BASIC STARTER KIT

BEGINNERS GUIDE

QUAD STORE
WWW.QUADSTORE.IN

Preface

Quad Store is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

This is an entry-level learning kit for Arduino. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of Arduino, and be able to make fascinating works based on Arduino.

Contents

Getting Started with Arduino

What is an Arduino?

Arduino is an open-source physical computing platform designed to make experimenting with electronics more fun and intuitive. Arduino has its own unique, simplified programming language, a vast support network, and thousands of potential uses, making it the perfect platform for both beginner and advanced DIY enthusiasts.
www.arduino.cc

A Computer for the Physical World:

The friendly blue board in your hand (or on your desk) is the Arduino. In some ways you could think of Arduino as the child of traditional desktop and laptop computers. At its roots, the Arduino is essentially a small portable computer. It is capable of taking inputs (such as the push of a button or a reading from a light sensor) and interpreting that information to control various outputs (like a blinking LED light or an electric motor).
That's where the term "physical computing" is born - an Arduino is capable of taking the world of electronics and relating it to the physical world in a real and tangible way. Trust us - this will all make more sense soon.

Arduino UNO SMD R3

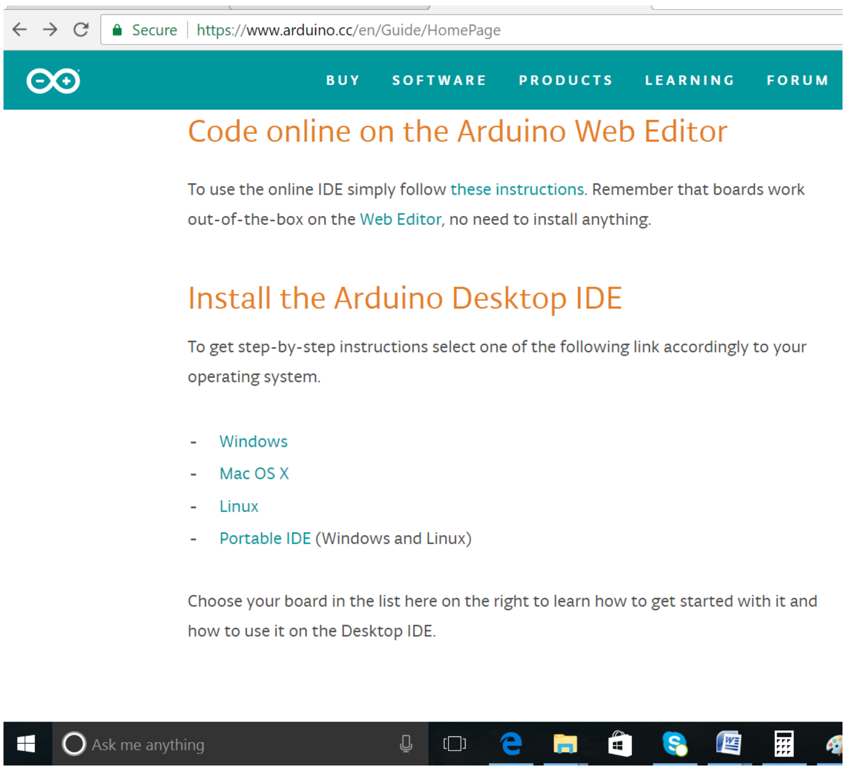
The Arduino Uno is one of several development boards based on the ATmega328. We like it mainly because of its extensive support network and its versatility. It has 14 digital input/output pins (6 of which can be PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. Don't worry, you'll learn about all these later.

Installing Arduino IDE and Using Uno R3 board

STEP-1: Download the Arduino IDE (Integrated Development Environment)

Access the Internet:
In order to get your Arduino up and running, you'll need to download some software first from www.arduino.cc (it's free!). This software, known as the Arduino IDE, will allow you to program the Arduino to do exactly what you want. It's like a word processor for writing programs. With an internet-capable computer, open up your favorite browser and type in the following URL into the address bar:

www.arduino.cc/en/Main/Software



Code online on the Arduino Web Editor

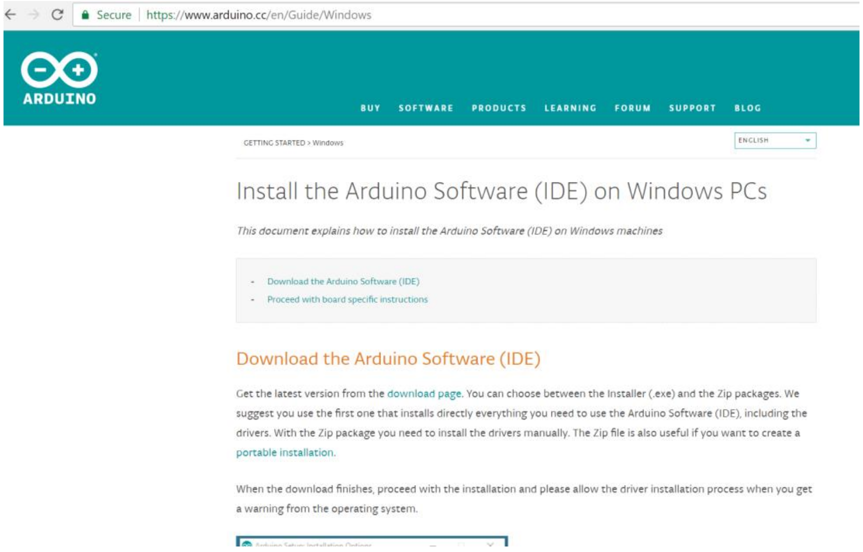
To use the online IDE simply follow these instructions. Remember that boards work out-of-the-box on the Web Editor, no need to install anything.

Install the Arduino Desktop IDE

To get step-by-step instructions select one of the following link accordingly to your operating system.

- Windows
- Mac OS X
- Linux
- Portable IDE (Windows and Linux)

Choose your board in the list here on the right to learn how to get started with it and how to use it on the Desktop IDE.


Secure | https://www.arduino.cc/en/Guide/Windows
ARDUINO
BUY
SOFTWARE
PRODUCTS
LEARNING
FORUM
SUPPORT
BLOG
GETTING STARTED > Windows
ENGLISH

Install the Arduino Software (IDE) on Windows PCs

This document explains how to install the Arduino Software (IDE) on Windows machines

- Download the Arduino Software (IDE)
- Proceed with board specific instructions

Download the Arduino Software (IDE)

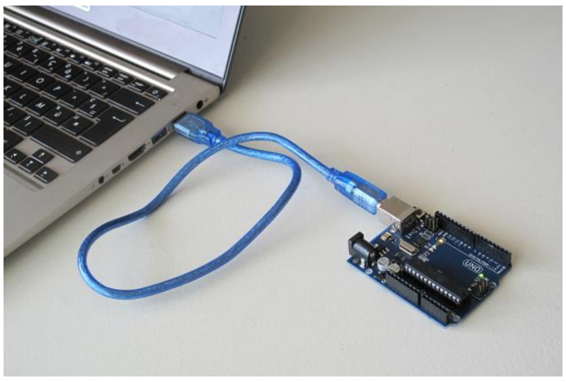
Get the latest version from the download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.

For different operating system platforms, the way of using Arduino IDE is different. Please refer to the following links: Windows User：http://www.arduino.cc/en/Guide/Windows Mac OS X User：http://www.arduino.cc/en/Guide/MacOSX Linux User：http://playground.arduino.cc/Learning/Linux For more detailed information about Arduino IDE, please refer to the following link: http://www.arduino.cc/en/Guide/HomePage

STEP-2: Connect your Arduino Uno to your Computer:

Use the USB cable provided in the kit to connect the Arduino to one of your computer's USB inputs.



STEP-3: Install Drivers

Depending on your computer's operating system, you will need to follow specific instructions. Please go to the URLs below for specific instructions on how to install the drivers onto your Arduino Uno.

Windows Installation Process:
Go to the web address below to access the instructions for installations on a Windows-based computer.

http://arduino.cc/en/Guide/Windows

Macintosh OS X Installation Process:
Macs do not require you to install drivers. Enter the following URL if you have questions. Otherwise proceed to next page.

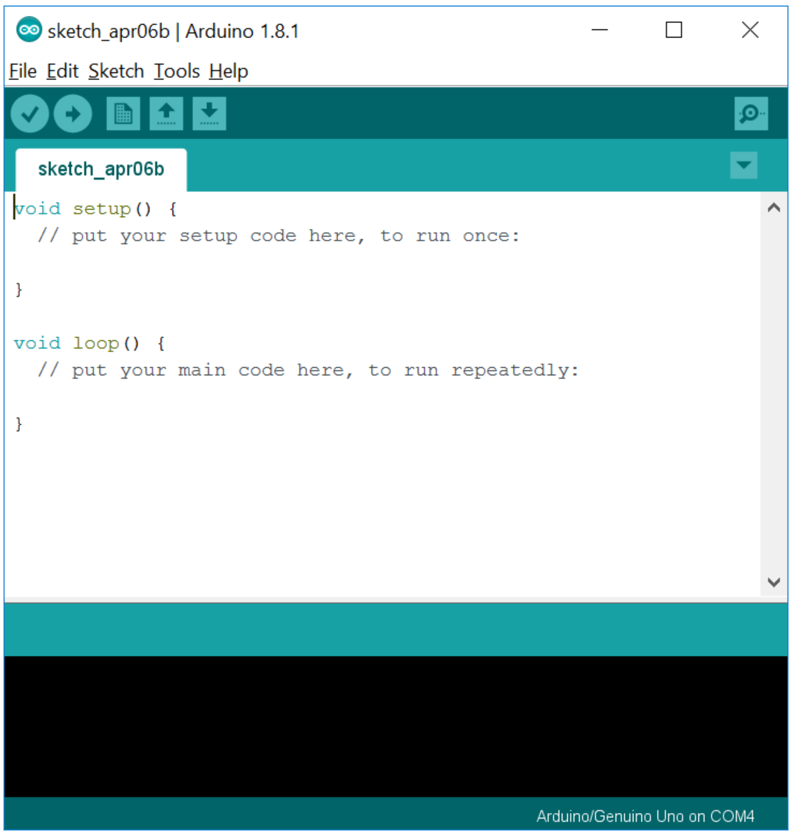
http://arduino.cc/en/Guide/MacOSX

Linux:
32 bit / 64 bit, Installation Process Go to the web address below to access the instructions for installations on a Linux-based computer.

http://www.arduino.cc/playground/Learning/Linux

STEP-4: Open the Arduino IDE

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away, this is just an introduction. The step is to set your IDE to identify your Arduino Uno.



```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

Arduino/Genuino Uno on COM4

GUI (Graphical User Interface)

Verify
Checks your code for errors compiling it.

Upload
Compiles your code and uploads it to the configured board. See [uploading](#) below for details.
Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

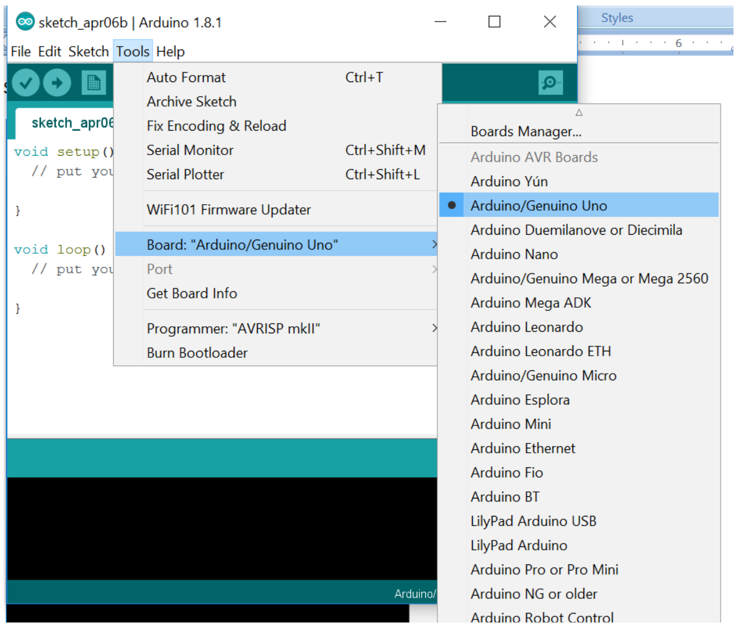
New
Creates a new sketch.

Open
Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.
Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File | Sketchbookmenu instead.

Save
Saves your sketch.

Serial Monitor
Opens the [serial monitor](#).

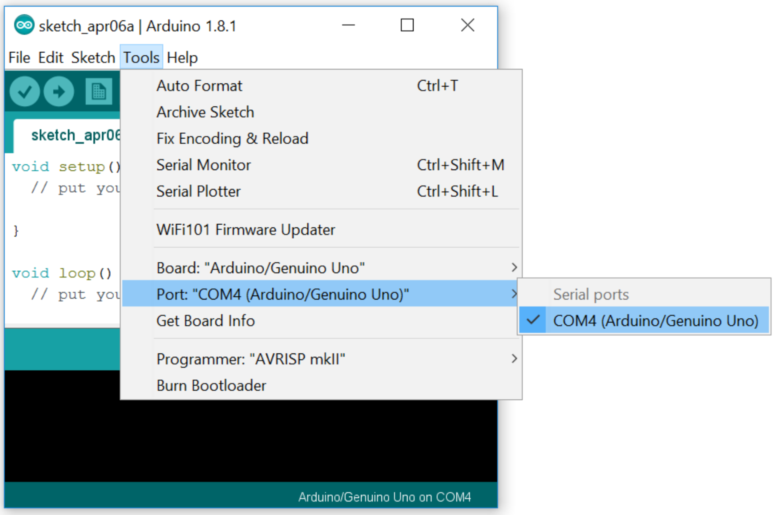
STEP-5: Select your board: Arduino Uno



```
void setup() {
  // put you

}

void loop() {
  // put you

}
```

STEP-6: Select your Serial Device

Windows: Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be com3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.


Board: "Arduino/Genuino Uno"
Port: "COM4 (Arduino/Genuino Uno)"
✓ COM4 (Arduino/Genuino Uno)
Programmer: "AVRISP mkII"
Arduino/Genuino Uno on COM4

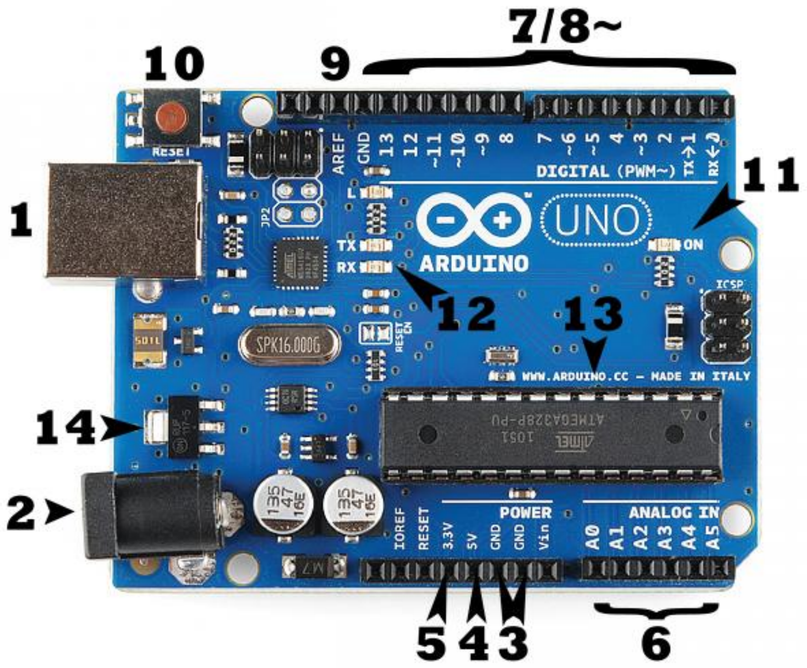
Mac OS: Select the serial device of the Arduino board from the Tools > Serial Port menu. On the Mac, this should be something with /dev/tty.usbmodem (for the Uno or Mega 2560) or /dev/tty.usbserial (for older boards) in it.

Linux: http://playground.arduino.cc/Learning/Linux

About Arduino Uno R3 board

What's on the board?

There are many varieties of Arduino boards that can be used for different purposes. Some boards look a bit different from the one below, but most Arduino have the majority of these components in common:



Power (USB / Barrel Jack)

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply that is terminated in a barrel jack. In the picture above the USB connection is labeled **(1)** and the barrel jack is labeled **(2).**

.

NOTE: Do NOT use a power supply greater than 20 Volts as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.
Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjuction with a breadboard and some wire. They usually have black plastic 'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- **GND** **(3):** Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- **5V** **(4)** **& 3.3V** **(5):** As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
- **Analog** **(6):** The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- **Digital** **(7):** Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
- **PWM** **(8):** You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM). We have a tutorial on PWM, but for now, think of these pins as being able to simulate analog output (like fading an LED in and out).
- **AREF** **(9):** Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset Button

Just like the original Nintendo, the Arduino has a reset button **(10).** Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code doesn't repeat, but you want to test it multiple times. Unlike the original Nintendo however, blowing on the Arduino doesn't usually fix any problems.

Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there's a tiny LED next to the word 'ON' **(11).** This LED should light up whenever you plug your Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong. Time to re-check your circuit!

TX RX LEDs

TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear – once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs **(12).** These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we're loading a new program onto the board).

Main IC

The black thing with all the metal legs is an IC, or Integrated Circuit **(13).** Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC. If you want to know more about the difference between various IC's, reading the datasheets is often a good idea.

Voltage Regulator

The voltage regulator **(14)** is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says – it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don't hook up your Arduino to anything greater than 20 volts.

Lesson 1 - Blinking LED

Overview:

In this tutorial, we will start the journey of learning Arduino UNO. To begin, let's learn how to make an LED blink.

Components:

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * 220Ω Resistor
- 1 * LED
- 1 * Breadboard
- 2 * Jumper Wires

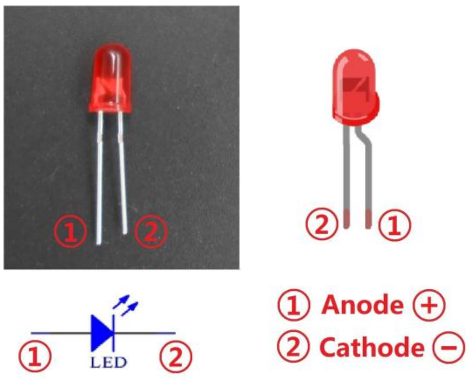
Principle:

In this lesson, we will program the Arduino's GPIO output high level (+5V) and low level (0V), and then make the LED which is connected to the Arduino's GPIO flicker with a certain frequency.

1. What is the LED?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphide semiconductor materials. The LED has two electrodes: a positive electrode and a negative one. It lights up only when a forward current passes, and it can flash red, blue, green, yellow, etc. The color of the light depends on the material it is made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.



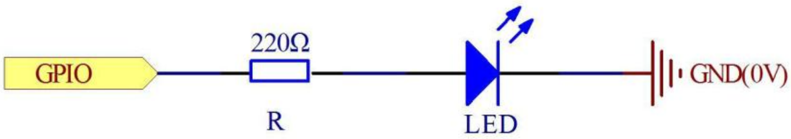
2. What is resistor?

The main function of the resistor is to limit currents. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm(Ω).

A band resistor is used in this experiment. It is a resistor with a surface coated with some particular color through which the resistance can be identified directly.

There are two methods for connecting LED to pins of an Arduino board:

①



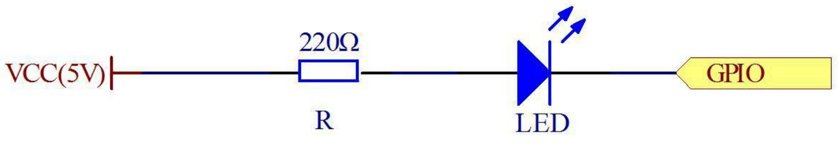
As shown in the schematic diagram, the anode of the LED is connected to Arduino's GPIO via a resistor, and the cathode to the ground (GND). When the GPIO outputs high level, the LED is on; when it outputs low, the LED is off.

The resistance of a current-limiting resistor is calculated as follows: 5~20mA current is required to make an LED on, and the output voltage of the Arduino UNO's GPIO is 5V, so we can get the resistance:

$$R = U / I = 5V / (5\sim20mA) = 250\Omega\sim1k\Omega$$

Since an LED is a resistor itself, here we use a 220ohm resistor.

②



As shown in the schematic diagram above, the anode of LED is connected to VCC(+5V), and the cathode of LED is connected to the Arduino's GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

The experiment is made based on method ① – use pin D8 of the Arduino board to control an LED. When D8 is programmed to output high level, the LED will be turned on. Next, delay for some time. Then D8 is programmed to output low level to turn the LED off. Repeat the above process and you can get a blinking LED then.

3. Key functions:

●setup()

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

●loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

●pinMode()

Configures the specified pin to behave either as an input or an output.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

●digitalWrite()

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

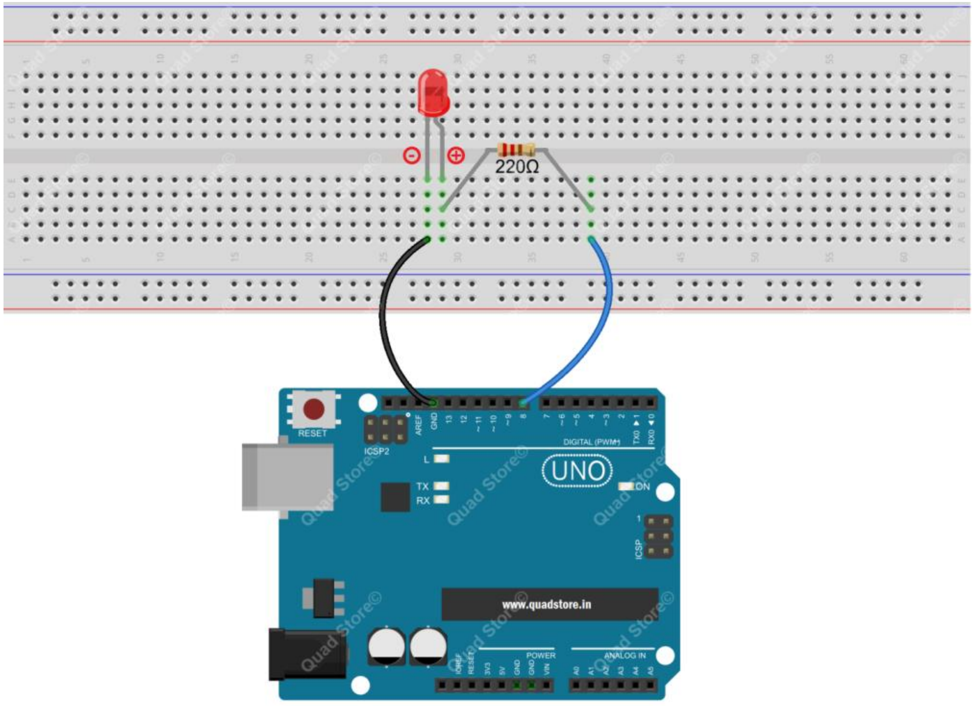
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

●delay()

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Procedure:

Step 1: Build the circuit as below:



Step 2: Program : You can copy paste the below program in the IDE or open the code directly from the “CODE” folder that comes with the DVD/from the downloaded Zip folder.

```
/*************************************************************
File name:   01_blinkingLed.ino
Description:  Lit LED, let LED blinks.
Website: www.quadstore.in
*************************************************************/
int ledPin=8; //definition digital 8 pins as pin to control the LED
void setup()
{
   pinMode(ledPin,OUTPUT);    //Set the digital 8 port mode, OUTPUT: Output mode
}
void loop()
{
   digitalWrite(ledPin,HIGH); //HIGH is set to about 5V PIN8
   delay(1000);              //Set the delay time, 1000 = 1S
   digitalWrite(ledPin,LOW);  //LOW is set to about 5V PIN8
   delay(1000);              //Set the delay time, 1000 = 1S
}
```

Step 3: Compile the program and upload to Arduino UNO board. Now you can see the LED blinking

Lesson 2 – Active/Passive Buzzer

Overview

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

Component Required:

(1) x Quaduino Uno R3

(1) x Active buzzer

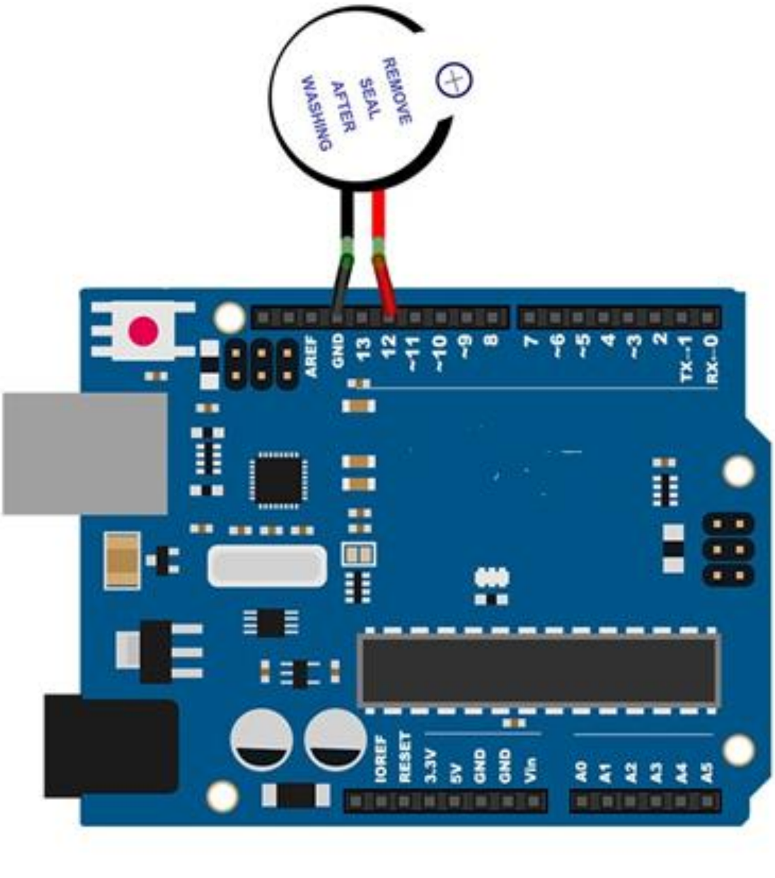
(2) x F-M wires (Female to Male DuPont wires)

Component Introduction

BUZZER:

Electronic buzzers are DC-powered and equipped with an integrated circuit. They are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products for voice devices. 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.

Wiring diagram



‘

Code

After wiring, please open the program in the code folder- Lesson 6 Making Sounds and click UPLOAD to upload the program. See Lesson 2 for details about program uploading if there are any errors.

Lesson 3 - Controlling an LED by a Button

Overview

In this lesson, we will learn how to detect the state of a button, and then toggle the state of the LED based on the state of the button.

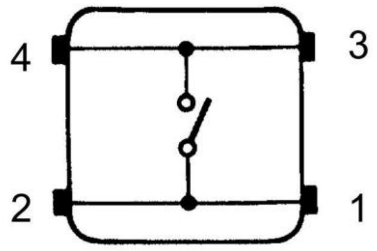
Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Button
- 1 * LED
- 1 * 10kΩ Resistor
- 1 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

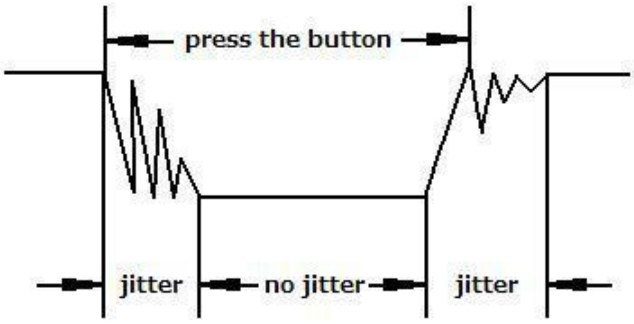
1. Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown below.

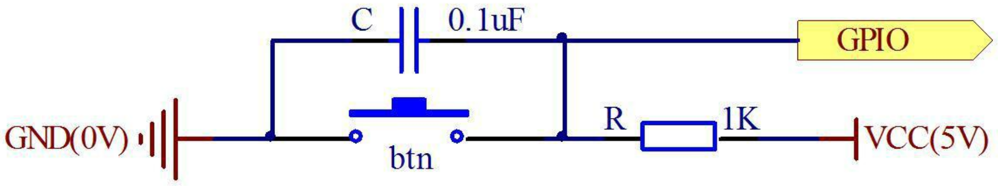


The button we used is a normally open type one. The two contacts of a button are in the off state under the normal conditions; only when the button is pressed they are closed.

The button jitter must happen in the process of using. The jitter waveform is as the flowing picture:



Each time you press the button, the Arduino will regard you have pressed the button many times due to the jitter of the button. You should deal with the jitter of buttons before using. You can eliminate the jitter through software programming. Besides, you can use a capacitor to solve the issue. Take the software method for example. First, detect whether the level of button interface is low level or high level. If it is low level, 5~10ms delay is needed. Then detect whether the level of button interface is low or high. If the signal is low, you can infer that the button is pressed once. You can also use a 0.1uF capacitor to avoid the jitter of buttons. The schematic diagram is as shown below:



2. Interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem

. Key functions:

●attachInterrupt(interrupt, ISR, mode)

Specifies a named Interrupt Service Routine (ISR) to call when an interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).

Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be ignored (turned off) until the current one is finished. as delay() and millis() both rely on interrupts, they will not work while an ISR is running. delayMicroseconds(), which does not rely on interrupts, will work as expected.

Syntax:

`attachInterrupt(pin, ISR, mode)` Parameters:
pin: the pin number

ISR: the ISR will be called when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

-LOW to trigger the interrupt whenever the pin is low,

-CHANGE to trigger the interrupt whenever the pin changes value -RISING to trigger
when the pin goes from low to high, -FALLING for when the pin goes from high to low.

●digitalRead()

Reads the value from a specified digital pin, either HIGH or LOW. Syntax:
`digitalRead(pin)`

Parameters:

pin: the number of the digital pin you want to read (int) Returns:

HIGH or LOW ●delayMicroseconds(us)

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax:

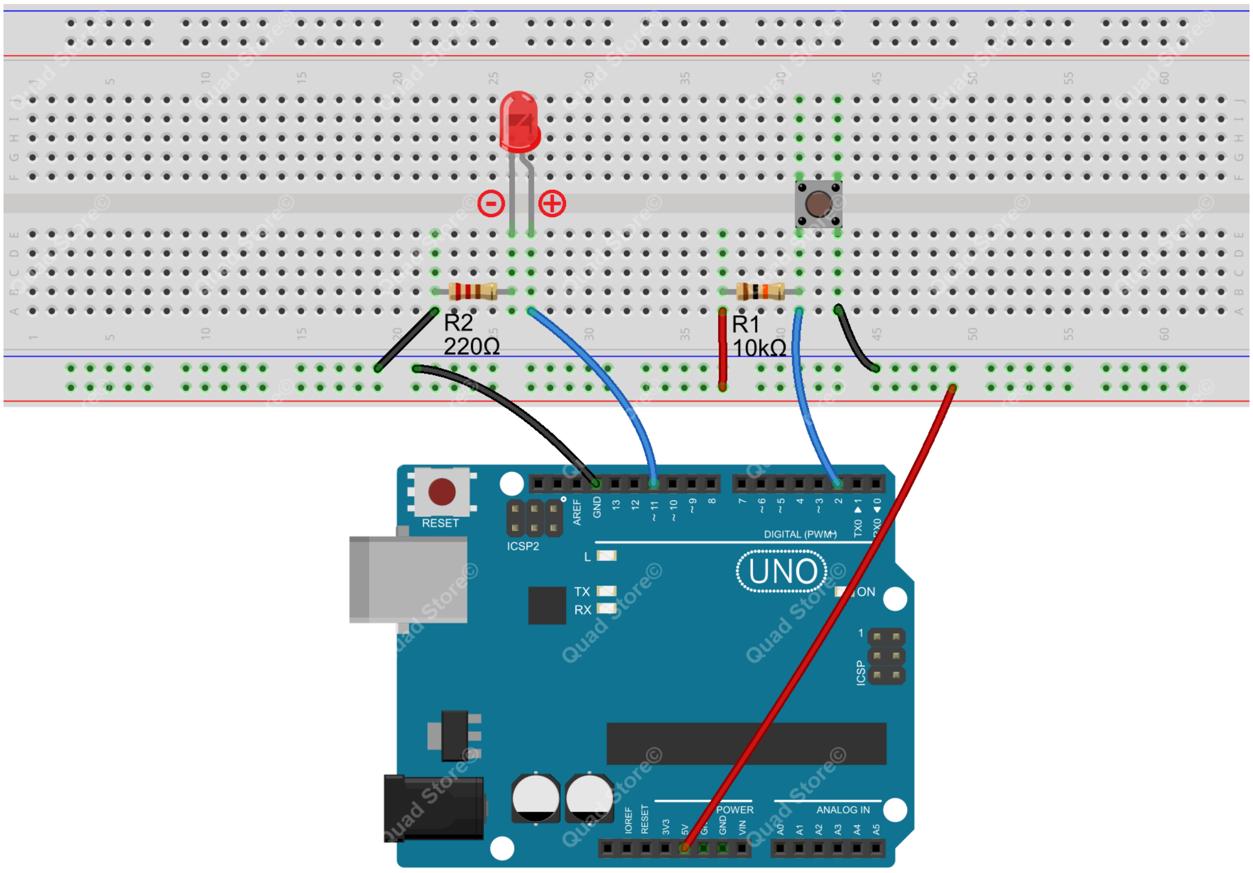
`delayMicroseconds(us)`

Parameters:

us: the number of microseconds to pause (unsigned int) Returns:
None

Procedure:

Step 1: Build the circuit



Step 2: Program: Open /Copy the code from the “CODE” Folder

Step 3: Compile the program and upload to Arduino UNO board

Now press the button, and you can see the state of the LED will be toggled between ON and OFF.

Lesson 4 - LED Flowing Lights

Overview

In the first lesson, we have learned how to make an LED blink by programming the Arduino. Today, we will use the Arduino to control 8 LEDs to make the LEDs show the effect of flowing.

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 8 * LED
- 8 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

The principle of this experiment is very simple and is quite similar with that in the first lesson.

Key function:

●for statements

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

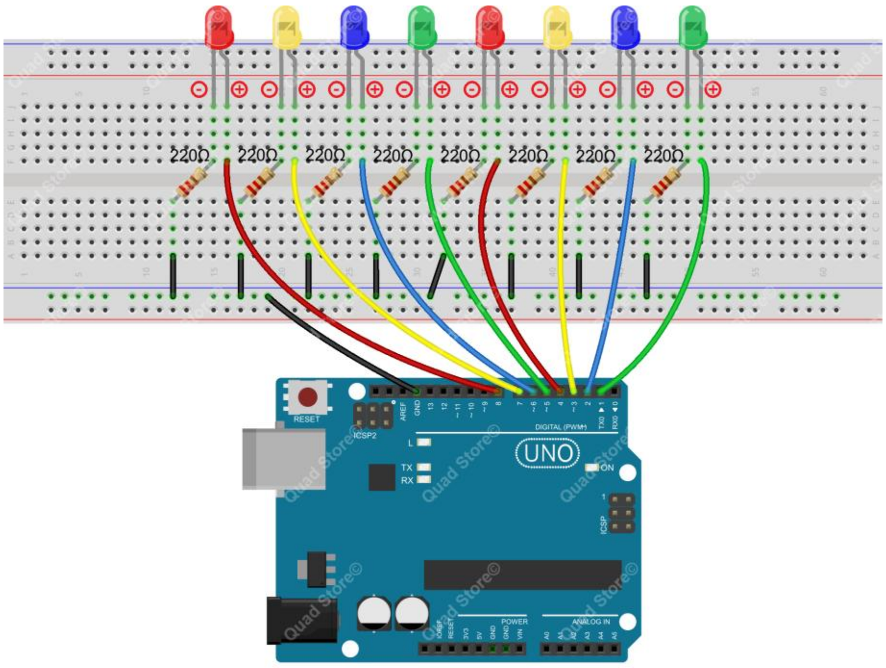
There are three parts to the for loop header:

```
for (initialization; condition; increment) { //statement(s);
}
```

The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino UNO board

Now, you can see 8 LEDs light up in sequence from the green one on the right side to others on the left, and next from the left to the right. The LEDs flash like flowing water repeatedly in a circular way.

Lesson 5 - Breathing LED

Overview

In this lesson, we will learn how to program the Arduino to generate PWM signals. And then we use the PWM square-wave signals to control an LED gradually getting brighter and then slowly dimmer, much like human breath.

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * LED
- 1 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the following figure, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

Key function:

●analogWrite()

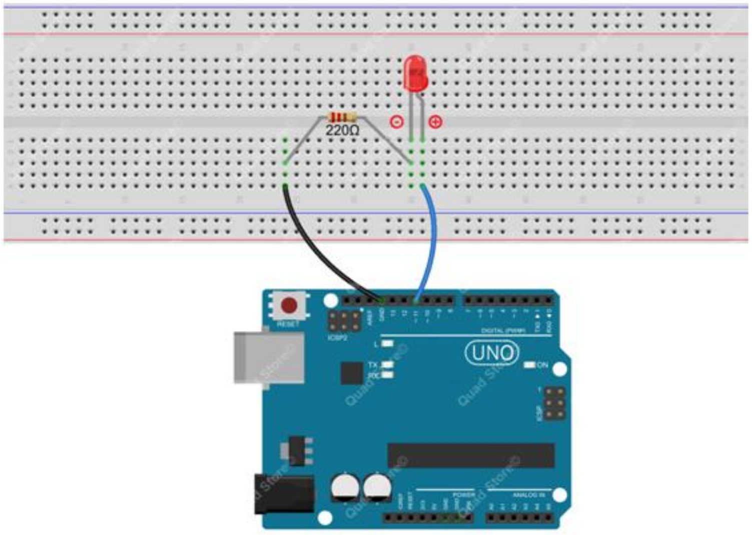
Writes an analog value (PWM wave) to a pin. Can be used to light an LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). You do not need to call pinMode() to set the pin as an output before calling analogWrite().

Syntax: analogWrite(pin, value)
Parameters:
pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on). Returns:
nothing

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino UNO board.

Now, you should see the LED lights up and gets gradually brighter, and then slowly turns dimmer. The process repeats circularly, and with the particular rhythm it looks like animals' breath.

Lesson 6 - Controlling an RGB LED by PWM

Overview

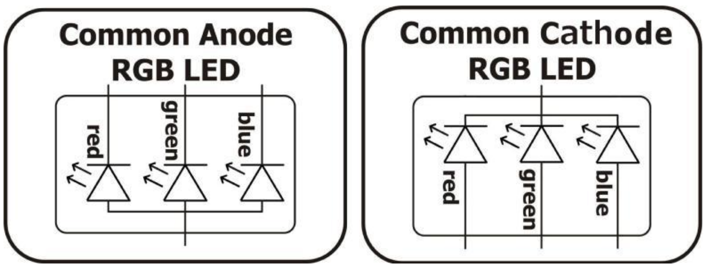
In this lesson, we will program the Arduino for RGB LED control, and make RGB LED emits several of colors of light.

Components

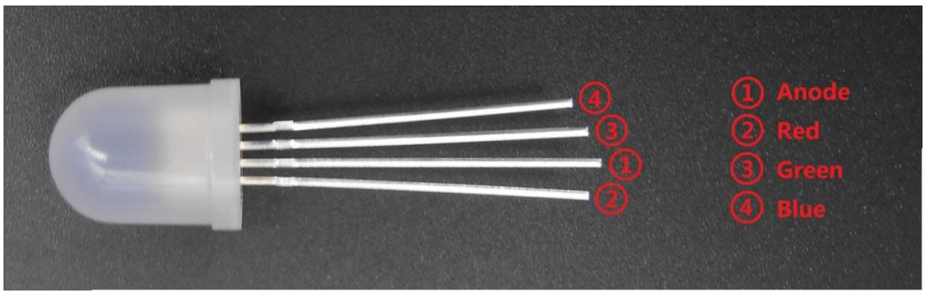
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * RGB LED
- 3* 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

RGB LEDs consist of three LEDs: red, green and blue. These three colored LEDs are capable of producing any color. Tri-color LEDs with red, green, and blue emitters, in general using a four-wire connection with one common lead (anode or cathode).



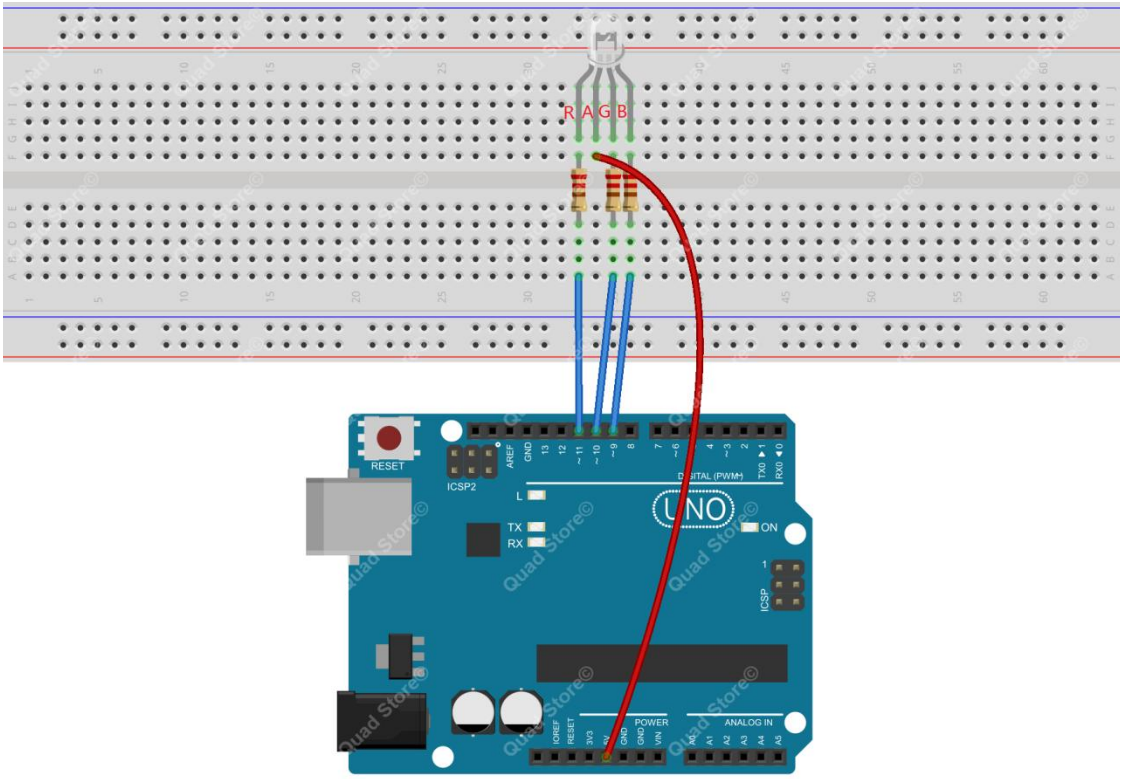
What we use in this experiment is a common anode RGB LED. The longest pin is the common anode of the three LEDs. The pin is connected to the +5V pin of the Arduino, and the rest pins are connected to pin D9, D10, and D11 of the Arduino with a current limiting resistor between.



In this way, we can control the color of an RGB LED by 3-channel PWM signals.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino UNO board

Now, you can see the RGB LED flash red, green, blue, yellow, white and purple light, and then go out. Each state lasts for 1s each time, and the LED flashes colors repeatedly in such sequence.

Lesson 7 - Tilt Switch

Overview

In this lesson, we will learn how to use the tilt switch and change the state of an LED by changing the angle of tilt switch.

Components

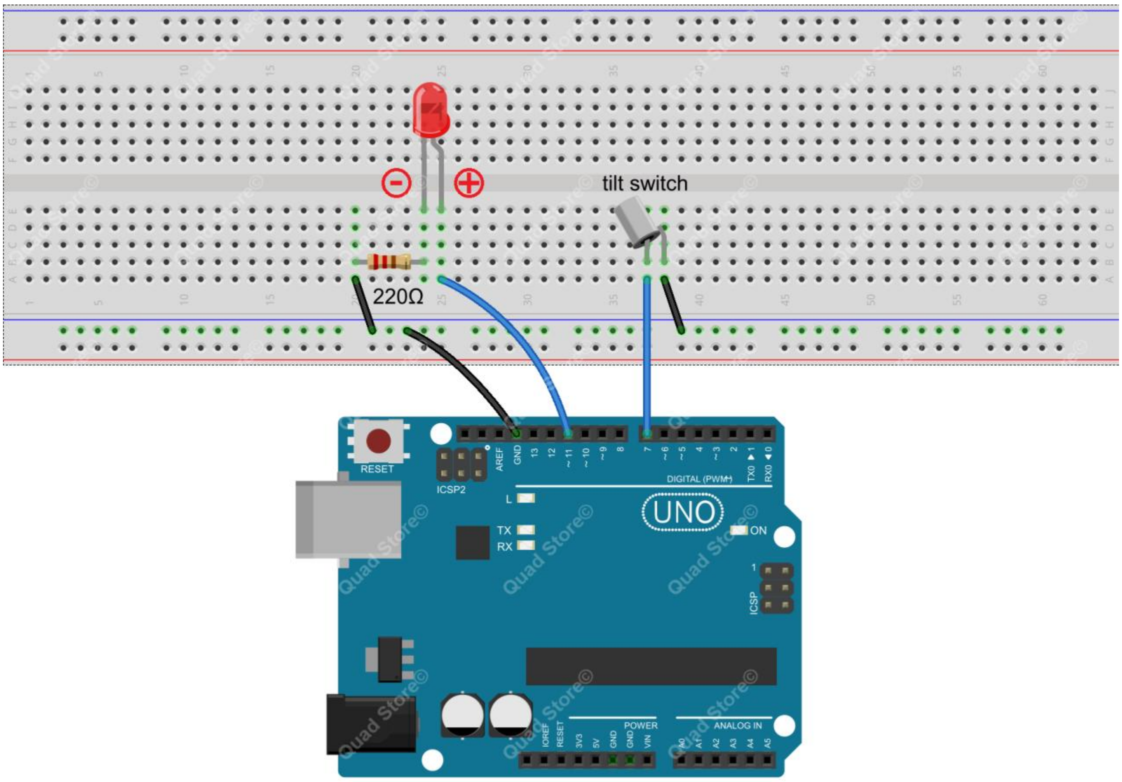
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Tilt Switch (SW-520D)
- 1 * LED
- 1 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

The tilt switch is also called the ball switch. When the switch is tilted in the appropriate direction, the contacts will be connected, tilting the switch the opposite direction causes the metallic ball to move away from that set of contacts, thus breaking that circuit.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino UNO board

Now, when you lean the breadboard at a certain angle, you will see the state of LED is changed.

Lesson 8 – Photoresistor

Overview

In this lesson, you will learn how to measure light intensity using an Analog Input. You will build on lesson 26 and use the level of light to control the number of LEDs to be lit.

The photocell is at the bottom of the breadboard, where the pot was above.

Component Required:

(1) x Quaduino Uno R3
(1) x Breadboard
(8) x leds
(8) x 220 ohm resistors
(1) x 1k ohm resistor
(1) x 74hc595 IC
(1) x Photoresistor (Photocell)
(16) x M-M wires (Male to Male jumper wires)

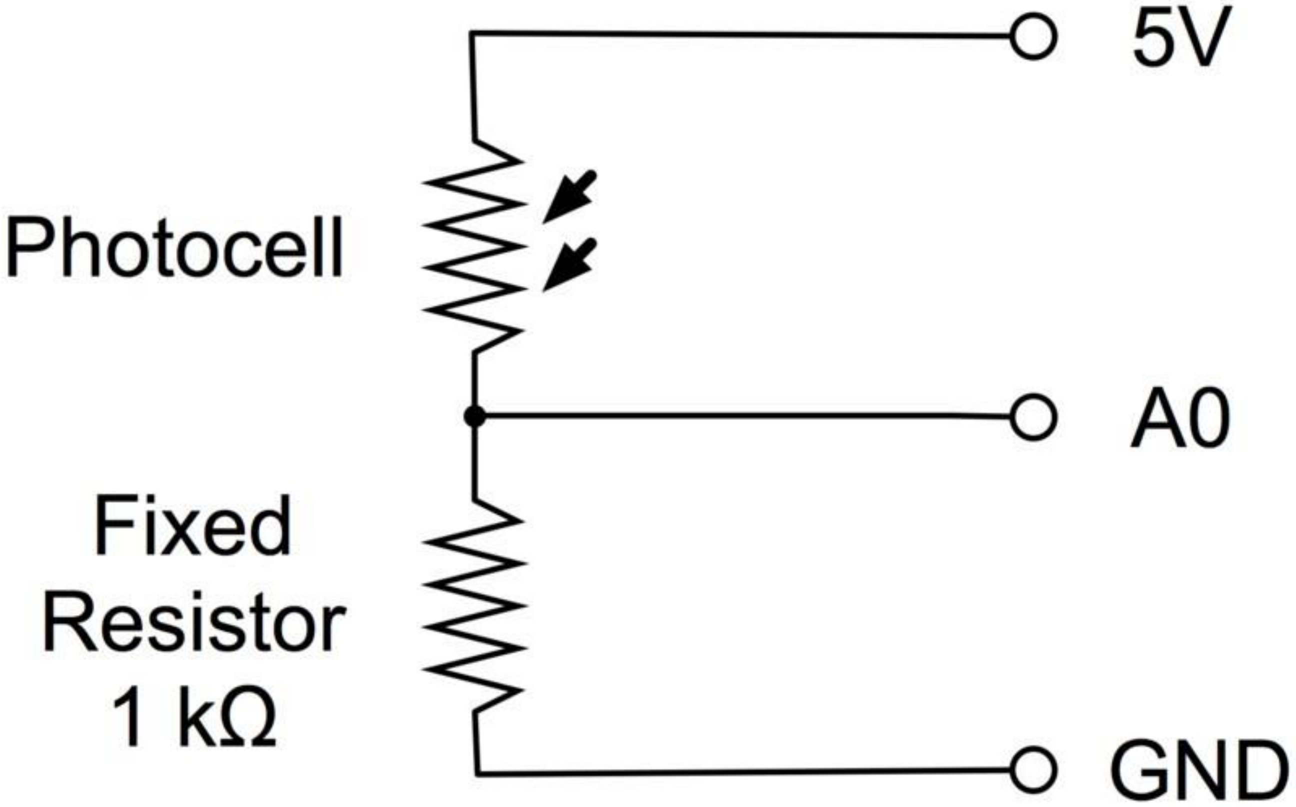
Component Introduction

PHOTOCELL:

The photocell used is of a type called a light dependent resistor, sometimes called an LDR. As the name suggests, these components act just like a resistor, except that the resistance changes in response to how much light is falling on them.

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.

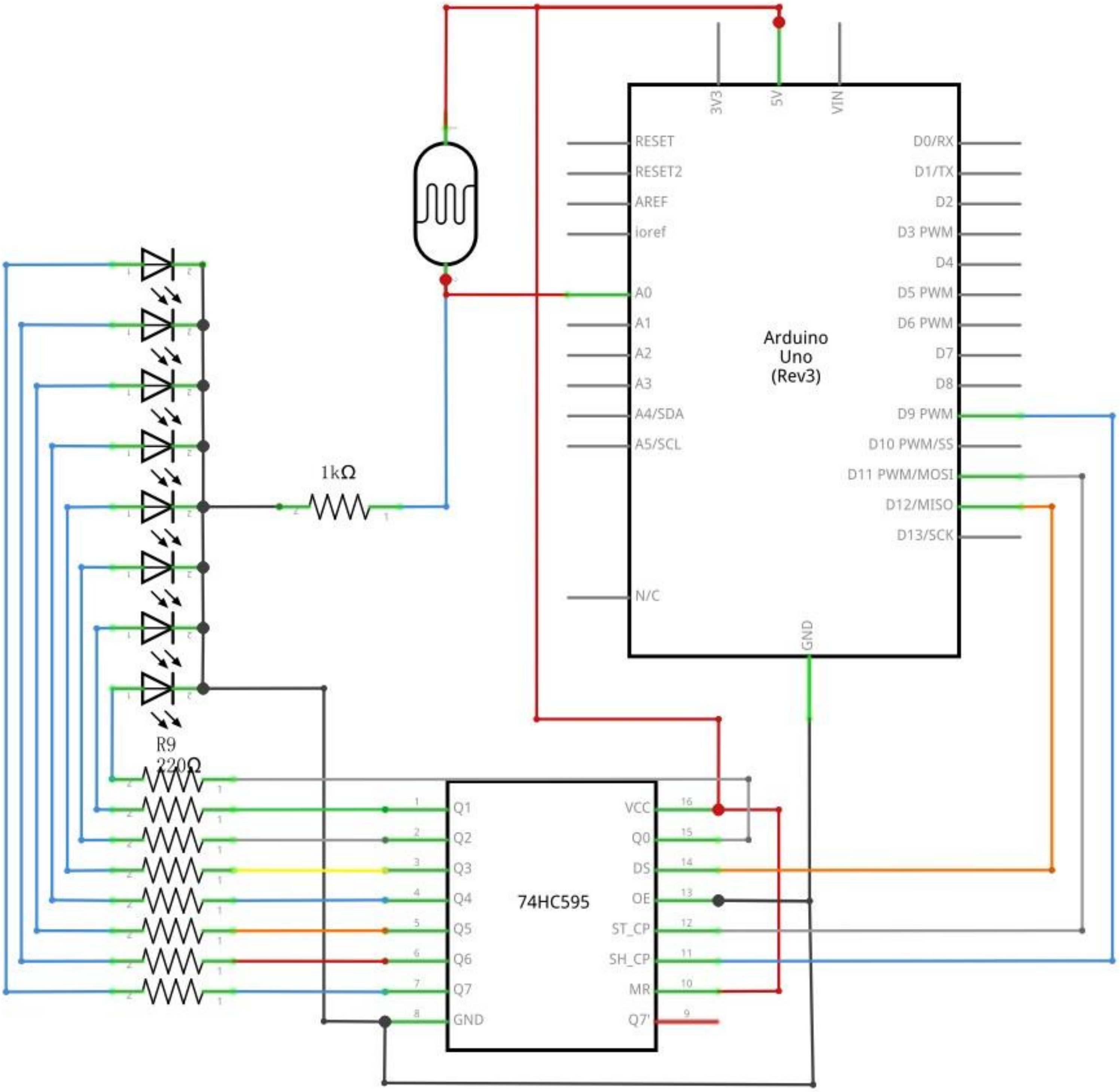


The resistor and photocell together behave like a pot. When the light is very bright, then the resistance of the photocell is very low compared with the fixed value resistor, and so it is as if the pot were turned to maximum.

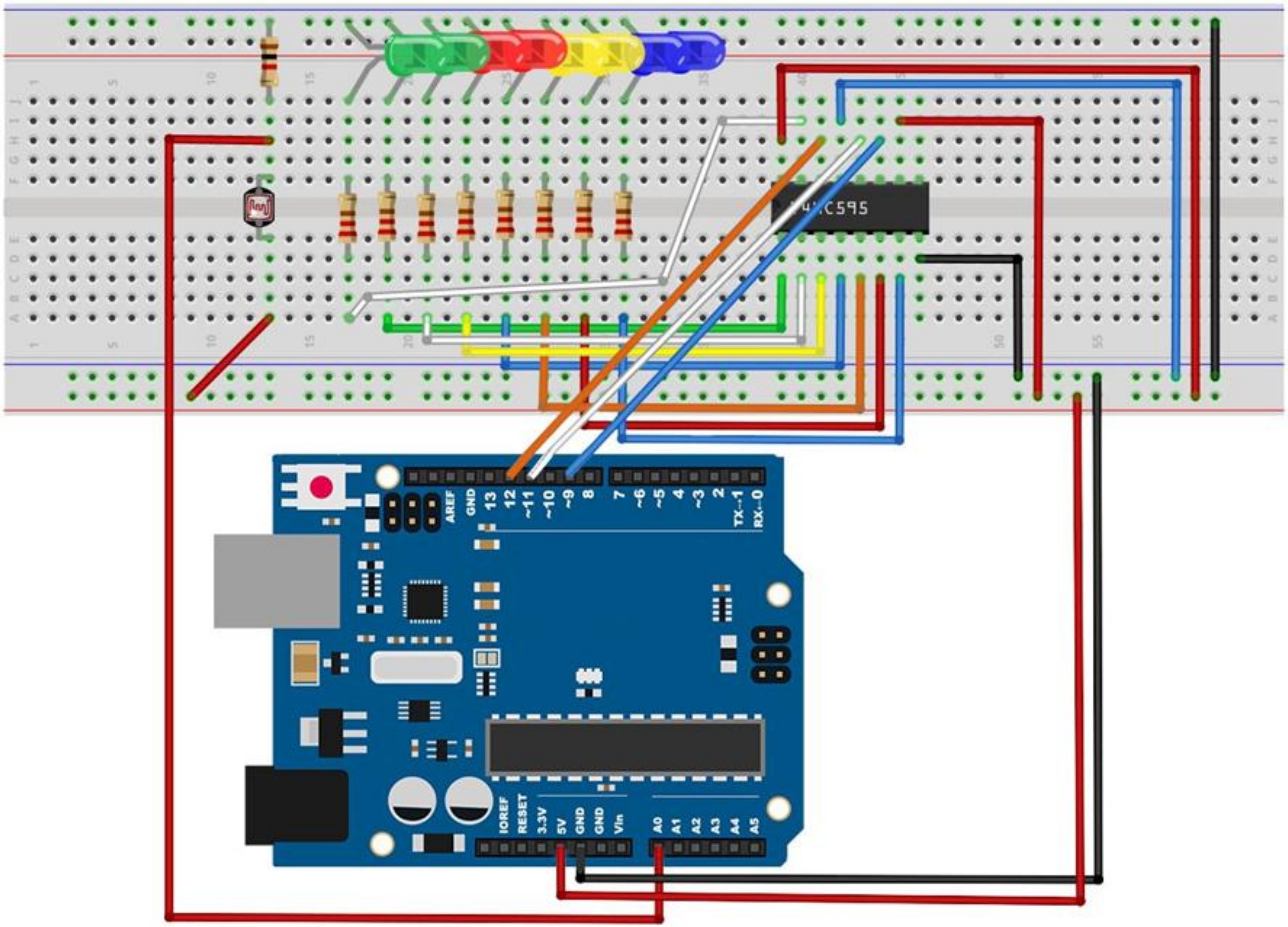
When the photocell is in dull light, the resistance becomes greater than the fixed 1 kΩ resistor and it is as if the pot were being turned towards GND.

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

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

The first thing to note is that we have changed the name of the analog pin to be 'lightPin' rather than 'potPin' since we no longer have a pot connected.

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.