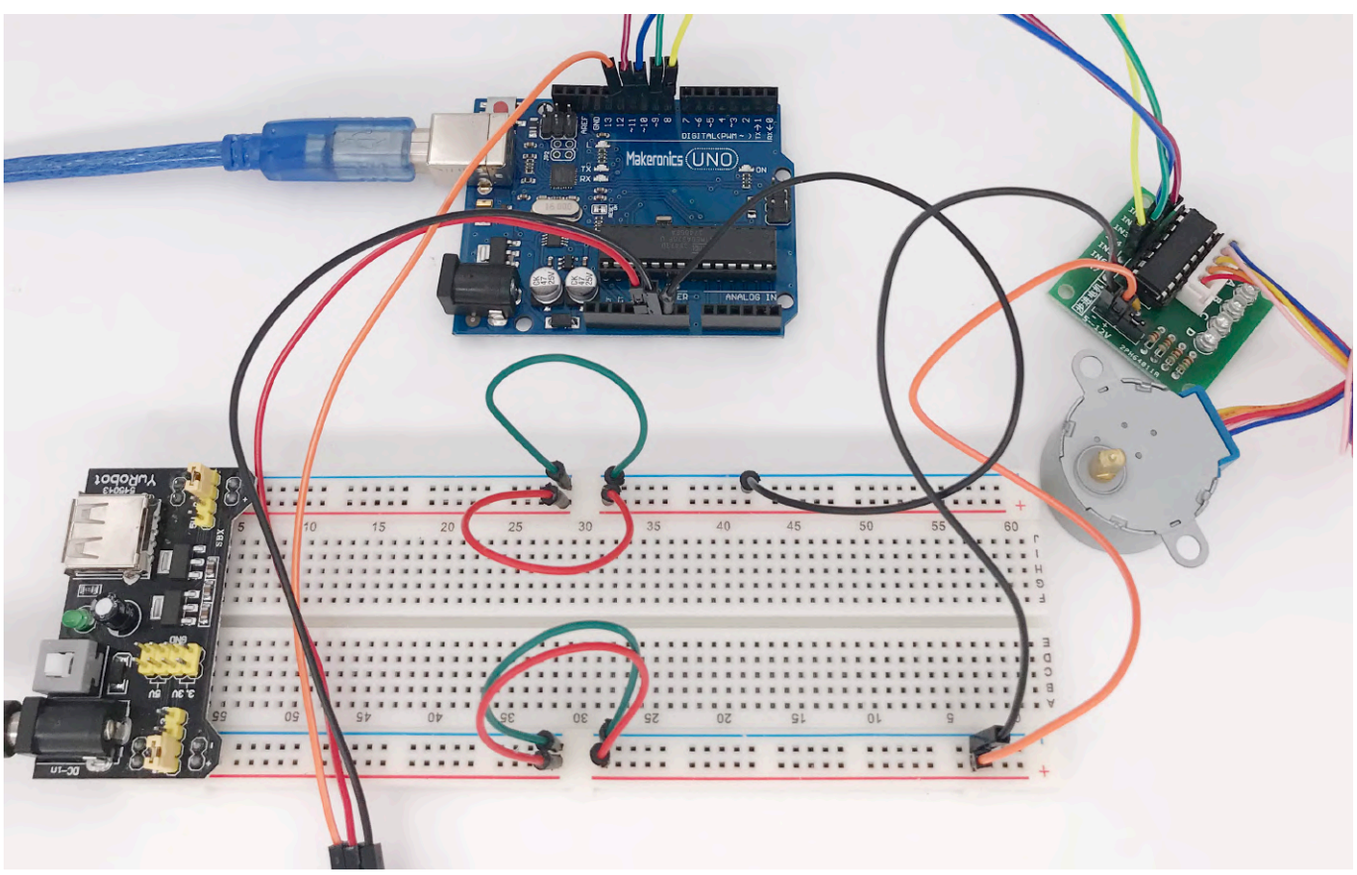
Makeronics

Super Starter Kit For Uno R3

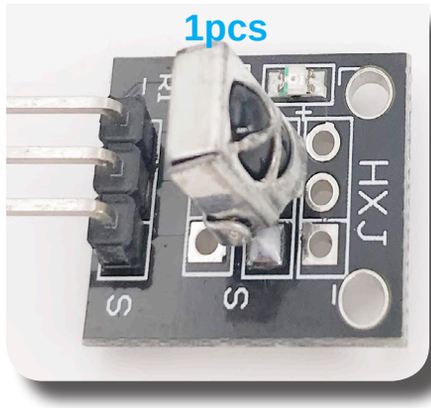


Parts List

Stepper Motor
1 pcs

Servo
1pcs

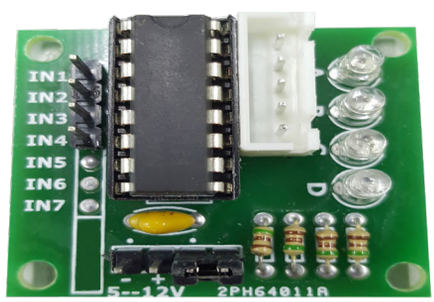
IR Receiver Module
1pcs



Power supply Module
1 pcs

Makeronics Uno R3
1pcs

Stepper Motor Driver
1 pcs



Fan Blade
1 pcs

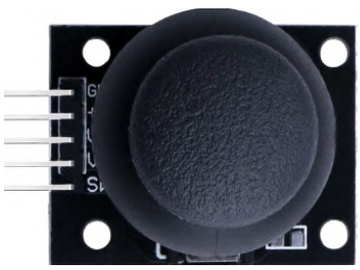
DC motor
1 pcs

Female-to-male dupont wires
10 pcs

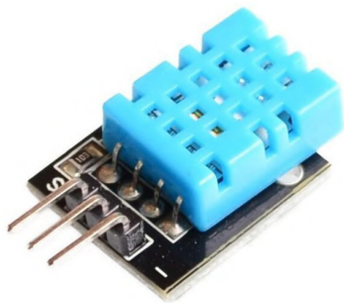
Motor bracket
1 pcs

Tilt switch
1 pcs

Joystick Module
1 pcs



DHT11 Temperature and Humidity Module
1 pcs

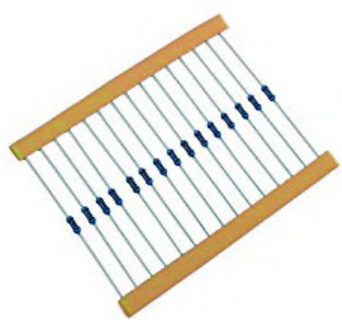


LCD 1602 Module with Pin header
1 pcs

LED
40 pcs

9V Battery with DC
1 pcs

Resistor 30 pcs

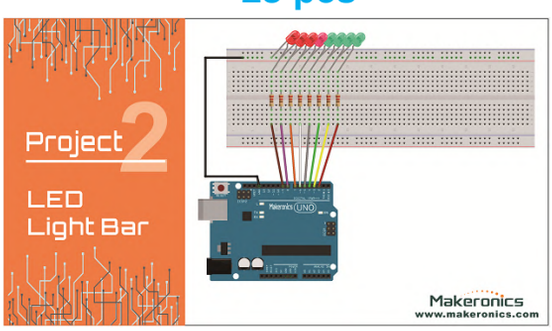


Passive Buzzer
1 pcs

Ultra sonic sensor
1 pcs

USB cable
1 pcs

Instruction Card
25 pcs

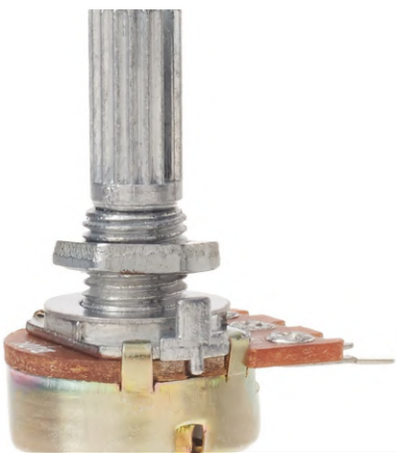


1 digit 7-segment display
1 pcs

4 digit 7-segment display 1 pcs

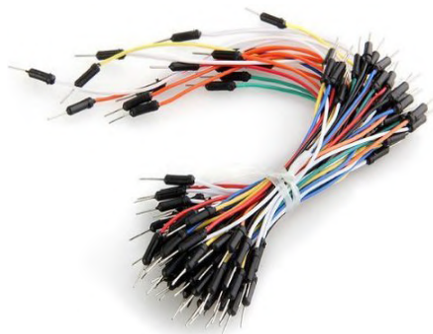
Small button 10 pcs

Potentiometer
1 pcs



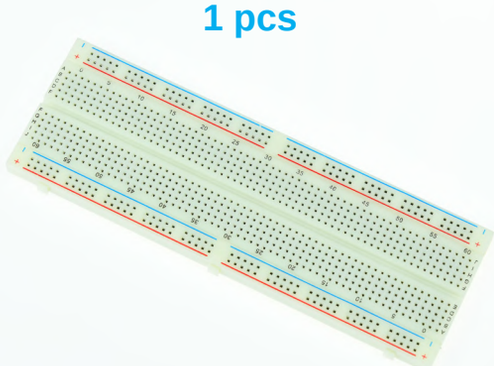
Active Buzzer
1 pcs

65 jumper wires
1 pcs



L293D
1 pcs

830 Tie-points breadboard
1 pcs



74HC595
1 pcs

RGB LED
1 pcs

Remote
1 pcs

Photoresistor
1 pcs

Thermistor
1 pcs

10 Grid Adjustable Storage Container
1pcs

Index

Lesson1 Arduino IDE 1
Lesson2 Installing Additional Arduino Libraries 6
Lesson3 Blink 13
Lesson4 External LED (Project 1) 24
Lesson5 LED Light Bar (Project 2) 36
Lesson6 RGB LED (Project 3) 41
Lesson7 Digital Inputs (Project 4) 50
Lesson8 Active Buzzer (Project 5) 55
Lesson9 Passive Buzzer (Project 6) 59
Lesson10 Piano (Project 7) 63
Lesson11 Tilt Ball Switch (Project 8) 67
Lesson12 Servo motor (Project 9) 71
Lesson13 Ultrasonic Sensor Module (Project 10) 76
Lesson14 Intruder Sensor (Project 11) 82
Lesson15 DHT11 Temperature and Humidity Sensor (Project 12) 87
Lesson16 Analog Joystick Module (Project 13) 93
Lesson17 IR Receiver Module (Project 14) 99
Lesson18 LCD Display (Project 15) 106
Lesson19 Thermometer (Project 16) 113
Lesson20 Eight LEDs with 74HC595 (Project 17) 119
Lesson21 The Serial Monitor 128
Lesson22 Photoresistor (Project 18) 135
Lesson23 Photoresistor And 74HC595 (Project 19) 141
Lesson24 74HC595 And Segment Display (Project 20) 146
Lesson25 Four Digital Seven Segment Display (Project 21) 152
Lesson26 DC Motors (Project 22) 157
Lesson27 Relay (Project 23) 169
Lesson28 Stepper Motor (Project 24) 175
Lesson29 Controlling Stepper Motor With Remote (Project 25) 184

1

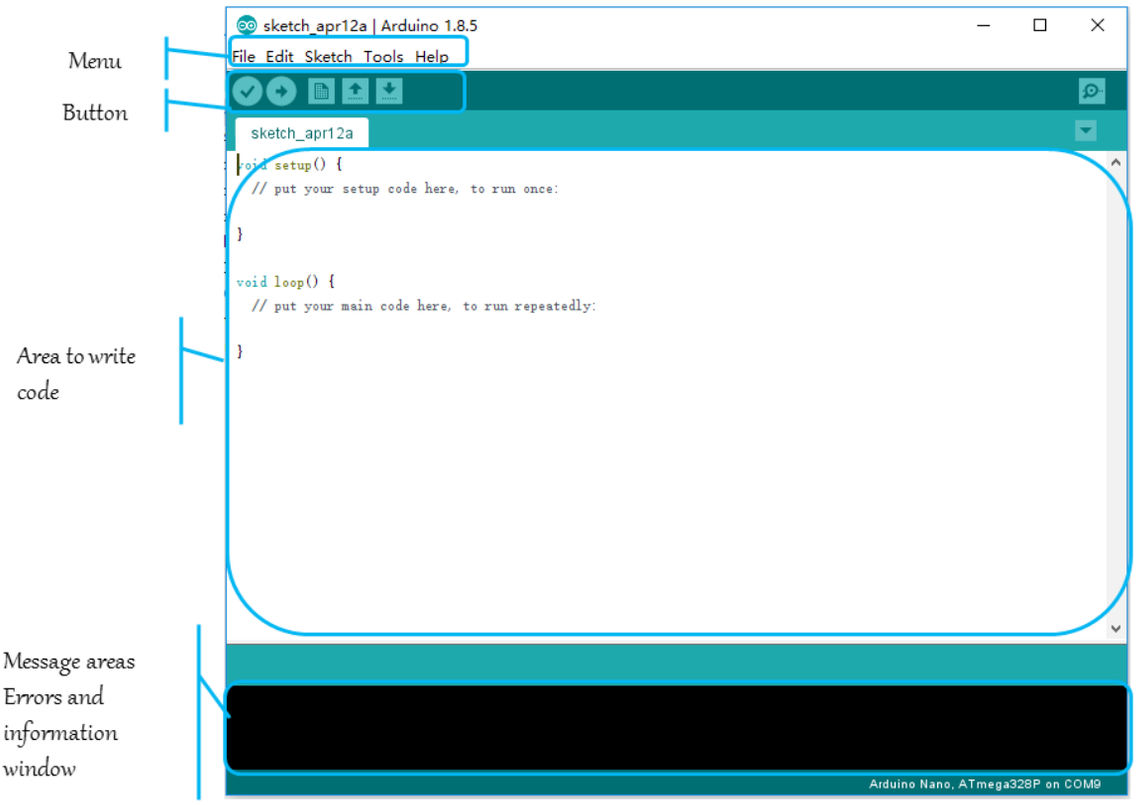
Arduino IDE

WHAT'S AN IDE?

An integrated development environment (IDE) is a software application that allows you to write code and test that code out in the programming language the IDE supports.
You can test your code in the IDE and solve any emerging problems with the help of a message area that shows errors in your code and a console that provides more detail about the nature of these errors. It has buttons so you can check your code, save it, create a new code window, upload it to your Arduino, and more.

WHAT'S IN IDE?

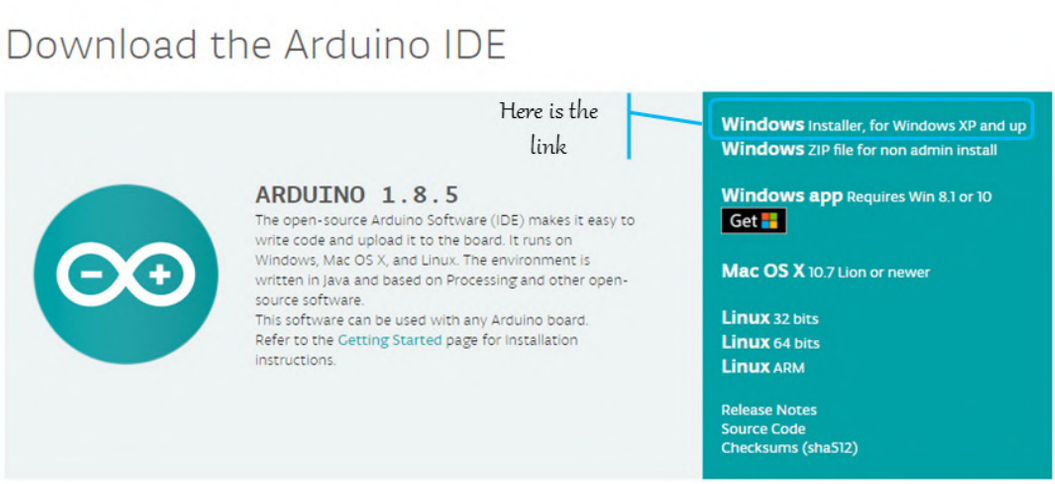
- A code editor window where you write your code
- A message area that gives information about your code
- An error console that gives detailed information and helps in debugging
- Menus that allow you to set properties for your Uno and load code examples and other functions
- Buttons to check code, upload it to Arduino, save code, create a new code window, and more



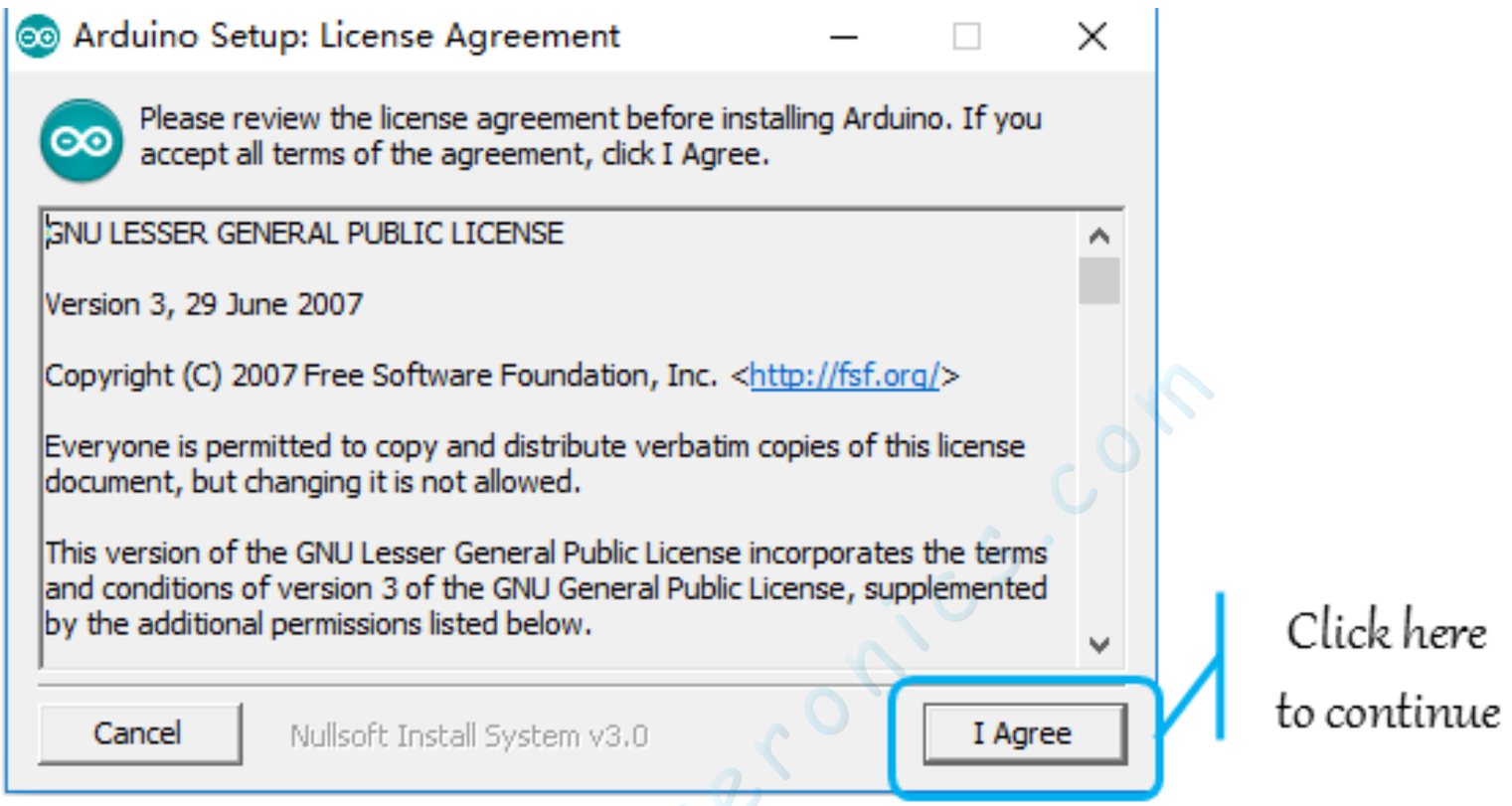
Installing Arduino IDE?

On Windows

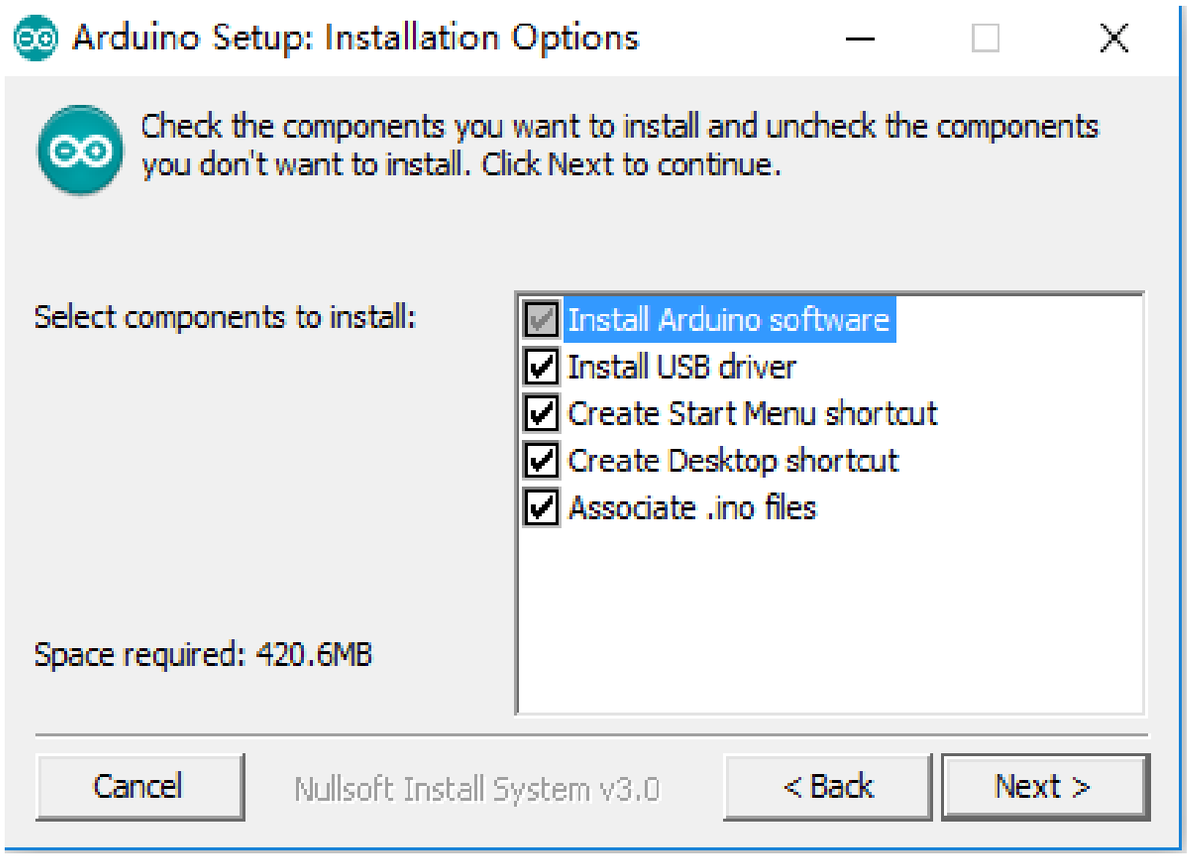
1. Go to https://www.arduino.cc/en/Main/Software
2. Download "Windows Installer"



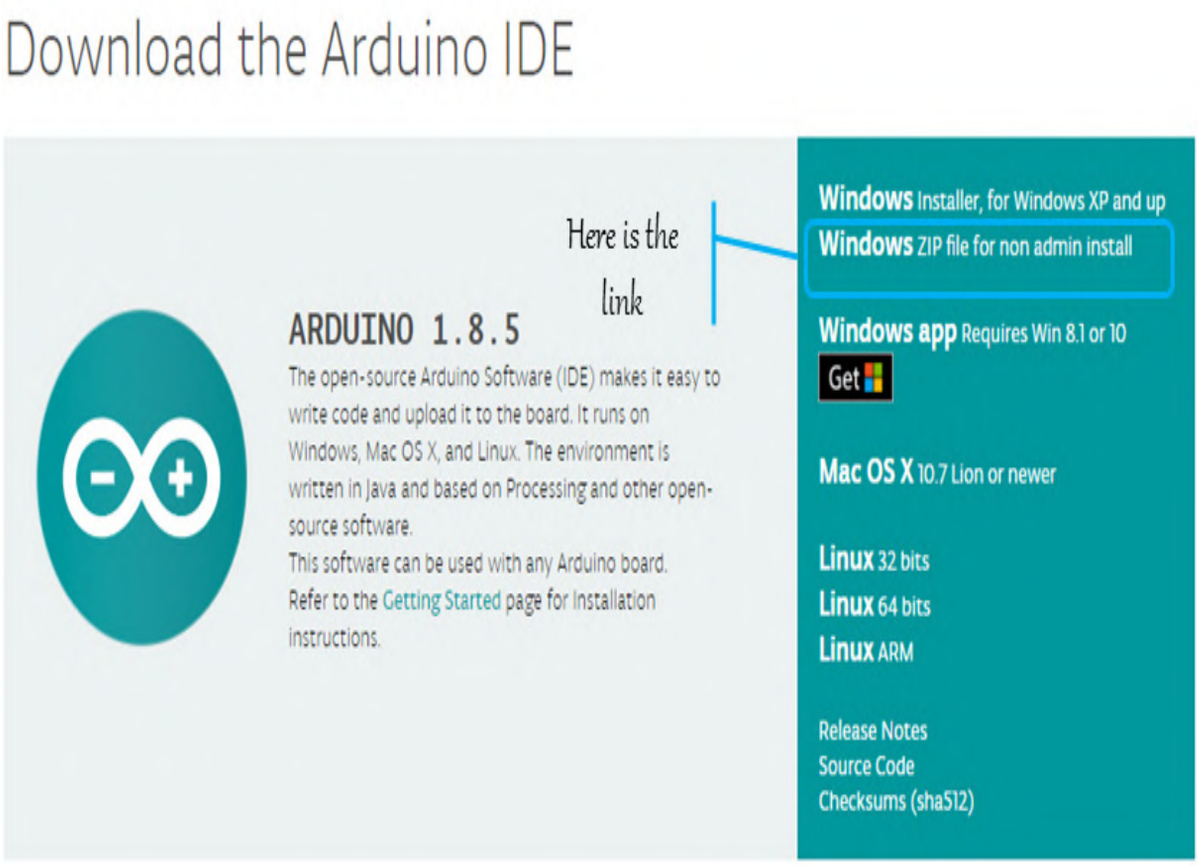
3. If you have downloaded "Windows Installer", double click on it and it will be installed. The first dialog box asks you to agree to the Arduino License Agreement. Clicking "I Agree" will take you to the next step of the installation.



With the Arduino Setup Installation Options, make sure that the
Install USB Driver and the Associate .ino Files boxes are checked. Create Start Menu Shortcut and Create Desktop Shortcut are optional but will help you navigate to the Arduino IDE in the future.



4.If you have downloaded "Windows ZIP file for non admin install", extract it and you will find arduino.exe. Double click on it to get started with the Arduino IDE.



On Linux

1. Go to https://www.arduino.cc/en/Main/Software.
2. Download "Linux 32 bits" or "Linux 64 bits" depending on your OS type.
3. Extract it and run the Arduino executables to get started with the Arduino IDE.

On Mac

1. Go to https://www.arduino.cc/en/Main/Software.
2. Download "Mac OS X 10.7 Lion or newer".
3. Extract it and run the Arduino executables to get started with the Arduino IDE.

Installing Additional Arduino Libraries

2

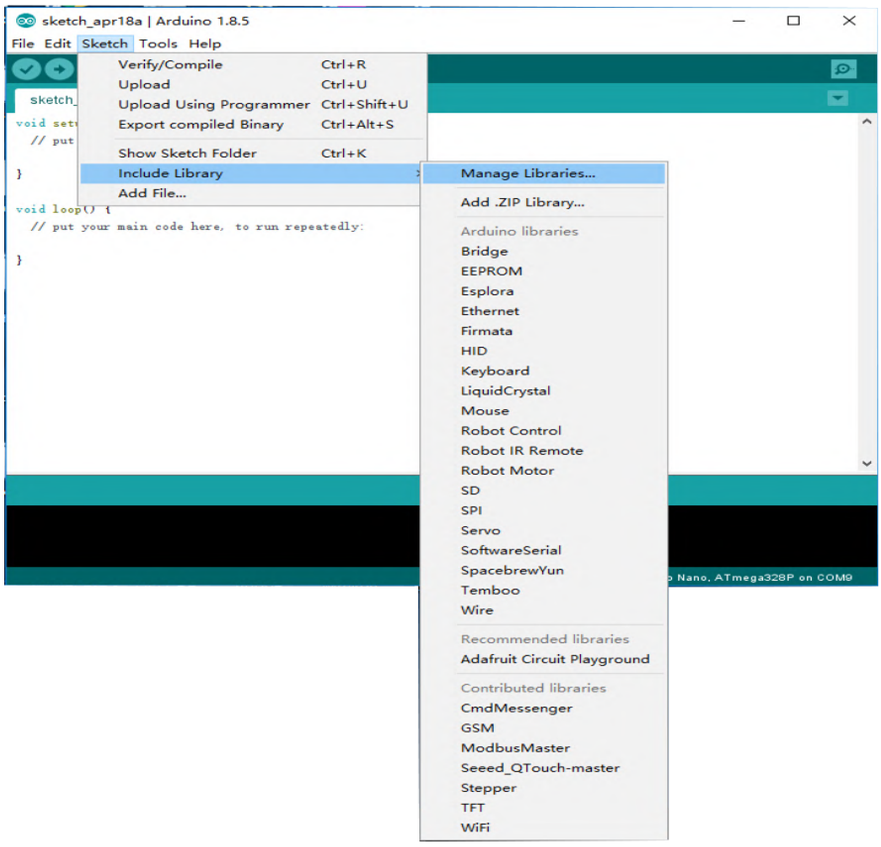
What are Libraries?

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in LiquidCrystal library makes it easy to talk to character LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. To use the additional libraries, you will need to install them.

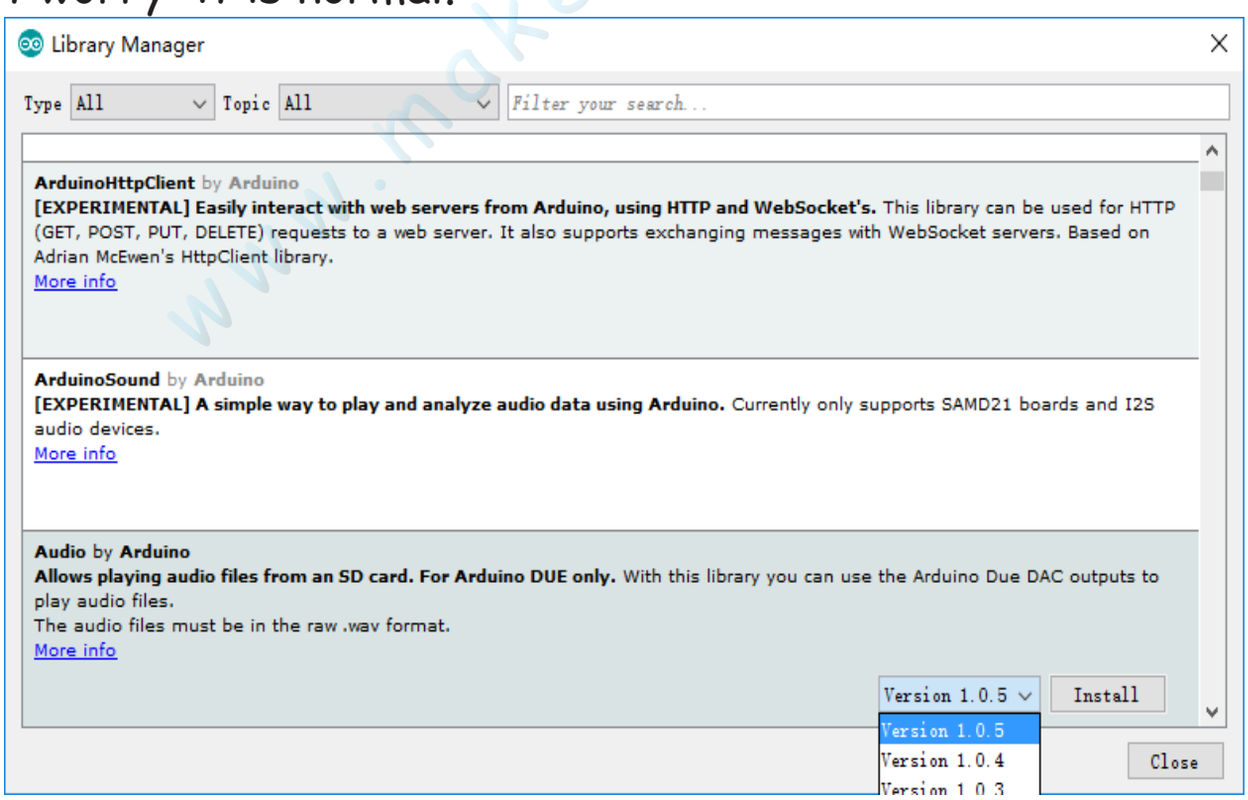
How to Install a Library?

Using the Library Manager

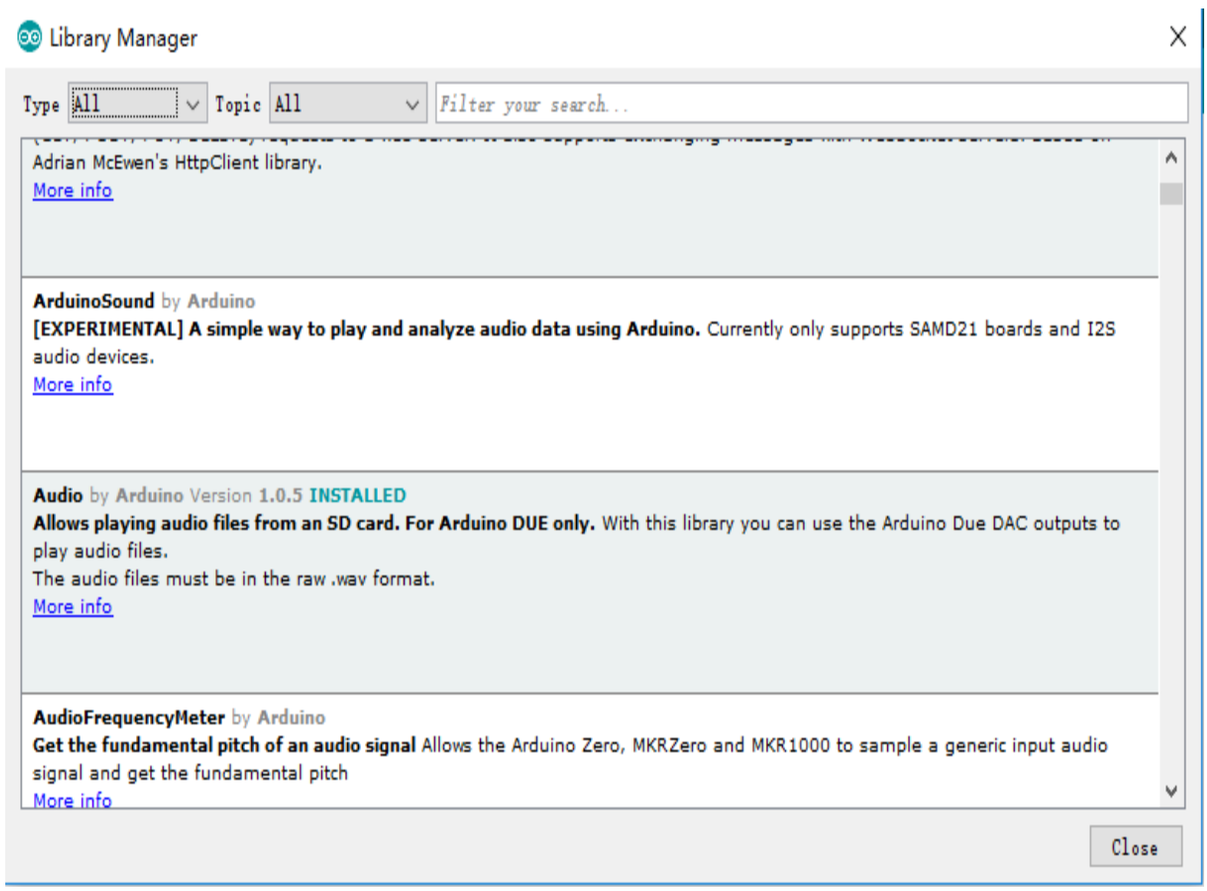
To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.6.2). Open the IDE and click to the "Sketch" menu and then Include Library > Manage Libraries.



Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation. In this example we will install the Bridge library. Scroll the list to find it, click on it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry: it is normal.



Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an Installed tag should appear next to the Bridge library. You can close the library manager.

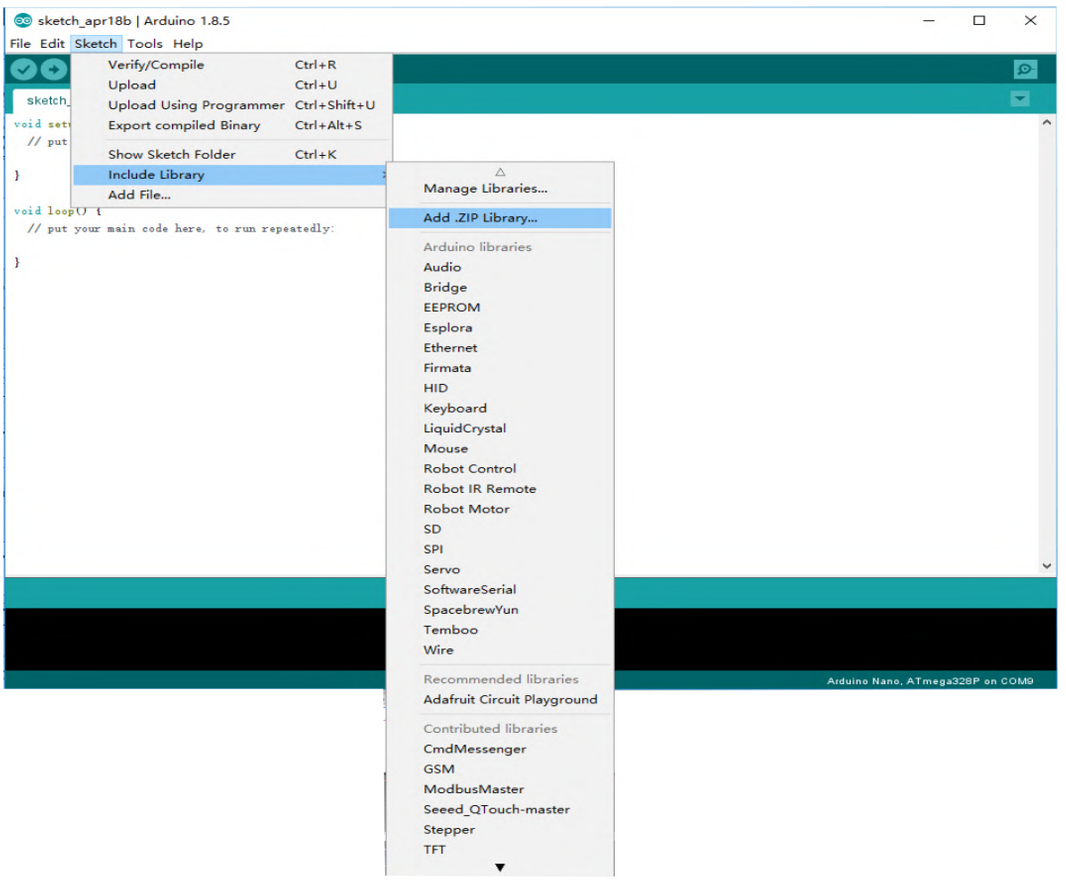


You can now find the new library available in the Sketch > Include Library menu. If you want to add your own library to Library Manager, follow these instructions.

Importing a .zip Library

Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be a .cpp file, a .h file and often a keywords.txt file, examples folder, and other files required by the library. Starting with version 1.0.5, you can install 3rd party libraries in the IDE. Do not unzip the downloaded library, leave it as is.

In the Arduino IDE, navigate to Sketch > Include Library > Add .ZIP Library. At the top of the drop down list, select the option to "Add .ZIP Library''.



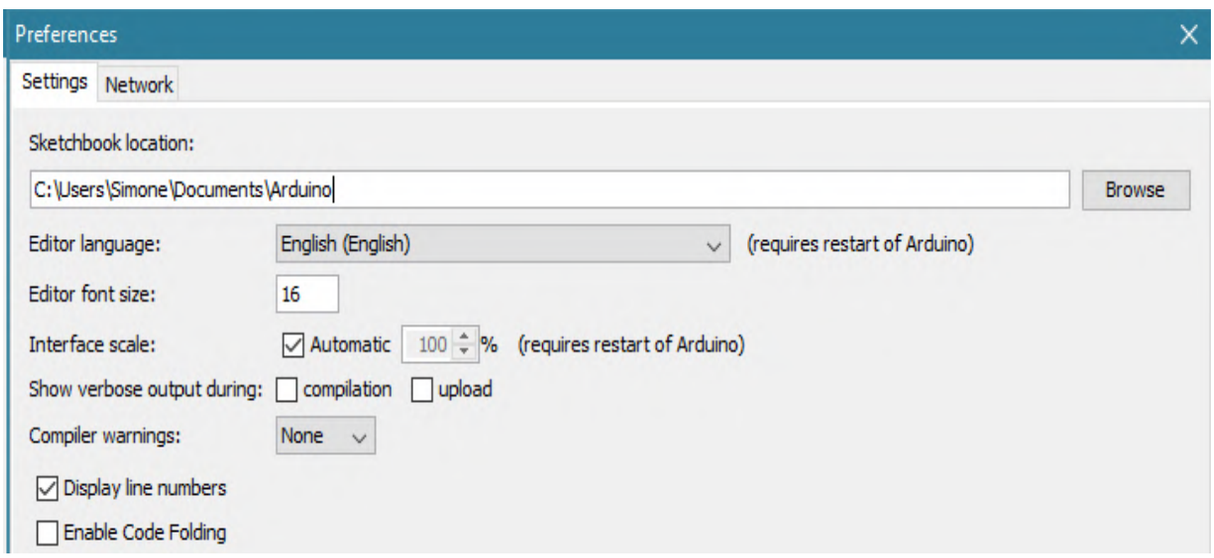
You will be prompted to select the library you would like to add. Navigate to the .zip file's location and open it.

Return to the Sketch > Include Library menu. menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in your sketch. The zip file will have been expanded in the libraries folder in your Arduino sketches directory.

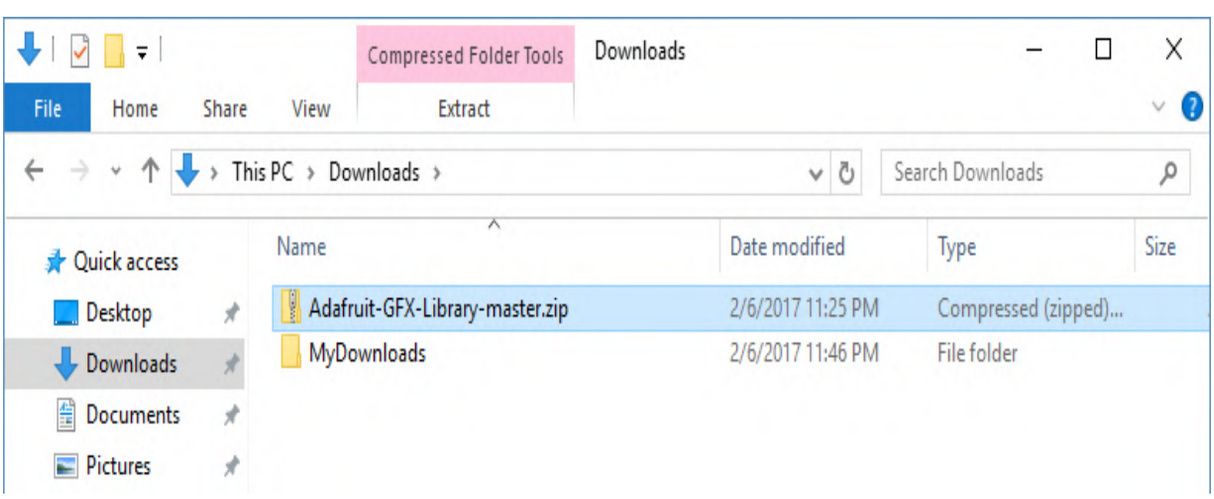
Note: the Library will be available to use in sketches, but with older IDE versions examples for the library will not be exposed in the File > Examples until after the IDE has restarted.

Manual installation

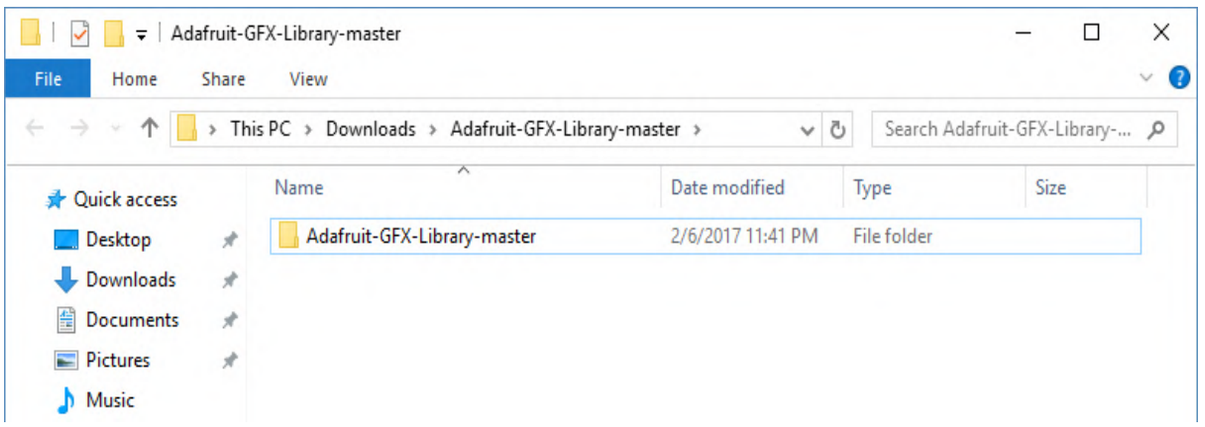
When you want to add a library manually, you need to download it as a ZIP file, expand it and put in the proper directory. The ZIP file contains all you need, including usage examples if the author has provided them. The library manager is designed to install this ZIP file automatically as explained in the former chapter, but there are cases where you may want to perform the installation process manually and put the library in the libraries folder of your sketchbook by yourself.
You can find or change the location of your sketchbook folder at File > Preferences > Sketchbook location.



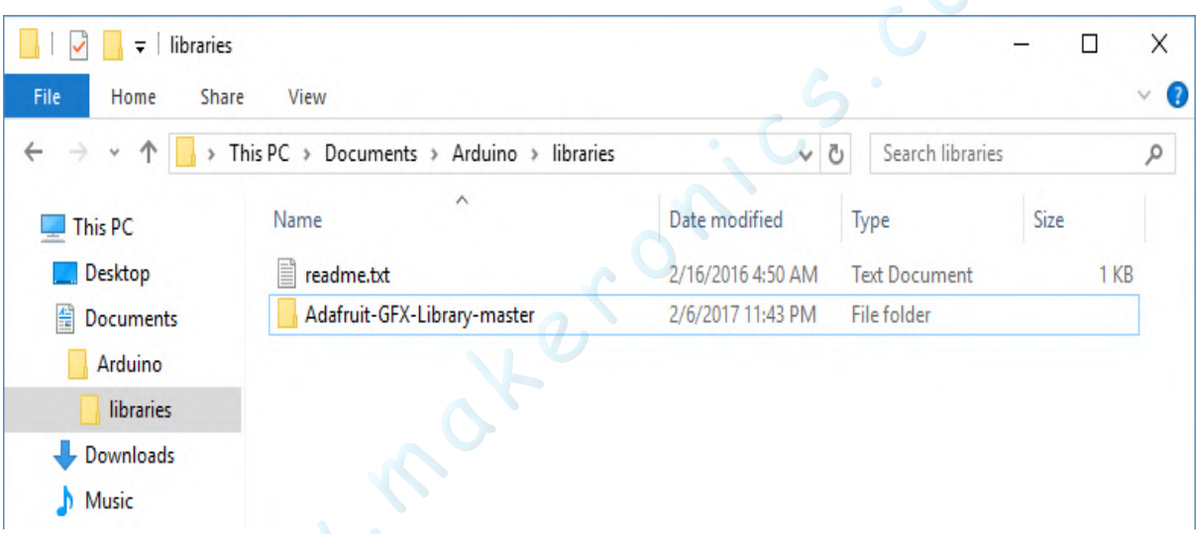
Go to the directory where you have downloaded the ZIP file of the library.



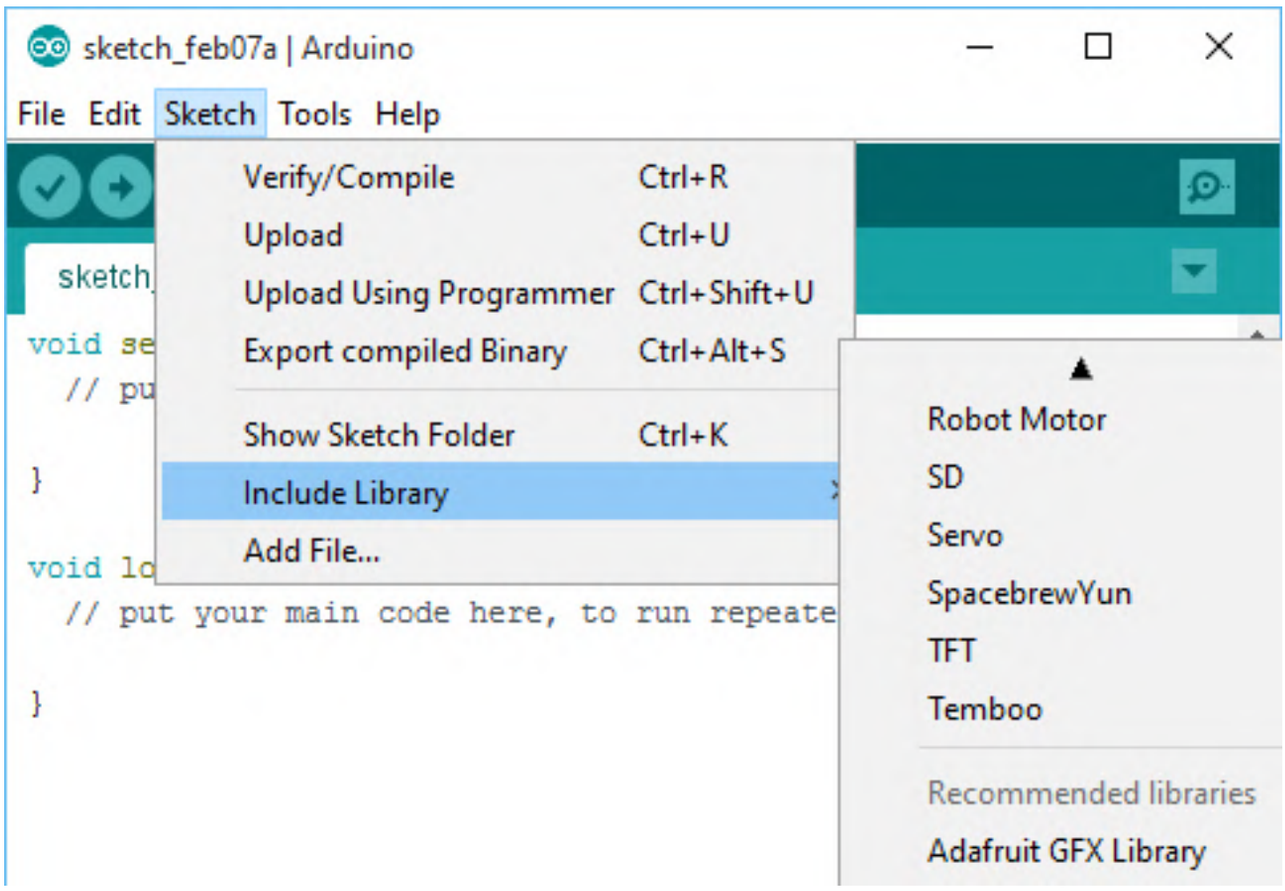
Extract the ZIP file with all its folder structure in a temporary folder, then select the main folder, that should have the library name.



Copy it in the "libraries" folder inside your sketchbook.



Start the Arduino Software (IDE), go to Sketch > Include Library. Verify that the library you just added is available in the list.


sketch_feb07a | Arduino
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Robot Motor
SD
Servo
SpacebrewYun
TFT
Temboo
Recommended libraries
Adafruit GFX Library
// put your main code here, to run repeate

3

Blink

In this lesson, you will learn how to program your UNO R3 controller board to blink the Arduino 's built-in LED, and how to load programs by basic steps.

Component Required:

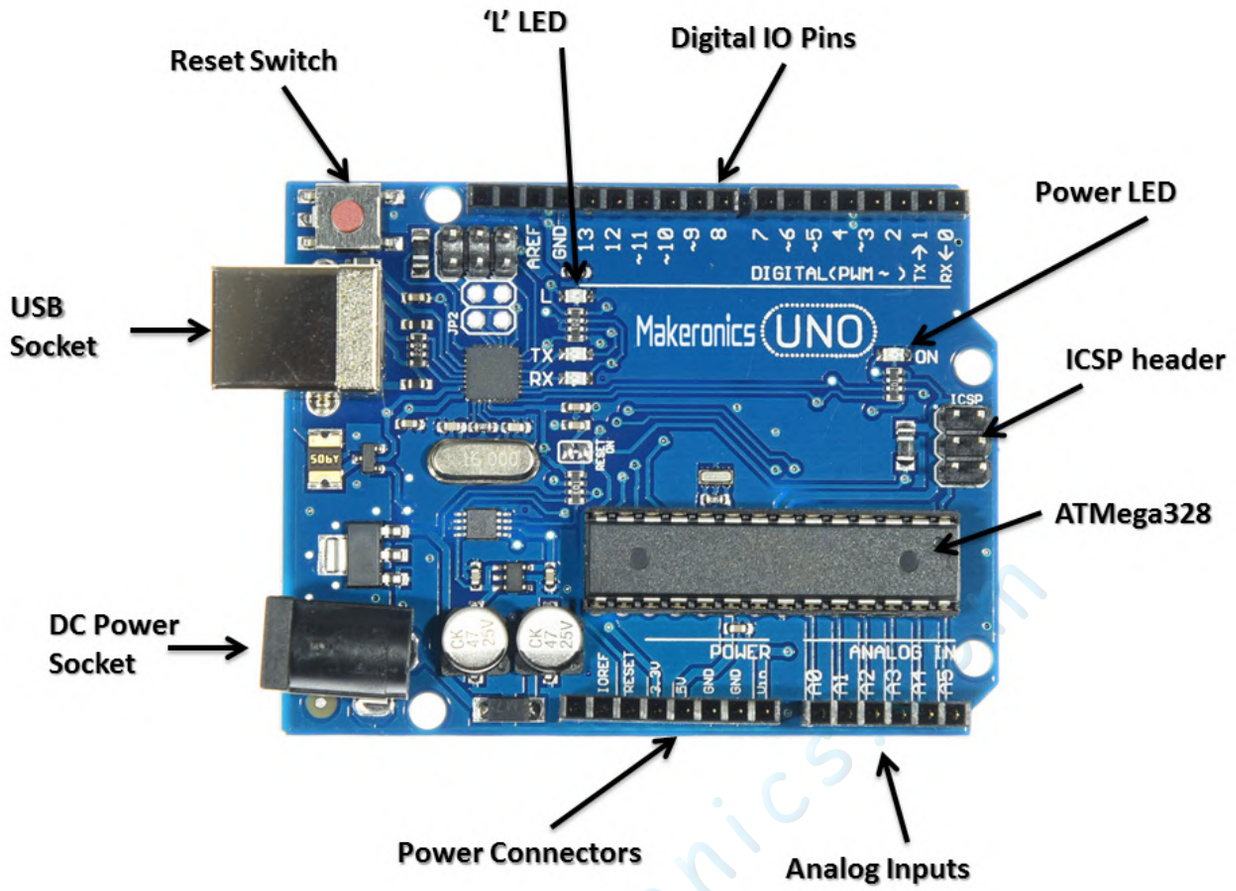
- 1x Makeronics UNO R3

Principle:

There are various types of Arduino boards, but by far the most common and the one that is used for all the projects in this book is the Arduino Uno.

Lets start our tour of the Arduino with USB socket, which serves several purposes: it can be used to provide power to the Arduino, for programming the Arduino from your computer, and finally, as a communicatons link.

The little red button next to the USB socket is the Reset button. When you press this, the Arduino will restart and run the program that is installed on it.



Along both the top and bottom edges of the Arduino, there are connection sockets to which electronics can be attached. The digital inputs and outputs, identified by a number between 0 and 13. Each pin can be configured in your programs to be either an input or an output. You might connect a switch to a digital input, and the digital input will be able to tell if the switch is pressed or not. Alternatively, you might connect an LED to a digital output and by changing the digital output from low to high, you can turn the LED on. In fact there is one LED built onto the board called the "L" LED that is connected to digital pin 13.

Beneath the digital I/O pins, there is a power LED that simply indicates that the board is powered. The ICSP (In-Circuit Serial Programming) header is only for advanced programming of the Arduino without using the USB connection. Most users of Arduino will never use the ICSP header.

It is worth highlighting the ATMega328 (the brains of the Arduino). The ATMega328 is a microcontroller IC (Integrated Circuit). This chip stores the program that the Arduino will run (it contains 32 KB of flash memory).

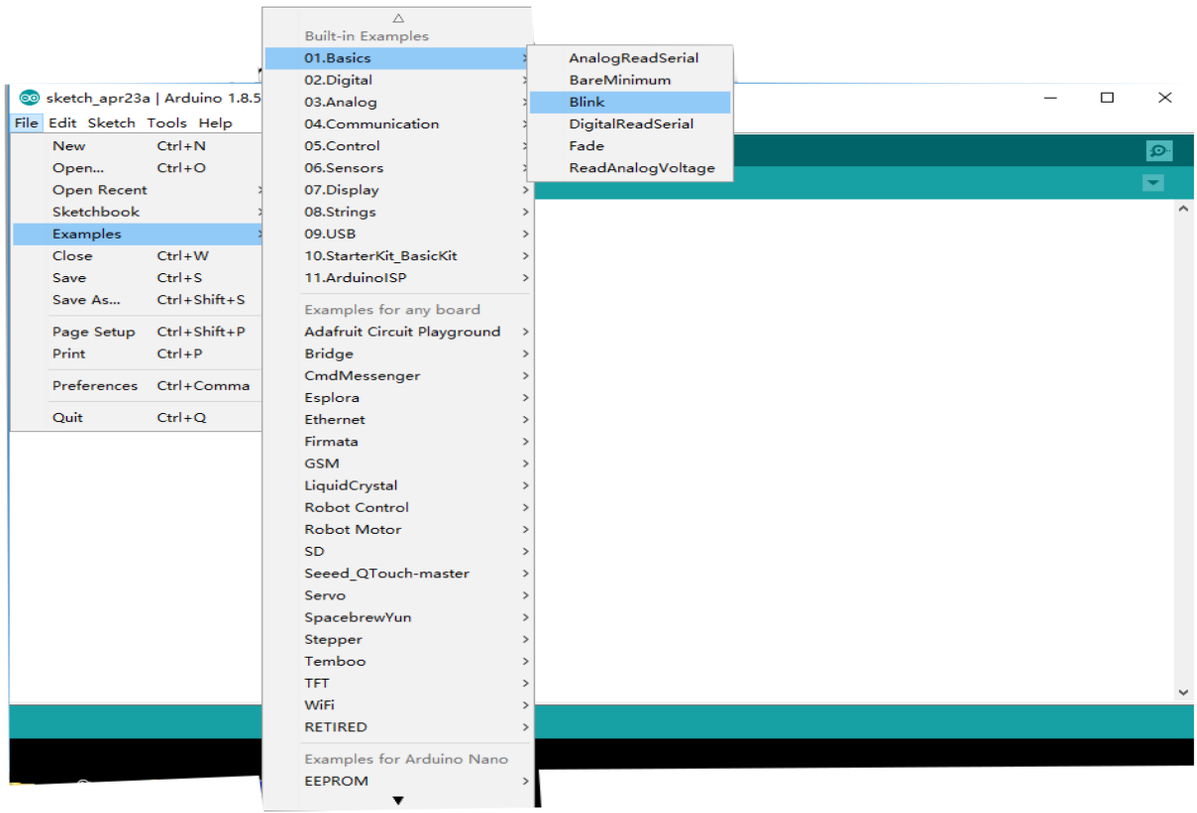
Below the ATMega328, there is a row of analog input pins labeled A0 to A5. Whereas digital inputs can only tell if something is on or off, analog inputs can measure the voltage at the pin (as long as the voltage is between 0 and 5V). Such a voltage could be from a sensor of some sort. If you run short of digital inputs and outputs, then these analog input pins can also be used as digital inputs and outputs.

Next to this, there is a row of power connectors that can be used as alternative ways of supplying power to the Arduino. They can also be used to supply power to other electronics that you build.

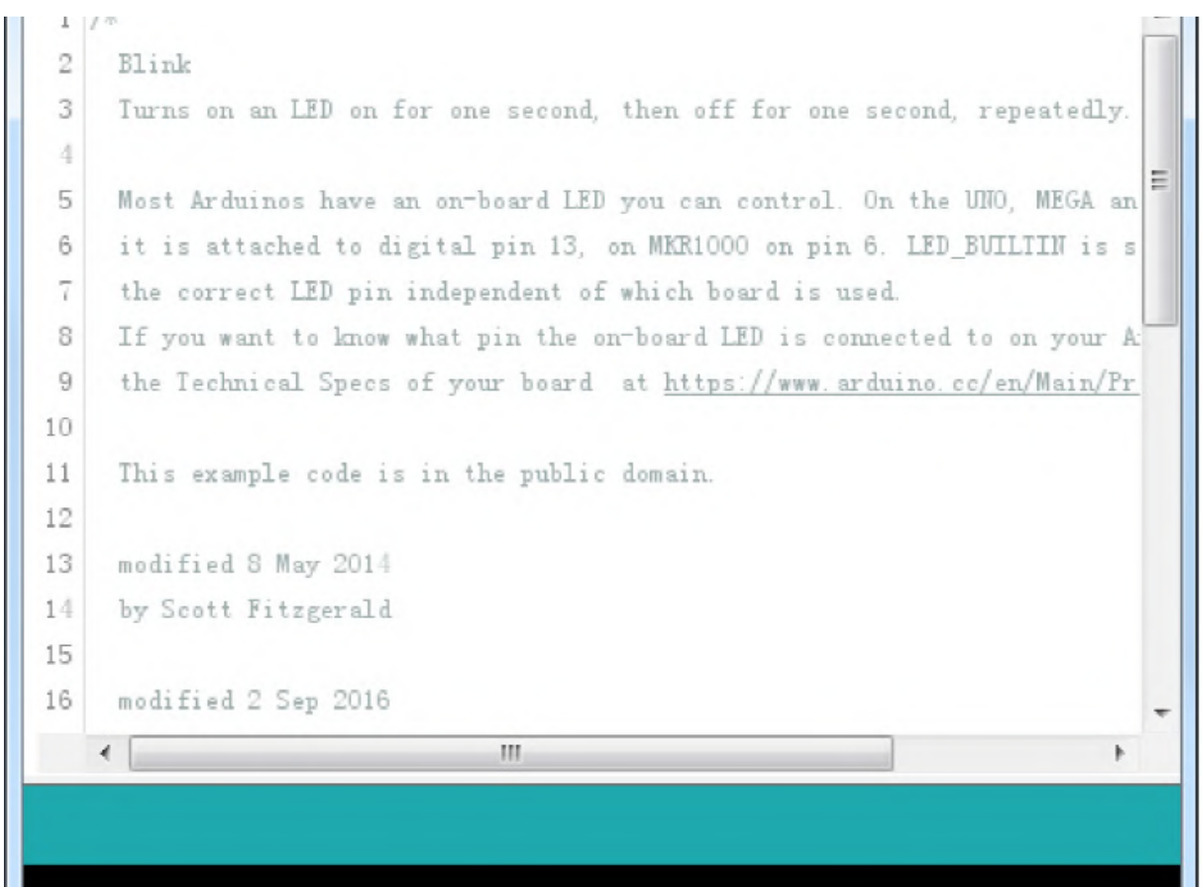
The Arduino also has a DC power jack. This can accept anything between 7V and 12V DC and uses a voltage regulator to provide the 5V that the Arduino operates on. The Arduino will automatically accept power from the USB socket and power from the DC connector depending on which is connected.

In this lesson, we will reprogram the UNO R3 board with our own Blink sketch and then change the rate at which it blinks. In Lesson 2, you set up your Arduino IDE and made sure that you could find the right serial port for it to connect to your UNO R3 board. Now it's time to put that connection to the test and program your UNO R3 board.
The Arduino IDE includes a large collection of example sketches that you can load and use. This includes an example sketch for making the 'L' LED blink.

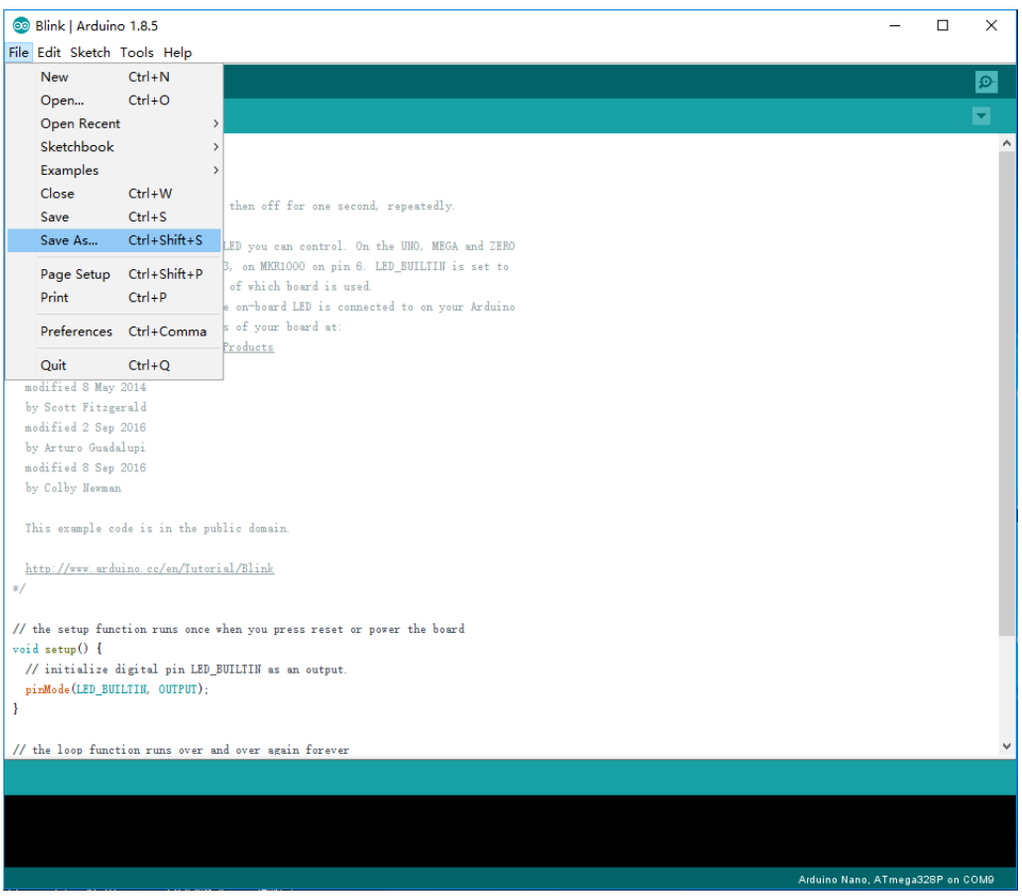
Load the 'Blink' sketch that you will find in the IDE's menu system under File > Examples > 01.Basics

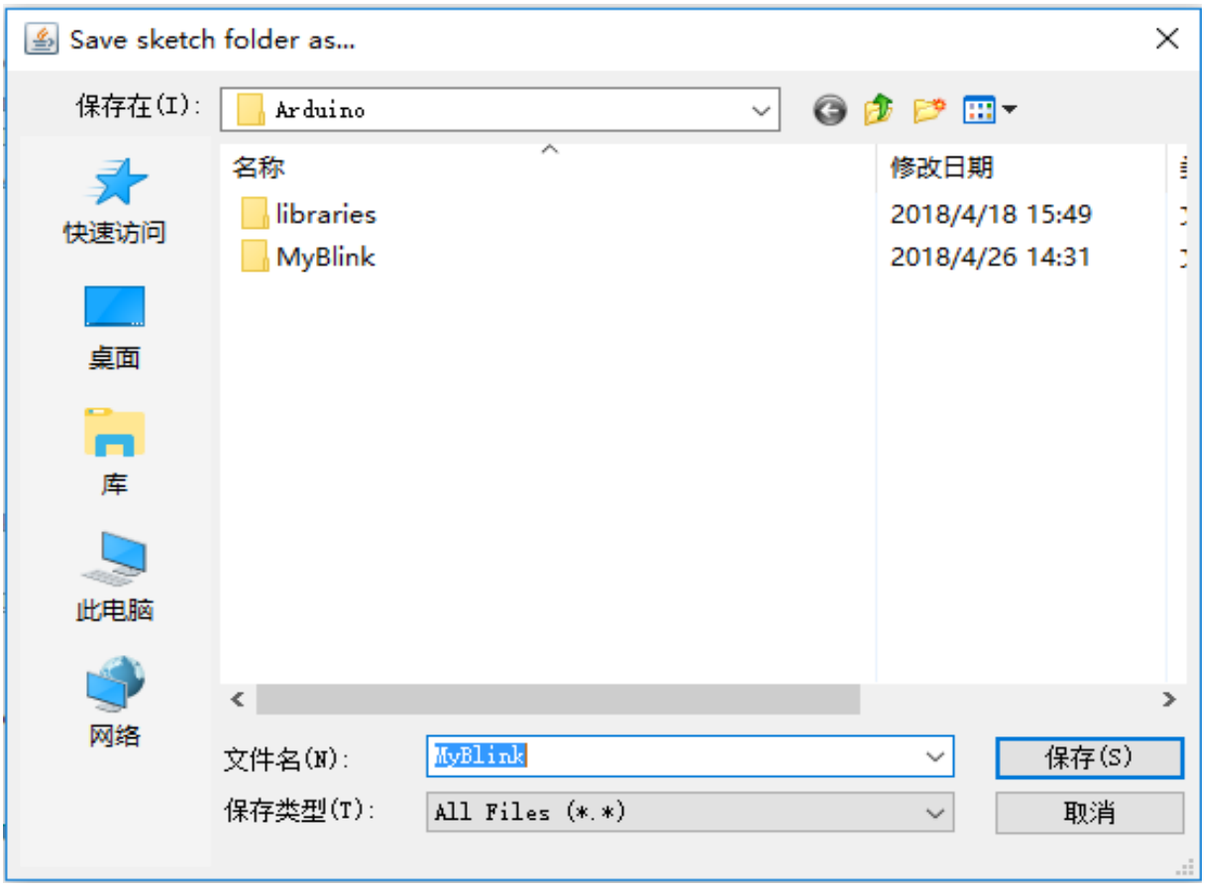


When the sketch window opens, enlarge it so that you can see the entire sketch in the window.

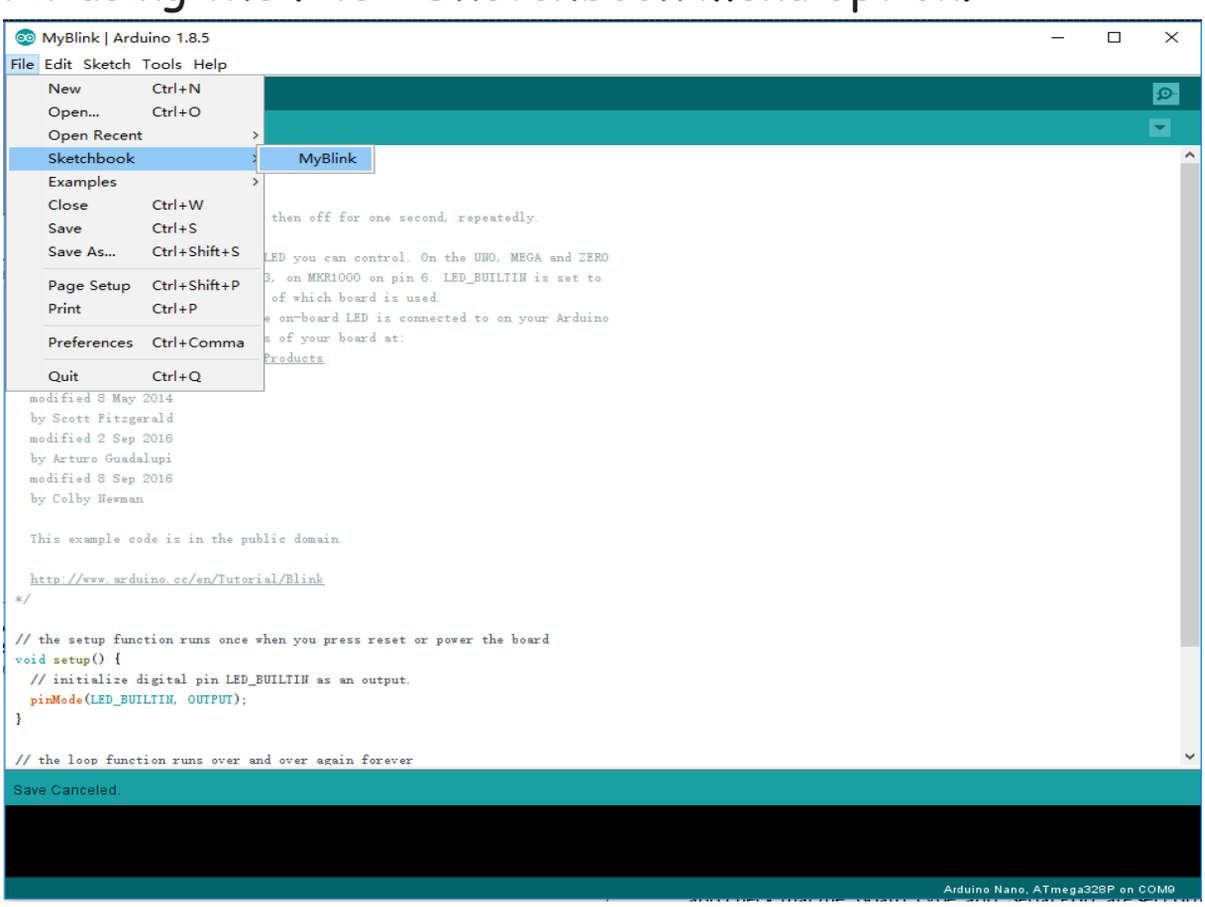


The example sketches included with the Arduino IDE are 'read-only'. That is, you can upload them to an UNO R3 board, but if you change them, you cannot save them in the same file. Since we are going to change this sketch, the first thing you need to do is save your own copy.
From the File menu on the Arduino IDE, select 'Save As..' and then save the sketch with the name 'MyBlink'.

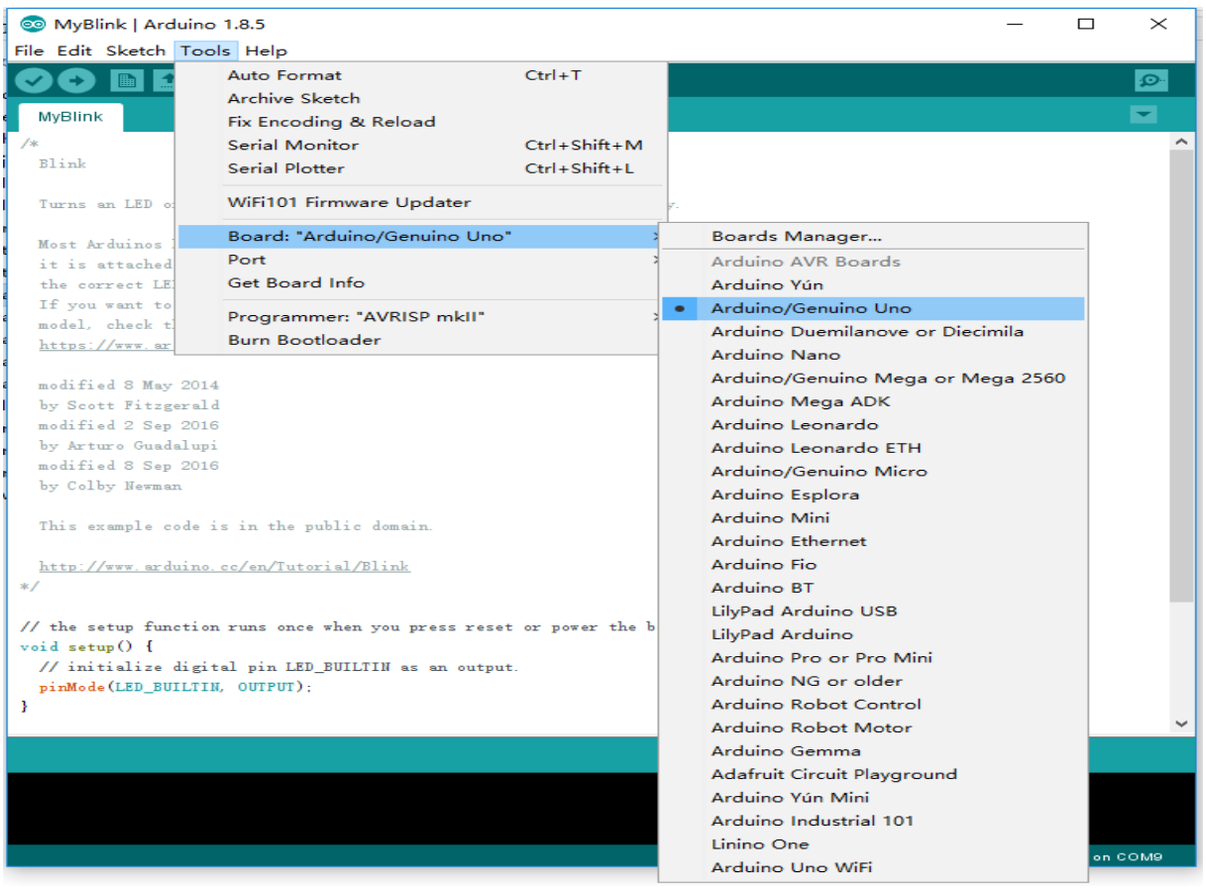


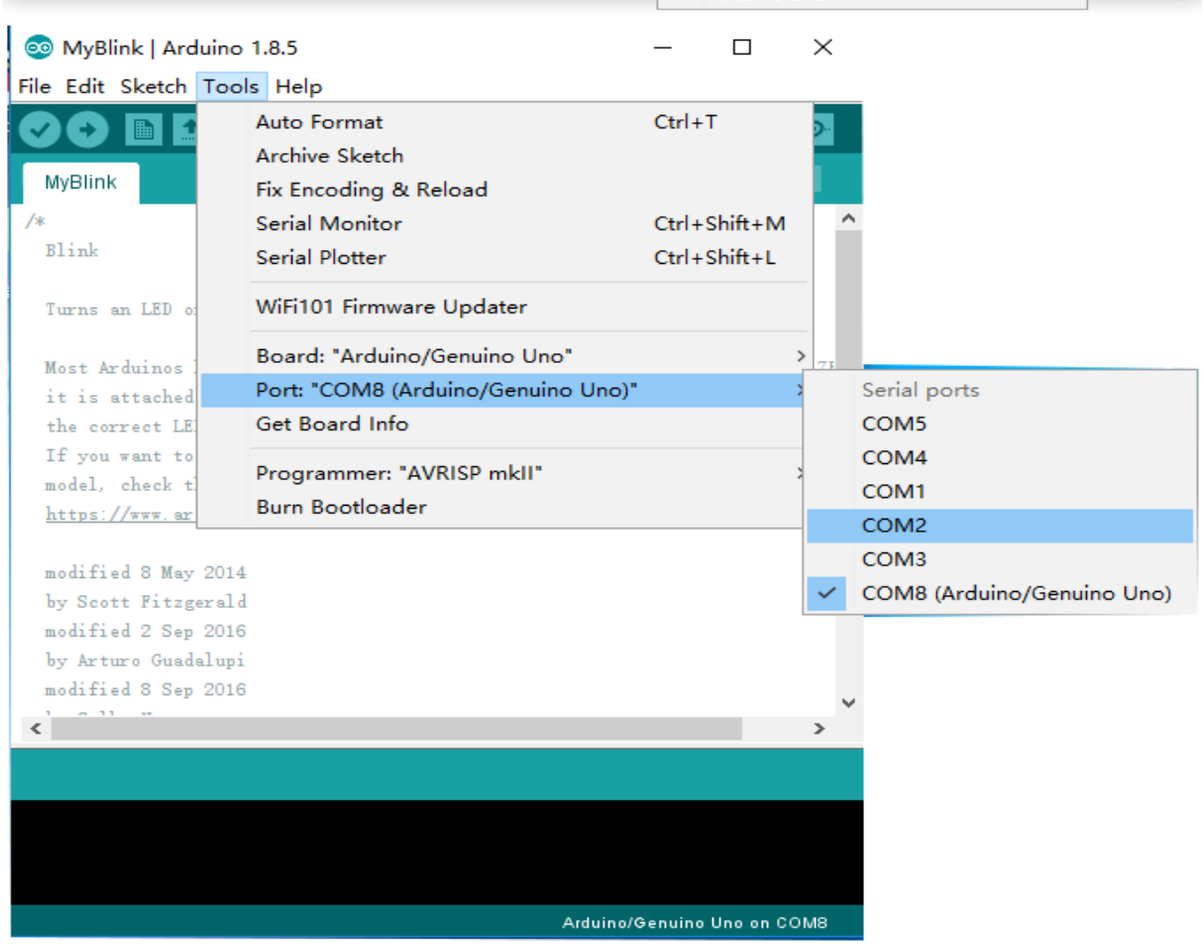


You have saved your copy of 'Blink' in yourown sketchbook. This means that if you ever want to find it again, you can just open it using the File > Sketchbook menu option.



Attach your Arduino board to your computer with the USB cable and check that the 'Board Type' and 'Serial Port' are set correctly.



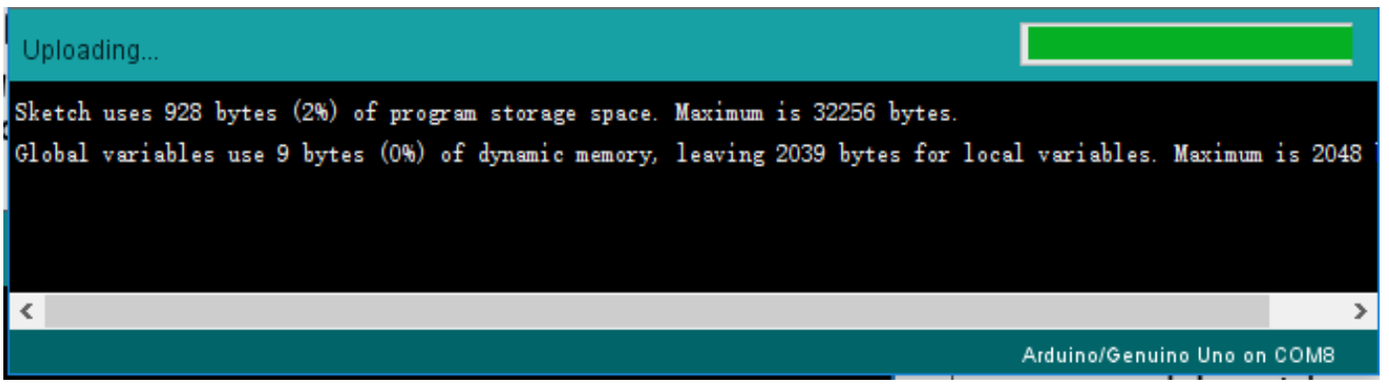


Note: The Board Type and Serial Port here are not necessarily the same as shown in picture. If you are using 2560, then you will have to choose Mega 2560 as the Board Type, other choices can be made in the same manner. And the Serial Port displayed for everyone is different, despite COM 8 chosen here, it could be COM3 or COM4 on your computer. A right COM port is supposed to be COMX (arduino XXX), which is by the certification criteria.
The Arduino IDE will show you the current settings for board at the bottom of the window.

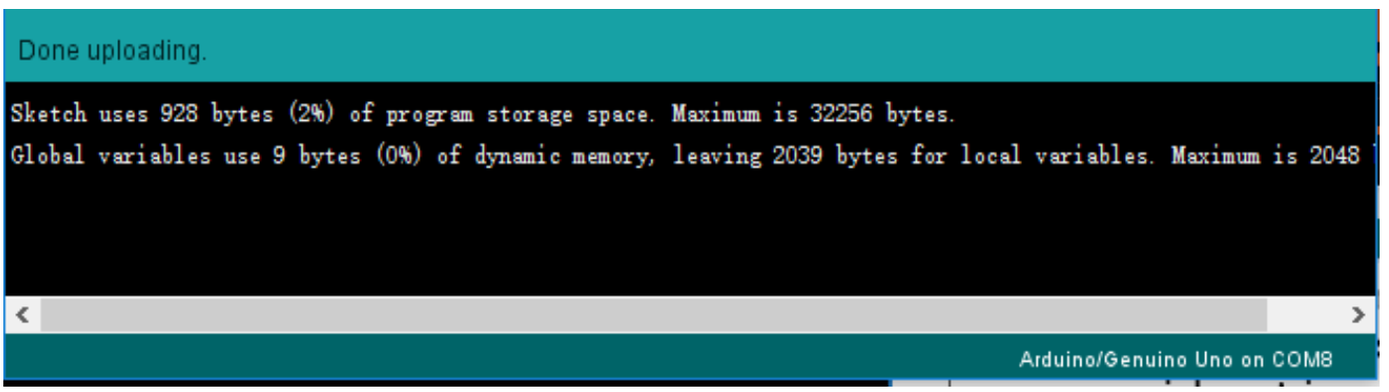
Click on the 'Upload' button. The second button from the left on the toolbar.

If you watch the status area of the IDE, you will see a progress bar and a series of messages. At first, it will say 'Compiling Sketch...'. This converts the sketch into a format suitable for uploading to the board.

Next, the status will change to 'Uploading'. At this point, the LEDs on the Arduino should start to flicker as the sketch is trrassferred.



Finally, the staus will change to 'Done'.



The other message tells us that the sketch is using 928 bytes of the 32,256 bytes available.
After the 'Compiling Sketch..' stage you could get the following error message:

It means that your board is not connected at all, or the drivers have not been installed (if necessary) or that the wrong serial port is selected.
If you encounter this error, go back to Lesson 1 and check your installation.
Once the upload has completed, the board should restart and start blinking.

Note that a huge part of this sketch is composed of comments. These are not actual program instructions; rather, they just explain how the program works. They are there for your benefit.
Everything between /* and */ at the top of the sketch is a block comment, it explains what the sketch is for.

code:

Single line comments start with // and everything up until the end of that line is considered as a comment.
The first line of code is:

```
int led = 13;
```

As the comment above it explains, this is giving a variable name to the pin that the LED is attached to. This is 13 on most Arduinos, including the UNO and Mega2560.

Next, we have the 'setup' function. Again, as the comment says, this is executed when the reset button is pressed. It is also executed whenever the board resets for any reason, such as power first being applied to it, or after a sketch has been uploaded.

```
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
```

Every Arduino sketch must have a 'setup' function, and the place where you might want to add instructions of your own is between the { and the }.
In this case, there is just one command there, which, as the comment states tells the Arduino board that we are going to use the LED pin as an output.

It is also mandatory for a sketch to have a 'loop' function. Unlike the 'setup' function that only runs once, after a reset, the 'loop' function will, after it has finished running its commands, immediately start again.

```
void loop() {
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Inside the loop function, the commands first of all turn the LED pin on (HIGH), then 'delay' for 1000 milliseconds (1 second), then turn the LED pin off and pause for another second.

You are now going to make your LED blink faster. As you might have guessed, the key to this lies in changing the parameter in () for the 'delay' command.

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                        // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                        // wait for a second
}
```

This delay period is in milliseconds, so if you want the LED to blink twice as fast, change the value from 1000 to 500. This would then pause for half a second each delay rather than a whole second.

Upload the sketch again and you should see the LED start to blink more quickly.

Project 1

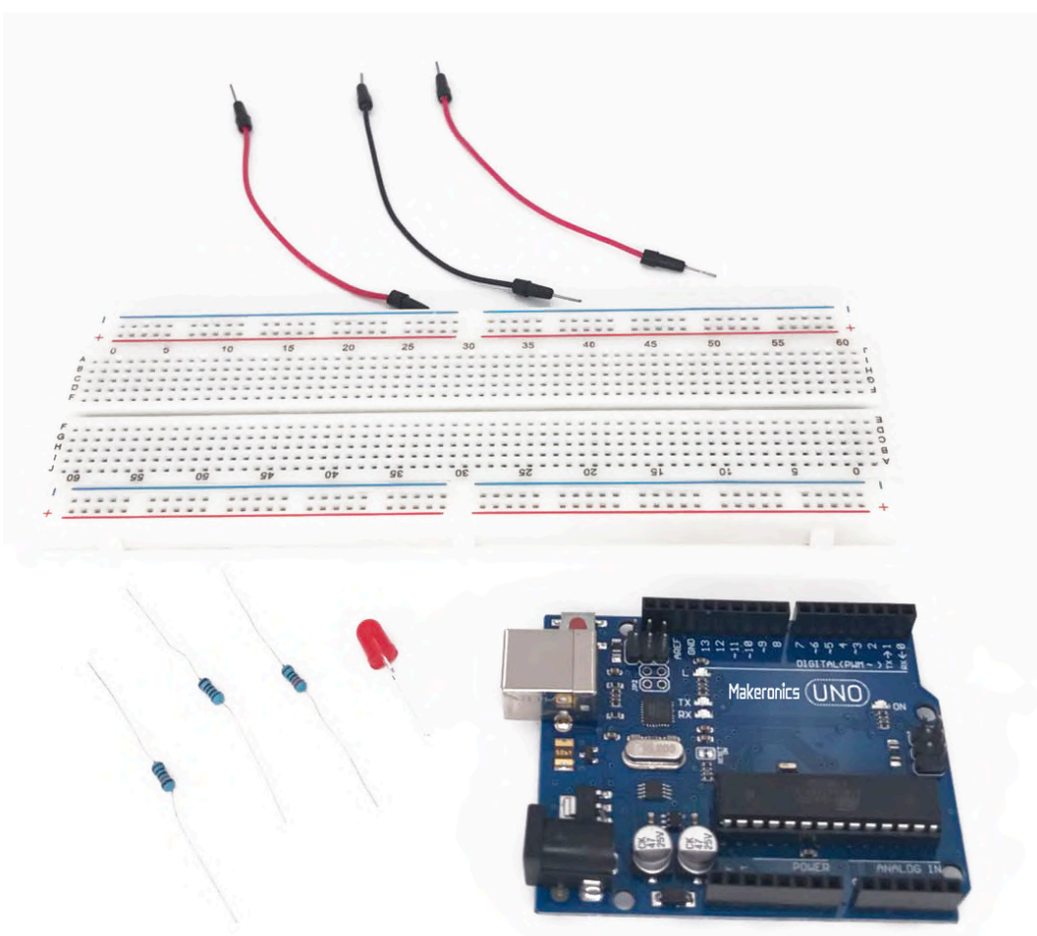
4

External LED

In this lesson, you will learn how to change the brightness of an LED by using different values of resistor.

Component Required:

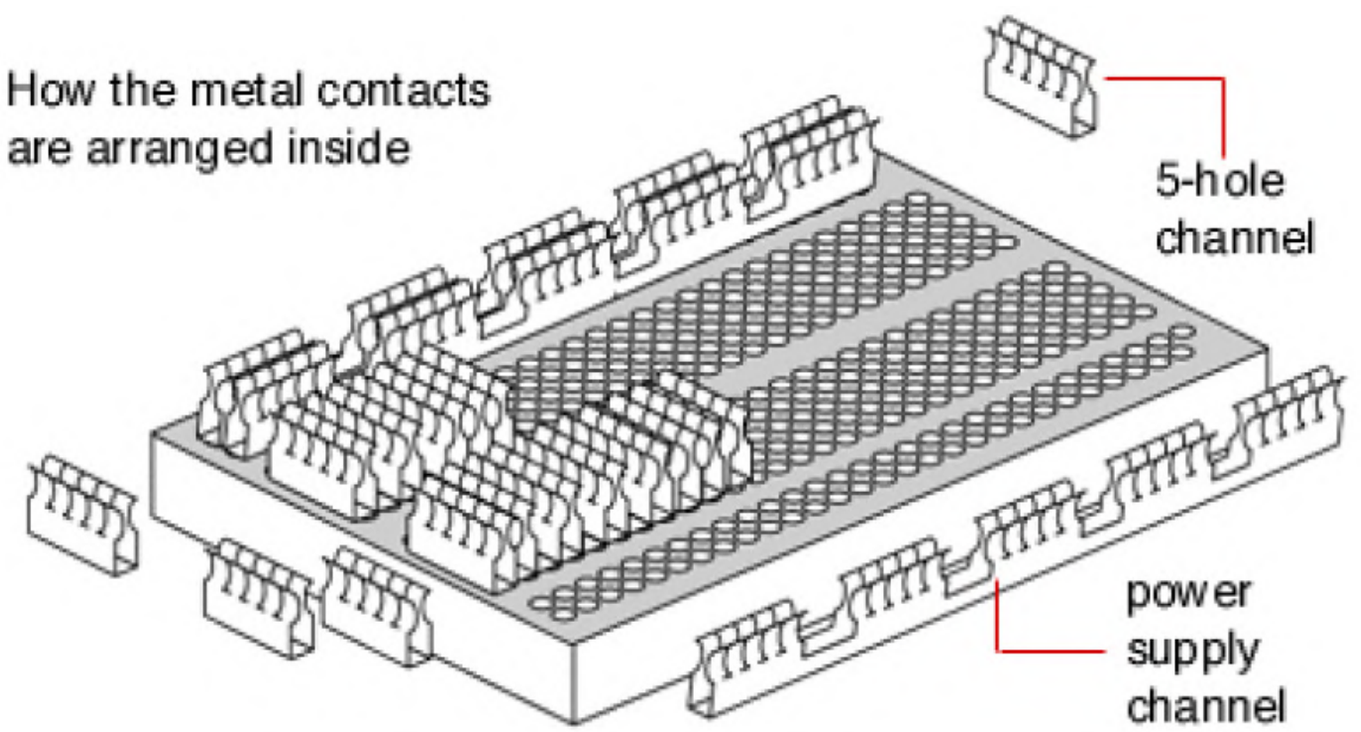
- 1 x Makeronics Uno R3
- 1 x 830 Tie Points Breadboard
- 1 x 5mm red LED
- 1 x 220 ohm resistor
- 1 x 1k ohm resistor
- 1 x 10k ohm resistor
- 3 x M-M wires (Male to Male jumper wires)



Component Introduction:

BREADBOARD:

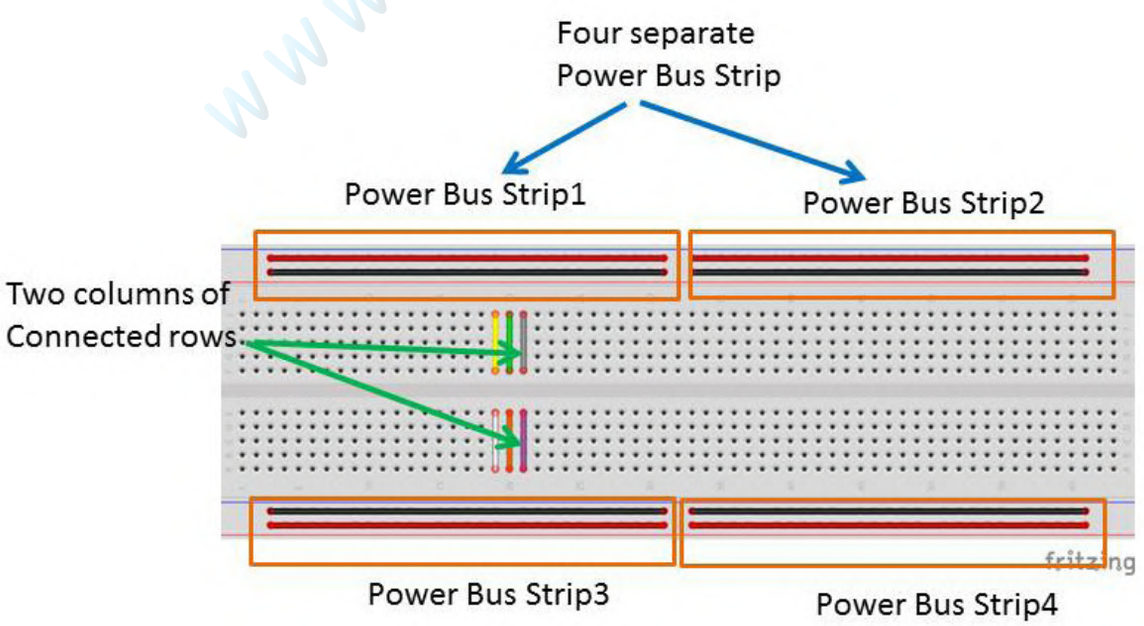
the solderless breadboard does not require soldering, it is reusable. This makes it easy to use for creating temporary prototypes and experimenting with circuit design.



A solderless breadboard socket consists of a perforated block of plastic with numerous tin plated phosphor bronze or nickel silver alloy spring clips under the perforations. The clips are often called tie points or contact points. The number of tie points is often given in the specification of the breadboard. The breadboard used throughout this book is often referred to as a full-sized breadboard or 830-point breadboard (because it has 830 holes in it). There are bigger and smaller breadboards available, you can goto www.makeronics.com website to select your breadboard. But this size is about right for all the projects and experiments in this book.

Solderless breadboards connect pin to pin by metal strips inside the breadboard. The layout of a typical solderless breadboard is made up from two types of areas, called strips. Strips consist of interconnected electrical terminals.

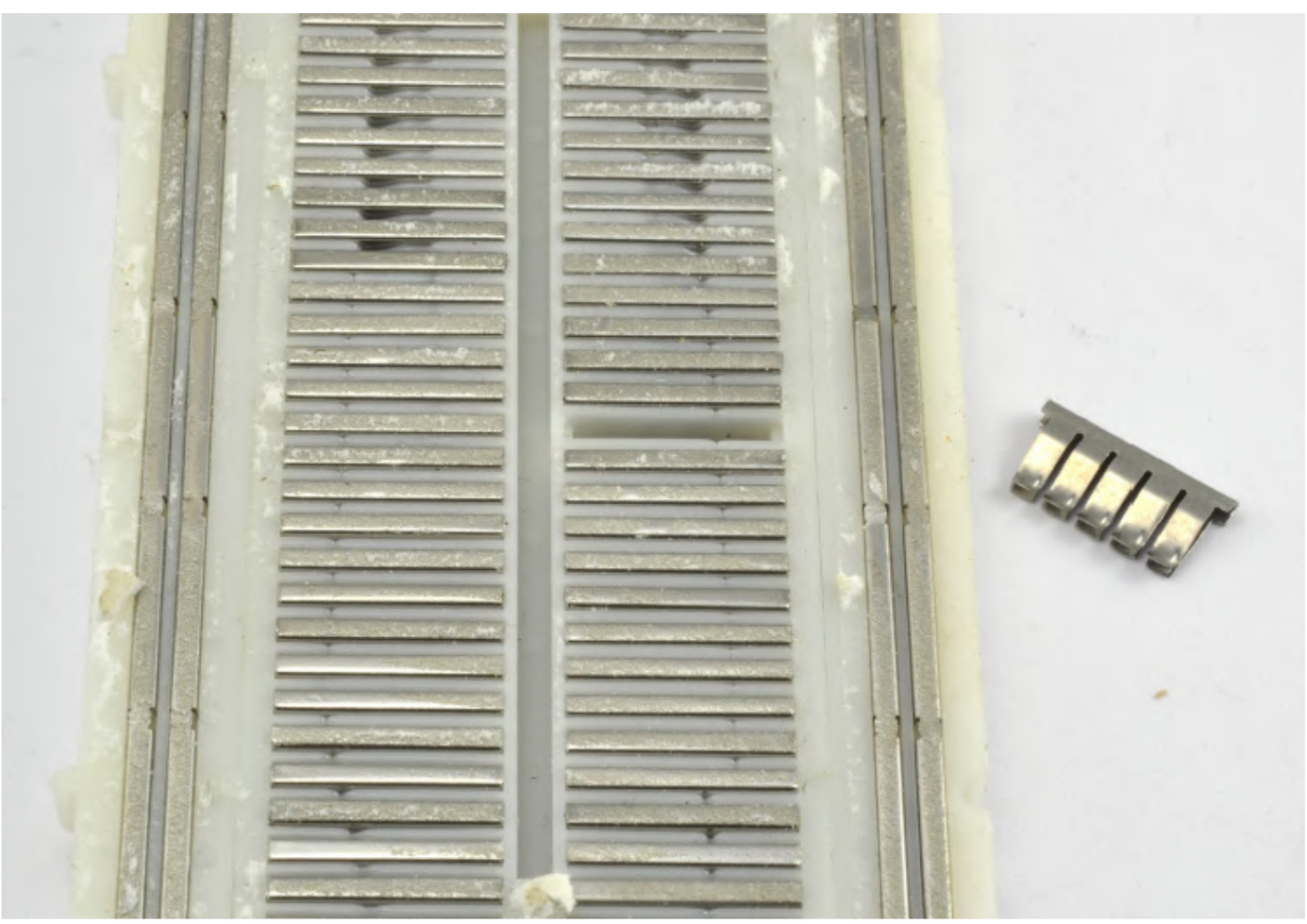
This type of breadboard has two rows on each side of the breadboard. These are usually marked with red and blue lines.



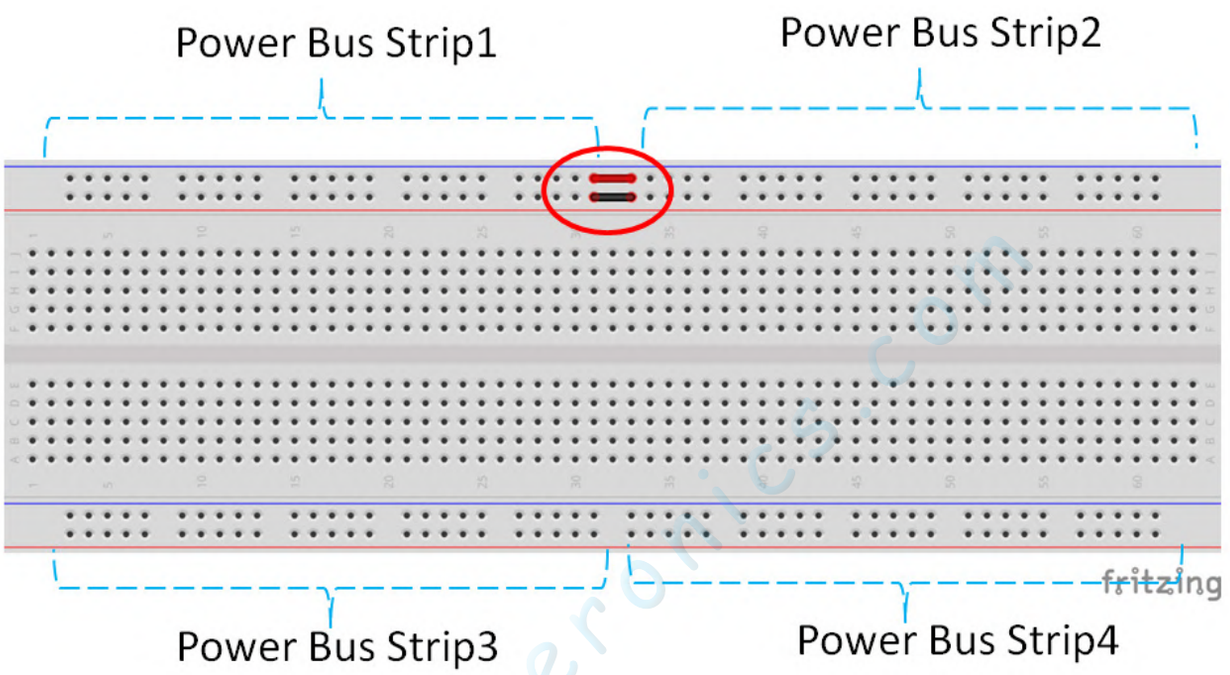
For any one of these rows, all the holes in that row are connected together electrically behind the plastic of the board. Although these rows can be used for anything, they are most often used for the positive and negative supply to your circuit.

The main body of the board is split into two banks, or columns, of five holes each down the whole length of the board. Each of these columns has all five holes connected together by a clip behind the plastic. To connect the leg of one component to another, both leads just need to be pushed into holes on the same column of the breadboard.

Behind the holes in the front plastic face of a breadboard, there are metal clips designed to grip wires and component leads. Please look at the photo below for a dismantled breadboard with one of the clips removed.



Note:
The breadboard 830 inside this starterkit has four seperate power bus strip. it can provide more power input than the other breadboard in market. If you need power bus strip2 keep same power input as power bus strip1, you need plug 2 dupont wires to bridge them like the shown in the figure below.

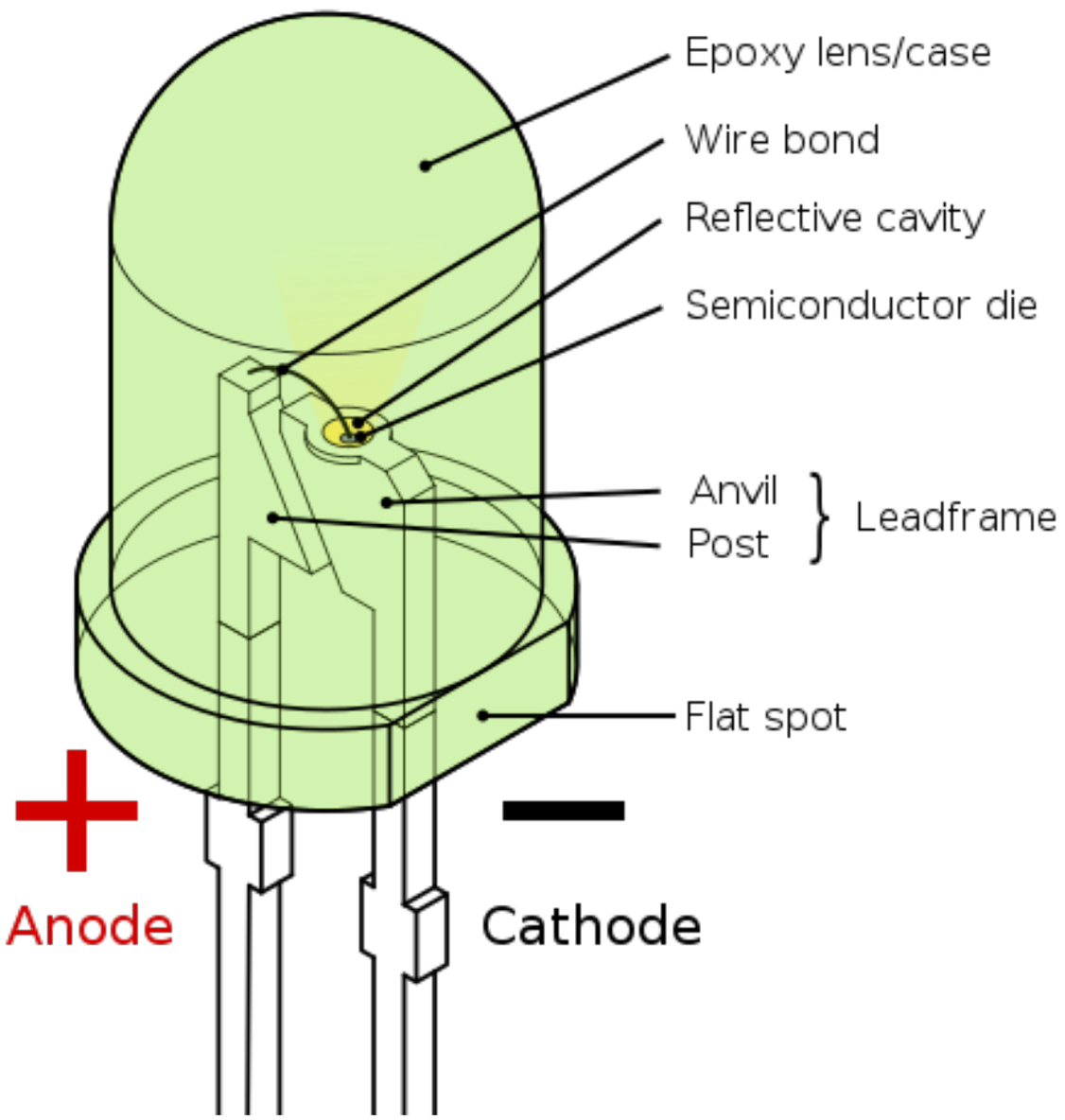


While breadboards are great for prototyping, they have some limitations. Because the connections are push-fit and temporary, they are not as reliable as soldered connections. If you are having intermittent problems with a circuit, it could be due to a poor connection on a breadboard.

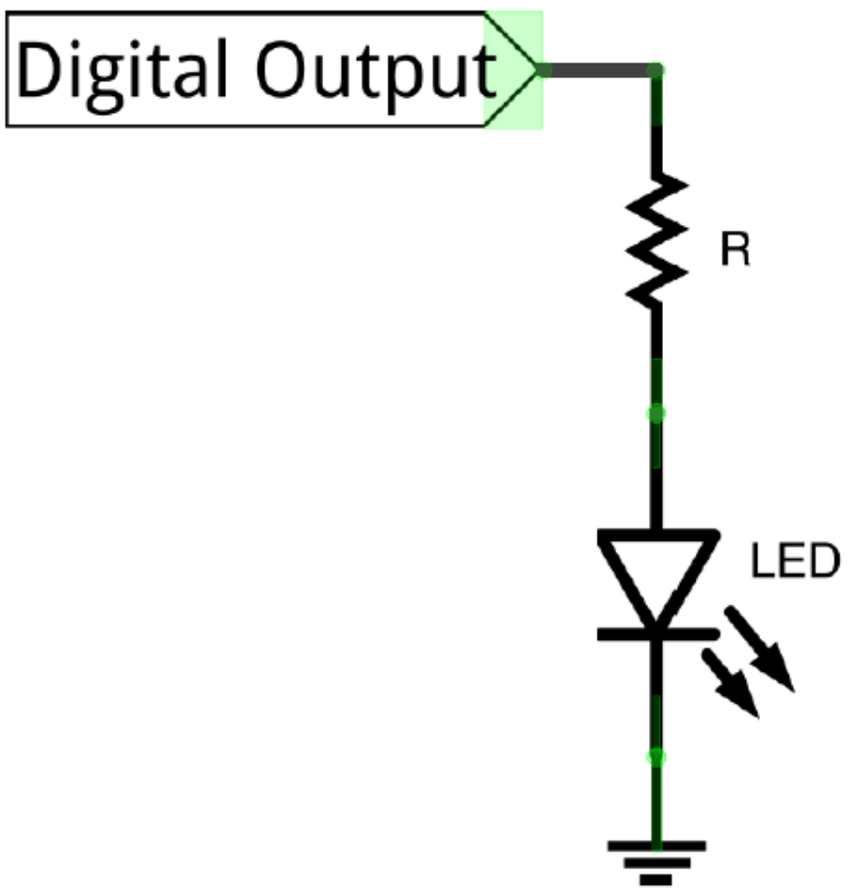
LED:

LED stands for Light Emitting Diode and, in its core, it's just a diode that emits light.

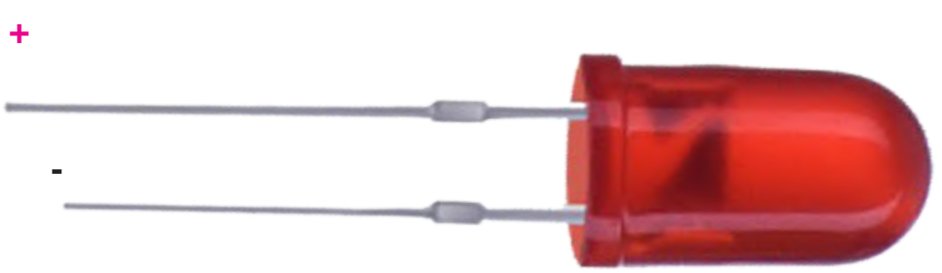
By regular LEDs, those small colorful devices that often come in a 5mm diameter (sometimes 3mm or 10mm) and require a modest current to make them light, so that they can be controlled directly from the output of an Arduino.



The resistor is needed to limit the current flowing through the LED for two reasons: first, so that the LED's maximum current is not exceeded (which will shorten its life); and second, so that the output-current capability of the output pin or the cumulative total of all the output pins has not been exceeded.



You'll notice that the LED has a short leg and a long leg. If you look really closely, you'll also see that the edge of the LED bulb has a flat surface on the same side as the short leg. These help you identify the polarity of the legs; the LED's long leg is the positive leg, and the short leg on the side of the flat bulb surface is the negative, or ground, leg.



RESISTORS:

A resistor is a passive two-terminal electrical component that implements electrical resistance as a circuit element. In electronic circuits, resistors are used to reduce current flow, adjust signal levels, to divide voltages, bias active elements, and terminate transmission lines, among other uses.

The unit of resistance is called the Ohm, which is usually shortened to Ω the Greek letter Omega. Because an Ohm is a low value of resistance (it doesn't resist much at all), we also denote the values of resistors in kΩ (1,000 Ω) and MΩ (1,000,000 Ω). These are called kilo-ohms and mega-ohms.

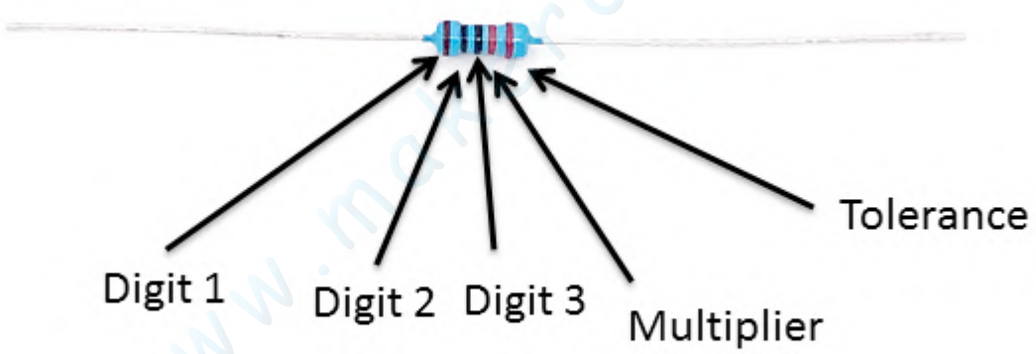
Resistors are colorful little devices. If you are trying to work out their value, you can just measure their resistance with a multimeter, or you can read their value from the colored stripes.
Each color has a number associated with it, as shown in table below.

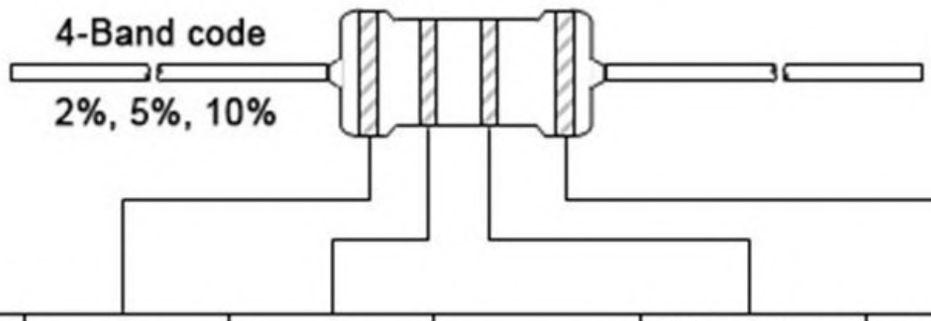
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9
Gold	1/10
Silver	1/100

Gold and silver, as well as representing the fractions 1/10 and 1/100, are also used to indicate how accurate the resistor is, so gold is +-5% and silver is +-10%.

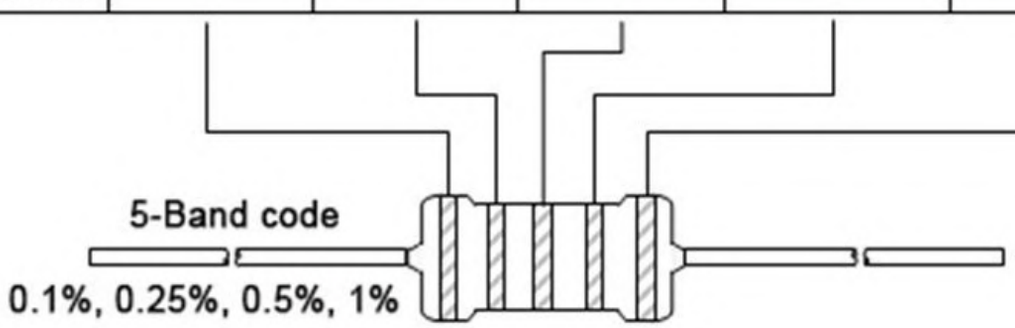
There will generally be four of these bands together starting at one end of the resistor, a gap, and then a single band at the other end of the resistor. The single band indicates the accuracy of the resistor value.

The figure below shows the arrangement of the colored bands. The resistor value uses just the three bands. The first band is the first digit, the second the second digit, the third is third digit, and the fourth "multiplier" band is how many zeros to put after the first two digits.

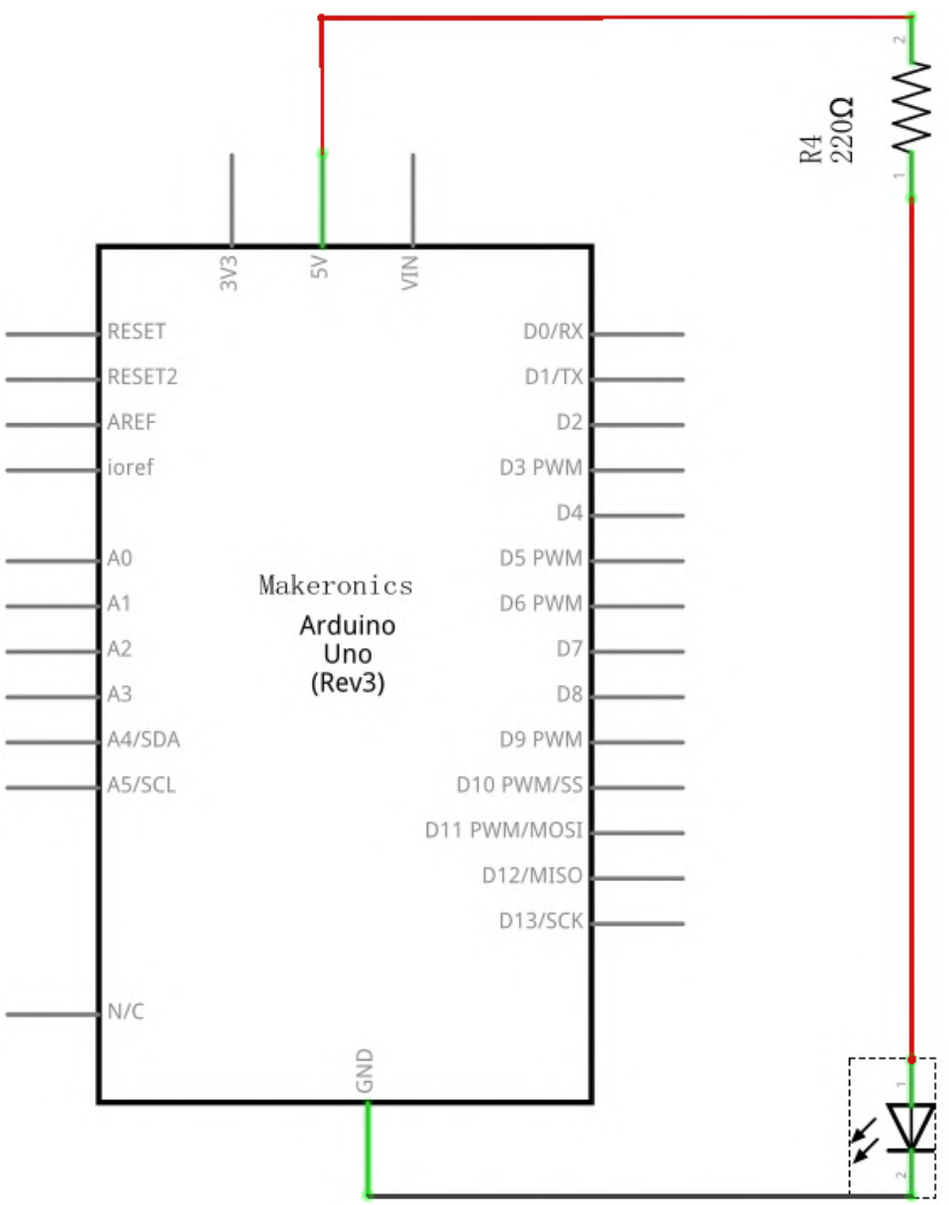




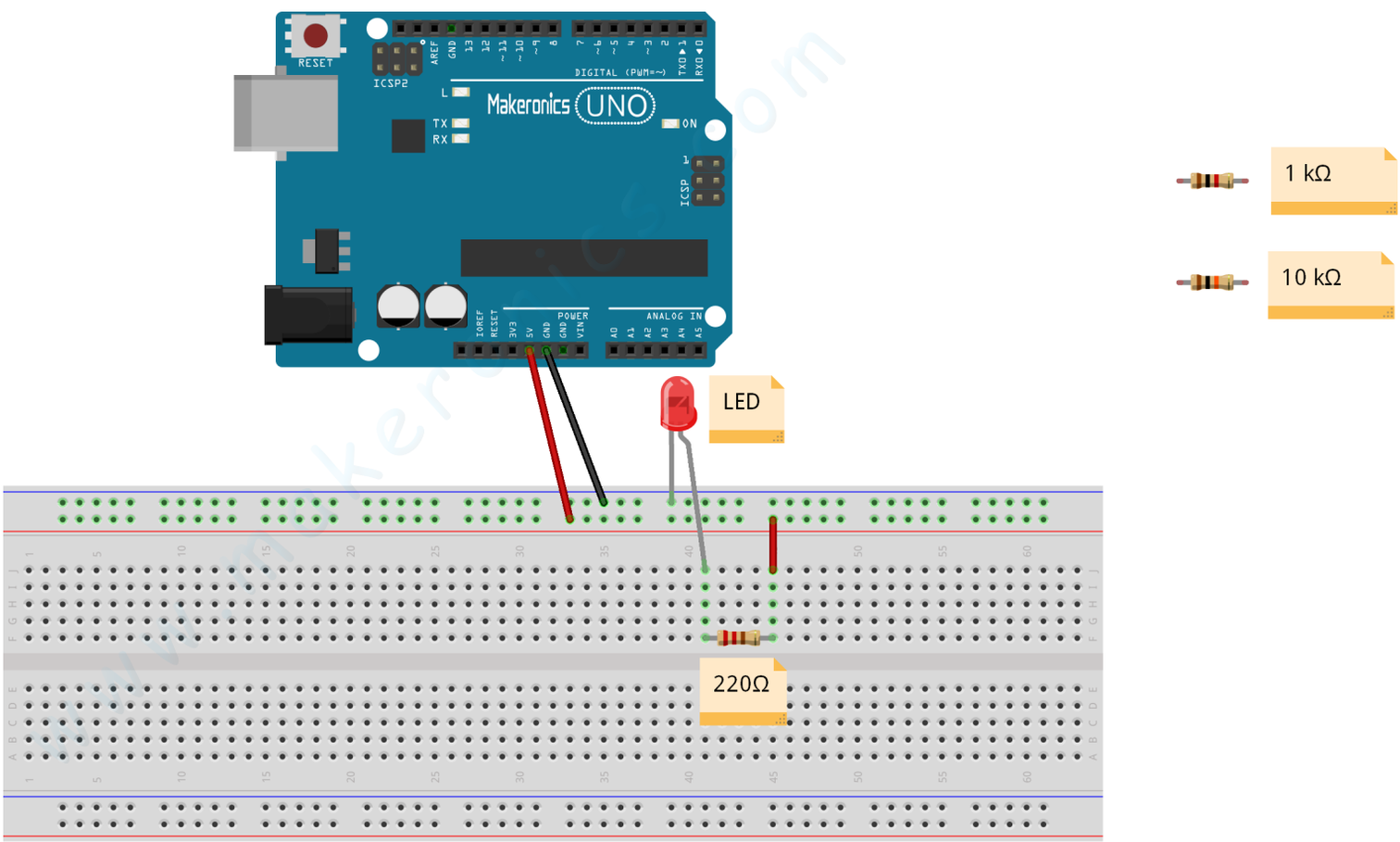
Color	1st Ring	2nd Ring	3rd Ring	4th Ring (Multipliter)	5th Ring (Tolerance)
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	±1%
Red	2	2	2	100Ω	±2%
Orange	3	3	3	1kΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100kΩ	±0.5%
Blue	6	6	6	1MΩ	±0.25%
Vlolet	7	7	7	10MΩ	±0.10%
Gray	8	8	8		±0.05%
White	9	9	9		
Gold					±5%
Silver					±10%
No Color					±20%



Connection Schematic:



Wiring diagram:

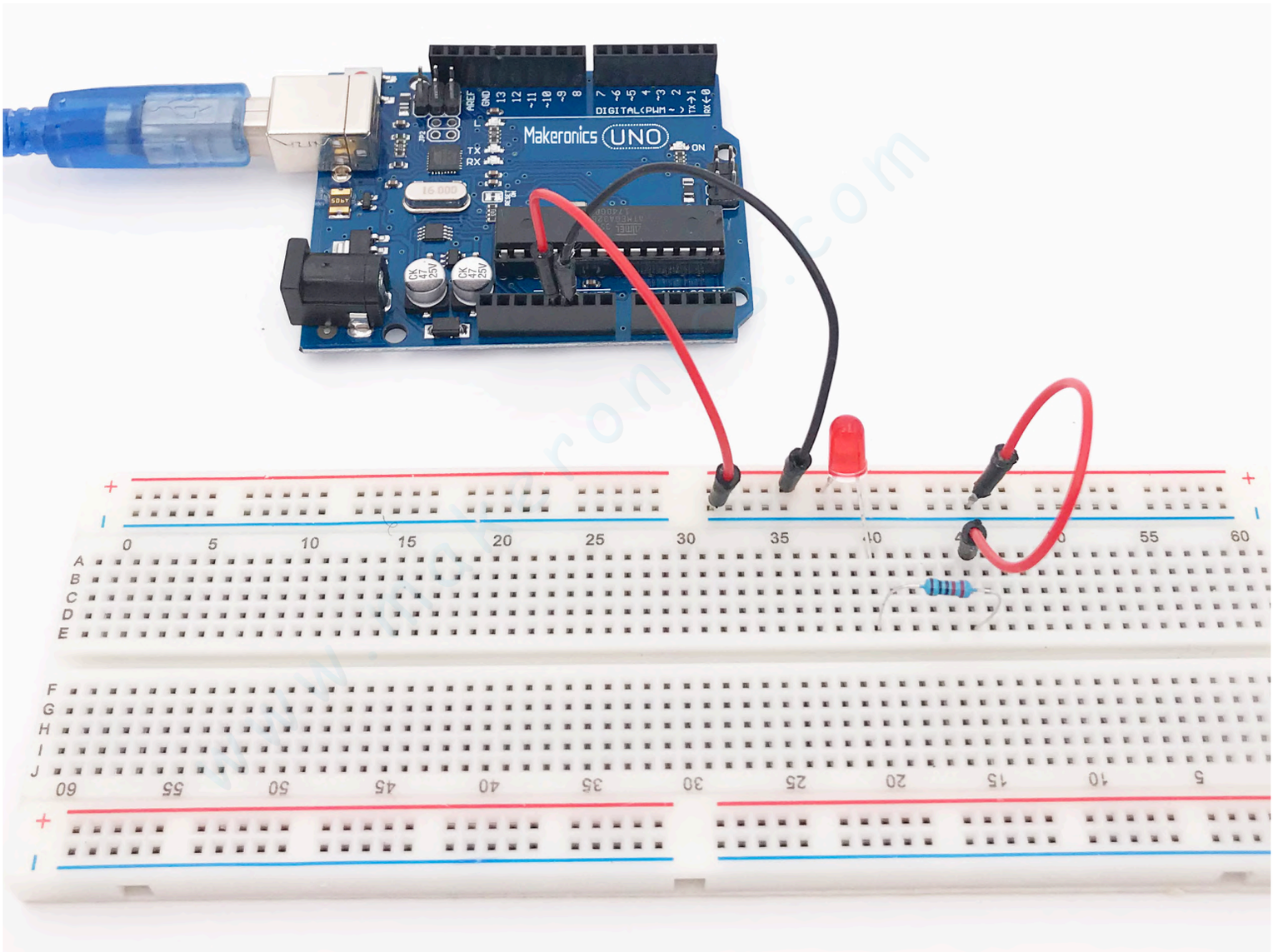


The UNO is a convenient source of 5 volts, which we will use to provide power to the LED and the resistor. You do not need to do anything with your UNO, except to plug it into a USB cable.

With the 220 Ω resistor in place, the LED should be quite bright. If you swap out the 220 Ω resistor for the 1kΩ resistor, then the LED will appear a little dimmer. Finally, with the 10 kΩ resistor in place, the LED will be just about visible. Pull the red jumper lead out of the breadboard and touch it into the hole and remove it, so that it acts like a switch. You should just be able to notice the difference.

At the moment, you have 5V going to one leg of the resistor, the other leg of the resistor going to the positive side of the LED and the other side of the LED going to GND. However, if we moved the resistor so that it came after the LED, as shown below, the LED will still light.
You will probably want to put the 220Ω resistor back in place. It does not matter which side of the LED we put the resistor, as long as it is there somewhere

Demo:


Makeronics UNO
ON
TX
RX
DIGITAL(PWM~)
0 5 10 15 20 25 30 35 40 55 60
A B C D E
F G H I J

Project 2

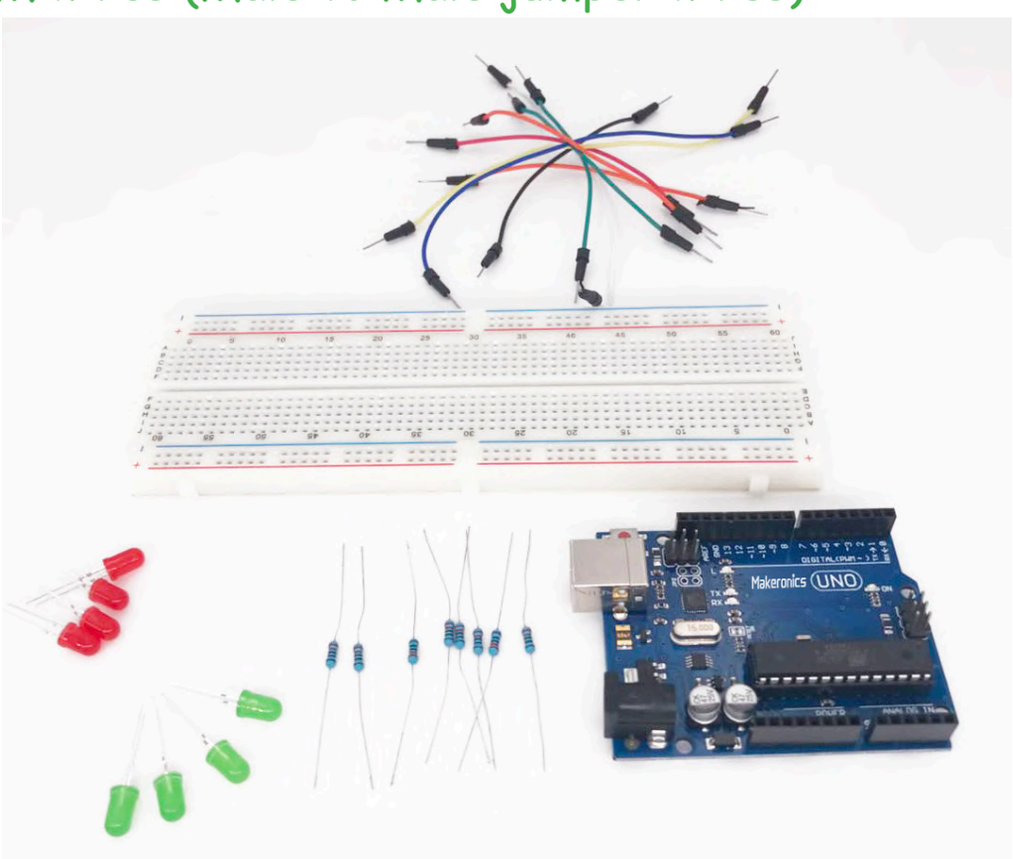
5

LED Light Bar

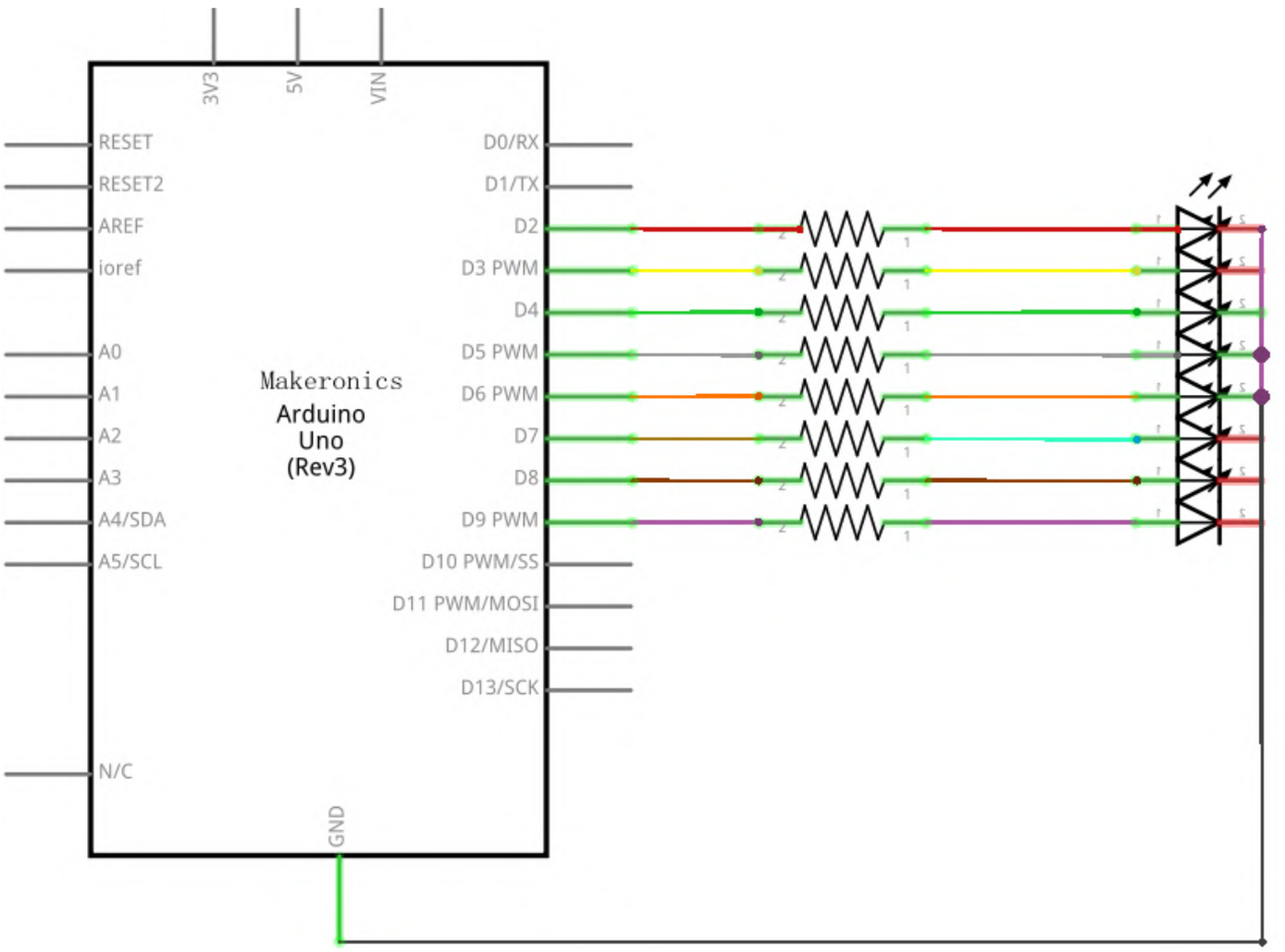
An LED bar graph is just a bunch of LEDs put together in a fancy case, but there are many users for it. We can display the date from a sensor, show a critical condition, or make a funny light show with it. In this lesson, you will learn how to light multiple LEDs.

Component Required:

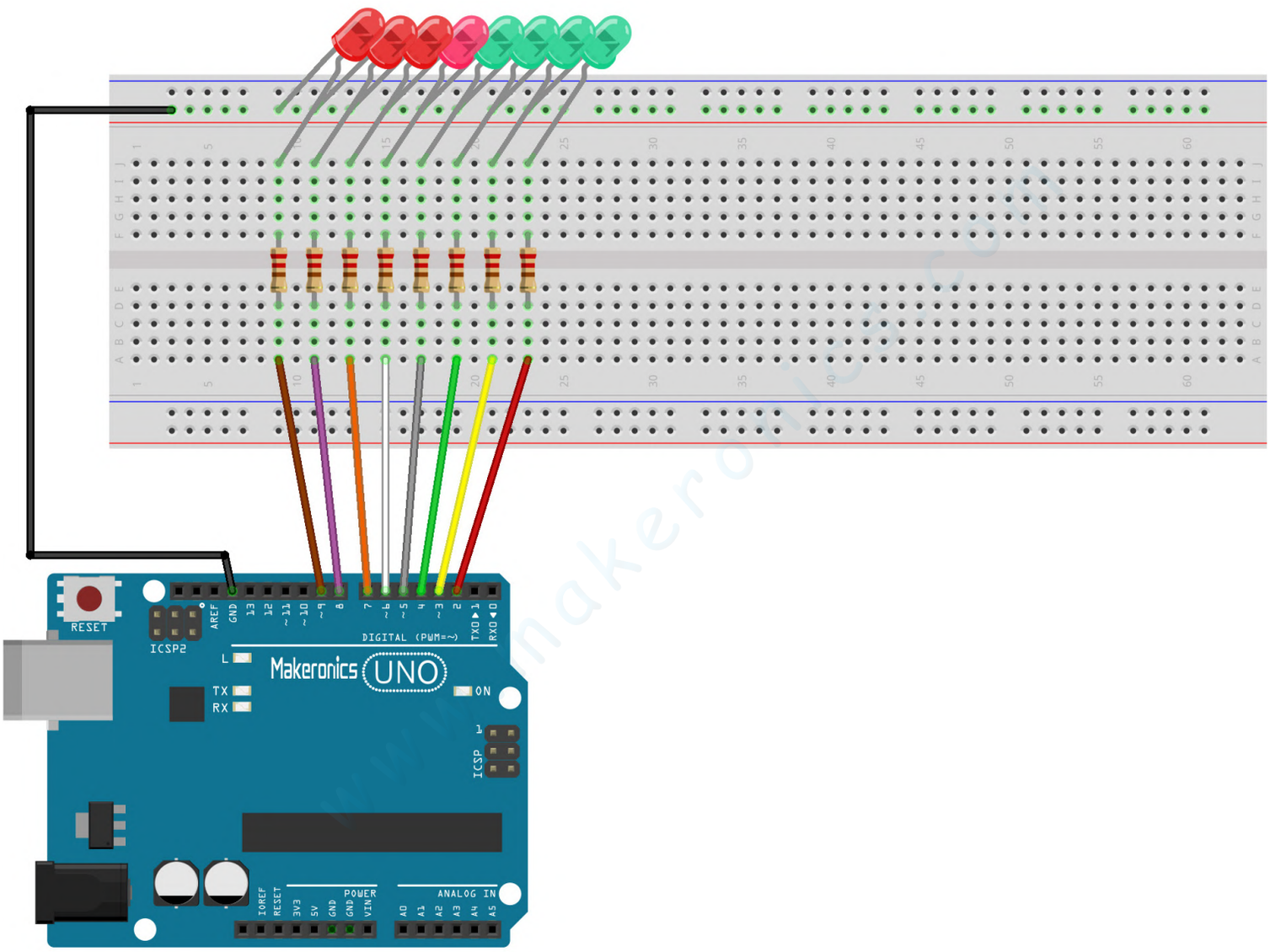
- 1 x Makeronics Uno R3
- 1 x 830 Tie Points Breadboard
- 4 x 5mm red LED
- 4 x 5mm green LED
- 8 x 220 ohm resistor
- 9 x M-M wires (Male to Male jumper wires)



Connection Schematic:



Wiring diagram:



Code:

In the setup() function, we set each LED pin as an output. This simple trick, used here, helps to set all the pins between pin1 and pin8 as outputs:

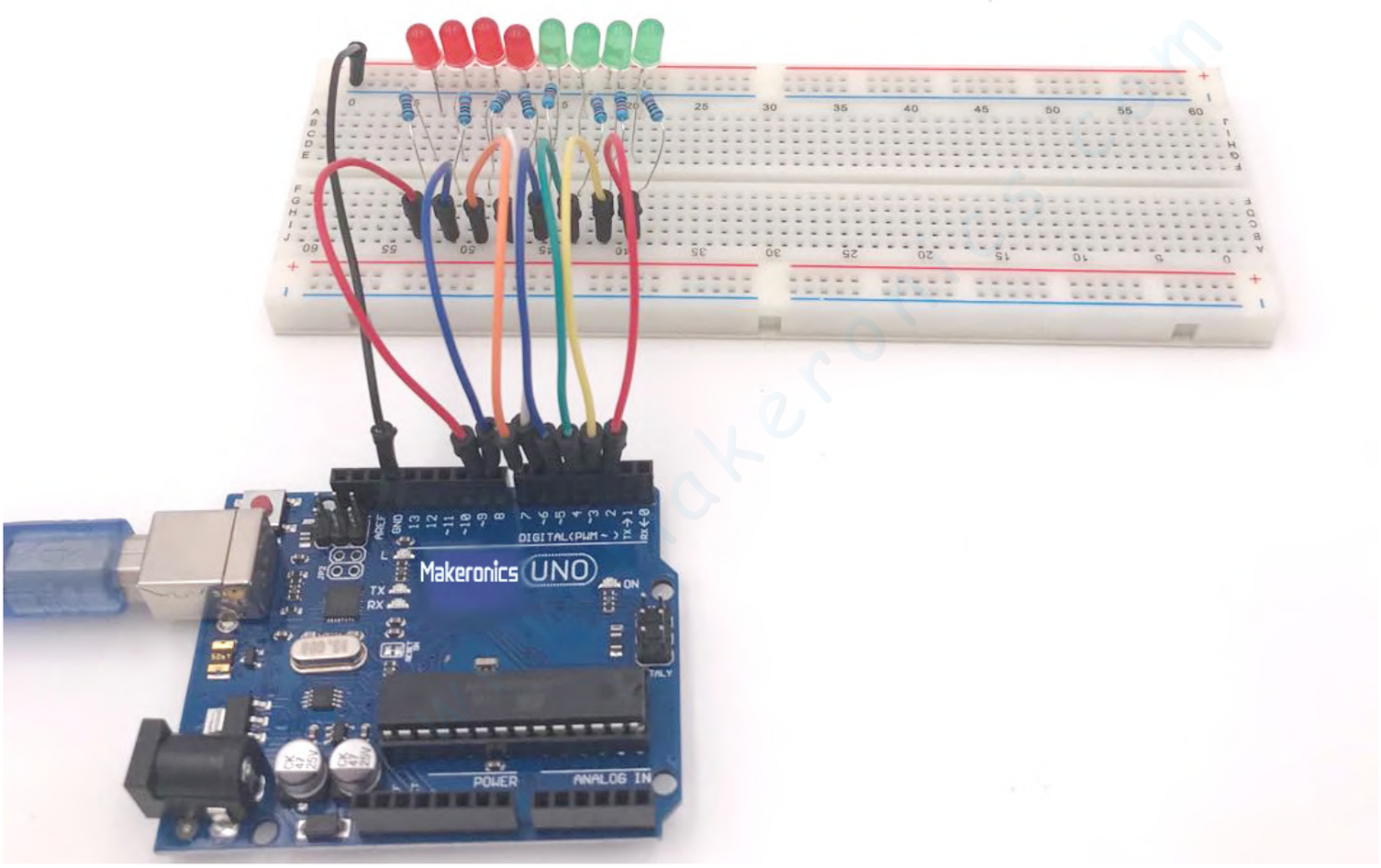
```
for (int i = 2; i < 9; i++) {
pinMode(i,OUTPUT);
}
```

In the custom allLEDsOff() function, it turns off all the LEDs at the same time.

```
void allLEDsOff(void) {
for (int i =2 ; i < 9; i++) {
digitalWrite(i, LOW);
}
}
```

This function is called in the loop cycle to turn the LEDs off, and then the LEDs are turned on one at a time -- with a 200-millisecond delay between each one --to create a sweeping effect. Another loop sends the sequence back the other way.

Demo:



Project 3

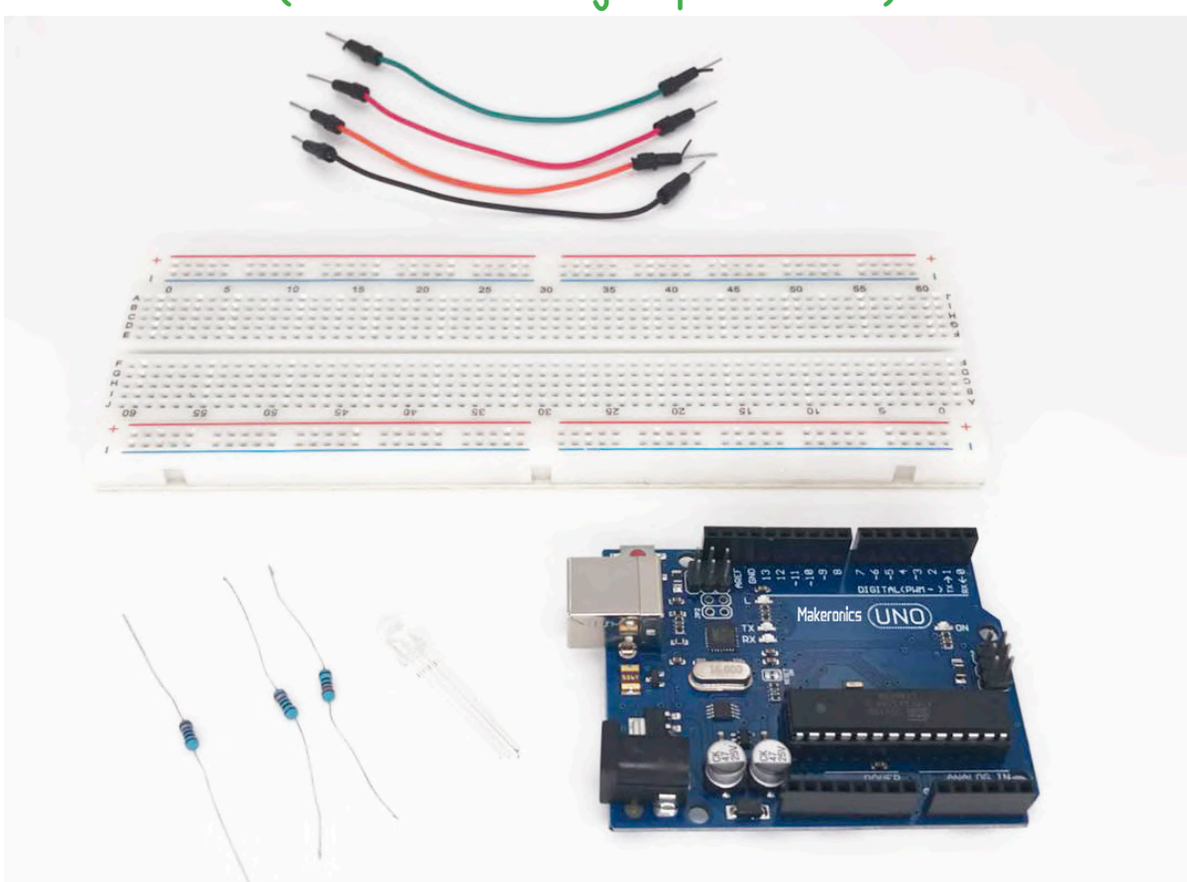
RGB LED

6

In this lesson, you will learn how to control RGB LED by using PWM. In our sketch, we will start with the LED in the Red color state, then fade to Green, then fade to Blue and finally back to the Red color. By doing this we will cycle through most of the color that can be achieved.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 Tie Points Breadboard
- 1 x RGB LED
- 3 x 220 ohm resistors
- 4 x M-M wires (Male to Male jumper wires)



Component Introduction:

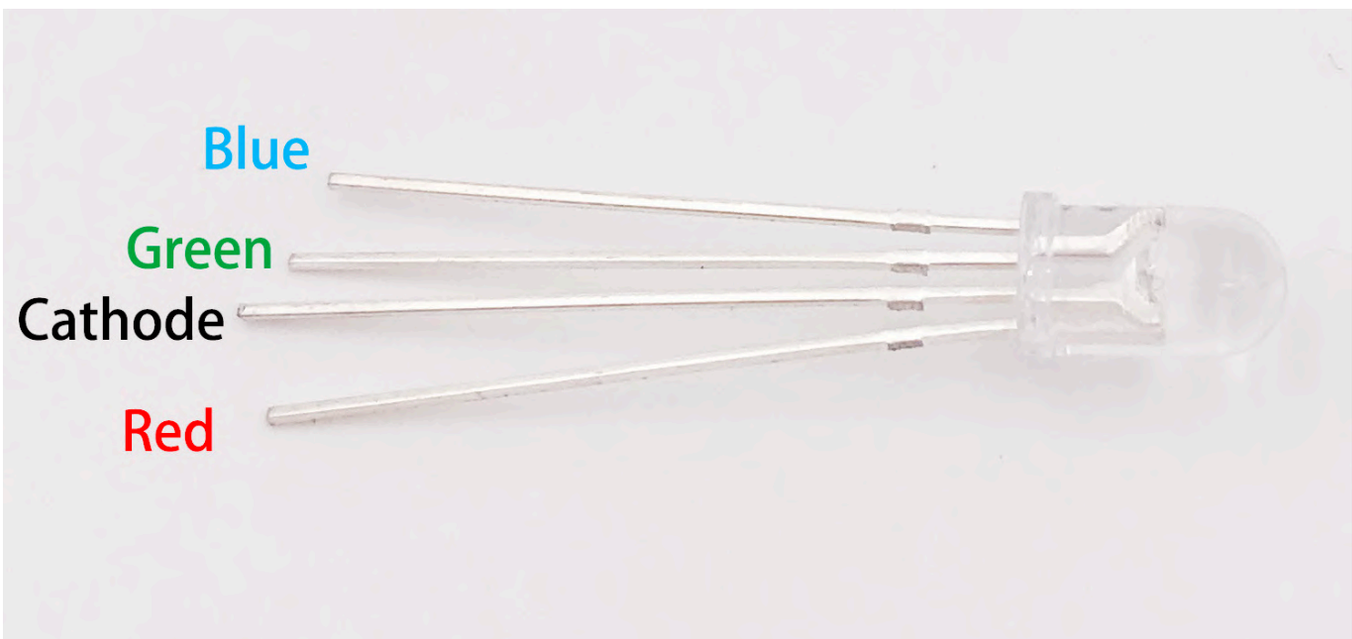
RGB LED:

Red, green, and blue LEDs (RGB LEDs) are a single LED case that contains three actual LEDs. The three LEDs are red, green, and blue. By using PWM to control the brightness to each of the LED colors, you can make the LED appear any color.

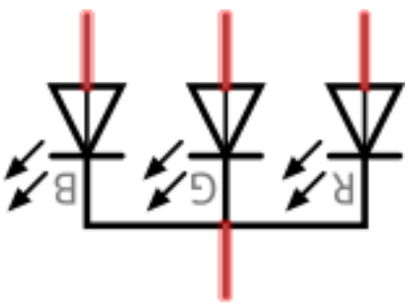
Although an RGB LED contains three normal, two-pin LEDs, this does not mean that the LED body has to have six pins, because one terminal of each LED can be connected together as a common pin.

If the negative connections to each LED are connected together, the resulting lead is called a common cathode and if the positive connections are common, the lead is called a common anode. The RGB LED that provided by this starterkit is common cathode.

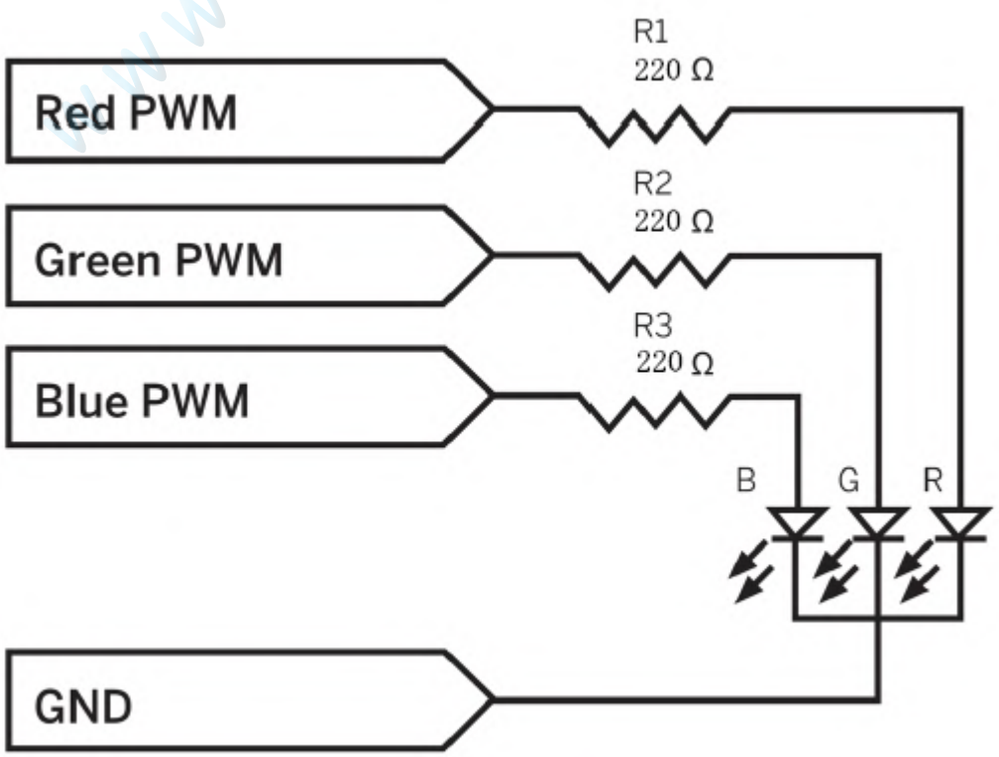
If you look closely at the LED in Figure below, you'll notice that the legs are all different lengths. With regular LEDs, the short leg is the negative leg, but with the RGB LED, the longest leg is the negative leg.



The circuit diagram for this component is usually drawn like the Figure below. Notice that it shows three separate LEDs connected together, and they each share a single negative connection.

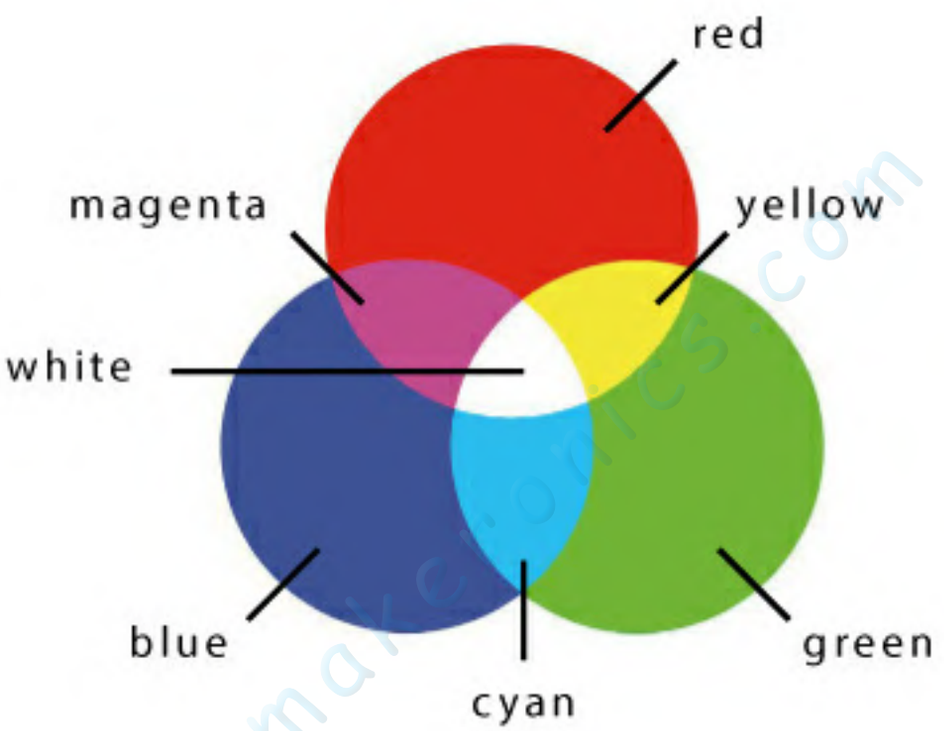


Keeping in mind which positive leg corresponds to which color, you can wire this LED into a circuit just like you would three separate LEDs. Just connect the positive leg(s) you want to use to power or to an Arduino output pin through a current-limiting resistor, and connect the common cathode to ground.



COLOR:

RGB LEDs are cool because you can use them to create a slew of colors. Red, green, and blue are the primary colors in the additive color scheme, and the LED can mix these colors to create light in other colors. The additive color wheel in figure below shows how primary colors can combine to create any color in the rainbow.

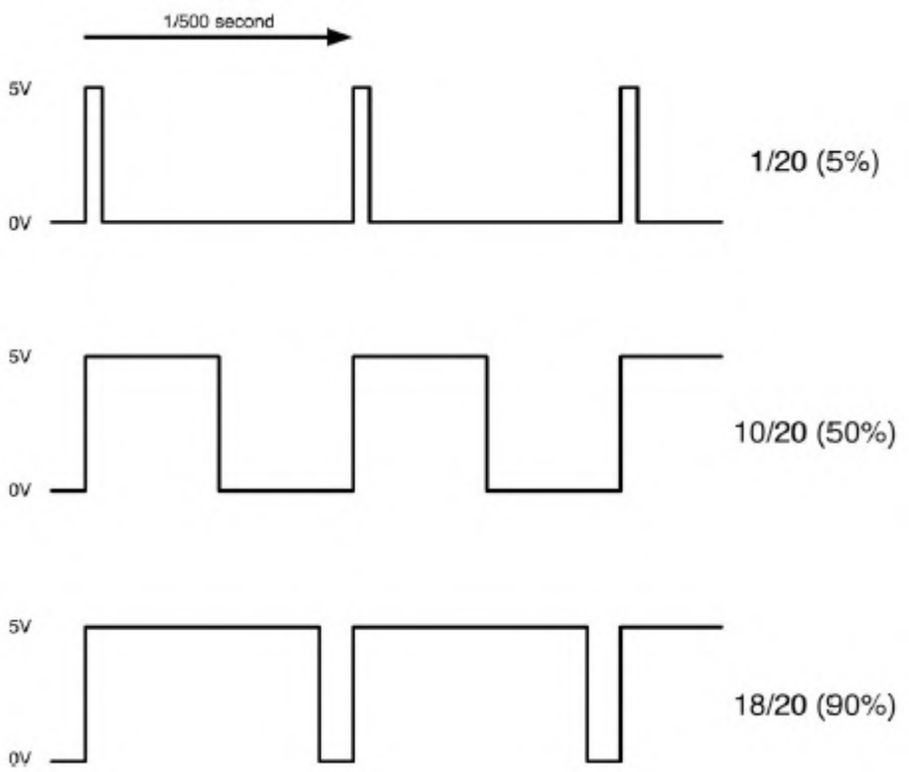


With your RGB LED, if you turn on the blue LED and the red LED together, you get magenta light. Combine the red and green LEDs, and you get yellow. If all the LEDs are on, you get white light. This concept is the foundation for how an LED TV or monitor works: each pixel on your screen is essentially an RGB LED.

Theory (PWM):

If you try controlling the brightness of an LED by adjusting the voltage across it, it generally doesn't work so well, because there is a big dead-zone until the voltage gets high enough for the LED to emit any light at all.

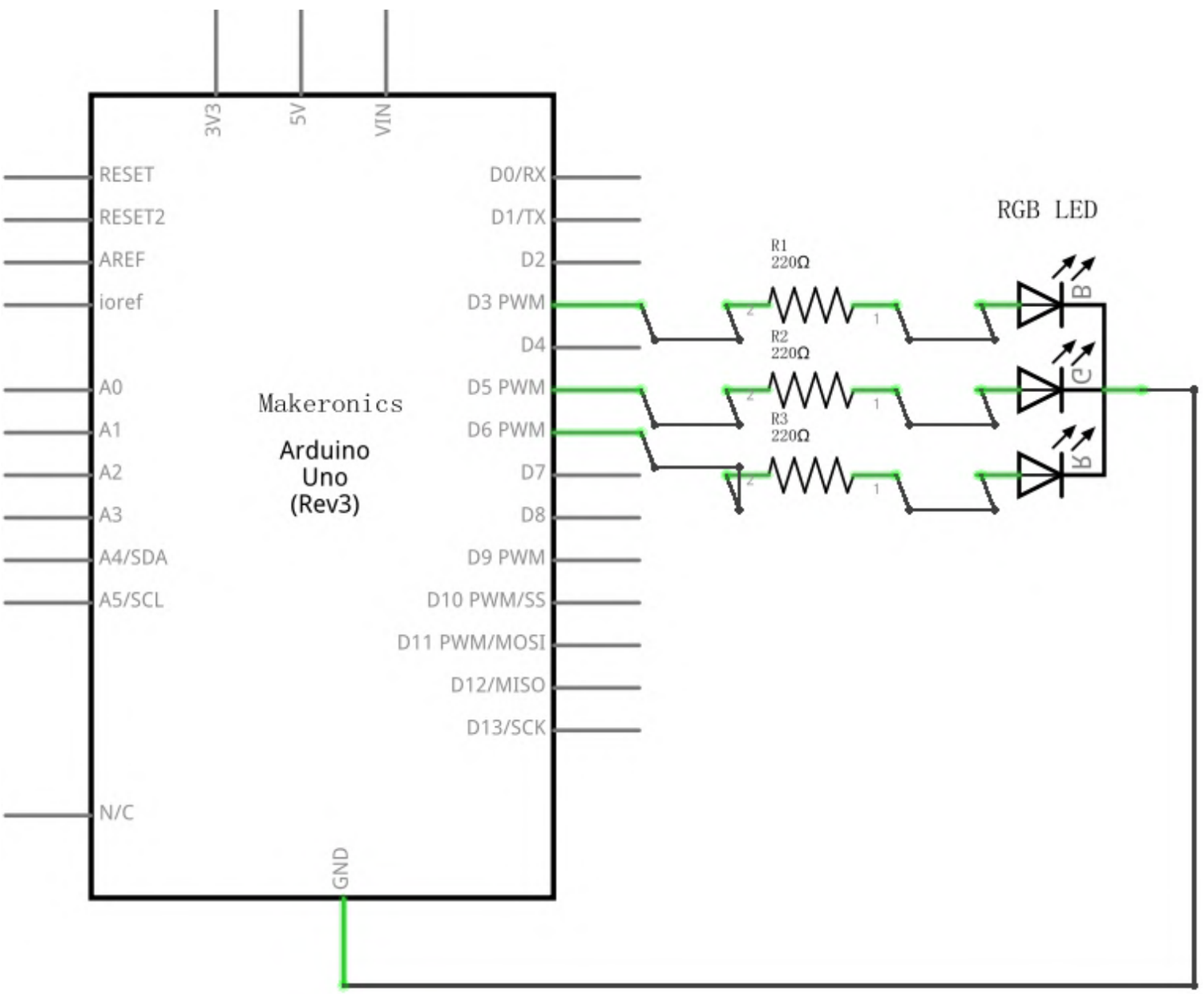
PWM analog outputs are perfect for controlling the brightness of the LED. LEDs can turn on and off very quickly, certainly less than a millionth of a second, so when using PWM with LEDs, they are actually flashing at the PWM frequency, but the eye sees them as on but with a brightness that varies as the proportion of the time that the LED is actually on changes.



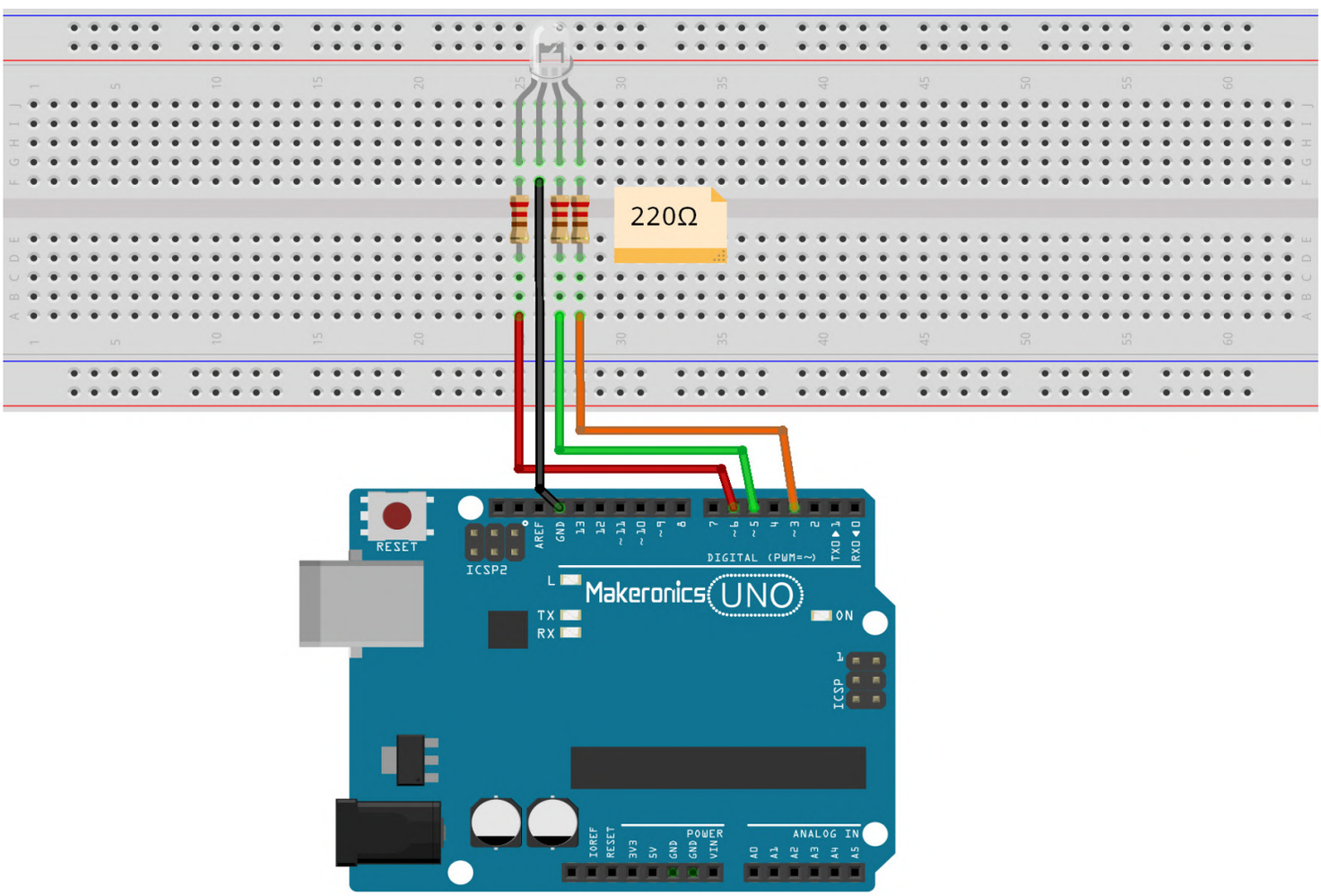
The proportion of the time that the pulses are high is called the duty cycle. If the pulses are only on for 5% of the time (duty cycle of 5%), then not much energy will be arriving at the LED, so the LED will be dim Increase the duty cycle to 50% and the LED will receive half power and probably brightness. If the duty cycle is increased to 90%, LED fully lit. Turning LED off is just a matter of setting the duty cycle to 0 and fully lit setting it to 100%. Setting the duty cycle to 100% will leave the Arduino pin high all the time.

The Arduino has the ability to do PWM on their output pins. On an Arduino Uno, this is restricted to pins D3, D5, D6, D9, D10, and D11. These pins are marked with a ~ sign next to the pin on the Arduino itself. The frequency of these pulses can be varied on Arduino. On an Arduino Uno, it is 490Hz (pulses per second) for most pins except pins 5 and 6, which run at 980Hz. For controlling the brightness of an LED, the default Arduino PWM frequency of 490Hz works just fine.

Connection Schematic:



Wiring diagram:



Code:

After wiring, please open the program in the code folder- Lesson 6 RGB LED, and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Our code will use FOR loops to cycle through the colors.
The first FOR loop will go from RED to GREEN.
The second FOR loop will go from GREEN to BLUE.
The last FOR loop will go from BLUE to RED.
Try the sketch out and then we will dissect it in some detail.....

The sketch starts by specifying which pins are going to be used for each of the colors:

```
// Define Pins
#define BLUE 3
#define GREEN 5
#define RED 6
```

The next step is to write the 'setup' function. As we have learnt in earlier lessons, the setup function runs just once after the Arduino has reset. In this case, all it has to do is define the three pins we are using as being outputs.

```
void setup()
{
pinMode(RED, OUTPUT);
pinMode(GREEN, OUTPUT);
pinMode(BLUE, OUTPUT);
digitalWrite(RED, HIGH);
digitalWrite(GREEN, LOW);
digitalWrite(BLUE, LOW);
}
```

Before we take a look at the 'loop' function, let's look at the last function in the sketch.

The define variables

```
redValue = 255; // choose a value between 1 and 255 to change the color.
```

```
greenValue = 0;
blueValue = 0;
```

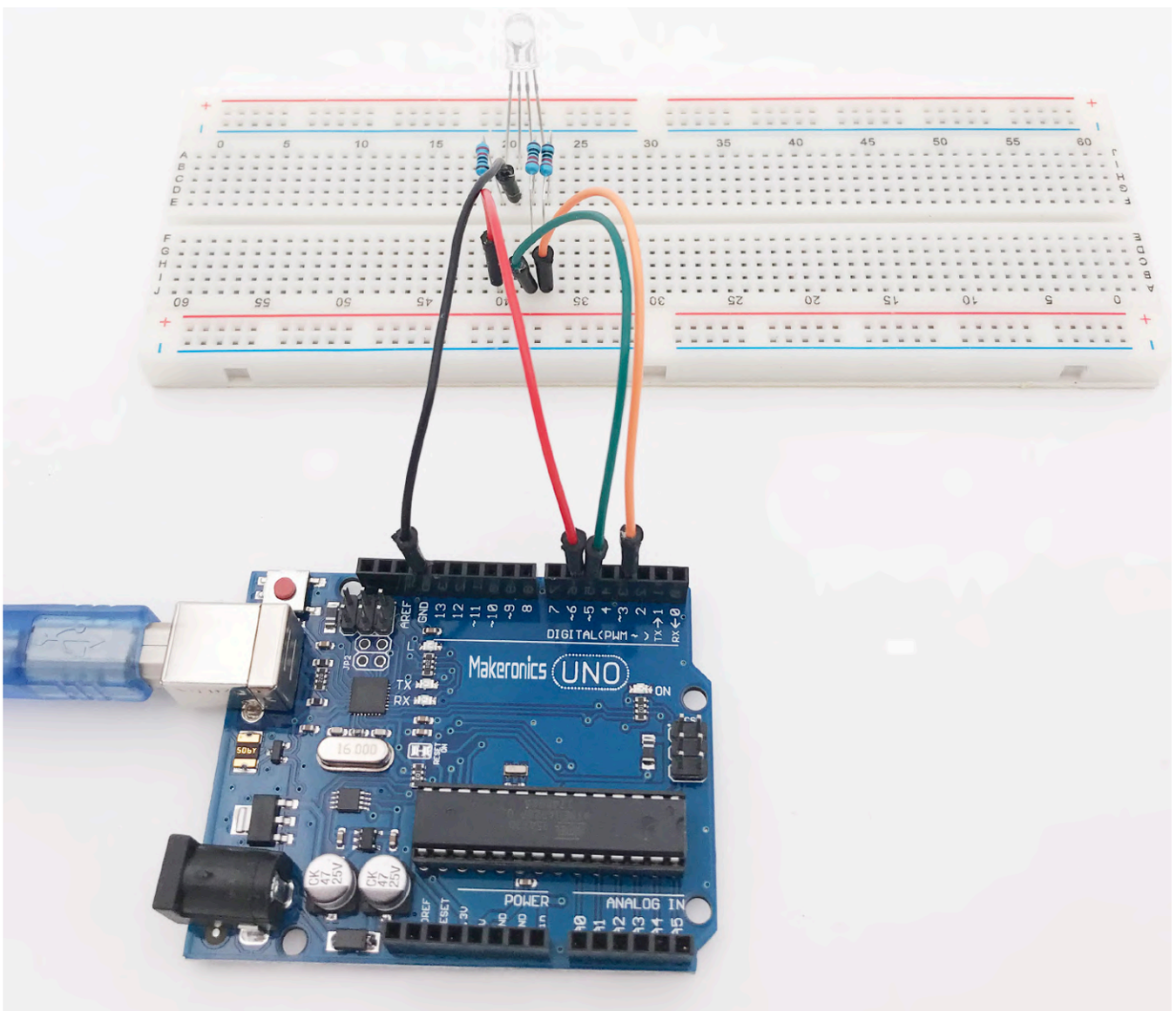
This function takes three arguments, one for the brightness of the red, green and blue LEDs. In each case the number will be in the range 0 to 255, where 0 means off and 255 means maximum brightness. The function then calls 'analogWrite' to set the brightness of each LED.

If you look at the 'loop' function you can see that we are setting the amount of red, green and blue light that we want to display and then pausing for a second before moving on to the next color.

```
#define delayTime 10 // fading time between colors
Delay(delayTime);
```

Try adding a few colors of your own to the sketch and watch the effect on your LED.

Demo:



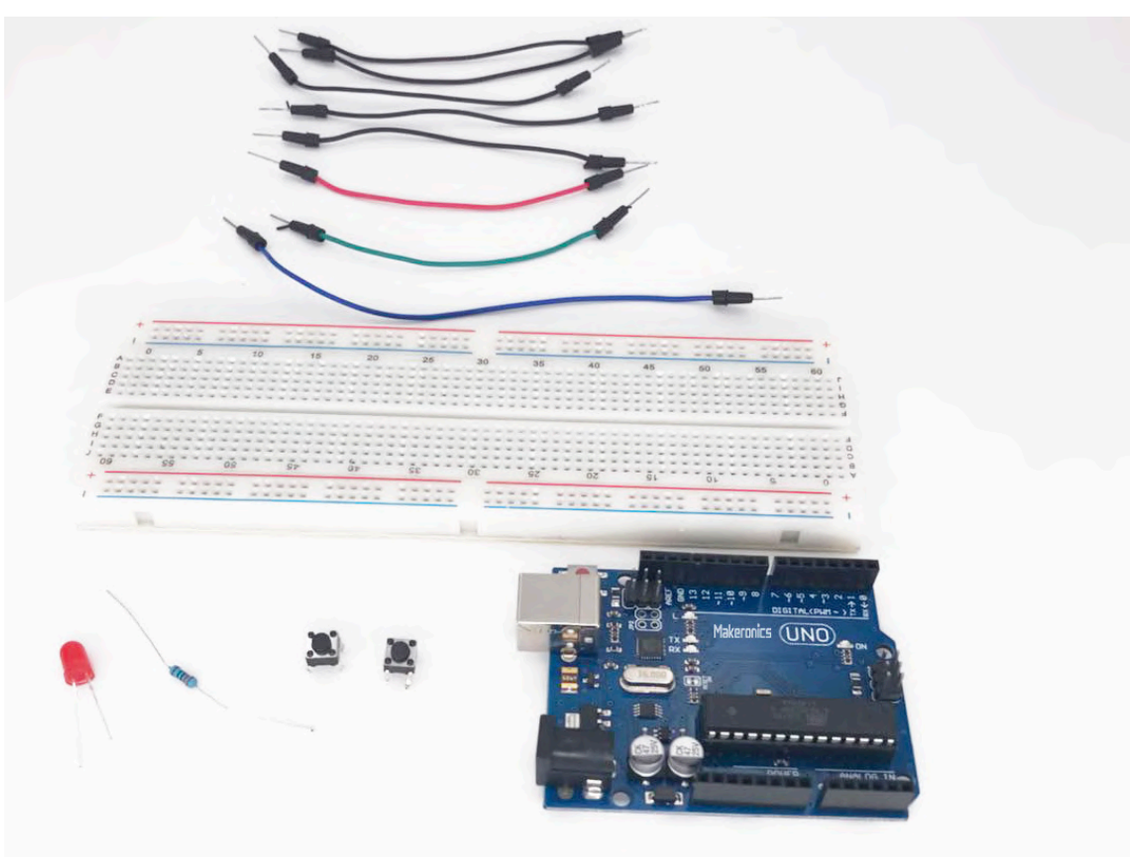
Project 4

7 Digital Inputs

In this lesson, you will learn to use push buttons with digital inputs to turn an LED on and off.
Pressing the button will turn the LED on; pressing the other button will turn the LED off.

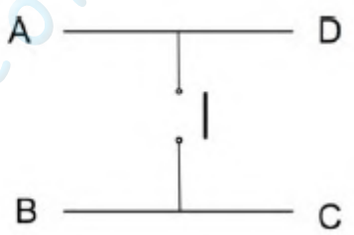
Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 Tie-points Breadboard
- 1 x 5mm green LED
- 1 x 220 ohm resistor
- 2 x pushbuttons
- 8 x M-M wires (Male to Male jumper wires)



Component Introduction: pushbuttons:

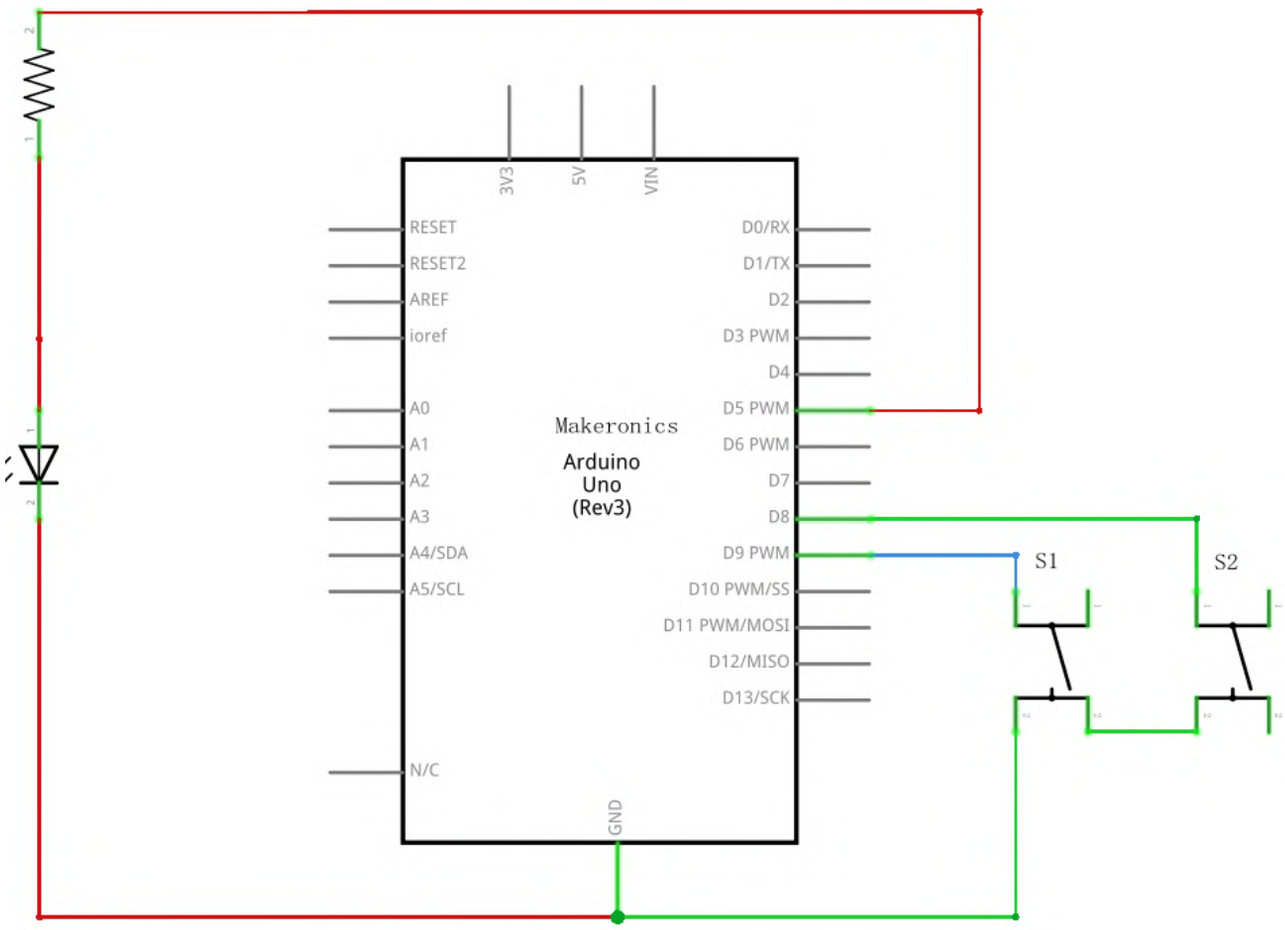
When pressed, a pushbutton completes a circuit, turning it on. As soon as the button is released, the connection will spring back and break that circuit, turning it off. The pushbutton switch is also known as a momentary or normally open switch, and is used in, for example, computer keyboards.



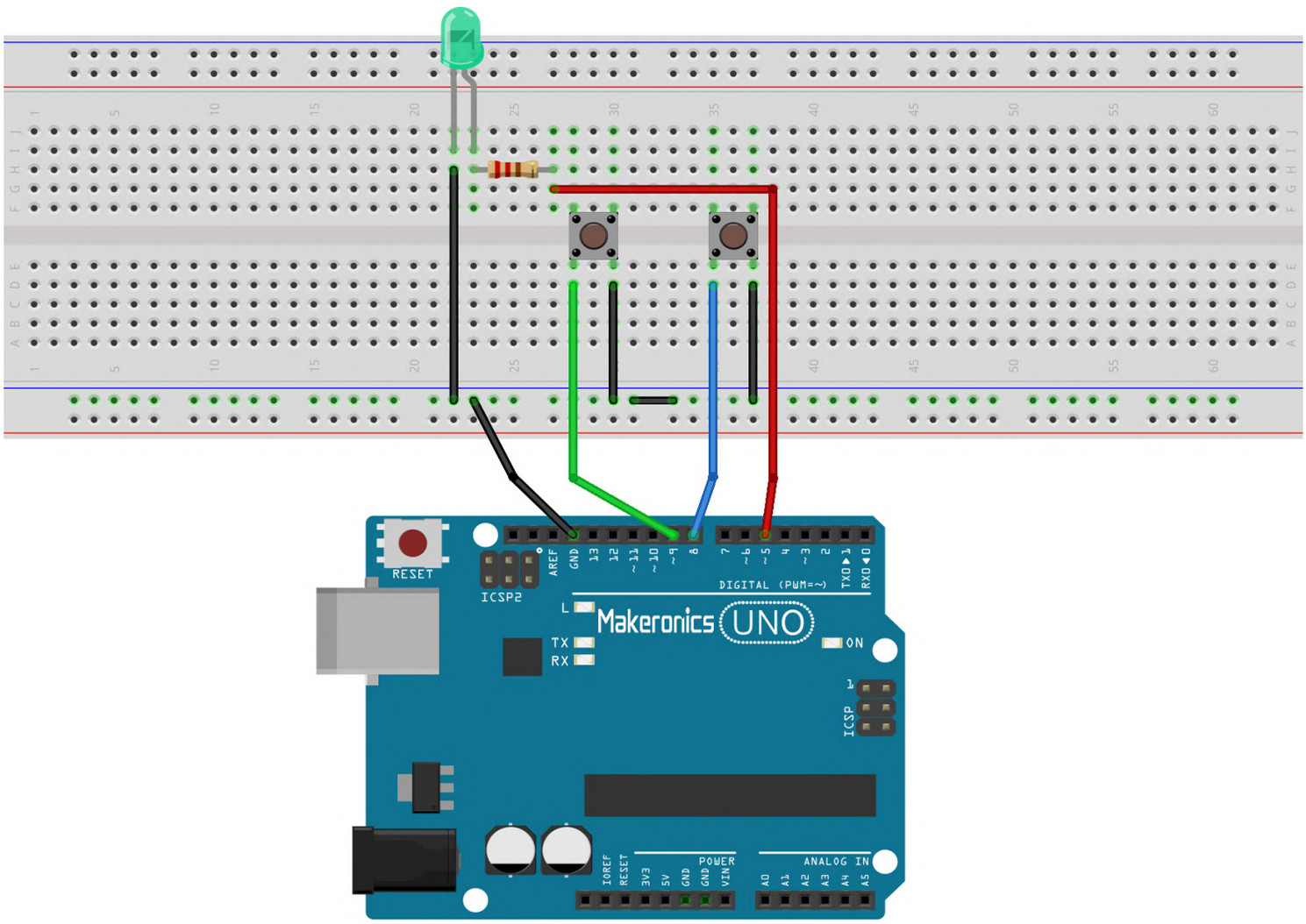
This is in contrast to a toggle switch, which stays either on or off until you toggle it to the other position, like a light switch.

This type of pushbutton has four pins, but you generally use only two at a time for connection. You'll use the top connections in this project, although the two unused pins at the bottom would do the same job. As Figure above shows, the pins work in a circuit. Pins A and D are always connected, as are pins B and C. When the button is pressed, the circuit is complete.

Connection Schematic:



Wiring diagram:



Note:
Please use one dupont wire to bridge the ground rows of power bus strip3 and power bus strip 4.

Code:

After wiring, please open program in the code folder- Lesson 7 Digital Inputs, and press UPLOAD to upload the program. If errors are prompted, see Lesson 2 for details about the tutorial on program upload.

Load the sketch onto your UNO R3 board. Pressing the left button will turn the LED on while pressing the right button will turn it off.

The first part of the sketch defines three variables for the three pins that are to be used. The 'ledPin' is the output pin and 'buttonApin' will refer to the left button on the breadboard and 'buttonBpin' to the other button.
The 'setup' function defines the ledPin as being an OUTPUT as normal, but now we have the two inputs to deal with. In this case, we use the set the pinMode to be 'INPUT_PULLUP' like this:

```
pinMode(buttonApin, INPUT_PULLUP);
pinMode(buttonBpin, INPUT_PULLUP);
```

The pin mode of INPUT_PULLUP means that the pin is to be used as an input, but that if nothing else is connected to the input, it should be 'pulled up' to HIGH. In other words, the default value for the input is HIGH, unless it is pulled LOW by the action of pressing the button.

This is why the button are connected to GND. When a button is pressed, it connects the input pin to GND, so that it is no longer HIGH.

Since the input is normally HIGH and only goes LOW when the button is pressed, the logic is a little upside down. We will handle this in the 'loop' function.

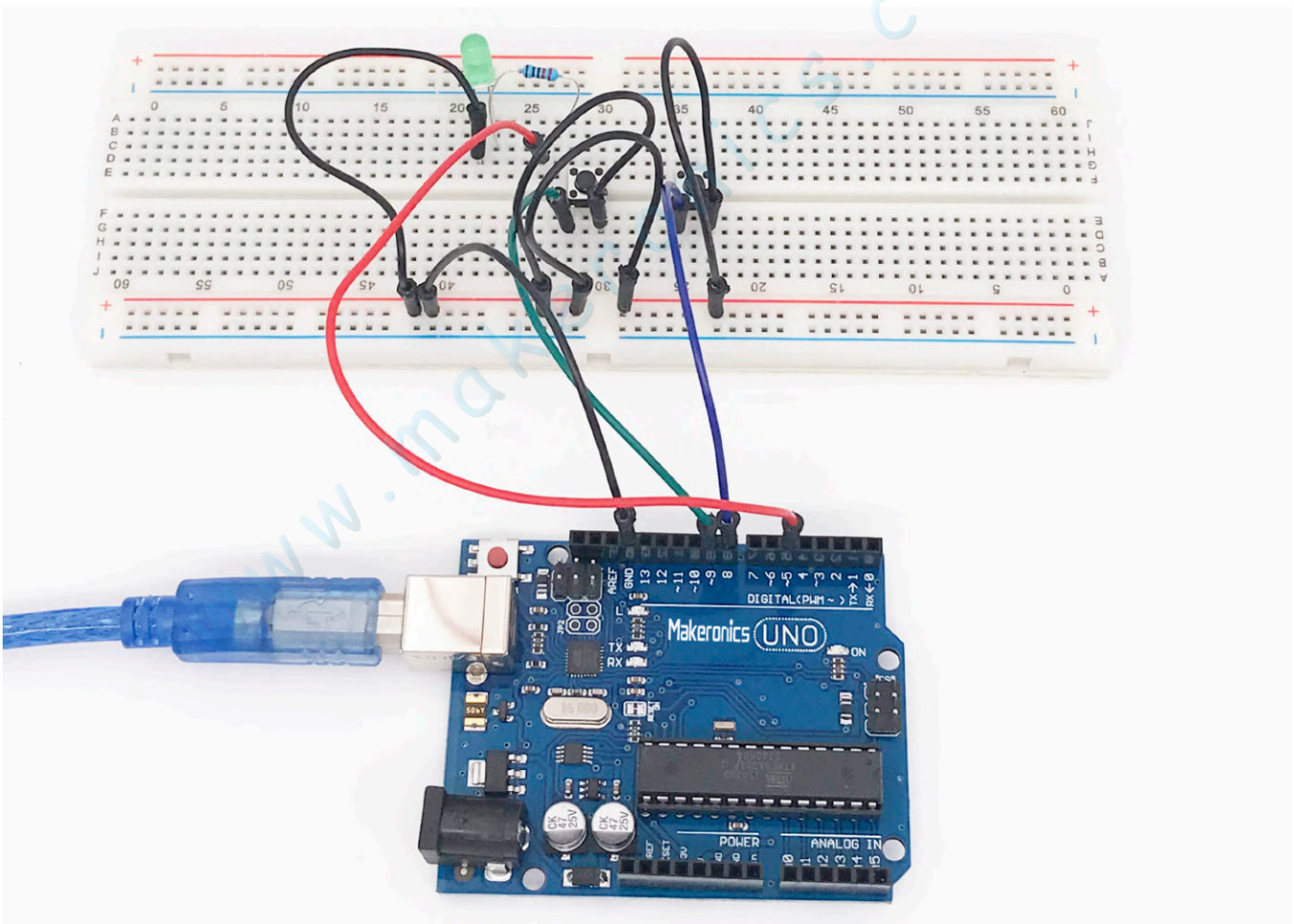
```
void loop()
{
if (digitalRead(buttonApin) == LOW)
{
digitalWrite(ledPin, HIGH);
}
if (digitalRead(buttonBpin) == LOW)
{
digitalWrite(ledPin, LOW);
}
}
```

In the 'loop' function there are two 'if' statements. One for each button. Each does an 'digitalRead' on the appropriate input.

Remember that if the button is pressed, the corresponding input will be LOW, if button A is low, then a 'digitalWrite' on the ledPin turns it on.

Similarly, if button B is pressed, a LOW is written to the ledPin.

Demo:



Project 5

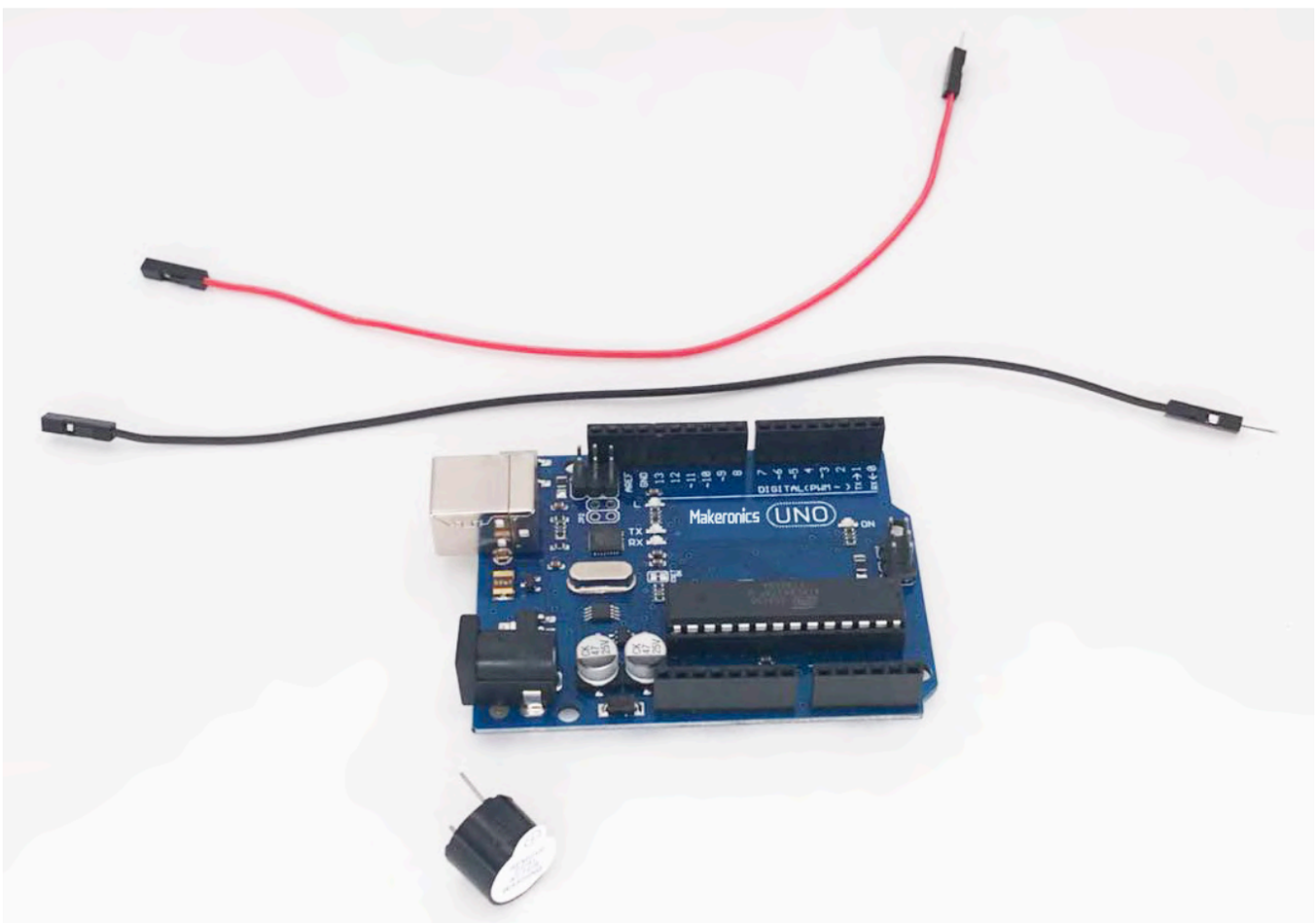
8

Active Buzzer

In this lesson, you will learn how to generate a sound with an active buzzer.

Component Required:

- 1 x Makeronics Uno R3
- 1 x Active buzzer
- 2 x F-M wires (Female to Male DuPont wires)



Component Introduction

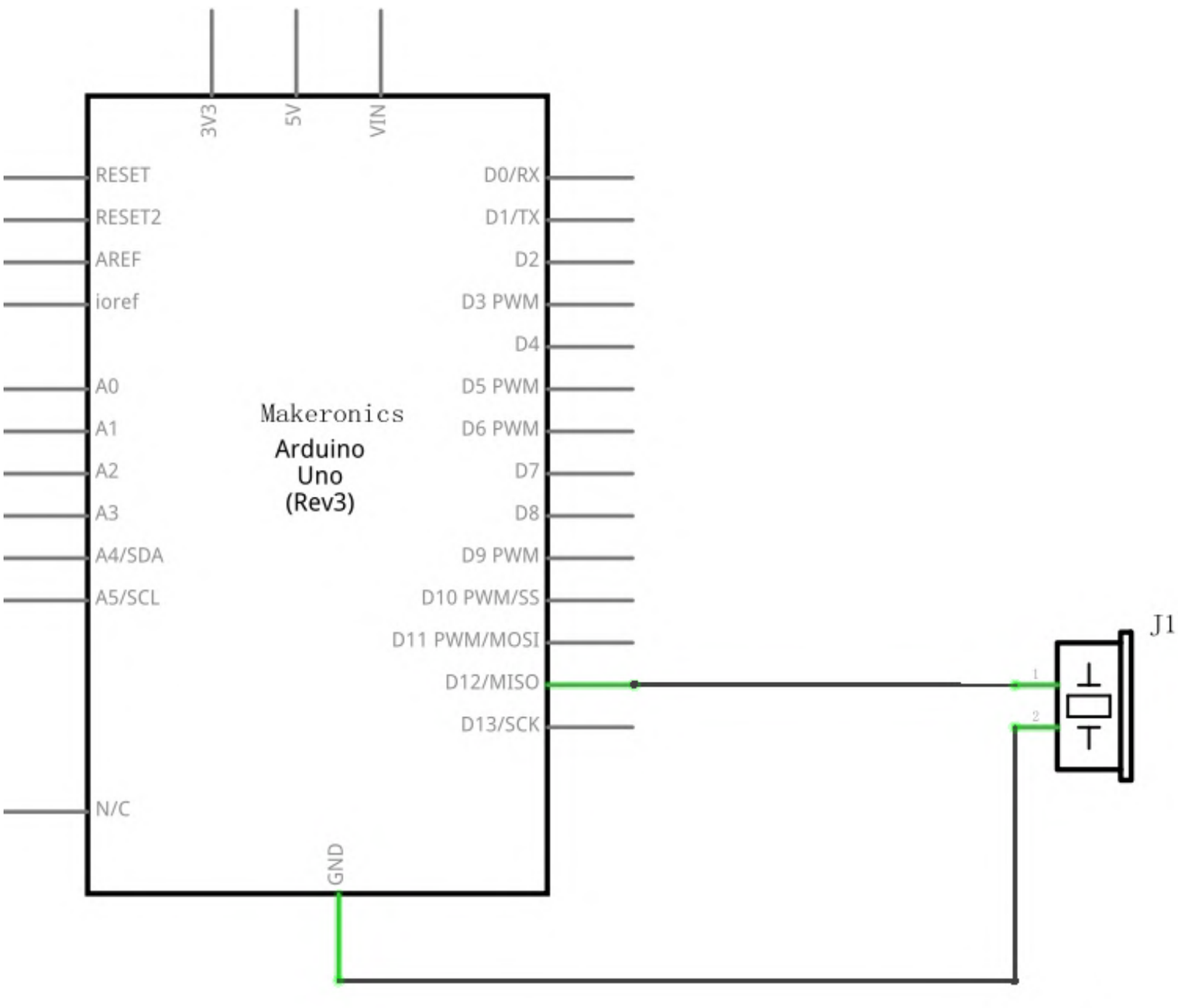
BUZZER:

The buzzer is similar to a speaker and produces an audible click when you apply a voltage to the two leads; these clicks happen very fast, several hundreds or even thousands of times per second, and their frequency creates a tone.

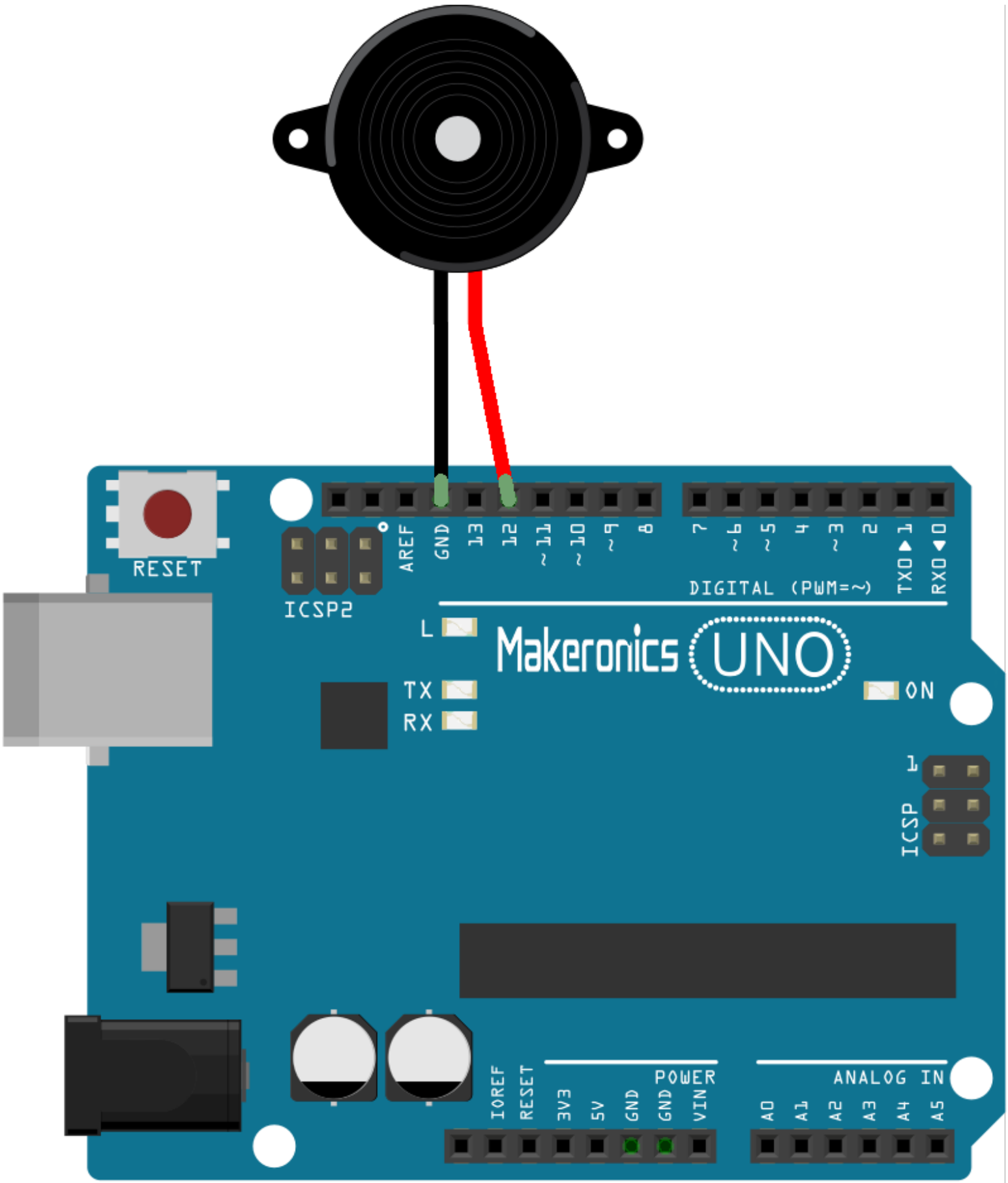
Buzzers can be categorized as active and passive ones. Turn the pins of two buzzers face up. The one with a green circuit board is a passive buzzer, while the other enclosed with a black tape is an active one.

An active buzzer will generate a tone using an internal oscillator, so all that is needed is a DC voltage. A passive buzzer requires an AC signal or PWM to make a sound. It is like an electromagnetic speaker, where a changing input signal produces the sound, rather than producing a tone automatically.

Connection Schematic:



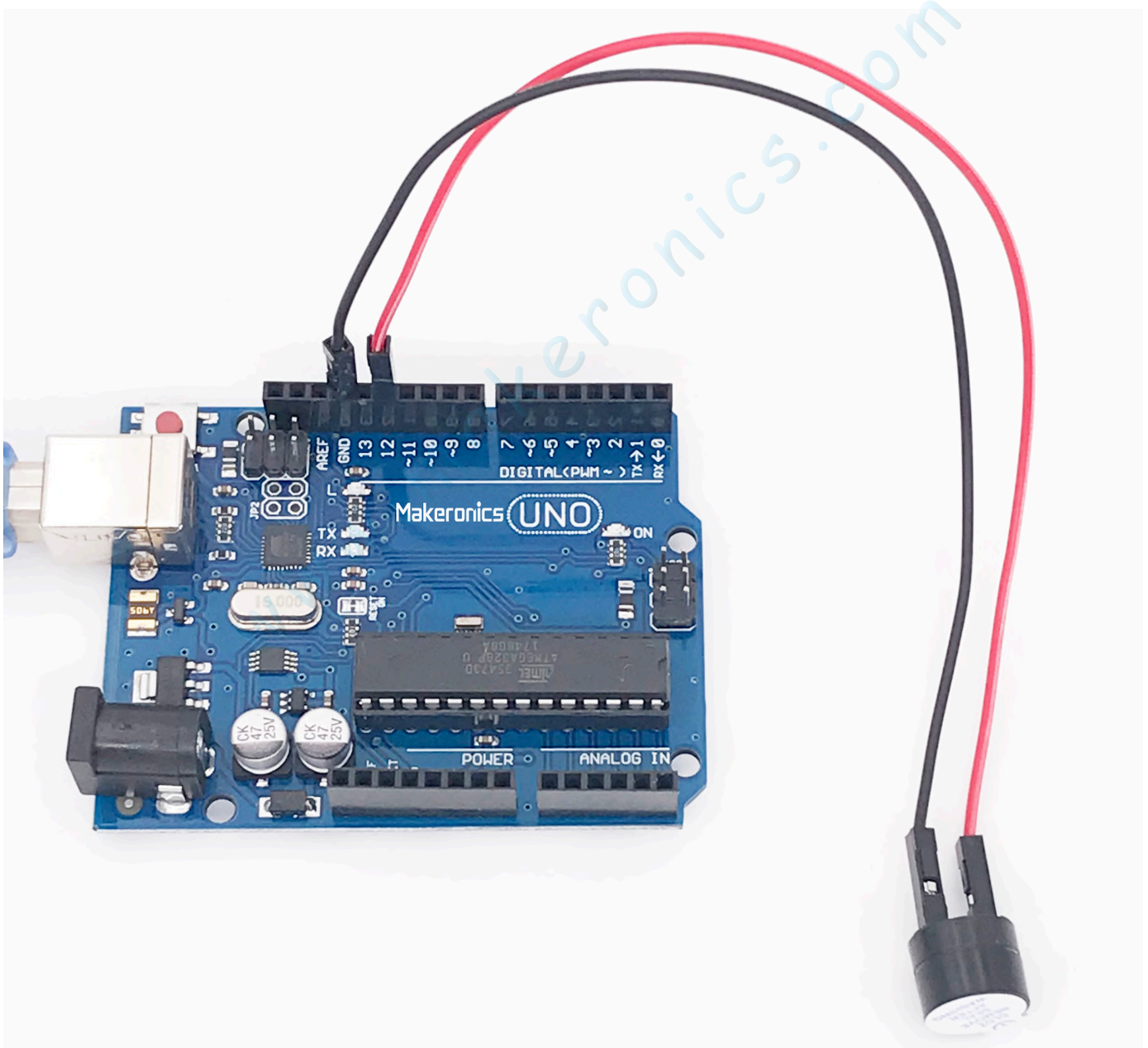
Wiring diagram:



Code:

After wiring, please open the program in the code folder-Lesson 8 Making Sounds and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Demo:



Project 6

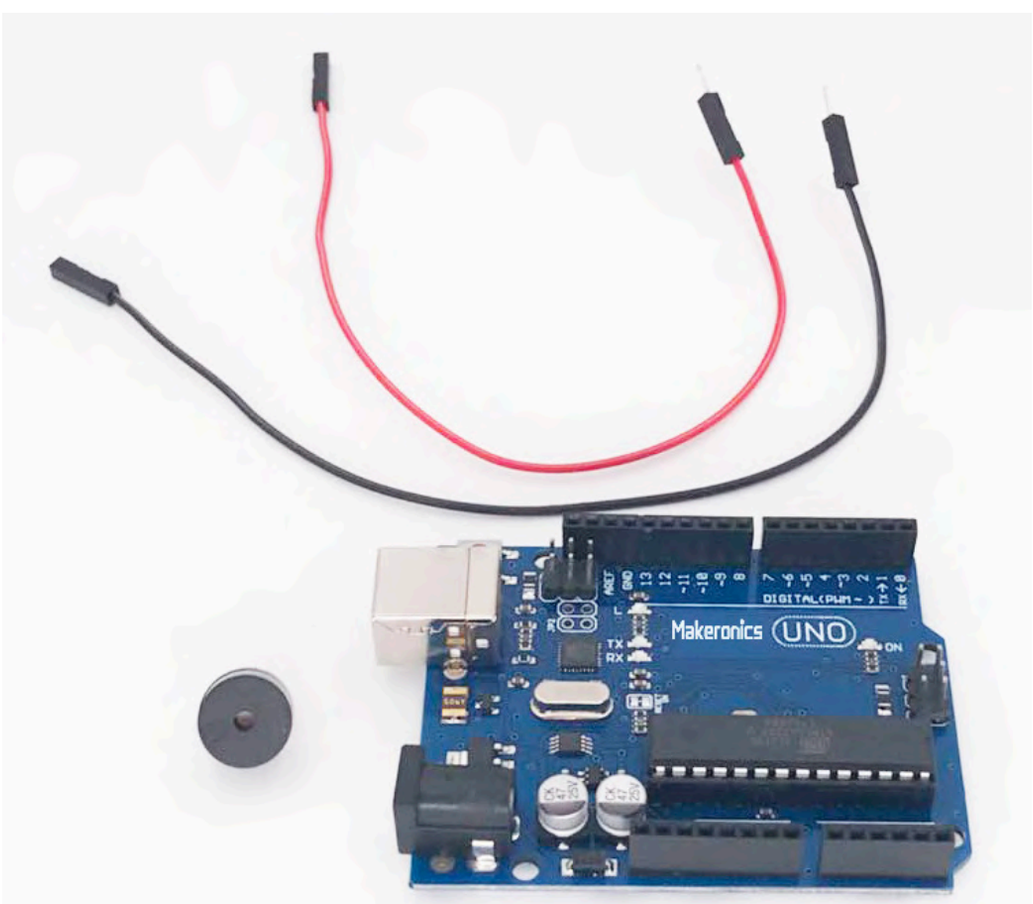
9

Passive Buzzer

In this lesson, you will learn how to use a passive buzzer.
The purpose of the experiment is to generate eight different sounds, each sound lasting 0.5 seconds: from Alto Do (523Hz), Re (587Hz), Mi (659Hz), Fa (698Hz), So (784Hz), La (880Hz), Si (988Hz) to Treble Do (1047Hz).

Component Required:

- 1 x Makeronics Uno R3
- 1 x Passive buzzer
- 2 x F-M wires (Female to Male DuPont wires)



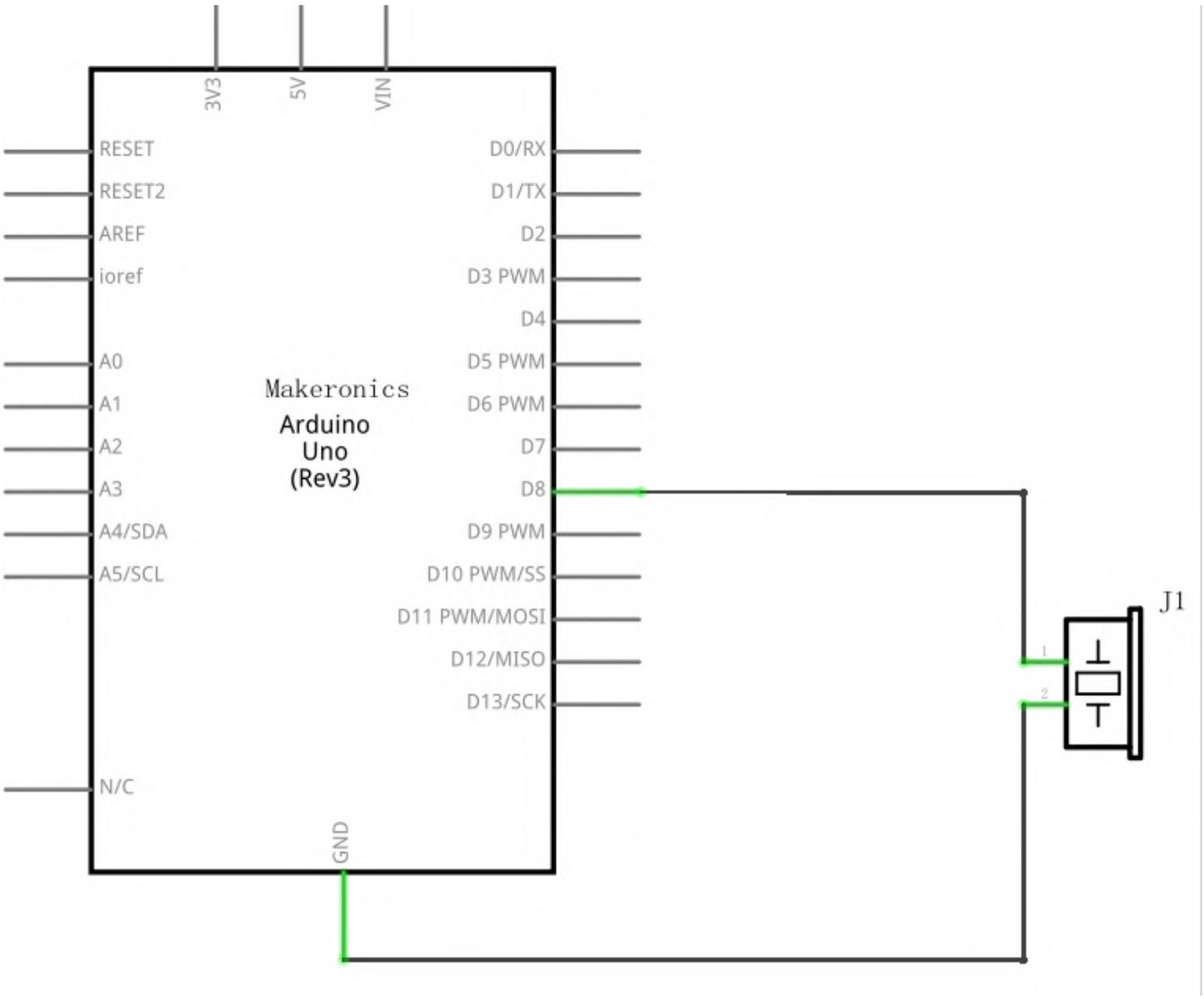
Component Introduction:

Passive Buzzer:

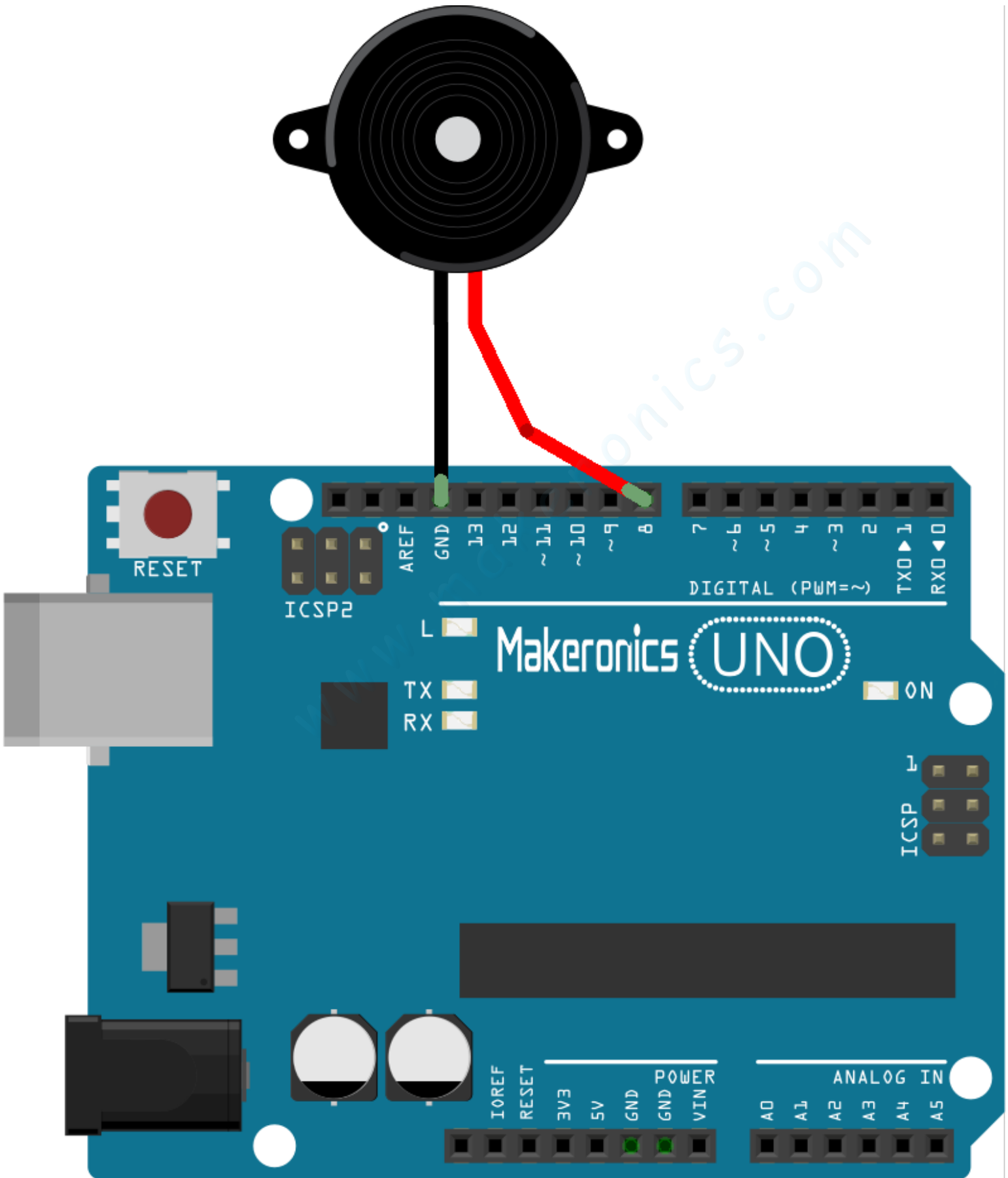
The working principle of passive buzzer is using PWM generating audio to make the air to vibrate. Appropriately changed as long as the vibration frequency, it can generate different sounds. For example, sending a pulse of 523Hz, it can generate Alto Do, pulse of 587Hz, it can generate midrange Re, pulse of 659Hz, it can produce midrange Mi. By the buzzer, you can play a song.

We should be careful not to use the UNO R3 board analog Write () function to generate a pulse to the buzzer, because the pulse output of analog Write () is fixed (500Hz).

Connection Schematic:



Wiring diagram:



Wiring the buzzer connected to the UNO R3 board, the red (positive) to the pin8, black wire (negative) to the GND.

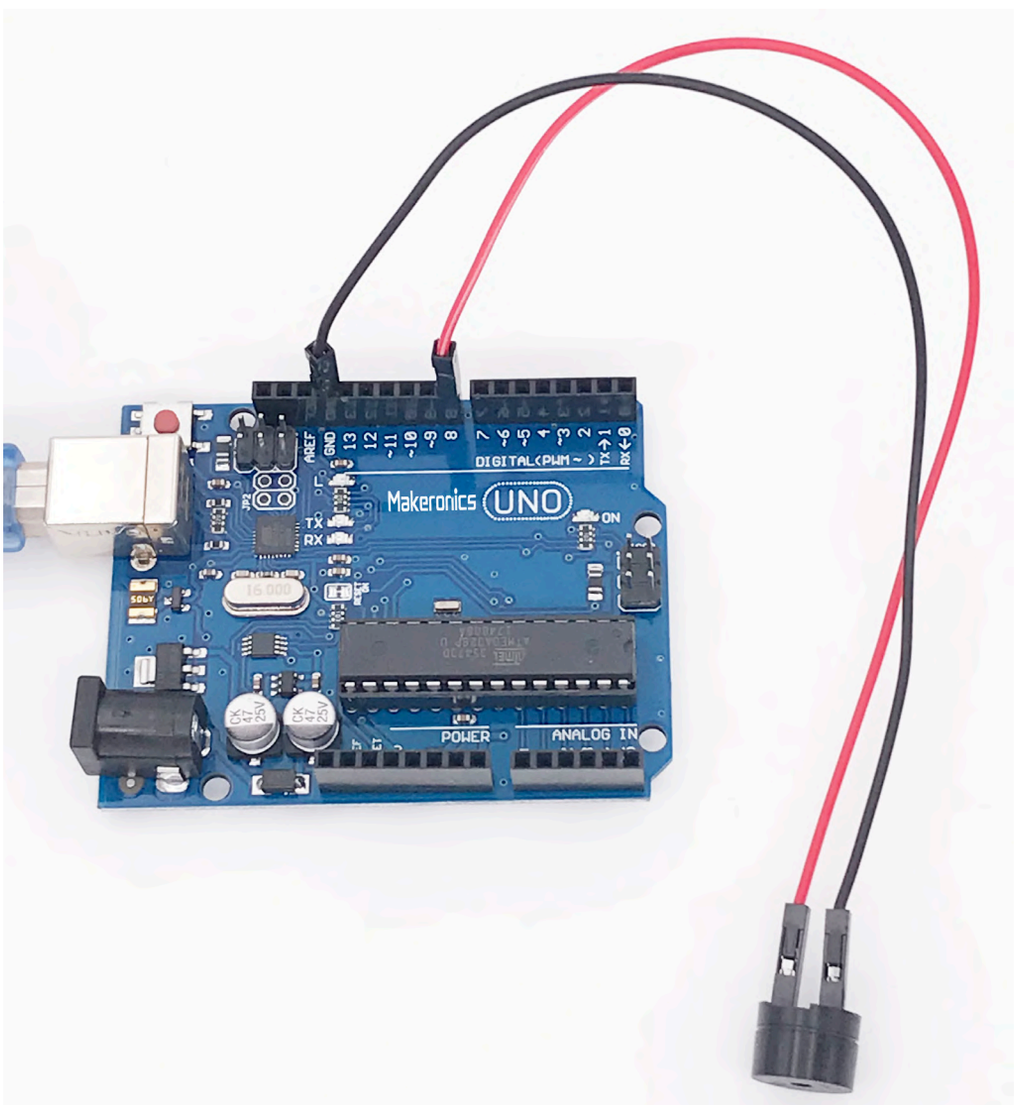
Code:

After wiring, please open the program in the code folder-Lesson 10 Passive Buzzer and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Before you can run this, make sure that you have installed the <pitches> library or re-install it, if necessary. Otherwise, your code won't work.

For details about loading the library file, see Lesson 2.

Demo:



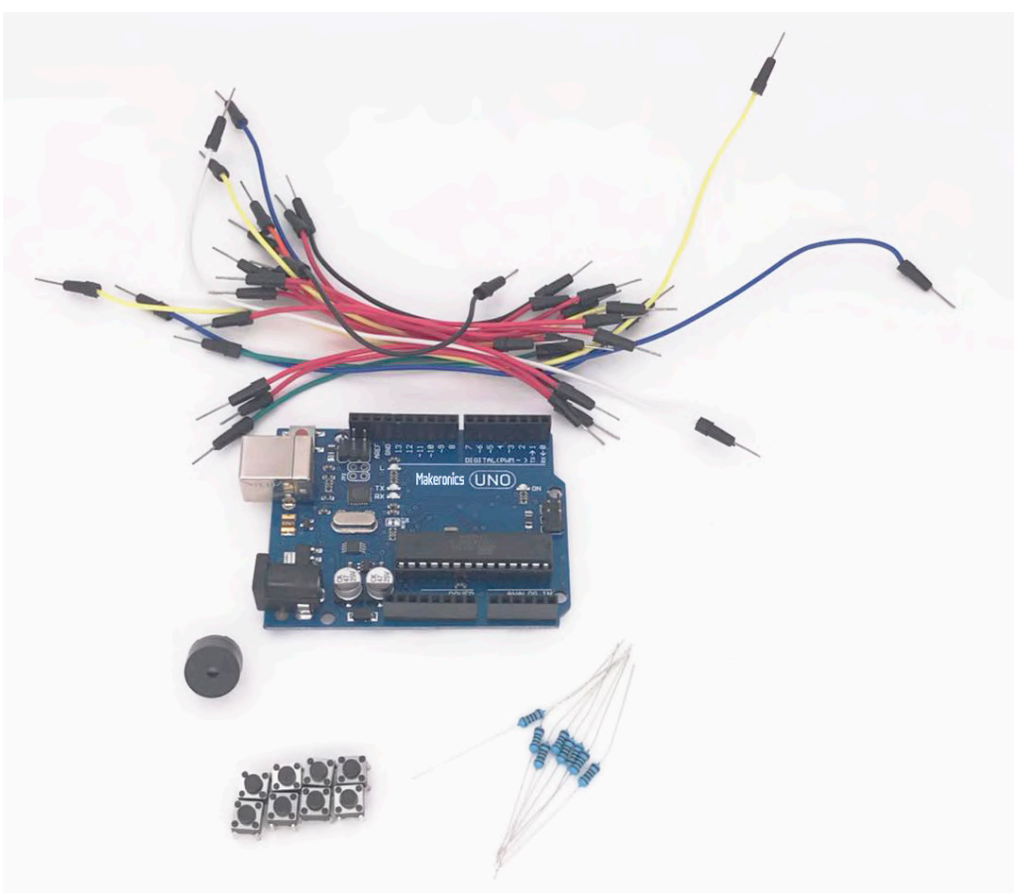
Project 7

10 Piano

In this lesson, we'll use some momentary pushbuttons and a Active buzzer to create a simple piano.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 Tie-points Breadboard
- 1 x Passive buzzer
- 8 x pushbuttons
- 8 x 1k-ohm resistors
- 2 x F-M wires (Female to Male DuPont wires)
- 20 x M-M wires (Male to Male jumper wires)

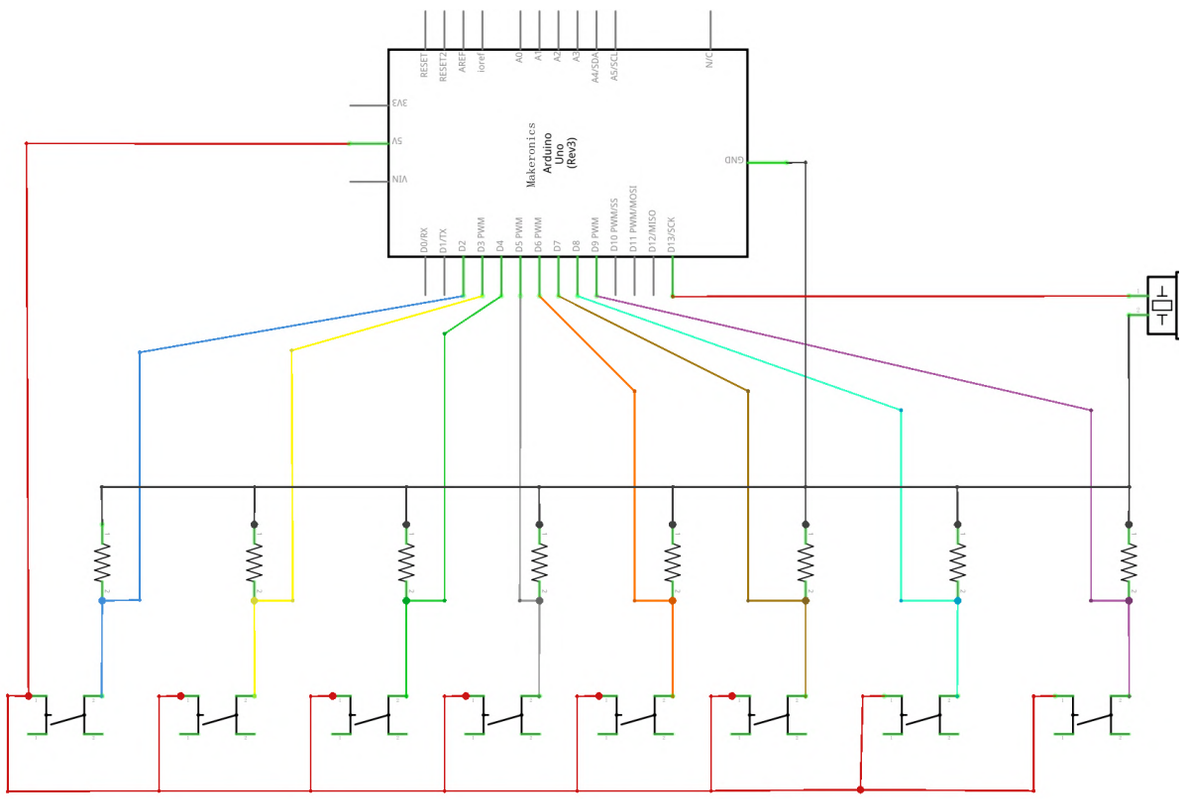


We can make recognizable notes by getting the passive buzzer to click hundreds of times a second at a particular frequency, so first we need to know the frequency of the different tones we want. The following table shows the notes and their corresponding frequencies. Period is the duration of the cycle, in microseconds, at which the frequency is created. For example, to get a C note (261 Hz), we need the piezo to cycle at a period of 3,830 microseconds.

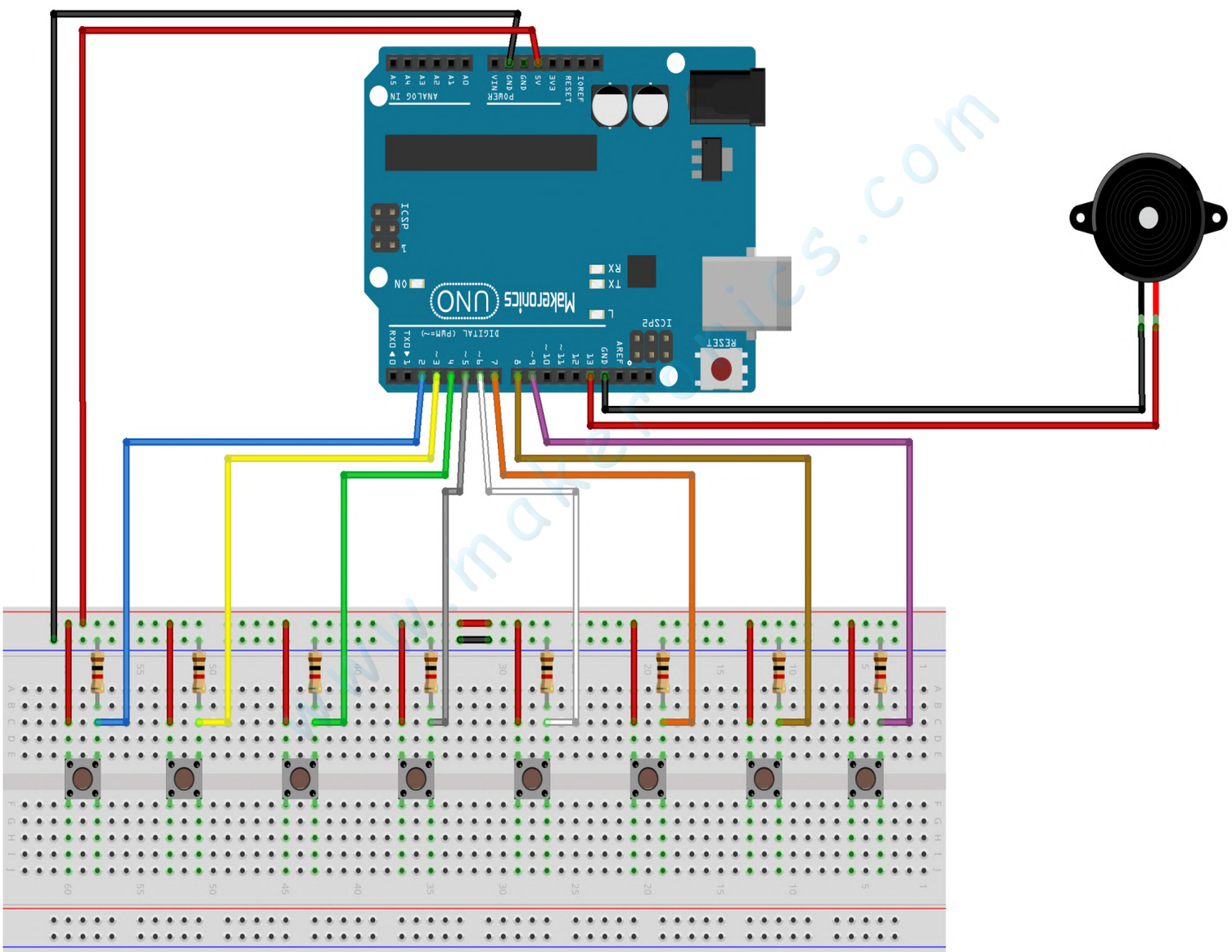
We halve the period to get the timeHigh value, which is used in the code to create the note. (The tone is caused by the piezo being turned on and off very quickly, so the time that the piezo is on, or HIGH, is half the period.)

Note	Frequency	Period	Timehigh
c	261 Hz	3,830	1915
d	294 Hz	3,400	1700
e	329 Hz	3,038	1519
f	349 Hz	2,864	1432
g	392 Hz	2,550	1275
a	440 Hz	2,272	1136
b	493 Hz	2,028	1014
C	523 Hz	1,912	956

Connection Schematic:



Wiring diagram:

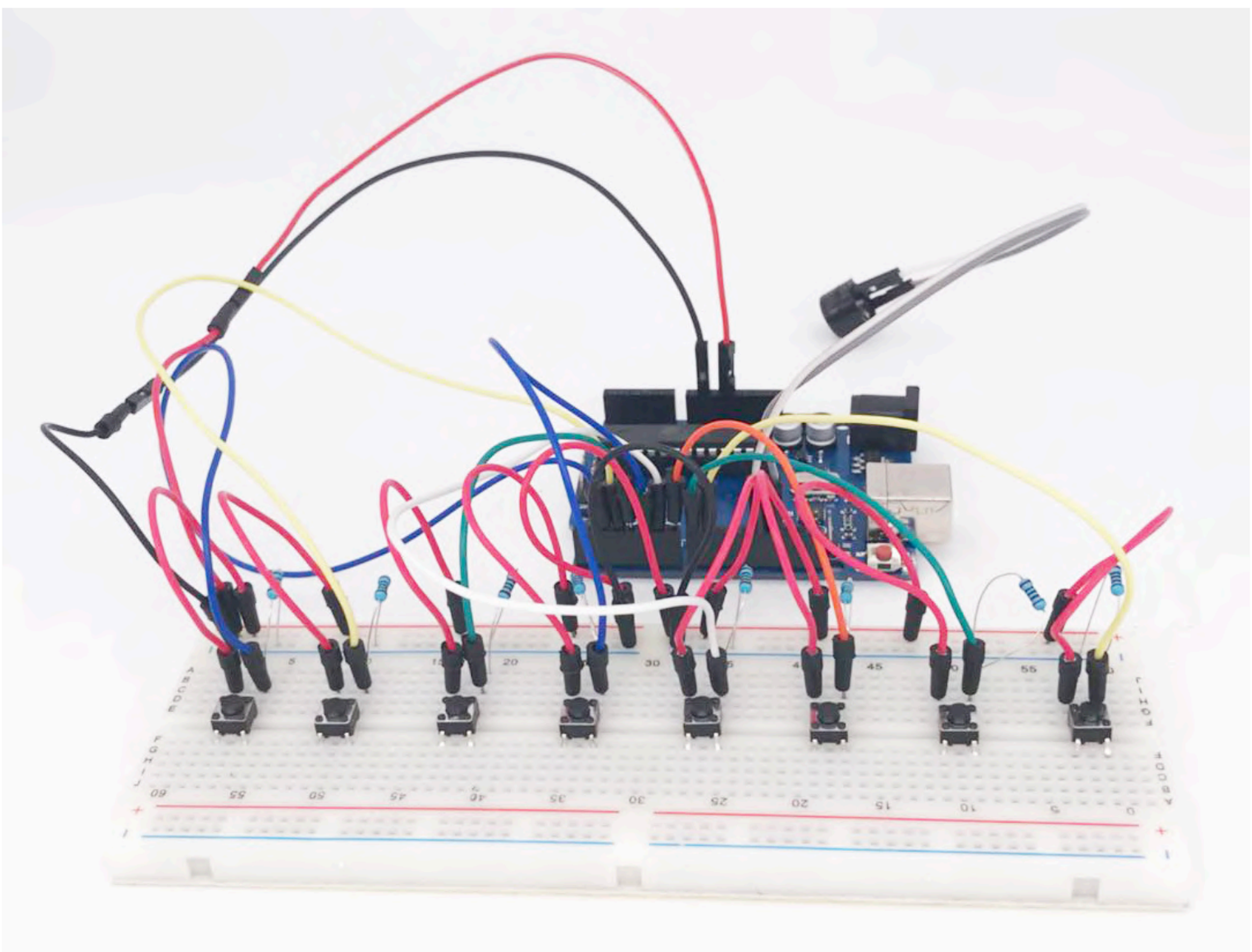


Code:

After wiring, please open the program in the code folder- Lesson 9 Piano and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Before you can run this, make sure that you have installed the <pitches> library or re-install it, if necessary. Otherwise, your code won't work.

Demo:



Project 8

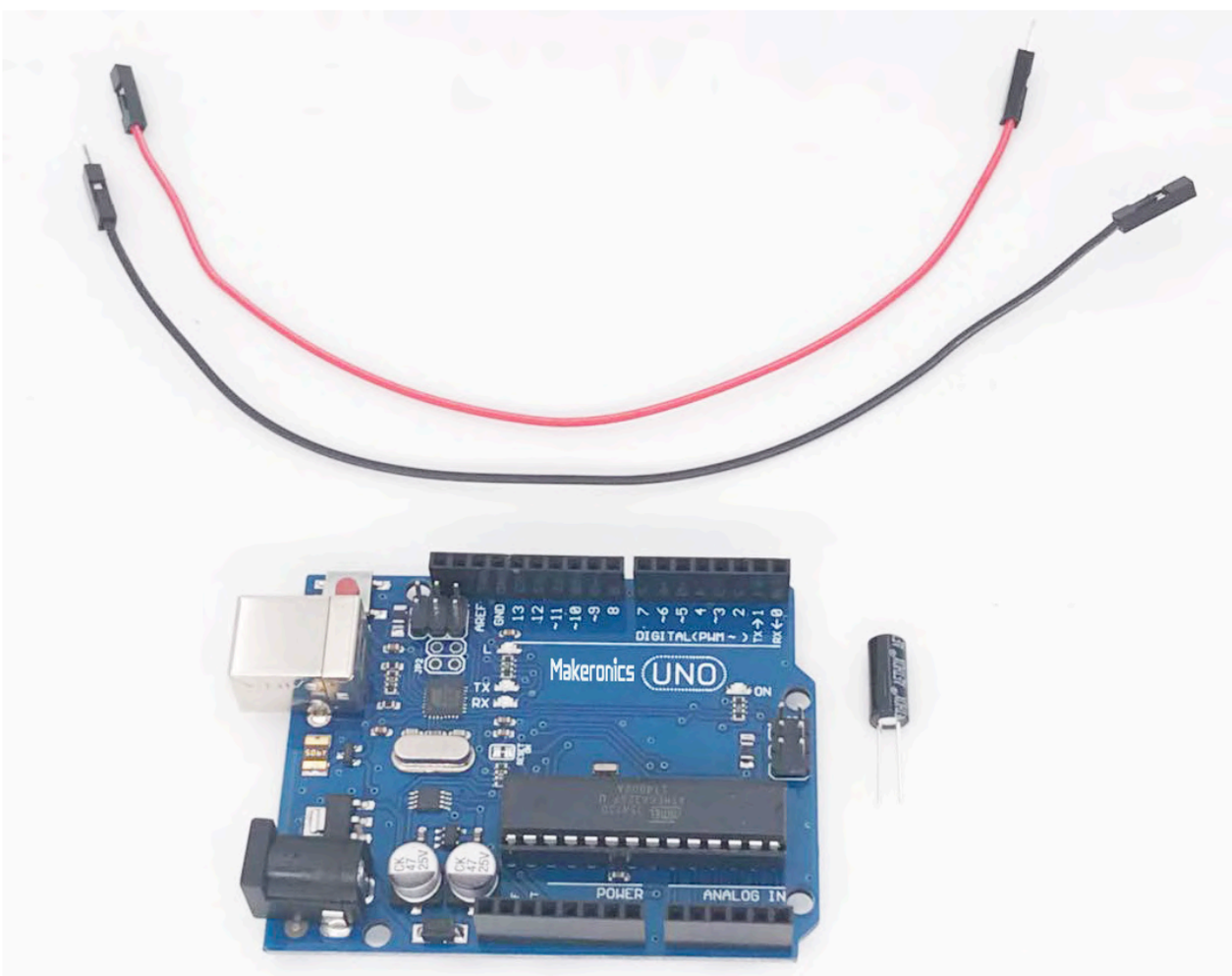
11

Tilt Ball Switch

In this lesson, you will learn how to use a tilt ball switch in order to detect small angle of inclination.

Component Required:

- 1 x Makeronics Uno R3
- 1 x Tilt Ball switch
- 2 x F-M wires (Female to Male DuPont wires)



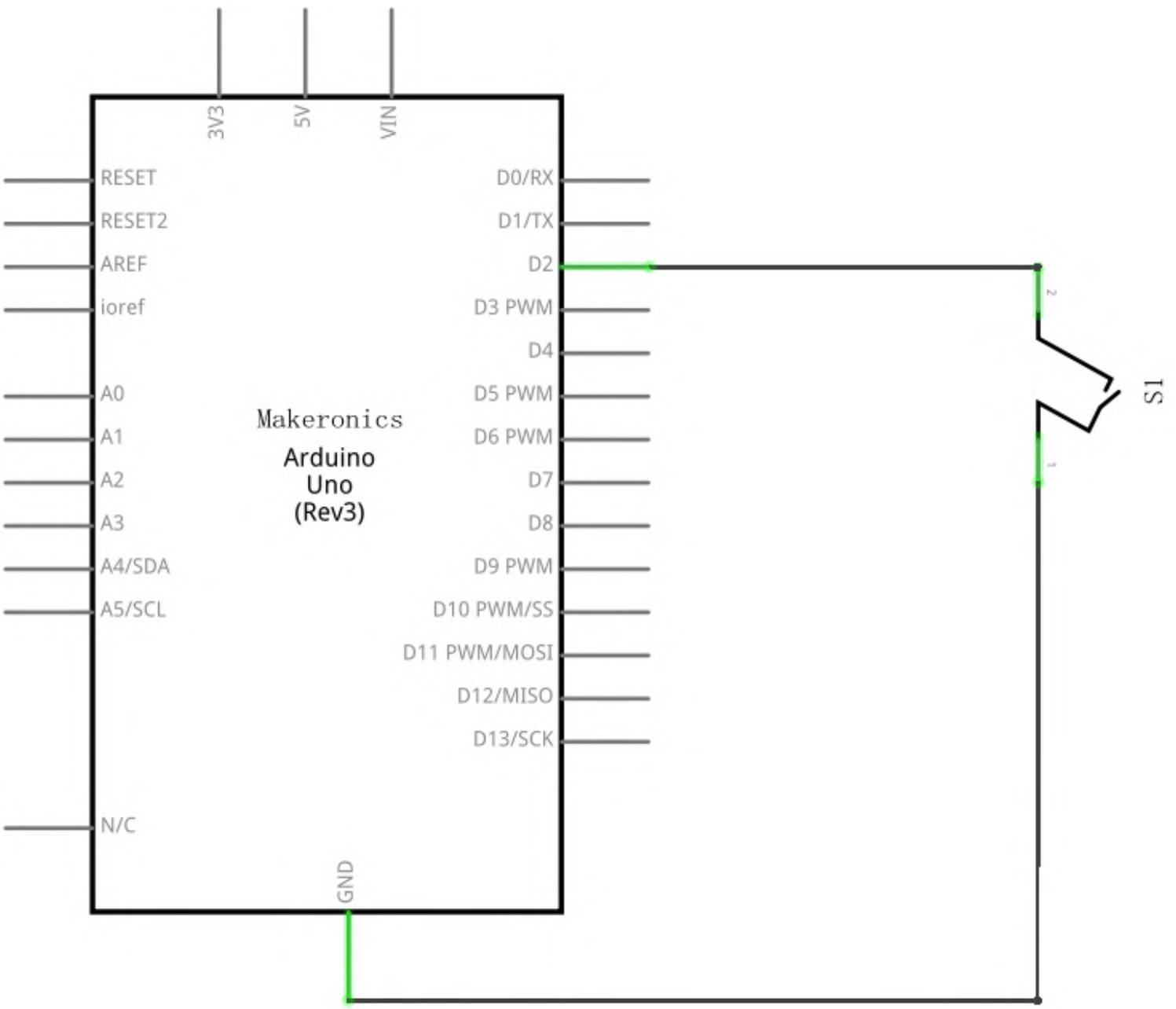
Component Introduction:

Tilt sensor:

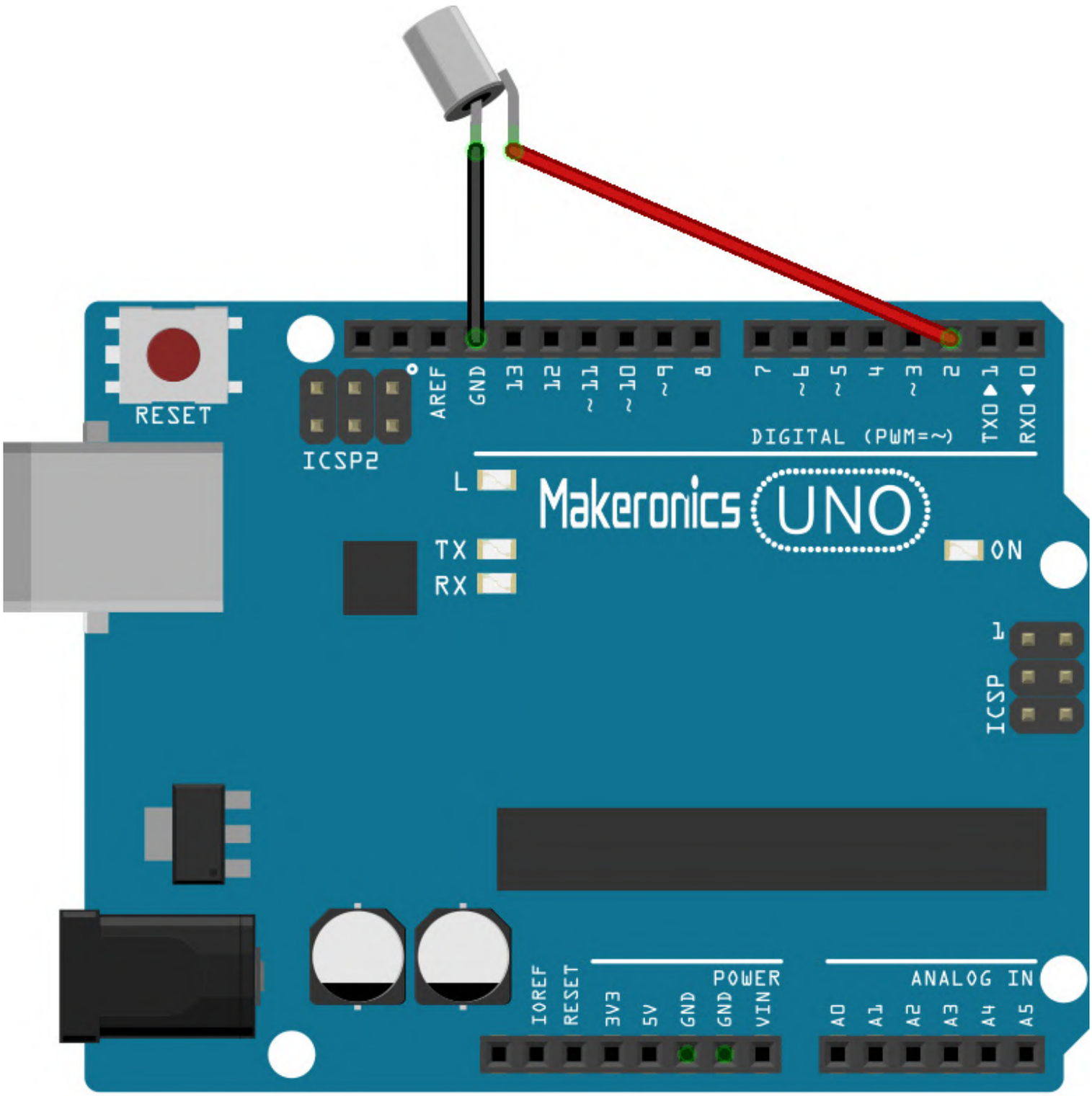
The tilt sensor is a component that can detect the tilting of an object. However it is only the equivalent to a pushbutton activated through a different physical mechanism. This type of sensor is the environmental-friendly version of a mercury-switch. It contains a metallic ball inside that will commute the two pins of the device from on to off and viceversa if the sensor reaches a certain angle.

The tilt ball switch is composed of a metal ball inside a metal casing that makes a connection when the switch is in an upright position. If you tilt the switch, the ball shifts and the connection is broken.

Connection Schematic:



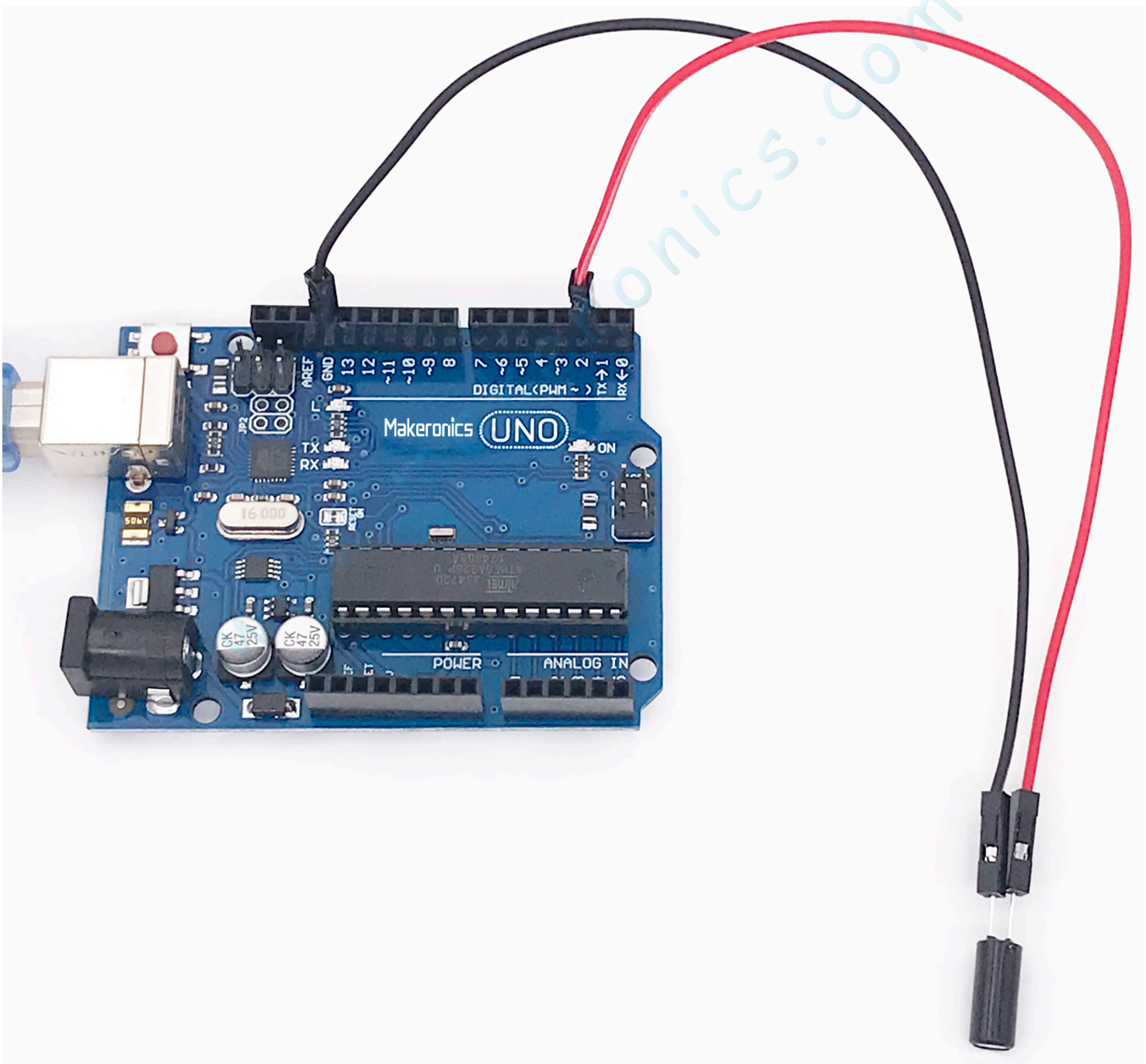
Wiring diagram:



Code:

After wiring, please open the program in the code folder- Lesson 11 Ball Switch and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Demo:



Project 9

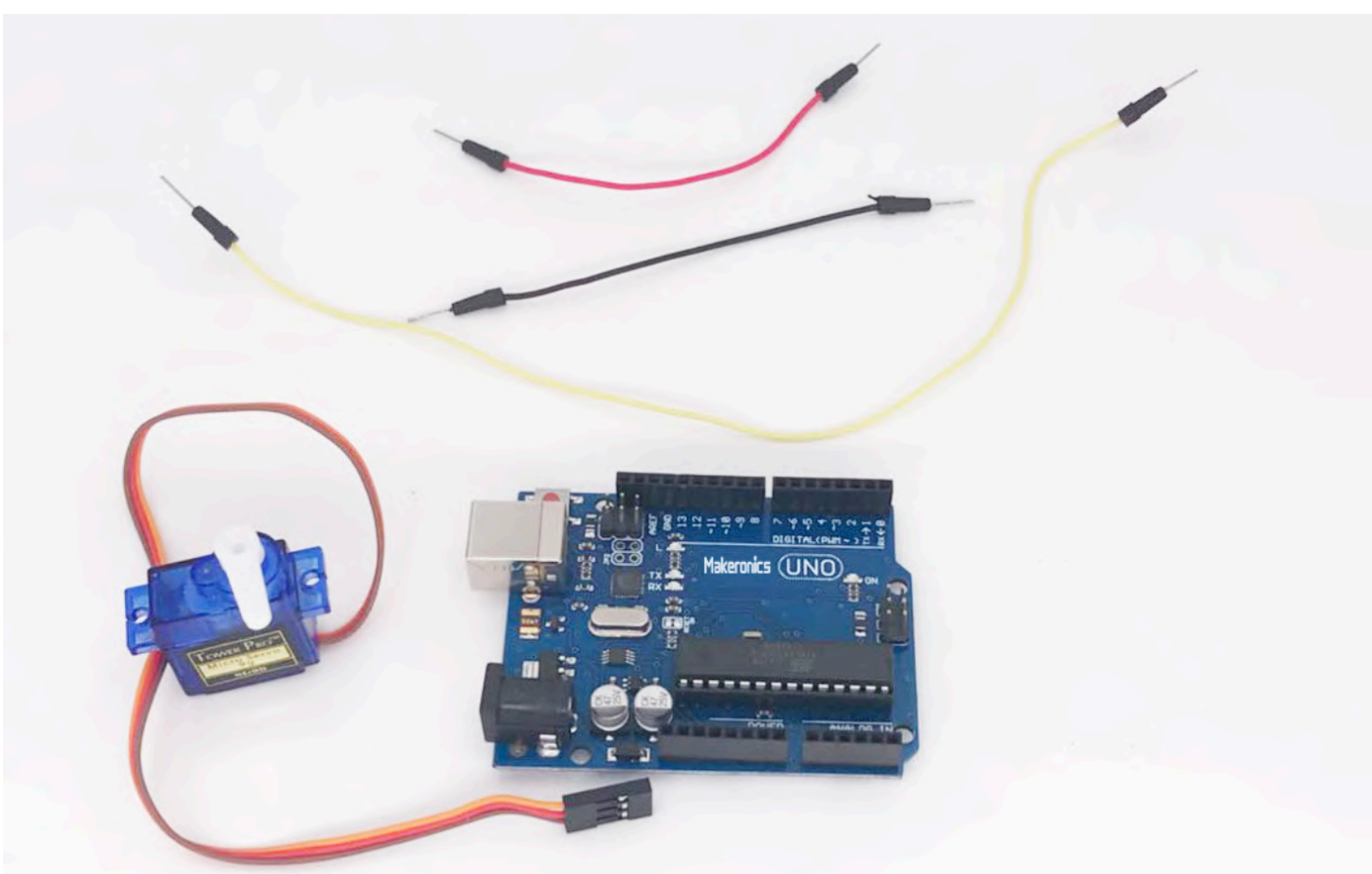
12

Servo motor

In this lesson, you will learn how to control a servomotor's arm from 0° to 180°.

Component Required:

- 1 x Makeronics Uno R3
- 1 x Servo (SG90)
- 3 x M-M wires (Male to Male jumper wires)



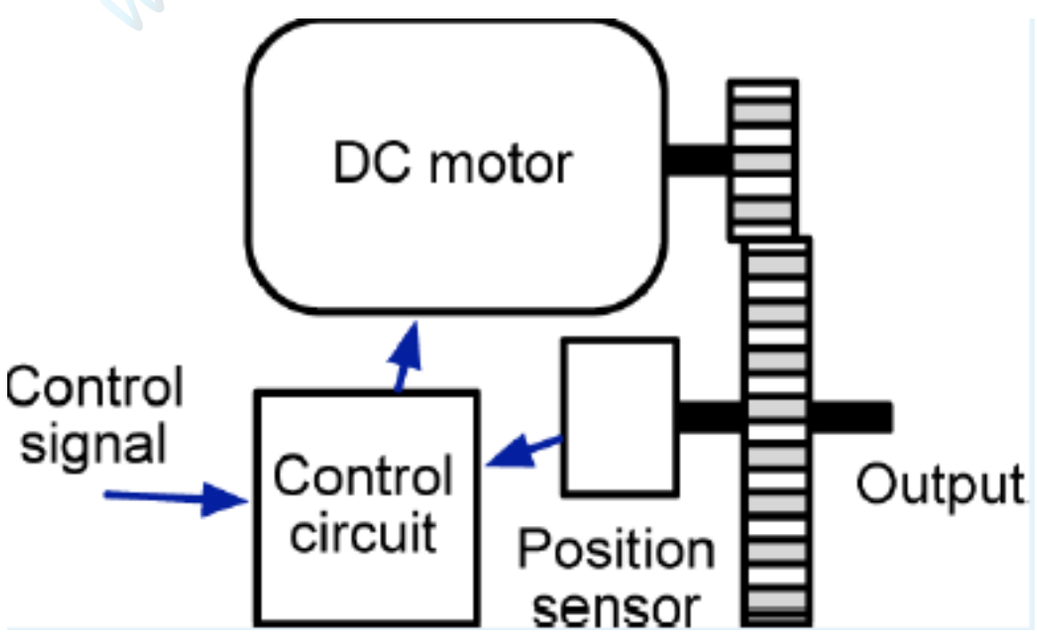
Component Introduction

Servo:

A servomotor is a closed-loop servomechanism that uses position feedback to control its motion and final position. The input to its control is a signal (either analogue or digital) representing the position commanded for the output shaft.

A servo is made up of a DC motor that drives a reduction gearbox to reduce the rotational speed of the motor while at the same time increasing the torque. The output drive shaft is connected to a position sensor (usually a variable resistor) to monitor the position of the shaft. A control circuit uses this input from the position sensor combined with a control signal that sets the desired position and uses that to control the power and direction that the DC motor turns. The control unit takes the desired position and subtracts the actual position to give the "error," which may be positive or negative. This error is then used to power the motor.

The larger the difference between the desired and actual positions of the output shaft, the faster the motor will turn toward the desired position. As it gets closer to zero error, the power to the motor decreases.



Controlling a Servo:

The control signal to the servo is not, as you might expect, a voltage, but rather it is a PWM signal.
The Arduino has a servo library that will generate pulses on any of the pins, so you do not need to use a pin that is marked as PWM capable.

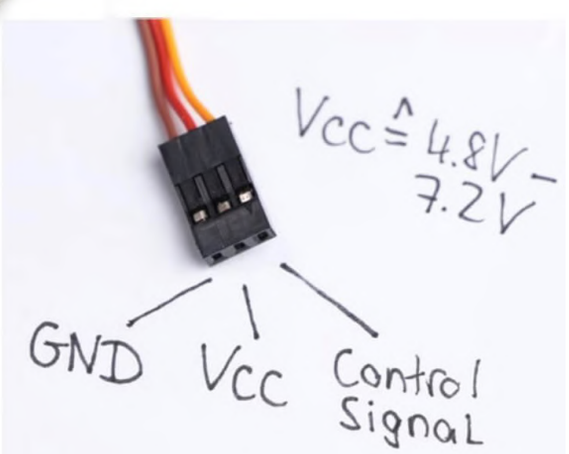
SG90:

Universal for JR and FP connector
Cable length : 25cm
No load; Operating speed: 0.12 sec / 60 degree (4.8V), 0.10 sec / 60 degree (6.0V)
Stall torque (4.8V): 1.6kg/cm
Temperature : -30~60'C
Dead band width: 5us
Working voltage: 3.5~6V
Dimension : 1.26 in x 1.18 in x 0.47 in (3.2 cm x 3 cm x 1.2 cm)
Weight : 4.73 oz (134 g)

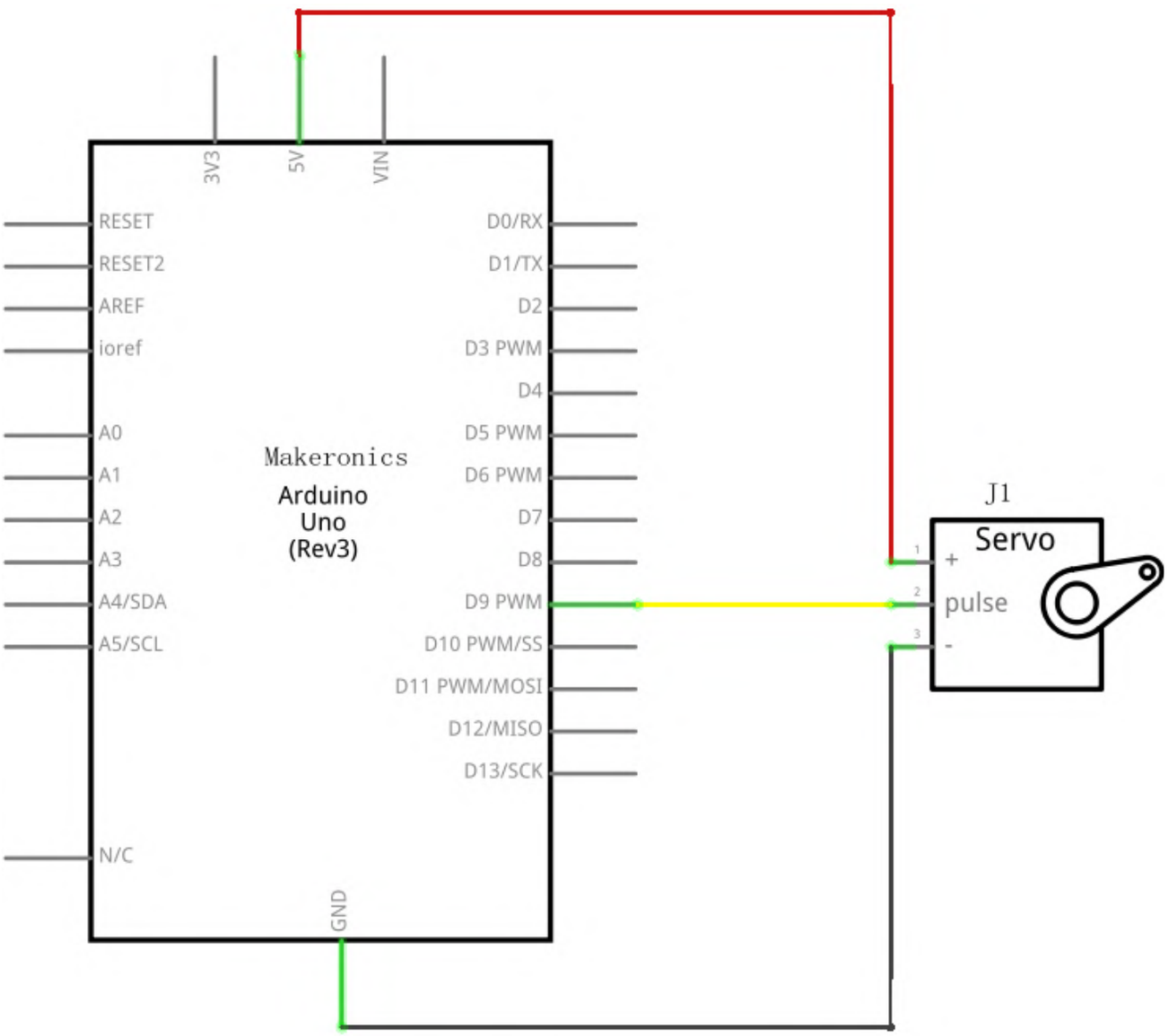
Note:

VCC=Power input

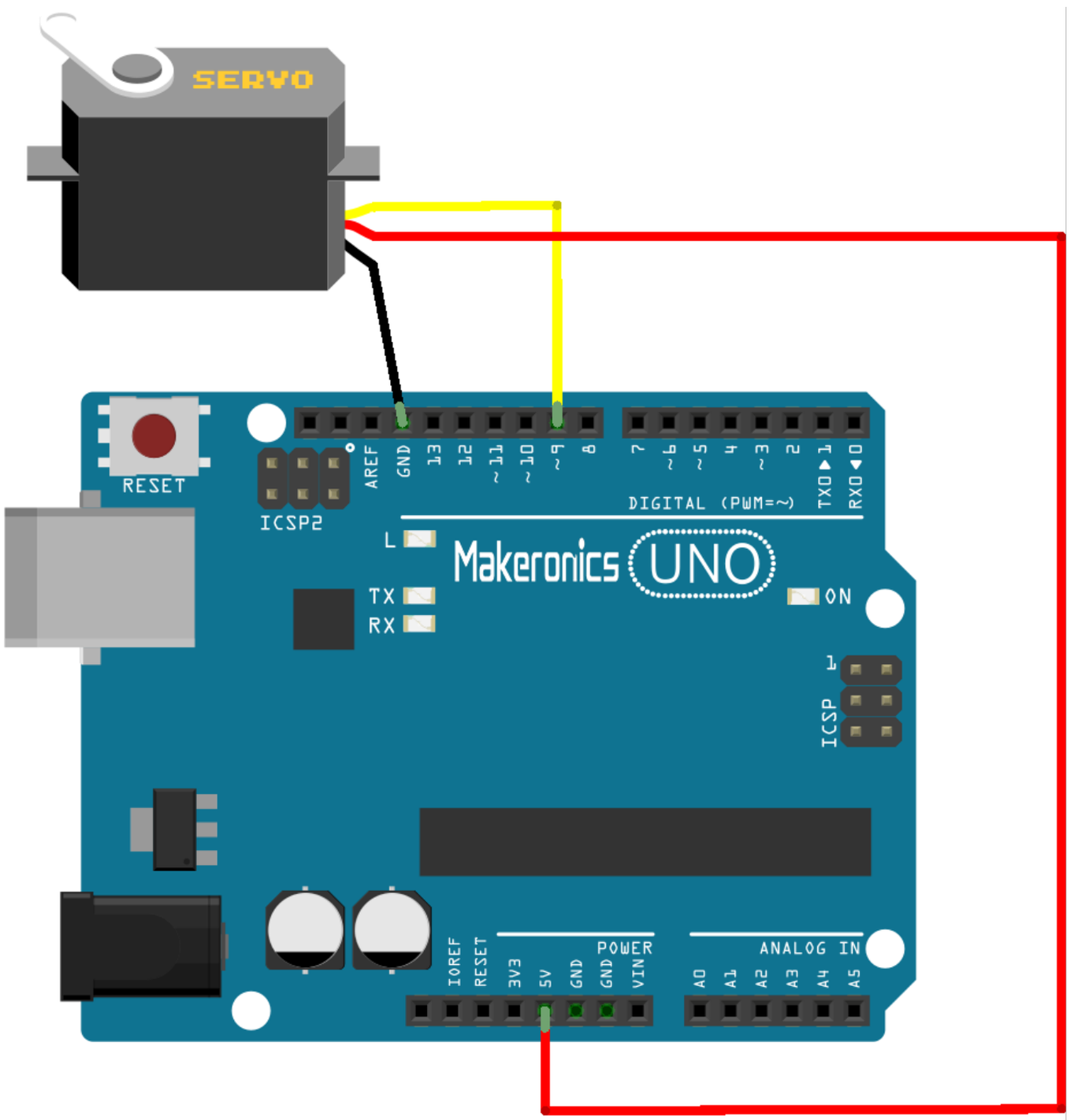
GND=Ground input



Connection Schematic:



Wiring diagram:

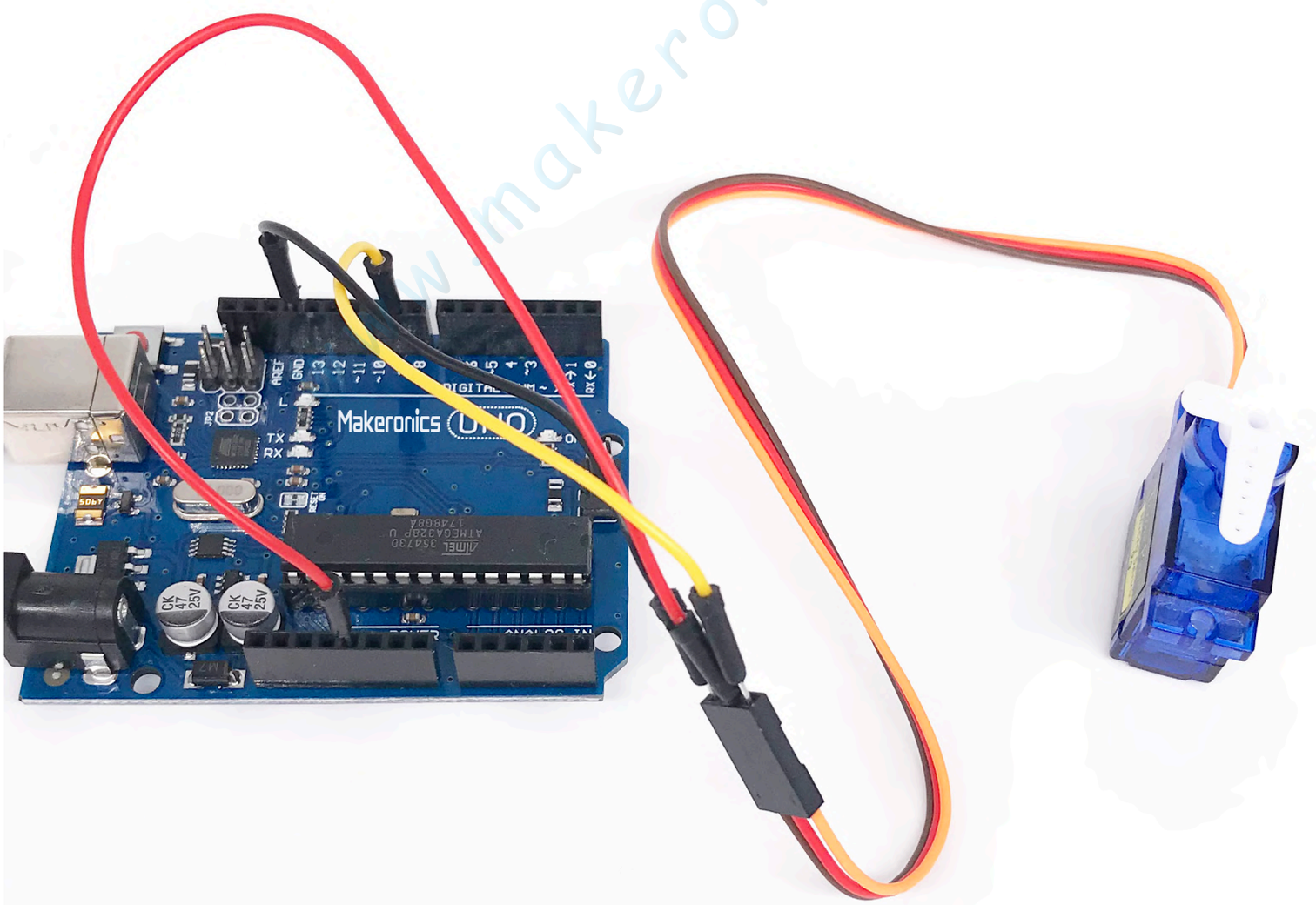


Code:

After wiring, please open the program in the code folder-Lesson 12 Servomotor and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < Servo> library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

In the picture, the brown wire of servo is adapted via the black M-M wires, the red one is adapted via the red M-M wires, and the orange one is adapted via the yellow M-M wires.

Demo:



Project 10

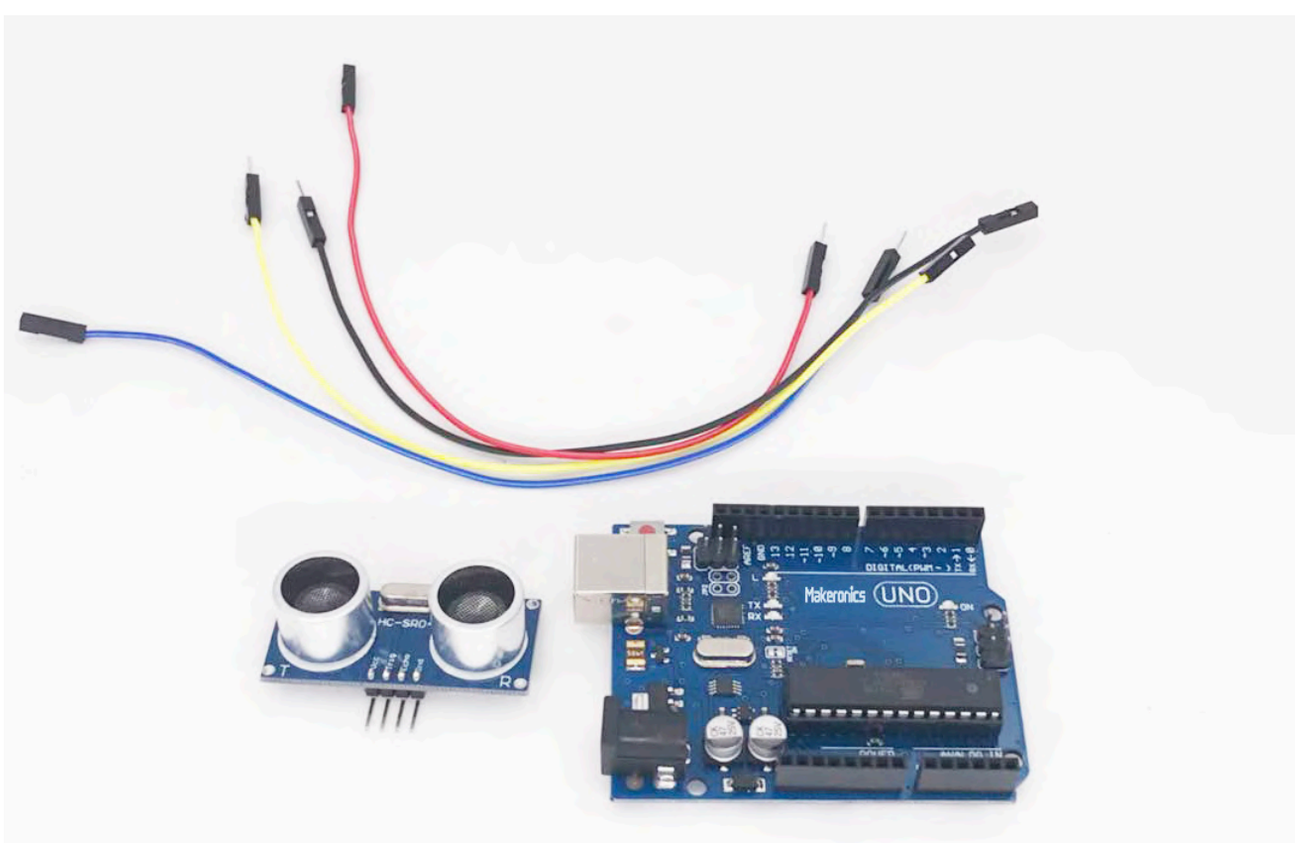
13

Ultrasonic Sensor Module

In this lesson, you will learn how to use ultrasonic sensor to measure distance.

Component Required:

- 1 x Makeronics Uno R3
- 1 x Ultrasonic sensor module
- 4 x F-M wires (Female to Male DuPont wires)



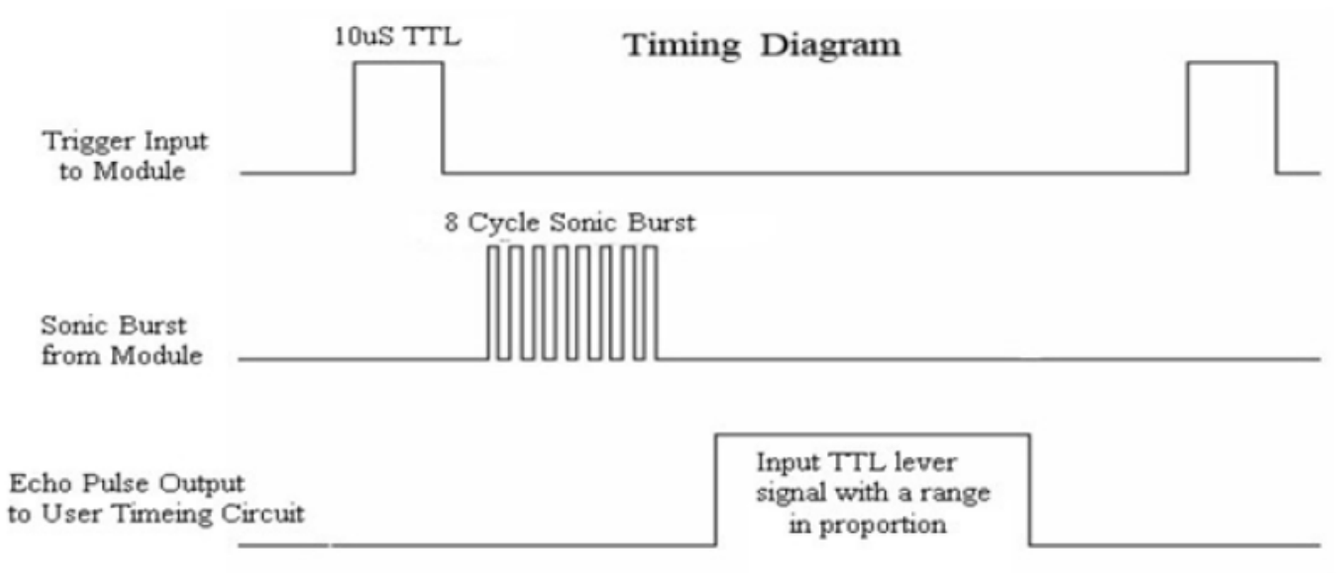
Component Introduction:

Ultrasonic sensor:

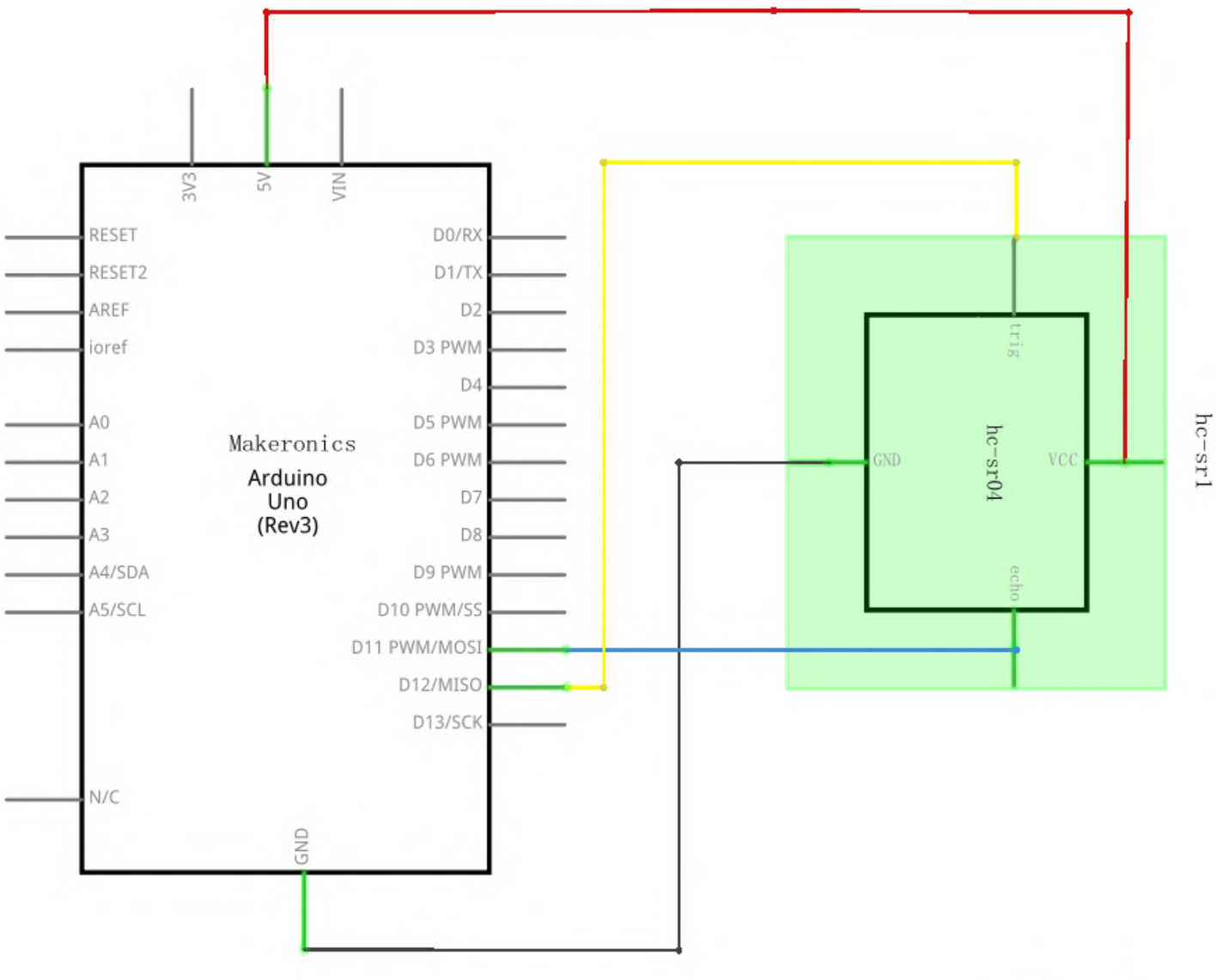
An ultrasonic sensor, typically called sonar, uses sound above the normal hearing frequency to detect distance. It does so by emitting a short sound pulse and waiting for it to return. It measures the time it takes for the sound to travel, bounce on objects, and then travel back to the sensor.
Because the speed of sound is known, the total time it takes for the sound to return is dependent on the distance to the objects. This creates a very wide sensing beam. It is useful in many applications, especially when we need to detect large, complex objects. However, the wide beam will create interference if we have two objects in range. It will always detect the closer one.

Ultrasonic sensor module HC-SR04 provides 2cm-400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic tore turning.
Test distance = (high level time × velocity of sound (340m/s) /2

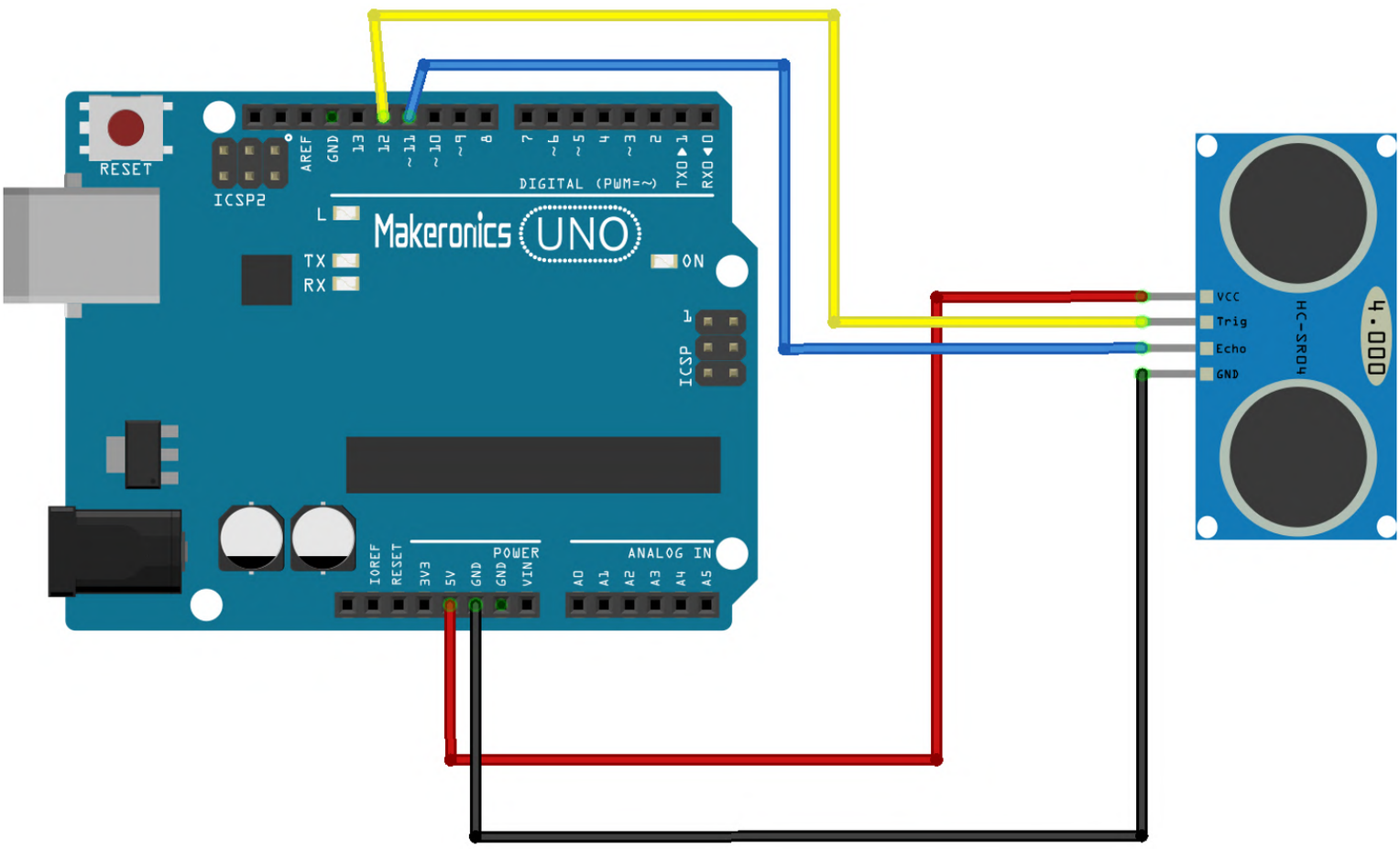
The Timing diagram is shown below. You only need to supply a short 10us pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: us / 58 = centimeters or us / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Connection Schematic:



Wiring diagram:



Code:

Using a Library designed for these sensors will make our code short and simple.

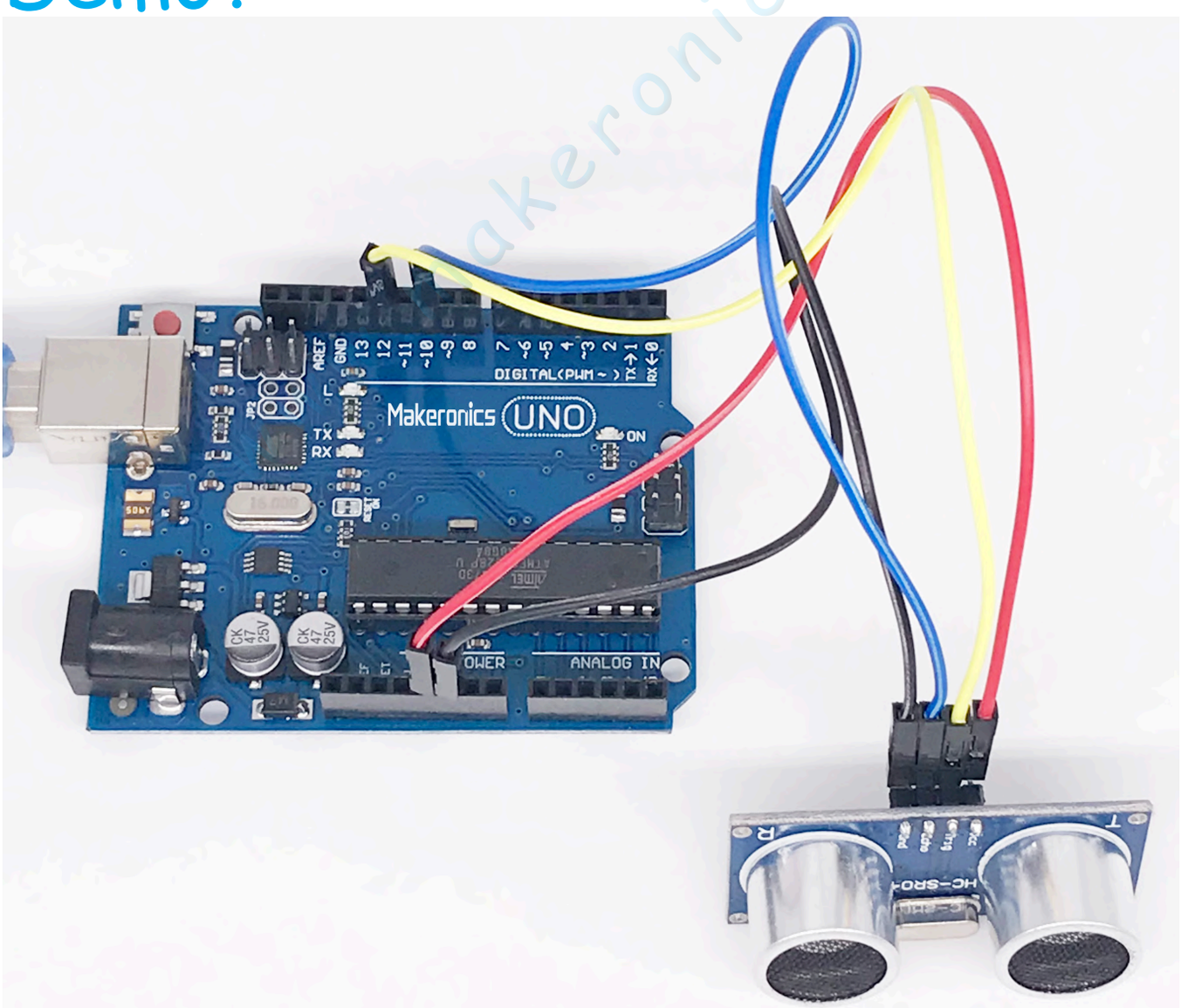
We include the library at the beginning of our code, and then by using simple commands we can control the behavior of the sensor.

After wiring, please open the program in the code folder- Lesson 13 Ultrasonic Sensor Module and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

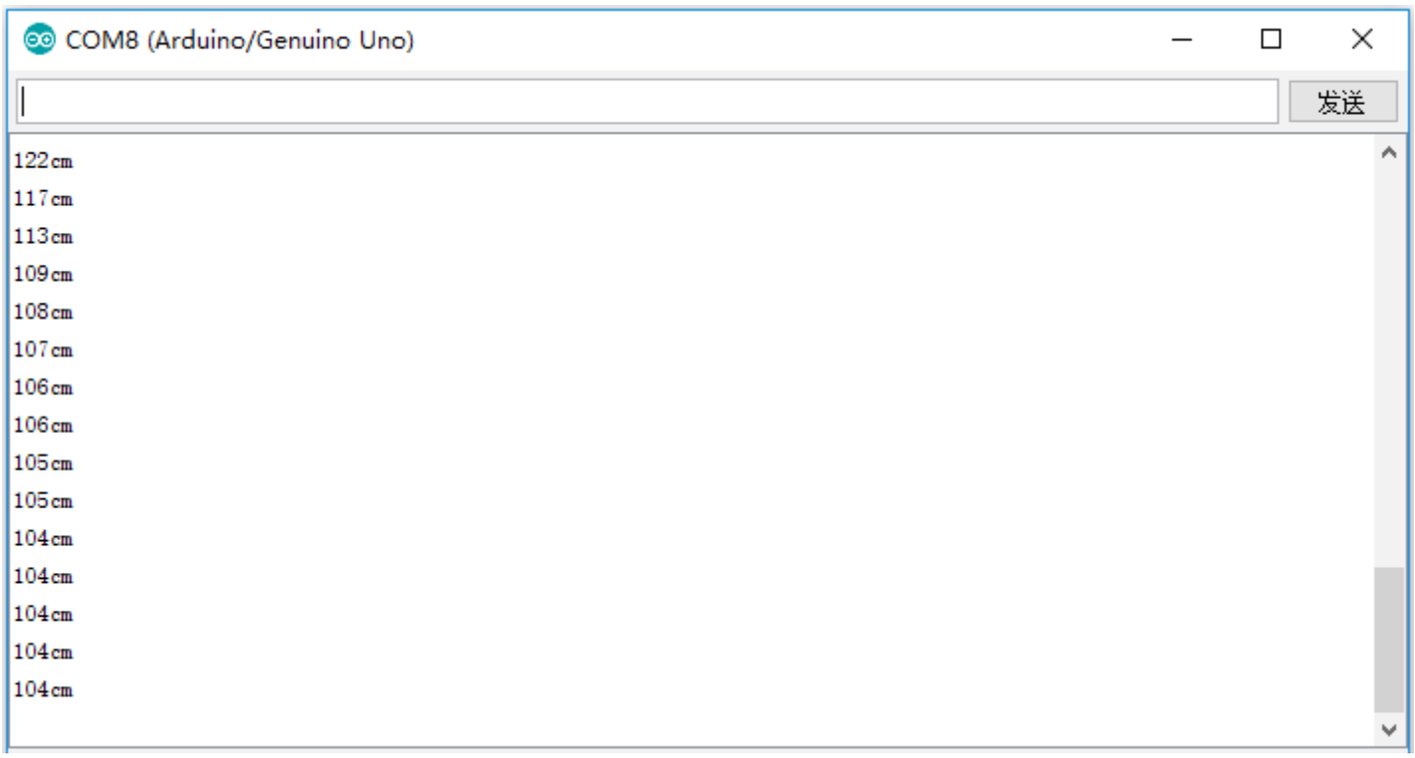
Before you can run this, make sure that you have installed the < HC-SR04> library or re-install it, if necessary. Otherwise, your code won't work.

For details about loading the library file, see Lesson 2.

Demo:



Open the serial monitor then you can see the data as blow:
Click the Serial Monitor button to turn on the serial monitor. The basics about the serial monitor are introduced in details in Lesson 2.



Project 11

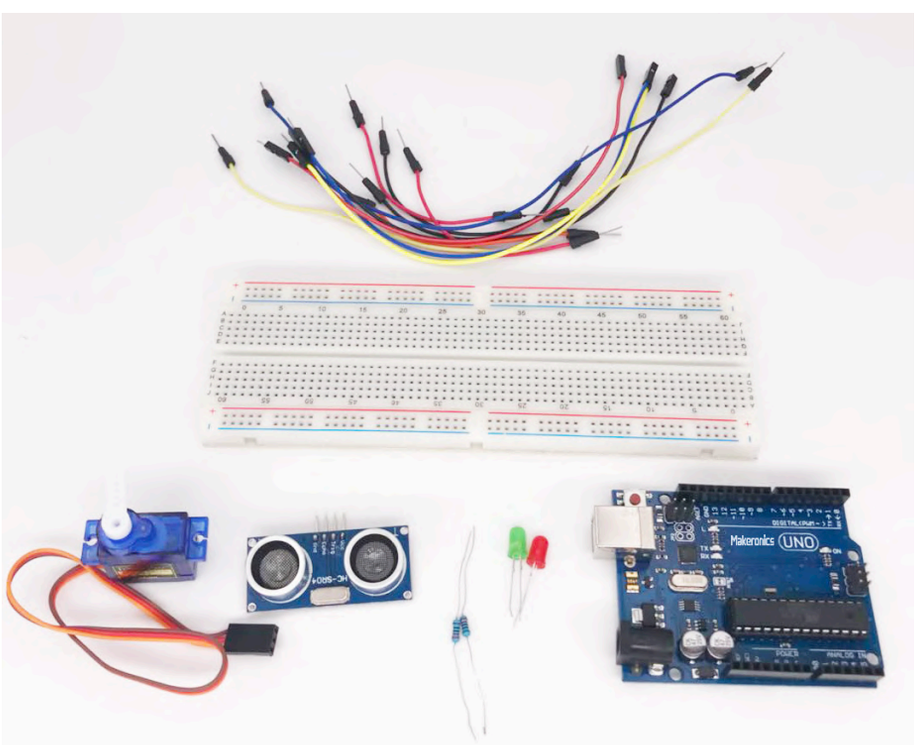
14

Intruder Sensor

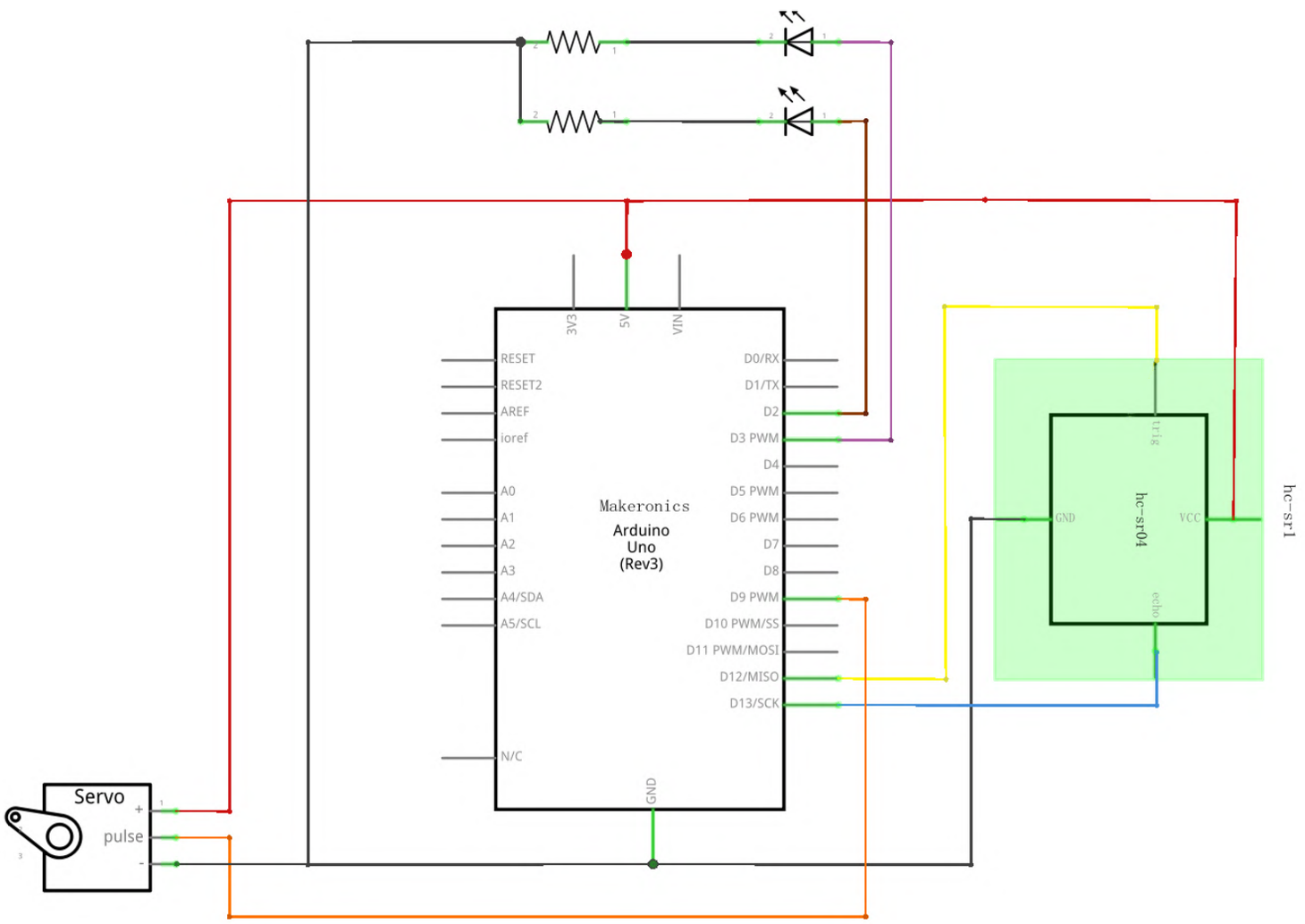
In this tutorial we will learn how to use ultrasonic sensor and servomotor as intruder sensor.

Component Required:

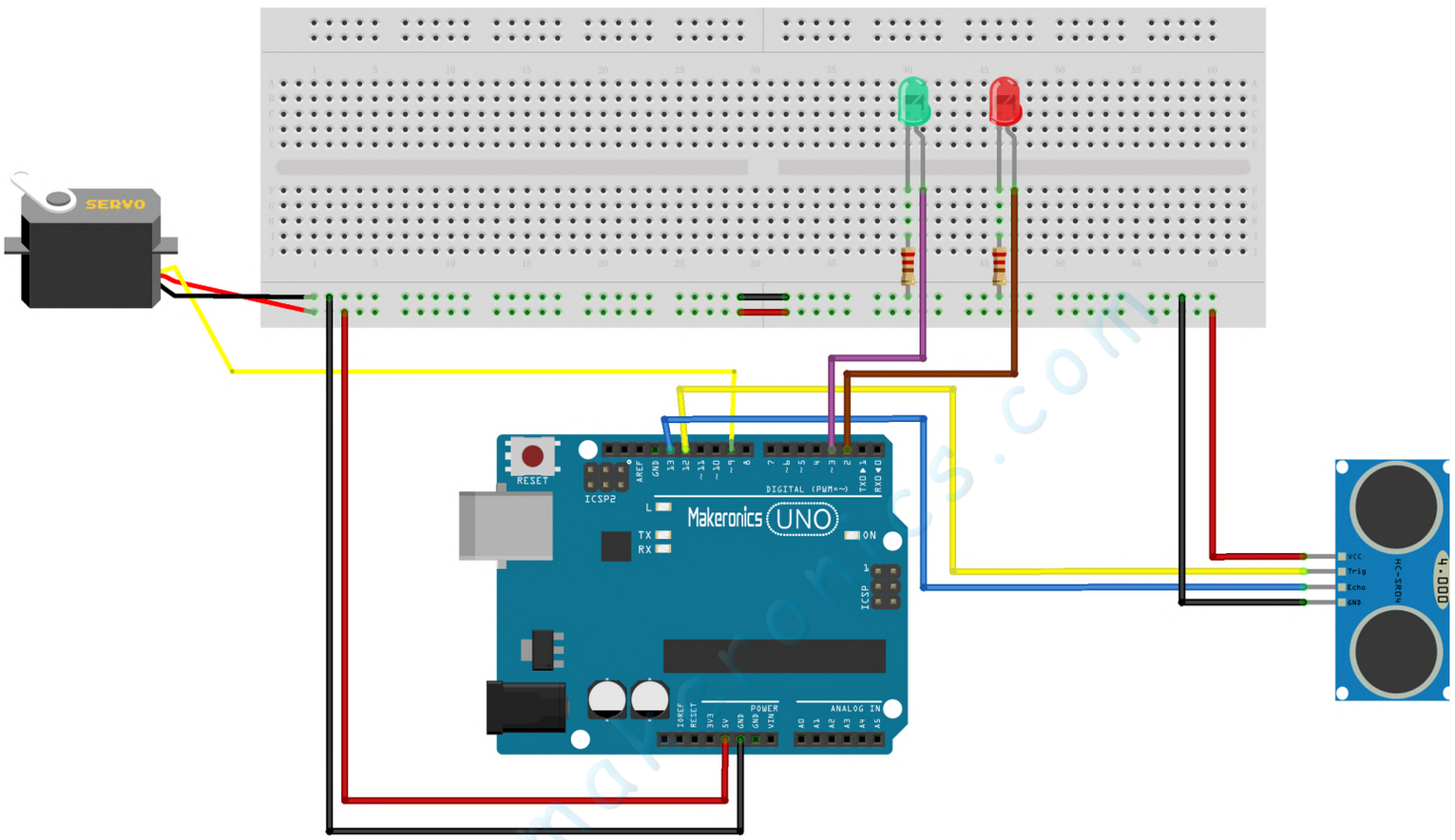
- 1 x Makeronics Uno R3
- 1 x 830 Tie-points Breadboard
- 1 x Ultrasonic sensor module
- 1 x Servo (SG90)
- 1 x 5mm Red LED
- 1 x 5mm Green LED
- 2 x 220ohm resistors
- 4 x F-M wires (Female to Male DuPont wires)
- 7 x M-M wires (Male to Male jumper wires)



Connection Schematic:



Wiring diagram:

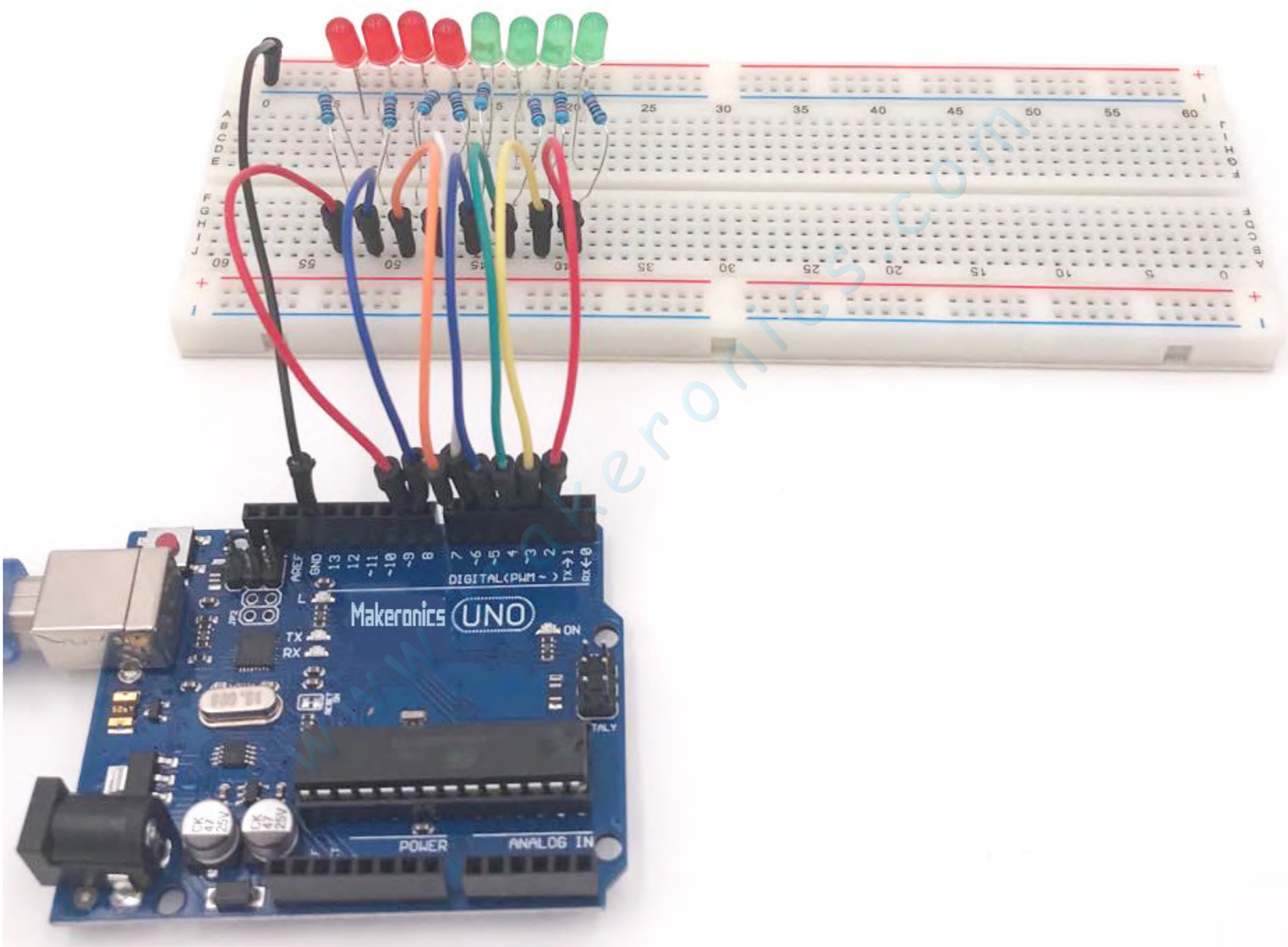


The Trig pin on the sensor is connected to Arduino pin 12 and emits an ultrasonic signal or ping. When the signal reaches an object, it bounces back to the module, and this echo is sent to Arduino pin 13. The time difference between the two signals gives us our distance reading. If the distance is more than our set minimum, the green LED stays on; if not, the red LED lights and the servo moves.

Code:

Using a Library designed for these sensors will make our code short and simple.
We include the library at the beginning of our code, and then by using simple commands we can control the behavior of the sensor.
After wiring, please open the program in the code folder- Lesson 14 Intruder Sensor and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < HC-SR04> library and <Servo> library or re-install them, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

Demo:



Project 12

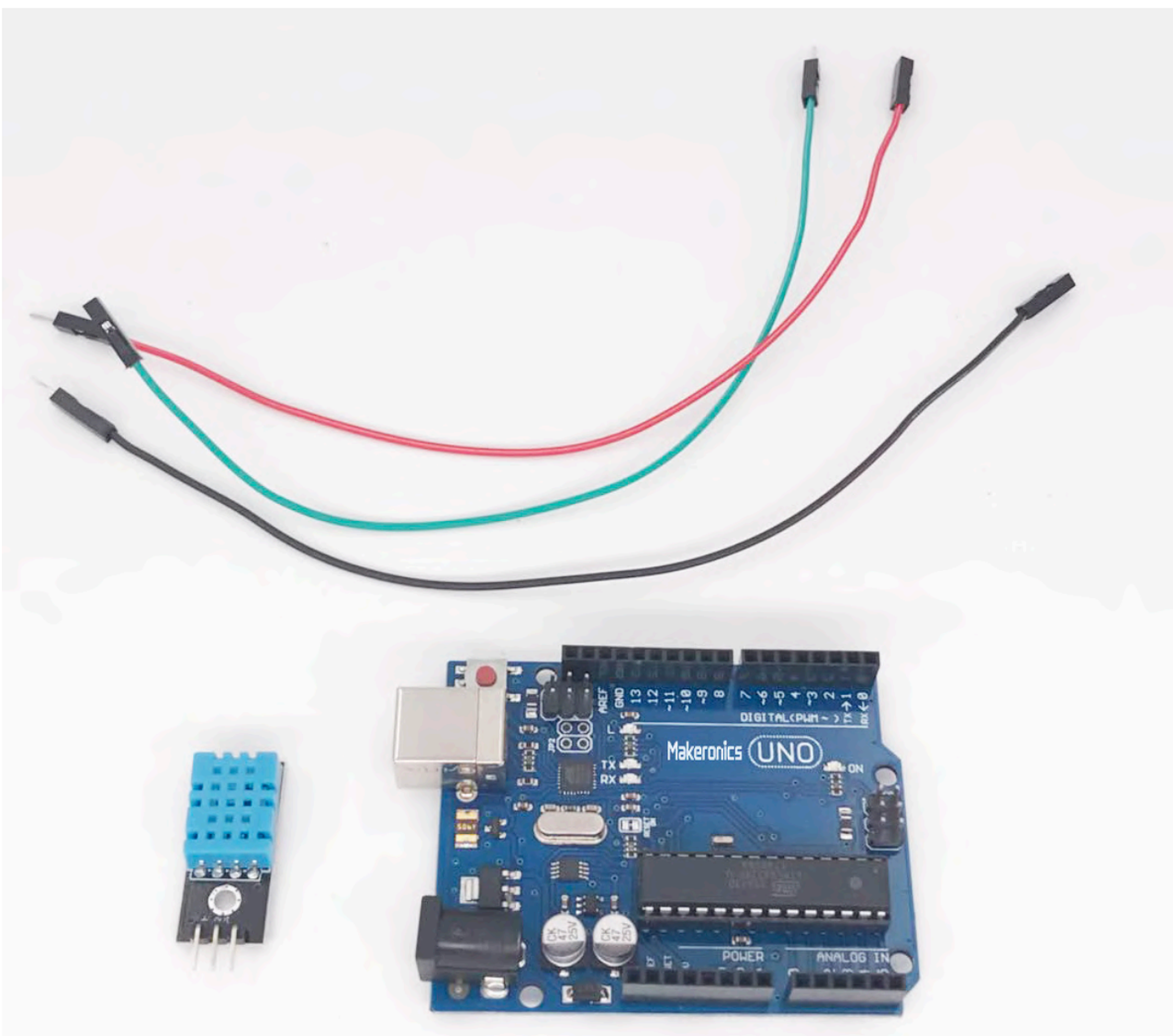
DHT11 Temperature and Humidity Sensor

15

In this tutorial we will learn how to use a DHT11 Temperature and Humidity Sensor.

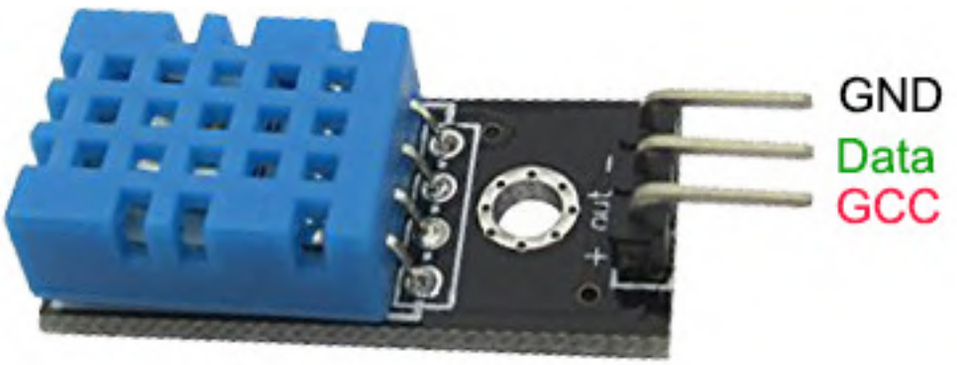
Component Required:

- 1 x Makeronics Uno R3
- 1 x DHT11 Temperature and Humidity module
- 3 x F-M wires (Female to Male DuPont wires)



Component Introduction:

Temp and humidity sensor:



The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds.

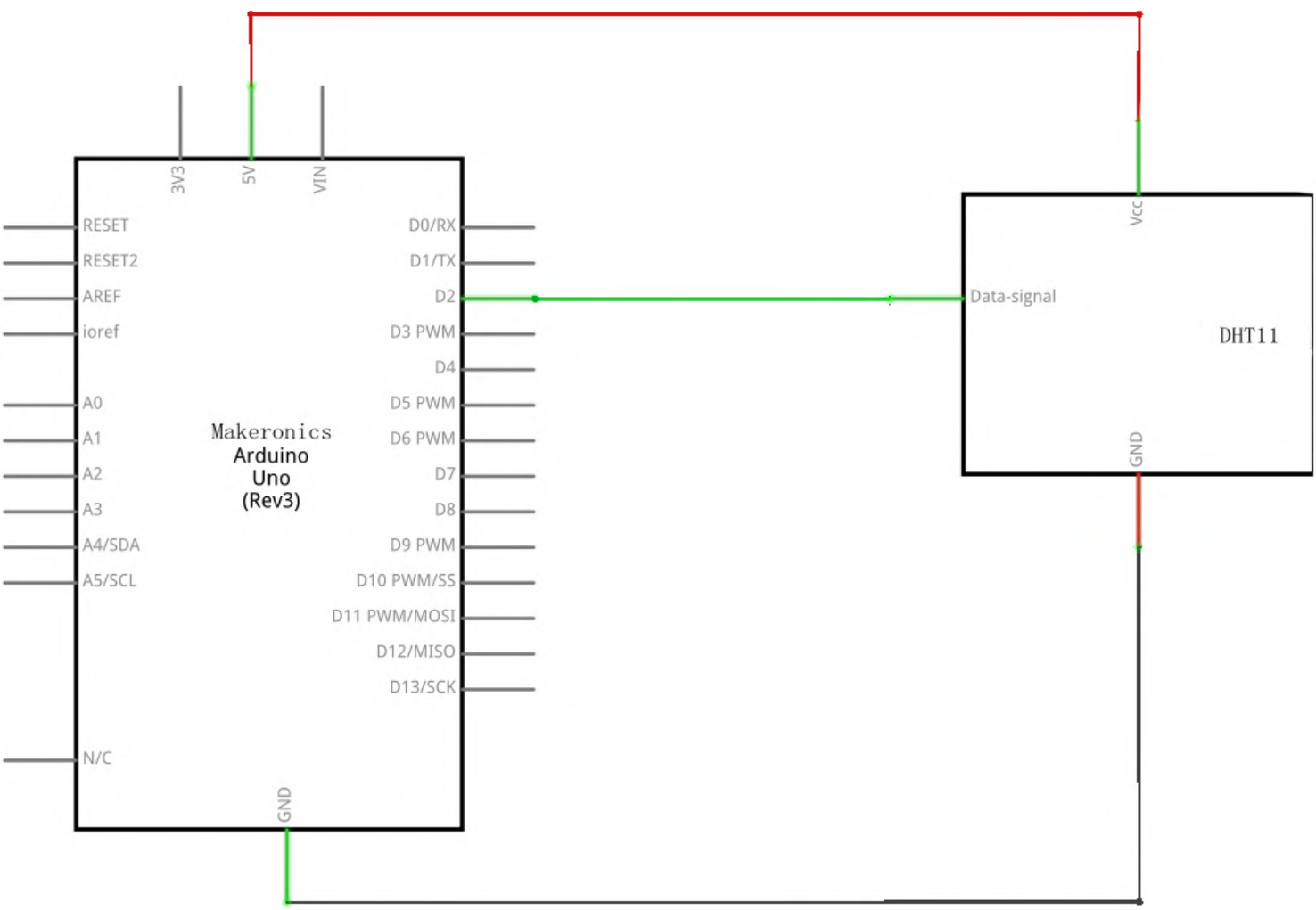
Product spec:

Resolution: 16Bit
Repeatability: ±1% RH
Accuracy: At 25℃ ±5% RH
Interchangeability: fully interchangeable
Response time: 1 / e (63%) of 25℃ 6s
1m / s air 6s
Hysteresis: <± 0.3% RH
Long-term stability: <± 0.5% RH / yr in
Temperature:
Resolution: 16Bit
Repeatability: ±0.2℃
Range: At 25℃ ±2℃
Response time: 1 / e (63%) 10S
Electrical Characteristics
Power supply: DC 3.5 ～ 5.5V
Supply Current: measurement 0.3mA standby 60μA
Sampling period: more than 2 seconds

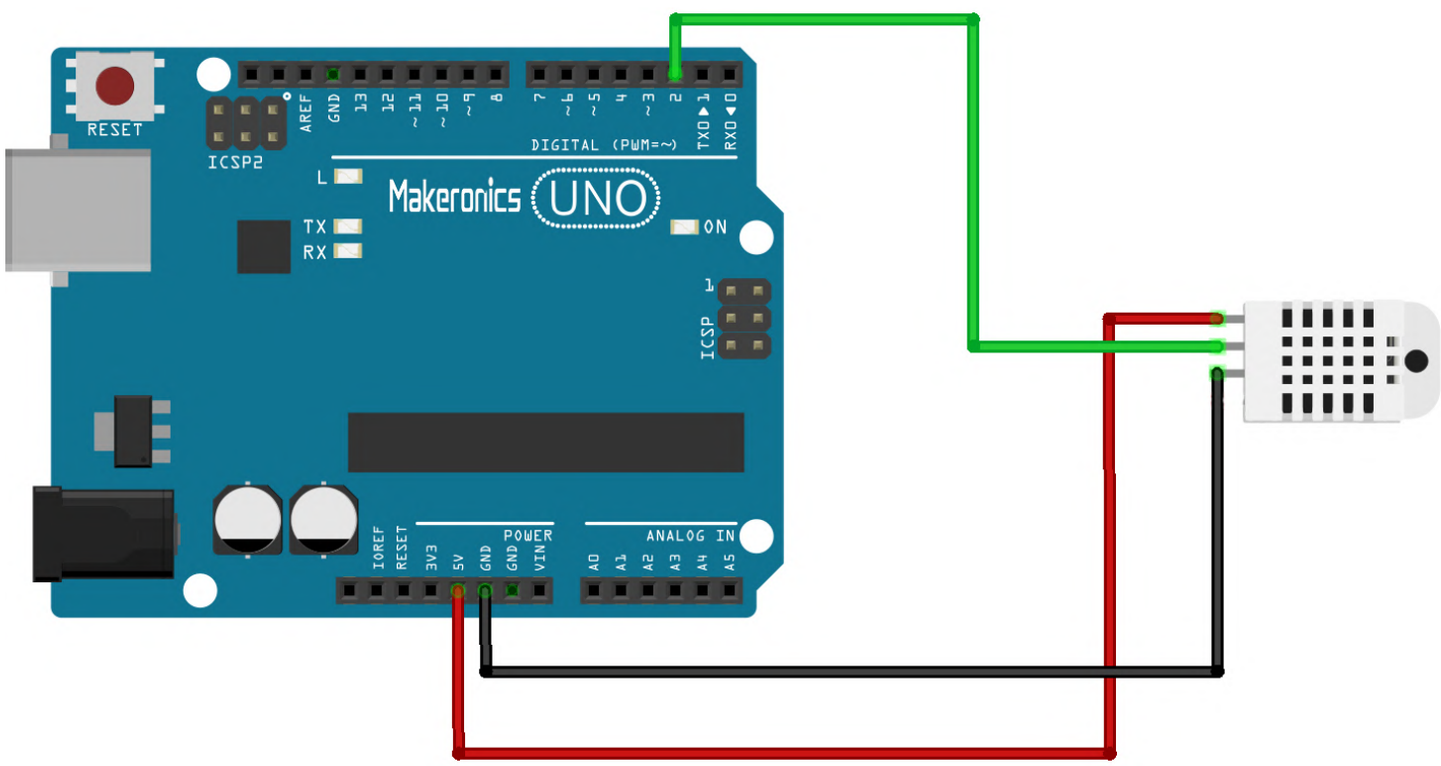
Pin Description:

1. the VDD power supply 3.5 ~ 5.5V DC
2. DATA serial data, a single bus
3. NC, empty pin
4. GND ground, the negative power

Connection Schematic:



Wiring diagram:

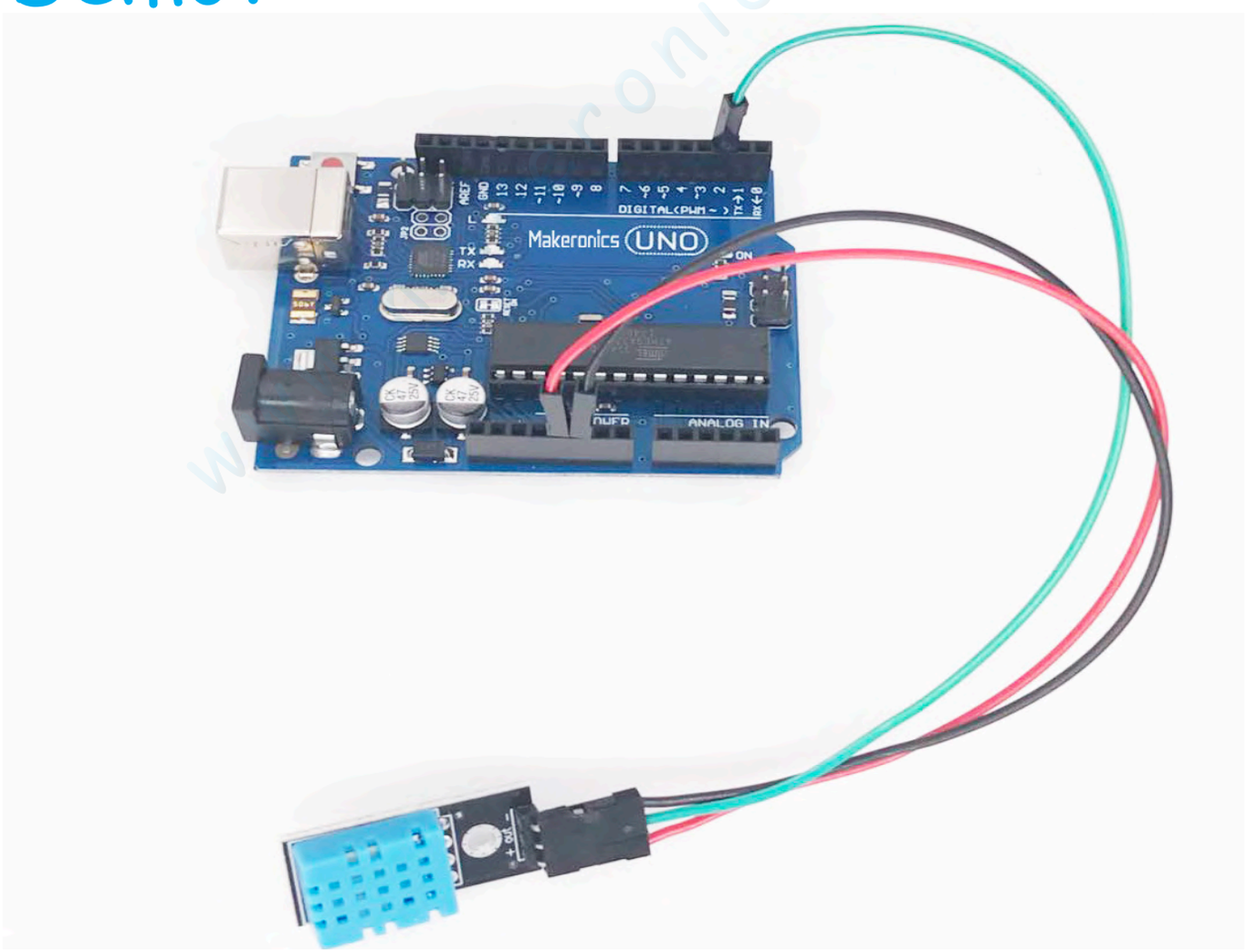


As you can see we only need 3 connections to the sensor, since one of the pin is not used.
The connections are: Voltage, Ground and Signal which can be connected to any Pin on our UNO.

Code:

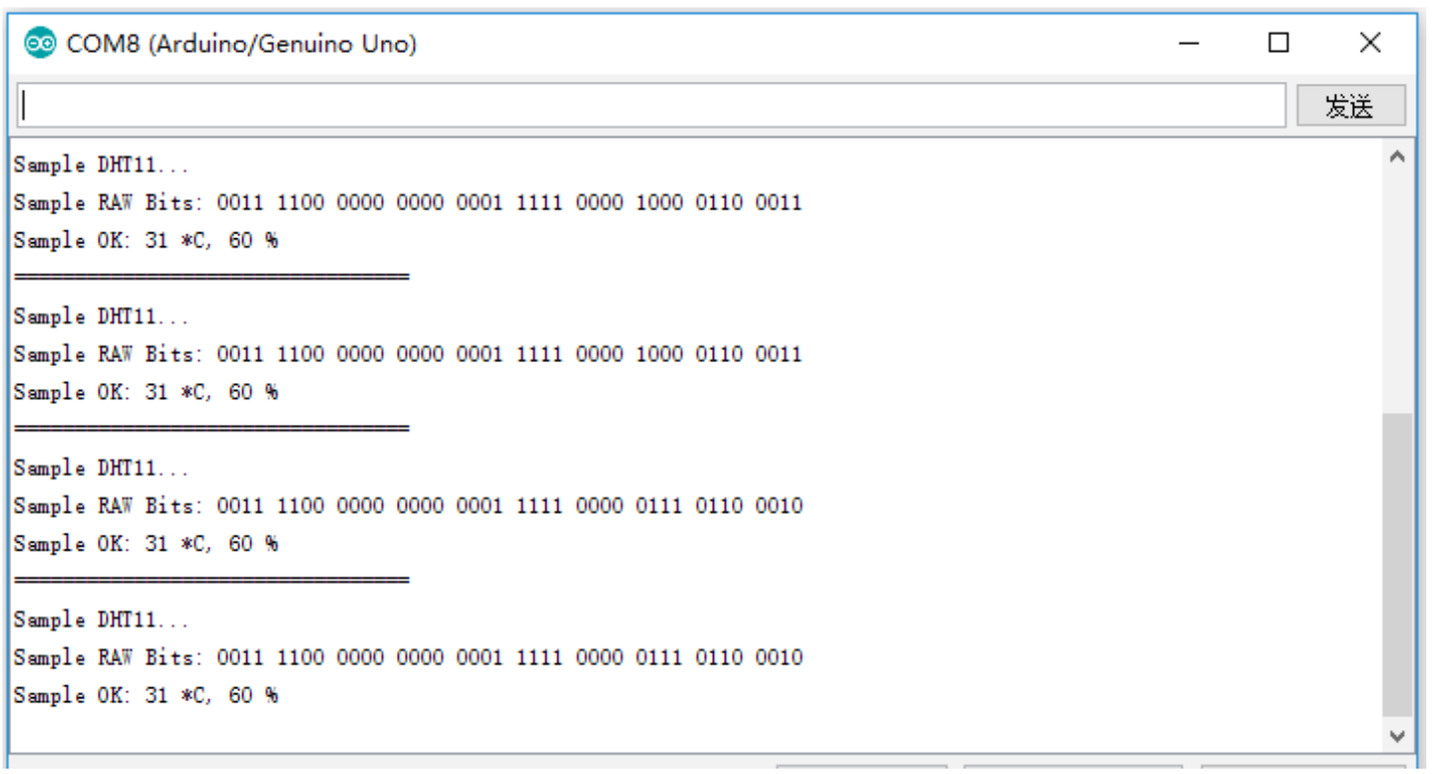
After wiring, please open the program in the code folder- Lesson 15 DHT11 Temperature and Humidity Sensor and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < SimpleDHT> library or re-install it, if necessary. Otherwise, your code won't work.
For details about the tutorial on the loading of library file, see Lesson 2.

Demo:



Upload the program then open the monitor, we can see the data as below: (It shows the temperature of the environment, we can see it is 31 degree)

Click the Serial Monitor button to turn on the serial monitor. The basics about the serial monitor are introduced in details in Lesson 2.



Project 13

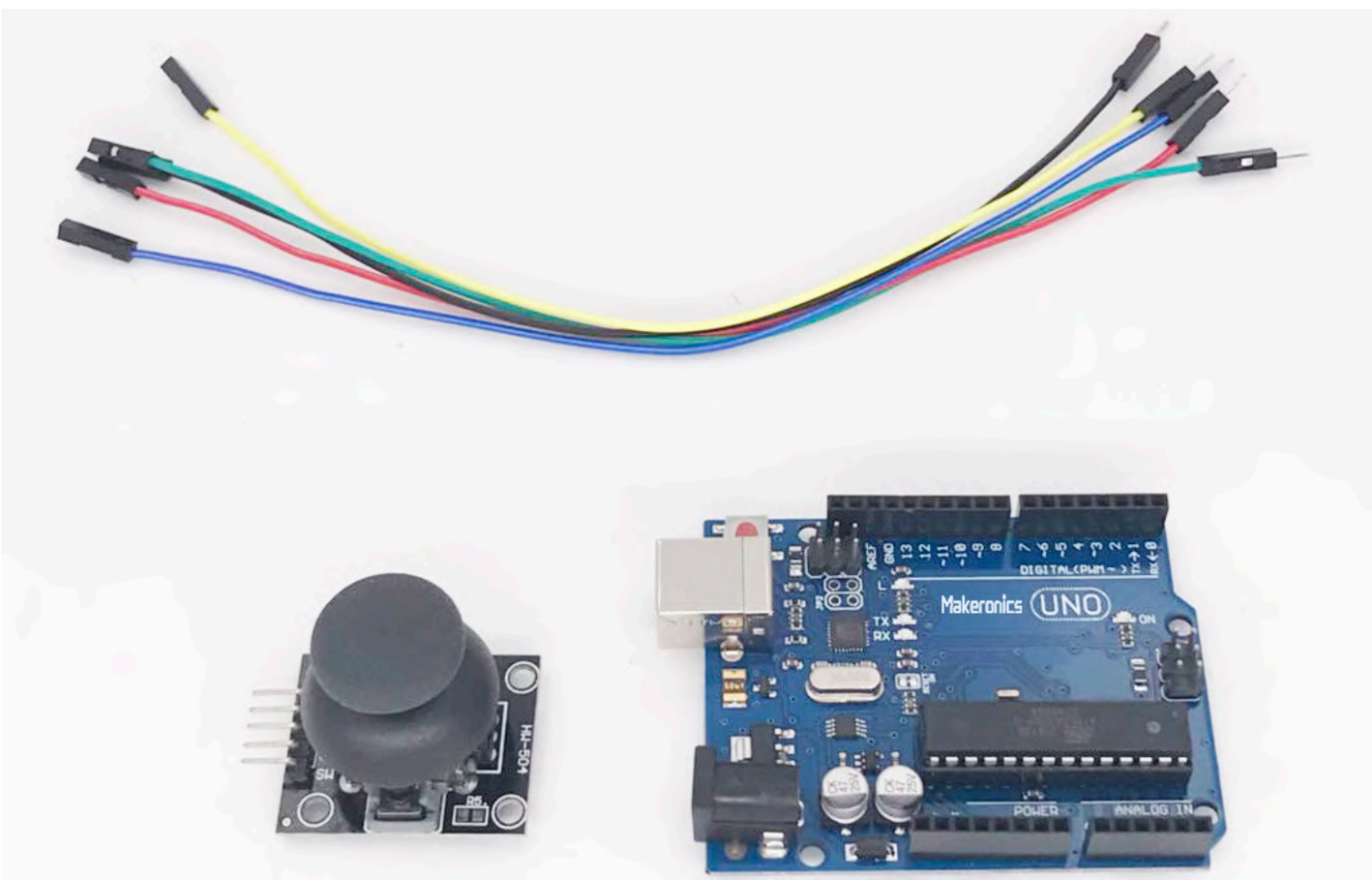
16

Analog Joystick Module

Analog joysticks are a great way to add some control in your projects. In this tutorial we will learn how to use the analog joystick module.

Component Required:

- 1 x Makeronics Uno R3
- 1 x Joystick module
- 5 x F-M wires (Female to Male DuPont wires)



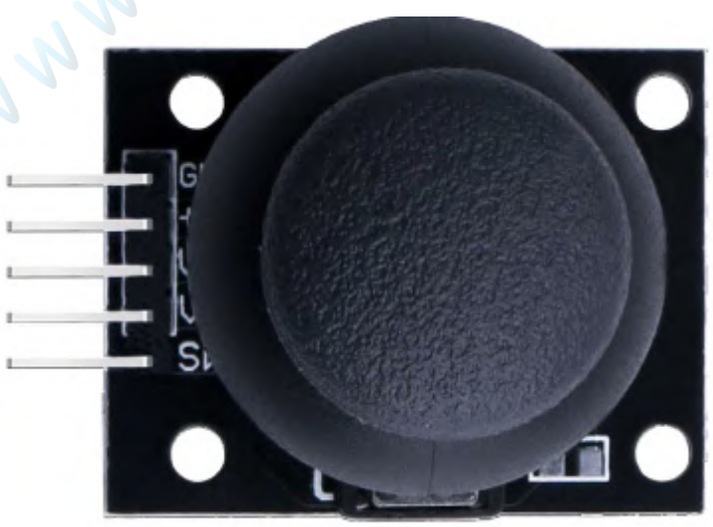
Component Introduction:

Joystick:

The joystick is basically two potentiometers and a button that allow us to measure the movement of the stick in two dimensions.

Potentiometers are variable resistors and act as sensors that provide us with a voltage that varies depending on the rotation of the device around its shaft. So as you move the joystick around its center, its resistance—and therefore its output—varies. The outputs from the potentiometers are analog, so they can have a value only between 0 and 1,023 when read by the analog pin of the Arduino.

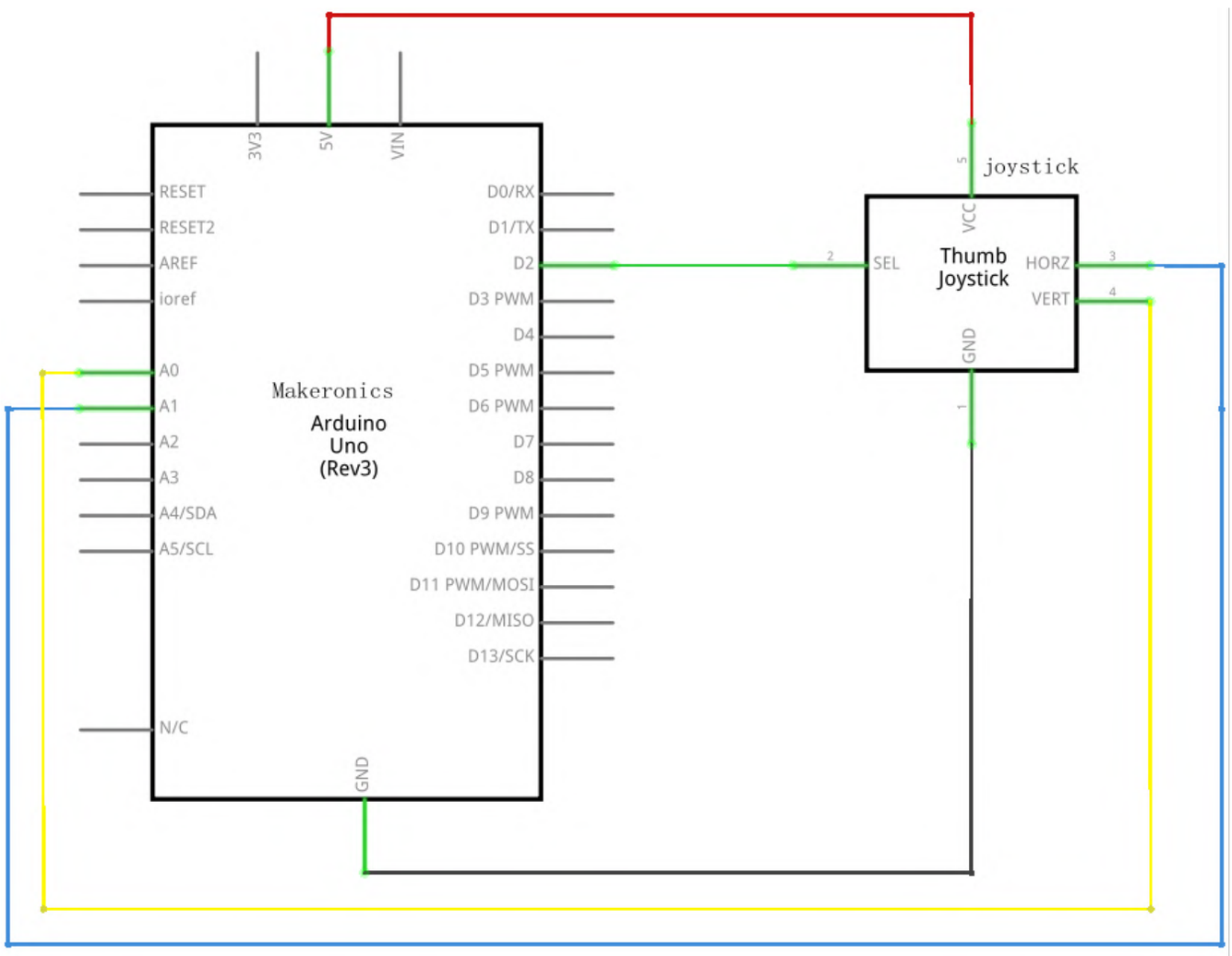
A joystick typically has five pins: VRx (the x-axis signal), VRy (the y-axis signal), SW (a pushbutton we won't be using in this project), and GND and +5V for power.



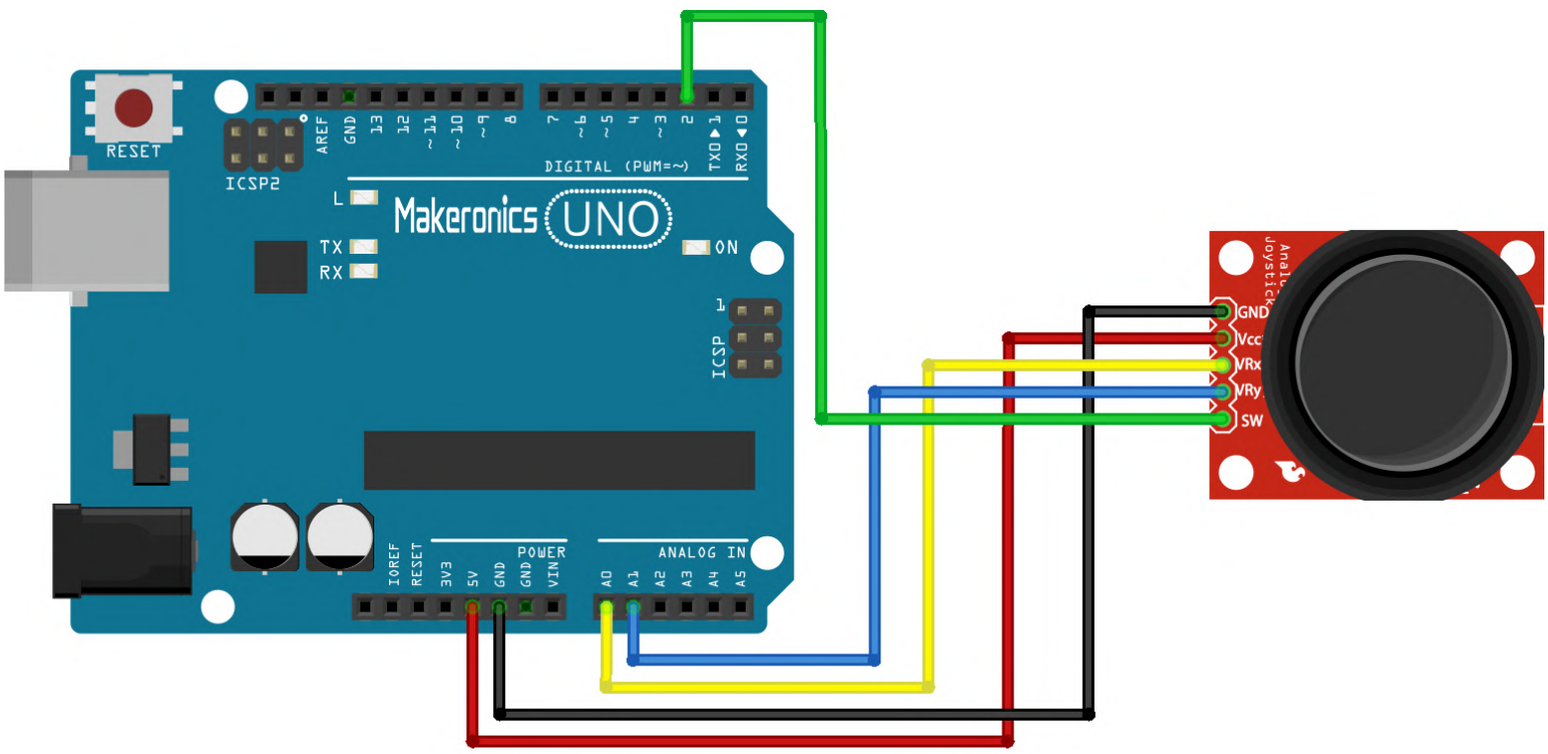
The thumb stick is analog and should provide more accurate readings than simple 'directional' joysticks tact use some forms of buttons, or mechanical switches. Additionally, you can press the joystick down (rather hard on mine) to activate a 'press to select' push-button.

We have to use analog Arduino pins to read the data from the X/Y pins, and a digital pin to read the button. The Key pin is connected to ground, when the joystick is pressed down, and is floating otherwise. To get stable readings from the Key /Select pin, it needs to be connected to VCC via a pull-up resistor. The built in resistors on the Arduino digital pins can be used. For a tutorial on how to activate the pull-up resistors for Arduino pins, configured as inputs.

Connection Schematic:



Wiring diagram:



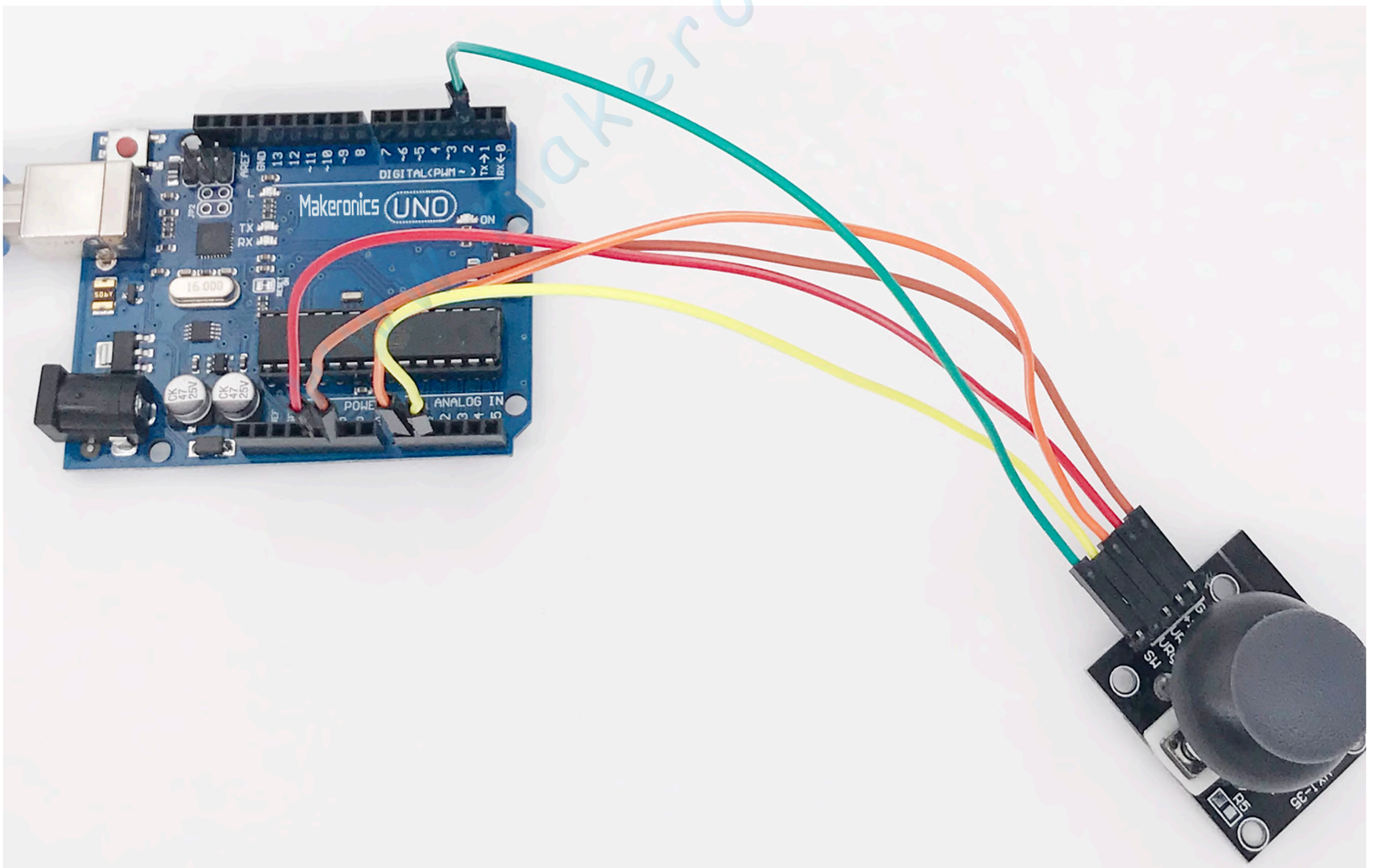
We need 5 connections to the joystick.
The connections are: Key, Y, X, Voltage and Ground.
"Y and X" are Analog and "Key" is Digital. If you don't need the switch then you can use only 4 pins.

Code:

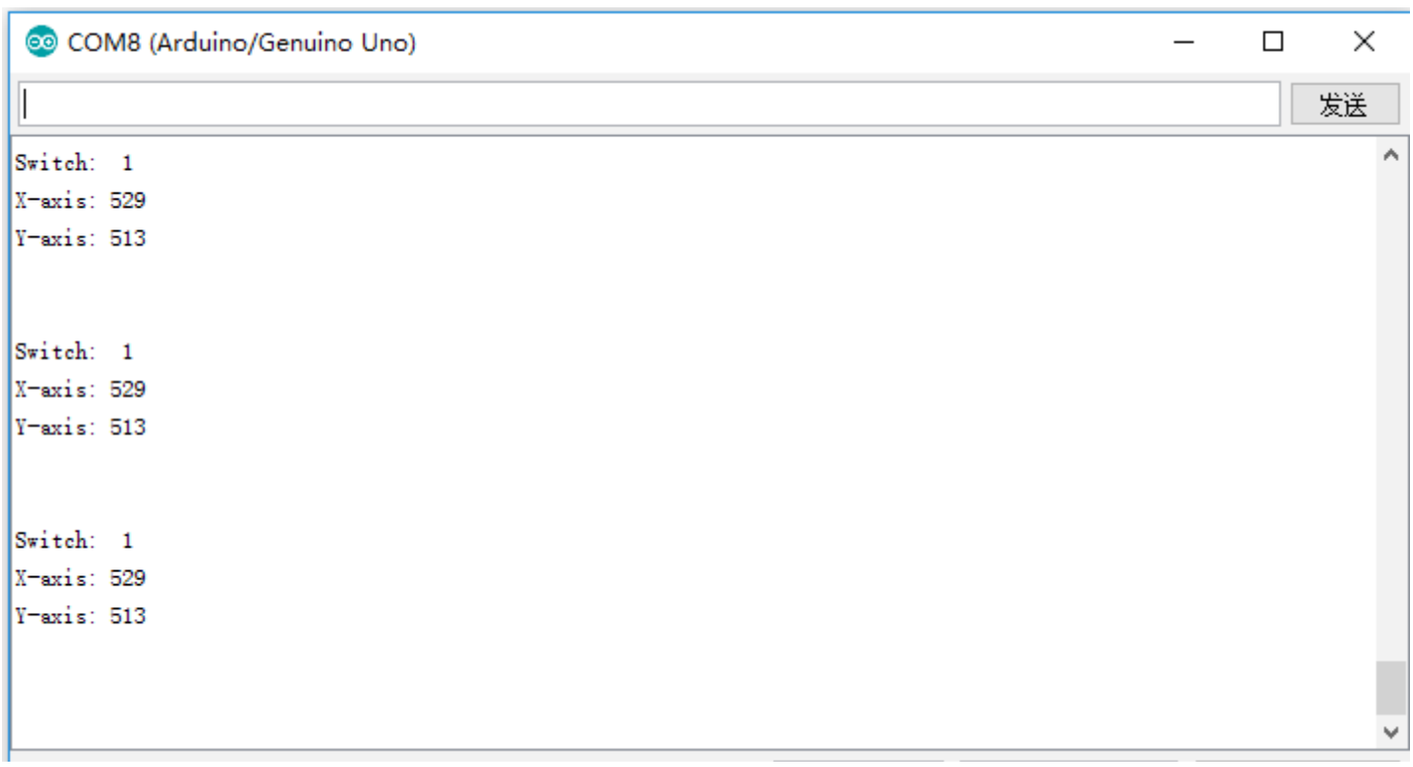
After wiring, please open the program in the code folder- Lesson 16 Analog Joystick Module and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

When the joystick is in the resting position or middle, it should return a value of about 512.
The range of values goes from 0 to 1024.

Demo:



Open the monitor then you can see the data as blow:
Click the Serial Monitor button to turn on the serial monitor. The basics about the serial monitor are introduced in details in Lesson 2.



Project 14

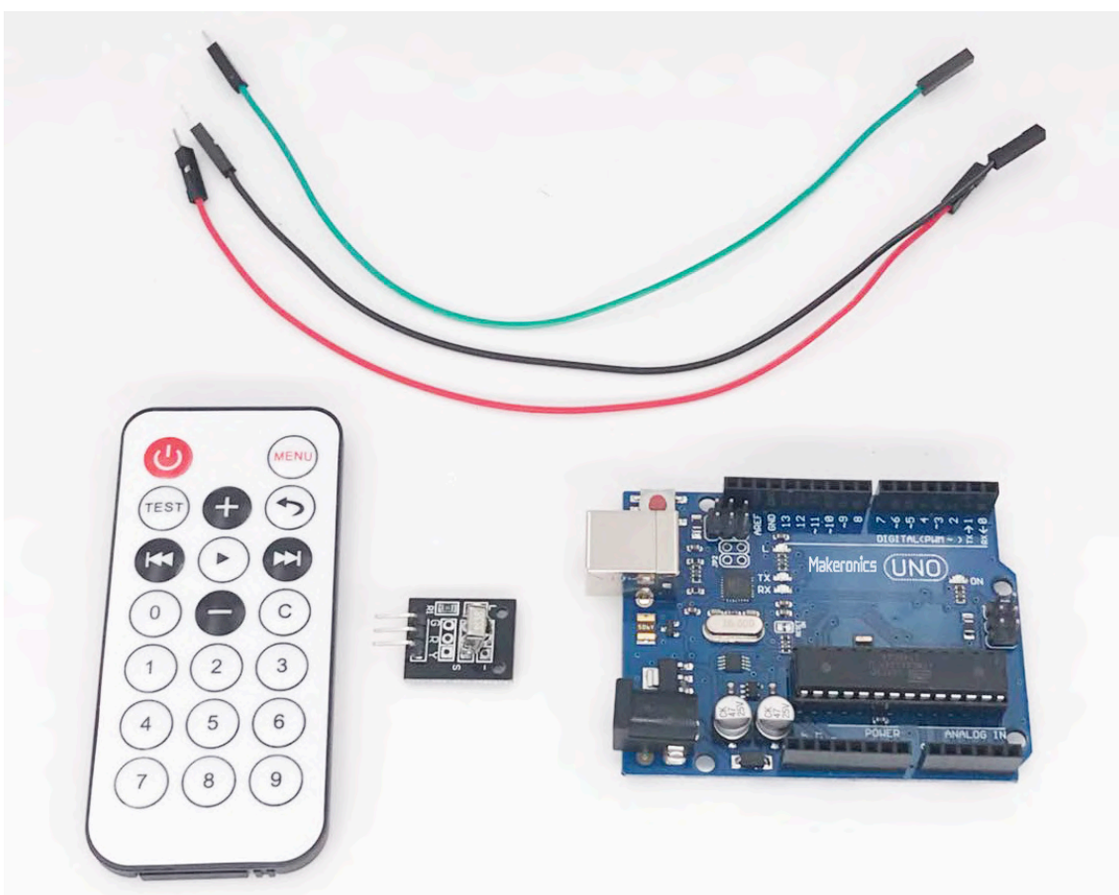
17

IR Receiver Module

IR Receiver Module is a great way to add remote control in your projects. In this tutorial we will learn how to use the IR Receiver Module.

Component Required:

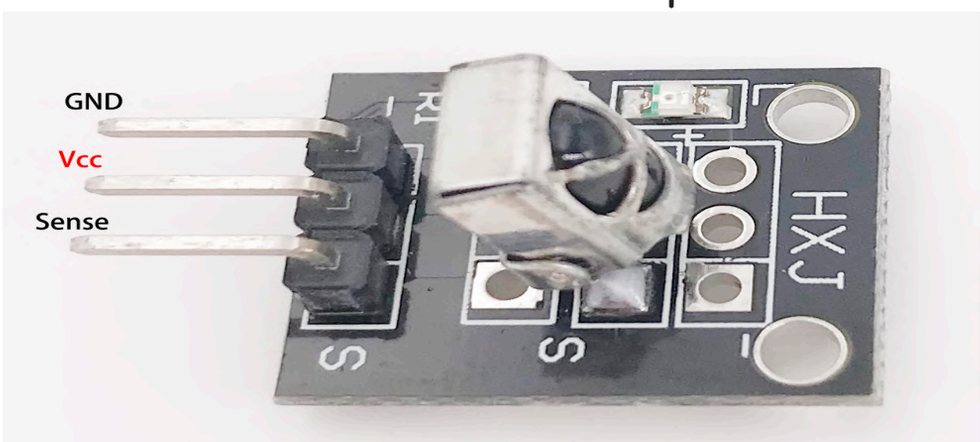
- 1 x Makeronics Uno R3
- 1 x IR receiver module
- 1 x IR remote
- 3 x F-M wires (Female to Male DuPont wires)



Component Introduction:

IR RECEIVER SENSOR

IR detectors are little microchips with a photocell that are tuned to listen to infrared light. They are almost always used for remote control detection - every TV and DVD player has one of these in the front to listen for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels. IR light is not visible to the human eye, which means it takes a little more work to test a setup.

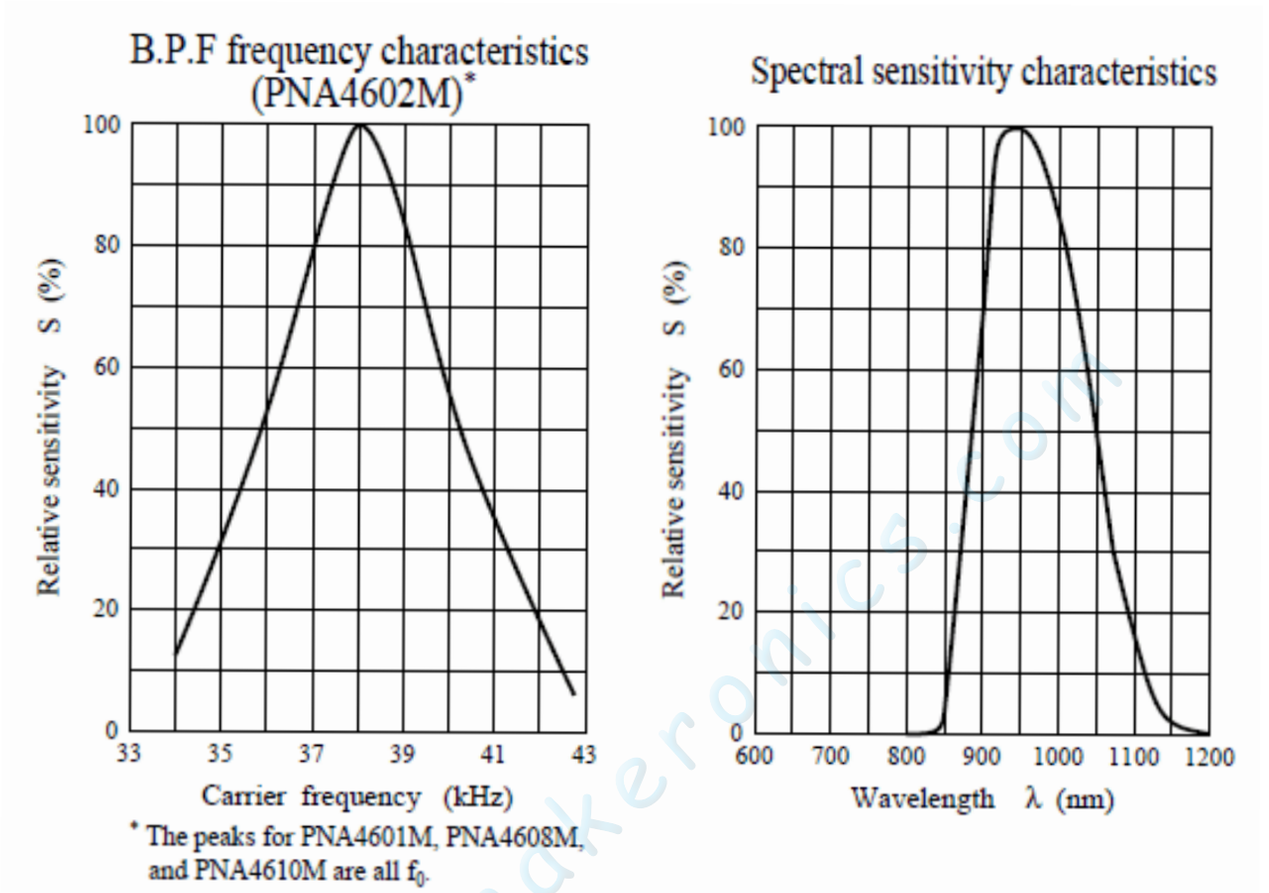


There are a few difference between these and say a CdS Photocells:

IR detectors are specially filtered for IR light, they are not good at detecting visible light. On the other hand, photocells are good at detecting yellow/green visible light, and are not good at IR light.

IR detectors have a demodulator inside that looks for modulated IR at 38 KHz. Just shining an IR LED won't be detected, it has to be PWM blinking at 38KHz. Photocells do not have any sort of demodulator and can detect any frequency (including DC) within the response speed of the photocell (which is about 1KHz)

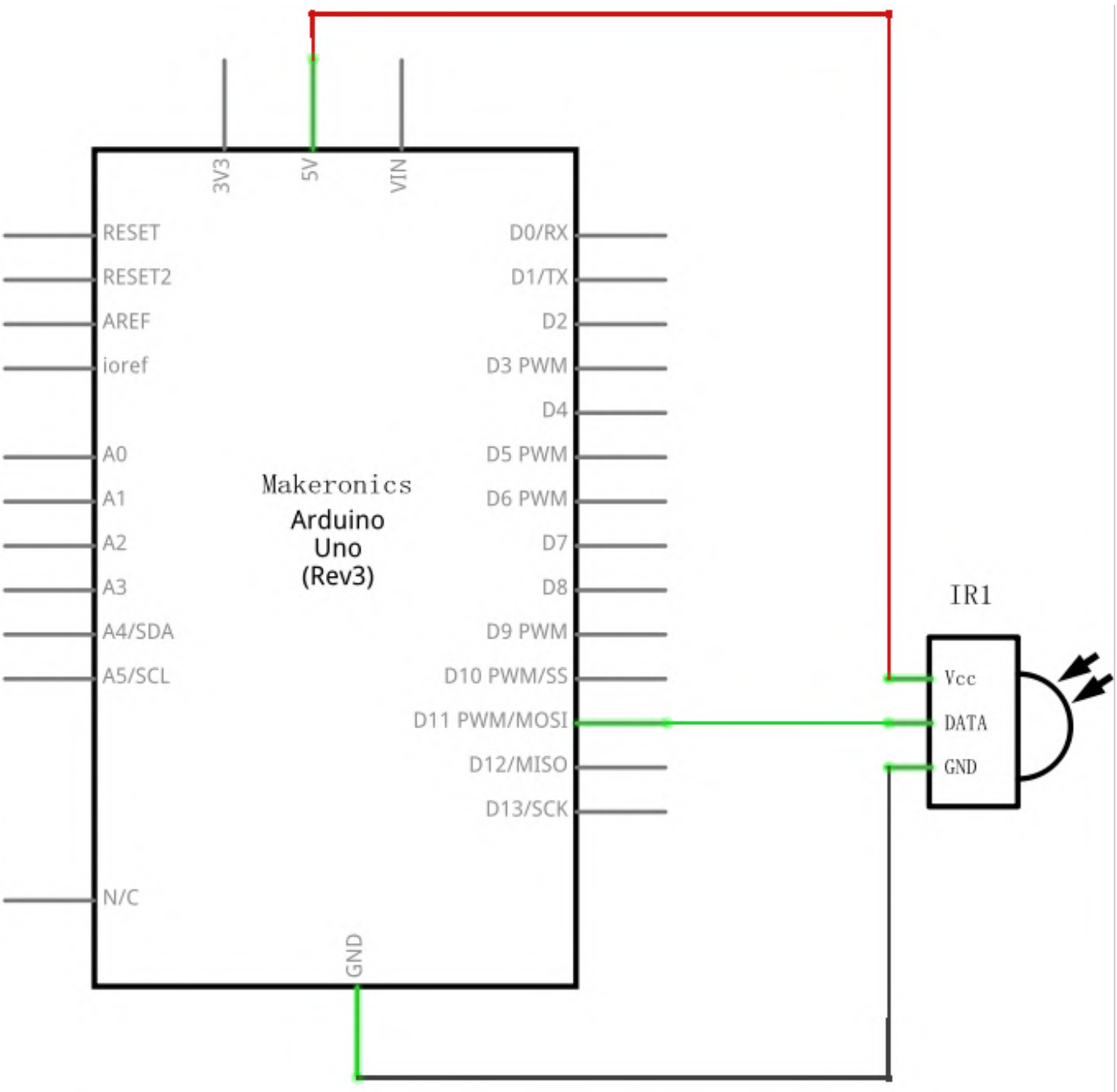
IR detectors are digital out - either they detect 38KHz IR signal and output low (0V) or they do not detect any and output high (5V). Photocells act like resistors, the resistance changes depending on how much light they are exposed to.



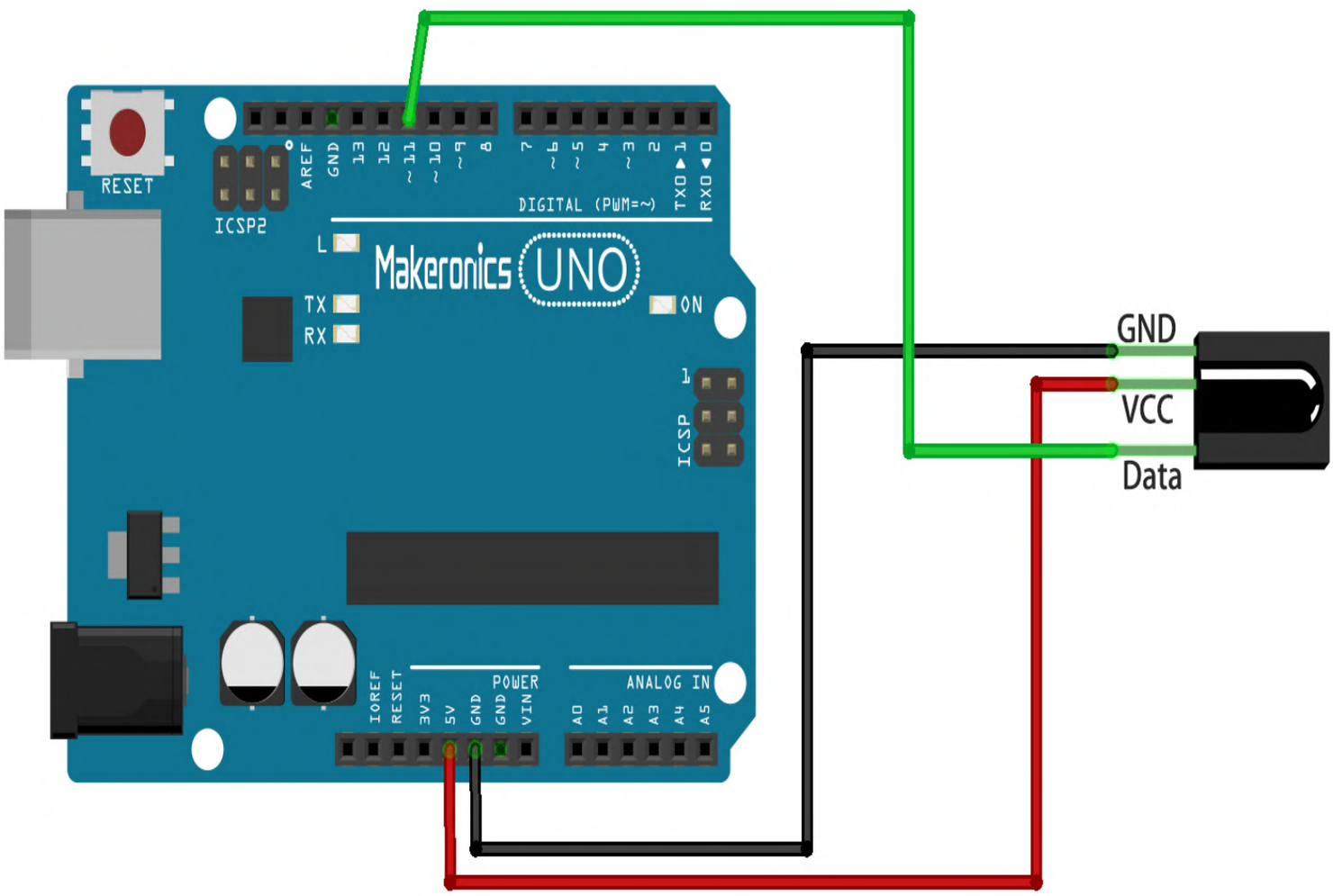
As you can see from these datasheet graphs, the peak frequency detection is at 38 KHz and the peak LED color is 940 nm. You can use from about 35 KHz to 41 KHz but the sensitivity will drop off so that it won't detect as well from afar. Likewise, you can use 850 to 1100 nm LEDs but they won't work as well as 900 to 1000nm so make sure to get matching LEDs! Check the datasheet for your IR LED to verify the wavelength.

Try to get a 940nm - remember that 940nm is not visible light!

Connection Schematic:



Wiring diagram:



There are 3 connections to the IR Receiver.
The connections are: Signal, Voltage and Ground.
The "-" is the Ground, "S" is signal, and middle pin is Voltage 5V.

When you press a button on the remote, it sends out a digital value that is picked up by the receiver. This value is different for each button. We'll decode the values for each button with the Arduino.

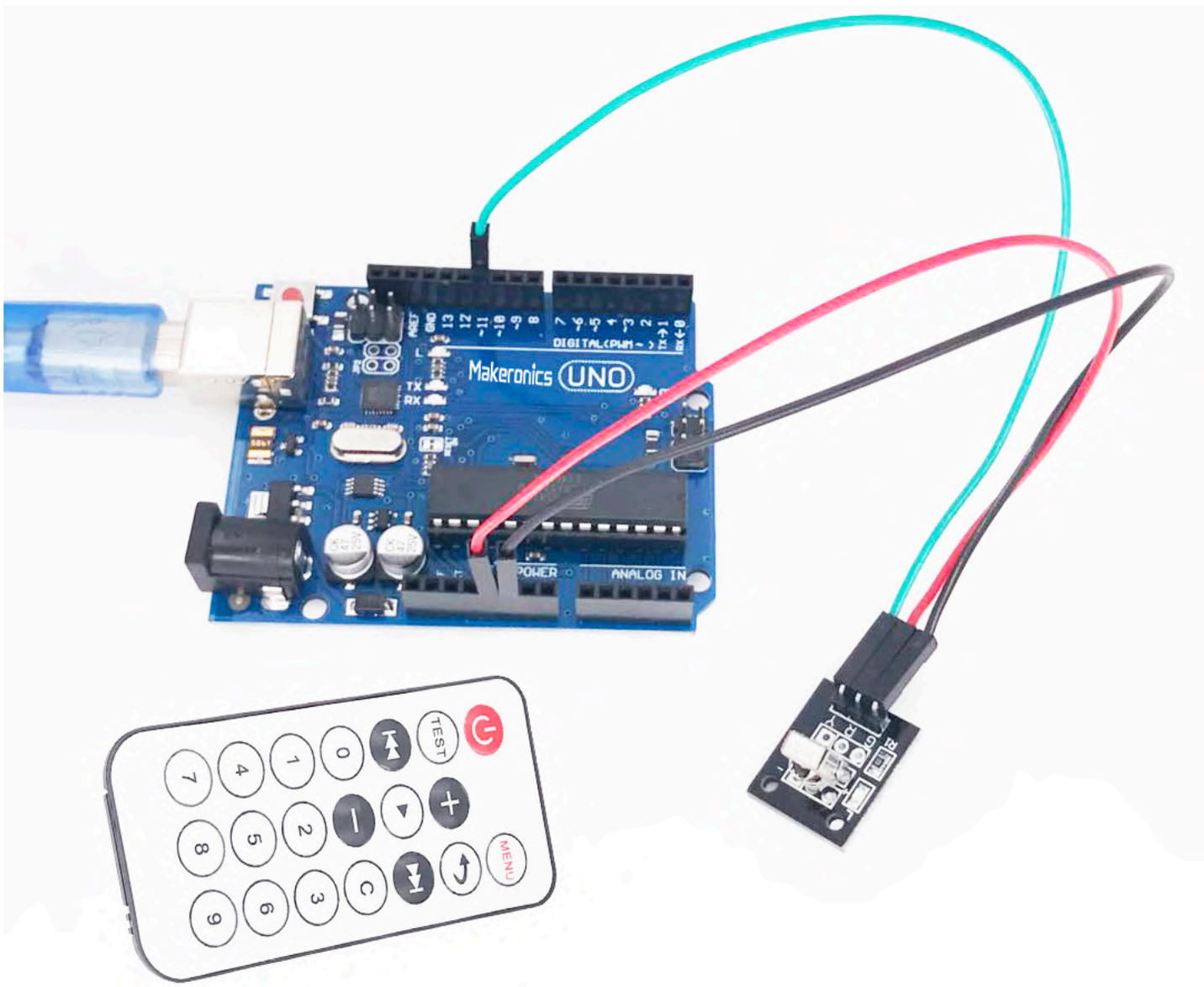
Code:

After wiring, please open the program in the code folder- Lesson 17 IR Receiver Module and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

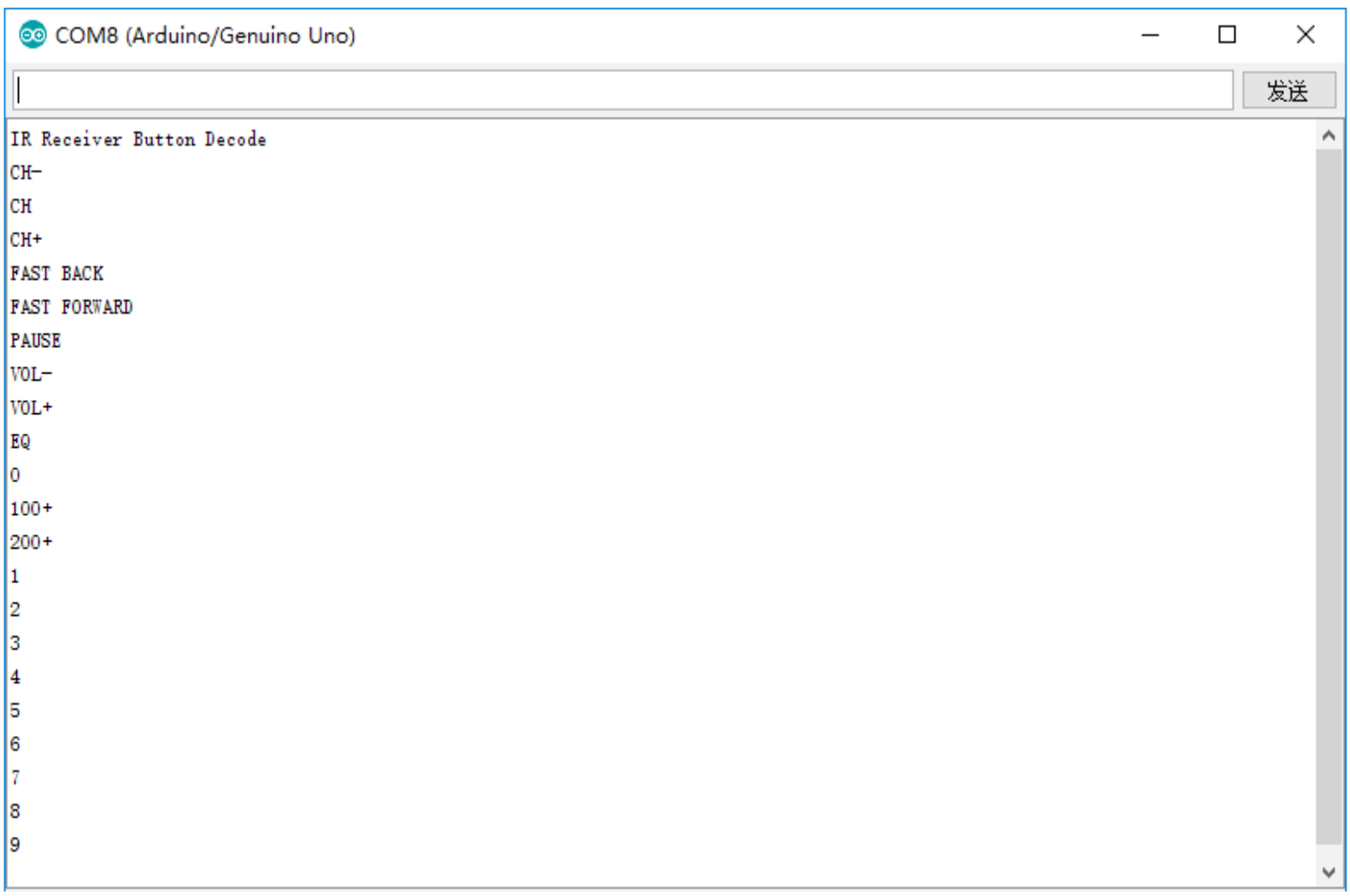
Before you can run this, make sure that you have installed the < IRremote > library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

Next we will move the <RobotIRremote> out of the Library folder, we do this because that library conflicts with the one we will be using. You can just drag it back inside the library folder once you are done programming your microcontroller.
Once you have installed the Library, just go ahead and restart your IDE Software.

Demo:



Open the monitor then you can see the data as blow:
Click the Serial Monitor button to turn on the serial monitor. The basics about the serial monitor are introduced in details in Lesson 2.



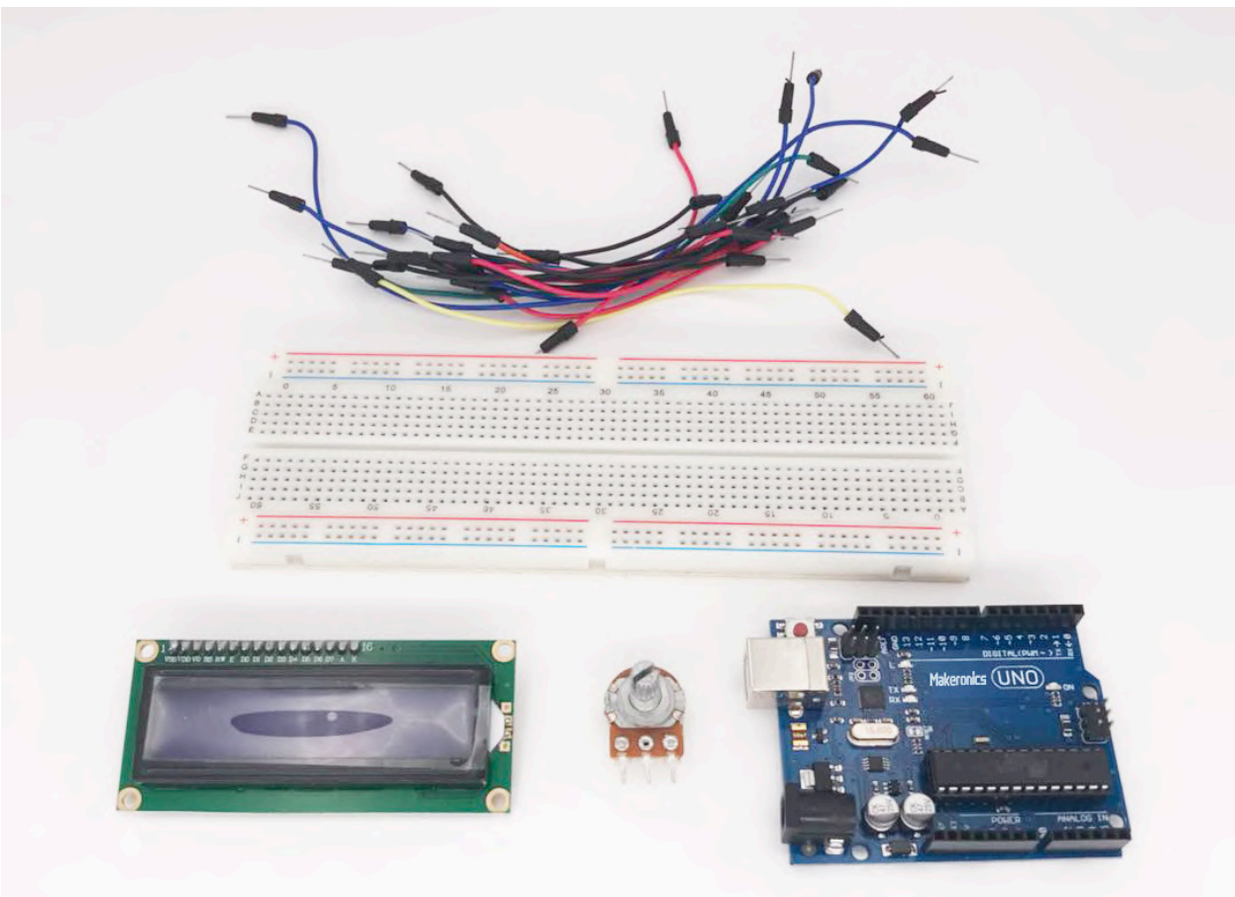
Project 15

18 LCD Display

In this lesson, we will run the program for the LCD library, but in the next lesson, we will get our display to show the temperature, using sensors.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points Breadboard
- 1 x LCD1602 module
- 1 x Potentiometer (10k)
- 18 x M-M wires (Male to Male jumper wires)



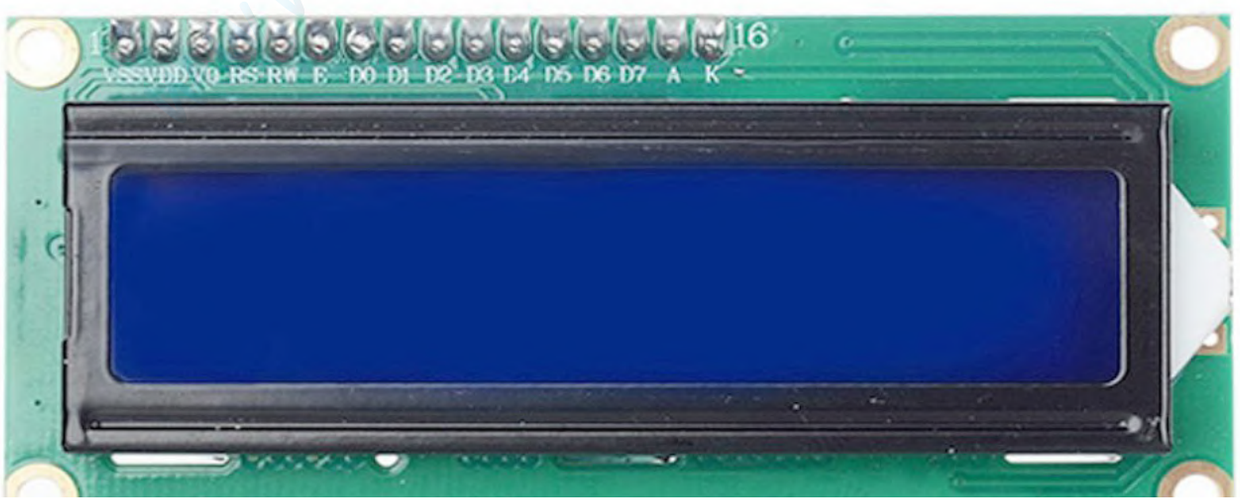
Component Introduction:

LCD1602

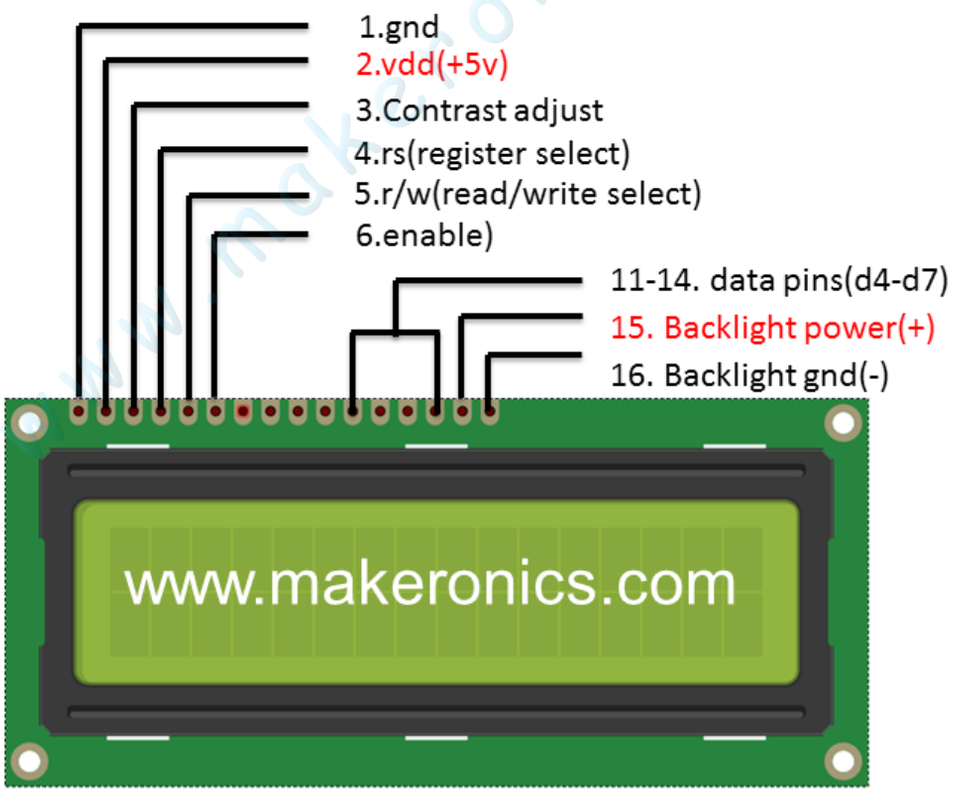
LCD is short for liquid crystal display. Invented over 40 years ago, liquid crystal technology is used in digital watches, alarm clocks, projectors, televisions, computer monitors, and more.

The LCD you'll use in this project is a simple monochromatic display, meaning it displays only one color. Beneath the screen of the display is a layer of liquid crystal. This is a unique chemical that, when a small electric current is applied to it, changes from transparent to opaque. Combined with a backlight or a reflective mirror, liquid crystal is used to build very simple displays. Light comes through or is blocked depending on which areas of the liquid crystal electricity is applied to—which means you can make shapes if you can control the current.

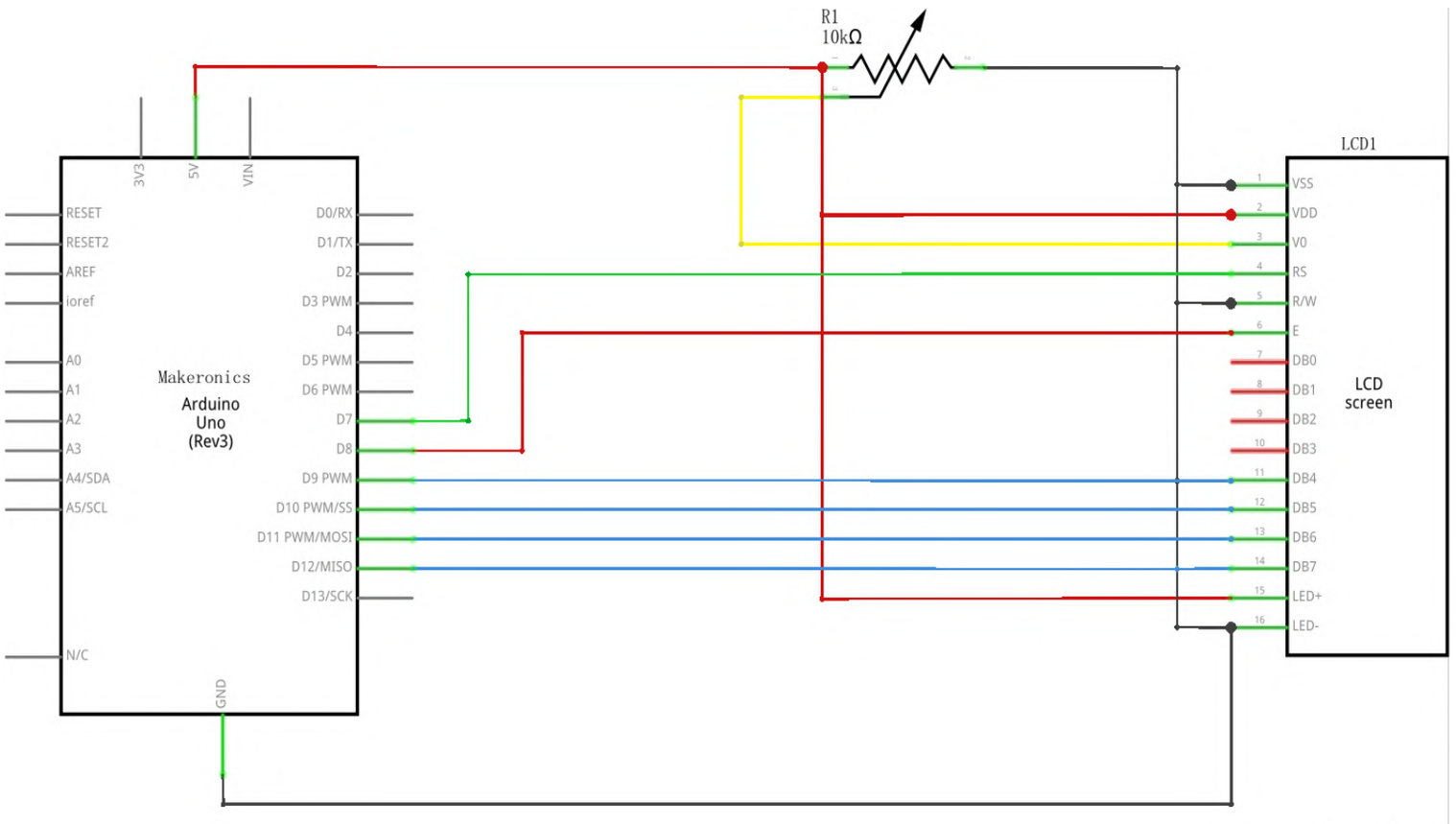
The 16 × 2 character LCD displays up to 32 characters of information, each of which is broken down into a 5 × 8 pixel matrix. Each individual pixel can be made either opaque or transparent depending on the applied electric current, controlled by the Arduino.



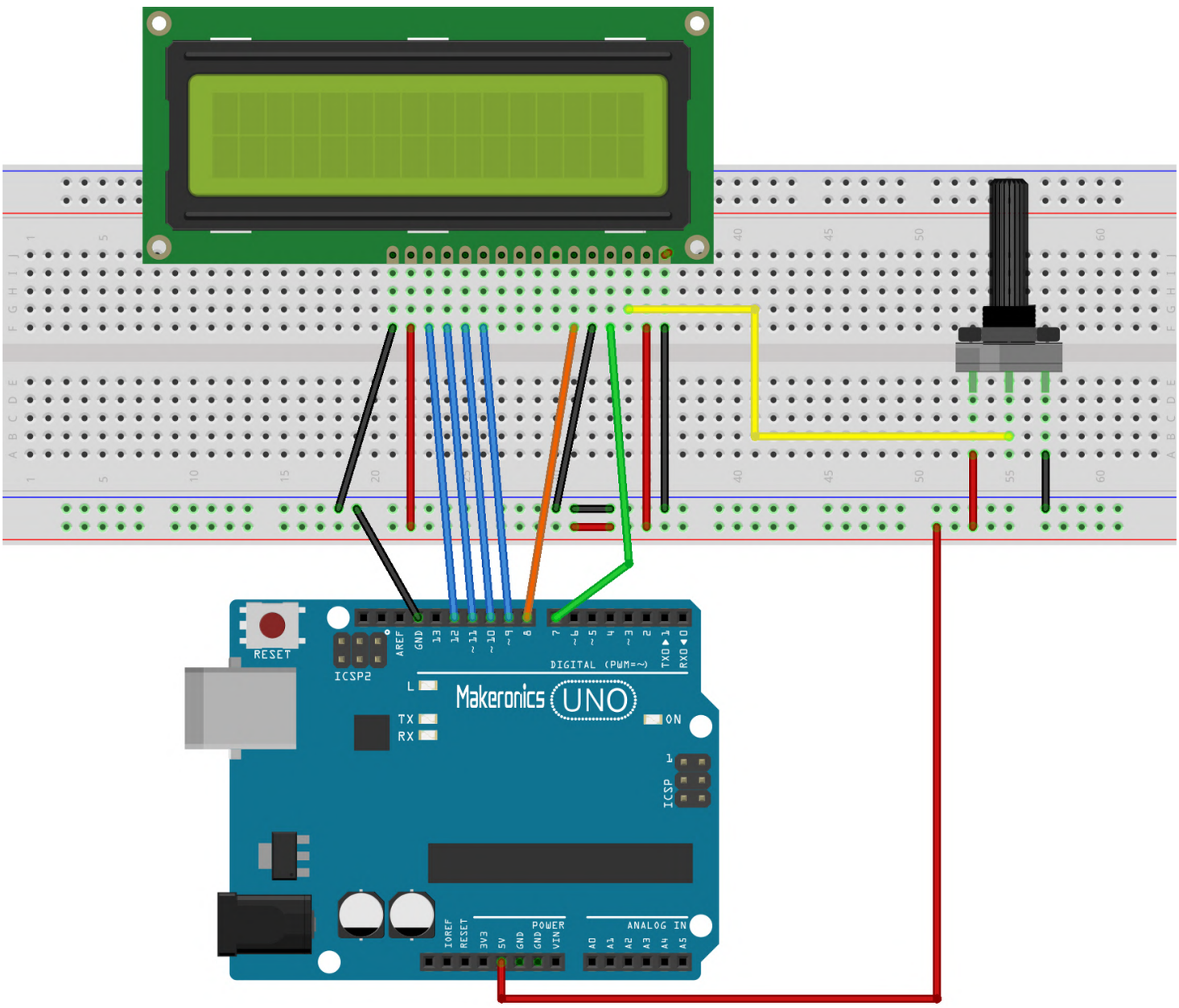
Introduction to the pins of LCD1602:
VSS: A pin that connects to ground
VDD: A pin that connects to a +5V power supply
VO: A pin that adjust the contrast of LCD1602
RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
R/W: A Read/Write pin that selects reading mode or writing mode
E: An enabling pin that, when supplied with low-level energy, causes the LDC module to execute relevant instructions.
D0-D7: Pins that read and write data
A and K: Pins that control the LED backlight



Connection Schematic:



Wiring diagram:



There are 40 individual pixels in a single character, each controlled by the Arduino, meaning there are 1,280 different control lines! Thankfully, the LCD used in this project has a special parallel interface LCD driver IC by Hitachi called the HD44780. This chip allows you to display almost any character on the screen using just six control lines from the Arduino.

Rather than having to control each of the 40 pixels for each character separately, the HD44780 driver chip interprets data sent over by the Arduino using four data lines and two control lines and converts this into the character to display. To further simplify the interface, the Arduino community has written an LCD library for writing code to the LCD. We'll look at that in the code.

Code:

After wiring, please open the program in the code folder- Lesson 18 LCD Display and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

Before you can run this, make sure that you have installed the < LiquidCrystal > library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

Upload the code to your Arduino board and you should see the message 'hello, world' displayed, followed by a number that counts up from zero.
The first thing of note in the sketch is the line:

```
#include <LiquidCrystal.h>
```

This tells Arduino that we wish to use the Liquid Crystal library.

Next we have the line that we had to modify. This defines which pins of the Arduino are to be connected to which pins of the display.

```
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

After uploading this code, make sure the backlight is lit up, and adjust the potentiometer all the way around until you see the text message
In the 'setup' function, we have two commands:

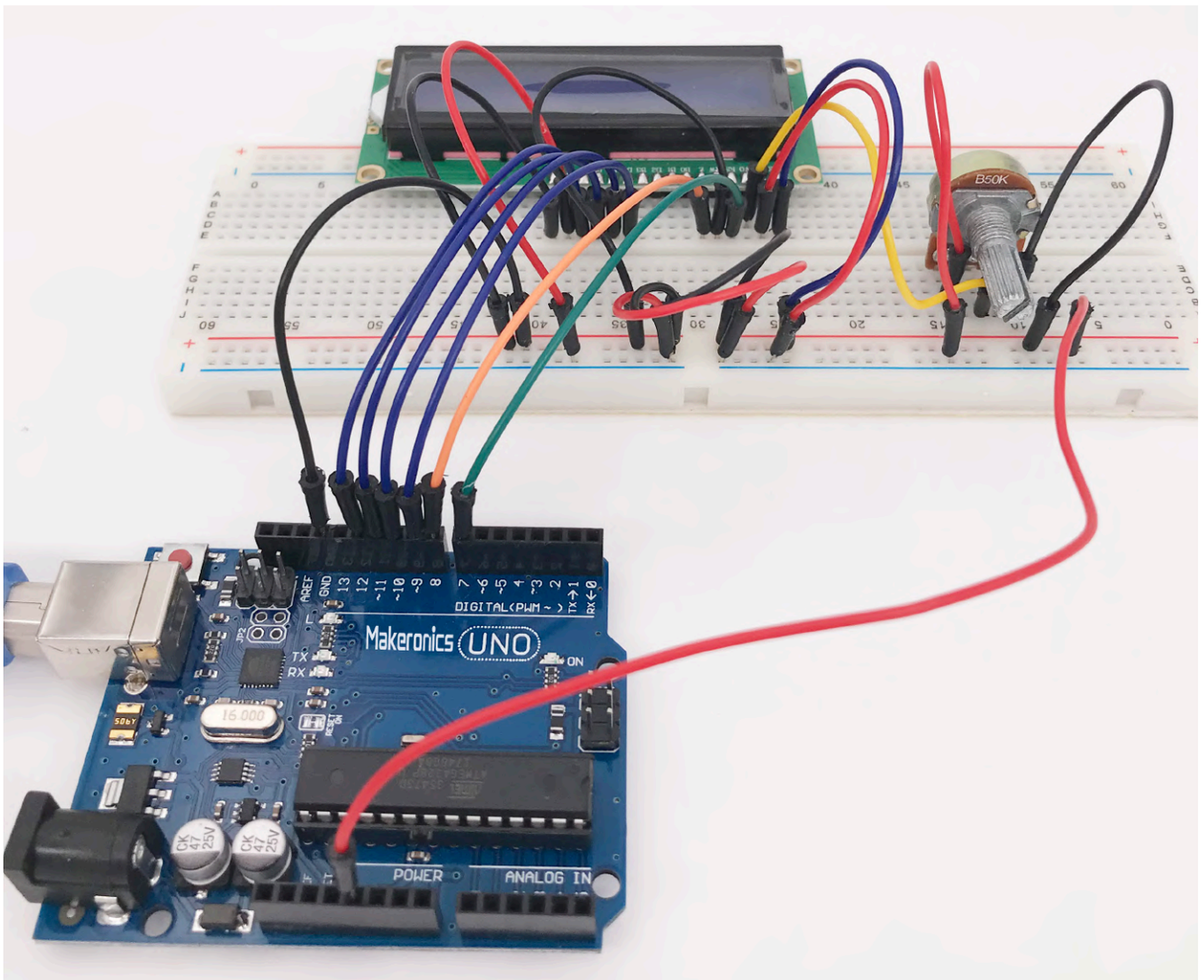
```
lcd.begin(16, 2);
lcd.print("Hello, World!");
```

The first tells the Liquid Crystal library how many columns and rows the display has. The second line displays the message that we see on the first line of the screen.
In the 'loop' function, we aso have two commands:

```
lcd.setCursor(0, 1);
lcd.print(millis()/1000);
```

The first sets the cursor position (where the next text will appear) to column 0 & row 1. Both column and row numbers start at 0 rather than 1.
The second line displays the number of milliseconds since the Arduino was reset.

Demo:



Project 16

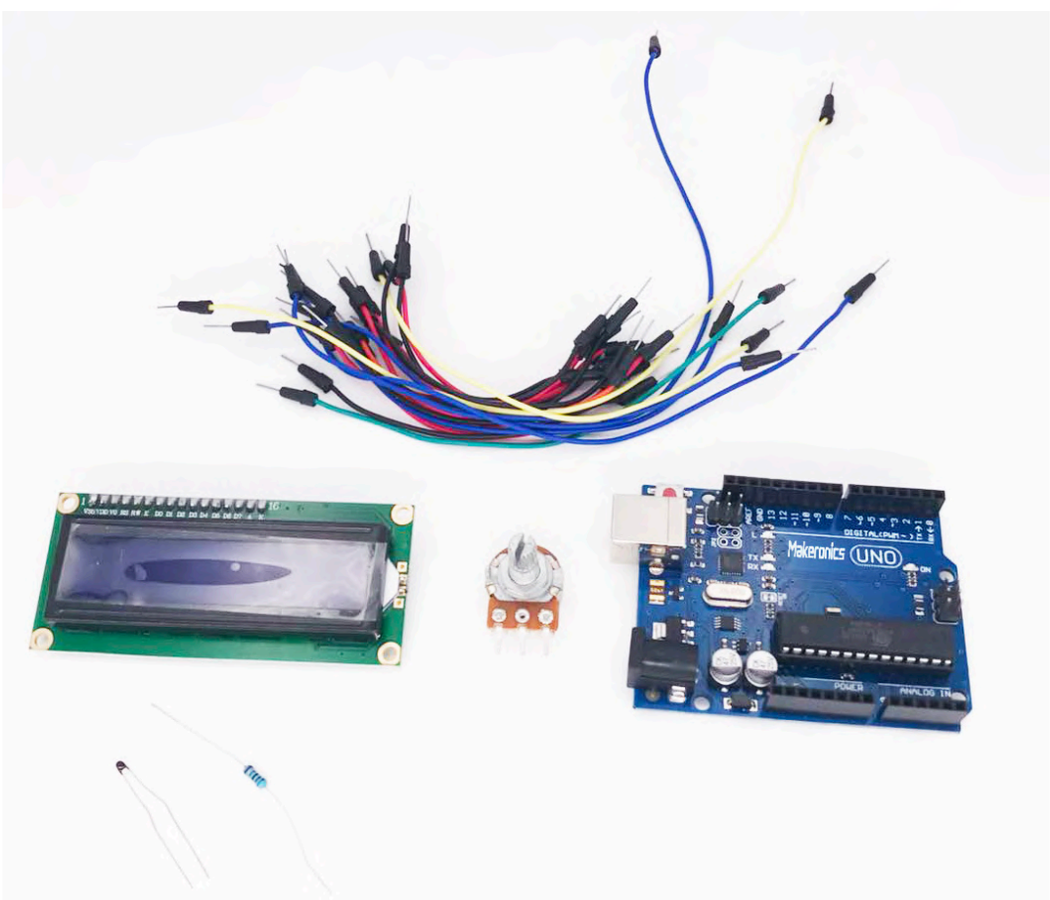
19

Thermometer

In this lesson, you will use an LCD display to show the temperature.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points Breadboard
- 1 x LCD1602 Module
- 1 x 10k ohm resistor
- 1 x Thermistor
- 1 x Potentiometer （10K)
- 20 x M-M wires (Male to Male jumper wir



Component Introduction:

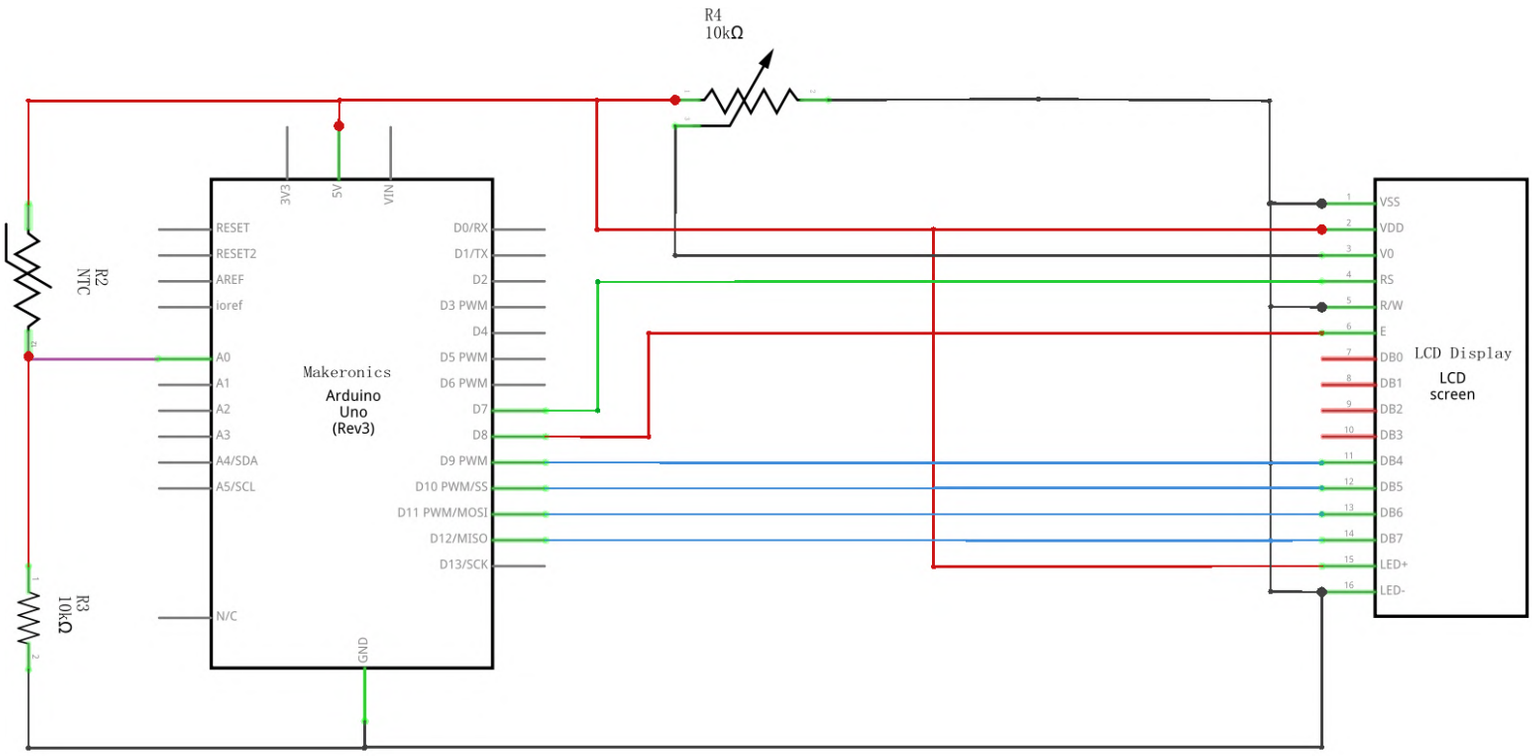
Thermistor:

A thermistor is a type of resistor whose resistance is dependent on temperature, more so than in standard resistors.
Thermistors are of two opposite fundamental types:

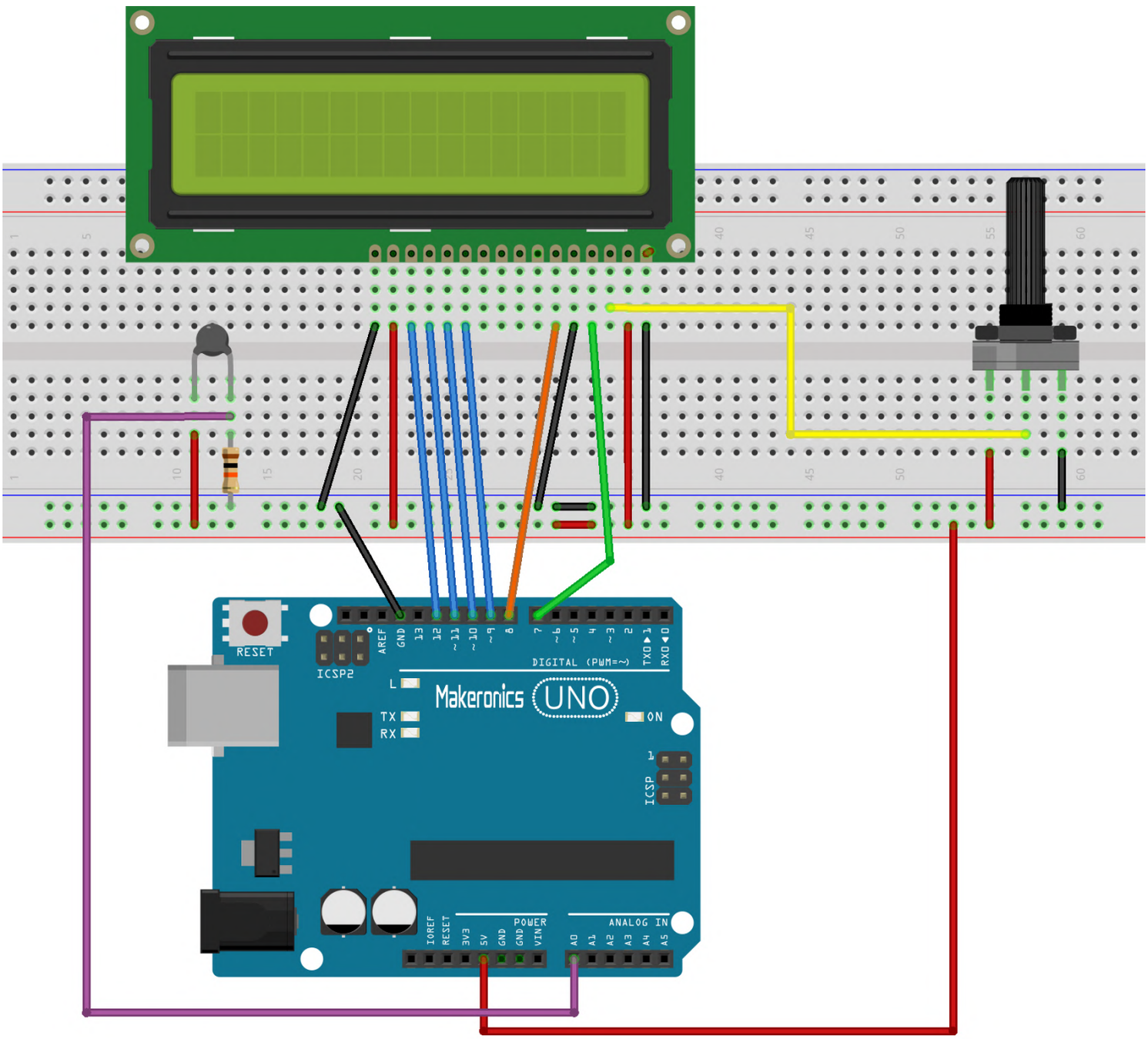
•With NTC thermistors, resistance decreases as temperature rises. An NTC is commonly used as a temperature sensor, or in series with a circuit as an inrush current limiter.

•With PTC thermistors, resistance increases as temperature rises. PTC thermistors are commonly installed in series with a circuit, and used to protect against overcurrent conditions, as resettable fuses.

Connection Schematic:



Wiring diagram:



The breadboard layout is based on the layout from Lesson 18, so it will simplify things if you still have this on the breadboard.
There are a few jumper wires near the pot that have been moved slightly on this layout.

Code:
After wiring, please open the program in the code folder- Lesson 19 Thermometer and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < LiquidCrystal > library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

The sketch for this is based on that of lesson 18. Load it to your Arduino and you should find that warming the temperature sensor by putting your finger on it will increase the temperature reading.
I find it useful to put a comment line above the 'lcd' command.

```
// BS E D4 D5 D6 D7
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

This makes things easier if you decide to change which pins you use next time.
In the 'loop' function there are now two interesting things going on. Firstly we have to convert the analog from the temperature sensor into an actual temperature, and secondly we have to work out how to display them.

First of all, let's look at how to calculate the temperature.

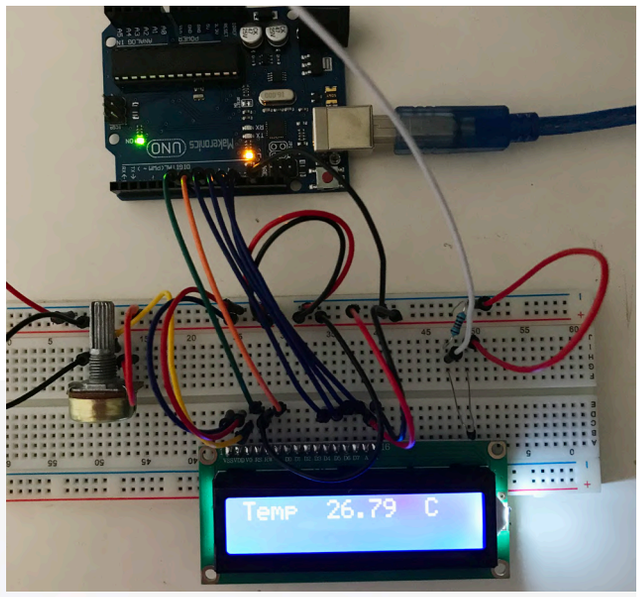
```
int tempReading = analogRead(tempPin);
double tempK = log(10000.0 * ((1024.0 / tempReading - 1)));
tempK = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 *
tempK * tempK )) * tempK );
float tempC = tempK - 273.15;
float tempF = (tempC * 9.0)/ 5.0 + 32.0;
```

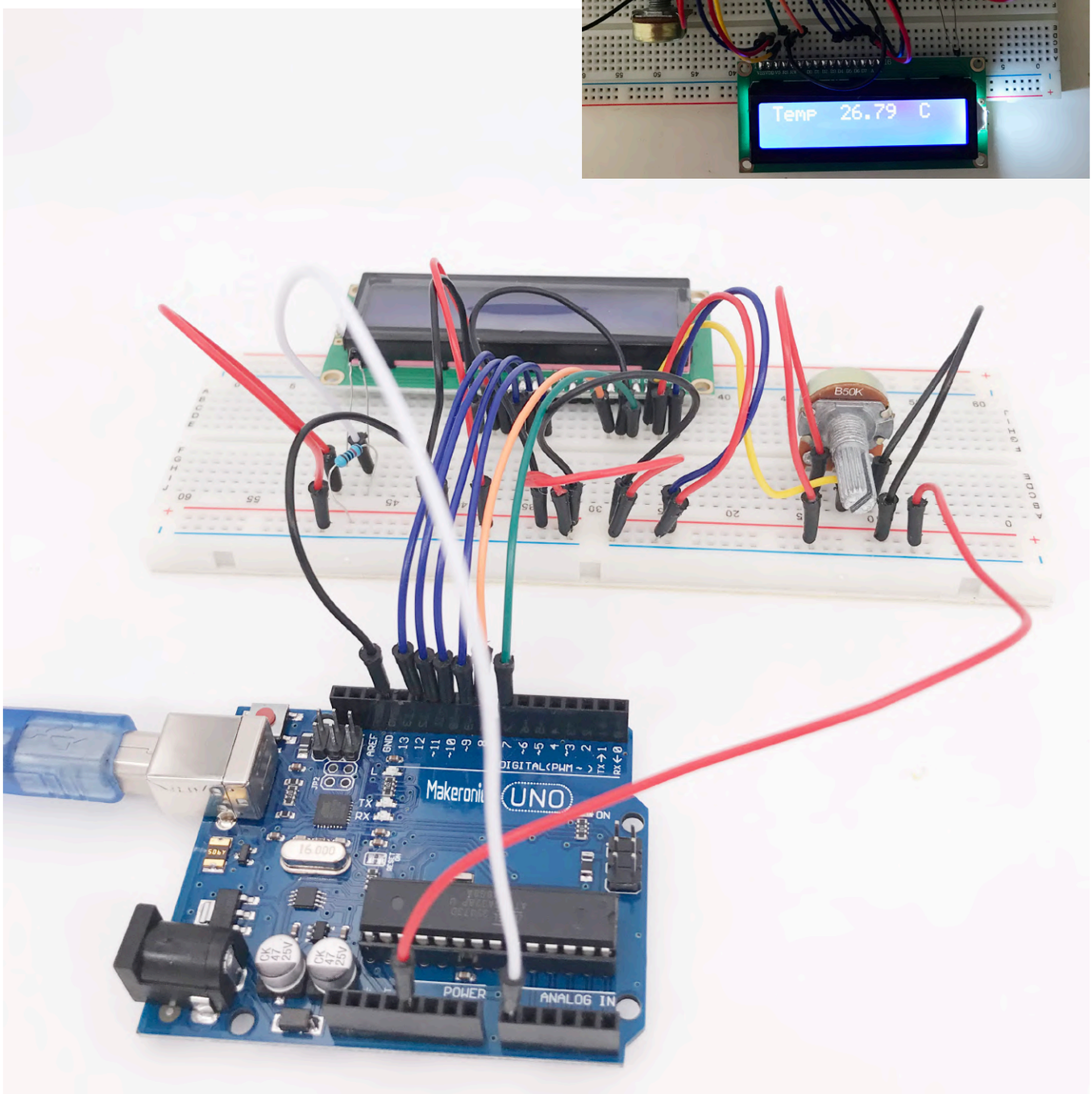
Displaying changing readings on an LCD display can be tricky. The main problem is that the reading may not always be the same number of digits. So, if the temperature changed from 101.50 to 99.00 then the extra digit from the old reading is in danger of being left on the display.
To avoid this, write the whole line of the LCD each time around the loop.

```
lcd.setCursor(0, 0);
lcd.print("Temp C ");
lcd.setCursor(6, 0);
lcd.print(tempF);
```

The rather strange comment serves to remind you of the 16 columns of the display. You can then print a string of that length with spaces where the actual reading will go.
To fill in the blanks, set the cursor position for where the reading should appear and then print it.

Demo:





Project 17

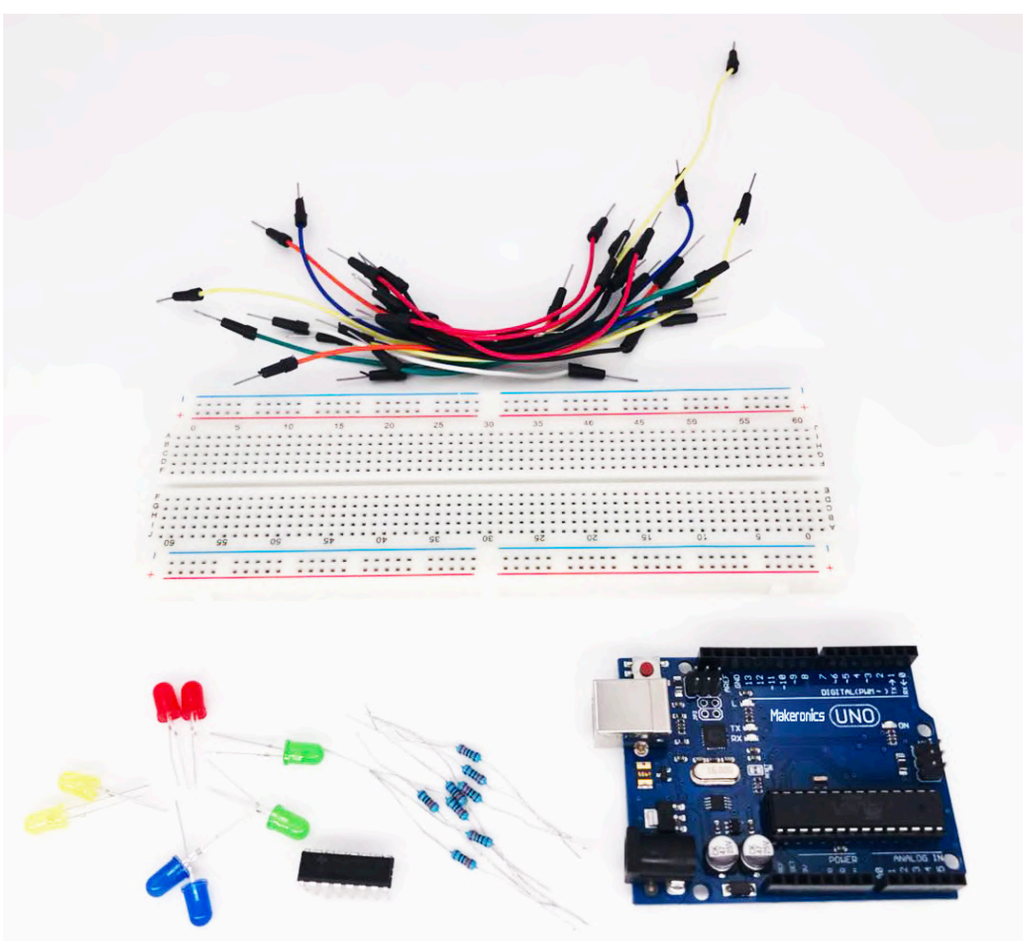
Eight LEDs with 74HC595

20

In this lesson, you will learn how to use eight large red LEDs with an UNO without needing to give up 8 output pins(refer to lesson 5)!

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 8 x leds(redx2,greenx2,yellowx2,bluex2)
- 8 x 220 ohm resistors
- 1 x 74hc595 IC
- 19 x M-M wires (Male to Male jumper wires)



Component Introduction:

Arduino has a limited number of digital pins. Sometimes, we want to build projects that require more pins than we have available on our boards. This is actually a common problem in electronics, which led to the invention of the shift register.
A shift register transforms serial data to parallel output. Basically, we tell the register what value to set for each output pin it has. If, for example, it has eight output pins, we will first say the value of the 8th pin and then the 7th pin until we get to the first one. The advantage is that we are using around three Arduino pins to get eight or even sixteen, which is very convenient.

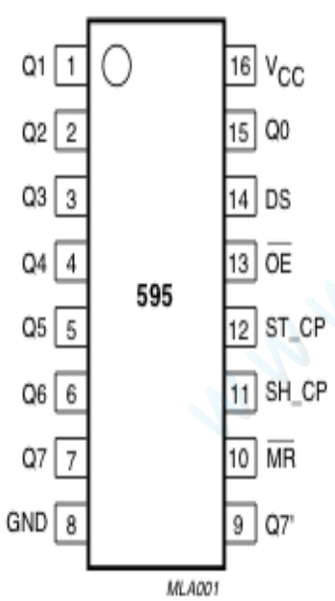
There are a lot of shift registers available and they mostly work the same. For simplicity, we will only address the commonly available 74HC595 to control eight LEDs with just three pins.

74HC595 Shift Register

The shift register is a type of chip that holds what can be thought of as eight memory locations, each of which can either be a 1 or a 0.

The shift register has an input pin to which we send the values of the outputs. For example, on a shift register with four output pins—Q0, Q1, Q2, and Q3—if we send on the data pin in succession: 1,0,1,0, it will make Q0 high, Q1 low, Q2 high, and Q3 low.

The shift register has an input pin to which we send the values of the outputs. For example, on a shift register with four output pins—Q0, Q1, Q2, and Q3—if we send on the data pin in succession: 1,0,1,0, it will make Q0 high, Q1 low, Q2 high, and Q3 low. However, we want to be able to write without affecting the current output of the shift register, and after we finish writing every output, we want to make the register apply the new values to its output pins. For this we use the latch pin. When we get the pin at LOW, in our case, it will not change the current output values until we put the pin at HIGH again. Also, it needs a way of knowing when we send a 1 or a 0; it needs a clock to make sure it reads each bit at the right interval. For that, we use the clock pin.

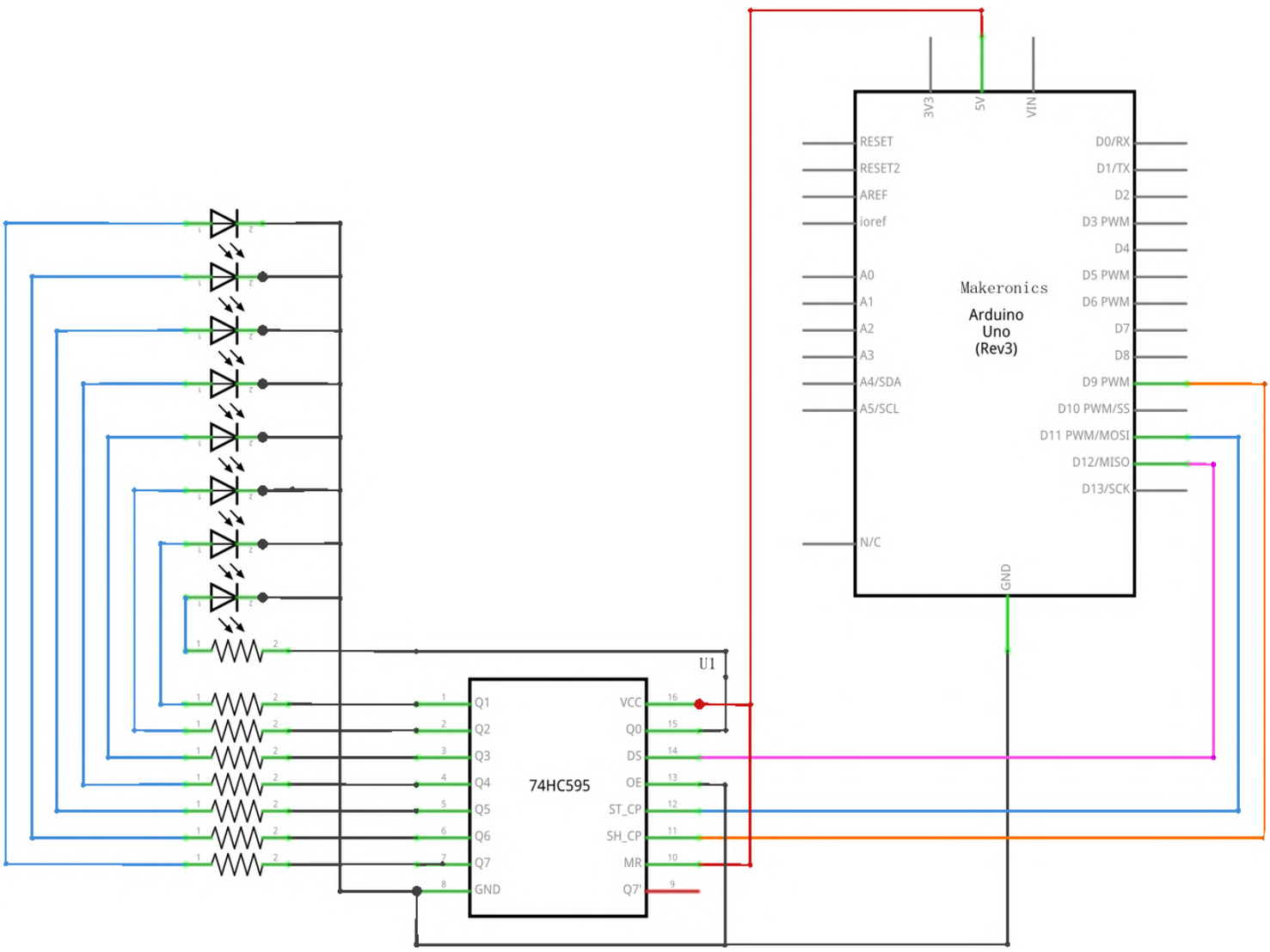


PINS 1-7, 15	Q0 " Q7	Output Pins
PIN 8	GND	Ground, Vss
PIN 9	Q7"	Serial Out
PIN 10	MR	Master Reclear, active low
PIN 11	SH_CP	Shift register clock pin
PIN 12	ST_CP	Storage register clock pin (latch pin)
PIN 13	OE	Output enable, active low
PIN 14	DS	Serial data input
PIN 16	Vcc	Positive supply voltage

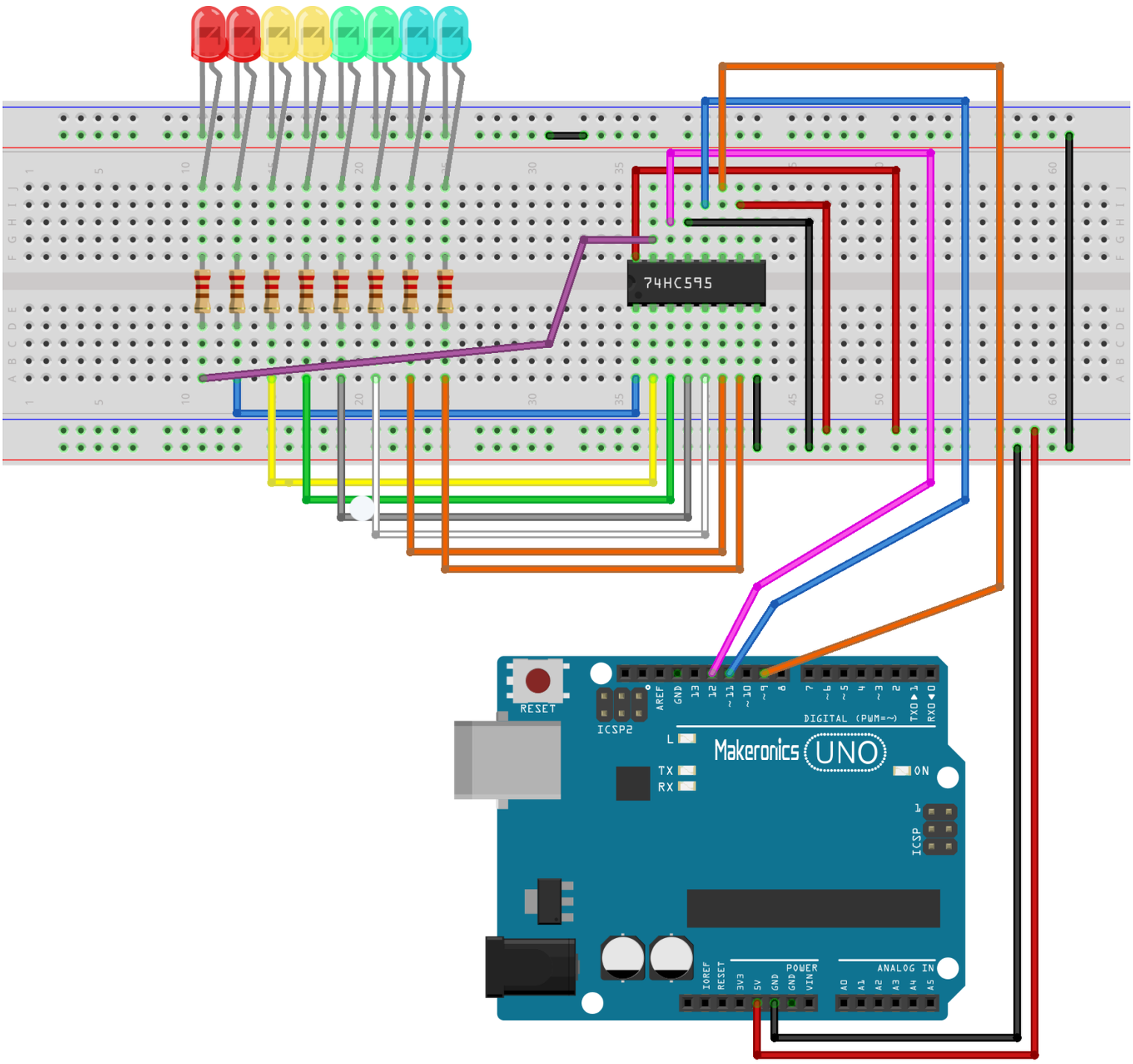
The clock pin needs to receive eight pulses. At each pulse, if the data pin is high, then a 1 gets pushed into the shift register; otherwise, a 0. When all eight pulses have been received, enabling the 'Latch' pin copies those eight values to the latch register. This is necessary; otherwise, the wrong LEDs would flicker as the data is being loaded into the shift register.

The chip also has an output enable (OE) pin, which is used to enable or disable the outputs all at once. You could attach this to a PWM-capable UNO pin and use 'analogWrite' to control the brightness of the LEDs. This pin is active low, so we tie it to GND.

Connection Schematic:



Wiring diagram:



Make the following connections to turn on 74HC595 chip:

GND (pin 8) to ground,
Vcc (pin 16) to 5V
OE (pin 13) to ground
MR (pin 10) to 5V

This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by controlling the MR and OE pins from your Arduino board too, but this way will work and leave you with more open pins.

add 8 LEDs

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called sourcing current. Some shift registers can't source current, they can only do what is called sinking current. If you have one of those it means you will have to flip the direction of the LEDs, putting the anodes directly to power and the cathodes (ground pins) to the shift register outputs. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 220-ohm resistor in series to protect the LEDs from being overloaded.

Code:

After wiring, please open the program in the code folder- Lesson 20 Eight LEDs with 74HC595 and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

The first thing we do is define the three pins we are going to use. These are the UNO digital outputs that will be connected to the latch, clock and data pins of the 74HC595.

```
int latchPin = 11;
int clockPin = 9;
int dataPin = 12;
```

Next, a variable named 'leds' is defined. This will be used to hold the pattern of which LEDs are currently turned on or off. Data of type 'byte' represents numbers using eight bits. Each bit can be either on or off, so this is perfect for keeping track of which of our eight LEDs are on or off.

```
byte leds = 0;
```

The 'setup' function just sets the three pins we are using to be digital outputs.

```
void setup()
{
pinMode(latchPin, OUTPUT);
pinMode(dataPin, OUTPUT);
pinMode(clockPin, OUTPUT);
}
```

The 'loop' function initially turns all the LEDs off, by giving the variable 'leds' the value 0. It then calls 'updateShiftRegister' that will send the 'leds' pattern to the shift register so that all the LEDs turn off. We will deal with how 'updateShiftRegister' works later.

The loop function pauses for half a second and then begins to count from 0 to 7 using the 'for' loop and the variable 'i'. Each time, it uses the Arduino function 'bitSet' to set the bit that controls that LED in the variable 'leds'. It then also calls 'updateShiftRegister' so that the leds update to reflect what is in the variable 'leds'.
There is then a half second delay before 'i' is incremented and the next LED is lit.

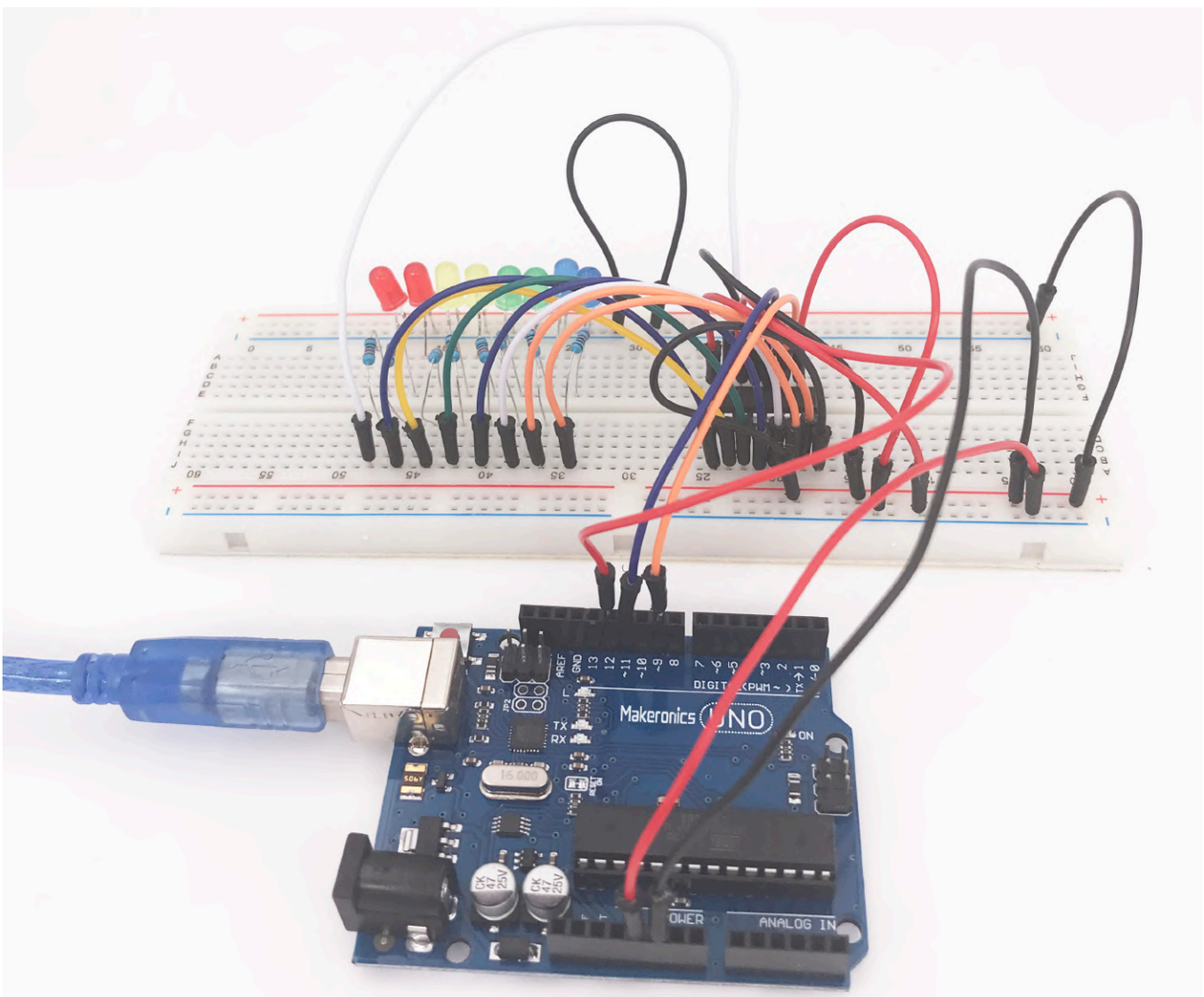
```
void loop()
{
leds = 0;
updateShiftRegister();
delay(500);
for (int i = 0; i < 8; i++)
{
bitSet(leds, i);
updateShiftRegister();
delay(500);
}
}
```

The function 'updateShiftRegister', first of all sets the latchPin to low, then calls the UNO function 'shiftOut' before putting the 'latchPin' high again. This takes four parameters, the first two are the pins to use for Data and Clock respectively.
The third parameter specifies which end of the data you want to start at. We are going to start with the right most bit, which is referred to as the 'Least Significant
Bit' (LSB).
The last parameter is the actual data to be shifted into the shift register, which in this case is 'leds'.

```
void updateShiftRegister()
{
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, LSBFIRST, leds);
digitalWrite(latchPin, HIGH);
}
```

If you wanted to turn one of the LEDs off rather than on, you would call a similar Arduino function (bitClear) with the 'leds' variable. This will set that bit of 'leds' to be 0 and you would then just need to follow it with a call to 'updateShiftRegister' to update the actual LEDs.

Demo:



21

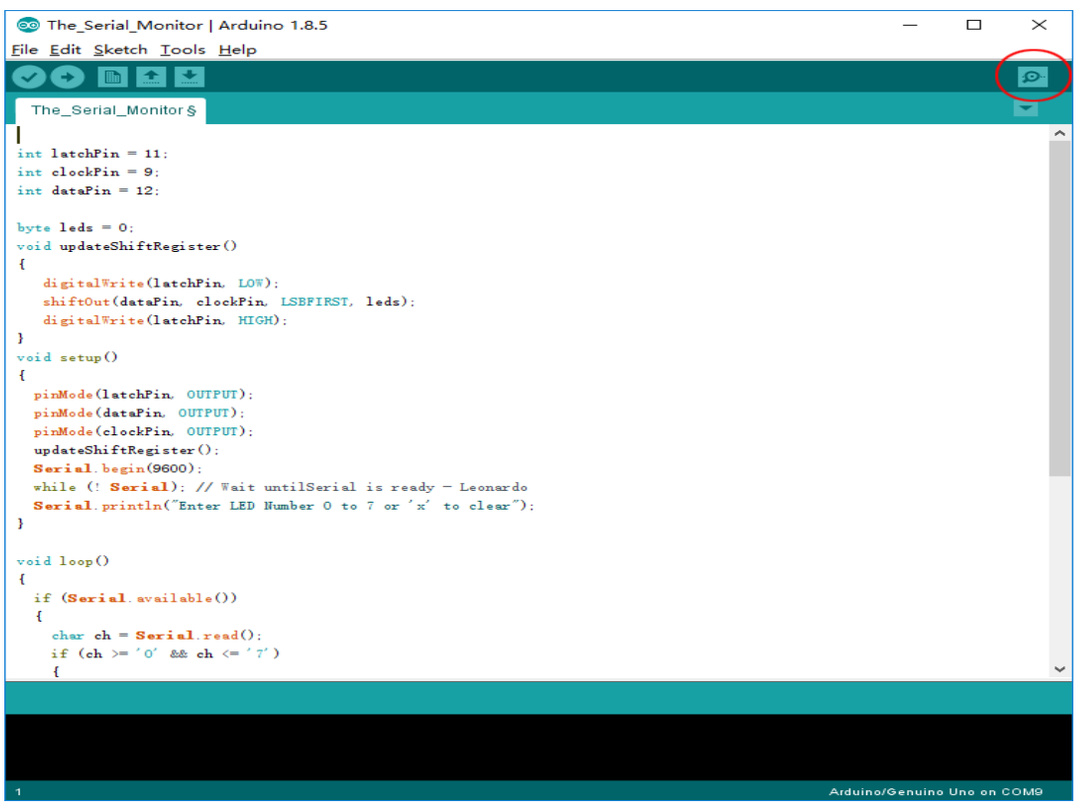
The Serial Monitor

In this lesson, you will build on Lesson 20, adding the facility to control the LEDs from your computer using the Arduino Serial Monitor. The serial monitor is the 'tether' between the computer and your UNO. It lets you send and receive text messages, handy for debugging and also controlling the UNO from a keyboard! For example, you will be able to send commands from your computer to turn on LEDs.

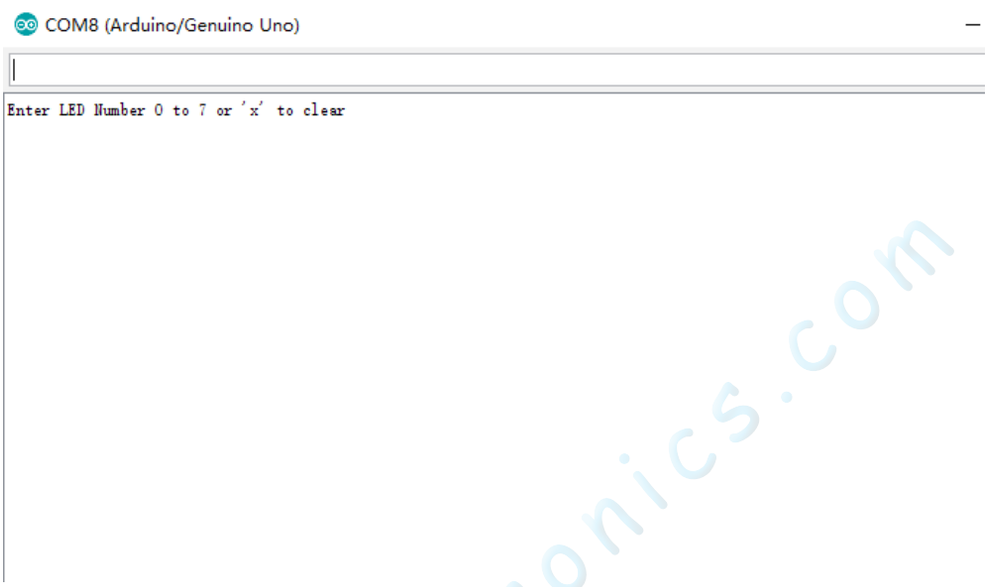
In this lesson, you will use exactly the same parts and a similar breadboard layout as Lesson 20. So, if you have not already done so, follow Lesson 20 now.

Steps taken

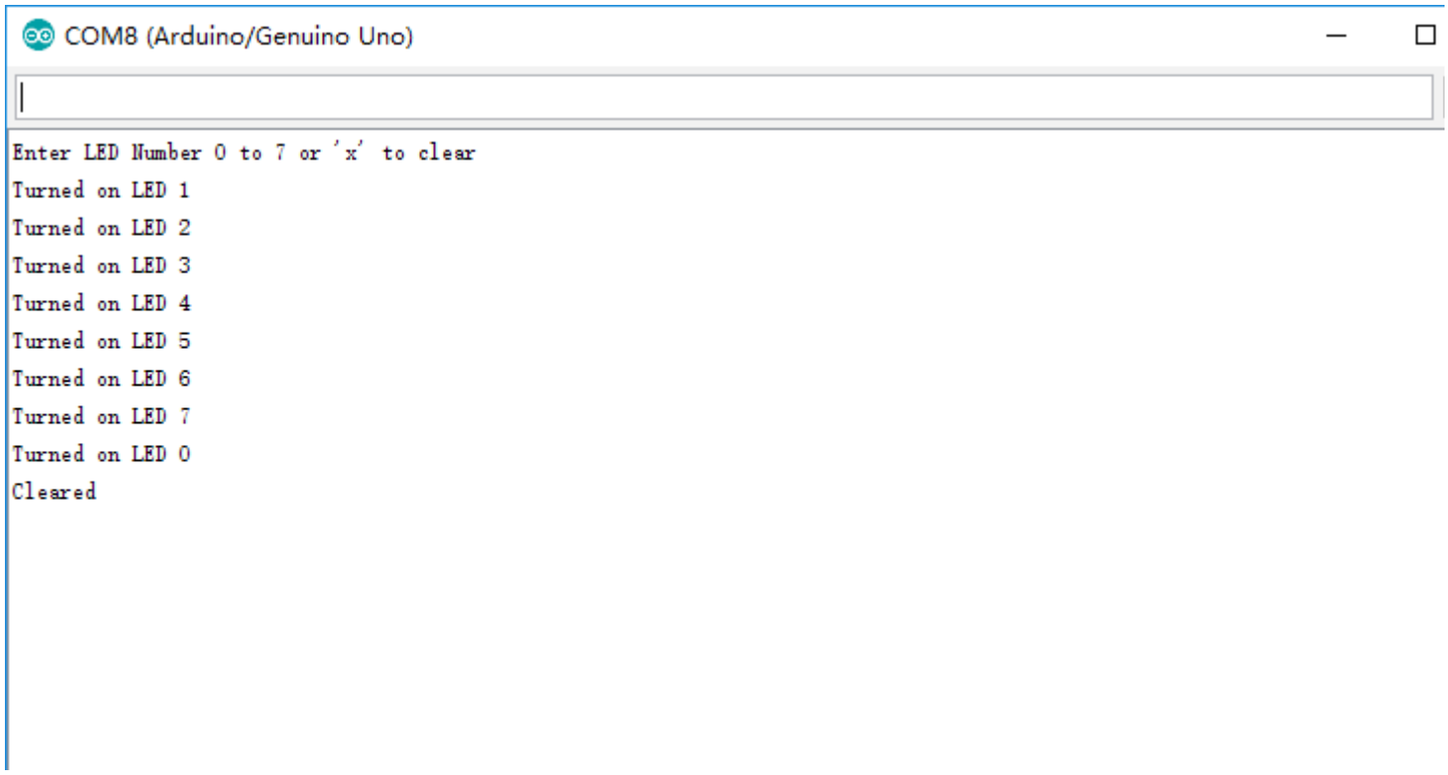
After you have uploaded this sketch onto your UNO, click on the right-most button on the toolbar in the Arduino IDE. The button is circled below.



The following window will open.
Click the Serial Monitor button to turn on the serial monitor. The basics about the serial monitor are introduced in details in Lesson 2.



This window is called the Serial Monitor and it is part of the Arduino IDE software. Its job is to allow you to both send messages from your computer to an UNO board (over USB) and also to receive messages from the UNO.
The message "Enter LED Number 0 to 7or 'x' to clear" has been sent by the Arduino. It is telling us what commands we can send to the Arduino: either send the 'x' (to turn all the LEDs off) or the number of the LED you want to turn on (where 0 is the bottom LED, 1 is the next one up, all the way to 7 for the top LED).
Try typing the following commands into the top area of the Serial Monitor that is level with the 'Send' button. Press 'Send', after typing each of these characters: x 0 3 5
Typing x will have no effect if the LEDs are already all off, but as you enter each number, the corresponding LED should light and you will get a confirmation message from the UNO board. The Serial Monitor will appear as shown below.



Type x again and press 'Send' to turn off all LEDs.

Code

After wiring, please open program in the code folder- Lesson 21 The Serial Monitor and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

As you might expect, the sketch is based on the sketch used in Lesson 20. So, we will just cover the new bits here. You will find it useful to refer to the full sketch in your Arduino IDE.

In the 'setup' function, there are three new lines at the end:

```
void setup()
{
pinMode(latchPin, OUTPUT);
pinMode(dataPin, OUTPUT);
pinMode(clockPin, OUTPUT);
updateShiftRegister();
Serial.begin(9600);
while (! Serial); // Wait until Serial is ready - Leonardo
Serial.println("Enter LED Number 0 to 7 or 'x' to clear");
}
```

Firstly, we have the command 'Serial.begin(9600)'. This starts serial communication, so that the UNO can send out commands through the USB connection. The value 9600 is called the 'baud rate' of the connection. This is how fast the data is to be sent. You can change this to a higher value, but you will also have to change the Arduino Serial monitor to the same value.

The line beginning with 'while' ensures that there is something at the other end of the USB connection for the Arduino to talk to before it starts sending messages. Otherwise, the message might be sent, but not displayed. This line is actually only necessary if you are using an Arduino Leonardo because the Arduino UNO automatically resets the Arduino board when you open the Serial Monitor, whereas this does not happen with the Leonardo.

The last of the new lines in 'setup' sends out the message that we see at the top of the Serial Monitor.
The 'loop' function is where all the action happens:

```
void loop()
{
if (Serial.available())
{
char ch = Serial.read();
if (ch >= '0' && ch <= '7')
{
int led = ch - '0';
bitSet(leds, led);
updateShiftRegister();
Serial.print("Turned on LED ");
Serial.println(led);
}
```

```
if (ch == 'x')
{
leds = 0;
updateShiftRegister();
Serial.println("Cleared");
}
}
}
```

Everything that happens inside the loop is contained within an 'if' statement. So unless the call to the built-in Arduino function 'Serial.available()' is 'true' then nothing else will happen.
Serial.available() will return 'true' if data has been send to the UNO and is there ready to be processed. Incoming messages are held in what is called a buffer and Serial. available() returns true if that buffer is Not empty.

If a message has been received, then it is on to the next line of code:

```
char ch = Serial.read();
```

This reads the next character from the buffer, and removes it from the buffer. It also assigns it to the variable 'ch'. The variable 'ch' is of type 'char' which stands for 'character' and as the name suggests, holds a single character.
If you have followed the instructions in the prompt at the top of the Serial Monitor, then this character will either be a single digit number between 0 and 7 or the letter 'x'.

The 'if' statement on the next line checks to see if it is a single digit by seeing if 'ch' is greater than or equal to the character '0' and less than or equal to the character '7'. It looks a little strange comparing characters in this way, but is perfectly acceptable.

Each character is represented by a unique number, called its ASCII value. This means that when we compare characters using <= and >= it is actually the ASCII values that were being compared.

If the test passes, then we come to the next line:

```
int led = ch – '0';
```

Now we are performing arithmetic on characters! We are subtracting the digit '0' from whatever digit was entered. So, if you typed '0' then '0' - '0' will equal 0. If you typed '7' then '7' - '0' will equal the number 7 because it is actually the ASCII values that are being used in the subtraction.

Since that we know the number of the LED that we want to turn on, we just need to set that bit in the variable 'leds' and update the shift register.

```
bitSet(leds, led);
updateShiftRegister();
```

The next two lines write back a confirmation message to the Serial Monitor.

```
Serial.print("Turned on LED ");
Serial.println(led);
{
leds = 0;
updateShiftRegister();
Serial.println("Cleared");
}
```

If it is, then it clears all the LEDs and sends a confirmation message.

The first line uses Serial.print rather than Serial.println. The different between the two is that Serial.print does not start a new line after printing whatever is in its parameter. We use this in the first line, because we are printing the message in two parts. Firstly the general bit: 'Turned on LED ' and then the number of the LED.

The number of the LED is held in an 'int' variable rather than being a text string. Serial.print can take either a text string enclosed in double-quotes, or an 'int' or for that matter pretty much any type of variable.

After the 'if' statement that handles the case, when a single digit has been handled, there is a second 'if' statement that checks to see if 'ch' is the letter 'x'.

```
if (ch == 'x')
 {
 leds = 0;
 updateShiftRegister();
 Serial.println("Cleared");
 }
```

If it is, then it clears all the LEDs and sends a confirmation message.

Project 18

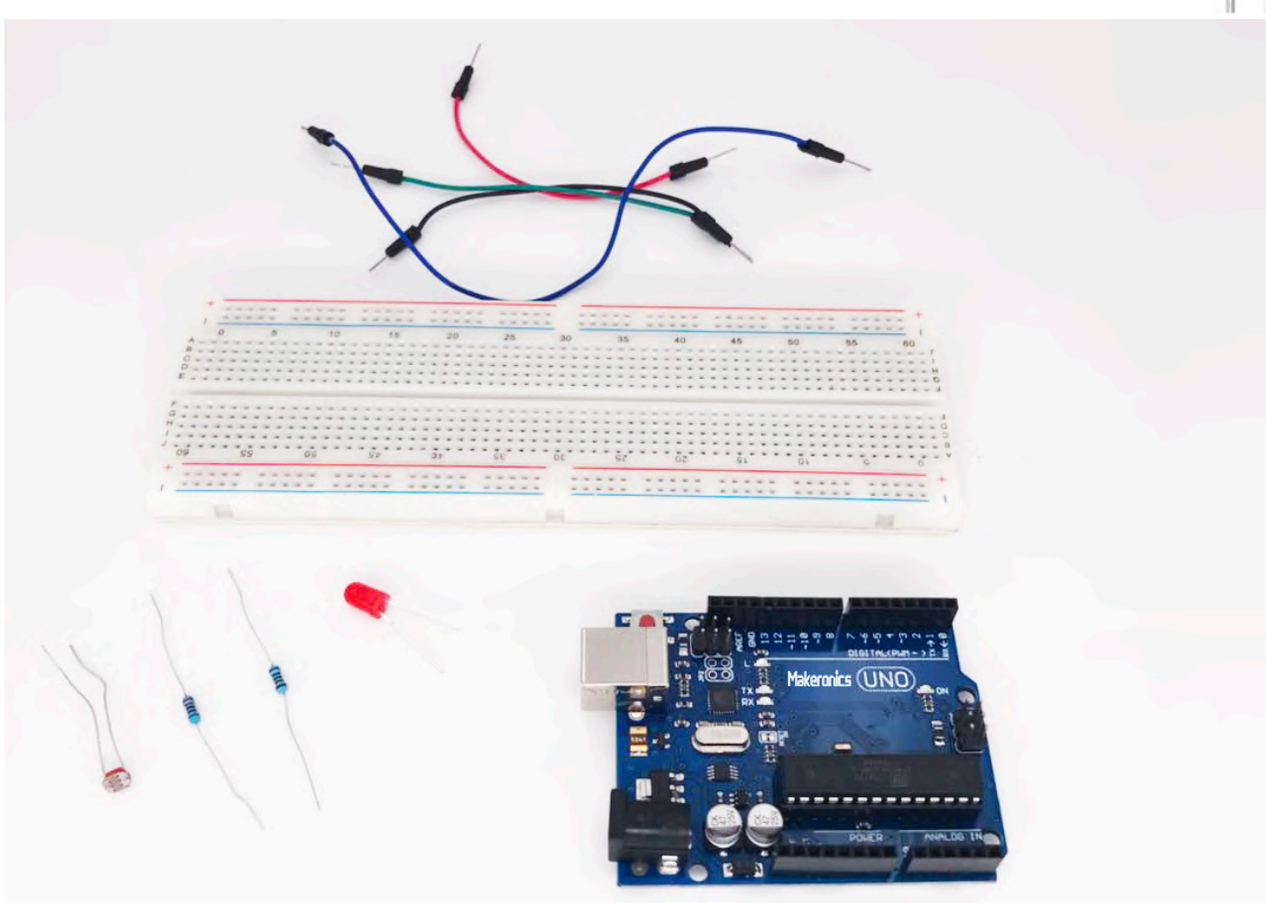
22

Photoresistor

In this lesson, you will learn how to create a night light that gets brighter depending on the amount of light detected.

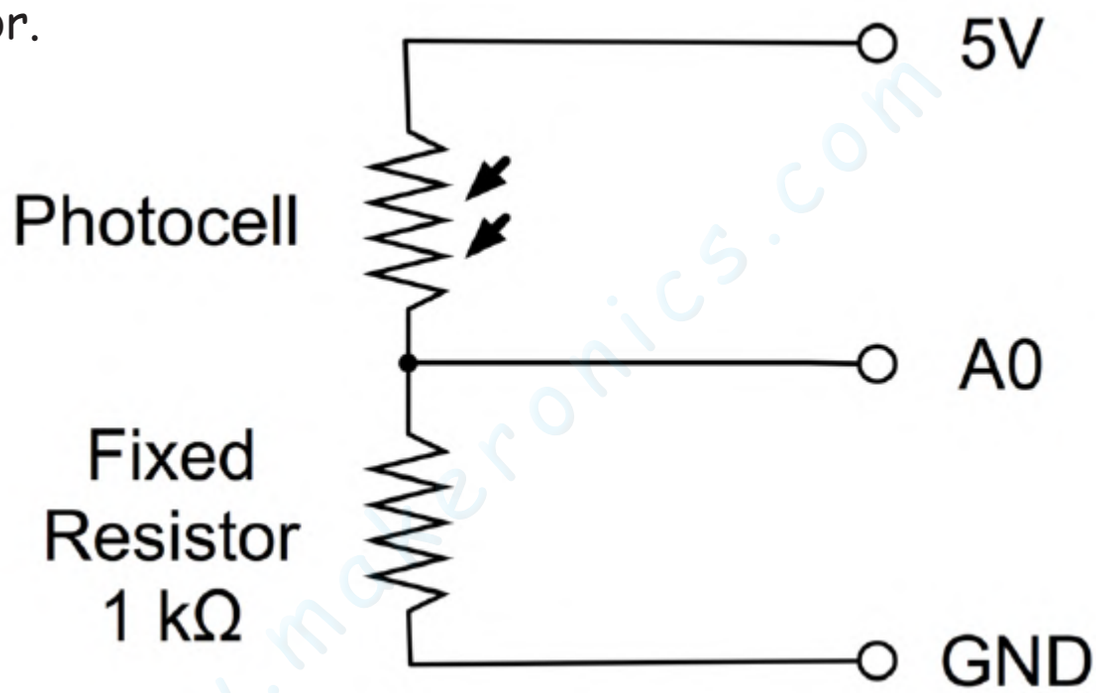
Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 1 x Red led
- 1 x 220 ohm resistors
- 1 x 1k ohm resistor
- 1 x Photoresistor (Photocell)
- 4 x M-M wires (Male to Male jumper wires)



Component Introduction: Photoresistor

A photoresistor is a variable resistor that reacts to light; the less light that shines on it, the higher the resistance it provides..
This one has a resistance of about 50 kΩ in near darkness and 500 Ω in bright light. To convert this varying value of resistance into something we can measure on an UNO R3 board's analog input, it needs to be converted into a voltage. The simplest way to do that is to combine it with a fixed resistor.

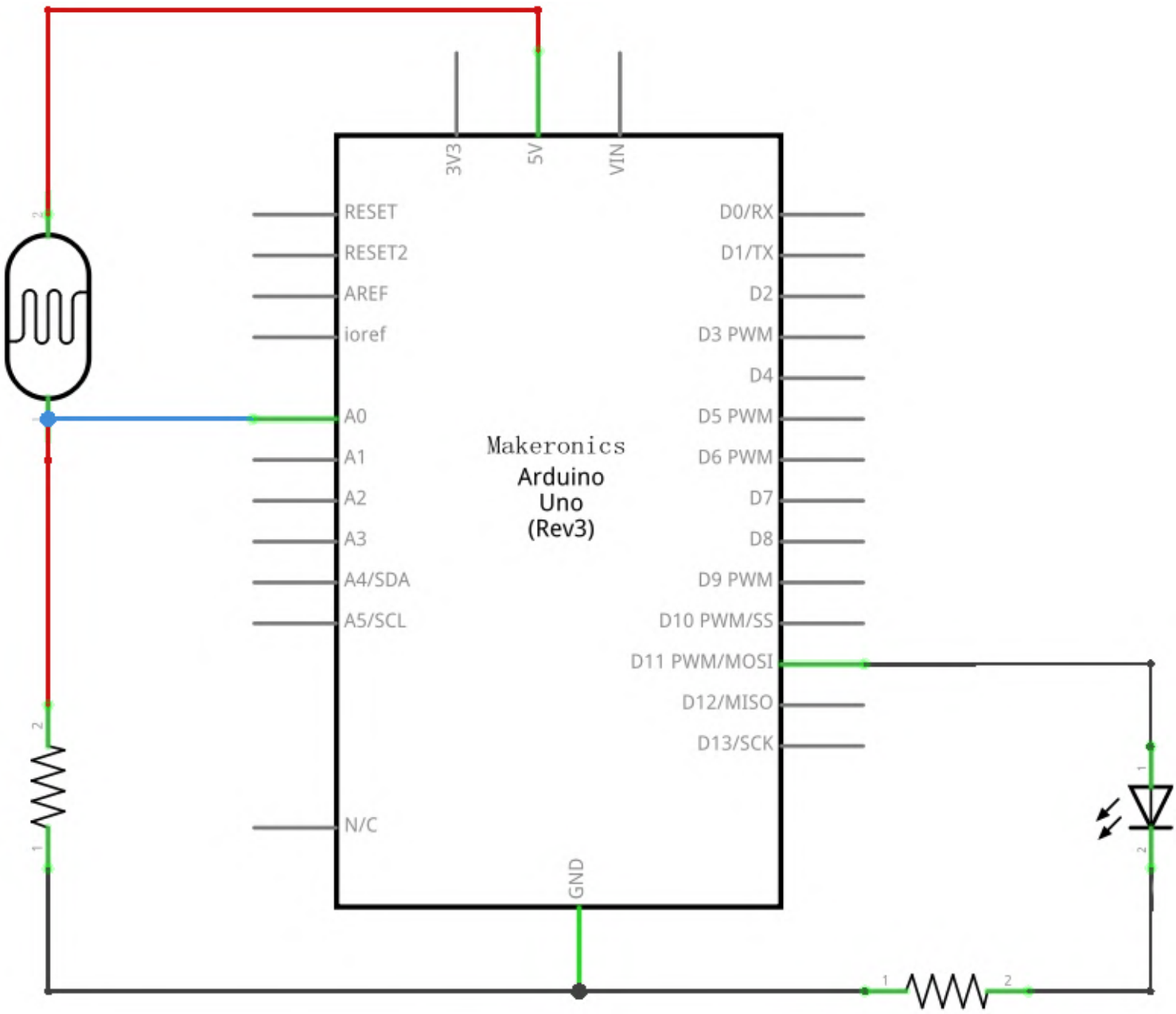


The resistor and photocell together behave like a pot. When the light is very bright, then the resistance of the photocell is very low compared with the fixed value resistor, and so it is as if the pot were turned to maximum.

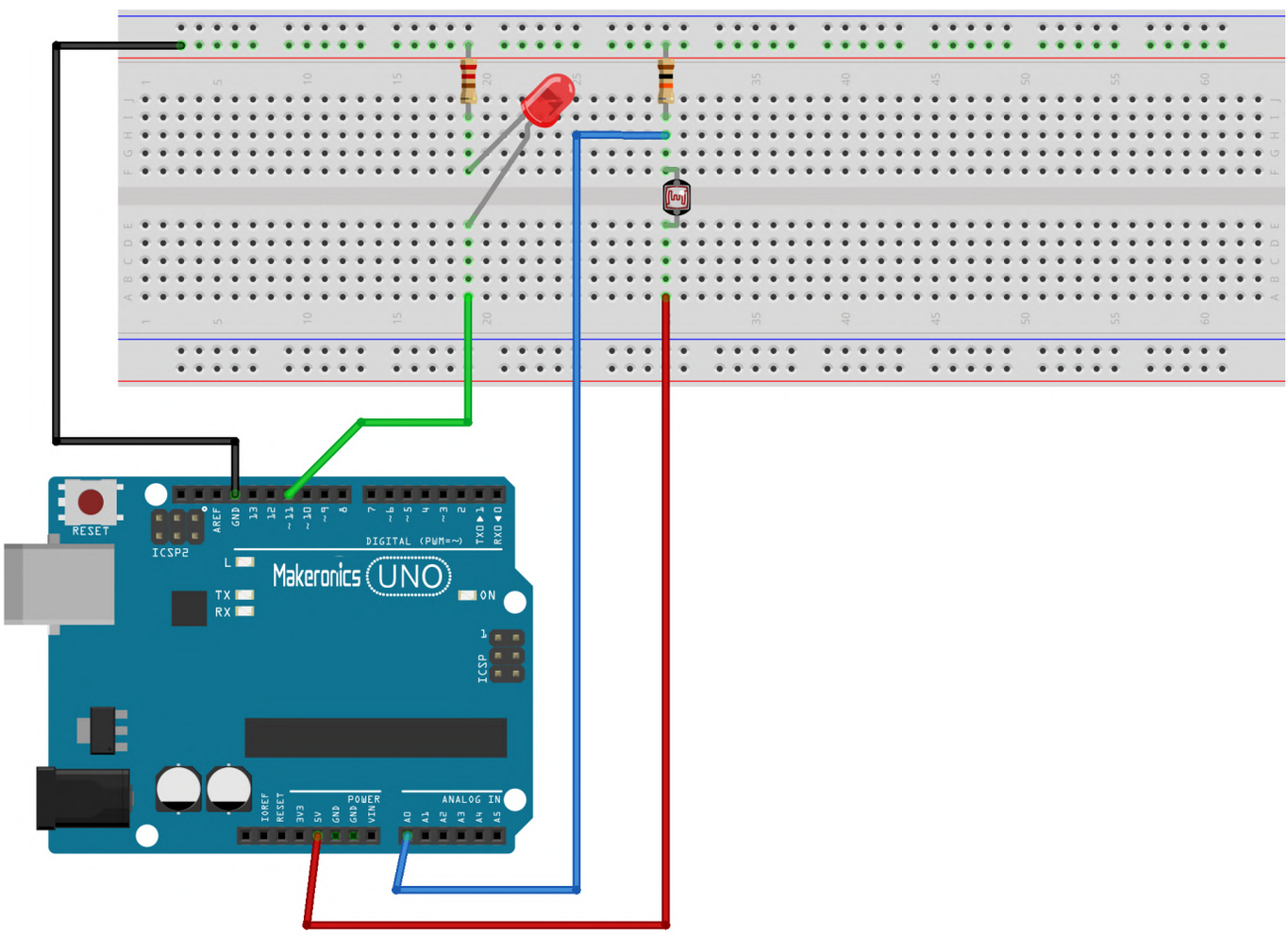
When the photocell is in dull light, the resistance becomes greater than the fixed 1 kΩ resistor and it is as if the pot were being turned towards GND.

Load up the sketch given in the next section and try covering the photocell with your finger, and then holding it near a light source.

Connection Schematic:



Wiring diagram:



Code:

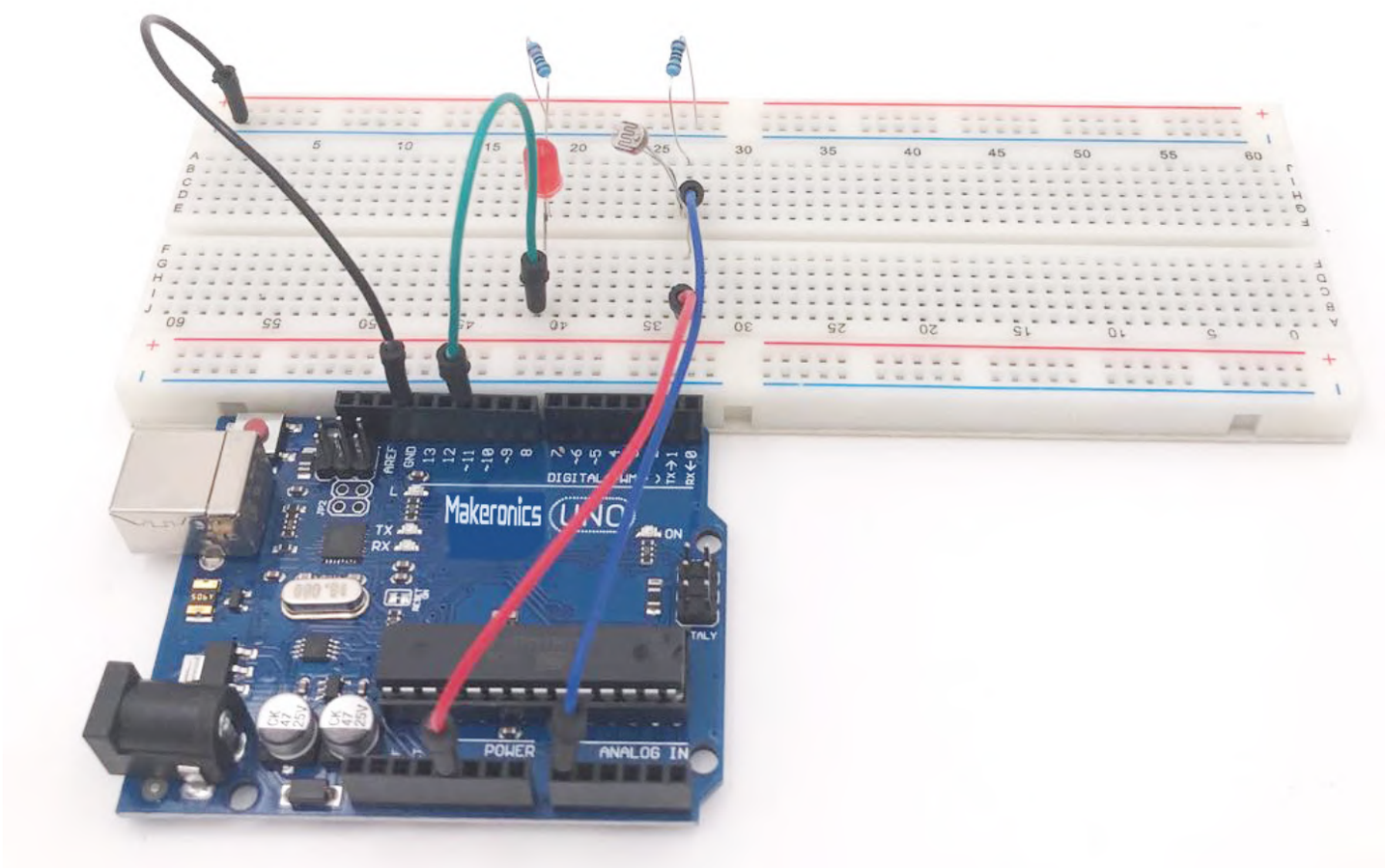
This resistance value varies the voltage that's sent to the input pin of the Arduino, which in turn sends that voltage value to the output pin as the power level of the LED, so in low light the LED will be bright.

```
int ledPin = 13; // Pin connected to the LED
analogWrite(ledPin, analogRead(lightPin) / 4);
```

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). The frequency of the PWM signal is approximately 490 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 through 13. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11. You do not need to call pinMode() to set the pin as an output before calling analogWrite().

Demo:



Project 19

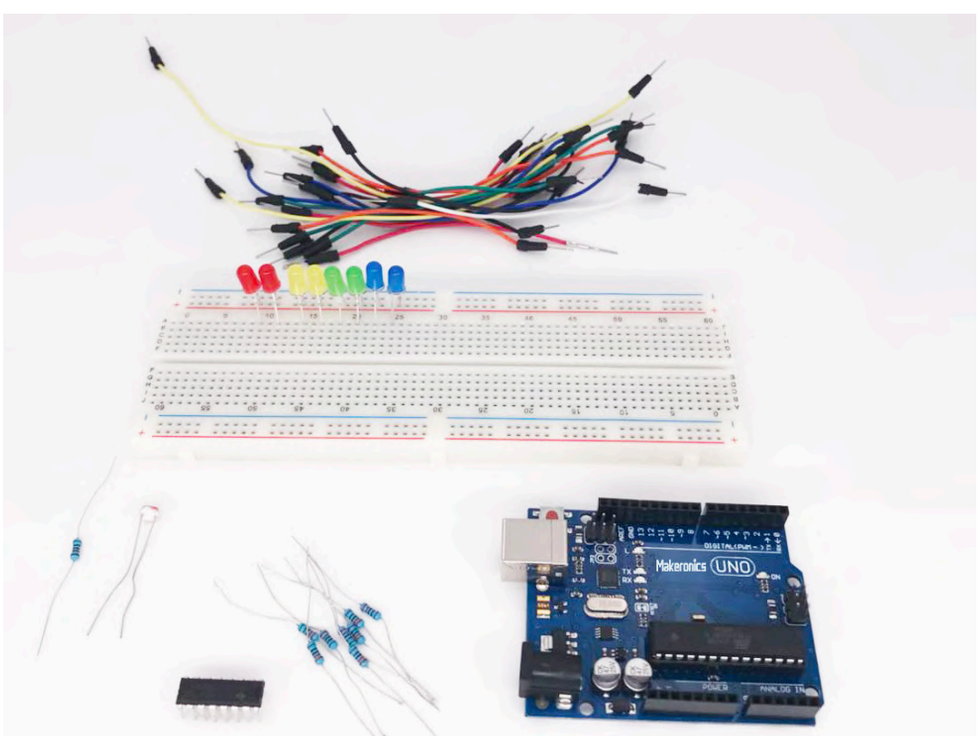
Photoresistor And 74HC595

23

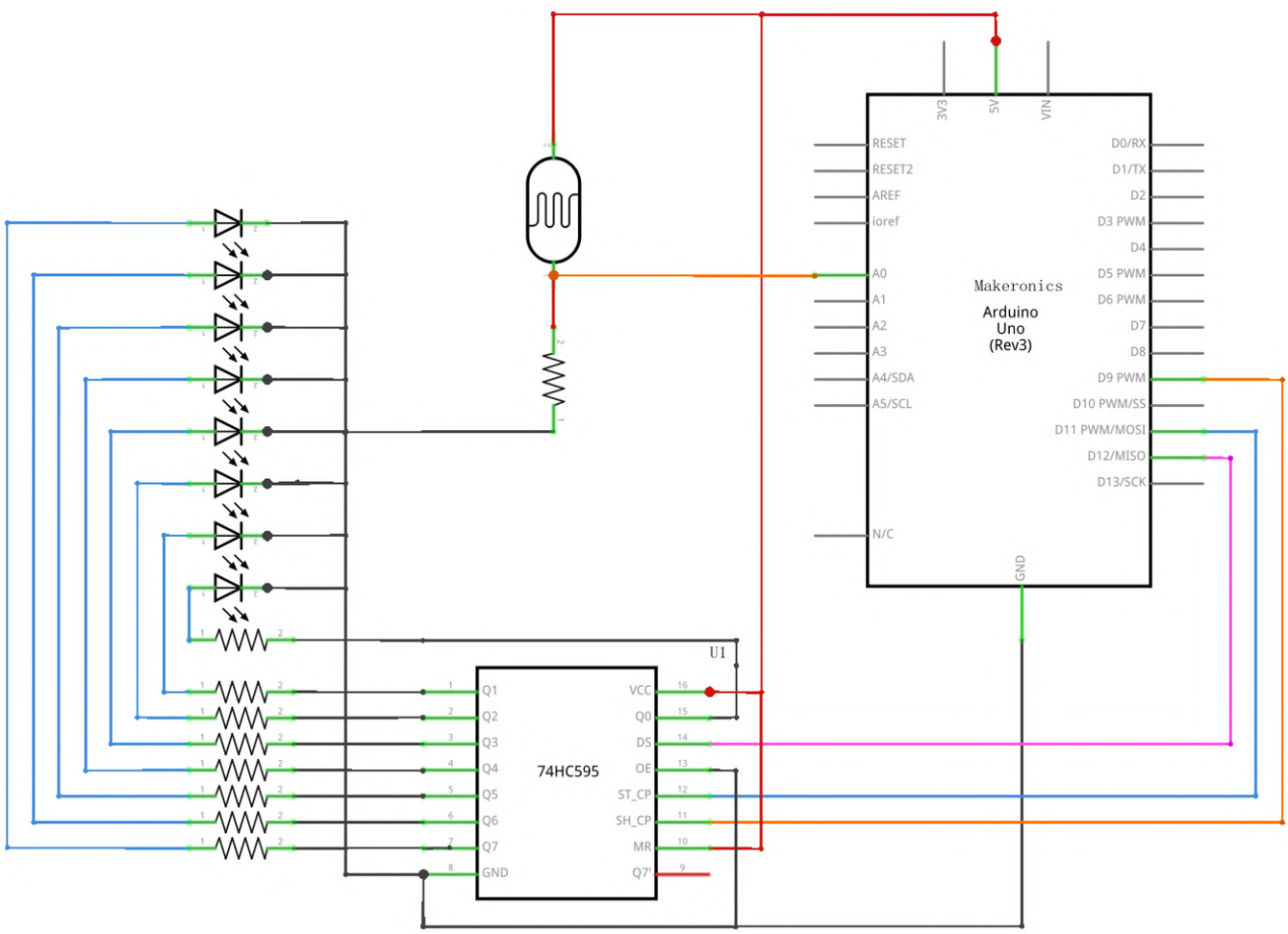
In this lesson, you will learn how to measure light intensity using an Analog Input. You will build on lesson 2 and use the level of light that measured by photocell to control the number of LEDs to be lit.

Component Required:

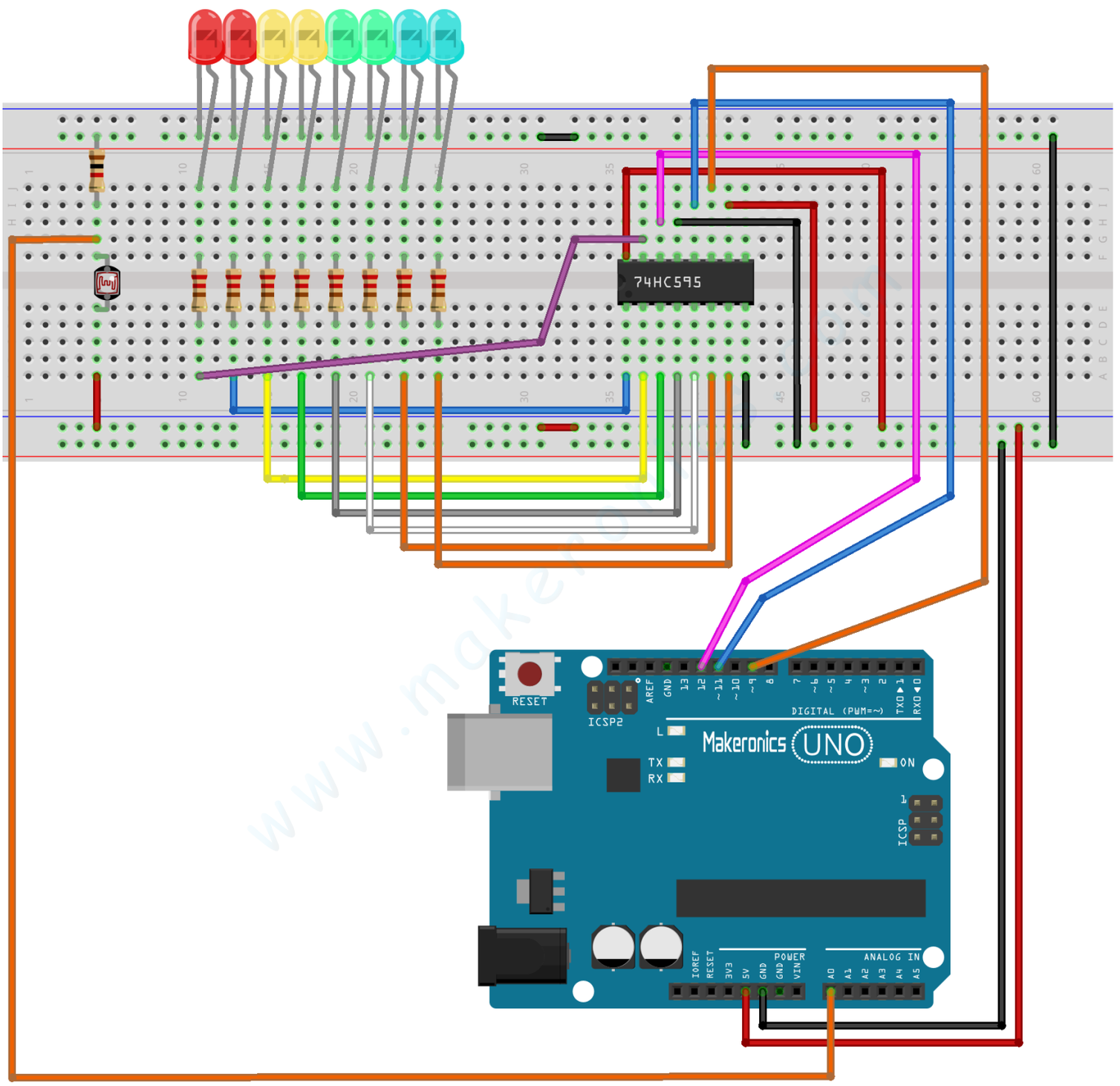
- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 8 x leds(redx2,greenx2,yellowx2,bluex2)
- 8 x 220 ohm resistors
- 1 x 1k ohm resistor
- 1 x 74hc595 IC
- 1 x Photoresistor (Photocell)
- 22 x M-M wires (Male to Male jumper wires)



Connection Schematic:



Wiring diagram:



Code:

After wiring, please open the program in the code folder- Lesson 23 Photocell and 74HC595 and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.

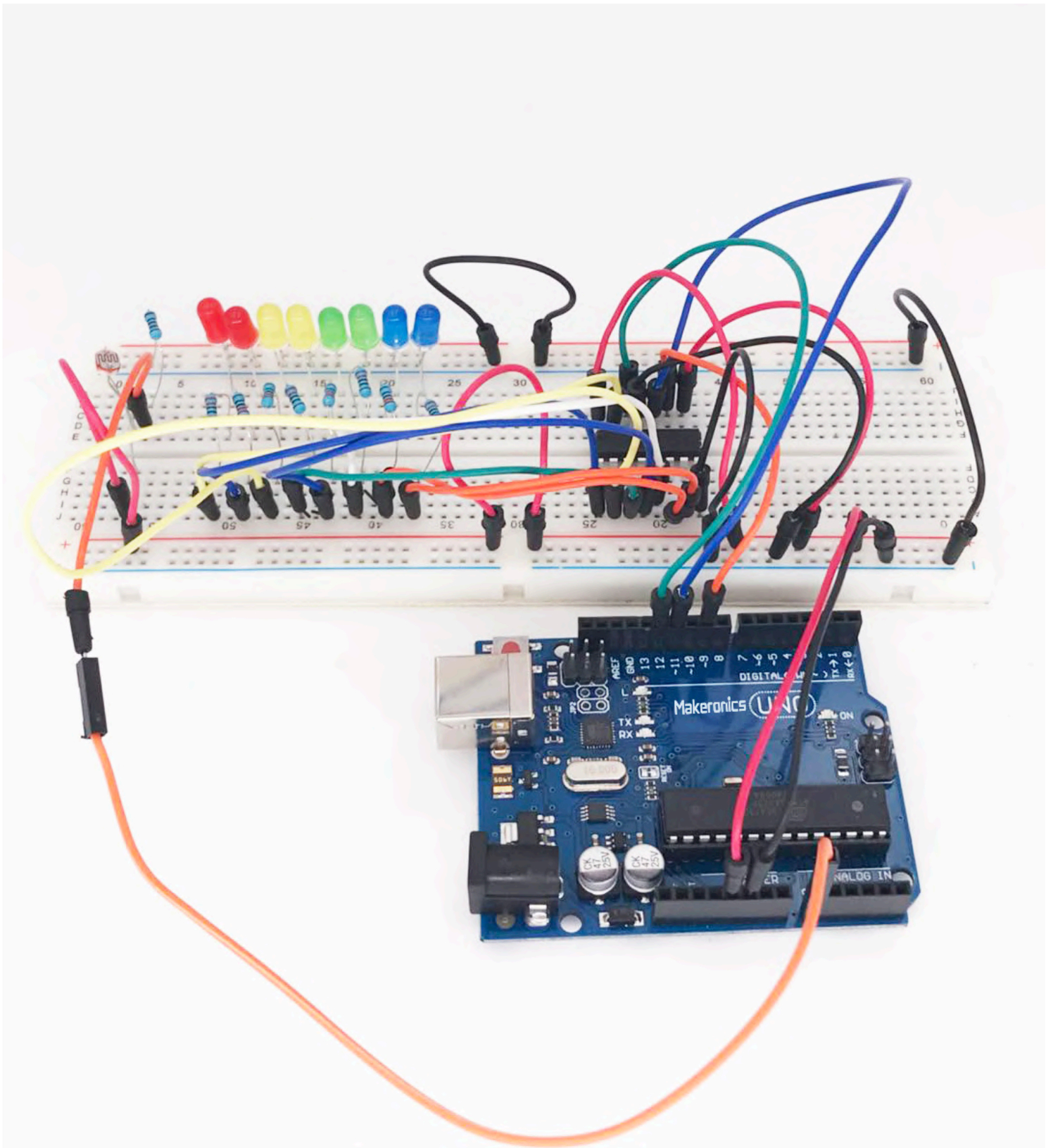
The first thing to note is that we have added a variable named 'lightPin' to match the analog pin.

The only other substantial change to the sketch is the line that calculates how many of the LEDs to light:

```
int numLEDSLit = reading / 57; // all LEDs lit at 1k
```

This time, we divide the raw reading by 57 rather than 114. In other words, we divide it by half as much as we did with the pot to split it into nine zones, from no LEDs lit to all eight lit. This extra factor is to account for the fixed 1 kΩ resistor. This means that when the photocell has a resistance of 1 kΩ (the same as the fixed resistor), the raw reading will be 1023 / 2 = 511. This will equate to all the LEDs being lit and then a bit (numLEDSLit) will be 8.

Demo:



Project 20

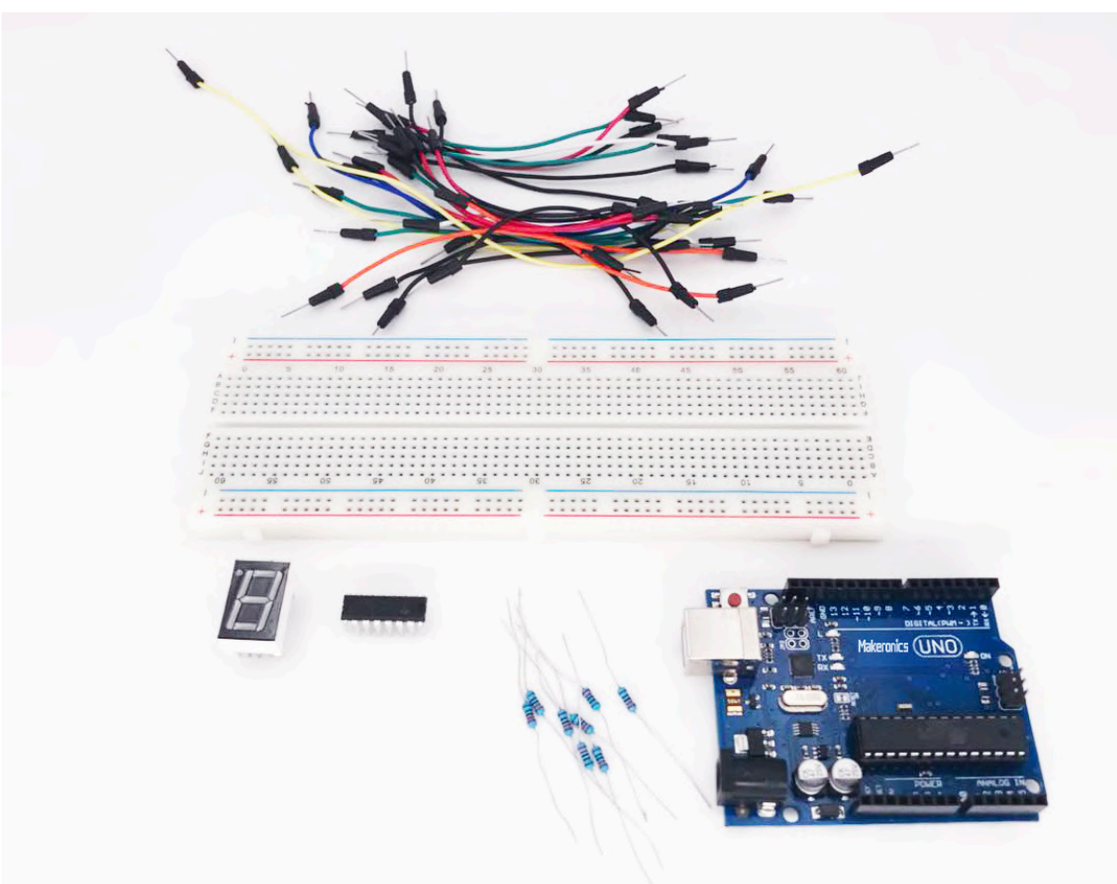
74HC595 And Segment Display

24

After learning Lesson 20、21 and Lesson 23, we will use the 74HC595 shift register to control the segment display. The segment display will show number from 9-0.

Component Required:

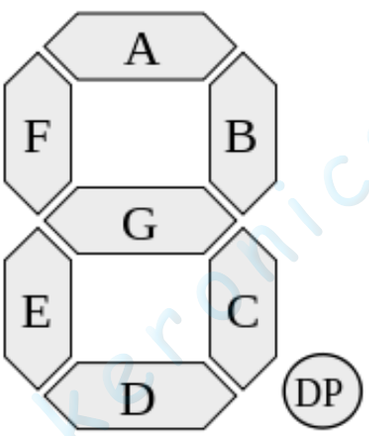
- (1) x Makeronics Uno R3
- (1) x 830 tie-points breadboard
- (1) x 74HC595 IC
- (1) x 1 Digit 7-Segment Display
- (8) x 220 ohm resistors
- (26) x M-M wires (Male to Male jumper wires)



Component Introduction:

Seven segment display

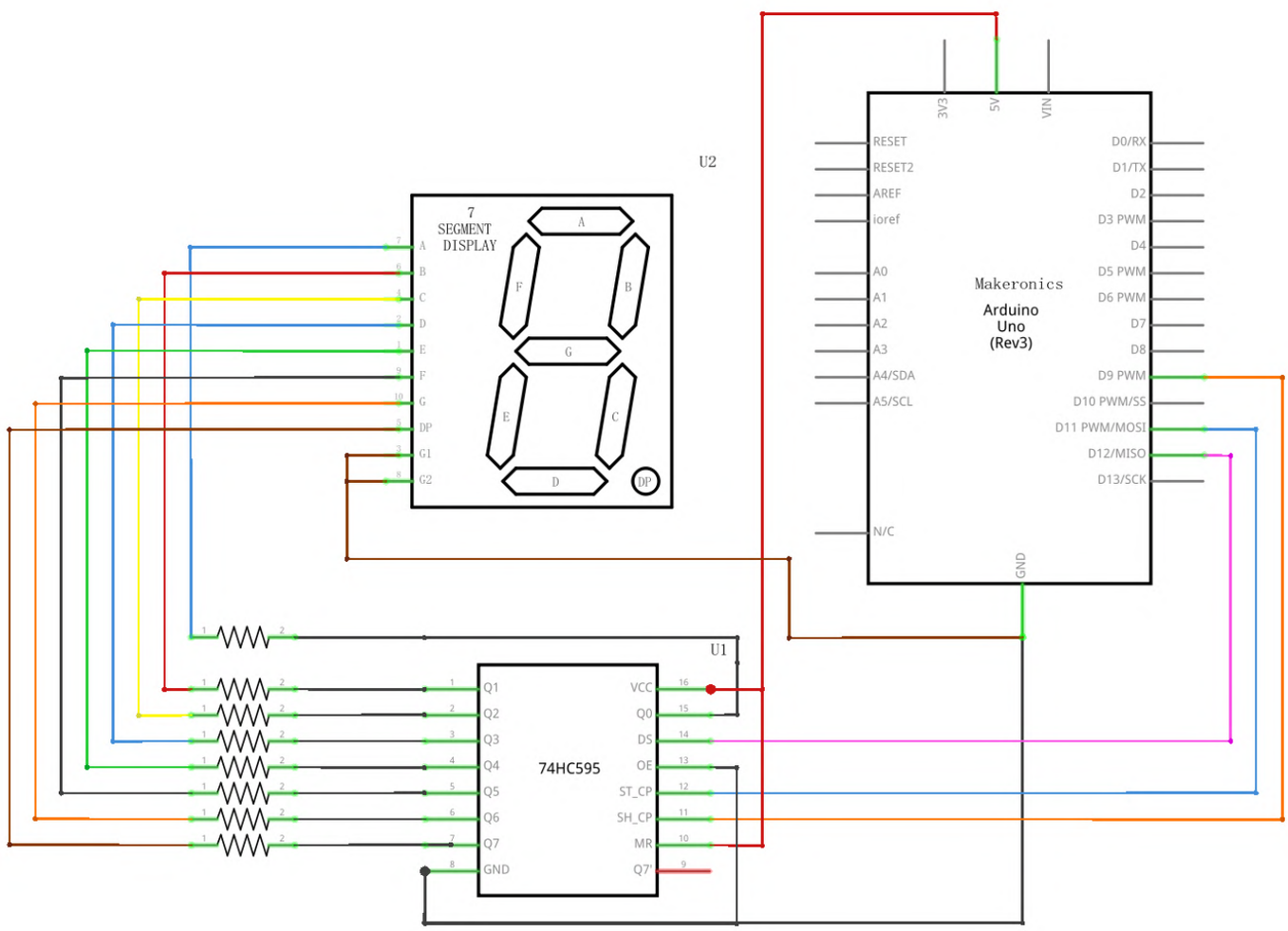
The 7-segment displays can be used in multiple applications. Displaying a digit value is the most used, Each segment is an LED that you can control. There are seven individual segments (plus the decimal points), as shown in figure below, and by turning specific segments on and off, you can make numbers and most letters of the English language.



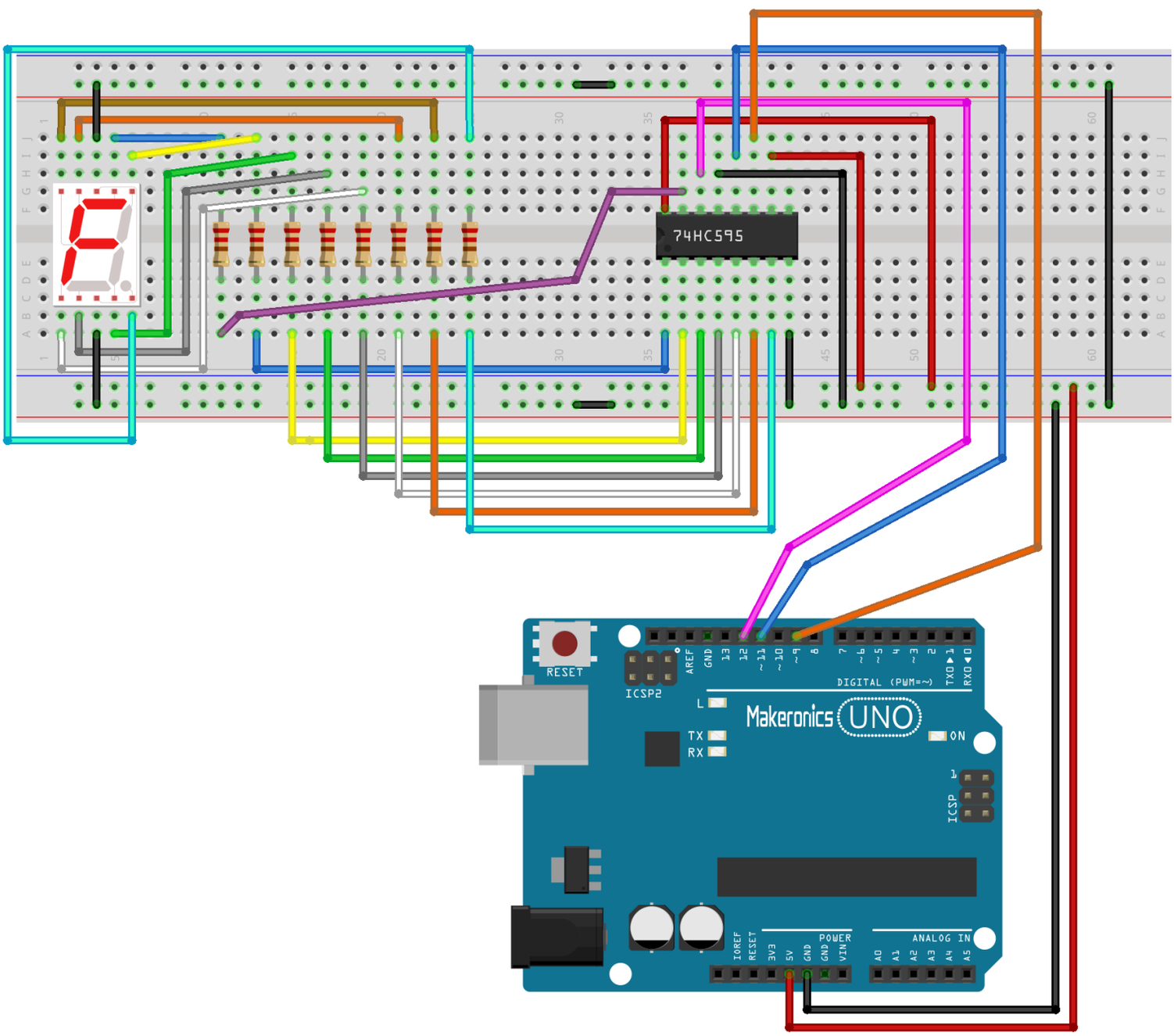
0-9 ten digits correspond with each segment are as follows (the following table applies common cathode seven segment display device, if you are using a common anode, the table should be replaced every 1 0 0 should all replaced by 1):

Digit	Display	gfedcba	abcdefg	a	b	c	d	e	f	g
0	0	0x3F	0x7E	**on**	**on**	**on**	**on**	**on**	**on**	off
1	1	0x06	0x30	off	**on**	**on**	off	off	off	off
2	2	0x5B	0x6D	**on**	**on**	off	**on**	**on**	off	**on**
3	3	0x4F	0x79	**on**	**on**	**on**	**on**	off	off	**on**
4	4	0x66	0x33	off	**on**	**on**	off	off	**on**	**on**
5	5	0x6D	0x5B	**on**	off	**on**	**on**	off	**on**	**on**
6	6	0x7D	0x5F	**on**	off	**on**	**on**	**on**	**on**	**on**
7	7	0x07	0x70	**on**	**on**	**on**	off	off	off	off
8	8	0x7F	0x7F	**on**	**on**	**on**	**on**	**on**	**on**	**on**
9	9	0x6F	0x7B	**on**	**on**	**on**	**on**	off	**on**	**on**
A	A	0x77	0x77	**on**	**on**	**on**	off	**on**	**on**	**on**
b	b	0x7C	0x1F	off	off	**on**	**on**	**on**	**on**	**on**
C	C	0x39	0x4E	**on**	off	off	**on**	**on**	**on**	off
d	d	0x5E	0x3D	off	**on**	**on**	**on**	**on**	off	**on**
E	E	0x79	0x4F	**on**	off	off	**on**	**on**	**on**	**on**
F	F	0x71	0x47	**on**	off	off	off	**on**	**on**	**on**

Connection Schematic:



Wiring diagram:



The following table shows the seven-segment display 74HC595 pin correspondence table:

74HC595 pin	Seven shows remarkable control pin (stroke)
Q0	7 (A)
Q1	6 (B)
Q2	4 (C)
Q3	2 (D)
Q4	1 (E)
Q5	9 (F)
Q6	10 (G)
Q7	5 (DP)

Connect 74HC595

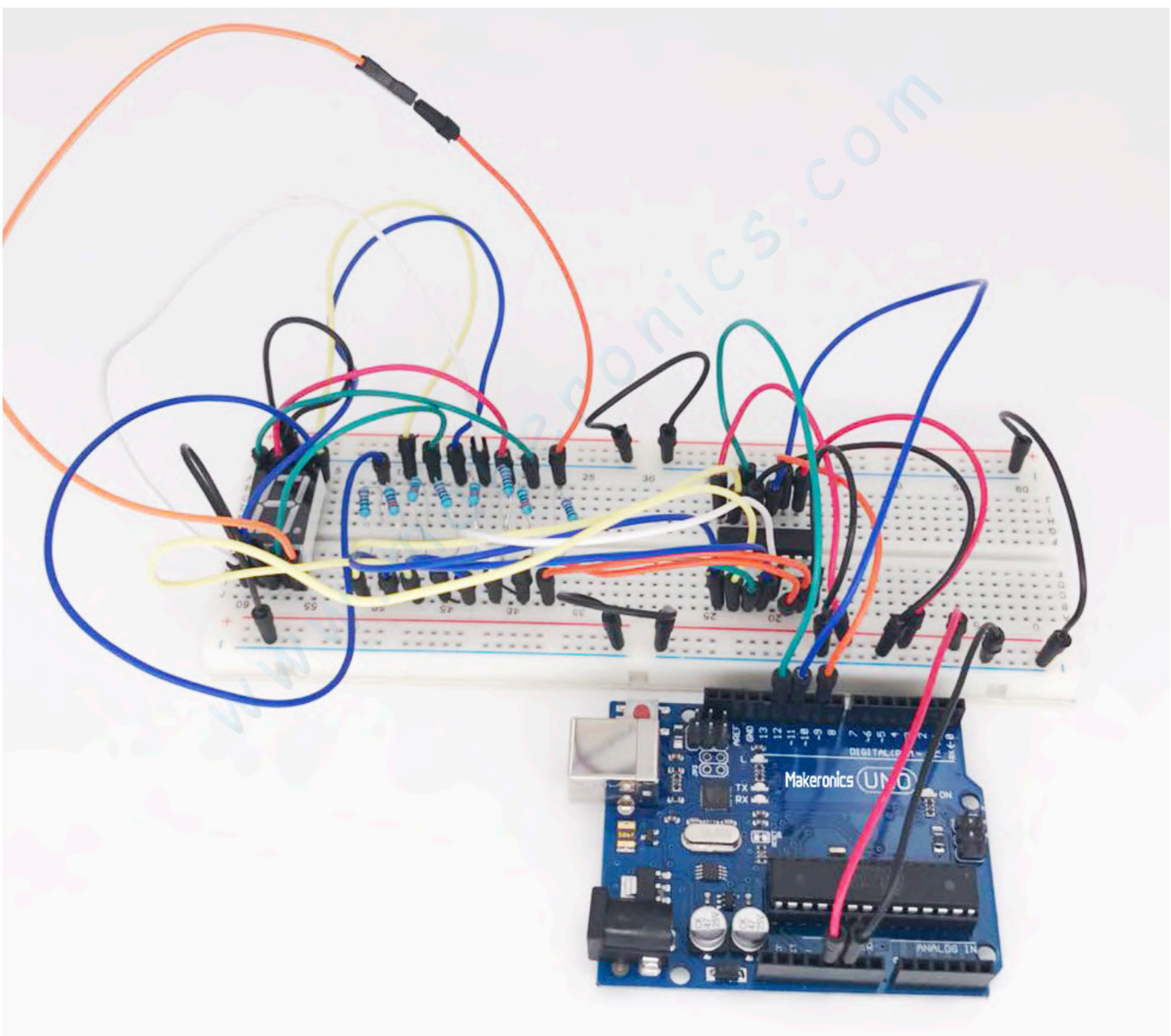
First, the wiring is connected to power and ground:
VCC (pin 16) and MR (pin 10) connected to 5V
GND (pin 8) and OE (pin 13) to ground
Connection DS, ST_CP and SH_CP pin:
DS (pin 14) connected to UNO R3 board pin 2
ST_CP (pin 12, latch pin) connected to UNO R3 board pin 3
SH_CP (pin 11, clock pin) connected to UNO R3 board pin 4

seconde, Connect the seven segment display:
The seven-segment display 3, 8 pin to UNO R3 board GND (This example uses the common cathode, if you use the common anode, please connect the 3, 8 pin to UNO R3 board + 5V)
According to the table above, connect the 74HC595 Q0 ~ Q7 to seven-segment display corresponding pin (A ~ G and DP), and then each foot in a 220 ohm resistor in series.

Code

After wiring, please open the program in the code folder- Lesson 24 74HC595 And Segment Display and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

Demo:



Project 21

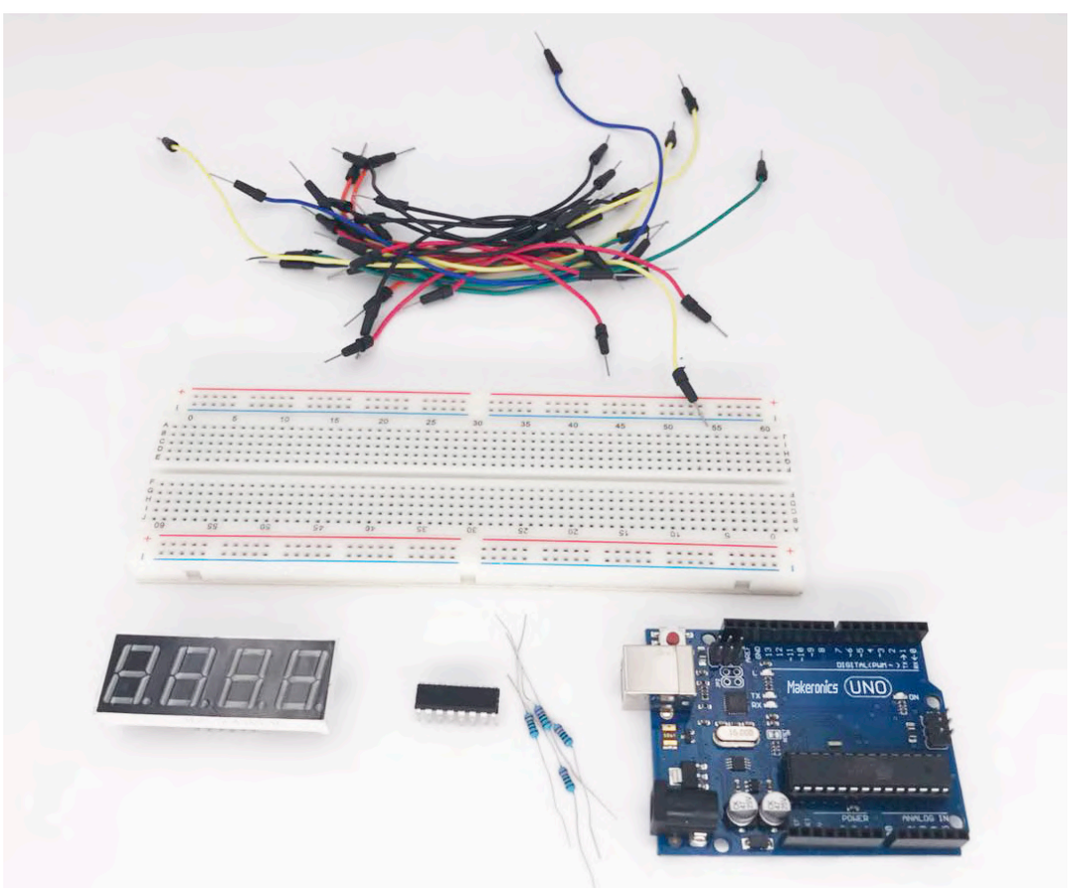
Four Digital Seven Segment Display

25

In this lesson, you will learn how to use a 4-digit 7-segment display.

Component Required:

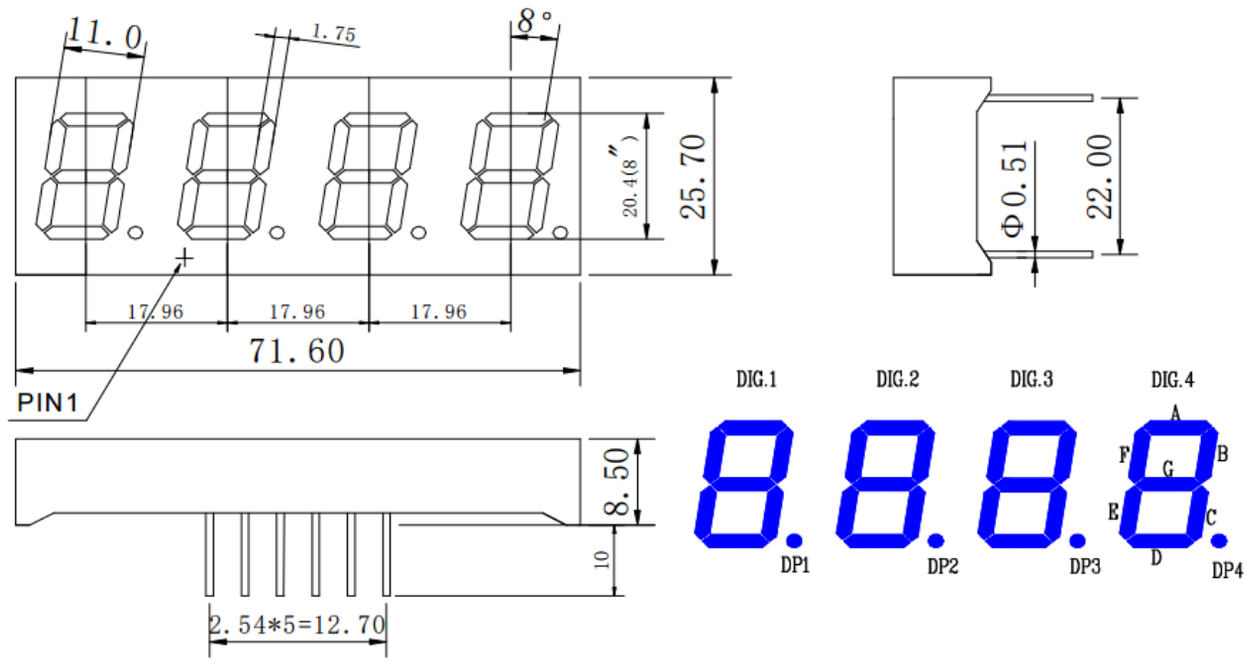
- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 1 x 74HC595 IC
- 1 x 4 Digit 7-Segment Display
- 4 x 220 ohm resistors
- 24 x M-M wires (Male to Male jumper wires)

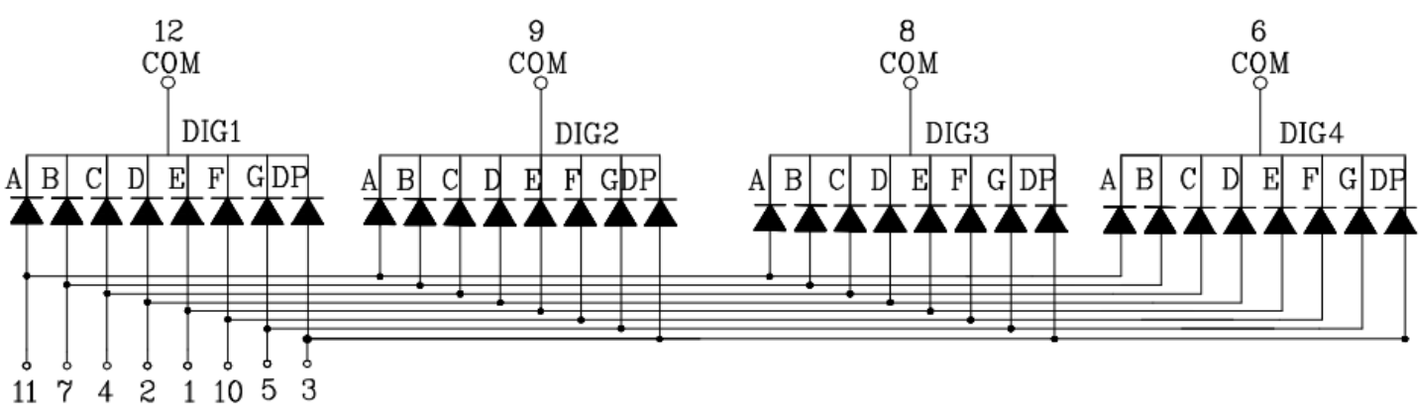


Component Introduction:

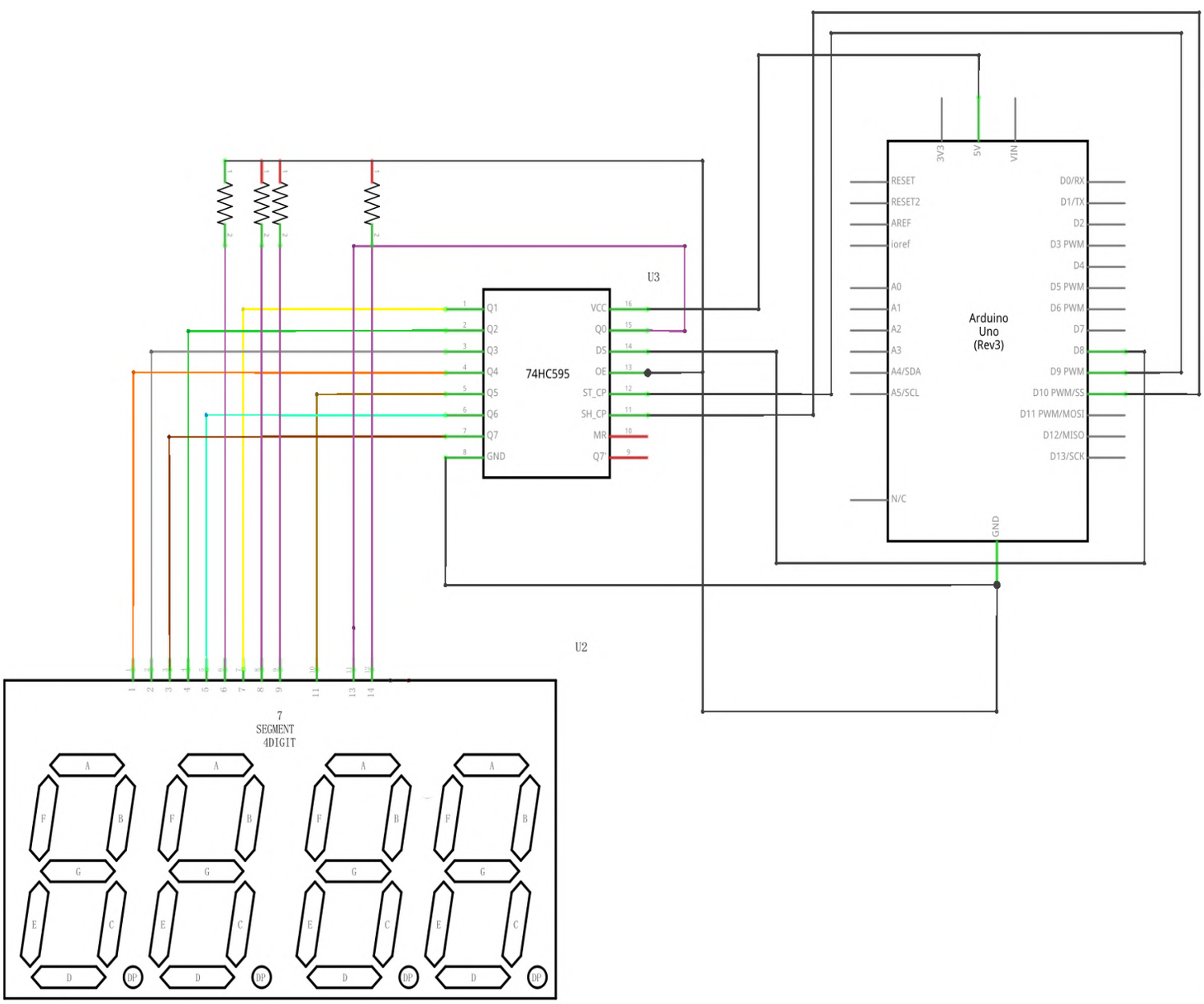
Four Digit Seven segment display

These common-cathode displays feature 4 x 7-segment digits and one decimal point per digit. The LEDs have a forward voltage of 1.9VDC and a max forward current of 20mA.

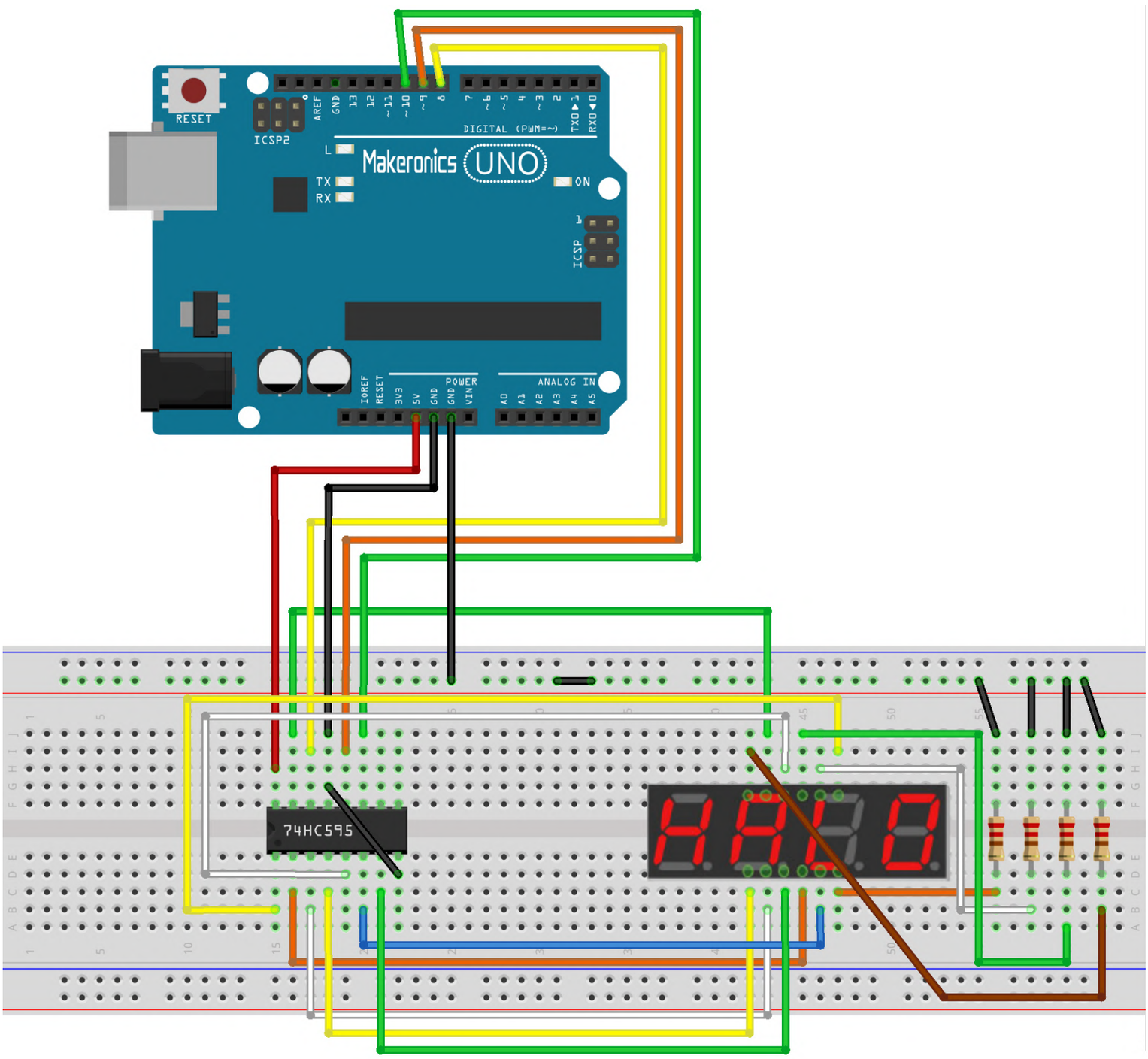




Connection Schematic:



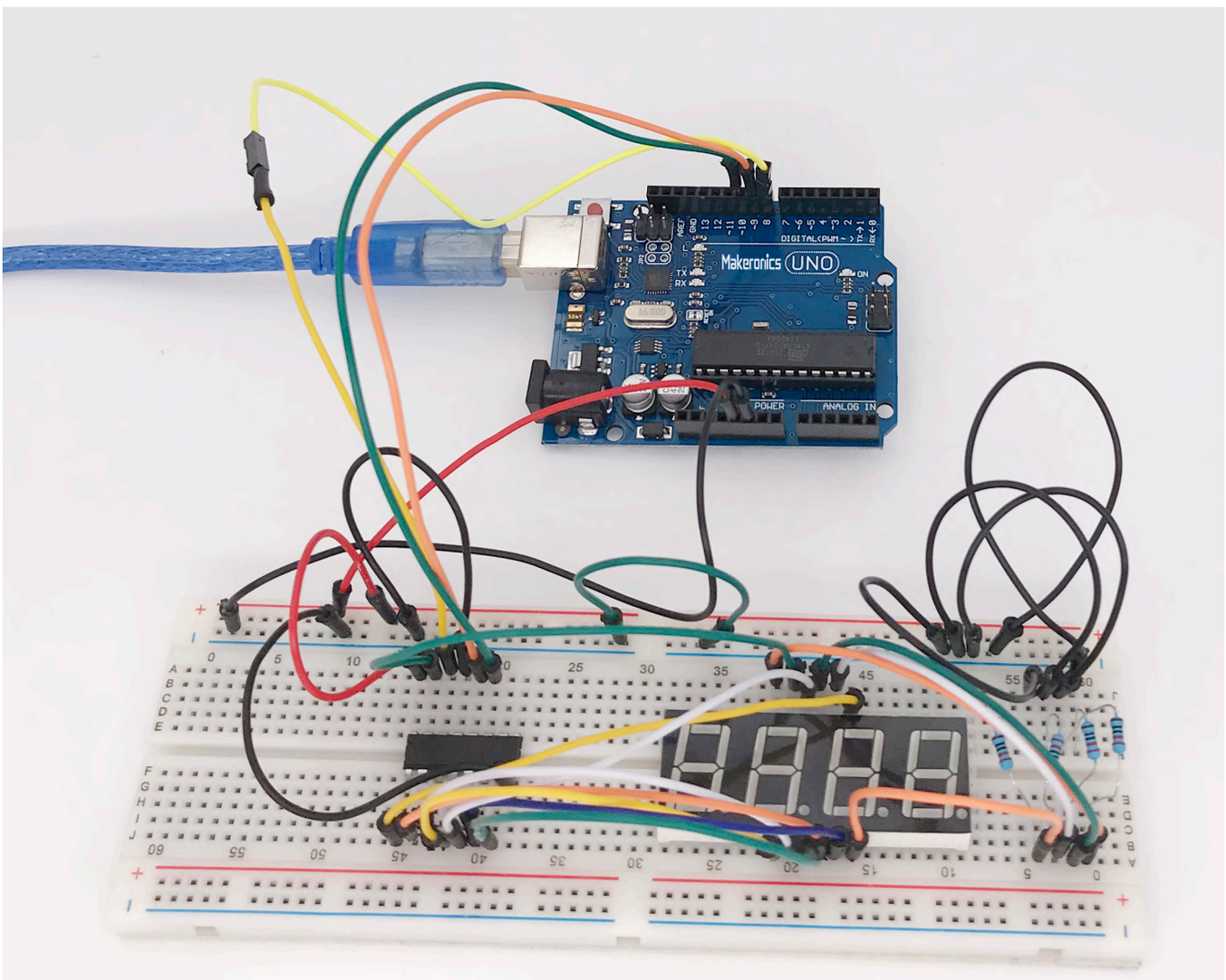
Wiring diagram:



Code:

After wiring, please open the program in the code folder- Lesson 25 Four Digital Seven Segment Display and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

Demo:



Project 22

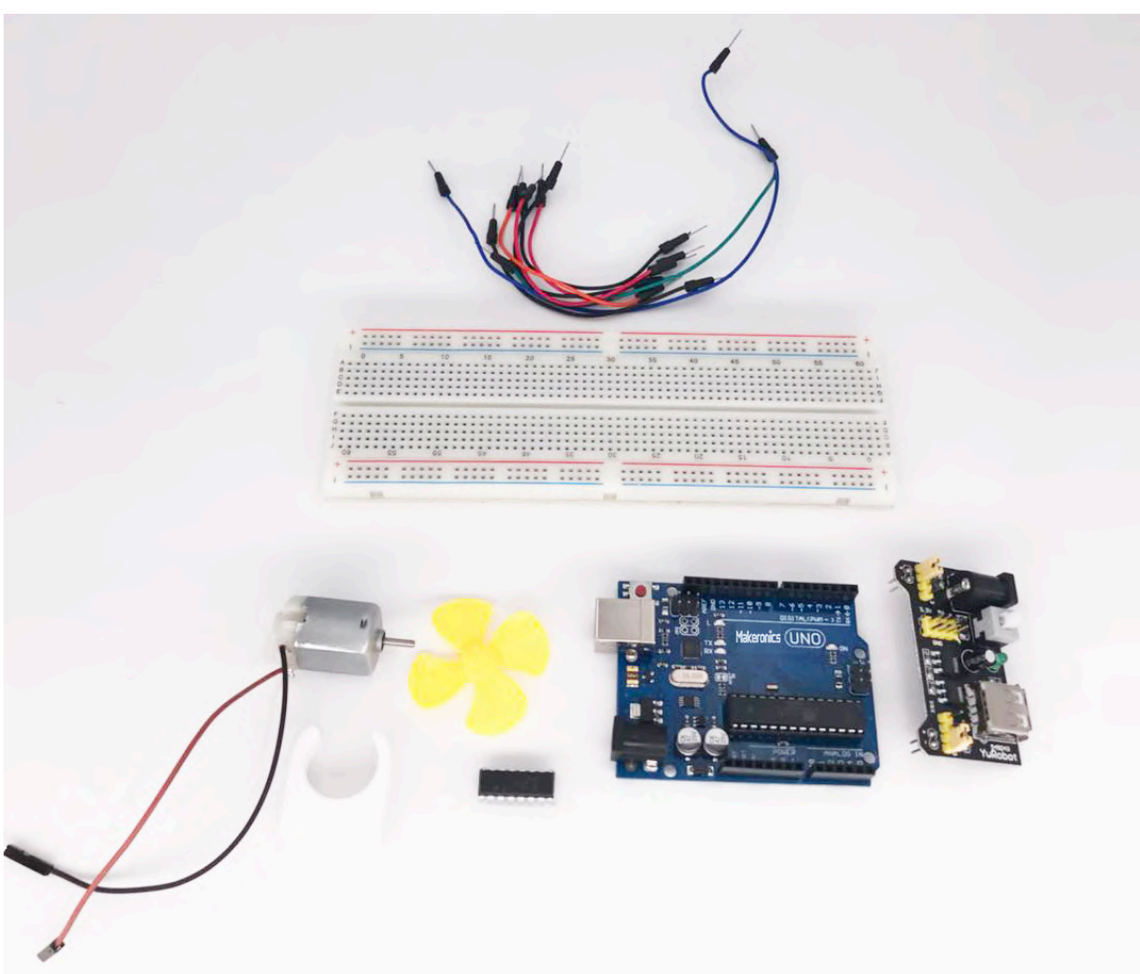
26

DC Motors

In this lesson, you will learn how to control a small DC motor using an UNO R3 and a transistor.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 1 x L293D IC
- 1 x Fan blade and 3-6v motor, motor bracket
- 8 x M-M wires (Male to Male jumper wires)
- 1 x Power Supply Module
- 1 x 9V battery

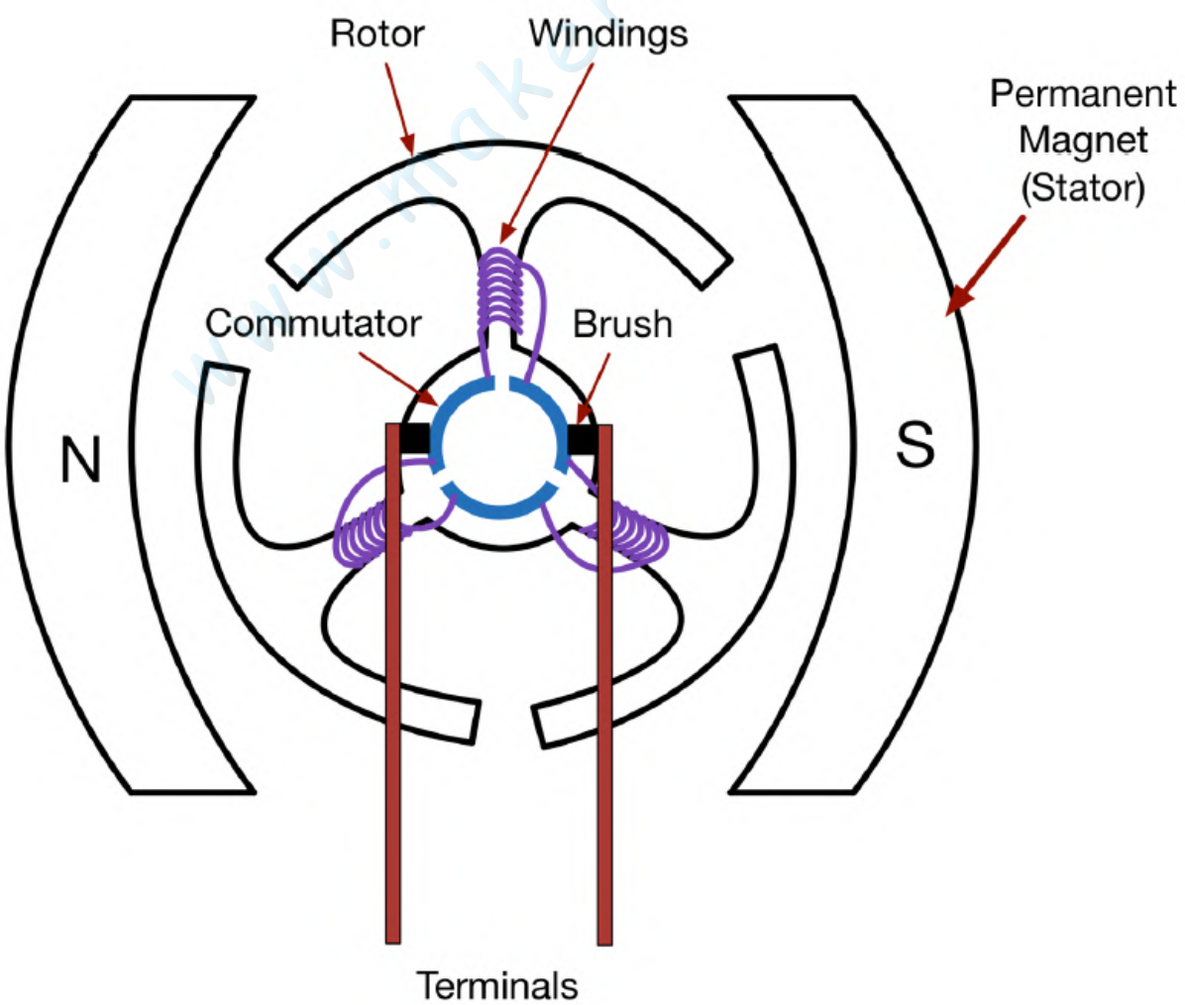


Component Introduction:

DC motor

Motors are also the driving force behind lots of other useful output devices like pumps and linear actuators.

DC motors generally have three major components, as shown in figure below. They have stationary magnets (stator) around the outside of the motor, a rotor (the bit that moves) and a commutator.
Coils of wire are wound around the rotor. In the case of figure below, there are three coils around three shaped parts of the rotor.These coils are connected to the commutator. The commutator's job is to energize the coils with the right polarity, in turn, as the rotor rotates, so that the next coil is always being pushed and pulled against the permanent magnets in the stator, so that the overall effect is that the rotor rotates.



The commutator is made up of a ring split into segments (in this case, three) and brushes are used to connect the terminals to the separate segments of the commutator as it rotates with the rotor.
A useful feature of DC motors is that when you reverse the polarity of the voltage across its terminals, it will spin in the opposite direction.

Breadboard Power Supply

The small DC motor is likely to use more power than an UNO R3 board digital output can handle directly. If we tried to connect the motor straight to an UNO R3 board pin, there is a good chance that it could damage the UNO R3 board. So we use a power supply module provides more power.

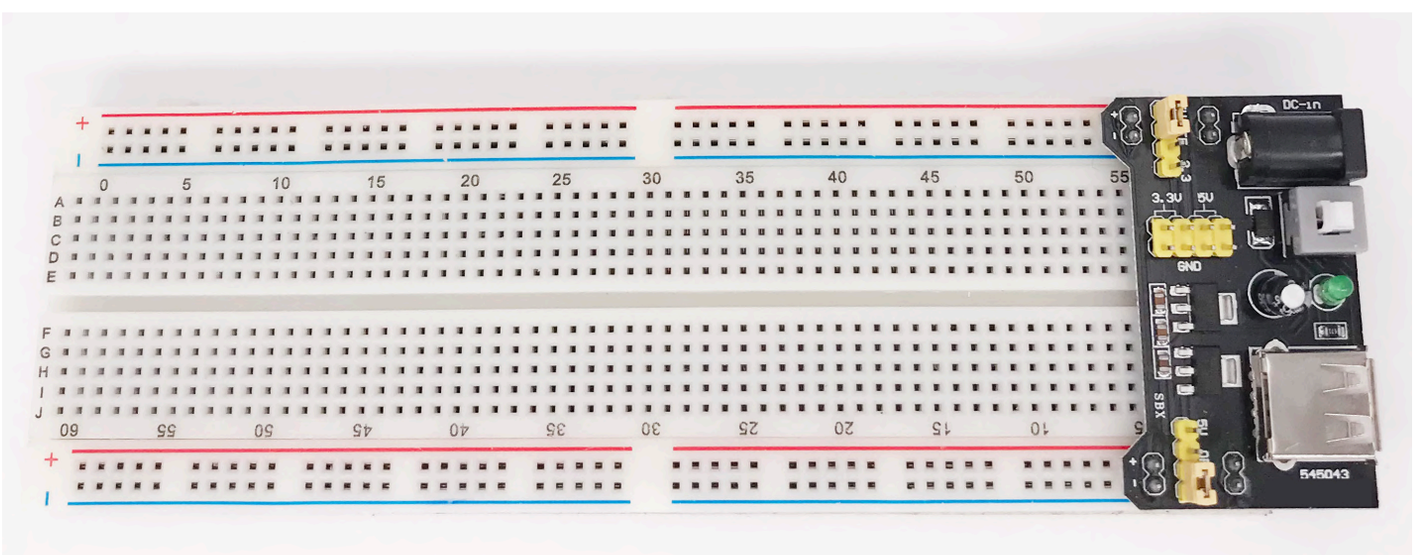
Product Specifications:

- Locking On/Off Switch
- LED Power Indicator
- Input voltage: 6.5-9v (DC) via 5.5mm x 2.1mm plug
- Output voltage: 3.3V/5v
- Maximum output current: 700 mA
- Independent control rail output. 0v, 3.3v, 5v to breadboard
- Output header pins for convenient external use
- Size: 2.1 in x 1.4 in
- USB device connector onboard to power external device

Setting up output voltage:

5V 3V

The left and right voltage output can be configured independently. To select the output voltage, move jumper to the corresponding pins. Note: power indicator LED and the breadboard power rails will not power on if both jumpers are in the "OFF" position.



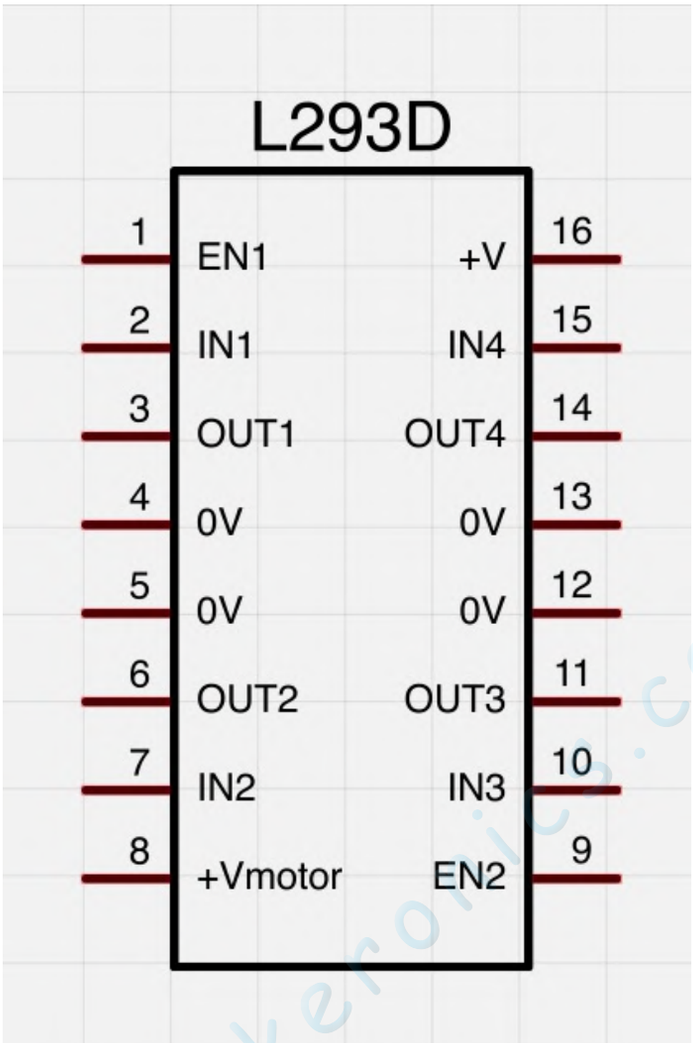
Important note:
Make sure that you align the module correctly on the breadboard. The negative pin(-) on module lines up with the blue line(-) on breadboard and that the positive pin(+) lines up with the red line(+). Failure to do so could result in you accidently reversing the power to your project

L293D

This is a very useful chip. It can actually control two motors independently. We are just using half the chip in this lesson, most of the pins on the right hand side of the chip are for controlling a second motor.

Product Specifications:

• Featuring Unitrode L293 and L293D Products Now From Texas Instruments
• Wide Supply-Voltage Range: 4.5 V to 36 V
• Separate Input-Logic Supply
• Internal ESD Protection
• Thermal Shutdown
• High-Noise-Immunity Inputs
• Functionally Similar to SGS L293 and SGS L293D
• Output Current 1 A Per Channel (600 mA for L293D)
• Peak Output Current 2 A Per Channel (1.2 A for L293D)
• Output Clamp Diodes for Inductive T ransient Suppression (L293D)



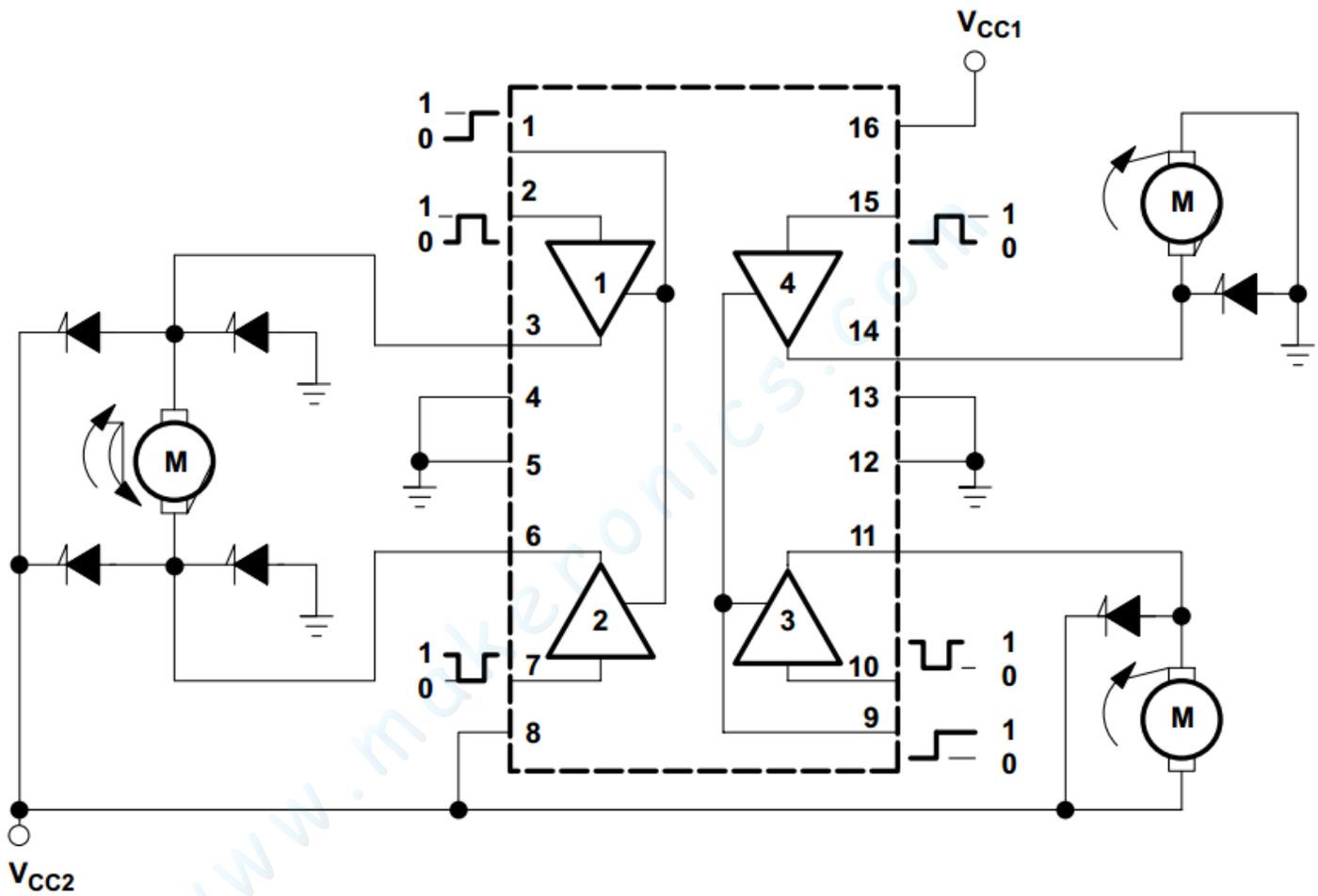
Description/ordering information

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/ high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase

with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

Block diagram



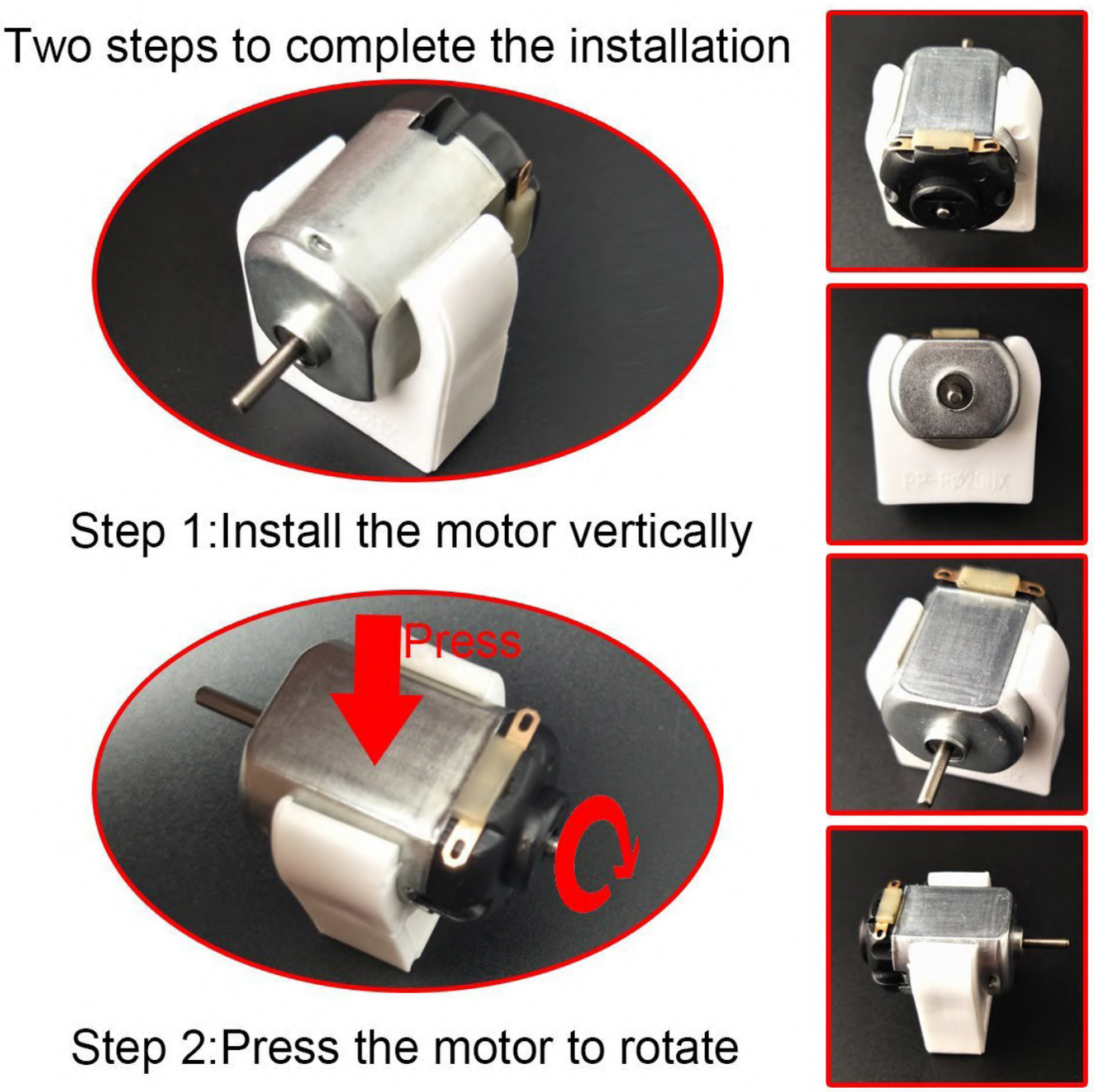
I got fed up with indecipherable pinout diagrams within datasheets, so have designed my own that I think gives more pertinent information.
There are 3 wires connected to the Arduino, 2 wires connected to the motor, and 1 wire connected to a battery.

To use this pinout:

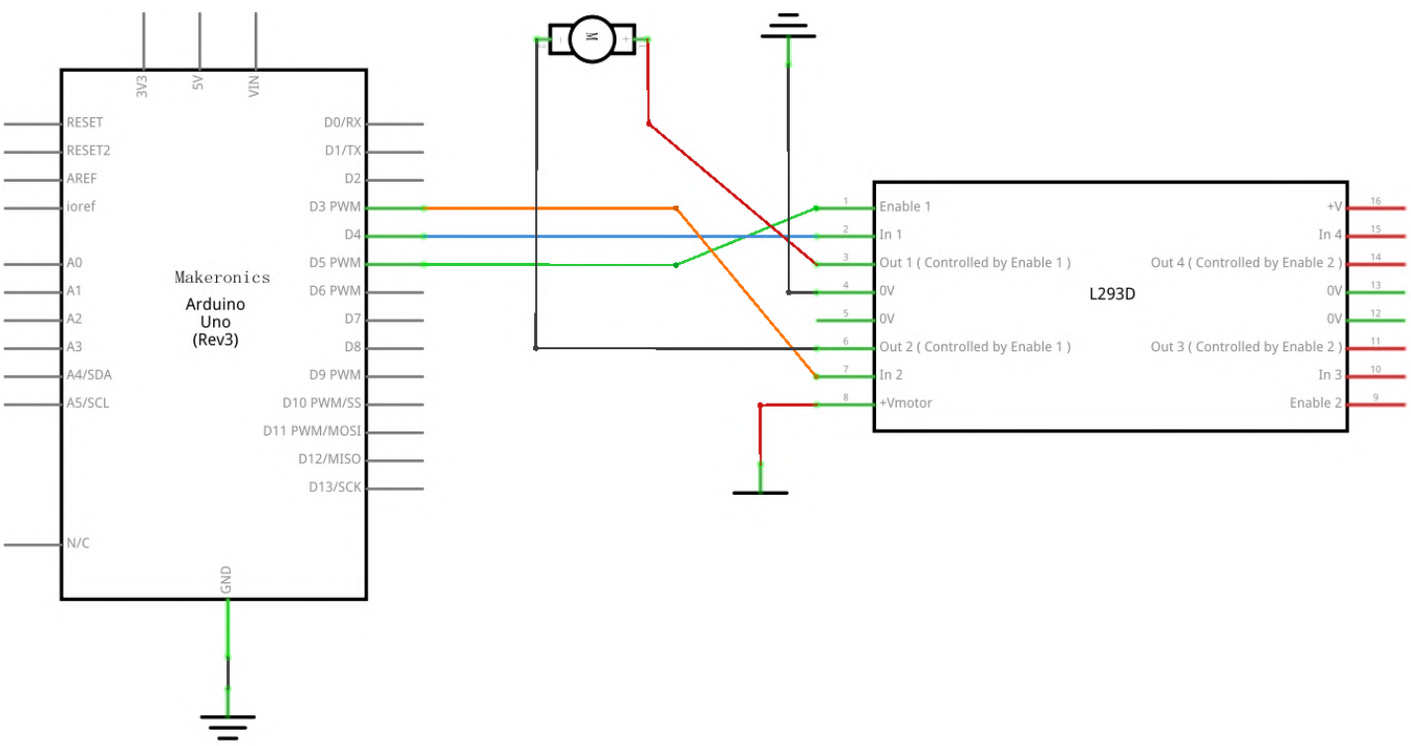
The left hand side deals with the first motor, the right hand side deals with a second motor.
Yes, you can run it with only one motor connected.

Arduino Connections
M1 PWM - connect this to a PWM pin on the Arduino. They're labelled on the Uno, pin 5 is an example. Output any integer between 0 and 255, where 0 will be off, 128 is half speed and 255 is
max speed.
M1 direction 0/1 and M1 direction 1/0 - Connect these two to two digital Arduino pins. Output one pin as HIGH and the other pin as LOW, and the motor will spin in one direction. Reverse the outputs to LOW and HIGH, and the motor will spin in the other direction.

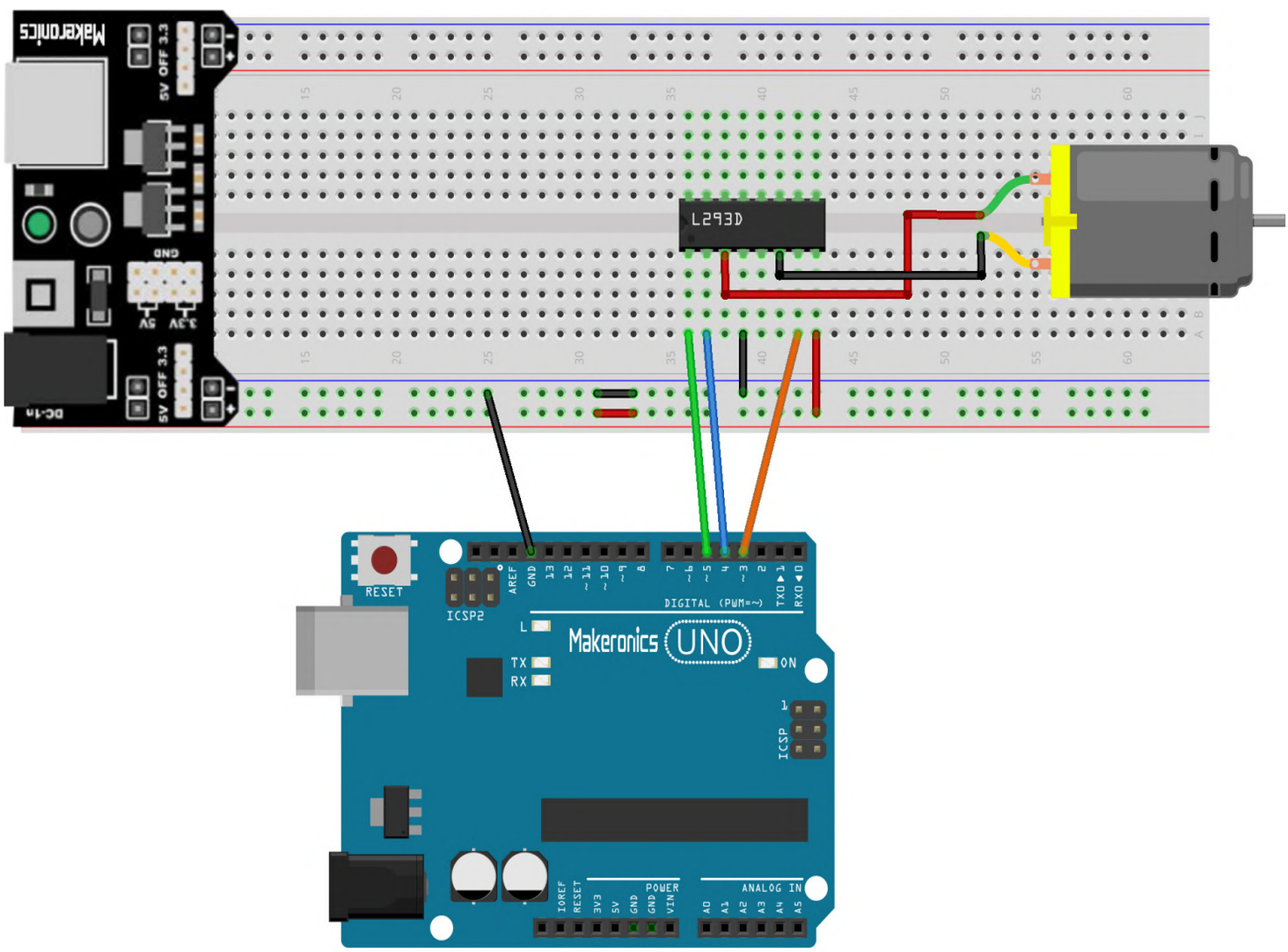
Bracket for motor:
The bracket is easy for motor to fix well. You can refer to the 2 steps below for installation. In fact, we just complete the step 1, which is enough for this experience.



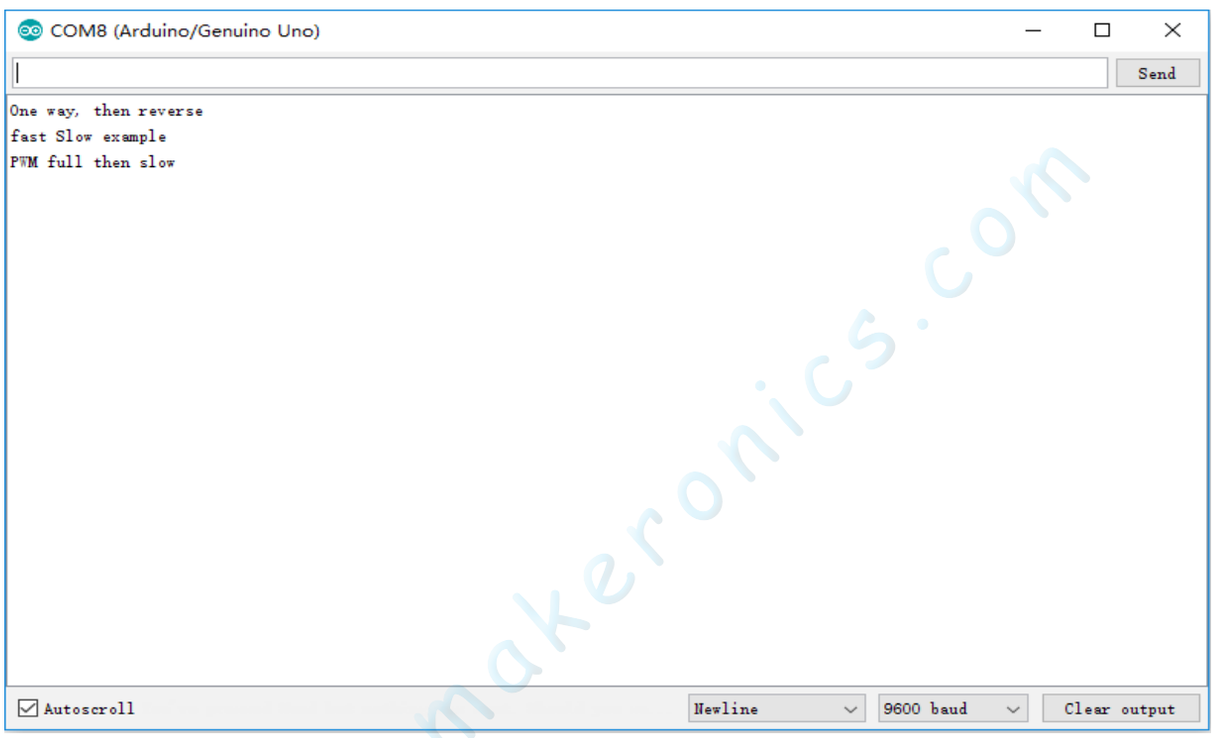
Connection Schematic:



Wiring diagram:



Like a motor, the coil of a relay is capable of producing voltage spikes when it is switched on and off. So note that this would be risky without the L293D controlling it. You should never connect a motor directly to the Arduino, because when you switch a motor off you get an electrical feedback. With a small motor, this will damage your Arduino, and with a large motor, you can watch an interesting flame and sparks effect.

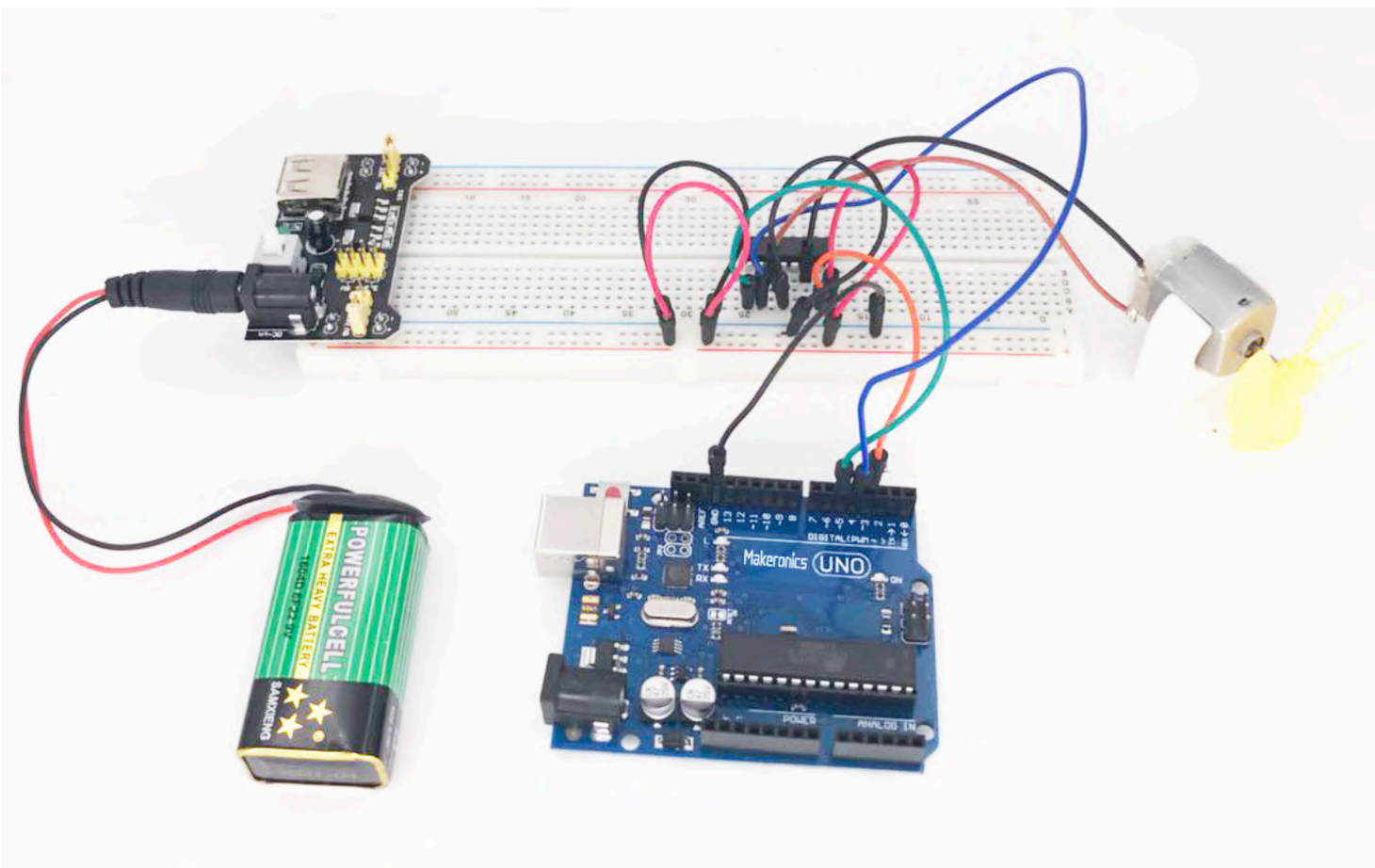


Code:

After wiring, please open the program in the code folder-Lesson 26 DC Motors and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

After program loading, turn on all the power switches. The motor will slightly rotate clockwise and anticlockwise for 5 times. Then, it will continue to dramatically rotate clockwise. After a short pause, it will dramatically rotate anticlockwise. Then the controller board will send PWM signal to drive the motor, the motor will slowly reduce its maximum RPM to the minimum and increase to the maximum again. Finally, it comes to a stop for 10s until the next cycle begins.

Demo:



Project 23

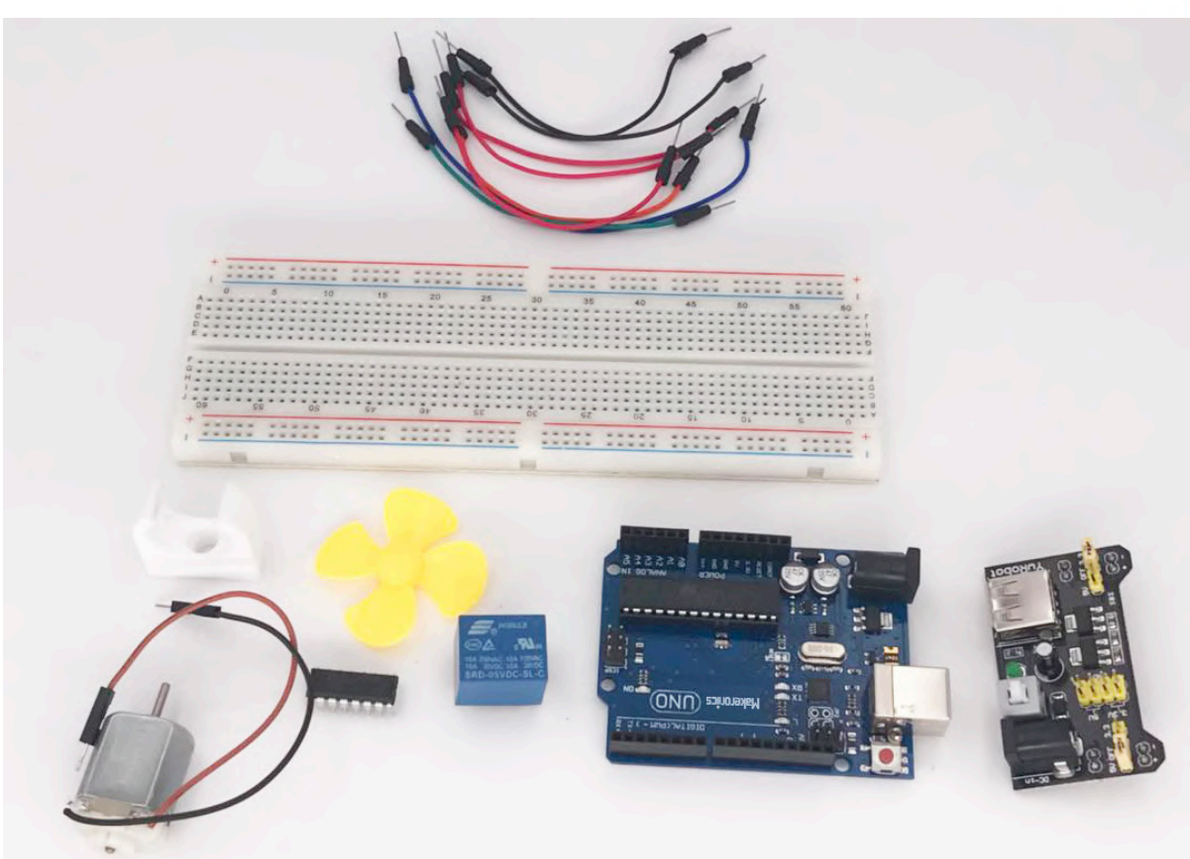
27

Relay

In this lesson, you will learn how to use a relay to turn on /off a motor.

Component Required:

(1) x Makeronics Uno R3
(1) x 830 tie-points breadboard
(1) x Fan blade and 3-6v dc motor, motor bracket
(1) x L293D IC
(1) x 5v Relay
(1) x Power Supply Module
(1) x 9V battery
(8) x M-M wires (Male to Male jumper wires)



Component Introduction:

If you only need your Arduino to occasionally turn a motor on and off, then one approach is to use a relay. Although this is often considered a rather old-fashioned way of doing things, it has a number of advantages:

• It's easy to do, requiring few components.
• Extremely good isolation between the electrically noisy, high-current motor and the Arduino.
• High-current handling (with the right relay).
• You can use ready-made relay modules directly with a Raspberry Pi or Arduino.

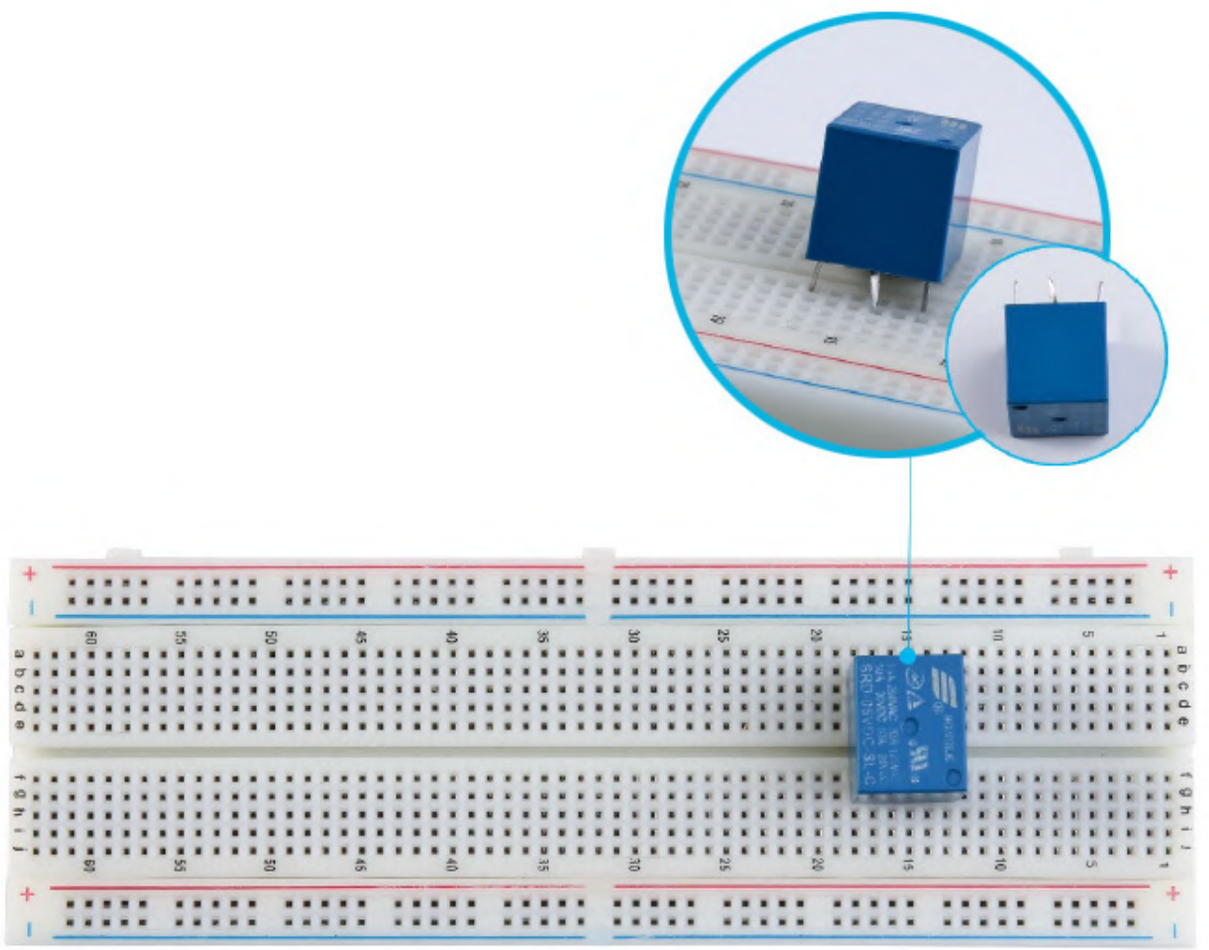
The downsides to using a relay or a relay module include the following:

• They are relatively large components.
• They can only switch the motor on and off—they cannot control the speed.
• They are electromechanical devices and typically are expected to survive perhaps 10,000,000 switching operations before something in them breaks.

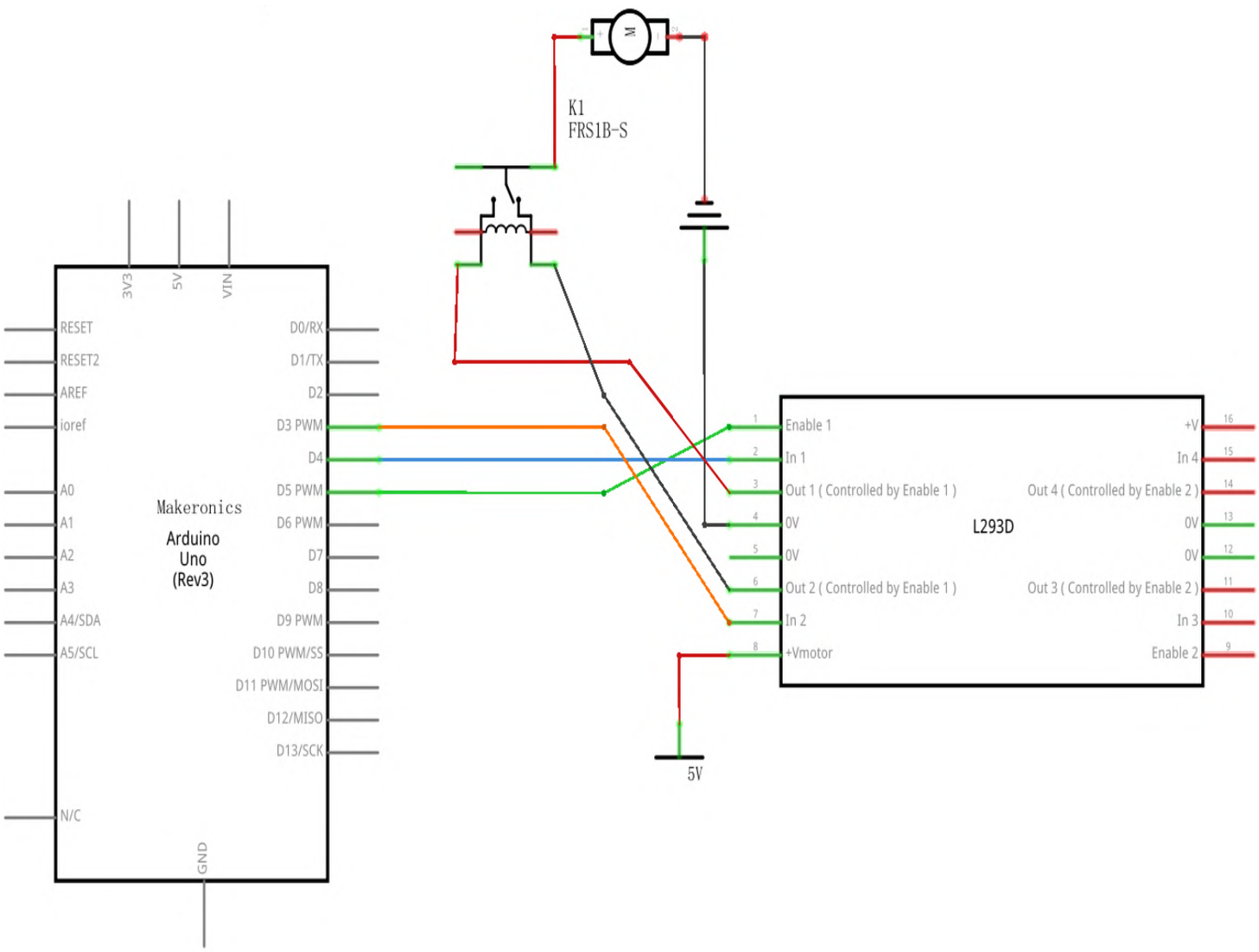
Relay

Relays like this are called Single Pole Change Over (SPCO), because rather than have just two terminals that are either connected or not, they actually have three. They have what is called a common terminal, usually labeled COM, and two other terminals: normally open (NO) and normally closed (NC). In this context, normally means without power applied to the relay coil. So, the NO and COM connections will be "open" (not connected) until the coil is energized. The NC contact will behave in the opposite manner, so the NC and COM terminals will "normally" be connected, but will disconnect when the coil is energized. Generally speaking, just the NO and COM terminals of the relay are used for switching.

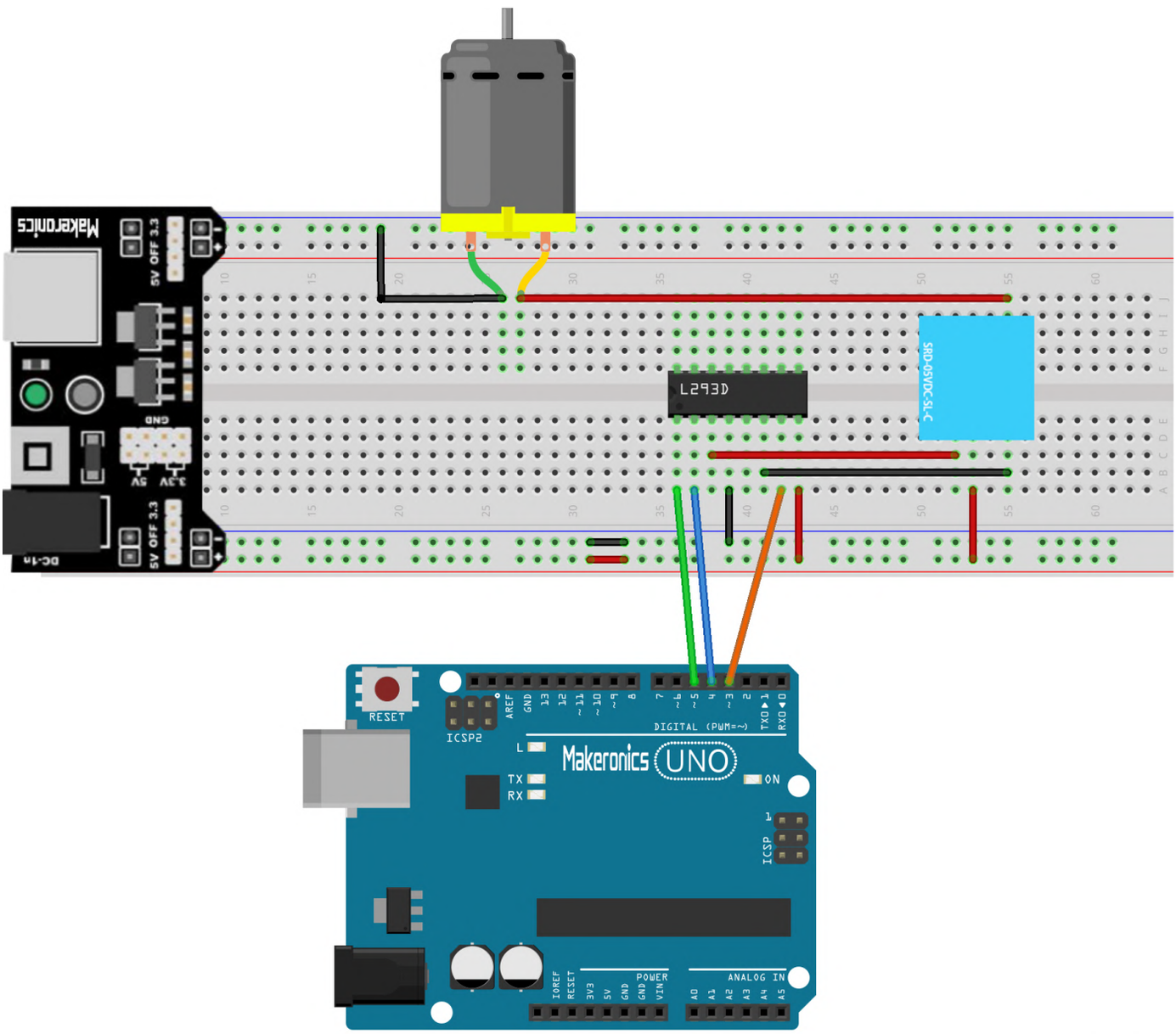
The Below is the schematic of how to drive relay with Arduino.
You may be confused about how to insert the relay into the bread board. As the picture below shows, you will have to bend one of the pins of the relay slightly then you can insert it into the bread board.



Connection Schematic:



Wiring diagram:

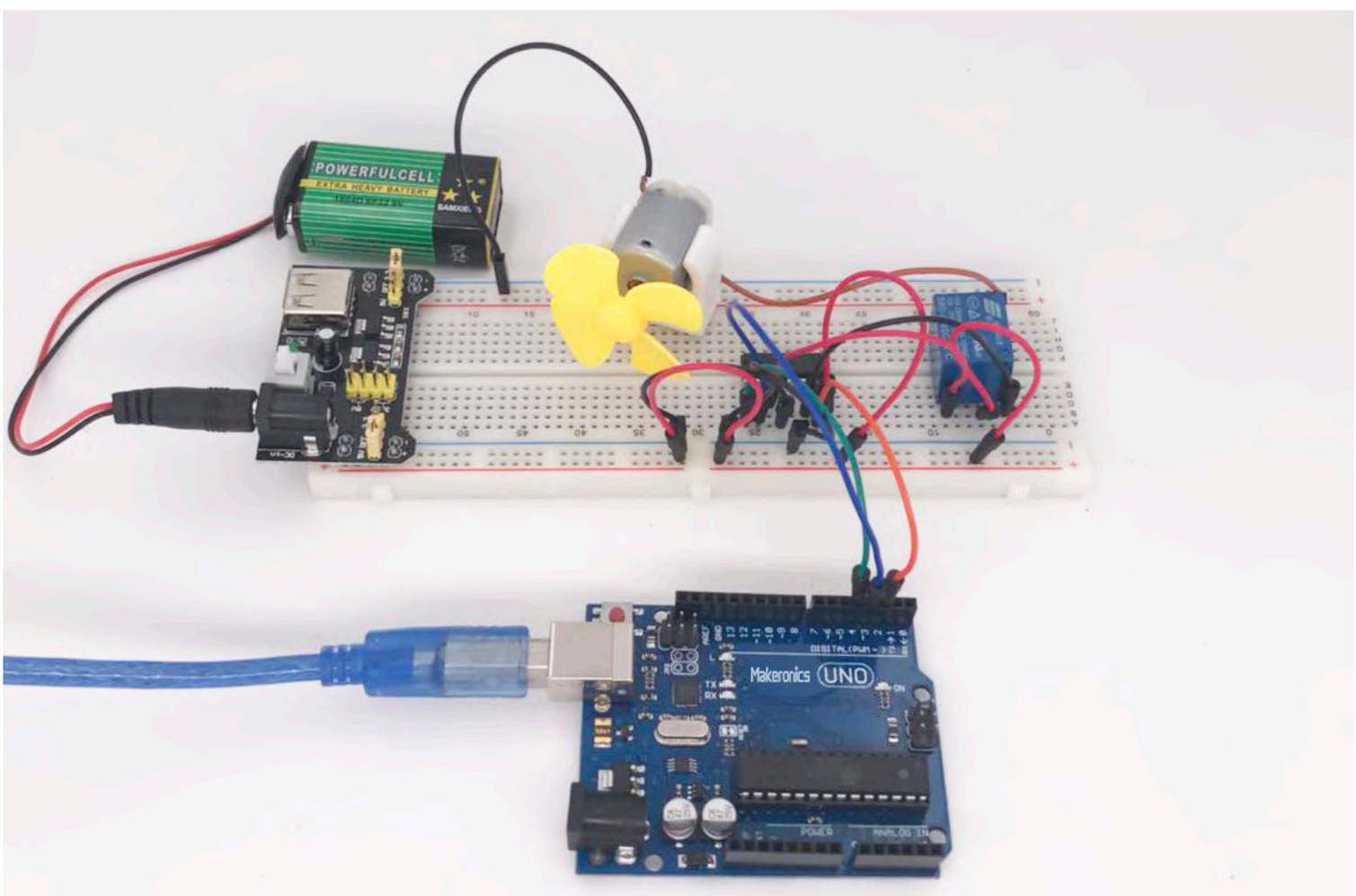


Code:

After wiring, please open the program in the code folder-Lesson 27 Relay and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

After program loading, turn on all the power switches. The relay will pick up with a ringing sound. Then, the motor will rotate. After a period of time, the relay will be released, and the motor stops.

Demo:



Project 24

28

Stepper Motor

In this lesson, you will learn a fun and easy way to drive a stepper motor.
The stepper we are using comes with its own driver board making it easy to connect to our UNO.
Since we don't want to drive the motor directly from the UNO, we will be using an inexpensive little breadboard power supply that plugs right into our breadboard and power it with a 9V 1Amp power supply.

Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 1 x ULN2003 stepper motor driver module
- 1 x Stepper motor
- 1 x 9V battery
- 1 x Power supply module
- 6 x F-M wires (Female to Male DuPont wires)
- 5 x M-M wire (Male to Male jumper wire)

Component Introduction:

Stepper Motor

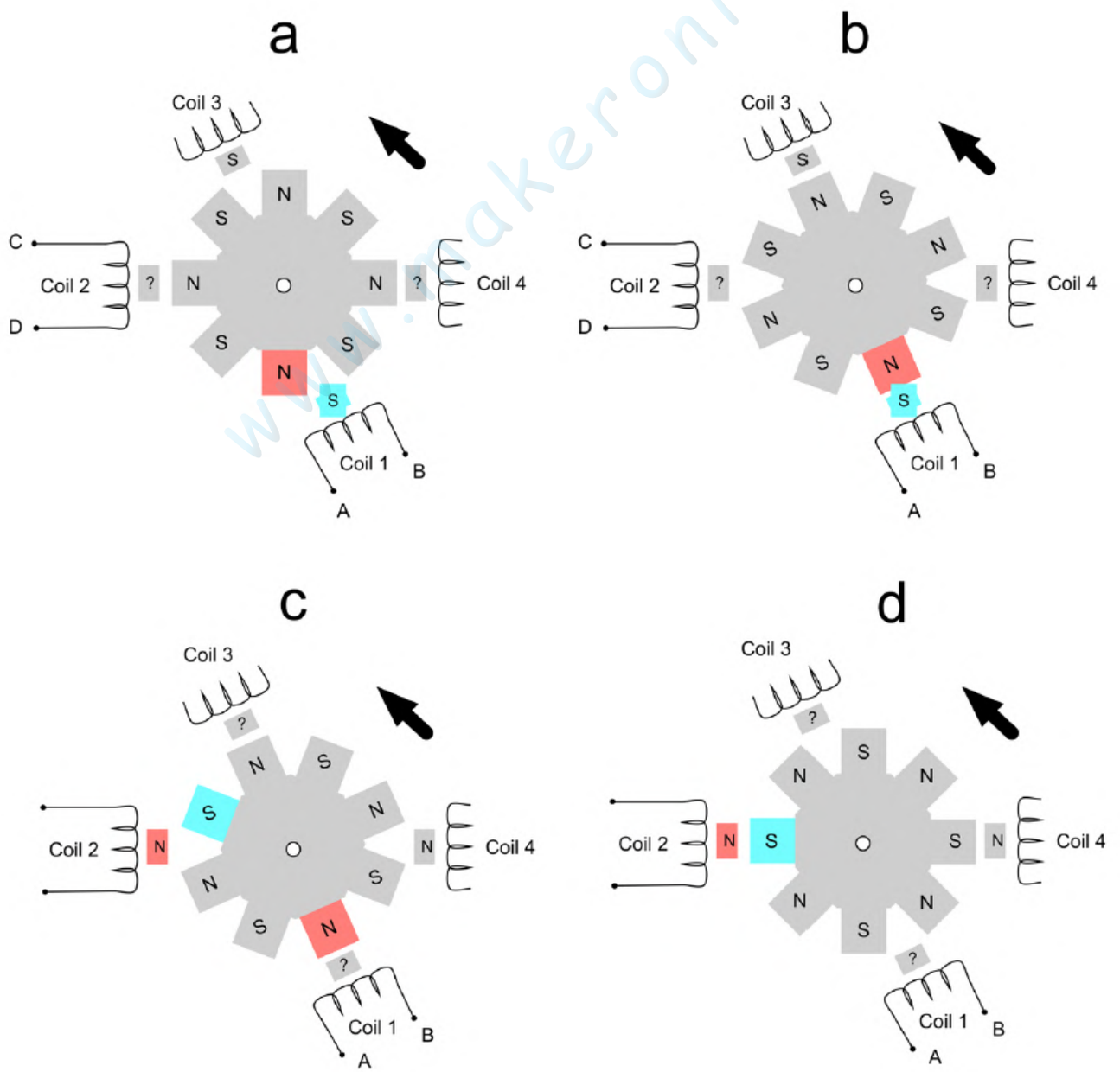
Stepper motors fall somewhere in between a regular DC motor and a servo motor. They have the advantage that they can be positioned accurately, moved forward or backwards one 'step' at a time, but they can also rotate continuously.

As the name suggests, stepper motors rotate in a series of short steps. This has the advantage that if the motor is free to turn, then by counting the steps you tell it to turn through, you know how much it has turned. This is one reason why stepper motors get used in printers, both of the 2D and 3D variety, to accurately position paper, a 3D printer bed, or print nozzle.

Stepper motor 28BYJ-48 Parameters

- Model: 28BYJ-48
- Rated voltage: 5VDC
- Number of Phase: 4
- Speed Variation Ratio: 1/64
- Stride Angle: 5.625° /64
- Frequency: 100Hz
- DC resistance: 50Ω±7%(25℃)
- Idle In-traction Frequency: > 600Hz
- Idle Out-traction Frequency: > 1000Hz
- In-traction Torque >34.3mN.m(120Hz)
- Self-positioning Torque >34.3mN.m
- Friction torque: 600-1200 gf.cm
- Pull in torque: 300 gf.cm
- Insulated resistance >10MΩ(500V)
- Insulated electricity power: 600VAC/1mA/1s
- Insulation grade: A
- Rise in Temperature <40K(120Hz)
- Noise <35dB(120Hz,No load,10cm)

The bipolar stepper motor usually has four wires coming out of it. Unlike unipolar steppers, bipolar steppers have no common center connection. They have two independent sets of coils instead. You can distinguish them from unipolar steppers by measuring the resistance between the wires. You should find two pairs of wires with equal resistance. If you've got the leads of your meter connected to two wires that are not connected (i.e. not attached to the same coil), you should see infinite resistance (or no continuity).

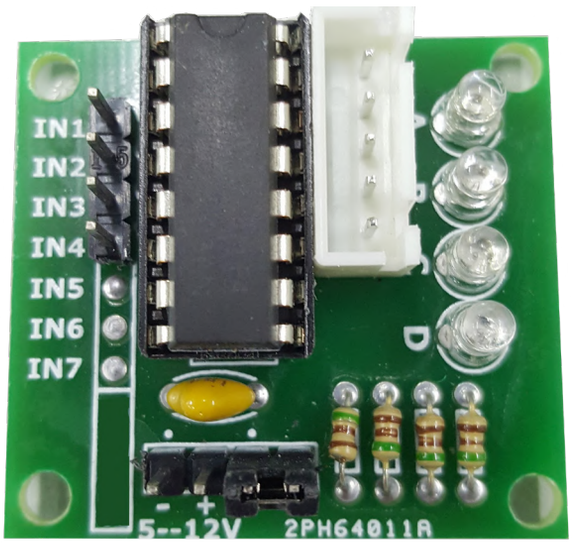


Typically, there will be four coils, the opposite coils being connected together so that they work in unison. The coils are all on the stator (outside of the motor) and therefore there is no need for a commutator and brushes as there is in a DC motor.

In a stepper motor, the rotor is made in the form of a cog with alternating north and south poles marked by N and S. There will normally be a lot more cogs on the rotor than the number shown in figure above. Each of the coils can be energized to be magnetic north or south depending on the direction of the current through the coil. Coils 1 and 3 operate together, so that if Coil 1 is N so will Coil 3 be. The same applies to Coils 2 and 4.

Starting with figure a above, if Coil 1 (and therefore Coil 3) is energized to be S then opposites attract and like poles repel, so the rotor will rotate conter-clockwise until the S of Coil 1 is aligned with the nearest N cog, as shown in figure b above. To keep the clockwise turning, the next step is to energize Coils 2 and 4 to be N (figure c above), pulling the nearest S cog to Coil 2 level with the coil.

After all this effort, the motor has now advanced one step. To continue rotating counterclockwise, Coil 1 now needs to be N.



Product Description

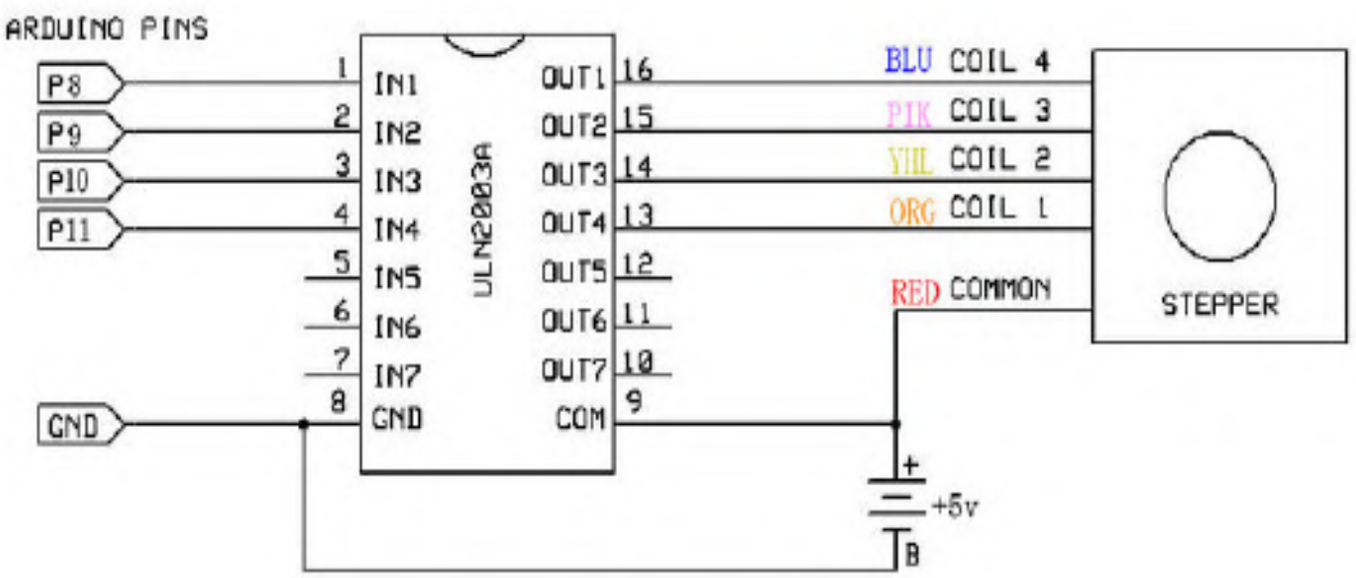
• Size: 42mmx30mm
• Use ULN2003 driver chip, 500mA
• A. B. C. D LED indicating the four phase stepper motor working condition.
• White jack is the four phase stepper motor standard jack.
• Power pins are separated
• We kept the rest pins of the ULN2003 chip for your further prototyping.

The simplest way of interfacing a unipolar stepper to Arduino is to use a breakout for ULN2003A transistor array chip. The ULN2003A contains seven Darlington transistor drivers and is somewhat like having seven TIP120 transistors all in one package. The ULN2003A can pass up to 500 mA per channel and has an internal voltage drop of about 1V when on. It also contains internal clamp diodes to dissipate voltage spikes when driving inductive loads. To control the stepper, apply voltage to each of the coils in a specific sequence.

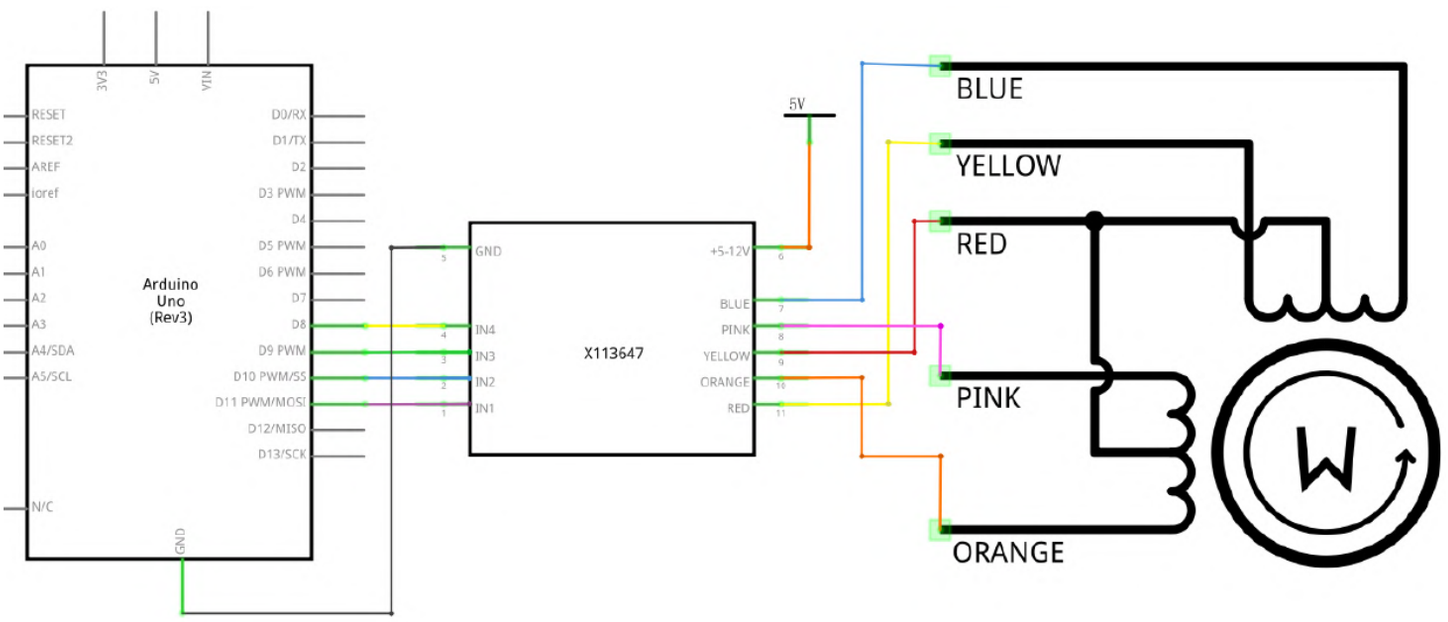
The sequence would go like this:

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 ORG	-	-						-
3 YEL		-	-	-				
2 PIK				-	-	-		
1 BLU						-	-	-

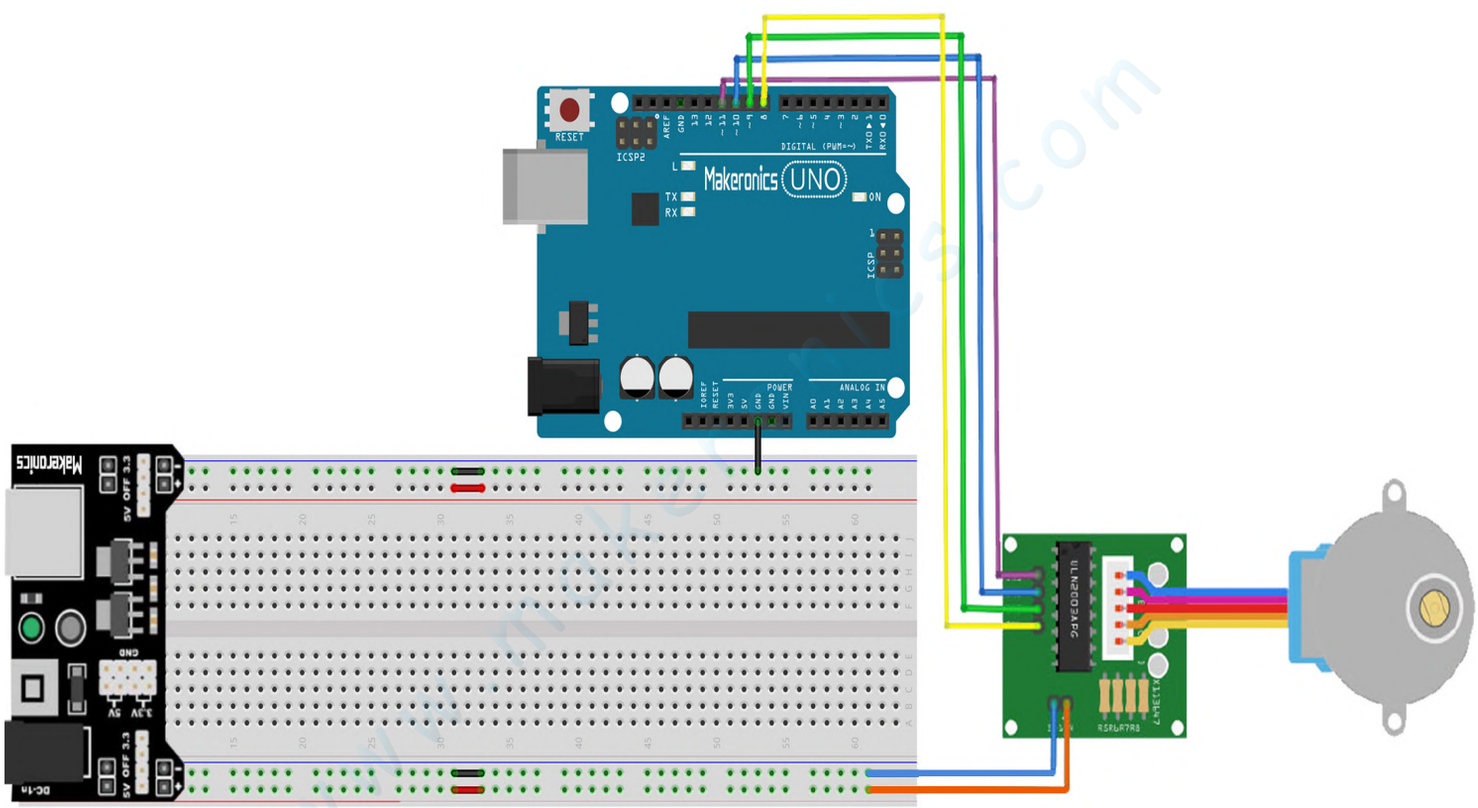
Here are schematics showing how to interface a unipolar stepper motor to four controller pins using a ULN2003A, and showing how to interface using four com.



Connection Schematic:

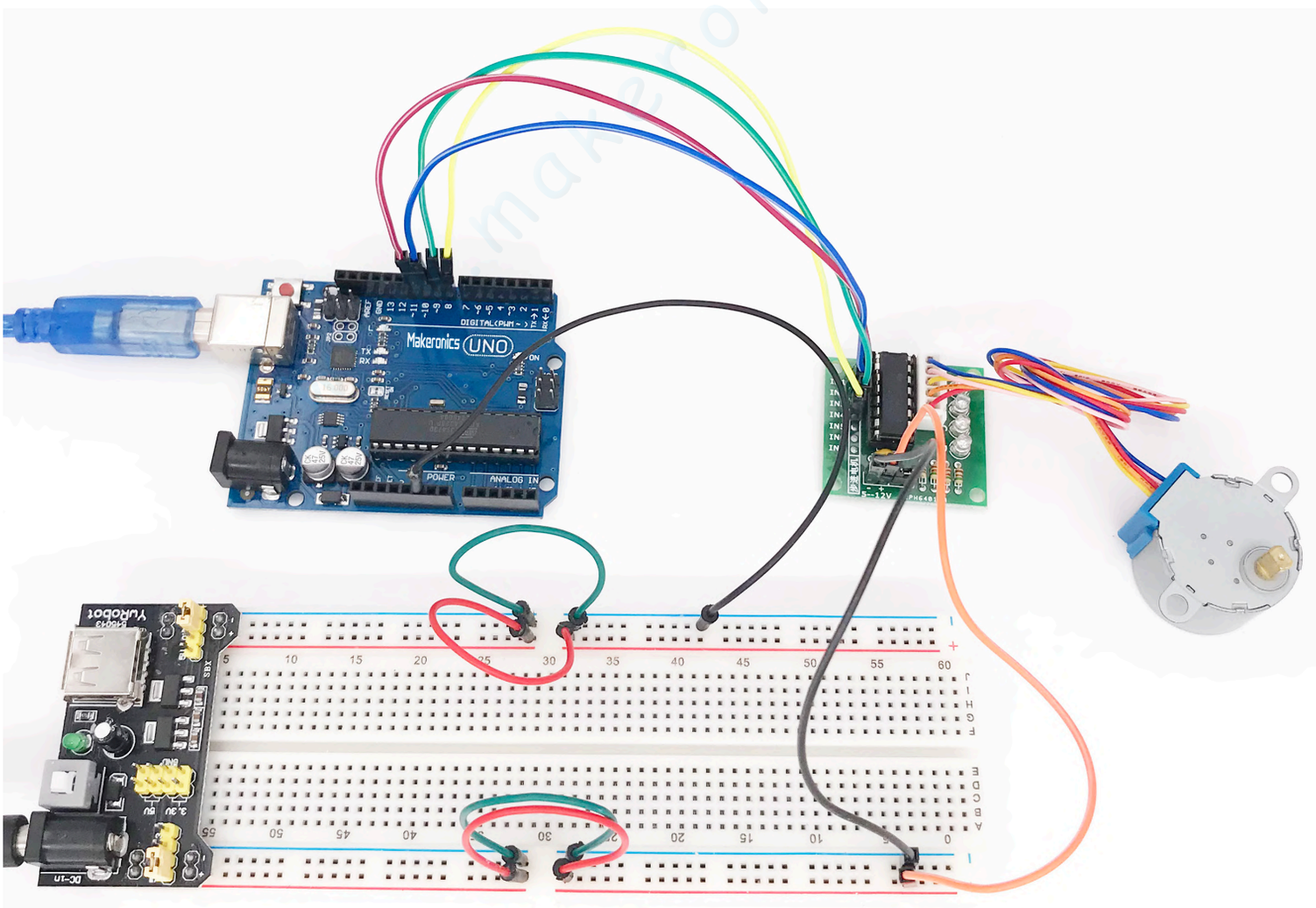


Wiring diagram:



We are using 4 pins to control the Stepper.
Pin 8-11 are controlling the Stepper motor.
We connect the Ground from to UNO to the Stepper motor.
Code
After wiring, please open the program in the code folder- Lesson 28 Stepper Motor and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < Stepper > library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.

Demo:



Project25

Controlling Stepper Motor With Remote

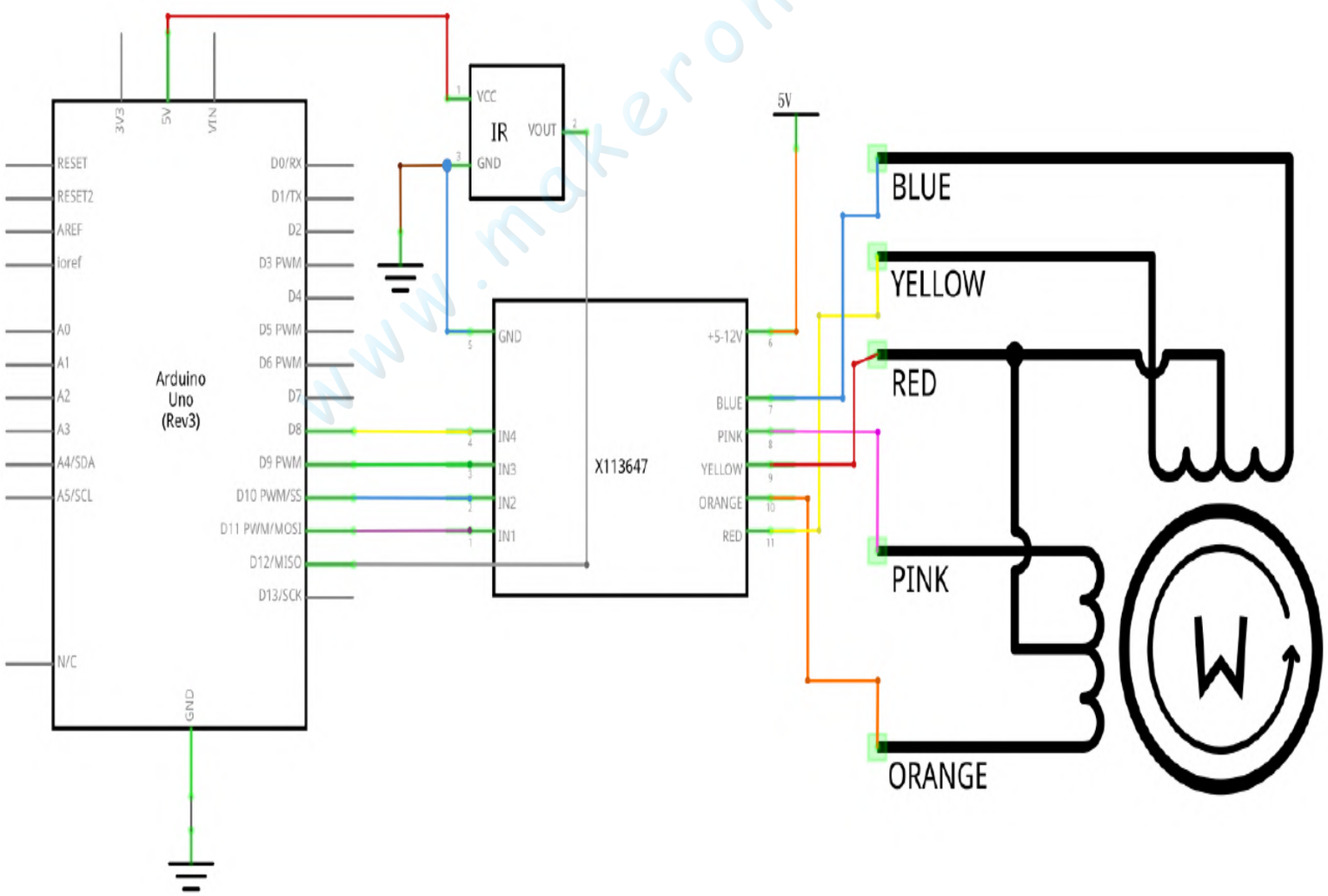
29

In this lesson, you will learn a fun and easy way to control a stepper motor from a distance using an IR remote control.
The stepper we are using comes with its own driver board making it easy to connect to our UNO.
Since we don't want to drive the motor directly from the UNO, we will be using an inexpensive little breadboard power supply that plugs right into our breadboard and power it with a 9V 1Amp power supply.
The IR sensor is connected to the UNO directly since it uses almost no power.

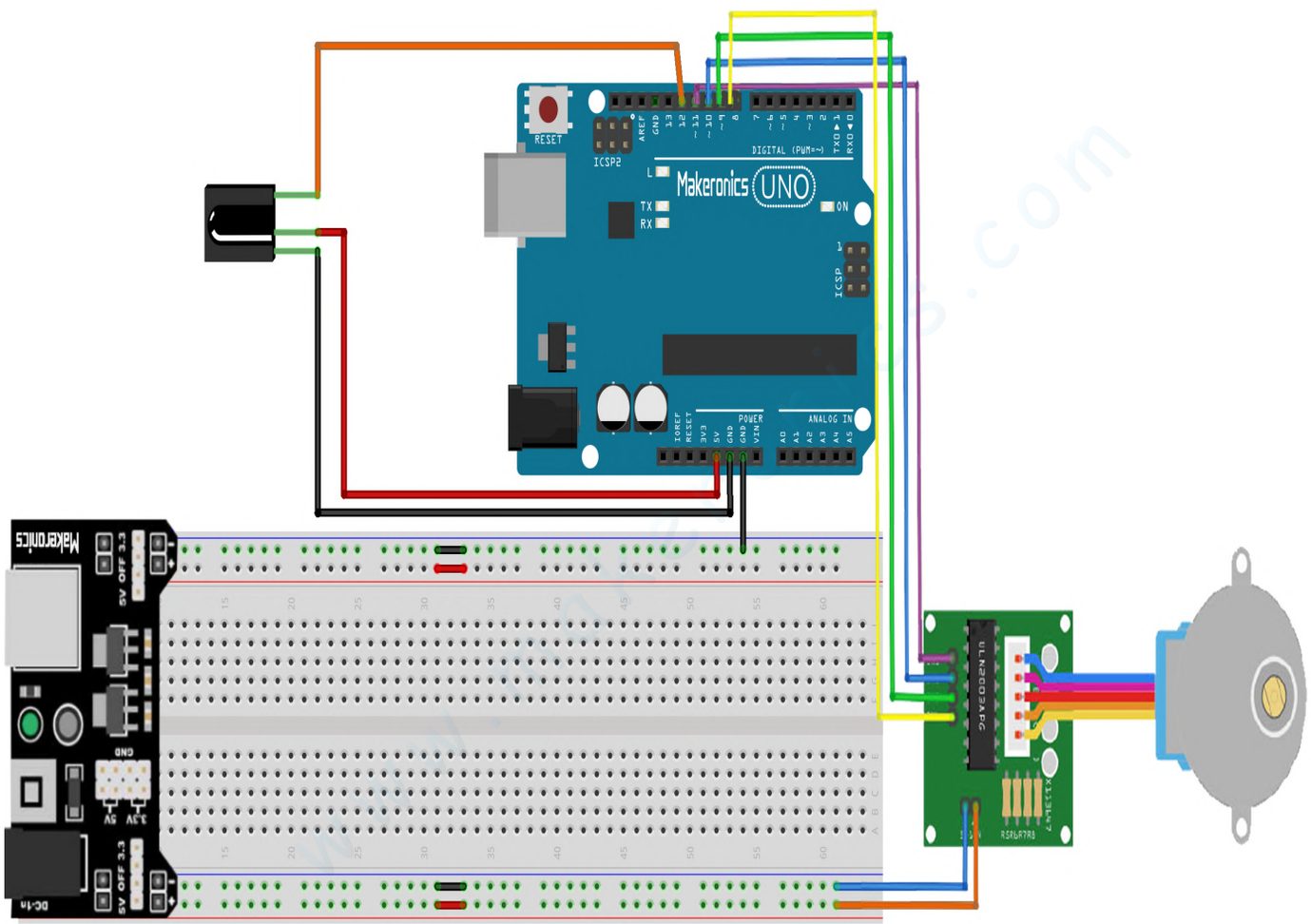
Component Required:

- 1 x Makeronics Uno R3
- 1 x 830 tie-points breadboard
- 1 x IR receiver module
- 1 x IR remote
- 1 x ULN2003 stepper motor driver module
- 1 x Stepper motor
- 1 x Power supply module
- 1 x 9V battery
- 9 x F-M wires (Female to Male DuPont wires)
- 5 x M-M wire (Male to Male jumper wire)

Connection Schematic:



Wiring diagram:



We are using 4 pins to control the Stepper and 1 pin for the IR sensor.

Pins 8-11 are controlling the Stepper motor and pin 12 is receiving the IR information.
We connect the 5V and Ground from the UNO to the sensor.
As a precaution, use a breadboard power supply to power the stepper motor since it can use more power and we don't want to damage the power supply of the UNO.

Code

After wiring, please open program in the code folder- Lesson 29 Controlling Stepper Motor With Remote and click UPLOAD to upload the program. See Lesson 3 for details about program uploading if there are any errors.
Before you can run this, make sure that you have installed the < IRremote >,< Stepper >library or re-install it, if necessary. Otherwise, your code won't work.
For details about loading the library file, see Lesson 2.
The code only recognize 2 values from the IR Remote control: VOL+ and VOL-. When VOL+ is pressed on the remote the motor will make a full rotation clockwise. VOL- will make a full rotation counter-clockwise.

Demo: