programme, i.e. information that are skipped when sending a sketch to Arduino, and they are only meant to make it easier for programmers to understand the code. Comments can be placed in one line (then they should be preceded by "//" characters) or in many lines (then the comment text should be placed between "/*" and "*/" characters).

```
void setup() {
   // comment in one line

   /*
   multi-line
   comment
   */

}
```

Comments can be placed in any part of the programme. Often they are also used to temporarily "turn off" a part of the programme, e.g. for testing time.

Uploading the programme to the board

To upload the programme to the Arduino, first you need to (one time only):

1. connect the board with a USB cable to the computer,
2. from the Arduino IDE toolbar choose: *Tools* > *Board* > *Arduino/Genuino UNO*,
3. from the Arduino IDE toolbar select*: Tools* > *Port* and there check the number of the COM port, next to which the name Arduino UNO appeared.

Then, in order to compile the programme (i.e. translate it into a language understandable for Arduino) and upload it to the board, click the option *Upload*, which is visible in the Arduino IDE menu in the form of a round icon with an arrow to the right. After programme is loaded correctly, a corresponding message will appear at the bottom of the IDE, e.g. "Upload completed", and after a while the code will start working on the board.

If an optional Bluetooth module is connected to the system,
the appropriate slide switch should be set to **BT OFF**
when the programme is loaded onto the board.

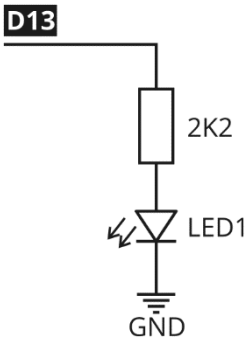
Peripherals support in practice

This part of the manual describes the most important information related to turning on all peripherals that are located on the board. The elements have been described in order from the easiest to the most difficult, thanks to which it is possible to gradually learn different aspects of the Arduino language in practice.

Simple, single-colour LED (LED1 - D13)

The educational board has been equipped with one, standard LED diode, which is located in the lower left corner of the board. There is a marking **D13** next to the diode, which means that this element can be operated in a digital way (*D from digital*). The number "13" is the number of the Arduino pin to which the element is connected with the default configuration of jumpers.

Similar descriptions have been placed next to all peripherals.



The diode was connected to the Arduino output with an anode (through the resistor). It means that in order to turn on the diode, the logic high (or "1") should be set at the digital output **13**.

The programme that turns the diode on should start with setting the pin as an output. To do this, use the *pinMode* function (*pin_number*, OUTPUT), in which we replace *pin_number* with the pin number to be used in the *output* mode.

From that moment on, it will be possible to switch on or off the diode with the *digitalWrite(pin_number, status)* function anywhere in the

programme. Where, similarly in place of *pin_number* we insert pin number to which the diode was connected, and in the place described as *status*, we give one of the two possible options: LOW, that is logical low state (or "0") or HIGH, that is logical high state (or "1").

Setting the low state is equivalent to giving the mass potential to the output, and setting the high state is equivalent to giving a positive power bus to the output, i.e. 5 V in this case.

The programme that turns the diode on will look like this:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(13, OUTPUT);

  // turn the LED on
  digitalWrite(13, HIGH);
}

// the loop function runs over and over again forever
void loop() {

}
```

Both instructions are included in the *setup* function, because after switching on Arduino, we turn the diode on only once and we do not have to do anything with it.

Using definitions

In the above programme, the pin number to which the diode is connected had to be entered in two places (in the configuration of the output and in the instruction turning on the diode). To simplify later changes, information about the pin number can be written in the form of definitions e.g.:

```
#define LED1 13
```

From now on, the compiler will automatically substitute the number "13" at every place in the programme where "LED1" is placed. This is particularly convenient when making changes to the pins to which the peripherals are connected in more complex programmes.

The programme turning on the diode with the use of definitions may look like this:

```
#define LED1 13

// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(LED1, OUTPUT);
```

```
  // turn the LED on
  digitalWrite(LED1, HIGH);
}

// the loop function runs over and over again forever
void loop() {

}
```

Delays

When creating many programmes, it is useful to be able to enter a delay. A popular example is a programme flashing a diode. In practice, this means it's necessary to turn on the diode, wait for a certain time, turn off the diode, wait for a certain time and start the cycle from the beginning. For this purpose, in Arduino programming, the *delay(time)* function is used, where for *time* we enter the duration of the delay in milliseconds (1 second = 1000 milliseconds).

During the delay, the entire programme is stopped (and stands still)!

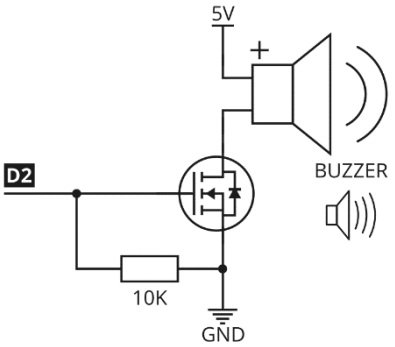
An example of a programme flashing a diode:

```
#define LED1 13

// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(LED1, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // turn the LED on
  digitalWrite(LED1, HIGH);
  delay(500); // wait for 500 ms

  // turn the LED off
  digitalWrite(LED1, LOW);
  delay(200); // wait for 200 ms
}
```

This time, the diode control instructions have been moved to the *loop* function, because they are to be executed all the time (in the loop). Of course, the instruction configuring the pin (to which the diode is connected) as an output remained inside the *setup* function (because the configuration is done only once).

Buzzer with generator (D2)

A buzzer with a generator (**sound emitting device**) is connected to pin D2. Controlling of this device is performed in the same way as the above example with a diode. The logical "1" on the Arduino output will make the buzzer sound.

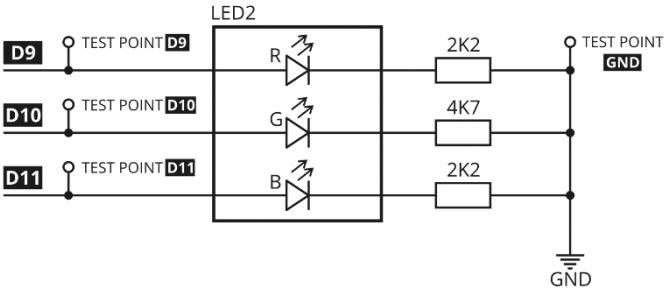


An sample programme using a sound generator:

```
#define BUZ 2

// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(BUZ, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // turn the buzzer on
  digitalWrite(BUZ, HIGH);
  delay(1000); // wait for 1000 ms
  // turn the buzzer off
  digitalWrite(BUZ, LOW);
  delay(200); // wait for 200 ms
}
```

Simple, three-colour (RGB) LED (LED2 - D9, D10, D11)

There is LED2 diode on the board, which is an RGB diode. This means that its housing has 3 luminous structures: red (R), green (G) and blue (B).



Using the descriptions that are on the board, you can create the appropriate definitions:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11
```

Controlling of individual LEDs is performed in an analogous way to the previously described LED1. We start from the pin configuration as input, and then control each colour separately:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11

// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(LED2R, OUTPUT);
  pinMode(LED2G, OUTPUT);
  pinMode(LED2B, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // turn selected color on and turn off other
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, LOW);
  delay(500); // wait for 500 ms

  // turn selected color on and turn off other
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, LOW);
  delay(500); // wait for 500 ms

  // turn selected color on and turn off other
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, HIGH);
  delay(500); // wait for 500 ms
}
```

Of course, it is also possible to light two / three colours at the same time. Thanks to this, it is possible to mix colours and obtain different, intermediate colours. An example of such a programme:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11

// the setup function runs once when you press reset or power the board
void setup() {

// pin configuration
  pinMode(LED2R, OUTPUT);
  pinMode(LED2G, OUTPUT);
  pinMode(LED2B, OUTPUT);
}
```

```
// the loop function runs over and over again forever
void loop() {
  // different color settings
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, LOW);
  delay(500); // wait for 500 ms

  // different color settings
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, HIGH);
  delay(500); // wait for 500 ms

  // different color settings
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, HIGH);
  delay(500); // wait for 500 ms
}
```

Control of the RGB diode with PWM signal

On some Arduino terminals (marked with the "~" sign) it is possible to obtain a square wave signal with variable duty cycle (PWM). In the TME-EDU-ARD-2 educational board all colours of the RGB diode have been connected to such terminals. Thanks to this, it is additionally possible to control the brightness of each of them.

In order to generate the PWM signal, use the *analogWrite(pin_number, duty_cycle)*, where in place of the *pin_number*, we enter the terminal number to which the diode is connected, and in place of the *duty_cycle* we enter a number in the range from 0 to 255.

> Setting 0 will be equivalent to lighting the diode with the brightness of 0%, and 255 will mean lighting with the brightness equal to 100%.

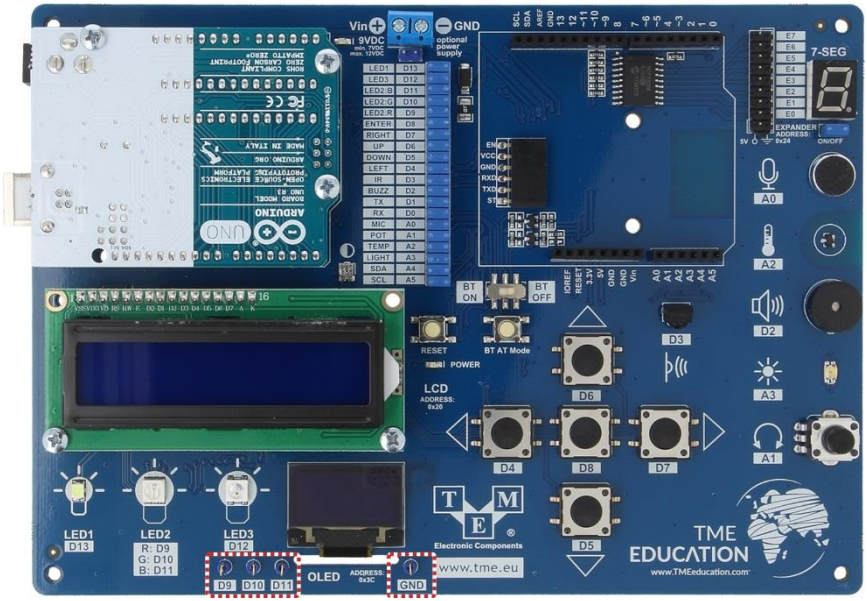
Following example demonstrates how to change the brightness of a red diode:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11

void setup() {
  // pin configuration
  pinMode(LED2R, OUTPUT);
}

void loop() {
  // set LED brightness
```

```
  analogWrite(LED2R, 0);
  delay(500); // wait for 500ms

  // set LED brightness
  analogWrite(LED2R, 50);
  delay(500); // wait for 500ms

  // set LED brightness
  analogWrite(LED2R, 150);
  delay(500); // wait for 500ms

  // set LED brightness
  analogWrite(LED2R, 255);
  delay(500); // wait for 500ms
}
```

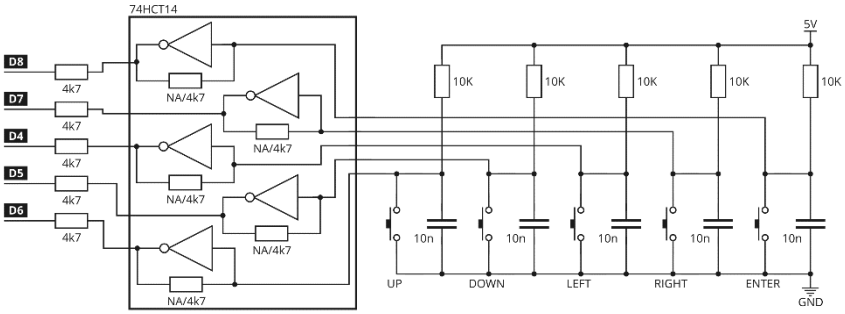
Using testing terminals described as D9, D10, D11 and GND, it is possible to connect the board to the oscilloscope, which will show changes in the signal duty cycle that the diode is being controlled with.

If you don't have an oscilloscope, a voltmeter can be connected to this point (between any measuring point and GND). With the increase of the PWM signal duty cycle, the voltmeter should indicate higher and higher voltage (from 0 to 5V).



5-button keyboard (D4, D5, D6, D7, D8)

There are 5 monostable buttons on the board that are connected to digital pins D4, D5, D6, D7 and D8. All buttons have been **equipped with RC filters**, thanks to which the readings are stable and there is no need to program the inputs in order to avoid the so-called. contact vibrations.



The buttons are connected via reversing buffers, so when the button is not pressed, the **low state** (logic "0") is visible at the Arduino input. After pressing the button, the input state will be high (logic "1").

> Depending on the version of the board, the resistors marked as "NA" might not be soldered in, which doesn't influence operation of the circuit.

To use the buttons, the pin should be configured as an input using the *pinMode(pin_number, INPUT)* function, in which as *pin_number* we enter the pin number to be used in the input mode. In the further part of the programme, we read the input status using the *digitalRead(pin_number)* function.

The programme, which will turn on the LED1 diode after pressing the button connected to the D4 pin (to the left):

```
#define LED1 13
#define SW_LEFT 4

// the setup function runs once when you press reset or power the board
void setup() {
  // pin configuration
  pinMode(LED1, OUTPUT);
  pinMode(SW_LEFT, INPUT);
}

// the loop function runs over and over again forever
void loop() {
  if (digitalRead(SW_LEFT) == HIGH) { // if the pushbutton is pressed
    digitalWrite(LED1, HIGH);
```

```
  } else {
    digitalWrite(LED1, LOW); // if not
  }
}
```

Communication with a computer via UART

The USB cable connected to Arduino can power the board, it is used for programming it, but it can also be used for communication between the set and the computer. For this purpose, it is necessary to use the UART interface, which should be turned on one time at the beginning with the below instruction:

```
Serial.begin(115200);
```

The value given in brackets, i.e. "115200" means the selected transfer rate, which in this case was set to one of the most popular, standard values.

Then, using the following command, it is possible to send various values to the computer. In this case, it is the word "TEST":

```
Serial.println("TEST");
```

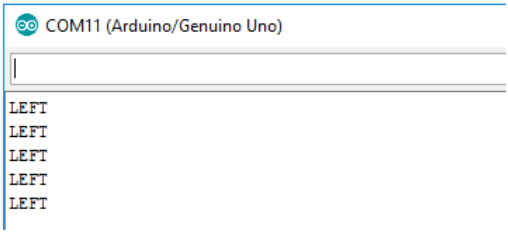
In order to check the operation of this mechanism in practice, you can expand the previous programme that used the button and add to it sending information about the button pressed to the computer.

Additional delay added at the end of the programme means that the loop is executed only every 200 ms, so the information will not be sent to the computer too often.

```
#define LED1 13
#define SW_LEFT 4

void setup() {
  // pin configuration
  pinMode(LED1, OUTPUT);
  pinMode(SW_LEFT, INPUT);
  Serial.begin(115200);
}

void loop() {
  if (digitalRead(SW_LEFT) == HIGH) { // if the pushbutton is pressed
    digitalWrite(LED1, HIGH);
```

```
    Serial.println("LEFT");
  } else { // if not
    digitalWrite(LED1, LOW);
  }

  delay(200);
}
```

After uploading this programme, run the *Serial Port Monitor*, which can be found in the Arduino IDE menu in the *Tools* option. From now on, after pressing the button on the board (D4), the text "LEFT" should appear in a new window:



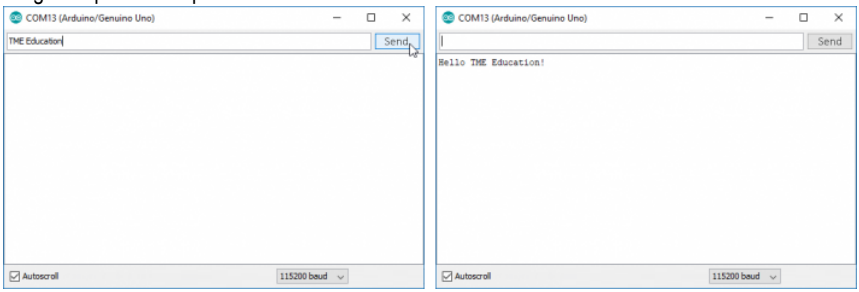
If instead of "LEFT" text, random, unreadable characters are displayed, make sure that the appropriate transfer rate is selected at the bottom of the window (in the drop-down menu), i.e.: 115200.

Sending information to Arduino

On a similar basis, you can also send information to Arduino. A sample program of this type is shown in the example below. The program continually checks if Arduino received any data from the computer. If yes, it is saved in the *text* variable, and a welcome screen is displayed.

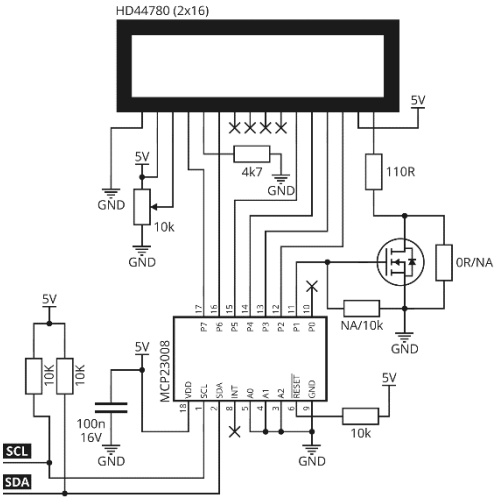
```
String text = "";
void setup() {
  Serial.begin(115200);
}
void loop() {
  if(Serial.available() > 0) {             //Is there any text received?
    text = Serial.readStringUntil('\n');   //Read that text
    Serial.println("Hello " + text + "!"); //Print information
  }
}
```

Program operation in practice:



Text LCD display 2x16 characters

One of the main elements of the educational board is a text LCD display, on which you can display 2 lines of text (16 characters each). The display is connected via an additional intermediary system that communicates with Arduino via the digital I2C interface. Thanks to this, only 2 signal lines are used to control this display.



To run the display, it is necessary to add the appropriate libraries. For this purpose, you can use the *Library Manager* built into the Arduino IDE, which is located in the menu: *Sketch* > *Add Library* > *Manage libraries...*

You need Internet access to run the *Library Manager*! Alternatively, libraries can be imported from previously prepared ZIP archives. To do this, use the option: *Sketch* > *Add Library* > *Add .ZIP library...*

We add libraries in the programme using the *#include* command:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
```

Then, it is necessary to provide information about connection method of the LCD display to the expander. First information given to the called function (0x20) is the address of the expander controlling this display.

```
// information about the LCD connection
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);
```

The next step is a display configuration, in which we set the allowed number of columns (16) and rows (2). This information is closely related to the display used and should always be set in the same way:

```
void setup() {
  // lcd configuration
  lcd.begin(16, 2);
}
```

In order to display information on the display, use the *lcd.setCursor(x, y)* command, which will position the cursor in the appropriate location. Lines and positions in lines are counted from zero (not from 1), so for example:

- *lcd.setCursor(0, 0)* - start with the first character in the first line,
- *lcd.setCursor(0, 1)* - start with the first character in the second line,
- *lcd.setCursor(5, 0)* - start with the sixth character in the first line,
- *lcd.setCursor(5, 1)* - start with the sixth character in the second line.

After setting the cursor, you can use the *lcd.print("TEXT")* function, which will display the indicated text on the LCD. In order to delete display content, you can write a string of spaces on it or use the special function *lcd.clear()*.

Demo of an example that displays text alternately in two lines:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// information about the LCD connection
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
  // lcd configuration
  lcd.begin(16, 2);
}

void loop() {
  // set the cursor to position
  lcd.setCursor(0, 0);
  // print text on lcd
  lcd.print("TME");
  // set the cursor to position
  lcd.setCursor(0, 1);
  // print text on lcd
  lcd.print("TESTER - 2");

  delay(1000);
  lcd.clear();

  // set the cursor to position
  lcd.setCursor(0,0);
```

```
  // print text on lcd
  lcd.print("TESTER - 1");
  // set the cursor to position
  lcd.setCursor(0, 1);
  // print text on lcd
  lcd.print("TME");

  delay(1000);
  lcd.clear();
}
```

If the text is not visible on the display in spite of the programme being loaded, it may be necessary to **adjust the contrast**, which can be done using a small, silver potentiometer located above the top right-hand corner of the display. All you have to do is to turn the potentiometer gently with the loaded programme and running system, e.g. using a flat-head screwdriver.

Support for analog sensors

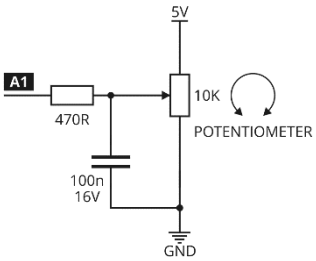
There are several analog sensors on the board, and the way of operating all of them is very similar. Analog sensors **measure physical value** (e.g. temperature), and show the result in as a certain voltage value. For example, the warmer it is, the higher is the voltage from the sensor supply range on the sensor output (in this case from 0 to 5V).

To use the analog input you do not have to declare it in any way. It is enough to read the voltage value, which is represented by a number between 0 and 1023. Where 0 means ~0V and 1023 means ~5V. The information is read using the *analogRead(pin_number)* function.

Range 0-1023 results from the fact that Arduino has an analog-digital converter (ADC) with a resolution of 10 bits.

Analog sensors - potentiometer (A1)

The simplest analog sensor located on the board is a potentiometer connected to the A1 pin (A for “analog”). In this case, the use of this element as a voltage divider is used.

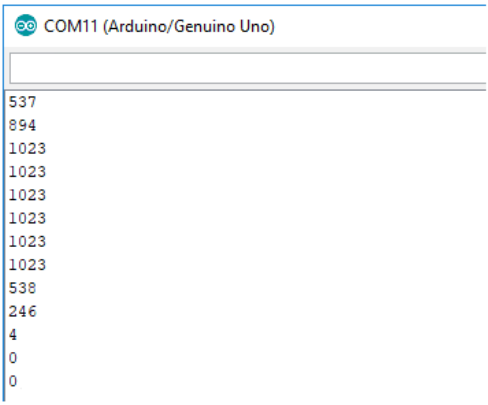


The following programme will read information from this sensor and send it to the computer via UART. New values will be sent every 150 ms:

```
void setup() {
  // UART configuration
  Serial.begin(115200);
}
```

```
void loop() {
  // read sensor data
  int pot = analogRead(A1);

  // send information to PC
  Serial.println(pot);

  // wait for 150 ms
  delay(150);
}
```

After uploading the programme and launching the *Serial Port Monitor*, a sequence of numbers should be visible, the values of which change during manual adjustment of the potentiometer.



A more convenient form of displaying values read from an analog sensor can be an LCD. In this case, the content of the top line is printed only once, at the beginning of the programme, and only the content displayed in the second line is replaced in the loop:

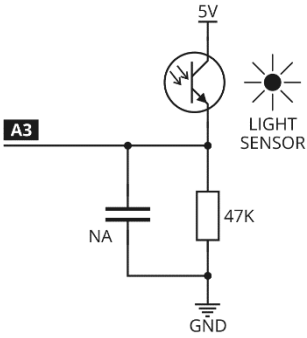
```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// information about the LCD connection
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
  // lcd configuration
  lcd.begin(16, 2);

  // set the cursor to position
  lcd.setCursor(0, 0);
  // print text on lcd
  lcd.print("POT:");
}
```

```
void loop() {
  // read sensor data
  int pot = analogRead(A1);

  // set the cursor to position
  lcd.setCursor(0, 1);
  // print text on lcd
  lcd.print(pot);

  // wait for 150 ms
  delay(150);

  // clear second line of LCD
  lcd.setCursor(0, 1);
  lcd.print("    ");
}
```

After uploading this programme, any change of the potentiometer position should be immediately visible on the display in form of a changing number between 0 and 1023.

Analog sensors - light sensor (A3)

Another analog sensor placed on the board is the **KPS-3227 light sensor**, which by default is connected to the A3 input. By testing the voltage at the output of this sensor, you can measure the amount of light falling on the board - the brighter the light is, the higher voltage should be.



Example displaying a value on the display:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("LIGHT:");
}

void loop() {
  int light = analogRead(A3);    // read sensor data

  // set the cursor to position
  lcd.setCursor(0, 1);
  // print text on lcd
  lcd.print(light);

  // wait for 150 ms
  delay(150);
```

```
  // clear second line of LCD
  lcd.setCursor(0, 1);
  lcd.print("    ");
}
```

The sensor indications may be unstable with some sources of artificial light (e.g. in case of fluorescent lamps).

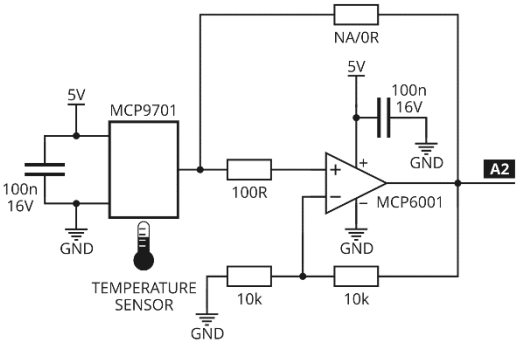
For easier interpretation of the results, you can use the *map* function, which allows you to scale the values. The following code will give you a value between 0 and 100, instead of a value between 0 and 1023, because you can interpret it for example as a percentage level of lighting:

```
lcd.print(map(light, 0, 1023, 0, 100));
```

A list of all Arduino functions is available at:
https://www.arduino.cc/reference/en/

Analog sensors - temperature sensor (A2)

The board has an analog temperature sensor MCP9701 (connected to A2 by default). This sensor can be used identically to a light sensor. In this case, an increase in voltage at its output will mean a rise in temperature.



Voltage reading at the sensor output can also be converted into temperature in degrees Celsius. To do this, following formula can be used: *temp*0.125-22.0*. Sample thermometer programme:

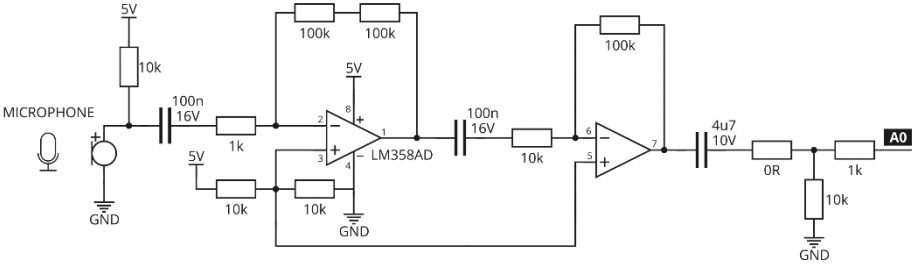
```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("TEMP [C]:");
}


void loop() {
```

```
  // read sensor data
  int temp = analogRead(A2);

  lcd.setCursor(0, 1);

  temp = (int)(temp*0.125 - 22.0);
  lcd.print(temp);

  delay(200);

  lcd.setCursor(0, 1);
  lcd.print("    ");
}
```

A temperature sensor might work incorrectly if exposed to magnetic field.

Analog sensors - microphone (A0)

There is a microphone on the board that has been connected to the Arduino in a way that allows it to be used as a **noise level sensor**. This microphone can be used, for example, to switch diodes on by clapping or signalling too high sound levels in a room.



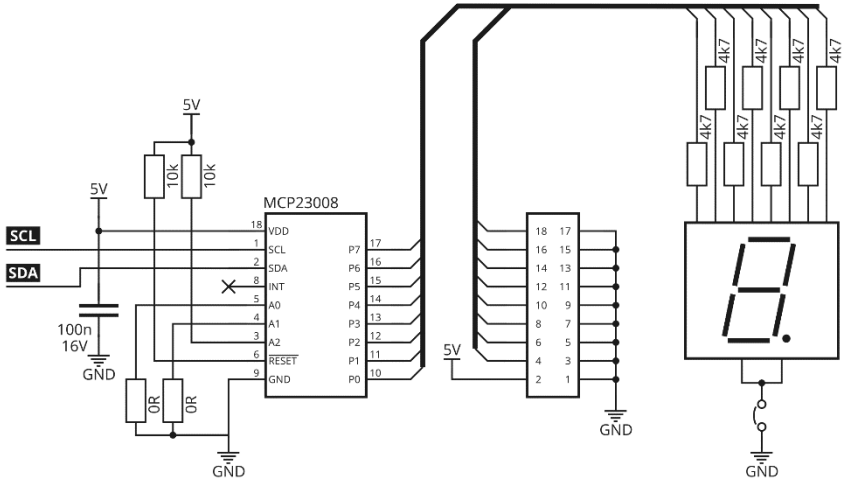
The following code is the simplest form of using this sensor. Arduino regularly checks the voltage level at the input to which the microphone system is connected. This information is shown on the display. If the measured value exceeds 250, the condition will be met and the LED1 will be switched on for a second. The value triggering the diode switching on **was selected experimentally** in such a way that the system responded to the clap.

```
#define LED1 13

#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
  pinMode(LED1, OUTPUT);
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
```

```
  lcd.print("MIC:");
}

void loop() {
  // read sensor data
  int mic = analogRead(A0);

  lcd.setCursor(0, 1);
  lcd.print(mic);
  if (mic > 250) {   // if a clap was detected
    // turn the LED on
    digitalWrite(LED1, HIGH);
    delay(1000);
  } else {  // if not
    // turn the LED off
    digitalWrite(LED1, LOW);
  }

  delay(150);

  lcd.setCursor(0, 1);
  lcd.print("    ");
}
```

Expander and 7-segment display

There is a 7-segment display with a common anode in the top right-hand corner of the board, which was connected to Arduino via the I2C expander (0x24 address). The use of different addresses allows simultaneous use of many components communicating with the same data lines. This allows simultaneous use of text LCD for example.



There is a jumper described as "ON / OFF" below the display. Using this jumper it is possible to disconnect the common anode from the mass of the system, which will completely turn the display off. This jumper is useful in situations where we want to use the expander pins in a different way. You can then remove the jumper, turn off the display and use the expander pins led out to the double connector next to the display.

You should start using the expander from adding a library and declaring a new object (here named as *seg7*), which will allow you to control individual pins:

```
#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;
```

We initialize the expander and set all its pins in as inputs. The names of functions and how they are used are similar to those used to control the pins built into Arduino:

```
  seg7.begin(0x4);

  seg7.pinMode(0, OUTPUT);
  seg7.pinMode(1, OUTPUT);
  seg7.pinMode(2, OUTPUT);
  seg7.pinMode(3, OUTPUT);
  seg7.pinMode(4, OUTPUT);
  seg7.pinMode(5, OUTPUT);
  seg7.pinMode(6, OUTPUT);
  seg7.pinMode(7, OUTPUT);
```

Address of the expander responsible for the 7-segment display is 0x24, however the library used here only requires providing the offset value above address 0x20 as an address. Therefore, we only pass the address **in form of 0x4** to the function. (However, this method is specific only to this library, more information on this subject can be found in the *Adafruit_MCP23008* library documentation.)

From now on, after the configuration of the display, it will be possible to control all the LEDs that are part of it. Turning on the diode is as follows: *seg7.digitalWrite(pin_number, HIGH)*. An example of a programme flashing with two elements of the display:

```
#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;

void setup() {
  seg7.begin(0x4);   // expander pin configuration

  seg7.pinMode(0, OUTPUT);
  seg7.pinMode(1, OUTPUT);
  seg7.pinMode(2, OUTPUT);
  seg7.pinMode(3, OUTPUT);
  seg7.pinMode(4, OUTPUT);
  seg7.pinMode(5, OUTPUT);
  seg7.pinMode(6, OUTPUT);
  seg7.pinMode(7, OUTPUT);
}
```

```
void loop() {
  seg7.digitalWrite(5, HIGH);
  seg7.digitalWrite(2, LOW);
  delay(250);

  seg7.digitalWrite(5, LOW);
  seg7.digitalWrite(2, HIGH);
  delay(250);
}
```

Expander and 7-segment display (digits)

Usually, this type of display is used to display numbers from 0 to 9. To make setting such values easier, it is worth adding an array with 10 elements to the programme. Each element of the array is a binary number, which represents information on the switching on (1) or off (0) of each diode.

```
uint8 t digit[10] = {
  B00111111, // "0"
  B00000110, // "1"
  B01011011, // "2"
  B01001111, // "3"
  B01100110, // "4"
  B01101101, // "5"
  B01111101, // "6"
  B00000111, // "7"
  B01111111, // "8"
  B01101111, // "9"
};
```

Thanks to this, using the new *seg7.writeGPIO()* function it will be possible to display any digit using a single line of code. The following demonstration programme displays all values (0-9) in a loop:

```
#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;

uint8_t digit[10] = {
  B00111111, // "0"
  B00000110, // "1"
  B01011011, // "2"
  B01001111, // "3"
  B01100110, // "4"
  B01101101, // "5"
  B01111101, // "6"
  B00000111, // "7"
  B01111111, // "8"
  B01101111, // "9"
};

void setup() {
  // expander pin configuration
  seg7.begin(0x4);
  seg7.pinMode(0, OUTPUT);
```

```
  seg7.pinMode(1, OUTPUT);
  seg7.pinMode(2, OUTPUT);
  seg7.pinMode(3, OUTPUT);
  seg7.pinMode(4, OUTPUT);
  seg7.pinMode(5, OUTPUT);
  seg7.pinMode(6, OUTPUT);
  seg7.pinMode(7, OUTPUT);
}

void loop() {
  for (int i=0; i<10; i++) {
    seg7.writeGPIO(digit[i]);
    delay(250);
  }
}
```

For loop

In the example above, we used the *for* loop. By using the loop, we can execute the same instructions multiple times (without the need for re-typing them), and therefore we can shorten the notation. In this example, the loop will execute 10 times (from 0 to 9). Each time, a value from the digit array will be displayed on the screen, indicating the current cycle of the loop. This is possible because the value of variable "*i*" will be incremented (increased) after each execution of the instructions in the *for* loop. After the defined number of executions, the program will proceed to further instructions (in this case, it will go back to the beginning and restart execution of instructions in the *for* loop).

While loop

Another equally common loop in Arduino is the *while* loop, which executes itself until the condition in the bracket is met. The for loop described above can be replaced with the new *while* loop in the following manner:

```
int i = 0;
while(i < 10){
  seg7.writeGPIO(digit[i]);
  delay(250);
  i++;
}
```

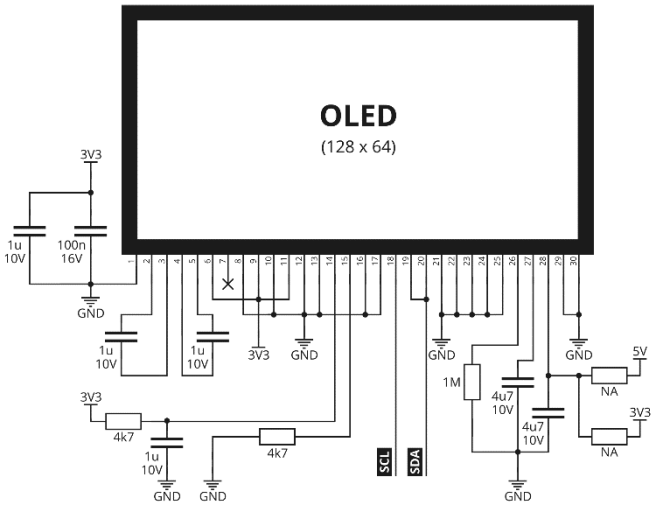
At the beginning, a counter variable is created (here, it is the "*i*" variable). Right after while in the bracket, a condition is added, which makes the loop execute itself until the value of the "*i*" variable is less than 10. The loop contains instructions to be performed several times. At the end of the loop, an increment operator is added (*i*++), i.e. the increase of the variable by 1. Thanks to that, after the 10th cycle the condition is not met, and the program will exit the loop.

Loops of this type are also used for creating infinite loops. In such cases, there's no need for adding a condition – you can simply use "*1*", which means that the condition will always be true:

```
while(1){
  //infinite loop
}
```

OLED graphic display

Another element connected to the I2C interface is the OLED graphic display (address 0x3C), which is next to the LEDs. The display used on the board **has the popular SSD1306 driver**.



In order to use the display, you need to add the required library to the project:

```
#include <Adafruit_SSD1306.h>
```

And create a display object:

```
Adafruit_SSD1306 display(NULL);
```

Then, inside the *setup* function configuration of OLED is necessary, which means its initialization (with entering the address 0x3C), waiting for a short time for the controller initialization, clearing the display content and setting the colour to white (the only possible one):

```
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  display.clearDisplay();
  display.setTextColor(WHITE);
```

Displaying the text on the screen consists in selecting the cursor position (set in pixels), entering the text, and then "sending" all information to the screen.

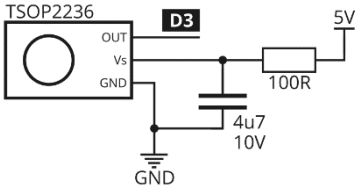
The following programme first displays two test lines (one under the other with 50 pixels shift to the right). Then, slides the text *"TEST LINE 12345"* diagonally through the screen:

```
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(NULL);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  display.clearDisplay();
```

```
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("TEST LINE 1");
  display.setCursor(50,10);
  display.println("TEST LINE 2");
  display.display();

  delay(2000);
}

void loop() {
  for (int i = 0; i < 8; i++) {
     display.clearDisplay();
     display.setCursor(5*i,8*i);
     display.println("TEST LINE 12345");
     display.display();
     delay(250);
  }
}
```

IR receiver (D3)

The board is equipped with an **integrated infrared receiver TSOP2236**, which can be used to receive information sent by IR remotes working in the RC5 standard. This method of communication was developed by Philips over 30 years ago for remote control of consumer electronics and is still very popular.



The remote control working in the RC5 standard sends an infrared beam with a frequency of 36 kHz. Each time, **14 bits of data are sent**, which create a data frame. If the button on the remote control is pressed, data frames are transmitted all the time (every ~114 ms).

The use of a sensor connected to the D3 pin should be started from adding a library, creating an object and declaring a new structure to store the received information:

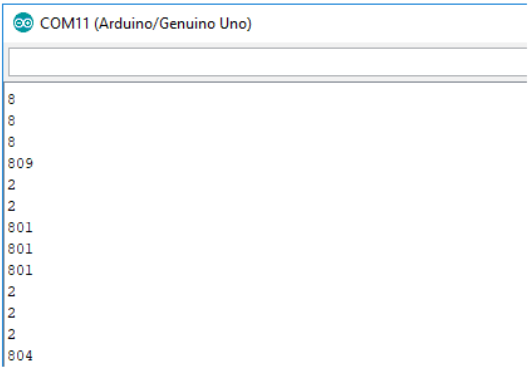
```
#include <IRremote.h>

IRrecv irrecv(3);
decode_results results;
```

Sample demo programme that decodes signals from the remote and sends them to a computer via UART is shown below:

```
#include <IRremote.h>

IRrecv irrecv(3);
decode results results;
```

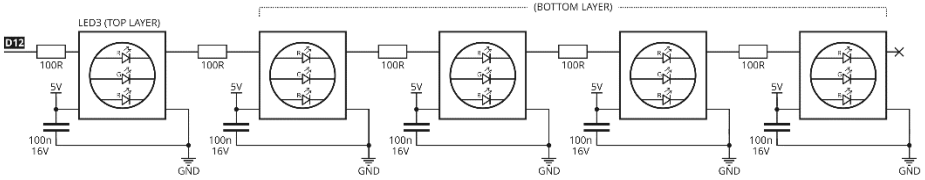
```
void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn();    // start IR receiver
}

void loop() {
  if (irrecv.decode(&results)) { // if data detected
    // send data via UART to PC in HEX format
    Serial.println(results.value, HEX);
    irrecv.resume();    // resume receiver
  }
  delay(100);
}
```

As a result of its work decoded codes should appear in the *Serial Port Monitor*, which are sent by the IR remote. Remote codes may vary, depending on the particular model. Some of the remotes also send other codes depending on whether the button is pressed for the first or for the next time.



Support of digitally controlled RGB LEDs (D12)

There are 5 digitally controlled RGB LEDs on the board. All these diodes are connected in series to one Arduino output (D12). **WS2812B** diodes have a built-in controller, so that different colour can be assigned to each diode.The LEDs are numbered from 0 to 4. The LED number 0 is next to the OLED display, the remaining 4 LEDs have been placed on the other side of the board (in the corners), thanks to which they can illuminate the board from the bottom. Diodes should be turned on beginning from adding a library and configuring an object that contains information about them.



The following entry is a declaration of 5 LEDs connected to pin 12:

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(5, 12, NEO_GRB + NEO_KHZ800);
```

To enable each colour of LED number 0, use the *setPixelColor* function:

```
  pixels.begin();

  pixels.setPixelColor(0, pixels.Color(10,20,30));
  pixels.show();
```

This code means that the first diode in the series (No. 0) will be set to red with intensity 10/255, green with intensity 20/255 and blue with intensity 30/255. After assigning colours, simply call the *show* function, which will send information to the LEDs.

The following test programme runs all the colours of the LED next to the OLED display and then alternately blinks with 4 LEDs on the bottom of the board (in blue and red):

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(5, 12, NEO_GRB + NEO_KHZ800);

void setup() {
  pixels.begin();

  pixels.setPixelColor(0, pixels.Color(10,20,30));
  pixels.show();
}

void loop() {
  pixels.setPixelColor(1, pixels.Color(0,0,255));
  pixels.setPixelColor(2, pixels.Color(255,0,0));
  pixels.setPixelColor(3, pixels.Color(0,0,255));
  pixels.setPixelColor(4, pixels.Color(255,0,0));
  pixels.show();
  delay(250);
  pixels.setPixelColor(1, pixels.Color(255,0,0));
  pixels.setPixelColor(2, pixels.Color(0,0,255));
  pixels.setPixelColor(3, pixels.Color(255,0,0));
  pixels.setPixelColor(4, pixels.Color(0,0,255));
  pixels.show();
  delay(250);
}
```

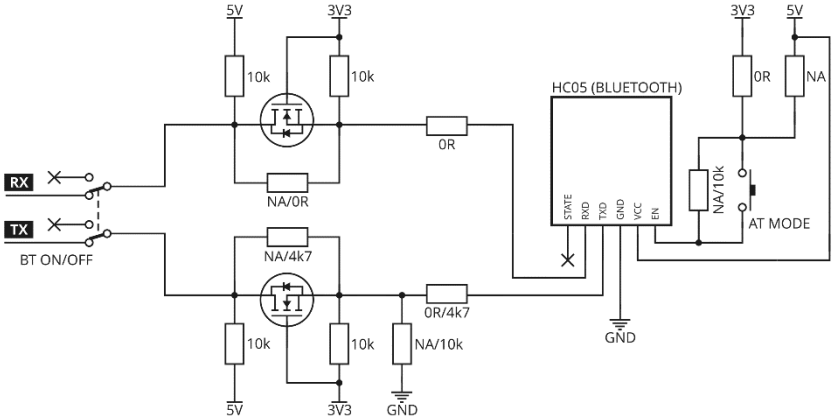
In case of a problem with turning on the diodes, make sure that after setting the colours, function *show* was called, which is responsible for sending information to the controllers.

Bluetooth module

The board has a space for attaching an optional Bluetooth module, e.g. HC-05. The angled 6-pin connector for its connection is located to the right of the jumper strip. The following signals have been led out there:

- power supply for the BT module,
- 2 pins controlling the module,
- 2 pins for data transmission (UART).

The connection was made using a logical level converter:



Bluetooth module can be used, for example, to communicate the set with mobile phones running Android operating system. Communication is performed via the UART interface - the same one which is used for communication with a PC. Therefore, it is possible to use only one of these two ways of communication at the same time.

To verify the operation of the module, the following test code can be used. It works by sending the text "TME BT" (every 250 ms). At the same time, the system receives all data sent to it from the phone. If the "*TEST*" string is received, the system will respond by sending "*TME BT TEST OK*" and will stop sending new information for 1 second.

```
void setup() {
  Serial.begin(115200);
}

String data = "";
void loop() {
  if(Serial.available() > 0) { // if data received

    // read string until new line char
    data = Serial.readStringUntil('\n');
```

```
    // if there is "TEST" string received
    if (data == "TEST") {
      // send back "TME BT TEST OK"
      Serial.println("TME BT TEST OK");
      delay(1000);
    }
  }

  Serial.println("TME BT");
  delay(250);
}
```

After uploading the programme, slide switch should be set to the BT ON position. Set with the BT module should be detected by the phone as a new Bluetooth device that can be paired using the default pin code: "1234". After pairing, application in the mobile phone should display messages sent by the board.

Arduino shields connectors

There are sockets in the top right-hand corner of the board that allow you to connect expansion boards in Arduino Shield standard. However, you need to remember that before connecting any additional module, **you have to disconnect all pins that are used by the overlay using the jumpers!**

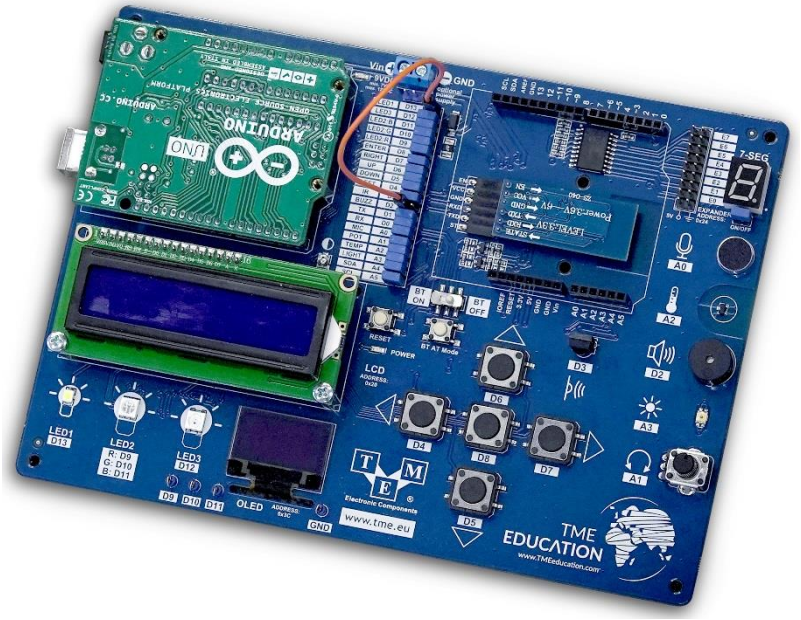
In an extreme situation, the conflict of used signals
may damage the Arduino, an educational board or an additional module.

Configuration jumpers

All peripherals on the board are connected to Arduino by default. However, it is possible to change the configuration very easily. By removing jumpers, you can **disconnect the peripherals from the controller**. If it is necessary, you can **change the assignment of peripherals** to specific pins using an additional connection cable (not included).

Changing the configuration of connections using jumpers can be particularly convenient when it is necessary to connect certain peripheral of the board to another pin, e.g.: connecting the LED1 diode to the pin on which hardware PWM signal generation is possible.

The following example shows how you can disconnect the buzzer and LED1 and change the configuration in such a way that the buzzer is controlled by pin D13 (instead of D2).



List of Arduino functions

Below you can find the abridged list of all the features with their short description included. Its full version with detailed descriptions can be found on https://www.arduino.cc/reference/en/.

digitalRead(*pin*) – digital input status check.
pin – lead on which the condition is to be checked
Returned value: HIGH (for high state) or LOW (for low state)

digitalWrite(*pin, state*) – setting digital output state.
pin – lead on which state is to be set
state – state to be set (LOW or HIGH)
Returned value: none.

pinMode(*pin, mode*) – configuration of the digital pin operating mode.
pin – lead to be configured
mode – lead working mode:
INPUT – digital input
INPUT_PULLUP – input with pull-up resistor activated
OUTPUT – digital output
Returned value: none.

analogRead(*pin*) – voltage measurement using an ADC converter.
pin – analog lead (A0-A5) on which the measurement is to be made
Returned value: a number between 0 and 1023 (the higher the input voltage, the higher the value returned by the function).

analogWrite(*pin, value*) – setting PWM signal duty cycle on the specified lead.
pin – lead on which the PWM signal should be generated
value – duty cycle value from 0 to 255, where 0 is 0% of the PWM signal duty cycle, and 255 is 100% of the PWM signal duty cycle.
Returned value: none.

delay(*ms*) – function that pauses the operation of the program for a specified number of milliseconds (1 second is 1000 milliseconds). Function operation completely holds the program!
ms – duration of the delay / pause of the program in milliseconds, maximum value is 4 294 967 295 (size of the unsigned long variable).
Returned value: none.

delayMicroseconds(*us*) – function that pauses the operation of the program for a specified number of milliseconds (1 second is 1000 milliseconds). Function operation completely holds the program!
us – duration of the delay / pause of the program in microseconds, maximum value is 16 383.
Returned value: none.

abs(*x*) – function that returns the absolute value, for the correct operation of the function you should avoid using other functions inside of the arguments.
x – number of which the absolute value is to be calculated.
Returned value: absolute value from the number x.

constrain(*x, a, b*) – function that checks if the number is in the given range.
x – tested number (of any type)
a – lower range value
b – upper range value
Returned value:
x – if the number is greater than a and smaller than b.
a - if the number x is smaller than a
b – if the number x is greater than b

map(*value, fromLow, fromHigh, toLow, toHigh*) – function that scales a number from one range to the other. It is useful, for example, for rescaling the value measured by the ADC converter (0 ... 1023) to a percentage value (0 ... 100%).
value – value to be scaled
fromLow – lower range of the current range
fromHigh – upper range of the current range
toLow – lower range of the future range
toHigh – upper range of the future range
Returned value: scaled number.

max(*x, y*) – function that compares two numbers.
x – first number
y – second number
Returned value: larger of the numbers

min(*x, y*) – function that compares two numbers.
x – first number
y – second number
Returned value: smaller of the numbers

pow(*base, exponent*) – function calculating power.
base – number to be raised to power
exponent – value of the power
Returned value: base number raised to exponent power.

sqrt(*x*) – square root function.
x – number
Returned value: square root of the number x.

sq(*x*) – square function.
x – number
Returned value: x raised to the power of 2.

cos(*rad*) – function calculating cosine of an angle.
rad – angle in Radians
Returned value: cosine of the given angle.

sin(*rad*) – function calculating sine of an angle.
rad – angle in Radians
Returned value: sine of the given angle.

tan(*rad*) – function calculating tangent of an angle.
rad – angle in Radians
Returned value: tangent of the given angle.

millis() – function that returns the number of milliseconds since Arduino was started. The value is reset after about 50 days.
Returned value: time since start in milliseconds (variable of unsigned long type)

pulseIn(*pin, value, timeout*) – function calculating duration of the pulse on a given input. For example, measurement starts when state changes from low to high and stops when the low state reappears. The function works properly for pulses from 10 microseconds to 3 minutes.
pin – output on which the pulse is to be measured
value – pulse type to be measured (LOW or HIGH)
timeout – optional number of microseconds after which the system will stop measuring if the state on the pin does not change before
Returned value: duration – duration of the pulse in microseconds or 0 if no measurement

isAlpha(*char*) – function checking if the given character is a letter.
char – character to be checked
Returned value: true if the character is a letter

isAlphaNumeric(*char*) – function checking if the given character is a letter or a digit.
char – character to be checked
Returned value: true if the character is a letter or a digit

isDigit(*char*) – function checking if the given character is a digit.
char – character to be checked
Returned value: true if the character is a digit

random(*min, max*) – function that returns a pseudo-random value. The use of the function should be preceded by a single call of the randomSeed() function.
min – start of the range
max – end of the range
Returned value: pseudo-random value from min to (max-1)

randomSeed(*seed*) – function that initializes the pseudorandom number generator. Without the correct use of this function, values returned by the random function will be the same each time Arduino is started.
seed – numerical value that initializes a pseudo-random number generator.
Returned value: none
Example: randomSeed(analogRead (0)) ADC value read from the A0 pin which can be used by sensors connected to the system or may be unused is given as an argument to the function. Random reading of ADC makes that after each start of Arduino, the pseudo-random number generator can return different values in subsequent draws.

byte(*x*) – function converting the variable x.
x – variable to be converted
Returned value: variable of type byte.

char(*x*) – function converting the variable x.
x – variable to be converted
Returned value: char variable.

int(*x*) – function converting the variable x.
x – variable to be converted
Returned value: variable of type int.

long(*x*) – function converting the variable x.
x – variable to be converted
Returned value: variable of type long.

for (*initialization; condition; update*) { } – the *for* loop is used for multiple execution of instructions provided in the body of the function; *initialization* is used only once, and it is usually used to create a counter variable in the loop (e.g. "*int i = 0*"); next, a condition is added to define how long the loop shall be executed (e.g. "*i < 10*"); the final element is the information on what happens with the counter variable after each cycle of the loop (e.g. "*i*++").

while(*condition*) { } – instructions in the while loop will be executed indefinitely as long as the condition in parentheses is true; the condition may be a numeric value or e.g. the status of a connected button or sensor.

Operators:

- % modulo division
- * multiplication
- + addition
- - subtraction
- / division
- = assignment
- != not equals
- < lower
- <= lower or equal
- > greater
- >= greater or equal
- == equal
- ! negation
- && logical AND
- || logical OR

List of libraries with licenses

Extensible hd44780 LCD library – library for hd44780 based LCD display
Project website: www.github.com/duinoWitchery/hd44780
Author: Bill Perry
Licence: GPL

Adafriut_SSD1306 – library for our Monochrome OLEDs based on SSD1306 drivers
Project website: www.github.com/adafruit/Adafruit_SSD1306
Author: firma Adafruit
Licence: BSD

Adafruit_NeoPixel – library for controlling single-wire-based LED pixels
Project website: www.github.com/adafruit/Adafruit_NeoPixel
Author: firma Adafruit
Licence: GPL

Arduino RC5 remote control decoder library – library for decoding infrared remote control commands encoded with the Philips RC5 protocol
Project website: www.github.com/guyc/RC5
Author: Guy Carpenter
Licence: BSD

Adafruit-MCP23008-library – library for the MCP23008 I2C I/O expander
Project website: www.github.com/adafruit/Adafruit-MCP23008-library
Author: firma Adafruit
Licence: BSD