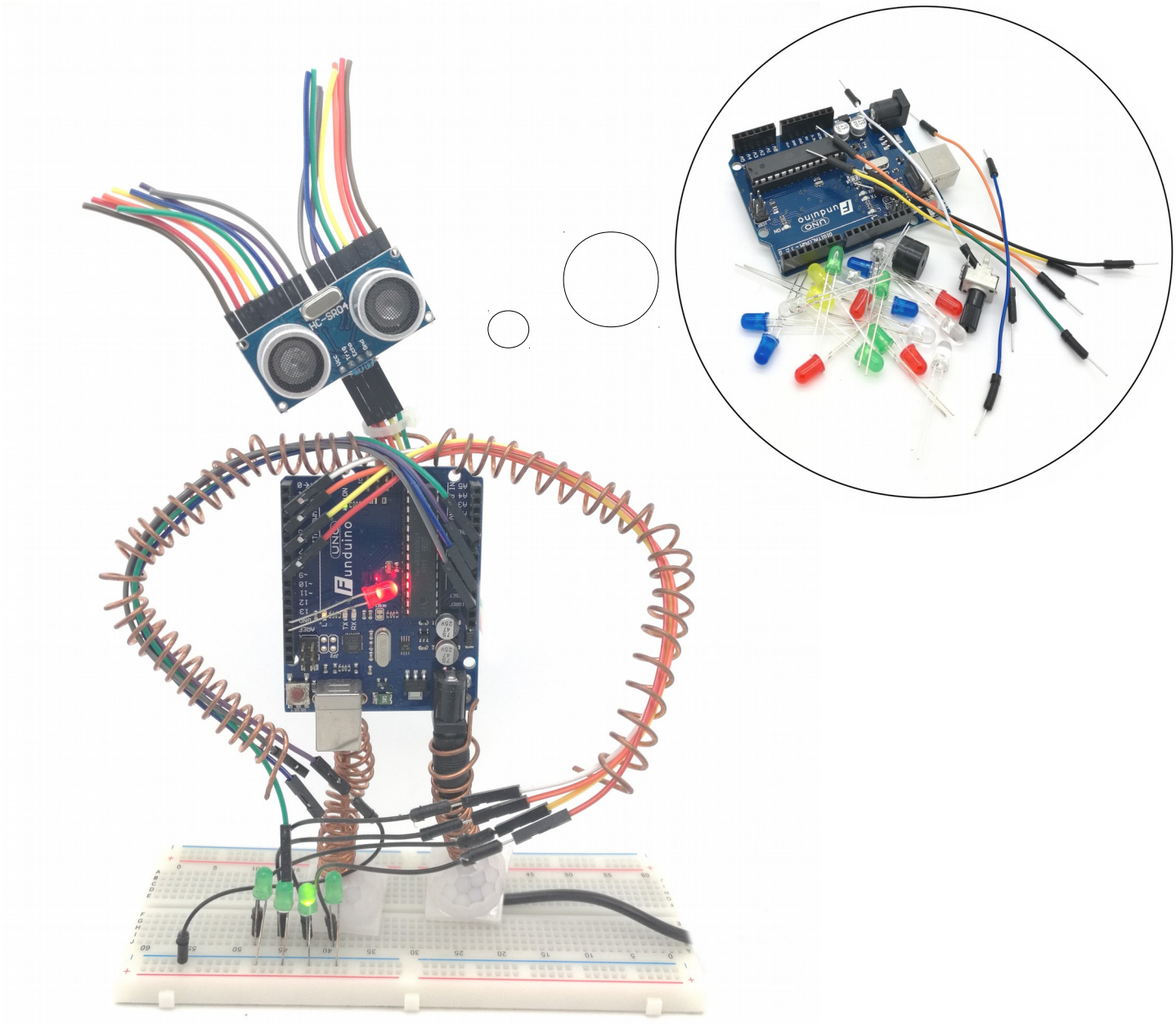
Tutorials for Arduino



This version of our tutorials in english language is a new one (april 2016). Please contact us in case you notice any mistakes: info@funduino.de

Have fun with our tutorials! Kind regards,

Funduino Service Team

Contents

Preface

1. Preface to the Arduino tutorials

These tutorials are meant to be an entry to the Arduino basis. Beginners should get an interesting lead-in the world of Arduino. Our tutorials are all based on practical tasks with theoretical introductions at the beginning. We really recommend to read the theoretical part to successfully complete the practical tasks.

These tutorials were created in the context of a teaching unit. They can be used for free to learn about Arduino, but it's not allowed to copy and use the tutorials without any permission. These tutorials have been created carefully and are continuously maintained, however we can't give any warranty about the accuracy and completeness of the tutorials.

For the practical tasks you'll need some technical equipment. On our website www.funduinoshop.com you can buy especially customized Funduino kits for our tutorials.

What is actually Arduino?

Arduino is an Open-source-electronic-prototyping-base for simple used hardware and software in the field of microcontrolling. It is suitable to realize fascinating projects in a short time. Many of them can be found on Youtube under „Arduino“. It is mostly used by artists, designer or tinkers to realize creative ideas.

But Arduino is also increasingly used by universities and schools to teach an interesting and simple beginning to the world of microcontrolling.

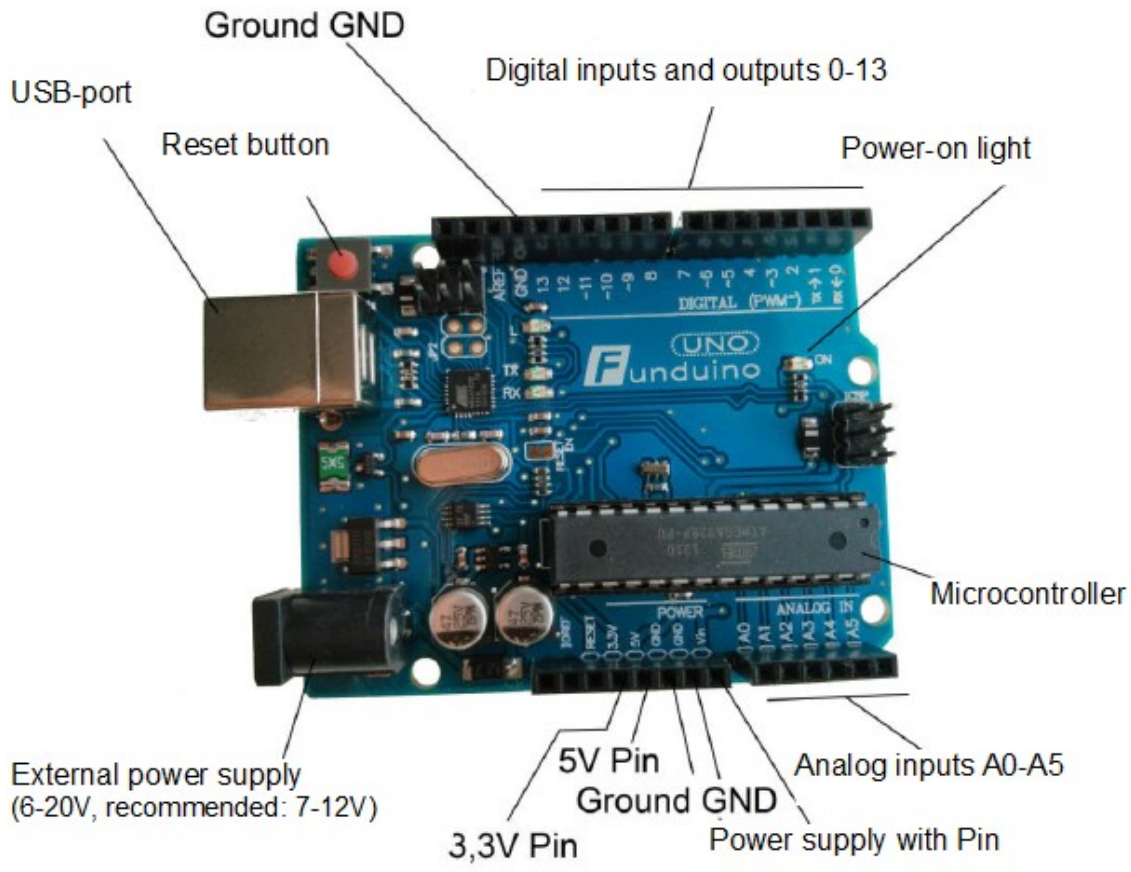
2. Hardware and Software

The term „Arduino“ ist mostly used for both components. The hardware (Arduino Boards) and the corresponding software (Arduino).

2.1 Hardware

The Arduino hardware is a so-called microcontrolling board (Following called „board“). Basically it is a circuit board with many electronic parts around the actual microcontroller. On the edge of the board are many pins with whom it is possible to connect different components. Some of them are for example: Switches, LED's, Ultrasonic sensors, temperature sensors, displays, stepper, etc..

There are different kind of boards, that can be used with the Arduino sofware. Different sized “**official”** boards, with the official “Aduino” name on it, but also many, mostly cheaper, but equivalent Arduino **“fitting”** boards. Typical official boards are called Arduino UNO, Arduino MEGA, Arduino Mini, etc. Arduino compatible boards are for example Funduino UNO, Funduino MEGA, Freeduino, Seeduino, Sainsmart UNO etc..



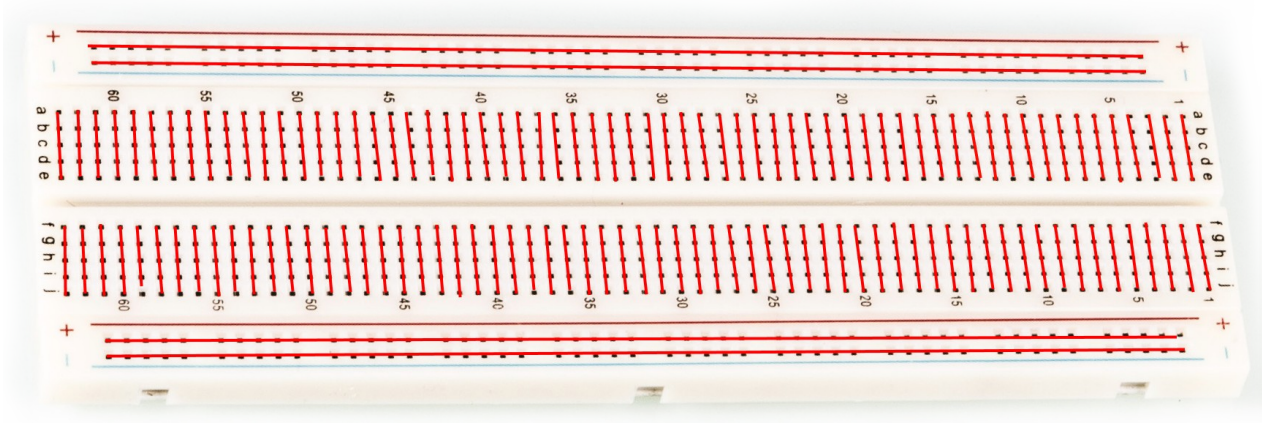
2.1.2 Description of typical equipment

Beside sensors and actuators you need, as a base for quick and flexible experimental setups, jumper cable combined with a breadboard. This way you won't need to solder. Furthermore the LEDs are useful to check the signal output of the board.

2.1.2.1 Breadboard

A Breadboard is a helpful tool to build circuits without any soldering. Certain contacts are connected with each other. Therefore it is possible to connect many cables with each other without soldering or screwing them together.

This image below shows in color, which contacts are connected.



2.1.2.2 LED (light emitting diode)

With LEDs it is possible to check the results of projects real quick. Because of that they're useful for almost every Arduino project. On the internet are many information about LEDs. The most important information:

The electricity can only get through the LED in one direction. So the LED has to be connected the right way to work. There is a longer and a shorter contact at the LED. The longer one is the positive (+) and the shorter one is the negative (–) contact.

The LED is only designed for a specific voltage. If there isn't enough voltage the LED won't shine as bright as it should. If there's to much voltage for the LED, it will get really hot (ATTENTION) and burn out.

Typical voltage data for the different colors of LEDs: blue: 3,1V, white: 3,3V, green: 3,7V, yellow: 2,2V, red: 2,1V. The voltage on the microcontrollerboards is 5V. So the LED shouldn't be connected to the board directly, but with a resistor between it in the circuit.

Non-committal recommendation for resistors at different LEDs (while connecting to the 5V pins on the microcontroller boards):

LED:	white	red	yellow	green	blue	IR
resistor:	100 Ohm	200 Ohm	200 Ohm	100 Ohm	100 Ohm	100 Ohm

2.2 Software

The software that is used to program the microcontroller, is open-source-software and can be downloaded for free on www.arduino.cc. With this “Arduino software” you can write little programs witch the microcontroller should perform. This programs are called “Sketch”.

In the end the sketches are transferred to the microcontroller by USB cable.
More on that later on the subject “programing”.

2.2.1 Installation

Now one after another the Arduino software and the USB driver for the board have to be installed.

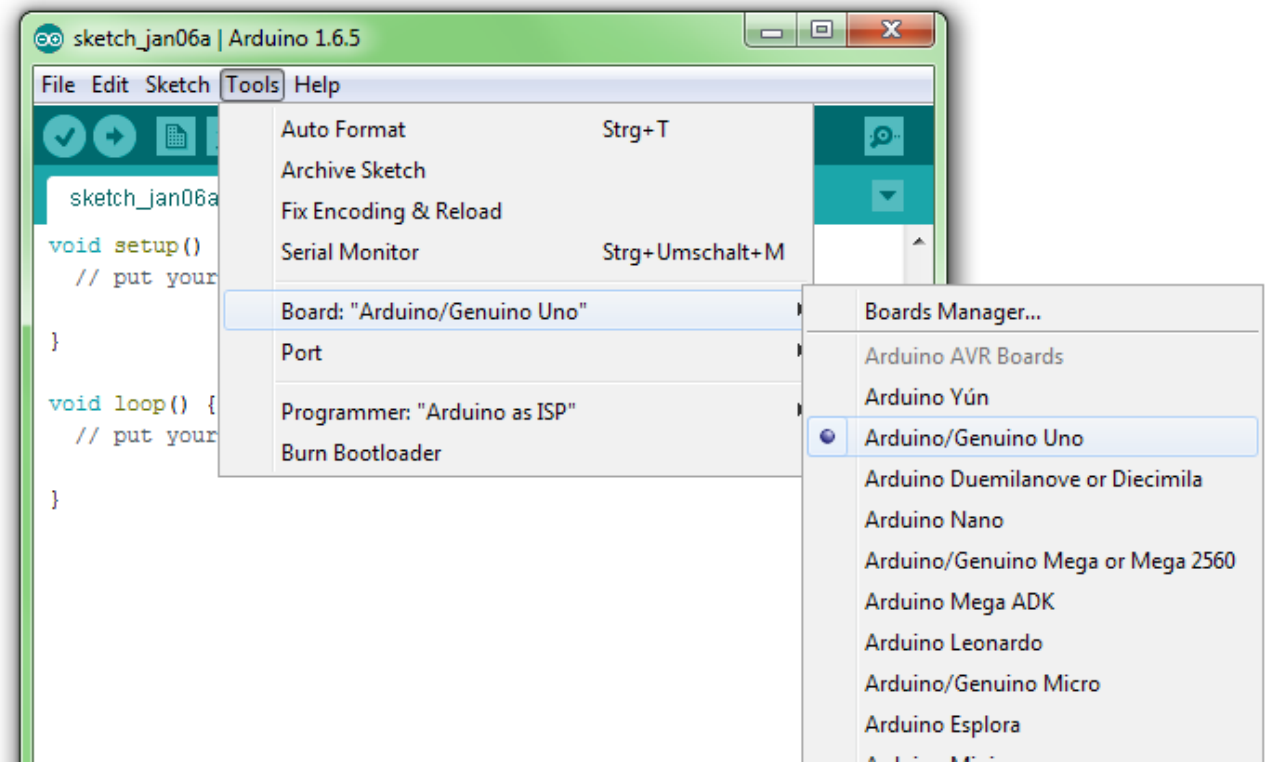
2.2.1.1 Installation and set up of the Arduino software

1. Download the Arduino software on www.arduino.cc and install it on the computer (The microcontroller NOT connected to the PC). After that you open the software file and start

the program named arduino.exe.

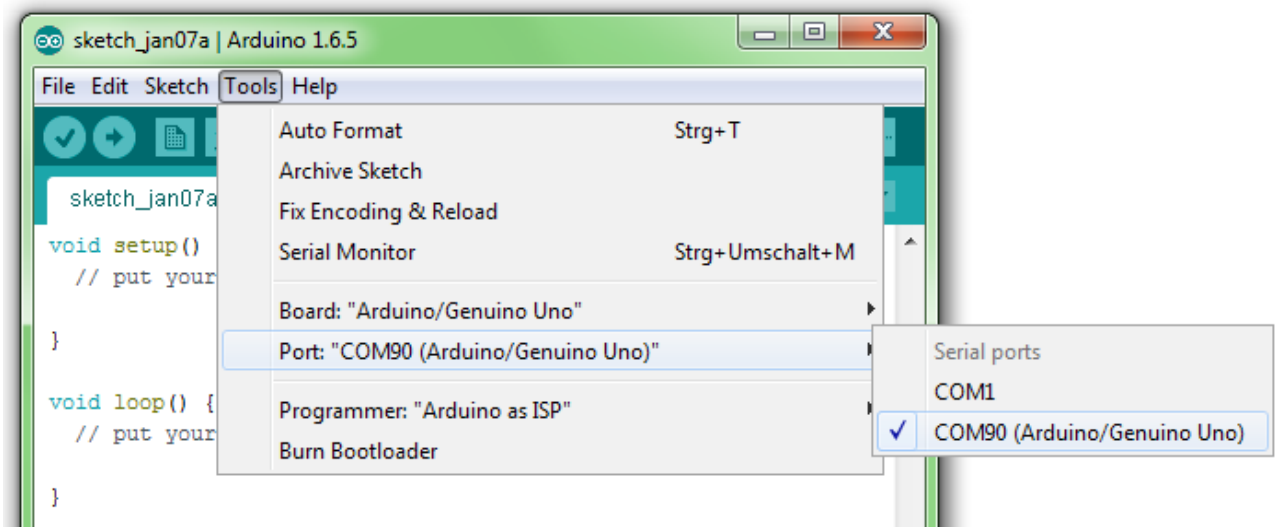
Two set ups on the program are important and should be considered.

a) The board that you want to connect, has to be selected on the arduino software. The “Funduino Uno” is here known as “Arduino / Genuino Uno”.



b) You have to choose the right “Serial-Port”, to let the Computer know to which port the board has been connected. That is only possible if the USB driver has been installed correctly. It can be checked this way:

At the moment the Arduino isn't connected to the PC. If you now choose “Port”, under the field “Tool”, you will already see one or more ports here (COM1/ COM2/ COM3…).
The quantity of the shown ports doesn't depend on the quantity of the USB ports on the computer. When the board gets connected to the computer, YOU WILL FIND ONE MORE PORT.



2.2.1.2 Installation of the USB driver

How it should be:

1. You connect the board to the computer.

2. The Computer recognizes the board and suggests to install a driver automatically.

ATTENTION: Wait a second! Most of the time the computer can't find the driver automatically to install it. You might choose the driver by your own to install it. It can be found in the Arduino file under “Drivers”.

Control: At the control panel of the Computer you can find the “Device manager”. If the board has been installed successfully, it should appear here. When the installation has failed, there is either nothing special to find or you will find an unknown USB device with a

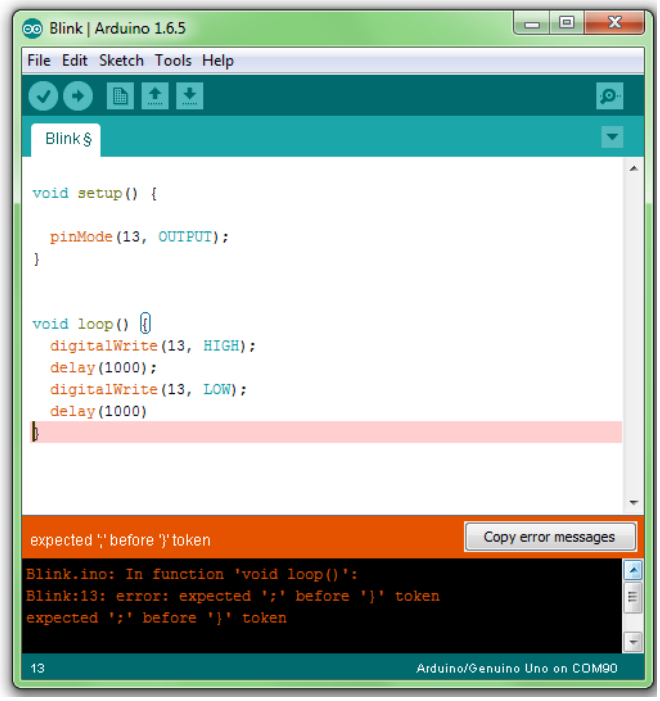
yellow exclamation mark. In this case: Click on the unknown device and choose “update USB driver”. Now you can start over with the manual installation.

3. Programming

Now we can start properly. Without to much theoretical information we start directly with programming. Learning by doing. On the left side you can find the “sketches”, on the right the accompanying explanation for the commands in grey. If you work trough the tutorials with this system, you will soon understand the code and be able to use it by yourself. Later on you can familiarize yourself with other features. These tutorials are only meant as first steps to the Arduino world. All possible program features and codes are referred on www.arduino.cc under „reference“.

First of all a short explanation for possible error reports that can appear while working with the Arduino software. The two most common ones are:

1) The board is not installed right or the wrong board is selected. After uploading the sketch, there will appear an error report underneath the sketch. It looks like the one in the picture on the right. The note “not in sync” shows up in the error report.



2.) There is a mistake in the sketch. For example, a word is misspelled or a bracket is missing. In the example on the left the last semicolon in the sketch is missing. In this Case the error report often starts with “excepted..”. This means that the program is still expecting something that is missing.

Basic structure of a sketch:

A sketch can be divided in three parts.

1. Name variable

In the first part elements of the program are named (This will be explained in program no. 3). This part is not absolutely necessary.

2. Setup (absolutely necessary for the program)

The setup will be performed only once. Here you are telling the program for example what Pin (slot for cables) should be an input and what should be an output on the boards.

Defined as Output: The pin should put out a voltage. For example: With this pin a LED is meant to light up.

Defined as an Input: The board should read out a voltage. For example: A switch is actuated. The board recognized this, because it gets a voltage on the Input pin.

3. Loop (absolutely necessary for the program)

This loop part will be continuously repeated by the board. It assimilates the sketch from beginning to end and starts again from the beginning and so on.

Let's start!

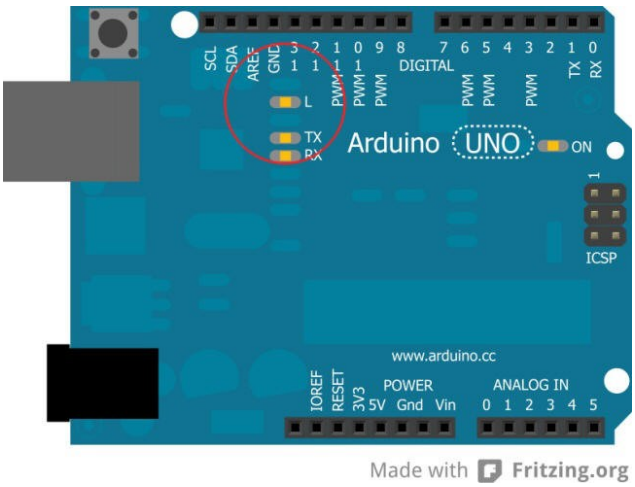
Tutorials:

Blinking LED

Task: Get a LED to blink.

Required equipment: Just the microcontroller board with the USB cable.

On the board a LED is already build on the pin 13 (for test purpose). This LED often already blinks, after connecting a new board to the computer, because during manufacturing the blink program is uploaded for test purposes. We are going to program this blinking by ourself.



Circuit:

The meant LED is circled in red on the image above.

You only have to connect the board properly with the computer.

1.1 First part of the program: Name variables

– First we don't do anything here.

1.2 Second part of the program: Setup

We only have one output – Pin 13 should put out voltage (The LED should light up.).

We start writing in the white area of the arduino software:

```
void setup()  //The setup begins here

{  //opening curly bracket – A program part begins here

}  //closing curly bracket – A program part is ending here
```

Now we are going to write the setup information between the curly brackets.

In this case: “pin 13 is supposed to be an output” :

```
void setup() //The setup begins here

{  //A program part begins here

pinMode(13, OUTPUT);  //Pin 13 is supposed to be an ouput.

}  //A program part is ending here.
```

1.3 Third part of the program: Loop (main part):

```
void setup()  //The setup begins here
{   //A program part begins here
pinMode(13, OUTPUT);  //Pin 13 is supposed to be an ouput.
}  //A program part is ending here.
void loop()  //The main part of the program begins here
{   //A program part begins here
}   //A program part is ending here.
```

Now we bring in the loop part (main part).

THIS IS THE COMPLETE SKETCH:

```
void setup()  //The setup begins here
{   //A program part begins here
pinMode(13, OUTPUT);  //Pin 13 is supposed to be an ouput.
}  //A program part is ending here.
void loop()  //The main part of the program begins here
{  //program part begins here
digitalWrite(13, HIGH);  //Turn on the voltage on pin 13 (LED on)
delay(1000);  //Wait for 1000 milliseconds (one second)
digitalWrite(13, LOW);  //Turn off the voltage on pin 13 (LED off)
delay(1000);  //Wait for 1000 milliseconds (one second)
}  //Program ends here
```

Done! The sketch should look just like the one in the image on the right. Now we have to upload it to the board. By clicking on the button with the arrow on the upper left of the software, you will upload the sketch to the board.

1.4 Now you have the option to modify the program. Example: You want the LED to blink really fast. Therefore we will shorten the waiting time (From 1000ms to 200ms)

```
void setup()  //The setup begins here
{  //A program part begins here
pinMode(13, OUTPUT); //Pin 13 is supposed to be an ouput.
}  //A program part is ending here.
void loop() //The main part of the program begins here
{  //program part begins here
digitalWrite(13, HIGH);  //Turn on the voltage on pin 13 (LED on)
delay(200);  //Wait for 200 milliseconds
digitalWrite(13, LOW);  //Turn off the voltage on pin 13 (LED off)
delay(200);  //Wait for 200 milliseconds
}  //program part ends.
```

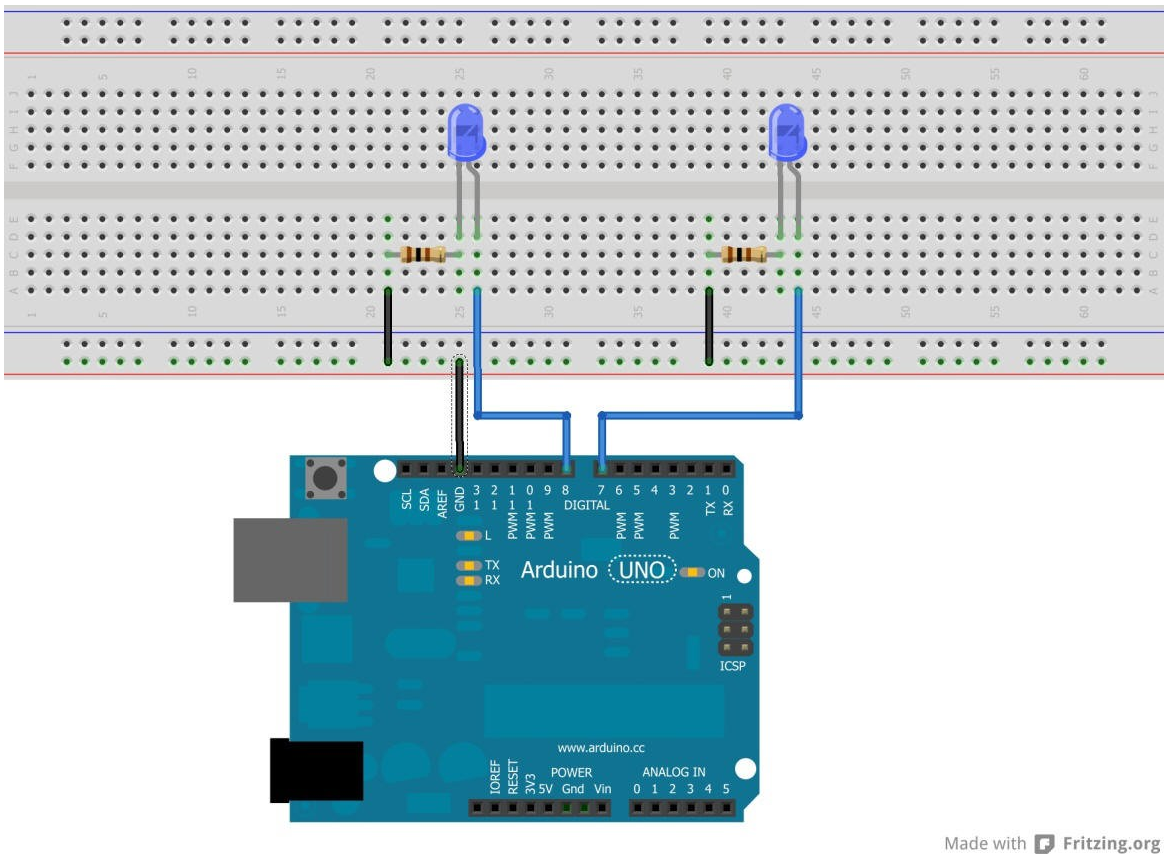
The new modified sketch has to be uploaded to the board again. Now if everything has worked properly the LED should blink faster.

Alternately blinking LED

Task: We want to let two LEDs blink alternately.

Required equipment: Microcontroller / two LEDs (blue) / two resistors with 100 Ohm / Breadboard / cables

Setup:



Code:

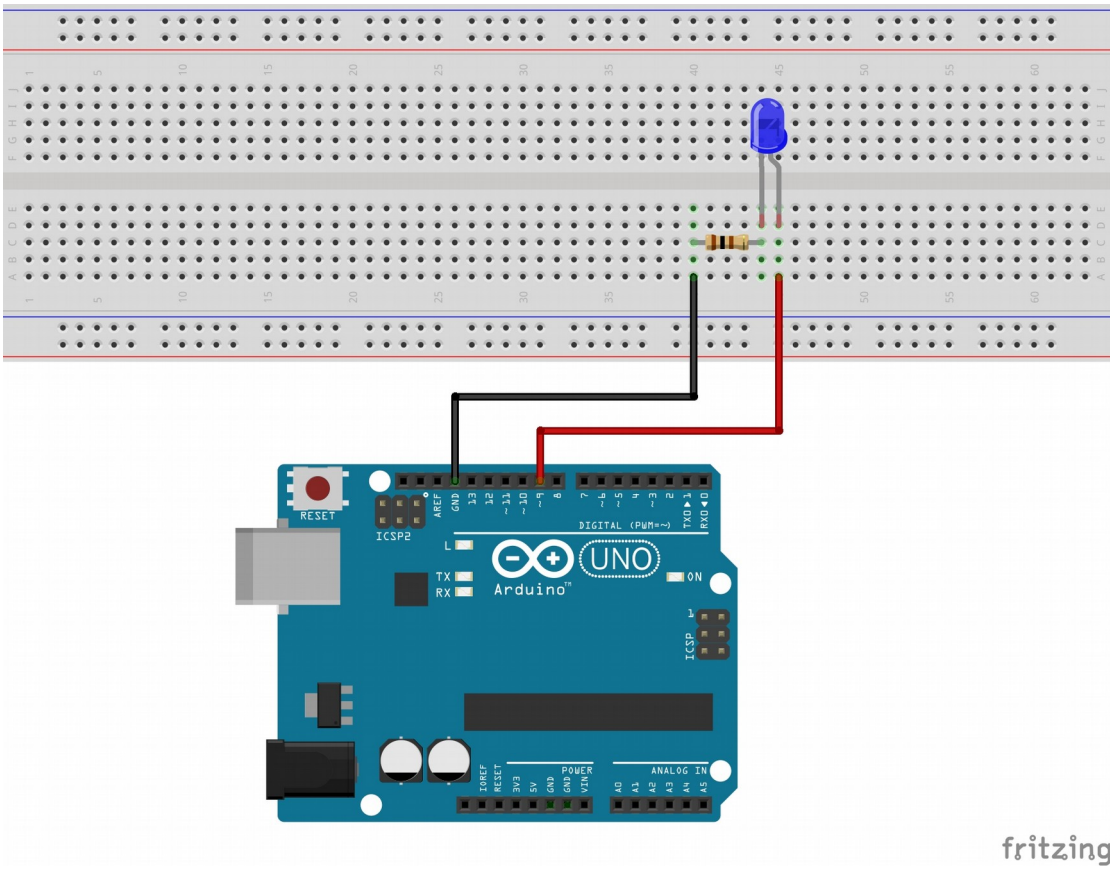
```
void setup()

{   //We are starting with the setup

pinMode(7, OUTPUT);   //Pin 7 is defined as output

pinMode(8, OUTPUT);   //Pin 8 is defined as output

}

void loop()

{   //The main part starts

digitalWrite(7, HIGH);   //turn on the LED on pin 7

delay(1000);   //wait for 1000 milliseconds

digitalWrite(7, LOW);  //turn off the LED on pin 7

digitalWrite(8, HIGH);   //turn on the LED on pin 8

delay(1000);   //wait for 1000 milliseconds

digitalWrite(8, LOW);  //turn off the LED on pin 8

}   //Here at the end of the loop the program starts again from the beginning of
//the loop part. So..turn on LED on pin 7..wait for 1000
//milliseconds..etc..etc..
```

Fading LED

Task: A LED should pulsating get brighter and darker (fade).

Required equipment: Microcontroller / one LED (blue) / resistor with 100 Ohm / Breadboard / cables

Setup:



The Arduino is a digital microcontroller. It only knows “5 Volt on” and “5 Volt off” on its outputs. But to vary the brightness of an LED, it ought be vary the voltage on the outputs. For example 5 Volts if the LED shines bright, 4 Volts if it shines a bit darker and so on. THIS DOESN'T WORK ON DIGITAL PINS. But there is an other option. It is called pulse width modulation (short PWM). PWM lets the 5 V voltage fade. The voltage is turned on and off for milliseconds. With a really high PWM the 5 V signal nearly gets constant on the pin. With a low PWM it is the other way around and the 5 V signal is barely there (This is only a reduced summary, so you should look it up on the internet, if you need more information). With this PWM it is possible to get nearly the same effect on a LED, as if the

voltage would get varied. Not every pin on the board has the PWM function. The pins with this function are specially labelled, for example with a little wave in front of the pin number (see image on the bottom of page 22).

Let's go!

Code:

```
int LED=9;  //The word “LED” stands for the number 9.

int brightness=0;  //The word “brightness” stands for the value that is emitted
//at the PWM.The number “0” is only used as an initial value.

int fading=5;  //”fading” sets up the speed of the fading.

void setup()

{    // The setup begins here.

pinMode(LED, OUTPUT);     //The pin with the LED is supposed to be an output.

}

void loop()

{   //The loop part begins here.

analogWrite(LED, brightness);    //The function analogWrite is used to activate
// the PWM output on the pin with the LED. The value of the PWM is saved under
//the word “brightness”. In this case it is “0”.

brightness=brightness + fading; // Now we modify the value of the PWM output. We
//add the value of the fading to the value of the brightness. In this case:
//brightness = 0+5. The new value that is standing for brightness isn't 0 any
// longer but 5. When the program has ran through the loop part once it will
//start over again. The next pass the value will be 10. After that it will be
// 15... etc.

delay(25); //The LED should only stay bright for a really short time like 25
//milliseconds. If you reduce that time the fading will also get faster.
```

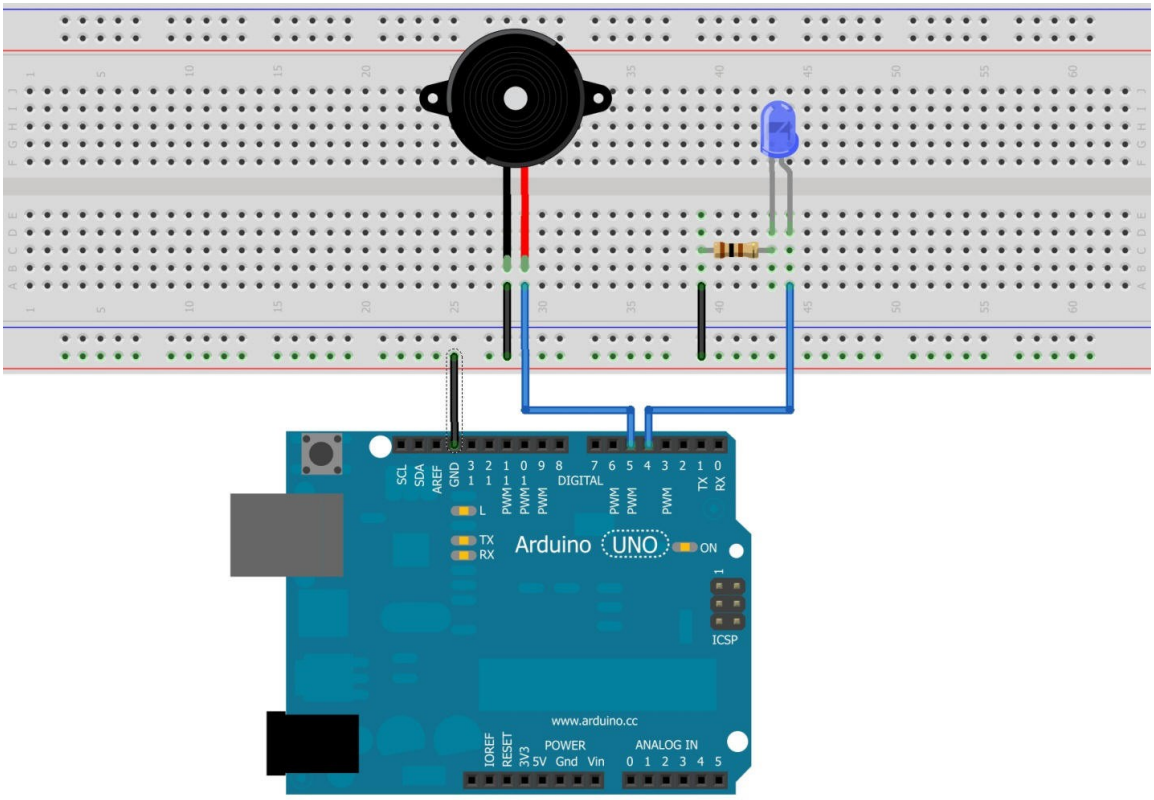
```
if(brightness==0 || brightness== 255){

fading= -fading;

} //Meaning of this command: If the Value of brightness will reach 0 OR 255, the
//value of fading will change from negative to positive and the other way
//around. The Reason why is, that while the program is running trough the loop
//part, the LED will shine brighter and brighter. But at the point of 255 as a
//value of the PWM, it will reach it's maximum. At this point the LED should get
//darker step by step. Because of this the fading will be negate at this point.
//This means the next time the program runs through the loop the part
//“brightness=brightness+fading;” will effect that the LED gets darker. Example:
//“brightness=255+(-5)”. The value of brightness will be 250 after that. The
//next time it will be 245..etc. As the value of brightness will reach 0 the
//value of fading will get positive again.

} // With this bracket the loop part ends.
```

Light and sound

Task: A LED and a piezo speaker are supposed to blink or beep continuously.

Required equipment: Microcontroller / one LED / resistor with 200 Ohm / Breadboard / piezo speaker / cables

Setup:



Code:

```
int LED=4;  //this time we also going to use the first part of the program. Here
//we are going to put in variables. This means that there will be a letter or a
//word standing for a number. In this example the LED is connected to pin 4 and
//the speaker to pin 5, so we rename pin 4 and pin 5, to avoid confusion. The
//word “LED” now stands for the number 4 and the word “beep” for the number 5.

int beep=5;

void setup()

{   //We are starting with the setup

pinMode(LED, OUTPUT);   //pin 4 (pin “LED”) is supposed to be an output

pinMode(beep, OUTPUT);   //Pin 5 (pin “beep”)  is supposed to be an output

}

void loop()

{   //The main part starts

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

digitalWrite(beep, HIGH);   //turn on the speaker

delay(1000);   //wait for 1000 milliseconds (sound and light)

digitalWrite(LED, LOW);    //turn off the LED

digitalWrite(beep, LOW);   //turn off the speaker

delay(1000);   //wait for 1000 milliseconds (no sound and no light)

}   //Here at the end of the loop the program starts again from the beginning of
//the loop. So it will beep and light up again. If you change the break (delay)
//it will be either beep and light up faster or slower.
```

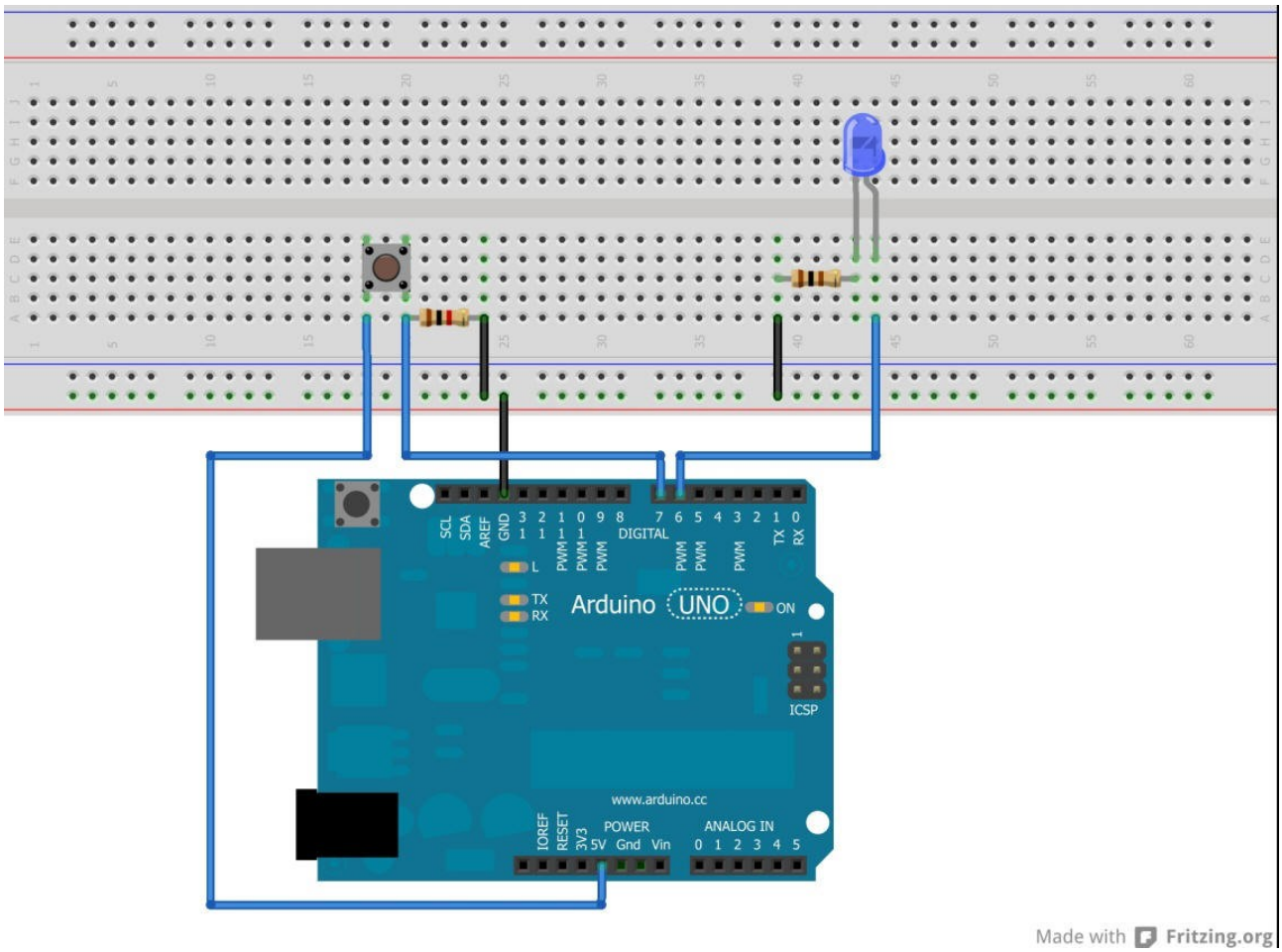
Push button and LED

Task: After pushing the button an LED is supposed to light up for 5 seconds.

Required equipment: Arduino / one LED (blue) / one resistor with 100Ohm / one resistor with 1K Ohm (1000 Ohm) / Breadboard / Cables / Push button

The digital pins of the microcontroller are not only able to put out voltage, they are also able to read out voltage. We are going to try this with the following program. This time there is something special in the setup. If we would simply connect the push button with the microcontroller and push the button, there would be voltage on the pin. You can imagine it like many electrons floating around the pin. When you now release the button, there wouldn't get any more electrons to the pin. Now the difficulty: The electrons that are already floating around the pin are only escaping extremely slow. So the microcontroller thinks that the button has been pushed longer than it actually has been. The microcontroller thinks that the button has been pushed until the electrons have escaped completely from the pin. This problem can be fixed by grounding the pin with a 1000 Ohm (1K Ohm) resistor. Now the electrons are able to escape from the pin faster and the microcontroller recognizes that the button only has been pushed briefly. The resistor is called “PULLDOWN”- resistor, because the resistors is always “pulling down” the voltage to 0V. ATTENTION: If you are using a smaller valued resistor, you can get an electrical short on the microcontroller while pushing the button.

Setup:



Code:

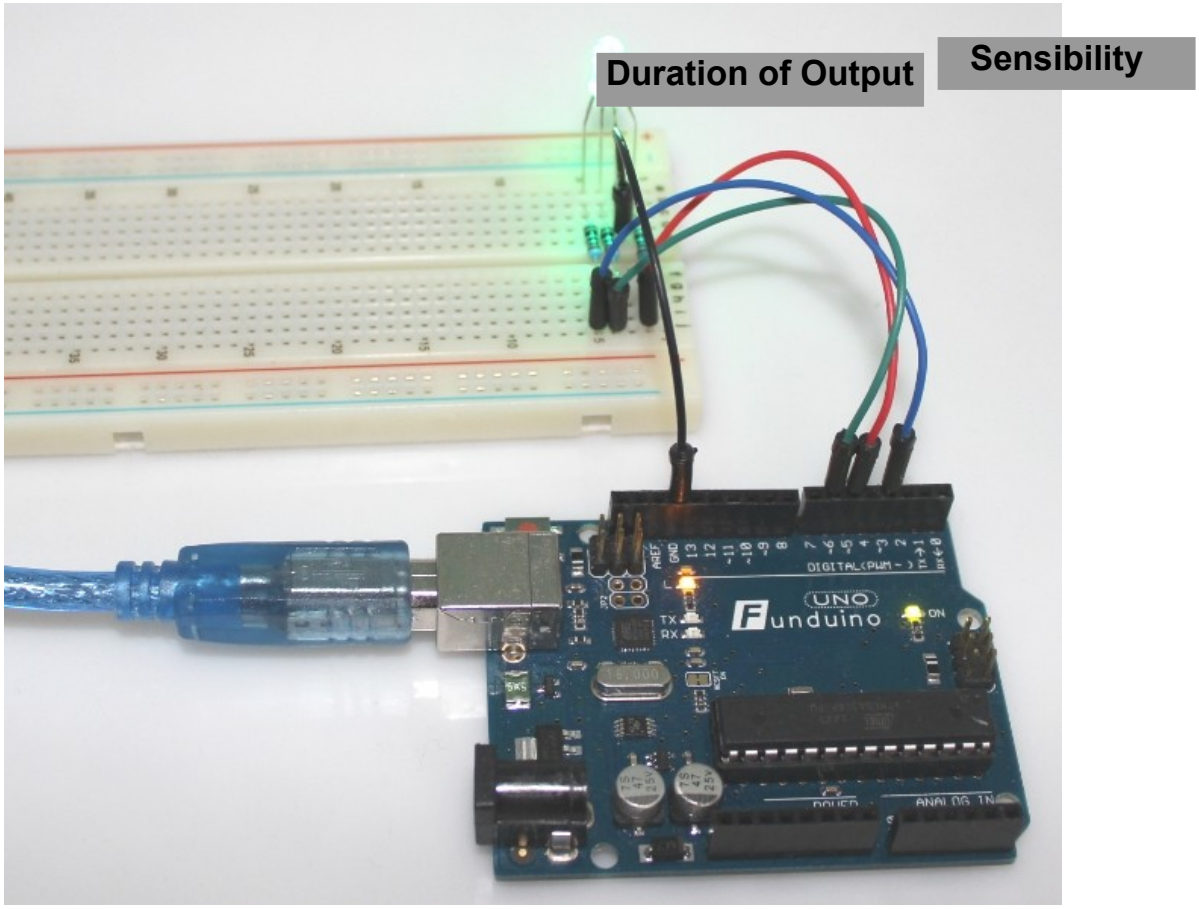
```
int LEDblue=6;    //The word “LEDblue” stands for the value 6.

int button=7;      //The word “button” stands for the value 7.

int buttonstatus=0;    //The word “buttonstatus” stands for the value 0. Later
//on there will be safed wheter the button is pushed or not.

void setup()

{     //The setup starts here

pinMode(LEDblue, OUTPUT);    //The pin connected to the LED (pin 6) is an output

pinMode(button, INPUT);    //The pin connected to the button (pin 7) is an input.
```

```
}

void loop()

{   //with this bracket the loop part starts

buttonstatus=digitalRead(button);   //The value on pin 7 is read out (command:
//digitalRead). The result will be safed under “buttonstatus”. (HIGH means 5V
//and LOW means 0V)

if (buttonstatus == HIGH)  //If the button gets pushed (high voltage value)...


{   //open program part of the IF-command

digitalWrite(LEDblue, HIGH);   //...the LED should light up

delay(5000);     //5000 miliseconds (5 seconds) long

digitalWrite(LEDblue, LOW);   //after 5seconds the LED should turn off

}   //close the program part of the IF-command

else

{    //open the program part of the else-command

digitalWrite(LEDblue, LOW);   //the LED shouldn't light up

}    //close the program part of the else-command

}    //with this bracket the whole loop parts gets closed
```

RGB LED

Task: We want a RGB LED to light up in different colours.

Required equipment: Microcontroller / one RGB LED / three resistors each with 200 Ohm / Breadboard / cables

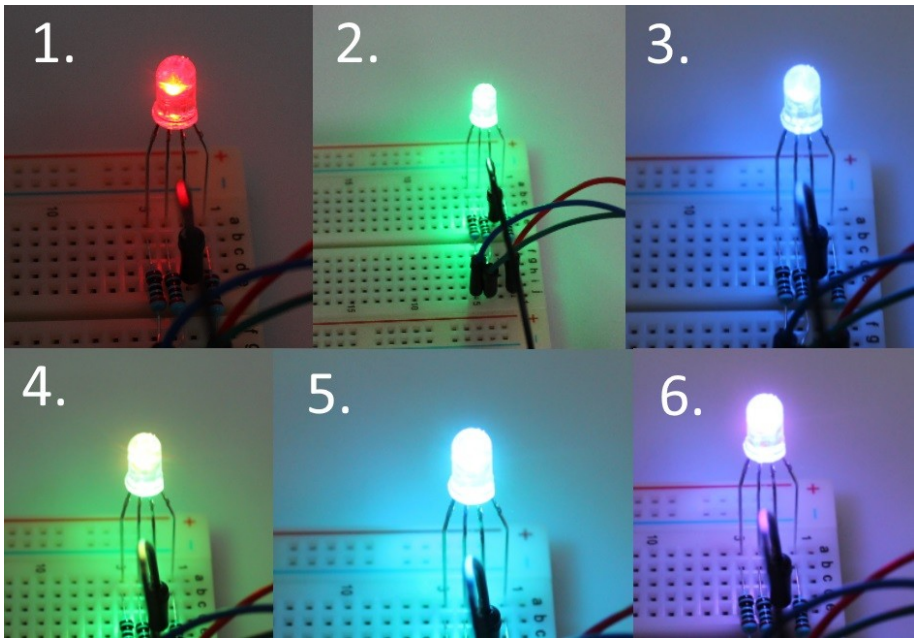


What is an RGB LED? The RGB LED is an LED that is able to shine in different colours. RGB stands for the three colours “red”, “green” and “blue”. Inside the RGB LED are three separate LEDs available, which can be turned on and off individually and shine in three different colours. That is reason behind the four contacts of the RGB LED. The longest one can be (depending on the version) the anode(+) or the cathode(-). The other contacts belong to the different colours.

Version a; “Common cathode” – The longest contact of the RGB LED is “-”. In this case the other three contacts are supposed to get positive gate voltage (5V) “+”.

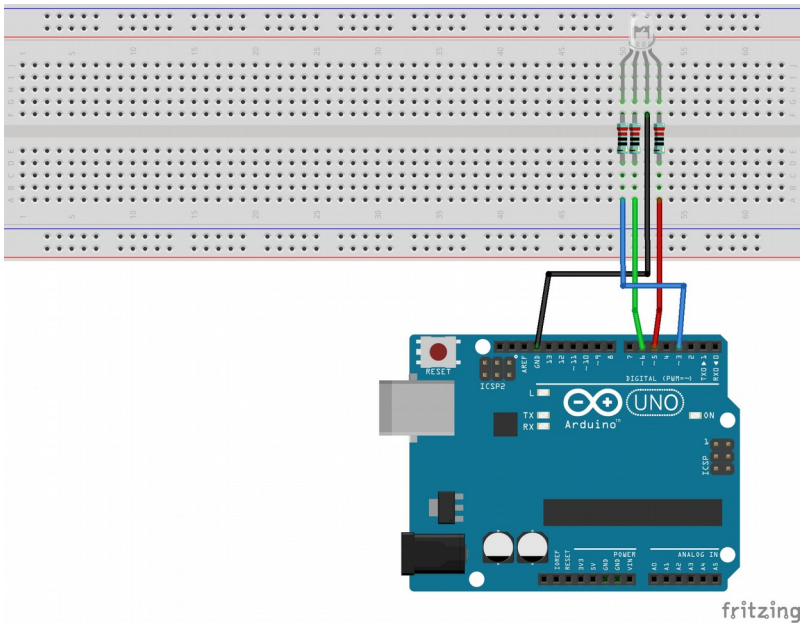
Version b: “Common anode” – The longest contact of the RGB LED is “+”. This means that the other contacts are supposed to get negative gate Voltage (GND) “-”.

It is possible to create much more colours if you temper the colours. For example; you will get “yellow” by tempering “blue” and “green”.

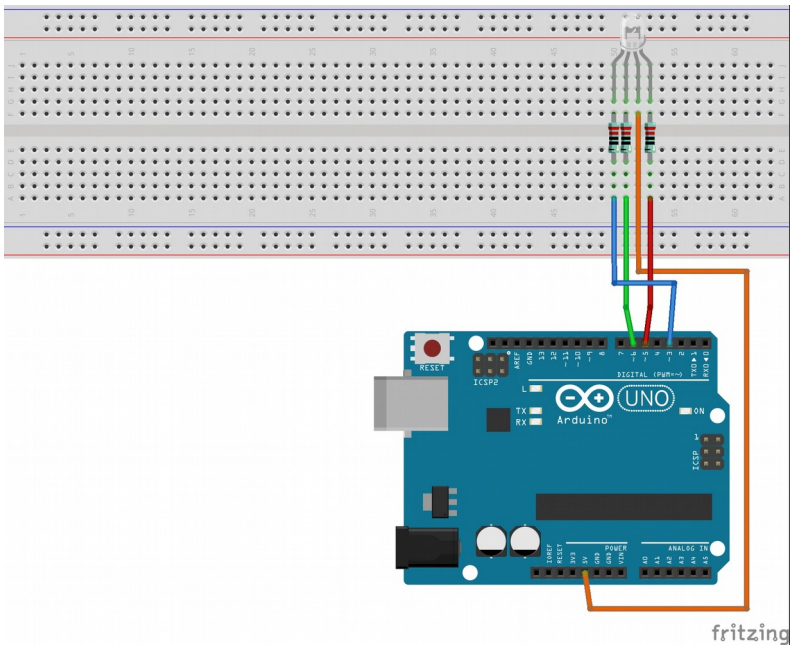


There is a simple way to find out what RGB LED version you have. Just switch “+” and “-” on the LED. Only if the LED is connected the right way it will work and shine.

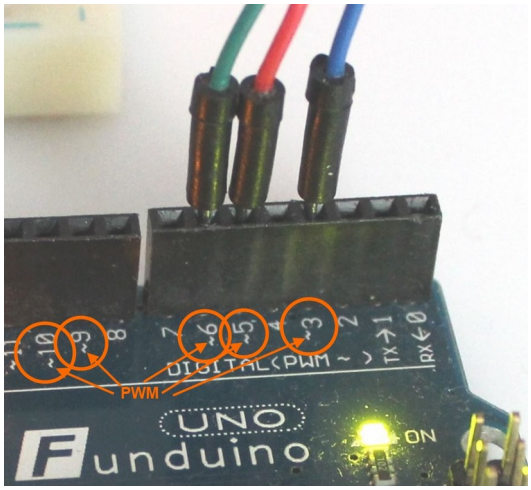
Setup with version a:



Setup for version b:



The Arduino is a digital Microcontroller. On its Outputs it only can “turn on” 5V or “turn off” 5V. But to create the different colours, the three colours of the LED have to be actuate more specific. That is the reason to use the Pulse Width Modulation. The PWM can be used on the pins with the little wave in front of the number.



The PWM lets the voltage pulse from +5V to 0V. So the voltage gets turned off and on for only milliseconds. With a really high PWM the 5 V signal nearly gets constant on the pin. With a low PWM it is the other way around and the 5 V signal is barely there (This is only a reduced summary, so you should look it up on the internet, if you need more information). With PWM it is possible to get nearly the same effect as if the voltage would vary.

The following codes are working for both RGB versions:

Sketch 1:

With this code you can turn on and off the three different colours one by one.

```
int LEDblue=3;  //Blue colour on pin 3

int LEDred=5;   //Red colour on 5

int LEDgreen=6;   //Green colour on pin 6

int b=1000;   //b defines a break of 1000ms (1 second)
```

```
int brightnessblue=150;  //Value between 0 and 255 – defines the brightness of
//the single colour

int brightnessred=150;   //Value between 0 and 255 – defines the brightness of
//the single colour

int brightnessgreen=150;   //Value between 0 and 255 – defines the brightness of
//the single colour

int dark=0;  //Value 0 stands vor 0V voltage – therefore LED off

void setup()

{

pinMode(LEDblue, OUTPUT);

pinMode(LEDgreen, OUTPUT);

pinMode(LEDred, OUTPUT);

}

void loop()

{

analogWrite(LEDblue, brightnessblue);   //Turn on blue

delay(b);  //Break



analogWrite(LEDblue, dark);   //Turn off blue

analogWrite(LEDred, brightnessred);  //Turn on red

delay(b);   //Break

analogWrite(LEDred, dark); // Turn off red
```

```
analogWrite(LEDgreen, brightnessgreen);  //Turn on green

delay(b);  //Break

analogWrite(LEDgreen, dark);   //Turn off  green

}
```

Sketch 2:

With this code always two different colours will be turned on and off together. This way we are able to create the colours yellow, turquoise and purple.

```
int LEDblue=3;  //Blue colour on pin 3

int LEDred=5;   //Red colour on 5

int LEDgreen=6;   //Green colour on pin 6

int b=1000;   //b defines a break of 1000ms (1 second)

int brightnessblue=150;  //Value between 0 and 255 – defines the brightness of
//the single colour

int brightnessred=150;   //Value between 0 and 255 – defines the brightness of
//the single colour

int brightnessgreen=150;   //Value between 0 and 255 – defines the brightness of
//the single colour

int dark=0;  //Value 0 stands vor 0V voltage – therefore LED off

void setup()

{
```

```
pinMode(LEDblue, OUTPUT);
pinMode(LEDgreen, OUTPUT);
pinMode(LEDred, OUTPUT);
}



void loop()
{
analogWrite(LEDgreen, brightnessgreen);  //Green and red on = yellow
analogWrite(LEDred, brightnessred);
delay(b);
analogWrite(LEDgreen, dark);  //Green and red off = yellow off
analogWrite(LEDred, dark);
analogWrite(LEDgreen, brightnessgreen);  //Green and blue on = turquoise
analogWrite(LEDblue, brightnessblue);
delay(b);
analogWrite(LEDgreen, dark);   //Green and blue off = turqouise off
analogWrite(LEDblue, dark);
analogWrite(LEDred, brightnessred);    //Red an blue on = purple
analogWrite(LEDblue, brightnessblue);
```

```
delay(b);

analogWrite(LEDred, dark);   //Red and blue off = purple off

analogWrite(LEDblue, dark);

}
```

Motion detector

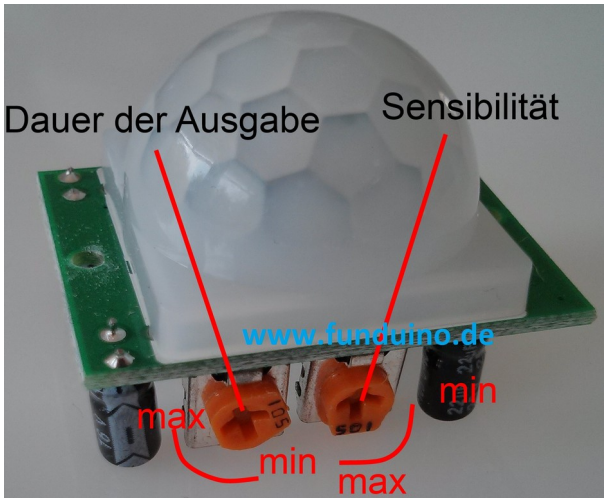
Task: As soon as a motion gets detected, a piezo speaker should beep.

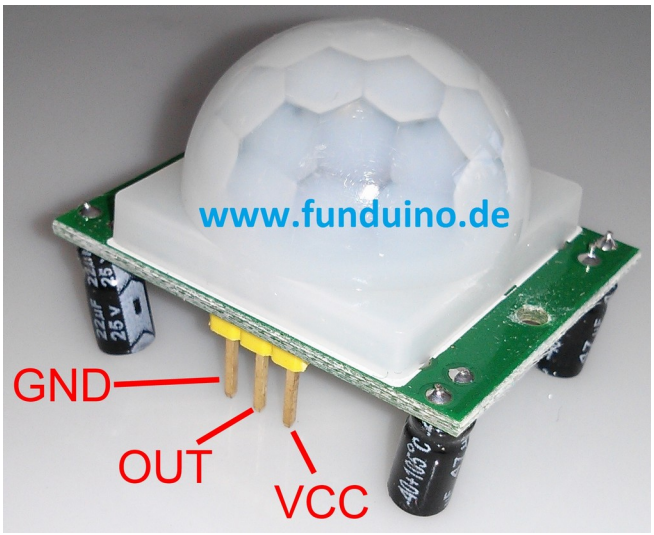
Required equipment: Arduino / motion detector / breadboard / cables / piezo speaker

Learning content: Read out the voltage values of a motion detector and use them for an output.

The motion detector, also known as PIR sensor, is very simply constructed. Once it has detected a movement, it puts out 5V voltage on a pin. Now the microcontroller just has to read this voltage out and processes it.

The duration of the of the output signal and the sensibility (reach) of the motion detector can be adjusted with the two knobs on it (see image on the right).





The plastic lens on top of the motion detector can easily be removed. Underneath of it, there is the IR-detector and the lettering of the three contacts. GND (-), OUT (Signal output), VCC (+).

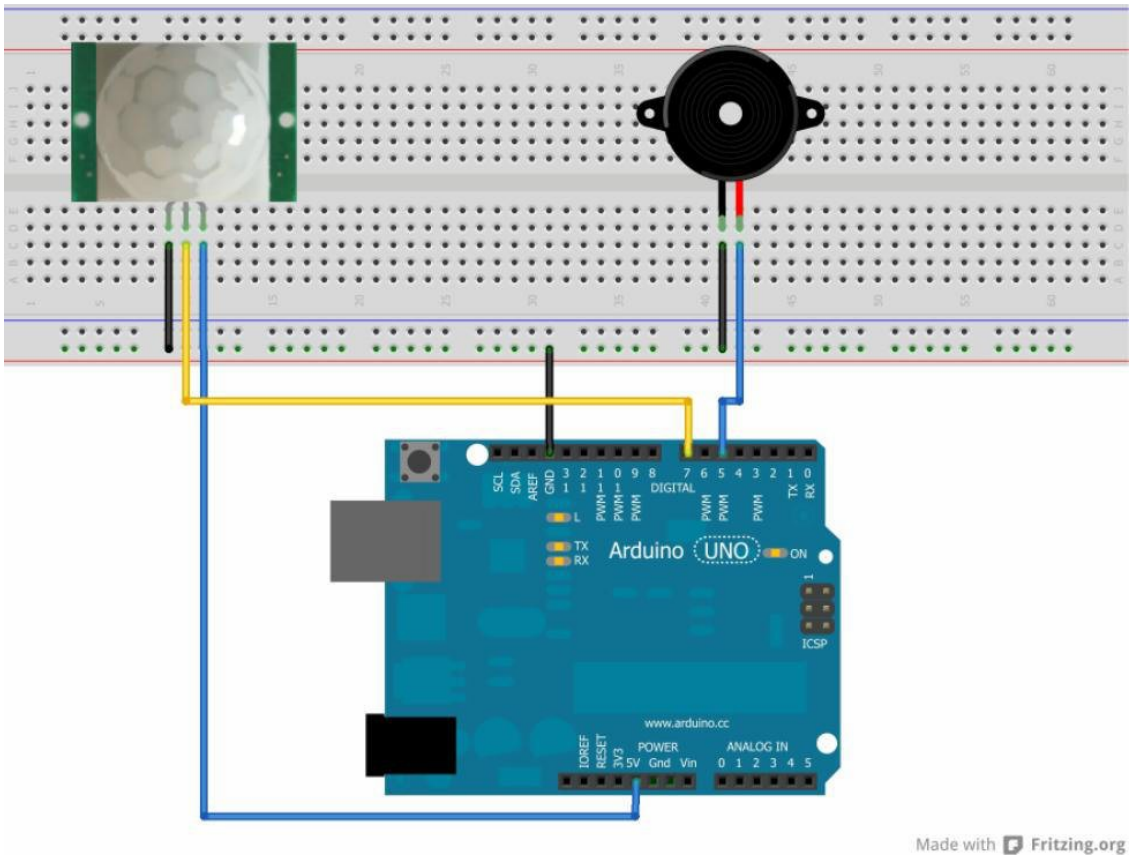
As shown in the image on the left.

Furthermore there is a jumper on the bottom of the detector. This jumper makes it possible to switch between two different modes.

1) Jumper on the outermost: The signal of the output will be maintained for a certain time, after the detector has recognized a movement. But after that certain time the signal will be deactivated, even if movement could be detected. The signal will be activated again after some time.

2) The other mode gets activated if the jumper is placed on the two inner contacts. The output signal will stay constantly active, as long as a movement is detected.
This mode is recommended for projects with Arduino.

Setup:



Code:

```
int piezo=5;      //The word “piezo” stands for the value 5.

int movement=7;   //The word “movement” stands for the value 7.

int movementstatus=0;  //The word “movementstatus” stands for the value 0. Later
//on there will be saved if a movement is detected or not

void setup()

{   //The setup starts here

pinMode(piezo, OUTPUT);  //The pin connected to the piezo speaker (pin 5) is
//defined as an output.
```

```
pinMode(movement, INPUT);  //The pin connected to the moition detector (pin 7)is
//defined as an input.

}

void loop()

{    //The loop part starts here

movementstatus=digitalRead(movement);  //The value on pin 7 is read out
//(command: digitalRead). The result will be safed under “movementstatus”. (HIGH
//means 5V and LOW means 0V)

if(movementstatus==HIGH)   //if a movement is detected (voltage signal high) ..

{   //open program part of the IF-command

digitalWrite(piezo,HIGH);  //..the piezo should beep

delay(5000);  //5 seconds long

digitalWrite(piezo, LOW);   //after that the piezo should be quiet

}  //close program part of the IF-command

else

{   //open else-command

digitalWrite(piezo,LOW); //the piezo speaker should be turned off

} //close else-command

} //close loop part
```

Photo resistor

Now it will get a little bit more complicated!

Task: A LED should light up if it gets dark or rather the photo resistor is covered.

Required equipment: Arduino/ One LED / Resistor with 200 Ohm / Resistor with 10K Ohm / Breadboard / Cables / Photo resistor

Learning content: Read out voltage and show out the values via “serial monitor”.

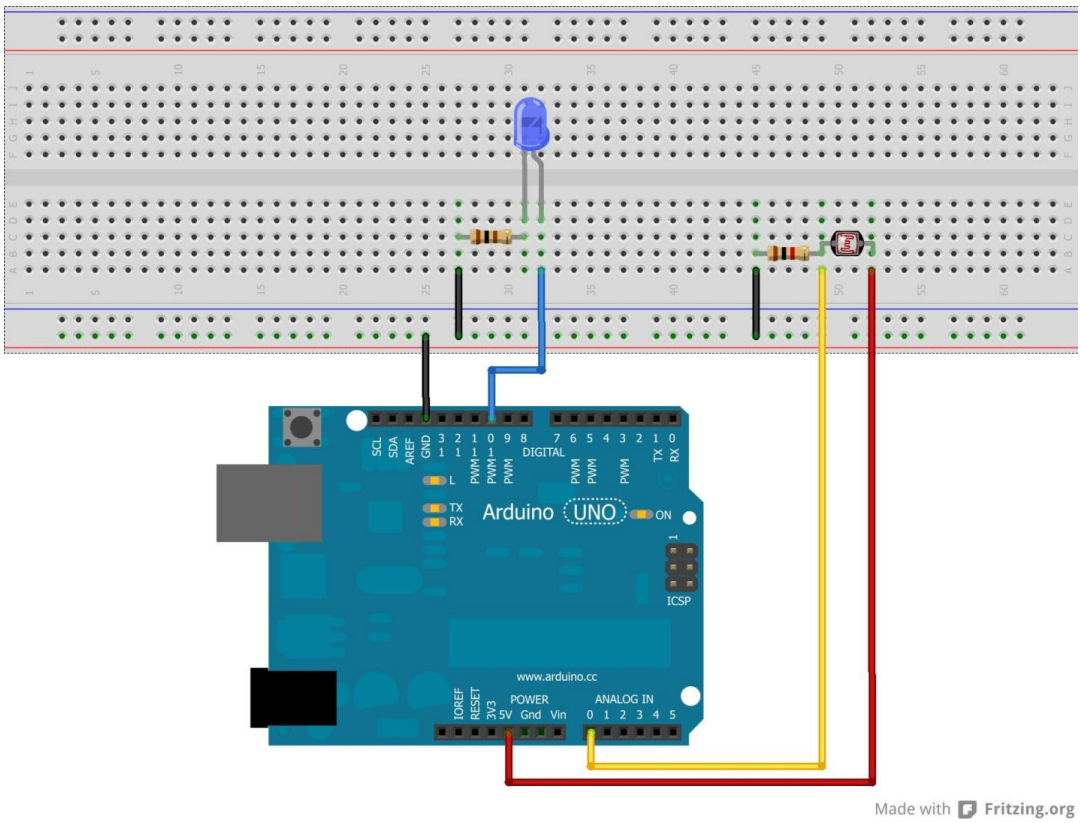
Read out voltage:

We want the microcontroller to recognize how bright it is by reading out a photo resistor. For this purpose we will use a simple physical principle. If two consumers are connected in one circuit (series connection), they also “share” the applied voltage. Example: Two same lamps are connected in series with 6V applied voltage. With a voltmeter we are now able to measure only 3V on each lamp. If we would connect two different lamps (with different values of resistance), than we would be able to measure two different voltage values on each lamp, for example: 1,5V and 4,5V.
A photo resistor changes its resistance depending on the luminous intensity. We will use this effect to get a value for the brightness or darkness by reading out the applied voltage. To create a split voltage we need to connect the photo resistor with a resistor (1-10K Ohm, depending on what kind of photo resistor is used. The resistor should have nearly the same value of resistance as the used photo resistor.) in series. Now the circuit should get connected to ground (GND) and 5V (see setup).
The microcontroller is able to read out analogue signals (voltage) and work with the values. For this function the analog Inputs on the board are used. They are converting the voltage into a number, with that we are able to work with. 0 Volt corresponds to the number 0 and the highest measurable value 5V corresponds to the number 1023 (0 to 1023 are 1024 numbers = 10 Bit). Example: A voltage of 2,5V is measured, so the microcontroller would give out the value 512 (1024:2).

The “serial monitor”

An important tool of the arduino software is the “serial monitor”. With this “serial monitor” it is possible to show the read out data from the microcontroller (numbers or text). It is quite useful, because you will not always have an LCD-Monitor connected to show some values. In this sketch we will use the “serial monitor” to get the values shown, which the microcontroller get from the photo resistor.
Why does that make sense? If we would want the LED to light up if it gets dark, there must be a function in the sketch, that defines: “If the value of the photo resistor gets lower that x, the LED is supposed to light up.” Therefore we have to know how high x is, while it starts to get darker.
Solution: We are going to display the value “x” from the photo resistor (while it is getting darker) on the “serial monitor”. Knowing this, we are later on able to get a function like this in the sketch: “If the read out voltage of the photo resistor gets lower that “x”, turn on the LED”.

Setup:



Sketch:

```
int input=A0;  //"input" stands for the value "A0" (Label of the analog port 0)

int LED=10;   //"LED" stands for the value 10

int sensorvalue=0;   //variable for the sensor value with 0 as starting value

void setup()  //The setup starts here
{

Serial.begin(9600);    //Start the communication with the serial port. We will
//need this to get the read out value of the photo resistor on the serial
//monitor.

pinMode (LED,OUTPUT);    //The pin connected with the LED gets defined as an
//output. We won't need to define the analog pin.

}

void loop()
```

```
{    //loop part starts here

sensorvalue=analogRead(input);   //Read out the voltage of the photo resistor
//and save it under “sensorvalue”



Serial.print("Sensor value =");  //Show “Sensor value=” on serial monitor

Serial.print(sensorvalue);   //Send the value of the photo resistor as a number
//between 0 and 1023 to the serial monitor

if(sensorvalue > 512)    //If the sensor value gets higher than 512…

{

digitalWrite(LED,HIGH);   //...the LED should light up...

}

else

{

digitalWrite(LED,LOW);   //else the LED should be turned off

}

delay(50);  //short break where the LED is turned on or off.

}   //This last bracket closes the loop part

//If the sensor value for example at normal brightness only reaches 100 (this
//value depends on the used resistor, the brightness and the current
//direction), it would make sense to use a lower value than 512 (for example
//90), to turn on the LED. You can look up the current sensor value on the
//“serial monitor”. You //can find it in the Arduino software at “tools”.
```

Potentiometer

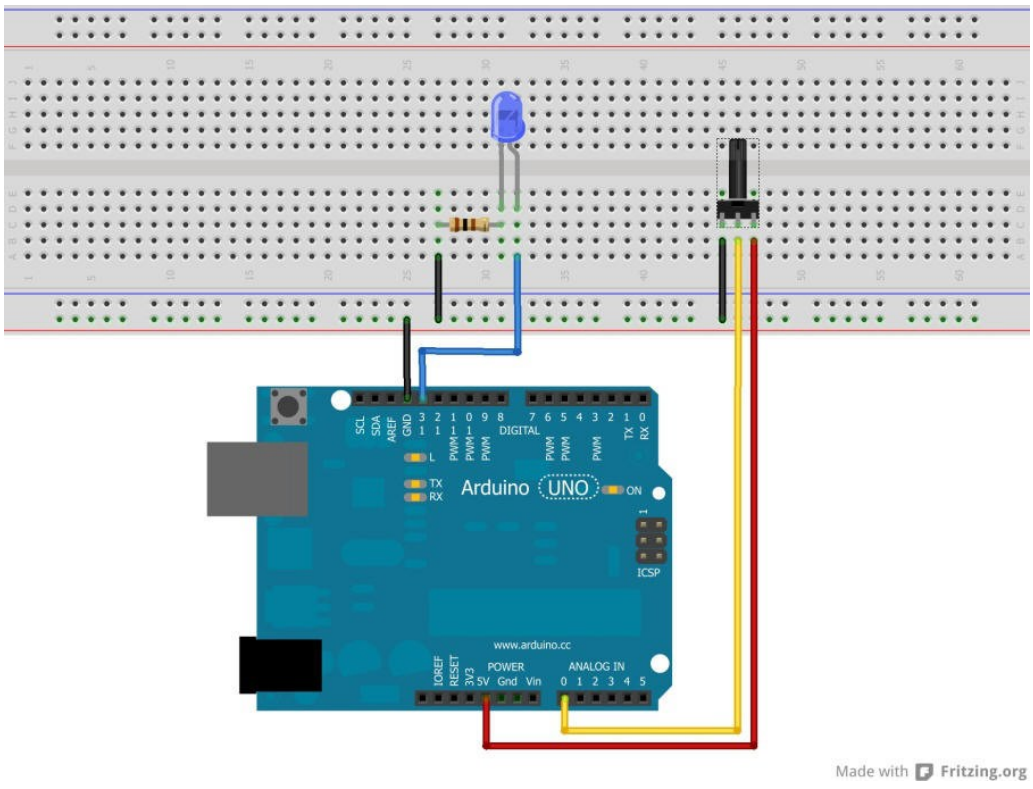
Task: The speed of a blinking LED should be regulated with a potentiometer.

Required equipment: Arduino / Potentiometer / Breadboard / Cables

Learning content: Reading out a potentiometer and working with sensor data on a mathematical basis (in this case for the duration of a break).

There are three contacts on a potentiometer. The outer ones should be connected with + and –. The contact in the middle has to be connected with a analog Input on the microcontroller. If the potentiometer gets turned around, the pin in the middle will give out a voltage between 0V and 5V. Potentiometer on the most left position: 0V and potentiometer on the most right position: 5V, depending on the way it's connected (+ and -).

We will use the already fixed LED with pin 13 on the microcontroller. But it's also possible to connect an additional LED on the Breadboard like it's shown here:



Sketch:

```
int input=A0;  //The word “input” now stands for the value “A0”

int LED=13;   //”LED” now stand for the value 13

int sensorvalue=0;   //Variable for the sensorvalue with 0 as starting value

void setup()

{        //The setup begins here

pinMode (LED,OUTPUT); //The pin 13 connected to the LED is defined as an output

}

void loop()

{       //The loop part begins here

sensorvalue= analogRead(input);    //The voltage at the potentiometer is read
//out and gets saved as a number between 0 and 1023 under “sensorvalue”

digitalWrite (LED,HIGH);    //Turn on LED

delay(sensorvalue);   //The value, which is saved under “sensorvalue” defines
//how long the LED will light up (milliseconds)

digitalWrite(LED, LOW);  //Turn off LED

delay(sensorvalue);   //The value, which is saved under “sensorvalue” defines
//how long the LED is turned off (milliseconds)

}  //End of the loop part

//Now the loop part will be restarted. If the read out value of the
//potentiometer has changed, the time between the on and off stage will also
//change. The LED will be blinking faster or slower. The longest possible delay
//in this sketch can be 1023ms (milliseconds) long. If needed longer delays are
//also possible. Therefore you would have to add a litte mathimatical function
```

```
//into the sketch. Example: The line “sensorvalue=analogRead(input);” has to be
//changed into “sensorvalue=analogRead(input)*2;”. The saved sensor value will
//be increased by the factor of 2. So the longest delay would be 2046ms and so
//on ..
```

Temperature measurement

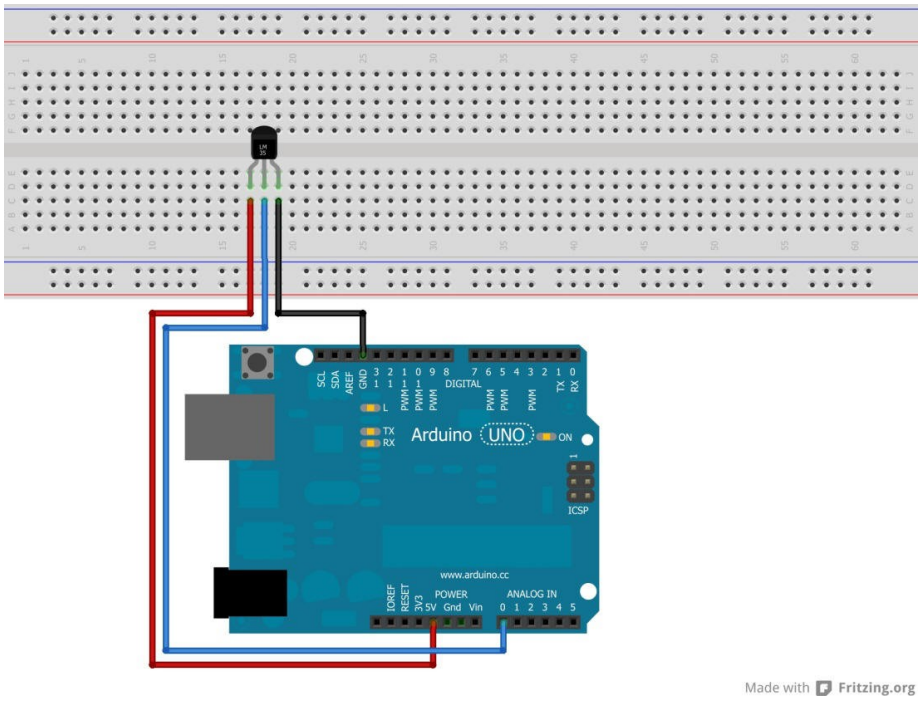
Task: Read out the temperature with theTMP36 sensor and display the temperature on the serial monitor.

Required equipment: Arduino / Breadboard / cables / temperature sensor TMP36 / external power-supply

The sensor has three terminals. 5V, GND, and the pin for the temperature signal. On this pin, the sensor puts out a voltage between 0 and 2.0 volts.
0V = -50 ° C and 2.0V = 150 ° C.
The sensor is supposed to be quite precise ($\pm 2^{\circ}C$) between -40°C and 150°C, according to the manufacturing. The voltage on this pin must be read out by the microcontroller board and then has to be converted into a temperature value.

- CAUTION: If the sensor is connected incorrectly it gets destroyed.

- Use a external power supply for more sensor accuracy (as possible 9V battery or 9V power supply).



Code:

```
int TMP36 = A0;  //The sensor should be connected to the analog Pin A0. We will
//call the pin “TMP36” from now on

int temperature = 0;   //Under the variable “temperature” we will later on save
//the temperature value
int temp[10];  //To get good values we have to read out some values to get the
//mean value out of them. The square brackets “[10]” create ten different
//variables at one time:„temp[0]“, „temp[2]“, „temp[3]“, …till… „temp[9]“.So
//with this spelling [10] we will save some space.

int time= 20;  //The value after “time” sets the distance between each
//measurement.

void setup() {

Serial.begin(9600);   //In the setup we are going to start the serial
//communication, to see the temperature values on the serial monitor. The
//microcontroller will be sending values to the computer. The serial monitor can
//be started at “Tools” in the arduino software.

}

void loop() {   //main part

temp[0] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[2] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[3] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);
```

```
temp[4] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[5] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[6] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[7] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);



temp[8] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[9] = map(analogRead(TMP36), 0, 410, -50, 150);   //Until here the
//temperature gets read out ten times. In between there is always a little
//break. We will take a closer look on the used command:
//temp[1]=map(analogRead(TMP36),0,410,-50,150);
//temp[1] – The name of the first variable.
//“map(a,b,c,d,e)” - This is a “Map command”. With this command it is possible
//to change a read out value (a) from one region between (b) and (c) into a
//region between (d) and (e).
//In our case this means: The sensor value gets read out right in the map
//command “analogRead(TMP36)”. The value should be between 0 and 410 (= values
//between 0V and 2V at the analog port). The sensor outputs this voltage values
//if it measures temperature between -50°C and 150°C. With the “map command”
//these values get converted into degree values between -50°C and 150°C.

temperature=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]+temp[6]+temp[7]+tem
p[8]+temp[9])/10; //Everything in one row! In this line, all ten values get
//summarized and get divided by ten. This average value gets saved under
//“temperature”.
```

```
Serial.print(temperature);  //Now the temperature values gets send to the
//computer and can be looked up on the serial monitor.

Serial.println("degree");

}
```

Extension of the sketch:

If the temperature reaches 30°C, a noise from the piezo speaker appears.

```
int TMP36 = A0;

int temperature = 0;

int temp[10];

int time= 20;

int piezo=5;  //Piezo speaker on pin 5

void setup() {

Serial.begin(9600);

pinMode (piezo, OUTPUT);  //Pin 5 is an output

}

void loop() {

temp[0] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);
```

```
//...

temp[9] = map(analogRead(TMP36), 0, 410, -50, 150);

temperature=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]+temp[6]+temp[7]+tem
p[8]+temp[9])/10; // Everything in one row!

Serial.print(temperature);

Serial.println("degrees");

if (temperature>=30)    //If the temperature reaches 30°C or more….

{

digitalWrite(piezo,HIGH);  //...the piezo speakers beeps..

}

else

{

digitalWrite(piezo,LOW);   //...it is quiet.

}

}
```

Measurement of distance

Task: Measure a distance with the HC-SR04 ultrasonic sensor and show the result on the serial monitor.

Required equipment: microcontroller board / cables / Breadboard / HC-SR04 ultrasonic sensor

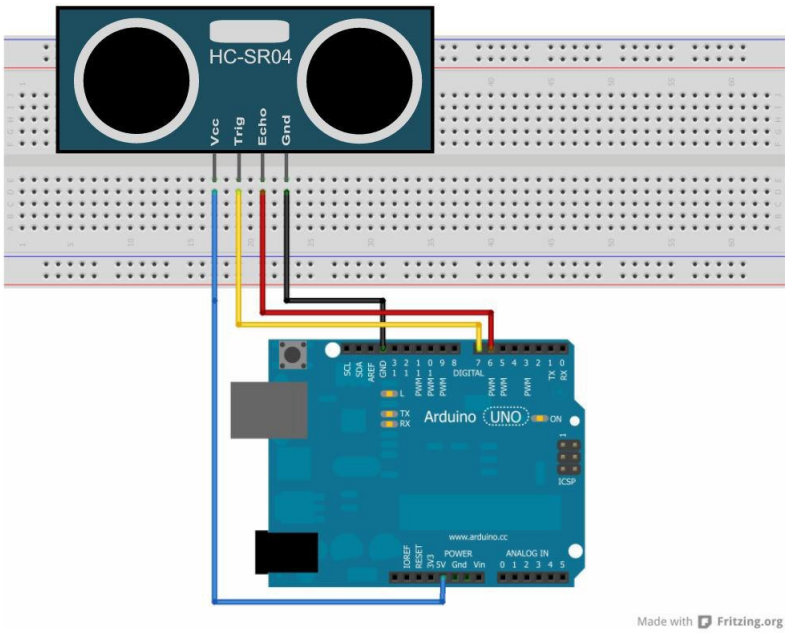
How does the ultrasonic sensor work?

The sensor has four pins.

a) 5V (+) b) GND (-) c) echo d) trigger

The contacts 5V and GND are meant for the power supply. Trough the "trigger" pin the sensor gets a short signal (5V) from the microcontroller board to create a sound wave. As soon as the sound wave hits a wall or other objects, it will be reflected and comes back to the ultrasonic sensor. When the sensor detects this returned sound wave, the sensor will send a signal to the Arduino microcontroller trough the "echo" pin. The Arduino-board measures the time between the transmission and the return of the sound wave, and converts this time into a distance.

Setup:



Code:

```
int trigger=7;  //"trigger" on pin 7.

int echo=6;   //"echo" on pin 6.

long time=0;   //The value "time" will safe the time between transmission and
//returning of the soundwave.

long dist=0;  //The value "dist" will save the calculated distance. It will
//start with "0". Instead of "int" we are using "long" for this value, to save a
//bigger number.

void setup()

{

Serial.begin (9600);   //Starting the serial communication. It will send the
//data from the arduino board to the computer to show it on the serial monitor.

pinMode(trigger, OUTPUT);    //"trigger" (Pin 7) is an output.

pinMode(echo, INPUT);  //"echo" (Pin 6) is an input.

}

void loop()

{

digitalWrite(trigger, LOW);   //Low voltage on the trigger pin to produce a
//clear signal.

delay(5);   //….for 5 milliseconds.

digitalWrite(trigger, HIGH);    //Creating the soundwave.

delay(10);   //..for 10 milliseconds.

digitalWrite(trigger, LOW);  //Stop creating the soundwave.
```

```
time = pulseIn(echo, HIGH);  //With the command pulseIn (Capital “i” in the
//front of the “n”) the arduino board measures the time between sending and
//receiving the soundwave.

dist = (time/2) / 29.1;  //This calculation transforms the measured time into
//the distance in centimeter. (The sound needs 29,1 seconds for one centimeter.
//The time gets divided with two, because we only want to get one distance and
//not the two ways that the soundwave has to take).

if (dist >= 500 || dist <= 0)   //If the distance gets over 500cm OR under 0cm,
//the measurement is no longer accurate.

{

Serial.println("No measurement");  //So the serial monitor displays “No
//measurement”
}

else   //otherwise
{

Serial.print(dist);  //The calculated distance is shown on the serial monitor.

Serial.println("cm");
}

delay(1000);  //This command causes a short break between the measurements.
}
```

Extension of the sketch

If the distance is less than 80cm, the piezo speaker should beep.

```
int trigger=12;

int echo=13;

long time=0;

long dist=0;

int piezo=5;     //Piezo speaker on pin 5.

void setup()

{

Serial.begin (9600);

pinMode(trigger, OUTPUT);

pinMode(echo, INPUT);

pinMode(piezo, OUTPUT);  //The pin (5) connected to the piezo is a output.

}

void loop()

{

digitalWrite(trigger, LOW);

delay(5);

digitalWrite(trigger, HIGH);

delay(10);

digitalWrite(trigger, LOW);

time = pulseIn(echo, HIGH);

dist = (time/2) / 29.1;
```

```
if (dist >= 500 || dist <= 0)

{

Serial.println("No measurement");
}

else
{


Serial.print(dist);

Serial.println("cm");

}

if (dist <= 80)    //If the measured distance gets 80 or shorter...

{

digitalWrite(piezo,HIGH);  //..the piezo should beep

}

else     //If not ...

{

digitalWrite(piezo,LOW);   //..the speaker should be quiet.

}

delay(1000);

}
```

Reverse warning system

With this code we are able to create a reverse warning system. Additional to the ultrasonic sensor we are going to connect a LED with pin 12.

Are you able to construct a reverse warning system without any images?

```
int trigger=7;

int echo=6;

long time=0;

int LED=12;

long dist=0;

void setup()

{

Serial.begin (9600);

pinMode(trigger, OUTPUT);

pinMode(echo, INPUT);


pinMode(12, OUTPUT);

}

void loop()

{

digitalWrite(trigger, LOW);

delay(5);
```

```
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
time = pulseIn(echo, HIGH);
dist = (time/2) / 29.1;
if (dist >= 500 || dist <= 0)
{
Serial.println("No measurement");
}
else
{
Serial.print(dist);
Serial.println("cm");
}
if (dist <= 40)
{
digitalWrite(LED, HIGH);
delay(dist*3);
digitalWrite(LED, LOW);
delay(dist*3);
}
```

```
}
```

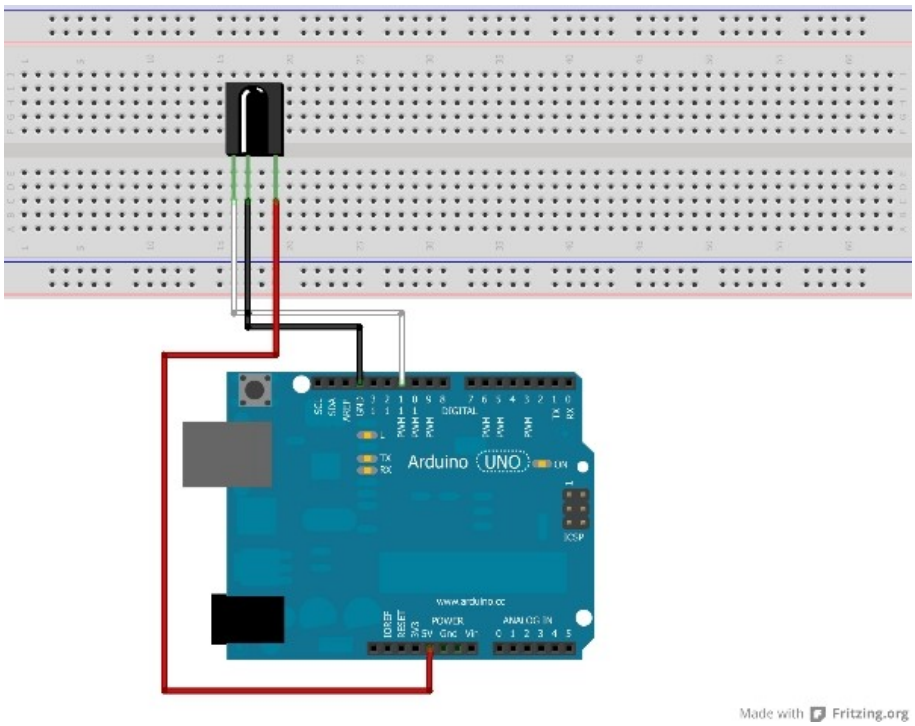
Usage of an infrared remote

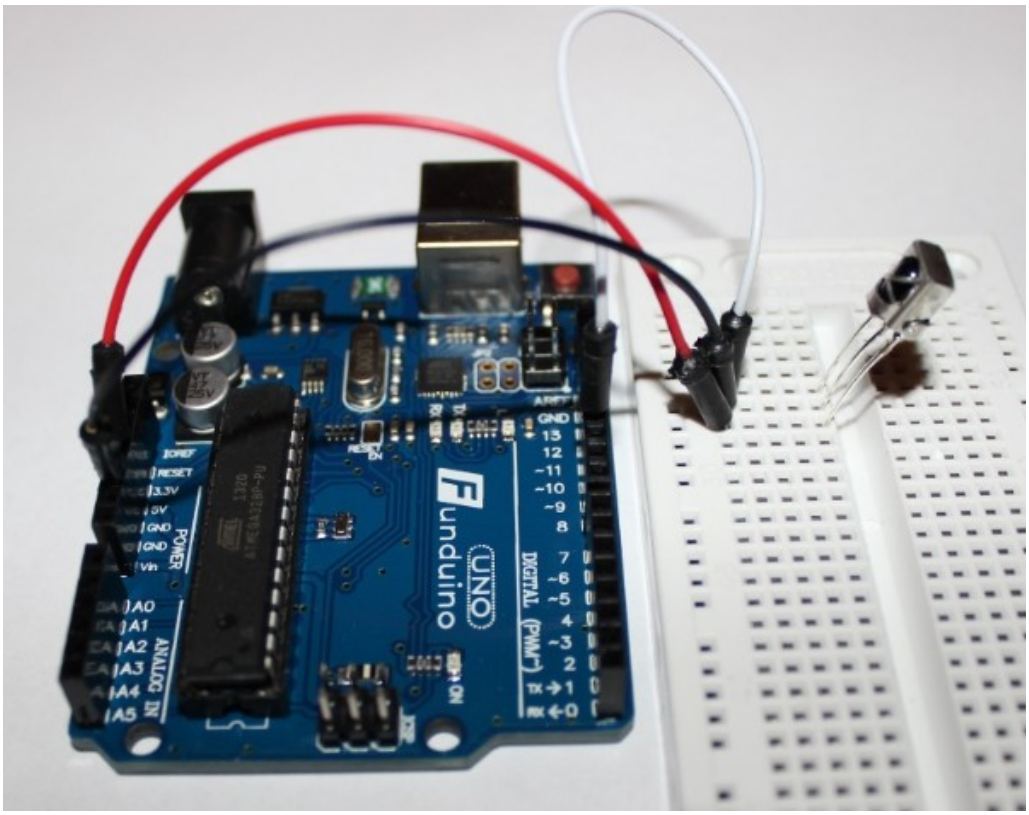
Task: Read out the signal from an infrared remote with an infrared sensor.

Required equipment: Arduino / breadboard / cable / infrared sensor / infrared remote control

With an infrared receiver, the Arduino board can receive the commands of an infrared remote control. The data are sent with infrared light from remote control to the receiver. Since our eyes can not perceive this light, we can not see this light. With a little trick you can see the light. Take your mobile-phone and look with the camera on the infrared diode of the remote while pressing a button on the remote. You will see the flashing infrared diode on the display of the mobile phone.

Setup:





The sketch is a variation of the sketch “IRrecvDemo”, and can be downloaded together with the very important IR-Remote-Library on the following website:
https://github.com/shirriff/Arduino-IRremote

Download the zip-package and copy the files into your “libraries” directory in the arduino software. Rename the downloaded directory to "Irremote".
Now you can open the sketch in the sample-files in the arduino-software:
File -> Examples -> IRremote -> IrrecvDemo

Code:

```
/*
* IRremote: IRrecvDemo - demonstrates
receiving IR codes with IRrecv
* An IR detector/demodulator must be
connected to the input RECV_PIN.
* Version 0.1 July, 2009
* Copyright 2009 Ken Shirriff
```

```
* http://arcfn.com
*/    //Information about the original program “IrrecvDemo”.

#include <Irremote.h>   //The program uses a library in this sketch. This saves us a lot of work, with reading out the code of the infrared light.

int RECV_PIN = 11;  //The contact which outputs the data gets connected with pin 11.



IRrecv irrecv(RECV_PIN);   //Here we are defining an object, that is supposed to read out the infrared sensor on pin 11.

decode_results results;  //This command defines that the read out infrared data gets saved  under “results”.

void setup()
{

Serial.begin(9600);    //In the setup we are starting the serial connection, to see the data from the remote on the serial monitor.

pinMode (13, OUTPUT);

irrecv.enableIRIn();   //This command initializes the infrared sensor.

}

void loop()

{   //The loop part is quite short because of the used library.

if (irrecv.decode(&results)) {    //If data is received...

Serial.println(results.value, DEC); //..they should show up on the serial monitor as decimal number (DEC).

irrecv.resume();  //Receive the next value.
```

```
}

}
```

Pressing the "1" button on the infrared remote control causes (in this case) the serial-monitor to write the number "16724175". This is the decrypted number code behind this button.

If you hold the button permanently pressed, the number "4294967295" appears. This is the code that indicates that a button is pressed continuously. This number does not depend on which button is pressed.

There can also appear other numbers if a key is pressed only very short or pulsating. In that case the sensor may not read a unique value.

Extension of the sketch: Switch on a LED by pressing button1 and switch it off with button2.

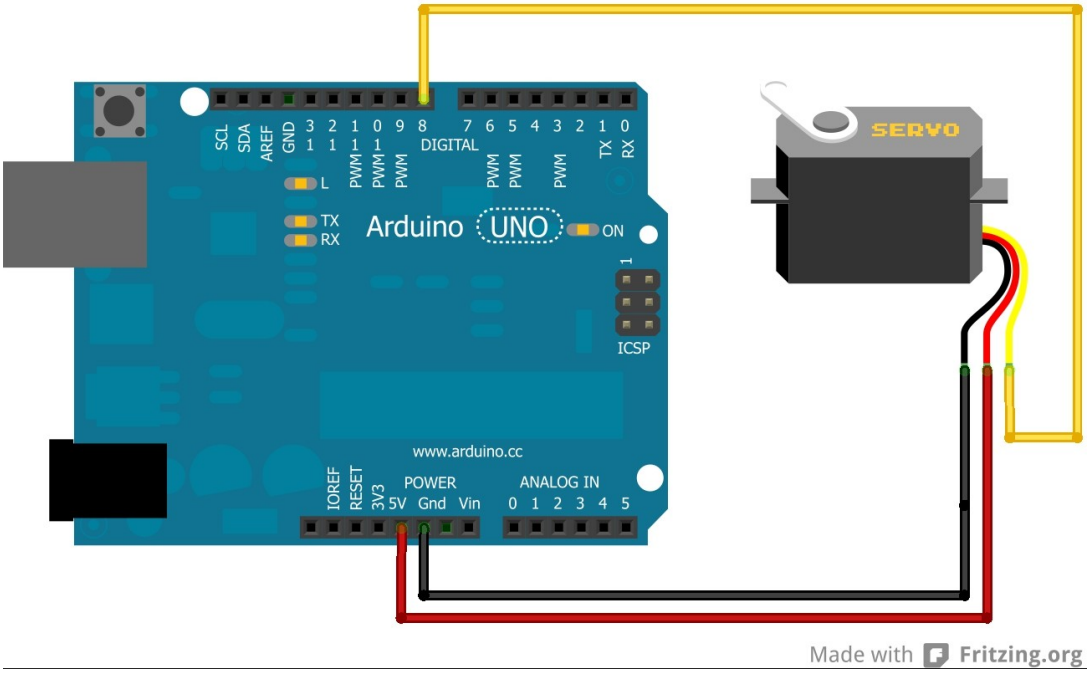
```
#include <Irremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);


decode_results results;

void setup()

{

Serial.begin(9600);

pinMode (13, OUTPUT);   //Pin 13 gets connected with a LED (Output).
```

```
digitalWrite(13, LOW);    //At first the LED should be tuned off.

irrecv.enableIRIn();

}

void loop() {

if (irrecv.decode(&results)) {

Serial.println(results.value, DEC);

if (results.value == 16724175)   //If The IR receiver receives the number
//16724175 (button 1)...

{digitalWrite (13, HIGH);}   //...the LED gets turned on.

if (results.value == 16718055)   //If the IR receiver receives the number
//16718055 (button 2)...

{digitalWrite (13, LOW);}  //….the LED gets turned off.

irrecv.resume(); // Receive the next value

}

}
```

Control a servo

Task: A servo has to turn to three different positions. Between the movements there should be a short break.

Required equipment: Arduino / one servo / three cables



Code:

```
#include <Servo.h>   //Include the servo library

Servo servoblue;   //The servo gets the name “servoblue”

void setup()
{

servoblue.attach(8);   //The signal line of the servo is on pin 8
}

void loop()
{
```

```
servoblue.write(0);    //Position 1 with an angle of 0°

delay(3000);    //Wait 3 seconds

servoblue.write(90);   //Position 2 with an angle of 90°

delay(3000);   //Wait 3 seconds

servoblue.write(180);   //Position 3 with an angle of 180°

delay(3000);   //Wait 3 seconds

servoblue.write(20);  //Position 4 with an angle of 20°

delay(3000);   //Wait 3 seconds
}
```

LCD Display

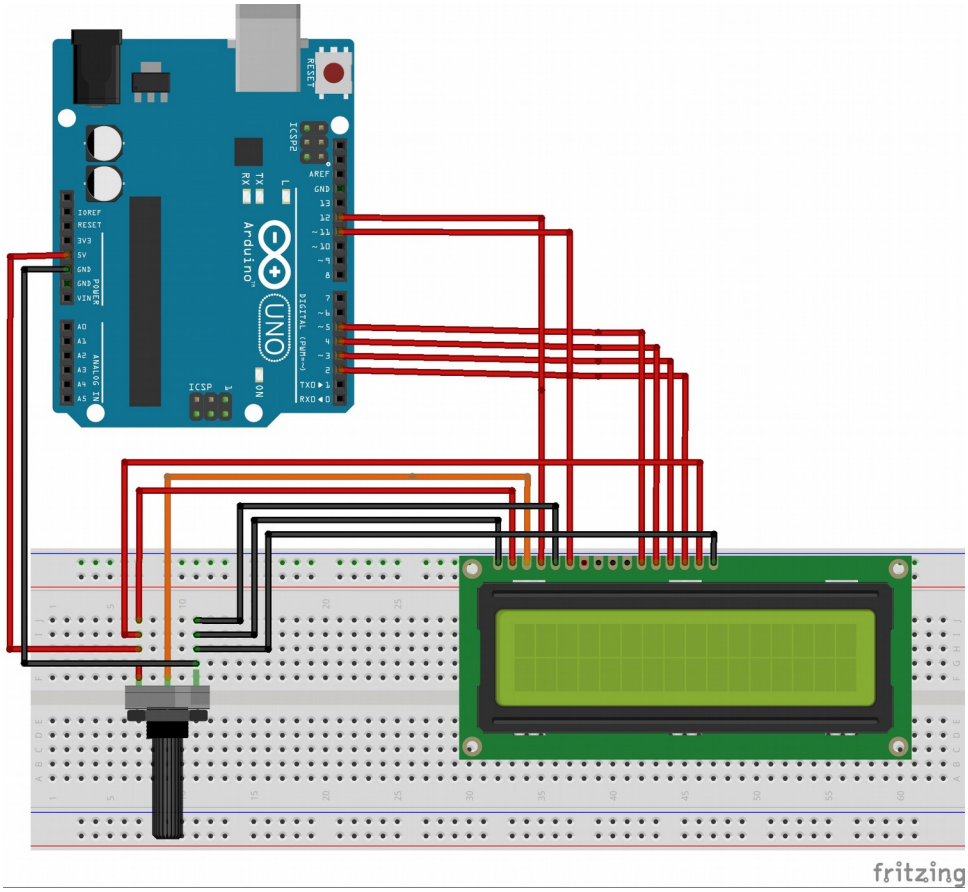
Task: Show a text on an LCD Display.

Required equipment: Arduino / potentiometer / 14 cables / breadboard

As you may see on the image, the wiring for this project is not that easy. It is because the LCD has to be connected with pretty many cables. Another difficulty is that there is no plug strip already on the LCD. You can either solder a plug strip to the LCD or solder the cable directly to the contacts of the LCD. If you want to solder the cables directly with the LCD we recommend to use flat cables (e.g. from an old hard drive or an old CD drive). Without any soldering it is quite impossible to get good results.

The potentiometer is needed to adjust the contrast. The back light of the LCD is powered with 5V. Because it is difficult to show the lettering on the LCD, in the image underneath you should try to count the contacts of the LCD. (Example: The first contact from the right to the left gets connected to GND. The second contact from the right to the left gets connected to 5V ….). Information: It is easier to use an LCD Keypad shield or an I2C LCD for more complicated projects because you don't have to take so much time for the wiring of the LCD. But these two other options are more expensive than the simple LCD module.

Setup:



Now if the LCD is successfully connected we are ready to start with the programming. Like many other components the LCD also needs to revert to a library in the code. This library is already a component of the arduino software. So you don't have to install something on your own.

Code:

```
#include <LiquidCrystal.h>   //Load the LCD library

LiquidCrystal lcd(12, 11, 6, 5, 4, 3);  //In this line we define which pins on
//the microcontroller get connected to the LCD (Better don't change yet).

void setup() {

lcd.begin(16, 2);   //In the setup we indicate how many signs and how many rows
//we are using. In this case: 16 signs in two rows.

}

void loop() {


lcd.setCursor(0, 0);   //Start position of the cursor on the LCD (0,0 = first
//character in the first row).

lcd.print("www.funduino.de");  // Write the text “www.funduino.de”

lcd.setCursor(0, 1);  //Start position of the cursor on the LCD (0,0 = first
//character in the second row).

lcd.print("good luck!!!");  // Write the text “good luck!!!”.

}
```

Extension of the sketch: We want to print alternately text on the first and on the second row. For this example we will use the words “up” and “down”.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {

lcd.begin(16, 2);

}


void loop() {

lcd.setCursor(0, 0);  //Start position first sign in the first row...

lcd.print("up");   //..show the word “up”

delay (2000);   //Wait 2 seconds

lcd.clear();   //Clear the display

lcd.setCursor(5, 1);   //Start position fifth sigh in the second row...

lcd.print("down");  //...print “down”

delay (2000);   //Wait 2 seconds

lcd.clear();   //Clear the display

}
```

The LCD module is especially useful to show sensor values or other outputs from the microcontroller. You can also find help in the arduino software at the example sketches. There you can find many different examples at “LiquidCrystal”.

Relay shield

Task: Use a relay shield.

Required equipment: Relay shield / Arduino / cables

A relay is a switch, which can be activated with a low current from the Arduino board. So you can switch on and off electrical things, that need much more power than a Arduino board can provide.

The relay needs a permanent power supply trough 5V+ and – (In the picture VCC and GND). With the “IN”-pin, the switch can be activated by the Arduino board. The three terminals on the left side are meant to be connected with the electrical component that needs more power than the Arduino can provide. If the relay doesn't get any signal from the Arduino on the “IN” contact, the contacts A and B are connected. Once the relay gets a signal from the Arduino the contacts B and C get connected.

ATTENTION: There are two types of relay shields. They differ in the way you get them to switch. Either you have to connect the “IN” pin to 5V+ on the Arduino, or you have to connect the “IN” pin to GND on the Arduino. You can easily find out which type of relay you got. Try connecting the “signal” pin with 5V+ or with GND. The version where the relay switches (you can hear a loud crack) will be the right version.

For testing purpose, you can use the “blink”-sketch. Instead of the LED, you connect the output-pin from the Arduino board with the “IN” pin from the relay. With that sketch, the relays will switch on and off in a 1 second rhythm.

Code:

```
void setup()
{

pinMode(6, OUTPUT);
}

void loop()
{

digitalWrite(6, HIGH);   //At this point the relay turns on

delay(1000);  //..wait one second

digitalWrite(6, LOW);   //turn off the relay again

delay(1000);   //...wait a second
}
```

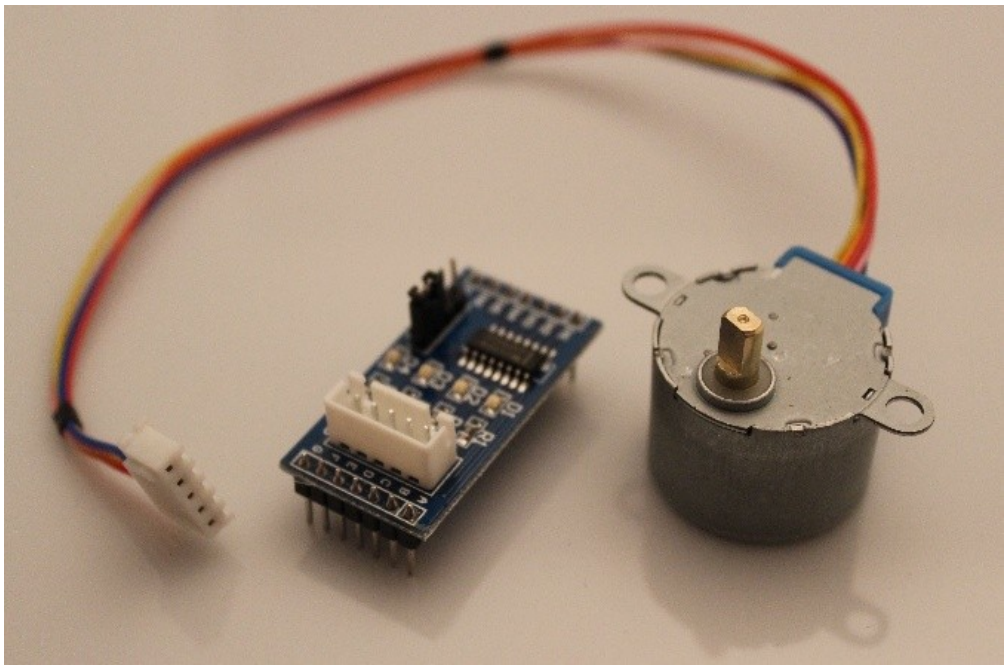
Stepper

Task: A stepper should turn around back and forth.

Required equipment: Arduino / stepper with control board / 6 cables

This stepper is especially useful for small projects with the arduino board. The stepper can operate without any external power supply. It can get a quite high torque. This is possible because of a gear, which is installed inside of the metal case in front of the actual step motor. One full rotation of the drive shaft can be divided in 2048 separate steps. One little disadvantage of this can be the slow maximum rotation speed.

The stepper gets connected with a control board. This control board supplies the stepper with enough power, to prevent the digital pins of having to do this. There are two versions of the control board. One with the outer pins on the top and one with the outer pins on the bottom. The connection is the same on both versions.



Wiring

PIN2 on Arduino board to IN4 on control board !!! 2 with 4 !!!

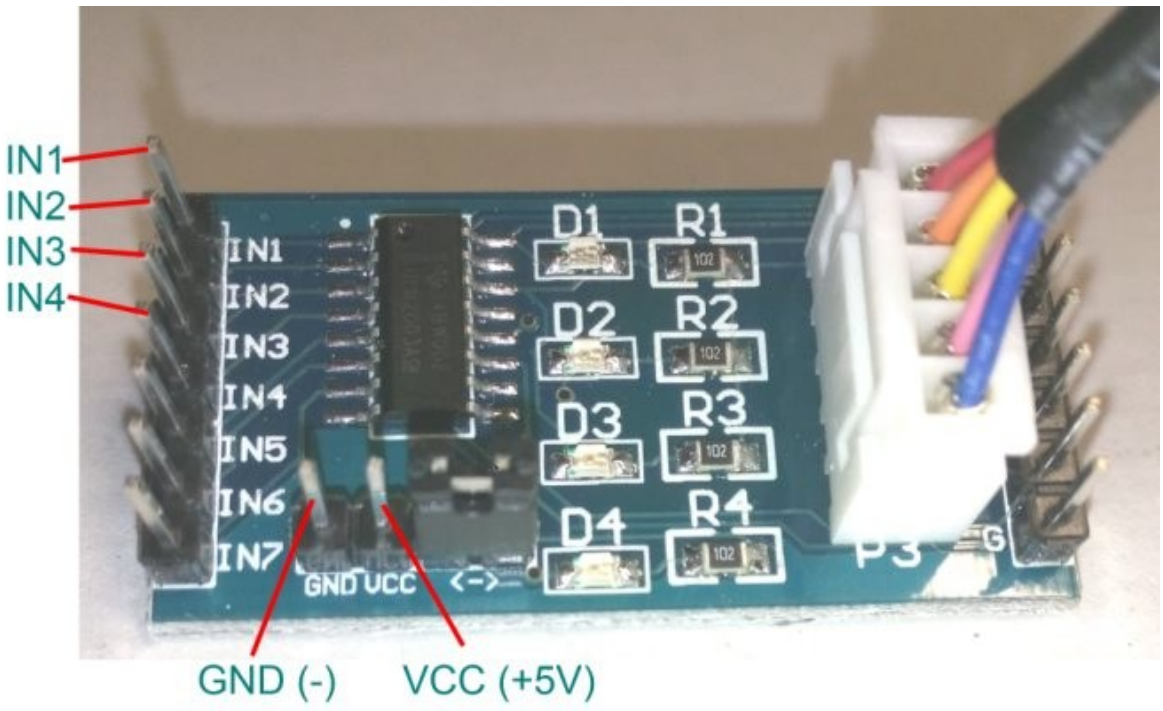
PIN3 on Arduino board to IN2 on control board !!! 3 with 2 !!!

PIN4 on Arduino board to IN3 on control board !!! 4 with 3 !!!

PIN5 on Arduino board to IN1 on control board !!! 5 with 1 !!!

PIN “GND” on Arduino board to GND on control board

PIN “5V” on Arduino board to VCC on control board



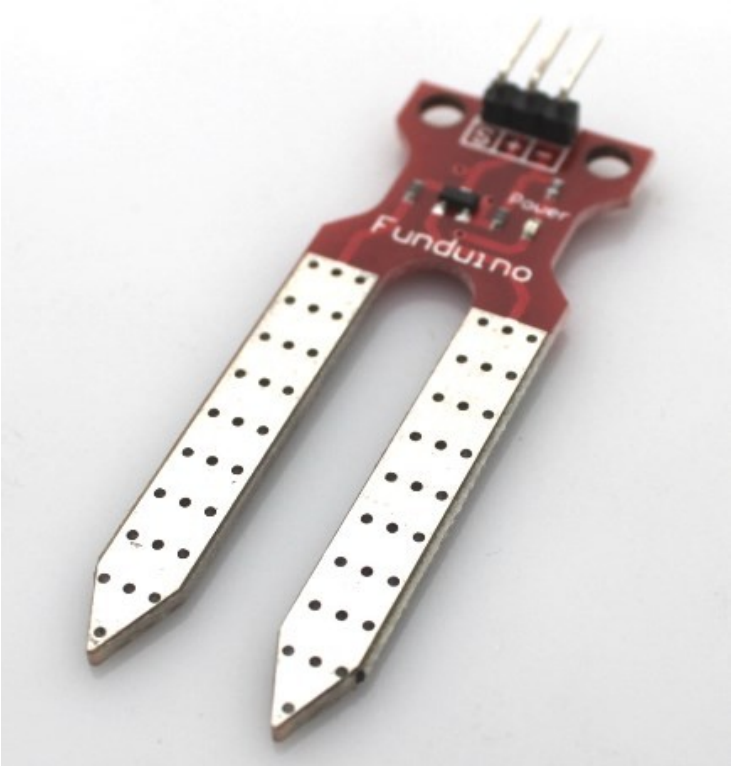
The following Code is an example and lets the stepper do one rotation (2048 steps) back and forth.

```
#include <Stepper.h>   //Load stepper library (already including the arduino software)
int SPMU = 32;
Stepper myStepper(SPMU, 2,3,4,5);
void setup()
{
myStepper.setSpeed(500);
}
void loop() {
myStepper.step(2048);
delay(500);
myStepper.step(-2048);
delay(500);
}
```

Moisture sensor

Task: We want to measure moisture and display the read out value on the serial monitor.

Required equipment: Arduino / moisture sensor / cables

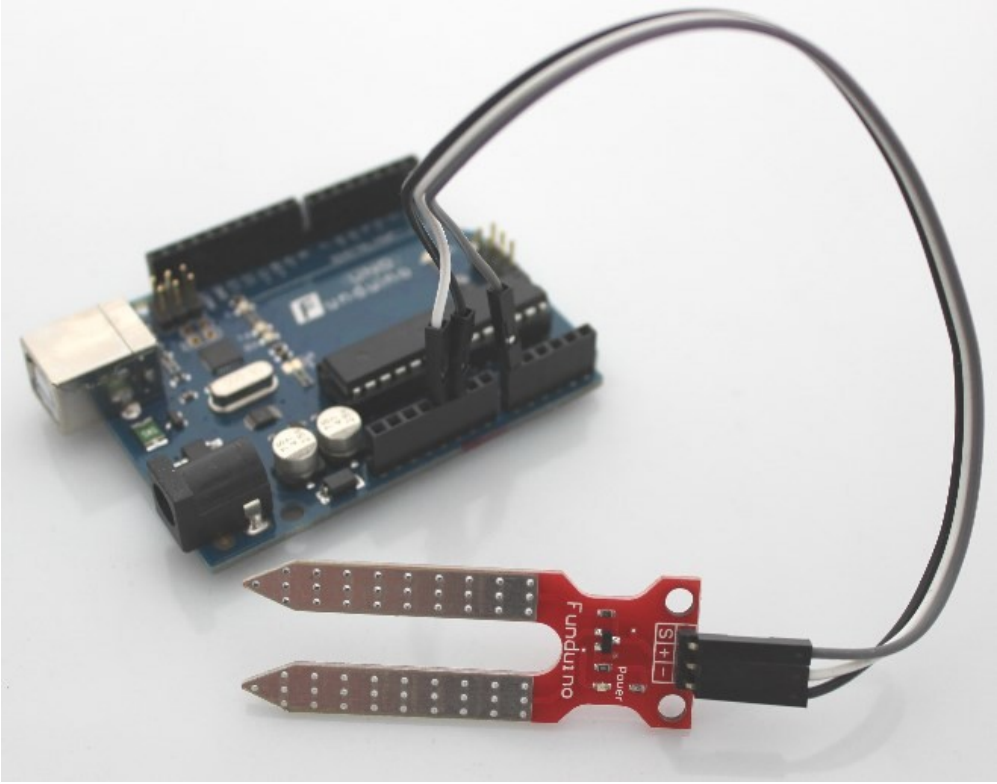


As the name suggests, the moisture sensor is able to measure moisture. That means that it can measure the directly adjacent moisture, like Skin moisture or ground humidity, but not the air moisture. One example of use can be the moisture sensor used to measure the ground humidity of a plant. If the ground of the plant gets to dry, an alarm could set off or an automatic water pump system could water the plant. The moisture sensor is also suitable for measuring a water level, in the region of the sensor.
The way of functioning is quite simple. There is a voltage on the two contacts of the moisture sensor. The higher the level of moisture gets between the contacts, the better the currant can flow from one contact to the other. This value gets electronically processed from the moisture sensor and gets transmitted as an analog signal to an analog input of the board. Since the board, as described in previous tutorials, isn't able to measure electrical voltage as such, it converts the analog signal into a numerical value. 0V to 12V

corresponds to a numerical value from 0 to 1023 (That are 1024 numbers, since the zero is counted as the first numerical value).

But the upper level of the moisture sensor is around 800, if the sensor gets completely under water. The accurate calibration depends on the sensor and the type of liquid/ moisture that is measured (e.g. salt water has a better conductivity so the value would be higher).

Setup:



The programming of the moisture sensor isn't that complicated and is very similar to the programming of the potentiometer, because there is just an analog value read out.

```
int measurement=0;   //Variable for the measurement with 0 as starting value

void setup()

{     //The setup begins here
```

```
Serial.begin(9600);  //Starting the serial communication to show the sensor
//values on the serial monitor later on

}

void loop()

{   //The loop part starts here

measurement=analogRead(A0);   //The voltage on the moisture sensor gets read out
//and gets saved under “measurement”

Serial.print("Moisture measurement:");  //Show the words “Moisture measurement:”
//on serial monitor and ...

Serial.println(measurement);   //...show the read out moisture sensor value

delay(500);   //Short break to avoid to many confusing values on serial monitor
}
```

Extension of the code:

Now we want a piezo speaker to beep if the value gets under a certain limit.

Here we are going to define “200” as limit.

Code:

```
int measurement=0;

int beep=6;   //”beep” stands for the pin 6, which gets connected with the piezo

void setup()
{
Serial.begin(9600);

pinMode (6,OUTPUT);  //The pin 6 gets defined as an output
}
```

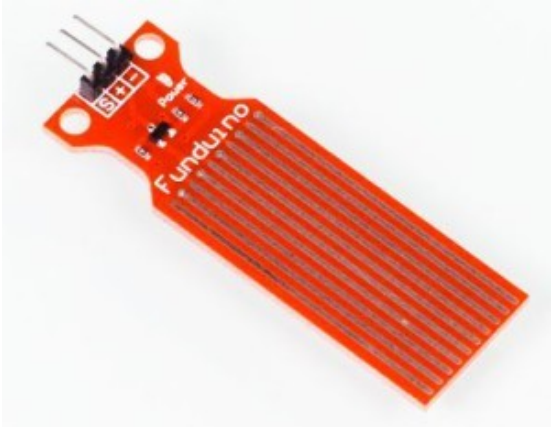
```
void loop()

{

measurement=analogRead(A0);

Serial.print("Moisture measurement:");

Serial.println(measurement);

delay(500);

if (measurement <200)   //Starting IF command: If the sensor value gets under
//“200”...

{

digitalWrite(beep, HIGH);  //...the piezo should beep...

}

else   //..if not...

{

digitalWrite(beep, LOW);  //...the piezo should be quiet

}

}
```

Drop sensor

Task: We want to detect a drop of water and display the read out value on the serial monitor.

Required equipment: Arduino / drop sensor / cables / (piezo and breadboard for the extension of the sketch)

The drop sensor, or liquid sensor, is able to detect liquid on its surface. A little drop would already be enough to get a clear measurement.



One example of the usage can be a rain detector. If the drop sensor measures a rain drop, the Arduino board could e.g. close the blinds, set off an alarm or turn on windshield wipers.

The way of functioning again is quite simple. On the long contacts that are running trough the surface of the drop sensor a voltage is applied (+ or -). As soon as a liquid e.g. in form of a drop touches the surface of the sensor, a small current flows from one contact to another. The sensor converts this value into an analog signal and transfers it to the microcontroller. The microcontroller, as mentioned in other tutorials, isn't able to read out a voltage and has to convert the analog signal into a numerical value. 0V to 12V corresponds to a numerical value from 0 to 1023 (That are 1024 numbers, since the zero is counted as the first numerical value).

If the drop sensor is completely dry the value would be “0”. Whenever a drop of water touches the contacts of the sensor the value would be at about “480”. The more drops

are touching the surface of the sensor, the higher the value would get.

Setup:

See image on page 58 (The two cables on the top (black and red) and the piezo speaker on the breadboard will be needed later on).

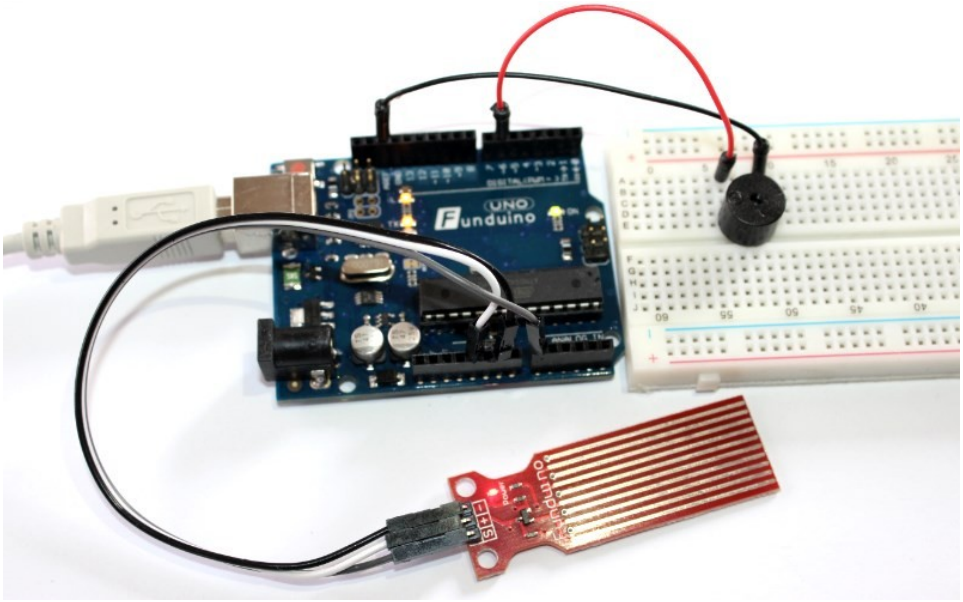
The programming of the moisture sensor isn't that complicated and is very similar to the programming of the potentiometer or the moisture sensor, because there is just an analog value read out.

```
int measurement=0;   //Variable for the measurement with 0 as starting value

void setup()

{     //The setup begins here

Serial.begin(9600);  //Starting the serial communication to show the sensor
//values on the serial monitor later on

}

void loop()

{   //The loop part starts here

measurement=analogRead(A0);   //The voltage on the drop sensor gets read out and
//gets saved under “measurement”

Serial.print("Moisture measurement:"); //Show the words “Moisture measurement:”
//on serial monitor and ...

Serial.println(measurement);   //...show the read out moisture sensor value

delay(500);  //Short break to avoid to many confusing values on serial monitor

}
```

Extension of the code:

Now we want a piezo speaker to beep as soon as a rain drop touches the sensor. As limit we are going to use the sensor value 400, because if a drop touches the sensor we are expecting a value at about 480.

Setup:



Code:

```
int measurement=0;

int beep=6;   //"beep" stands for the pin 6, which gets connected with the piezo

void setup()

{

Serial.begin(9600);

pinMode (6,OUTPUT);  //The pin 6 gets defined as an output
```

```
}

void loop()

{

measurement=analogRead(A0);

Serial.print("Moisture measurement:");

Serial.println(measurement);

delay(500);

if (measurement >400)   //Starting IF command: If the sensor value gets higher
//than “400”...

{

digitalWrite(beep, HIGH);  //...the piezo should beep...

}

else   //..if not...

{

digitalWrite(beep, LOW);  //...the piezo should be quiet

}

}
```

RFID Kit

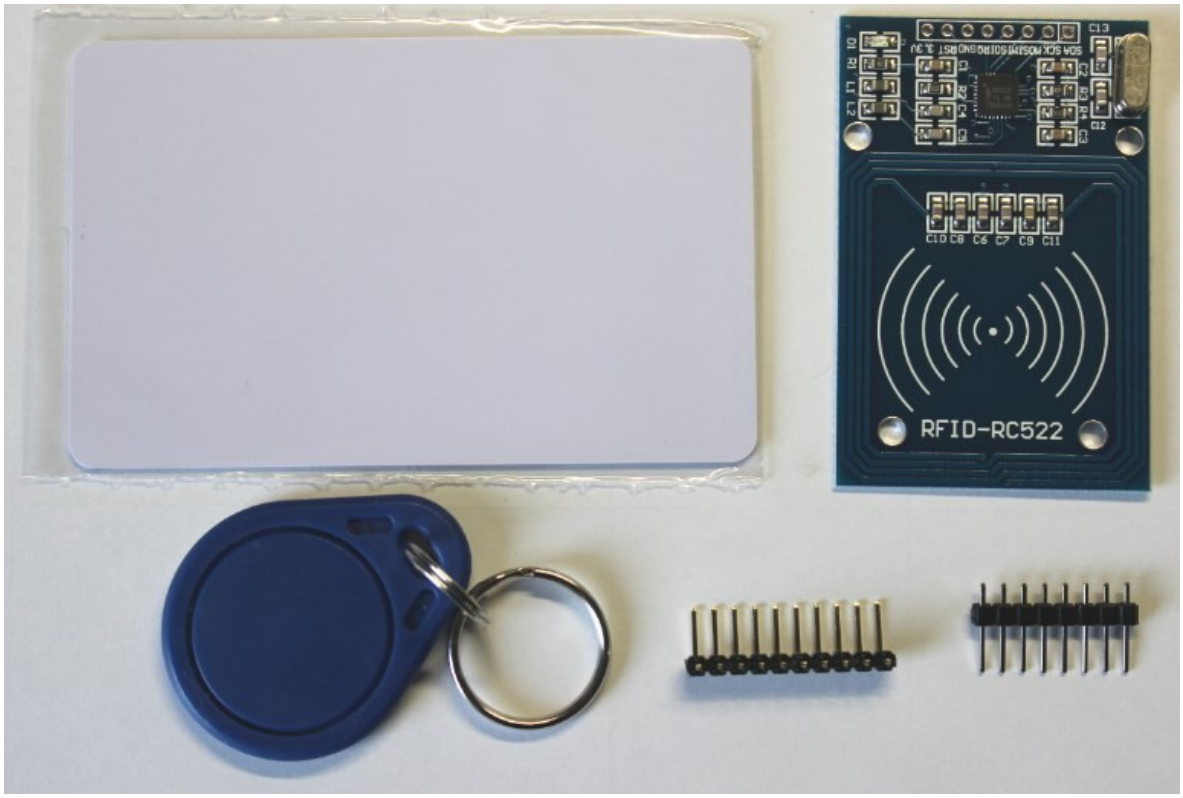
Task: Read out the UID of a RFID tag and display the UID as one contiguous decimal number on the serial monitor.

Required equipment: Arduino / RFID Kit / cables

The RFID (“radio frequency identification”) reader is used to read out a certain code, which is send from a RFID transmitter (also called “RFID tag”) by radio. Each RFID tag has only one unique code. The RFID Kit is useful to realize projects like for example a locking mechanism or other similar projects in which a person should be identified with a tag.

RFID tags may come in different shapes, like a key chain or a card in credit card format.

On the following image you can see on the left side two RFID tags, on the right the RFID receiver RFID RC522 and pin header which have to be soldered to the receiver (There are also versions with already soldered pin headers on the receiver).



How does it work? A RFID receiver contains a small copper coil that generates a magnetic field. An RFID transmitter also includes a copper coil that picks up the magnetic field and generates an electrical voltage insinde the transmitter. This voltage is used by a small electronic chip to get it to emit an electrical code by radio. The transmitter directly receives this code and processes it, so that the microcontroller is able to process the received code.

It is also possible to codify a RFID tag. Due to the complexity it is not mentioned in this tutorial. But you can find several other tutorials for this on the web.

Read out and process the data of RFID tags with the Arduino

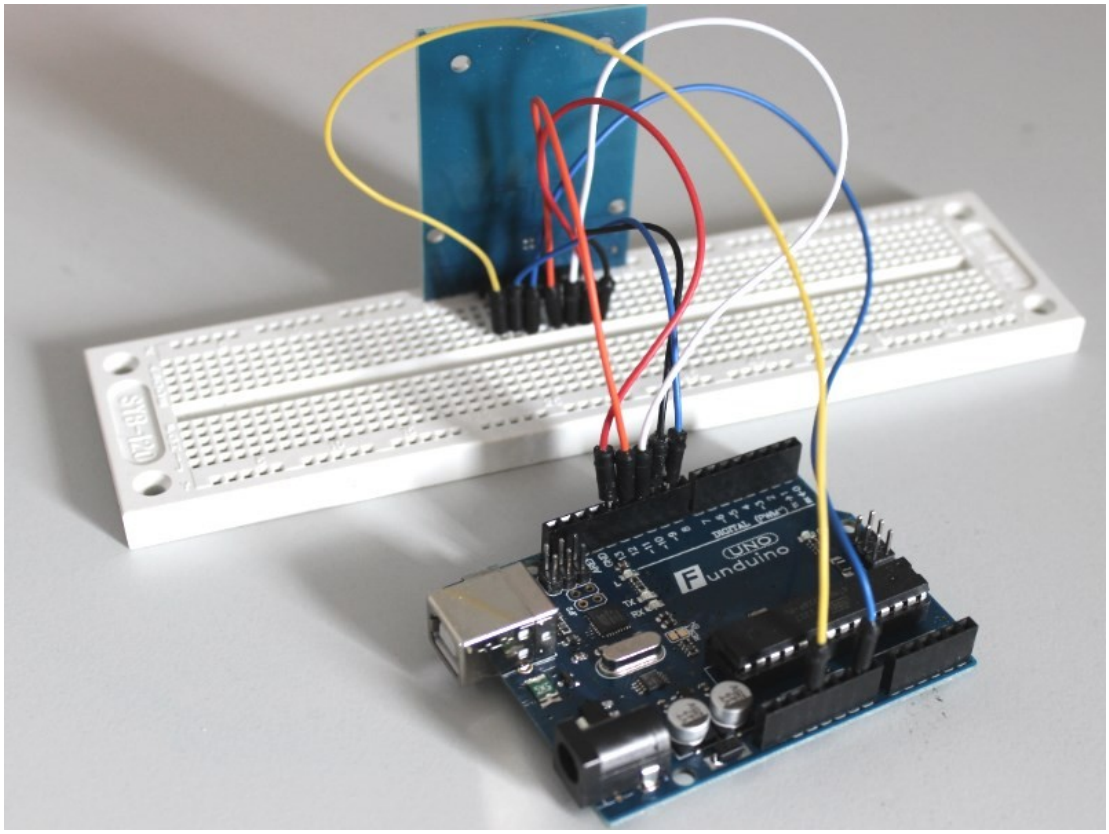
Required equipment: Arduino UNO or MEGA, RFID reader, at least one RFID tag, breadboard, cables, one LED, one 200 Ohm resistor

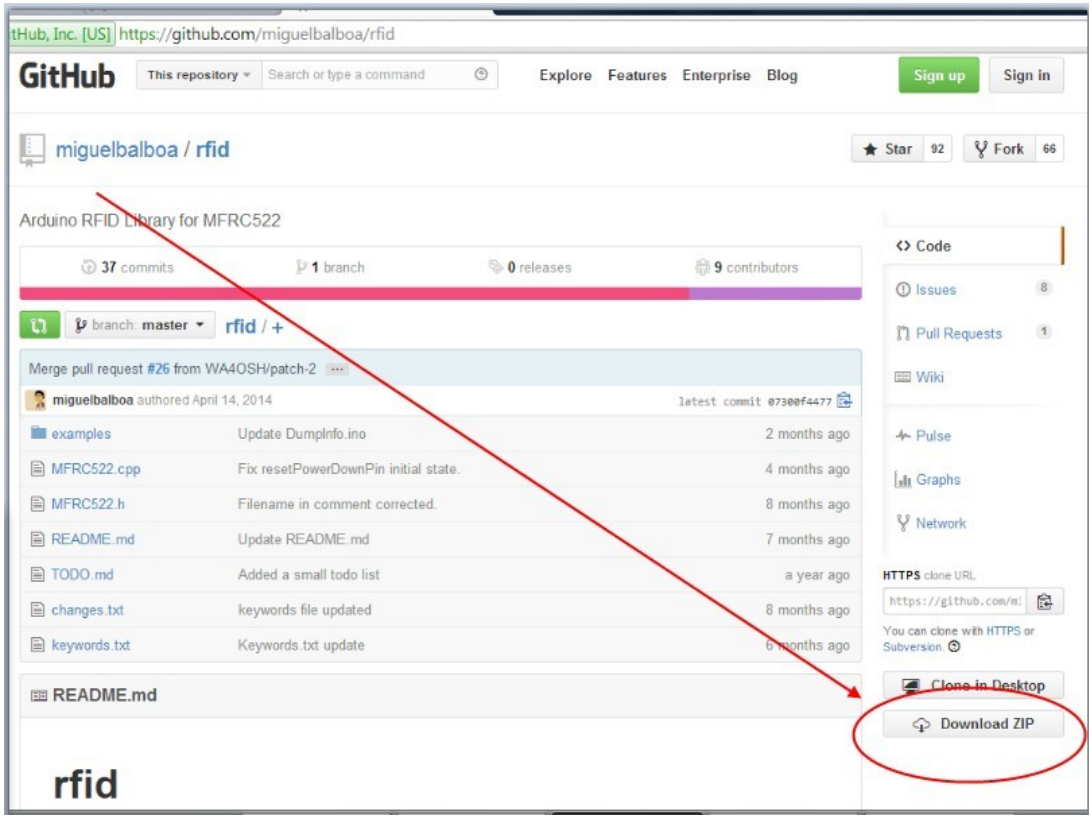
Using a Arduino microcontroller, we want to read out a RFID tag. If it is the right tag the LED should light up for 5 seconds.

Wiring of the RFID reader with the Arduino board:

Board:	Arduino Uno	Arduino Mega	MFRC522-READER
Pin:	10	53	SDA
Pin:	13	52	SCK
Pin:	11	51	MOSI
Pin:	12	50	MISO
Pin:	nothing	nothing	IRQ
Pin:	GND	GND	GND
Pin:	9	5	RST
Pin:	3,3V	3,3V	3,3V

On the image underneath the RFID reader has soldered pins bent by 90° on it (like the already soldered version). This way it is possible to plug the reader vertical on the breadboard.




https://github.com/miguelbalboa/rfid
GitHub
This repository
Search or type a command
Explore
Features
Enterprise
Blog
Sign up
Sign in
miguelbalboa / rfid
Star 92
Fork 66
Arduino RFID Library for MFRC522
37 commits
1 branch
0 releases
9 contributors
branch: master
rfid / +
Merge pull request #26 from WA4OSH/patch-2
miguelbalboa authored April 14, 2014
latest commit 0730ef4477
examples Update DumpInfo.ino 2 months ago
MFRC522.cpp Fix resetPowerDownPin initial state. 4 months ago
MFRC522.h Filename in comment corrected. 8 months ago
README.md Update README.md 7 months ago
TODO.md Added a small todo list a year ago
changes.txt keywords file updated 8 months ago
keywords.txt Keywords.txt update 6 months ago
README.md
rfid
Code
Issues 8
Pull Requests 1
Wiki
Pulse
Graphs
Network
HTTPS clone URL
You can clone with HTTPS or Subversion.
Clone in Desktop
Download ZIP

Programming

Reading out and processing the data of an RFID receiver would require,as well as other complex tasks, a lot lines of code. Therefore we are going to use one of many existing libraries from the internet. The one we used in this tutorial can be found on https://github.com/miguelbalboa/rfid.To work with the library you have to download it and save it in the Arduino program folder. You just have to click on “Download ZIP” and save the unziped data on your hard drive.

You have to unpack the folder into the Arduino software folder and save it under “libraries”. Usually, the folder has been saved under "C: Programmearduinolibraries…" (If you have saved it somewhere else you have to use the “libraries” folder there).

After the data is unzipped and saved correctly, there should appear a data with the name “rfid-master” in the “libraries” folder. Now you MUST delete the hyphen from the name. So you rename the data into “rfidmaster”. If you have followed this steps correctly, the library is ready to be used in the Arduino software.

Sketch 1

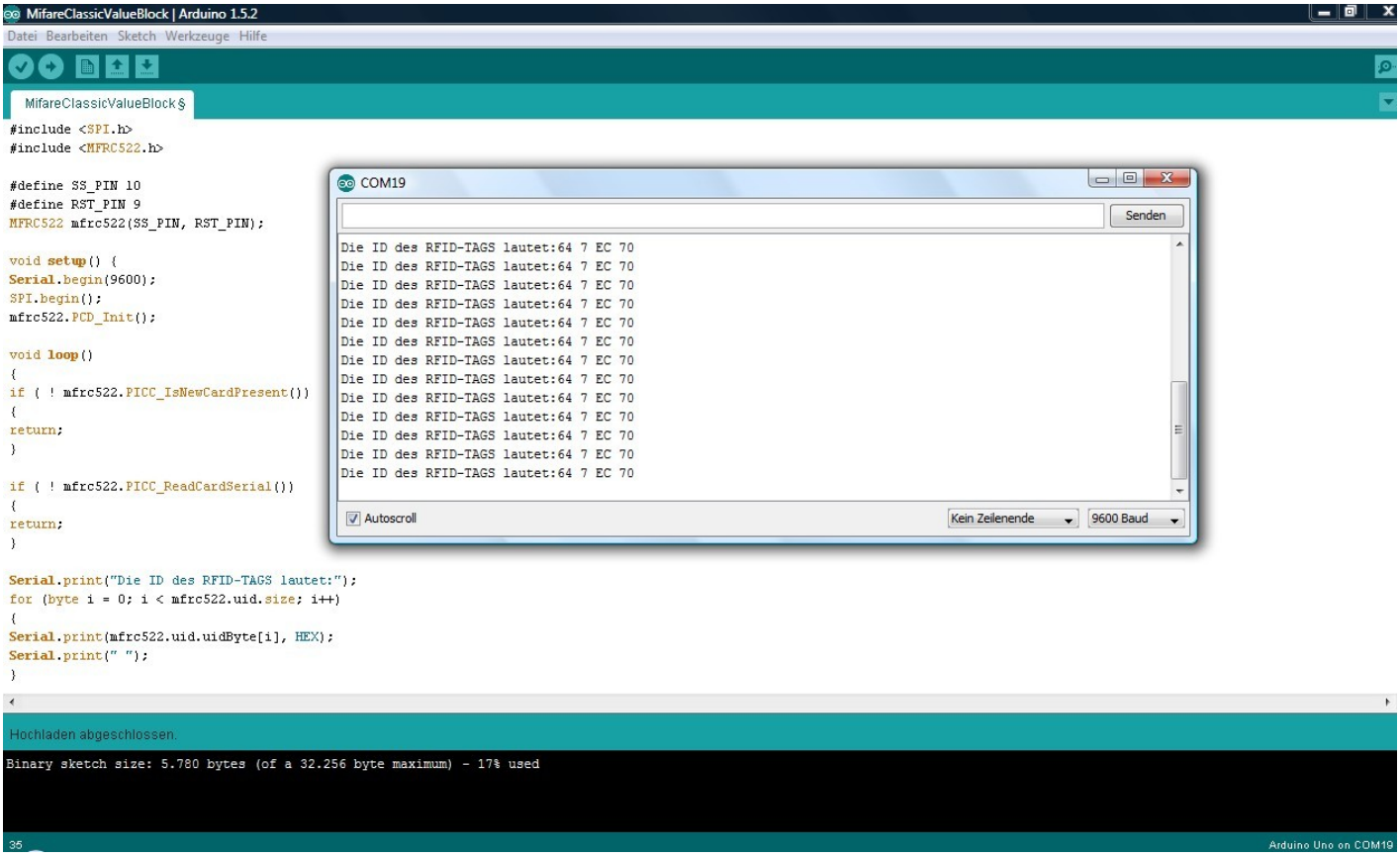
First of all we are going to read out the UID (“Unique Identification Number”). It is the individual sign of every RFID tag. We are going to use the sketch underneath (Attention, the sketch only works if the library has been added to the software the way it is explained before). This program is only destined for UNO R3 microcontroller. If you want to use MEGA2560 or other controllers, you have to adjust the pins.

```
#include <SPI.h> //Include SPI library

#include <MFRC522.h> //Include RFID library

#define SS_PIN 10 //SDA on Pin 10 (different on MEGA)

#define RST_PIN 9 //RST on Pin 9 (different on MEGA)
```

```
MFRC522 mfrc522(SS_PIN, RST_PIN); //Name RFID receiver

void setup() //Starting setup

{

Serial.begin(9600); //Starting serial connection

SPI.begin(); //set up SPI connection

mfrc522.PCD_Init(); //Initialize RFID receiver

}

void loop() //Loop part starts here

{

if (! mfrc522.PICC_IsNewCardPresent()) //If a card is near by...

{

return; //move on..

}

if (! mfrc522.PICC_ReadCardSerial()) //If a RFID tag has been chosen...

{

return; //move on...

}

Serial.print("The ID of the RFID-TAG is:"); // Show “The ID of the RFID-TAG is:"
//on the serial monitor

for (byte i = 0; i < mfrc522.uid.size; i++)

{
```

```
Serial.print(mfrc522.uid.uidByte[i], HEX); //Now the UID, which consists out of
//four separate blocks is read out and sent to the Serial Monitor one by one.
//The ending Hex means that the four blocks of the UID output as HEX number
//( including letters ).

Serial.print(" "); //The command " Serial.print ( " " ); " ensures that there is
//a space between the read blocks.

}

Serial.println(); //This line creates a line break on the serial monitor

}
```

If everything has worked it should look like this on the serial monitor (except of your own UID):



It is not very easy to work with these HEX numbers one after another. So we change the line " Serial.print (mfrc522.uid.uidByte [i] , HEX) ; " to " Serial.print (mfrc522.uid.uidByte [i] , DEC ; " . Then you will get the individual parts of the UID as a decimal number.

Sketch 2

Now the UID code is shown as a decimal number, but it is still divided into four blocks. We are going to change the code in a mathematical way to effect that the UID is shown as a single contiguous number (decimal number).

Why are we doing this? If we want to use this sketch later on, for example to let a LED light up or turn a stepper around depending on the right recognized RFID tag, it will be easier to use a IF command with one contiguous number. Example:

“If the RFID Code is 1031720, a LED should turn for 5 seconds”.

The command would be more complicated this way: “If the first block is 195 and the second block is 765 and the third block is 770 and the fourth block is 233 a LED should turn on for 5 seconds”.

A disadvantage of the decimal way is the fact that the sketch can be a little bit unsafe because not all four blocks (max. 12 numbers) can be shown as a contiguous number. If you want it to be completely safe you would have to test every single block.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

```
void setup()
{
Serial.begin(9600);
SPI.begin();
mfrc522.PCD_Init();
}

void loop()
{

if ( ! mfrc522.PICC_IsNewCardPresent())
{
return;
}

if ( ! mfrc522.PICC_ReadCardSerial())
{
return;
}

long code=0; //We are using “code” as a new variable to save the UID as
//contiguous number later on. By using “long” instead of “int” we are able to
//save a longer number.

for (byte i = 0; i < mfrc522.uid.size; i++)
{
code=((code+mfrc522.uid.uidByte[i])*10); //Now the four blocks are read out and
//the code gets “stretched” by the factor 10 every passing (Actually, we should
//use the factor 1000, but the number would become too large).
}

Serial.print("The Card number is:"); //Finally the number code (you can't say
//UID anymore)  gets displayed on the serial monitor.

Serial.println(code);
}
```

Great, we are now able to get the individual identification number (on the serial monitor) from a RFID Tag. In this case the number of the tag is 1031720.

And now? We want a LED to turn on for 5 seconds if the wanted RFID Tag gets hold in front of the RFID Reader.

Sketch 3

```
#include <SPI.h>

#include <MFRC522.h>

#define SS_PIN 10

#define RST_PIN 9

MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup()
{

Serial.begin(9600);

SPI.begin();

mfrc522.PCD_Init();

pinMode (2, OUTPUT); //Pin 2 gets defined as output (here we are going to
//connect the LED)
}

void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
{
return;
}

if ( ! mfrc522.PICC_ReadCardSerial())
{
return;
```

```
}

long code=0;

for (byte i = 0; i < mfrc522.uid.size; i++)
{
code=((code+mfrc522.uid.uidByte[i])*10);
}



Serial.print("The Card number is:");

Serial.println(code);

//Here begins the extension of the sketch

if (code==1031720) //If the number code is 1031720...

{ //Open program part

digitalWrite (2, HIGH); //...the LED on Pin 2 should light up ...

delay (5000); //..for 5 seconds..

digitalWrite (2, LOW); // …and than turn off.

} //End of the program part

} //End of the sketch
```

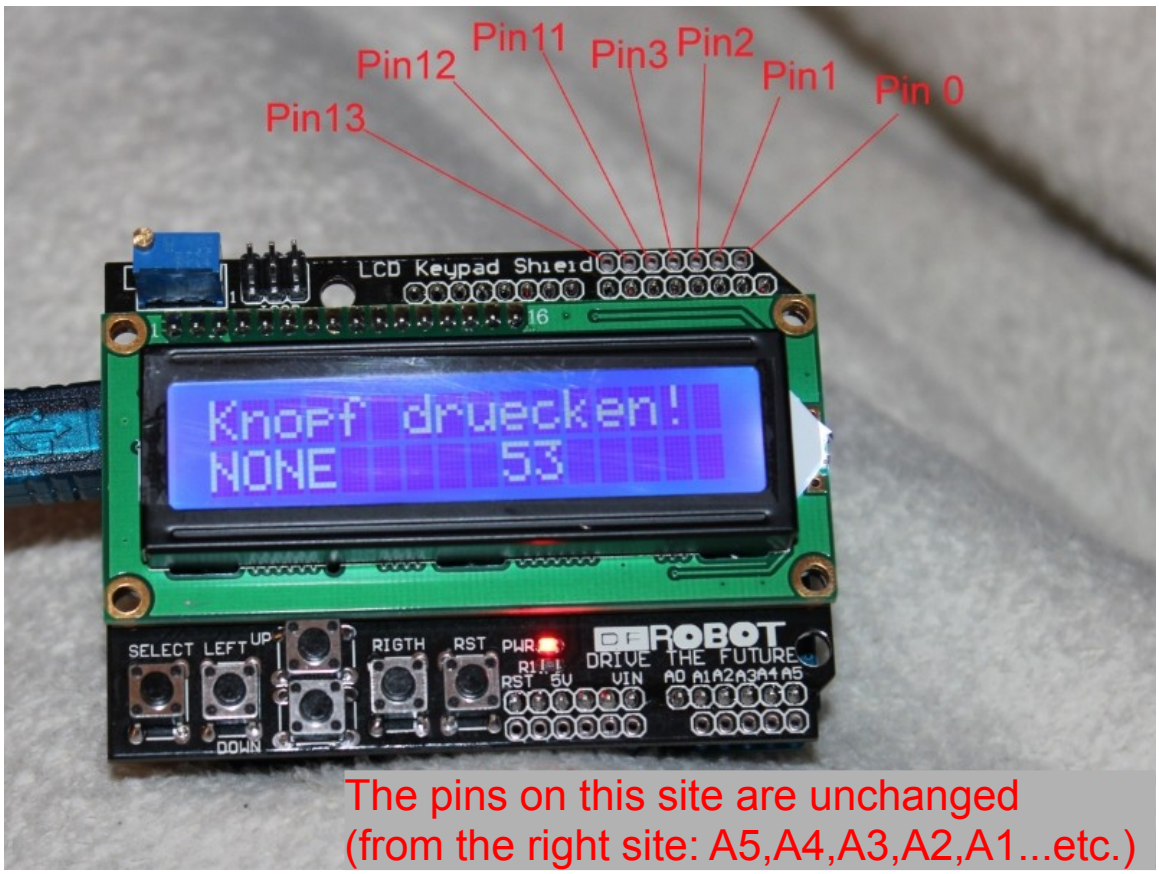
Tutorials with additional Equipment

Keypad shield

Task: Use a keypad shield with Arduino.

Required equipment: Arduino / Keypad shield

The keypad shield has some advantages over the simple LCD module. It doesn't has to be wired that complicated and has six push button, which can be used. These six buttons can be read out trough the analog pins of the microcontroller. The buttons are all connected, trough different resistors, with a analog pin (A0). That's why the analog pin A0 can only be used limited for different purposes. The shield doesn't even has the A0 pin.



The keypad shield can be plugged on the UNO board or the MEGA board for example. The power supply pins of the keypad shield should be plugged into the power supply pins of the microcontroller board (on the middle bottom you can find the power supply pins of the keypad shield. The label VIN or 5V can also help you to find them). The pins on the top of the Arduino are also used by the keypad shield (pin 0-13). Some of them are used for the LCD on the keypad shield. The other free pins are combined in a row of slots (see on the top of the image).

If you want to use these slots, we recommend to solder a connector strip onto them.

One example of a sketch can be the following:

Code:

```
//Sample using LiquidCrystal library

#include <LiquidCrystal.h>

/*********************************************************
This program will test the LCD panel and the buttons
Mark Bramwell, July 2010
*********************************************************/

// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);


// define some values used by the panel and buttons
int lcd_key = 0;
int adc_key_in = 0;
#define btnRIGHT 0
#define btnUP 1
#define btnDOWN 2
#define btnLEFT 3
#define btnSELECT 4
#define btnNONE 5
```

```
// read the buttons
int read_LCD_buttons()
{
adc_key_in = analogRead(0); // read the value from the sensor
// my buttons when read are centered at these valies: 0, 144, 329, 504, 741
// we add approx 50 to those values and check to see if we are close
if (adc_key_in > 1000) return btnNONE; // We make this the 1st option for speed
reasons since it will be the most likely result

if (adc_key_in < 50) return btnRIGHT;
if (adc_key_in < 195) return btnUP;
if (adc_key_in < 380) return btnDOWN;
if (adc_key_in < 555) return btnLEFT;
if (adc_key_in < 790) return btnSELECT;

return btnNONE; // when all others fail, return this...
}

void setup()
{
lcd.begin(16, 2); // start the library
lcd.setCursor(0,0);
lcd.print("Message"); // print a simple message
pinMode (2, OUTPUT);
}

void loop()
{
digitalWrite (2, HIGH);
lcd.setCursor(9,1); // move cursor to second line "1" and 9 spaces over
lcd.print(millis()/1000); // display seconds elapsed since power-up


lcd.setCursor(0,1); // move to the begining of the second line
lcd_key = read_LCD_buttons(); // read the buttons

switch (lcd_key) // depending on which button was pushed, we perform an action
{
case btnRIGHT:
{
lcd.print("RIGHT ");
digitalWrite (2, LOW);
```

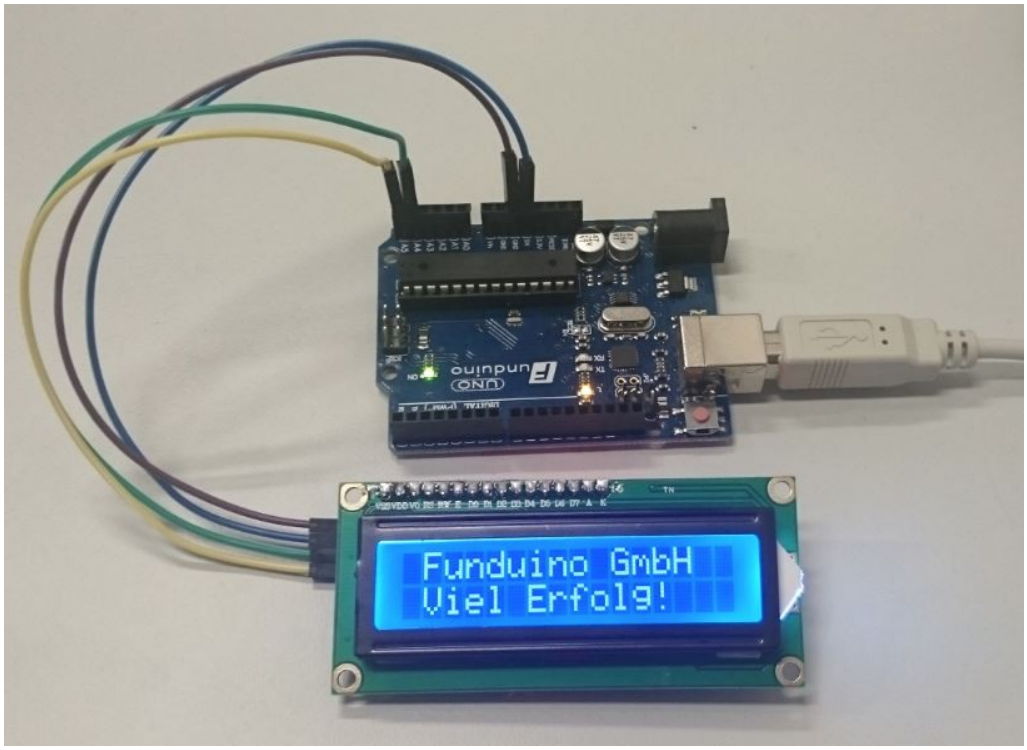
```
break;
}
case btnLEFT:
{
lcd.print("LEFT ");
break;
}
case btnUP:
{
lcd.print("UP ");
break;
}
case btnDOWN:
{
lcd.print("DOWN ");
break;
}
case btnSELECT:
{
lcd.print("SELECT");
break;
}
case btnNONE:
{
lcd.print("NONE ");
break;

}
}
}
```

I²C Display

Task: Show a text on an I²C LCD Module.

Required equipment: microcontroller (in this example UNO R3), LCD with I²C module, cables

The LCD module with the already soldered i²C module on it, makes it possible to use an LCD without the complicated wiring. This can be useful for more complex projects. Another difference to the simple LCD is, that the module on the back of the LCD, already has got a knob for the adjusting of the back light.



Wiring: The connection of the I²C LCD is very simple. The module has only four contacts. GND gets connected to GND on the Arduino, VCC to 5V on the Arduino, SDA with the analog Input A4 and SCL to the analog Input A5.

LCD >> UNO

GND >> GND

VCC >> 5V

SDA >> A4

SCL >> A5

Attention!: The MEGA2560 microcontroller has its own SDA and SCL pins. You can find them on pin 20 and pin 21.

Programing:

We will need another library to work with the I²C LCD, which isn't already installed in the arduino software. You can download the zip for example here: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library .
After you have downloaded the library you have to add it to the arduino software.

You can easily do this at the arduino software at “Sketch” then “Include Library” and “add .ZIP Library..”. Now you can use the library while writing a sketch.

Sketch:

```
#include <Wire.h> //Include Wire library
#include <LiquidCrystal_I2C.h>//Include the previous downloaded
//LiquidCrystal_I2C library

LiquidCrystal_I2C lcd(0x27, 16, 2);   //Here we are going to define what kind of
//display we are using. In this case it is one with 16 signs and 2 rows.

void setup()

{

lcd.begin(); //In the setup part we are starting the LCD (without anything
//inside the brackets, because we defined the LCD already).

}

void loop()

{

lcd.setCursor(0,0); //At the loop part the I²C LCD is programed just like the
//simple LED

lcd.print("Funduino GmbH");
```

```
lcd.setCursor(0,1);  // lcd.setCursor to define sign and row where the text
//should start

lcd.print("Good Luck!"); // lcd.print to show text on the LCD

}
```

Extension:

Just like with the simple LCD, you can show the result of measurements on the I²C LCD.

Example code (moisture sensor gets connected to pin A0):

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

int measurement=0;

void setup()

{

lcd.begin();

}

void loop()

{

measurement=analogRead(A0);  //The value on A0 gets read out and saved under
“measurement”
```

```
lcd.setCursor(0,0); //”Measurement:” is shown in the first row

lcd.print("Measurement:");

lcd.setCursor(0,1);  //In the second row we want to show the read out value of
the moisture sensor

lcd.print(measurement);

delay(500);

}
```
