A project Report on

AUTOMATED COVID TEMPERATURE BASED GATE OPENING AND VISITOR COUNTER SYSTEM

Submitted in partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY
in
ELECTRICAL AND ELECTRONICS ENGINEERING
by

G.CH.SAI MANIKANTA	**18P35A0258**
G. LEELA ABHIRAM	**18P35A0257**
S.J. VENKATA ARAVIND	**18P35A0276**
N. SUMAN RAJU	**18P35A0270**
A. VARA PRASAD	**17P35A0201**

Under the esteemed guidance of

Mr. S. REDDY RAMESH M.Tech.,

Assistant Professor

Department of Electrical and Electronics Engineering

SRI SAI ADITYA INSTITUTE OF SCIENCE AND TECHNOLOGY

(Affiliated to JNTUK, Approved by AICTE, New Delhi)

Surampalem, ADB road, East Godavari District, A.P, India

2018-2021

CERTIFICATE

This is to certify that project report entitled **"AUTOMATED COVID TEMPERATURE BASED GATE OPENING AND VISITOR COUNTER SYSTEM"** is being submitted by

G.CH.SAI MANIKANTA	**18P35A0258**
G. LEELA ABHIRAM	**18P35A0257**
S.J. VENKATA ARAVIND	**18P35A0276**
N. SUMAN RAJU	**18P35A0270**
A. VARA PRASAD	**17P35A0201**

in the partial fulfilment of the requirements for the award of degree of the **Bachelor of Technology** in **Electrical and Electronics Engineering.** It is the record of bona fide work carried out by them under the esteemed guidance and supervision of

Mr. S. REDDY RAMESH M.Tech., Assistant Professor.

Project Guide	**Head of the Department**
Mr. S. REDDY RAMESH M.Tech.,	**Sri J. PAVAN M. Tech., (Ph.D.)**
Assistant Professor	Associate Professor

Department of Electrical and Electronics Engineering

SRI SAI ADITYA INSTITUTE OF SCIENCE & TECHNOLOGY

(Affiliated to JNTU, Kakinada and Approved by AICTE, New Delhi)

Aditya Nagar, ADB road, Surampalem, E.G.Dist.

2018-2021

ACKNOWLEDGEMENT

We are thankful to our guide **Mr. S. REDDY RAMESH** **M.Tech.,** spared his valuable time append novel ideas to guide us in intelligent. We are indebted to him without whom we not have culminated to pinnacle of the project.

We wish to convey our sincere thanks to Head of the Department of Electrical and Electronics Engineering **Mr. J. PAVAN** **M. Tech., (Ph.D.)** for provide vital information in carrying out our project.

We are thankful to our Principal **Dr. T. K. RAMA KRISHNA RAO** for providing appropriate environment required for the project.

Not to forget our teaching, non-teaching staff, our friends and our parents who had directly and indirectly helped and supported us in completing our work successfully within given time.

With sincere regards,

G.CH.SAI MANIKANTA	**18P35A0258**
G. LEELA ABHIRAM	**18P35A0257**
S.J. VENKATA ARAVIND	**18P35A0276**
N. SUMAN RAJU	**18P35A0270**
A. VARA PRASAD	**17P35A0201**

ABSTRACT

This project describes the automated covid temperature based gate opening and visitor counting system where it scans the temperature and at certain temperature it releases the gate and opens if the temperature is beyond preprogrammed temperature then the gate is kept closed and then this will also scans the visitors who visits the area or who cross the gate so that we can have a count on the visitors. This is based on the current situation covid 19 break out so that we are doing this project which will help the current society. We are using IOT for this project to make it easier to maintain the output values of the system. Where we can save and monitor these values for future purpose.

This project briefly describes the “automated covid temperature based gate opening and visitor counting system” where it scans the temperature and at certain temperature it releases the gate and opens if the temperature is beyond preprogrammed temperature then the gate is kept closed and then this will also scans the visitors who visits the area or who cross the gate so that we can have a count on the visitors. This is based on the current situation covid 19 break out so that we are doing this project which will help the current society. We are using IOT for this project to make it easier to maintain the output values of the system. Where we can save and monitor these values for future purpose.

In this project, Automated covid temperature-based gate opening and visitor counting system which is verified through an IOT based system to monitor its output performance. The performance results are discussed.

TABLE OF CONTENTS

1.Insight into ESP32 Features & Using It with Arduino IDE

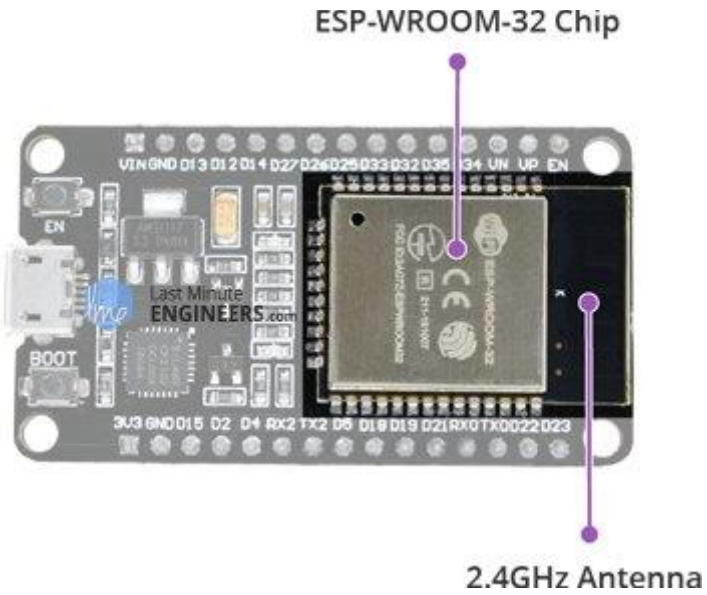
Few years back, ESP8266 took the embedded IoT world by storm. For less than $3, you could get a programmable, WiFi-enabled microcontroller being able to monitor and control things from anywhere in the world.
Now Espressif (The semiconductor company behind the ESP8266) has released a perfect super-charged upgrade: the ESP32. Being successor to ESP8266; not only does it have a WiFi support, but it also features Bluetooth 4.0 (BLE/Bluetooth Smart) – perfect for just about any IoT project.

1.1.ESP-WROOM-32 Module

The development board equips the ESP-WROOM-32 module containing Tensilica Xtensa® Dual-Core 32-bit LX6 microprocessor. This processor is similar to the ESP8266 but has two CPU cores (can be individually controlled), operates at 80 to 240 MHz adjustable clock frequency and performs at up to 600 DMIPS (Dhrystone Million Instructions Per Second).

- ESP-WROOM-32 Chip
- Xtensa® Dual-Core 32-bit LX6

- Upto 240MHz Clock Freq.
- 520kB internal SRAM
- 4MB external flash
- 802.11b/g/n Wi-Fi transceiver
- Bluetooth 4.2/BLE



There's also 448 KB of ROM, 520 KB of SRAM and 4MB of Flash memory (for program and data storage) just enough to cope with the large strings that make up web pages, JSON/XML data, and everything we throw at IoT devices nowadays.

The ESP32 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can not only connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. The ESP32 supports WiFi Direct as well, which is a good option for peer-to-peer connection without the need of an access point. The WiFi Direct is easier to setup and the data transfer speeds are much better than Bluetooth.

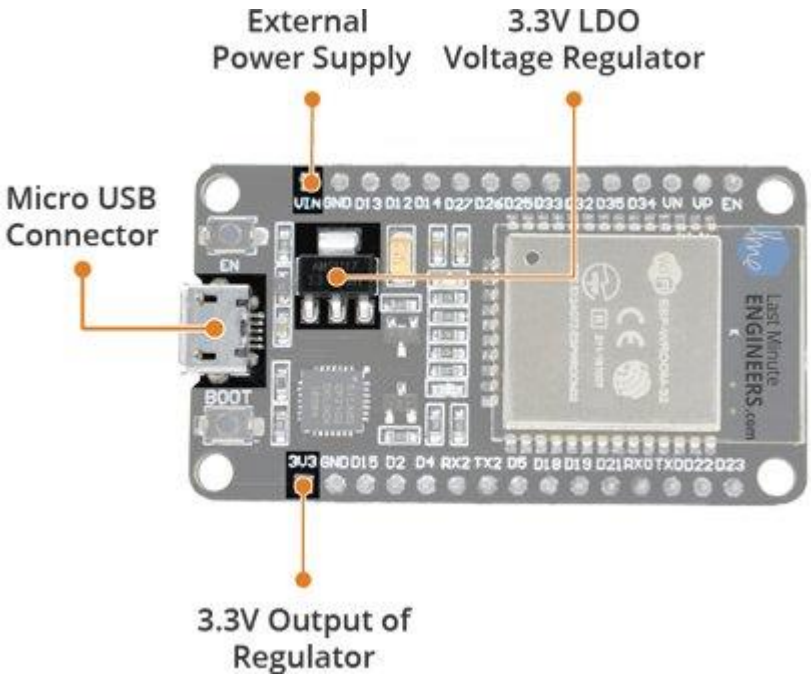
The chip also has dual mode Bluetooth capabilities, meaning it supports both Bluetooth 4.0 (BLE/Bluetooth Smart) and Bluetooth Classic (BT), making it even more versatile.

1.2. Power Requirement

As the operating voltage range of ESP32 is 2.2V to 3.6V, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP32 pulls as much as 250mA during RF transmissions. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

Power Requirement

- Operating Voltage: 2.2V to 3.6V
- On-board 3.3V 600mA regulator
- 5 µA during Sleep Mode
- 250mA during RF transmissions



Power to the ESP32 development board is supplied via the on-board MicroB USB connector. Alternatively, if you have a regulated 5V voltage source, the VIN pin can be used to directly supply the ESP32 and its peripherals.

Also, the sleep current of the ESP32 chip is less than 5 µA, making it suitable for battery powered and wearable electronics applications.

Warning:

The ESP32 requires a 3.3V power supply and 3.3V logic levels for communication. The GPIO pins are not 5V-tolerant! If you want to interface the board with 5V (or higher) components, you'll need to do some level shifting.

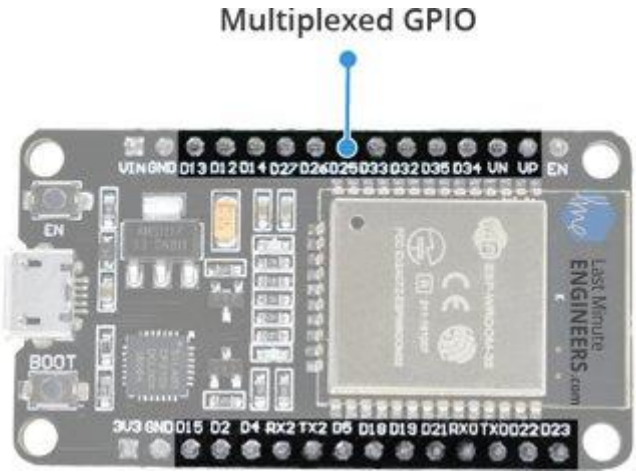
1.3. Peripherals and I/O

Although the ESP32 has total 48 GPIO pins, only 25 of them are broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

- 15 ADC channels – 15 channels of 12-bit SAR ADC's. The ADC range can be set, in firmware, to either 0-1V, 0-1.4V, 0-2V, or 0-4V
- 2 UART interfaces – 2 UART interfaces. One is used to load code serially. They feature flow control, and support IrDA too!
- 25 PWM outputs – 25 channels of PWM pins for dimming LEDs or controlling motors.
- 2 DAC channels – 8-bit DACs to produce true analog voltages.
- SPI, I2C & I2S interface – There are 3 SPI and 1 I2C interfaces to hook up all sorts of sensors and peripherals, plus two I2S interfaces if you want to add sound to your project.
- 9 Touch Pads – 9 GPIOs feature capacitive touch sensing.

Multiplexed I/Os

- 15 ADC channels
- 2 UART interfaces
- 25 PWM outputs
- 2 DAC channels
- SPI, I2C & I2S interface
- 9 Touch Pads



Thanks to the ESP32's pin multiplexing feature (Multiple peripherals multiplexed on a single GPIO pin). Meaning a single GPIO pin can act as an ADC input/DAC output/Touch pad.

1.3.1. Input Only GPIOs

Pin D34, D35, VP and VN cannot be configured as outputs, but they can be used as either digital inputs, analog inputs, or for other unique purposes. Also note that they do not have internal pull-up or pull-down resistors, like the other GPIO pins.

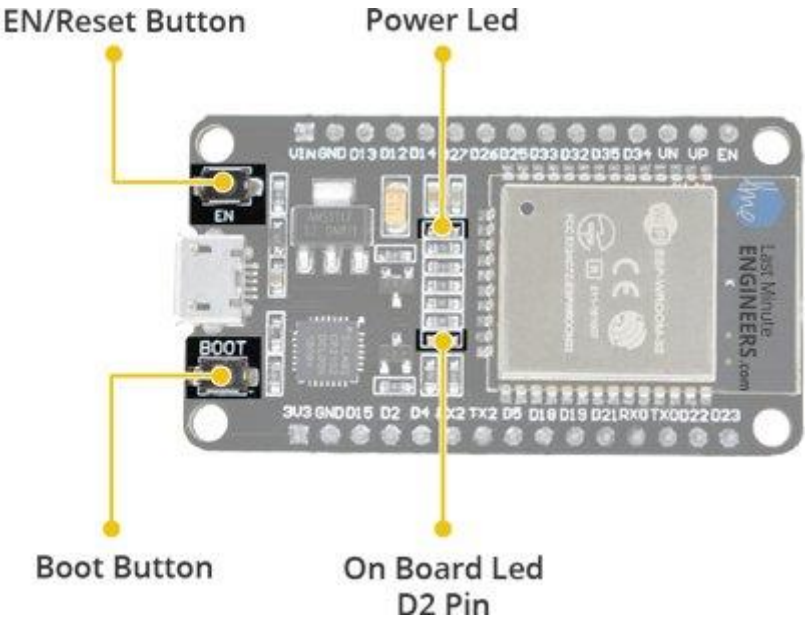
Also GPIO pins VP and VN are an integral part of the ultra-low-noise pre-amplifier for the ADC, which help to configure the sampling time and noise of the pre-amp.

1.4. On-board Switches & LED Indicators

The ESP32 development board features two buttons. One marked as EN located on the top left corner is the Reset button, used of course to reset the ESP32 chip. The other Boot button on the bottom left corner is the download button used while downloading the new sketch/programs.

Switches & Indicators

- EN – Reset the ESP32 chip
- Boot – Download new programs
- Red LED – Power Indicator
- Blue LED – User Programmable



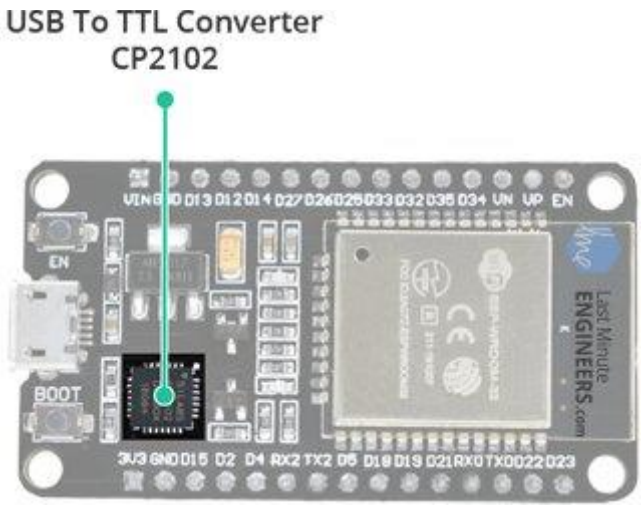
The board also has 2 LED indicators viz. Red LED & Blue LED. A Red LED indicates that the board is powered up and has 3.3V from the regulator. The Blue LED is user programmable and is connected to the D2 pin of the board.

1.5. Serial Communication

The board includes CP2102 USB-to-UART Bridge Controller from Silicon Labs, which converts USB signal to serial and allows your computer to program and communicate with the ESP32 chip.

Serial Communication

- CP2102 USB-to-UART converter
- 5 Mbps communication speed
- IrDA support

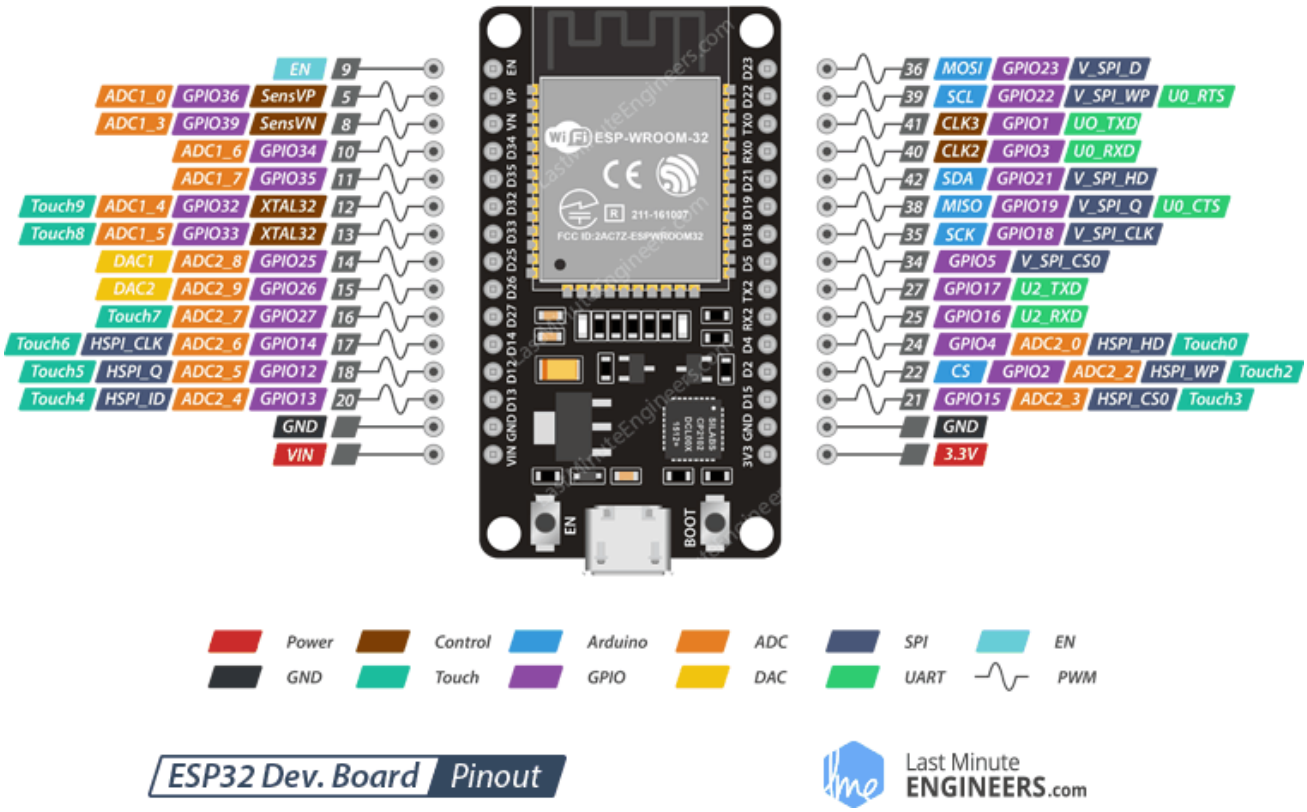


If you have an older version of CP2102 driver installed on your PC, we recommend upgrading now.

CP2102 Driver

1.6. ESP32 Development Board Pinout

The ESP32 development board has total 30 pins that interface it to the outside world. The connections are as follows:



For the sake of simplicity, we will make groups of pins with similar functionalities.

Power Pins There are two power pins viz. VIN pin & 3.3V pin. The VIN pin can be used to directly supply the ESP32 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pin is the output of an on-board voltage regulator. This pin can be used to supply power to external components.
GND is a ground pin of ESP32 development board.
Arduino Pins are nothing but ESP32's hardware I2C and SPI pins to hook up all sorts of sensors and peripherals in your project.

GPIO Pins ESP32 development board has 25 GPIO pins which can be assigned to various functions programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

ADC Channels The board integrates 12-bit SAR ADCs and supports measurements on 15 channels (analog enabled pins). Some of these pins can be used to build a programmable gain amplifier which is used for the measurement of small analog signals. The ESP32 is also designed to measure the voltages while operating in the sleep mode.

DAC Channels The board features two 8-bit DAC channels to convert digital signals into true analog voltages. This dual DAC can drive other circuits.

Touch Pads The board offers 9 capacitive sensing GPIOs which detect capacitive variations introduced by the GPIO's direct contact or close proximity with a finger or other objects.

UART Pins ESP32 development board has 2 UART interfaces, i.e. UART0 and UART2, which provide asynchronous communication (RS232 and RS485) and IrDA support, and communicate at up to 5 Mbps. UART provides hardware management of the CTS and RTS signals and software flow control (XON and XOFF) as well.

SPI Pins SPI Pins ESP32 features three SPIs (SPI, HSPI and VSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

- 4 timing modes of the SPI format transfer
- Up to 80 MHz and the divided clocks of 80 MHz
- Up to 64-Byte FIFO

All SPIs can also be used to connect to the external Flash/SRAM and LCD.

~ PWM Pins The board has 25 channels (Nearly All GPIO pins) of PWM pins controlled by Pulse Width Modulation (PWM) controller. The PWM output can be used for driving digital motors and LEDs. The controller consists of PWM timers and the PWM operator. Each timer provides timing in synchronous or independent form, and each PWM operator generates the waveform for one PWM channel.

EN Pin is used to enable ESP32. The chip is enabled when pulled HIGH. When pulled LOW the chip works at minimum power.

1.7. ESP32 Development Platforms

Now, let's move on to the interesting stuff!

There are a variety of development platforms that can be equipped to program the ESP32. You can go with Espruino – JavaScript SDK and firmware closely emulating Node.js, or use Mongoose OS – An operating system for IoT devices (recommended platform by Espressif Systems and Google Cloud IoT) or use a software development kit (SDK) provided by Espressif or one of the platforms listed on WiKiPedia.

Fortunately, the amazing ESP32 community recently took the IDE selection a step further by creating an Arduino add-on. If you're just getting started programming the ESP32, this is the environment we recommend beginning with, and the one we'll document in this tutorial. Check out the ESP32 Arduino GitHub repository for more information.

1.8. Installing the ESP32 Core On – Windows OS

Espressif's official ESP32 Arduino core is hosted here on GitHub. They don't have an Arduino board manager install yet (Like we do while installing ESP8266 core on Arduino IDE). It should be available soon. Until then, we have to install it manually.

Let's proceed with the installing ESP32 Arduino core.

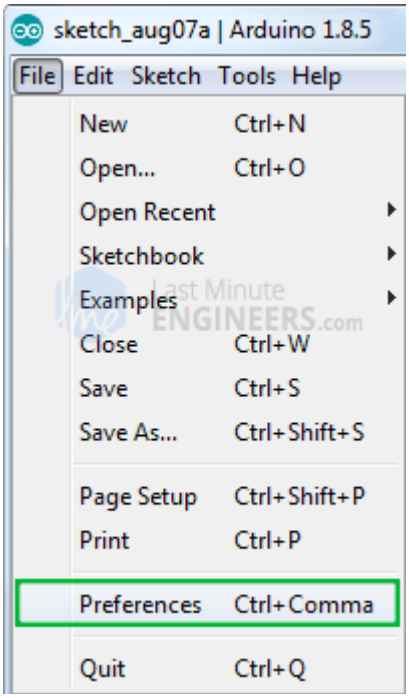
The first thing is having latest Arduino IDE (Arduino 1.8.5 or higher) installed on your PC. If not, we recommend upgrading now.

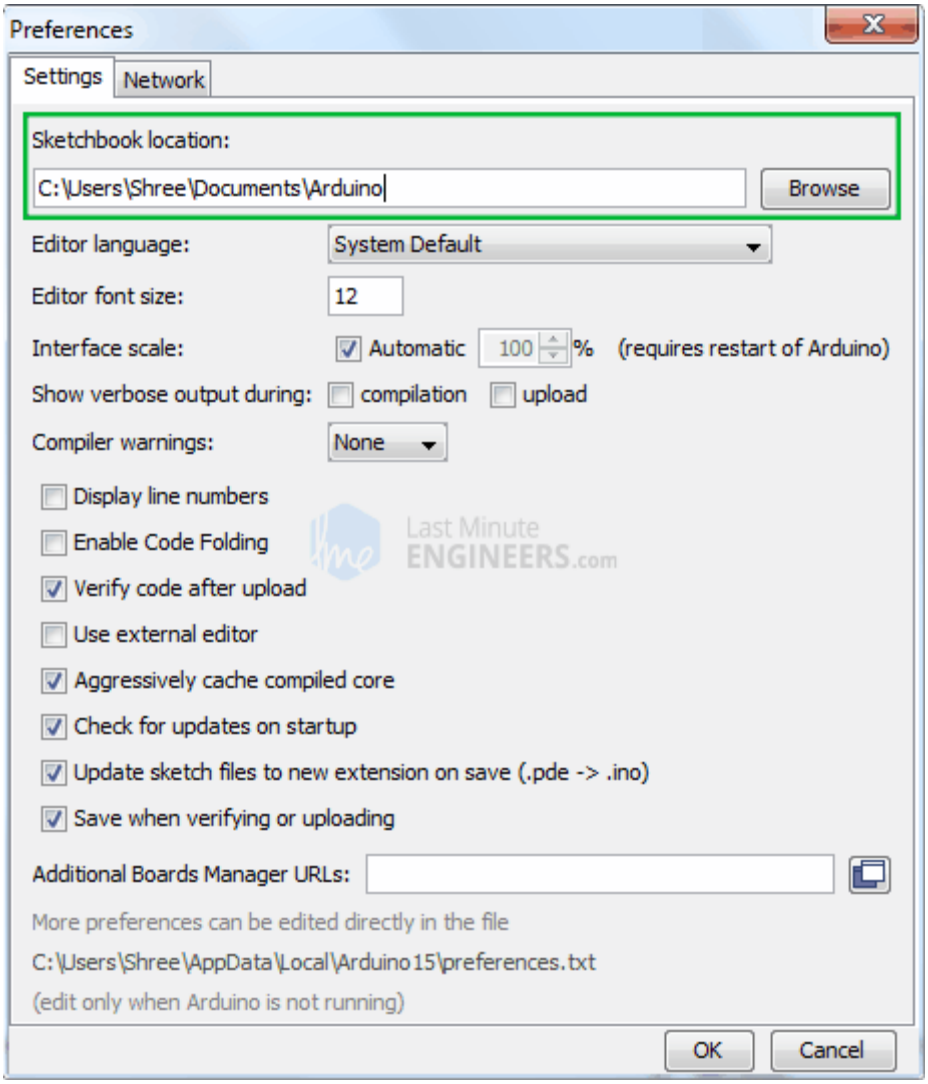
Latest Arduino IDE

Next, you need to download the contents of the esp32-arduino GitHub repository. You can visit GitHub and download it manually or simply click below download button.

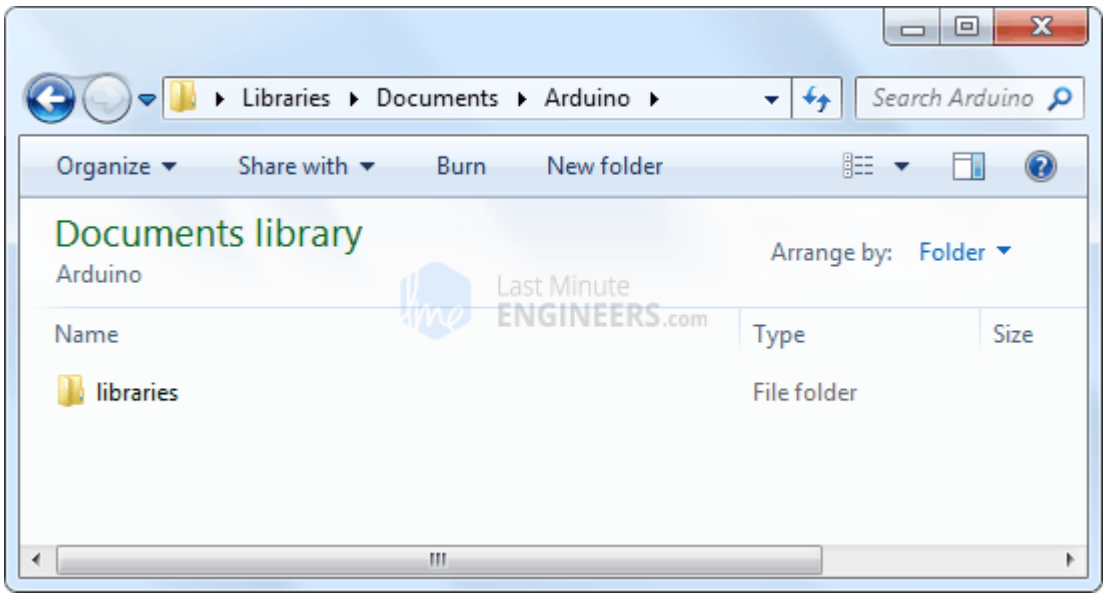
ESP32 Arduino Core

Now go to your Arduino sketchbook directory. It’s by default Arduino directory in My Documents unless otherwise changed. You can verify it by opening Arduino IDE > File > Preferences > Sketchbook Location.



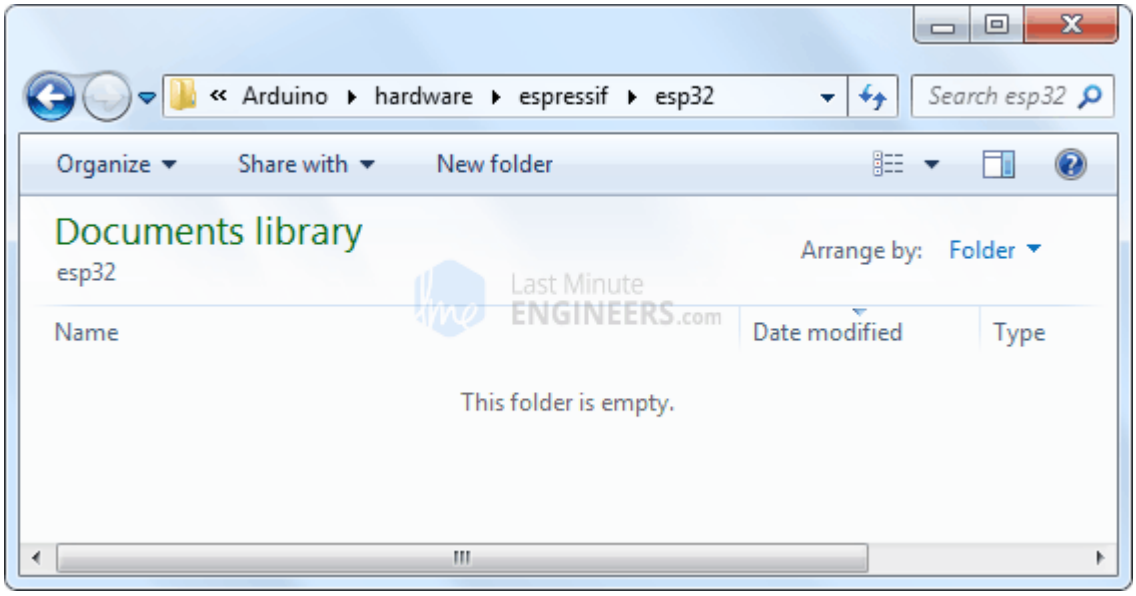


Now considering your sketchbook directory is located in My Documents > Arduino, open the directory. You should see libraries directory inside it.

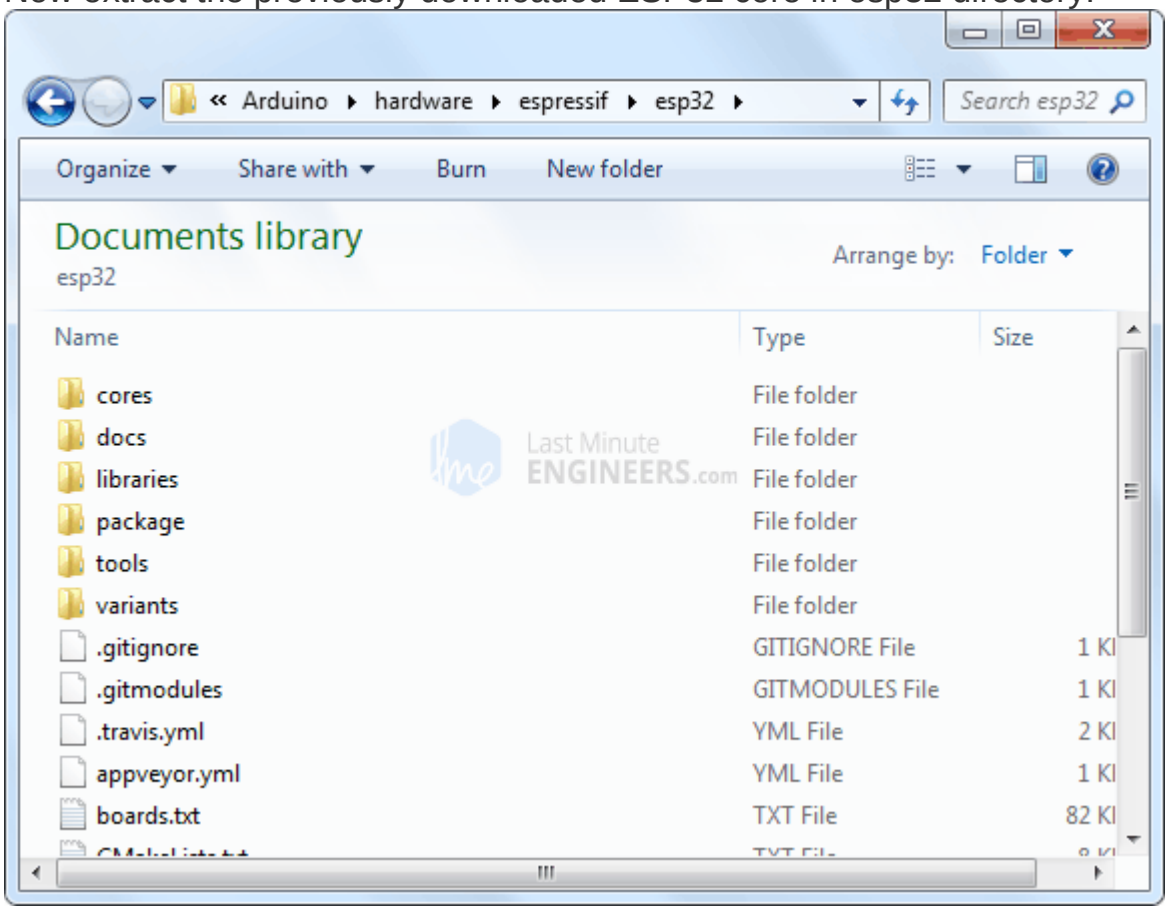


Now create a new directory called hardware. Inside it create another directory called espressif. Inside it create another directory called esp32.

The directory structure should look like My Documents > Arduino > hardware > espressif > esp32

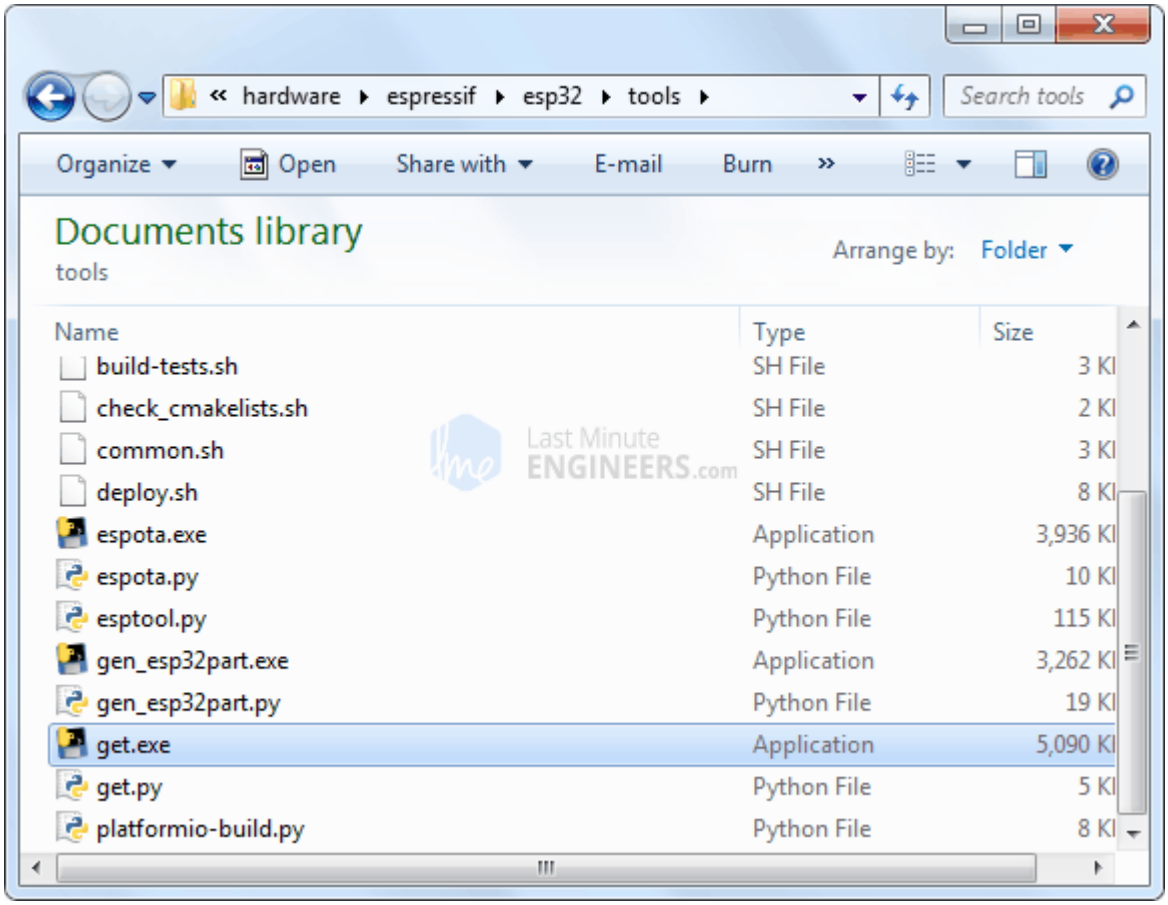


Now extract the previously downloaded ESP32 core in esp32 directory.

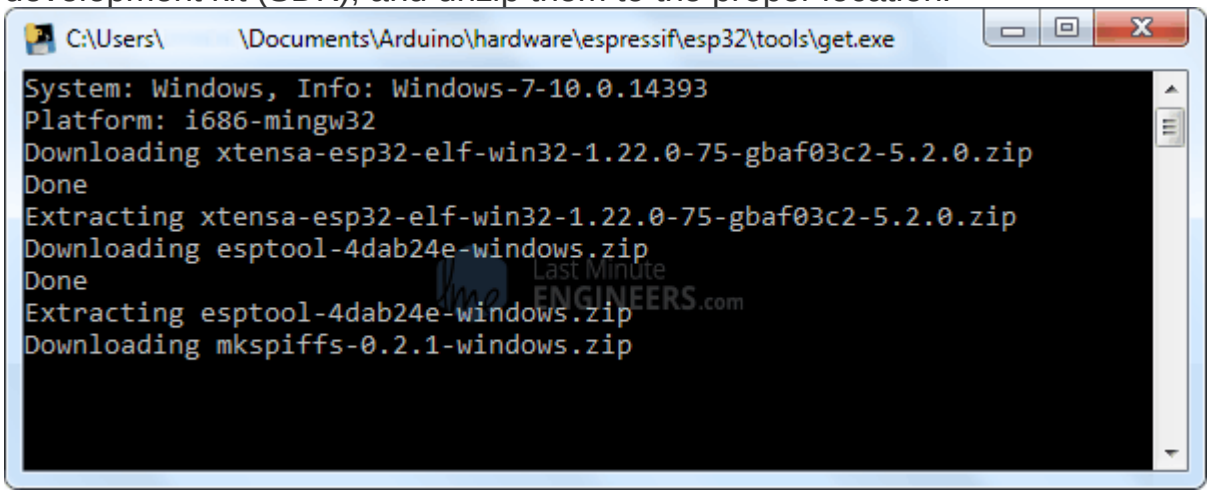


Once you are done, verify that “boards.txt”, “platform.txt”, and the cores, doc, tools, etc. folders are there inside esp32 directory.

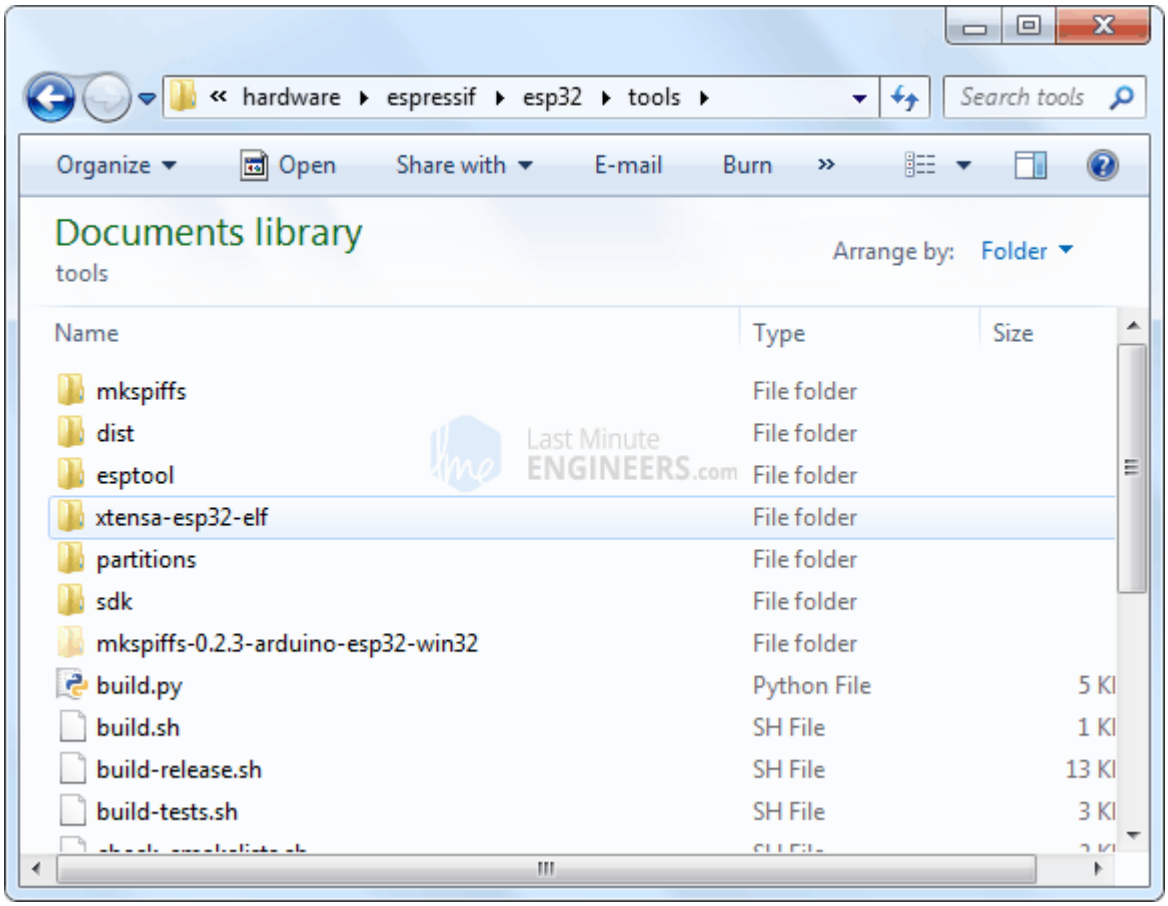
In order to compile code for the ESP32, you need the Xtensa GNU compiler collection (GCC) installed on your machine. Go to esp32 > tools folder and execute get.exe



This executable will download the Xtensa GNU tools and the ESP32 software development kit (SDK), and unzip them to the proper location.



You should see a few new folders in the “tools” directory, including “sdk” and “xtensa-esp32-elf” once it’s done.



1.9. Installing the ESP32 Core On – Mac OS

Espressif’s official ESP32 Arduino core is hosted here on GitHub. They don’t have an Arduino board manager install yet (Like we do while installing ESP8266 core on Arduino IDE). It should be available soon. Until then, we have to install it manually.

Let’s proceed with the installing ESP32 Arduino core.

The first thing is having latest Arduino IDE (Arduino 1.8.5 or higher) installed on your PC. If not, we recommend upgrading now.

Latest Arduino IDE

Next, Open your Terminal and execute the following commands.

```
curl -o get-pip.py https://bootstrap.pypa.io/get-pip.py && \
```

```
sudo python get-pip.py && \
sudo pip install pyserial
mkdir -p ~/Documents/Arduino/hardware/espressif && \
cd ~/Documents/Arduino/hardware/espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32 && \
git submodule update --init --recursive && \
cd tools && \
python get.py
```

Here ~/Documents/Arduino represents your sketch book location as per Arduino IDE > File> Preferences > Sketchbook location. Adjust the command above accordingly if necessary!

If you get the below error, Install the command line dev tools and try above commands again:


```
xcode-select --install
```

1.10. Installing the ESP32 Core On – Debian/Ubuntu Linux OS

Espressif's official ESP32 Arduino core is hosted here on GitHub. They don't have an Arduino board manager install yet (Like we do while installing ESP8266 core on Arduino IDE). It should be available soon. Until then, we have to install it manually.

Let's proceed with the installing ESP32 Arduino core.

The first thing is having latest Arduino IDE (Arduino 1.8.5 or higher) installed on your PC. If not, we recommend upgrading now.

Latest Arduino IDE

Next, Open your Terminal and execute the following commands

```
sudo usermod -a -G dialout $USER && \
sudo apt-get install git && \
wget https://bootstrap.pypa.io/get-pip.py && \
sudo python get-pip.py && \
sudo pip install pyserial && \
mkdir -p ~/Arduino/hardware/espressif && \
cd ~/Arduino/hardware/espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32 && \
git submodule update --init --recursive && \
cd tools && \
python2 get.py
```

If you have Arduino.app installed to /Applications/, modify the installation as follows, beginning at mkdir -p ~/Arduino…:

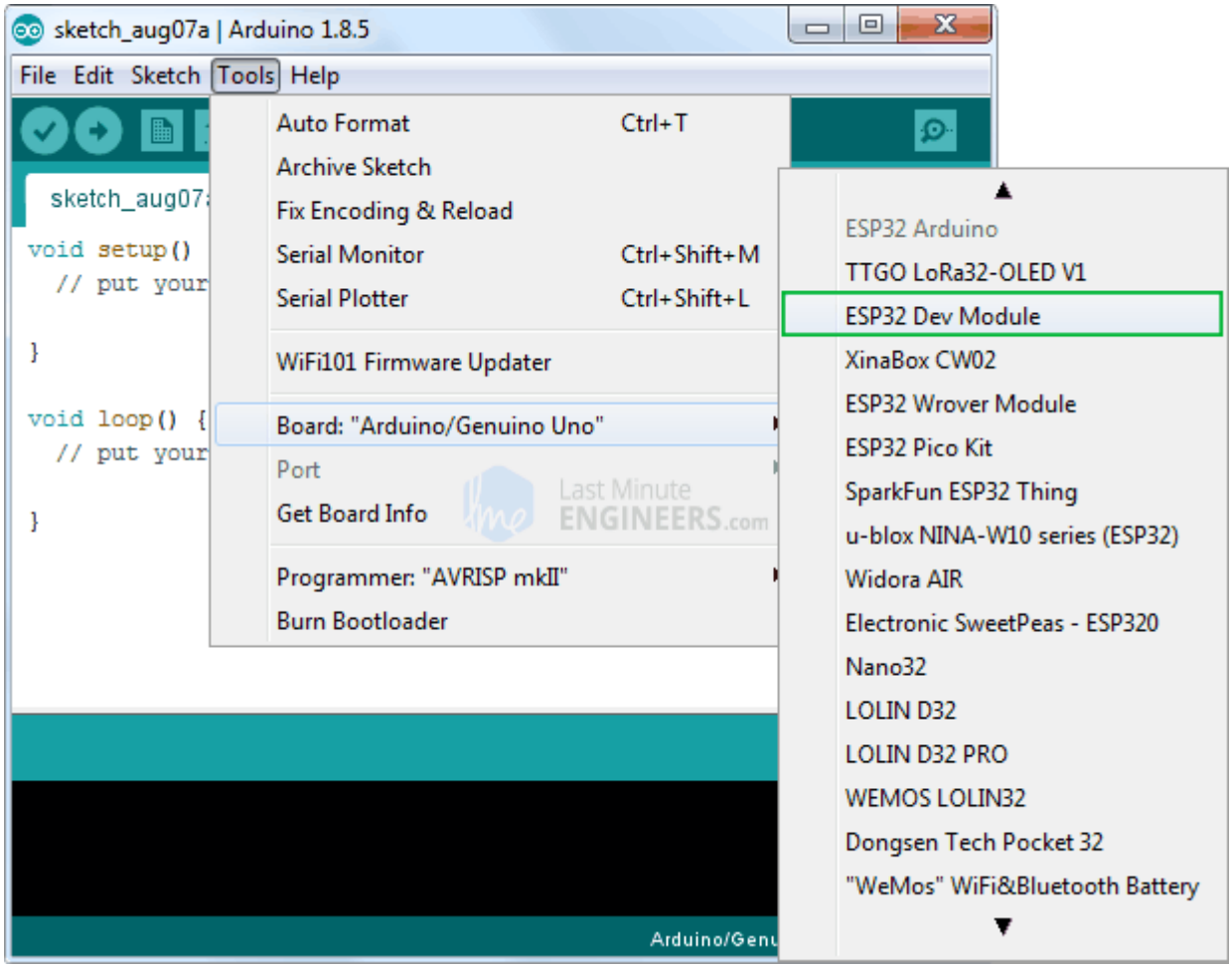
```
cd /Applications/Arduino_*/Contents/java/hardware/
mkdir -p espressif && \
cd espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32 && \
git submodule update --init --recursive && \
cd tools && \
python2 get.py```
```

1.11. Arduino Example: Blink

To make sure ESP32 Arduino core and the ESP32 development board are properly set up, we'll upload the simplest sketch of all – The Blink!

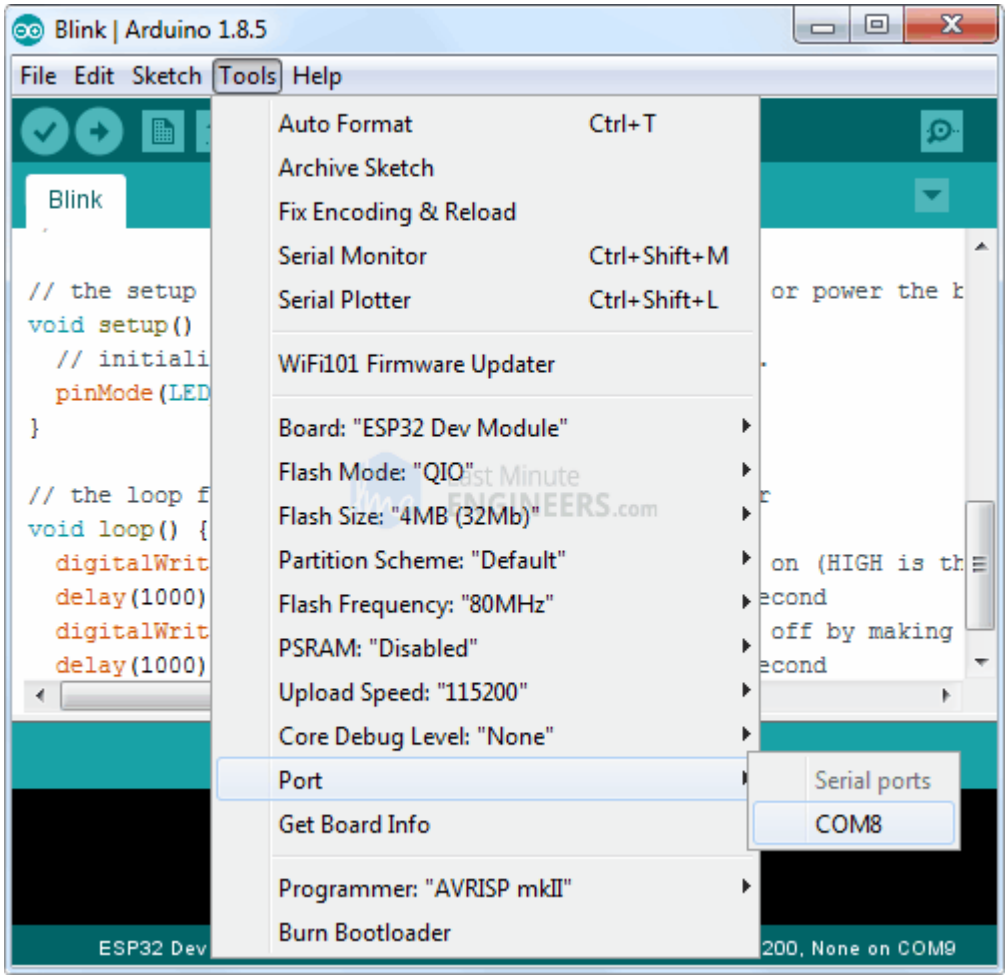
We will use the on-board LED for this test. As mentioned earlier in this tutorial, D2 pin of the board is connected to on-board Blue LED & is user programmable. Perfect!

Before we get to uploading sketch & playing with LED, we need to make sure that the board is selected properly in Arduino IDE. Open Arduino IDE and select ESP32 Dev Module option under your Arduino IDE > Tools > Board menu.



Now, plug your ESP32 development board into your computer via micro-B USB cable. Once the board is plugged in, it should be assigned a unique COM port. On Windows machines, this will be something like COM#, and on Mac/Linux computers it will come in the form of /dev/tty.usbserial-XXXXXX. Select this serial port under the Arduino IDE > Tools > Port menu.

Also the upload speed is selected to 921600 by default. Try lowering it to Upload Speed : 115200 as many users complained about getting espcomm_sync failed error when trying to upload the sketch at 921600 speed.



Once you are done, try the example sketch below.

```
int ledPin = 2;
void setup()
{
        pinMode(ledPin, OUTPUT);
}
void loop()
{
        digitalWrite(ledPin, HIGH);
        delay(500);
        digitalWrite(ledPin, LOW);
        delay(500);
}
```

Once the code is uploaded, LED will start blinking. You may need to tap the EN button to get your ESP32 to begin running the sketch.

2. MLX90614

ESP32 and MLX90614 infrared thermometer.

The MLX90614 is a non-contact infrared thermometer with a measurement range from -70 to +380 degree Celsius. Just connect the four leads to your Wemos and you will have a accurate thermometer with a resolution of 0.01 and a accuracy of 0.5 degrees, or for that matter you can use any microcontroller that can communicate with it through it’s I2C interface.

Being an I2C device you simply need to connect to the SDA, SCL and choose a suitable GND and Vin. I used 3.3v to be safe, although the breakout states 3 to 5v.

This version I chose comes with a breakout board with all of the components needed for operation. Here is a picture of that breakout board

MLX90614

2.1.Features:

Small size, low cost
Mounted on a breakout board with two types of pins
10k Pull up resistors for the I2C interface with optional solder jumpers
Factory calibrated in wide temperature range:
-40 … + 125 ° C for sensor temperature and
-70 … + 380 ° C for object temperature.
High accuracy of 0.5 ° C over wide temperaturerange (0 … + 50 ° C for both Ta and To) High (medical) accuracy calibration
Measurement resolution of 0.02 ° C
Single and dual zone versions
SMBus compatible digital interface
Customizable PWM output for continuous reading
Sleep mode for reduced power consumption

2.2. Connection

As sensor use I2C communication, wiring is simple:

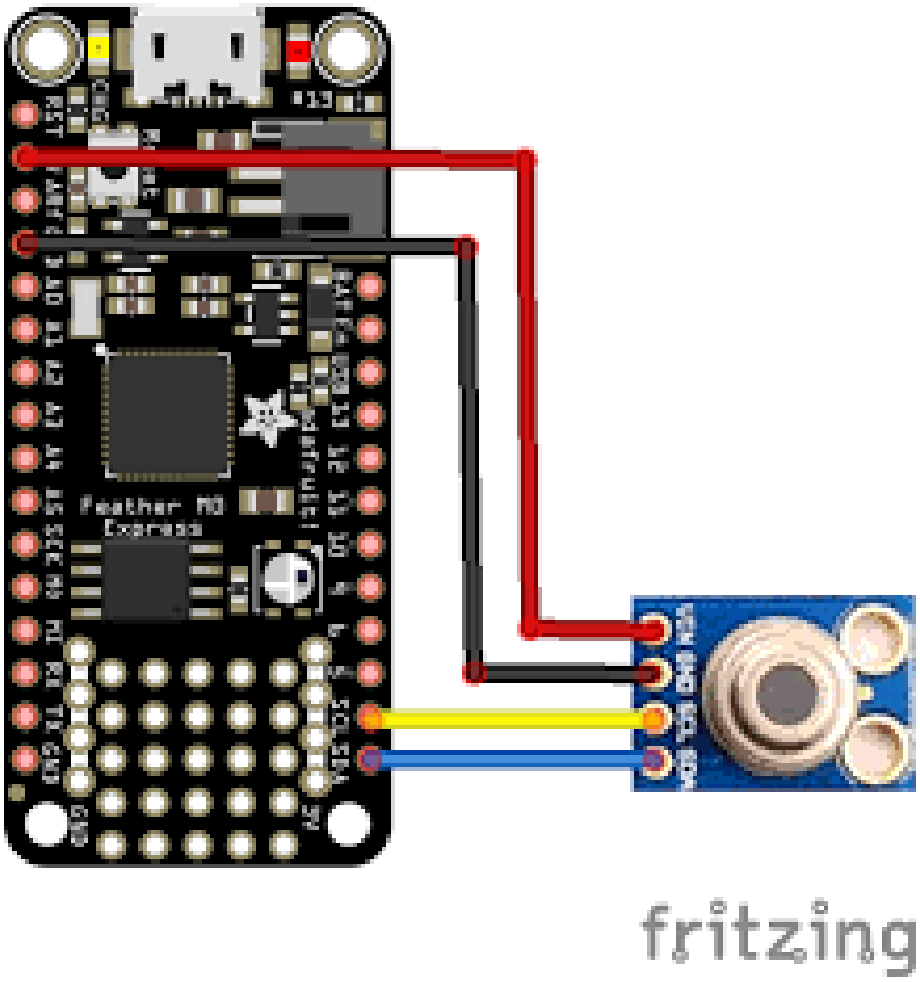
MLX90614 - ESP32

VIN - 3.3V (never connect to 5V, because ESP32 use 3.3V logic!)

GND - GND

SCL - GPIO 22 (marked as D22 on board)

SDA - GPIO 21 (marked as D21 on board)



2.3. Code

There is a library from Adafruit and rather than reinvent the wheel, here is the basic code example. In practice you connect to an LCD, warning LED or perhaps a buzzer to warn if a certain maximum temperature was reached

The sketch below is fairly straightforward, most of the work is done in the Adafruit MLX96014 library which outputs the result via the serial monitor

```
?
1    #include <Wire.h>
2    #include <Adafruit_MLX90614.h>
3
4    Adafruit_MLX90614 mlx = Adafruit_MLX90614();
```

```
5
6   void setup()
7   {
8   Serial.begin(9600);
9   mlx.begin();
10  }
11
12  void loop()
13  {
14  Serial.print("Ambient = ");
15  Serial.print(mlx.readAmbientTempC());
16  Serial.print("*C\tObject = ");
17  Serial.print(mlx.readObjectTempC());
18  Serial.println("*C");
19  Serial.print("Ambient = ");
20  Serial.print(mlx.readAmbientTempF());
21  Serial.print("*F\tObject = ");
22  Serial.print(mlx.readObjectTempF());
23  Serial.println("*F");
24
25  Serial.println();
26  delay(1000);
27  }
```

2.4. Output

Open up the Serial monitor window and you should see something like the following, the interesting one is the object temperature and how it varied

when I placed an object in front of the sensor, the ambient reading stayed the same

Ambient = 22.13*C Object = 46.25*C
Ambient = 22.13*C Object = 46.25*C

Ambient = 71.83*F Object = 115.25*F
Ambient = 22.91*C Object = 68.71*C

Ambient = 73.24*F Object = 155.68*F
Ambient = 23.89*C Object = 64.25*C

Ambient = 75.00*F Object = 147.65*F
Ambient = 24.79*C Object = 58.79*C

Ambient = 76.62*F Object = 137.82*F
Ambient = 25.45*C Object = 54.07*C

Ambient = 77.81*F Object = 129.33*F
Ambient = 26.09*C Object = 50.89*C

3.Servo Motor

3.1.How Servo Motor Works & Interface It With Arduino

Want to add motion to your next Arduino project without building a motor controller? Then servo motors might be the solid launching point for you.

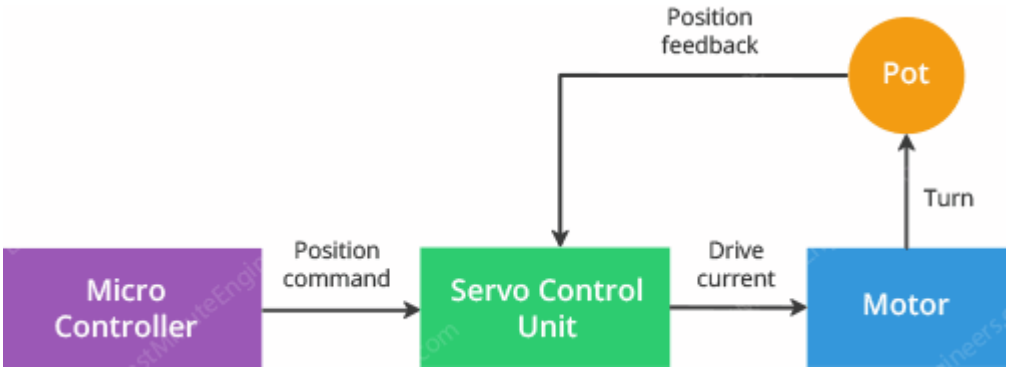
Unlike DC motors, you can precisely control the positioning of these motors. Instruct them where to point, and they'll do it for you.

They're useful in many robotics projects, such as for turning the front wheels on an RC model for steering or pivoting a sensor to look around on a robotic vehicle.

3.2.What is Servo?

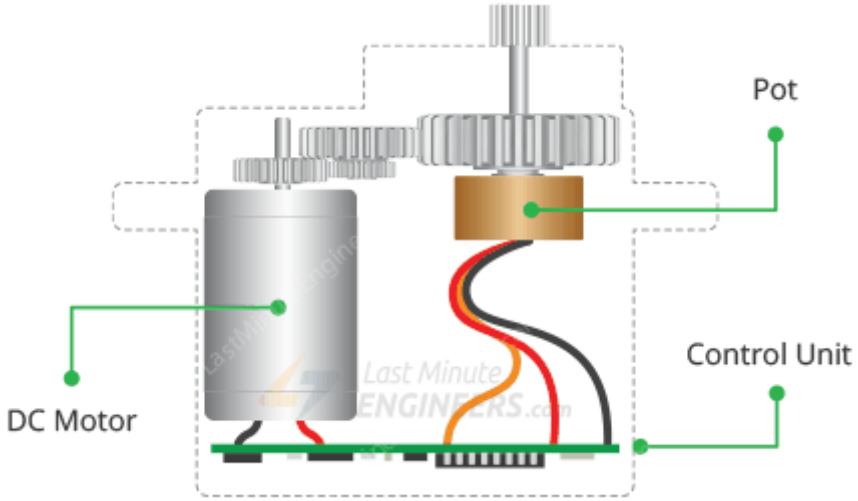
Servo is a general term for a closed loop control system.

A closed loop system uses the feedback signal to adjust the speed and direction of the motor to achieve the desired result.



RC servo motor works on the same principal. It contains a small DC motor connected to the output shaft through the gears.

The output shaft drives a servo arm and is also connected to a potentiometer (pot).



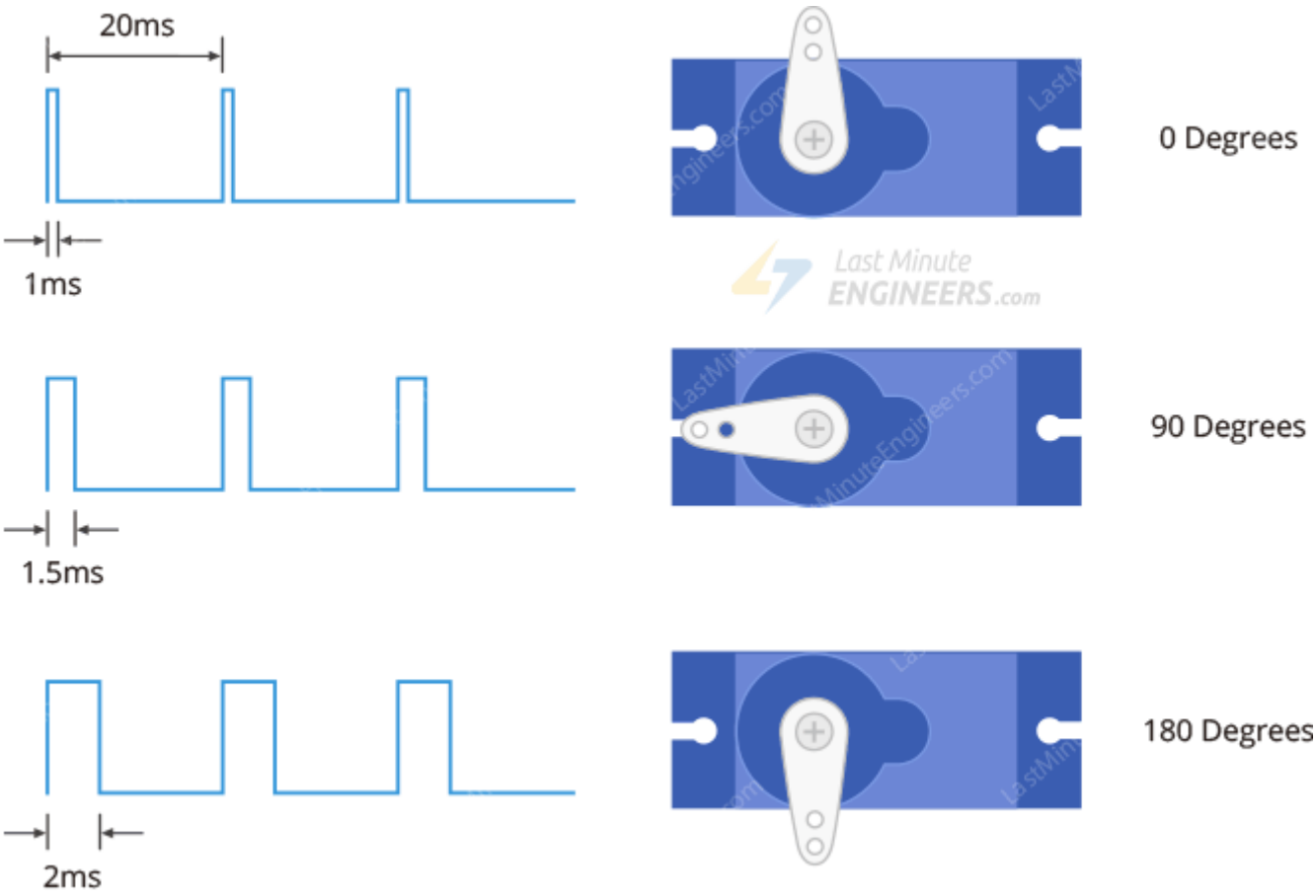
The potentiometer provides position feedback to the servo control unit where the current position of the motor is compared to the target position.

According to the error, the control unit corrects the actual position of the motor so that it matches the target position.

3.3.How Servo Motors Work?

You can control the servo motor by sending a series of pulses to the signal line. A conventional analog servo motor expects to receive a pulse roughly every 20 milliseconds (i.e. signal should be 50Hz).

The length of the pulse determines the position of the servo motor.



If the pulse is high for 1ms, then the servo angle will be zero.

If the pulse is high for 1.5ms, then the servo will be at its center position.

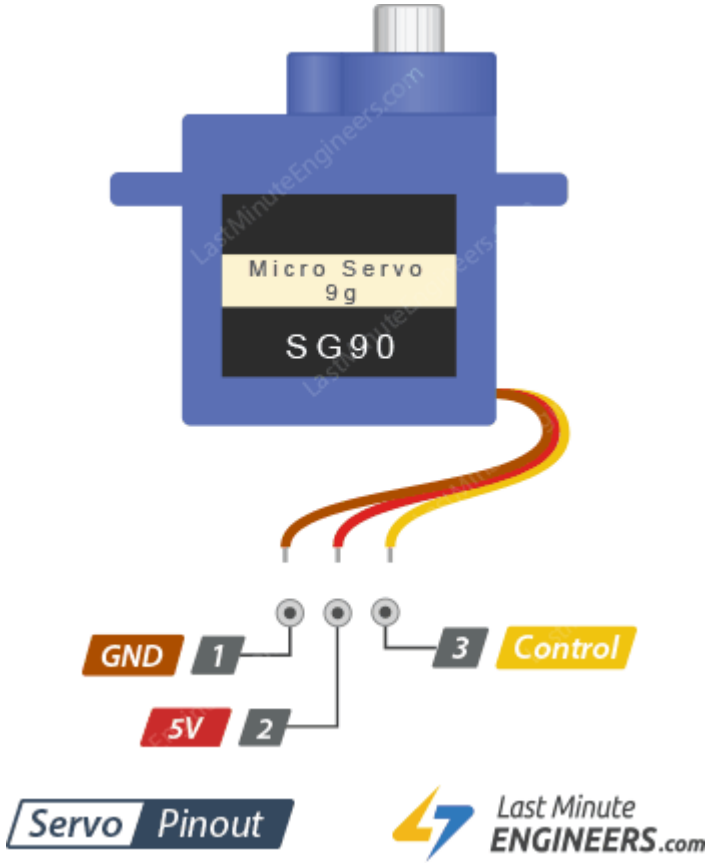
If the pulse is high for 2ms, then the servo will at 180 degrees.

Pulses ranging between 1ms and 2ms will move the servo shaft through the full 180 degrees of its travel.

The duration of the pulses may sometimes vary with different brands and they can be 0.5ms for 0 degrees and 2.5ms for 180 degrees.

3.4. Servo Motor Pinout

Servo motors typically have three connections and are as follows:



GND is a common ground for both the motor and logic.

5V is a positive voltage that powers the servo.

Control is input for the control system.

The color of the wires varies between servo motors, but the red wire is always 5V and GND will either be black or brown. The control wire is usually orange or yellow.

Wiring Servo Motor to Arduino UNO

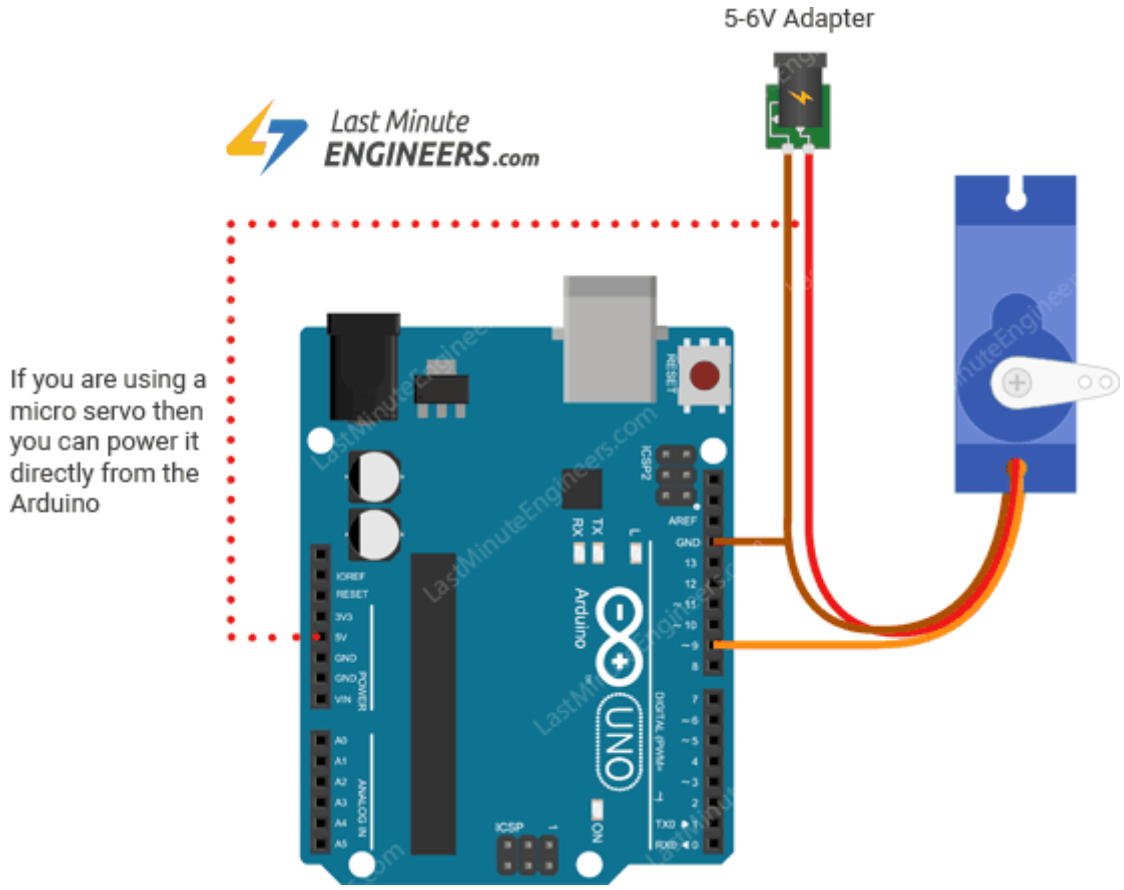
Let’s hook the servo motor up to the Arduino.

For example let’s use SG90 Micro Servo Motor. It runs on 4.8-6VDC (5V Typical) and can rotate approximately 180 degrees (90 in each direction).

It consumes around 10mA at idle and 100mA to 250mA when moving, so we can power it up through 5-volt output on the Arduino.

If you have a servo that consumes more than 250mA, consider using a separate power supply for your servo.

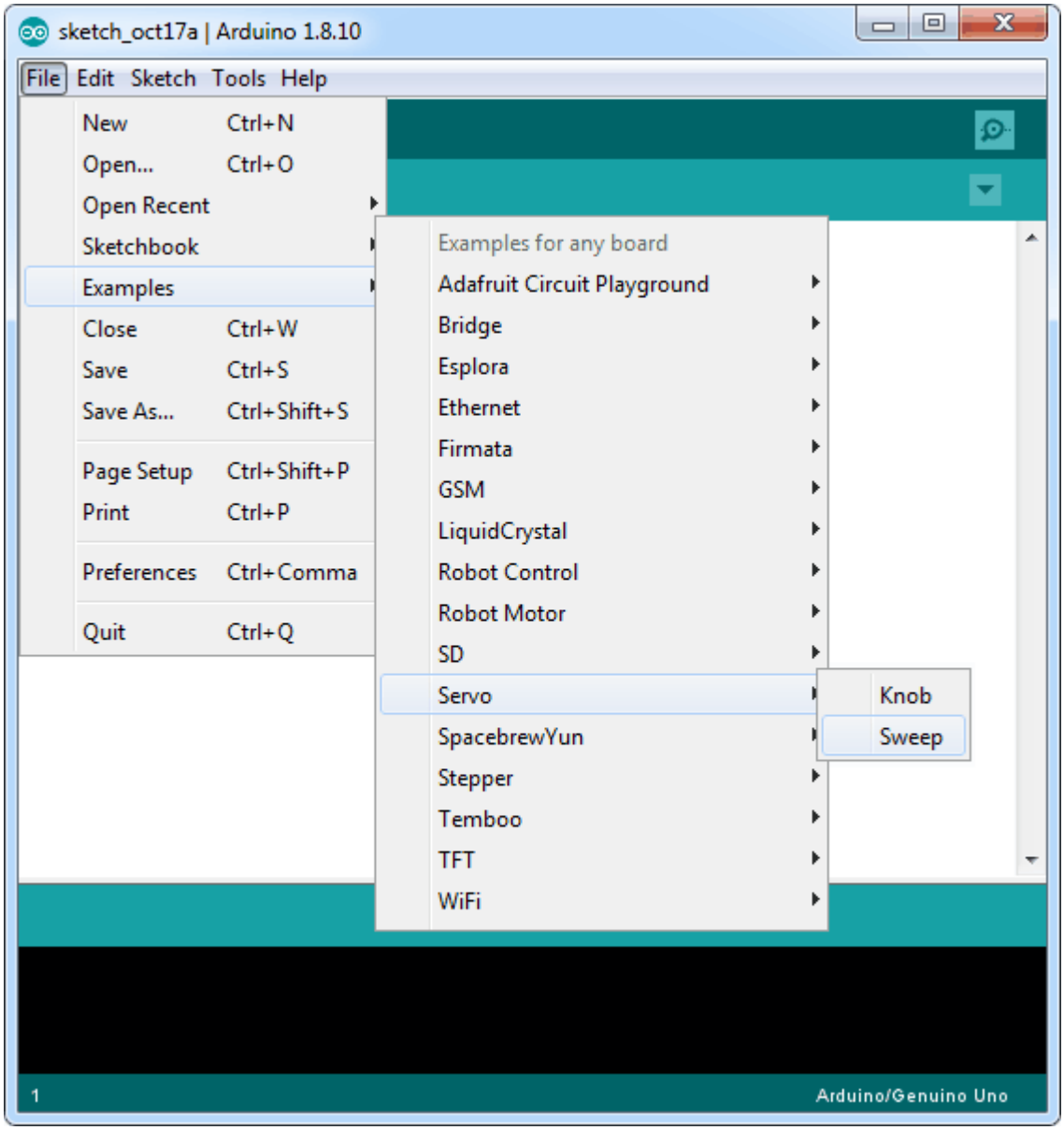
Connect the Red wire to the 5V on Arduino (or DC jack) and Black/Brown wire to ground. Finally connect the Orange/Yellow wire to the PWM enabled pin 9.



3.5.Arduino Code – Sweep

For our first Arduino sketch, we will use one of the built-in examples that come with the Arduino IDE.

Go to the Examples sub-menu. Select the Servo and Load the Sweep sketch.



Go ahead and upload the sketch. You will immediately see the motor moving in one direction and then going back in another.

```
#include <Servo.h>


int servoPin = 9;

Servo servo;

int angle = 0;  // servo position in degrees


void setup() {
```

```
  servo.attach(servoPin);
}

void loop() {

  // scan from 0 to 180 degrees
  for(angle = 0; angle < 180; angle++) {
    servo.write(angle);
    delay(15);
  }

  // now scan back from 180 to 0 degrees
  for(angle = 180; angle > 0; angle--) {
    servo.write(angle);
    delay(15);
  }
}
```

3.6.Explanation:

Controlling servos is not an easy task, but luckily for us, Arduino IDE already contains a very nice library called Servo. It includes simple commands so that you can quickly instruct the servo to turn to a particular angle.

If you are going to use these commands, you need to tell the Arduino IDE that you are using the library with this command:

#include <Servo.h>

The next thing we do is declare the Arduino pin to which the control pin of the servo motor is connected.

int servoPin = 9;

Below line creates a servo object.

Servo servo;

You can actually define up to eight servos in this way, for example, if we had two servos, then we could write something like this:

Servo servo1;

Servo servo2;

The variable angle is used to store the current angle of the servo in degrees.

int angle = 0;

In the setup function, we link the servo object to the pin that will control the servo using this command:

servo.attach(servoPin);

The loop function actually contains two for loops. The first loop increases the angle in one direction and the second in the opposite direction.

Below command tells the servo to update its position to the specified angle.

servo.write(angle);

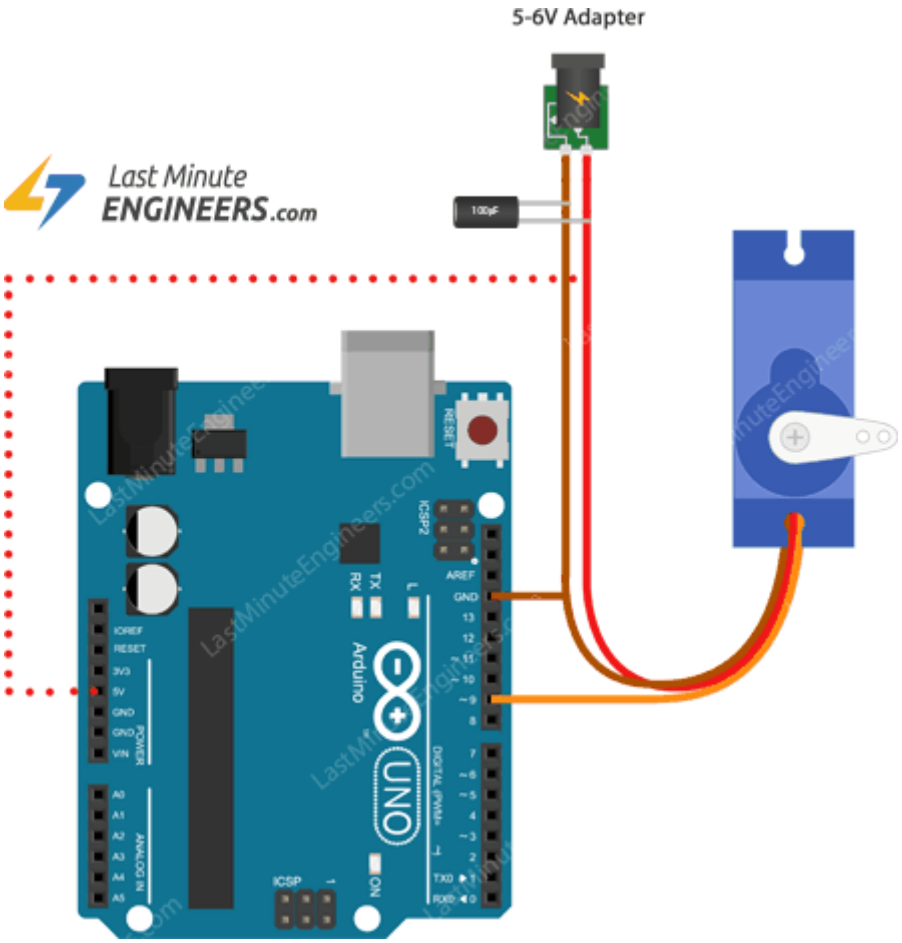
Troubleshooting

Sometimes your servo may misbehave if you decide to run it directly from the Arduino. The reason for this is that the servo draws considerable power, especially during start-up, and this can cause the Arduino board to reset.

If this happens, you can usually fix this by placing a fairly large electrolytic capacitor (470uF – 1000uF) between GND and 5V.

The capacitor acts as a reservoir of electricity, so that when the motor starts, it takes charge from the capacitor as well as the Arduino supply.

The longer lead of the capacitor should be connected to 5V and the negative lead to GND.



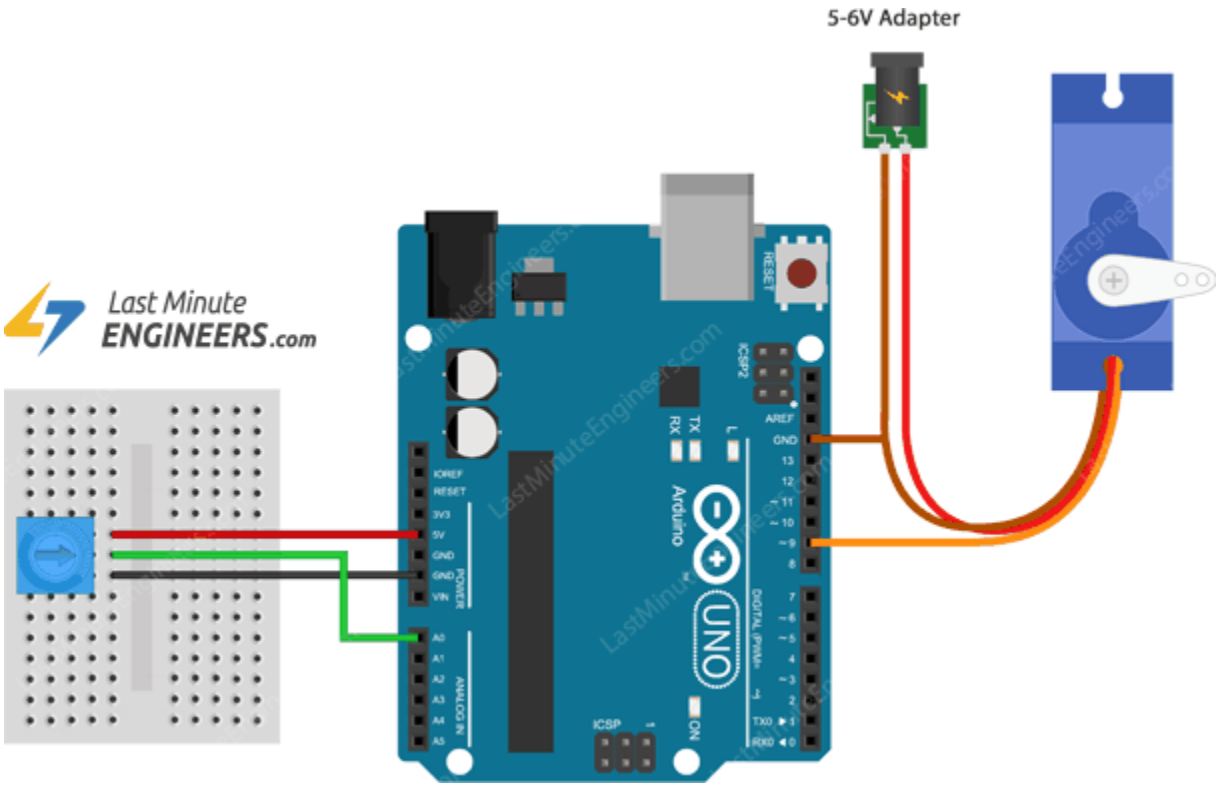
Controlling Servo with a Potentiometer

Our next step is to add a potentiometer so that we can control the position of the servo by turning the knob.

This project can be very useful when you want to control the pan and tilt of a sensor connected to the servo.

3.7.Wiring

As the wiring diagram shows you'll need a potentiometer, any value from 10k up will be OK. Connect one end of the pot to ground, the other end to the Arduino 5V and the wiper to analog input A0.



Arduino Code

The code to make the servo follow the knob's position is simpler than to make it sweep.

```
#include <Servo.h>


int potPin = 0;

int servoPin = 9;

Servo servo;


void setup() {

  servo.attach(servoPin);

}


void loop() {

  int reading = analogRead(potPin);
```

```
  int angle = map(reading, 0, 1023, 0, 180);

  servo.write(angle);

}
```

Notice that there is now a new variable called potPin.

In the loop function, we start by reading the value from the analog pin A0.

int reading = analogRead(potPin);

This gives us a value of between 0 and 1023. But we need to scale it down, since the servo can only rotate through 180 degrees.

One way to do this is to use the Arduino map() Function which re-maps a number from one range to another. So, below line changes the reading to represent the angle between 0 and 180 degrees.

int angle = map(reading, 0, 1023, 0, 180);

Finally, we use the write() command that tells the servo to update its position to the angle selected by the potentiometer.

4.Active Buzzer

The active buzzer will only generate sound when it will be electrified. It generates sound at only one **frequency**. This buzzer operates at an audible frequency of 2 KHz.

Specifications

The specifications of the active buzzer are as follows

- It operates at a voltage range of 3.3 – 5V
- It operates at a frequency of near 2 KHz
- It has small size: 3.3 x 1.3 x 1.2 cm

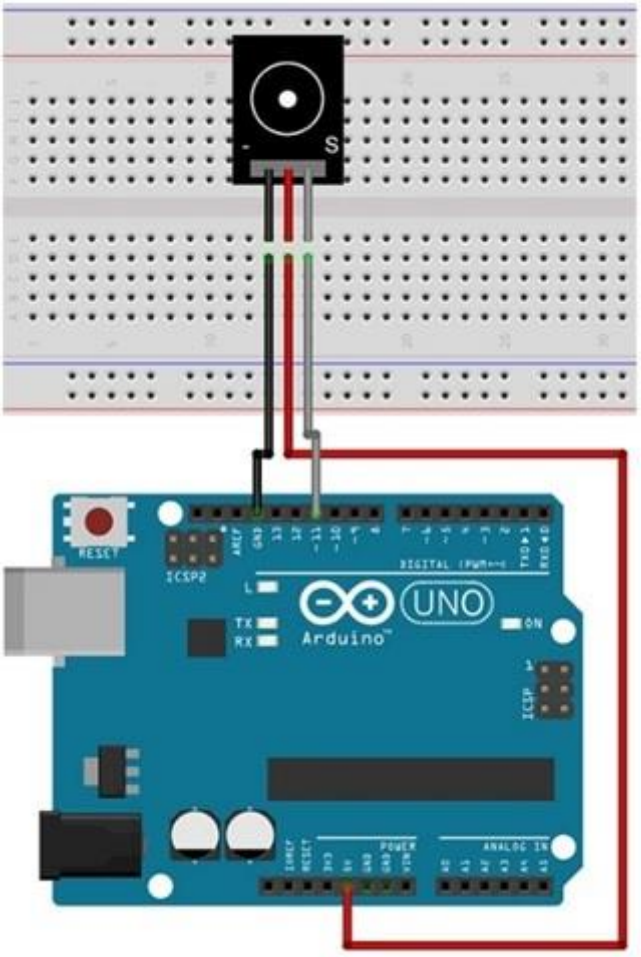
4.1. Pin Out

The module has only three pins. The pin out from left to right is as follows

S: Signal pin
5V: input pin
Ground: Ground pin

active buzzer interfacing with Arduino

The connection scheme is very easier. Just connect the wires as shown in the figure.



Code :

```
// This code is for testing the active buzzer


int buzzerPin = 11;                                  // initializing the buzzer pin
at pin 11 of Arduino


void setup() {                                       // Code written in it will
only run once


  pinMode(buzzerPin, OUTPUT);                        // This will set the pin 11 as
output


  beep(50);                                          // This will make a beep
sound Beep


  beep(50);


  delay(1000);                                       //Adding a delay of 1 sec.


}
```

```
void loop() {                                   // Code written in it will
run continuously


  beep(50);                                     // This will make a beep
sound after every 500 milliseconds


  delay(1000);                                  // Adding a delay of one
second.


}


void beep(unsigned char delayms) {              // Created a function for beep


  analogWrite(buzzerPin, 20);                   // This will set pin 11 to high


  delay(delayms);                               // Giving a delay


  analogWrite(buzzerPin ,0);                    // This will set pin 11 to LOW


  delay(delayms);                               // Giving a delay
```

}

4.2. Passive Buzzer

The passive buzzers need a sound signal to generate a tone. This can be done by either giving a PWM signal to the buzzer or by turning it ON and OFF at different frequencies. It can generate a range of sound signals depending on the input frequency. It can generate tones of frequencies between 1.5 to 2.5 KHz.

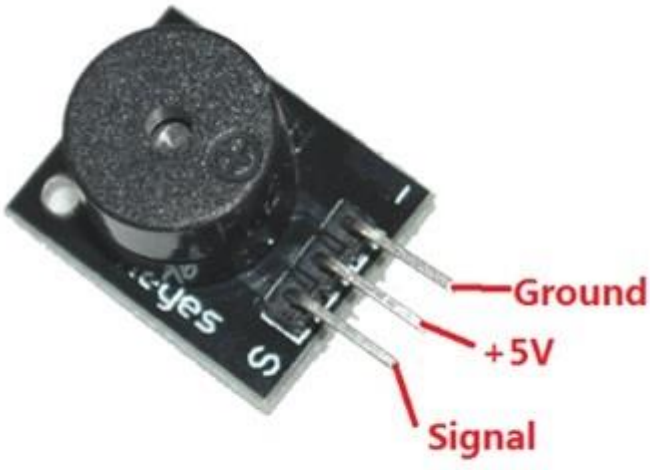
Specification

The specifications of the passive buzzer are as follows.

Operating Voltage	1.5 – 15V DC
Tone Generating Range	1.5 – 2.5 KHz
Dimensions [0.728in X 0.591in]	18.5mm X 15mm

Pin Out

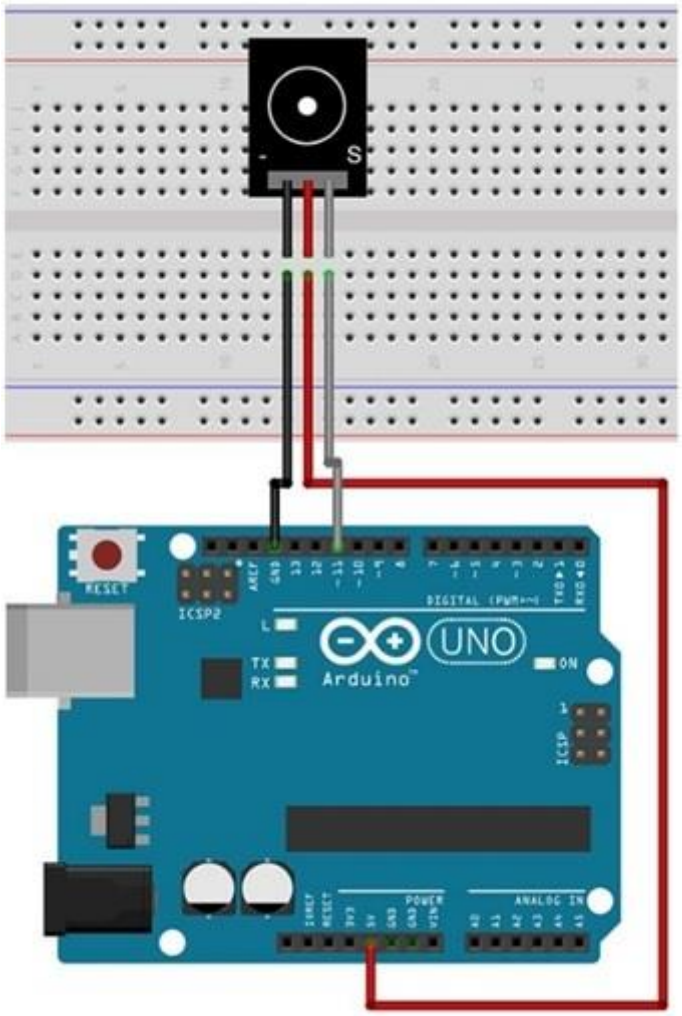
The module has only three pins. The pin out from left to right is as follows



S: Signal pin
5V: input pin
Ground: Ground pin

4.3. Passive Buzzer interfacing with Arduino

The connection scheme is very easier. Just connect the wires as shown in the figure.



4.4. Code

// This code is for testing the passive buzzer

int out =11; // initializing pin 11 as buzzer pin

```
void setup ()                                   // Code written in
it will only run once.

 {

   pinMode(out, OUTPUT);                        // Setting pin 11 as
output pin

 }



void loop ()                                    // Code written in
it will run repeatedly

 {

    unsigned  char  i,  j  ;                    //  Declaring
variables

   while (1){

     for (i = 0; i <80; i++){                   // 100 cycles of
sound

      digitalWrite (out, HIGH);                 // This will turn the
buzzer ON

       delay (1) ;                              // Giving a Delay
of 1ms will set frequency 1

      digitalWrite (out, LOW);                  // This will turn the
buzzer OFF

       delay (1) ;                              // Giving a delay
ms

    }
```

```
    for (i = 0; i <100; i++){                                   // 100 cycles of
sound

     digitalWrite (out, HIGH);                                  // This will turn the
buzzer ON

      delay (2) ;                                               // Giving a delay
of 2ms will set frequency 2

     digitalWrite (out, LOW);                                   // This will turn the
buzzer OFF

      delay (2) ;                                               // Giving a delay
of 2ms

    }

  }

 }
```

4.4.1Playing a Melody using passive buzzer

In this example, we will play a melody using the passive buzzer. We will use the capability of Arduino to produce the pwm signal through which the buzzer will generate tone at different frequencies. The connections are same as we did for the passive buzzer. Just upload the code and the buzzer will play a melody.

4.4.2. Melody game Code

```
// This code is for playing a melody



int buzzerpin = 11;

 int DEBUG = 1;
```

```
 void setup() {
   pinMode(buzzerpin, OUTPUT);                                          //
Setting pin 11 as output
   if (DEBUG) {
     Serial.begin(9600);                                                // Setting
baud rate at 9600
   }
 }


 int melody[] = {  C,  b,  g,  C,  b,   e,  R,  C,  c,  g, a, C };          // initializing
variables for playing melody
 int beats[]  = { 16, 16, 16,  8,  8,  16, 32, 16, 16, 16, 8, 8 };    //  initializing  the
beat values
 int MAX_COUNT = sizeof(melody)


 long tempo = 10000;                                                  //  This  will  Set
overall tempo

 int pause = 1000;                                                  // initializing the variable
for pause between tones

 int rest_count = 100; //<-BLETCHEROUS HACK; See NOTES

 // Initialize core variables
 int tone_ = 0;
 int beat = 0;
 long duration  = 0;

 // PLAY TONE  ==========================================
 // Pulse the speaker to play a tone for a particular duration
 void playTone() {
   long elapsed_time = 0;
   if (tone_ > 0) { // if this isn't a Rest beat, while the tone has
     //  played less long than 'duration', pulse speaker HIGH and LOW
     while (elapsed_time < duration) {

       digitalWrite(buzzerpin,HIGH);
       delayMicroseconds(tone_ / 2);
```

```
      // DOWN
      digitalWrite(buzzerpin, LOW);
      delayMicroseconds(tone_ / 2);

      // Keep track of how long we pulsed
      elapsed_time += (tone_);
    }
  }
  else { // Rest beat; loop times delay
    for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
      delayMicroseconds(duration);
    }
  }
}

// LET THE WILD RUMPUS BEGIN =============================
void loop() {
  // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    tone_ = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A pause between notes...
    delayMicroseconds(pause);

    if (DEBUG) { // If debugging, report loop, tone, beat, and duration
      Serial.print(i);
      Serial.print(":");
      Serial.print(beat);
      Serial.print(" ");
      Serial.print(tone_);
      Serial.print(" ");
      Serial.println(duration);
    }
  }
}
```

IR sensors

5. IR SENSORS

An infrared sensor is an electronic device, that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measure only infrared radiation, rather than emitting it that is called a passive IR sensor. Usually, in the infrared spectrum, all the objects radiate some form of thermal radiation. These types of radiations are invisible to our eyes, which can be detected by an infrared sensor. The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode that is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, the resistances and the output voltages will change in proportion to the magnitude of the IR light received.

5.1. Working Principle

The working principle of an infrared sensor is similar to the object detection sensor. This sensor includes an IR LED & an IR Photodiode, so by combining these two can be formed as a photo-coupler otherwise optocoupler. The physics laws used in this sensor are planks radiation, Stephan Boltzmann & weins displacement.

IR LED is one kind of transmitter that emits IR radiations. This LED looks similar to a standard LED and the radiation which is generated by this is not visible to the human eye. Infrared receivers mainly detect the radiation using an infrared transmitter. These infrared receivers are available in photodiodes form. IR Photodiodes are dissimilar as compared with usual photodiodes because they detect simply IR radiation. Different kinds of infrared receivers mainly exist depending on the voltage, wavelength, package, etc.

Once it is used as the combination of an IR transmitter & receiver, then the receiver's wavelength must equal the transmitter. Here, the transmitter is IR LED whereas the receiver is IR photodiode. The infrared photodiode is responsive to the infrared light that is generated

through an infrared LED. The resistance of photo-diode & the change in output voltage is in proportion to the infrared light obtained. This is the IR sensor's fundamental working principle.

Once the infrared transmitter generates emission, then it arrives at the object & some of the emission will reflect back toward the infrared receiver. The sensor output can be decided by the IR receiver depending on the intensity of the response.

5.2. Types of Infrared Sensor

Infrared sensors are classified into two types like active IR sensor and passive IR sensor.

5.2.1. Active IR Sensor

This active infrared sensor includes both the transmitter as well as the receiver. In most of the applications, the light-emitting diode is used as a source. LED is used as a non-imaging infrared sensor whereas the laser diode is used as an imaging infrared sensor.

These sensors work through energy radiation, received & detected through radiation. Further, it can be processed by using the signal processor to fetch the necessary information. The best examples of this active infrared sensor are reflectance and break beam sensor.

5.2.2. Passive IR Sensor

The passive infrared sensor includes detectors only but they don't include a transmitter. These sensors use an object like a transmitter or IR source. This object emits energy and detects through infrared receivers. After that, a signal processor is used to understand the signal to obtain the required information.

The best examples of this sensor are pyroelectric detector, bolometer, thermocouple-thermopile, etc. These sensors are classified into two types like thermal IR sensor and quantum IR sensor. The thermal IR sensor doesn't depend on wavelength. The energy source used by these sensors is heated. Thermal detectors are slow with their response and detection time. The quantum IR sensor depends on the wavelength and these sensors include high response and detection time. These sensors need regular cooling for specific measurements.

5.2.3. Circuit Working

Once the infrared LED is detected, then the reflected light from the thing will activate a small current that will supply throughout the IR LED detector. This will activate the NPN transistor & the PNP; therefore the LED will switch ON. This circuit is applicable for making different projects like automatic lamps to activate once a person approaches close to the light.

5.3. Hardware Interfacing

IR Sensor have four pins:

1. VCC +5V

2. GND

3. D connects with any digital pin of Arduino when IR pair use as Digital Sensor.

4. A connects with analog input pin of Arduino when IR pair use as Analog Sensor

```
const int ProxSensor=A0;

int inputVal = 0;




void setup()

{

  pinMode(13, OUTPUT);           // Pin 13 has an LED
connected on most Arduino boards:

  pinMode(ProxSensor,INPUT);     //Pin 2 is connected to
the output of proximity sensor
```

```
  Serial.begin(9600);

}

void loop()

{

  if(digitalRead(ProxSensor)==HIGH)       //Check the
sensor output

  {

    digitalWrite(13, HIGH);   // set the LED on

  }

  else

  {

    digitalWrite(13, LOW);    // set the LED off

  }

inputVal = analogRead(ProxSensor);

Serial.println(inputVal);

delay(1000);                // wait for a second

}
```

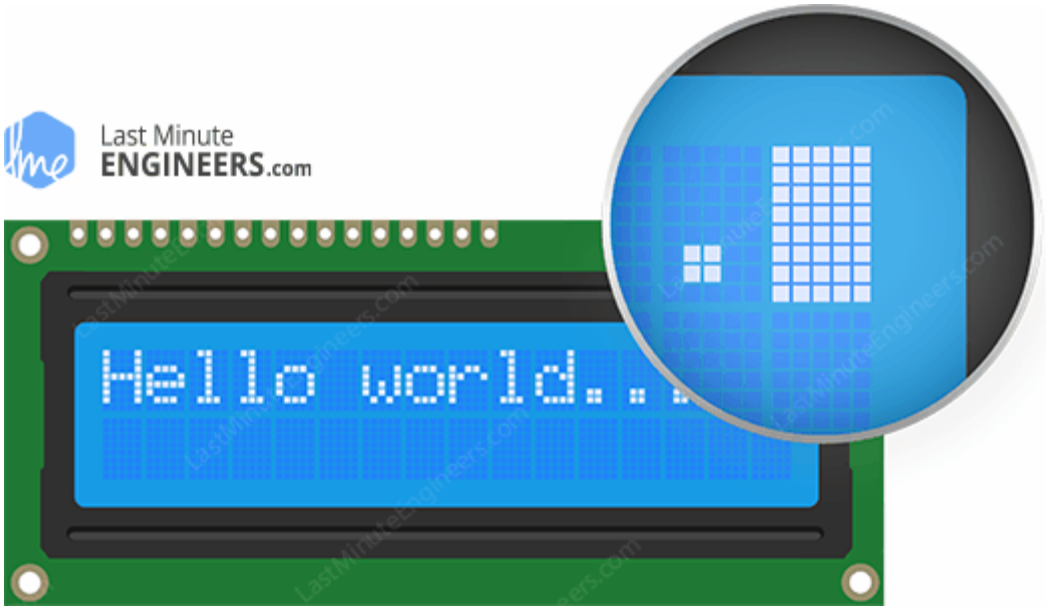
6.Interfacing 16×2 Character LCD Module with Arduino

Want your Arduino projects to display status messages or sensor readings? Then these LCD displays might be the perfect fit. They are extremely common and a fast way to add a readable interface to your project.

This tutorial will cover everything you need to know to get up and running with Character LCDs. Not just 16×2(1602) but any character LCDs (for example, 16×4, 16×1, 20×4 etc.) that are based on parallel interface LCD controller chip from Hitachi called the HD44780. Because, the Arduino community has already developed a library to handle HD44780 LCDs; so we'll have them interfaced in no time.

6.1. Hardware Overview

These LCDs are ideal for displaying text/characters only, hence the name 'Character LCD'. The display has an LED backlight and can display 32 ASCII characters in two rows with 16 characters on each row.



Each rectangle contains grid of 5×8 pixels

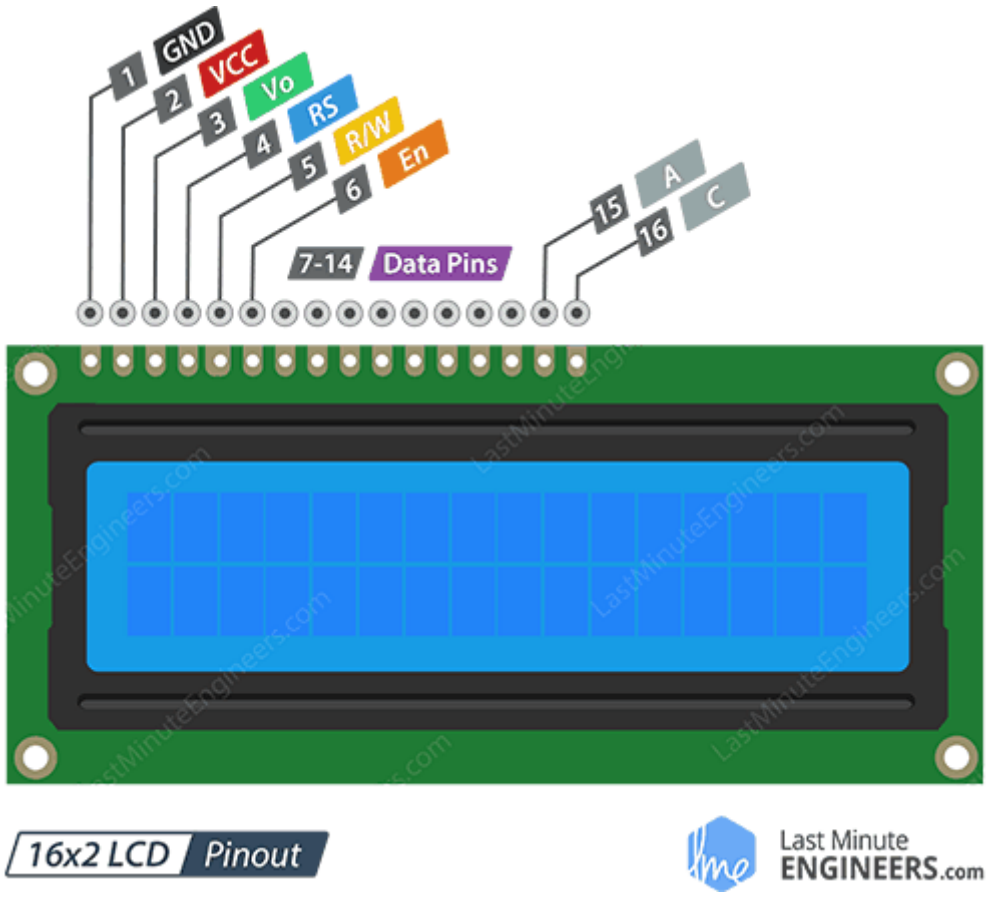
If you look closely, you can actually see the little rectangles for each character on the display and the pixels that make up a character. Each of these rectangles is a grid of 5×8 pixels.

Although they display only text, they do come in many sizes and colors: for example, 16×1, 16×4, 20×4, with white text on blue background, with black text on green and many more.

The good news is that all of these displays are ‘swappable’ – if you build your project with one you can just unplug it and use another size/color LCD of your choice. Your code may have to adjust to the larger size but at least the wiring is the same!

6.2.16×2 Character LCD Pinout

Before diving into hookup and example code, let’s first take a look at the LCD Pinout.



GND should be connected to the ground of Arduino.
VCC is the power supply for the LCD which we connect the 5 volts pin on the Arduino.
Vo (LCD Contrast) controls the contrast and brightness of the LCD. Using a simple voltage divider with a potentiometer, we can make fine adjustments to the contrast.
RS (Register Select) pin lets the Arduino tell the LCD whether it is sending commands or the data. Basically, this pin is used to differentiate commands from the data.

For example, when RS pin is set to LOW, then we are sending commands to the LCD (like set the cursor to a specific location, clear the display, scroll the display to the right and so on). And when RS pin is set on HIGH, we are sending data/characters to the LCD.

R/W (Read/Write) pin on the LCD is to control whether or not you're reading data from the LCD or writing data to the LCD. Since we're just using this LCD

as an OUTPUT device, we're going to tie this pin LOW. This forces it into the WRITE mode.

E (Enable) pin is used to enable the display. Meaning, when this pin is set to LOW, the LCD does not care what is happening with R/W, RS, and the data bus lines; when this pin is set to HIGH, the LCD is processing the incoming data.

D0-D7 (Data Bus) are the pins that carries the 8-bit data we send to the display. For example, if we want to see the uppercase 'A' character on the display we will set these pins to 0100 0001(according to the ASCII table) to the LCD.

A-K (Anode & Cathode) pins are used to control the backlight of the LCD.

6.3. Testing Character LCD

Now we're onto the interesting stuff. Let's test your LCD.

First, connect the 5V and GND pins from the Arduino Uno to the breadboard power rails and get your LCD plugged into the breadboard.

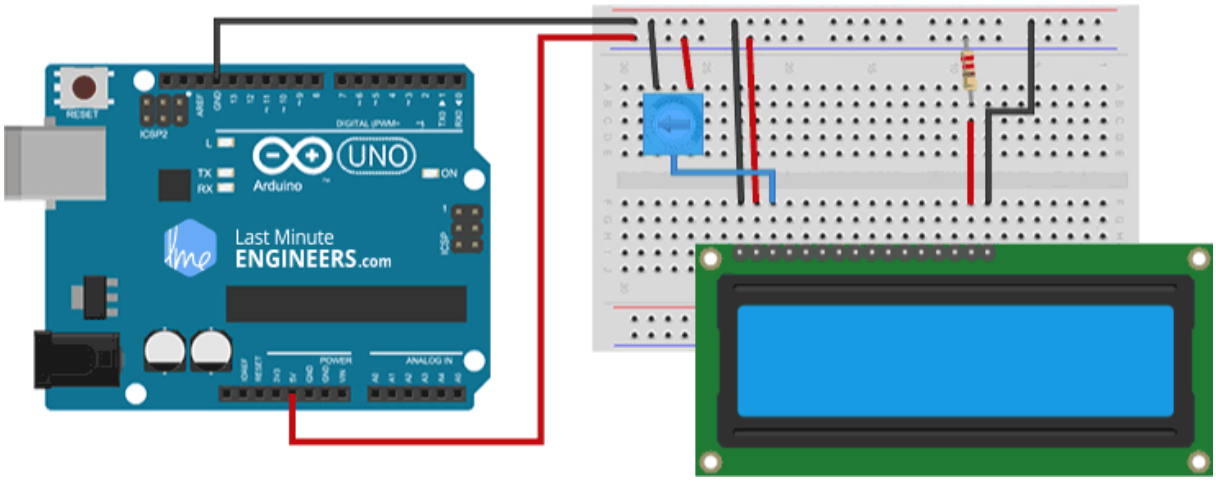
Now we'll power up the LCD. The LCD has two separate power connections; One (Pin 1 and Pin 2) for the LCD itself and another one (Pin 15 and Pin 16) for the LCD backlight. Connect pins 1 and 16 on the LCD to GND and pins 2 and 15 on the LCD to 5V.

Vast majority of LCDs have a built-in series resistor for the LED backlight. If you happen to have an LCD that does not include a resistor, you'll need to add one between 5V and pin 15.

To calculate the value of the series resistor, look up the maximum backlight current and the typical backlight voltage drop from the data sheet. And using simple ohm's law you can calculate the resistor value.

If you can't find the data sheet, then it should be safe to use a 220-ohm resistor, however a value this high may make the backlight slightly dim.

Next, we'll make connections for Pin 3 on the LCD which controls the contrast and brightness of the display. In order for fine contrast adjustments we'll connect a 10K potentiometer between 5V and GND; connect the centre pin (wiper) of the potentiometer to pin 3 on the LCD.



Adjusting LCD contrast by rotating potentiometer knob

That's it! Now turn on the Arduino, you'll see the backlight light up. And as you rotate the knob on the potentiometer, you should notice the first line of rectangles appear. If this happens, Congratulations! Your LCD is doing just fine.

6.4. Wiring – Connecting 16×2 Character LCD with Arduino Uno

Before we get to uploading code and sending data to the display, let's hook the LCD up to the Arduino.

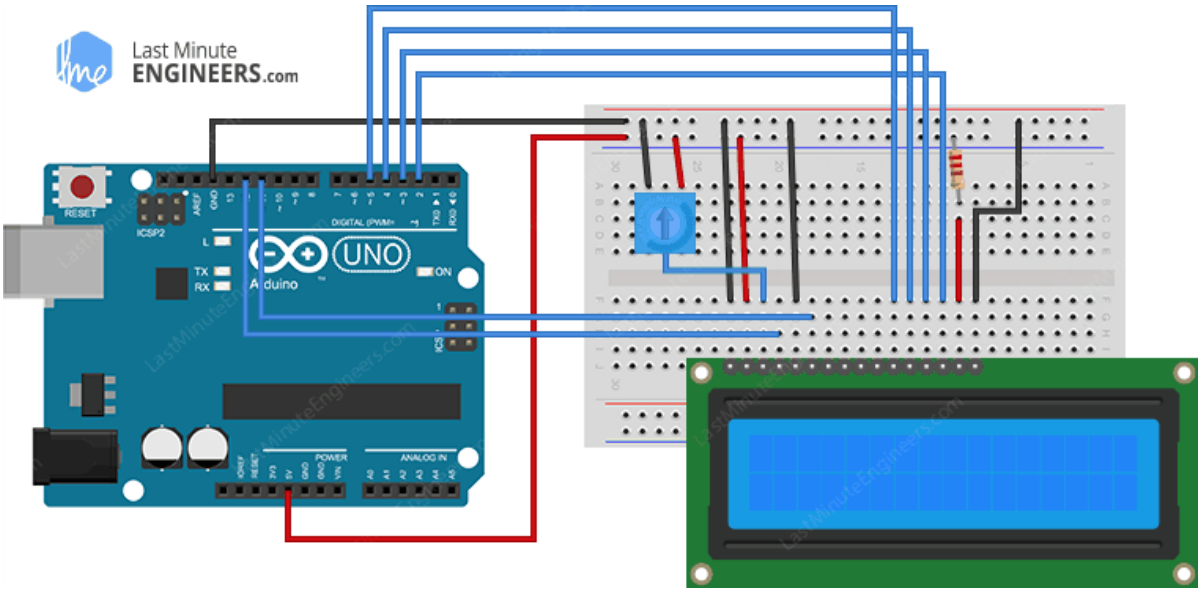
The LCD has a lot of pins (16 pins in total) that we'll show you how to wire up. But the good news is that not all these pins are necessary for us to connect to the Arduino.

We know that there are 8 Data lines that carry raw data to the display. But, HD44780 LCDs are designed in a way that we can talk to the LCD using only 4 data pins (4-bit mode) instead of 8(8-bit mode). This saves us 4 pins!

So, to recap, we will be interfacing LCD using 4-bit mode and hence we need only 6 pins: RS, EN, D7, D6, D5, and D4 to talk to the LCD.

Now, let's connect the LCD Display to the Arduino. Four data pins (D4-D7) from the LCD will be connected to Arduino's digital pins from #4-7. The Enable pin on LCD will be connected to Arduino #2 and the RS pin on LCD will be connected to Arduino #1.

The following diagram shows you how to wire everything.



Wiring connections of 16×2-character LCD and Arduino UNO

With that, you're now ready to upload some code and get the display printing.

6.5. Arduino Code

The following test sketch will print 'Hello World!' message on the LCD. Try the sketch out and then we will dissect it in some detail.

```
// include the library code:
#include <LiquidCrystal.h>

// Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
        // set up the LCD's number of columns and rows:
        lcd.begin(16, 2);

        // Clears the LCD screen
        lcd.clear();
}

void loop()
{
```

```
        // Print a message to the LCD.
        lcd.print(" Hello world!");

        // set the cursor to column 0, line 1
        // (note: line 1 is the second row, since counting begins with 0):
        lcd.setCursor(0, 1);
        // Print a message to the LCD.
        lcd.print(" LCD Tutorial");
}
```

If everything goes right, you should see something like this on the display.

6.6. Code Explanation:

The sketch starts by including Liquid Crystal library. As mentioned earlier in this tutorial, the Arduino community has a library called Liquid Crystal that makes programming the LCD module less difficult. You can explore more about the library on Arduino's official website.

```
// include the library code:
#include <LiquidCrystal.h>
```

Next, we have to create an Liquid Crystal object. This object uses 6 parameters and specifies which Arduino pins are connected to the LCD's RS pin, Enable pin, and data pins: d4, d5, d6, and d7.

```
// Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Now that you have declared a Liquid Crystal object, you can access special methods (aka functions) that are specific to the LCD.

In the 'setup' function: we will use two functions: The first function is `begin()`. This is used to specify the dimensions of the display i.e. how many columns and rows the display has. If you are using 16×2-character LCD, pass the parameters 16 & 2; If you are using 20×4 LCD, pass the parameters 20 & 4. You got the point!

The second function is `clear()`. This clears the LCD screen and moves the cursor to the upper left corner.

```
lcd.begin(16, 2);
lcd.clear();
```

In the 'loop' function: we will use the `print()` function which displays the message that we see on the first line of the screen.

```
// Print a message to the LCD.
lcd.print(" Hello world!");
```

Following that we will set the cursor position to second row, by calling function `setCursor()` The cursor position specifies the location where you need the new text to be displayed on the LCD. The upper left corner is considered col=0, row=0

```
lcd.setCursor(0, 1);
lcd.print(" LCD Tutorial");
```

6.7. Other useful functions in Liquid Crystal Library

There are a few useful functions you can use with Liquid Crystal object. Few of them are listed below:

- If you just want to position the cursor in the upper-left of the LCD without clearing the display, use `home()`
- There are many applications like turbo C++ compiler or notepad++, in which pressing 'insert' key on the keyboard changes cursor. Just like that you can change the cursor on the LCD using `blink()` or `lcd.cursor()` .

- `blink()` function displays the blinking block of 5×8 pixels, while `lcd.cursor()` displays an underscore (line) at the position to which the next character will be written.
- You can use the `noBlink()` function to turns off the blinking LCD cursor and `lcd.noCursor()` to hide the LCD cursor.
- You can scroll the contents of the display one space to the right using `lcd.scrollDisplayRight()` or one space left using `lcd.scrollDisplayLeft()`. If you want to scroll the text continuously, you need to use these functions inside a 'for loop'.

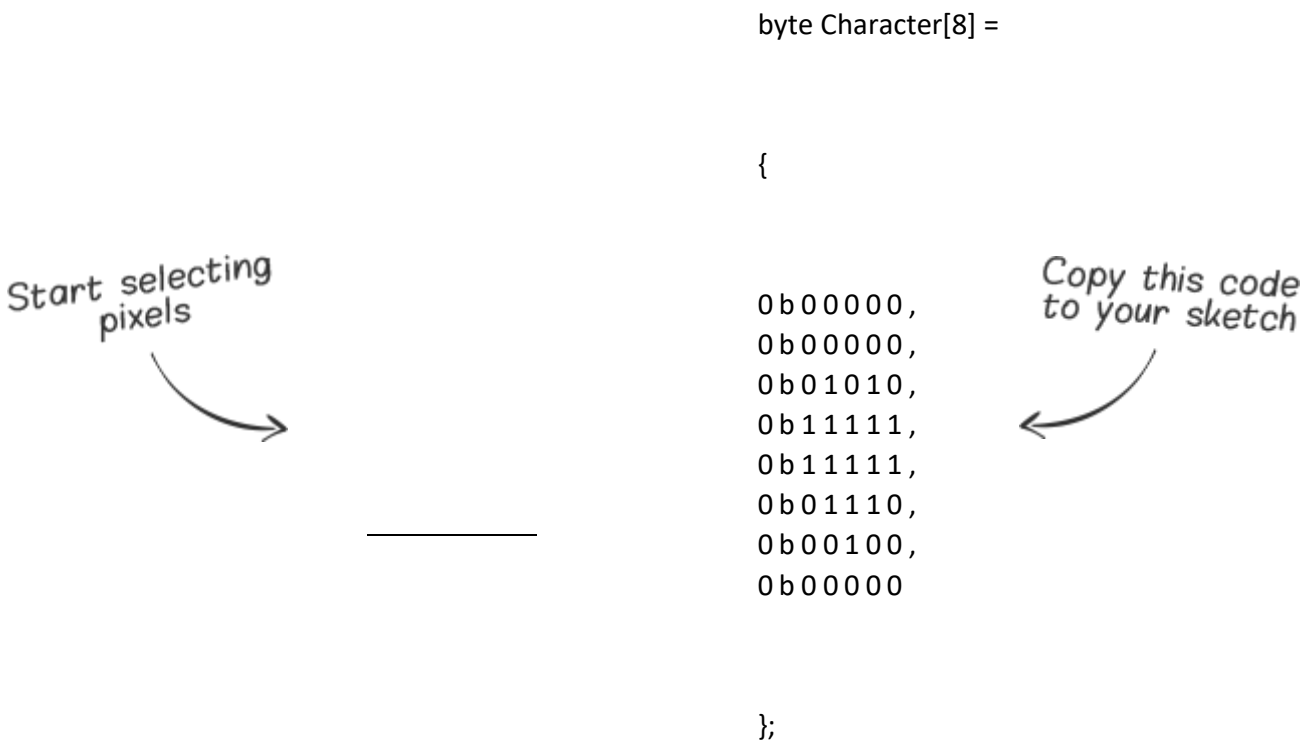
6.8. Custom character generation for 16×2-character LCD

If you are finding characters on the display dull and unexciting, you can create your own custom characters (glyph) and symbols for your LCD. They are extremely useful when you want to display a character that is not part of the standard ASCII character set.

As we discussed earlier in this tutorial that a character on the display is formed in a 5×8 matrix of pixels so you need to define your custom character within that matrix. To define the character you'll use the createChar() function of the Liquid Crystal library.

To use `createChar()` you first set up an array of 8 bytes. Each byte (only 5 bits are considered) in the array defines one row of the character in the 5×8 matrix. Whereas, the zeros and ones in the byte indicate which pixels in the row should be on and which ones should be off.

Creating custom character was not easy until now! We have created a small application called Custom character generator for character LCD. Can you see the blue grid below? That's it. That's the application. You can click on any of the 5×8 pixels to set/clear that particular pixel. And as you click on pixels, the code for the character is generated next to the grid. This code can directly be used in your Arduino sketch.



Your imagination is limitless. The only limitation is that the Liquid Crystal library supports only eight custom characters. But don't get disappointed, look at the bright side, at least we have eight characters.

The following sketch demonstrates how you can use these custom characters on the display.

```
// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// make some custom characters:
byte Heart[8] = {
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
```

```
0b00000,
0b00000
};

byte Bell[8] = {
0b00100,
0b01110,
0b01110,
0b01110,
0b11111,
0b00000,
0b00100,
0b00000
};


byte Alien[8] = {
0b11111,
0b10101,
0b11111,
0b11111,
0b01110,
0b01010,
0b11011,
0b00000
};

byte Check[8] = {
0b00000,
0b00001,
0b00011,
0b10110,
0b11100,
0b01000,
0b00000,
0b00000
};

byte Speaker[8] = {
0b00001,
```

```
0b00011,
0b01111,
0b01111,
0b01111,
0b00011,
0b00001,
0b00000
};


byte Sound[8] = {
0b00001,
0b00011,
0b00101,
0b01001,
0b01001,
0b01011,
0b11011,
0b11000
};


byte Skull[8] = {
0b00000,
0b01110,
0b10101,
0b11011,
0b01110,
0b01110,
0b00000,
0b00000
};

byte Lock[8] = {
0b01110,
0b10001,
0b10001,
0b11111,
0b11011,
0b11011,
```

```
0b11111,
0b00000
};

void setup()
{
        // initialize LCD and set up the number of columns and rows:
        lcd.begin(16, 2);

        // create a new character
        lcd.createChar(0, Heart);
        // create a new character
        lcd.createChar(1, Bell);
        // create a new character
        lcd.createChar(2, Alien);
        // create a new character
        lcd.createChar(3, Check);
        // create a new character
        lcd.createChar(4, Speaker);
        // create a new character
        lcd.createChar(5, Sound);
        // create a new character
        lcd.createChar(6, Skull);
        // create a new character
        lcd.createChar(7, Lock);

        // Clears the LCD screen
        lcd.clear();

        // Print a message to the lcd.
        lcd.print("Custom Character");
}

// Print All the custom characters
void loop()
{
        lcd.setCursor(0, 1);
        lcd.write(byte(0));

        lcd.setCursor(2, 1);
```

```
        lcd.write(byte(1));

        lcd.setCursor(4, 1);
        lcd.write(byte(2));

        lcd.setCursor(6, 1);
        lcd.write(byte(3));

        lcd.setCursor(8, 1);
        lcd.write(byte(4));

        lcd.setCursor(10, 1);
        lcd.write(byte(5));

        lcd.setCursor(12, 1);
        lcd.write(byte(6));

        lcd.setCursor(14, 1);
        lcd.write(byte(7));
}
```

After including the library, we need initialize the custom character array of eight bytes.

```
byte Heart[8] = {
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000,
0b00000
};
```

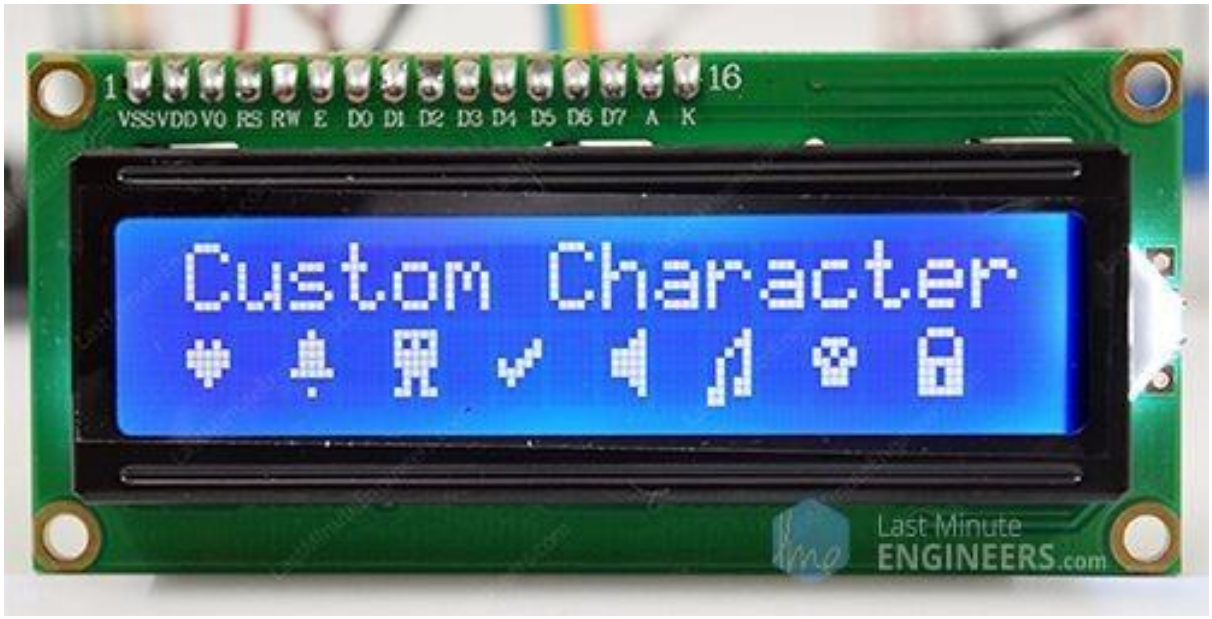
In the setup we have to create the custom character using the `createChar()` function. This function takes two parameters. The first one is a number between 0 and 7 in order to reserve one of the 8 supported custom characters. The second parameter is the name of the array of bytes.

```
// create a new character
lcd.createChar(0, Heart);
```

Next in the loop, to display the custom character we use `write()` function and as a parameter we use the number of the character we reserved.

// byte(0) represents Heart character.

lcd.write(byte(0));



7.REAL LIFE CONCLUSIONS

7.1. ADVANTAGES

- This is done at low cost and very economical.
- Temperature can be scanned so that when we use at hospital premises, we can have healthy and safe roaming of visitors.
- This project has a capability of counting number of visitors or persons that crossed the gate or the entrance or the exit. So that we can have a real time monitoring.
- As we are linking this to IOT so that we can have a good monitoring and minute to minute updates.
- We are having a IOT based application so we can monitor from our android phones itself.
- It can replace thermal guns.
- Helps to maintain COVID Norms, by restricting the persons with high temperature.
- Easy to Implement near any existing entries.
- Easy configuration and setup process.
- On-board or cloud data reporting to shield from future liability.

7.2. LIMITATIONS

- Weather can affect the temperature of the human body.
- Only measure surface temperatures and NOT the internal temperature of food or other materials.
- Are not thought to be as highly accurate as surface probes measurements of the same.
- Can be temporarily affected by frost, moisture, dust, fog, smoke or other particles in the air.
- By this man power can be reduced so that employment shortage will occurs.
- Distance can affect the IR scanner(temperature).
- Allow only Low Temperature (Below 98.5F) person to enter the Premises.
- Can be used only inside not outside.

7.3. APPLICATIONS

There are some real time applications the we use our project or circuit to make our premises better.

- This can be implemented at schools, colleges, educational institutions, railway stations, bus stations, airports Restaurants, hotels…etc
- This can be more useful where there are a greater number of public gatherings and crowded places.
- This automated temperature scanning and entry opening system can be more useful and needed when the person who checks our temperature with a thermal gun is not well trained.
- This automated project can be used to reduce the manual scanning system in the large crowds.
- There will warning beeps so that it can alert people and prevent the entry or exit.
- Easily accessed by the user as it is linked to android application.
- Self-servicing, no secondary temperature taker required.

- Screening for employee health to keep sick people home and building safety and employee wellbeing.

According to the FDA, automated thermal scanning may help reduce cross-contamination and the risk of spreading diseases, as opposed to traditional infrared thermometers. They are easy to clean and use, making them accessible and safe for many people, not just trained medical professionals. however, often refers to a thermometer that uses infrared but must be operated by a person, which is less safe and pulls staff away from their work.

The biggest issue with manual scanning system is that the operator must bring the thermometer relatively close to the person being scanned, usually within a few inches. They may be too close to maintain appropriate safety and social distancing measures. Even with proper personal protective equipment, this puts both parties at unnecessary risk. With an automated, touchless temperature screening machine, there is no other human involved. There isn't even an item to touch. It is fully contactless, allowing workers and customers to abide by social distancing and stay safer.

8.REFERENCES

- Gostic, K.M. Gomez, A.C.R., Mummah, R.O., Kucharski, A.J., Lloyd-Smith J.O., Estimated effectiveness of symptom and risk screening to prevent the spread of COVID-19. eLife, 2020. 9 e55570
- Quilty, B.J. and S. Clifford, Effectiveness of airport screening at detecting travellers infected with novel coronavirus (2019-nCoV)External Link Disclaimer. Euro Surveillance, 2020. 25(5).
- Hewlett, A.L., et al., Evaluation of an infrared thermal detection system for fever recognition during the H1N1 influenza pandemic.External Link Disclaimer Infection Control & Hospital Epidemiology, 2011. 32(5): p. 504-506.
- Priest, P.C., et al., Thermal Image Scanning for Influenza Border Screening: Results of an Airport Screening Study.External Link Disclaimer Plos One, 2011. 6(1): p. e14490.
- Tay, M., et al., Comparison of Infrared Thermal Detection Systems for mass fever screening in a tropical healthcare settingExternal Link Disclaimer. Public health, 2015. 129(11): p. 1471-1478.
- Chan, L., et al., Utility of infrared thermography for screening febrile subjects.External Link Disclaimer Hong Kong Medical Journal, 2013. 19(2): p. 109-115.
- Nguyen, A.V., et al., Comparison of 3 infrared thermal detection systems and self-report for mass fever screening. Emerging Infectious Diseases, 2010. 16(11): p. 1710-1717.
- Mouchtouri, V.A., et al., Exit and entry screening practices for infectious diseases among travelers at points of entry: looking for evidence on public health impact. International Journal of Environmental Research and Public Health, 2019. 16(23): p. 4638.
- Bitar, D., A. Goubar, and J. Desenclos, International travels and fever screening during epidemics: a literature review on the effectiveness and potential use of non-contact infrared thermometers. Euro Surveillance, 2009. 14(6): p. 19115.
- Liu, C.-C., R.-E. Chang, and W.-C. Chang, Limitations of forehead infrared body temperature detection for fever screening for severe acute respiratory syndrome. Infection Control & Hospital Epidemiology, 2004. 25(12): p. 1109-1111.
- Fletcher, T., et al., Comparison of non-contact infrared skin thermometers. Journal of medical engineering & technology, 2018. 42(2): p. 65-71.
- Morán-Navarro, R., et al., Validity of skin, oral and tympanic temperatures during exercise in the heat: effects of wind and sweatExternal Link Disclaimer. Annals of biomedical engineering, 2019. 47(1): p. 317-331.
- Mouchtouri, V.A., et al., Exit and Entry Screening Practices for Infectious Diseases among Travelers at Points of Entry: Looking for Evidence on Public Health Impact. Int J Environ Res Public Health, 2019 16(23) 4638
- Bwire, GM and Paulo, LS, Coronavirus disease-2019: is fever an adequate screening for the returning travelers?External Link Disclaimer Tropical Medicine and Health, 2020 48:18
- Duong, A, et al., Rapid Temperature Screening for Workplace HealthExternal Link Disclaimer, Infection Control Tips, 2017