SSG EMBEDDED SOLUTIONS

SSG

info@ssges.co.in

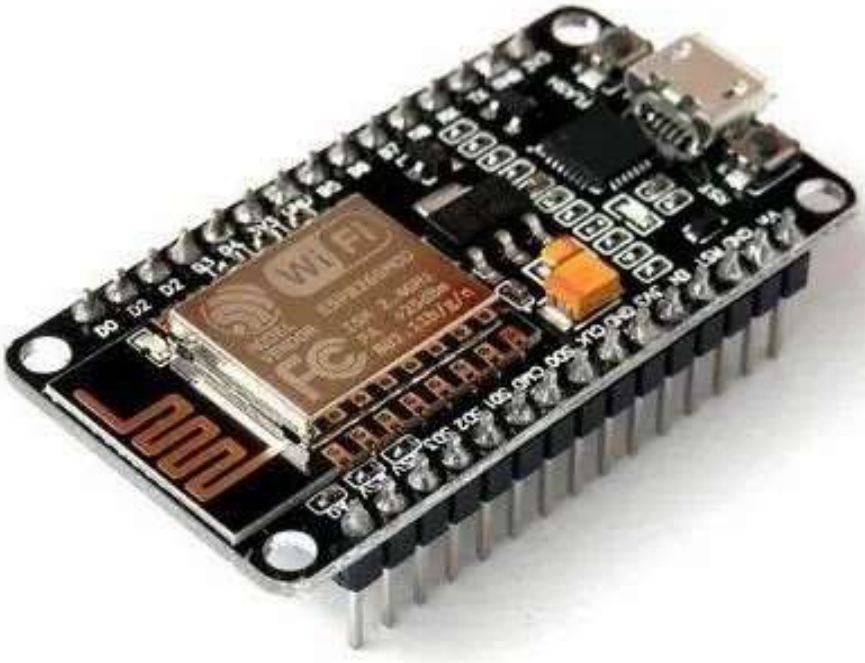
Embedded solutions

Microcontroller Development Kit

ESP 8266(NodeMCU)

Documentation

Wi-Fi Module



SSG EMBEDDED SOLUTIONS | Ph. No. 7123559635

List of Contents

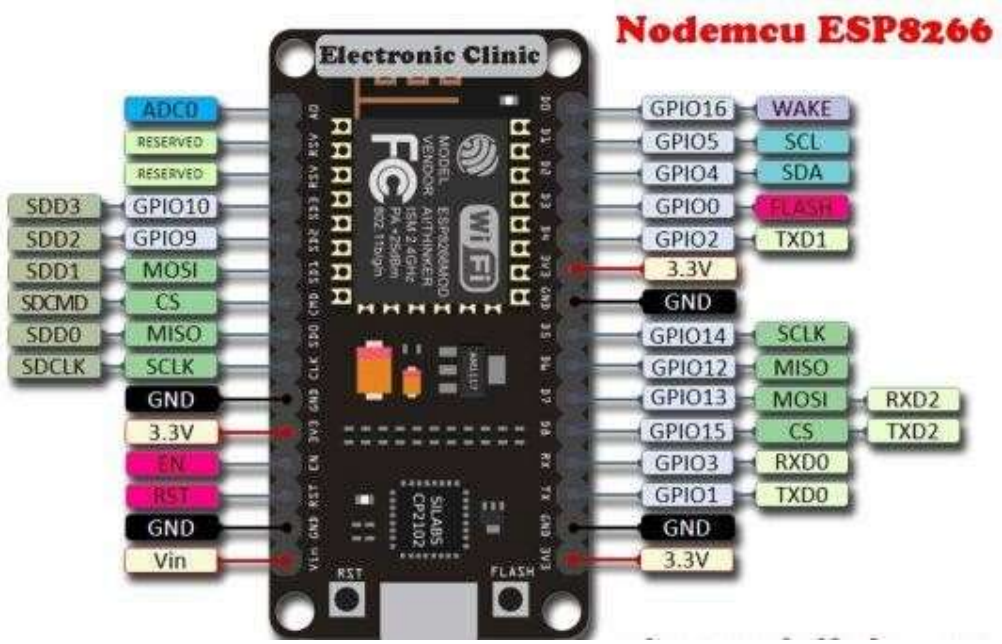
Introduction

An ESP8266 Wi-Fi module is a SOC microchip mainly used for the development of end-point IoT (Internet of things) applications. It is referred to as a standalone wireless transceiver, available at a very low price. It is used to enable the internet connection to various applications of embedded systems.

Espressif systems designed the ESP8266 Wi-Fi module to support both the TCP/IP capability and the microcontroller access to any Wi-Fi network. It provides the solutions to meet the requirements of industries of IoT such as cost, power, performance, and design.

It can work as either a slave or a standalone application. If the ESP8266 Wi-Fi runs as a slave to a microcontroller host, then it can be used as a Wi-Fi adaptor to any type of microcontroller using UART or SPI. If the module is used as a standalone application, then it provides the functions of the microcontroller and Wi-Fi network.

Pin Diagram



Specifications

Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
Operating Voltage: 3.3V
Input Voltage: 7-12V
Digital I/O Pins (DIO): 16
Analog Input Pins (ADC): 1
UARTs: 1
SPIs: 1
I2Cs: 1
Flash Memory: 4 MB
SRAM: 64 KB
Clock Speed: 80 MHz
USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
PCB Antenna
Small Sized module to fit smartly inside your IoT projects

Applications

- Access points portals
- IoT projects
- Wireless data logging
- Used in learning the networking fundamentals
- Sockets and smart bulbs
- Smart home automation systems

Installing Arduino IDE

To download the Arduino IDE, visit the following URL:

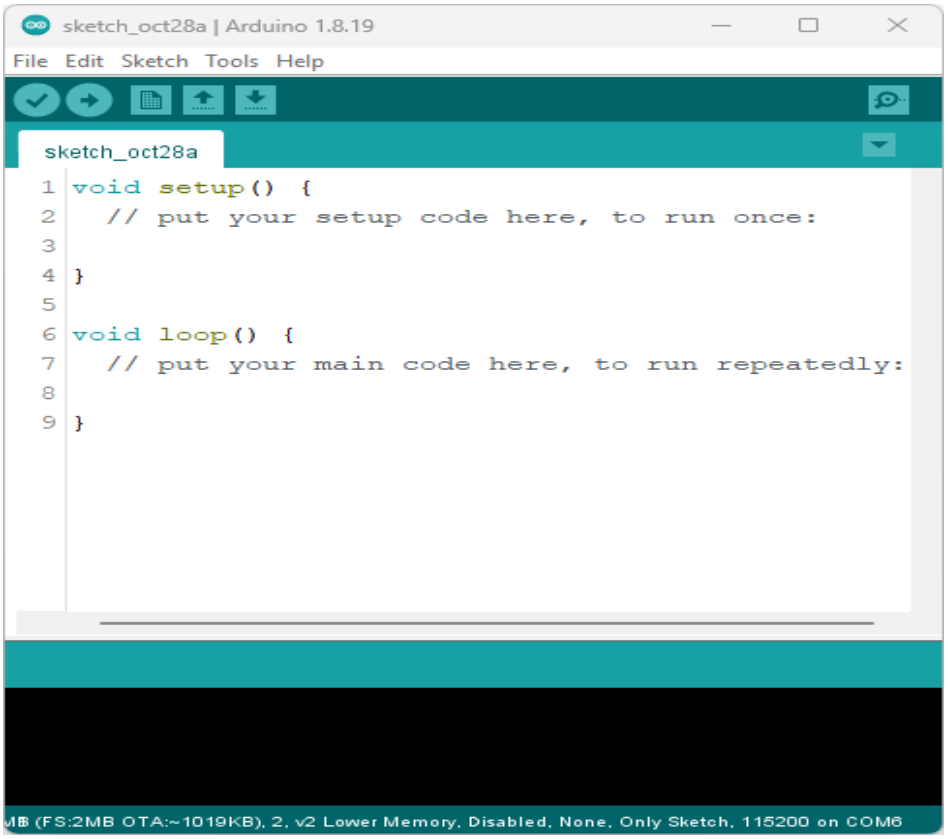
https://www.arduino.cc/en/Main/Software

Don't install the 2.0 version. At the time of writing this tutorial, **we recommend using the legacy version (3.8.39)** with the ESP32. While version 2 works well with Arduino, there are still some bugs and some features that are not supported yet for the ESP32.
Scroll down until you find the legacy version section.

Select your operating system and download the software. For Windows, we recommend downloading the "**Windows ZIP file**".

revisions.txt	10/28/2022 4:10 PM	Text Document	97 KB
wrapper-manifest.xml	10/28/2022 4:10 PM	Microsoft Edge H...	1 KB
arduino.exe	10/28/2022 4:10 PM	Application	72 KB
arduino.l4j.ini	10/28/2022 4:10 PM	Configuration sett...	1 KB
arduino_debug.exe	10/28/2022 4:10 PM	Application	69 KB
arduino_debug.l4j.ini	10/28/2022 4:10 PM	Configuration sett...	1 KB
arduino-builder.exe	10/28/2022 4:10 PM	Application	23,156 KB
libusb0.dll	10/28/2022 4:10 PM	Application exten...	43 KB
msvcp100.dll	10/28/2022 4:10 PM	Application exten...	412 KB
msvcr100.dll	10/28/2022 4:10 PM	Application exten...	753 KB
tools-builder	10/28/2022 4:12 PM	File folder	
tools	10/28/2022 4:12 PM	File folder	
libraries	10/28/2022 4:12 PM	File folder	
lib	10/28/2022 4:11 PM	File folder	
java	10/28/2022 4:11 PM	File folder	
hardware	10/28/2022 4:11 PM	File folder	
examples	10/28/2022 4:10 PM	File folder	
drivers	10/28/2022 4:10 PM	File folder	

The Arduino IDE window should open.

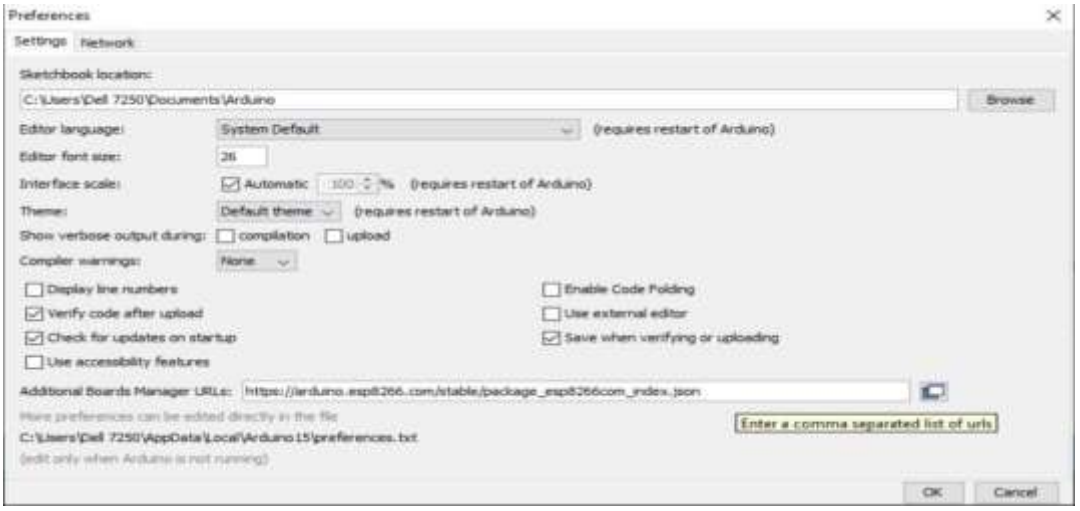


Installing using Arduino IDE

Now, to install the ESP8266 board in the Arduino IDE, you need to follow the below steps −

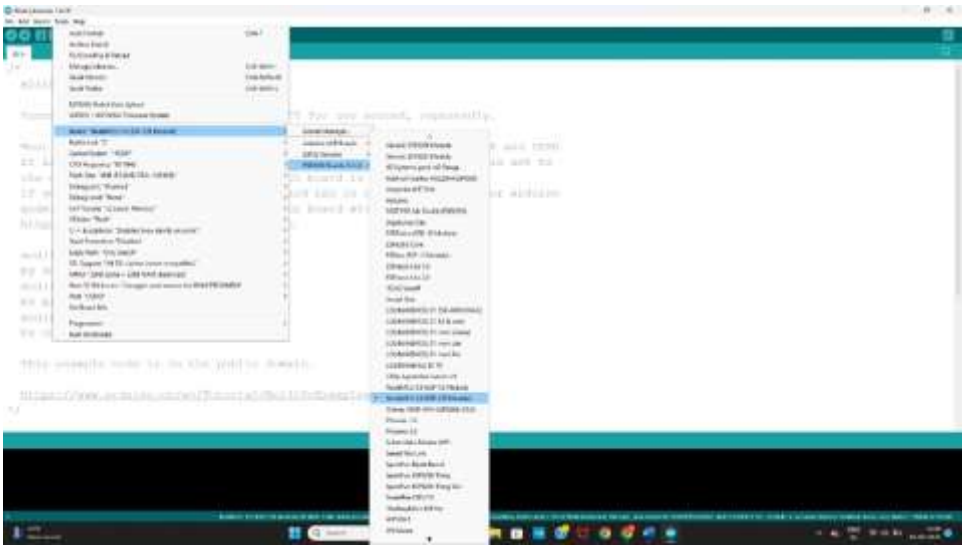
- Make sure you have Arduino IDE (preferably the latest version) installed on your machine
- Open Arduino and go to File −> Preferences
- In the Additional Boards Manager URL, enter

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```



- Go to Tools −> Board−> Boards Manager. A pop−up would open up. Search for ESP8266 and install the **esp8266** by **Espressif Systems** board. The image below shows the board already installed because I had installed the board before preparing this tutorial.

- Once your ESP8266 board has been installed, you can verify the installation by going to Tools −> Boards. You can see a whole bunch of boards under the **ESP8266 Arduino** section. Choose the board of your choice. If you are not sure which board best represents the one you have, you can choose **ESP8266 Dev Module**.



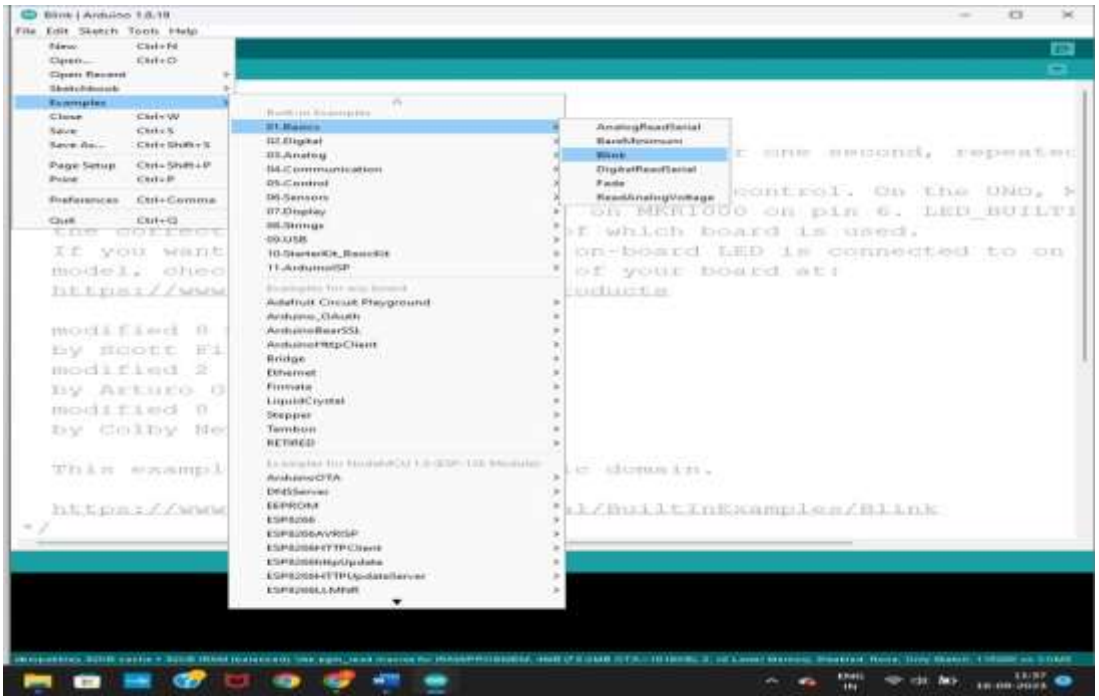
Next, connect your board to your machine using the USB Cable. You should see an additional COM Port under Tools−> Port. Select that additional port. In case you see multiple ports, you can disconnect the USB and see which port disappeared. That port corresponds to ESP8266.

Materials Required

- ESP8266 Module
- Arduino IDE
- Programming cable (micro-USB cable)

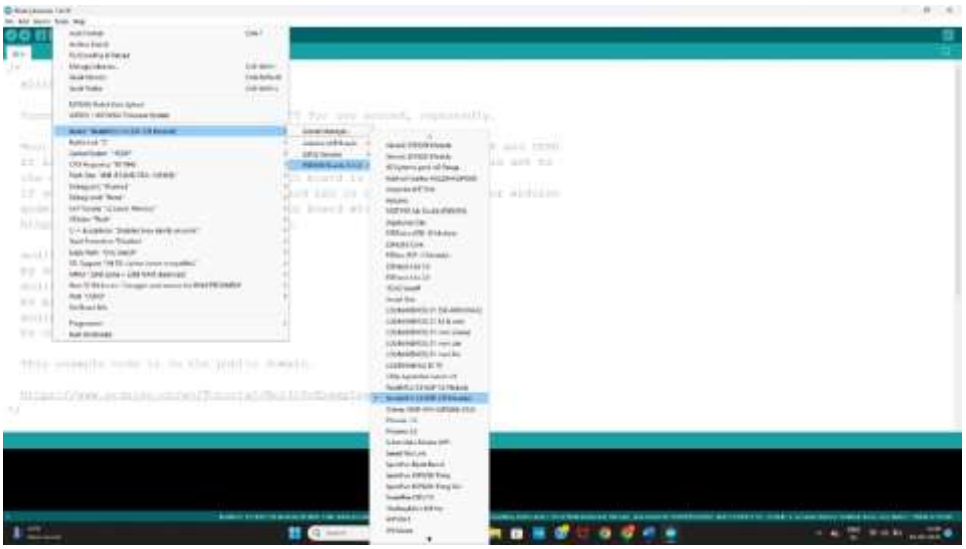
Inbuilt led blinking

Step 3: Open a Blink basic project in Arduino IDE on *File>Examples>03.Basics>Blink.*

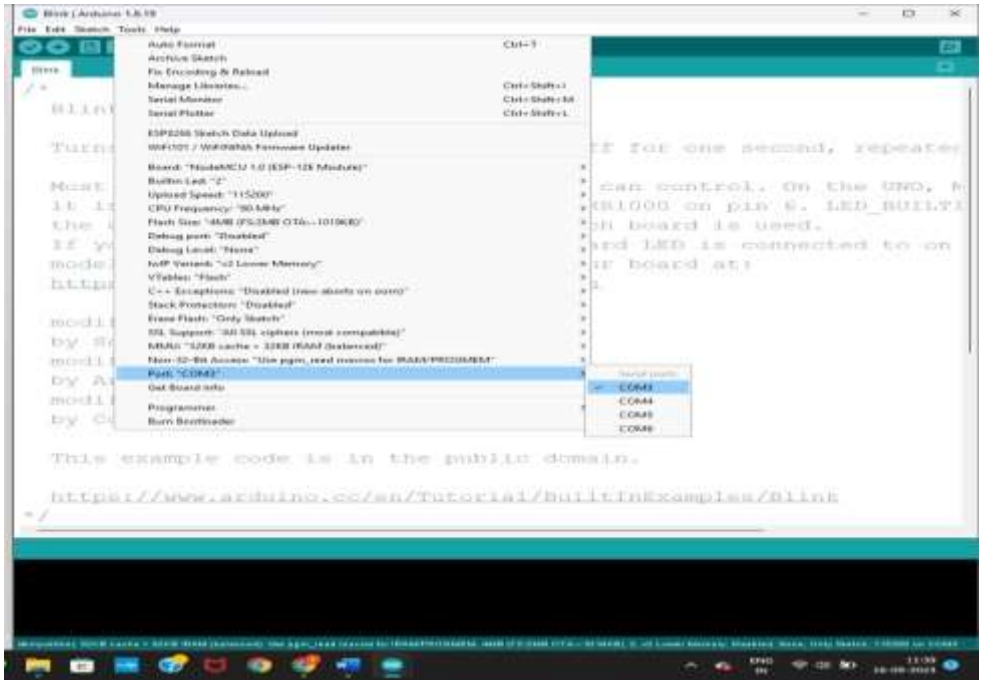


Step 2: Connect ESP8266 board to computer using Micro USB to USB cable.

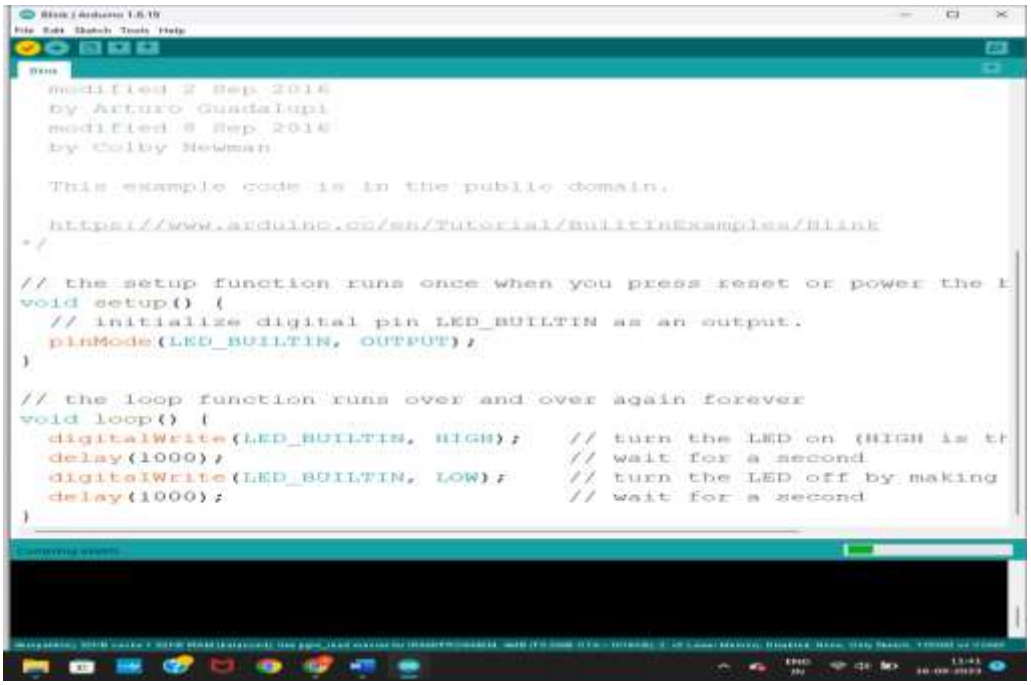
Step 3: Change the board configuration on *Tools>Board>ESP8266Arduino.*



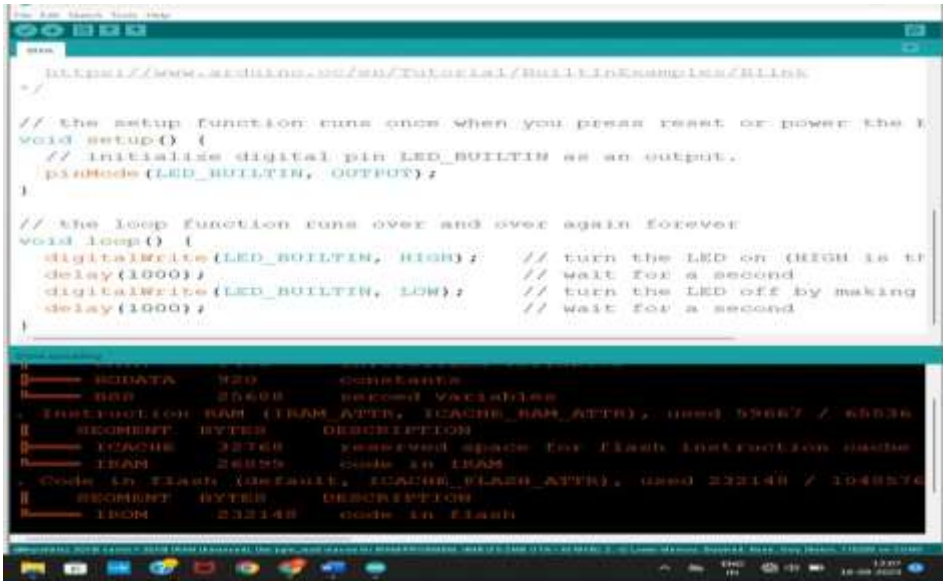
Step 4: Change the port configuration to *USBtoUART* on *Tools>Port*.



Step 5: Click *Verify* to compile the project.

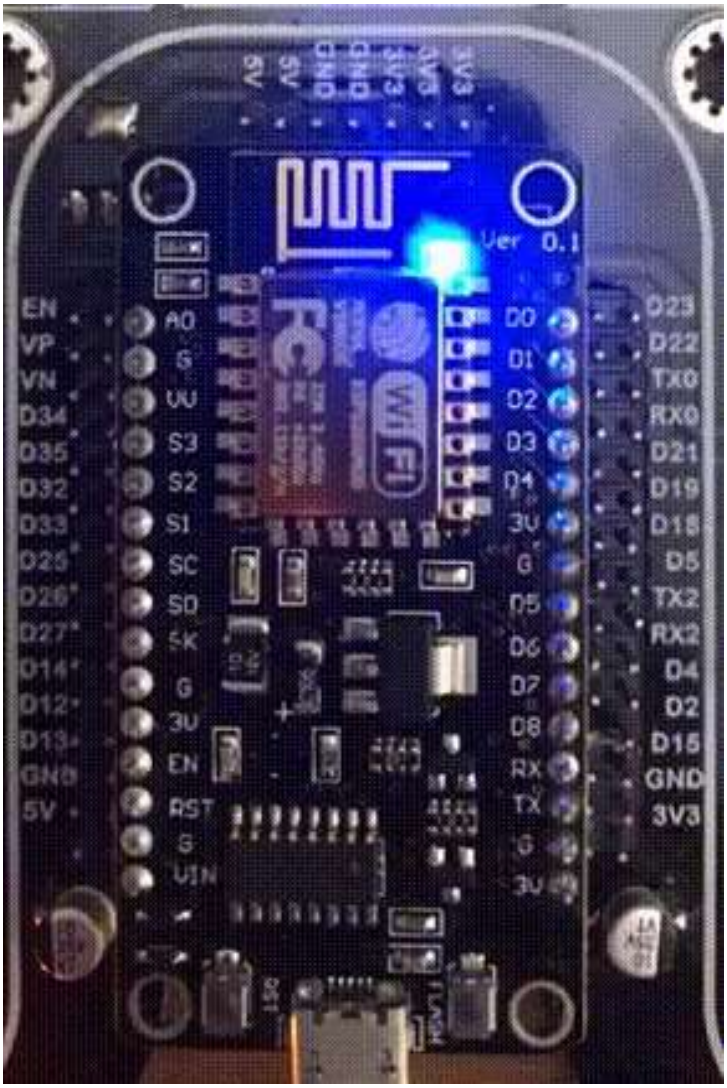


Step 6: Click *Upload* to run the code on ESP8266.



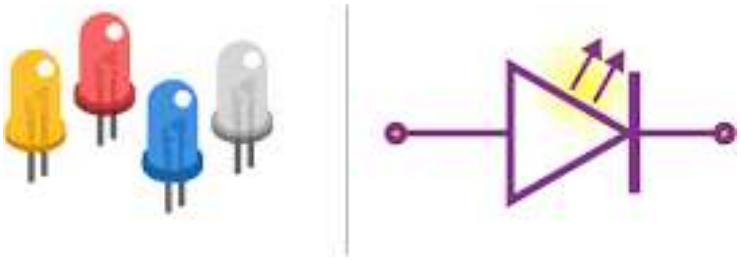
Important! *Press the BOOT button on ESP8266 when Arduino IDE shows "Connecting" status!*

Step 7: The built-in LED will blink with 3 second interval.



LED

It is most widely used semiconductor which emit either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diodes (LED) is an optical electrical energy into light energy when voltage is applied.



These are the applications of LEDs:

- Digital computers and calculators.
- Traffic signals and Burglar alarms systems.
- Camera flashes and automotive heat lamps
- Picture phones and digital watches.

LED Blinking Code

```
//connect D1,D2,D4 &D3 with LED PINS

int LED1 = 0; // Assign LED pin i.e: D3 on NodeMCU

int LED2 = 5; // Assign LED pin i.e: D1 on NodeMCU

int LED3 = 4; // Assign LED pin i.e: D2 on NodeMCU

int LED4 = 2; // Assign LED pin i.e: D4 on NodeMCU

void setup() {

// initialize GPIO 5 as an output

pinMode(LED1, OUTPUT);

pinMode(LED2, OUTPUT);

pinMode(LED3, OUTPUT);

pinMode(LED4, OUTPUT);
```

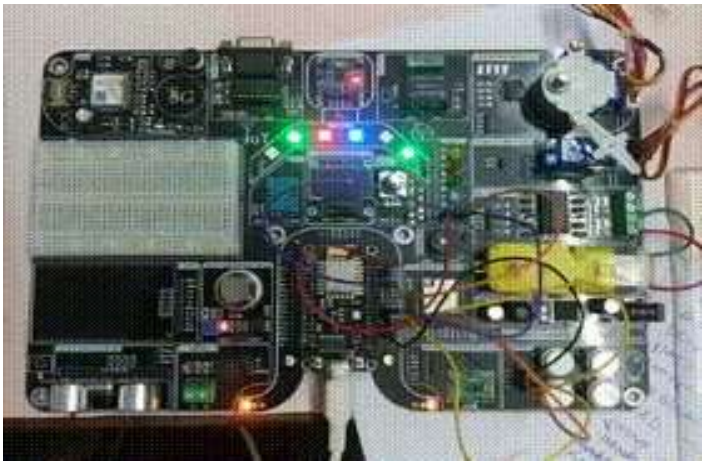
```
}
// the loop function runs over and over again forever
void loop() {
 digitalWrite(LED1, HIGH); // turn the LED on
digitalWrite(LED2, HIGH); // turn the LED on
digitalWrite(LED3, HIGH); // turn the LED on
digitalWrite(LED4, HIGH); // turn the LED on
delay(1000); // wait for a second
digitalWrite(LED1, LOW); // turn the LED off
digitalWrite(LED2, LOW); // turn the LED off
digitalWrite(LED3, LOW); // turn the LED off
digitalWrite(LED4, LOW); // turn the LED off
delay(1000); // wait for a second
}
```

LED Blinking with Switch Code

```
//Connect D0 with LEDPIN1 &TX with PUSHBUTTONPIN1
//Connect D1 with LEDPIN2 &RX with PUSHBUTTONPIN2
//Connect D6 with LEDPIN3 &D5 with PUSHBUTTONPIN3
//Connect D3 with LEDPIN4 &D7 with PUSHBUTTONPIN4
const int LEDPIN1 = 16;
const int LEDPIN2 = 5;
const int LEDPIN3 = 12;
const int LEDPIN4 = 0;

const int PushButton1 =1;
const int PushButton2 =3;
const int PushButton3 =14;
const int PushButton4 =13;

// This Setup function is used to initialize everything
void setup()
{
// This statement will declare pin 22 as digital output
pinMode(LEDPIN1, OUTPUT);
pinMode(LEDPIN2, OUTPUT);
pinMode(LEDPIN3, OUTPUT);
pinMode(LEDPIN4, OUTPUT);

// This statement will declare pin 15 as digital input
pinMode(PushButton1, INPUT_PULLUP);
pinMode(PushButton2, INPUT_PULLUP);
pinMode(PushButton3, INPUT_PULLUP);
```

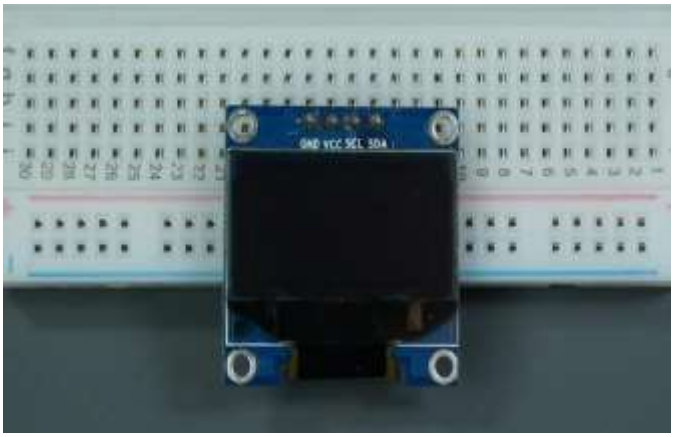
```
pinMode(PushButton4, INPUT_PULLUP);
}
void loop()
{
// digitalRead function stores the Push button state
// in variable push_button_state
int Push_button_state1 = digitalRead(PushButton1);
int Push_button_state2 = digitalRead(PushButton2);
int Push_button_state3 = digitalRead(PushButton3);
int Push_button_state4 = digitalRead(PushButton4);
if ( Push_button_state1 == LOW )
{
digitalWrite(LEDPIN1, HIGH);
}
else
{
  digitalWrite(LEDPIN1, LOW);
}
 if ( Push_button_state2 == LOW )
{
digitalWrite(LEDPIN2, HIGH);
}
else
{
  digitalWrite(LEDPIN2, LOW);
}
 if ( Push_button_state3 == LOW )
{
```

```
digitalWrite(LEDPIN3, HIGH);
}
else
{
  digitalWrite(LEDPIN3, LOW);
}
 if ( Push_button_state4 == LOW )
{
digitalWrite(LEDPIN4, HIGH);
}
else
{
  digitalWrite(LEDPIN4, LOW);
}
}
```



OLED

OLED is the acronym for **Organic Light Emitting Diode**. OLED is a modern display technology used in a wide range of electronic display devices, such as TVs, monitors, laptops, smartphones, bulletin boards, stadium screens, etc.



OLED displays consist of organic semiconductor compounds that emit a bright light on the passage of electric current through them, and hence it is termed as OLED. Since, OLED displays can emit light on their own, thus they are considered as self-emissive types of display. There is no need of backlight panel with LEDs to illuminate the screen.

The primary advantages of OLED displays include better picture quality, relatively wider viewing angles, greater flexibility in design, compact size, faster response time, and low power consumption.

OLED Code

```
//CONNECT D1 TO SCL & D2 TO SDA OF SSD1306 DISPLAY

#include <SPI.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128 // OLED display width, in pixels
```

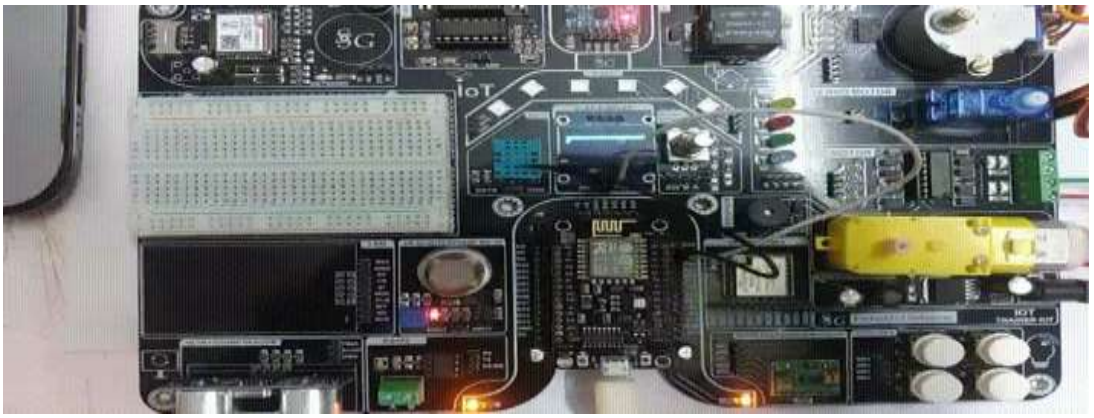
```
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET     -1 // Reset pin
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

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

 // initialize the OLED object
 if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;); // Don't proceed, loop forever
 }
 // Clear the buffer.
 display.clearDisplay();

 // Display Text
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,28);
 display.println("SSG EMBEDDED SOLUTION");
 display.display();
 delay(2000);
 display.clearDisplay();
```

```
// Display Inverted Text
display.setTextColor(BLACK, WHITE); // 'inverted' text
display.setCursor(0,10);
display.println("SSG EMBEDDED SOLUTIONS");
display.display();
delay(2000);
display.clearDisplay();

// Changing Font Size
display.setTextColor(WHITE);
display.setCursor(0,24);
display.setTextSize(2);
display.println("SSG");
display.display();
delay(2000);
display.clearDisplay();

// Display Numbers
display.setTextSize(1);
display.setCursor(0,28);
display.println(123456789);
display.display();
delay(2000);
display.clearDisplay();

// Specifying Base For Numbers
```

```
display.setCursor(0,28);
display.print("0x"); display.print(0xFF, HEX);
display.print("(HEX) = ");
display.print(0xFF, DEC);
display.println("(DEC)");
display.display();
delay(2000);
display.clearDisplay();

// Display ASCII Characters
display.setCursor(0,24);
display.setTextSize(2);
display.write(3);
display.display();
delay(2000);
display.clearDisplay();

// Scroll full screen
display.setCursor(0,0);
display.setTextSize(1);
display.println("SSG");
display.println("EMBEDDED");
display.println("SOLUTIONS!");
display.display();
display.startscrollright(0x00, 0x07);
delay(2000);
display.stopscroll();
```

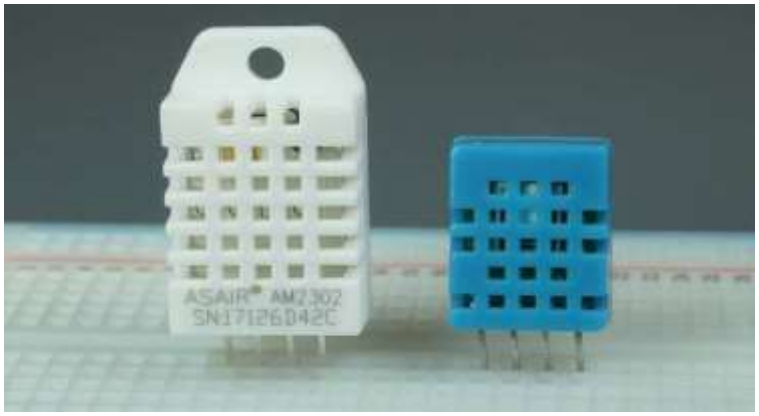
```
  delay(1000);
  display.startscrollleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  display.clearDisplay();

  // Scroll part of the screen
  display.setCursor(0,0);
  display.setTextSize(1);
  display.println("Scroll");
  display.println("some part");
  display.println("of the screen.");
  display.display();
  display.startscrollright(0x00, 0x00);
}
void loop() {
   // Scroll full screen
  display.setCursor(0,0);
  display.setTextSize(1);
  display.println("SSG");
  display.println("EMBEDDED");
```

```
  display.println("SOLUTIONS");
  display.display();
  display.startscrollright(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  display.clearDisplay();
}
```



DHT11

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity. These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.



The DHT22 sensor has a better resolution and a wider temperature and humidity measurement range. However, it is a bit more expensive, and you can only request readings with 2 seconds interval. The DHT33 has a smaller range and it's less accurate. However, you can request sensor readings every second. It's also a bit cheaper.

DHT11 Code

```
//connect D2 with DATA pin of DHT11

#include "DHT.h"


// Uncomment one of the lines below for whatever DHT sensor type you're using!

#define DHTTYPE DHT11 // DHT 11

//#define DHTTYPE DHT21  // DHT 21 (AM2301)

//#define DHTTYPE DHT22  // DHT 22  (AM2302), AM2321
```

```
// DHT Sensor
uint8_t DHTPin = 4;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;

void setup() {
  Serial.begin(115200);
  delay(100);

  pinMode(DHTPin, INPUT);

  dht.begin();
  }

void loop() {

 Temperature = dht.readTemperature(); // Gets the values of the temperature
  Serial.print("Temperature=");
  Serial.println(Temperature);
  Humidity = dht.readHumidity(); // Gets the values of the humidity
   Serial.print("Humidity=");
   Serial.println( Humidity);
   delay(1000);
```

}

DHT11 ThingSpeaks Code

```
//DHT11 is connected to  D4

#include <DHT.h>

#include <ESP8266WiFi.h>


#define DHTPIN 2          //DHT11 is connected to GPIO Pin 2 ieD4


String apiKey = "RFCN0AQCAF9B8YPB";     //  Enter your Write API key from ThingSpeak

const char* ssid =  "enter ssid";     // Enter your WiFi Network's SSID

const char* pass = "enter password"; // Enter your WiFi Network's Password

const char* server = "api.thingspeak.com";


float humi;

float temp;


DHT dht(DHTPIN, DHT11);
```

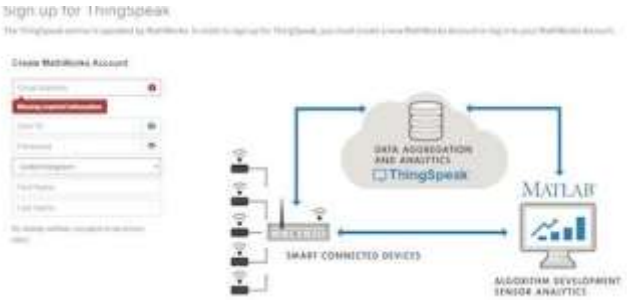
```
WiFiClient client;

void setup()
{
      Serial.begin(115200);
      delay(10);
      dht.begin();

      Serial.println("Connecting to ");
      Serial.println(ssid);


      WiFi.begin(ssid, pass);

     while (WiFi.status() != WL_CONNECTED)
    {
          delay(100);
          Serial.print("*");
    }
     Serial.println("");
     Serial.println("***WiFi connected***");

}

void loop()
{
```

```
    humi = dht.readHumidity();
    temp = dht.readTemperature();

    if (client.connect(server,80))  //   "184.106.153.149" or api.thingspeak.com
    {
     String sendData =
apiKey+"&field1="+String(temp)+"&field2="+String(humi)+"\r\n\r\n";

     //Serial.println(sendData);

     client.print("POST /update HTTP/1.1\n");
     client.print("Host: api.thingspeak.com\n");
     client.print("Connection: close\n");
     client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
     client.print("Content-Type: application/x-www-form-urlencoded\n");
     client.print("Content-Length: ");
     client.print(sendData.length());
     client.print("\n\n");
     client.print(sendData);

     Serial.print("Temperature: ");
     Serial.print(temp);
     Serial.print("deg C. Humidity: ");
     Serial.print(humi);
     Serial.println("%. Connecting to Thingspeak.");
     }
```

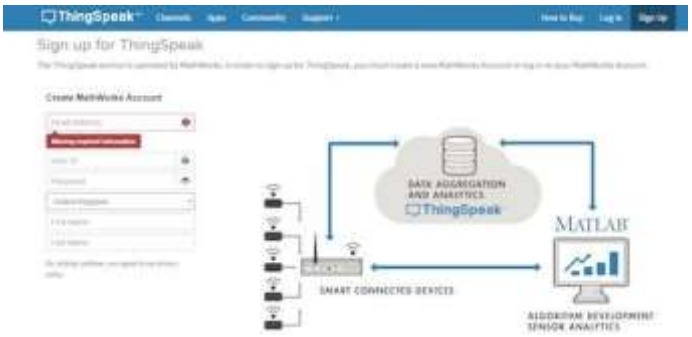
```
    client.stop();

    Serial.println("Sending. ..");

 delay(10000);
}
```

How to use thingspeak

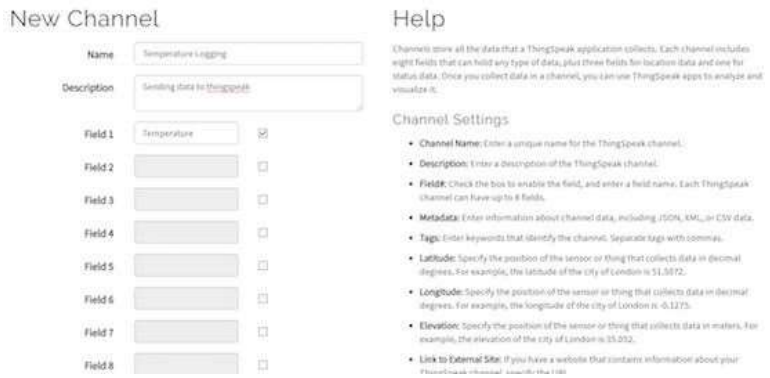
Step 1: Make Account With Thing Speak (MATLAB) and sign up

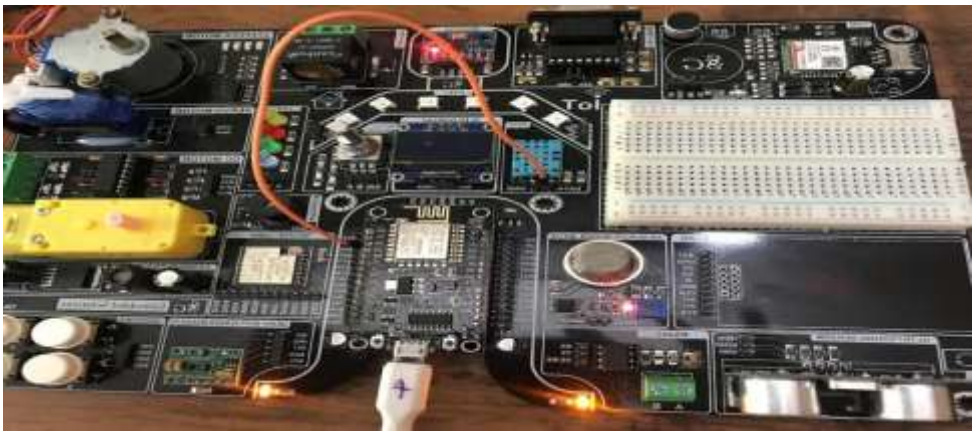


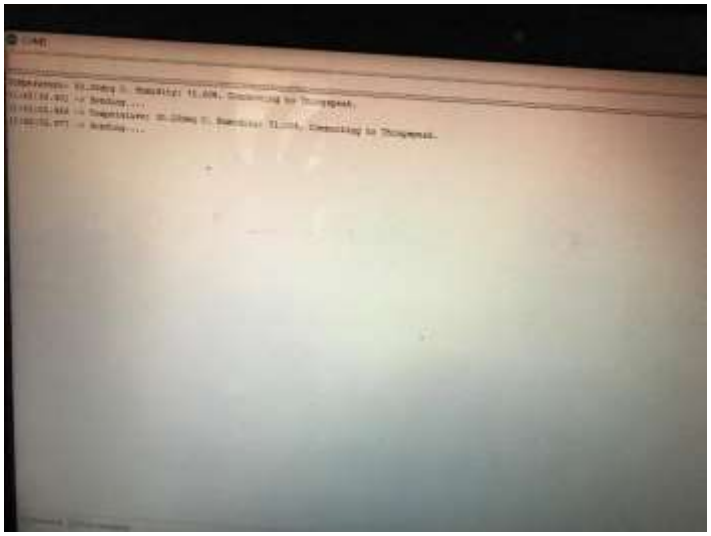
s



Step:2 Create channel


New Channel
Name
Temperature Logging
Description
Sending data to thingspeak
Field 1
Temperature
Field 2
Field 3
Field 4
Field 5
Field 6
Field 7
Field 8
Help
Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.
Channel Settings
Channel Name: Enter a unique name for the ThingSpeak channel.
Description: Enter a description of the ThingSpeak channel.
Field#: Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
Metadata: Enter information about channel data, including JSON, XML, or CSV data.
Tags: Enter keywords that identify the channel. Separate tags with commas.
Latitude: Specify the position of the sensor or thing that collects data in decimal degrees. For example, the latitude of the city of London is 51.5072.
Longitude: Specify the position of the sensor or thing that collects data in decimal degrees. For example, the longitude of the city of London is -0.1275.
Elevation: Specify the position of the sensor or thing that collects data in meters. For example, the elevation of the city of London is 35.052.
Link to External Site: If you have a website that contains information about your





NeoPixel LED

NeoPixel LED Strip Lights are programmable RGB LED strip which can be programmed to generate any desired lighting pattern. NeoPixel can produce multiple colors in any combination and brightness.

Features:

- Individually addressable RGB LEDs
- 36.8 million colors per pixel
- Single-wire digital control
- Operating Voltage: 5V DC
- Current Requirement: 60mA per LED
- Flexible LED structure
- 5050 RGB LED with WS2832 driver

NeoPixel LED Code

```
// NeoPixel test program showing use of the WHITE channel for RGBW

// pixels only (won't look correct on regular RGB NeoPixel strips).

//connect D1 to DIN of RGB strip

#include <Adafruit_NeoPixel.h>

#ifdef__AVR__

 #include <avr/power.h> // Required for 16 MHz Adafruit Trinket

#endif
```

```
// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1:
#define LED_PIN    5

// How many NeoPixels are attached to the Arduino?
#define LED_COUNT  120

// NeoPixel brightness, 0 (min) to 255 (max)
#define BRIGHTNESS 50 // Set BRIGHTNESS to about 1/5 (max = 255)

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRBW + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
// Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
//  NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//  NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)

void setup() {
  // These lines are specifically to support the Adafruit Trinket 5V 16 MHz.
  // Any other board, you can remove this part (but no harm leaving it):
#if defined(_AVR_ATtiny85_) && (F_CPU == 16000000)
  clock_prescale_set(clock_div_1);
```

```
#endif
  // END of Trinket-specific code.

  strip.begin();          // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();           // Turn OFF all pixels ASAP
  strip.setBrightness(BRIGHTNESS);
}

void loop() {
  // Fill along the length of the strip in various colors...
  colorWipe(strip.Color(255,   0,   0)     , 1); // Red
  colorWipe(strip.Color(  0, 255,   0)     , 1); // Green
  colorWipe(strip.Color(  0,   0, 255)     , 1); // Blue
  colorWipe(strip.Color(  0,  0,   0, 255), 1); // True white (not RGB white)

  whiteOverRainbow(75, 5);

  pulseWhite(5);

  rainbowFade2White(3, 3, 1);
}

// Fill strip pixels one after another with a color. Strip is NOT cleared
// first; anything there will be covered pixel by pixel. Pass in color
// (as a single 'packed' 32-bit value, which you can get by calling
// strip.Color(red, green, blue) as shown in the loop() function above),
// and a delay time (in milliseconds) between pixels.
```

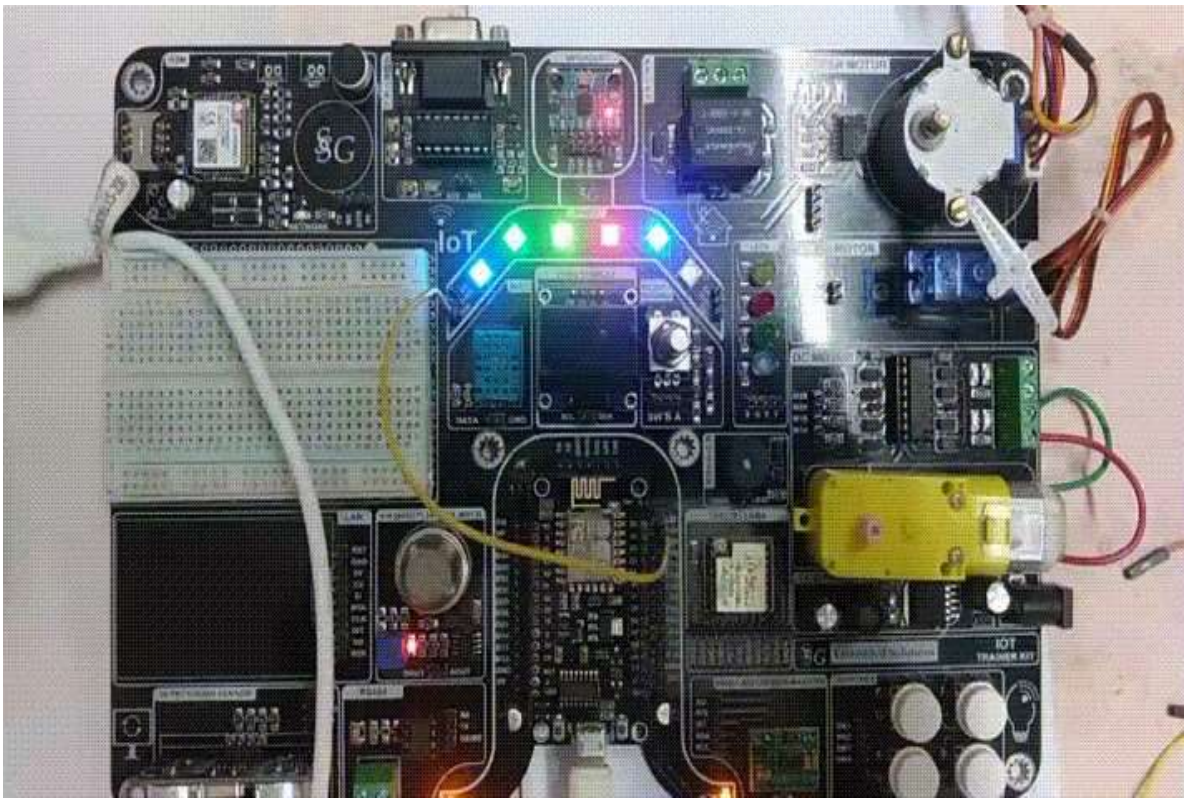
```
void colorWipe(uint32_t color, int wait) {
  for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
    strip.setPixelColor(i, color);// Set pixel's color (in RAM)
    strip.show();                 // Update strip to match
    delay(wait);                  //  Pause for a moment
  }
}

void whiteOverRainbow(int whiteSpeed, int whiteLength) {

  if(whiteLength >= strip.numPixels()) whiteLength = strip.numPixels() - 1;

  int     head          = whiteLength - 1;
  int     tail          = 0;
  int     loops         = 3;
  int     loopNum       = 0;
  uint32_t lastTime     = millis();
  uint32_t firstPixelHue = 0;

  for(;;) { // Repeat forever (or until a 'break' or 'return')
    for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
      if(((i >= tail) && (i <= head)) ||      //  If between head & tail...
         ((tail > head) && ((i >= tail) || (i <= head)))) {
        strip.setPixelColor(i, strip.Color(0, 0, 0, 255)); // Set white
      } else {                                    // else set rainbow
        int pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
        strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue)));
```

```
    }
   }

   strip.show(); // Update strip with new contents
   // There's no delay here, it just runs full-tilt until the timer and
   // counter combination below runs out.

   firstPixelHue += 40; // Advance just a little along the color wheel

   if((millis() - lastTime) > whiteSpeed) { // Time to update head/tail?
    if(++head >= strip.numPixels()) {      // Advance head, wrap around
    head = 0;
     if(++loopNum >= loops) return;
    }
    if(++tail >= strip.numPixels()) {      // Advance tail, wrap around
     tail = 0;
    }
    lastTime = millis();                   // Save time of last movement
   }
  }
}

void pulseWhite(uint8_t wait) {
  for(int j=0; j<256; j++) { // Ramp up from 0 to 255
   // Fill entire strip with white at gamma-corrected brightness level 'j':
   strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
   strip.show();
```

```
    delay(wait);
  }

  for(int j=255; j>=0; j--) { // Ramp down from 255 to 0
    strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
    strip.show();
    delay(wait);
  }
}

void rainbowFade2White(int wait, int rainbowLoops, int whiteLoops) {
  int fadeVal=0, fadeMax=100;

  // Hue of first pixel runs 'rainbowLoops' complete loops through the color
  // wheel. Color wheel has a range of 65536 but it's OK if we roll over, so
  // just count from 0 to rainbowLoops*65536, using steps of 256 so we
  // advance around the wheel at a decent clip.
  for(uint32_t firstPixelHue = 0; firstPixelHue < rainbowLoops*65536;
    firstPixelHue += 256) {

    for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...

      // Offset pixel hue by an amount to make one full revolution of the
      // color wheel (range of 65536) along the length of the strip
      // (strip.numPixels() steps):
      uint32_t pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
```

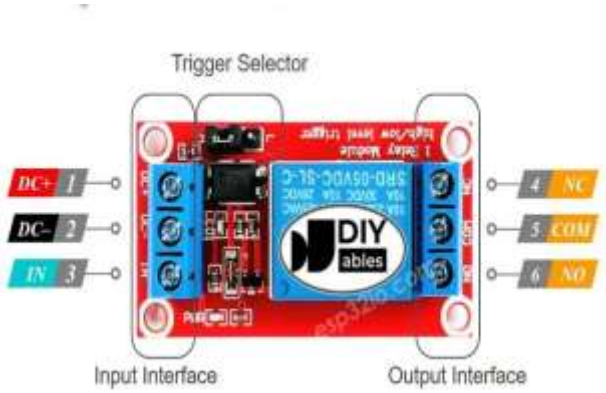
```
    // strip.ColorHSV() can take 1 or 3 arguments: a hue (0 to 65535) or
    // optionally add saturation and value (brightness) (each 0 to 255).
    // Here we're using just the three-argument variant, though the
    // second value (saturation) is a constant 255.
    strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue, 255,
      255 * fadeVal / fadeMax)));
  }

  strip.show();
  delay(wait);

  if(firstPixelHue < 65536) {                 // First loop,
    if(fadeVal < fadeMax) fadeVal++;          // fade in
  } else if(firstPixelHue >= ((rainbowLoops-1) * 65536)) { // Last loop,
    if(fadeVal > 0) fadeVal--;                // fade out
  } else {
    fadeVal = fadeMax; // Interim loop, make sure fade is at max
  }
}

for(int k=0; k<whiteLoops; k++) {
  for(int j=0; j<256; j++) { // Ramp up 0 to 255
    // Fill entire strip with white at gamma-corrected brightness level 'j':
    strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
    strip.show();
  }
  delay(1000); // Pause 1 second
```

```
  for(int j=255; j>=0; j--) { // Ramp down 255 to 0
    strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
    strip.show();
  }
 }

 delay(500); // Pause 1/2 second
}
```



Relay

Relays are the switches that aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts of an electronic circuit. When the relay contact is open (NO), the relay isn't energized with the open contact. However, if it is closed (NC), the relay isn't energized given the closed contact. However, when energy (electricity or charge) is supplied, the states are prone to change.

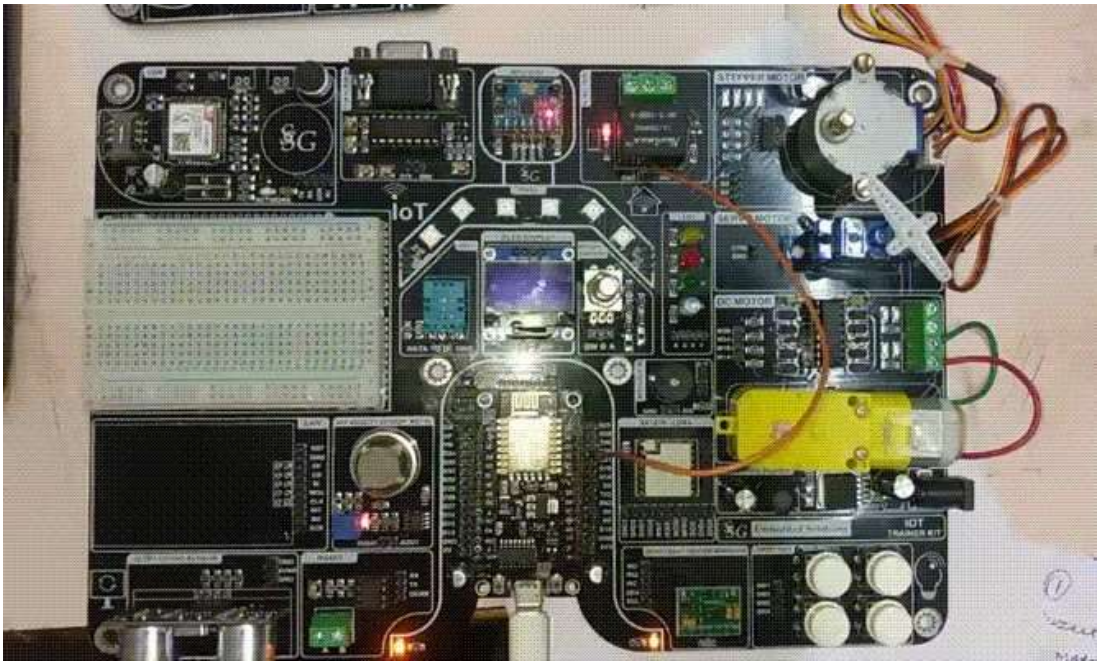


Relays are normally used in the control panels, manufacturing, and building automation to control the power along with switching the smaller current values in a control circuit. However, the supply of amplifying effect can help control the large amperes and voltages because if low voltage is applied to the relay coil, a large voltage can be switched by the contacts.

Relay Code

```
//connect D3 with SIG Pin


int RELAY1 = 0; // Assign LED pin i.e: D3 on NodeMCU

void setup()

{


// initialize GPIO 5 as an output
```

```
 pinMode(RELAY1, OUTPUT);
}
// the loop function runs over and over again forever
void loop()
{
 digitalWrite(RELAY1, HIGH); // turn the LED on
 delay(1000); // wait for a second
digitalWrite(RELAY1, LOW); // turn the LED off
delay(1000); // wait for a second
}
```



Buzzer

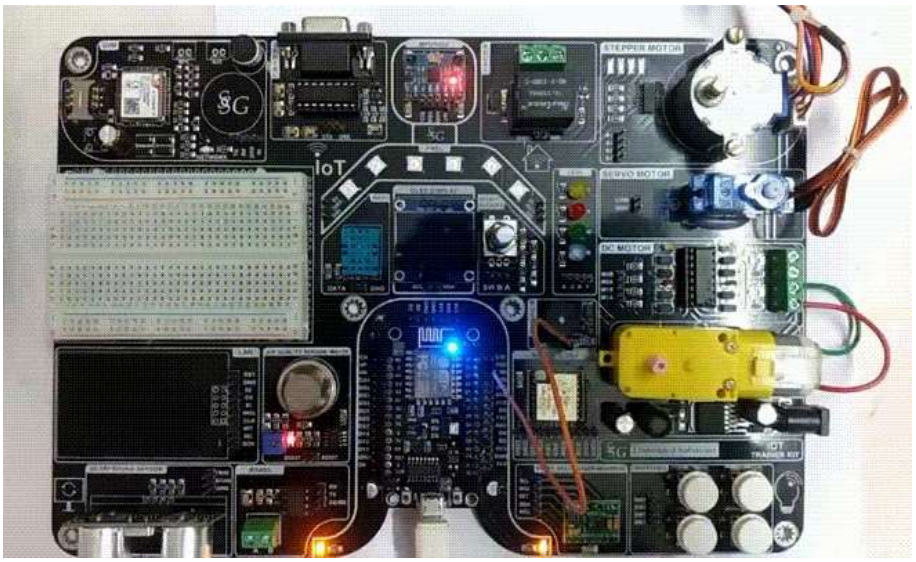
A **buzzer** is an electronic device that generates sound by converting electrical energy into sound energy. It typically consists of a piezoelectric crystal, which expands and contracts when an alternating current is applied to it, creating sound waves.

Buzzers are commonly used in a wide range of applications such as alarms, timers, and warning systems. They can also be used in electronic devices such as mobile phones, computers, and other electronic devices to generate different sounds and tones.

Buzzer Code

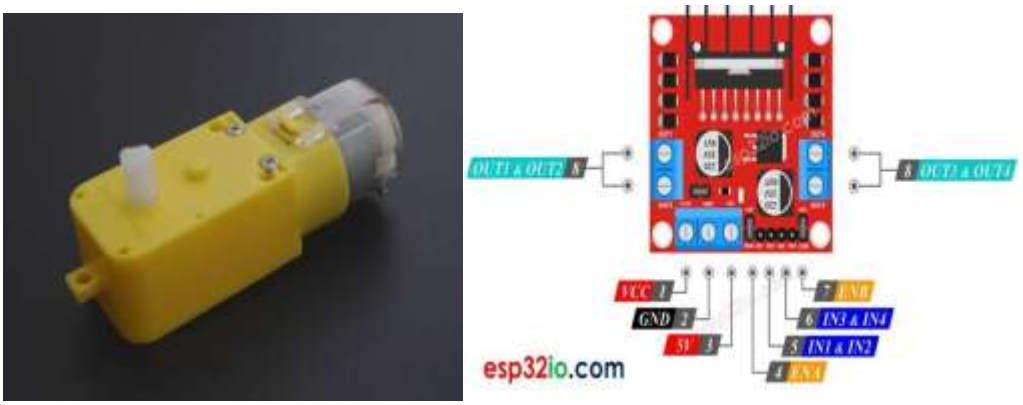
```
//connect D1, with BUZZER PINS

int BUZZER = 5; // Assign LED pin i.e: D1 on NodeMCU

void setup() {

// initialize GPIO 5 as an output

pinMode(BUZZER, OUTPUT);

}
```

```
// the loop function runs over and over again forever

void loop() {

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

delay(1000); // wait for a second

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

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



DC Motor

DC motor uses Direct Current (electrical energy) to produce mechanical movement i.e. rotational movement. When it converts electrical energy into mechanical energy then it is called as DC motor and when it converts mechanical energy into electrical energy then it is called as DC generator.



The working principle of DC motor is based on the fact that when a current carrying conductor is placed in a magnetic field, it experiences a mechanical force and starts rotating. Its direction of rotation depends upon Fleming's Left Hand Rule.

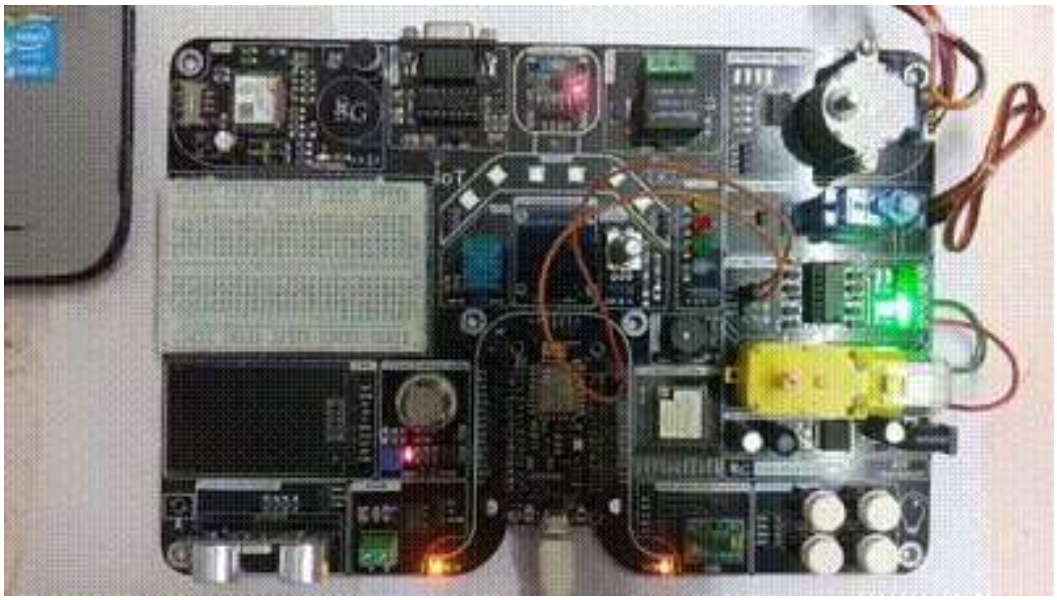
DC motors are used in many applications like robot for movement control, toys, quadcopters, CD/DVD disk drive in PCs/Laptops etc.

DC Motor Code

```
//CONNECT D1 & D2 WITH M1A & M1B OR M2A &M2B RESPECTIVELY

int ENA = D7;

int IN1 = D1;

int IN2 = D2;


void setup() {
```

```
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}
void loop() {
  setDirection();
  delay(1000);
  changeSpeed();
  delay(1000);
}
void setDirection() {
  analogWrite(ENA, 255);

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2,  LOW);
  delay(5000);

  digitalWrite(IN1,  LOW);
  digitalWrite(IN2, HIGH);
  delay(5000);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}
```

```
void changeSpeed() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);

  for (int i = 0; i < 256; i++) {
    analogWrite(ENA, i);
    delay(20);
  }

  for (int i = 255; i >= 0; --i) {
    analogWrite(ENA, i);
    delay(20);
  }
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}
```



Servo Motor

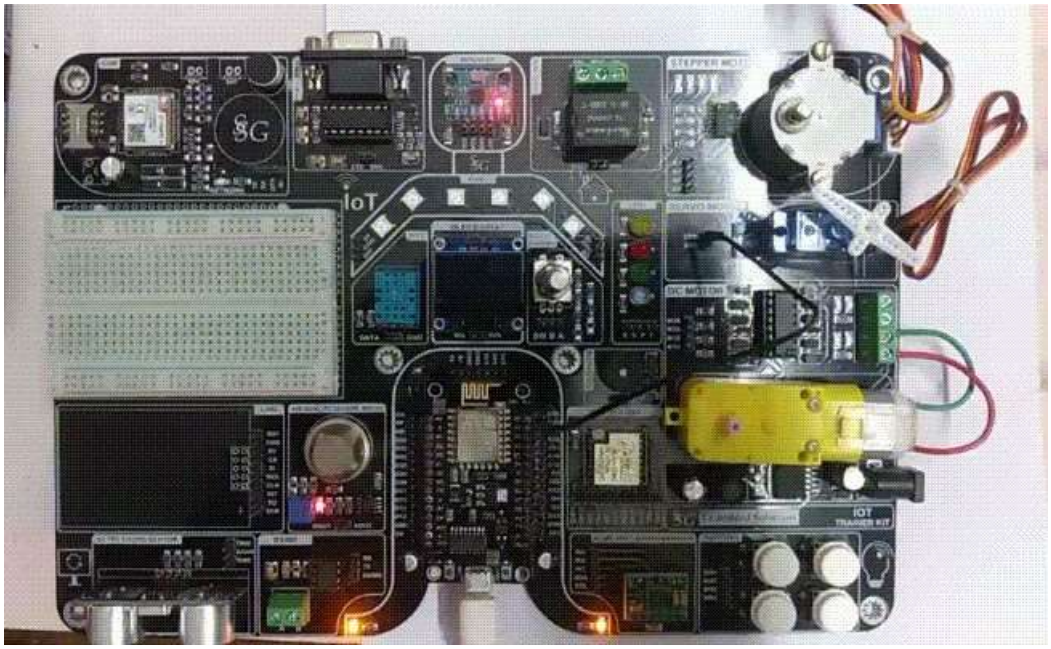
Servo motor is an electrical device which can be used to rotate objects (like robotic arm) precisely.Servo motor consists of DC motor with error sensing negative feedback mechanism. This allows precise control over angular velocity and position of motor. In some cases, AC motors are used.

It is a closed loop system where it uses negative feedback to control motion and final position of the shaft.It is not used for continuous rotation like conventional AC/DC motors.It has rotation angle that varies from 0° to 380°.

Servo Motor Code

```
//CONNECT D1 PIN to SERVO CTRL PIN

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position
```

```
void setup() {
  myservo.attach(5);  // attaches the servo on pin 9 to the servo object
}
void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
}
```



Stepper Motor

Stepper Motor is a brushless DC Motor. Control signals are applied to stepper motor to rotate it in steps.

Its speed of rotation depends upon rate at which control signals are applied. There are various stepper motors available with minimum required step angle.

Stepper motor is made up of mainly two parts, a stator and rotor. Stator is of coil winding and rotor is mostly permanent magnet or ferromagnetic material.

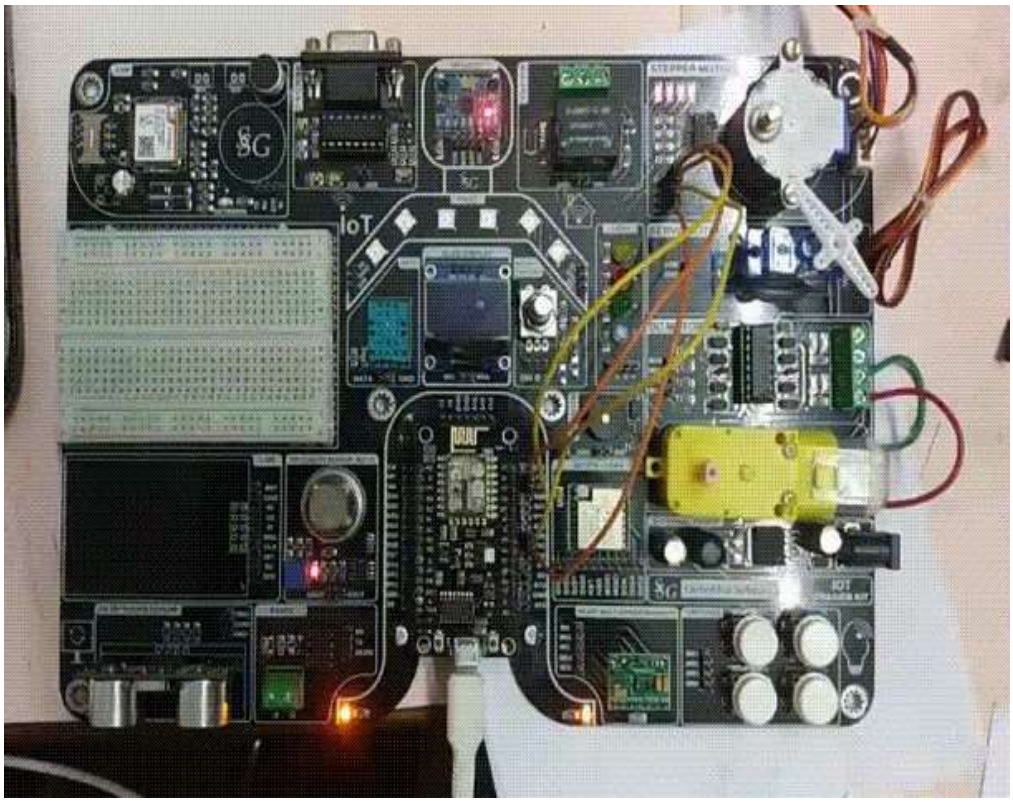
Step angle is the minimum angle that stepper motor will cover within one move/step. Number of steps required to complete one rotation depends upon step angle. Depending upon stepper motor configuration, step angle varies e.g. 0.72°, 3.8°, 3.75°, 7.5°, 35° etc.

Stepper Motor Code

```
//connect D1,D2,D5 &D6 with B1,B2,B3 & B4

#include <AccelStepper.h>
```

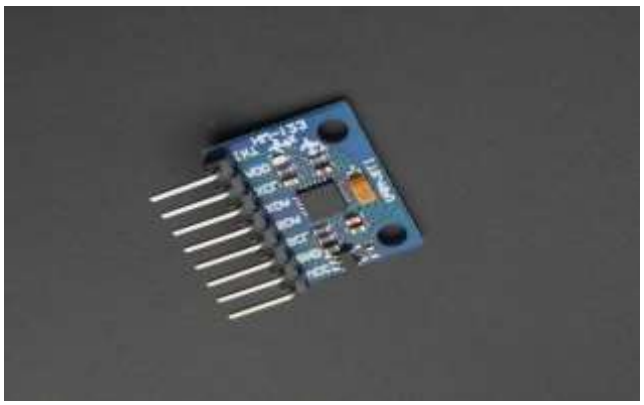
```
const int stepsPerRevolution = 2048; // change this to fit the number of steps per revolution

// ULN2003 Motor Driver Pins

#define IN1 5

#define IN2 4

#define IN3 14

#define IN4 12


// initialize the stepper library

AccelStepper stepper(AccelStepper::HALF4WIRE, IN1, IN3, IN2, IN4);


void setup() {

  // initialize the serial port

  Serial.begin(115200);


  // set the speed and acceleration

  stepper.setMaxSpeed(500);

  stepper.setAcceleration(100);


// set target position

  stepper.moveTo(stepsPerRevolution);

}


void loop() {

  // check current stepper motor position to invert direction

  if (stepper.distanceToGo() == 0){

    stepper.moveTo(-stepper.currentPosition());
```

```
  Serial.println("Changing direction");
 }

 // move the stepper motor (one step at a time)
 stepper.run();
}
```



MPU6050

MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.



It has Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor etc.

If 3-axis Magnetometer is connected to auxiliary I2C bus, then MPU6050 can provide complete 9-axis Motion Fusion output.

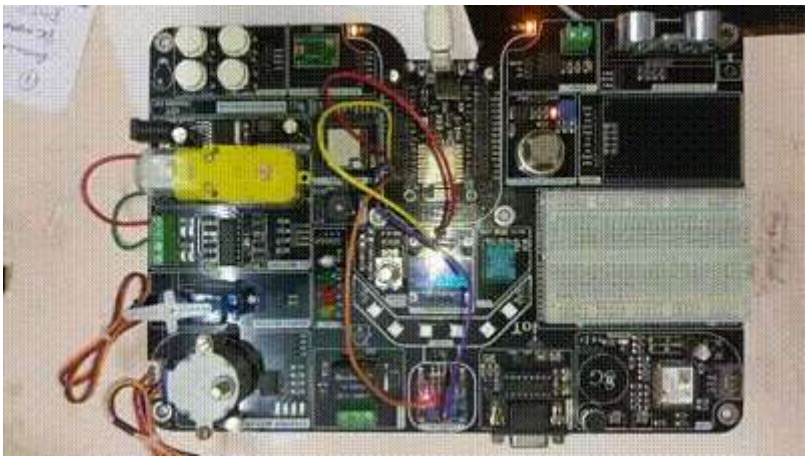
MPU6050 Code

```
//CONNECT D2 TO SDA & D1 TO SCL OF OLED & MPU6050

#include <Adafruit_MPU6050.h>

#include <Adafruit_SSD1306.h>

#include <Adafruit_Sensor.h>


Adafruit_MPU6050 mpu;

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 64, &Wire);

void setup() {
```

```
  Serial.begin(115200);
  // while (!Serial);
  Serial.println("MPU6050 OLED demo");

  if (!mpu.begin()) {
    Serial.println("Sensor init failed");
    while (1)
      yield();
  }
  Serial.println("Found a MPU-6050 sensor");

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }
  display.display();
  delay(500); // Pause for 2 seconds
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setRotation(0);
}

void loop() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
```

```
display.clearDisplay();
display.setCursor(0, 0);

Serial.print("Accelerometer ");
Serial.print("X: ");
Serial.print(a.acceleration.x, 1);
Serial.print(" m/s^2, ");
Serial.print("Y: ");
Serial.print(a.acceleration.y, 1);
Serial.print(" m/s^2, ");
Serial.print("Z: ");
Serial.print(a.acceleration.z, 1);
Serial.println(" m/s^2");

display.println("Accelerometer - m/s^2");
display.print(a.acceleration.x, 1);
display.print(", ");
display.print(a.acceleration.y, 1);
display.print(", ");
display.print(a.acceleration.z, 1);
display.println("");

Serial.print("Gyroscope ");
Serial.print("X: ");
Serial.print(g.gyro.x, 1);
Serial.print(" rps, ");
```

```
  Serial.print("Y: ");
  Serial.print(g.gyro.y, 1);
  Serial.print(" rps, ");
  Serial.print("Z: ");
  Serial.print(g.gyro.z, 1);
  Serial.println(" rps");

  display.println("Gyroscope - rps");
  display.print(g.gyro.x, 1);
  display.print(", ");
  display.print(g.gyro.y, 1);
  display.print(", ");
  display.print(g.gyro.z, 1);
  display.println("");

  display.display();
  delay(100);
}
```



MAX30300 Pulse and Heartbeat Sensor

The MAX30300 Pulse Oximeter is a medical device that is used to measure blood oxygen saturation levels, heart rate, and pulse strength.

It uses a non-invasive method to measure oxygen saturation levels in the blood. This module has a pair of LEDs (**L**ight **E**mitting **D**iode) that emit a monochromatic red light at a wavelength of 660nm and infrared light at a wavelength of 940nm.As the photodiode emits light, it falls on the finger and gets absorbed by the oxygenated blood rest light is reflected through the finger and falls on the detector.The detector detects and processes the signals and gives the output.The MAX30300 sensor works on the I2C Serial Communication protocol.

Specification

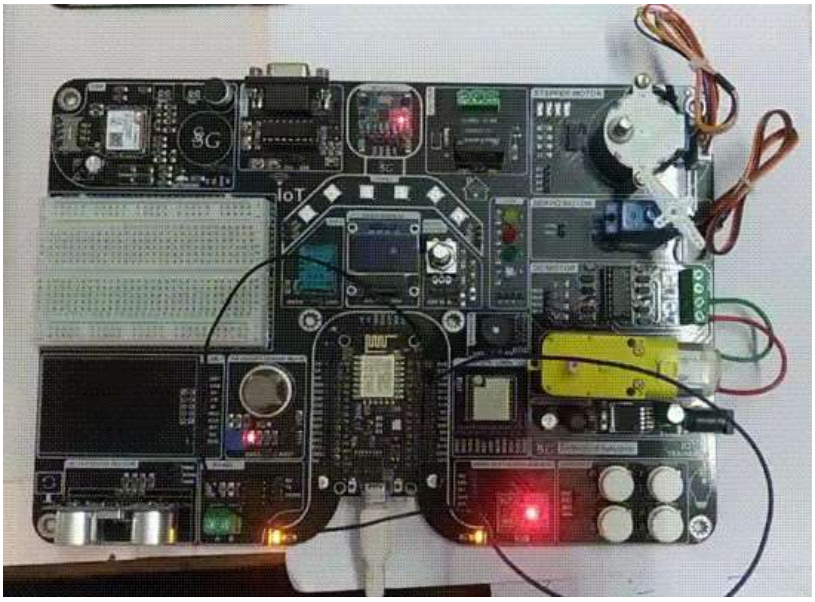
- Operating voltage of the module is 3.7V to 3.3V.
- Supply current of 3200uA.
- The operating temperature range of the module is -40C to +85C.
- LED Current range 0mA to 50 mA.
- LED Pulse width range from 200us to 3.6ms

MAX30300 Pulse and Heartbeat Sensor Code

```
//connect D2,D1 to SDA,SCL of HEART RATE SENSOR and also connect VCC=3V &
GND=GND of HEART RATE SENSOR module to NODE MCU8266 BOARD

#include <Wire.h>

#include "MAX30100_PulseOximeter.h"


#define REPORTING_PERIOD_MS     1000


// PulseOximeter is the higher level interface to the sensor

// it offers:

//  * beat detection reporting

//  * heart rate calculation

//  * SpO2 (oxidation level) calculation

PulseOximeter pox;


uint32_t tsLastReport = 0;


// Callback (registered below) fired when a pulse is detected

void onBeatDetected()

{

   Serial.println("Beat!");

}


void setup()

{

   Serial.begin(115200);
```

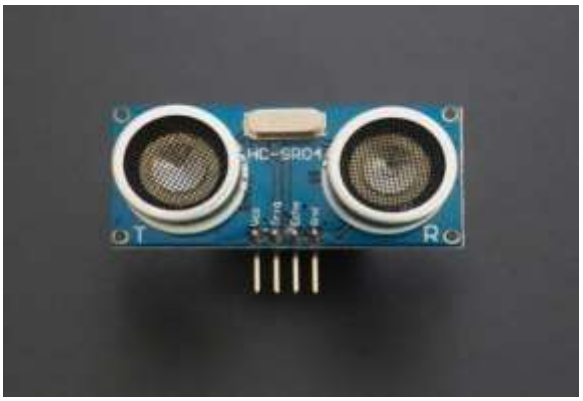
```
    Serial.print("Initializing pulse oximeter..");

    // Initialize the PulseOximeter instance
    // Failures are generally due to an improper I2C wiring, missing power supply
    // or wrong target chip
    if (!pox.begin()) {
        Serial.println("FAILED");
        for(;;);
    } else {
        Serial.println("SUCCESS");
    }

    // The default current for the IR LED is 50mA and it could be changed
    //  by uncommenting the following line. Check MAX30100_Registers.h for all the
    //  available options.
    // pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);

    // Register a callback for the beat detection
    pox.setOnBeatDetectedCallback(onBeatDetected);
}

void loop()
{
    // Make sure to call update as fast as possible
    pox.update();

    // Asynchronously dump heart rate and oxidation levels to the serial
```

```
    // For both, a value of 0 means "invalid"
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        Serial.print("Heart rate:");
        Serial.print(pox.getHeartRate());
        Serial.print("bpm / SpO2:");
        Serial.print(pox.getSpO2());
        Serial.println("%");

        tsLastReport = millis();
    }
}
```



Ultrasonic

Ultrasonic Module HC-SR04 works on the principle of SONAR and RADAR systems. It can be used to determine the distance of an object in the range of 2 cm – 400 cm. An ultrasonic sensor generates high-frequency sound (ultrasound) waves. When this ultrasound hits the object, it reflects as echo which is sensed by the receiver



HC-SR-04 has an ultrasonic transmitter, receiver and control circuit.

In the ultrasonic module HCSR04, we have to give trigger pulse, so that it will generate ultrasound of frequency 40 kHz. After generating ultrasound i.e. 8 pulses of 40 kHz, it makes echo pin high. Echo pin remains high until it does not get the echo sound back. So the width of echo pin will be the time for sound to travel to the object and return back. Once we get the time we can calculate distance, as we know the speed of sound.

Ultrasonic Code

```
//CONNECT D6 TO TRIG & D5 TO ECHO

const int trigPin = 12;

const int echoPin = 14;


//define sound velocity in cm/uS

#define SOUND_VELOCITY 0.034
```

```
#define CM_TO_INCH 0.393701

long duration;
float distanceCm;
float distanceInch;

void setup() {
  Serial.begin(115200); // Starts the serial communication
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
}

void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance
  distanceCm = duration * SOUND_VELOCITY/2;
```

```
  // Convert to inches
  distanceInch = distanceCm * CM_TO_INCH;

  // Prints the distance on the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);
  Serial.print("Distance (inch): ");
  Serial.println(distanceInch);

  delay(1000);
}
```

RS232 Serial Port

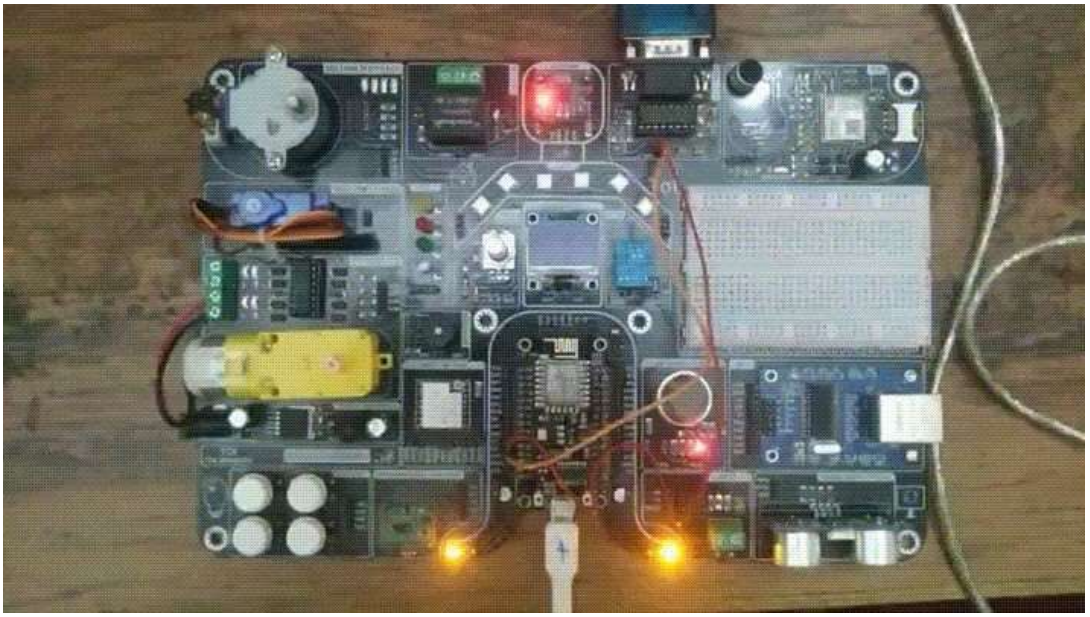
The term RS232 stands for "Recommended Standard 232" and it is a type of serial communication used for transmission of data normally in medium distances. It was introduced back in the 3960s and has found its way into many applications like computer printers, factory automation devices etc. Today there are many modern communication protocols like the RS485, SPI, I2C, CAN etc..

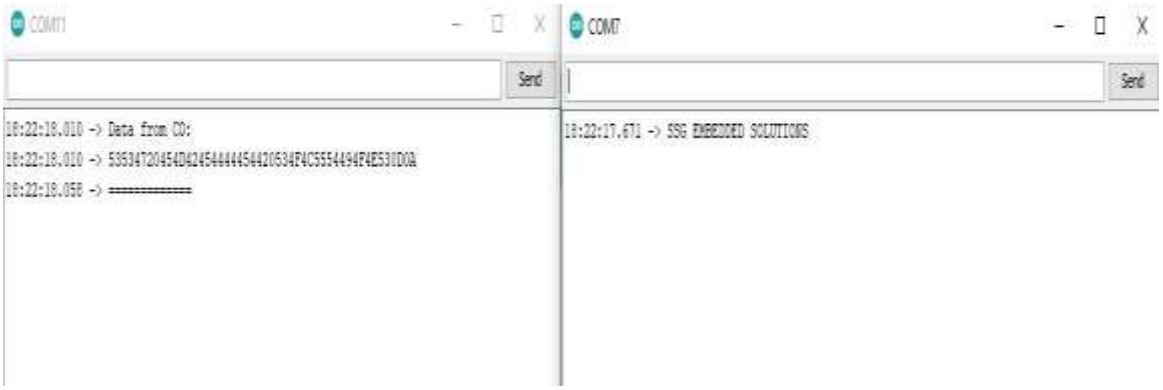
RS232 works on the two-way communication that exchanges data to one another. There are two devices connected to each other, **(DTE) Data Transmission Equipment& (DCE) Data Communication Equipment** which has the pins like **TXD, RXD, and RTS& CTS**. Now, from **DTE** source, the **RTS** generates the *request to send* the data. Then from the other side **DCE**, the **CTS,** clears the path for receiving the data. After clearing a path, it will give a signal to **RTS** of the **DTE** source to send the signal. Then the bits are transmitted from **DTE** to **DCE**. Now again from **DCE** source, the request can be generated by **RTS** and **CTS** of **DTE** sources clears the path for receiving the data and gives a signal to send the data.

RS232 Serial Port Code

```
#include <SoftwareSerial.h>

int rx_pin = 3;
int tx_pin = 4;

SoftwareSerial COSerial(rx_pin, tx_pin); // RX, TX
void setup()
 {
  pinMode (rx_pin, INPUT_PULLUP);
  Serial.begin(9600);
  while (!Serial);
  Serial.println("HW Serial - Ready");

  COSerial.begin(9600);
  while (!COSerial);
  Serial.println("SW Serial - Ready");
}

String readSerial()
 {
  int inChar;
  String inStr = "";
  char buff[2];
  long startTime = millis();
  if (COSerial.available())
 {
```

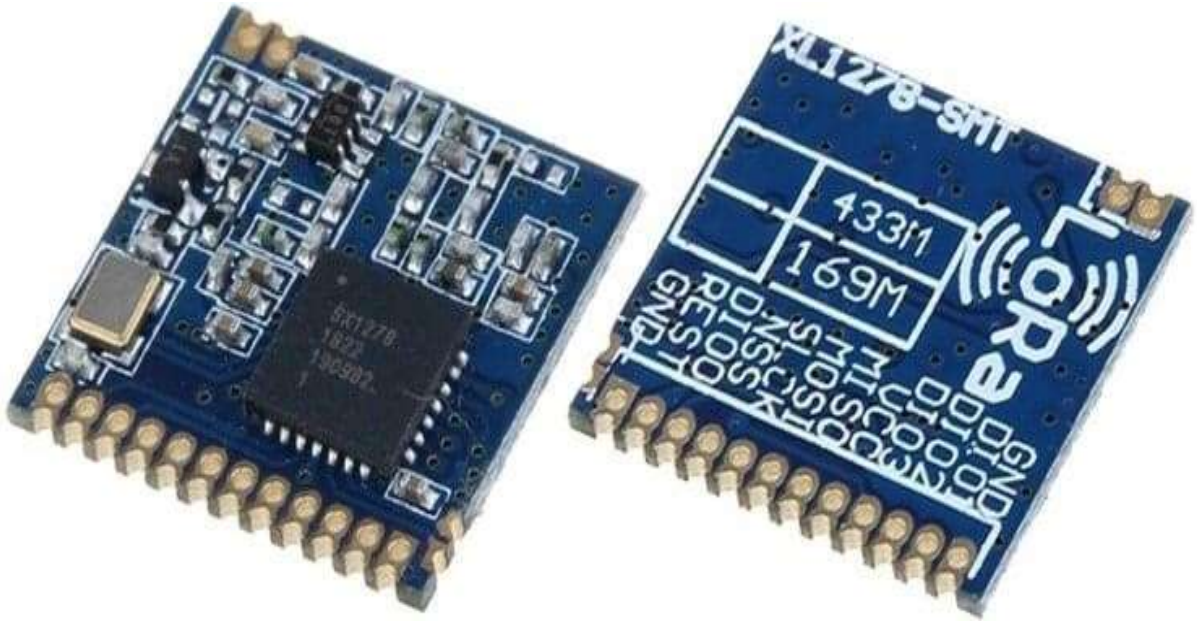
```
   while (millis() - startTime < 300)
{
     inChar = -1;
     inChar = COSerial.read();
     if (inChar > -1)
 {
       sprintf(buff,"%02X",inChar);
       inStr = inStr + buff;
     }
   }
 }
 return inStr;
}
void loop()
 {
 String fromSerial = readSerial();
 if (fromSerial.length() > 0)
 {
   Serial.println("Data from CO:");
   Serial.println(fromSerial);
   Serial.println("=============");
 }
}
```




COM11
Send
18:22:18.010 -> Data from CO:
18:22:18.010 -> 53534720454D4245444444454420534F4C5554494F4E531D0A
18:22:18.050 -> ==========
COM7
Send
18:22:17.671 -> SSG EMBEDDED SOLUTIONS

LORA

The **SX3276/77/78/79** transceivers feature the LoRa® long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimizing current consumption.



SX3278 can achieve a sensitivity of over **-348dBm** using a low-cost crystal. The high sensitivity combined with the integrated **+20dBm power amplifier** yields industry leading link budget making it optimal for any application requiring range or robustness. Lora SX3278 also provides significant advantages in both blocking and selectivity over conventional **modulation techniques**, solving the traditional design compromise between range, interference immunity and energy consumption.

LORA Transmitter Code

////Connect D8,D7,D6,D5,D0 & D1 to LORA SX1278---NSS,MOSI,MISO,SCK,RST & DIO0 respectively

#include <SPI.h>

#include <LoRa.h>

```
//=====================================================================
==//

#define PIN_LORA_COPI   13
#define PIN_LORA_CIPO   12
#define PIN_LORA_SCK    14
#define PIN_LORA_CS     15
#define PIN_LORA_RST    16
#define PIN_LORA_DIO0  5

#define LORA_FREQUENCY  433E6
int counter = 0;
void setup() {
 Serial.begin (115200);
 while (!Serial);
 delay (1500);
 Serial.println ("LoRa Sender");

 LoRa.setPins (PIN_LORA_CS, PIN_LORA_RST, PIN_LORA_DIO0);
 LoRa.setSPIFrequency (20000000);
 LoRa.setTxPower (20);

 if (!LoRa.begin (LORA_FREQUENCY)) {
  Serial.println ("Starting LoRa failed!");
  while (1);
 }
 else {
```

```
   Serial.print ("LoRa initialized with frequency ");
   Serial.println (LORA_FREQUENCY);
  }
}
void loop() {
  Serial.print ("Sending packet: ");
  Serial.println (counter);

  // send packet
  LoRa.beginPacket();
  LoRa.print ("Hello LoRa ");
  LoRa.print (counter);
  LoRa.endPacket();

  counter++;

  delay (1000);
}
```

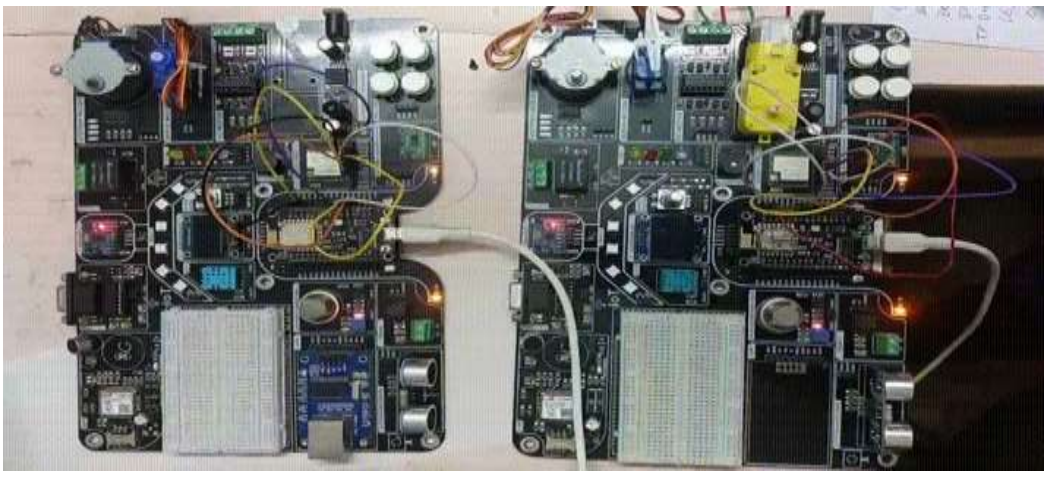
LORA Receiver Code

////Connect D8,D7,D6,D5,D0 & D1 to LORA SX1278---NSS,MOSI,MISO,SCK,RST & DIO0 respectively

```
#include <SPI.h>
#include <LoRa.h>
```

```
#define PIN_LORA_COPI   13
#define PIN_LORA_CIPO   12
#define PIN_LORA_SCK    14
#define PIN_LORA_CS     15
#define PIN_LORA_RST    16
#define PIN_LORA_DIO0  5

#define LORA_FREQUENCY  433E6

void setup() {
  Serial.begin (115200);
  while (!Serial);
  delay (1500);
  Serial.println ("LoRa Receiver");

  LoRa.setPins (PIN_LORA_CS, PIN_LORA_RST, PIN_LORA_DIO0);
  LoRa.setSPIFrequency (20000000);

  if (!LoRa.begin (LORA_FREQUENCY)) {
    Serial.println ("Starting LoRa failed!");
    while (1);
  }
  else {
    Serial.print ("LoRa initialized with frequency ");
    Serial.println (LORA_FREQUENCY);
  }
}
```

```
void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();

  if (packetSize) {
    // received a packet
    Serial.print ("Received packet '");

    // read packet
    while (LoRa.available()) {
      Serial.print ((char) LoRa.read());
    }

    // print RSSI of packet
    Serial.print ("' with RSSI ");
    Serial.println (LoRa.packetRssi());
  }
}
```



GSM

The SIM800C is a Quad-Band GSM/GPRS module in a LCC type which supports GPRS up to 85.6kbps data transfer. It has strong extension capability with abundant interfaces including UART, USB2.0, GPIO etc. The module provides much flexibility and ease of integration for customer's applications

General features

Frequency Band: 850/900/3800/3900MHz

GPRS multi-slot cass: 32/30

Compliment GSM phase 2/2+: Class 4 (2w 850/900MHz) and Class 3 (3w 3800/3900MHz)

Control via AT commands (3GPP TP 27.007, 27.005 & SIMCom enhanced AT Commands)

Low power comsumption

GPRS mobile station class B, SMS cell boardcast

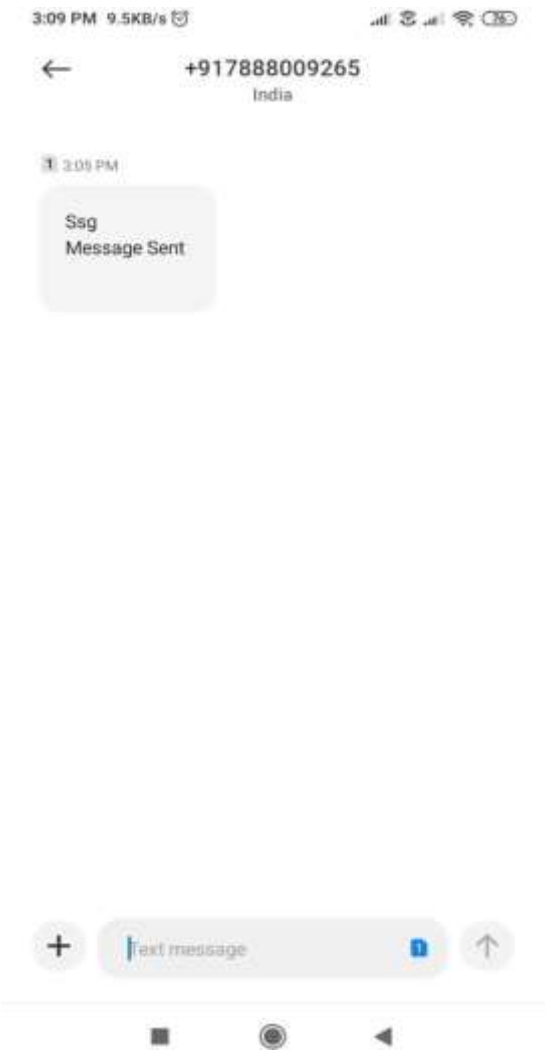
Embedded TCP/UDP prototcol, FTP/HTTP. Audio record, SSL/TLC, Speech code mode

GSM Code

```
//Connect TX2 to RX of GSM & RX2 to TX of GSM


void setup() {
  Serial.begin(9600);
  Serial.begin(9600);
  delay(3000);
  test_sim800_module();
  send_SMS();
}
void loop() {
  updateSerial();
}
void test_sim800_module()
{
  Serial.println("AT");
  updateSerial();
  Serial.println();

  Serial.println("AT+CSQ");
  updateSerial();

  Serial.println("AT+CCID");
  updateSerial();

  Serial.println("AT+CREG?");
  updateSerial();
```

```
  Serial.println("ATI");
  updateSerial();
  Serial.println("AT+CBC");
  updateSerial();
}
void updateSerial()
{
  delay(500);
  while (Serial.available())
  {
    Serial.write(Serial.read());//Forward what Serial received to Software Serial Port
  }
  while (Serial.available())
  {
    Serial.write(Serial.read());//Forward what Software Serial received to Serial Port
  }
}
void send_SMS()
{
  Serial.println("AT+CMGF=1"); // Configuring TEXT mode
  updateSerial();

  Serial.println("AT+CMGS=\"08482937644\"");//change ZZ with country code and
xxxxxxxxxxx with phone number to sms

  updateSerial();
  Serial.print("Ssg"); //text content

  updateSerial();
```

```
Serial.println();


  Serial.println("Message Sent");
  Serial.write(26);
}
```



Rotary Encoder

A rotary encoder is a type of position sensor that converts the angular position (rotation) of a knob into an output signal that can be used to determine which direction the knob is turned.

Rotary encoders are classified into two types: absolute and incremental. The absolute encoder reports the exact position of the knob in degrees, whereas the incremental encoder reports the number of increments the shaft has moved.

Rotary encoders are the modern digital equivalent of potentiometers. It can rotate 360° without stopping

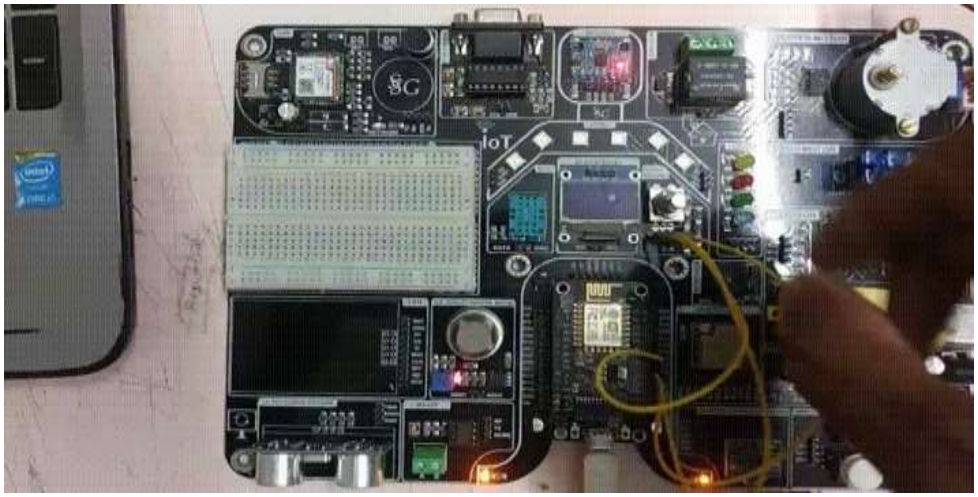
Rotary Encoder Code

```
//Set SERIAL MONITER BAUD RATE 9600

// CLK    ...  PIN D5 --- OUT_A

// DT     ...  PIN D6 ----- OUT_B

// SW     ...  PIN D7-------- SW

// Rotary Encoder Inputs

#define CLK 14

#define DT 12
```

```
#define SW 13

int counter = 0;
int currentStateCLK;
int lastStateCLK;
String currentDir ="";
unsigned long lastButtonPress = 0;

void setup() {

  // Set encoder pins as inputs
  pinMode(CLK,INPUT);
  pinMode(DT,INPUT);
  pinMode(SW, INPUT_PULLUP);

  // Setup Serial Monitor
  Serial.begin(9600);

  // Read the initial state of CLK
  lastStateCLK = digitalRead(CLK);
}

void loop() {

  // Read the current state of CLK
  currentStateCLK = digitalRead(CLK);
```

```
// If last and current state of CLK are different, then pulse occurred
// React to only 1 state change to avoid double count
if (currentStateCLK != lastStateCLK  && currentStateCLK == 1){

  // If the DT state is different than the CLK state then
  // the encoder is rotating CCW so decrement
  if (digitalRead(DT) != currentStateCLK) {
  counter --;
    currentDir ="CCW";
  } else {
    // Encoder is rotating CW so increment
    counter ++;
    currentDir ="CW";
  }

  Serial.print("Direction: ");
  Serial.print(currentDir);
  Serial.print(" | Counter: ");
  Serial.println(counter);
}

// Remember last CLK state
lastStateCLK = currentStateCLK;

// Read the button state
int btnState = digitalRead(SW);
```

```
  //If we detect LOW signal, button is pressed
  if (btnState == LOW) {
    //if 50ms have passed since last LOW pulse, it means that the
    //button has been pressed, released and pressed again
    if (millis() - lastButtonPress > 50) {
      Serial.println("Button pressed!");
    }

    // Remember last button press event
    lastButtonPress = millis();
  }

  // Put in a slight delay to help debounce the reading
  delay(1);
}
```



Air Quality MQ335

The MQ-135 “air quality” sensor is part of the MQ-135 sensor belongs to the MQ series that are used to detect different gasses present in the air. The MQ-135 sensor is used to detect gases such as NH3,NOx, alcohol, Benzene, smoke,CO2 ,etc. steel exoskeleton houses a sensing device within the gas sensor module.

The analog output voltage lies between 0-5V where the output voltage increases relatively with the concentration of gas vapors coming in contact with the sensor. Under standard conditions, this output voltage from the sensor is directly proportional to the concentration of CO2 gas in PPM. This output voltage is converted to a digital value (0-1023) via the analog to digital converter in Arduino. This value is equal to the gas concentration in PPM.

Air Quality MQ335 Code

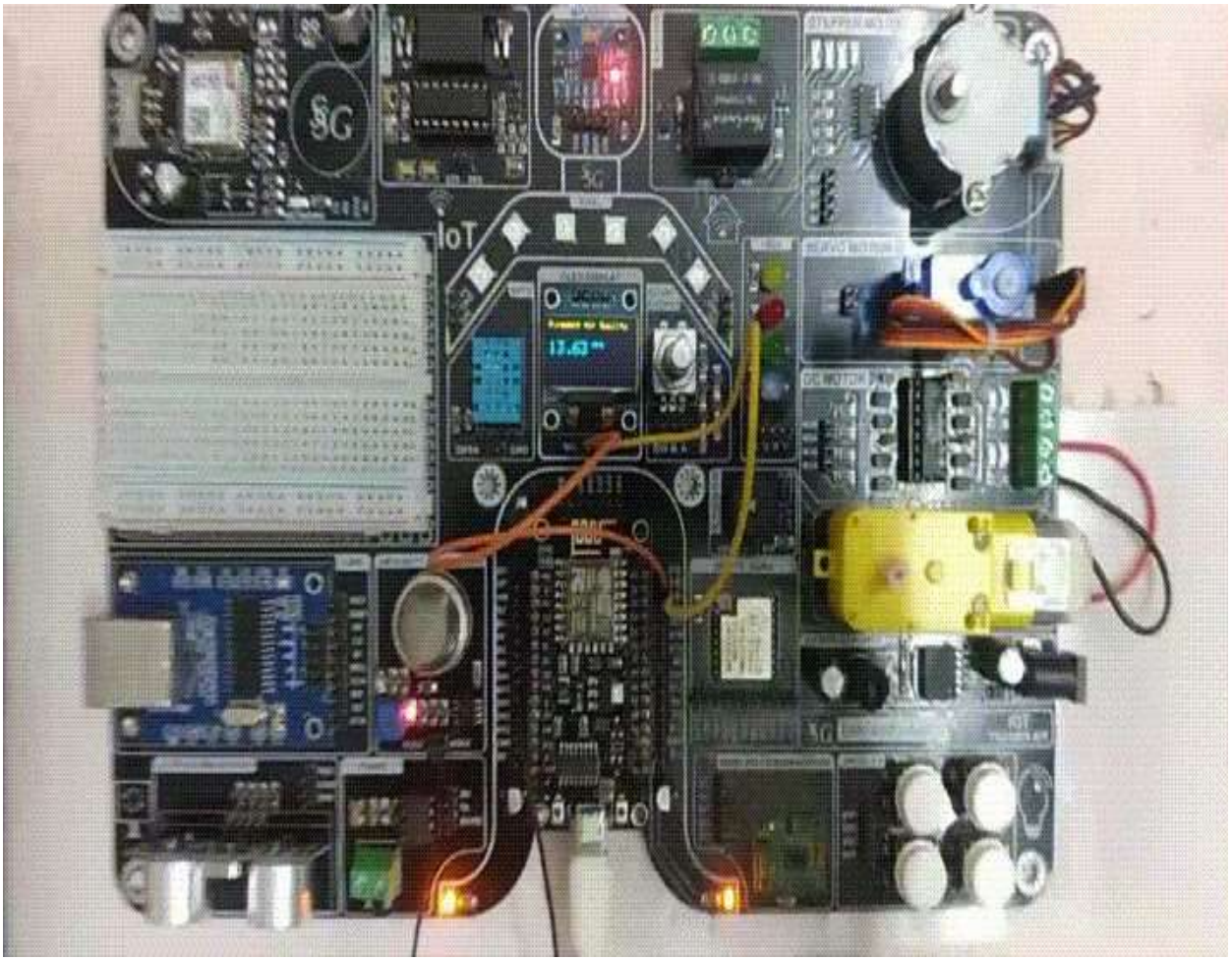
```
//A0 is connected to AOUT pin of MQ135

#include <ESP8266WiFi.h>

#include <SPI.h>

#include <Wire.h>

#include "MQ135.h"
```

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128    // OLED display width, in pixels
#define SCREEN_HEIGHT 64    // OLED display height, in pixels
#define OLED_RESET -1       // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

String apiKey = "4YX1CCOH67CT53F0"; // Enter your Write API key from ThingSpeak
const char *ssid = "ssg solutions 2 4 ghz";     // replace with your wifi ssid and wpa2 key
const char *pass = "ssgabhay";
const char* server = "api.thingspeak.com";

WiFiClient client;



void setup()
{
  Serial.begin(115200);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C (128x64)
  display.clearDisplay();
  delay(10);

  Serial.println("Connecting to ");
  Serial.println(ssid);
```

```
 display.clearDisplay();
 display.setCursor(0,0);
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.println("Connecting to ");
 display.setTextSize(2);
 display.print(ssid);
 display.display();

 WiFi.begin(ssid, pass);

 while (WiFi.status() != WL_CONNECTED)
 {
  delay(500);
  Serial.print(".");
 }
  Serial.println("");
  Serial.println("WiFi connected");

  display.clearDisplay();
  display.setCursor(0,0);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.print("WiFi connected");
  display.display();
  delay(4000);
}
```

```
void loop()
{
  MQ135 gasSensor = MQ135(A0);
  float air_quality = gasSensor.getPPM();
  Serial.print("Air Quality: ");
  Serial.print(air_quality);
  Serial.println(" PPM");
  Serial.println();

  display.clearDisplay();
  display.setCursor(0,0);  //oled display
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.println("P.resent Air Quality");


  display.setCursor(0,20);  //oled display
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.print(air_quality);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.println(" PPM");
  display.display();
```

```
  if (client.connect(server, 80)) // "184.106.153.149" or api.thingspeak.com
 {
  String postStr = apiKey;
  postStr += "&field1=";
  postStr += String(air_quality);
  postStr += "r\n";

  client.print("POST /update HTTP/1.1\n");
  client.print("Host: api.thingspeak.com\n");
  client.print("Connection: close\n");
  client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
  client.print("Content-Type: application/x-www-form-urlencoded\n");
  client.print("Content-Length: ");
  client.print(postStr.length());
  client.print("\n\n");
  client.print(postStr);

  Serial.println("Data Send to Thingspeak");
 }
  client.stop();
  Serial.println("Waiting...");

  delay(2000);     // thingspeak needs minimum 15 sec delay between updates.
}
```


SG
IoT