Laboratory focused on basics of Arduino

ROBa - Laboratory number 1

https://github.com/Adam-Fabo/ROB-laboratories

Brno university of technology
Faculty of informatics
Adam Fabo
29.4.2023

Basics of Arduino Laboratory

Welcome to the first laboratory. This laboratory is aimed at students with no previous experience with Arduino, robotics or electronics.

Prerequisites

Prerequisites are basic knowledge of programming in C/C++ (syntax, conditions, loops, functions). Also, knowledge in prototyping of electronics circuits is welcomed.

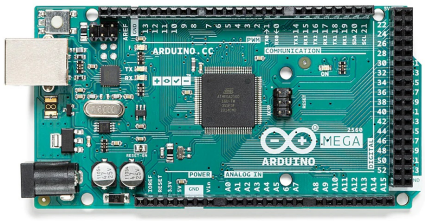
Goals of the laboratory

Goals of this laboratory are set to introduce you with basics of Arduino and prototyping electrical circuits. You will learn:

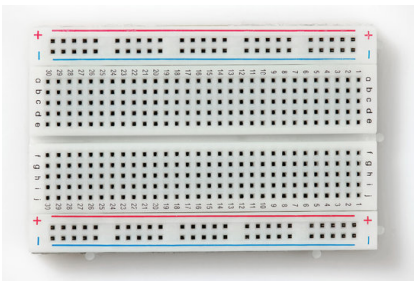
- Work with Arduino and Breadboard
- Creating simple electrical circuits using buttons, LEDs, potentiometers, ...
- Using Arduino's Input/Output capabilities
- Reading/Writing digital and analog signal, using PWM
- Using Arduino's serial communication and debugging your code

Hardware used in this laboratory

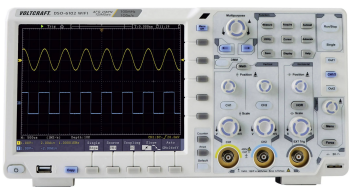
For this laboratory, you will need the following:



(a) Arduino Mega

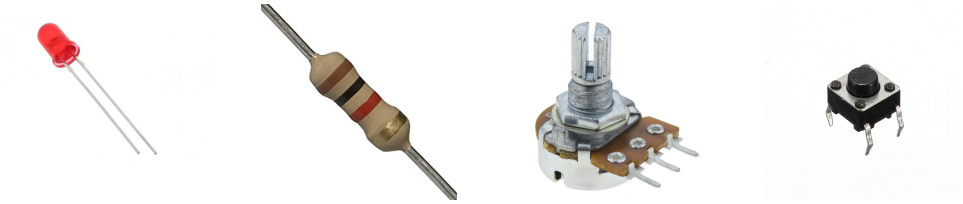


(b) Breadboard



(c) Oscilloscope

Figure 1: Core of the laboratory



(a) LED (b) Resistor (c) Potentiometer (d) Button

Figure 2: Electrical parts

Arduino Mega	-	Contains *ATmega2560* processor. Your code will run on it
Breadboard	-	Used for creating electronics circuits
Oscilloscope	-	Used for displaying electronic signals
LED	-	Light Emitting Diode
Resistor	-	Used for limiting current for electronics parts (like LED)
Potentiometer	-	Variable resistor
Button	-	Used for interrupting circuits

Theory

Arduino overview

Arduino is a set of open source hardware and software projects used for fast prototyping of electronics circuits. It offers lots of **boards** (Arduino Mega, Arduino Uno, Arduino Micro, ...), **shields** (Ethernet shield, Motor shield, ...), **sensors** (Gas sensor, Hall sensor, Motion sensor, ...) and **components** (Universal PCBs, Jumping wires, Connectors, ...) that can be used for fast prototyping and learning basic concepts of electronics and programming.

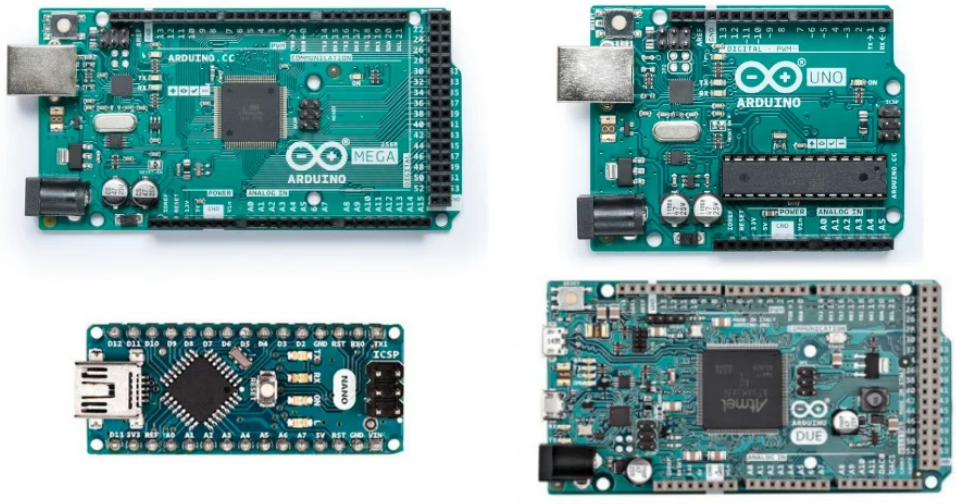


Figure 3: Prewiev of Arduino Boards

Arduino Mega

Arduino board (later on just Arduino) used in this course will be Arduino Mega. It's one of the more powerful boards and offers great amount of **GPIO** - General Purpose Input Output pins.

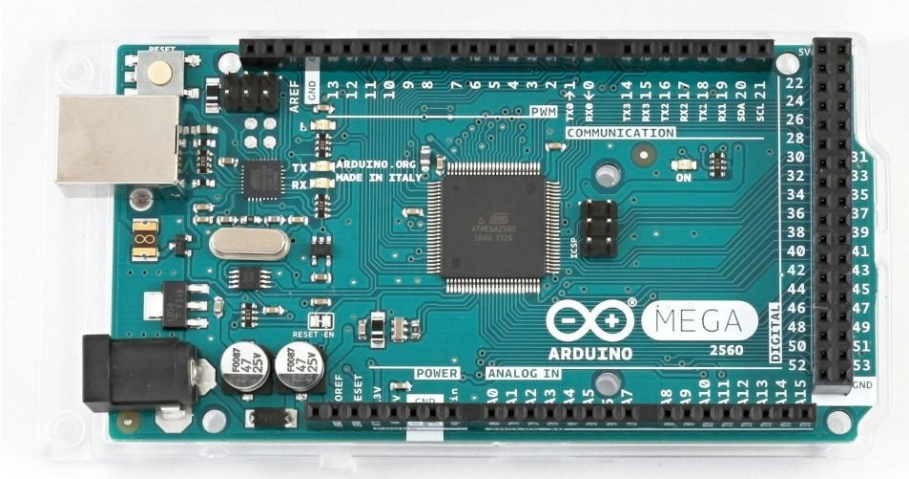


Figure 4: Arduino Mega used in this laboratory

Arduino uses USB type B for communication with PC and power (for powering Arduino can also be used power-jack). It also offers 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4x UART (hardware serial ports).

Microcontroller	ATmega2560
Operating Voltage	5V
Clock Speed	16 MHz
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
LED_BUILTIN	13

Table 1: Some of the specifications of Arduino Mega

Arduino IDE

Arduino also offers software for programming its boards. It can also be programmed by other software, but most convenient way is to use Arduino software. It's called Arduino IDE. There are **2 versions** of it - **Arduino IDE** and **Arduino IDE 2**.

Arduino IDE is an older version of these 2 and offers fully functional IDE. Its disadvantage is its older design. The difference between Arduino IDEs and other editors is that Arduino IDE offers easy way to compile and upload code to the Arduino boards. **Verify/Compile** can be done by button with **Green tick** in left top corner, and **Upload** can be done by button with **Green arrow** next to Verify button. Arduino IDE also offers **Serial monitor** and Serial plotter. Both of them are useful for debugging/looking at result from a code that runs on Arduino.

Figure 5: Preview of Arduino IDE

Arduino IDE 2 is newer and still being developed. It offers modern looking IDE, better Serial plotter etc. Its drawback is that it still has bugs in places, and its functionality is not fully polished.

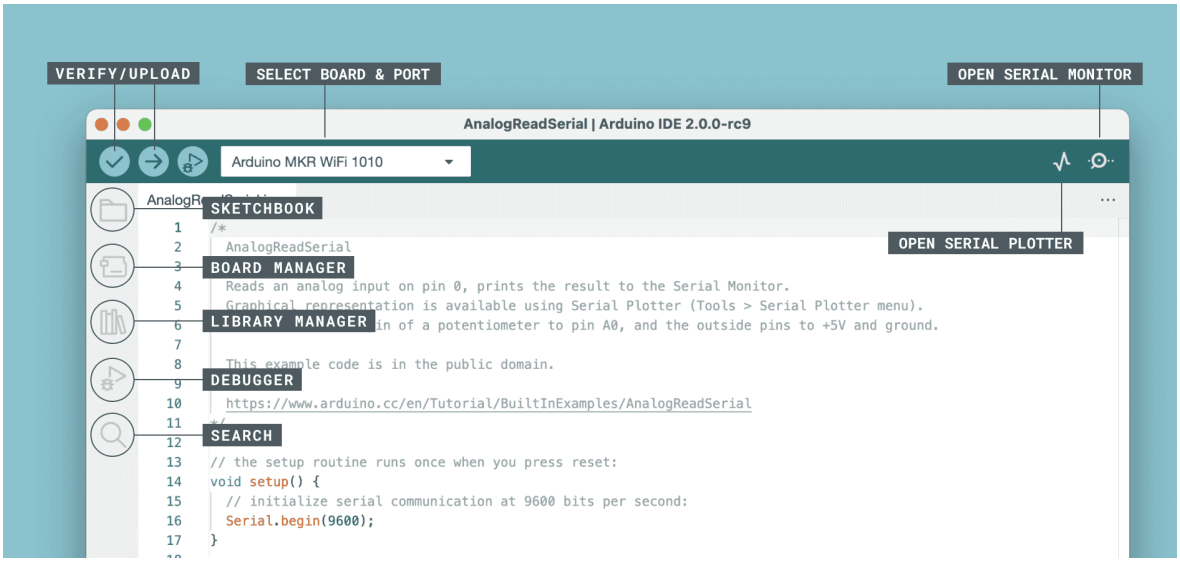


Figure 6: Preview of Arduino IDE 2

Both of the IDEs contain **settings for uploading** your code into the Arduino board. You need to make sure that the correct **Board** and **Port** is set before uploading. Communication port depends on which USB port you chose and Board is which Arduino board you are using. In this laboratory, it is **Arduino Mega**.

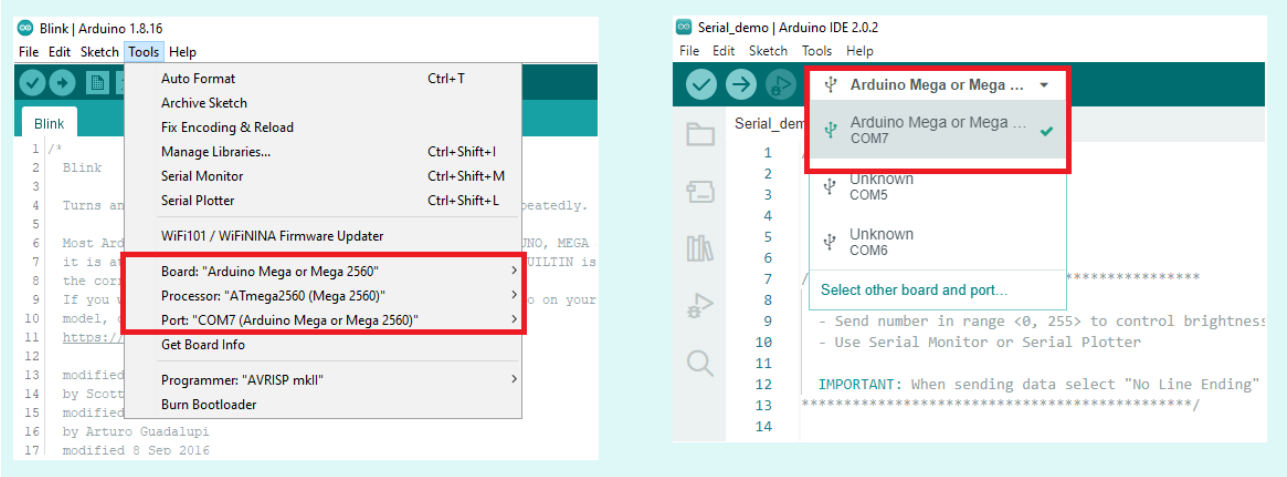


Figure 7: Settings for **Port** and **Board** in both IDEs

Breadboard

Breadboard is a construction base used for solderless prototyping of electronics circuits. Even tough on the market is huge amount of variants, the working principle of each of them is the same.

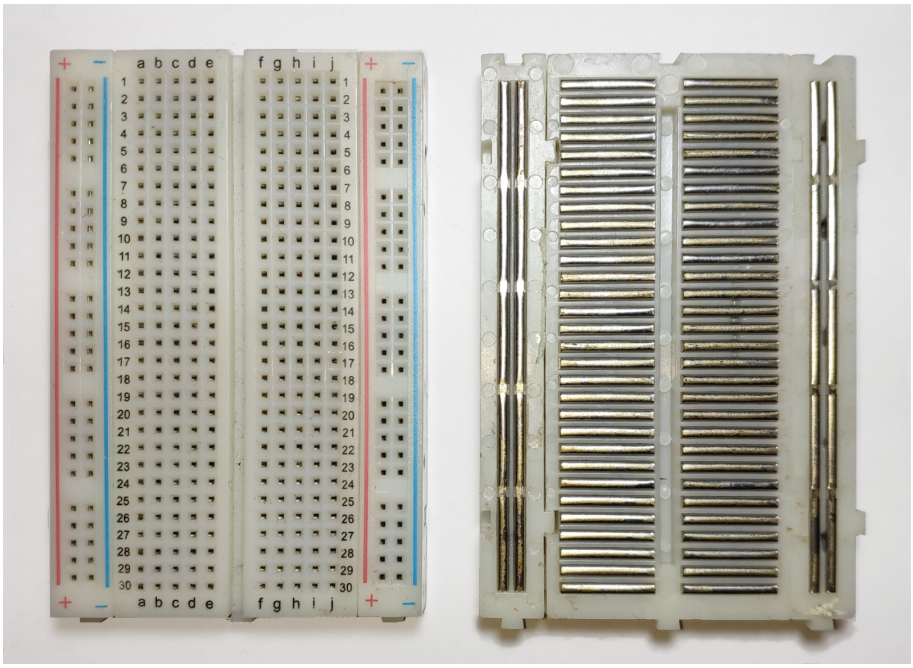


Figure 8: Breadboard

Breadboard consists of 2 types of rails - **horizontal** and **vertical**.

Vertical rails, usually at the sides (bigger boards can have them also in the middle), are used for *supply voltage* and *ground*. They are connected all the way through the breadboard.

Horizontal rails are used for making the circuit itself. This kind of lines are also connected but not trough center divider.

Connections are demonstrated at Figure 8, where can be sees disassembled breadboard from the backside.

Resistor

Resistor is a basic passive electronic part. Its goal is usually to **limit current** flow through the circuit or to **accumulate voltage drop**. It has unit of *Ohm* [Ω].

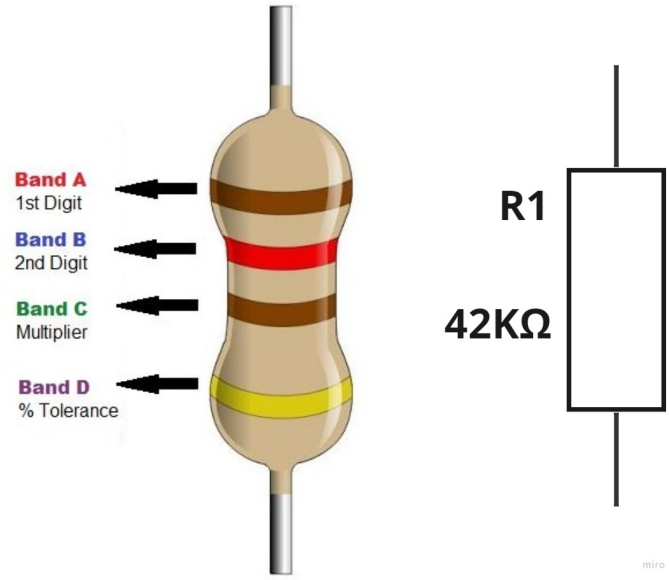


Figure 9: Resistor and its schematic symbol

There is no point knowing the resistor without knowing the **Ohms law**. It is one of the core electrical laws and in its principle is really simple. Ohms laws describes relationship between *current*, *voltage* and *resistance*. Ohm law says that current flowing through a circuit is proportional to the voltage applied with constant resistance.

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

A good analogy is shown at Figure 10. Voltage tries to "*push*" as much current trough circuit, while resistance (and resistor itself) limits the current.

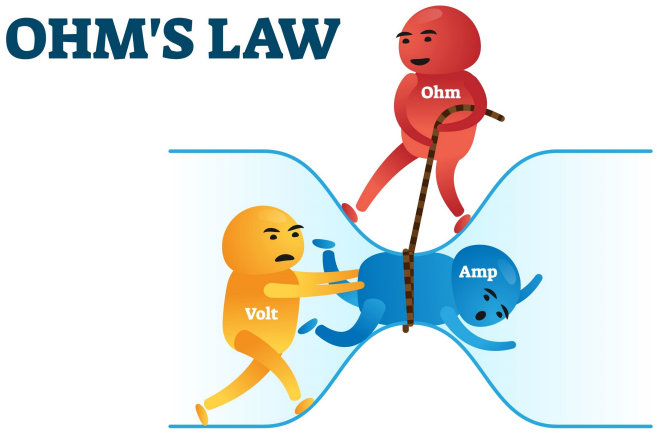


Figure 10: Ohm's law

Variable Resistor

A variable resistor - as the name suggests - is a **resistor which value can be changed**. Unlike the resistor which has 2 pins, the variable resistor (or *potentiometer*) has 3 pins as can be seen on Figure 11. It has the same goal as resistor - **limit current flow** in the circuit or to act as a **voltage divider**.

It's working principle is simple and goes as follows: two side pins act together as a normal resistor. Middle pin - a slider - slides across resistive path and changes resistance relative to slider and another side pin. As the resistance changes, also the voltage drop across the slider and other pins change.

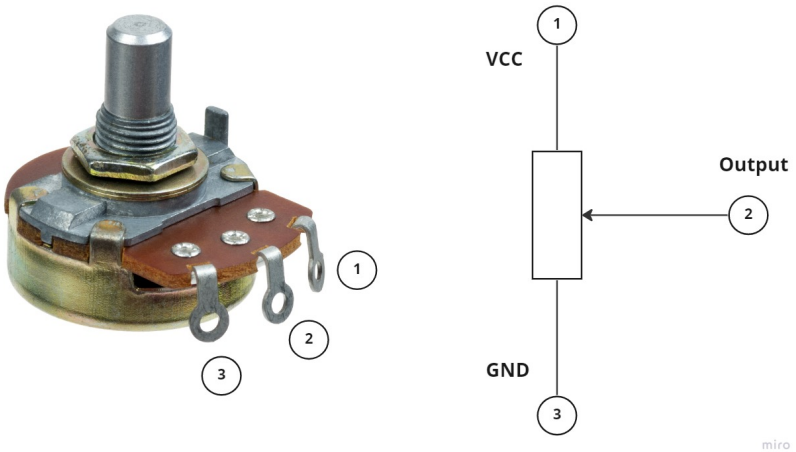


Figure 11: Potentiometer and its schematic symbol

It is used for example as volume control for speakers. As you rotate the knob at the potentiometer, voltage at the slider pin changes - which can be translated by other electrical circuits into sound level.

LED

LED or **Light Emitting Diode** is a type of diode that emits light when current flows through it. The diode itself as an electrical part allows current to flow only one way. It has 2 pins, **Anode** and **Cathode**. In most applications there needs to be a **resistor in series** with the diode, in order to limit current flow, to prevent burning the LED. LEDs are mainly used for indication.

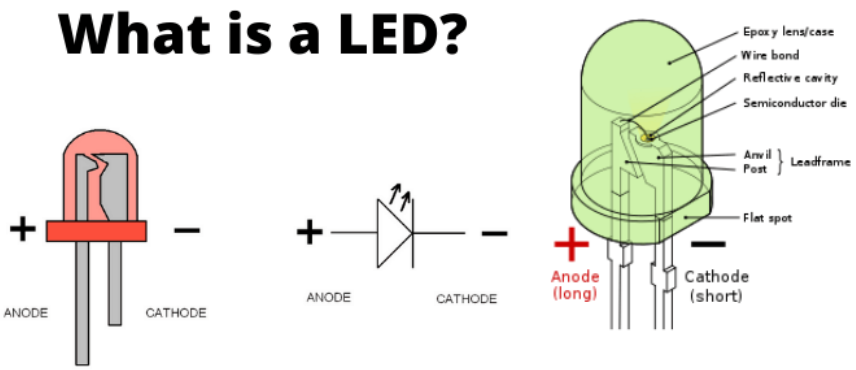


Figure 12: Diagram of a LED

Push Button

Another simple electrical part covered in this laboratory is push button. A push button (or just button) acts like an element that can **disconnect or connect the conducting path** in an electrical circuit. There are many types of buttons, but the button used in this laboratory is a momentary **mechanical push button** - meaning that when not pressed, the button returns into its original state (either on/off).

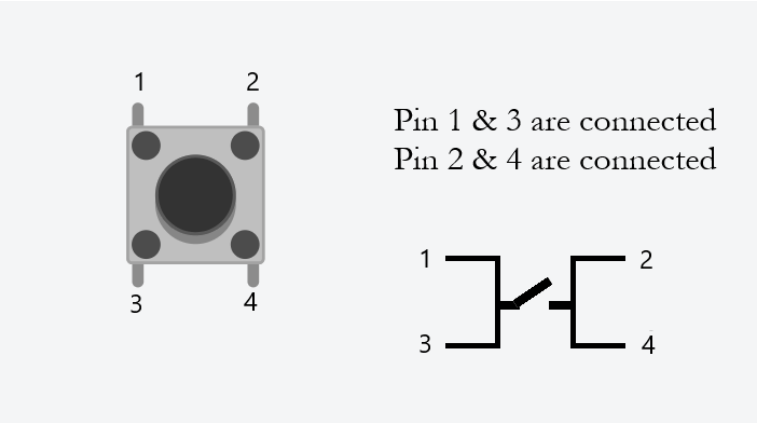


Figure 13: Push button and its schematic symbol

In many cases, there is either **pull-up** or **pull-down resistor** in combination with the button. The function of these resistors is to **ensure a known state for a signal**. If there were no pull-down/pull-up resistors and button would simply translate signal from one pin to other, in case of a disconnected conducting path, button output side would not be connected to anything. The pin would be in state of a **high impedance** and reading analog or a logic value would consist of an undefined behavior.

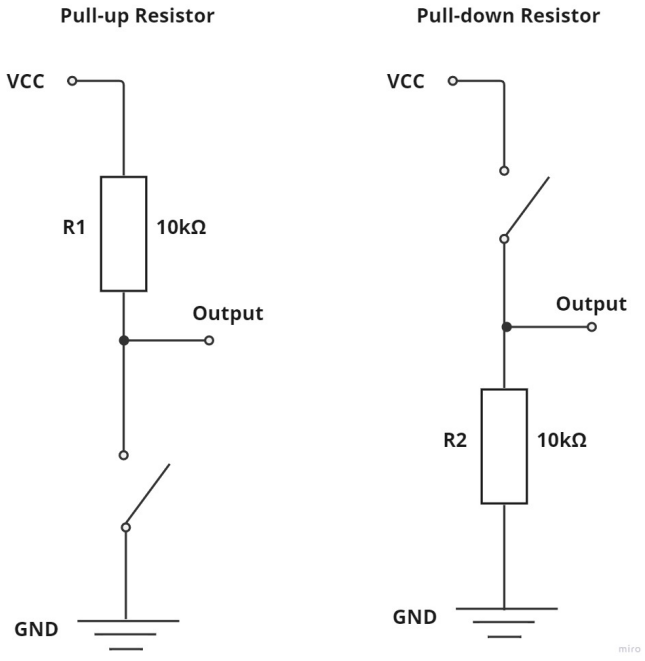


Figure 14: Pull-up and Pull-down resistors

Oscilloscope

An oscilloscope is a type of electronic test instrument that **graphically displays varying electrical voltages** as a two-dimensional plot of one or more signals as a function of time. It has for each channel a probe. **The probe** is simply a cable that connects the measured circuit to the oscilloscope (probes are more complex than that, but in case of this laboratory this explanation is sufficient). It has 2 clips - **Alligator clip** which is used for ground reference and **Measuring clip** which is used for measured signal.

(a) Oscilloscope (b) Probe

Figure 15: Oscilloscope and its probe

Digital signal

A digital signal is type of signal that **represents data as a sequence of discrete values**. Vast majority of digital devices in this world use binary values 1/0. This binary value is translated into the level of voltage in electrical circuits. **Logic 1** can be 12V, 5V, 3.3V, or 1.25V. In case of an Arduino mega used in this laboratory, it is **5 volts**. **Logic 0** is usually in every case translated onto **0 volts**.

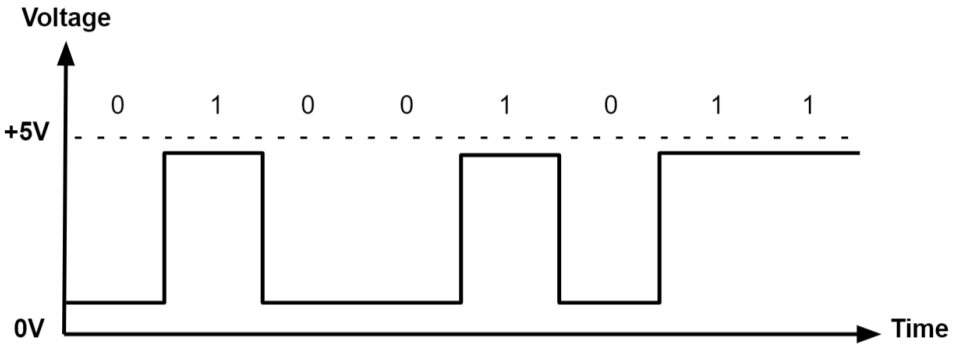


Figure 16: Digital signal

PWM

PWM or **Pulse-Width Modulation** is a method of reducing the **average power delivered** by an electrical signal, by effectively chopping it up into discrete parts. It has 2 attributes - **Frequency and Duty**. Switching frequency has to be fast enough for the controlled device (such as motor or LED) for the waveform to be perceived as a smooth signal. Duty cycle describes how long from one period is signal "ON" or in logic state 1.

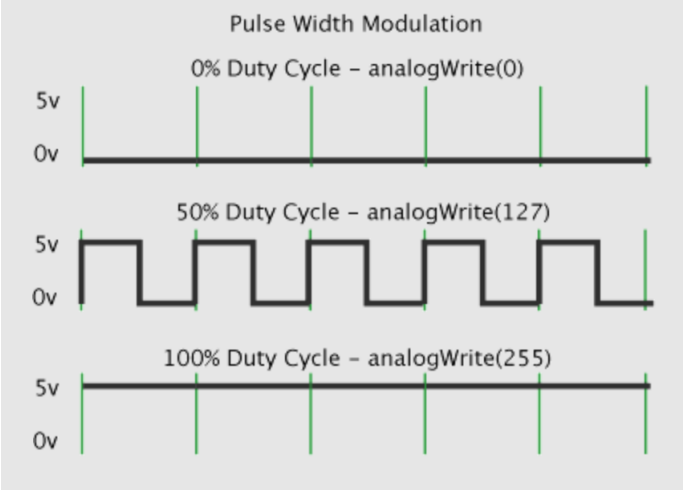


Figure 17: PWM

Analog signal

Analog signal, unlike digital signal, can have **any value in a given range**. In case of Arduino Mega, given range is **<0 Volts, 5 Volts>**. In order for an Arduino to be able to work with analog values, voltage level needs to be translated into digital signal. The component that is used for this is called **Analog to Digital Converter** - or **ADC**, shortened. Arduino has multiple analog to digital converter connected to analog pins. Its resolution is 10-bits, meaning that the analog value is stored in a 10-bit number. **Range for 10 bit number is <0, 1023>**.

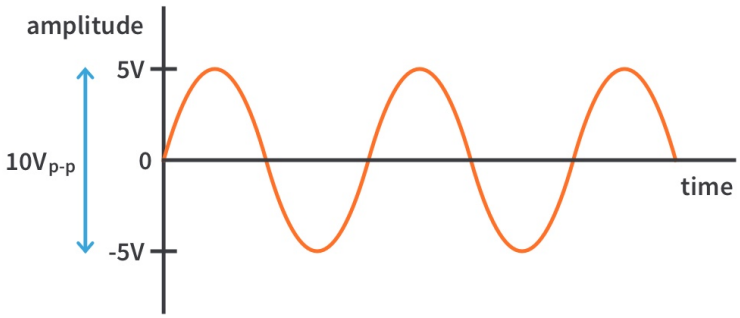


Figure 18: Analog signal

Programming Arduino

Arduino can be programmed by **Arduino IDE** and programming language needed to know is **c/c++**. Arduino offers many built-in functions that simplify work with GPIO, protocols, serial communication, etc. Therefore, the learning curve is not that steep.

Basic frame of code contains 2 function **void setup()** and **void loop()**. As their names suggest, setup() functions, is used for **setting up values, pins, communications, etc**. This function runs only once at the beginning of each code run. The function void loop() runs after setup() in endless loop. This is the place where you write the main part of your code.

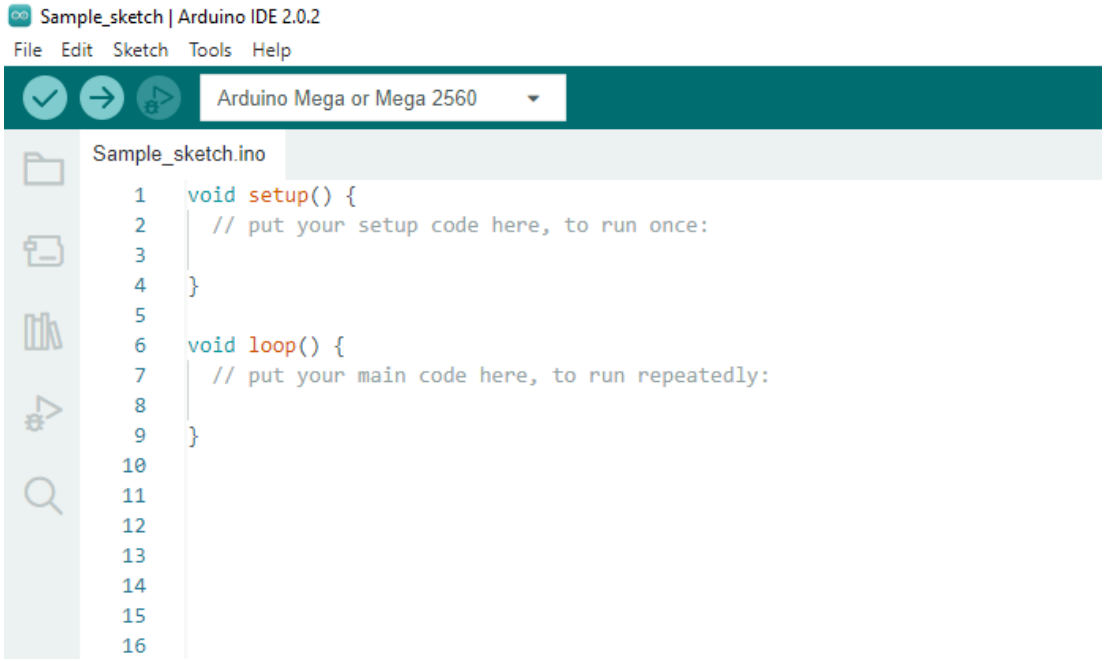


Figure 19: Empty project with starting **setup()** and **loop()** functions

Even though Arduino offers usage of a high level constructions in its code like classes, it is also possible to write and compile a low level code like Assembler. For purposes of this course there is no need to know any of the mentioned above, but you should know basic syntax and construction for loops, conditions and variable declarations.

```
// If condition

int x = 0;
if(x == 10){
  do_something();
}else{
  do_something_else();
}
```

```
// For loop

for(int i=0; i < 10; i++){
  do_something();
}
```

```
// While loop

int i = 0;
while(i < 10){
  do_something();
  i++;
}
```

Figure 20: Basic syntax of a code

For each of the described signals - **Digital, Analog, PWM** - Arduino has built-in functions. Arduino can either **read or write** these signals. Before reading or writing a signal using any **pin**, it **has to be initialized** as either input or output. This initialization is usually in setup function. Table below contains list of functions necessary to know for basic work with Arduino.

pinMode(pin, INPUT/OUTPUT)	-	Sets pin to be either INPUT or OUTPUT (choose one). Usually used in setup()
digitalWrite(pin, HIGH/LOW)	-	Writes digital value 1/0 (0 Volts, 5 Volts) to pin.
digitalRead(pin)	-	Reads digital value from pin. Returns read value
analogWrite(pin, value)	-	Writes analog value to pin. Value has to be from range <0, 255>. This range corresponds to <0 Volts, 5 Volts>
analogRead(pin)	-	Reads analog value from pin. Returns read value. Value is from range <0, 1023> which corresponds to <0 Volts, 5 Volts>
delay(value)	-	Stops the program for given number of milliseconds
delayMicroseconds(value)	-	Stops the program for given number of microseconds

Exercises

Basic LED blinking

Check if your Arduino is "Alive" and upload your first code.

1. Open file **Blinking_LED.ino** with Arduino IDE using **File -> Open...**
2. Take a look at the code
3. Specify correct port and board in Arduino IDE in tools-board and tools-port
4. Upload the code by clicking at  or pressing **Ctrl+U**
5. Built in LED on Arduino board should be blinking in 1 second intervals

Blinking LED on breadboard Part-1

Create a simple circuit with LED on breadboard and experiment with it.

1. Open file **Blinking_LED_Breadboard.ino** with Arduino IDE using **File -> Open...**
2. Take a look at scheme **LED_scheme.png** and create circuit as it shown at the image
3. Upload the code by clicking at  or pressing **Ctrl+U**
4. LED should be blinking in one second intervals

Blinking LED on breadboard Part-2

If you successfully finished previous exercise, continue experimenting.

1. Continue working with the **same code and scheme**
2. Modify the time intervals that are stored in constants **TIMETIME_ON** and **TIMEIME_OFF** as it is described in table below
3. **After each change, upload code to Arduino**
4. What behaviour of the LED do you observe?

	TIME_ON	TIME_OFF
1.	100	100
2.	30	30
3.	1	1
4.	1	10
5.	1	20

PWM - custom method

Create PWM waves using knowledge from last 2 exercises.

1. Open file PWM_custom_method.ino with Arduino IDE using **File -> Open...**
2. Reuse circuit from previous exercise
3. Upload the code
4. Try and modify constant **DUTY_CYCLE_PERCENTAGE** and set it to values 1, 2, 5, 10, 100
5. Observe behavior of LED
6. How would you do it, if you would have to blink win 2 LEDs with different duty cycles?

PWM - proper method

Use built-in Arduino function to create PWM wave.

1. Open file **PWM_proper_method.ino** with Arduino IDE using **File -> Open...**
2. Reuse circuit from previous exercise
3. Upload the code
4. Try and modify integer **brightness** and set it to values 0, 1, 5, 10, 50, 100, 255
5. Observe behavior of LED

PWM - Fading LED

Write your own code that will continually brighten and dim LED

1. Open file **PWM_fading_LED.ino** with Arduino IDE using **File -> Open...**
2. Reuse circuit from previous exercise
3. Follow the instructions and implement **brightening and dimming of the LED**
4. Upload the code
5. Test your implementation by uploading your code and observing LED

Serial communication

Control LED using serial communication.

1. Open file **Serial_Demo.ino** with Arduino IDE using **File -> Open...**
2. Reuse circuit from previous exercise
3. Upload the code
4. Open Serial Plotter (or Serial Monitor), select "**No Line Ending**" and send values in range <0,255> to control brightness of the LED

Button

Sending signals is only half of the knowledge. Other half is to know how to read them.

1. Open file **Input_button.ino** with Arduino IDE using **File -> Open...**
2. Take a look at scheme **Button_scheme.png** and create circuit as it shown at the image
3. Upload the code
4. Press the button and observe the behavior using Serial Plotter (or Serial Monitor)

Potentiometer

Read analog signal from potentiometer.

1. Open file **Input_potentiometer.ino** with Arduino IDE using **File -> Open...**
2. Take a look at scheme **Potentiometer_scheme.png** and create circuit as it shown at the image
3. Upload the code
4. Rotate the potentiometer and observe the behavior using Serial Plotter (or Serial Monitor)

Oscilloscope

Use an oscilloscope to read Arduino output signal.

1. Reuse any code that controls LED
2. Take a look at scheme **LED_Oscilloscope_scheme.png** and create circuit as it shown at the image
3. Upload the code
4. Press **autoset** and wait for the oscilloscope to set
5. Take a look at the measurements taken by the oscilloscope
6. Rotate knobs that are used for horizontal and vertical scaling