Technical Tutorial for Teachers

Abstract

Contents

List of Figures

List of Tables

1 Arduino Board

Arduino is an open-source electronic platform, which heart is an 8-bit microcontroller Atmel AVR. This platform was created as a simple and inexpensive tool for prototyping for students without the electronic and programming background. By its simplicity, it gained supporters around the world, especially in so-called makers. Due to the growing demand Arduino began to develop by adding new versions of boards dedicated to various purposes, e.g. Internet of Things (IoT), medical measurements, construction of 3D printers or embedded systems. The list of Arduino board versions is long but the most common used boards are shown in figure 1.

The Arduino UNO was the first USB Arduino board. It is a reference model for Arduino platform. The heart of this board is the ATmega328P microcontroller. The Arduino UNO is equipped with 14 digital input/output pins and 6 analog inputs. The 6 of 14 digital pins also can be used as PWM outputs [1].

The Arduino Leonardo board is dedicated to projects, when USB communication is needed. Especially, this board can be connected to a computer as a mouse or keyboard. The heart of Arduino Leonardo is the ATmega32u4 microcontroller and it is equipped with 20 digital input/output pins, in which 7 of them can be used as PWM outputs, and 12 analog inputs [1].

The Arduino Mega 2560 is based on the ATmega 2560 microcontroller and it was designed for more complex projects especially for the 3D printers and robotics projects. This board is equipped with 54 inputs/outputs, in which 15 of them can work as PWM outputs, and there are 16 analog inputs.

The Arduino LilyPad is special version dedicated to wearables projects, based on the ATmega328 microcontroller with 9 digital inputs/outputs and 4 analog inputs. This board can be attached to the textile by special conductive thread. The Arduino LilyPad can be washed by hand with a mild detergent after removing the batteries and other power supplies. Then the board should be dried according to the description of this board [1]. The disadvantage of the Arduino LilyPad is lack of stabilized power supply, which the user have to deliver.

The Arduino Nano is the smallest board with dimensions: 18 x 45 mm. The heart of this board is the ATmega328 and it is equipped with 22 digital inputs/outputs, in which 6 of them can be used as PWM outputs, and there are 8 analog inputs. The size of the board is compatible with breadboards.

The building of the Arduino UNO board is shown in figure 2 and the Arduino Mega 2560 in figure 3.

2 Programming Software

The main programming software dedicated to the Arduino Board is the Arduino integrated development environment (Arduino IDE). The Arduino IDE supports the C++ language but the code must be written according to a specific scheme. Additionally, the Arduino boards can be programmed using block-based programming languages as Snap4Arduino or graphical programming languages as LabVIEW.

2.1 Snap4Arduino

The Snap4Arduino is modified version of the Snap! visual programming language. The software can be downloaded from page http://snap4arduino.rocks/. The Snap4Arduino

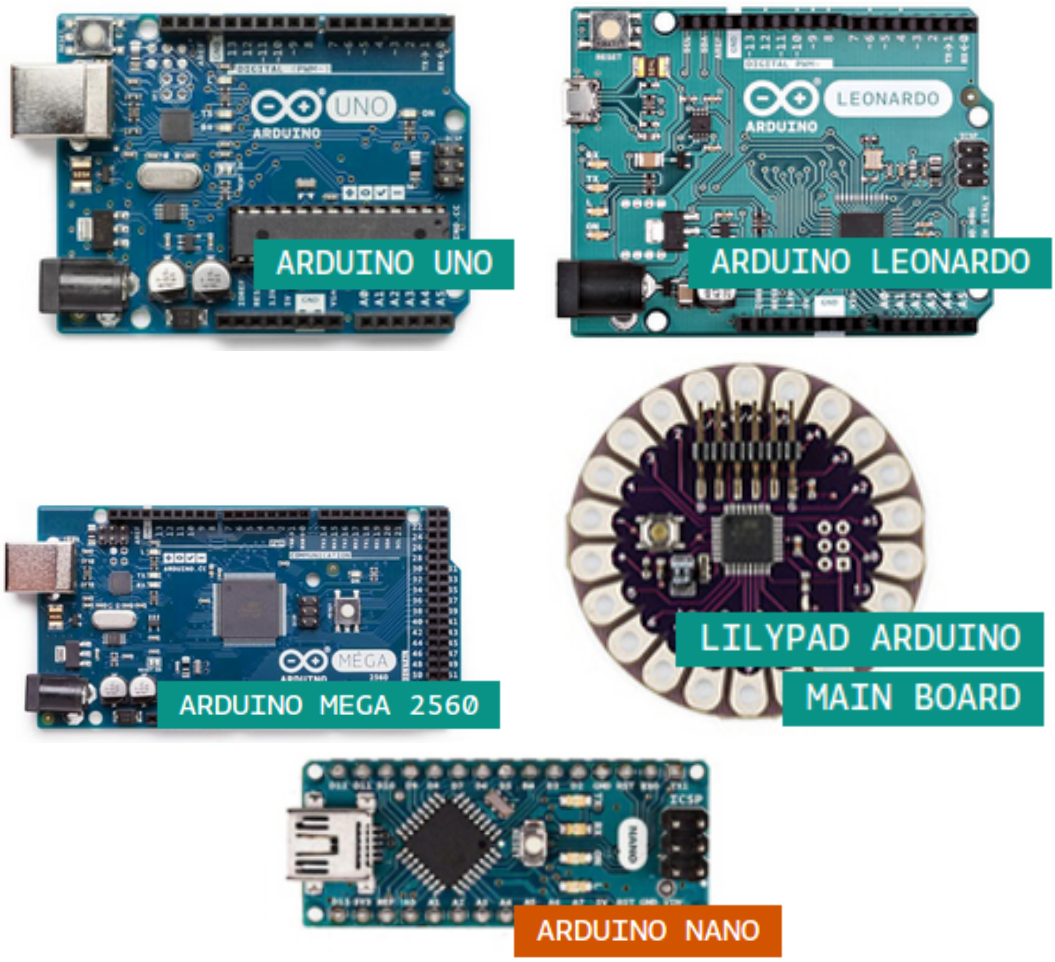


Figure 1: The most popular Arduino Boards: Uno, Leonardo, Mega 2560, LilyPad and Nano.

can be used also on the website but only on Chrome browser with special plugin, which can be download from page http://snap4arduino.rocks/.

Before running the Snap4Arduino, the StandardFirmata should be installed on the Arduino board. Please follow instructions below to install StandardFirmata [2]:

1. Download and install the Arduino IDE environment from page https://www.arduino.cc.
2. Open the Arduino IDE and choose: *File* → *Examples* → *Firmata* → *StandardFirmata*.
3. Connect your board to an USB port in your computer.
4. In the *Tools* menu, select the board version and the serial port where the board is connected.
5. Verify and upload the code.

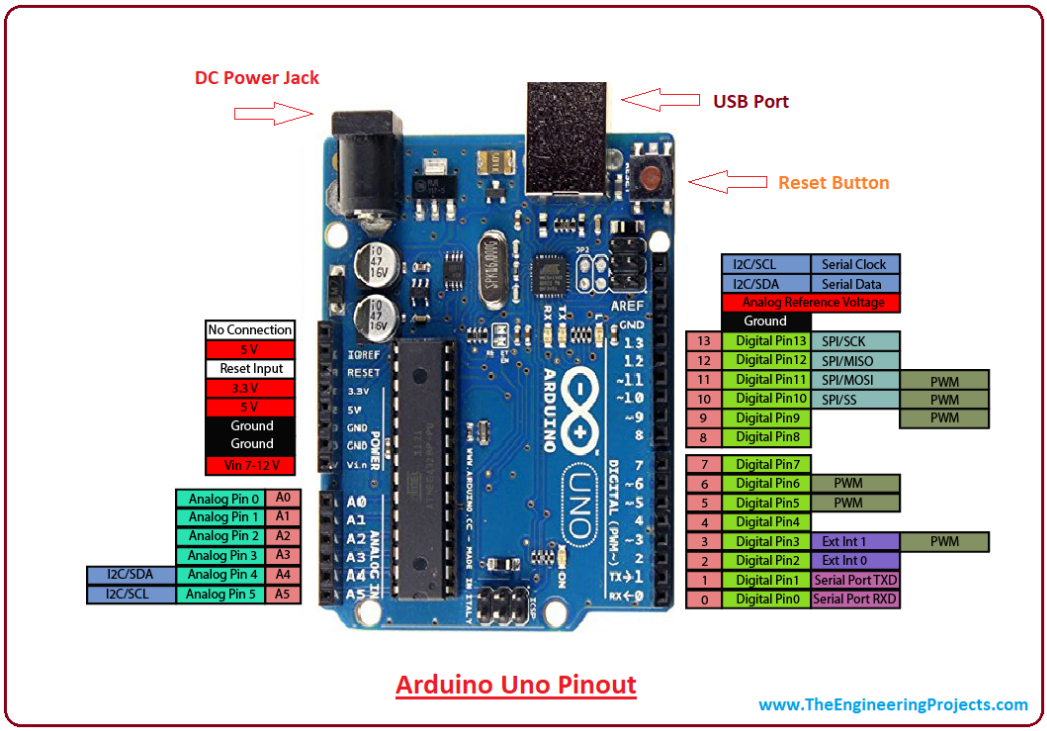


Figure 2: The Arduino UNO pinout [3].

6. Open the Snap4Arduino.

After opening the Snap4Arduino software, choose the *Arduino* section on top left corner as is shown in figure 4. Then click on *Connect Arduino* button.

2.2 Arduino IDE

The Arduino integrated development environment (Arduino IDE) is dedicated software for Arduino boards, which can be download from page https://www.arduino.cc. The Arduino IDE window is shown in figure 5. After running the software, open the *Tools* section and choose board and port from the list.

All the programs always consist of two basic functions:

- *setup* - run only ones at the beginning of the program.
- *loop* - run constantly. All instructions put here will be repeated many times.

Users can extend programs with own functions but these two are mandatory.

The Arduino IDE allows to run examples, which are helpful to understand how the libraries work. Such example can be opened after clicking on *File→Examples* and then the user will see the list of libraries and examples for them.

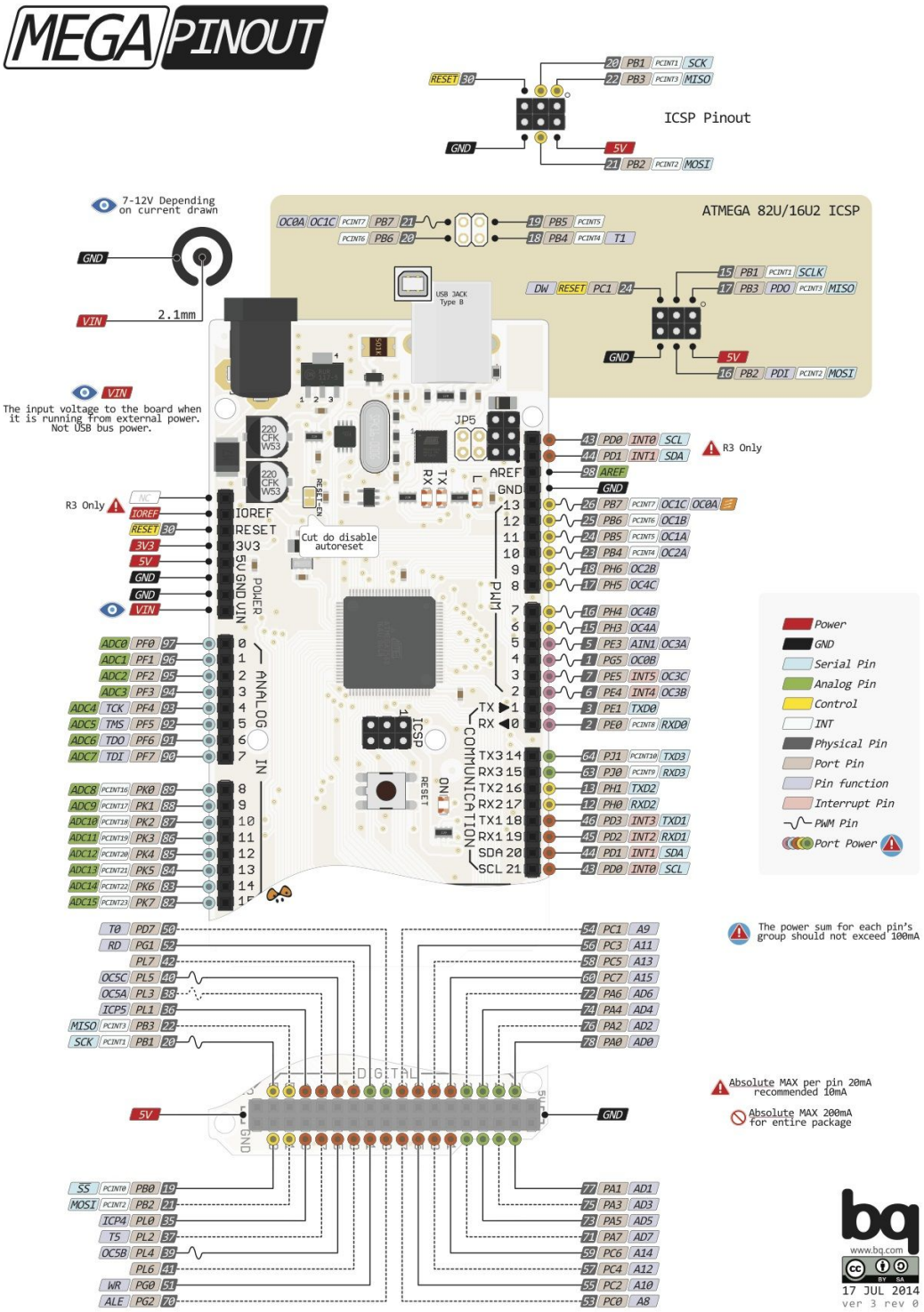


Figure 3: The Arduino Mega 2560 pinout [4].

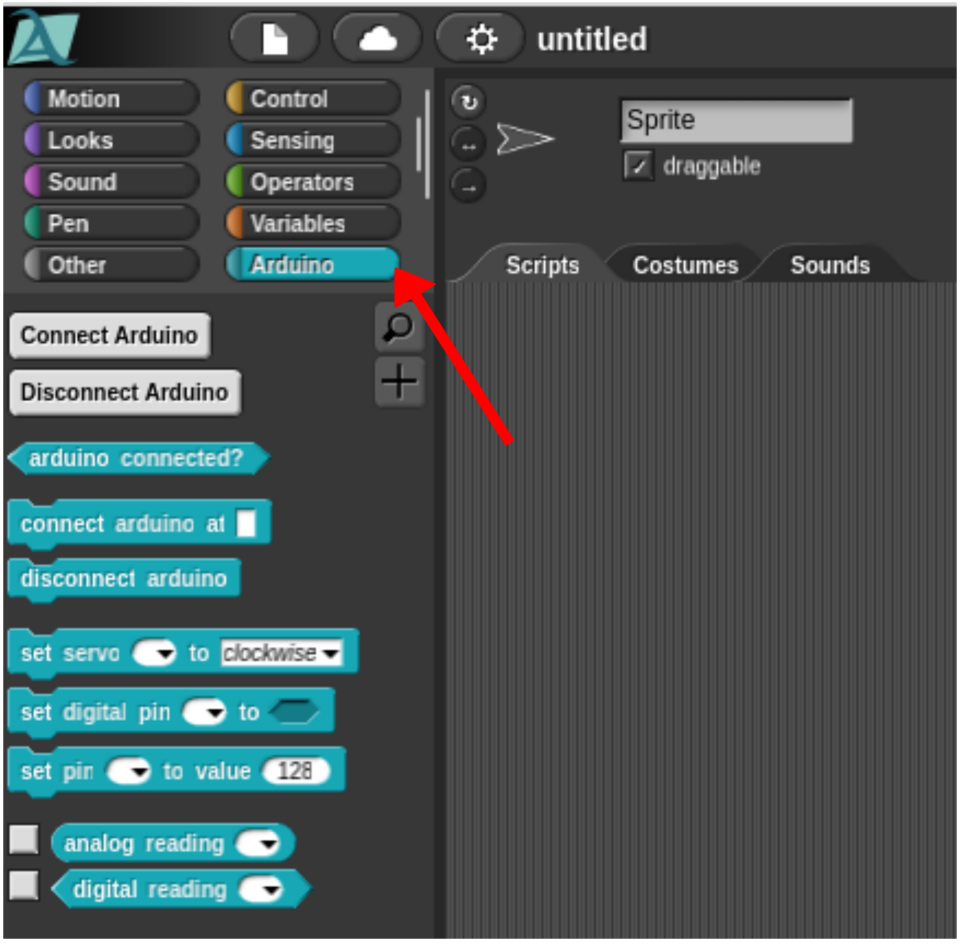


Figure 4: The connection with the Arduino board on the Snap4Arduino software.

3 Electronic circuit design framework

The electronic circuit should be designed before it will be assembled. There are many programs, which will assist in this process but the simplest ones are: TinkerCad Circuits and Fritzing. The first one is an online software and the second one is a desktop application. Both programs are similar and useful for beginners.

3.1 TinkerCad Circuits

TinkerCad Circuits is a free online software, which help to design the electronic circuits and simulate them. The software can be found on website https://www.tinkercad.com/circuits.

An example of simple electronic circuit is shown in figure 6, in which the LED is connected to the Arduino Uno board through a resistor. The resistor is used to limit the LED current. The typical LED voltage drop is about 1.7 - 2.0 V (U_{LED}) and current should not exceed 20 mA. The Arduino boards generate the 5 V (U_{DO}) on the digital outputs so that resistor value should be between (220;3300) Ω according to equation 2. In this example, the resistor has 330 Ω value.

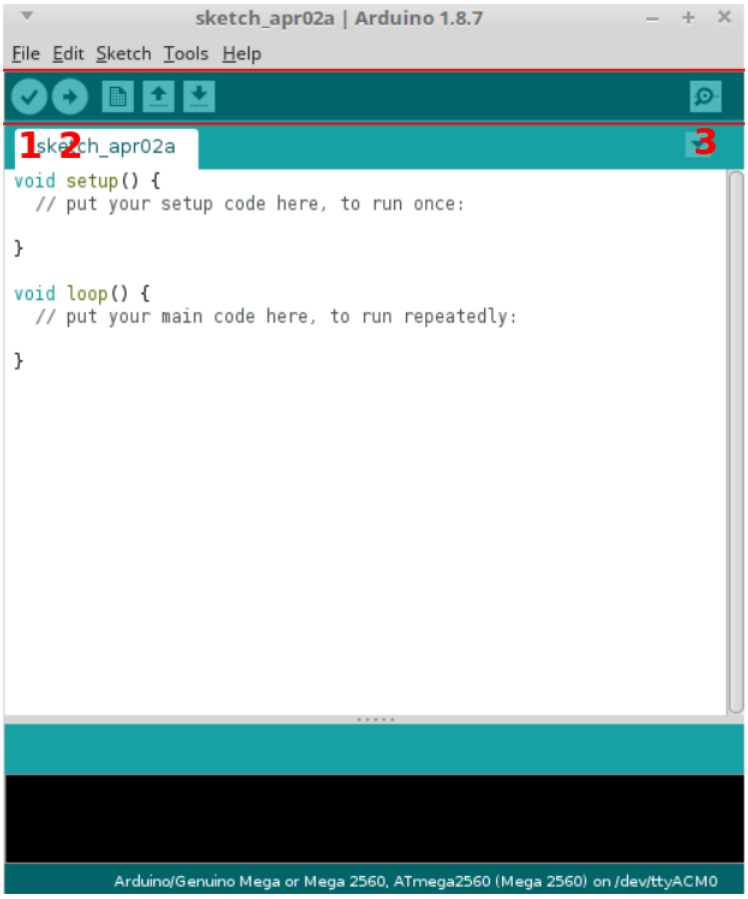


Figure 5: The Arduino IDE window, where 1 - button to verify the code, 2 - button to upload the code to Arduino board and 3 - button to open the Serial Monitor.

$$R = \frac{U_{DO} - U_{LED}}{I} \tag{1}$$

The all possible components are located on the right panel on the software window like:

- General elements: resistors, capacitors, diodes or inductors.
- Input elements: buttons, potentiometers, photoresistors, IR sensors, ultrasonic distance sensors, PIR sensors, keypads etc.
- Outputs elements: LEDs, DC motors, servos, piezos, LCD displays etc.
- Power supply: batteries.
- Breadboards.
- Microcontrollers: Arduino Uno only.
- Instruments: multimeters, oscilloscopes etc.

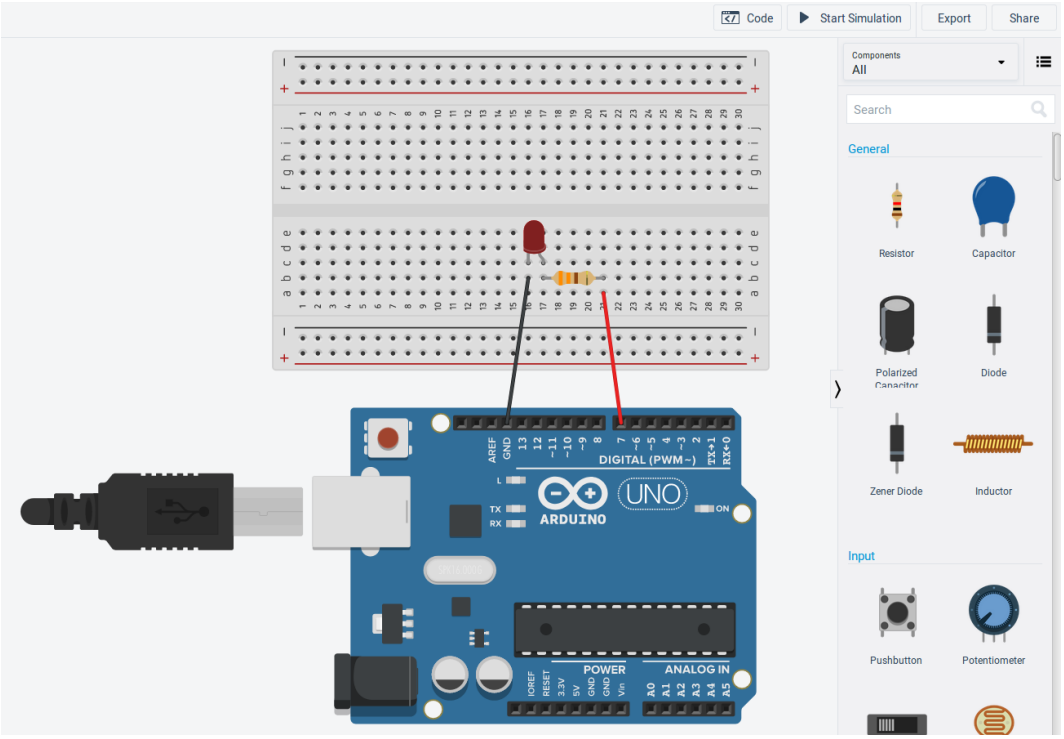


Figure 6: The simple example of circuit creation done by using TinkerCad Circuit software.

The TinkerCad Circuits allows to simulate the electronic circuits by using the simple block language. The code section will open after clicking on *Code* button on the right top corner of website. In the figure 7 there is an example code block shown, which simulate the LED blinking for every 200 ms. The LED is turn on/off by using set pin function, in which user should fill in the pin number and the value, which will be set on the pin. The code consists of 4 lines:

1. The LED is turn on because HIGH value is passed to set pin function. The pin number is 7 because the LED is connected to this pin.
2. The wait function is run so that the LED will shine for 200 ms.
3. The LED is turn off because LOW value is passed to set pin function.
4. The wait function is run so that the LED will not shine for 200 ms.

An advantage of TinkerCad Circuit is simplicity and simulation option. However, this software is an online tool, so that it is not possible to use it without internet access. In such situation, the user can use the Fritzing software, which works similar to TinkerCad Circuit.

3.2 Fritzing

Fritzing is an open-source software, which can be used to design electronic circuits. The software can be found on website: `http://fritzing.org/`.

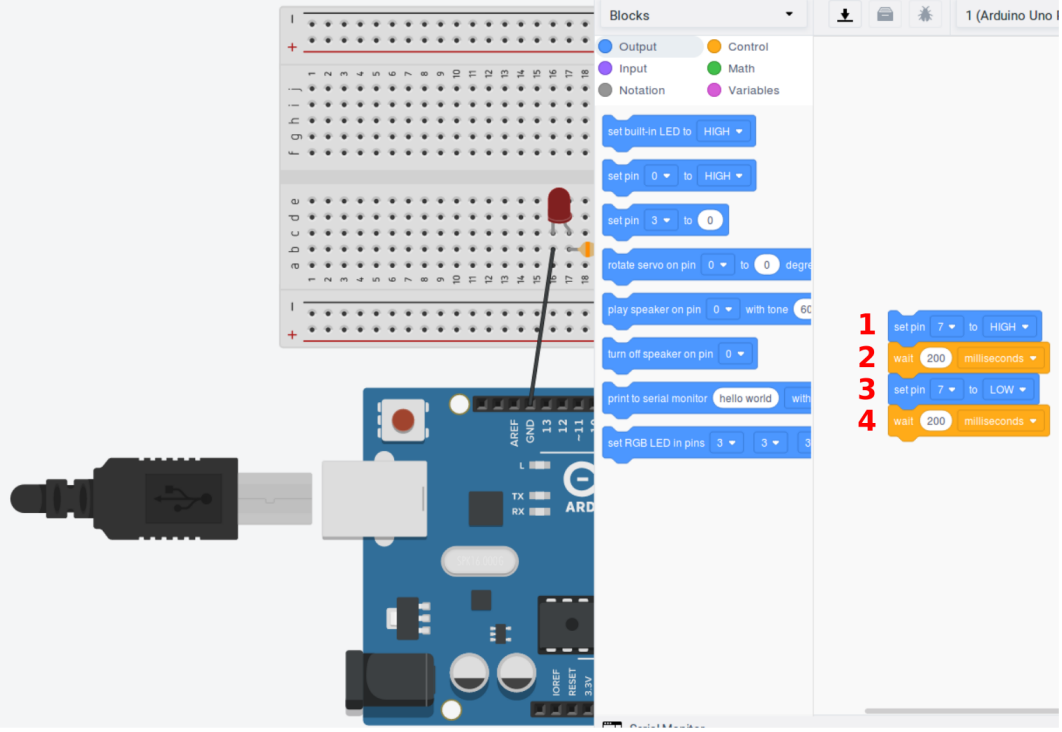


Figure 7: The simple example of code in TinkerCad Circuit, which simulates the LED blinking every 200 ms. The red digits numerate the code lines.

Creation of electronic circuit is the same simple as in TinkerCad Circuit. Right panel contains part images, which user can drag and drop onto project. The list of elements is much more wider in comparison with TinkerCad Circuits.User can use:

- the most popular Arduino boards like: UNO, Mega 2560, Leonardo, Yun, Due, Esplora, Mini and Nano.
- the basic elements like: resistors, capacitors, inductors, transistors etc.
- input elements like: potentiometers, encoders, buttons, photocells, sensors etc.
- the output elements like: diodes, LCD displays, piezos, DC motors, servos, stepper motors etc.
- power supply.

Additionally, a large number of elements can be downloaded from GitHub repositories.

The example of simple electronic circuit is shown in figure 8, in which the LED diode is connected to the Arduino Mega 2560 board through a resistor.

The Fritzing allows automatic generation of electronic schematics by clicking on *Schematic* button. In figure 9 there is automatically generated schematic for example used in figure 8.

Additional feature of the Fritzing is a possibility of writing code and sending it directly to the Arduino board. At the beginning, the user should configure the Fritzing by clicking

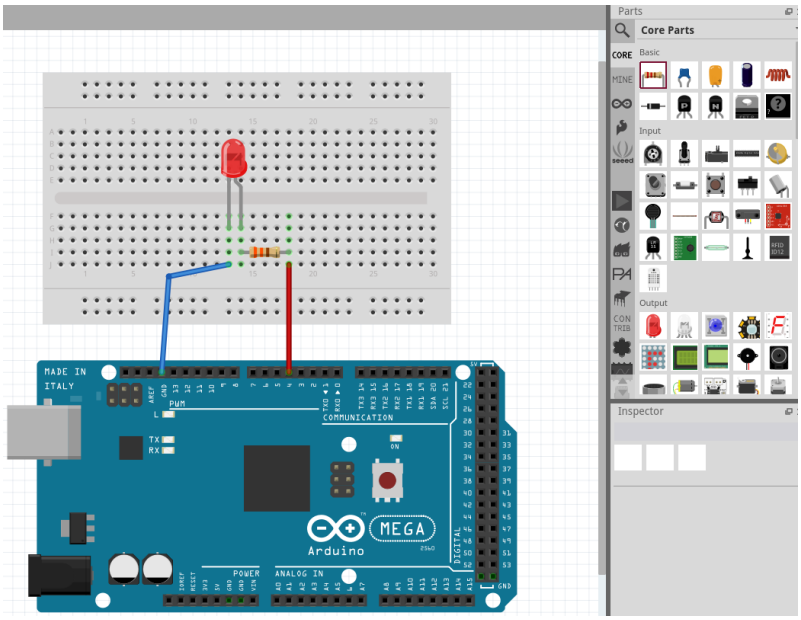


Figure 8: The example of electronic circuit created with Fritzing software.

the *Edit* → *Preferences...* → *Code View* and choose the localization of Arduino IDE as is shown in figure 10.

Next step is pressing the *Code* button. The example code of blinking LED diode every 100 ms is shown in figure 11. The code consists of two functions: *setup* and *loop* and the lines:

1. **pinMode** function determines if the pin should be input or output. The first argument is the pin number. The second argument is the word **INPUT** or **OUTPUT**. The "input" mode of pin should be selected when the signal from other elements will be received by Arduino board e.g. event of button press, etc. The output mode of pin should be selected when the Arduino will drive the logic value on pin, e.g. controlling LED, servo motors, piezo etc. In our case, the pin mode should be **OUTPUT**, because LED will be turned on/off when the **HIGH** or **LOW** logic value will be driven by Arduino on pin. The LED is connected to pin number 4 so that the line looks: *pinMode(4, OUTPUT)*. All instructions should end by semicolon ";". The configuration of pin is done one time at the beginning of program so that this function should be located in *setup* function.

2. **digitalWrite** function is used to set the *HIGH* or *LOW* logic value on selected pin. The first argument is the number of pin and the second is the word *HIGH* or *LOW*. The *HIGH* signal corresponds to the 5 V voltage and *LOW* signal to 0 V voltage. At the beginning, the LED will be turned on, so that the *HIGH* state should be set on pin number 4.

3. **delay** is the function, which stops running of the program for chosen amount of time specified in ms. In this case, the program will wait 100 ms and for this time the LED will shine.

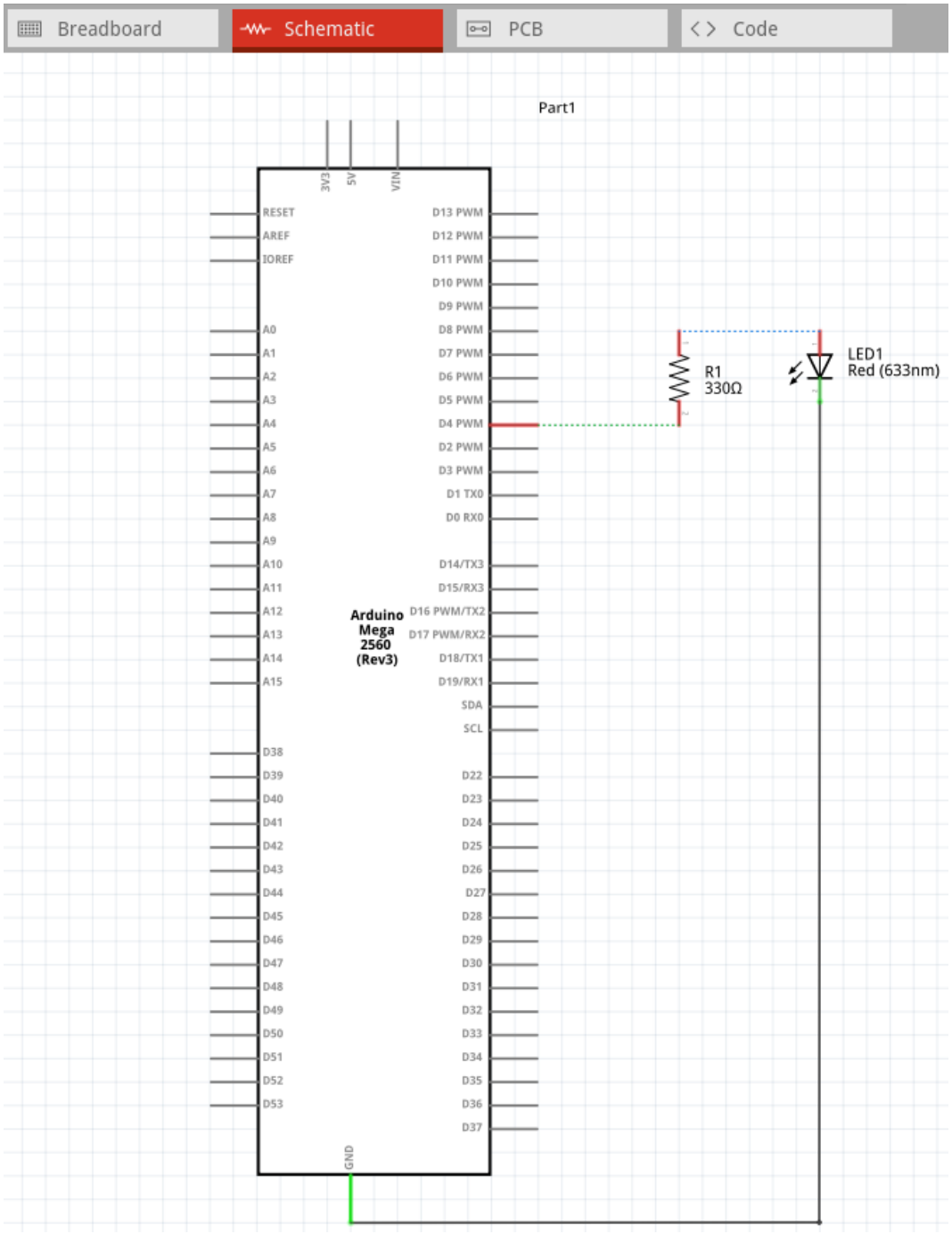


Figure 9: The example of automatically generated schematic by using the Fritzing software.

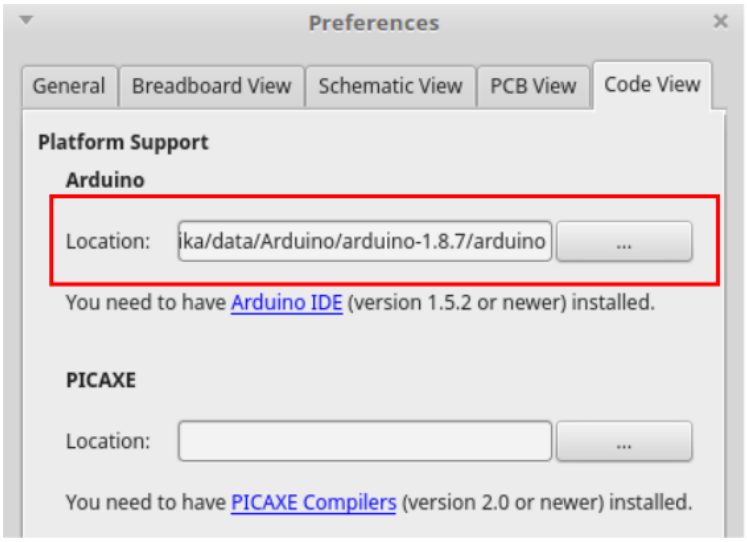


Figure 10: The *Preferences* window.

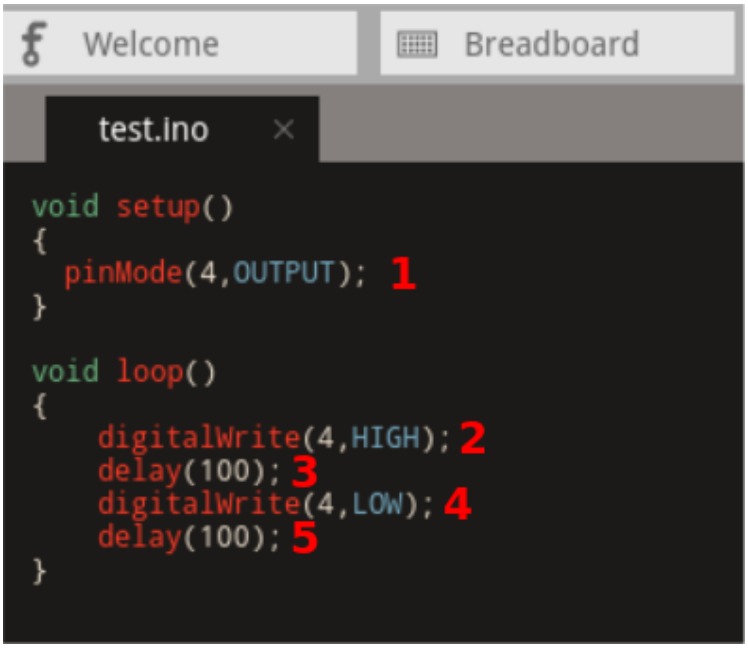


Figure 11: The example of code, which makes the LED blinking every 100 ms, wrote in Fritzing software.

4. Next step is turning off the LED so that the *LOW* state should be set on pin number 4.

5. The last step is waiting for 100 ms (in this case the LED will not shine).

The steps: 2 to 5 should repeat, so we could observe the effect of blinking LED. For that, these lines are put in *loop* function.

Before uploading the code to the Arduino board, the type of board and port should be chosen as it is shown in figure 12. Then, the code can be uploaded to board by clicking

the *Upload* button.

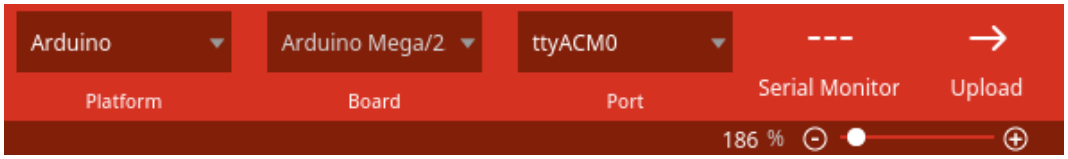


Figure 12: Options, which should be chosen before uploading the code to the Arduino board.

Summarizing, the TinkerCad Circuit and Fritzing are useful and quite similar software for electronic circuit creation. The TinkerCad Circuit is an online software and it advantage is a possibility to simulate the circuit before building it in reality. It helps to verify if circuit will work according to expectations. A disadvantage of TinkerCad Circuits is having only Arduino UNO board and a need for internet access. The Fritzing is a desktop software, which is similar to TinkerCad Circuit and its advantages are possibility to automatic electronic schematics generation and the wide list of elements. Fritzing disadvantage is a lack of circuit simulation option but user can write the code in Fritzing in the same way as in Arduino IDE and upload it to the real Arduino board.

4 Introduction to electricity

Before starting the practical electronic circuit creation, it is important to understand the basic concepts of electricity.

4.1 Basics concepts of electricity

The electric current is an orderly movement of electric charges, which are transferred via charge carriers. The type of charge carries depend on type of materials. In metals, charge carriers are electrons and in liquid and gas materials carriers can be positive ions (cations), negative ions (anions) and electrons.

The electric field that causes the electric current in the conductor, is determined by the potential difference called the voltage, which is expressed in Volts (V). The higher the voltage is, the faster the charge flows (current) through the circuit. The direction of the current flow is from a point with higher potential to a point with lower potential. Ground (GND) describes the point with the lowest potential energy in the circuit [5, 6, 7, 8] .

The electric current is a ratio of electric charge flowing through the cross-section of the conductor to time, expressed in Ampere (A). If the current does not change its direction of flow and its value, the current is called direct current (DC). Current is called alternating current when the electric current changes its direction periodically. Current which comes from electric outlet changes direction 50-60 times per second depending on the country.[5, 6, 7, 8] .

A resistance describes how much materials or components resist the flow of charges. The value of resistance is defined as the ratio of voltage (U) to the current (I) (2). More information in [5, 6, 7, 8] :

$$R = \frac{U}{I} \tag{2}$$

4.2 Basic elements

The basic elements which are needed to design the electronic circuits are described in this chapter.

Resistor

The first element, which was used in chapter number 3 is the **resistor**, which converts part of current to heat energy. This element is used to limit the current flowing through elements, which work for smaller current e.g. LED diode. The higher value of resistor is used, the smaller current will flow through a circuit. This dependence is described by Ohm law (3).

$$I = \frac{U}{R} \tag{3}$$

The example of resistor is shown in figure 13.

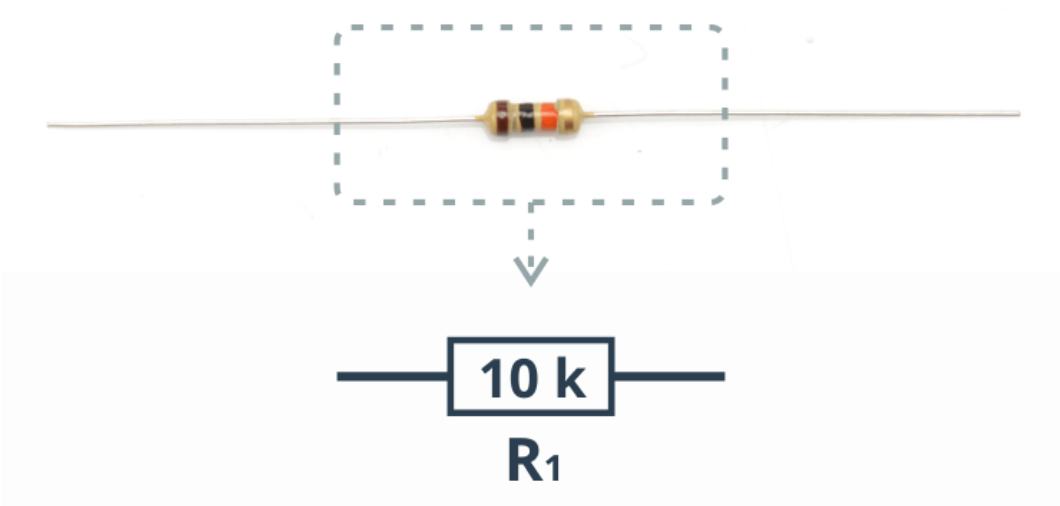


Figure 13: Example of resistor and its schematic symbol. Source: [9].

Color	1st digit	2nd digit	3rd digit (multiplier)	4th digit (tolerance)
black	0	0	1	-
brown	1	1	10	±1%
red	2	2	100	± 2%
orange	3	3	1k	-
yellow	4	4	10 k	-
green	5	5	100 k	±0.5%
blue	6	6	1M	±0.25%
violet	7	7	10M	±0.1%
gray	8	8	-	±0.05%
white	9	9	-	-
gold	-	-	0.1	±5%
silver	-	-	0.01	±10%

Table 1: The table with the description of values of resistor band codes.

The value of resistance is usually shown as color code on through-hole resistors, because the resistors usually are too small to print the value on them. The value of resistance reads from left side to the right side according with the table 1. The first and second band represent the value. The third band is the multiplier, which is ten to the power of color number. The results from two first bars should by multipled by this value. The last band represents the tolerance, in which this resistor was done. The tolerance shows how much the real resistance can be different from the nominal value.

In figure 14 an example resistor and a way of reading the code was shown. The resistance value is: $15 \cdot$ 1k=15 $k\Omega$ and its tolerance is 5%.

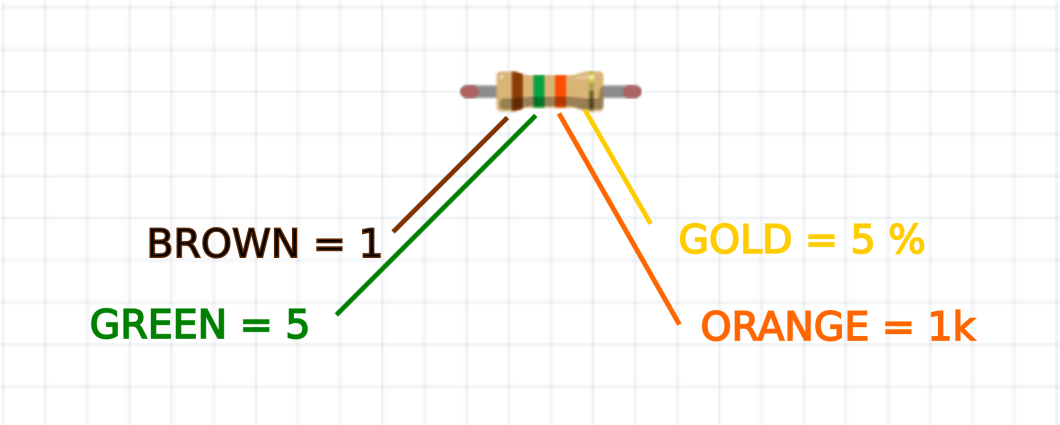


Figure 14: The example of resistance reading from color code.

LED diode

The second element, which was used in chapter 3 is **LED** diode, which converts the electric current to light. The LED diode is a polarized element, what means that current should flow through the LED only in one direction. The anode should be connected to the higher voltage and cathode to lower voltage. The anode is the longer leg of LED diode and cathode is shorter leg (see figure 15).

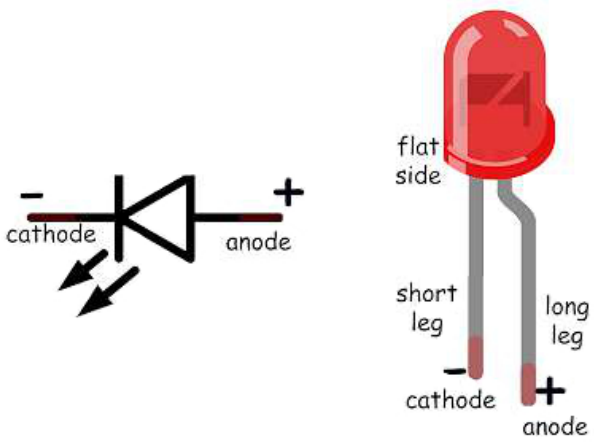


Figure 15: The example of LED diode and description of its legs. Source: [10]

The typical LED diode needs voltage near 1.7 V (U_{LED}) and current between 1 to 15 mA. The Arduino boards generate the 5 V (U_{DO}) on the digital outputs so that resistor should be used to limit the current. The value of resistance should be between (220;3300) Ω according to equation 4.

$$R = \frac{U_{DO} - U_{LED}}{I} \tag{4}$$

Breadboard

The breadboard is a very useful device which helps to connect elements. It should be used at the beginning of electronic circuit creation especially in a phase of circuit behaviour checking. The example of a breadboard is shown in figure 16. The black lines show, which holes are connected. In the middle of breadboard, the vertical holes are connected and horizontal holes are disconnected. A large gap separates two sides of breadboard. At the edge of breadboard, the user can see two lines: red and blue. Sometimes the breadboard does not show color lines. The red line is used to connect the (+) of power and all horizontal holes are connected along red line. The blue line is used to connect the ground (GND) to it and all horizontal holes are connected along blue line. The red and blue lines are separated from themselves.

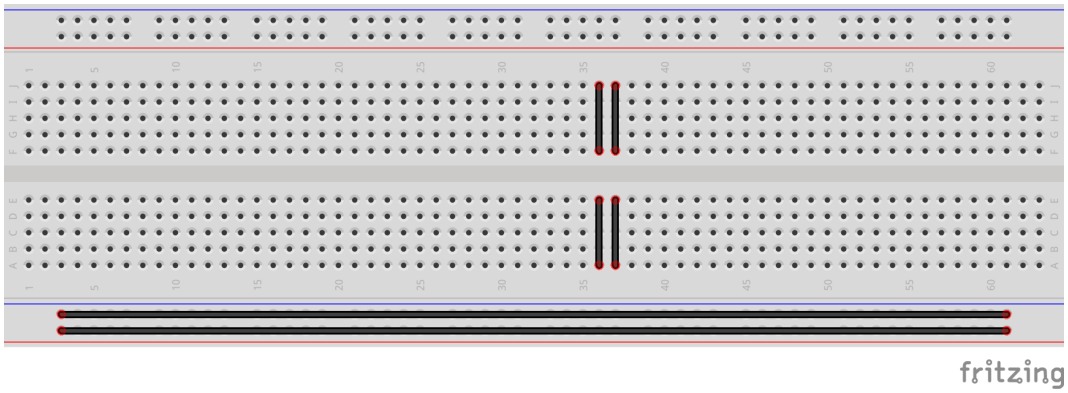


Figure 16: The example of breadboard. The black lines show, which holes are connected.

In figure 17 **wrongly placed** elements are shown. All legs of elements are shorted so that the elements will not work.

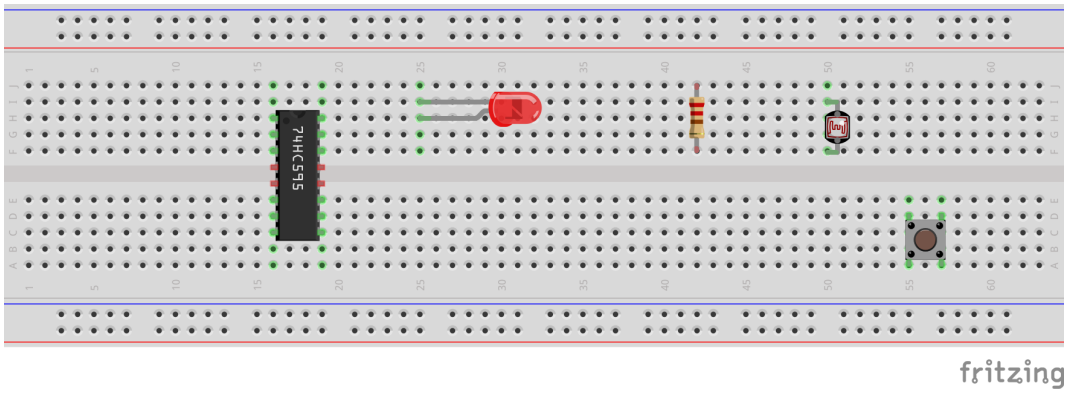


Figure 17: The example of **wrongly placed** elements on breadboard. The legs are shorted so that the elements will not work.

In figure 18 there is an example of correctly placed elements on breadboard.

Rectifying diode

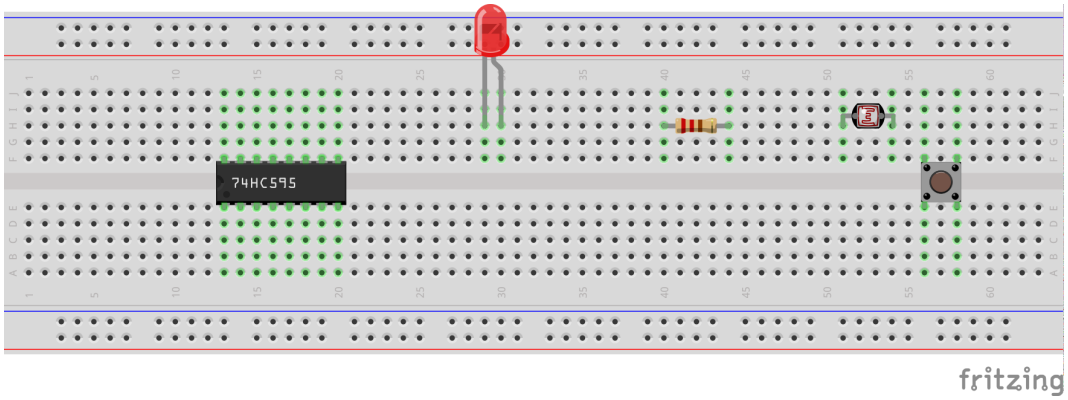


Figure 18: The example of **correctly** placed elements on breadboard.

The rectifying diode is useful element, because electric current can flow only in one direction through diode from anode to cathode. This element is used to protect sensitive elements from flowing of current in different direction than expected. In figure 19 the rectifying diode is shown.

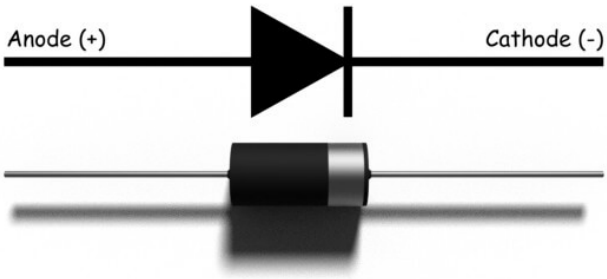


Figure 19: The rectifying diode and description of its legs. Source: [11]

Capacitor

The capacitor consists of two layers, in which charge gathers, and dielectric between them. It is used e.g. to filter power - because it does not let pass ripples of voltage but smoothes the signal. The capacitor can be used also to resonance circuits creation, which extract signal at the selected frequency e.g radio. Capacitors can be divided to: pole capacitors and non-polar capacitors. The electrolytic capacitors are included in polar capacitors and are shown in figure 20. Remember to connect properly electrolytic capacitors. The non-polar capacitors are divided on ceramic (see in figure 21) and foil, depending which material was used as dielectric. The non-polar capacitors can be connected freely.

The capacity is an ability to collect the charge. The electrolytic capacitors are characterized by high capacity but they are not efficient with high frequency signals due to the dissipation factor. In some frequencies such capacitors quite contrary may even behave as inductors. Ceramic capacitors do not dry out but they are not efficient with filtration of the low frequencies, because they have smaller capacitance.

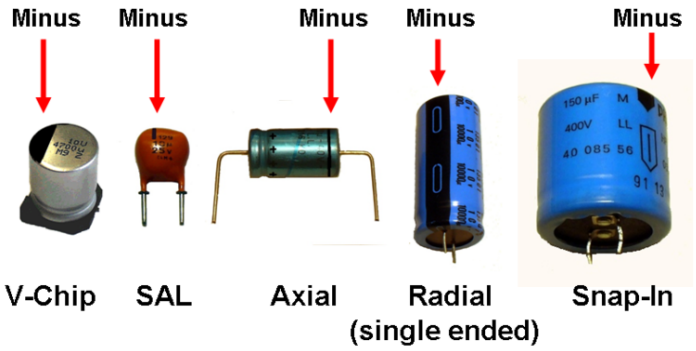


Figure 20: An example of electrolytic capacitors. Source: [12].

Figure 21: An example of ceramic capacitors. Source: [13].

Ceramic capacitors values frequently have 3 digits written on them. The two first digits are value in pico Farad (pF) and third digit is the multiplier (ten to the power of this number). This is quite simillar to resistors. In figure 21 many ceramic capacitors are shown. One of them has inscription 473, which means 47 pF $\cdot$ 10^3 = 47 pF $\cdot$ 1000 = 47 000 pF = 47 nF.

Potentiometer

The potentiometer is an element, which may regulate the voltage. It consists of three legs (see figure 22), in which the middle one is output leg. When the first leg is connected to e.g. 5 V and the last leg to 0 V (GND), the middle one will give the voltage between 0 to 5 V. This element also may be used as variable resistor if middle leg wil be connected to one of other legs. In such case regulated resistance will be between legs on the both ends.

Button

The button is an element, which close (shorts) the circuit when is pressed.

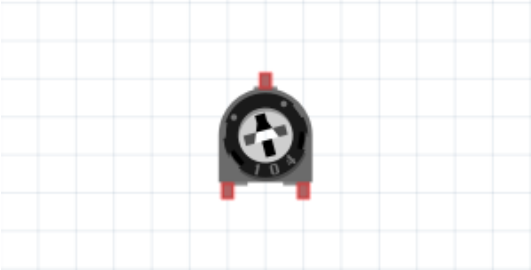


Figure 22: The example of potentiometer.

Servo motor

Servo motors are widely used in modeling, for example, remotely controlled aircrafts, in which the servo is responsible for ailerons movement. A servo motor is a motor with a gear that rotates by a given angle (see figure 23), usually in the range (0;180) degrees. Three wires are led out from the servo motor. The middle (usually red) should be connected to the power supply (e.g. 5 V). Black is ground (GND) and the last wire (usually in bright color: white / yellow) is a control line, which should be connected to the *pulse-width modulation* (PWM) pin.

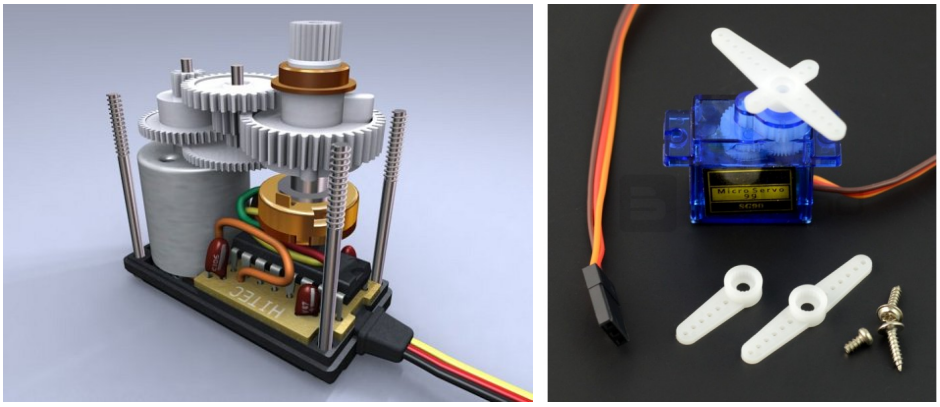


Figure 23: The example of servo motor construction. Source: [14, 15].

The controller placed in the servomechanism reads the PWM signal (see chapter no. 5.2) and on its basis determines the angle by which the gearbox should rotate (see figure 24). The PWM signal duty cycle determines the angle.

DC motor

DC motors (direct current motors), can be built in a variety of ways, but are generally easy to understand as devices that provide rotary motion of the shaft as a result of a repulsion of magnetic poles or an attraction (N-S). The DC motor is usually made of a permanent magnet in a stator (non-moving part) and electromagnets in the rotor (moving part). The electromagnet(s) in the rotor, depending on the direction of current flow through the winding, produce on the "outer side" either pole S or N. To ensure

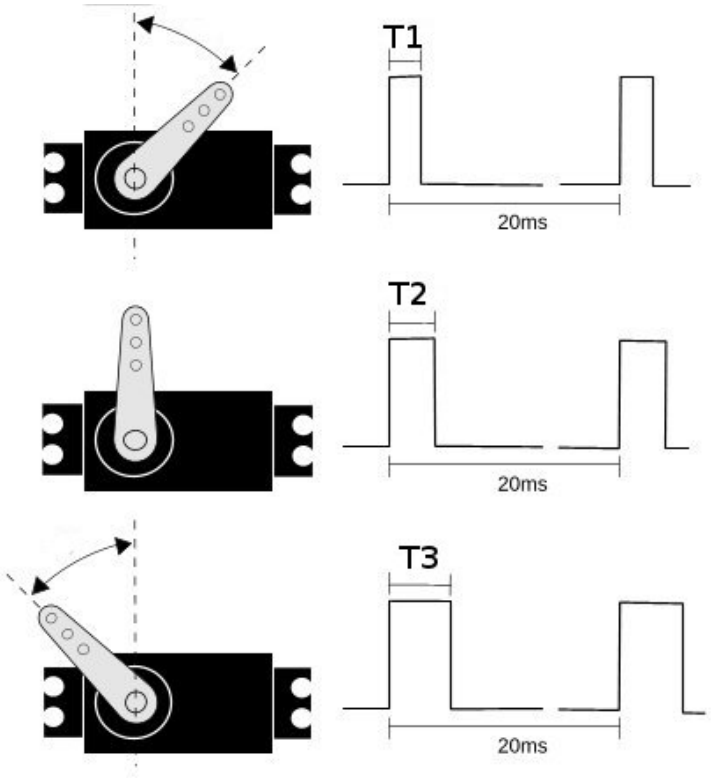


Figure 24: An idea of servo motor working. Source: [16].

rotary movement, the current direction must be changed in the right place. This is provided by a commutator, an element that changes the polarity of leads to coil windings in electromagnets as a result of rotation. The idea of DC motor working is shown in figure 25.

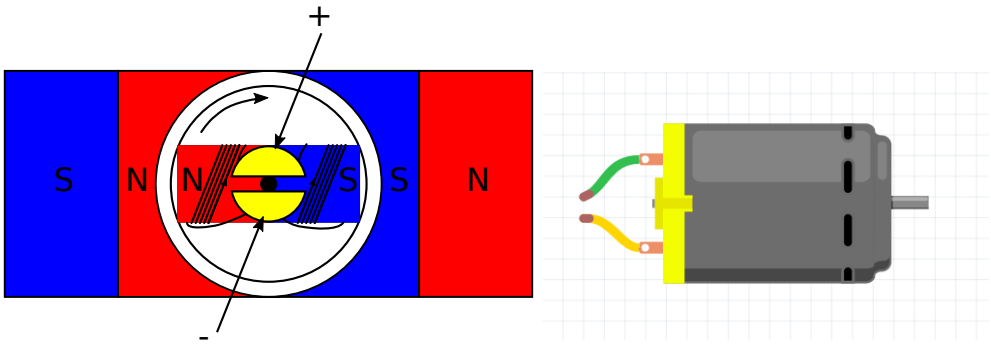


Figure 25: The schematic idea of DC motor working. Brushes (arrows) supply power to the commutator, which is marked in yellow. The rotor rotates clockwise.

4.3 Connecting the elements

The two elements can be connected in series or parallel e.g. this can be used with resistors and capacitors. If you connect two resistors in series (see figure 26), the total resistance (R_T) will be summarized:

$$R_T = R_1 + R_2 = 220\Omega + 470\Omega = 690\Omega \tag{5}$$

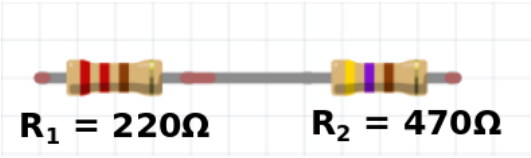


Figure 26: The series connection of two resistor.

If you connect two resistors in parallel (see figure 27), the total resistance will be calculated as:

$$\frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2} \rightarrow R_T = \frac{R_1 R_2}{R_1 + R_2} = \frac{220\Omega 470\Omega}{220\Omega + 470\Omega} \cong 150\Omega \tag{6}$$

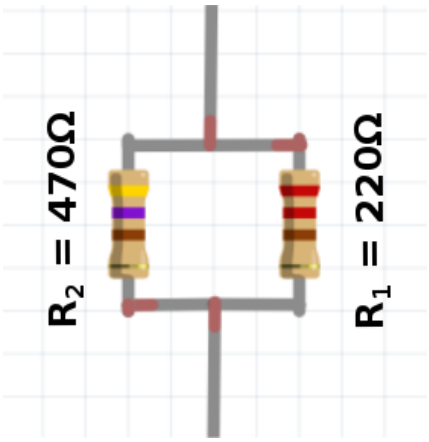


Figure 27: The parallel connection of two resistor.

If you connect two capacitor (C_1 and C_2) in series, the total capacity (C_T) will be defined as:

$$\frac{1}{C_T} = \frac{1}{C_1} + \frac{1}{C_2} \rightarrow C_T = \frac{C_1 C_2}{C_1 + C_2} \tag{7}$$

If you connect two capacitor (C_1 and C_2) in parallel, the total capacity (C_T) will be defined as:

$$C_T = C_1 + C_2 \tag{8}$$

5 The digital and analog signals

Elements can be divided into two groups: input and output elements. The first group consists on buttons and the all sensors, which measure some parameters like temperature, pressure, humidity, intensity of light, detected sounds etc. The value of measured parameter can be sent to the Arduino board as analog value or digital value. The second group consists of the actuators (executive devices), piezo buzzer, diodes, LED and LCD displays, servo and DC motors etc. These elements can be controlled by the Arduino board by sending digital or *pulse-width modulation* PWM. This chapter describes the way of sending digital and PWM signal and receiving the digital and analog signals.

All example codes are attached to the tutorial.

5.1 Digital signal

Sending adigital signal to chosen pin is always done in two steps:

1. Determining, which pin is output. You can read and write the digital signal from pins described as **Digital** or **PWM** on the Arduino board (see figures: 2, 3). Sometimes instead the PWM word, you can find "∼" sign.

2. Sending the *HIGH* or *LOW* value to chosen pin. The *HIGH* value corresponds to 5 V voltage and *LOW* value to 0 V.

Example 1

The blinking LED diode connected according to the figure 6 will be used as a example of sending a digital signal. The code for Snap4Arduino of turn on/off LED diode by every 1s is shown in figure 28.

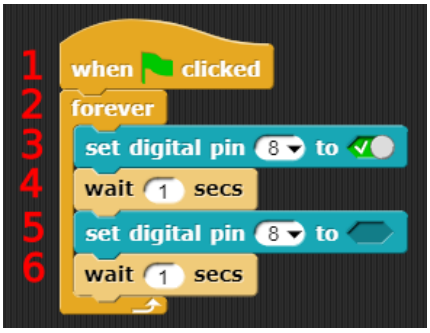


Figure 28: The example of blinking LED diode for every 1s prepared in Snap4Arduino software.

It consists of following steps:

1. The *when clicked* function runs the program after clicking on it.

2. The block *forever* will repeat all instructions inside.

3. The *set digital pin to* function will determine, which pin is output and sent the *HIGH/LOW* value to this pin. In this line, the *HIGH* value will be sent to the pin number 8. So that the LED diode will start shining.

4. The *wait* function will block the running program for chosen number of seconds. So that the LED diode will shine for 1s.

5. The *LOW* value will be sent to pin no 8, so that the LED diode will turn off.

6. The program will be blocked for 1s so that the LED diode will not shine for 1s.

Analogous code for Arduino IDE is shown below.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(8, OUTPUT); 1
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(8, HIGH); 2
  delay(1000); 3
  digitalWrite(8,LOW); 4
  delay(1000); 5
}
```

The whole code consists of two functions: *setup* and *loop*. The instructions, which are put in *setup* function, will run only ones. The instructions, which are put in *loop* function, will be repeated many times. The code consists of following lines:

1. **pinMode** function determines if the pin should be input or output. The first argument is the number of pin, in which element is connected. The second argument is the word **INPUT** or **OUTPUT**. The input pin should be chosen when the signal from elements will be received by Arduino board e.g. button, thermometer etc. The output pin should be chosen when the Arduino will set the signal on pin e.g. LED, servo motors, piezo etc. In this case, the pin should be **OUTPUT**, because LED will be turn on/off when the **HIGH** or **LOW** signal will be set by Arduino on pin. The LED is connected to pin number 8 so that the line looks: *pinMode(8, OUTPUT)*. All instructions should end by semicolon ";" . The configuration of pin is done only ones at the beginning of program so that this function should be located in *setup* function.

2. **digitalWrite** function is used to set the *HIGH* or *LOW* signal on chosen pin. The first argument is the number of pin and the second is the word *HIGH* or *LOW*. At the beginning, the LED will be turned on, so that the *HIGH* signal should be set on pin number 8.

3. **delay** is the function, which stops running the program for selected period of time in ms units. In this case, the program will wait 1000 ms = 1 s and for this time the LED will shine.

4. Next step is turning off the LED so that the *LOW* signal should be set on pin number 8.

5. The last step is waiting for 1000 ms and the LED will not shine.

The step between 2-5 should repeat to see the effect of blinking LED so that these lines are put in *loop* function.

Example 1.2

The previous example shows how to turn on/off one LED diode. This example is a modification by adding additional LED diodes. Let us assume that we would like to make traffic lights so we need three LED diodes: red, yellow and green. The electronic circuit is shown in figure 29.

The code for Snap4Arduino is shown in figure 30 and consists of following lines:

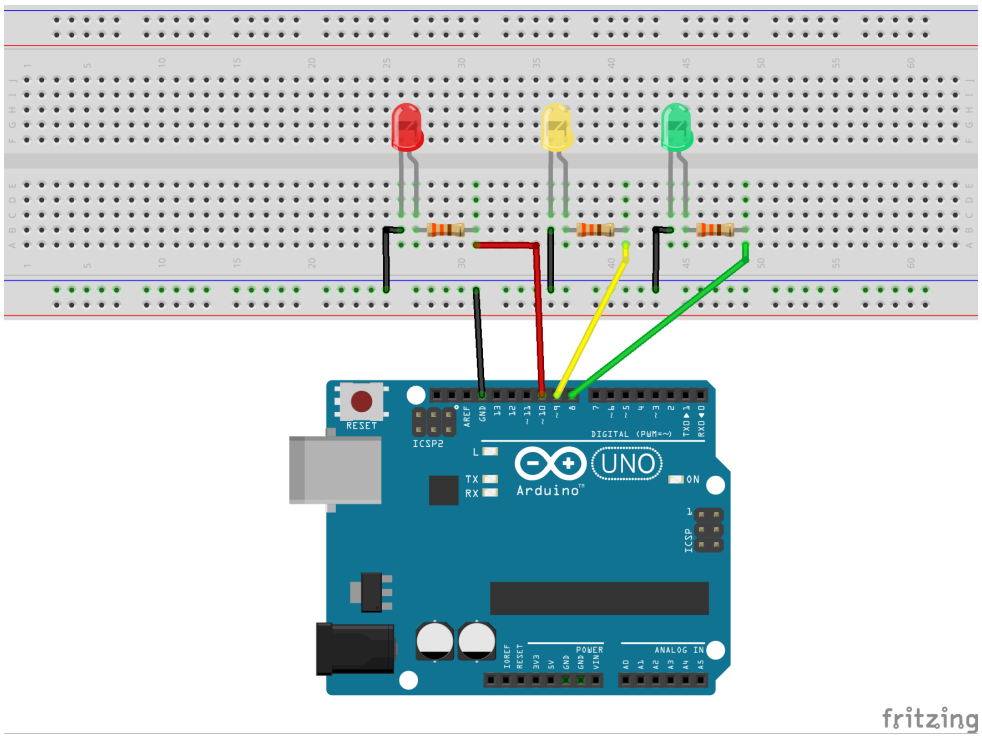


Figure 29: An example of traffic lights electronic circuit scheme.

1. Sometimes, it is needed to use one value in many places of code. If you need to modify it in the future, you may need to change it many times. Variables are used to save time and make code easier to read. The variables consists of name and the value. In this example, three variables are created: *red*, *yellow* and *green*. The

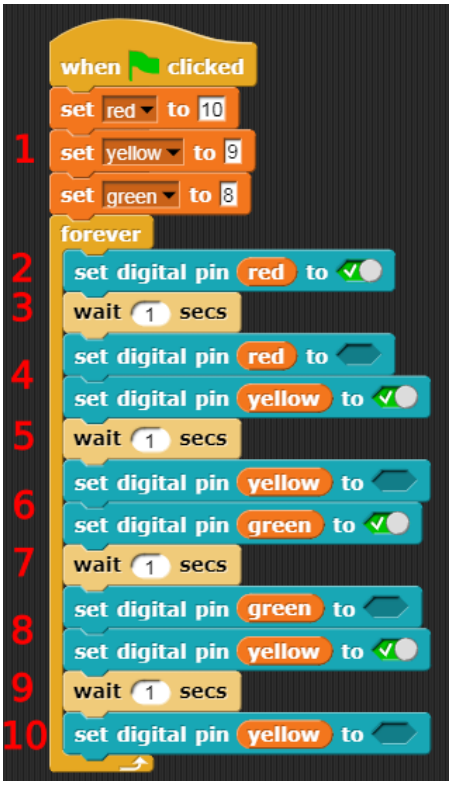


Figure 30: An example of code for traffic lights prepared in Snap4Arduino Software.

values 8, 9 and 10 are saved to variables *red*, *yellow* and *green* respectively. If the *red* variable will be used in code, the program will read and use the value 10 from it. If the red LED diode will be connected to other pin number, then you need only to place new value to the *red* variable only once.

2. The *set digital* function will turn on the red LED diode by sending the *HIGH* value. The pin number is forwarded as the *red* variable.

3. The *wait* function will block program for 1 s and the red LED diode will shine.

4. The red LED diode will be turned off by sending *LOW* value to *set digital* function. The yellow LED diode will be turned on.

5. The *wait* function will block program for 1 s and the yellow diode will shine.

6. The yellow LED diode will be turned off and the green LED diode will be turned on.

7. The *wait* function will block program for 1 s and the green diode will shine.

8. The green LED diode will be turned off and the yellow LED diode will be turned on.

9. The *wait* function will block program for 1 s and the yellow diode will shine.

10. The green LED diode will be turned off.

Analogous code for Arduino IDE is shown below:

```
int LED_red = 10;
int LED_yellow = 9; 1
int LED_green = 8;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_red, OUTPUT);
  pinMode(LED_yellow,OUTPUT); 2
  pinMode(LED_green,OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_red, HIGH); 3
  delay(1000); 4
  digitalWrite(LED_red, LOW);
  digitalWrite(LED_yellow, HIGH); 5
  delay(1000); 6
  digitalWrite(LED_yellow, LOW);
  digitalWrite(LED_green, HIGH); 7
  delay(1000); 8
  digitalWrite(LED_green, LOW);
  digitalWrite(LED_yellow, HIGH); 9
  delay(1000); 10
  digitalWrite(LED_yellow, LOW); 11
}
```

Before we start to analyse the code, the variables will be introduced. The variables are created to save time and make code easier to read. They consist of three elements: type, name and value. The type determines, how value should look and how much place in memory this value will occupy. The basic types of variables are shown in table 2. The name is a label we can use in other places of code and value is an initial value which name refers to.

Type	**Range**	**Description**	**Examples**
int	$(-2147483648, 2147483647)$	integer	-3 , 5 , 20000
unsigned int	$(0, 4294967295)$	unsigned integer	0, 50, 1000
float	$(1.2E-38, 3.4E+38)$	floating numbers	1.3, -6.789
double	$(2.2E-308, 1.8E+308)$	double precision floating number	-0.987, 1.356
char	$(-128, 127)$	stores characters in ASCII code	'a', 'f'
bool	*True*, *False*	boolean value	True

Table 2: The basic type of variables in C++ language.

The code of traffic lights consists of following lines:

1. Creation of three variables, which store the pin numbers for red, yellow and green LED diodes.
2. Determining, which pin should be *OUTPUT* using the *pinMode* function.
3. Writing the *HIGH* value to pin connected with the red LED diode by using *digitalWrite* function. The red LED diode will start shinning.
4. Block the program running for 1000 ms by using *delay* function. The red LED diode will shine.
5. Turn off the red LED diode by sending *LOW* value to the pin. The yellow LED diode will be turned on.
6. Block the program running for 1000 ms by using *delay* function. The yellow LED diode will shine.
7. Turn off the yellow LED diode by sending *LOW* value to the pin. The green LED diode will be turned on.
8. Block the program running for 1000 ms by using *delay* function. The green LED diode will shine.
9. Turn off the green LED diode by sending *LOW* value to the pin. The yellow LED diode will be turned on.
10. Block the program running for 1000 ms by using *delay* function. The yellow LED diode will shine.
11. Turn off the yellow LED diode by sending *LOW* value to the pin.

Example 2

Previous examples show how to send the digital signals to the pins. This example will show how read digital signal from pin. For this purpose the electronic circuit from example 1.2 will be modified by adding the button (see figure 31). The capacitor is used to eliminate contact vibrations and randomly generated spikes. There is also a resistor 10 kΩ which is connected from button to GND. This is so-called pull-down resistor, which enforces the *LOW* state on input pin when button is not pressed. When button is pressed 5 V (*HIGH* state) is connected through button to input pin. In this situation small current will flow through resistor to GND.

The goal is to turn on the LEDs after pushing button. The code for Snap4Arduino is shown in figure 32 and consists of following lines:

1. Initialization of variables.

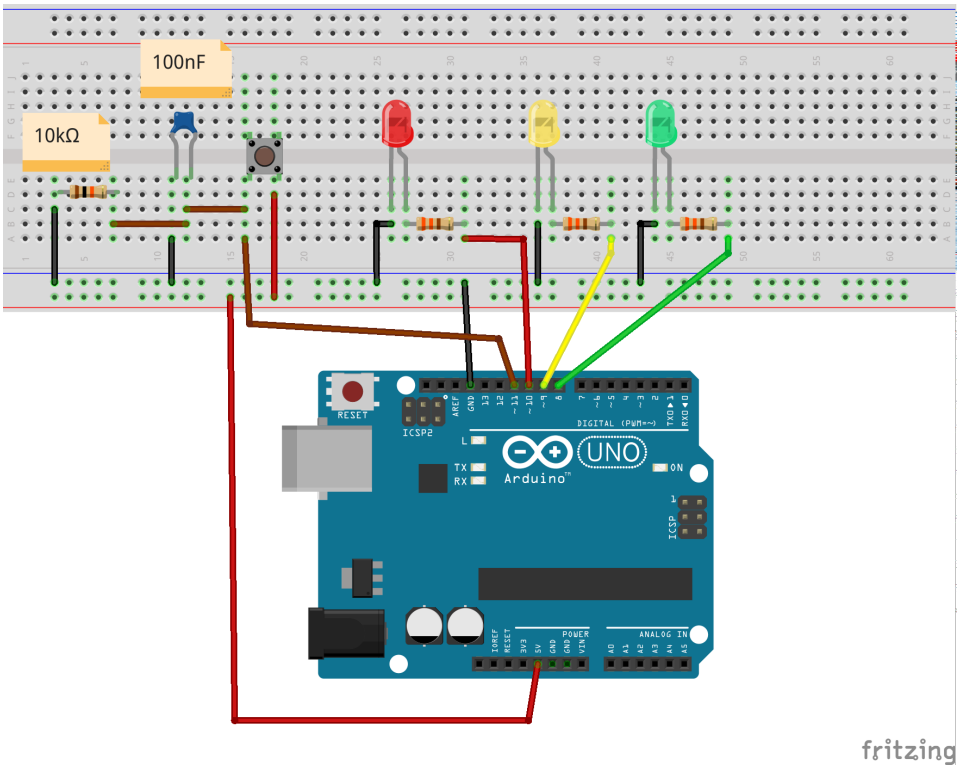


Figure 31: The electronic circuit design for example 2.

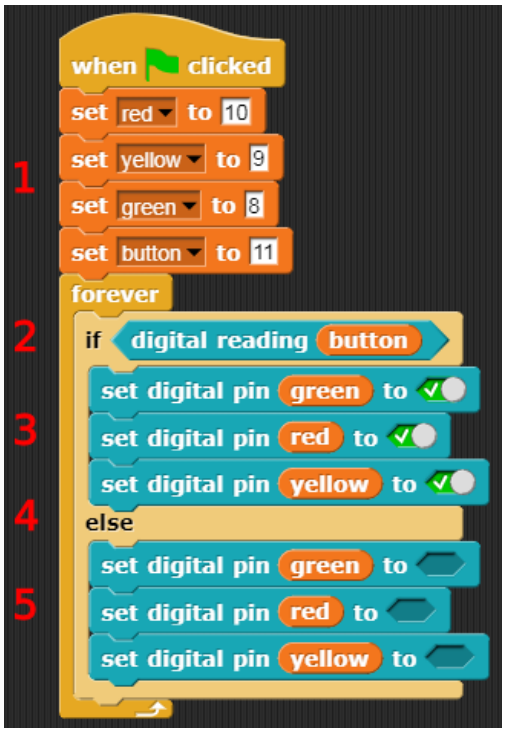


Figure 32: Code for example 2 written in Snap4Arduino.

2. A block *if...else* is used, which means that if condition after the word *if* is met, instructions which are put inside (lines with number 3) will be executed. If the condition will not be met, these lines (3) will not execute. In this case, if the button is pressed, the *if* block will execute instructions inside it.

3. LEDs will be turned on if the button was pressed.

4. A part of block *else* will run, if the condition will not be met.

5. LEDs will be turned off if the button was not pressed.

In this example the *set digital pin* function is repeated. If you connect more than three LED diodes, the code will start to grow. An example of code, which can be easily modified for more LED diodes without growing the code, is shown in figure 33.

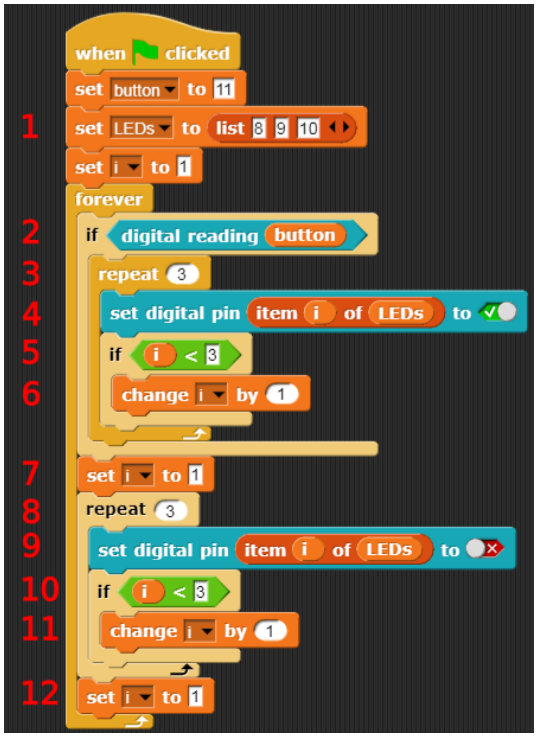


Figure 33: A code for example 2 written in Snap4Arduino, which can be easily modified for more LED diodes.

This code consists of following lines:

1. Previously, the variables were created for single values. Sometimes we would like to store more than one number. For this purpose a list can be used. The list name is *LEDs* and has following values: 8, 9 and 10. The first element of list is 8, the second element of list is 9 and the third element is 10.

2. If user has pushed the button, instructions inside the block would be executed.

3. The *repeat* block will repeat the instructions put inside the chosen amount of times. In this case the instruction will be repeated 3 times.

4. The *set digital pin* function is used to turn on LED. Normally, as the first argument of this function, a number of chosen pin is used. Actually, all pins are stored in a list. For first iteration of *repeat* block we would like to set number 8 to *set digital pin* function. For the second iteration we would like to use number 9. For the third iteration we would like to use number 10. So that we can use additional variable called i to iterate through the list and turn on all LED diodes. The *item ... of ...* function allows to choose one value from the list. The i variable will have value equal to 1 at the beginning of *repeat* block, so that the first LED diode connected to pin number 8 will be turned on.

5. This line checks if i variable is less than 3. The list has only 3 elements, so that if the i variable will have value greater than 3, the program will return an error (you cannot iterate over non-existant elements of the list).

6. The *change ... by ...* function allows to increase or decrease the value of variable. In this case the i variable will increase the value by 1 so that during the next iteration of *repeat* block, the second LED diode will be turned on by *set digital pin* function.

7. This line sets i variable to initial value (1).

8. This line will execute *repeat* block for three times to turn off the LEDs.

9. The *set digital pin* function will turn off the LED diode.

10. This line checks if i variable is less than 3.

11. The i variable will increase the value by 1 so that during the next iteration of *repeat* block, the second LED diode will be turned off by *set digital pin* function.

12. This line sets i variable to initial value (1).

Analogous code for Arduino IDE is shown below:

```
int LED_red = 10;
int LED_yellow = 9;
int LED_green = 8;
int button = 11;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_red, OUTPUT);
  pinMode(LED_yellow,OUTPUT);
  pinMode(LED_green,OUTPUT);
  pinMode(button, INPUT); 1
}

void loop() {
  // put your main code here, to run repeatedly:
```

```
  if(digitalRead(button)) 2
  {
    digitalWrite(LED_red, HIGH);
    digitalWrite(LED_yellow, HIGH); 3
    digitalWrite(LED_green,HIGH);
  }
  else 4
  {
    digitalWrite(LED_red, LOW);
    digitalWrite(LED_yellow, LOW); 5
    digitalWrite(LED_green, LOW);
  }
}
```

1. Button sends *HIGH* signal to the Arduino board, when pressed. So that the button is the *INPUT* element. The *pinMode* function determines that the pin number 11 connected to the button will be *INPUT*.

2. Block *if...else* is used, which means that if condition after the word *if* is met, the instructions put inside (lines with number 3) will be executed. If the condition will not be met, these lines (3) will not execute. In this case, if the button is pressed, the *if* block will execute instructions inside it.

3. LEDs will be turned on if the button was pressed.

4. Part of block *else* will run, if the condition will not be met.

5. LEDs will be turned off if the button was not pressed.

In this example the *digitalWrite* function is repeated. If you connect more than three LED diodes, the code will start to grow. An example of code, which can be easily modified for more LED diodes without growing the code, is shown below:

```
int LEDs[] = {10,9,8}; 1
int button = 11;

void setup() {
  // put your setup code here, to run once:
  for(unsigned int i=0; i<3; i++) 2
  {
    pinMode(LEDs[i], OUTPUT); 3
  }
  pinMode(button, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(button))
  {
    for(unsigned int i=0; i<3;i++) 4
    {
```

```
      digitalWrite(LEDs[i], HIGH); 5
    }
  }
  else
  {
    for(unsigned int i=0; i<3;i++) 6
    {
      digitalWrite(LEDs[i], LOW); 7
    }
  }
}
```

This code consists of following lines:

1. Pin numbers are stored in table. The table is a variable, which consists of a type, a name and values. An advantage of the table is that you can store more than one value inside of it. In this example, the table type is *int* and table is named *LEDs*. Values of the table are 10, 9, 8 and are put in {} brackets. You can easily iterate the table to receive one chosen value. If you would like to read first element of table, the code will look like: *LEDs[0]*. The number in brackets (called index) informs, which element from table will be received. The index starts from 0 so that the first element will have index 0. The second element you can read by writing *LEDs[1]* and the third by *LEDs[2]*.

2. The *pinMode* function should be executed three times with three values from *LEDs* table separately. The *for* loop allows to repeat the instructions until the condition will be met. The *for* loop is used according to the scheme:

   for (creating variables **;** loop termination condition **;** changing the value of variables)

   In this example, the *for* loop should run three times to initialize three different pins as *OUTPUT*. The index of table will be changed by variable called i starting from 0. The second argument of *for* loop says that the loop will run as long as i variable will be smaller than 3. The third argument says that after one execution of loop, the i variable will be increased by 1.

3. The *pinMode* function is used to determine, which pins will be *OUTPUT*. The number of pin is transferred as chosen element from table. During the first iteration of *for* loop, the i variable will be equal to 0 and the *LEDs[i]=LEDs[0]=10*. After one iteration, the i variable will increase by 1. During second iteration, the i variable will be equal to 1 and the *LEDs[i]=LEDs[1]=9*. Similarly, during the third iteration the i variable will be equal to 2 and the *LEDs[i]=LEDs[2]=8*.

4. The *for* loop should run three times to turn on three LED diodes separately.

5. The *digitalWrite* function is used to send *HIGH* value to pin and turn on the LED diode.

6. The *for* loop should run three times to turn off three LED diodes separately.
7. The *digitalWrite* function is used to send *LOW* value to pin and turn off the LED diode.

5.2 PWM

Pulse Width Modulation (PWM) is a technique to generate the digital square wave, in which the *HIGH* value is set for specified time. In figure 34 an idea of PWM generation is presented, in which the green lines depict a PWM period which equals to about 2 ms for Arduino *analogWrite* function.

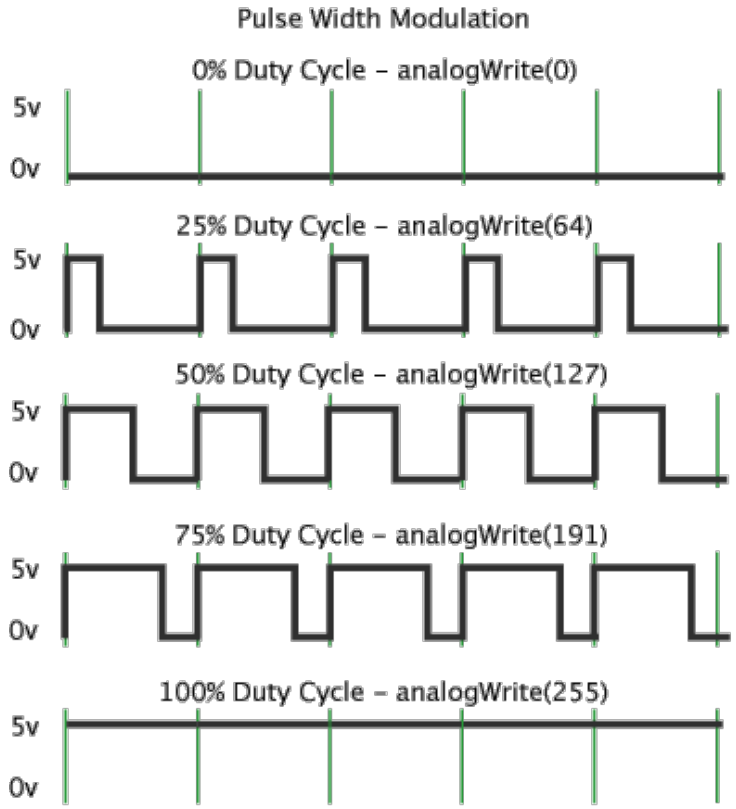


Figure 34: The idea of pulse width modulation. Source: [1].

You can generate the PWM signal from pins described as **PWM** on the Arduino board (see figures: 2, 3). Sometimes instead the PWM word, you can find "∼" sign.

The function, which is used to control the width of signal is ***analogWrite(pin, width)*** for Arduino IDE, which needs the width as the second argument. The width is determined as the value between 0-255. The maximum value 255 represents the 100% duty cycle so that the *HIGH* state will last whole period. The value 127 corresponds to the 50% of duty cycle, so that the *HIGH* and *LOW* states will be generated alternately for half time of period (about 1 ms). The 64 value corresponds to 25% duty cycle so that the *HIGH* state will last 25% of time (about 0.5ms) and the *LOW* signal will last 75% of time (about 1.5 ms).

The PWM technique is commonly used to change brightness of LED diodes, to change the angle of servo motors etc. The PWM signal can be also used to generate analog signal from the Arduino board. Normally, the Arduino board allows only to read analog signal. But you can use the PWM signal and low pass filter, which converts the PWM signal to constant voltage proportionally to the percentage of the PWM duty cycle. The scheme of the low pass filter is shown in figure 35. Simple low pass filter can be assembled using 100 nF capacitor and 3.9 kΩ resistor.

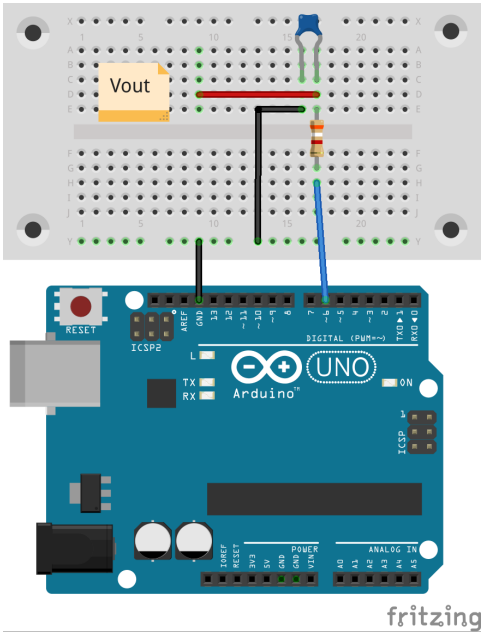


Figure 35: The electronic circuit scheme of low pass filter connected to PWM output from the Arduino UNO board.

Example 3

This example shows how to simulate the brightness of LED diode. The human eyes work slowly so that we cannot notice the changing between *HIGH* and *LOW* signal during generation of PWM signal. When the duty cycle is smaller, the human eyes see that LED diode shines at lower intensity. When duty cycle is higher the LED diode will shine with higer intensity. In fact, the brightness of LED diode does not change but the effect is connected only with restrictions of speed of image registration by human eye. During this example, we will change the LED diode brighteness by changing the PWM duty cycle from 0 to 100% and than from 100% to 0%. The electronic circuit is shown in figure 36.

The code for Snap4Arduino is shown on figure 37 and consists on line:

1. A *LED* variable is created and its value is set to a pin number, where LED diode is connected. This pin should be PWM type.

2. i variable is created and its value of is set to 0.

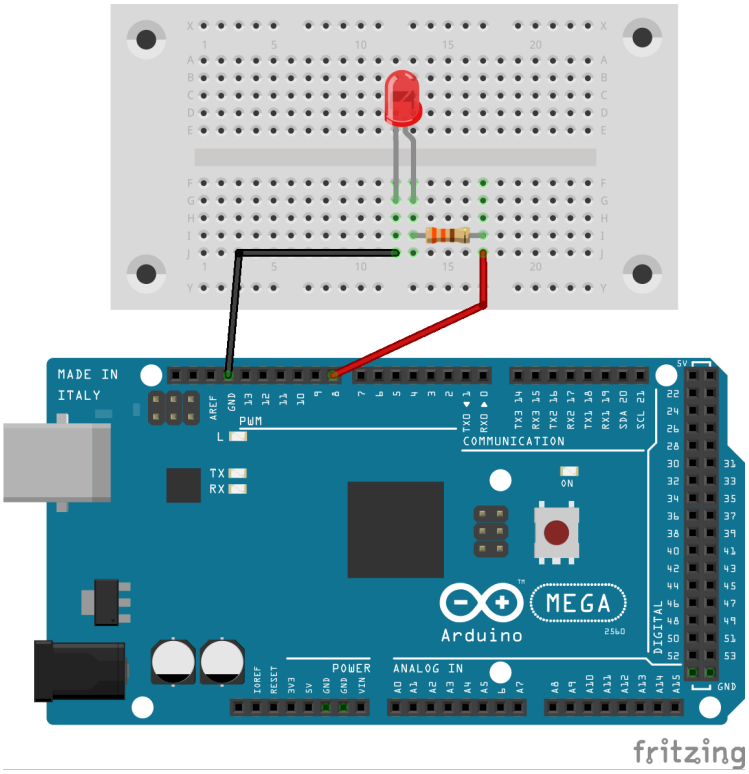


Figure 36: An electronic circuit of LED diode for example 3.

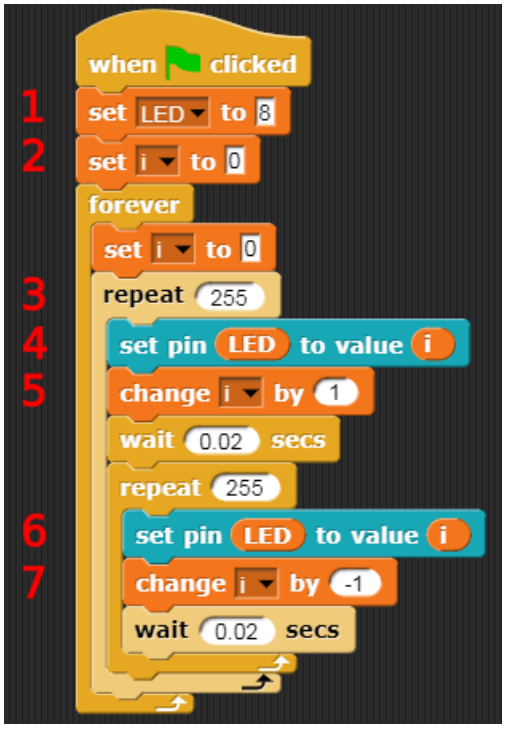


Figure 37: A code for example 3 prepared in Snap4Arduino software.

3. Acounterpart of *analogWrite* function is *set pin ... to value ...* function in Snap4Arduino. The PWM signal should be generated from 0 to 255 to see brightening of LED diode. So that *repeat* block is used to repeat the *set pin ... to value ...* function 255 times.

4. The *set pin ... to value ...* function is used to change the width of signal. The value of width is stored in i variable.

5. The i variable is changed by 1 after one *repeat* block iteration.

6. The PWM signal should be generated from 255 to 0 to see darkening of LED diode. So that *repeat* block is used to repeat the *set pin ... to value ...* function 255 times. The value of width is stored in i variable.

7. The i variable is changed by -1 after one *repeat* block iteration.

Analogous code for Arduino IDE is shown below and consists of following lines:

```
int LED=8;

void setup() {
  // put your setup code here, to run once:
  pinMode(8,OUTPUT); 1
}

void loop() {
  // put your main code here, to run repeatedly:
  for(unsigned int i=0; i<256; i++) 2
  {
      analogWrite(LED, i); 3
      delay(20);
  }
  for(unsigned int i=255; i>0; i--) 4
  {
      analogWrite(LED, i); 5
      delay(20);
  }
}
```

1. First, the pin number 8, which is used to connect LED diode, is initialized as *OUTPUT*. The pin number 8 can work as PWM pin o the Arduino Mega 2560 board.

2. Duty cycle of PWM signal should be changed from 0 to 255 to see brightening of LED diode. So that the *for* loop is used and will increase the i variable value by 1 after every iteration of loop as soon as i will be smaller than 256.

3. The *analogWrite* function is used to generate the PWM signal and the i variable is used to change the duty cycle from 0 to 255.

4. Duty cycle of PWM signal should be changed from 255 to 0 to see darkening of LED diode. So that the *for* loop is used and will decrease the i variable value of 1 after every iteration of loop as soon as i will be greater than 0. The initial value of i variable is equal to 255.

5. The *analogWrite* function is used to generate the PWM signal and the i variable is used to change the duty cycle from 255 to 0.

Example 4

PWM signal is commonly used to change the color of RGB diode, which terminals are described in figure 38. Following program will generate the colors from rainbow: red, orange, yellow, green, blue and violet, which will change after 1 s. The electronic circuit is shown in figure 39.

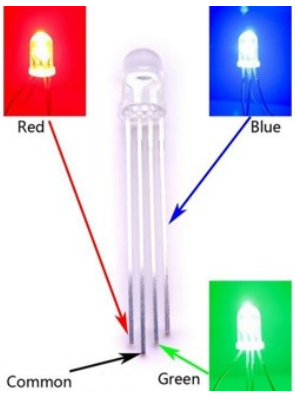


Figure 38: Terminals of RGB diode. Source: [17].

Colors of rainbow with the duty cycles are shown in table 3.

Color	Red	Green	Blue
red	255	0	0
orange	255	150	0
yellow	255	255	0
green	0	255	0
blue	0	0	255
violet	150	0	255

Table 3: Values of duty cycles for red, green and blue LED diodes to generate the rainbow colors.

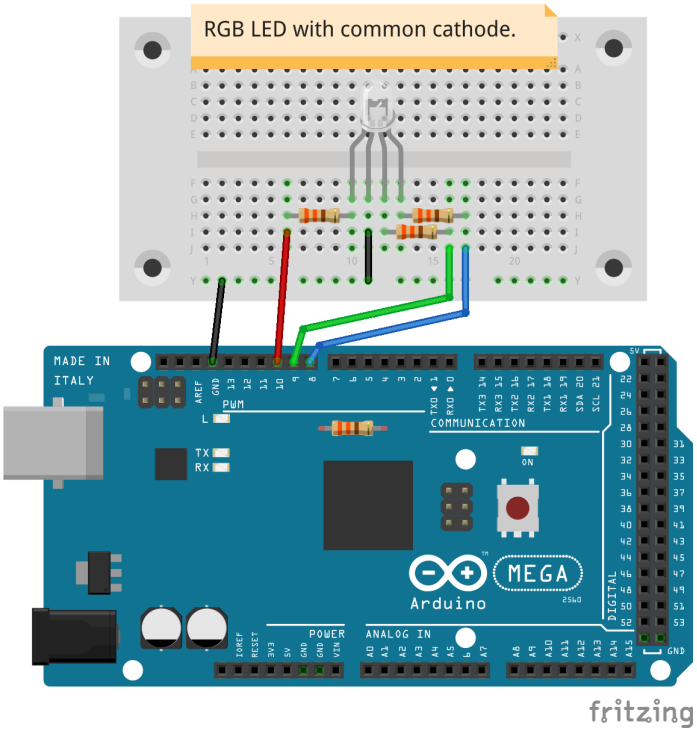


Figure 39: The electronic circuit for example 4.

A code for changing the RGB diode colors is shown in figure 40 and consists of following lines:

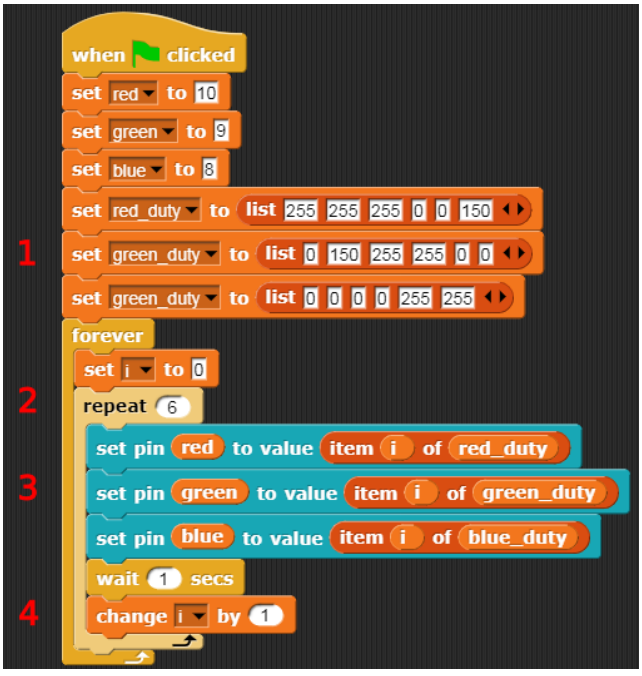


Figure 40: A code for example 4 prepared in Snap4Arduino software.

1. The variables of duty cycles for all colors for red, green and blue LED diodes are created as lists separately.

2. Rainbow is constructed with 6 colors: red, orange, yellow, green, blue and violet so that the *set pin ... to value ...* function should be repeated 6 times by using *repeat* block.

3. The duty cycles are set separately for red, green and blue diodes by using *set pin ... to value ....* The width is transferred as element from list by using an index i variable.

4. After one iteration of *repeat* block, the i variable is increased by 1.

Analogous code for Arduino IDE is shown below and consists of following lines:

```
int LED_red=10;
int LED_green=9;
int LED_blue=8;

int red_values[]={255, 255, 255, 0, 0, 150};
int green_values[]={0, 150, 255, 255, 0, 0}; 1
int blue_values[]={0, 0, 0, 0, 255, 255};

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_red,OUTPUT);
  pinMode(LED_green,OUTPUT); 2
  pinMode(LED_blue,OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  for(unsigned int i=0; i<6; i++) 3
  {
    analogWrite(LED_red, red_values[i]);
    analogWrite(LED_green, green_values[i]); 4
    analogWrite(LED_blue, blue_values[i]);
    delay(1000);
  }
}
```

1. The variables of duty cycles for all colors for red, green and blue LED diodes are created as a table separately.

2. Pins connected to the red, green and blue LED diodes are initialized as *OUTPUT*.

3. Rainbow is constructed with 6 colors: red, orange, yellow, green, blue and violet so that the *analogWrite* function should be repeated 6 times by using *for* loop. An i variable is created as index, which will be used to chose value of duty cycles from table for chosen color.

4. The *analogWrite* function is used to change the duty cycle for the red, green and blue LED diodes separately.

5.3 Analog signal

The Arduino board allows to read the analog signal, which is returned by many sensors. Analog pins are described as *Analog IN* on the Arduino board (see figures: 2, 3). A function, which is used to read the analog value from chosen pin is ***analogRead(pin)*** in Arduino IDE. The maximum measured voltage by Arduino pins is 5V, the minimum voltage is 0V. The *analogRead* function returns value between 0 to 1023. So that the read value can be transferred to the voltage (V_{read}) according to equation:

$$V_{read} = \frac{5.0}{1024} \cdot value \tag{9}$$

The Arduino IDE provides useful function, which recalculate the value from one range to the another. This function is called ***map(value, minimum value from old range, maximum value from old range, minimum value from new range, maximum value from new range)***. So that the read value from *analogRead* function can be transferred to voltage by line:

```
int value = analogRead(pin);
double voltage = map(value, 0, 1023, 0, 5);
```

Example 5

This example will be used to read the temperature from LM35 sensor, connected according to the figure 41.

Read value should be displayed somehow. It can be done in many ways e.g. using the LCD display or Serial Monitor. The *Serial* is a communication interface, which consists of two lines: *Rx* (to receive data) and *Tx* (to transmit data). Transmission by Serial is asynchronous. Both communicating devices, which one receive data and second transmits data, must have the same *baud-rate*, which specifies the number of transmitted bits per second. In this example, the read value, calculated voltage and temperature will be displayed in *Serial Monitor* for Arduino IDE. Unfortunately, the *Serial* monitor is not available for Snap4Arduino, but the actual values of variables are display in the right corner of this software.

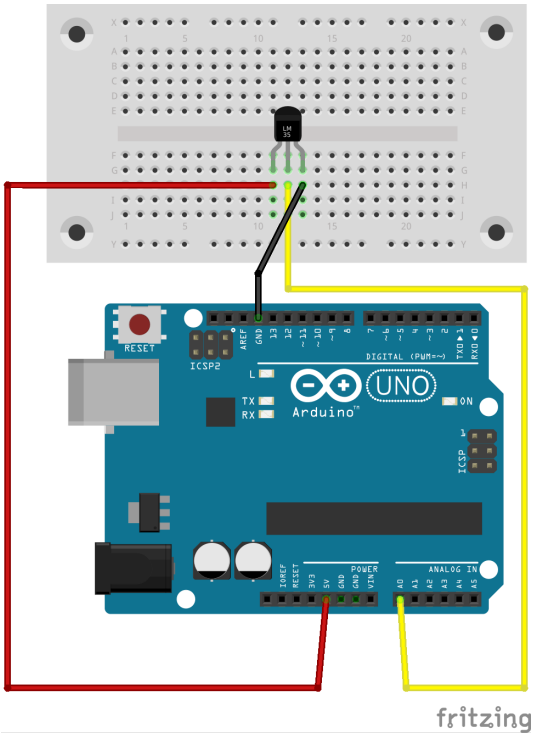


Figure 41: An electronic circuit for example 5.

A code for example 5 for Snap4Arduino is shown in figure 42 and consists of following lines:

1. An analog value from temperature sensor is read by function *analog reading* and saved to *value* variable.
2. A voltage is calculated according to equation 9.
3. According to documentation of LM35 sensor, the temperature is changing by 10 $mV/^oC$ so that the voltage should be multiplied by 100 to receive temperature.

Analogous code for Arduino IDE is shown in figure 43 and consists of following lines:

1. A pin, to which the LM35 is connected, is initialized as *INPUT*.
2. An analog value is read by *analogRead* function and saved to *value* variable.
3. A voltage is calculated according to equation 9.
4. According to documentation of LM35 sensor, the temperature is changing by 10 $mV/^oC$ so that the voltage should be multiplied by 100 to receive temperature.
5. Function *print* called on the object *Serial* is used to display the text or value to *Serial Monitor*, which is shown on the right side of figure 43.

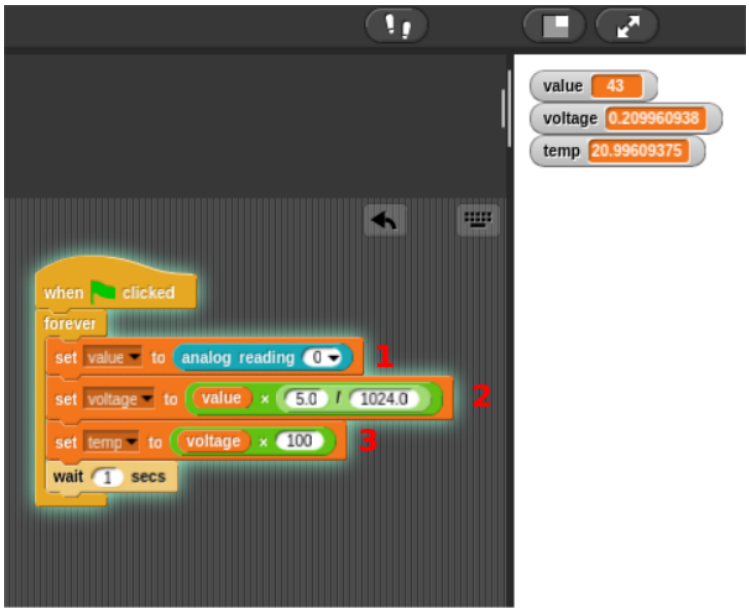


Figure 42: A code for example 5 prepared in Snap4Arduino software.

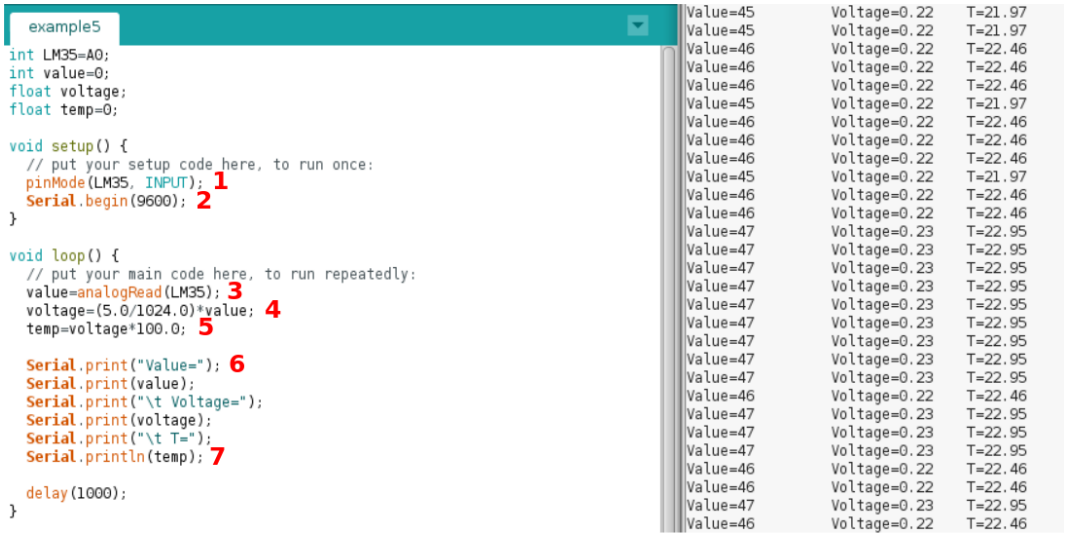


Figure 43: A code for example 5 prepared in Arduino IDE software.

6. Function *println* called on the object *Serial* is used to display the inscription or value to *Serial Monitor* with addition of new line tarminating character. The effect will be the same as "pushing the *Enter* key".

6 Sensors

Sensors are used to register the measured quantity as temperature, humidity, pressure, distance, etc. The measured value can by read by the Arduino board as the analog signals, time between two signals or digital values using data communication interfaces (bus). Depending on the type of sensor, the different way of reading measured data needs to be applied. The most popular sensors will be described with examples in this chapter.

6.1 Basic sensors

Example 6

The first example of sensor readout will be based on ultrasonic distance sensor HC-SR04, which is shown in Figure 44. This sensor generates the ultrasonic wave, which reflects from the objects and come back to the sensor. Time between generating and registering the wave front is proportional to the distance and depends on the speed of propagation of a given wave in the medium e.g. air. In the similar way, the bats assess the distance.

This example can be used as a parking sensor in car, which measures the distance from objects and alarm the driver by a buzzer sound.

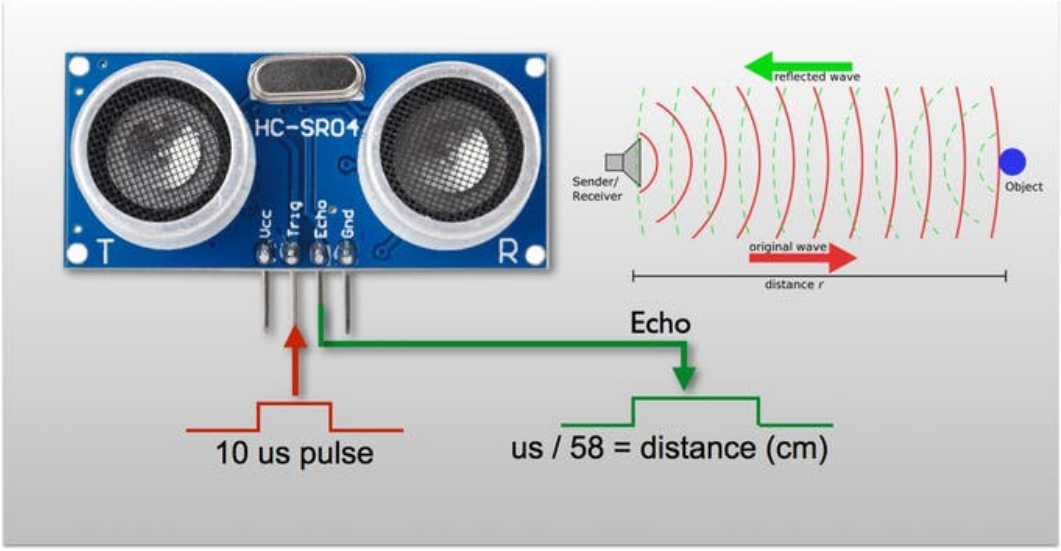


Figure 44: The HC-SR04 ultrasonic distance sensor. Source: [18].

The electronic circuit is shown in Figure 45.

The ultrasonic sensor usage can be programmed with a ready library from GitHub repository or by writing own function, which is much more didactic and showing the idea of a distance measurement by using the ultrasonic waves. The first version of code is prepared for Arduino IDE and shown in Figure 46.

The code consists of the following lines:

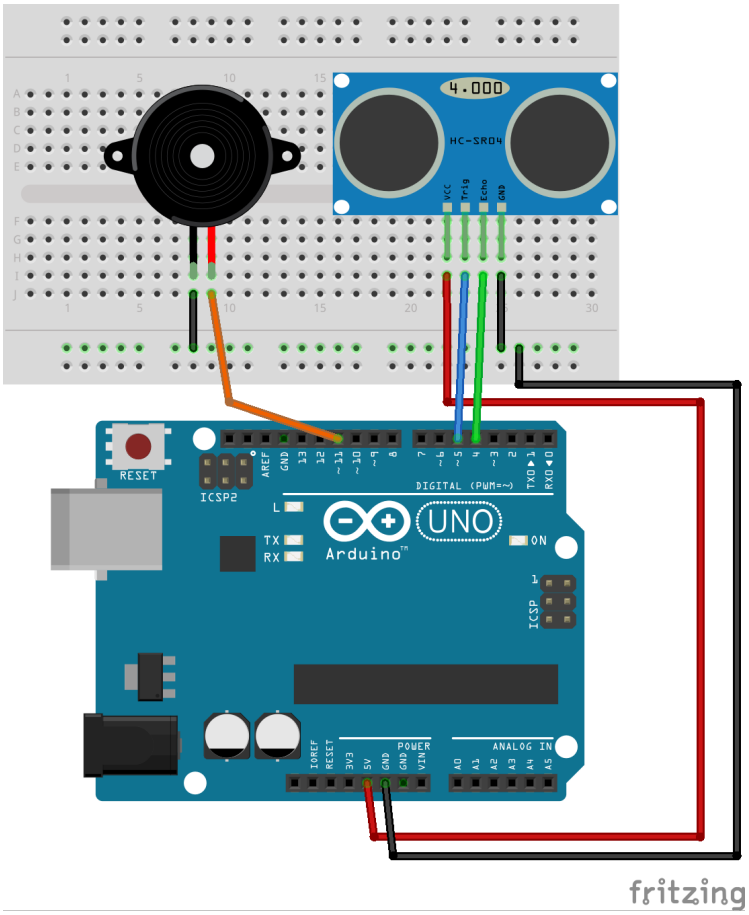


Figure 45: The electronic circuit for example 6.

1. Creating the variables, which determine used pins by numbers.

2. The two variables: *Time* and *distance* are created and their type is long. The Long type is extended version of int type, which value can be in range of: (-2 147 483 648, 2 147 483 647).

3. The ultrasonic sensor has two main legs called: *trigger* and *echo*. The leg *trigger* is used to release the wave generation and the *echo* is used to listen for the arrival of the wave. So that the *trigger* pin should be *OUTPUT* and *echo* pin should be *INPUT*. The buzzer, which is also used to generate sound, should be *OUTPUT*.

4. Releasing the wave generation is done after sending the *HIGH* signal level to the *trigger* pin for 10 μs. Before it, the *LOW* signal level should occurs for 2μs. This line set the *LOW* signal level to the *trigger* pin.

5. The *delayMicroseconds* function is used to generate the delay in μs. This line blocks running the program for 2μs so that the *LOW* signal on the *trigger* pin will last.

6. The *HIGH* signal level is set on the *trigger* pin.

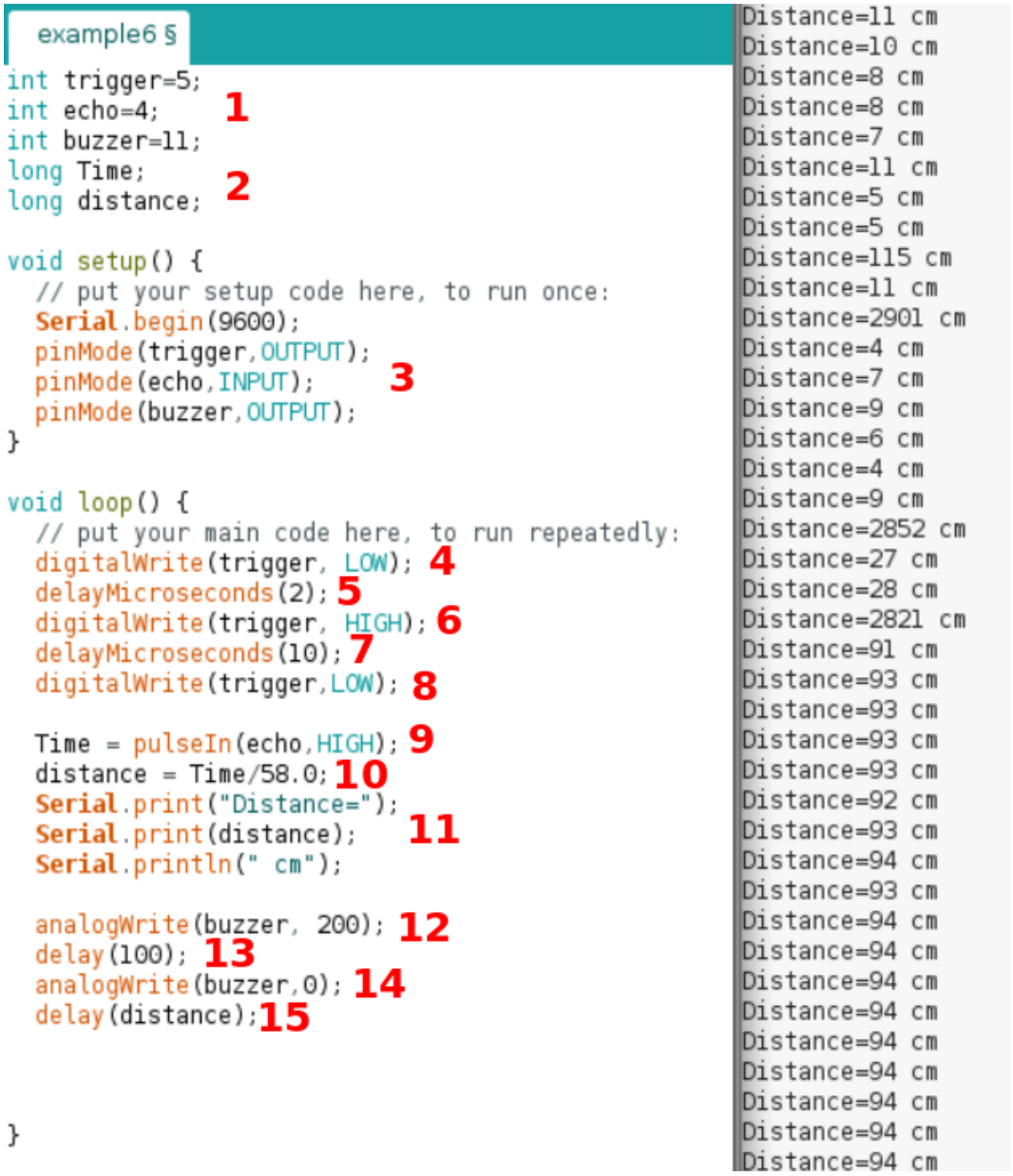


Figure 46: The code for example 6 written using the Arduino IDE software.

7. The program will be stopped for 10 μs and the *HIGH* signal level on the *trigger* pin will last.

8. The *LOW* signal level is set on the *trigger* pin to finish the releasing of the ultrasonic wave.

9. The *pulseIn* function measures the time duration of the chosen signal: *LOW* or *HIGH* in μs. According to the figure 44, the distance is proportional to the duration of *HIGH* signal on the *echo* pin.

10. The time duration between two *HIGH* signals is equal to the time needed to make

the wave travel to the object and back, that is, it traveled twice the distance. The distance can be calculated using the equation from documentation to HC-SR04:

$$d(cm) = \frac{t(\mu s)}{58} \tag{10}$$

or it can be calculated by using equation for the velocity:

$$v = \frac{2d}{t} \rightarrow d = v \cdot \frac{t}{2} = 0.034 \frac{\mu s}{cm} \cdot \frac{t(\mu s)}{2} \approx \frac{t(\mu s)}{58} \tag{11}$$

11. The value of distance is displayed in the *Serial Monitor*.
12. The piezo buzzer sound is adjustable by using the PWM signal. At the beginning the width of signal is set to 200.
13. This line blocks execution of the program for 100 ms and the piezo buzzer will generate the sound. You can modify this value for your needs.
14. This line turn off the piezo buzzer.
15. The delay between generating sound and silence depends on the distance. If distance is small the sound will be generated with higher frequency.

The lines between 4-10 will be repeated if you use another ultrasonic distance sensor. So that the code will grow. When you will use some parts of code many times, the good practise is to create the function. The function consists of:

type name(type argument)
{
code inside;
}

Example 6.2

The modified example 6 with own function usage is shown in Figure 47. The setup function is exactly the same as in example 6. The own function, which returns the value of distance from ultrasonic sensor was added after setup function. It consists of the following lines:

1. The declaration of function. First part is type of data, which function will return. The function should return distance, which is long value. Second part is a name of the function. The last part is located in brackets and it is declaration of variables, which user should transfer to function and temporary names of variables, which will be used inside the function.
2. The declaration of variable *Time* was moved to function body because this variable will be used only in this function.
3. This part is copied from example 6 (lines 4-10)

4. The word *return* determines what the function will return. For ultrasonic sensor, the distance calculated as time divided by 58 should be returned.

5. The line 5 shows function calling.

Example6.2

```
int trigger=5;
int echo=4;
int buzzer=11;
long distance;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(trigger,OUTPUT);
  pinMode(echo,INPUT);
  pinMode(buzzer,OUTPUT);
}

long Distance(int trigger, int echo) 1
{
  long Time; 2

  digitalWrite(trigger, LOW);
  delayMicroseconds(2);
  digitalWrite(trigger, HIGH);     3
  delayMicroseconds(10);
  digitalWrite(trigger,LOW);

  Time = pulseIn(echo,HIGH);
  return Time/58.0; 4
}

void loop() {
  // put your main code here, to run repeatedly:
  distance=Distance(trigger, echo); 5
  Serial.print("Distance=");
  Serial.print(distance);
  Serial.println(" cm");

  analogWrite(buzzer, 200);
  delay(100);
  analogWrite(buzzer,0);
  delay(distance);
}
```

```
Distance=2 cm
Distance=4 cm
Distance=6 cm
Distance=9 cm
Distance=14 cm
Distance=17 cm
Distance=21 cm
Distance=24 cm
Distance=26 cm
Distance=27 cm
Distance=29 cm
Distance=31 cm
Distance=33 cm
Distance=36 cm
Distance=41 cm
Distance=43 cm
Distance=47 cm
Distance=48 cm
Distance=50 cm
Distance=3184 cm
Distance=3168 cm
Distance=3177 cm
Distance=3178 cm
Distance=3169 cm
Distance=3181 cm
Distance=3181 cm
Distance=3164 cm
Distance=3159 cm
Distance=3156 cm
Distance=3171 cm
Distance=3160 cm
Distance=3179 cm
Distance=3158 cm
Distance=3165 cm
Distance=3150 cm
Distance=3197 cm
Distance=3167 cm
Distance=3175 cm
Distance=3166 cm
Distance=3150 cm
Distance=3154 cm
```

Figure 47: The code of example 6.2 prepared in the Arduino IDE.

The Snap4Arduino has only very basic functions, so that it is impossible (or very hard depending on the point of view - however not intended for a novice user) to write own

functions to measure the distance from ultrasonic sensor. The special libraries with this function can be easily downloaded from GitHub repository:

1. You can find many kinds of these libraries in GitHub repositories. One of them is: https://github.com/mdcanham/Snap4Arduino-Firmata/tree/master/Ultrasound_HC_SR04Firmata.
2. After downloading the repository, extract it to the *libraries* directory located inside the Arduino IDE installed directory.
3. Then unpack the contents of the archive. After unpacking, you should have directories and files inside the created folder, such as in Figure 48.
4. Open the *Ultrasound_HC_SR04Firmata.ino* in Arduino IDE software. Depending on the libraries the Firmata file can be named a little different.
5. Compile and upload to Arduino board Firmata firmware.
6. Open the Snap4Arduino software.
7. Drag and drop the *Ultrasound_HC_SR04_blocks.xml* file to Snap4Arduino software.

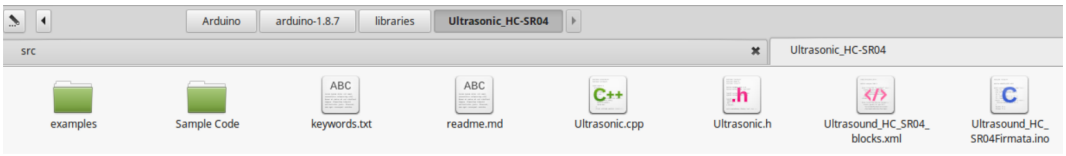


Figure 48: The interior of the Ultrasonic library folder.

The libraries for ultrasonic sensors forces user to connect the sensor to the specific pins. For this purpose, the electronic circuit was modified and shown in Figure 49.

The code for demonstration of parking sensor working using the Snap4Arduino is shown in Figure 50.

The code consists of following lines:

1. The *buzzer* variable was created and the pin number, to which buzzer is connected, is transferred to variable.
2. The *ultrasound distance reading* function returns the distance value. This value is transferred to created variable called *distance*.
3. The piezo buzzer sound is adjustable by using the *PWM* signal. At the beginning the width of signal is set to 200.
4. This line blocks program execution for 0.1 s and the piezo buzzer generates the sound.
5. This line turns off the piezo buzzer.

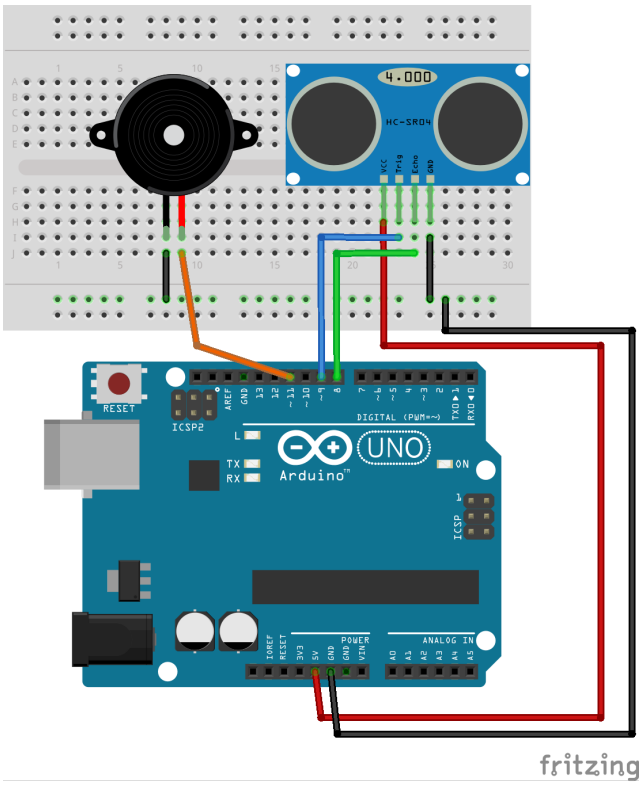


Figure 49: The electronic circuit for example 6 for Snap4Arduino software.

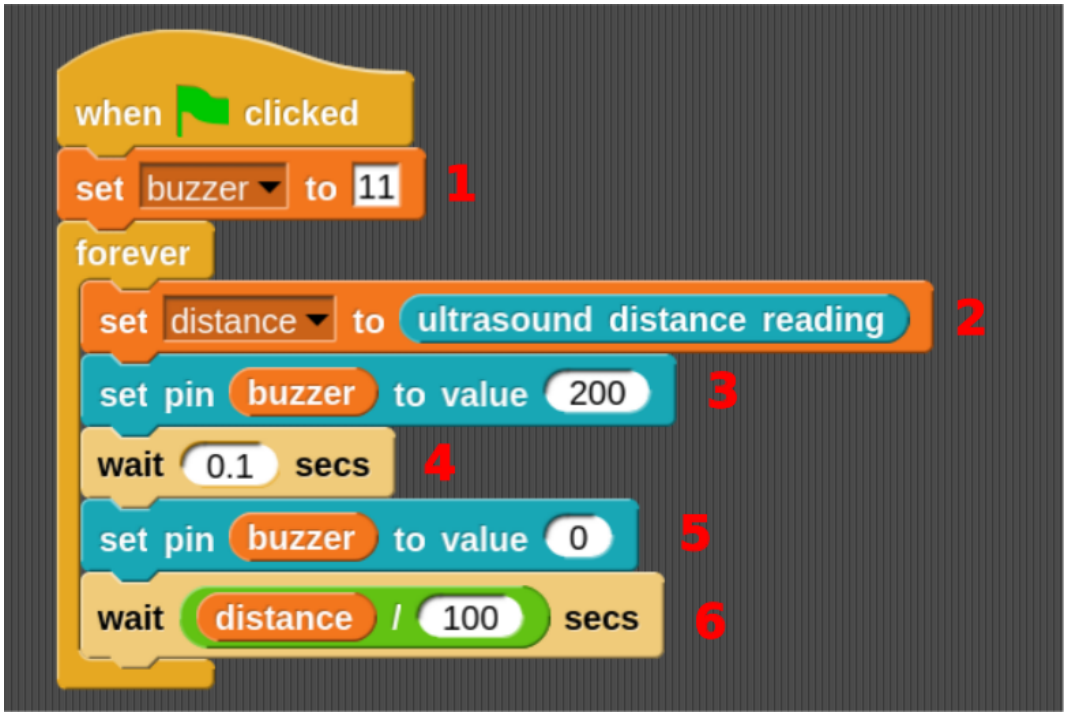


Figure 50: The code for example 6 written in Snap4Arduino.

6. Delay between sound generation and silence depends on the distance. If distance is small the sound will be generated with higher frequency.

Example 7

The sound sensor is an another popular sensor used in many projects. This sensor is easy to use because it returns just digital state. During this exercise, the sound sensor V2 is used, which returns 0 when sound is detected and 1 when sound is not detected. When sound is detected, the LED should light up. The electronic circuit scheme is shown in Figure 51.

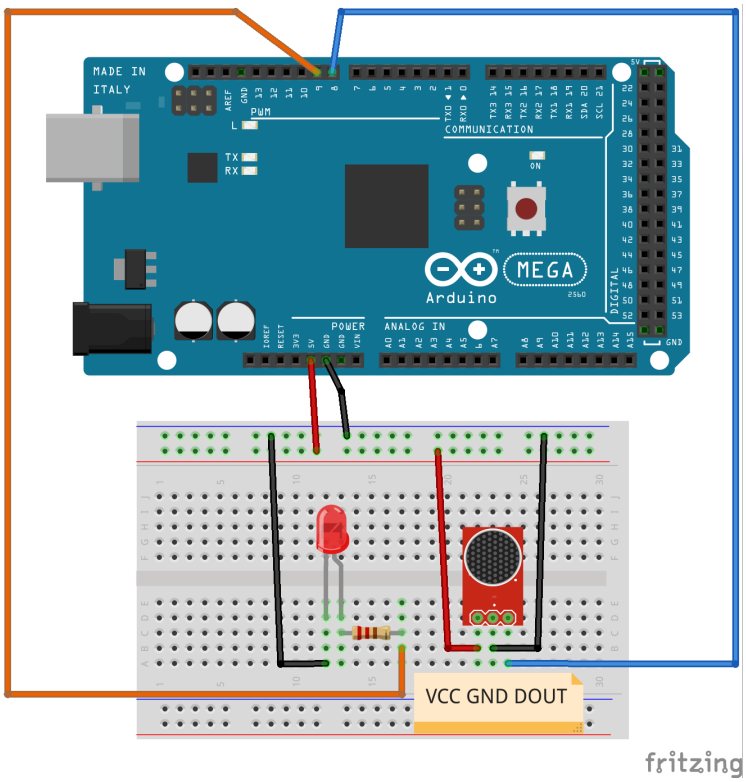


Figure 51: The electronic circuit for example 7.

The code for Arduino IDE is shown below and consists of the following lines:

```
int soundSensor=8;
int LED=9;
int sound=0;
boolean LEDstatus=false;

void setup() {
  // put your setup code here, to run once:
  pinMode(soundSensor,INPUT); 1
  pinMode(LED,OUTPUT);
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(soundSensor)==0) 2
  {
    if(LEDstatus) 3
    {
      digitalWrite(LED,LOW); 4
      LEDstatus=false;
    }
    else 5
    {
      digitalWrite(LED,HIGH); 6
      LEDstatus=true;
    }
  }
}
```

1. The sound sensor returns 0 or 1 value depending on detection of sound. So that pin number 8 should be *INPUT*. The LED will be turned on/off, so that the pin number 9 will be *OUTPUT*.

2. The value from sensor is read by using *digitalRead* function. If this value is equal 0, the sound is detected.

3. If the LED was turned on (LEDstatus equal true), the lines in if block will be executed.

4. The LED will be turned off and the LEDstatus will be equal false.

5. If the LED was turned off, the lines in else block will be executed.

6. The LED will be turned on and the LEDstatus will be equal to true.

When the program will be uploaded to Arduino board, the LED should light up after clapping hands or speaking loudly.

The analogous code for the Snap4Arduino software is shown in Figure 52 and consists of the following lines:

1. The *LED* and *SoundSensor* are created and the pin number of connected LED and sound sensor are put to these variables, respectively.

2. If the sound sensor detects sound (the value will be equal to *false*), the LED diode will light up.

3. If the *LEDstatus* variable had *true* value previously, the *LEDstatus* will change the value to *false*.

4. If the *LEDstatus* variable had *false* value previously, the *LEDstatus* will change the value to *true*.

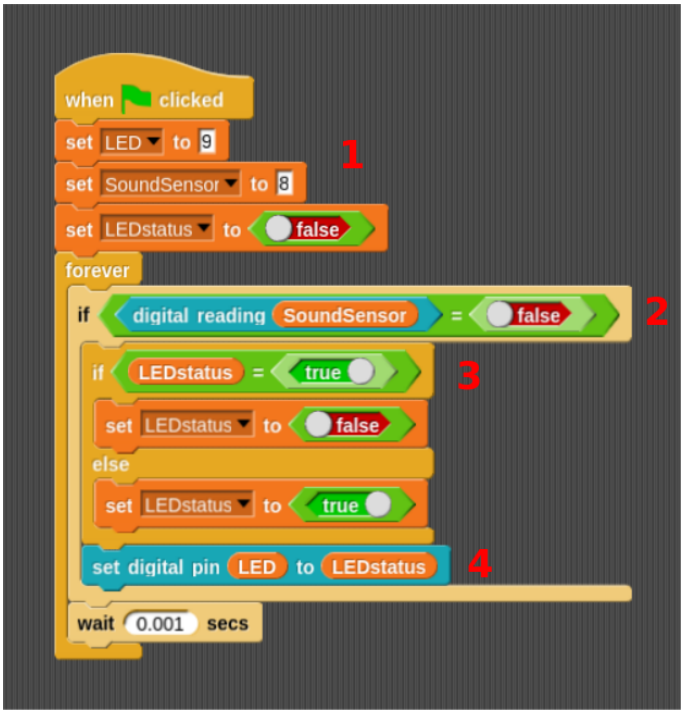


Figure 52: The code for example 7 written in Snap4Arduino.

5. The *LEDstatus* value will be set to the pin connected with LED, so the LED will be turned on/off.

Example 8

Previous sensor was an example of digital sensor. This example focuses on UV sensor (determining the ultra-violet radiation intensity on sunny days), which returns analog value proportional to the UV intensity. The analog value should be transformed to voltage and then the UV index can be calculated on the basis of Figure 54. In this example, the returned voltage from sensor and UV index will be displayed on LCD screen, which is connected to Arduino board as is shown in Figure 53.

The code for Arduino IDE is shown below and consists of the following lines:

```
#include <LiquidCrystal.h> 1
#define RS 22
#define E 24
#define D4 26 2
#define D5 28
#define D6 30
#define D7 32

int UV_pin = A0; 3
int UV_analogVal = 0;
double UV_value = 0.0;
int UV_index = 0;
```

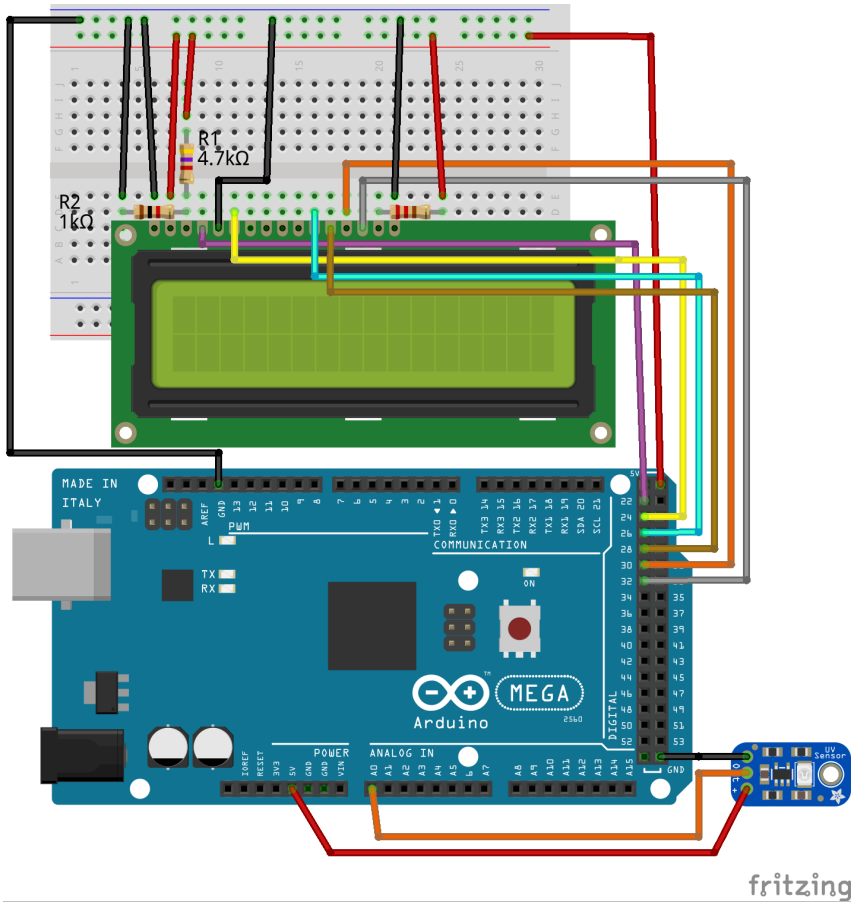


Figure 53: The electronic circuit for example 8.

```
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); 4


void setup() {
  // put your setup code here, to run once:
  pinMode(UV_pin, INPUT); 5
  lcd.begin(16,2); 6
  lcd.clear(); 7
}

void loop() {
  // put your main code here, to run repeatedly:
  UV_analogVal=analogRead(UV_pin); 8
  UV_value=UV_analogVal*5.0/1024.0; 9
  UV_index=map(UV_value,0.99,2.6,0,15); 10

  lcd.clear(); 11
  lcd.setCursor(0,0); 12
  lcd.print("A0=");
  lcd.print(UV_analogVal);
```

```
  lcd.print(" U=");
  lcd.print(UV_value);
  lcd.print("V");
  lcd.setCursor(0,1);
  lcd.print("Index=");
  lcd.print(UV_index);
  delay(100);
}
```

1. First, the library for LCD screen is added.
2. The RS, E, D4, D5, D6 and D7 variables are defined. They keep information about pin number to which the corresponding output of LCD has been connected on the Arduino board. The word *define* is preprocessor definition used to define named values. These values does not occupy random access memory (RAM) in microcontroller, but these are just inserted to code during compilation. These lines could be written as e.g. *int RS=22;* alternatively. However in this way, they will occupy some space in RAM during program execution (for int type, single value will occupy 2 bytes). RAM in microcontrollers is often very limited and priceless. For example Arduino UNO board (with ATmega328 microcontroller) has 2000 bytes of SRAM (static RAM) memory. In the next chapter, the advanced sensor will be described and values will be often defined by using *define* processor definition.
3. The needed variables are defined, which will store information about read analog value from pin, calculated UV value and index.
4. The lcd object of LiquidCrystal class is created.
5. The analog pin connected to UV sensor is defined as *INPUT*, because UV sensor will return the voltage. This value will be read by Arduino board and converted to number between (0;1023) using internal Analog to Digital converter (ADC).
6. The LCD screen is defined. The *begin* function needs the number of columns (16) and rows (2).
7. The LCD screen is cleared.
8. The analog value returned from UV sensor is read and stored in *UV_analogVal* variable.
9. The analog value is transformed to voltage. The more details about analog values can be found in Section 5.3.
10. The UV index is calculated according with the Figure 54. The voltage keeps value between (0.99;2.6)V for room temperature what corresponds (0;15) UV index. The *map* function allows to quickly transform the value between one to another range. The first argument of this function is value. Then the actual range of variable should be put e.g. (0.99;2.6). The last two arguments are a new range of variable e.g. (0;15).

11. The LCD screen is cleared.

12. The cursor on LCD screen is set to (0;0) position. First value means the column number numerated by starting from 0. The next value is a row number.

13. The *print* function allows to display chars on LCD screen. This function accepts numerical values and string values too.

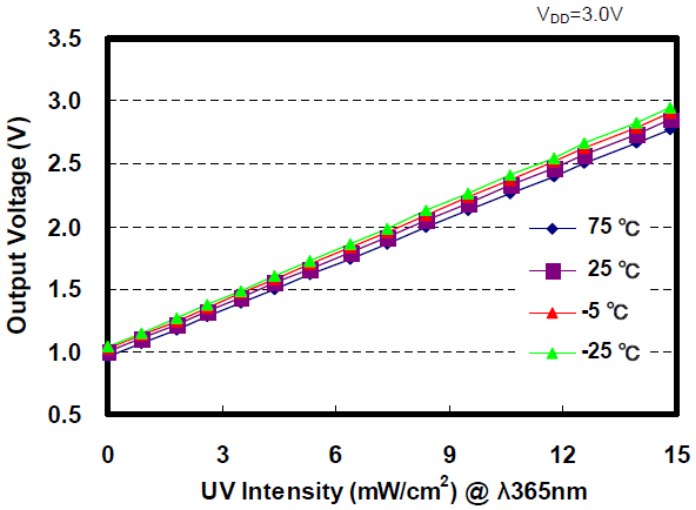


Figure 54: The dependence between returned voltage from sensor and UV index. Source: [19].

The Arduino board can work without PC computer connected. The standard 5 V Power Bank can be connected to USB port of the Arduino board as a power supply. Then the UV sensor can be tested outdoors to see the UV intensity changes on sunny weather.

The Snap4Arduino has only very basic functionality built-in, therefore the special Firmata should be uploaded to Arduino board in similar way as for the ultrasonic sensor. The special libraries with LCD functions can be downloaded from website:

http://blog.s4a.cat/2015/03/24/LCD-Library-for-Snap4Arduino.html

The libraries for LCD screen forces user to connect the LCD screen to specific pins: RS=12, E=11, D4=5, D5=4, D6=3 and D7=2.

The analogous code for the Snap4Arduino is shown in Figure 55 and consists of the following lines:

1. The needed variables are defined, which will store information about read analog value from pin, calculated UV value and index.

2. The analog value returned by UV sensor is read and stored in the *UV_analogVal* variable.

3. The analog value is transformed to the voltage. The more details about analog values can be found in Section 5.3.

4. The UV index is calculated according to the Figure 54. The voltage keeps value between (0.99;2.6)V for room temperature which corresponds to (0;15) UV index. The voltage can be transformed to index value by using simple function: $index = 9.32 \cdot Voltage - 9.23$.

5. The LCD screen is cleared.

6. The cursor in LCD screen is set to position (0,0) using *set cursor at* function.

7. The values are displayed on LCD screen using *print* function.

For older version of Snap4Arduino, the *print* function can not work properly. Snap4Arduino needs to send commands to Arduino board for all the time. Therefore, the Arduino board cannot be disconnected from computer during testing the UV sensor.

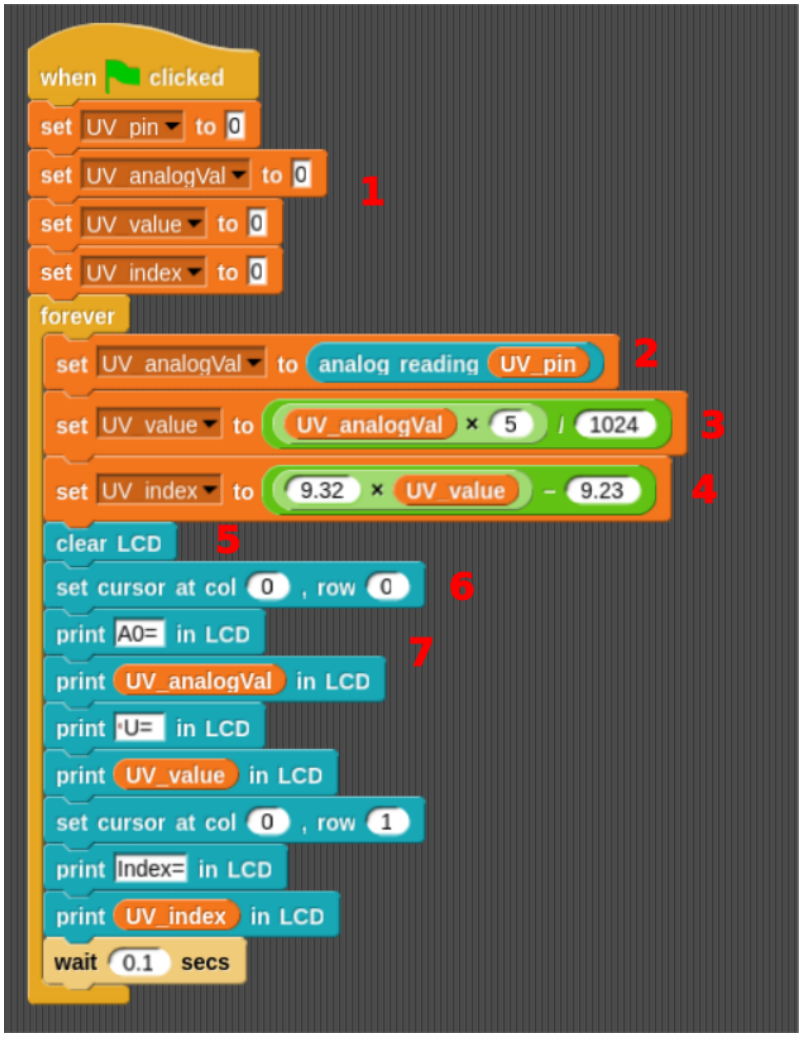


Figure 55: The code for example 8 written in Snap4Arduino.

6.2 The advanced sensors

The advanced sensors return a digital values, which can be read by Arduino board using a serial communication interfaces (buses). The serial communication interface is a group of lines, which are used to send data between connected devices. They can be divided in two groups: synchronous and asynchronous. The first group use additional serial clock line, which synchronize all devices connected to communication interface. The most popular synchronous bus are SPI (*Serial Peripheral Interface*) and I2C (*Inter-Integrated Circuit*), which is also marked as I^2C or IIC. The most popular asynchronous serial communication interface are UART (*Universal asynchronous receiver-transmitter*) and 1-wire.

6.2.1 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface consists on four lines:

- SCK - serial clock, which allows to synchronize devices,
- MOSI (*Master Output Slave Input*) - line, which transmits bits from master device to all slave devices,
- MISO (*Master Input Slave Output*) - line, which transmits bits from slave devices to master device,
- SS (*Slave select*) - line, which activate communication with chosen slave device. The slave device starts to listen and respond when a low value is set on its SS line. This line is sometimes called *Chip select* and then marked as CS. To point that active state is low in line name frequently a line above the text is placed, like $\overline{\text{SS}}$ or $\overline{\text{CS}}$.

The schematic connection between devices (block diagram) is shown in Figure 56. The master is only one and this device receives information from slave devices on demand. The master device generates the clock signal, which will synchronize all devices. It is possible to connect many slave devices (e.g. sensors) to master (e.g. Arduino board) but all slave devices need to have separate SS/CS lines. It is worth to mention that SPI is full-duplex bus. It means that data can be transmitted and received by device in the same time.

The dedicated library to use SPI bus in Arduino IDE is called *SPI* and this library is installed in Arduino IDE software by default. The most important functions from *SPI* library are given in Table 4.

Most sensors have ready libraries for Arduino IDE, which can be found in `https://www.arduino.cc` or github repository. The next example will show how use ready libraries for sensors, which send data using SPI device.

Example 9

During this example, the accelerometer LIS3DH will be used. This sensor sends data using SPI or I2C buses. The sensor should be connected according to the Figure 57.

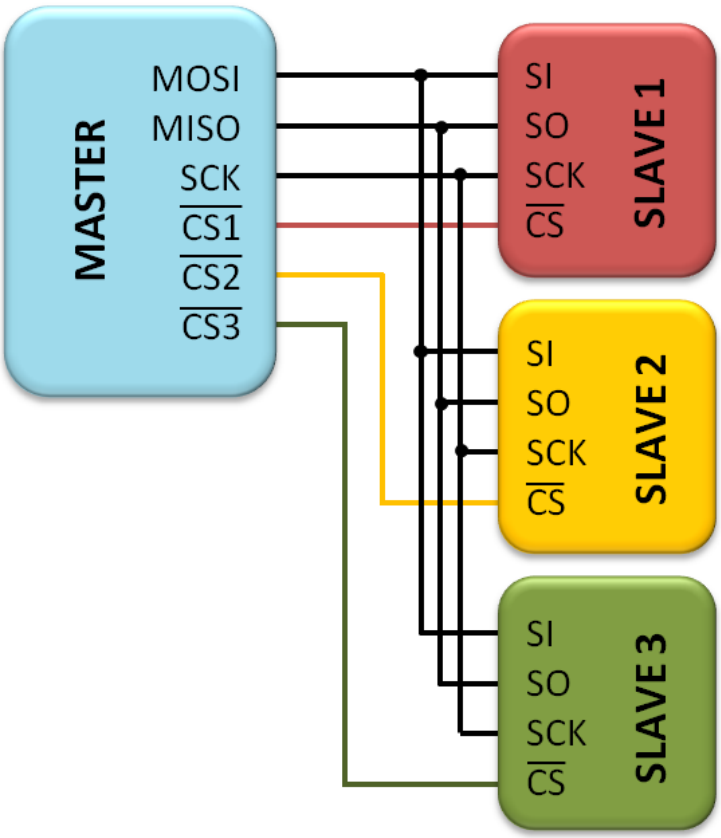


Figure 56: The schematic connection between devices using SPI bus. Source: http://extronic.pl/content/60-kurs-xmega-interfejs-spi .

Function	Description
begin	Initialization of SPI bus
setBitOrder(MSBFIRST/LSBFIRST)	Choose order of a bit transmitted by SPI bus from two options: LSBFIRST (least-significant bit first) or MSBFIRST (most-significant bit first).
transfer(byte)	Sending the byte of data.
transfer(0)	Reading data from slave devices.

Table 4: The most popular commands for SPI bus.

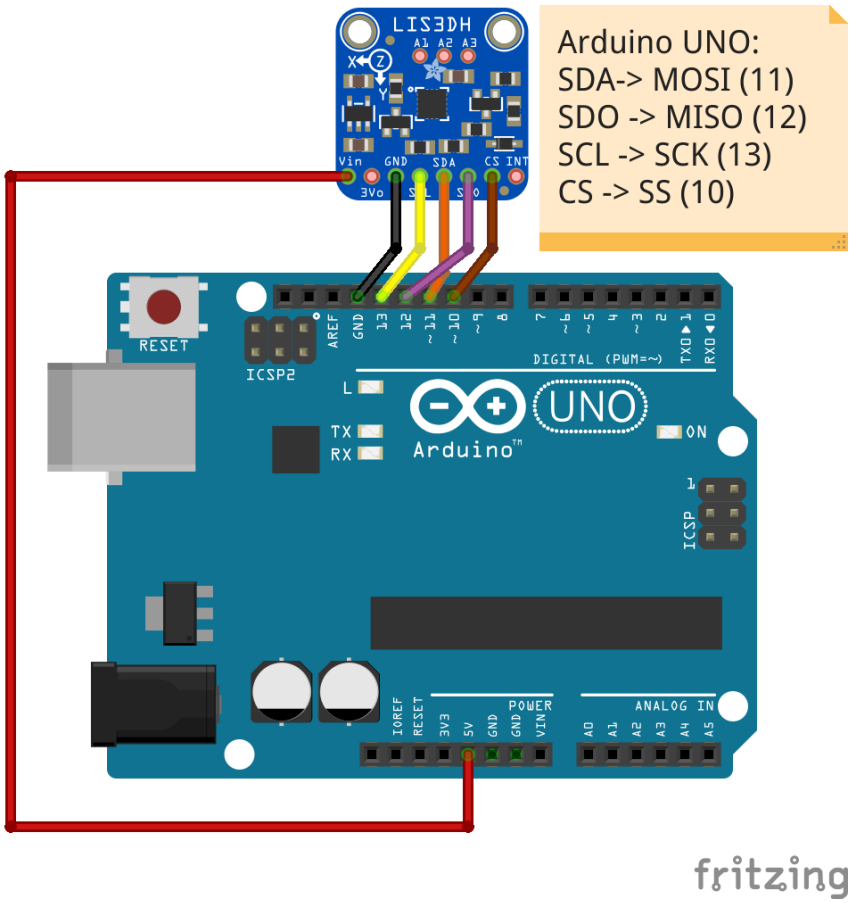


Figure 57: The electronic circuit for example 9.

The dedicated library for LIS3DH can be downloaded from github repository: https://github.com/adafruit/Adafruit_LIS3DH . When you download library, unzip it inside *library* folder in Arduino IDE installed path. Then, open Arduino IDE software once again. Most ready libraries have examples to start working with sensors. You can find them in Arduino IDE software by choosing *File* → *Examples* → *Adafruit LIS3DH* → *acceldemo* . This example can be helpful to understand how to use a library. Based on example, you can create your own code as is shown here:

```
// Basic demo for accelerometer readings from Adafruit LIS3DH

#include <SPI.h> 1
#include <Adafruit_LIS3DH.h>
#include <Adafruit_Sensor.h>

#define LIS3DH_CS 10 2

Adafruit_LIS3DH lis = Adafruit_LIS3DH(LIS3DH_CS); 3

void setup(void)
{
  Serial.begin(115200);
  Serial.println("LIS3DH test!");

  if (! lis.begin(0x18))  4
  {
    Serial.println("Couldn't start");
    while (1) yield();
  }
  Serial.println("LIS3DH found!");

  Serial.print("Range = "); Serial.print(2 << lis.getRange());
  Serial.println("G");

  // lis.setDataRate(LIS3DH_DATARATE_50_HZ);
  Serial.print("Data rate set to: "); 5
  switch (lis.getDataRate())
  {
    case LIS3DH_DATARATE_1_HZ: Serial.println("1 Hz"); break;
    case LIS3DH_DATARATE_10_HZ: Serial.println("10 Hz"); break;
    case LIS3DH_DATARATE_25_HZ: Serial.println("25 Hz"); break;
    case LIS3DH_DATARATE_50_HZ: Serial.println("50 Hz"); break;
    case LIS3DH_DATARATE_100_HZ: Serial.println("100 Hz"); break;
    case LIS3DH_DATARATE_200_HZ: Serial.println("200 Hz"); break;
    case LIS3DH_DATARATE_400_HZ: Serial.println("400 Hz"); break;

    case LIS3DH_DATARATE_POWERDOWN:
      Serial.println("Powered Down"); break;
    case LIS3DH_DATARATE_LOWPOWER_5KHZ:
      Serial.println("5 Khz Low Power"); break;
    case LIS3DH_DATARATE_LOWPOWER_1K6HZ:
      Serial.println("16 Khz Low Power"); break;
  }
}

void loop()
{
  lis.read();      // get X Y and Z data at once 6
  // Then print out the raw data
  Serial.print("X:  "); Serial.print(lis.x); 7
  Serial.print("  \tY:  "); Serial.print(lis.y);
  Serial.print("  \tZ:  "); Serial.print(lis.z);
```

```
  /* Or.... get a new sensor event, normalized */
  sensors_event_t event; 8
  lis.getEvent(&event);

  /* Display the results (acceleration is measured in m/s^2) */
  Serial.print("\t\tX: "); Serial.print(event.acceleration.x); 9
  Serial.print(" \tY: "); Serial.print(event.acceleration.y);
  Serial.print(" \tZ: "); Serial.print(event.acceleration.z);
  Serial.println(" m/s^2 ");

  Serial.println();

  delay(200);
}
```

The code for Arduino IDE is shown above and consists of the following lines:

1. Adding *SPI* library.
2. Define, which pin will be used as a chip select. Different Arduino boards will have different pin, which is used as CS/SS. This pin can be arbitrarily chosen from any digital pin on board.
3. Initialize SPI bus on chosen CS pin.
4. Initialize LIS3DH sensor by using *begin* function. If function returns error (*false* value), than appropriate text will be printed in serial monitor.
5. Printing data rate in serial monitor.
6. Read the X, Y and Z values from sensor.
7. Printing read values in serial monitor.
8. Calculating coordinates of acceleration based on raw X,Y and Z data.
9. Printing results in serial monitor.

Example output after running code is shown in Figure 58.

If you compare official example with above code, you will see strong similarities. You can create your own code in similar way. If you are interested how exactly communication with sensor is realized, you can open downloaded library and analyze the source code. You will find the command from *SPI* library, which are listed in Table 4. Please remember that different Arduino boards have MISO, MOSI, SCK and SS pins realized in different digital pins. Therefore, you should check the pin numbers before you connect sensor to Arduino board.

```
LIS3DH found!
Range = 2G
Data rate set to: 400 Hz
X:  288         Y:  48          Z:  16144             X: 0.24         Y: 0.04         Z: 9.76 m/s^2

X:  288         Y:  64          Z:  16144             X: 0.14         Y: 0.02         Z: 9.69 m/s^2

X:  320         Y:  32          Z:  16048             X: 0.19         Y: 0.02         Z: 9.61 m/s^2

X:  352         Y:  0   Z:  16128             X: 0.22         Y: 0.02         Z: 9.66 m/s^2

X:  368         Y:  32          Z:  16032             X: 0.22         Y: 0.02         Z: 9.60 m/s^2

X:  288         Y:  48          Z:  16080             X: 0.18         Y: 0.05         Z: 9.60 m/s^2

X:  288         Y:  48          Z:  16112             X: 0.17         Y: 0.03         Z: 9.65 m/s^2

X:  336         Y:  16          Z:  16064             X: 0.24         Y: 0.00         Z: 9.67 m/s^2

X:  336         Y:  32          Z:  16080             X: 0.20         Y: 0.02         Z: 9.63 m/s^2

X:  368         Y:  16          Z:  16096             X: 0.22         Y: 0.01         Z: 9.64 m/s^2
```

Figure 58: The example output after running code from example 9.

6.2.2 Inter-Integrated Circuit (I2C)

The Inter-Integrated Circuit bus consists on two lines:

- SDA (*Serial Data Line*) is a data line, which is used to send data between master and slave devices.
- SCL (*Serial Clock Line*).

Both lines are connected to VCC by pull-up resistors. The value of resistance influences on the speed of transmission. The smaller the resistor value is used, the faster data transmission is possible but at a cost of the higher energy consumption and of course there is a limit: too low resistance may prevent the low state to be driven on the bus.

The example communication on I2C bus (protocol) is shown in Figure 59.

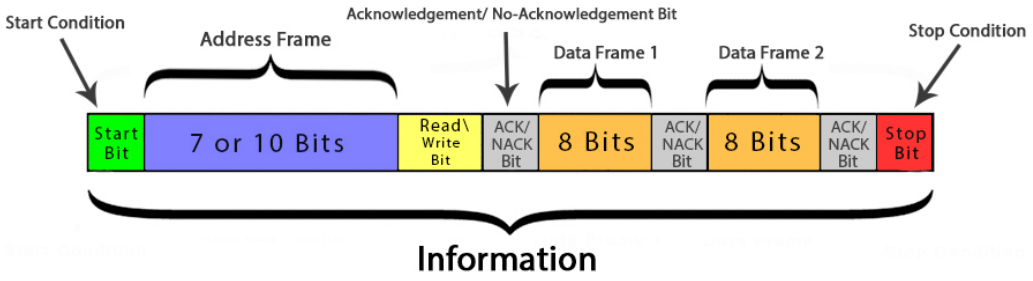


Figure 59: The I2C communication protocol. Source: [20].

The dedicated library to use I2C bus in Arduino IDE is called *Wire* and this library is installed in Arduino IDE software by default. The most important function from *Wire* library are given in Table 5.

Most sensors have ready libraries for Arduino IDE, which can be found in https://www.arduino.cc or github repository.

Function	Description
begin()	Initialization of I2C bus
beginTransmission(address)	Starting transmission with slave device. The function required the slave address.
write(data)	Sending the one byte of data to slave device.
endTransmission()	Finishing connection with slave device. The function returns value depending on how transmission was finished: **0** - no errors. **1** - the received data is too large. **2** - the device returned NACK after sending address. **3** - the device returned NACK after sending data. **4** - unknown error.
requestFrom(address,size of data)	This function is used to request data from slave device.
available()	This function checks if data are available to read.
read()	Reading one byte of data from slave devices.

Table 5: The most popular commands for I2C bus.

Example 10

This example will show how to use ready library to read value from pressure sensor MPL3115A2. This sensor sends data using I2C bus and it should be connected according to the Figure 60.

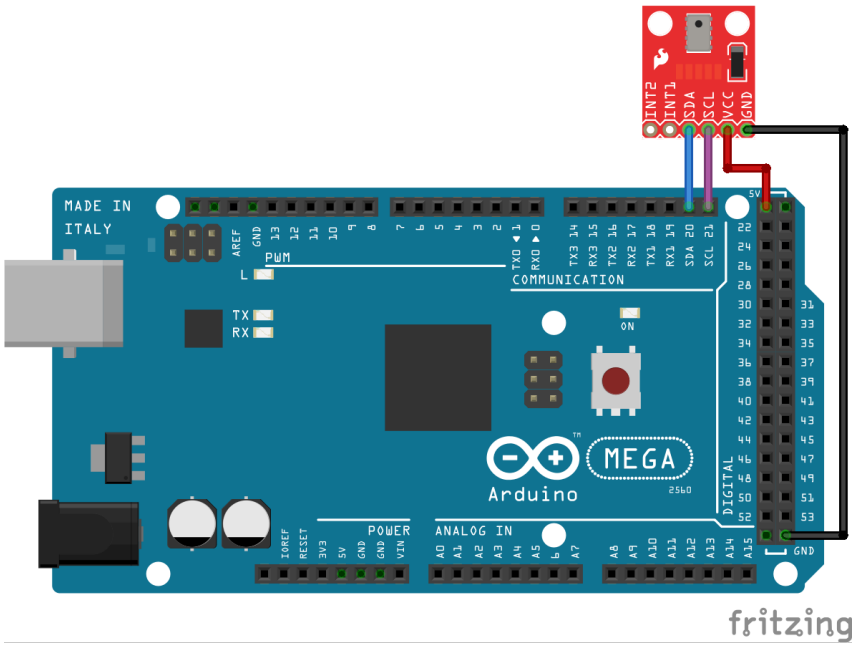


Figure 60: The electronic circuit for example 10.

The dedicated library for MPL3115A2 can be installed from libraries manager inside Arduino IDE software. After running the Arduino IDE software, you should choose *Tools* → *Manage Libraries....* Then, find the *SparkFunMPL3115A2* library and install it.

Most ready libraries have examples to start working with sensor. You can find them in Arduino IDE software by choosing *File* → *Examples* → *SparkFun MPL3115A2* → *SparkFunPressure* . This example can be helpful to understand how to use the library. Based on example, you can create your own code as is shown here:

```
#include <Wire.h>
#include "SparkFunMPL3115A2.h"

//Create an instance of the object
MPL3115A2 myPressure;

void setup()
{
  Wire.begin();         // Join i2c bus
  Serial.begin(9600);  // Start serial for output
  myPressure.begin(); // Get sensor online
  // Measure pressure in Pascals from 20 to 110 kPa
  myPressure.setModeBarometer();
  // Set Oversample to the recommended 128
  myPressure.setOversampleRate(7);
  // Enable all three pressure and temp event flags
  myPressure.enableEventFlags();
}

void loop()
{
  //Read the pressure and display in serial monitor
  float pressure = myPressure.readPressure();
  Serial.print("Pressure(Pa):");
  Serial.print(pressure, 2);
  Serial.println();
}
```

Example output after running code is shown in Figure 61.

If you compare official example with above code, you will see strong similarities. You can create your own code in similar way. If you are interested how exactly communication with sensor is realized, you can open downloaded library and analyze the source code. You will find commands from *I2C* library, which are listed in Table 5. Please remember that different Arduino boards have SDA and SCL lines realized in different pins. Therefore, you should check the pin numbers before you connect sensor to Arduino board.

Example 11

This example will show how to use *I2C* library, when the library to sensor is not available. In this example, the temperature sensor STM75 is connected to the Arduino board using ready shield.

If you do not have ready library, your first step should be searching for the documentation of sensor. You can find the documentation using web search engine in web browser

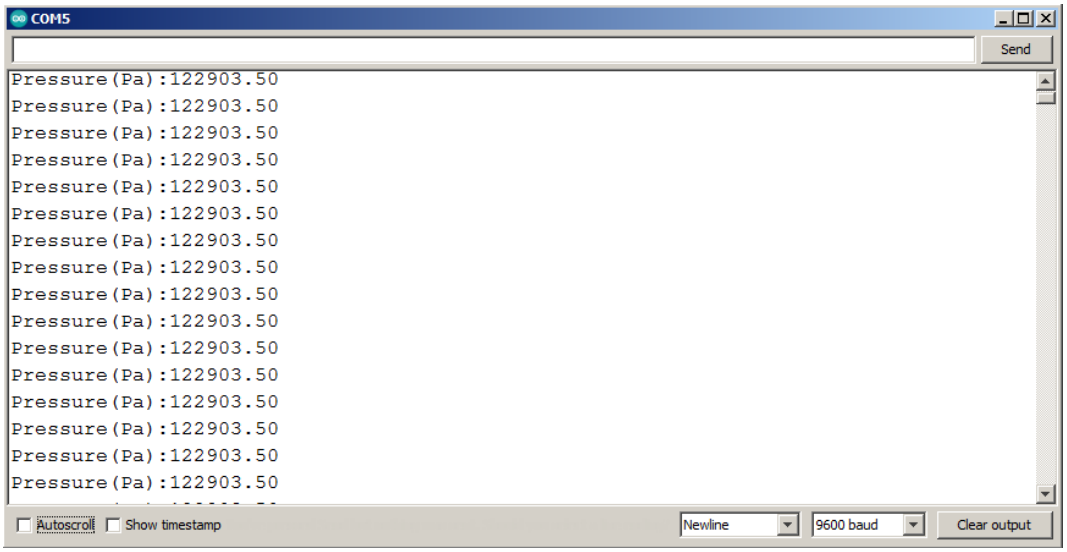


Figure 61: The example output after running code from example 10.

or directly on this website: http://fizyka.if.pw.edu.pl/%7Elabe/data/_uploaded/file/psw/laboratorium/Lab4/STM_LM75.pdf . Then, review the documentation.

The code should look like that:

```
#include <Wire.h>  1
#define adressTemp 0x48

float temp=0.0;
unsigned int msb=0, lsb=0;

void setup()
{
  Wire.begin(); 2
  Serial.begin(9600);
}

void loop() {
  Wire.beginTransmission(adressTemp);  3
  Wire.write(0x00);  4
  Wire.endTransmission();  5
  Wire.requestFrom(adressTemp, 2);  6
  while(Wire.available()) {  7
    msb = Wire.read();  8
    lsb = Wire.read();  9
    lsb = (lsb & 0x80 ) >> 7;  10
    temp = msb + 0.5 * lsb;  11
    Serial.print(" T=");
    Serial.println(temp);
  }
  delay(2000);
}
```

The code for Arduino IDE is shown in above and consists of the following lines:

1. The *Wire* library is included.
2. The I2C bus is initialized.
3. The transmission with slave device (temperature sensor) is started.
4. According to the documentation (see Figure 62), the temperature value will be returned by sensor if you send the value equal to 0 (hexadecimal 0x00). The 0 is the address of register, where the temperature value is stored.

Table 2. Register Functions

REGISTER NAME	ADDRESS (hex)	POR STATE (hex)	POR STATE (binary)	POR STATE (°C)	READ/ WRITE
Temperature	00	000X	0000 0000 0XXX XXXX	—	Read only
Configuration	01	00	0000 0000	—	R/W
T_{HYST}	02	4B0X	0100 1011 0XXX XXXX	75	R/W
T_{OS}	03	500X	0101 0000 0XXX XXXX	80	R/W

X = Don't care.

Figure 62: Register function description from documentation of STM75 sensor. The link to source of documentation is mentioned above.

5. Transmission with slave device is finished.
6. A request for 2 byte of data from slave device is sent.
7. If data are available, then while loop will be executed.
8. According to the documentation (see Figure 63), the temperature value will be returned with two bytes. First, the most significant part will be returned and store in *msb* variable.

Table 3. Temperature, T_{HYST}, and T_{OS} Register Definition

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = Don't care.

Figure 63: Bytes description from documentation to STM75 sensor. The link to source of documentation is mentioned above.

9. Then, the fractional part of the temperature will be returned and store in *lsb* variable.

10. Only the single most significant bit (8th bit) is important. Therefore, the rest bits should be replaced by 0. This is realized by AND operation. To understand this line better, let's follow an example. Suppose we read a value from temperature sensor: 10011010. We want to read only the 8 bit (marked red color). Remember that 0x80 is equal to 10000000 (binary). Let's make a prefix 0b to this value to distinguish from decimal, so it is 0b10000000. Therefore this operation will be realized as:

$$\begin{array}{l} 0b1001\ 1010 \\ \&\ 0b1000\ 0000 \\ \hline 0b1000\ 0000 \end{array}$$

Only 8th-bit has value 1 because *AND* operation gives 1 only if two bites are equal to 1. The result value is 0b1000 0000. After conversion to decimal value, the result will be equal to 128 but it should be equal to 1 or 0. Therefore, the 8th-bit should be moved to the right side. Right shift operation is performed by $value >> number\_of\_shifts$. The result of $0b10000000 >> 7$ operation is 0b00000001. After conversion to decimal value, the result will be equal to 1.

11. Combining two parts of temperature into one value.

Example output after running code is shown in Figure 64.

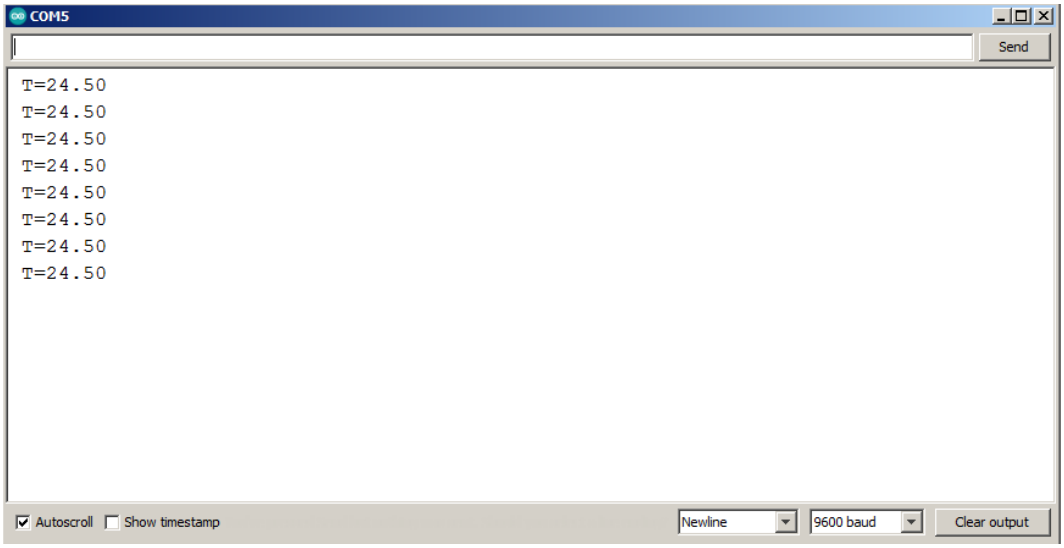


Figure 64: The example output after running code from example 11.

6.2.3 UART

The *Universal Synchronous and Asynchronous Receiver and Transmitter* (USART) is a interface, which consists of two lines:

- RxD - is a line used to receive data,
- TxD - is a line used to transmit data.

We will not utilize the "Synchronous" option of this interface (additional clock line is needed in such case), so we can treat this interface as UART. The UART data frame is shown in Figure 65. The frame consists of start bit, data bits and stop bit. The parity bit is optional.

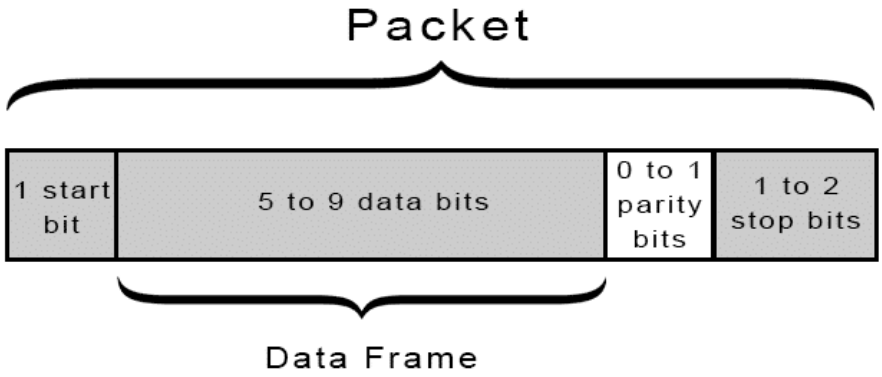


Figure 65: The UART data frame. Source: [21].

The UART should be configured by setting the speed of transmission in baud rate. The Arduino boards support the baud rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, and 115200. Supposing that you would like to send the packet, which consists of 1 bit start, 8 data bits and 1 bit stop (10 bits in total). If you configure the baud rate as 9600, the 960 packets (9600/10) can be sent by UART in 1 second.

The dedicated library to use UART in Arduino IDE is called *Serial* and this library is installed in Arduino IDE software by default. The most important function from *Serial* library are given in Table 6.

Function	**Description**
begin(baud rate)	Initialization of USART interface
print(string/numeric)	Sending the data
println(string/numeric)	Sending the data and the end line sign
available()	This function checks if data are available to read.
read()	Reading one byte of data.

Table 6: The most popular commands for USART interface.

Example 12

The aim of this example is to show how to use the GPS module (WaveShare NEO-6M/7M) connected to the Arduino Mega 2560 board (Figure 66).

In first step, the code will be prepared to read whole frame from GPS module:

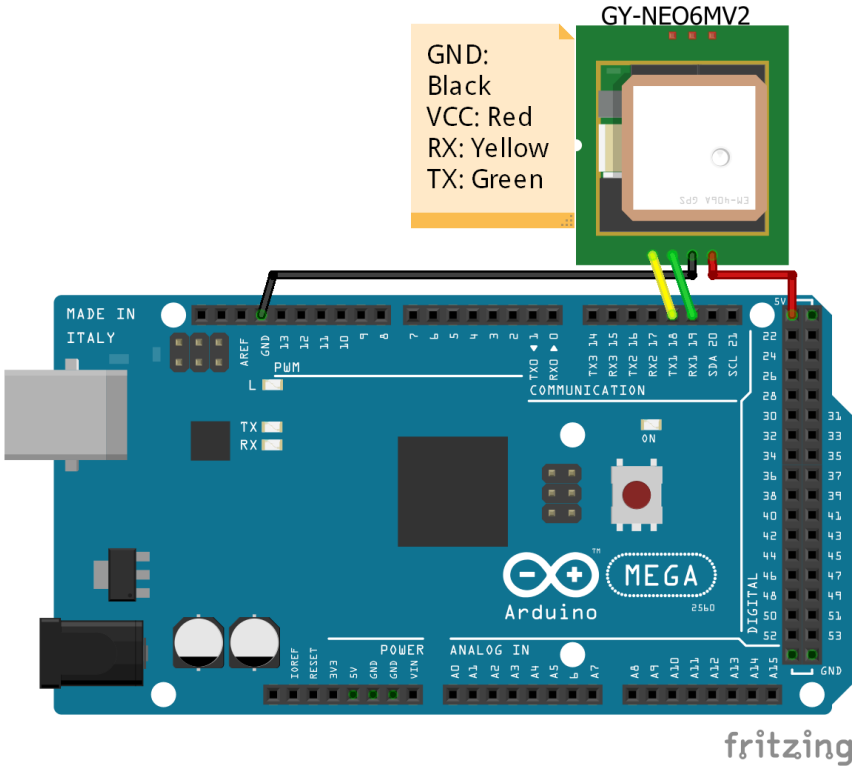


Figure 66: The electronic circuit for example 12.

```
void setup()
{
   Serial.begin(9600); 1
   Serial.println("WaveShare Neo-6M/7M module test");
   Serial1.begin(9600); 2
}

void loop()
{
   if(Serial1.available()) Serial.write(Serial1.read()); 3
}
```

The code for Arduino IDE is shown in above and consists of the following lines:

1. Initialization of serial monitor to show the collected data from GPS.
2. Initialization of *Serial 1*, where the GPS module is connected.
3. If data from GPS module are available, this data will be read and printed in serial monitor.

Example output after running code is:

```
WaveShare Neo-6M/7M module test
$GPGSV,3,3,11,27,05,274,,29,52,081,45,31,24,228,*40
$GPGLL,5222.14246,N,02100.70874,E,110311.00,A,A*61
$GPRMC,110312.00,A,5222.14246,N,02100.70874,E,0.009,,010621,,,A*77
```

Name	Example Data	Description
Sentence Identifier	**$GPGGA**	**Global Positioning System Fix Data**
Time	170834	
Latitude	18.5204, N	
Longitude	73.8567, E	
Fix Quality: - 0 = Invalid - 1 = GPS fix - 2 = DGPS fix	1	Data is from a GPS fix
Number of Satellites	5	5 Satellites are in view
Horizontal Dilution of Precision (HDOP)	1.5	Relative accuracy of horizontal position
Altitude	560.2, M	280.2 meters above mean sea level
Height of geoid above WGS84 ellipsoid	-34.0, M	-34.0 meters
Time since last DGPS update	blank	No last update
DGPS reference station id	blank	No station id
Checksum	*75	Used by program to check for transmission errors

[4]

Figure 67: The GPGGA (*Global Positioning System Fix Data*) frame. Source: [22].

```
$GPVTG,,T,,M,0.009,N,0.017,K,A*2C
$GPGGA,110312.00,5222.14246,N,02100.70874,E,1,07,1.18,123.3,M,35.4,M,,*58
```

The first world (GPGSV, GPGLL, GPRMC, GPVTG, GPGGA) is a id of GPS frame. The different frames present different data. The most important frame is GPGGA, which is shown in Figure 67. The parameters shown in row are separated by comma.

The previous code will be modified to show the time, latitude, longitude, quality of signal, number of satellites, precision and height:

```
int h=0, m=0, s=0;
long int t=0;

void setup()
{
   Serial.begin(9600);
   Serial.println("WaveShare Neo-6M/7M module test");
   Serial1.begin(9600);
```

```
    Serial.println("Time \t Latitude \t Longitude \t Quality \t Number of
    Satellites \t Precision \t Height");
}

void loop()
{
    if(Serial1.available())
    {
      String element=Serial1.readStringUntil(','); 1
      if(strstr(element.c_str(),"GPGGA")!=NULL) 2
      {
          String time=Serial1.readStringUntil(','); 3
          String width=Serial1.readStringUntil(',');
          String N=Serial1.readStringUntil(',');
          String lenght=Serial1.readStringUntil(',');
          String S=Serial1.readStringUntil(',');
          String quality=Serial1.readStringUntil(',');
          String nSatelites=Serial1.readStringUntil(',');
          String precision=Serial1.readStringUntil(',');
          String high=Serial1.readStringUntil(',');

          t=atol(time.c_str()); 4
          h=t/10000; 5
          m=(t-h*10000);
          m=m/100;
          s=t-h*10000-m*100;
          Serial.print(h); 6
          Serial.write(':');
          Serial.print(m);
          Serial.print(':');
          Serial.print(s);
          Serial.write('\t');
          Serial.write(width.c_str());
          Serial.write('\t');
          Serial.write(lenght.c_str());
          Serial.write('\t');
          Serial.write(quality.c_str());
          Serial.write('\t');
          Serial.write(nSatelites.c_str());
          Serial.write('\t');
          Serial.write(precision.c_str());
          Serial.write('\t');
          Serial.write(high.c_str());
          Serial.write('\n');
      }
    }
}
```

The code for Arduino IDE is shown above and consists of the following lines:

1. Reading data until comma sign and writing it to *element* variable.
2. The *strstr* function checks if GPGGA word exists in *element*. If yes, the function returns the localisation of word. Additionally, the *c_str* function is used to convert

the *element* variable from *String* data type (specific data type in Arduino software) to C-standard *string* data type (consisting of characters array ending with null char).

3. Reading the another parameter from GPGGA frame.
4. Time is converted from string to long integer value.
5. Time is divided to hours, minutes and seconds.
6. The read values are printed in serial monitor.

Example output after running the code looks like:

```
WaveShare Neo-6M/7M module test
Time    Latitude    Longitude    Quality    Number of Satellites    Precision
Height
11:31:12   5222.14246   02100.70874 1 07   1.27   116.4
```

6.2.4 1-wire

The 1-wire interface was developed by Dallas Semiconductor company. This bus consists of only one line. This interface works similar to I2C bus but the 0 and 1 bits are defined by time duration of low signal. The 1-wire protocol is shown in Figure 68. The 1-wire bus works slower than I2C. Maximal speed is 16kbit/s.

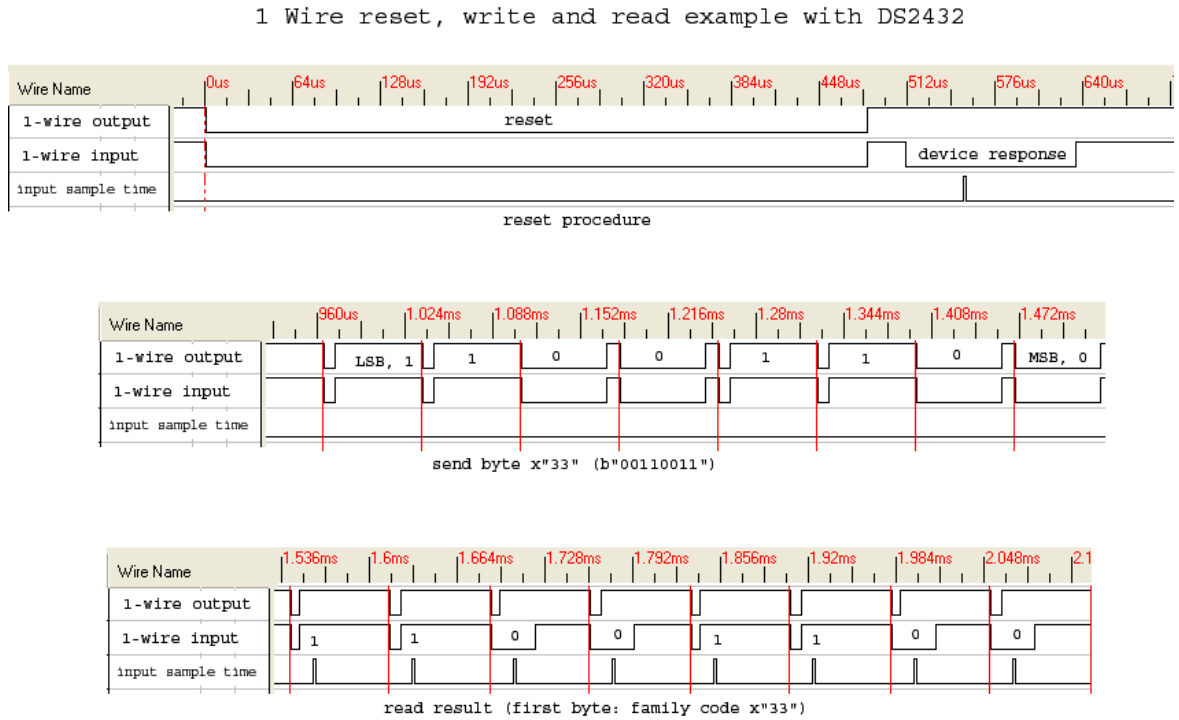


Figure 68: The 1-wire protocol

The dedicated library to use 1-wire bus in Arduino IDE is called *OneWire* and this library should be installed in Arduino IDE software. The most important function from *OneWire* library are given in Table 7.

Function	Description
search(byte array)	Returns addresses of modules connected to the 1-wire bus
reset()	Reset
select(address)	This function selects the slave device.
write(data, parasitic power - 1 or 0)	This function sends the data for slave device
read()	This function reads the data from slave device

Table 7: The most popular commands for 1-wire bus.

Example 13

During this example the digital thermometer (DS18B20) will be used. The program will read the actual temperature from sensor using the 1-wire bus. The DS18B20 should be connected to the Arduino board according to Figure 69.

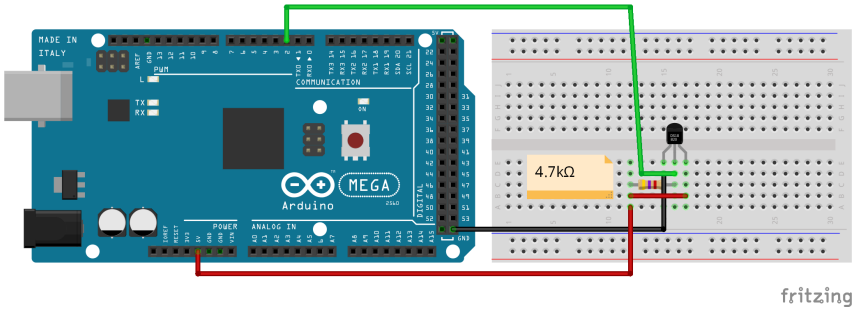


Figure 69: The electronic circuit for example 13.

The dedicated library for DS18B20 can be installed from libraries manager inside Arduino IDE software. After running the Arduino IDE software, you should chose *Tools* → *Manage Libraries...*. Then, find the *DallasTemperature* library and install it.

You can find example for *DallasTemperature* library in Arduino IDE software by choosing *File* → *Examples* → *DallasTemperature* → *Simple* . This example can be helpful to understand how to use the library. Based on example, you can create your own code as is shown here:

```
#include <Wire.h>
#include <OneWire.h> 1
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 3 2

OneWire oneWire(ONE_WIRE_BUS); 3
DallasTemperature sensors(&oneWire); 4
```

```
void setup()
{
   Serial.begin(9600);
   sensors.begin();   5
}

void loop()
{
   sensors.requestTemperatures();   6
   Serial.println(sensors.getTempCByIndex(0));   7
   delay(100);
}
```

The code for Arduino IDE is shown in above and consists of the following lines:

1. Adding necessary libraries for 1-wire bus and DS18B20 sensor.
2. Defining, which pin will be used for 1-wire bus.
3. Creating object of OneWire class.
4. Creating object of DallasTemperature class.
5. Initialization of DS18B20 sensor.
6. Request for temperature value from DS18B20 sensor.
7. Reading temperature value from sensor and displaying it in serial monitor.

Example output after running code is shown in Figure 70.

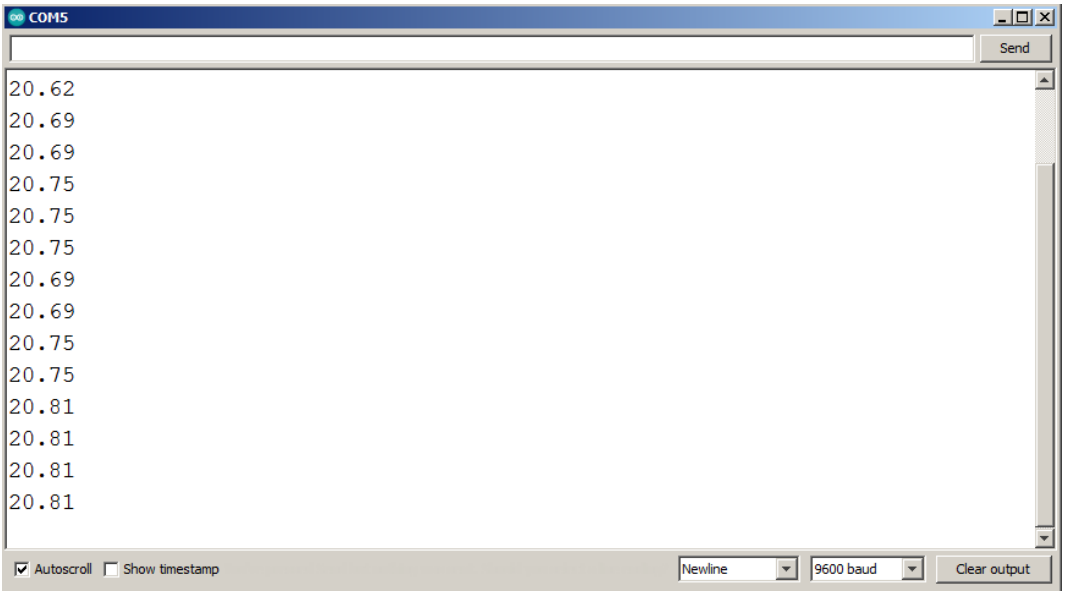
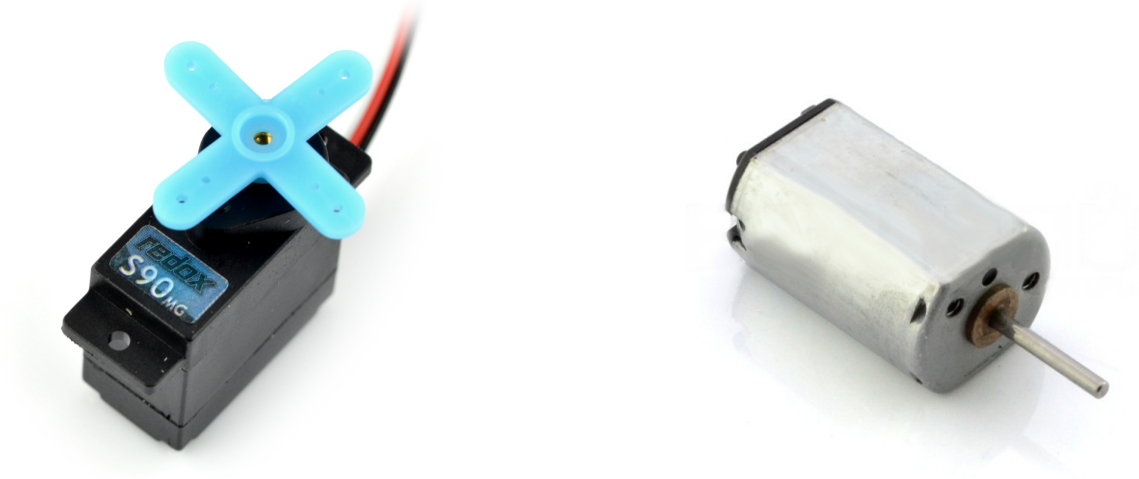


Figure 70: The example output after running code from example 13.



(a) Servo (b) DC motor

Figure 71: The example actuators. Source: [15].

7 Actuators

Actuators are elements making a movement. The basic actuators are: servo (Figure 71a) and motor (Figure 71b).

7.1 Servo

The servo is a motor with a gear that rotates by a given angle, usually in the range of $(0;180)^o$ or $(-90;90)^o$. The construction of the servo is shown in Figure 72.

The servo can be connected directly to the Arduino board. The three wires are brought out of the servo. The middle one (usually red) should be connected to the power supply (+5V). The black one is GND and the last wire (usually light colored: white/yellow) is the control that should be connected to the PWM pin.

The servo control is based on the PWM signal. The controller, located in the servo, reads the PWM signal and on its basis determines the angle by which the gear is to rotate. The idea of the servo operation is shown in Figure 73.

The servo can be controlled via Snap4Arduino and Arduino IDE. There is only one function in Snap4Arduino to control the servo: *set servo*. In this function, the angle should be between 0^o to 180^o. Therefore, the 0^o represents not a middle point but the minimum point in the right side.

The dedicated library to servo control in Arduino IDE is called *Servo* and this library is installed in Arduino IDE software by default. The most important function from *Servo* library are given in Table 8.

Example 14

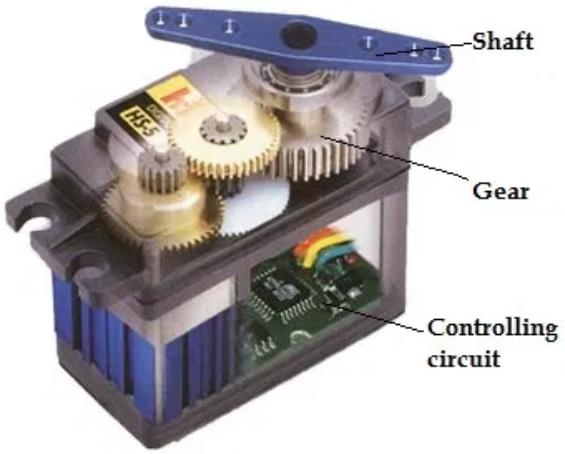


Figure 72: The construction of the servo. Source: [23].

Function	Description
attach(PWM pin number)	This function links the servo with chosen PWM pin.
detach()	This function removes the connection between servo and PWM pin.
read()	This function allows to read the angle of deflection of the servo.
write(angle)	This function allows to set the angle of deflection of the servo.

Table 8: The most popular commands to servo control.

In this example, the servo will move from minimum position through middle position to the maximum position and back again. The position will be changed every 1s. First, the servo should be connected to the Arduino board according to Figure 74.

The code for Snap4Arduino is shown in Figure 75 and consists of the following lines:

1. A *pin* variable is created, which will store the pin number where servo is connected. The value 7 is written to *pin* variable.
2. The servo position is changed to 0 (right corner).
3. The servo position is changed to 90 (middle).
4. The servo position is changed to 180 (left corner).
5. The servo position is changed to 90 (middle).

After running the code, servo will change position between minimum and maximum. The analogous code for the Arduino IDE software is shown here:

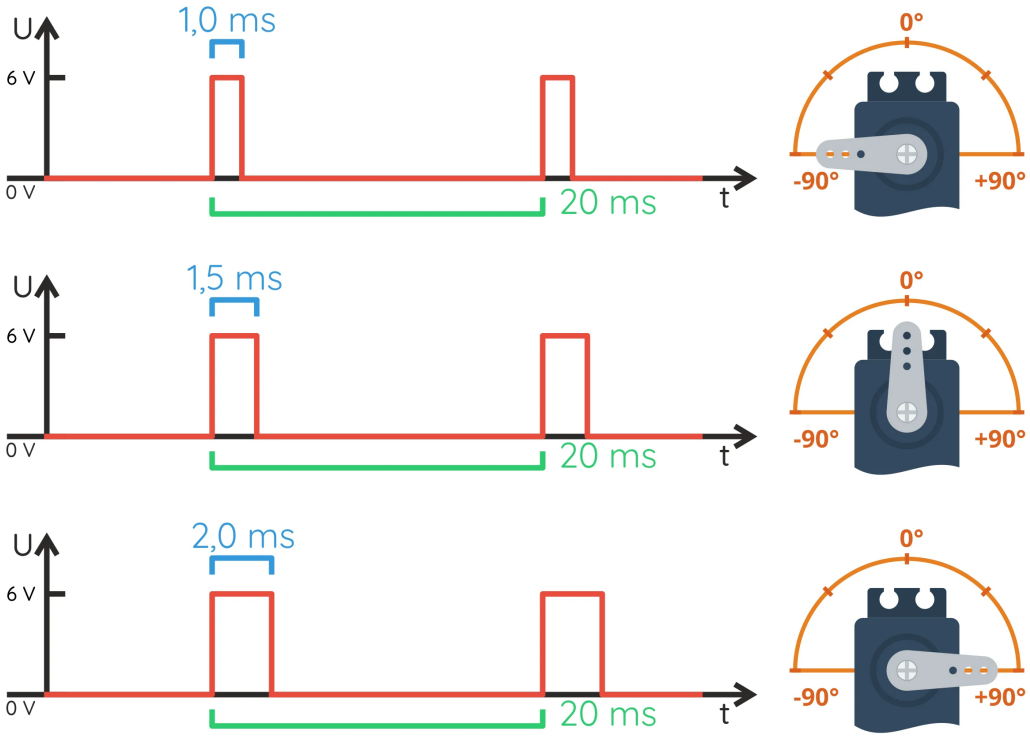


Figure 73: The idea of the servo operation. Source: [24].

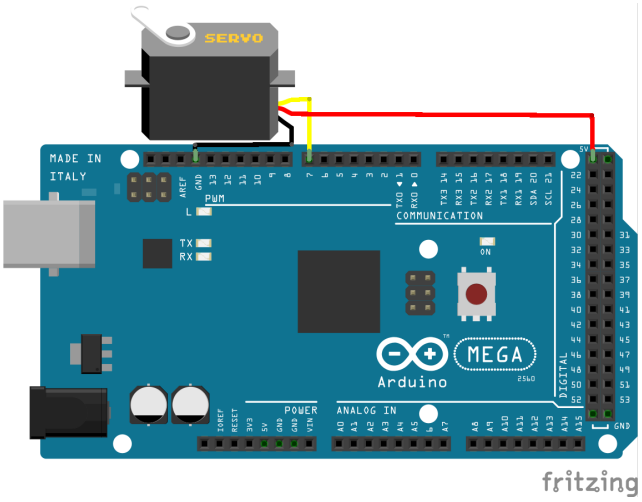


Figure 74: The electronic circuit for example 14.

```
#include <Servo.h>

int pin=7; 1
Servo servo; 2

void setup() {
  servo.attach(pin); 3
}
```

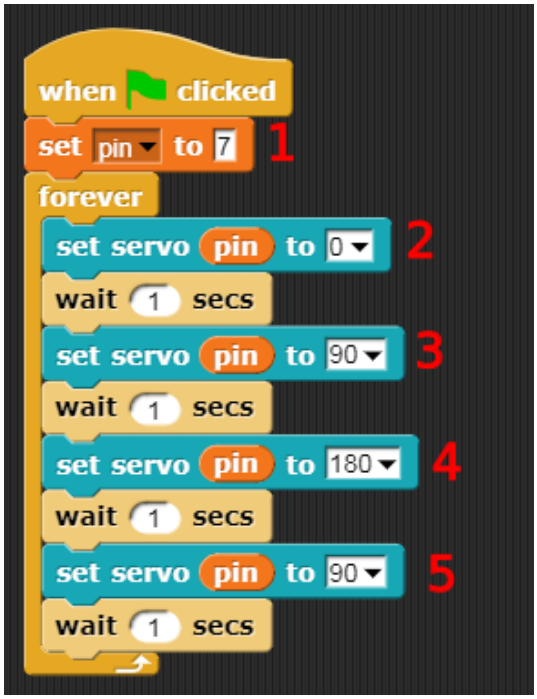


Figure 75: The code for example 14 written in Snap4Arduino.

```
void loop() {
  servo.write(0); 4
  delay(1000);
  servo.write(90); 5
  delay(1000);
  servo.write(180); 6
  delay(1000);
  servo.write(90); 7
  delay(1000);
}
```

The code for Arduino IDE consists of the following lines:

1. A *pin* variable is created, which will store the pin number where servo is connected. The value 7 is written to *pin* variable.
2. The object of *Servo* class is created.
3. The pin is linked with the servo.
4. The servo position is changed to 0 (right corner).
5. The servo position is changed to 90 (middle).
6. The servo position is changed to 180 (left corner).
7. The servo position is changed to 90 (middle).

7.2 Motor

The principle of DC motor was described in chapter 4.2. In this chapter, we will focus on programming part connected with DC motor. The motor should be connected via the controller to the Arduino board as is shown in Figure 76.

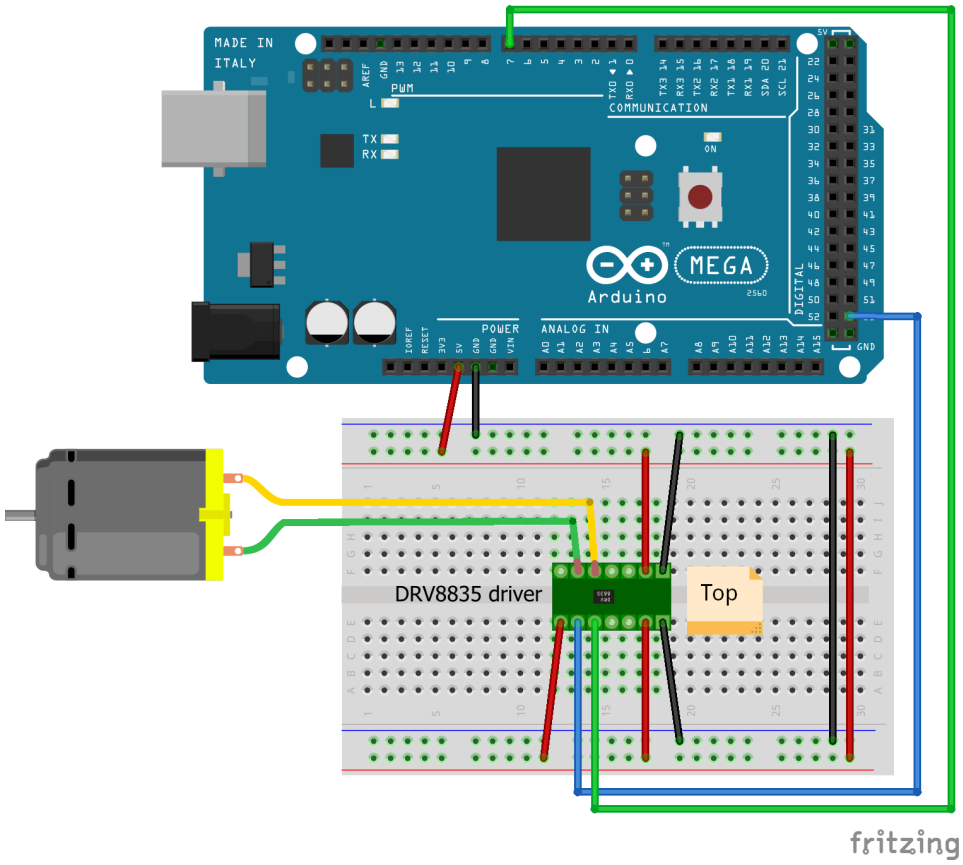


Figure 76: The electronic circuit for example 15.

Example 15

In this example, the motor speed will be increased from minimum to maximum. Then the direction of the motor rotation will be reversed and the motor rotation speed will be repeatedly increased from minimum to maximum.

The code for Snap4Arduino is shown in Figure 77 and consists of the following lines:

1. The *motor* variable is created, which will store the pin number connected to PWM input in motor driver. The value 7 is set to *motor* variable.
2. The *dir* variable is created, which will store the pin number connected to direction output from motor driver. The value 53 is set to *dir* variable.
3. The direction of rotation of the motor is set counter-clockwise.

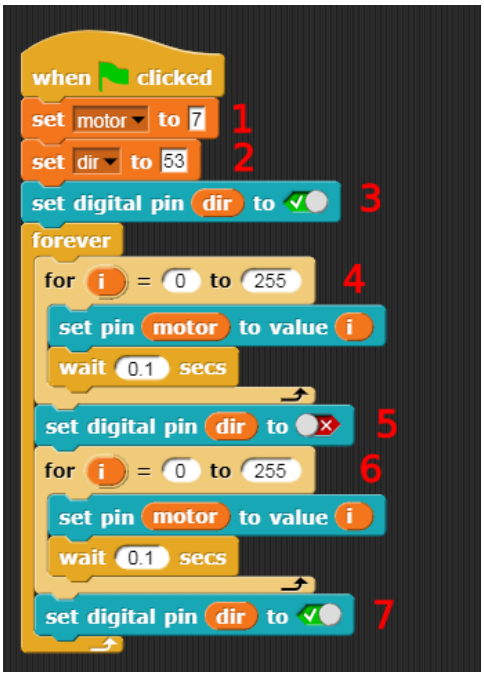


Figure 77: The code for example 15 written in Snap4Arduino.

4. Inside the for loop, the duty cycle of PWM signal is changing from 0 to 255. The delay between changes is 0.1s.

5. The direction of rotation of the motor is set clockwise.

6. Inside the for loop, the duty cycle of PWM signal is changing from 0 to 255. The delay between changes is 0.1s.

7. The direction of rotation of the motor is set counter-clockwise.

The analogous code for the Arduino IDE software is shown in here:

```
int motor=7; 1
int dir=53; 2

void setup() {
  pinMode(motor,OUTPUT); 3
  pinMode(dir,OUTPUT);
}

void loop() {
  digitalWrite(dir,HIGH); 4
  for(unsigned int i=0; i<255;i++) 5
  {
    analogWrite(motor,i);
    delay(100);
  }
  digitalWrite(dir,LOW); 6
```

```
  for(unsigned int i=0; i<255;i++) 7
  {
    analogWrite(motor,i);
    delay(100);
  }
}
```

The code for Arduino IDE consists of the following lines:

1. The *motor* variable is created, which will store the pin number connected to PWM input in motor driver. The value 7 is set to *motor* variable.
2. The *dir* variable is created, which will store the pin number connected to direction output from motor driver. The value 53 is set to *dir* variable.
3. The pins are configured as a outputs.
4. The direction of rotation of the motor is set counter-clockwise.
5. Inside the for loop, the duty cycle of PWM signal is changing from 0 to 255. The delay between changes is 0.1s.
6. The direction of rotation of the motor is set clockwise.
7. Inside the for loop, the duty cycle of PWM signal is changing from 0 to 255. The delay between changes is 0.1s.

8 Interrupts

The interrupts are mechanism, which allow to pause the execution of the code when an external signal occurs. As the external signal can be e.g. falling edge or *LOW* signal from button, sensors etc. The interrupts should be used when executing the part of code is critical when signal occurs e.g. in the alarm system when motion sensor detects the movement, the alarm should be turned on. Sometimes, the code is complex and continuous checking is impossible due to delays in different parts of code. In this situation, the interrupts should be used.

The *attachInterrupt(number of interrupt, function, react to)* defines the interrupt and function, which will run after interrupt occurs. On the Arduino board, only the specific pins allow to generate the interrupt according to the table 9.

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno	2	3				
Mega 2560	2	3	21	20	19	18

Table 9: The pin, which can be used to detect the external interrupts.

The interrupts can be caused by:

- **LOW** - the interrupt will be generated after *LOW* signal occurs on pin.

- **CHANGE** - the interrupt will be generated after changing *HIGH* signal to *LOW* signal or *LOW* signal to *HIGH* signal.
- **RISING** - the interrupt will be generated after changing *LOW* signal to *HIGH* signal.
- **FALLING** - the interrupt will be generated after changing *HIGH* signal to *LOW* signal.

The interrupts can be used only in the Arduino IDE software.

Example 16

In this example, we will build a simple alarm system that will consist of a PIR motion sensor and a buzzer. When motion is detected, the buzzer will be activated for 1s. The electronic circuit should be connected as in Figure 78.

The code in Arduino IDE is shown here:

```
int buzzer=7; 1
bool on=0; 2

void setup() {
  pinMode(buzzer,OUTPUT); 3
  attachInterrupt(0,alarm,RISING); 4
}

void alarm() 5
{
  on=1;
}

void loop() {
  // put your main code here, to run repeatedly:
  if(on) 6
  {
    analogWrite(buzzer,100);
    delay(1000);
    on=0;
  }
  else analogWrite(buzzer,0); 7
}
```

The code for Arduino IDE consists of following lines:

1. The *buzzer* variable is created, which will store the pin number where buzzer is connected to the Arduino board.
2. The *on* variable is created, which will store the status of alarm (0 - no movement, 1 - movement).
3. The pin where buzzer is connected is configured as output.

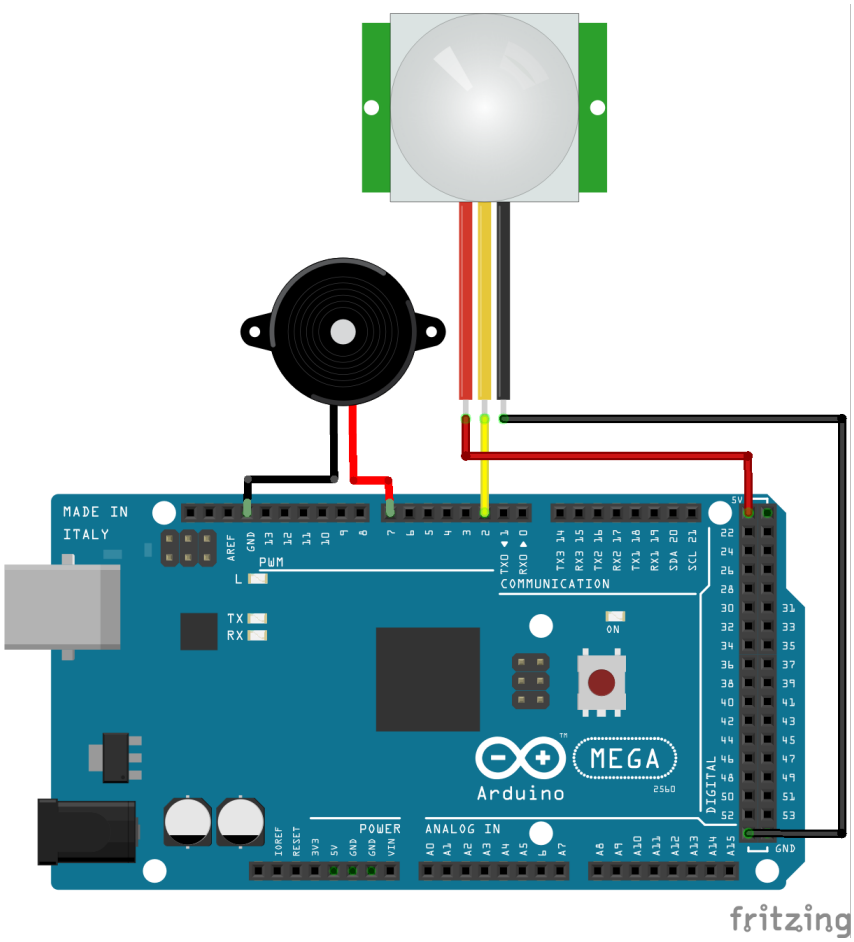


Figure 78: The electronic circuit for example 16.

4. The interrupt is configured. First argument is a number of the interrupt. The PIR motion sensor is connected to pin number 2. According to the table 9, this is interrupt number 0. Second argument is the name of function, which should be executed when interrupt occurs. The last argument is type of signal, which will stimulate the interrupt.

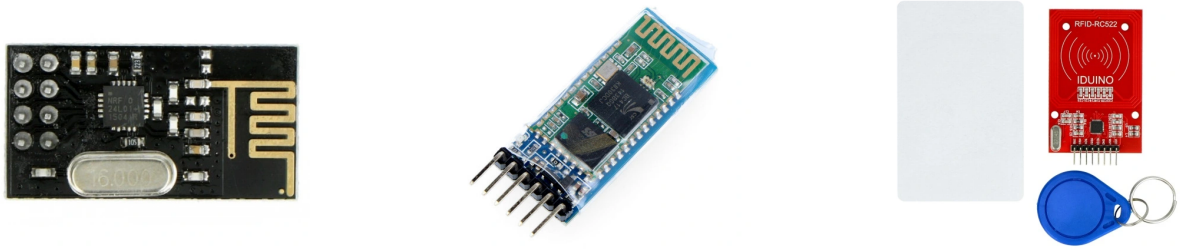
5. The interrupt function is created. In its middle, the value of the variable *on* is set to 1.

6. If motion is detected, the buzzer is activated by setting the PWM signal duty cycle to 100. Then a 1s delay is executed. The last step is to change the value of the variable *on* to 0.

7. If the motion is not detected, the buzzer is turned off.

9 Wireless communication

In some projects, the collected data is passed on, e.g. for the purpose of visualization or analysis. Often it is not possible to physically connect the Arduino board to another device, such as a computer, another Arduino board, Raspberry PI, etc. In such cases, you can use wireless communication and send data using, for example, radio modules, Bluetooth modules, RFID modules or network modules. In this chapter, we cover the first three types of modules. Sample modules are shown in the Figure 79.



(a) Radio module. (b) Bluetooth module (c) RFID module

Figure 79: The example wireless communication modules. Source: [15].

9.1 Radio communication

Radio communication takes place with the use of antennas that emit and receive electromagnetic waves adequately amplified. Depending on this amplification, such transmission may have a different range (from a few cm in the case of proximity cards to hundreds of thousands of kilometers in the case of space probes).

Radio communication modules are cheap and easy to use. However, you need to remember that most of them require a 3.3V power supply.

Example 17

In this example we will be using two Arduino boards. A radio module (nRF24L01) will be connected to both boards. The LM35 temperature sensor, discussed in chapter 5.3, will be additionally connected to the first Arduino board. The first Arduino board will be used to read the ambient temperature and send information about the read temperature via radio. So such a board will work as a transmitter. The second Arduino board will work as a receiver, i.e. it will receive the temperature value and display it on the serial monitor. The circuit should be connected according to Figure 80.

First, we will analyze the code in the Arduino IDE for the transmitter, which looks like this:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
```

```
#include <printf.h>

RF24 radio(48,53); 1
const byte rxAddress[6]="00001"; 2

int LM35=A0;
int value=0;
float voltage;
float temp=0.0;

void setup()
{
  pinMode(LM35,INPUT);

  Serial.begin(9600);
  printf_begin();

  if(!radio.begin()) Serial.println("Initialization failed!"); 3
  radio.setPALevel(RF24_PA_MAX); 4
  radio.setRetries(15,15); 5
  radio.openWritingPipe(rxAddress); 6
  radio.stopListening(); 7
}

void loop()
{
  value=analogRead(LM35);
  voltage=(5.0/1024.0)*value;
  temp=voltage*100.0;
  Serial.println(temp);

  radio.write(&temp, sizeof(float)); 8
  delay(1000);
}
```

The code for transmitter consists of following lines:

1. The creation of object of RF24 class. The arguments are: pin connected to CE and pin connected to CSN.

2. The creation of address.

3. Initialization of radio module.

4. Setting the output power of the transmitter. The option $RF24\_PA\_MAX$ means 0 dBm. Other options available: $RF24\_PA\_MIN$= -18dBm, $RF24\_PA\_LOW$= -12dBm and $RF24\_PA\_HIGH$ =-6dBm.

5. The *setRetries* function sets how many times a module should attempt to establish communication before returning an error. The first argument is the time between repetitions. The second argument is the number of repetitions.

6. Create a link (pipe) to send data.

7. End listening. Switching to transmission mode.

8. Sending the temperature value.

In the second step, we will analyze the receiver code:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <printf.h>

RF24 radio(48, 53);    1
const byte rxAddr[6] = "00001"; 2

void setup()
{
  while (!Serial);
  Serial.begin(115200);

  printf_begin();
  radio.begin(); 3
  radio.setPALevel(RF24_PA_MIN); 4
  radio.openReadingPipe(1, rxAddr); 5
  radio.startListening(); 6
}

void loop()
{
  if (radio.available()) 7
  {
    float temp;
    radio.read(&temp, sizeof(float));
    Serial.println(temp);
  }
}
```

The code for receiver consists of following lines:

1. The creation of object of RF24 class. The arguments are: pin connected to CE and pin connected to CSN.

2. The creation of address.

3. Initialization of radio module.

4. Setting the output power of the transmitter. The option $RF24\_PA\_MIN$ means -18 dBm.

5. Opening a link for reading pipe. The first argument is the link number (there can be a maximum of 6). The second argument is the address.

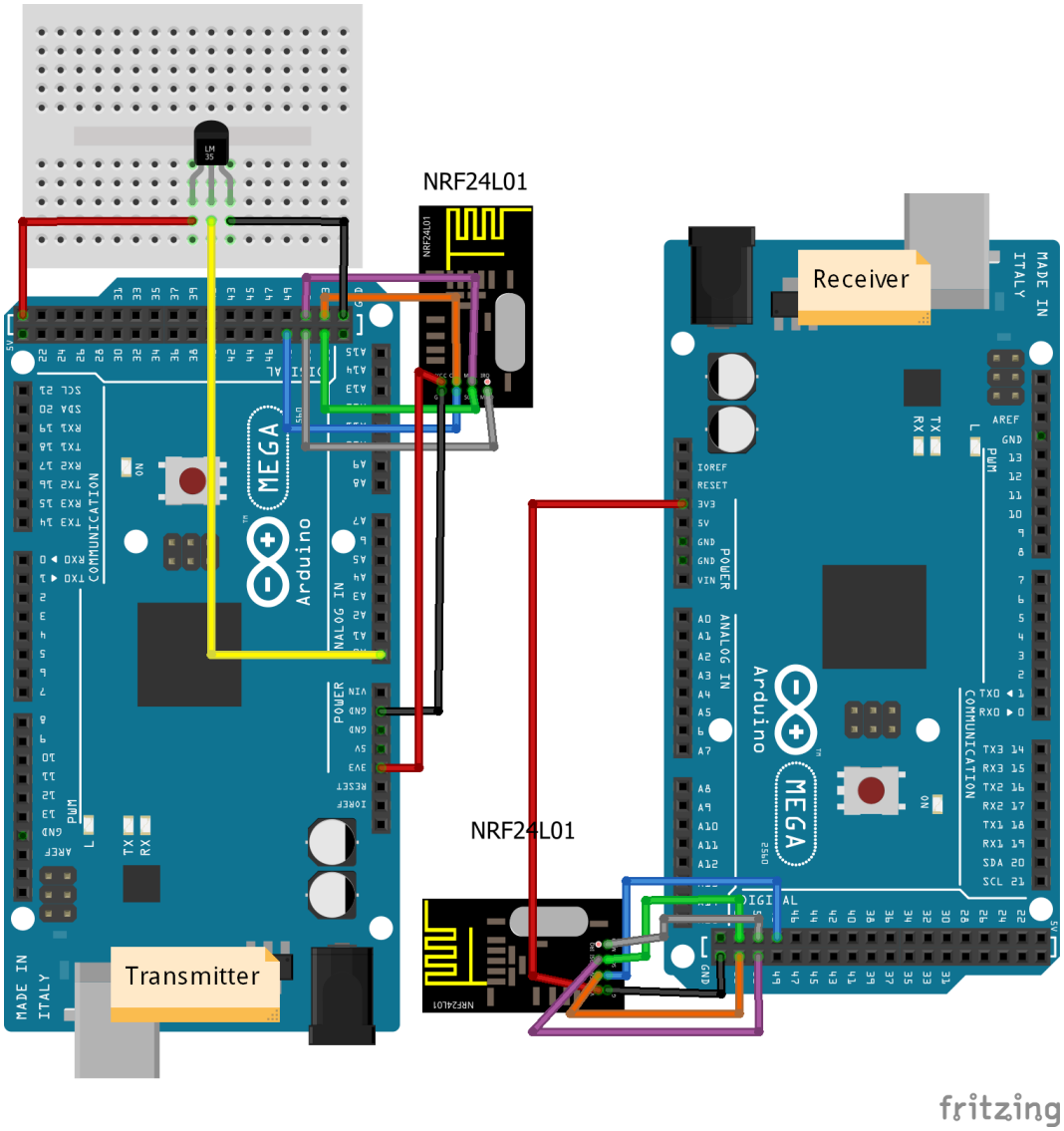


Figure 80: The electronic circuit for example 17.

6. Start listening.
7. If the data is ready for reading, the temperature is displayed on the serial monitor.

9.2 Bluetooth

The Bluetooth modules are cheap and easy to use. The most popular one is HC-05 or HC-06. You can use many libraries to program the Bluetooth module. The very interesting and user-friendly library is *RemoteXY*, which can be installed by using *Manage libraries* inside the Arduino IDE software.

Example 18

In this example, we will create a program that will control the position of the servo from the Android smartphone level. Connectivity between the Arduino board and the smartphone will be provided by the Bluetooth HC-05 module. We will use the *RemoteXY* library, which you should first install in the Arduino IDE using *Manage libraries*. In addition, a smartphone app, also called *RemoteXY*, should be installed. You will find it in the Google Play Store. When you install the necessary elements, we will be able to move on to the more interesting part, e.g. creating an application.

First, connect the circuit according to Figure 81.

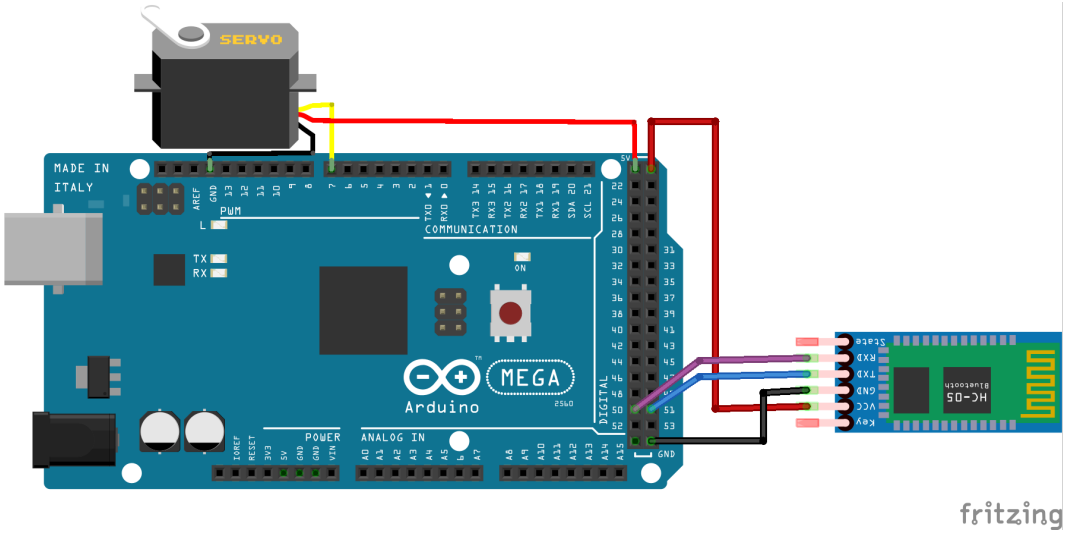


Figure 81: The electronic circuit for example 18.

Second, open up https://remotexy.com/en/editor/. Here you can create a user interface (GUI). In this case, let's use a slider that will store the selected servo position by the user. On the right side there is the *View* section where you can change the variable name assigned to the slider as well as its appearance. A sample GUI is shown in the Figure 82.

When the GUI looks satisfactory, click on the *Get source code* button. Then copy all visible code to the Arduino IDE.

In the next step, we will add a code that allows you to control the servo in the same way as in example 14.

Sample code looks like this:

```
////////////////////////////////////////////////
```

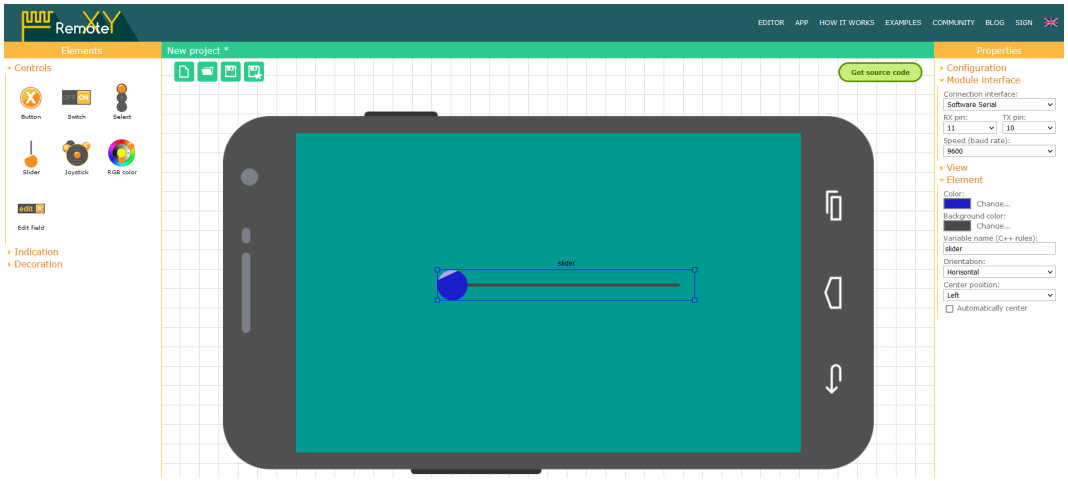


Figure 82: The example GUI.

```
//         RemoteXY include library          //
/////////////////////////////////////////////

// RemoteXY select connection mode and include library
#define REMOTEXY_MODE__SOFTSERIAL
#include <SoftwareSerial.h>
#include <RemoteXY.h>

// RemoteXY connection settings
#define REMOTEXY_SERIAL_RX 51 1
#define REMOTEXY_SERIAL_TX 50 2
#define REMOTEXY_SERIAL_SPEED 9600


// RemoteXY configurate
#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =
  { 255,1,0,0,0,11,0,11,13,0,
  4,128,28,27,51,6,6,26 };

// this structure defines all the variables and events of your control interface
struct {

    // input variables
  int8_t slider; // =0..100 slider position

    // other variable
  uint8_t connect_flag;  // =1 if wire connected, else =0

} RemoteXY;
#pragma pack(pop)

/////////////////////////////////////////////
//           END RemoteXY include          //
/////////////////////////////////////////////
```

```
#include <Servo.h> 3
int pin=7;
double angle;
Servo servo;

void setup()
{
  RemoteXY_Init (); 4
  servo.attach(pin); 5
}

void loop()
{
  RemoteXY_Handler (); 6
  angle=map(RemoteXY.slider ,0 ,100 ,0 ,180); 7
  servo.write(angle); 8
}
```

The code consists of following lines:

1. Definition which pin will work as RX. Remember that software serial can be created only on specific pins. Check, which pins can be used in https://www.arduino.cc.
2. Definition which pin will work as TX.
3. Adding of library and creating the necessary variables to control servo. This code is the same as in example 14.
4. Initialization of RemoteXY.
5. Connecting chosen pin number with servo.
6. This line allows to receive changes on GUI interface like changing the slider position.
7. The slider returns values between 0 to 100. The angle is between 0 to 180. Therefore, the read value from slider should be recalculated to new range (0-180). The map function allows to do it automatically. First you need to put the read value from slider, then the old range of variable and next is the new range of variable.
8. The chosen position is set on servo.

9.3 RFID

RFID (Radio-frequency identification) is a technique that uses radio waves to transmit data over short distances. Each device has a unique RFID key consisting of 14 numbers. RFID modules have found widespread use wherever quick identification is necessary, e.g. when opening the door, proximity cards. There are several libraries to support RFID modules. One of the most popular ones is: *MFRC522*, which needs to be installed from the Arduino IDE using *Manage libraries*.

Example 19

In this example, we will connect two LEDs (red and green) and an RFID module to the Arduino board. If the user brings the RFID pendant close to the module and the key is recognized, the green LED will light up. Otherwise the red LED will light. The electronic circuit should be connected according to Figure 83.

When you install *MFRC522* library, you can analyse the example called *readNUID*. Based on example, the code should look like:

```
#include <SPI.h>
#include <MFRC522.h>

int RED_LED=7; 1
int GREEN_LED=6;

#define RST_PIN 22 2
#define SS_PIN 53

MFRC522 mfrc522(SS_PIN,RST_PIN); 3
byte nuidPICC[4]; 4
int key[4]={23,189,10,179}; 5
bool same=true; 6

void setup()
{
  pinMode(RED_LED,OUTPUT); 7
  pinMode(GREEN_LED,OUTPUT);

  Serial.begin(9600);
  SPI.begin(); 8
  mfrc522.PCD_Init();
  Serial.println("Reading the RFID code");
}

void loop() {
  // put your main code here, to run repeatedly:
  if(mfrc522.PICC_IsNewCardPresent()) 9
  {
      if(mfrc522.PICC_ReadCardSerial()) 10
      {
          Serial.print("RFID card ID: ");
          for (byte i = 0; i < mfrc522.uid.size; i++)
          {
            nuidPICC[i] = mfrc522.uid.uidByte[i];
            Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
            Serial.print(mfrc522.uid.uidByte[i], DEC);
          }
      Serial.println();
      mfrc522.PICC_HaltA();
    }
  }
```

```
  //Key checking
  same=true;
  for(byte i = 0; i < 4; i++) 11
  {
    if(key[i]!=nuidPICC[i])same=false;
  }

  //Cleaning table
  for(byte i = 0; i < 4; i++) nuidPICC[i]=0; 12

  if(same) 13
  {
    Serial.println("Key recognized");
    digitalWrite(GREEN_LED,HIGH);
    digitalWrite(RED_LED,LOW);
  }
  else
  {
    digitalWrite(GREEN_LED,LOW);
    digitalWrite(RED_LED,HIGH);
  }
  delay(500);
}
```

The code consists of following lines:

1. The variables are created, which will store the pin numbers where red and green LED diodes are connected.
2. The variables are created, which will store the pin numbers where reset and chip select are connected.
3. The object of MFRC522 is created.
4. The *nuidPICC* table is created, which will store the read key from pendant.
5. The *key* table is created, which stores the proper key.
6. The *same* variable is created, which will inform if the read key is the same as proper key.
7. The pins, where LED diodes are connected, are defined as output.
8. The SPI bus and MFRC522 module are initialized.
9. This line checks if new pendant is close to RFID module.
10. This if structure reads the key of pendant and save to *nuidPICC* table.
11. The read key is checked with the proper one.
12. The *nuidPICC* table is cleaning.
13. If the key of pendant is proper, the green LED is turned on. Otherwise, the red LED is turned on.

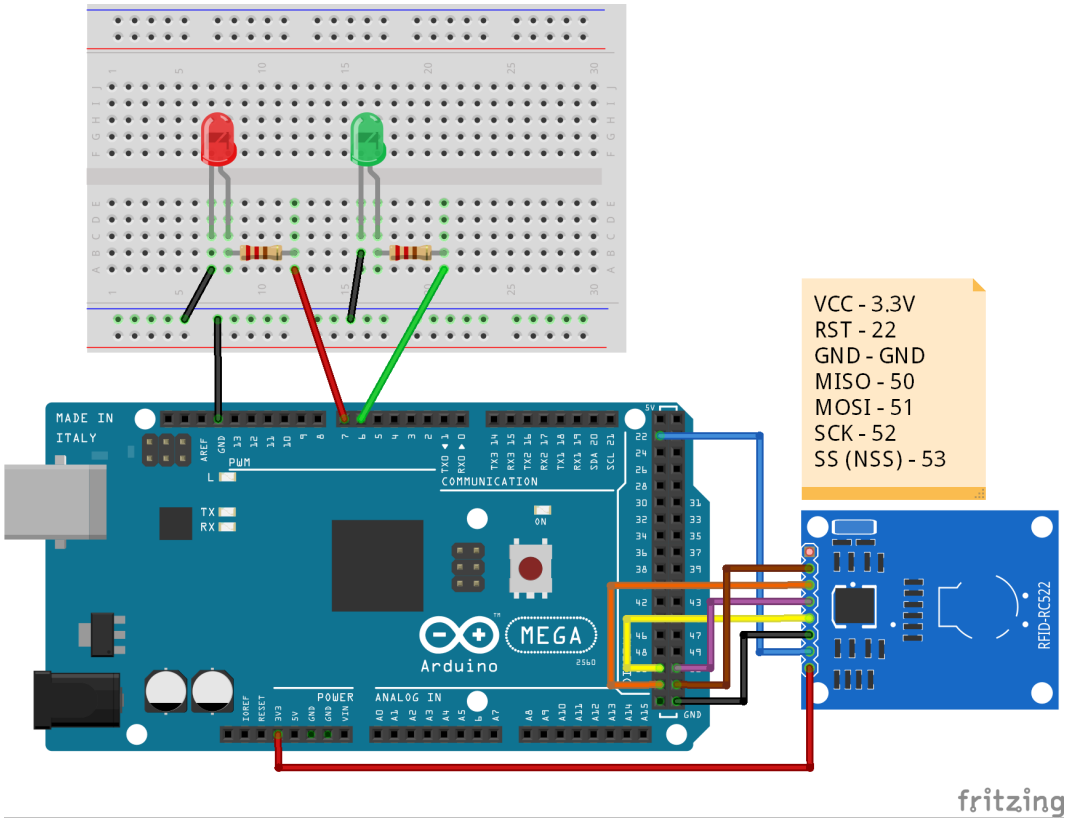


Figure 83: The electronic circuit for example 19.

10 Designing of the robot

10.1 Definition of a robot

Before we start our adventure with robotics, let's consider what a robot is. There are many definitions of a robot that better or less fit modern robots. The best matching definition can be found in book *The Robotics Primer*[25], which defines a robot as:

"A robot is an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals." [25]

What does the definition of a robot mean? A robot will be any device that will receive information from the environment using e.g. sensors and will make decisions based on them. The robot does not need to move, it can be static. It is important that it should be able to make decisions based on information from sensors. Let's take a look at the six proposed robotic projects to be implemented in the classrooms:

1. Lighthouse project - the project aims to create an intelligent lighthouse that will turn on the light only when it is dark outside and change the blinking frequency depending on whether the ship is near or far from the shore. Can the lighthouse be a robot? At first, most will say no. However, there is no such basis for not being a robot. It meets all three assumptions: it exists in the real world (it is not a simulation), receives information from the environment (light intensity using a photoresistor; the distance of the ship from the shore using an ultrasonic sensor) and makes decisions based on the input information (whether to turn on the light or not; how often to blink).

2. Sunflower project - The project aims to create a sunflower that moves towards the light. Here too, all three assumptions are met. Sunflower is a robot that receives information about the intensity of light based on photoresistors and depending on the value read, it will decide which way to move.

3. Smart light project - the aim of this project is to create intelligent lighting that will light up when motion is detected. Otherwise the lighting will be off. Here, too, we have the basis for calling a smart light robot. Based on the PIR motion sensor, a decision is made to turn on orturn off the lighting.

4. Theremin project - the goal of this project is to create Theremin that will be controlled with two hands. Depending on the distance of the hand from the ultrasonic distance sensor and the photoresistor, different pitch and beat will be produced. In this case, Theremin is also a robot that decides what pitch and beat to produce.

5. DIY automobile project - this project consists in creating a driving robot that will have a number of possibilities from detecting and avoiding obstacles to completing a route according to a complex algorithm (shape). This design is the most obvious example of a robot.

6. Weather station - this project is another extraordinary example of a robot. Its purpose is to collect information about the weather (temperature, humidity and pressure), but also to inform the user by changing the color of the LCD screen whether it is cold or warm. This part with color change is the key factor, therefore this weather station is a robot. Here we have a decision element in the form of information whether the temperature is appropriate or too low/high.

The described interdisciplinary robotic projects show how broad the concept of a robot is. The shape and purpose of the robot's work is limited only to our imagination. Let's just remember that each robot must somehow obtain information from the environment and make decisions and act on their basis.

Detailed descriptions of projects along with an exemplary implementation are available in the educational materials.

10.2 How does the robot work?

Before we start building a robot, a few things should be planned. The first is the way the robot works. There are three main robotic working architectures. The first of them is *reactive control*, which most of you will perform intuitively during the first attempts to create a robot. This architecture is based on a simple mechanism. Depending on the information received from the environment, a decision is made. An example is the smart light project described briefly in the previous section. When motion is detected, the light is turned on. So based on information (move or not), the decision is made (turn on or off the lights).

The second option is *deliberative control*, in this case, after reading information from the environment, the next actions are planned and then the action is performed. An example would be a robot playing checkers. After the opponent's move, the robot analyzes possible scenarios and performs the move according to the optimal scenario.

The last option is *hybrid control*, which combines the advantages of both methods. Those activities that require quick reactions are carried out according to *reactive control*. On the other hand, strategic decisions are made according to the *deliberative control* architecture.

10.3 How to start building a robot?

The construction of the robot can be divided into the following stages:

1. **Defining the purpose and requirements:**
   At the beginning, it is necessary to consider what the purpose of the robot is. Then, knowing the goal, you need to define a list of requirements, e.g. speed of movement of the robot, load capacity, frequency of activities, etc.

2. **Way of getting around:**
   If the robot should move, you should consider on what surfaces the robot will move.

If these are flat surfaces, you can consider the use of wheels. Often the wheels are mounted on a differential gear, i.e. two wheels connected to two motors on one axis of rotation are used. For the stability of the robot, an additional auxiliary wheel (so-called caster wheel) can be added. This solution was used in the DIY-automobile project. I encourage you to read the materials for this project, which shows how to mount the wheels on the differential gear.

If the robot is to move outside then the ground unevenness must be taken into account. If the terrain is relatively flat, it is enough to choose the right wheels. However, there may be times when the ground is not compacted enough and the wheels get stuck in the ground. Then the use of caterpillars will be a better solution.

An interesting idea is to create a robot that moves analogously to animals, e.g. spider, dog or fish. The construction of such a robot is a challenge, but it encourages the creators to get acquainted with many fields of science, e.g. biology, physics etc.

3. **Determining the method of collecting information from the environment:**

   In the next step, it is necessary to define what information from the environment will be read by the robot. Based on this, sensors should be selected.

4. **Defining the actuators:**

   Next, you should define what actuators will be needed. Depending on the purpose, these can be: motors, servos, etc. Sometimes our robot does not need to have actuators. Examples include the following projects: lighthouse, smart light, theremin and weather station. In these projects, no movement is made, but decisions are made.

5. **Defining the robot's work architecture and the robot's "heart":**

   In the previous section, the three main robotic architectures were discussed: reactive control, deliberative control and hybrid control. If the robot is to perform a simple goal based on current or average sensor readings, the most convenient architecture will be reactive control, which is used in all proposed robotic projects. A simple microcontroller, e.g. in an Arduino board, is enough to implement this architecture. If you choose architectures: deliberative control or hybrid control, it may be better to use a stronger computationally platform (other than Arduino), e.g. Raspberry Pi.

6. **Defining the method of communication with the robot (optional step):**

   In some cases, it is necessary to communicate with the robot using e.g. Bluetooth modules, radio modules or RFID modules. In such cases, it is necessary to specify how the modules are connected and what information is to be transferred. An example project showing remote communication with a robot is DIY-automobile (level 4).

7. **Identification of other necessary electronic components**:

   Most projects require the use of additional electronic components such as LEDs, resistors, capacitors, transistors, displays etc.

8. **Identification of mechanical elements:**

   Once you have a list of all electronic components that should be connected to the "heart" of the robot, e.g. Arduino, consider how to attach them. You can use everyday materials such as cardboard boxes, cardboard, plastic or metal packaging. You can use screws, hot glue or even tape to attach the elements. In simple projects, this approach is enough. In more advanced projects, you can use a 3D printer and make a base design with holders for sensors or actuators.

9. **Building a robot:**

   Combine all the elements according to the plan prepared in the previous point.

10. **Preparation of the robot's work algorithm:**

    Before you start writing the code, think carefully about what the robot's algorithm should look like. It is a good practice to create a code block diagram, which will show all the steps of the algorithm and the transitions between each stage. A sample algorithm in the form of a block diagram is shown in the Figure 84.

11. **Preparing the code and uploading to the selected platform.**

12. **Testing and improving the robot:**

    If the robot is ready to work, its operation should be tested at the beginning. When the robot's work does not meet our expectations, return to step 10 and work on improving the robot's work algorithm.

10.4 The example of robot construction

In order to illustrate all the steps necessary to construct a robot, we will use the sunflower project. We will start by following the steps described in the previous chapter step by step:

1. **Defining the purpose and requirements:**

   The robot will simulate the operation of a sunflower. So its goal will be to go towards the light.

2. **Way of getting around:**

   The robot does not need to move. It is enough for it to be able to rotate, so there is no need to attach wheels, tracks, etc.

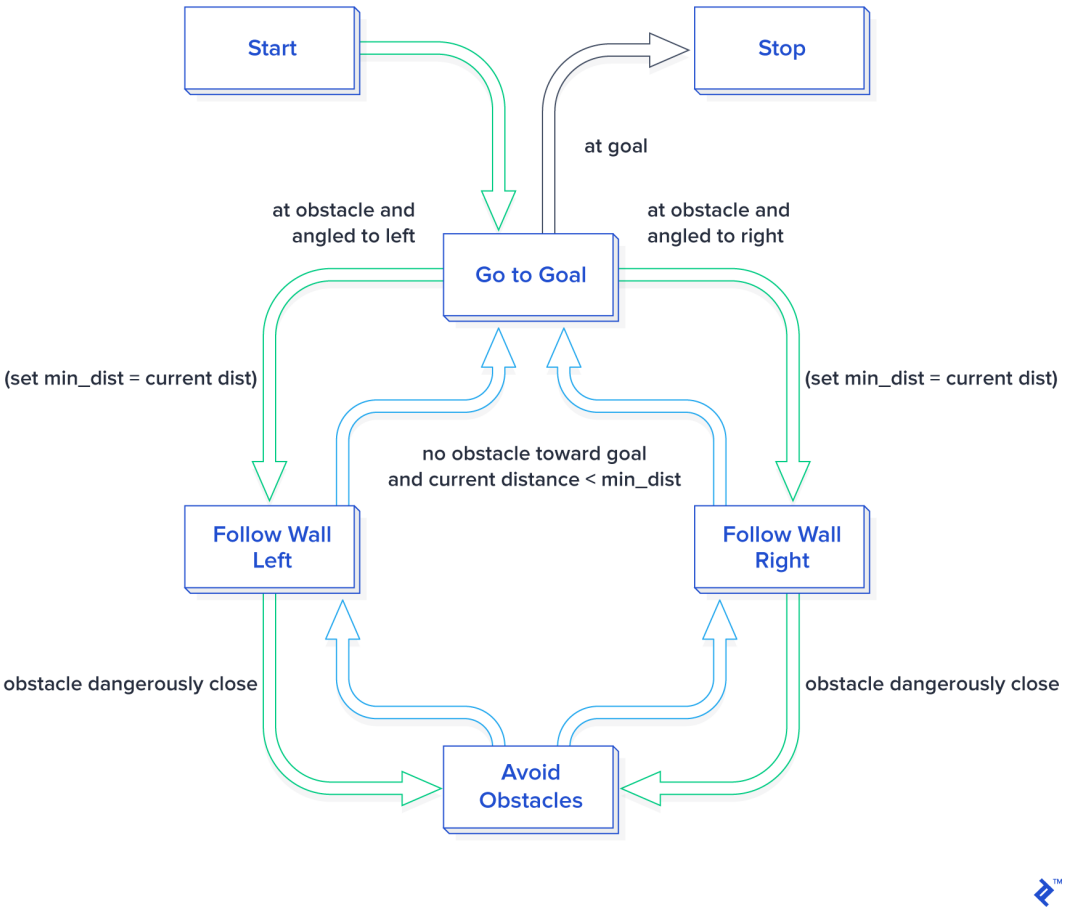


Figure 84: The example of code block diagram. Source: [26].

3. **Determining the method of collecting information from the environment:**

   The robot needs to read the intensity of the light. You can use modules measuring the light intensity or simply photoresistors. In this case, we choose the easier variant, i.e. photoresistors. To obtain information in which direction the light intensity is higher, we need at least two photoresistors.

4. **Defining the actuators:**

   The robot should rotate therefore the actuator should be used. In this case the optimal solution will be a servo. The robot will be tilted by an appropriate angle depending on the measurements from the photoresistor.

5. **Defining the robot's work architecture and the robot's "heart":**

   In this case, we have a simple situation. Depending on the measurements from the photoresistors, the robot should turn left or right. We have no way of predicting the direction here. Therefore, the simplest architecture, i.e. reactive control, is enough. The robot's "heart" can be Arduino board.

6. **Defining the method of communication with the robot (optional step):**

   Not needed.

7. **Identification of other necessary electronic components**:

   Photoresistors must be connected via a 10 $k\Omega$ resistors. Connections will be made on a breadboard, so male-male wires will also be useful.

8. **Identification of mechanical elements:**

   The robot should be similar to the real sunflower as possible. Hence, the base, stem and petals of the flower will be indispensable. The base will be made of a cup. The stem will be made of a piece of wood wrapped in green cloth. The flower petals will be made of cardboard and covered with a colored fabric. Photoresistors will be attached to the outermost petals of the flower. Then the wires from the photoresistors will be attached to the stem and hidden under the material. The wires will go to the base where the breadboard will be hidden. The wires will be connected to the resistors. The servo will also be hidden in the base and the stem will be attached to it. Arduino will also be placed in the base.

Figure 85: The exemplary construction of the robot. Source: "Ideas for crafting" - materials for sunflower project.

In summary, the necessary materials are: a cup, a piece of wood, cardboard and material in green and other colors of your choice.

9. **Building a robot:**

   An exemplary construction of the robot is shown in the Figure 85. The connection of electronics components is shown in materials for Sunflower project: http://www.roboscientists.eu/wp-content/uploads/2019/09/Sunflower-guidelines.pdf

10. **Preparation of the robot's work algorithm:**

    A example robot's work algorithm is shown in Figure 86.

11. **Preparing the code and uploading to the selected platform.**

12. **Testing and improving the robot:**

    The example of improving the robot is shown in materials for Sunflower project: http://www.roboscientists.eu/wp-content/uploads/2019/09/Sunflower-guidelines.pdf .

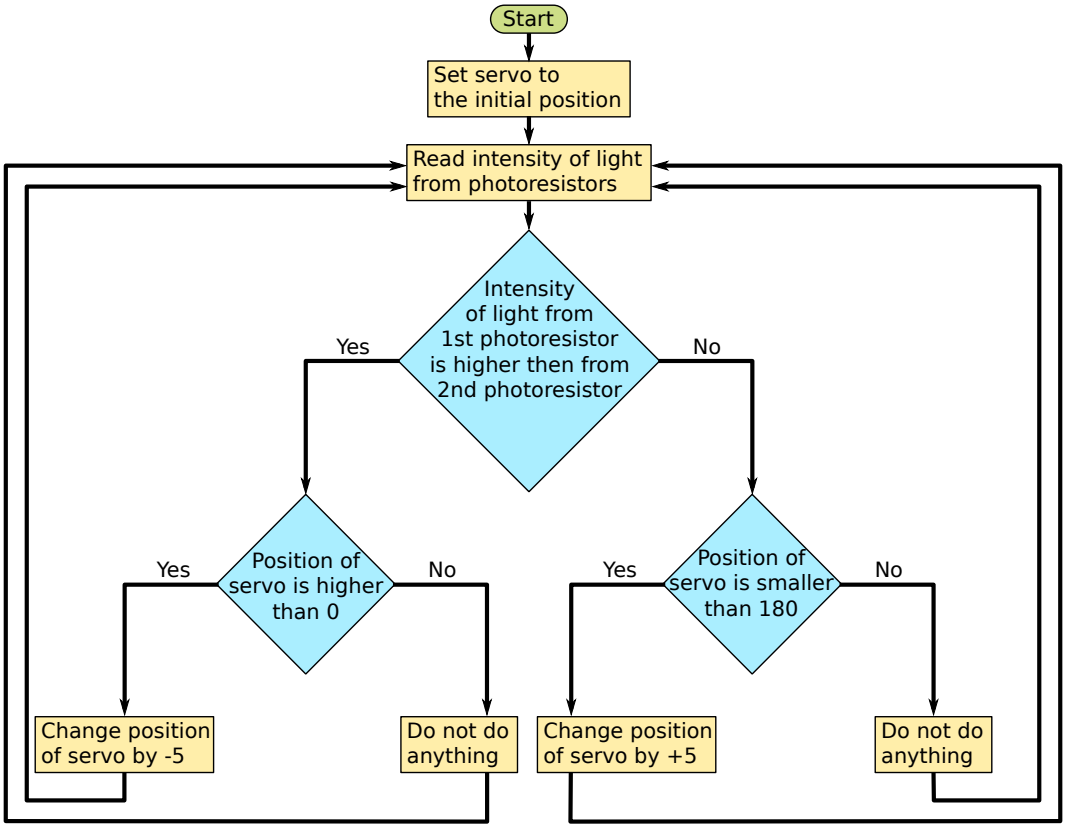


Figure 86: The example of robot's work algorithm.

11 The challenges

The proposed 6 interdisciplinary projects are a good starting material for working with students without robotic experience. They contain both sample solutions in Snap4Arduino and Arduino IDE. Depending on the work intensity of the students, these projects will last for 0.5-1 years. Depending on the level of advancement and commitment of the group, only the basic (first) level or higher levels may be completed. Each level includes new interesting expansions. In this chapter, I will present 10 ideas of robots that can be created with students after completing the 6 proposed robots. Here they are:

1. Carbon monoxide detector with notification to the phone.
2. Aquarium automation.
3. Plant automation. Watering plants depending on soil moisture.
4. ECG analysis. Biomedical sensors connected to e.g. an Arduino lilypad sewn into clothing. Information sent to the phone via Bluetooth.
5. Martian robot based on caterpillars.
6. A robot that moves like a cat.
7. A plane that is flying and remotely controlled.
8. Security system.
9. Robot going to a specific location (extension of the DIY-automobile project with a GPS module). In this project, students will learn what GPS frames look like. What information can we read.
10. Basic home automation. The topic can be developed in various ways. For example, you can adjust the blinds depending on the light intensity and temperature in the room.

Remember that the mentioned list should be a last resort. Better to motivate students to come up with a robot proposal they would like to create. Then their commitment and interest will be greater.

References

[1] The Arduino official website, https://www.arduino.cc .

[2] The Snap4Arduino official website, http://snap4arduino.rocks/ .

[3] The website: https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html .

[4] The website: https://www.pinterest.com/pin/292593307022200536/ .

[5] Czesław Bobrowski, "Fizyka - krótki kurs", Wydawnictwo Naukowo-Techniczne, 2003.

[6] Marian Kozielski, "Fizyka dla szkół średnich tom 2", Wydawnictwo B.Z.Kozielski, 1999.

[7] John Boxall, "Arduino 65 praktycznych projektów", Helion, 2014.

[8] Scott Fitzgerald, Michael Shiloh, "Arduino Projects Book", Arduino AG, 2017.

[9] The website: https://forbot.pl/ .

[10] The website: http://illinformedprojects.blogspot.com/2016/06/sound-controlled-led-matrix.html .

[11] The website: https://www.electrical4u.com/diode-working-principle-and-types-of-diod
.

[12] The website: https://commons.wikimedia.org/wiki/File:Alu-Elko-Polarit%C3%A4t.png .

[13] The website: http://nextews.com/5af00fbf/ .

[14] The website: http://www.serwo.wtx.pl/index.php?dzial=&kat=&nr=32 .

[15] The website: https://botland.com.pl/ .

[16] The website: http://avrkwiat.nstrefa.pl/omnie/index.php?option=com_content&view=article&id=244&Itemid=272 .

[17] The website: https://abc-rc.pl .

[18] The website: http://www.raspberrypirobotics.com .

[19] The website: https://learn.sparkfun.com/ .

[20] The website: https://dcubestore.com/blog/difference-between-i2c-and-spi/
.

[21] The website: https://www.circuitbasics.com/basics-uart-communication/ .

[22] S. Janwadkar, D. Bhavar and M. T. Kolte, "Design and implementation of a GPS based personal tracking system," 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), 2016, pp. 1-5, doi: 10.1109/ICPEICES.2016.7853253.

[23] The website: http://www.electricalidea.com/servo-motor/ .

[24] The website: https://forbot.pl/blog/kurs-elektroniki-ii-czujnik-przeszkod-sterownik .

[25] Maja J. Mataric, "The Robotics Primer", MIT Press, ISBN: 9780262633543.

[26] The website: https://www.toptal.com/robotics/programming-a-robot-an-introductory-tutorial

ROBOSCIENTISTS PROJECT

Motivating secondary school students towards STEM careers through robotic artefact making

Erasmus+ KA2 2018-1PL01-KA201-051129

Author:
Angelika Tefelska (WARSAW UNIVERSITY OF TECHNOLOGY)

Reviewers:
Rene Alimisi[2], Nikleia Eteokleous[3], George Keliris[4], Raphaela Neophytou[3], Chrissa Papasarantou[2], Konstantinos Salpasaranis[2], Costas Sisamos[4]

Partners:
[2]EDUMOTIVA, [3]FREDERICK UNIVERSITY, [4]ENGINO.NET LTD

Declaration
This report has been prepared in the context of the ROBOSCIENTISTS project. Where other published and unpublished source materials have been used, these have been acknowledged.

Funding Disclaimer
This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.