Introduction to Arduino Platform

I hear and I forget. I see and I remember. I do and I understand.

-Confucius

ARDUINO

WHAT IS ARDUINO?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

WHY ARDUINO?

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get

started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version

of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

Concept of Arduino

- The root of Arduino goes deep down to the development of Processing Language by MIT researchers. Processing language is an open source language designed to introduce the software development environment for the artistic people without the need of deep knowledge of programming of algorithms. Processing is based on java.
- In early year of 21st century, designing an electronics gadget was nearly impossible for a common man. The requirement of specific skill set and hefty prices of software and hardware created a full stop in the path of their creativity.
- In year 2003 Hernando Barragan, a programmer developed an open source electronics development platform with software IDE, where anyone with a small knowledge in electronics and programming could use his project to give wings to their creativity. His focus was to reduce the burden of complexity in designing electronics hardware and software. The project was named as Wiring. The software IDE of the Wiring used processing language to write the codes.
- As the program written in C\C++ is named as Project, in the same way the code written in Wiring (even in Processing and Arduino) is termed as Sketch. The name sketch gives a familiar look for an artist.
- The principle idea behind Wiring is that one can make the sketch of their idea on Wiring software and implement it using specially designed Wiring board. You need to write a few lines of codes on the software IDE and then download the program to the onboard microcontroller to see the output.
- Wiring has predefined libraries to make the programming language easy. Arduino uses these libraries. The predefined libraries are written in C and C++. One can even write his software in C\C++ and use them on Wiring boards. The difference between writing a program in C/C++ and Wiring is that the Wiring Application Programmable Interface (API) has simplified programming style and the user doesn't require detailed knowledge of the

concepts like classes, objects, pointers, etc. While sketching hardware you need to call the predefined functions and rest will be handled by the Wiring software.

- The basic difference between the Processing and the Wiring is that the Processing is use to write the program which can be used on other computers while Wiring program is used on microcontrollers.

What is the Arduino IDE?

Arduino provides an open-source and easy-to-use programming tool for writing code and uploading it to your board. It is often referred to as the Arduino IDE (Integrated Development Environment). The **Arduino Software (IDE) is easy-to-use for beginners**, yet flexible enough for advanced users.

The Arduino Software (IDE) is cross-platform, it runs on Windows, Mac OSX, and Linux operating systems.

You can tell your board what to do by writing code and uploading it to the microcontroller on it using the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

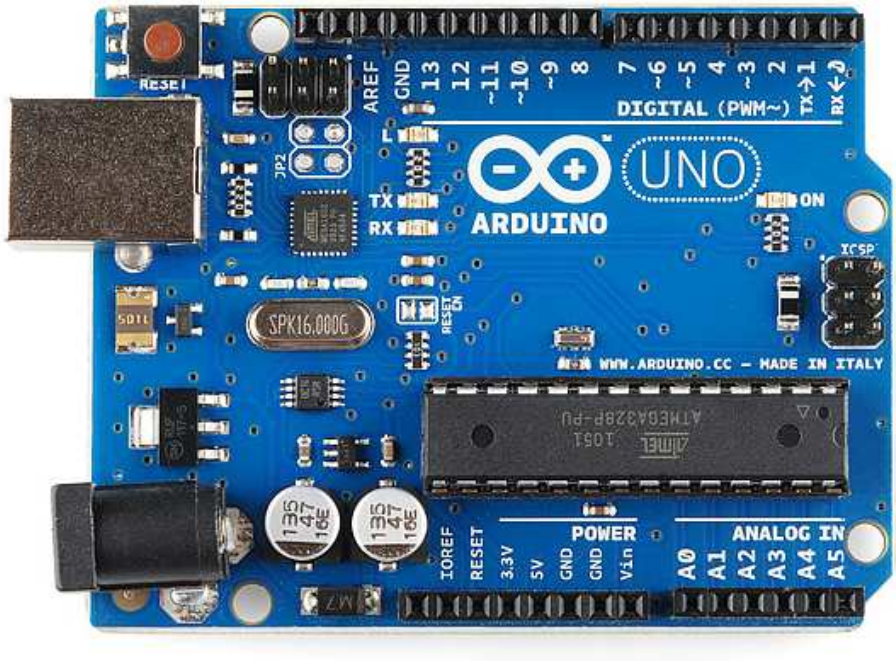
The Arduino Family

Arduino makes several different boards, each with different capabilities. In addition, part of being open source hardware means that others can modify and produce derivatives of Arduino boards that provide even more form factors and functionality. If you're not sure which one is right for your project, check this guide for some helpful hints. Here are a few options that are well-suited to someone new to the world of Arduino.

Arduino Uno (R3)

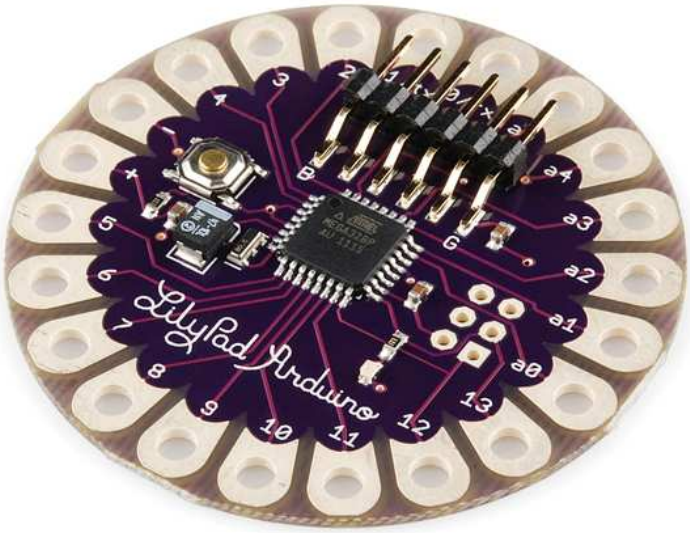
The Uno is a great choice for your first Arduino. It's got everything you need to get started, and nothing you don't. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a USB connection, a power jack, a reset button and more. It contains everything

needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



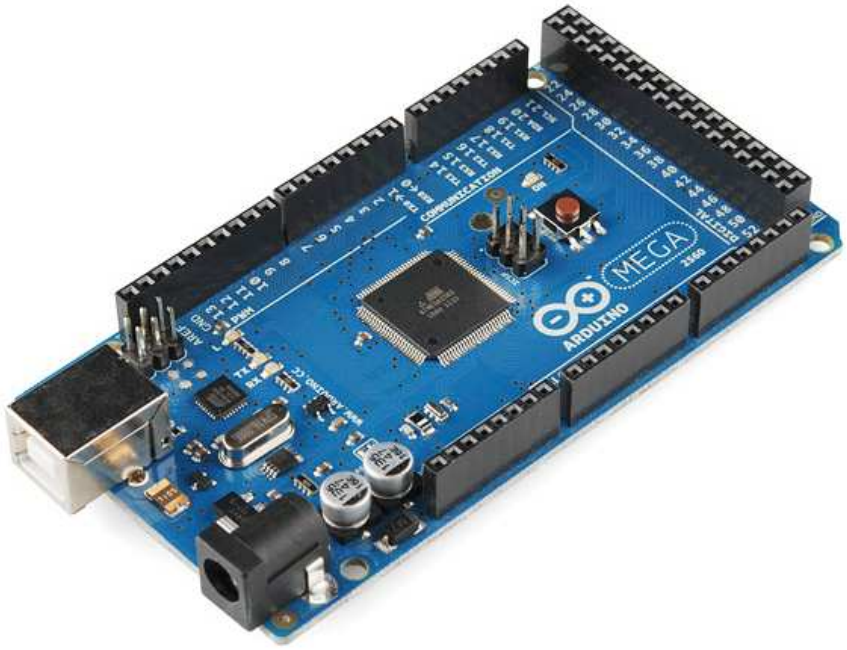
LilyPad Arduino

This is LilyPad Arduino main board! LilyPad is a wearable e-textile technology developed by Leah Buechley and cooperatively designed by Leah and SparkFun. Each LilyPad was creatively designed with large connecting pads and a flat back to allow them to be sewn into clothing with conductive thread. The LilyPad also has its own family of input, output, power, and sensor boards that are also built specifically for e-textiles. They're even washable!



Arduino Mega (R3)

The Arduino Mega is like the UNO's big brother. It has lots (*54!*) of digital input/output pins (14 can be used as PWM outputs), 16 analog inputs, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The large number of pins make this board very handy for projects that require a bunch of digital inputs or outputs (like lots of LEDs or buttons).

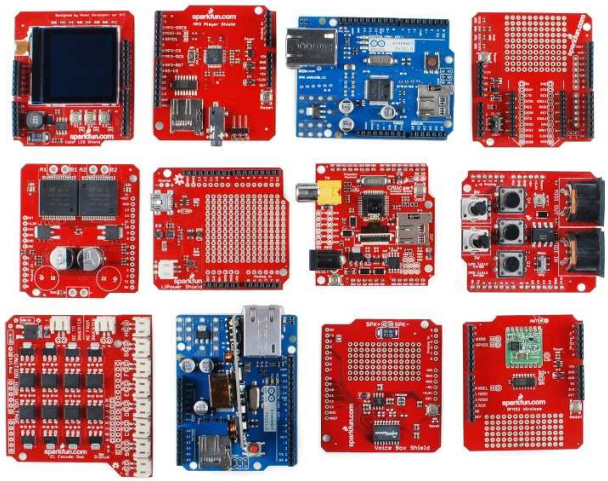


Arduino Leonardo

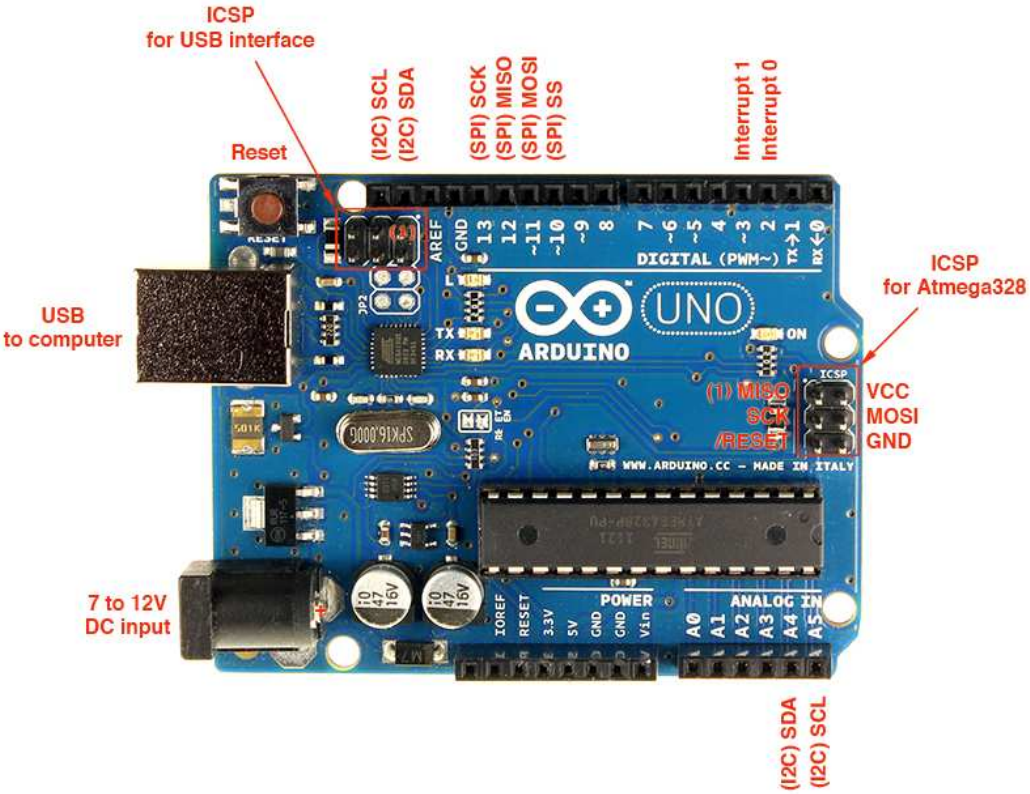
The Leonardo is Arduino's first development board to use one microcontroller with built-in USB. This means that it can be cheaper and simpler. Also, because the board is handling USB directly, code libraries are available which allow the board to emulate a computer keyboard, mouse, and more!

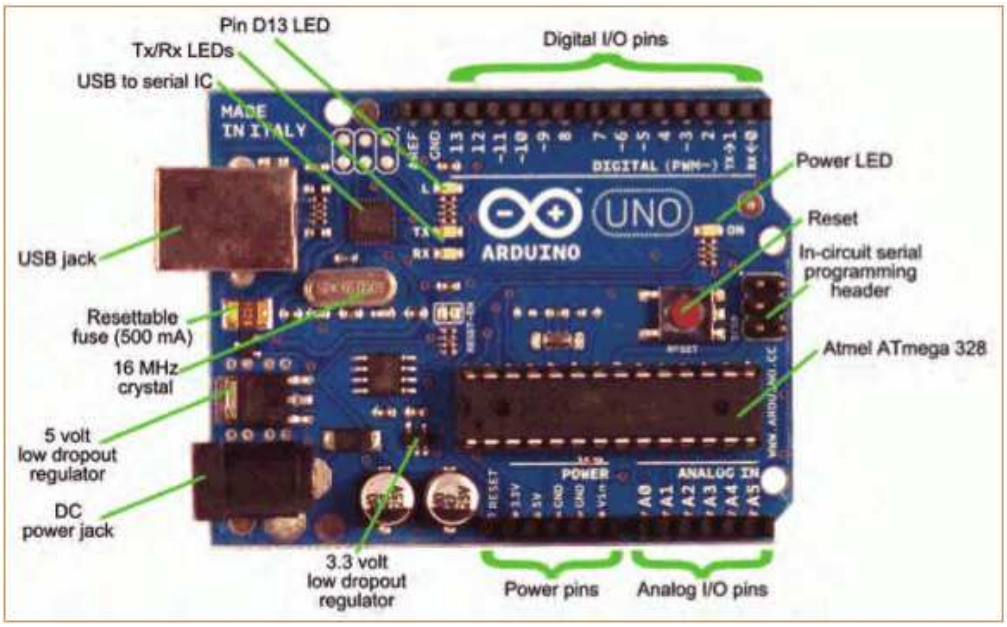
Shields

Additionally, there are these things called **shields** – basically they are pre-built circuit boards that fit on top of your Arduino and provide additional capabilities – controlling motors, connecting to the internet, providing cellular or other wireless communication, controlling an LCD screen, and much more.



Arduino Uno Pinout





Input and Output

Each of the 14 digital pins on the Arduino Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms.

Pins of A0-A5 on arduino Uno can be used to read analog values from analog input devices

In addition, some pins have specialized functions.

Serial: pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

TWI(Two Wire Interface): A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Compare board specs

This table shows a quick comparison between the characteristics of all the Arduino and Genuino boards.

Name	Processor	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
Mega 2560	ATmega 2560	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega 32U4	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Uno	ATmega 328P	16 MHz	6/0	14/6	1	2	32	Regular	1
Leonardo	ATmega 32U4	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Mini	ATmega 328P	16 MHz	8/0	14/6	1	2	32	-	-
Nano	ATmega 168 ATmega 328P	16 MHz	8/0	14/6	0.512 1	1 2	16 32	Mini	1

Getting Started with Arduino using Arduino Platform.

The Arduino Integrated Development Environment - or Arduino Software (IDE). The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

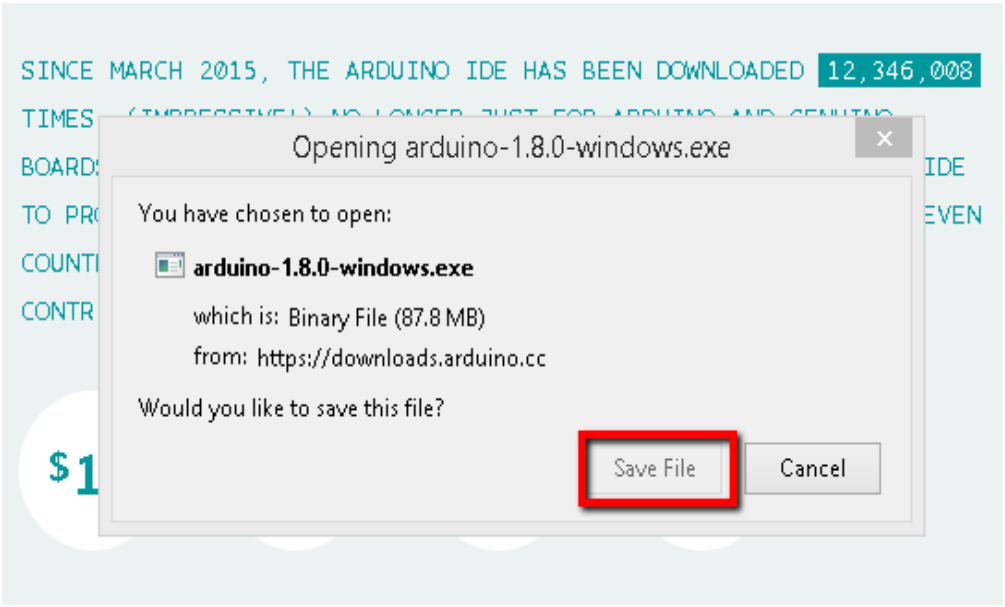
Installation of Arduino IDE Software

1. Installation steps for " Arduino IDE"

   Download the Arduino IDE Software from Google Arduino IDE.

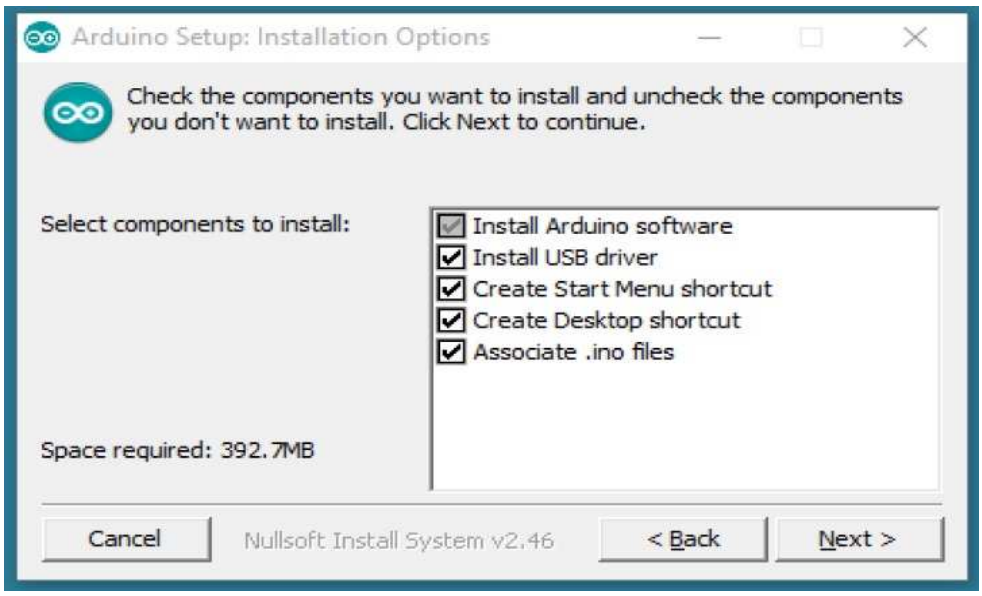
Click on just download.

Will get a pop-up asking for saving the file, click on save file. It will start downloading.

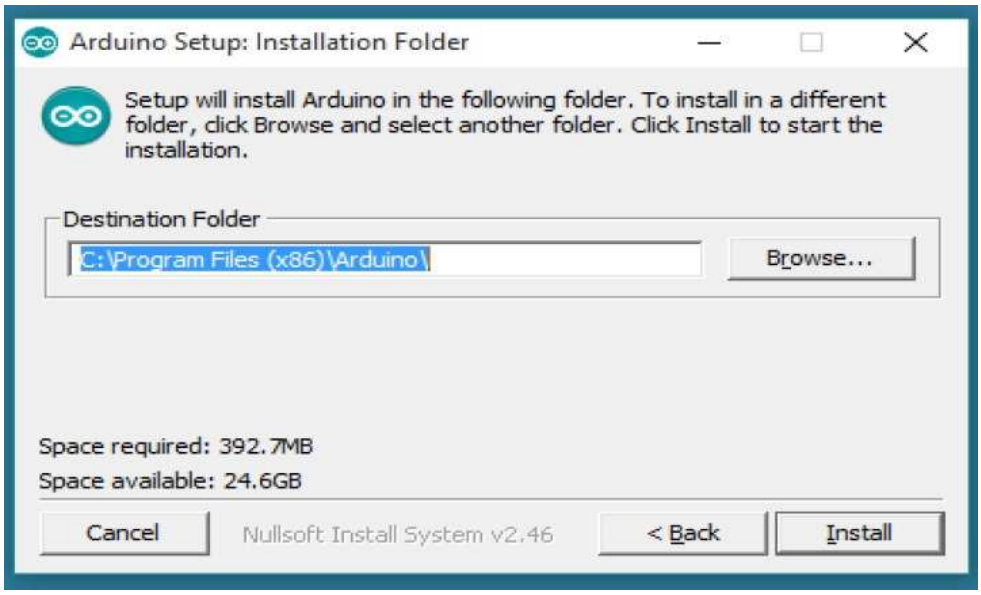


After the download finishes, proceed with the installation and you get a confirmation pop-up window with Yes and No buttons and select Yes button. Please allow the driver install.

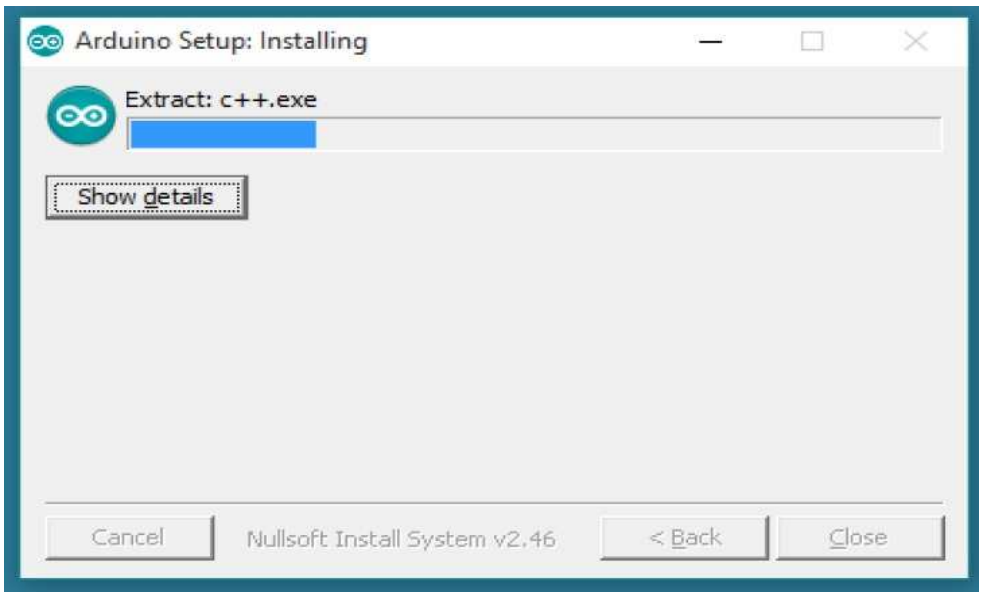
Choose the components to install and Click on next button as shown below.



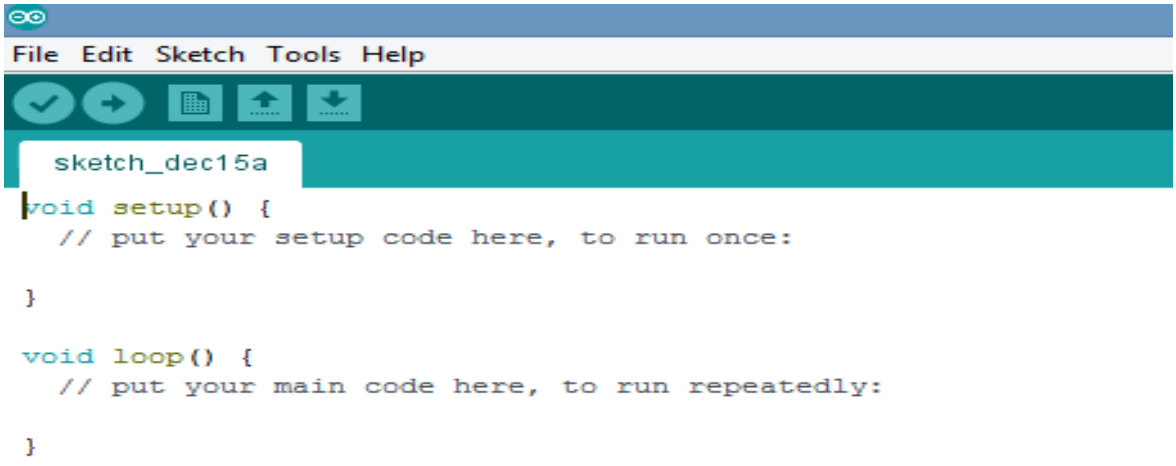
Choose the drive or directory where the software to be installed and click on install.



The process will extract and install all the required files to execute the Arduino Software.



Once the installation is completed, click on Arduino icon (we can see in Desktop), it will open the Arduino IDE as shown below.

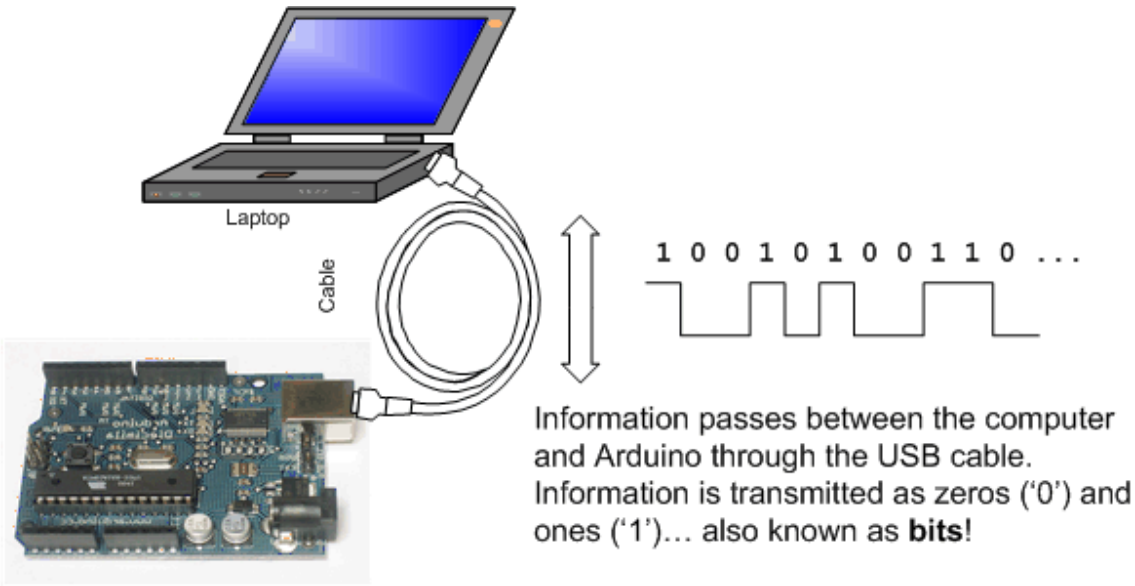


Serial Communication

Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().



Serial may sound like a tasty breakfast food, but it's actually quite different. The word serial means "one after the other." For example, a serial killer doesn't stop with one murder, but stabs many people one after the other. Serial data transfer is when we transfer data one bit at

a time, one right after the other.

Information is passed back & forth between the computer and Arduino by, essentially, setting a pin high or low. Just like we used that technique to turn an LED on and off, we can also send data. One side sets the pin and the other reads it. It's a little like Morse code, where you can use dots and dash to send messages by telegram. In this case, instead of a long cable, it’s only a few feet.

The Arduino Mega has three additional serial ports: Serial1 on pins 19 (RX) and 18 (TX), Serial2 on pins 17 (RX) and 16 (TX), Serial3 on pins 15 (RX) and 14 (TX). To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground.

The Arduino Due has three additional 3.3V TTL serial ports: Serial1 on pins 19 (RX) and 18 (TX); Serial2 on pins 17 (RX) and 16 (TX), Serial3 on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, SerialUSB'.

BIT RATE AND BAUD RATE

While bit rate and baud rate are closely linked, they aren't the same thing. Bit rate is a measure of the number of data bits -- zeros and ones -- that can be transmitted in one second. A bit rate of 2,400 bits per second (bps) for example, would mean 2,400 zeros and ones are transmitted each second.

Baud rate represents the number of times per second a signal (changing from zero to one or one to zero) or symbol (the connection's voltage, frequency or phase) in a communications channel changes state or varies. For example, a 2,400 baud rate means the channel is changing states up to 2,400 times per second. In this case, the baud rate is the same as the bit rate: 2,400 bps, and a bit rate of one equals a baud rate of one.

The two most common/confused words in digital communication – Bit rate and Baud rate. Generally, communication is concerned with transmission of data. In digital communication, there are two entities that are needed to carry out communication – the data to be transmitted and the signal over which the data is transmitted. Now, we have two entities to be worried about – the data and the signal. The most common misconception is that most people think both travel at the same speed! – NO!

The Difference:

Digital data is very different from digital signal. The process of converting digital data to digital signal is called as **line coding**. Now, to discriminate between data and signal, data is what we need to send. But signal is what we can send. So, signal is the carrier which carries data. Also, keep in mind that the smallest entity of the data, that can represent a piece of information is called data element and shortest meaningful unit of a signal is called signal element. Consider this as in the following scenario – Consider a train. Each carriage is a signal element. Each passenger inside the train is a data element. The train as a whole is a signal and all passengers together represent a data.

Data rate and Signal rate:

Data rate – Number of data elements transmitted per second.

Signal rate – Number of signal elements transmitted per second.

Now, the unit of data rate is bit rate. And the unit of signal rate is pulse rate/ modulation rate/ baud rate or simply baud. From the previous example, we can see that, a carriage in a train can carry more than one person. So, if you consider the number of person is more than one per carriage, you can say that bit rate is greater than baud rate for the signal.

Calculating the baud rate:

Baud rate is calculated using the below formula here, N is the bit rate and r is the number of data elements carried by each signal element. Here r must be as great as possible for better efficiency. (Stuff more people in a carriage :P)

$$S = \frac{N}{r} \quad baud$$

From the above text, it is clearly inferred that the bit rate must be greater than the baud rate for higher efficiency. The aim is to transmit as many data element as possible in a signal element. There are different methods to do this which are collectively called as line coding schemes. Some of the popular line coding schemes are: Non-return to zero (NRZ), Manchester, Alternate mark inversion (AMI) and also multi-level schemes are available.

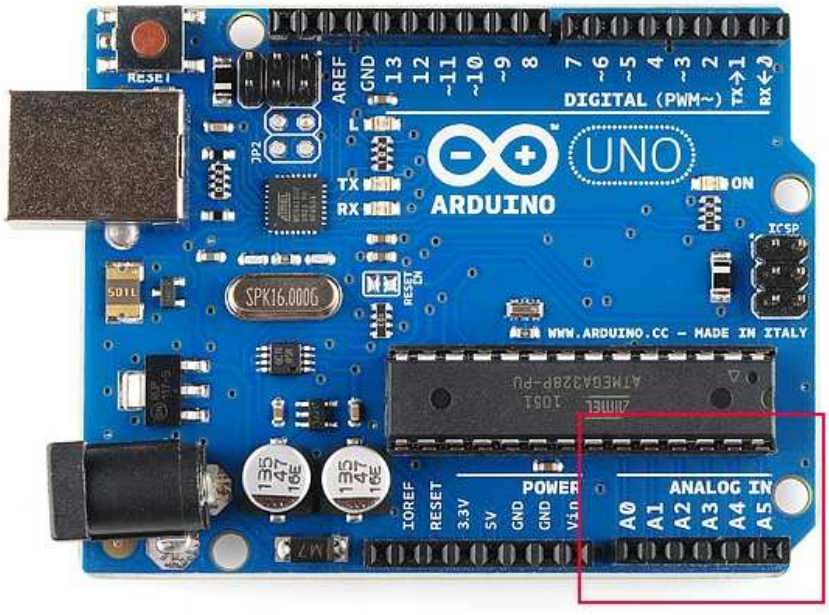
Analog to Digital Conversion

The Analog World

Microcontrollers are capable of detecting binary signals: is the button pressed or not? These are digital signals. When a microcontroller is powered from five volts, it understands zero volts (0V) as a binary 0 and a five volts (5V) as a binary 1. The world however is not so simple and likes to use shades of gray. What if the signal is 2.72V? Is that a zero or a one? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may output 0.01V or 4.99V or anything in between. Luckily, nearly all microcontrollers have a device built into them that allows us to convert these voltages into values that we can use in a program to make a decision.

What is the ADC?

An Analog to Digital Converter (ADC) is a very useful feature that converts an analog voltage on a pin to a digital number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.



Not every pin on a microcontroller has the ability to do analog to digital conversions. On the

Arduino board, these pins have an ‘An’ in front of their label (A0 through A5) to indicate these pins can read analog voltages.

ADCs can vary greatly between microcontrollers. The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (210) discrete analog levels. Some microcontrollers have 8-bit ADCs (28 = 256 discrete levels) and some have 16-bit ADCs (216 = 65,536 discrete levels).

The way an ADC works is fairly complex. There are a few different ways to achieve this feat (see Wikipedia for a list), but one of the most common technique uses the analog voltage to charge up an internal capacitor and then measure the time it takes to discharge across an internal resistor. The microcontroller monitors the number of clock cycles that pass before the capacitor is discharged. This number of cycles is the number that is returned once the ADC is complete.

Relating ADC Value to Voltage

The ADC reports a *ratiometric value*. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Analog to digital conversions are dependent on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

If your system is 3.3V, you simply change 5V out with 3.3V in the equation. If your system is 3.3V and your ADC is reporting 512, what is the voltage measured? It is approximately 1.65V.

If the analog voltage is 2.12V what will the ADC report as a value?

$$\frac{1023}{5.00V} = \frac{x}{2.12V}$$

Rearrange things a bit and we get:

$$\frac{1023}{5.00V} * 2.12V = x$$

$$x = 434$$

Ahah! The ADC should report 434.

Basic Arduino Programming Keywords

Keywords		Syntax	Example	Description
Data types	**int**	**int** variable_name;	**int** m; // int m=8	To define integer numbers. (16 bits in size)
	float	**float** variable_name;	**float** p; // float p=5.035	to define floating point numbers(32 bits)
	String	**string** variable_name;	**string** n; // n="college"	to define array of characters
	boolean	**boolean** variable_name;	**boolean** b;// b=1 or 0	defines simple logical true/false(high or low)
	char	**char** variable_name;	**char** p; //p='x'	To define characters (signed or unsigned)
	byte	**byte** variable_name;	**byte** r;	unsigned number from 0-255
	array	**array** variable_name[size];	**int array** a[10];// 'a' can store 10 elements from 0 to 9	collection of identical elements of same type
pinMode()		**pinMode**(Var1,OUTPUT/INPUT)	**Pinmode**(ledpin ,OUTPUT); **Pinmode**(13, INPUT);	Configures the specified pin to behave either as an input or output
digitalRead()		**digitalRead**(pin);	**Int** v1=12; Val= **digitalRead**(v1);	Reads the value from a specified digital pin , either HIGH or LOW
digitalWrite()		**digitalWrite**(pin, value);	**digitalWrite**(13, HIGH);	Write HIGH or LOW value to digital pin
analogRead()		**analogRead**(pin);	**int** m=**analogRead**(A0); **int** n=**analogRead**(A1);	Reads the analog value from a specified analog pin(A0-A5)
analogWrite()		**analogWrite**(pin);	**int** m=**analogWrite**(A0); **int** n=**analogWrite**(A1);	Write the analog value from a specified analog pin(A0-A5)
delay()		**delay**(value)**;** **millis**(value)**;**	**delay**(1000)**;** //one sec dealy **millis**(1000)**;** //one milli	Pauses the program for the amount of time specified

	micros(value)**;**	sec dealy **micros**(1000)**;** //one micro sec dealy	
Map()	**map(**variable, alow, ahigh, mlow ,mhigh**);**	**y=map(**x,0,1023,1,200**);**	Re-maps a number from one range to another
Serial.begin()	**Serial.begin**(baud rate)	**Serial.begin**(9600)	Sets the data rate in bits per second (baud) for serial data transmission.(9600,14400 etc)

Serial.print()	**Serial.print(**data**)**	**Serial.print(**78**) //gives 78**
		
		Serial.print("college"**) //prints**
		
		College
		
		Int v**=3; Serial.print(**v**) // prints 3**
			Prints data to the serial port as human readable ASCII text

Serial.println()	**Serial.println()**	**Serial.println(**56**);**	Prints the value 56 fallowed by a carriage return character(ASCII 13 or '\n') i.e new line character
Serial.read()	**Variable= Serial.read(**variable**)**	**v= Serial.read(**p**);** **m=Serial.read(**A0**);**	Reads incoming serial data
Serial.available()	**Serial.available()**	**if (Serial.available()** > 0**)** **{** **int** incomingByte **= Serial.read();** **}**	Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes)
Serial.write()	**Serial.write(val)**	**Serial.write(45);** **int** bytesent=**Serial.write("HEL LO");**	Writes the binary data to the serial port And it will return the number of bytes written on serial monitor
Serial.println(val, for	**Serial.println(val, format)**	**Serial.print(**78, BIN**)** **Serial.print(**78, OCT**)**	gives "1001110" gives "116"

mat)		**Serial.print(**78, DEC**)** **Serial.print(**78, HEX**)**	gives "78" gives "4E
Arrays	Data_type array_name[size];	**int** mypins[]={2,4,8}; **int** myval[4]={2,4,-8,3}**;**	An array defines the pin numbers of 2,4,8 which acts as input or output An array initializes four values of type **int**
lcd.begin()	**lcd.begin(**cols,rows**);**	**lcd.begin(16, 2);** //defines for 16,2 LCD display	Initializes the interface to the lcd screen **Note**: use #include <LiquidCrystal.h> to interface lcd with arduino
lcd.print()	**lcd.print(**data**)**	**lcd.print(**"Hello world"**)**	Prints text to LCD
lcd.cursor()	**lcd.cursor(**col,row**)**	**lcd.cursor(5,1**)//points to 5th coloumn,2nd row	an underscore (line) at the position to which the next character will be written
lcd.clear()	**lcd.clear()**	**lcd.clear()**	Clears the LCD screen

INTERFACING SENSORS WITH ARDUINO

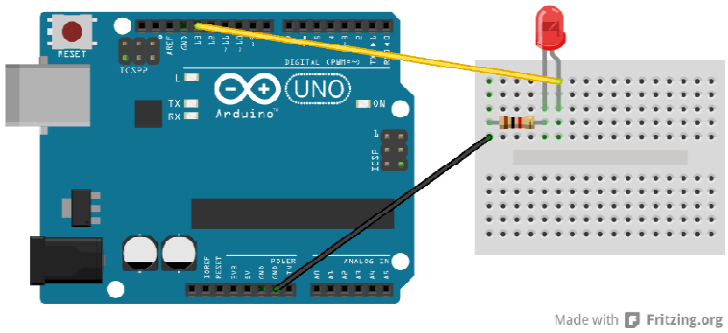
A. BLINKING OF LED USING ARDUINO

This example shows the simplest thing you can do with a Arduino to see physical output it blinks an LED.

a. Component Used

i. Arduino
ii. Breadboard
iii. LED
iv. Connecting Wires

a. Circuit Diagram



b. Connection Detail

i. Place the Arduino on the breadboard.
ii. Connect the LED on the Breadboard.
iii. Connect the VCC of led to 13 of the Arduino and GND to GND respectively.
iv. Upload the code.

c. Code

```
int LED=13;
void setup()   // the setup function runs once when you press reset or power the board
{  pinMode(LED, OUTPUT);  // initialize digital pin D1 as an output.
}
void loop() // the loop function runs over and over again forever
{ 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
}
```

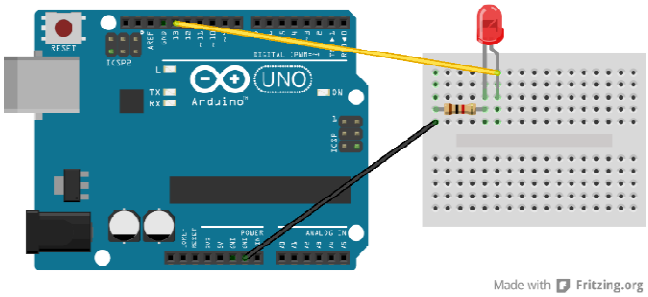
B. Fading of LED using Arduino

The example will demonstrate the fading of LED from Low brightness to high brightness and going back to low using Arduino.

a. Component Used

As earlier Program.

b. Circuit Diagram



c. Connection Detail

i. Place the Arduino on the breadboard.

ii. Connect the LED on the Breadboard.

iii. Connect the VCC of led to **13** of the Arduino and GND to GND respectively.

iv. Upload the code.

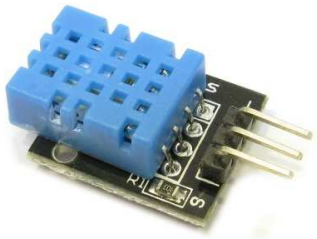
d. Code

```
int led = 13;          // the PWM pin the LED is attached
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by
void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
analogWrite(led, brightness);
brightness = brightness + fadeAmount;   // change the brightness for next time through the loop
if (brightness == 0 || brightness == 255) // reverse the direction of the fading at the ends
of the fade:
{
fadeAmount = -fadeAmount ;}
delay(30);  // wait for 30 milliseconds to see the dimming effect }
```

C. Interfacing Arduino with Temperature and Humidity Sensor (DHT11).

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).

The DHT11 detects water vapor by measuring the electrical resistance between two electrodes. The humidity sensing component is a moisture holding substrate with electrodes applied to the surface. When water vapor is absorbed by the substrate, ions are released by the substrate which increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity. Higher relative humidity decreases the resistance between the electrodes, while lower relative humidity increases the resistance between the electrodes. The DHT11 measures temperature with a surface mounted NTC temperature sensor (thermistor) built into the unit.



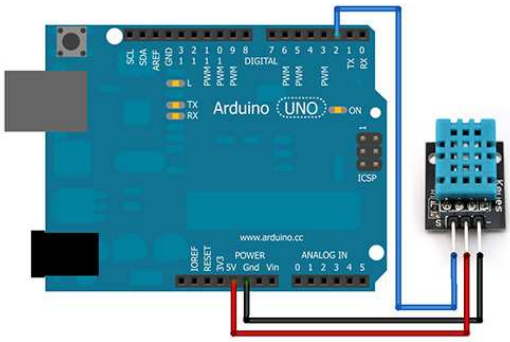
Technical Specification:

- Humidity Range: 20-90% RH
- Humidity Accuracy: ±5% RH
- Temperature Range: 0-50 °C
- Temperature Accuracy: ±2% °C
- Operating Voltage: 3V to 5.5V

a. Components Required

i. Arduino
ii. DHT11
iii. Breadboard
iv. Connecting Wires

b. Circuit Diagram



c. Connection Detail

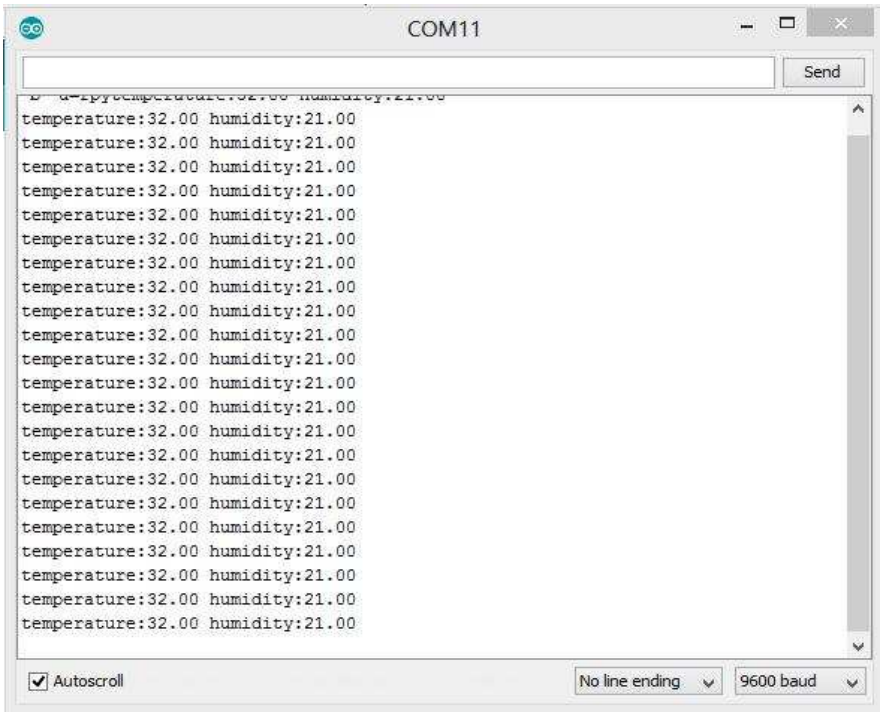
i. Place the Arduino on the breadboard.
ii. Place the DHT11 Temperature and Humidity Sensor on the breadboard (assuming DHT11library is installed).
iii. Connect VCC and GND of DHT11with 3.3v and GND to Arduino.
iv. Connect data pin from DHT11 to 2 of the Arduino.
v. Upload the code.

d. Code

```
#include <DHT11.h>            // including the library for dht11
int pin = 2;                  //Defining the pin D1
DHT11 dht11(pin);             //initialization of pin
void setup()
{
Serial.begin(9600);
}

Void loop()
{
int err;
float temp, humi;
if ((err = dht11.read(humi, temp)) == 0)// check if dht11 reads the value
{
Serial.print("temperature:");
Serial.print(temp);                    //Displaying the Temperature
Serial.print(" humidity:");
Serial.print(humi);                    //Displaying the Humidity
Serial.println();
}
}
```

e. Output



f. Application

i. Attention to chemical materials inside the chemical drums.
ii. Restoration process.
iii. Temperature Affect.
iv. Monitoring Cold storage vehicle.
v. Home Automation.

D. INTERFACING LDR WITH ARDUINO

A **Light Dependent Resistor** (LDR) or a photo resistor is a device whose resistivity is a function of the incident electromagnetic radiation. Hence, they are light sensitive devices. They are also called as photo conductors, photo conductive cells or simply photocells. They are made up of semiconductor materials having high resistance.

A **light dependent resistor** works on the principle of photo conductivity. Photo conductivity is an optical phenomenon in which the materials conductivity is increased when light is absorbed by the material. When light falls i.e. when the photons fall on the device, the electrons in the valence band of the semiconductor material are excited to the conduction band. These photons in the incident light should have energy greater than the band gap of the semiconductor material to make the electrons jump from the valence band to the conduction band. Hence when light having enough energy strikes on the device, more and more electrons are excited to the conduction band which results in large number of charge carriers. The result of this process is more and more current starts flowing through the device when the circuit is closed and hence it is said that the resistance of the device has been decreased.

a. Component Used

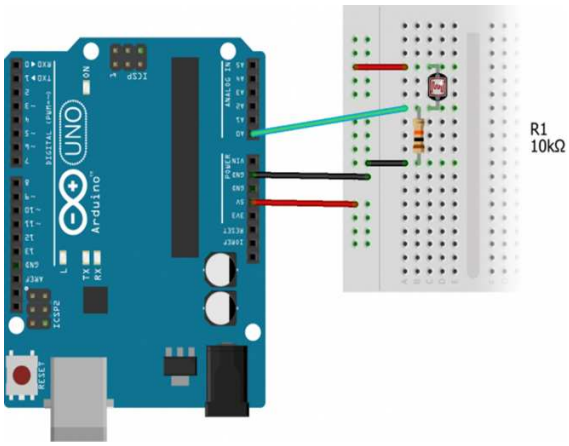
i. Arduino

ii. LDR(**Light Dependent Resistor)**

iii. 10K resistor

iv. Wire

b. Circuit Diagram



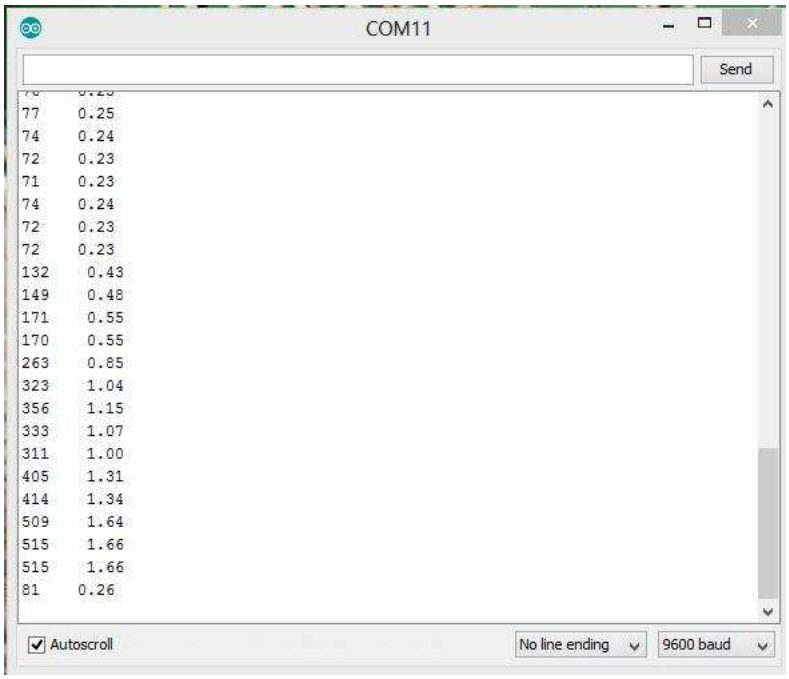
c. Connection Detail

i. Place the Arduino (ESP8266) on the breadboard.
ii. Take LDR and place it on Breadboard.
iii. Connect VCC (3.3V) toward any one pin (say pin 1) of the LDR.
iv. Connect Pull down Resistor with ground on the other pin (say pin 2) of LDR.
v. Connect a wire from the other pin (pin 2) to A0 of Arduino.

d. Code

```
void setup()
 {
   Serial.begin(9600);                    //setting up the serial monitor
  }
void loop()
 {
   int sensorValue = analogRead(A0);          //Accepting the analog value
   Serial.println(sensorValue);      //Printing the value on the serial monitor
   float voltage = sensorValue * (3.3 / 1023.0);  //Converting the value to Volt
   Serial.println(voltage);   // Printing Voltage
   delay(1000);
}
```

e. Output



f. Application

i. Camera Exposure Control
ii. Photocopy Machines - density of toner
iii. Street Light Control
iv. Automatic Headlight Dimmer
v. Night Light Control

E. Interfacing Ultrasonic Sensor with Arduino

This is the HC-SR04 ultrasonic ranging sensor. This economical sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit.

There are only four pins that you need to worry about on the HC-SR04: VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground). You will find this sensor very easy to set up and use for your next range-finding project.

It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.

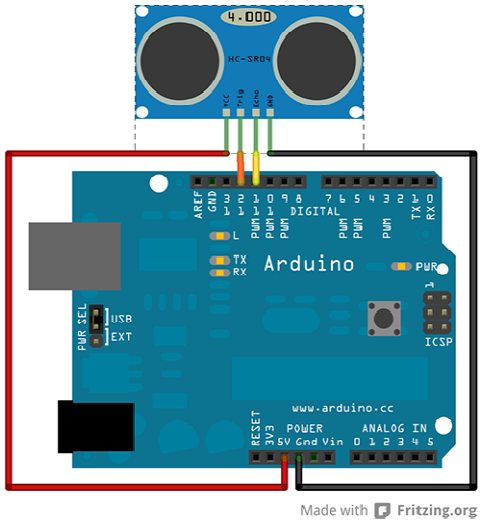
a. Components used

i. Arduino

ii. Ultrasonic Sensor

iii. Wire

b. Circuit Diagram



c. Connection Detail

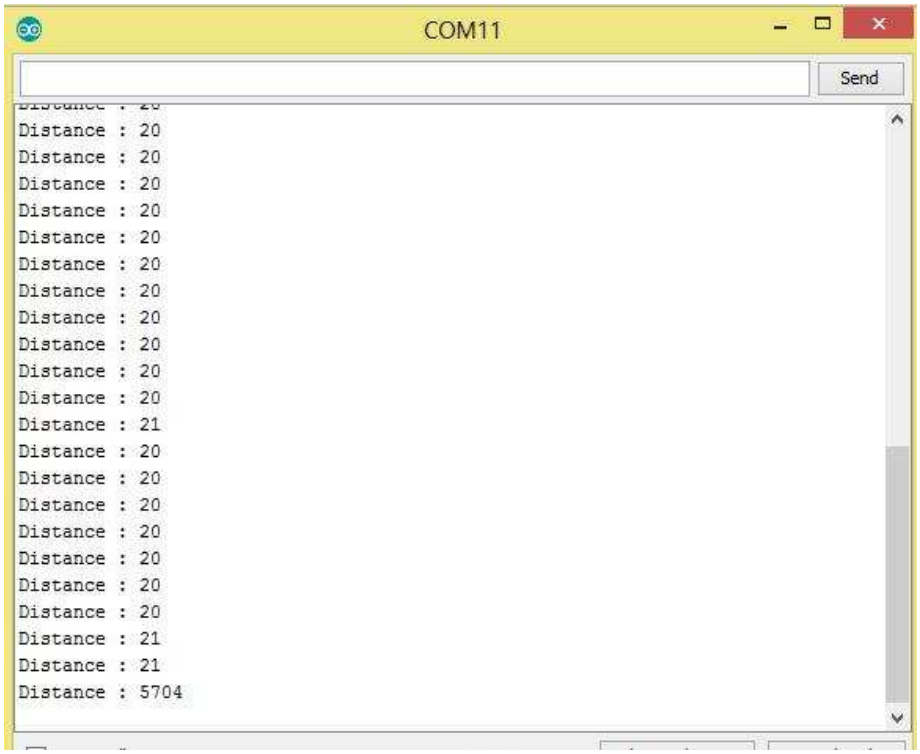
i. Place the Arduino (ESP8266) on the breadboard.

ii. Take Ultrasonic Sensor and place it on Breadboard.
iii. Connect VCC (3.3V) and GND along with TRIG and ECHO.
iv. Connect TRIG to pin 12 and ECHO to pin 11.
v. Upload the code given.

d. Code

```
#define TRIGGER 12        // defining trigger pin
#define ECHO    11        // defining echo pin
void setup()
{           Serial.begin(9600);
            pinMode(TRIGGER, OUTPUT);    //initializing trigger as output
            pinMode(ECHO, INPUT);//initialing trigger as input      }
void loop()
{
            int duration, dist;
            digitalWrite(TRIGGER, LOW);  // make trigger low
            delayMicroseconds(2);
            digitalWrite(TRIGGER, HIGH); // make trigger high
            delayMicroseconds(10);  //give 10 microsec delay
            digitalWrite(TRIGGER, LOW);
            duration = pulseIn(ECHO, HIGH);
            dist = (duration / 2) / 29.1;  // calibrate the distance using pulse
            Serial.print(dist); // Print the distance value
            Serial.println(" cm");
            delay(500);
                }
```

e. Output

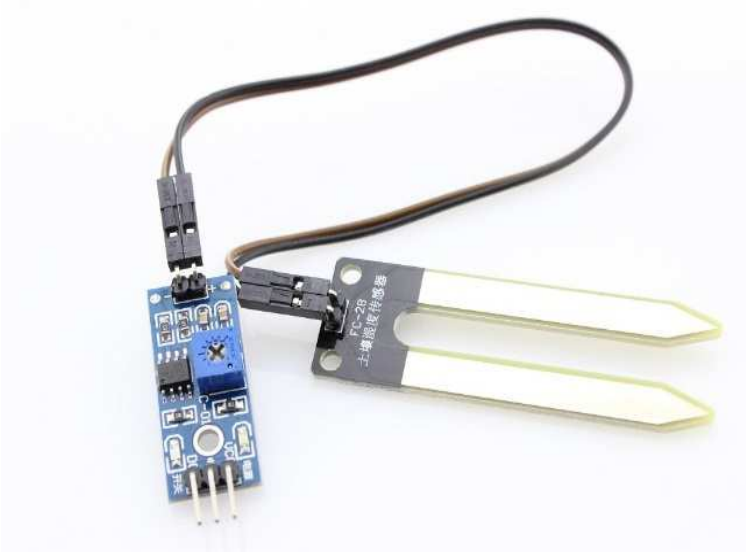


f. Application

i. Level Measuring
ii. Detecting Boom Height on Agricultural Machine
iii. Ultrasonic Sensors for Anti-Collision Detection on Aerial Work Platforms
iv. Bottle Counting on Drink Filling Machines
v. People detection counting

F. Interfacing Soil moister Sensor with Arduino

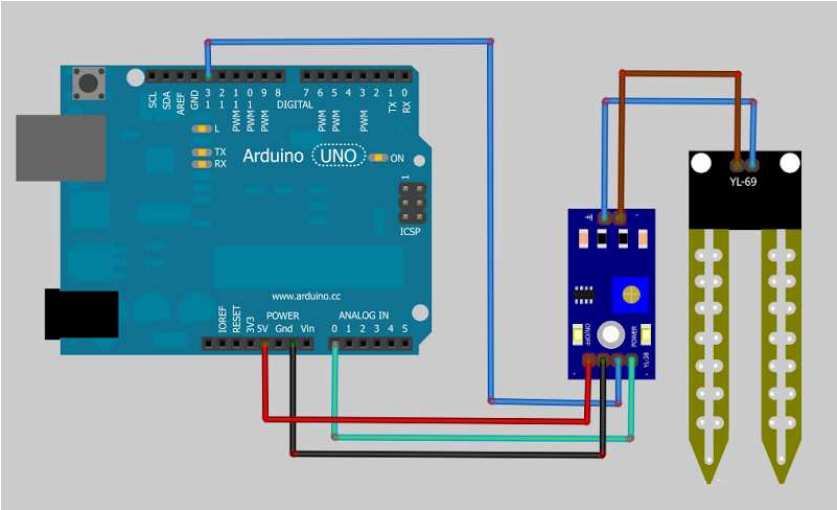
Soil moisture sensors measure the volumetric water content in soil. Since the direct gravimetric measurement of free soil moisture requires removing, drying, and weighting of a sample, soil moisture sensors measure the volumetric water content indirectly by using some other property of the soil, such as electrical resistance, dielectric constant, or interaction with neutrons, as a proxy for the moisture content. The relation between the measured property and soil moisture must be calibrated and may vary depending on environmental factors such as soil type, temperature, or electric conductivity. Reflected microwave radiation is affected by the soil moisture and is used for remote sensing in hydrology and agriculture. Portable probe instruments can be used by farmers or gardeners.



a. Component Required

i. Arduino
ii. Soil moisture sensor
iii. Connecting Wires
iv. Breadboard

b. Circuit Diagram



c. Connection Detail

i. Place the Arduino (ESP8266) on the breadboard.

ii. Take Ultrasonic Sensor and place it on Breadboard.

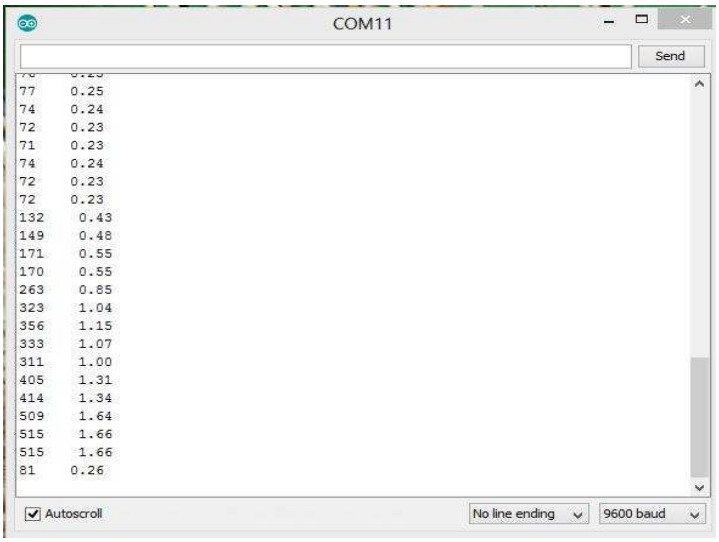
iii. Connect VCC and GND with 3.3 v and GND of Arduino.

iv. Connect the A0 with the A0 of the Arduino.

d. Code

```
void setup()
{
        Serial.begin(9600);
}
void loop()
{
        int soil_moisture=analogRead(A0);  // read from analog pin A0
        Serial.print("analog value: ");
        Serial.println(soil_moisture);
        delay(1000);
}
```

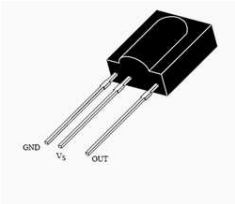
e. Output


COM11
Send
77 0.25
74 0.24
72 0.23
71 0.23
74 0.24
72 0.23
72 0.23
132 0.43
149 0.48
171 0.55
170 0.55
263 0.85
323 1.04
356 1.15
333 1.07
311 1.00
405 1.31
414 1.34
509 1.64
515 1.66
515 1.66
81 0.26
Autoscroll
No line ending
9600 baud

G. Interfacing TSOP (Infrared Receiver) with Arduino.

The TSOP 1738 is a member of IR remote control receiver series. This IR sensor module consists of a PIN diode and a pre amplifier which are embedded into a single package. The output of TSOP is active low and it gives +5V in off state. When IR waves, from a source, with a center frequency of 38 kHz incident on it, its output goes low. Lights coming from sunlight, fluorescent lamps etc. may cause disturbance to it and result in undesirable output even when the source is not transmitting IR signals. A band pass filter, an integrator stage and an automatic gain control are used to suppress such disturbances.

TSOP module has an inbuilt control circuit for amplifying the coded pulses from the IR transmitter. A signal is generated when PIN photodiode receives the signals. This input signal is received by an automatic gain control (AGC). For a range of inputs, the output is fed back to AGC in order to adjust the gain to a suitable level. The signal from AGC is passed to a band pass filter to filter undesired frequencies. After this, the signal goes to a demodulator and this demodulated output drives an npn transistor. The collector output of the transistor is obtained at pin 3 of TSOP module.



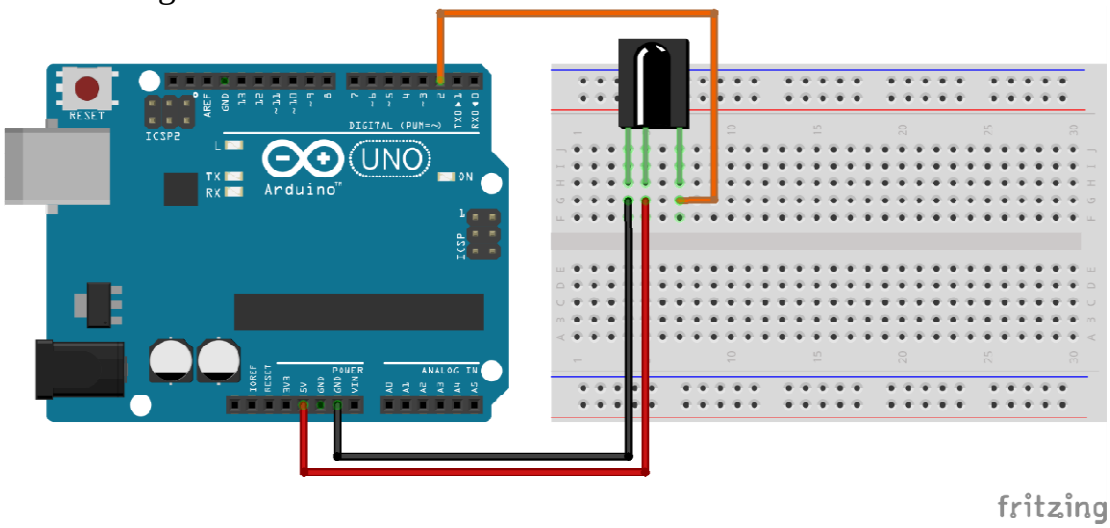
a. Component Required

i. Arduino
ii. TSOP
iii. Connecting Wires
iv. BreadBoard
v. Remote Control

b. Connection Detail

i. Place the Arduino on the Breadboard.
ii. Place the TSOP on the breadboard.
iii. Connect the 1st pin to GND followed by VCC rail.
iv. Connect the Data line to the pin D1 of the Arduino.
v. Install the Library "IRremote.h ".
vi. Upload the code.

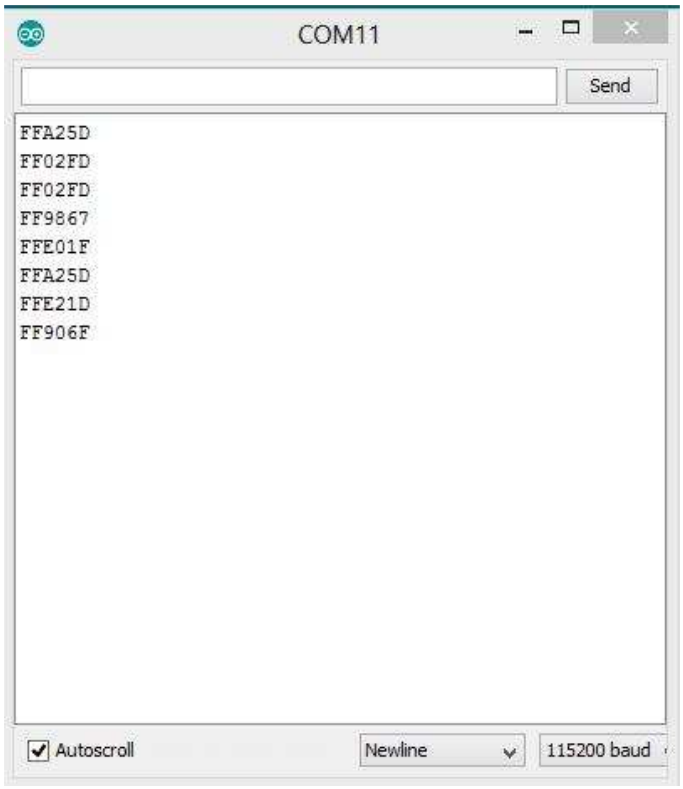
c. Circuit Diagram



d. Code

```
#include <IRremote.h>   //library include
int RECV_PIN = 2;          //defining pin for IR receiver
IRrecv irrecv(RECV_PIN);
decode_results results;
long lastMsg = 0;
char msg[50];
int value = 0;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn();                    // Start the receiver
}
void loop()
{
  long now = millis();
    if (now - lastMsg > 2000)
{
    if (irrecv.decode(&results))
    {
      Serial.println(results.value, HEX);          // printing the hex code or key
      irrecv.resume();                             // Receive the next value
      }
  }
}
```

e. Output



f. Application

i. Controlling fan and light with tv remote
ii. Remote access of other appliance.

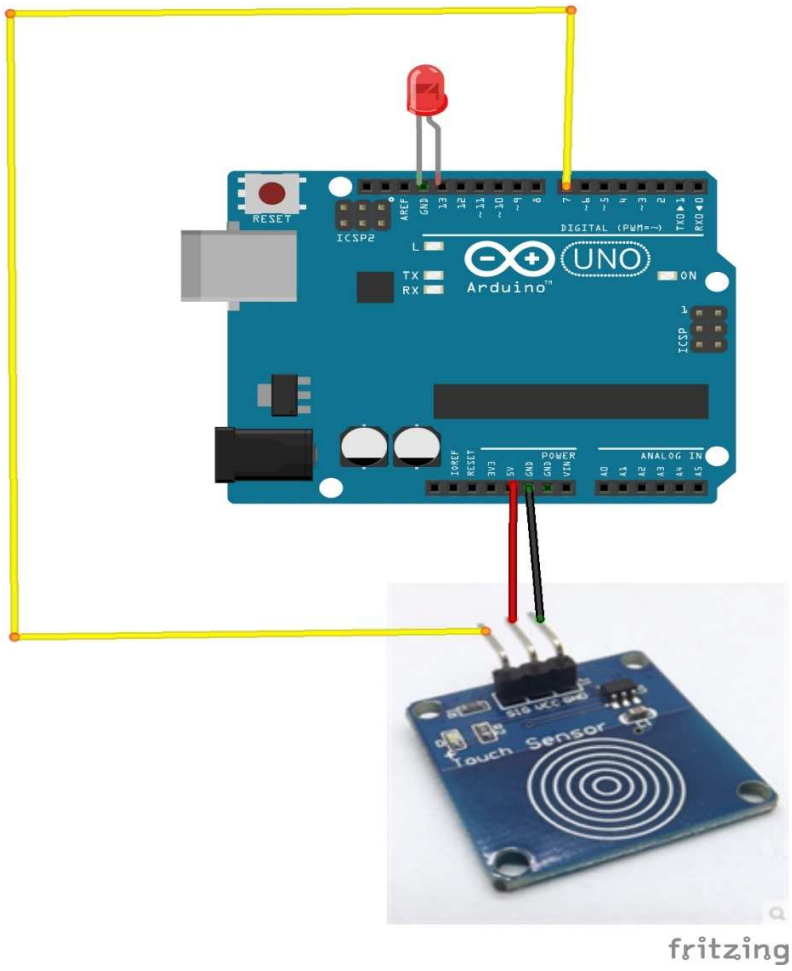
H. Interfacing Touch Sensor with Arduino.

This device uses your body as part of the circuit. When you touch the sensor pad, the capacitance of the circuit is changed and is detected. That detected change in capacitance results in the output changing states.

a. Component Required

i. Arduino
ii. Touch Sensor
iii. Led
iv. Wires
v. Breadboard

b. Circuit Diagram



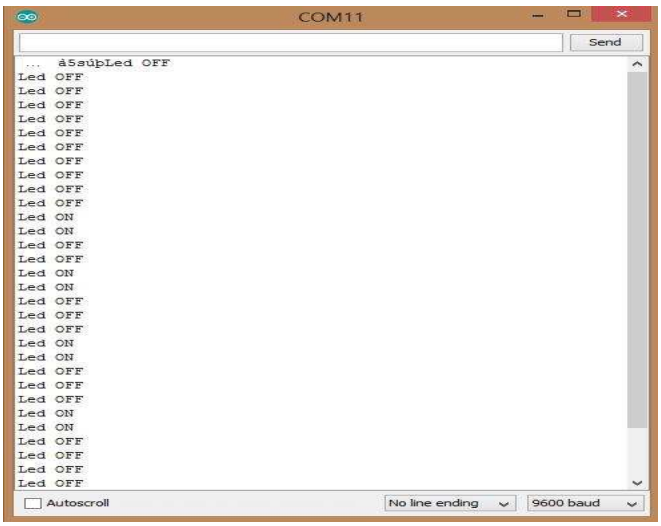
c. Connection Detail

i. Place the Arduino on the breadboard.
ii. Place the touch sensor and LED on the breadboard.
iii. Connect VCC and GND of the touch sensor to 3.3v and GND of the Arduino.
iv. Connect the –ve pin of the LED to the GND of the Arduino.
v. Connect +ve pin of the LED to the 13 of the Arduino.
vi. Connect the SIG pin of the touch sensor to the **7** of the Arduino.

d. Code

```
int TouchSensor = 7; //connected to Digital pin D1
int led = 13; // connected to Digital pin D4
void setup()
{
  Serial.begin(9600); // Communication speed
  pinMode(led, OUTPUT);
  pinMode(TouchSensor, INPUT);
}
void loop(){
  if(digitalRead(TouchSensor)==HIGH)       //Read Touch sensor signal
    {
     digitalWrite(led, HIGH);   // if Touch sensor is HIGH, then turn on
     Serial.println("Led ON");
    }
  else
    {
     digitalWrite(led, LOW);    // if Touch sensor is LOW, then turn off the led
     Serial.println("Led OFF");
    }
  delay(500); // Slow down the output for easier reading
}
```

e. Output



f. Applications

i. Remote controllers
ii. Light switches
iii. Sealed control panels
iv. Home appliances
v. Door-lock systems