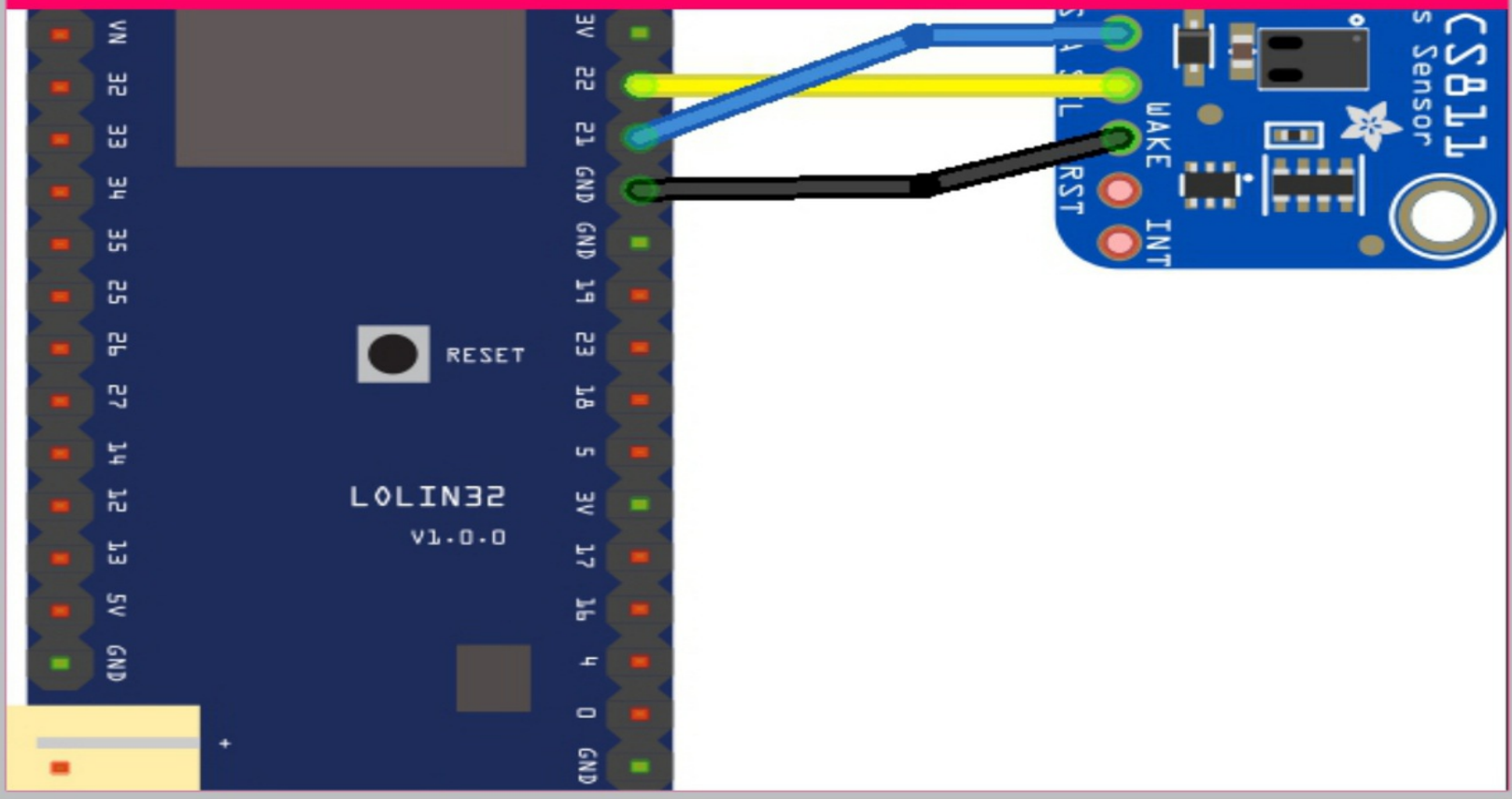
ESP32 Development using the Arduino IDE

IAIN HENDRY



ESP32 Development using the Arduino IDE

by

Iain Hendry

Contents

ESP32 and RFID-RC522 module example
LM35 and ESP32 example
ESP32 and MS5611 barometric pressure sensor example
ESP32 and MPL3115A2 absolute pressure sensor example
VEML6075 ultraviolet (UV) light sensor and ESP32
ESP32 and CCS811 gas sensor example
ESP32 and MPU-9250 sensor example
ESP32 and Max7219 8×8 LED matrix example
ESP32 and TM1637 7 segment display example
ESP32 and MAX44009 ambient light sensor example
ESP32 and OLED display example
ESP32 and Infrared receiver example
ESP32 and SD card example
MH ET LIVE ESP32 MINI KIT and WS2812B shield example
ESP32 and basic TEA5767 example
ESP32 and I2C LCD example
ESP32 and a Stepper motor
ESP32 and L9110 fan module example
ESP32 and GY-21P readings on a web page
ESP32 and CCS811 gas sensor data to Thingspeak example
In Review

About the ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations and includes in-built antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company, and is manufactured by TSMC using their 40 nm process. It is a successor to the ESP8266 microcontroller.

Features of the ESP32 include the following:

Processors:
CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
Ultra low power (ULP) co-processor
Memory: 520 KiB SRAM

Wireless connectivity:
Wi-Fi: 802.11 b/g/n
Bluetooth: v4.2 BR/EDR and BLE

Peripheral interfaces:
12-bit SAR ADC up to 18 channels
2 × 8-bit DACs
10 × touch sensors (capacitive sensing GPIOs)
4 × SPI
2 × I²S interfaces
2 × I²C interfaces
3 × UART
SD/SDIO/CE-ATA/MMC/eMMC host controller
SDIO/SPI slave controller
Ethernet MAC interface with dedicated DMA and IEEE 1588 Precision Time Protocol support
CAN bus 2.0
Infrared remote controller (TX/RX, up to 8 channels)
Motor PWM

LED PWM (up to 16 channels)
Hall effect sensor
Ultra low power analog pre-amplifier

Security:
IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
Secure boot
Flash encryption
1024-bit OTP, up to 768-bit for customers
Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)

Power management:
Internal low-dropout regulator
Individual power domain for RTC
5μA deep sleep current
Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Setting up the Arduino IDE

As you may have guessed from the title of this book we will be using the Arduino IDE, it is easy enough to setup the IDE and enable ESP32 support

Installation instructions using Arduino IDE Boards Manager

Starting with 1.6.4, Arduino allows installation of third-party platform packages using Boards Manager. There are packages available for Windows, Mac OS, and Linux (32 and 64 bit).

Install the current Arduino IDE 1.8 or later. The current version is available from the Arduino website.

Start Arduino and open Preferences window.

Enter https://dl.espressif.com/dl/package_esp32_index.json into Additional Board Manager URLs field. You can add multiple URLs, separating them with commas.

Open Boards Manager from Tools > Board menu and install esp32 platform (and don't forget to select your ESP32 board from Tools > Board menu after installation).

Stable release link: https://dl.espressif.com/dl/package_esp32_index.json

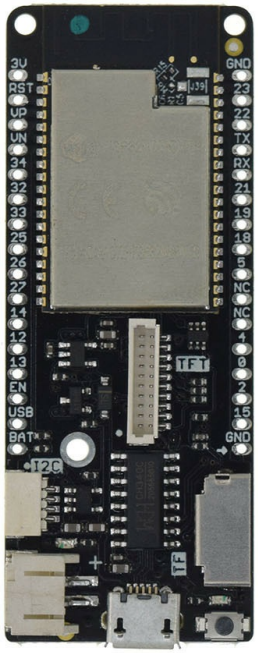
Development release link:
https://dl.espressif.com/dl/package_esp32_dev_index.json

Visit https://github.com/espressif/arduino-esp32 for other ways to install support

Choosing our hardware

Wemos Lolin32

This is my favourite ESP32 development board as its from the same company that makes the very popular ESP8266 based Wemos Mini boards and shields



The D32 comes in 2 formats, the basic one has the following features

Espressif official ESP32-WROOM-32 module
Lastest ESP32 Version: REV1
4MB FLASH
Lithium battery interface, 500mA Max charging current
Compatible with Arduino, MicroPython
Default firmware: lastest MicroPython

The pro one has the following features

Espressif official ESP32-WROVER module
Lastest ESP32 Version: REV1
4MB FLASH
4MB PSRAM
Lithium battery interface, 500mA Max charging current
LOLIN I2C port

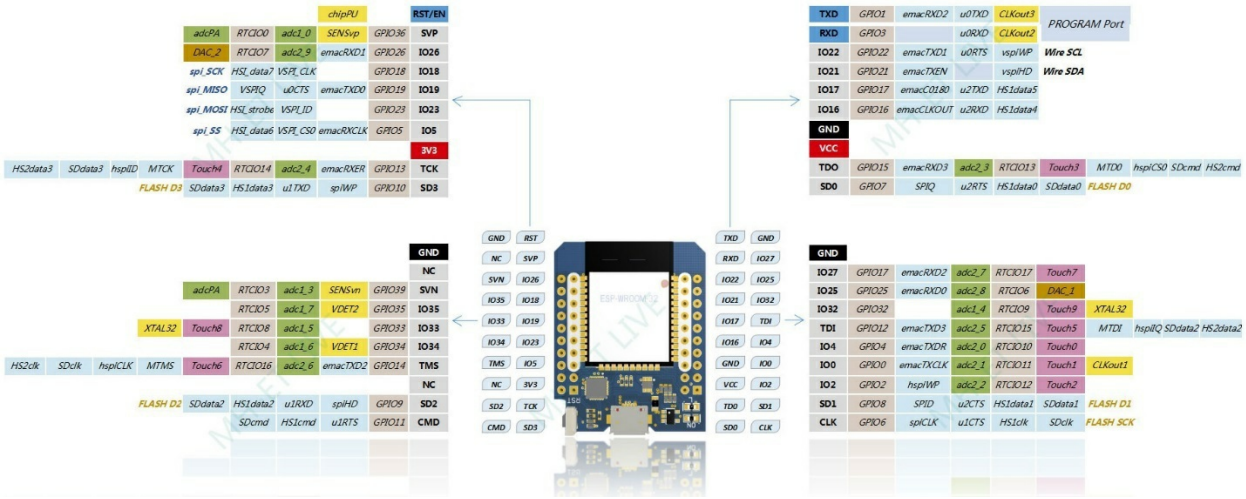
LOLIN TFT port

MH-ET LIVE MiniKit for ESP32

The MH-ET LIVE MiniKit is an ESP32 board which has the advantage of being able to utilise the various shields that have been developed for the Wemos Mini (ESP8266) module.

This is a picture of the module

Here is the pinout of the board, if you look at this you can see that the white rows are the same as the Wemos Mini, this means if you fit a suitable header you can use those shields



I have tried some Wemos mini shields and these work fine with this board

You can also see that there are many additional i/o pins you can use, hopefully people will develop shields for this board as well. The module comes with various header options so you can decide which ones you want to solder on

Links

There are various github resources for this board

https://github.com/MHEtLive/ESP32-MINI-KIT - various libraries

https://github.com/MHEtLive/ESP8266-Arduino-examples-lab

Basic examples and ESP32 features

Basic analog test example for an ESP32 board

This is a very basic example for the ESP32 board, the only reason for this is to show that unlike the ESP8266 boards the ESP32 has more than one Analog pins - in fact it has 12 analog pins

Once you have added ESP32 support to the Arduino IDE then select the Wemos Lolin 32 board and the correct port.

Code

```
int analog0;
int analog1;
int analog2;
int analogVal0 = 0;
int analogVal1 = 0;
int analogVal2 = 0;

void setup()
{
Serial.begin(9600); // use the serial port to send the values back to the computer
}

void loop()
{
analogVal0 = analogRead(analog0); // read the value from the sensor
analogVal1 = analogRead(analog1);
analogVal2 = analogRead(analog2);

Serial.println("analogVal0"); // print the value to the serial port
Serial.println(analogVal0);
Serial.println("analogVal1"); // print the value to the serial port
Serial.println(analogVal1);
Serial.println("analogVal2"); // print the value to the serial port
Serial.println(analogVal2);
}
```

Output

Open the serial monitor and you should see something like this

```
analogVal0
404
analogVal1
384
analogVal2
364
analogVal0
416
analogVal1
393
analogVal2
368
```

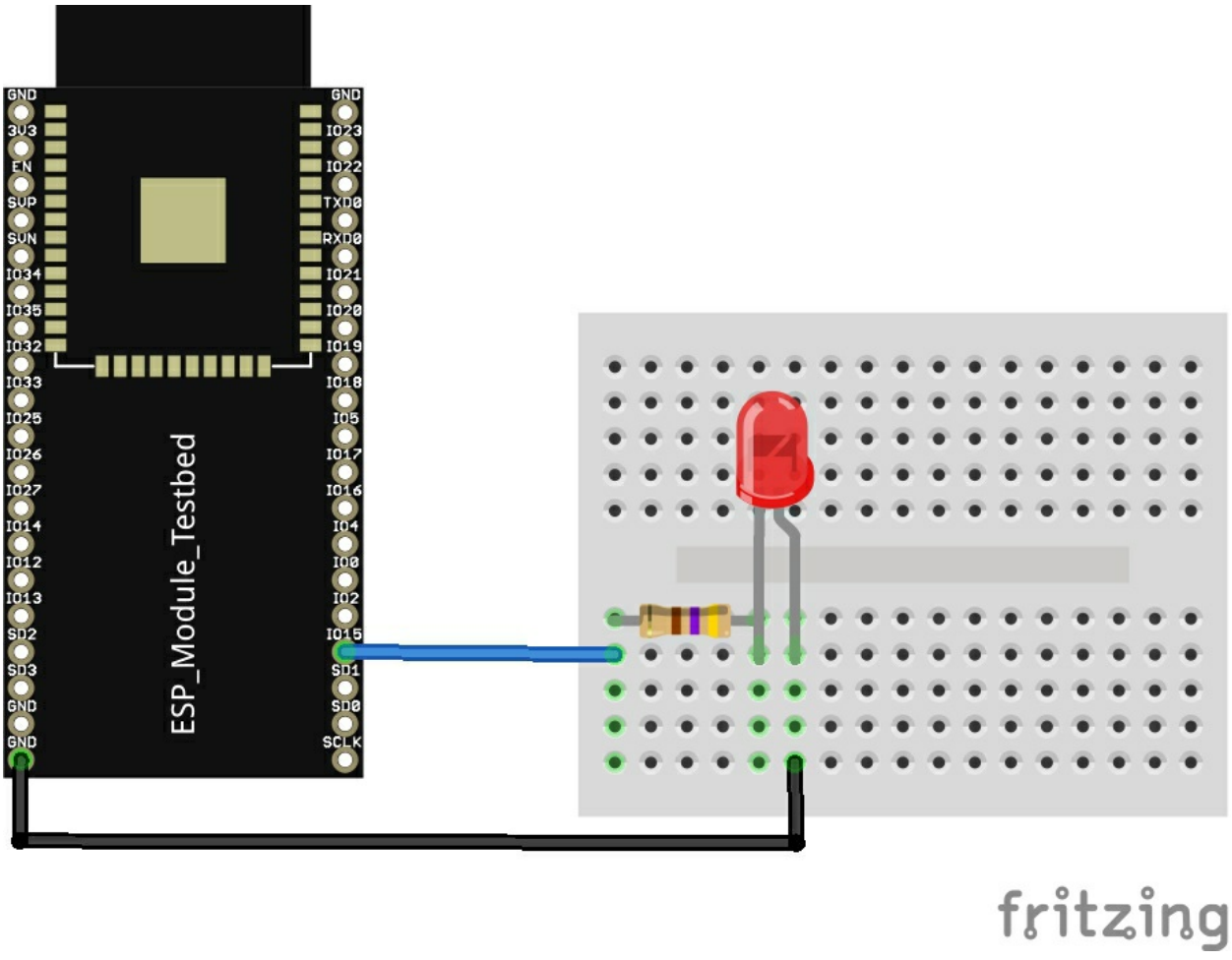
Basic WebServer example

In this example we will create a basic web server with an ESP32, we will then serve a web page with an on and off button which will switch an LED on and off

Parts Required

1x ESP32 Dev Module (Lolin32)
1x LED
1x Breadboard
1x 470 Ohm Resistor
Jumper wires

Layout



Code

Change the username and password to your own login information

A large code example which we have on our github repo – may be easier to read there

```
#include <WiFi.h>

// Replace with your network credentials
const char* ssid     = "username";
const char* password = "password";

WiFiServer server(80);

const int led =  15;      // the number of the LED pin

// Client variables
char linebuf[80];
int charcount=0;

void setup()
{
  // initialize the LED as an output:
  pinMode(led, OUTPUT);
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  while(!Serial) {
  }


   // We start by connecting to a WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);


   WiFi.begin(ssid, password);


   // attempt to connect to Wifi network:
  while(WiFi.status() != WL_CONNECTED)
  {
   // Connect to WPA/WPA2 network.
   delay(500);
   Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
```

```
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop()
{
  // listen for incoming clients
  WiFiClient client = server.available();
  if (client)
  {
    Serial.println("New client");
    memset(linebuf,0,sizeof(linebuf));
    charcount=0;
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected())
    {
      if (client.available())
      {
        char c = client.read();
        Serial.write(c);
        //read char by char HTTP request
        linebuf[charcount]=c;
        if (charcount<sizeof(linebuf)-1) charcount++;

              if (c == '\n' && currentLineIsBlank)
        {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");  // the connection will be closed after completion of the
response
          client.println();
          client.println("<!DOCTYPE HTML><html><head>");
          client.println("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">
</head>");
          client.println("<h1>ESP32 - Web Server example</h1>");
          client.println("<p>LED <a href=\"on\"><button>ON</button></a> <a href=\"off\">
<button>OFF</button></a></p>");
          client.println("</html>");
          break;
        }
        if (c == '\n')
        {
          // you're starting a new line
          currentLineIsBlank = true;
          if (strstr(linebuf,"GET /on") > 0)
          {
            Serial.println("LED ON");
            digitalWrite(led, HIGH);
          }
```

```
        else if (strstr(linebuf,"GET /off") > 0)
        {
          Serial.println("LED OFF");
          digitalWrite(led, LOW);
        }

              // you're starting a new line
        currentLineIsBlank = true;
        memset(linebuf,0,sizeof(linebuf));
        charcount=0;
       }
       else if (c != '\r')
       {
        // you've gotten a character on the current line
        currentLineIsBlank = false;
       }
      }
    }
    // give the web browser time to receive the data
    delay(1);

      // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}
```

Testing

Open the serial monitor to get the assigned IP address, in your favourite web browser navigate to this IP address This is my example

Now press the on and off button and check if the LED flashes

fade an LED using an ESP32

This example shows how easy it is to fade an LED using an ESP32

The code is fairly straightforward ledcSetup(ledChannel, freq, resolution) - this sets a channel of which there are 16 available, a frequency and a resolution which can be between 1 and 16 to the ledcsetup function. The frequency is a mystery at the moment but i wouldn't set it too high.

We now need to attach this to a pin where the led will be connected via ledcAttachPin(A13, ledChannel) - I chose A13 or 15 as its numbered on my LOLIN32

If you open the pins_arduino.h you will see where I get the A13 from

```
static const uint8_t A0 = 36;
static const uint8_t A3 = 39;
static const uint8_t A4 = 32;
static const uint8_t A5 = 33;
static const uint8_t A6 = 34;
static const uint8_t A7 = 35;
static const uint8_t A10 = 4;
static const uint8_t A11 = 0;
static const uint8_t A12 = 2;
static const uint8_t A13 = 15;
static const uint8_t A14 = 13;
static const uint8_t A15 = 12;
static const uint8_t A16 = 14;
static const uint8_t A17 = 27;
static const uint8_t A18 = 25;
static const uint8_t A19 = 26;
```

The two for loops fade the led in and out

Code

```
int freq = 10000;
int ledChannel = 0;
int resolution = 8;

void setup() {

ledcSetup(ledChannel, freq, resolution);
ledcAttachPin(A13, ledChannel);

}

void loop()
{
for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++)
{
ledcWrite(ledChannel, dutyCycle);
delay(5);
}

for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--)
{
ledcWrite(ledChannel, dutyCycle);
delay(5);
}

}
```

ESP32 capacitive touch example

Another great feature of the ESP32 is that it has the ability to detect touch on various pins by having capacitive touch sensors, in fact the ESP32 has 10 of these

If you look at the pin mapping you will see the following - these are the pins that support touch

```
static const uint8_t T0 = 4;
static const uint8_t T1 = 0;
static const uint8_t T2 = 2;
static const uint8_t T3 = 15;
static const uint8_t T4 = 13;
static const uint8_t T5 = 12;
static const uint8_t T6 = 14;
static const uint8_t T7 = 27;
static const uint8_t T8 = 33;
static const uint8_t T9 = 32;
```

Its easy to read the touch sensors by using the function: touchRead(Touch Pin #);

We will create an example where when we touch a pin an led will light, you may need to adjust the value called touch_value

These could be used for touch buttons where you could connect an external pads to the pins

Code

```
#define TOUCH_PIN T0 //connected to 4
#define LED_PIN A13 //connected to 15
int touch_value = 100;

void setup()
{
Serial.begin(9600);
Serial.println("ESP32 Touch Test");
pinMode(LED_PIN, OUTPUT);
digitalWrite (LED_PIN, LOW);
}

void loop()
{
touch_value = touchRead(TOUCH_PIN);
Serial.println(touch_value); // get value using T0
if (touch_value < 50)
{
digitalWrite (LED_PIN, HIGH);
}
else
{
digitalWrite (LED_PIN, LOW);
}
delay(1000);
}
```

ESP32 : perform a software reset

In this example we will show you how to perform a software reset on the ESP32 using the Arduino IDE. Luckily the ESP32 has a method in its library that makes this easy to do. You simply need to call the restart() method

Code

```
void setup()
{
Serial.begin(115200);
Serial.println("Restarting in 10 seconds");
delay(10000);
ESP.restart();
}

void loop()
{
}
```

Output

Open the serial monitor

```
E (102535) wifi: esp_wifi_stop 802 wifi is not init
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0008,len:8
load:0x3fff0010,len:160
load:0x40078000,len:10632
load:0x40080000,len:252
entry 0x40080034
Restarting in 10 seconds
E (102535) wifi: esp_wifi_stop 802 wifi is not init
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0008,len:8
load:0x3fff0010,len:160
load:0x40078000,len:10632
load:0x40080000,len:252
entry 0x40080034
Restarting in 10 seconds
```

Network Time Protocol example

Sometimes it is useful in any logging or displaying data to have a reasonably accurate time, by adding the NTPClient library you can do this quite easily Add support for the NTPClient library by doing the following

- Go To Library Manager and search for “NTP”
- Locate and Install NTPClient Library from Fabrice Weinberg

In the example below I am GMT, so no time zone offset

Code

You need to supply your own login credentials for username and password

```
#include <WiFi.h>
const char* ssid = "username";
const char* password = "password";

/* Time Stamp */
#include <NTPClient.h>
#include <WiFiUdp.h>

#define NTP_OFFSET 0 * 60 * 60 // In seconds
#define NTP_INTERVAL 60 * 1000 // In miliseconds
#define NTP_ADDRESS "0.pool.ntp.org"

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);

void setup()
{
Serial.begin(115200);
Serial.println("");
Serial.println("Time Stamp example");
Serial.println("");
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
timeClient.begin();
```

```
}

void loop()
{
timeClient.update();
String formattedTime = timeClient.getFormattedTime();
Serial.println(formattedTime);
delay(1000);
}
```

Output

Open the serial monitor and check the clock on your PC/Mac (whatever)
22:32:01
22:32:02
22:32:03
22:32:04
22:32:05
22:32:06
22:32:07
22:32:08
22:32:09
22:32:10

Links

i could go into great detail about NTP but these sites do it better
https://en.wikipedia.org/wiki/Network_Time_Protocol
http://tf.nist.gov/tf-cgi/servers.cgi

RGB LED example

In this example we will connect an RGB led to our ESP32, lets look at some information about RGB leds first

RGB LEDs consist of one red, one green, and one blue LED. By independently adjusting each of the three, RGB LEDs are capable of producing a wide color gamut. Unlike dedicated-color LEDs, however, these obviously do not produce pure wavelengths. Moreover, such modules as commercially available are often not optimized for smooth color mixing.

There are two primary ways of producing white light-emitting diodes (WLEDs), LEDs that generate high-intensity white light. One is to use individual LEDs that emit three primary colors[95]—red, green, and blue—and then mix all the colors to form white light. The other is to use a phosphor material to convert monochromatic light from a blue or UV LED to broad-spectrum white light, much in the same way a fluorescent light bulb works. It is important to note that the 'whiteness' of the light produced is essentially engineered to suit the human eye, and depending on the situation it may not always be appropriate to think of it as white light.

There are three main methods of mixing colors to produce white light from an LED:

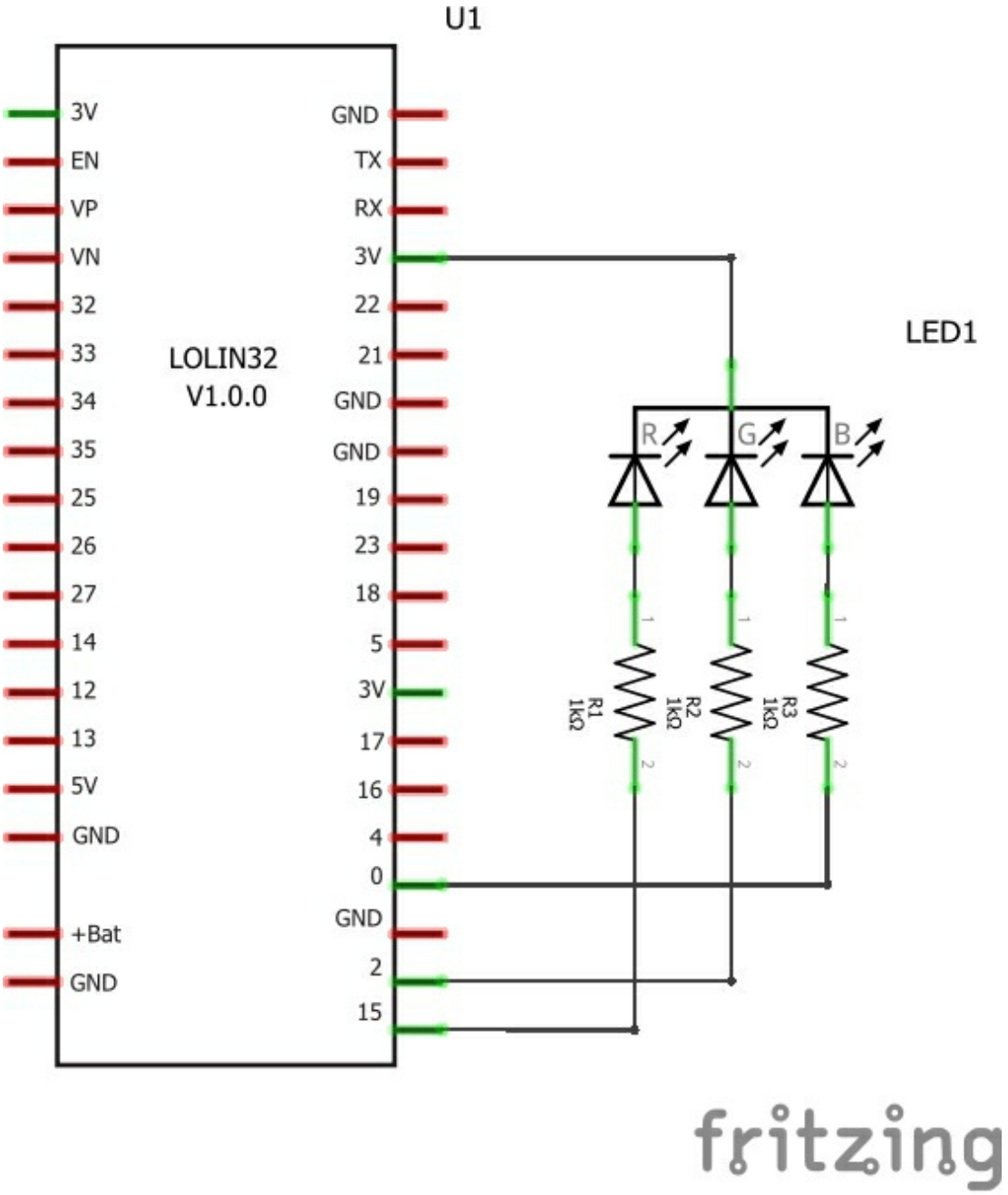
blue LED + green LED + red LED (color mixing; can be used as backlighting for displays)
near-UV or UV LED + RGB phosphor (an LED producing light with a wavelength shorter than blue's is used to excite an RGB phosphor)
blue LED + yellow phosphor (two complementary colors combine to form white light; more efficient than first two methods and more

commonly used)
Because of metamerism, it is possible to have quite different spectra that appear white. However, the appearance of objects illuminated by that light may vary as the spectrum varies.

Here is a picture of the RGB LED module I used, this is a common anode type.

Schematic

Here is a rough schematic, the LED and resistors are basically the module above



Code

This code example will cycle through the 3 main LED colours

```
int red = D6;
int green = D7;
int blue = D8;

// the setup routine runs once when you press reset:
void setup()
{
// initialize the digital pin as an output.
pinMode(red, OUTPUT);
pinMode(green, OUTPUT);
pinMode(blue, OUTPUT);
digitalWrite(red, HIGH);
digitalWrite(green, HIGH);
digitalWrite(blue, HIGH);
}

// the loop routine runs over and over again forever:
void loop() {
digitalWrite(red, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(red, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
digitalWrite(green, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(green, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
digitalWrite(blue, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(blue, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Light dependent resistor example

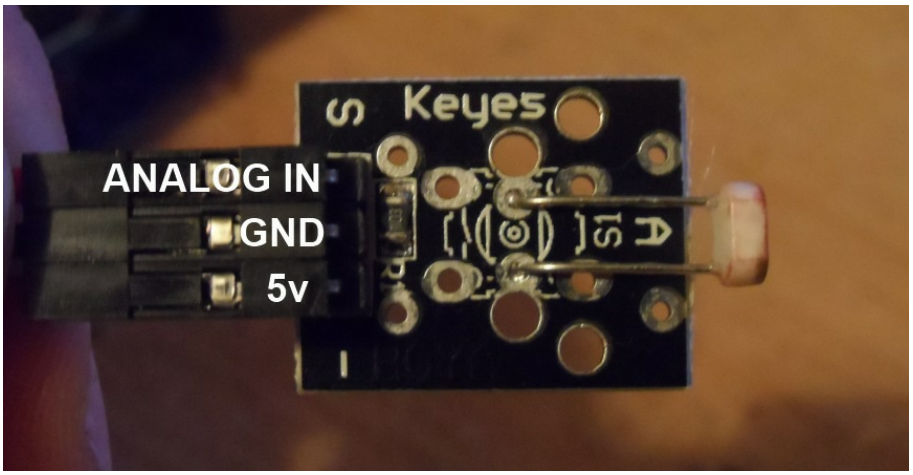
In this example we connect a photoresistor to a ESP32 Lolin32, the value read from the photoresistor corresponds to the amount of light present. The photoresistor is connected to A0 in this example.

A **photoresistor** (or **light-dependent resistor**, **LDR**, or **photocell**) is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibitsphotoconductivity. A photoresistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits. A photoresistor is made of a high resistance semiconductor.

In the dark, a photoresistor can have a resistance as high as several megohms (MΩ), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

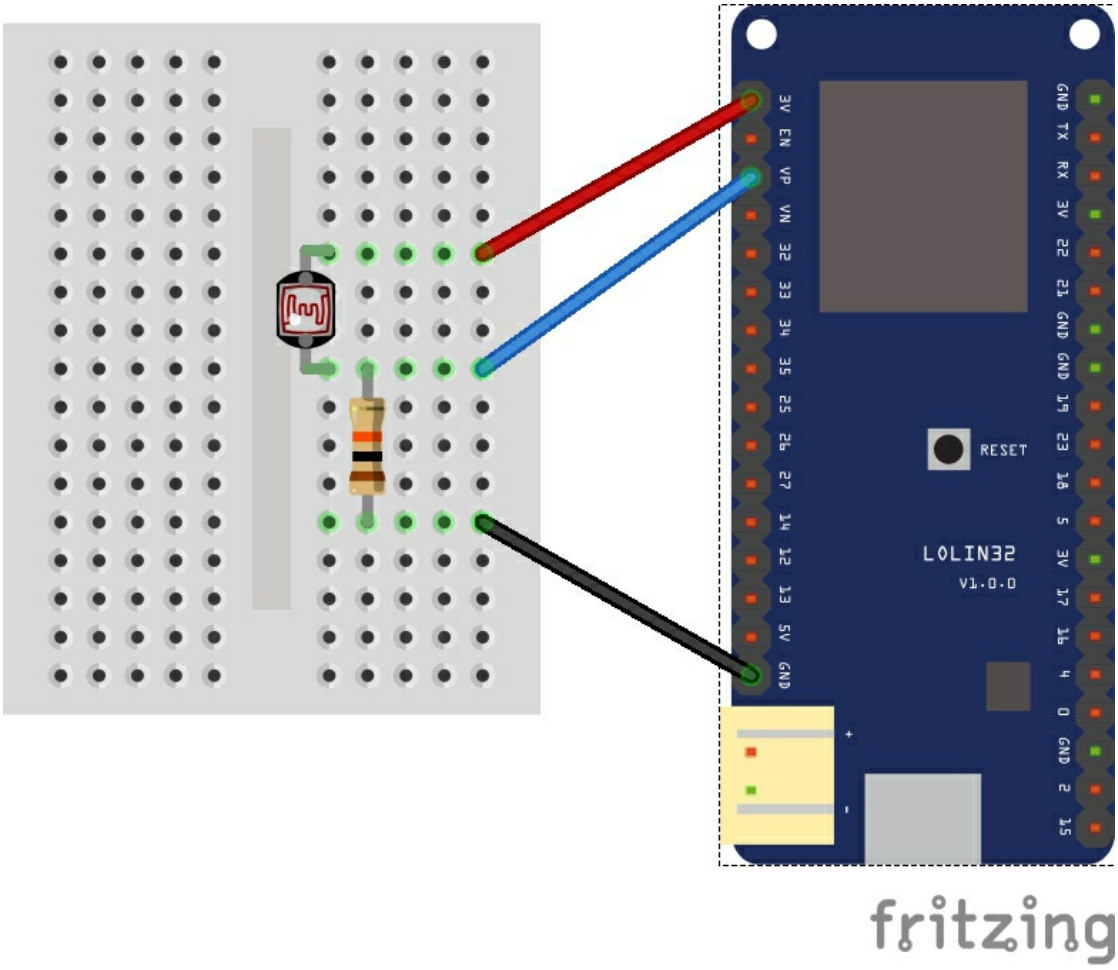
A practical example could be a dark room sensor for photography, if the reading approached a critical level an alarm could be activated or even a night light

Here is a sample module


ANALOG IN
GND
5v
Keyes

Schematic

In this example I connected 3v3 to the module and it worked fine



Code

In this example we simply output the reading via the serial port.

```
int sensorValue;

void setup()
{
  Serial.begin(9600); // starts the serial port at 9600
}

void loop()
{
  sensorValue = analogRead(A0); // read analog input pin 0
  Serial.print(sensorValue, DEC); // prints the value read
  Serial.print(" \n"); // prints a space between the numbers
  delay(1000); // wait 100ms for next reading
}
```

Testing

Open the serial monitor and move the LDR closer to a light, cover the LDR.

464
1535
4054
3999
1471
425
326

Using SHA-256 with an ESP32

In this example we will look at how you can generate the hash of a string using the SHA-256 algorithm.

We will use the Arduino IDE in which the ESP32 core by luck has a builtin mbed TLS libraries. If you want to ready more about SHA-2 then start at https://en.wikipedia.org/wiki/SHA-2 and read through this, its a nice introduction

Code

```
#include "mbedtls/md.h"

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

char *payload = "Hello SHA 256 from ESP32learning";
byte shaResult[32];

mbedtls_md_context_t ctx;
mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;

const size_t payloadLength = strlen(payload);

mbedtls_md_init(&ctx);
mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 0);
mbedtls_md_starts(&ctx);
mbedtls_md_update(&ctx, (const unsigned char *) payload, payloadLength);
mbedtls_md_finish(&ctx, shaResult);
mbedtls_md_free(&ctx);

Serial.print("Hash: ");

for(int i= 0; i< sizeof(shaResult); i++)
{
char str[3];
sprintf(str, "%02x", (int)shaResult[i]);
Serial.print(str);
}
}

void loop()
{
}
```

Testing

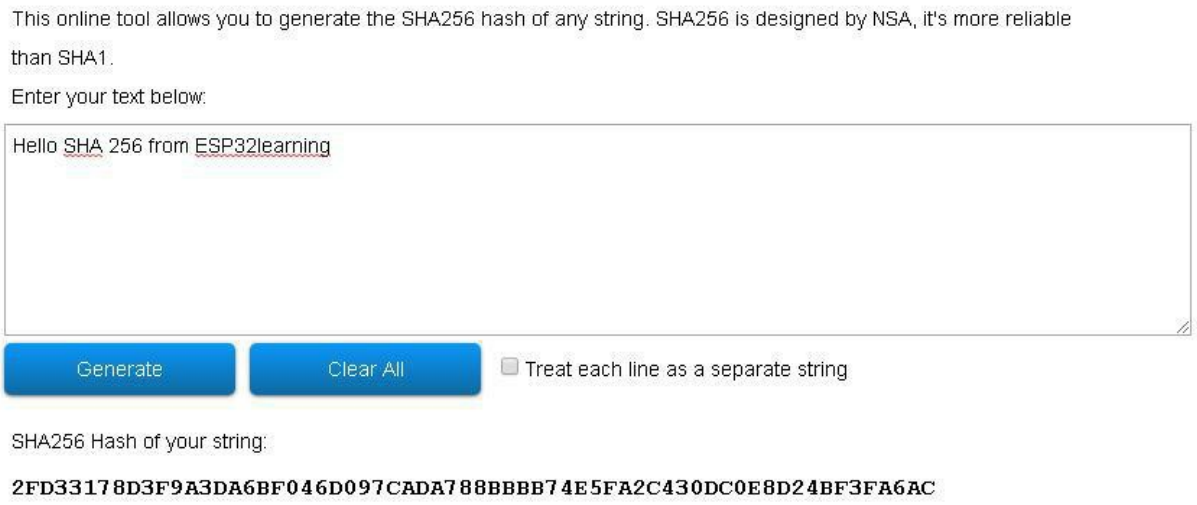
Open the serial monitor window , press the reset button on your board and you should see something like this

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x0
mode:DIO, clock div:1
load:0x3fff0008,len:8
load:0x3fff0010,len:160
load:0x40078000,len:10632
load:0x40080000,len:252
entry 0x40080034
Hash:
2fd33178d3f9a3da6bf046d097cada788bbbb74e5fa2c430dc0e8d24bf3fa6ac

The key thing is the hash, you then need to locate a sha256 hash generator tool on the internet and double check the hash above is correct. i will use the one at https://passwordsgenerator.net/sha256-hash-generator/ mainly because I could screen capture the text and the output

You can see my results here



ESP32 built in hall effect sensor example

Description

One feature of the ESP32 that sometimes goes unnoticed is the built in hall effect sensor. Lets look at a hall effect sensor and how it works - from wikipedia

A Hall effect sensor is a device that is used to measure the magnitude of a magnetic field. Its output voltage is directly proportional to the magnetic field strength through it.

Hall effect sensors are used for proximity sensing, positioning, speed detection, and current sensing applications.

Frequently, a Hall sensor is combined with threshold detection so that it acts as and is called a switch. Commonly seen in industrial applications such as the pictured pneumatic cylinder, they are also used in consumer equipment; for example some computer printers use them to detect missing paper and open covers. They can also be used in computer keyboards, an application that requires ultra-high reliability.

Hall sensors are commonly used to time the speed of wheels and shafts, such as for internal combustion engine ignition timing, tachometers and anti-lock braking systems. They are used in brushless DC electric motors to detect the position of the permanent magnet. In the pictured wheel with two equally spaced magnets, the voltage from the sensor will peak twice for each revolution. This arrangement is commonly used to regulate the speed of disk drives.

In a Hall effect sensor, a thin strip of metal has a current applied along it. In the presence of a magnetic field, the electrons in the metal strip are deflected toward one edge, producing a voltage gradient across the short side of the strip (perpendicular to the feed current). Hall effect sensors have an advantage over inductive sensors in that, while inductive sensors respond to a changing magnetic field which induces current in a coil of wire and produces voltage at its output, Hall effect sensors can detect static (non-changing) magnetic fields.

In its simplest form, the sensor operates as an analog transducer, directly returning a voltage. With a known magnetic field, its distance from the Hall

plate can be determined. Using groups of sensors, the relative position of the magnet can be deduced.

When a beam of charged particles passes through a magnetic field, forces act on the particles and the beam is deflected from a straight path. The flow of electrons through a conductor form a beam of charged carriers. When an conductor is placed in a magnetic field perpendicular to the direction of the electrons, they will be deflected from a straight path. As a consequence, one plane of the conductor will become negatively charged and the opposite side will become positively charged. The voltage between these planes is called the Hall voltage.[2]

When the force on the charged particles from the electric field balances the force produced by magnetic field, the separation of them will stop. If the current is not changing, then the Hall voltage is a measure of the magnetic flux density. Basically, there are two kinds of Hall effect sensors. One is linear which means the output of voltage linearly depends on magnetic flux density; the other is called threshold which means there will be a sharp decrease of output voltage at each magnetic flux density.

Code

```
int val = 0;
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  val = hallRead();
  // print the results to the serial monitor:
  Serial.print("sensor = ");
  Serial.println(val);//to graph
  delay(500);
}
```

Output

Open the serial monitor and if you have a magnet then put it close to your ESP32

sensor = 20
sensor = 14
sensor = 17
sensor = 19
sensor = 17
sensor = 14
sensor = 18
sensor = 68
sensor = 78

If you reverse the magnet so that the other polarity is close to the ESP32 the readings will be negative

ESP32 True random number generator

Description

ESP32 contains a hardware random number generator, values from it can be obtained using esp_random().

When Wi-Fi or Bluetooth are enabled, numbers returned by hardware random number generator (RNG) can be considered true random numbers. Without Wi-Fi or Bluetooth enabled, hardware RNG is a pseudo-random number generator

esp_random() description
Get one random 32-bit word from hardware RNG.

The hardware RNG is fully functional whenever an RF subsystem is running (ie Bluetooth or WiFi is enabled). For random values, call this function after WiFi or Bluetooth are started.

If the RF subsystem is not used by the program, the function bootloader_random_enable() can be called to enable an entropy source. bootloader_random_disable() must be called before RF subsystem or I2S peripheral are used. See these functions' documentation for more details.

Any time the app is running without an RF subsystem (or bootloader_random) enabled, RNG hardware should be considered a PRNG. A very small amount of entropy is available due to pre-seeding while the IDF bootloader is running, but this should not be relied upon for any use.

Code

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.println("-----------");
  Serial.println(esp_random());
  Serial.println(random(100));
  Serial.println(random(1,100));
  delay(1000);
}
```

Output

Open the serial monitor

```
3296550307
18
3
----------
1828082797
8
21
----------
3150364294
91
92
----------
1412556175
97
92
```

ESP32 Deep Sleep example

Description

The following is from https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/sleep_modes.html , I recommend you read these documents if you are interested in the sleep modes and the wakeup sources

ESP32 is capable of light sleep and deep sleep power saving modes.

In light sleep mode, digital peripherals, most of the RAM, and CPUs are clock-gated, and supply voltage is reduced. Upon exit from light sleep, peripherals and CPUs resume operation, their internal state is preserved.

In deep sleep mode, CPUs, most of the RAM, and all the digital peripherals which are clocked from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals (including ULP coprocessor), and RTC memories (slow and fast).

Wakeup from deep and light sleep modes can be done using several sources. These sources can be combined, in this case the chip will wake up when any one of the sources is triggered. Wakeup sources can be enabled using esp_sleep_enable_X_wakeup APIs and can be disabled using esp_sleep_disable_wakeup_source() API. Next section describes these APIs in detail. Wakeup sources can be configured at any moment before entering light or deep sleep mode.

Additionally, the application can force specific powerdown modes for the RTC peripherals and RTC memories using esp_sleep_pd_config() API.

Once wakeup sources are configured, application can enter sleep mode using esp_light_sleep_start() or esp_deep_sleep_start() APIs. At this point the hardware will be configured according to the requested wakeup sources, and RTC controller will either power down or power off the CPUs and digital peripherals.

esp_deep_sleep_start() function can be used to enter deep sleep once wakeup sources are configured. It is also possible to go into deep sleep with no wakeup sources configured, in this case the chip will be in deep sleep mode indefinitely, until external reset is applied.

esp_sleep_get_wakeup_cause() function can be used to check which wakeup source has triggered wakeup from sleep mode.

For touch pad and ext1 wakeup sources, it is possible to identify pin or touch pad which has caused wakeup using esp_sleep_get_touchpad_wakeup_status() and esp_sleep_get_ext1_wakeup_status() functions.

Code

```
/*
Simple Deep Sleep with Timer Wake Up
=====================================
ESP32 offers a deep sleep mode for effective power
saving as power is an important factor for IoT
applications. In this mode CPUs, most of the RAM,
and all the digital peripherals which are clocked
from APB_CLK are powered off. The only parts of
the chip which can still be powered on are:
RTC controller, RTC peripherals ,and RTC memories

This code displays the most basic deep sleep with
a timer to wake it up and how to store data in
RTC memory to use it over reboots

This code is under Public Domain License.

Author:
Pranav Cherukupalli <cherukupallip@gmail.com>
*/

#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  5        /* Time ESP32 will go to sleep (in seconds) */

RTC_DATA_ATTR int bootCount = 0;

/*
Method to print the reason by which ESP32
has been awaken from sleep
*/
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;

    wakeup_reason = esp_sleep_get_wakeup_cause();

    switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using
RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using
RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
  }
}

void setup(){
  Serial.begin(115200);
```

```
  delay(1000); //Take some time to open up the Serial Monitor

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  /*
  First we configure the wake up source
  We set our ESP32 to wake up every 5 seconds
  */
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
  " Seconds");

  /*
  Next we decide what all peripherals to shut down/keep on
  By default, ESP32 will automatically power down the peripherals
  not needed by the wakeup source, but if you want to be a poweruser
  this is for you. Read in detail at the API docs
  http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep.html
  Left the line commented as an example of how to configure peripherals.
  The line below turns off all RTC peripherals in deep sleep.
  */
  //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
  //Serial.println("Configured all RTC Peripherals to be powered down in sleep");

  /*
  Now that we have setup a wake cause and if needed setup the
  peripherals state in deep sleep, we can now start going to
  deep sleep.
  In the case that no wake up sources were provided but deep
  sleep was started, it will sleep forever unless hardware
  reset occurs.
  */
  Serial.println("Going to sleep now");
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called
}
```

Output
Open the serial monitor

```
rst:0x5 (DEEPSLEEP_RESET),boot:0x37 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x0
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:928
ho 0 tail 12 room 4
load:0x40078000,len:8424
ho 0 tail 12 room 4
load:0x40080400,len:5868
entry 0x4008069c
Boot number: 14
Wakeup caused by timer
Setup ESP32 to sleep for every 5 Seconds
Going to sleep now
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x37 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x0
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:928
ho 0 tail 12 room 4
load:0x40078000,len:8424
ho 0 tail 12 room 4
load:0x40080400,len:5868
entry 0x4008069c
Boot number: 15
Wakeup caused by timer
Setup ESP32 to sleep for every 5 Seconds
Going to sleep now
ets Jun  8 2016 00:22:57
```

ESP32 : a look at the Dual core

Description

The vanilla FreeRTOS is designed to run on a single core. However the ESP32 is dual core containing a Protocol CPU (known as CPU 0 or PRO_CPU) and an Application CPU (known as CPU 1 or APP_CPU). The two cores are identical in practice and share the same memory. This allows the two cores to run tasks interchangeably between them.

The ESP-IDF FreeRTOS is a modified version of vanilla FreeRTOS which supports symmetric multiprocessing (SMP). ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0. This guide outlines the major differences between vanilla FreeRTOS and ESP-IDF FreeRTOS. The API reference for vanilla FreeRTOS can be found via http://www.freertos.org/a00106.html

For information regarding features that are exclusive to ESP-IDF FreeRTOS, see ESP-IDF FreeRTOS Additions.

Tasks and Task Creation: Use xTaskCreatePinnedToCore() or xTaskCreateStaticPinnedToCore() to create tasks in ESP-IDF FreeRTOS. The last parameter of the two functions is xCoreID. This parameter specifies which core the task is pinned to. Acceptable values are 0 for PRO_CPU, 1 for APP_CPU, or tskNO_AFFINITY which allows the task to run on both.

Task Deletion: Task deletion behavior has been backported from FreeRTOS v9.0.0 and modified to be SMP compatible. Task memory will be freed immediately when vTaskDelete() is called to delete a task that is not currently running and not pinned to the other core. Otherwise, freeing of task memory will still be delegated to the Idle Task.

Tasks in ESP-IDF FreeRTOS are designed to run on a particular core, therefore two new task creation functions have been added to ESP-IDF FreeRTOS by appending PinnedToCore to the names of the task creation functions in vanilla FreeRTOS. The vanilla FreeRTOS functions of xTaskCreate() and xTaskCreateStatic() have led to the addition of xTaskCreatePinnedToCore() and xTaskCreateStaticPinnedToCore() in ESP-IDF FreeRTOS

The ESP-IDF FreeRTOS task creation functions are nearly identical to their

vanilla counterparts with the exception of the extra parameter known as xCoreID. This parameter specifies the core on which the task should run on and can be one of the following values.

0 pins the task to PRO_CPU
1 pins the task to APP_CPU
tskNO_AFFINITY allows the task to be run on both CPUs
For example xTaskCreatePinnedToCore(tsk_callback, “APP_CPU Task”, 1000, NULL, 10, NULL, 1) creates a task of priority 10 that is pinned to APP_CPU with a stack size of 1000 bytes. It should be noted that the uxStackDepth parameter in vanilla FreeRTOS specifies a task’s stack depth in terms of the number of words, whereas ESP-IDF FreeRTOS specifies the stack depth in terms of bytes.

More info : I recommend going through the following where the info above comes from https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/freertos-smp.html

Code Example 1 : Which core is running

This basic example will flash the LED as per the default example and we will use the xPortGetCoreID() to show the core that this runs on

```
void setup()
{
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop()
{
  Serial.print("This Task runs on Core: ");
  Serial.println(xPortGetCoreID());

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

You should see This Task runs on Core: 1 in the serial monitor

Code Example 2 : Adapt flashing led example to run on Core 0

In this example we move the example above to run on core 0

```
/*
* This sketch moves the blink sketch from Core 1 in loop to Core 0
*/

TaskHandle_t Task1;

void ExampleTask1( void * parameter )
{
  for (;;) {
    Serial.print("This Task runs on Core: ");
    Serial.println(xPortGetCoreID());

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

void setup()
{
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);

    xTaskCreatePinnedToCore(
    ExampleTask1,           /* Task function. */
    "Task_1",               /* name of task. */
    1000,                 /* Stack size of task */
    NULL,                   /* parameter of the task */
    1,                    /* priority of the task */
    &Task1,                 /* Task handle to keep track of created task */
    0);                   /* Core to use */
}

void loop()
{
  delay(1000);
}
```

Example 3

```
void setup()
{
  Serial.begin(112500);
  delay(1000);

    xTaskCreatePinnedToCore(
                taskOne,          /* Task function. */
                "TaskOne",        /* String with name of task. */
                10000,            /* Stack size in bytes. */
                NULL,             /* Parameter passed as input of the task */
                1,                /* Priority of the task. */
                NULL,             /* Task handle. */
                0);               /* Core to use */


    xTaskCreatePinnedToCore(
                taskTwo,          /* Task function. */
                "TaskTwo",        /* String with name of task. */
                10000,            /* Stack size in bytes. */
                NULL,             /* Parameter passed as input of the task */
                1,                /* Priority of the task. */
                NULL,             /* Task handle. */
                1);               /* Core to use */

}

void loop() {
  delay(1000);
}

void taskOne( void * parameter )
{
    for( int i = 0;i<10;i++ )
    {
        Serial.println("Hello from task 1");
        delay(1000);
    }

       Serial.println("Ending task 1");
    vTaskDelete( NULL );
}

void taskTwo( void * parameter)
{
    for( int i = 0;i<10;i++ ){
        Serial.println("Hello from task 2");
        delay(1000);
    }
    Serial.println("Ending task 2");
```

```
    vTaskDelete( NULL );
}
```

Output

Open the serial monitor

```
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Hello from task 1
Hello from task 2
Ending task 1
Ending task 2
```

ESP32 DAC example

Description

ESP32 has two 8-bit DAC (digital to analog converter) channels, connected to GPIO25 (Channel 1) and GPIO26 (Channel 2).

The DAC driver allows these channels to be set to arbitrary voltages.

The DAC channels can also be driven with DMA-style written sample data, via the I2S driver when using the “built-in DAC mode”.

Code

```
#define DAC1 25

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


}

void loop() {
  int Value = 255; //255= 3.3V 128=1.65V


    dacWrite(DAC1, Value);
  delay(1000);
}
```

Output

You will need to connect a voltmeter to Pin 25

Sensor and module examples

Temperature sensor example using a BMP180

BMP180 is the new digital barometric pressure sensor of Bosch Sensortec, with a very high performance, which enables applications in advanced mobile devices, such as smartphones, tablet PCs and sports devices. It follows the BMP085 and brings many improvements, like the smaller size and the expansion of digital interfaces. ultra-low power consumption down to 3 µA makes the BMP180 the leader in power saving for your mobile devices.

BMP180 is also distinguished by its very stable behavior (performance) with regard to the independency of the supply voltage. This bmp180 from Bosch is the best low-cost sensing solution for measuring barometric pressure and temperature. The sensor is soldered onto a PCB with a 3.3V regulator, I2C level shifter and pull-up resistors on the I2C pins. The BMP180 replaces the BMP085.

Specification

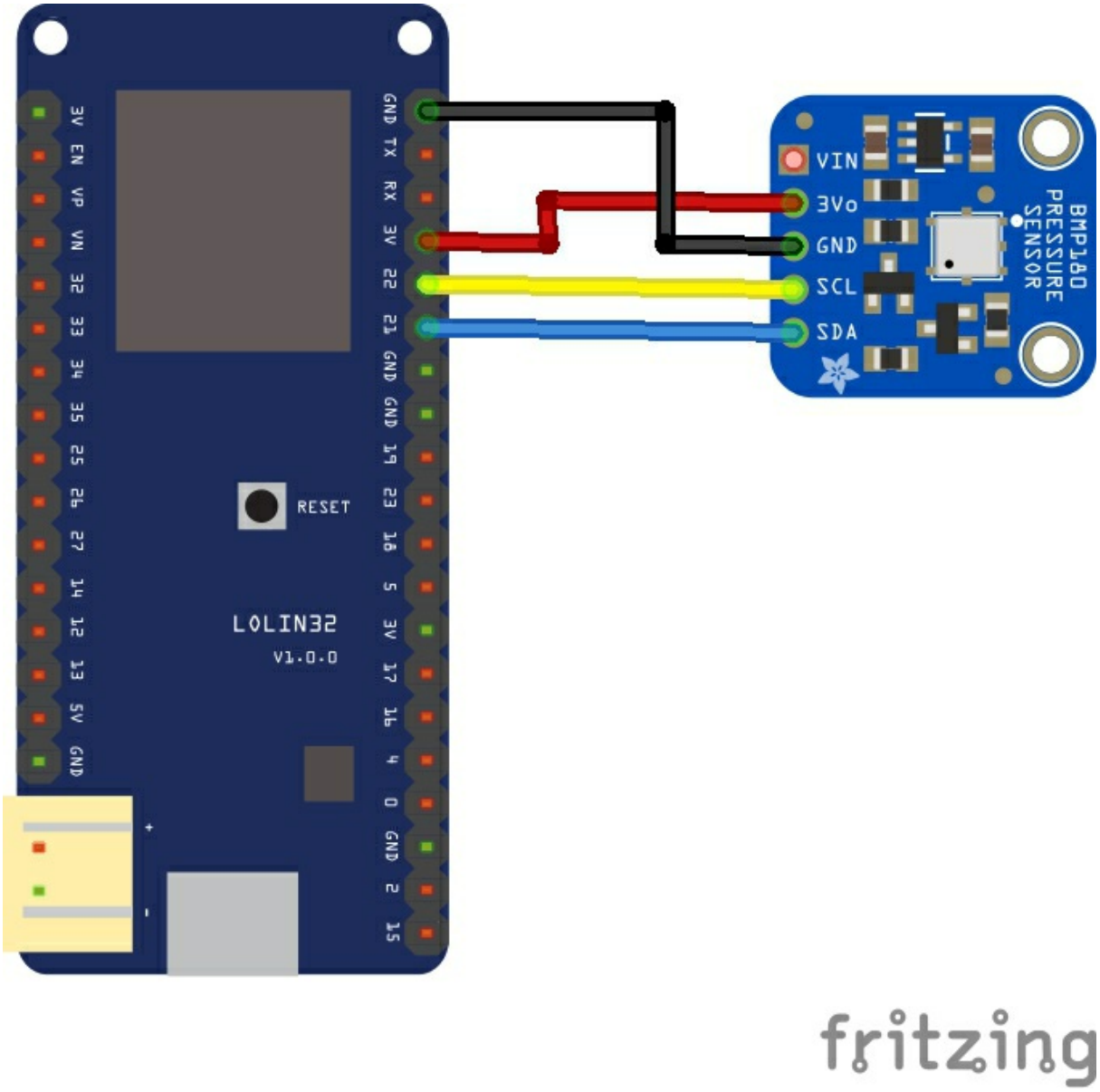
- Pressure sensing range: 300-1100 hPa (9000m to -500m above sea level)
- Up to 0.03hPa / 0.25m resolution
- -40 to +85°C operational range, +-2°C temperature accuracy

Here is a breakout which makes it easy to use the sensor. The sensor will cost less than $2.


BMP180

Layout

Make the following connections, this is a Wemos Lolin32, you may have to make small changes for other ESP32 boards.



Code

You will need the Adafruit BMP085 library for this example, you can either download it or use the library manager in newer Arduino IDEs.

https://github.com/adafruit/Adafruit-BMP085-Library

In this example I am only looking at the temperature and pressure but there are other functions in the library

```
#include <Wire.h>
#include <Adafruit_BMP085.h>

Adafruit_BMP085 bmp;

void setup()
{
Serial.begin(9600);
//Wire.begin (4, 5);
if (!bmp.begin())
{
Serial.println("Could not find BMP180 or BMP085 sensor at 0x77");
while (1) {}
}
}

void loop()
{
Serial.print("Temperature = ");
Serial.print(bmp.readTemperature());
Serial.println(" Celsius");

Serial.print("Pressure = ");
Serial.print(bmp.readPressure());
Serial.println(" Pascal");

Serial.println();
delay(5000);
}
```

Output

Open the Serial monitor and you should see something like this

Temperature = 19.80 Celsius
Pressure = 100831 Pascal

Temperature = 19.80 Celsius

Pressure = 100829 Pascal

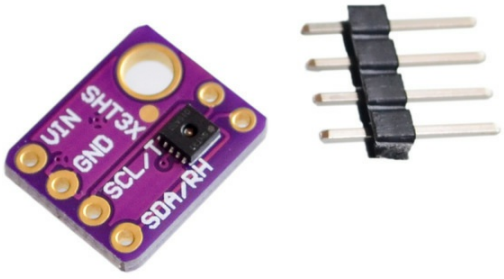
ESP32 and SHT31 sensor example

SHT31 is the next generation of Sensirion’s temperature and humidity sensors. It builds on a new CMOSens® sensor chip that is at the heart of Sensirion’s new humidity and temperatureplatform.
The SHT3x-DIS has increased intelligence, reliability and improved accuracy specifications compared to its predecessor. Its functionality includes enhanced signal processing, two distinctive and user selectable I2C addresses and communication speeds of up to 1 MHz. The DFN package has a footprint of 2.5 x 2.5 mm2 while keeping a height of 0.9 mm.

Features

Fully calibrated, linearized, and temperature compensated digital output
Wide supply voltage range, from 2.4 V to 5.5 V
I2C Interface with communication speeds up to 1 MHz and two user selectable addresses

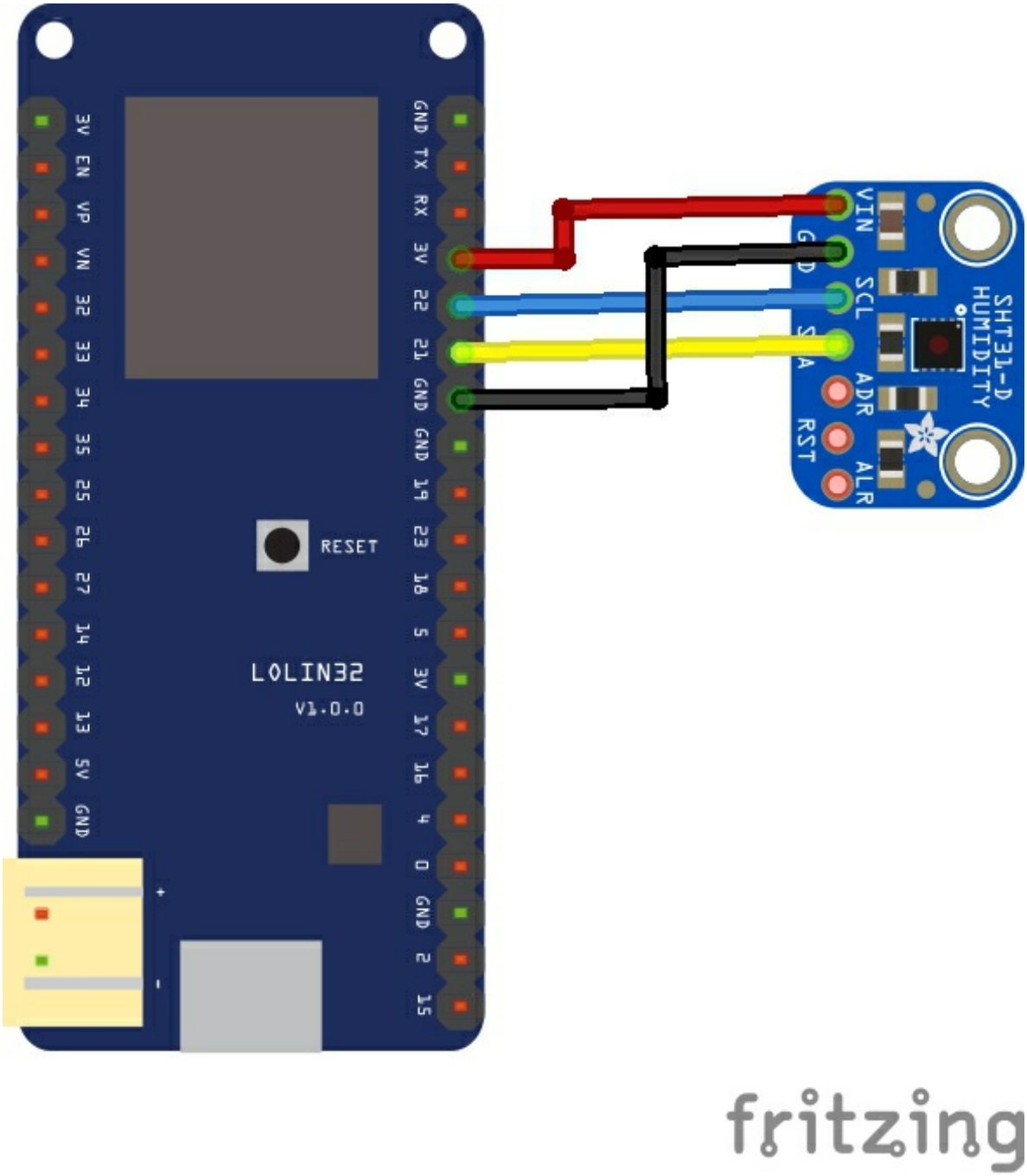
I bought the following module



Layout

If you’re using an Lolin32 simply connect the VIN pin to the 3v3 voltage pin, GND to ground, SCL to I2C Clock (22) and SDA to I2C Data (21).

Here is a layout drawn up in fritzing to illustrate this



Code

This example uses the adafruit sht31 library - https://github.com/adafruit/Adafruit_SHT31

```
#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_SHT31.h"

Adafruit_SHT31 sht31 = Adafruit_SHT31();

void setup()
{
Serial.begin(9600);
if (! sht31.begin(0x44))
{
Serial.println("Couldn't find SHT31");
while (1) delay(1);
}
}

void loop()
{
float t = sht31.readTemperature();
float h = sht31.readHumidity();

if (! isnan(t))
{
Serial.print("Temp *C = "); Serial.println(t);
}
else
{
Serial.println("Failed to read temperature");
}

if (! isnan(h))
{
Serial.print("Hum. % = "); Serial.println(h);
}
else
{
Serial.println("Failed to read humidity");
}
Serial.println();
delay(1000);
}
```

Output

Open the serial monitor and you should see something like this

Temp *C = 19.46
Hum. % = 45.33

ESP32 and HMC5883L sensor example

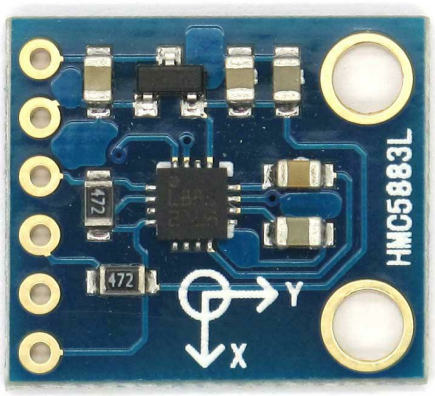
The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as lowcost compassing and magnetometry. The HMC5883L includes our state-of-theart, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy.

The I2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.

The HMC5883L utilizes Honeywell’s Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity.

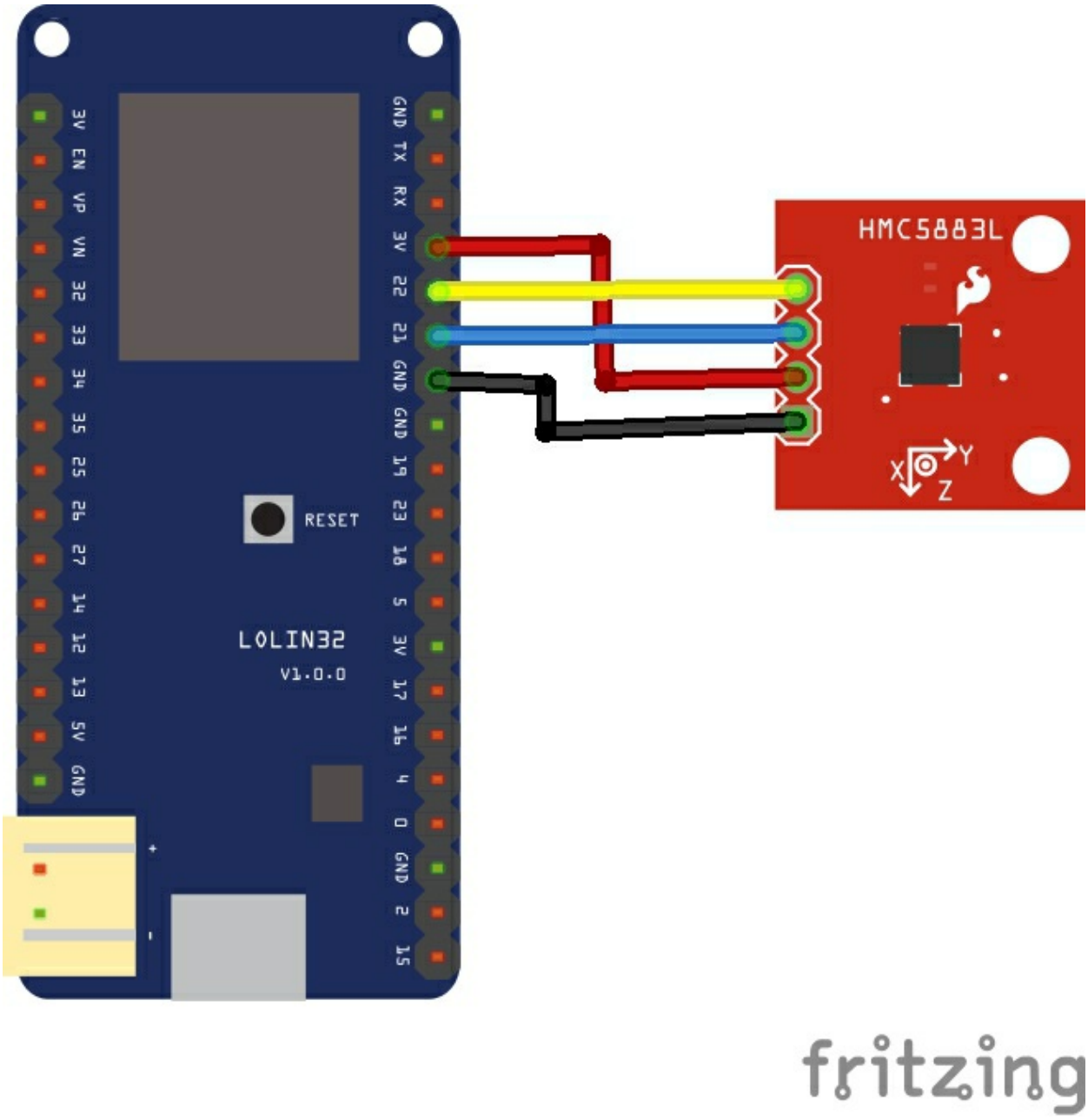
These sensors’ solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth’s magnetic fields, from milli-gauss to 8 gauss. Honeywell’s Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

Here is a typical module


HMC5883L
472
472
Y
X

Layout

This example layout shows an HMC5883 module connected to a LOLIN32



Code

This example uses the Adafruit libraries - the unified sensor and the HMC5883, you can add these libraries in the Arduino IDE using the library manager

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>

/* Assign a unique ID to this sensor at the same time */
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);

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

/* Initialise the sensor */
if(!mag.begin())
{
/* There was a problem detecting the HMC5883 ... check your connections */
Serial.println("Ooops, no HMC5883 detected ... Check your wiring!");
while(1);
}
}

void loop(void)
{
/* Get a new sensor event */
sensors_event_t event;
mag.getEvent(&event);

/* Display the results (magnetic vector values are in micro-Tesla (uT)) */
Serial.print("X: ");
Serial.print(event.magnetic.x);
Serial.print(" ");
Serial.print("Y: ");
Serial.print(event.magnetic.y);
Serial.print(" ");
Serial.print("Z: ");
Serial.print(event.magnetic.z);
Serial.print(" ");
Serial.println("uT");

// Hold the module so that Z is pointing 'up' and you can measure the heading with x&y
// Calculate heading when the magnetometer is level, then correct for signs of axis.
float heading = atan2(event.magnetic.y, event.magnetic.x);

// Once you have your heading, you must then add your 'Declination Angle', which is the 'Error' of the
```

```
magnetic field in your location.
// Find yours here: http://www.magnetic-declination.com/
// Mine is: -13* 2' W, which is ~13 Degrees, or (which we need) 0.22 radians
// If you cannot find your Declination, comment out these two lines, your compass will be slightly off.
float declinationAngle = 0.22;
heading += declinationAngle;

// Correct for when signs are reversed.
if(heading < 0)
heading += 2*PI;

// Check for wrap due to addition of declination.
if(heading > 2*PI)
heading -= 2*PI;

// Convert radians to degrees for readability.
float headingDegrees = heading * 180/M_PI;

Serial.print("Heading (degrees): ");
Serial.println(headingDegrees);

delay(500);
}
```

Testing

Open the serial monitor

```
Heading (degrees): 34.00
X: 17.45 Y: 6.91 Z: -41.53 uT
Heading (degrees): 34.20
X: 17.27 Y: 7.18 Z: -41.22 uT
Heading (degrees): 35.18
X: 16.18 Y: 8.82 Z: -41.53 uT
Heading (degrees): 41.19
X: 13.82 Y: 9.82 Z: -41.73 uT
```

ESP32 and MLX90614 infrared thermometer example

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

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

Connection

Again I connected this to a Wemos Lolin32

VIN -> Lolin32 3.3v
GND -> Lolin32 GND
SCL -> Lolin32 22
SDA -> Lolin32 21

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

```
#include <Wire.h>
#include <Adafruit_MLX90614.h>

Adafruit_MLX90614 mlx = Adafruit_MLX90614();

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

void loop()
{
Serial.print("Ambient = ");
Serial.print(mlx.readAmbientTempC());
Serial.print("*C\tObject = ");
Serial.print(mlx.readObjectTempC());
Serial.println("*C");
Serial.print("Ambient = ");
Serial.print(mlx.readAmbientTempF());
Serial.print("*F\tObject = ");
Serial.print(mlx.readObjectTempF());
Serial.println("*F");

Serial.println();
delay(1000);
}
```

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

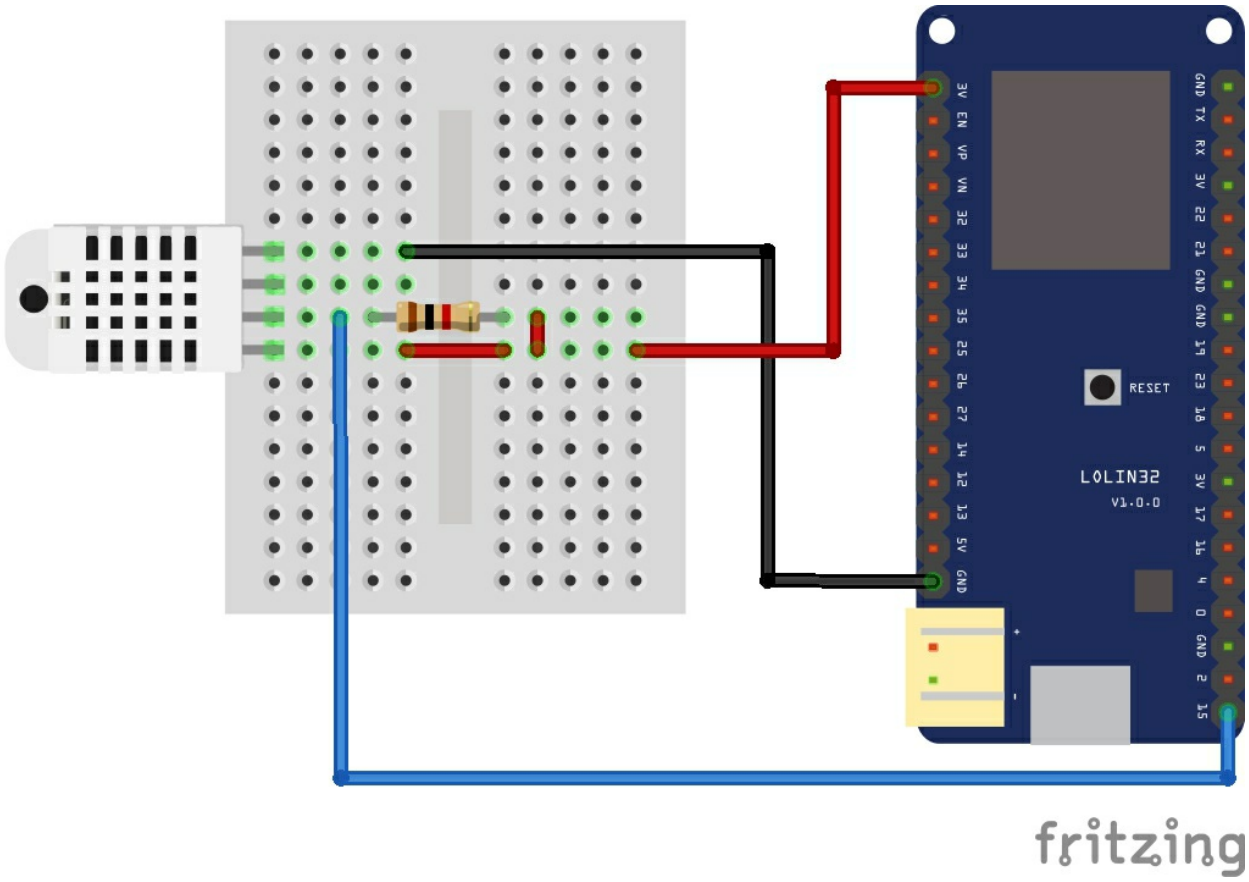
ESP32 and AM2302 example

AM2302 capacitive humidity sensing digital temperature and humidity module is one that contains the compound has been calibrated digital signal output of the temperature and humidity sensors. Application of a dedicated digital modules collection technology and the temperature and humidity sensing technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a capacitive sensor wet components and a high-precision temperature measurement devices, and connected with a high-performance 8-bit microcontroller. The product has excellent quality, fast response, strong anti-jamming capability, and high cost

Features

Ultra-low power, the transmission distance, fully automated calibration, the use of capacitive humidity sensor, completely interchangeable, standard digital single-bus output, excellent long-term stability, high accuracy temperature measurement devices.

Schematic


RESET
LOLIN32
V1.0.0
fritzing

Code

You need to add the DHT library from adafruit to the Arduino IDE - https://github.com/adafruit/DHT-sensor-library

```
#include "DHT.h"

#define DHTPIN A13
//our sensor is DHT22 type
#define DHTTYPE DHT22
//create an instance of DHT sensor
DHT dht(DHTPIN, DHTTYPE);
void setup() {
Serial.begin(115200);
Serial.println("DHT22 sensor!");
//call begin to start sensor
dht.begin();
}

void loop() {
//use the functions which are supplied by library.
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
Serial.println("Failed to read from DHT sensor!");
return;
}
// print the result to Terminal
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.println(" *C ");
//we delay a little bit for next read
delay(2000);
}
```

Output

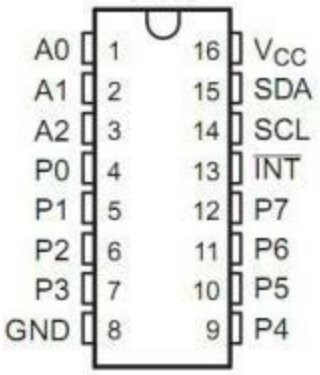
Open the serial monitor and you should see something like this

Humidity: 53.30 % Temperature: 28.90 *C
Humidity: 53.30 % Temperature: 28.90 *C

PC8574 and ESP32 example

The PCF8574 is an 8 bits I/O port expander that uses the I2C protocol. Using this IC, you can use only the SDA and SCL pins of your Arduino board to control up to 8 digital I/O ports.

A0,A1,A2 are address pins
P0,P1,P2,P3,P4,P5,P6,P7 are digital I/O ports
SDA,SCL are the I2C pins

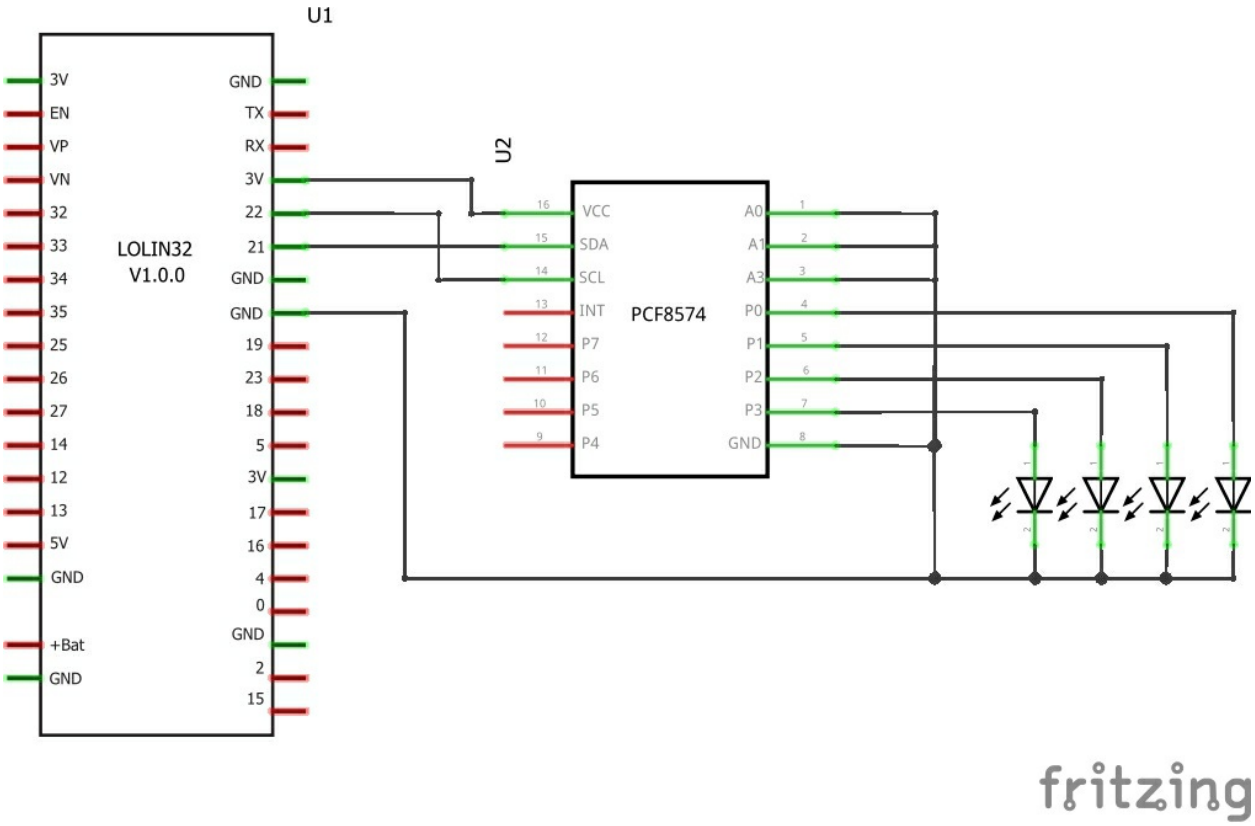


If we set pins A0 to A2 to GND, our device address in binary will be 0x20, thats exactly what I did in my example. To enable read and write there are different values required you can see these in the image below

Pin connectivity			Address of PCF8574								Address byte value		7-bit hexadecimal address without R/W
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	R/$\overline{W}$	Write	Read	
V_{SS}	V_{SS}	V_{SS}	0	1	0	0	0	0	0	-	40h	41h	20h
V_{SS}	V_{SS}	V_{DD}	0	1	0	0	0	0	1	-	42h	43h	21h
V_{SS}	V_{DD}	V_{SS}	0	1	0	0	0	1	0	-	44h	45h	22h
V_{SS}	V_{DD}	V_{DD}	0	1	0	0	0	1	1	-	46h	47h	23h
V_{DD}	V_{SS}	V_{SS}	0	1	0	0	1	0	0	-	48h	49h	24h
V_{DD}	V_{SS}	V_{DD}	0	1	0	0	1	0	1	-	4Ah	4Bh	25h
V_{DD}	V_{DD}	V_{SS}	0	1	0	0	1	1	0	-	4Ch	4Dh	26h
V_{DD}	V_{DD}	V_{DD}	0	1	0	0	1	1	1	-	4Eh	4Fh	27h

Schematic

Note that the PCF8574 is a current sink device so you do not require the current limiting resistors
I have only shown 4 LEDs, also note that I have shown A0, A1 and A2 tied to GND, this is what my test module had selected.



Code

This example flashes the led's

```
#include <Wire.h>

// address of PCF8574 IC
#define PCF8574_ADDR (0x20)

void setup()
{
  Wire.begin();
}

void loop()
{



    //send the data
  Wire.beginTransmission(PCF8574_ADDR);
  Wire.write(0xAA);
  Wire.endTransmission();
  delay(1000);
  Wire.beginTransmission(PCF8574_ADDR);
  Wire.write(0x55);
  Wire.endTransmission();
  delay(1000);
}
```

MAX6675 example

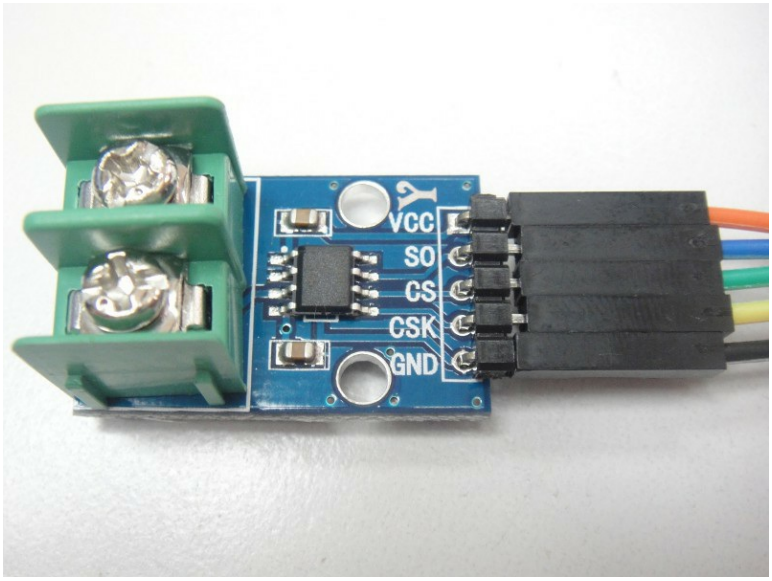
In this example we take a look at the MAX6675 Cold-Junction-Compensated K-Thermocouple-to-Digital Converter

The MAX6675 performs cold-junction compensation and digitizes the signal from a type-K thermocouple. The data is output in a 12-bit resolution, SPI™-compatible, read-only format. This converter resolves temperatures to 0.25°C, allows readings as high as +1024°C, and exhibits thermocouple accuracy of 8 LSBs for temperatures ranging from 0°C to +700°C.

Key Features

Cold-Junction Compensation
Simple SPI-Compatible Serial Interface
12-Bit, 0.25°C Resolution
Open Thermocouple Detection
Usually you use these in a breakout/module form and can also purchase a kit which includes a thermocouple.

Here is a picture of the module that I purchased



Connection

Here are the connections, sometimes these are named differently on the modules, in particular the CS connection

Vcc connected to 3.3v
Gnd connected to Gnd
SO connected to Pin 19 (MISO)
SS/CS connected to Pin 23 (MOSI)
CSK connected to Pin 5 (SS)

Code

You will need the MAX6675 library - https://github.com/adafruit/MAX6675-library - I needed to edit the library for it to compile, this version works - MAX6675_library

```
#include "max6675.h"

int thermoDO = 19;
int thermoCS = 23;
int thermoCLK = 5;

MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);

void setup()
{
Serial.begin(9600);
Serial.println("MAX6675 test");
delay(500);
}

void loop()
{
// basic readout test, just print the current temp

Serial.print("C = ");
Serial.println(thermocouple.readCelsius());
Serial.print("F = ");
Serial.println(thermocouple.readFahrenheit());

delay(1000);
}
```

Results

Open the serial monitor window and you should see something like this

C = 26.50
F = 79.70
C = 26.75
F = 80.15

ESP32 and RFID-RC522 module example

In this example we will connect an RFID-RC522 module and connect to an ESP32 Wemos LOLIN32

The microcontroller and card reader uses SPI for communication . The card reader and the tags communicate using a 13.56MHz electromagnetic field. (ISO 14443A standart tags) . Here is atypical module and smart card you can purchase

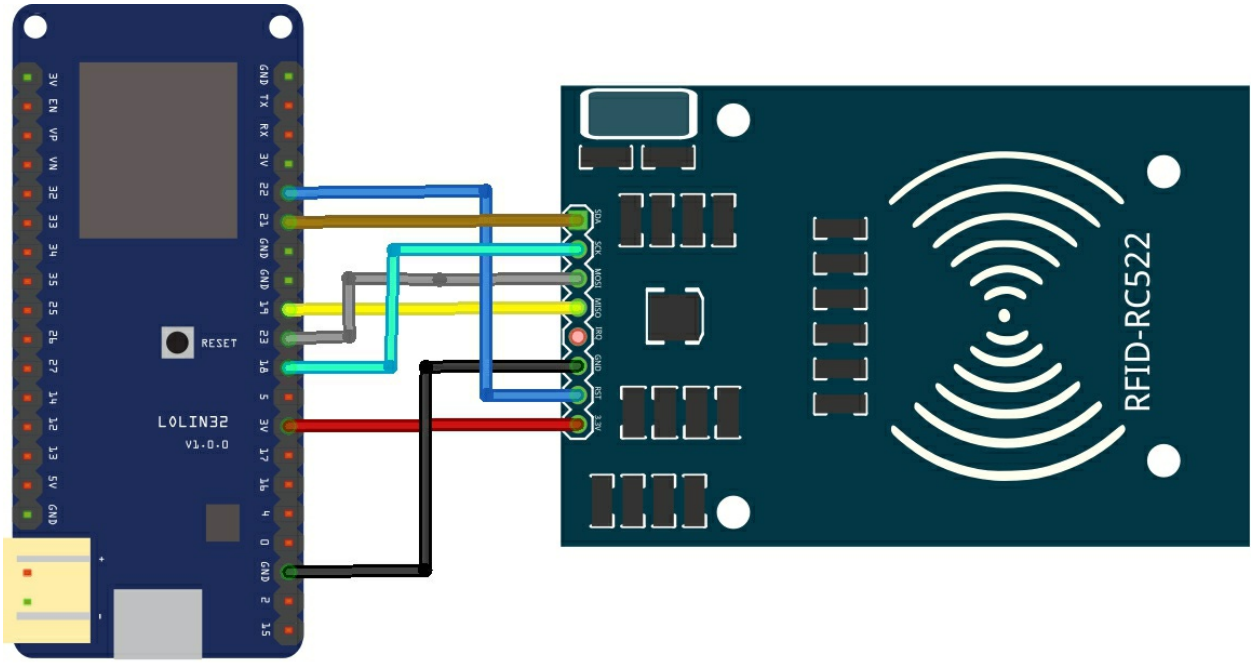
Features:

- MFRC522 chip based board
- Operating frequency: 13.56MHz
- Supply Voltage: 3.3V
- Current: 13-26mA
- Read Range: Approx 3cm with supplied card and fob
- SPI Interface
- Max Data Transfer Rate: 10Mbit / s
- Dimensions: 60mm × 39mm

Datasheet for the chip that used in modules can be found at:

http://www.nxp.com/documents/data_sheet/MFRC522.pdf

Layout



Code

Install the RFID522 library - https://github.com/miguelbalboa/rfid

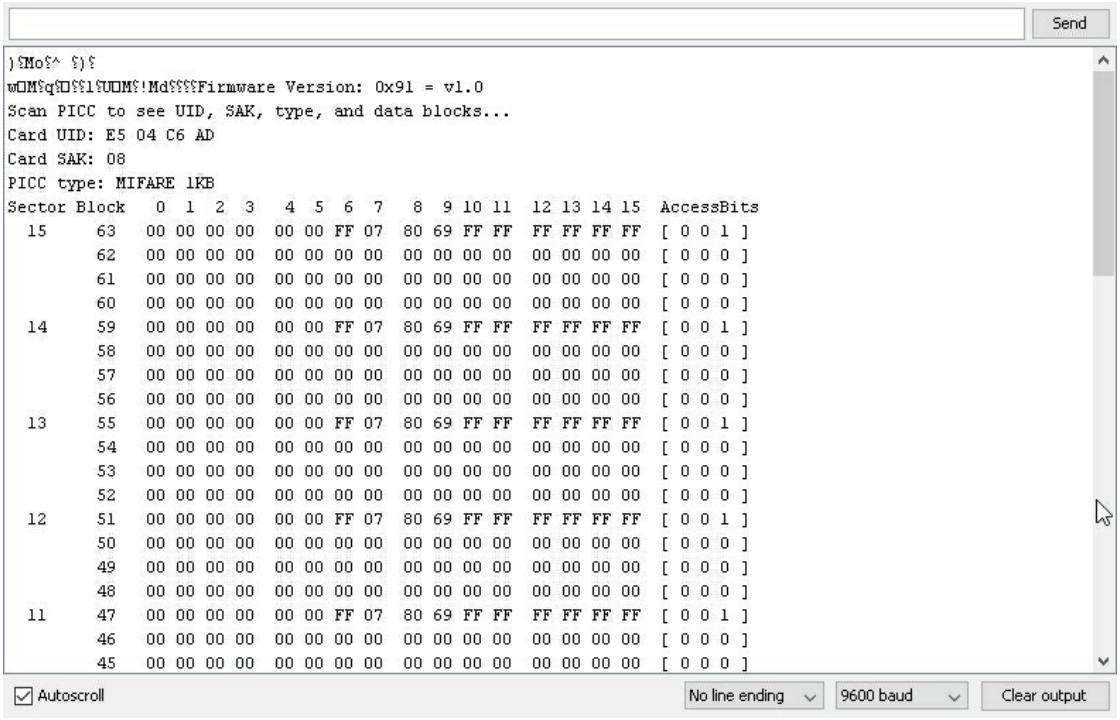
This is the DumpInfo example modified

```
#include <SPI.h>
#include <MFRC522.h>

//#define RST_PIN 9 // Configurable, see typical pin layout above
//#define SS_PIN 10 // Configurable, see typical pin layout above
const int RST_PIN = 22; // Reset pin
const int SS_PIN = 21; // Slave select pin

MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance

void setup() {
Serial.begin(9600); // Initialize serial communications with the PC
while (!Serial); // Do nothing if no serial port is opened (added for Arduinos based on ATMEGA32U4)
SPI.begin(); // Init SPI bus
mfrc522.PCD_Init(); // Init MFRC522
mfrc522.PCD_DumpVersionToSerial(); // Show details of PCD - MFRC522 Card Reader details
Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}
```

```
void loop() {
// Look for new cards
if ( ! mfrc522.PICC_IsNewCardPresent()) {
return;
}

// Select one of the cards
if ( ! mfrc522.PICC_ReadCardSerial()) {
return;
}

// Dump debug info about the card; PICC_HaltA() is automatically called
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Output

This is the output in the serial monitor



There are many other good examples in the library

LM35 and ESP32 example

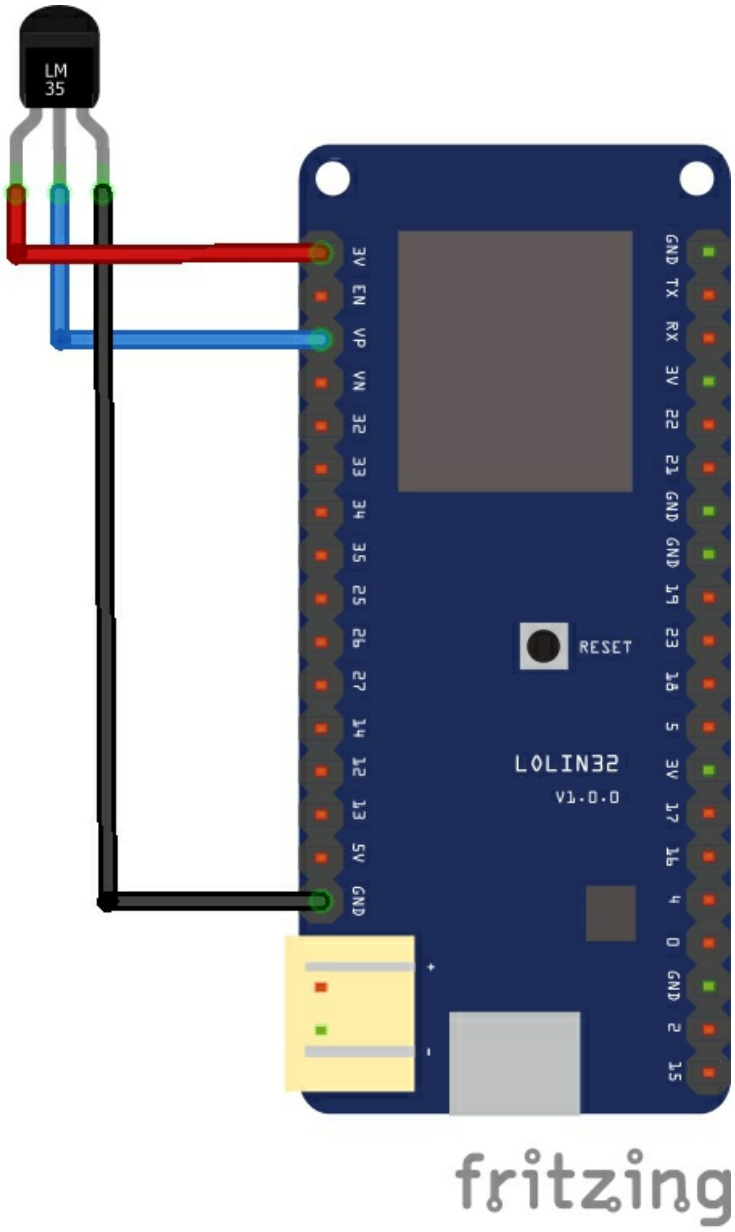
In this example we will connect an LM35 temperature sensor to our ESP32 module

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1/4°C at room temperature and ±3/4°C over a full -55 to +150°C temperature range

Here is a picture of the pins, its important to get these correct or you can damage the sensor, although its stated 5v in the image below we never had any issue connecting this to 3v3

Schematics

Very simple to connect Vcc is 3v3, Gnd is any Gnd and out goes to ESP32 LOLIN 32 A0, you can see this below



Code

```
const int analogIn = A0;

int RawValue= 0;
double Voltage = 0;
double tempC = 0;
double tempF = 0;

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

void loop(){

RawValue = analogRead(analogIn);
Voltage = (RawValue / 2048.0) * 3300; // 5000 to get millivots.
tempC = Voltage * 0.1;
tempF = (tempC * 1.8) + 32; // conver to F
Serial.print("Raw Value = " ); // shows pre-scaled value
Serial.print(RawValue);
Serial.print("\t milli volts = "); // shows the voltage measured
Serial.print(Voltage,0); //
Serial.print("\t Temperature in C = ");
Serial.print(tempC,1);
Serial.print("\t Temperature in F = ");
Serial.println(tempF,1);
delay(500);
}
```

Results

Here are the results via the serial monitor

Raw Value = 173 milli volts = 279 Temperature in C = 27.9 Temperature in F = 82.2
Raw Value = 173 milli volts = 279 Temperature in C = 27.9 Temperature in F = 82.2

ESP32 and MS5611 barometric pressure sensor example

This barometric pressure sensor is optimized for altimeters and variometers with an altitude resolution of 10 cm. The sensor module includes a high linearity pressure sensor and an ultra-low power 24 bit ΔΣ ADC with internal factory calibrated coefficients. It provides a precise digital 24 Bit pressure and temperature value and different operation modes that allow the user to optimize for conversion speed and current consumption.

A high resolution temperature output allows the implementation of an altimeter/thermometer function without any additional sensor. The MS5611-01BA can be interfaced to virtually any microcontroller. The communication protocol is simple, without the need of programming internal registers in the device.

Small dimensions of only 5.0 mm x 3.0 mm and a height of only 1.0 mm allow for integration in mobile devices. This new sensor module generation is based on leading MEMS technology and latest benefits from MEAS Switzerland proven experience and know-how in high volume manufacturing of altimeter modules, which have been widely used for over a decade. The sensing principle employed leads to very low hysteresis and high stability of both pressure and temperature signal.

features

- High resolution module, 10 cm
- Fast conversion down to 1 ms
- Low power, 1 µA (standby < 0.15 µA)
- QFN package 5.0 x 3.0 x 1.0 mm^3
- Supply voltage 1.8 to 3.6 V
- Integrated digital pressure sensor (24 bit ΔΣ ADC)
- Operating range: 10 to 1200 mbar, -40 to +85 °C
- I^2C and SPI interface up to 20 MHz
- No external components (Internal oscillator)
- Excellent long term stability

Connection

ESP32 (lolin32)	Module connection
3v3	Vcc
Gnd	Gnd
SCL (22)	SCL
SDA (21)	SDA

Code

This example comes from the https://github.com/jarzebski/Arduino-MS5611 library

```
#include <Wire.h>
#include <MS5611.h>

MS5611 ms5611;

double referencePressure;

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

   // Initialize MS5611 sensor
  Serial.println("Initialize MS5611 Sensor");

   while(!ms5611.begin())
  {
   Serial.println("Could not find a valid MS5611 sensor, check wiring!");
   delay(500);
  }

   // Get reference pressure for relative altitude
  referencePressure = ms5611.readPressure();

   // Check settings
  checkSettings();
}

void checkSettings()
{
  Serial.print("Oversampling: ");
  Serial.println(ms5611.getOversampling());
}

void loop()
{
  // Read raw values
```

```
  uint32_t rawTemp = ms5611.readRawTemperature();
  uint32_t rawPressure = ms5611.readRawPressure();

   // Read true temperature & Pressure
  double realTemperature = ms5611.readTemperature();
  long realPressure = ms5611.readPressure();

   // Calculate altitude
  float absoluteAltitude = ms5611.getAltitude(realPressure);
  float relativeAltitude = ms5611.getAltitude(realPressure, referencePressure);

   Serial.println("--");

   Serial.print(" rawTemp = ");
  Serial.print(rawTemp);
  Serial.print(", realTemp = ");
  Serial.print(realTemperature);
  Serial.println(" *C");

   Serial.print(" rawPressure = ");
  Serial.print(rawPressure);
  Serial.print(", realPressure = ");
  Serial.print(realPressure);
  Serial.println(" Pa");

   Serial.print(" absoluteAltitude = ");
  Serial.print(absoluteAltitude);
  Serial.print(" m, relativeAltitude = ");
  Serial.print(relativeAltitude);
  Serial.println(" m");

   delay(1000);
}
```

Output

Open the serial monitor and you will see something like this

rawTemp = 8493500, realTemp = 26.08 *C
rawPressure = 8579996, realPressure = 99777 Pa
absoluteAltitude = 129.68 m, relativeAltitude = -2.20 m

ESP32 and MPL3115A2 absolute pressure sensor example

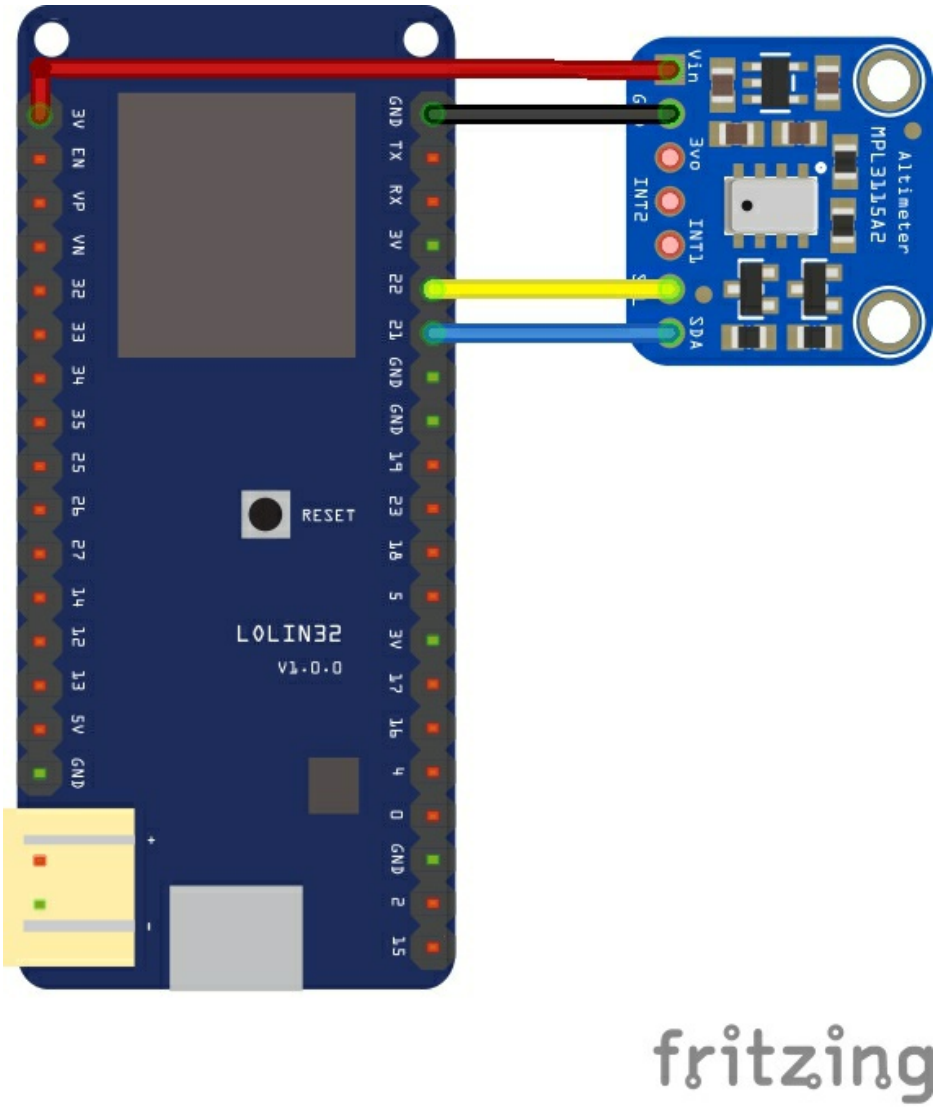
The MPL3115A2 is a compact, piezoresistive, absolute pressure sensor with an I2C digital interface. MPL3115A2 has a wide operating range of 20 kPa to 110 kPa, a range that covers all surface elevations on earth. The MEMS is temperature compensated utilizing an on-chip temperature sensor. The pressure and temperature data is fed into a high resolution ADC to provide fully compensated and digitized outputs for pressure in Pascals and

temperature in °C.

The compensated pressure output can then be converted to altitude, utilizing the formula stated in Section 9.1.3 "Pressure/altitude" provided in meters.

The internal processing in MPL3115A2 removes compensation and unit conversion load from the system MCU, simplifying system design

Schematics/Layout



Code

Again we use a library and again its an adafruit one - https://github.com/adafruit/Adafruit_MPL3115A2_Library

```
#include <Wire.h>
#include <Adafruit_MPL3115A2.h>

// Power by connecting Vin to 3-5V, GND to GND
// Uses I2C - connect SCL to the SCL pin, SDA to SDA pin
// See the Wire tutorial for pinouts for each Arduino
// http://arduino.cc/en/reference/wire
Adafruit_MPL3115A2 baro = Adafruit_MPL3115A2();

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

```
Serial.println("Adafruit_MPL3115A2 test!");
}

void loop() {
if (! baro.begin()) {
Serial.println("Couldnt find sensor");
return;
}

float pascals = baro.getPressure();
// Our weather page presents pressure in Inches (Hg)
// Use http://www.onlineconversion.com/pressure.htm for other units
Serial.print(pascals/3377); Serial.println(" Inches (Hg)");

float altm = baro.getAltitude();
Serial.print(altm); Serial.println(" meters");

float tempC = baro.getTemperature();
Serial.print(tempC); Serial.println("*C");

delay(250);
}
```

Output

Open the serial monitor - this is what I saw

```
Adafruit_MPL3115A2 test!
30.17 Inches (Hg)
-48.25 meters
35.75*C
30.17 Inches (Hg)
-47.81 meters
35.63*C
```

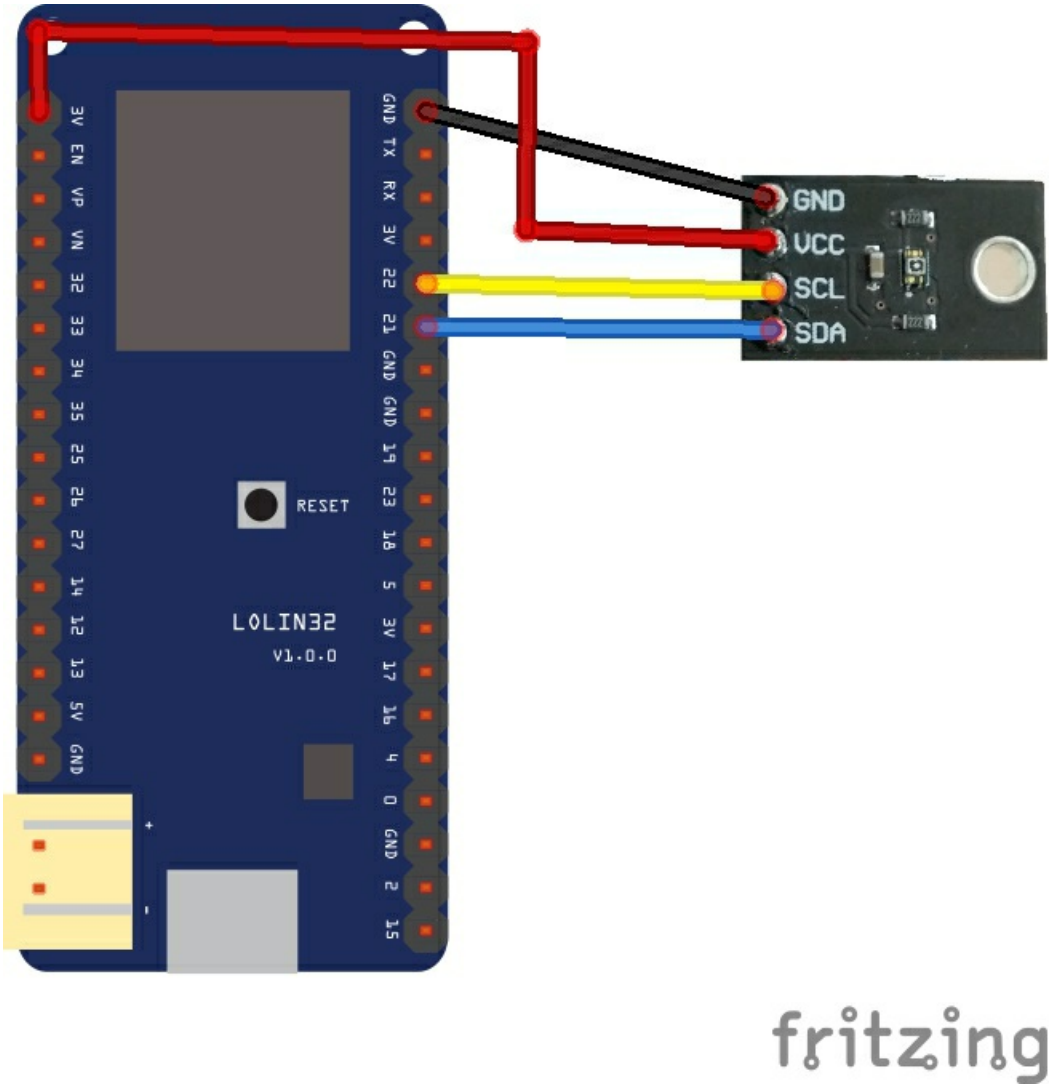
VEML6075 ultraviolet (UV) light sensor and ESP32

The VEML6075 senses UVA and UVB light and incorporates photodiode, amplifiers, and analog / digital circuits into a single chip using a CMOS process. When the UV sensor is applied, it is able to detect UVA and UVB intensity to provide a measure of the signal strength as well as allowing for UVI measurement.

The VEML6075 provides excellent temperature compensation capability for keeping the output stable under changing temperature. VEML6075’s

functionality is easily operated via the simple command format of I2C (SMBus compatible) interface protocol. VEML6075's operating voltage ranges from 1.7 V to 3.6 V.

Schematics/Layout



Code

Again we use a library and again its an adafruit one - https://github.com/NorthernWidget/VEML6075

```
#include <VEML6075.h>
```

```
VEML6075 UV;

void setup()
{
Serial.begin(38400); //Begin Serial
UV.begin(); //Begin the UV module

}

void loop()
{
Serial.print("UVA = ");
Serial.print(UV.GetUVA()); //Get compensated UVA value
Serial.print(" UVB = ");
Serial.println(UV.GetUVB()); //Get compensated UVB value
delay(1000);
}
```

Output

Open the serial monitor - this is what I saw but I tested this indoors

UVA = 0.00 UVB = 3.00
UVA = 0.00 UVB = 0.00
UVA = 0.00 UVB = 0.00
UVA = 0.00 UVB = 2.00

ESP32 and CCS811 gas sensor example

In this example we will connect a CCS811 gas sensor to an ESP32, first of all lets look at the sensor

The CCS811 is a low-power digital gas sensor solution, which integrates a gas sensor solution for detecting low levels of VOCs typically found indoors, with a microcontroller unit (MCU) and an Analog-to-Digital converter to monitor the local environment and provide an indication of the indoor air quality via an equivalent CO2 or TVOC output over a standard I2C digital interface.

I usually like to find a suitable module or breakout top use a sensor, here is the one I chose.

Features

Integrated MCU
On-board processing
Standard digital interface
Optimised low power modes
IAQ threshold alarms
Programmable baseline
2.7mm x 4.0mm LGA package
Low component count
Proven technology platform

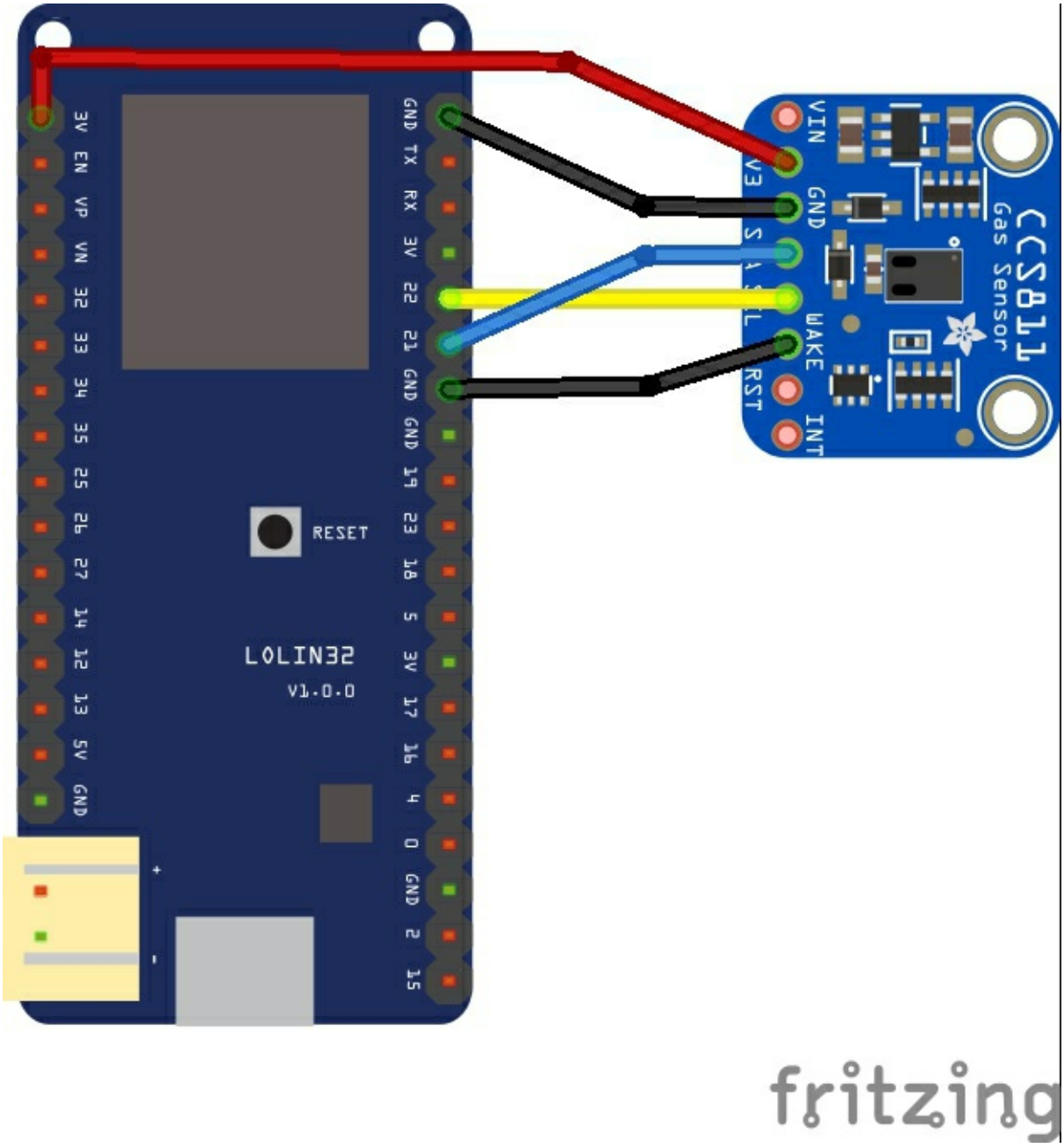
Specs

Interface	I^2C
Supply Voltage [V]	1.8 to 3.6
Power Consumption [mW]	1.2 to 46
Dimension [mm]	2.7 x 4.0 x 1.1 LGA
Ambient Temperature Range [°C]	-40 to 85
Ambient Humidity Range [% r.h.]	10 to 95

Schematics/Layout

Remember and connect WAKE to gnd

Layout



Code

Again we use a library this is the adafruit one - you can use the library manager and add this. And this is the out of the box example

```
#include "Adafruit_CCS811.h"

Adafruit_CCS811 ccs;

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

Serial.println("CCS811 test");

if(!ccs.begin()){
Serial.println("Failed to start sensor! Please check your wiring.");
while(1);
}

//calibrate temperature sensor
while(!ccs.available());
float temp = ccs.calculateTemperature();
ccs.setTempOffset(temp - 25.0);
}

void loop() {
if(ccs.available()){
float temp = ccs.calculateTemperature();
if(!ccs.readData()){
Serial.print("CO2: ");
Serial.print(ccs.geteCO2());
Serial.print("ppm, TVOC: ");
Serial.print(ccs.getTVOC());
Serial.print("ppb Temp:");
Serial.println(temp);
}
else{
Serial.println("ERROR!");
while(1);
}
}
delay(500);
}
```

Output

Open the serial monitor - this is what I saw. The higher CO2 level was when I breathed on the sensor

CO2: 954ppm, TVOC: 84ppb Temp:17.12
CO2: 400ppm, TVOC: 0ppb Temp:13.32
CO2: 400ppm, TVOC: 0ppb Temp:14.63
CO2: 889ppm, TVOC: 74ppb Temp:20.24
CO2: 400ppm, TVOC: 0ppb Temp:20.53

ESP32 and MPU-9250 sensor example

The MPU-9250 is the company's second generation 9-axis Motion Processing Unit™ for smartphones, tablets, wearable sensors, and other consumer markets. The MPU-9250, delivered in a 3x3x1mm QFN package, is the world's smallest 9-axis MotionTracking device and incorporates the latest InvenSense design innovations, enabling dramatically reduced chip size and power consumption, while at the same time improving performance and cost.

The MPU-9250 MotionTracking device sets a new benchmark for 9-axis performance with power consumption only 9.3µA and a size that is 44% smaller than the company's first-generation device. Gyro noise performance is 3x better, and compass full scale range is over 4x better than competitive offerings.
The MPU-9250 is a System in Package (SiP) that combines two chips: the MPU-6500, which contains a 3-axis gyroscope, a 3-axis accelerometer, and an onboard Digital Motion Processor™ (DMP™) capable of processing complex MotionFusion algorithms; and the AK8963, the market leading 3-axis digital compass. The MPU-9250 supports InvenSense's market proven MotionFusion. A single design can support the MPU-9250 or MPU-6500, providing customers the flexibility to support either device in different product SKUs.

Improvements include supporting the accelerometer low power mode with as little as 6.4µA of and it provides improved compass data resolution of 16-bits (0.15 µT per LSB). The full scale measurement range of ±4800µT helps alleviate compass placement challenges on complex pcb's

The MPU-9250 software drivers are fully compliant with Google's Android 4.1 Jelly Bean release, and support new low-power DMP capabilities that offload the host processor to reduce power consumption and simplify application development. The MPU-9250 includes MotionFusion and run-time calibration firmware that enables consumer electronics manufacturers to commercialize cost effective motion-based functionality.

More info - https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/

Connection

LOLIN32 Connection	MPU-9250 connection
3v3	Vcc
Gnd	Gnd
SDA - 21	SDA
SCL - 22	SCL

Code

I used the https://github.com/asukiaaa/MPU9250_asukiaaa - it was the easiest to use, I had to change the SDA and SCL defines for my LOLIN32 board from the default

#define SDA_PIN 21 #define SCL_PIN 22

```
#include <MPU9250_asukiaaa.h>

#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 21
#define SCL_PIN 22
#endif

MPU9250 mySensor;

void setup() {
while(!Serial);

Serial.begin(115200);
Serial.println("started");

#ifdef _ESP32_HAL_I2C_H_
// for esp32
Wire.begin(SDA_PIN, SCL_PIN); //sda, scl
#else
Wire.begin();
#endif

mySensor.setWire(&Wire);

mySensor.beginAccel();
mySensor.beginMag();

// you can set your own offset for mag values
// mySensor.magXOffset = -50;
// mySensor.magYOffset = -55;
// mySensor.magZOffset = -10;
}

void loop() {
mySensor.accelUpdate();
Serial.println("print accel values");
Serial.println("accelX: " + String(mySensor.accelX()));
Serial.println("accelY: " + String(mySensor.accelY()));
Serial.println("accelZ: " + String(mySensor.accelZ()));
Serial.println("accelSqrt: " + String(mySensor.accelSqrt()));

mySensor.magUpdate();
Serial.println("print mag values");
```

```
Serial.println("magX: " + String(mySensor.magX()));
Serial.println("maxY: " + String(mySensor.magY()));
Serial.println("magZ: " + String(mySensor.magZ()));
Serial.println("horizontal direction: " + String(mySensor.magHorizDirection()));

Serial.println("at " + String(millis()) + "ms");
delay(500);
}
```

Output

open the serial monitor and you should get something like this

print accel values
accelX: 0.91
accelY: -0.04
accelZ: 0.39
accelSqrt: 0.99
print mag values
magX: -17
maxY: 71
magZ: -46
horizontal direction: -13.47
at 156214ms

ESP32 and Max7219 8×8 LED matrix example

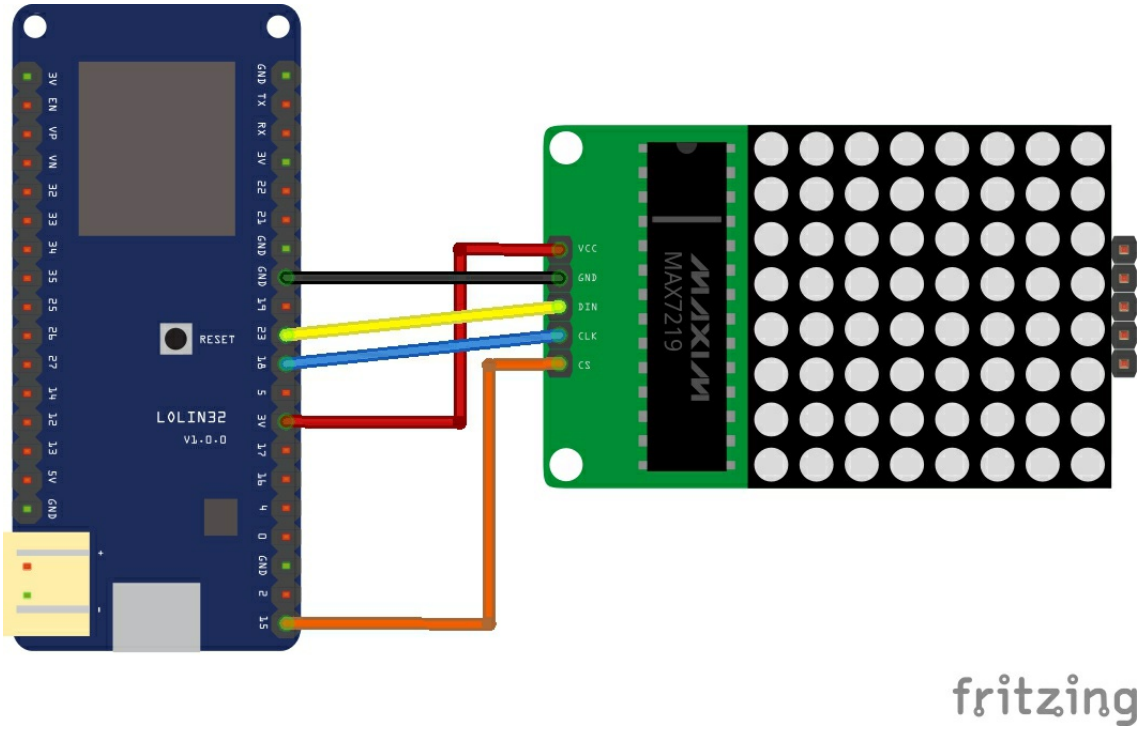
The MAX7219/MAX7221 are compact, serial input/output common-cathode display drivers that interface microprocessors (µPs) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs.

Included on-chip are a BCD code-B decoder, multiplex scan circuitry, segment and digit drivers, and an 8x8 static RAM that stores each digit. Only one external resistor is required to set the segment current for all LEDs.

The MAX7221 is compatible with SPI™, QSPI™, and MICROWIRE™, and has slew-rate-limited segment drivers to reduce EMI.

Here is a picture of a typical module that can be bought from many sources

Schematics



Code

You will need the library from https://github.com/squix78/MAX7219LedMatrix installed

```
#include <SPI.h>
#include "LedMatrix.h"

#define NUMBER_OF_DEVICES 1
#define CS_PIN 15
LedMatrix ledMatrix = LedMatrix(NUMBER_OF_DEVICES, CS_PIN);

void setup()
{
ledMatrix.init();
ledMatrix.setIntensity(4); // range is 0-15
ledMatrix.setText("The quick brown fox jumps over the lazy dog");
}

void loop()
{
ledMatrix.clear();
ledMatrix.scrollTextLeft();
ledMatrix.drawText();
```

```
ledMatrix.commit();
delay(200);
}
```

ESP32 and TM1637 7 segment display example

A common display module that you can buy on the internet contain the Tm1638 driver chip, I was interested in this one which is the TM1637 which appears to be a more basic version which can only control a display, the TM1638 can also control LED's, buttons and two displays at the same time.

This is a common anode 4-digit tube display module which uses the TM1637 driver chip; Only 2 connections are required to control the 4-digit 8-segment displays

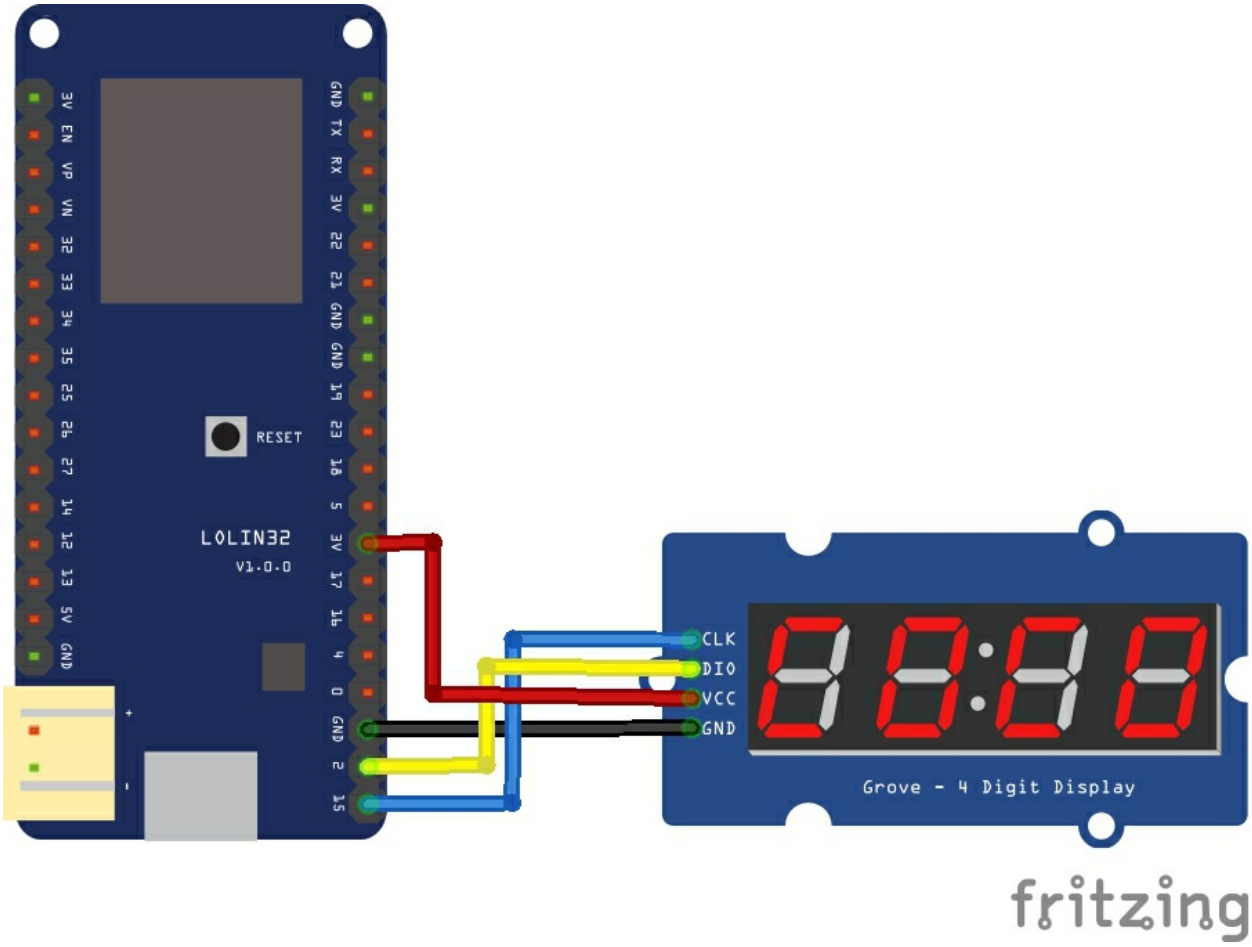
Here is the module

Features of the module

- Display common anode for the four red LED
- Powered supply by 3.3V/5V
- Four common anode tube display module is driven by IC TM1637
- Can be used for Arduino devices, two signal lines can make the MCU control 4 8 digital tube. Digital tube 8 segment is adjustable

Schematic

Here is how to hook the module up, the good news is this worked with my LOLIN32 and 3.3v



Code

There is a library for this IC, you can get it from https://github.com/avishorp/TM1637 , as usual there is a built in example but here is a simple sketch

```
#include <TM1637Display.h>

const int CLK = A13; //Set the CLK pin connection to the display
const int DIO = A12; //Set the DIO pin connection to the display

int numCounter = 0;

TM1637Display display(CLK, DIO); //set up the 4-Digit Display.

void setup()
{
display.setBrightness(0x0a); //set the diplay to maximum brightness
}

void loop()
{
for(numCounter = 0; numCounter < 1000; numCounter++) //Iterate numCounter
{
display.showNumberDec(numCounter); //Display the numCounter value;
delay(1000);
}
}
```

ESP32 and MAX44009 ambient light sensor example

The MAX44009 ambient light sensor features an I²C digital output that is ideal for a number of portable applications such as smartphones, notebooks, and industrial sensors. At less than 1µA operating current, it is the lowest power ambient light sensor in the industry and features an ultra-wide 22-bit dynamic range from 0.045 lux to 188,000 lux.

Low-light operation allows easy operation in dark-glass applications.

The on-chip photodiode's spectral response is optimized to mimic the human eye’s perception of ambient light and incorporates IR and UV blocking capability. The adaptive gain block automatically selects the correct lux range to optimize the counts/lux.

Features

Wide 0.045 Lux to 188,000 Lux Range
VCC = 1.7V to 3.6V
ICC = 0.65µA Operating Current
-40°C to +85°C Temperature Range
Device Address Options - 1001 010x and 1001 011x

Connection

Module Pin	**LOLIN32 Pin**
Vin	3v3
Gnd	Gnd
SCL	22
SDA	21

Code

```
#include<Wire.h>

#define Addr 0x4A

void setup()
{

Wire.begin();
// Initialise serial communication
Serial.begin(9600);

Wire.beginTransmission(Addr);
Wire.write(0x02);
Wire.write(0x40);
Wire.endTransmission();
delay(300);
}

void loop()
{
unsigned int data[2];
Wire.beginTransmission(Addr);
Wire.write(0x03);
Wire.endTransmission();

// Request 2 bytes of data
Wire.requestFrom(Addr, 2);

// Read 2 bytes of data luminance msb, luminance lsb
if (Wire.available() == 2)
{
data[0] = Wire.read();
data[1] = Wire.read();
}

// Convert the data to lux
int exponent = (data[0] & 0xF0) >> 4;
int mantissa = ((data[0] & 0x0F) << 4) | (data[1] & 0x0F);
float luminance = pow(2, exponent) * mantissa * 0.045;

Serial.print("Ambient Light luminance :");
Serial.print(luminance);
Serial.println(" lux");
delay(500);
}
```

Output

Open the serial monitor and change the light intensity on the sensor, here is an example

Ambient Light luminance :24.48 lux
Ambient Light luminance :21.42 lux
Ambient Light luminance :21.42 lux
Ambient Light luminance :9.94 lux
Ambient Light luminance :9.94 lux
Ambient Light luminance :13.77 lux

This handy little table from wikipedia shows some typical lux values

Illuminance (lux)	**Surfaces illuminated by**
0.0001	Moonless, overcast night sky (starlight)
0.002	Moonless clear night sky with airglow
0.05–0.36	Full moon on a clear night[4]
3.4	Dark limit of civil twilight under a clear sky
20–50	Public areas with dark surroundings
50	Family living room lights
80	Office building hallway/toilet lighting
100	Very dark overcast day
320–500	Office lighting
400	Sunrise or sunset on a clear day.
1000	Overcast day; typical TV studio lighting
10,000–25,000	Full daylight (not direct sun)
32,000–100,000	Direct sunlight

ESP32 and OLED display example

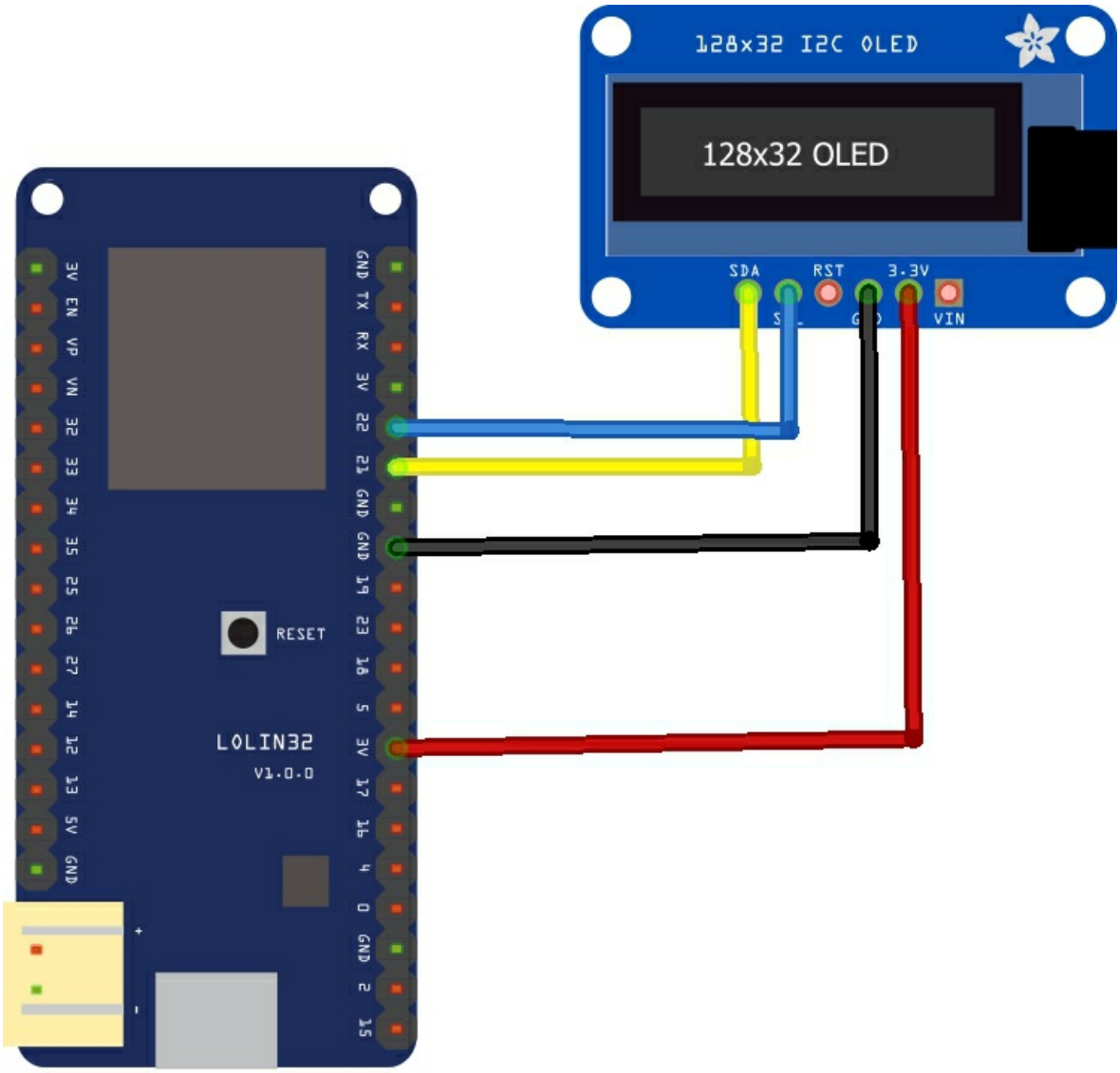
This example uses an OLED display these typically come in a couple of different sizes 128x32 and 128x64, this particular example will use the I2C connection from the Lolin32 to the display. There are a couple of libraries that make life easier. Lets look at a typical oled display

These will come in useful for various projects for example displaying the date and time or maybe temperature readings from a sensor

Connection

Pin Label	LOLIN32 PIN	I²C Function	Notes
GND	Ground	Ground	0V
VCC	Power	Power	Regulated 5V supply.
SDA	SDA / 21	SDA	Serial data in
SCL	SCL / 22	SCL	I²C clock

This layout shows a 128x32 connected to the LOLIN32, 128x64 I2C devices would be the same


128x32 I2C OLED
128x32 OLED
SDA
RST
3.3V
VIN
RESET
LOLIN32
V1.0.0

fritzing

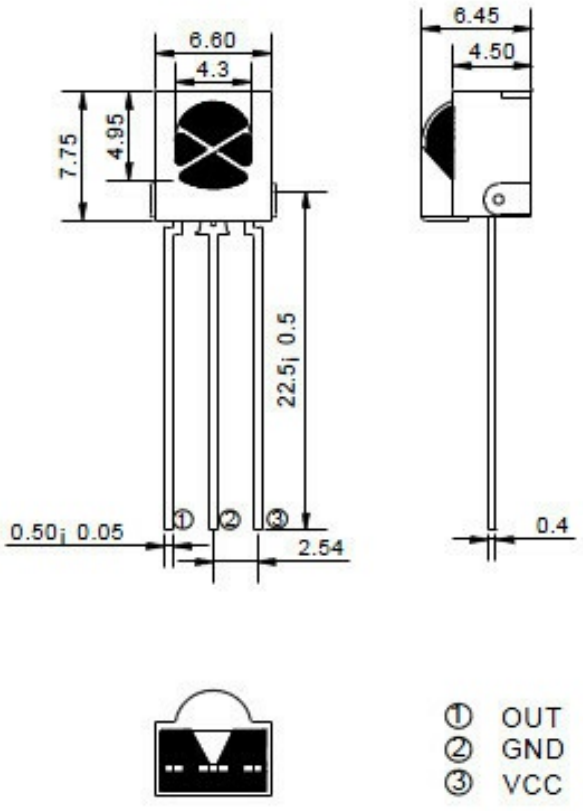
Code

This example uses the https://github.com/adafruit/Adafruit_SSD1306/archive/master.zip and https://github.com/adafruit/Adafruit-GFX-Library/archive/master.zip , there are several built in examples. I have modified one just to display text as further examples will write text to a display

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

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);


void setup()
{
  Serial.begin(9600);
  // by default, we'll generate the high voltage from the 3.3v line internally! (neat!)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C (for the
128x32)
  // init done
  display.clearDisplay();
  // text display tests
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Hello, world!");
  display.setTextColor(BLACK, WHITE); // 'inverted' text
  display.println(3.141592);
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.print("0x");
  display.println(0xDEADBEEF, HEX);
  display.display();
  display.clearDisplay();

}


void loop()
{


}
```

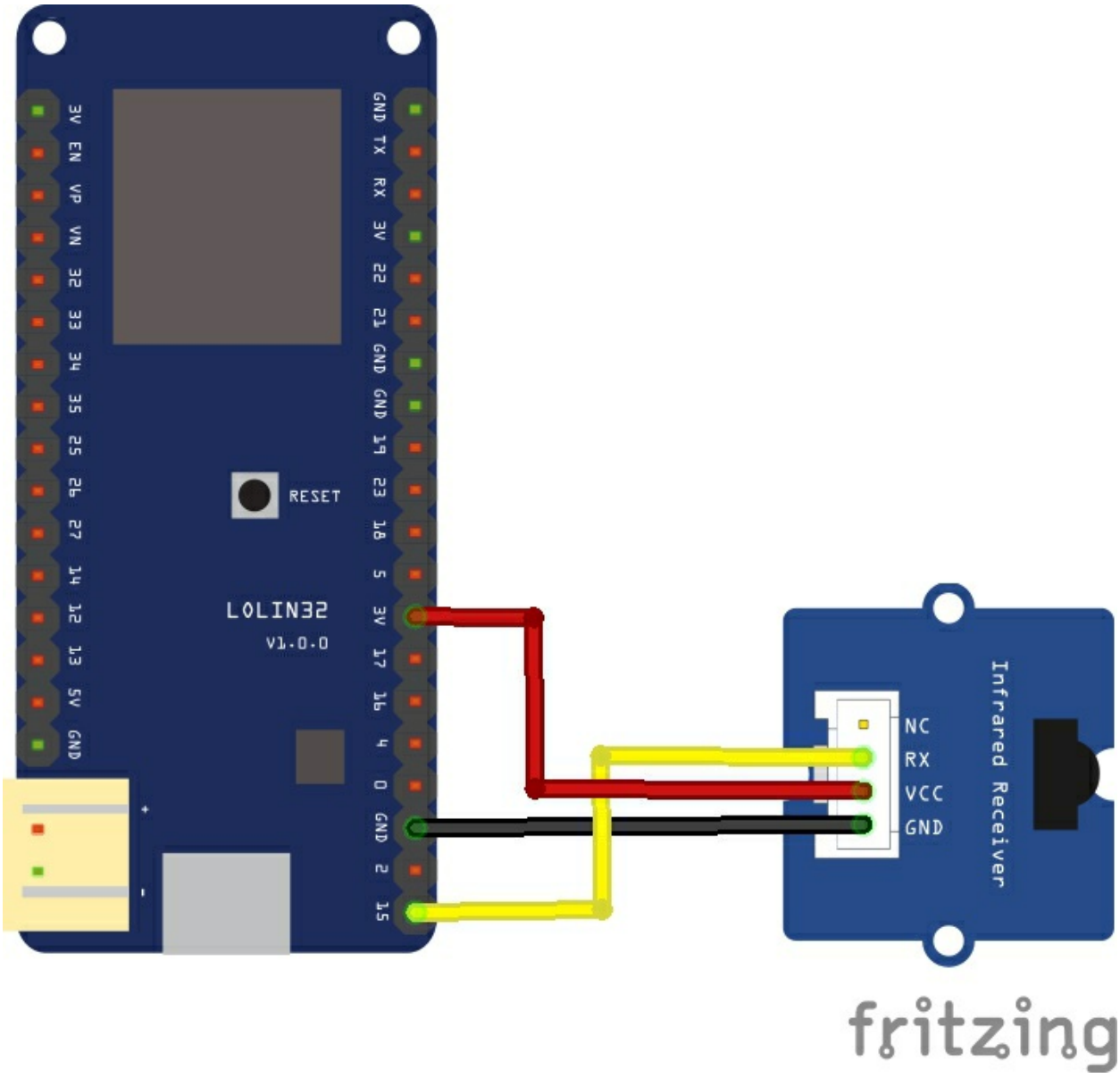
ESP32 and Infrared receiver example

In this example we look at how to connect an IR Reciever. Generally, they require Vcc(5v), GND and there is a data out which you connect to your ESP32. Here is a typical IR showing the pinout. I managed to get mine working just fine with the 3.3v from the ESP32 board



Many electronic shops online stock breakouts for these. Here is a picture of the remote control that I used for testing, there are many variants of these available

Layout


3V EN VP VN 32 33 34 35 25 26 27 14 12 13 5V GND
GND TX RX 3V 22 21 GND GND 19 23 18 5 3V 17 16 4 0 GND 2 15
RESET
LOLIN32
V1.0.0
NC
RX
VCC
GND
Infrared Receiver
fritzing

Code

You'll need the IR Remote library, you can get this from https://github.com/shirriff/Arduino-IRremote

Download and import or copy into your Arduino -> Library folder. As usual this library will be doing most of the work making it easier for ourselves.

```
#include <IRremote.h>

int RECV_PIN = 15;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
Serial.begin(9600);
irrecv.enableIRIn(); // Start the receiver
}

void loop()
{
if (irrecv.decode(&results))
{
Serial.println(results.value, HEX);
irrecv.resume();
}
}
```

Testing

I opened the serial monitor and pressed various keys on my remote here is what was displayed

FFA25D
FFFFFFFF
FFE21D
FF22DD
FFFFFFFF
FF02FD
FFFFFFFF
FFC23D
F076C13B
FFFFFFFF
FFA857
FF906F
FFFFFFFF
FF6897
FFFFFFFF
FFFFFFFF
FF9867
FFFFFFFF
FFB04F
FFFFFFFF
FF30CF

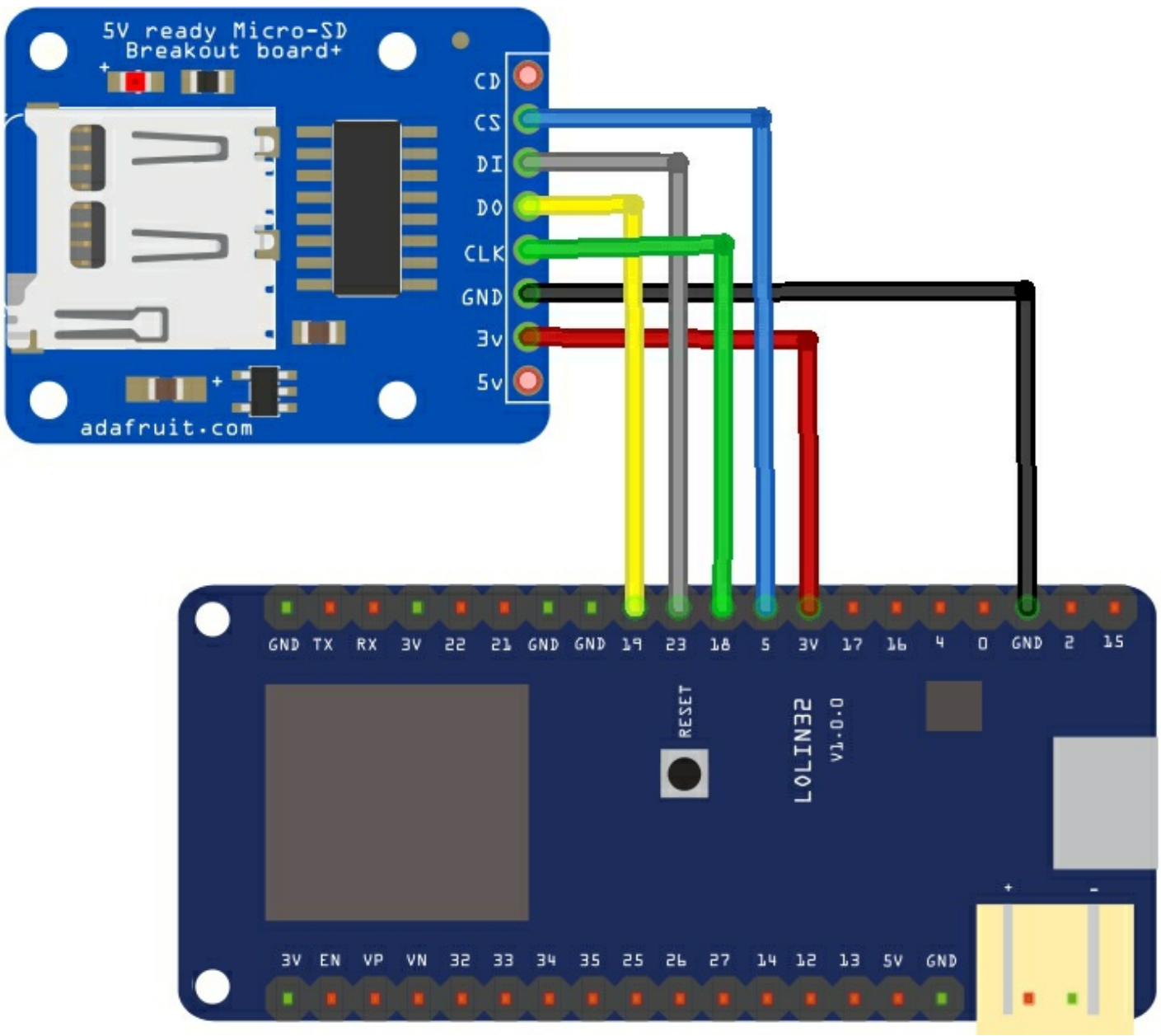
As you can see with a bit of programming we can take these values and put them to use.

ESP32 and SD card example

In this example we connect an SD card to our ESP32, we will log analog readings to a file on the SD card.
Here is our micro sd module

Here is the layout

Layout


5V ready Micro-SD
Breakout board+
CD
CS
DI
DO
CLK
GND
3v
5v
adafruit.com
GND TX RX 3V 22 21 GND GND 19 23 18 5 3V 17 16 4 0 GND 2 15
RESET
LOLIN32
V1.0.0
3V EN VP VN 32 33 34 35 25 26 27 14 12 13 5V GND
fritzing

Code

This example seems to get installed when you add support for ESP32 boards to the arduino IDE - the standard Arduino sd card example does not work

```
/*
* Connect the SD card to the following pins:
*
* SD Card | ESP32
* D2 -
* D3 SS
* CMD MOSI
* VSS GND
* VDD 3.3V
* CLK SCK
* VSS GND
* D0 MISO
* D1 -
*/
#include "FS.h"
#include "SD.h"
#include "SPI.h"

void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
Serial.printf("Listing directory: %s\n", dirname);

File root = fs.open(dirname);
if(!root){
Serial.println("Failed to open directory");
return;
}
if(!root.isDirectory()){
Serial.println("Not a directory");
return;
}

File file = root.openNextFile();
while(file){
if(file.isDirectory()){
Serial.print(" DIR : ");
Serial.println(file.name());
if(levels){
listDir(fs, file.name(), levels -1);
}
} else {
Serial.print(" FILE: ");
Serial.print(file.name());
Serial.print(" SIZE: ");
```

```
Serial.println(file.size());
}
file = root.openNextFile();
}
}

void createDir(fs::FS &fs, const char * path){
Serial.printf("Creating Dir: %s\n", path);
if(fs.mkdir(path)){
Serial.println("Dir created");
} else {
Serial.println("mkdir failed");
}
}

void removeDir(fs::FS &fs, const char * path){
Serial.printf("Removing Dir: %s\n", path);
if(fs.rmdir(path)){
Serial.println("Dir removed");
} else {
Serial.println("rmdir failed");
}
}

void readFile(fs::FS &fs, const char * path){
Serial.printf("Reading file: %s\n", path);

File file = fs.open(path);
if(!file){
Serial.println("Failed to open file for reading");
return;
}

Serial.print("Read from file: ");
while(file.available()){
Serial.write(file.read());
}
}

void writeFile(fs::FS &fs, const char * path, const char * message){
Serial.printf("Writing file: %s\n", path);

File file = fs.open(path, FILE_WRITE);
if(!file){
Serial.println("Failed to open file for writing");
return;
}
if(file.print(message)){
Serial.println("File written");
} else {
Serial.println("Write failed");
```

```
}
}

void appendFile(fs::FS &fs, const char * path, const char * message){
Serial.printf("Appending to file: %s\n", path);

File file = fs.open(path, FILE_APPEND);
if(!file){
Serial.println("Failed to open file for appending");
return;
}
if(file.print(message)){
Serial.println("Message appended");
} else {
Serial.println("Append failed");
}
}

void renameFile(fs::FS &fs, const char * path1, const char * path2){
Serial.printf("Renaming file %s to %s\n", path1, path2);
if (fs.rename(path1, path2)) {
Serial.println("File renamed");
} else {
Serial.println("Rename failed");
}
}

void deleteFile(fs::FS &fs, const char * path){
Serial.printf("Deleting file: %s\n", path);
if(fs.remove(path)){
Serial.println("File deleted");
} else {
Serial.println("Delete failed");
}
}

void testFileIO(fs::FS &fs, const char * path){
File file = fs.open(path);
static uint8_t buf[512];
size_t len = 0;
uint32_t start = millis();
uint32_t end = start;
if(file){
len = file.size();
size_t flen = len;
start = millis();
while(len){
size_t toRead = len;
if(toRead > 512){
toRead = 512;
}
```

```
file.read(buf, toRead);
len -= toRead;
}
end = millis() - start;
Serial.printf("%u bytes read for %u ms\n", flen, end);
file.close();
} else {
Serial.println("Failed to open file for reading");
}

file = fs.open(path, FILE_WRITE);
if(!file){
Serial.println("Failed to open file for writing");
return;
}

size_t i;
start = millis();
for(i=0; i<2048; i++){
file.write(buf, 512);
}
end = millis() - start;
Serial.printf("%u bytes written for %u ms\n", 2048 * 512, end);
file.close();
}

void setup(){
Serial.begin(115200);
if(!SD.begin()){
Serial.println("Card Mount Failed");
return;
}
uint8_t cardType = SD.cardType();

if(cardType == CARD_NONE){
Serial.println("No SD card attached");
return;
}

Serial.print("SD Card Type: ");
if(cardType == CARD_MMC){
Serial.println("MMC");
} else if(cardType == CARD_SD){
Serial.println("SDSC");
} else if(cardType == CARD_SDHC){
Serial.println("SDHC");
} else {
Serial.println("UNKNOWN");
}

uint64_t cardSize = SD.cardSize() / (1024 * 1024);
```

```
Serial.printf("SD Card Size: %lluMB\n", cardSize);

listDir(SD, "/", 0);
createDir(SD, "/mydir");
listDir(SD, "/", 0);
removeDir(SD, "/mydir");
listDir(SD, "/", 2);
writeFile(SD, "/hello.txt", "Hello ");
appendFile(SD, "/hello.txt", "World!\n");
readFile(SD, "/hello.txt");
deleteFile(SD, "/foo.txt");
renameFile(SD, "/hello.txt", "/foo.txt");
readFile(SD, "/foo.txt");
testFileIO(SD, "/test.txt");
}

void loop(){

}
```

Output

Open the serial monitor - this was my sample micro sd card

```
SD Card Type: SDSC
SD Card Size: 241MB
Listing directory: /
FILE: /MCP9808.TXT SIZE: 1496
FILE: /HDC1000.CSV SIZE: 836
FILE: /test.txt SIZE: 0
FILE: /foo.txt SIZE: 13
DIR : /System Volume Information
FILE: /miniwoof.bmp SIZE: 57654
FILE: /test.bmp SIZE: 230456
FILE: /woof.bmp SIZE: 230456
Creating Dir: /mydir
Dir created
Listing directory: /
FILE: /MCP9808.TXT SIZE: 1496
FILE: /HDC1000.CSV SIZE: 836
FILE: /test.txt SIZE: 0
FILE: /foo.txt SIZE: 13
DIR : /mydir
DIR : /System Volume Information
FILE: /miniwoof.bmp SIZE: 57654
FILE: /test.bmp SIZE: 230456
FILE: /woof.bmp SIZE: 230456
Removing Dir: /mydir
Dir removed
Listing directory: /
FILE: /MCP9808.TXT SIZE: 1496
FILE: /HDC1000.CSV SIZE: 836
```

FILE: /test.txt SIZE: 0
FILE: /foo.txt SIZE: 13
DIR : /System Volume Information
Listing directory: /System Volume Information
FILE: /System Volume Information/IndexerVolumeGuid SIZE: 76
FILE: /miniwoof.bmp SIZE: 57654
FILE: /test.bmp SIZE: 230456
FILE: /woof.bmp SIZE: 230456
Writing file: /hello.txt
File written
Appending to file: /hello.txt
Message appended
Reading file: /hello.txt
Read from file: Hello World!
Deleting file: /foo.txt
File deleted
Renaming file /hello.txt to /foo.txt
File renamed
Reading file: /foo.txt
Read from file: Hello World!
0 bytes read for 0 ms
1048576 bytes written for 18725 ms

MH ET LIVE ESP32 MINI KIT and WS2812B shield example

In this example we connect a Wemos Ws2812B RGB shield to a MH ET LIVE ESP32 MINI KIT (see earlier in this ebook for details of this module) , we will then loop through red, green and blue colors

The WS2812 is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant current control part, effectively ensuring the pixel point light color height consistent. The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel， the signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission. LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angle is large, good consistency,

low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

These are also sold and known as Neopixels

Code

You will need to add the Adafruit Neopixel library to your Arduino IDE - https://github.com/adafruit/Adafruit_NeoPixel

```
#include <Adafruit_NeoPixel.h>

#define PIN 21

//the Wemos WS2812B RGB shield has 1 LED connected to pin 2 (IO21)
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);

void setup()
{
pixels.begin(); // This initializes the NeoPixel library.
}

void loop()
{
setColor(255,0,0,1000); //red
setColor(0,255,0,1000); //green
setColor(0,0,255,1000); //blue
}

//simple function which takes values for the red, green and blue led and also
//a delay
void setColor(int redValue, int greenValue, int blueValue, int delayValue)
{
pixels.setPixelColor(0, pixels.Color(redValue, greenValue, blueValue));
pixels.show();
delay(delayValue);
}
```

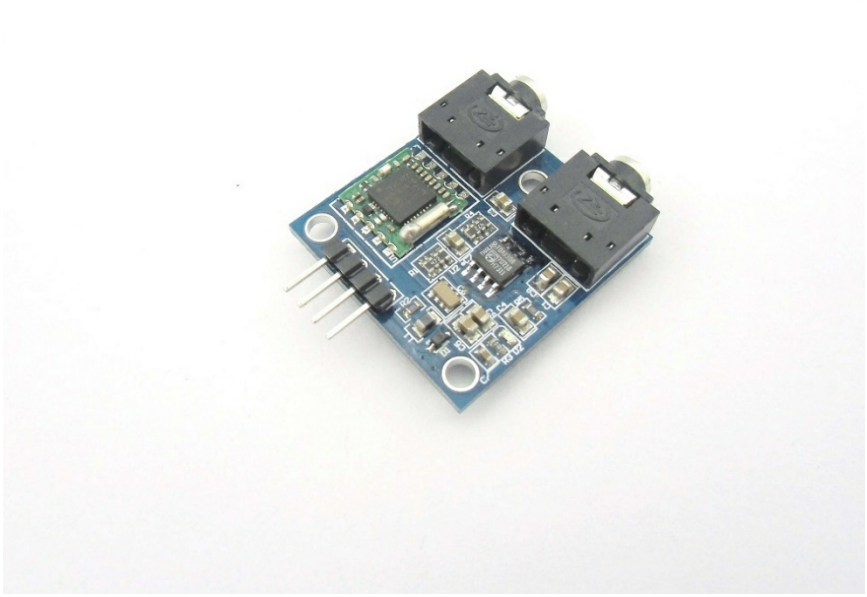
ESP32 and basic TEA5767 example

The TEA5767HN is a single-chip electronically tuned FM stereo radio for low-voltage applications with fully integrated Intermediate Frequency (IF) selectivity and demodulation. The radio is completely adjustment-free and only requires a minimum of small and low cost external components. The radio can be tuned to the European, US, and Japanese FM bands

At first I bought just the module and found it impossible to work with due to its small size, then I discovered this handy module which included the ability to connect speakers or headphone, an antenna connection and most usefully it has an onboard audio amplifier which removes the need for connecting one of these to the TEA5767. The module simply requires power and uses the I2C connection to your Arduino. Here is a picture of the module

Here are some of the key features of the module

Power Supply: 5V
Frequency range :76-108MHZ
You can plug the antenna port directly
I2C bus communication
Board uses the TDA1308 as audio amplifier
Onboard 3.5MM audio interface ,it can be directly connected to the headphone amplifier and so on.



Connection

LOLIN32 Pin	TEA5767 Pin
3v3	VCC
GND	GND
21	SDA
22	SCL

Code

This is a simple code example, which I have set to a local radio station where I live for testing. it requires the TEA 5767 library.

There are other libraries available but this does just fine for a quick example - https://github.com/simonmonk/arduino_TEA5767

```
// TEA5767 Example

#include <Wire.h>
#include <TEA5767Radio.h>

TEA5767Radio radio = TEA5767Radio();

void setup()
{
  Wire.begin();
  radio.setFrequency(102.8); // pick your own frequency
}

void loop()
{
}
```

Links

Here is the datasheet - www.voti.nl/docs/TEA5767.pdf

ESP32 and I2C LCD example

In this example we will interface to an I2C LCD using our ESP32. Now these I2C LCD's consist of 2 parts usually an **HD44780** 16×2 LCD and an I2C backpack which connects to the LCD exposing the standard power and I2C pins.

This is a typical module you can buy, you can see the backpack which is obviously on the back of the LCD

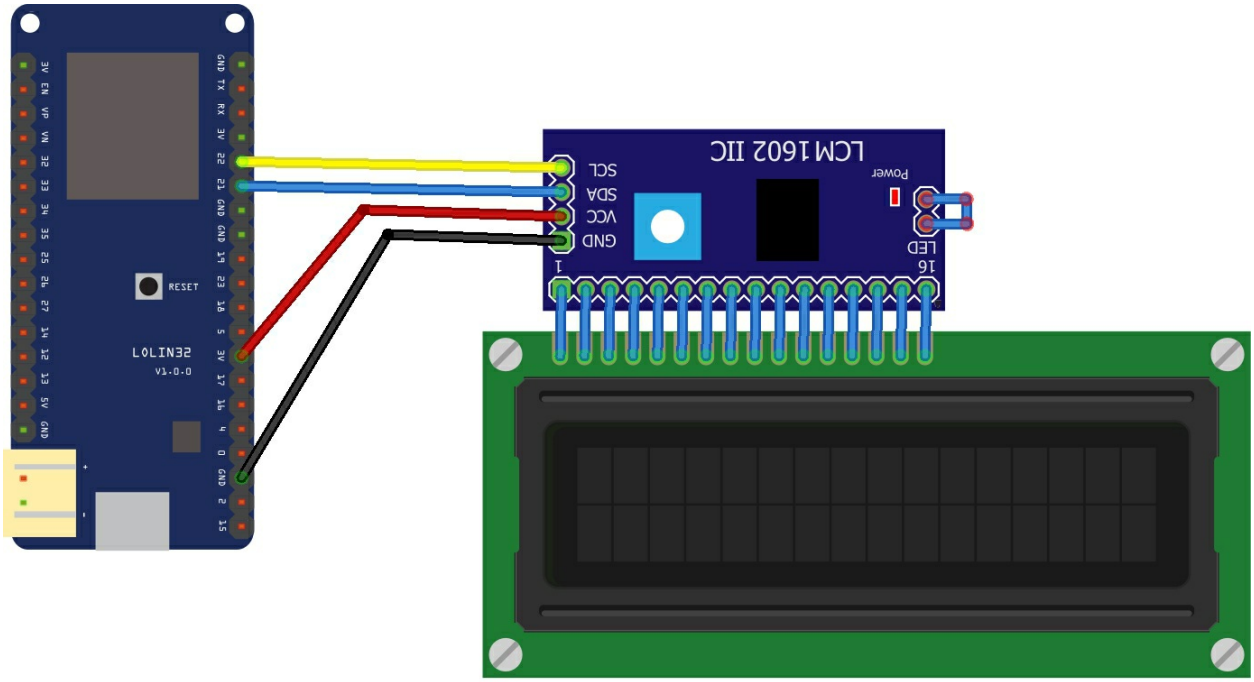
We will now look at the connections and a simple layout

Layout

Connect the pins as follows:

Wemos LOLIN32	**LCD1602**
GND	GND
3v3	VCC
SDA/21	SDA
SCL/22	SCL

Remember the backpack and lcd connections are already made for you, they are just shown in this layout for reference



Code

You will need an updated I2C LCD library, the original one I couldn't get to work but this one does seem to work - http://www.esp32learning.com/wp-content/uploads/2017/12/LiquidCrystal_I2C-master.zip

You will need to import this into the IDE as usual

Now my example below required the I2C address to be changed to 0x3F, a lot of the examples I have looked at are set to 0x27

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2); // set the LCD address to 0x3F for a 16 chars and 2 line display

void setup()
{
lcd.init(); // initialize the lcd
lcd.init();
// Print a message to the LCD.
lcd.backlight();
lcd.setCursor(0,0);
lcd.print("Hello world");
lcd.setCursor(1,0);
lcd.print("ESP32 I2C LCD");

}

void loop()
{
}
```

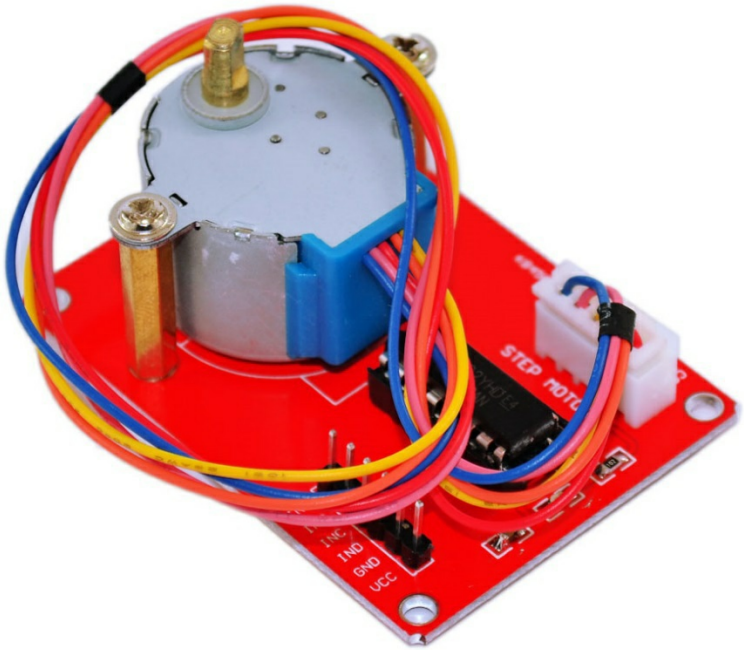
All going well you should see the messages in the code above on your LCD display

ESP32 and a Stepper motor

In this example we will show a basic stepper motor example, we use a driver board that can be bought on many sites which basically comprises of a ULN2003 IC and a few other components, you can use either 5v or 12v, the motor that comes quite often with this board is a 5v type, you can see it further down the page.

Here is the board that I used for this example and here are the connections

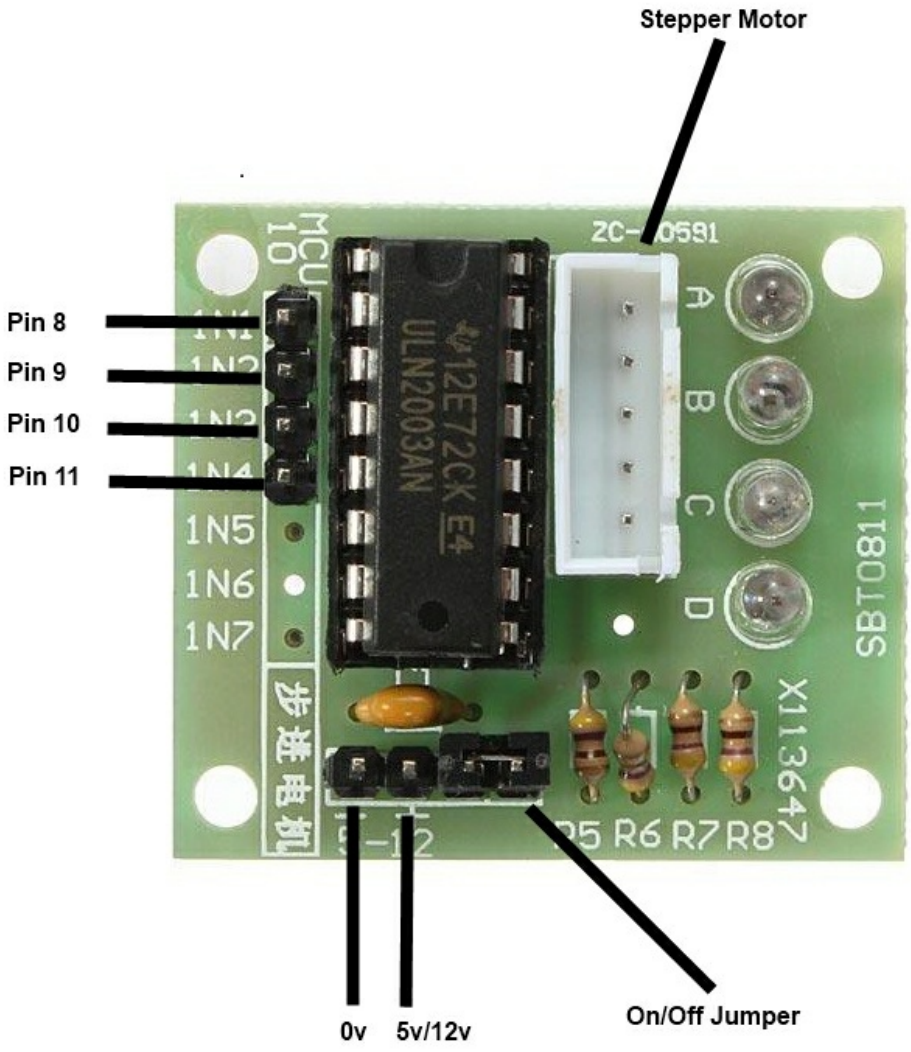
Wemos LOLIN32 IO15 -> IN1
Wemos LOLIN32 IO2 -> IN2
Wemos LOLIN32 IO0 -> IN3
Wemos LOLIN32 IO4 -> IN4



Diameter: 28mm; Voltage: 5V; Step angles: 5.625 x 1/64; Speed reduction ratio: 1/64; Power consumption: About 5V / 63mA; Load pull in frequency: >500Hz; Load pull out frequency: >900Hz; 5-wire 4-phase can be driven with an ordinary ULN2003A chip

I powered the module externally

There are other similar boards where the motor is separate from the board but it’s the same stepper motor and it uses a ULN2003 other similar boards where the motor is separate from the board but its the same stepper motor and it uses a ULN2003



The important point with these boards is that I do not recommend powering them from your ESP32 development board, you should use an external power source.

Code

The code uses the built in stepper library, this is a fairly basic example

```
#include <Stepper.h>

const int stepsPerRevolution = 200; // change this to fit the number of steps per revolution

// initialize the stepper library on pins 15,2,0,4
Stepper myStepper(stepsPerRevolution, 15,2,0,4);

int stepCount = 0; // number of steps the motor has taken

void setup() {

}

void loop() {
// step one step:
myStepper.step(1);
stepCount++;
delay(100);
}
```

ESP32 and L9110 fan module example

In this example we connect an ESP32 to a dual L9110 fan module.

This is a commonly found, basic low cost module which consists of an L9110 chip and a small motor attached. You need 4 connections between the ESP32 and the module. VCC, GND , INA and INB.

You should use an external power source for Vcc and Gnd

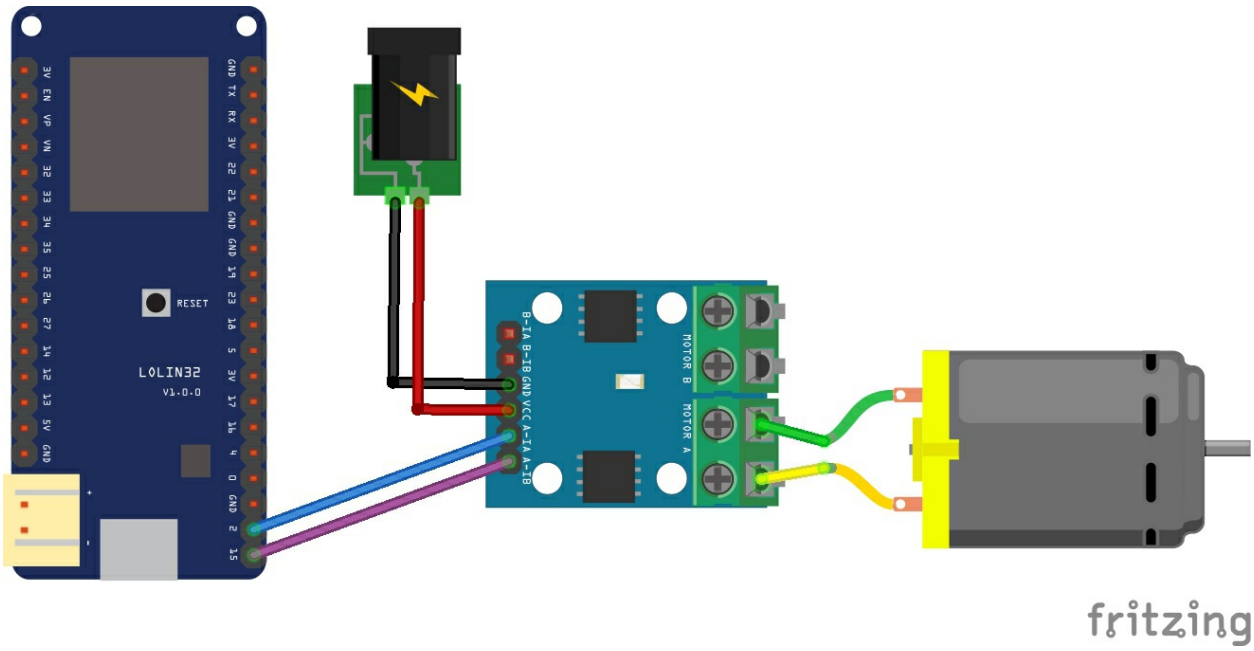
The L9110 The ASIC device control and drive motor design two-channel push-pull power amplifier discrete circuits integrated into a monolithic IC, peripheral devices and reduce the cost, improve the reliability of the whole. This chip has two TTL / CMOS compatible with the level of the input, with good resistance; two output terminals can directly forward and reverse movement of the drive motor, it has a large current driving capability, each channel through 750 ~ 800mA of continuous current, peak current capability up to 1.5 ~ 2.0A; while it has a low output saturation voltage; built-in clamp diode reverse the impact of the current release inductive load it in the drive relays, DC motors, stepper motor or switch power tube use on safe and reliable. The L9110 is widely used in toy car motor drives, stepper motor drive and switching power tube circuit.

- Motor Voltage: 2.5 ~12V
- Motor channels: 2
- Max Continuous Current per Channel: 800mA
- Size: 31mm x 22mm x 12mm

this is a picture of a typical module

Lets look at how to connect the ESP32 to the module

Layout



Code

No libraries needed in this example, fairly basic example this one, upload the sketch and the fan will just run in one direction at one speed. if you have one of the dual modules you can have 2 motors fitted and technically move in any direction

```
int INA = 2;
int INB = 15;

void setup()
{
pinMode(INA,OUTPUT);
pinMode(INB,OUTPUT);
}

void loop()
{
digitalWrite(INA,LOW);
digitalWrite(INB,HIGH);
```

```
delay(1000);
}
```

ESP32 and GY-21P readings on a web page

In this example we connect a GY-21P sensor to an ESP32 and then we will display the readings on a webpage

The GY-21P is an interesting module in that it combines a BMP280 sensor and an SI7021 sensor. The on-board BMP280+SI7021 sensor measures atmospheric pressure from 30kPa to 110kPa as well as relative humidity and temperature.

BMP280
Pressure range: 300-1100 hPa (9000 meters above sea level at -500m)
Relative accuracy (at 950 - 1050 hPa at 25 ° C): ± 0.12 hPa, equiv. to ± 1 m
Absolute accuracy (at (950 - 1050 hPa, 0 - +40 ° C): ± 0.12 hPa, equiv. To ± 1 m
Mains voltage: 1.8V - 3.6V
Power consumption: 2.7µA at 1Hz readout rate
Temperature range: -40 to + 85 ° C

SI7021
HVAC/R
Thermostats/humidistats
Respiratory therapy
White goods
Indoor weather stations
Micro-environments/data centers
Automotive climate control and defogging
Asset and goods tracking
Mobile phones and tablets
Size: 1.3*1cm/0.51*0.39"

Features

Operation Voltage: 3.3V
I2C & SPI Communications Interface
Temp Range: -40C to 85C
Humidity Range: 0 - 100% RH, =-3% from 20-80%

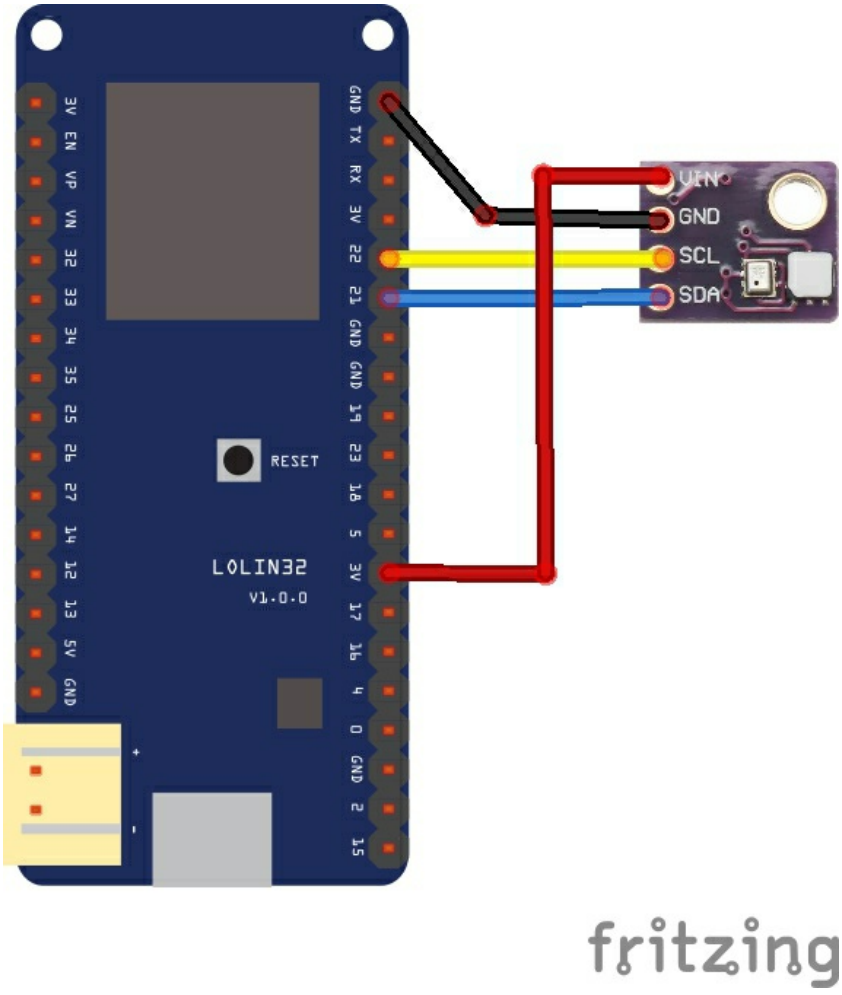
Pressure Range: 30,000Pa to 110,000Pa, relative accuracy of 12Pa, absolute accuracy of 100Pa
Altitude Range: 0 to 30,000 ft (9.2 km), relative accuracy of 3.3 ft (1 m) at sea level, 6.6 (2 m) at 30,000 ft.

Parts List

Part	Link
ESP32	New ESP-32 esp32 Lite V1.0.0 For Rev1 wifi Module + bluetooth board 4MB FLASH
GY-21P	GY-21P Atmospheric Humidity Temperature Sensor Breakout Barometric Pressure BMP280 SI7021 For Arduino
Connecting cable	Free shipping Dupont line 120pcs 20cm male to male + male to female and female to female jumper wire

Schematics/Layout

Connect the sensor to the ESP32



Code

I use a variety of Adafruit libraries, took the default examples and made the following out of them

https://github.com/adafruit/Adafruit_Sensor
https://github.com/adafruit/Adafruit_BMP280_Library
https://github.com/adafruit/Adafruit_Si7021

I got the sea level pressure value from
https://www.weatheronline.co.uk/weather/maps/current?LANG=en&CONT=euro&REGION=0003&LAND=UK&LEVEL=4&R=310

```
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include "Adafruit_Si7021.h"

const char* ssid = "yourname";
const char* password = "yourpasssword";

Adafruit_BMP280 bmp; // I2C
Adafruit_Si7021 sensor = Adafruit_Si7021();
WiFiServer server(80);

void setup()
{
Serial.begin(115200);
pinMode(5, OUTPUT); // set the LED pin mode
if (!bmp.begin())
{
Serial.println("Could not find a valid BMP280 sensor, check wiring!");
while (1);
}

if (!sensor.begin())
{
Serial.println("Did not find Si7021 sensor!");
while (true);
}
delay(10);

// We start by connecting to a WiFi network

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

```
WiFi.begin(ssid, password);

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

Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

server.begin();

}

int value = 0;

void loop(){
WiFiClient client = server.available(); // listen for incoming clients

if (client)
{ // if you get a client,
Serial.println("New Client."); // print a message out the serial port
String currentLine = ""; // make a String to hold incoming data from the client
while (client.connected())
{ // loop while the client's connected
if (client.available())
{ // if there's bytes to read from the client,
char c = client.read(); // read a byte, then
Serial.write(c); // print it out the serial monitor
if (c == '\n')
{ // if the byte is a newline character

// if the current line is blank, you got two newline characters in a row.
// that's the end of the client HTTP request, so send a response:
if (currentLine.length() == 0)
{
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Connnection: close");
client.println();
client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.println("<meta http-equiv=\"refresh\" content=\"5\">");
client.println("<br />");
//bmp280 part
client.println("<h3>BMP280 readings</h3>");
client.print("Pressure (Pa): ");
client.println((float)bmp.readPressure(), 1);
client.println("<br />");
client.print("Temperature (C): ");
```

```
client.println((float)bmp.readTemperature(), 1);
client.println("<br />");
client.print("Altitude (m): ");
client.println((float)bmp.readAltitude(1024), 1); // this should be adjusted to your local forcase
client.println("<br />");
//SI7021 part
client.println("<h3>SI7021 readings</h3>");
client.print("Humidity (%): ");
client.println((float)sensor.readHumidity(), 1);
client.println("<br />");
client.print("Temperature (C): ");
client.println((float)sensor.readTemperature(), 1);
client.println("<br />");

client.println("</html>");
break;
// break out of the while loop:
break;
}
else
{ // if you got a newline, then clear currentLine:
currentLine = "";
}
}
else if (c != '\r')
{ // if you got anything else but a carriage return character,
currentLine += c; // add it to the end of the currentLine
}
}
}
// close the connection:
client.stop();
Serial.println("Client Disconnected.");
}
}
```

Output

Open the serial monitor and take a note of the IP address

Open your favourite web browser and type in the IP address from above, you should see something like this

BMP280 readings

Pressure (Pa): 101516.0
Temperature (C): 22.7
Altitude (m): 73.1

SI7021 readings

Humidity (%): 47.7
Temperature (C): 22.1

ESP32 and CCS811 gas sensor data to Thingspeak example

In this example we connect a CCS811 gas sensor to an ESP32 and then we will upload the data to Thingspeak

The CCS811 is a low-power digital gas sensor solution, which integrates a gas sensor solution for detecting low levels of VOCs typically found indoors, with a microcontroller unit (MCU) and an Analog-to-Digital converter to monitor the local environment and provide an indication of the indoor air quality via an equivalent CO2 or TVOC output over a standard I2C digital

interface.

Thingspeak setup

You will now need to create a new account at thingspeak - https://thingspeak.com. Once done create a new channel and add fields called temperature, CO2 and TVOC.

You can see this in a screen capture of my simple channel, notice the ChannelID you will need that in your code later.

You can also fill in other fields such as Name, description and there are a few others as well. The key one(s) are Field1, Field 2 and Field 3 - this effectively is the data you send to thingspeak

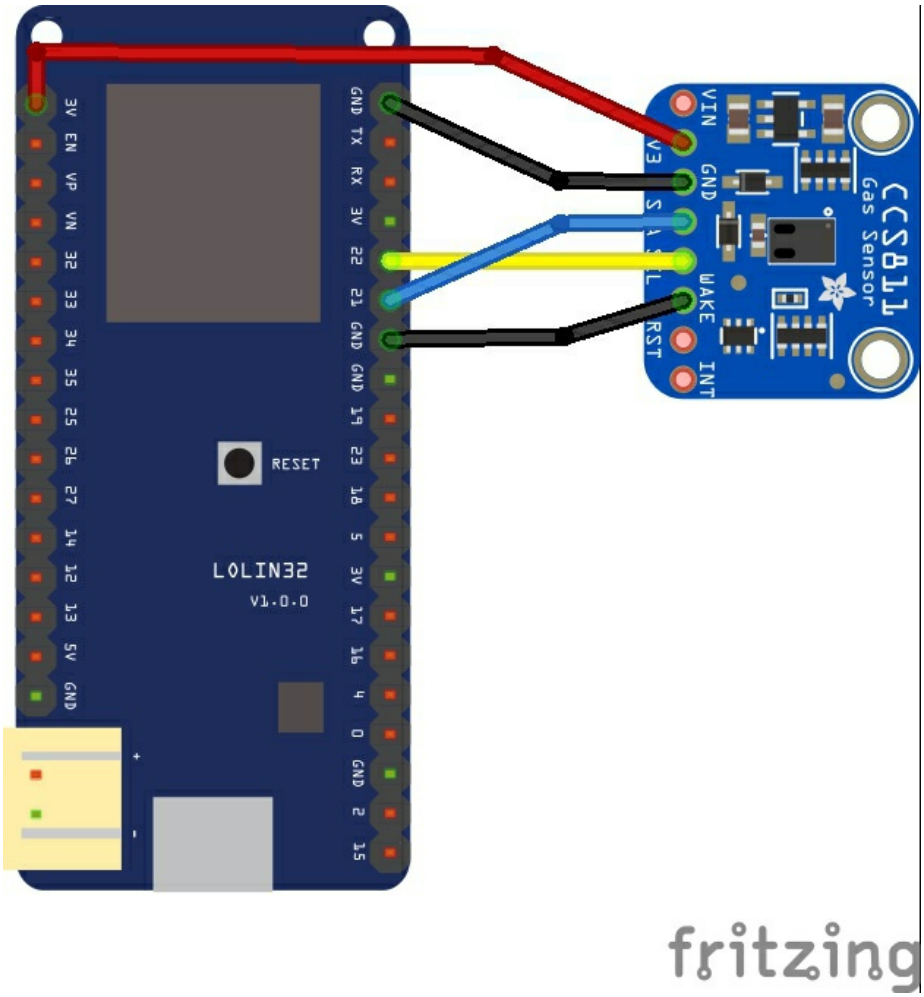
Channel Settings

Percentage complete	50%
Channel ID	629260
Name	CCS811 Example
Description	CCS811 Example to various microcontrollers
Field 1	C02 ☑
Field 2	TVOC ☑
Field 3	Temperature ☑
Field 4	☐
Field 5	☐

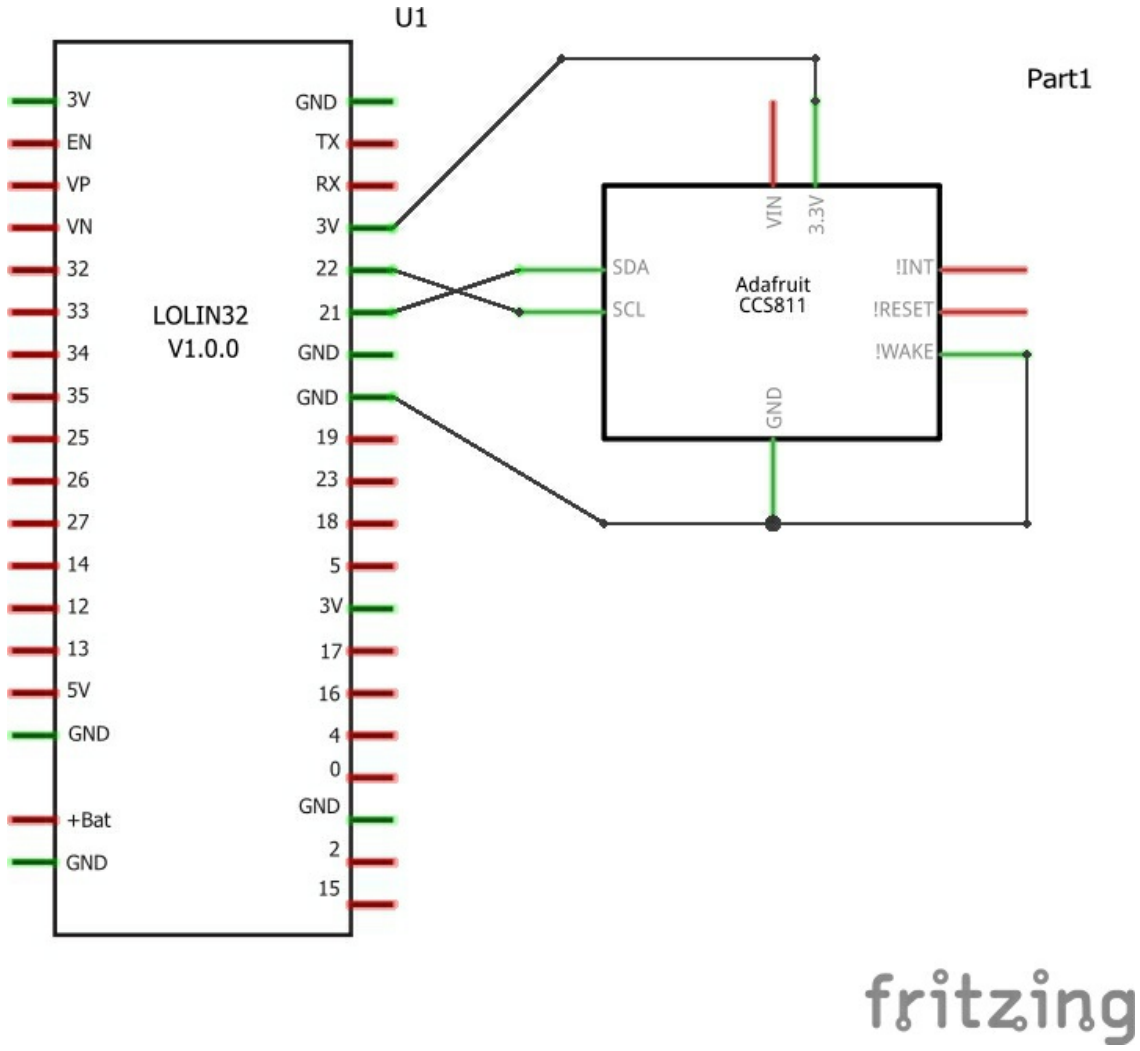
Schematics/Layout

Remember and connect WAKE to gnd

Layout



Schematic


U1
Part1
3V
EN
VP
VN
32
33
34
35
25
26
27
14
12
13
5V
GND
+Bat
GND
GND
TX
RX
3V
22
21
GND
GND
19
23
18
5
3V
17
16
4
0
GND
2
15
LOLIN32
V1.0.0
VIN
3.3V
SDA
SCL
Adafruit
CCS811
!INT
!RESET
!WAKE
GND
fritzing

Code

Again we use a library this is the adafruit CCS811 one - you can use the library manager and add this you will also the Thingspeak libraries for this example https://github.com/mathworks/thingspeak-arduino

```
#include "ThingSpeak.h"
#include <WiFi.h>
#include "Adafruit_CCS811.h"
char ssid[] = "networkssid"; // your network SSID (name)
char pass[] = "networkname"; // your network password
int keyIndex = 0; // your network key Index number (needed only for
WEP)
WiFiClient client;
unsigned long myChannelNumber = 00000;
const char * myWriteAPIKey = "apikey";
Adafruit_CCS811 ccs;
void setup()
{
Serial.begin(115200); //Initialize serial
if(!ccs.begin())
{
Serial.println("Failed to start sensor! Please check your wiring.");
while(1);
}
//calibrate temperature sensor
while(!ccs.available());
float temp = ccs.calculateTemperature();
ccs.setTempOffset(temp - 25.0);
delay(10);
WiFi.mode(WIFI_STA);
ThingSpeak.begin(client); // Initialize ThingSpeak
}
void loop()
{
// Connect or reconnect to WiFi
if(WiFi.status() != WL_CONNECTED)
{
Serial.print("Attempting to connect to SSID: ");
//Serial.println(SECRET_SSID);
while(WiFi.status() != WL_CONNECTED)
{
WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this
line if using open or WEP network
Serial.print(".");
delay(5000);
}
Serial.println("\nConnected.");
}
if(ccs.available())
```

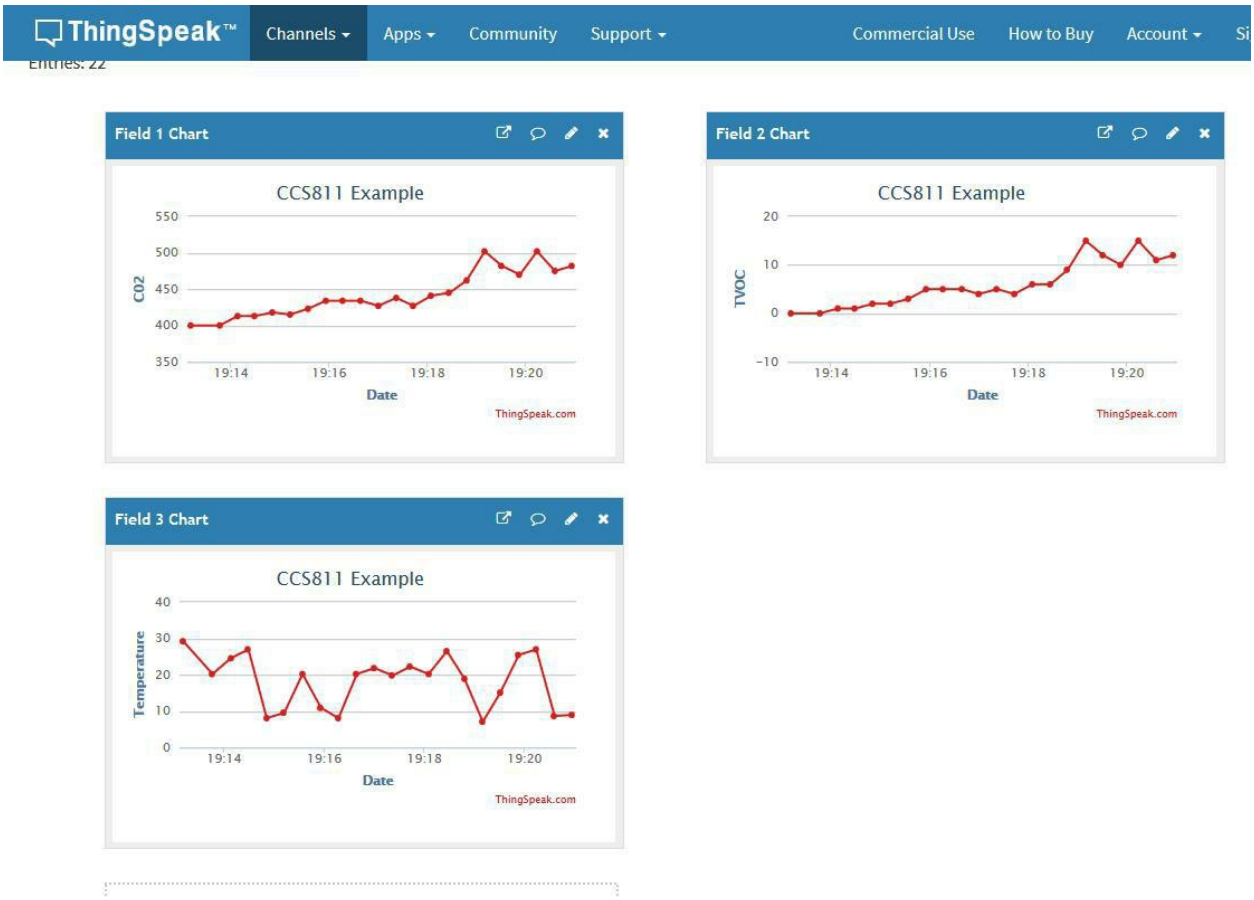
```
{
if(!ccs.readData())
{
float ccsCO2 = ccs.geteCO2();
float ccsTVOC = ccs.getTVOC();
float ccsTemp = ccs.calculateTemperature();
// Write to ThingSpeak. There are up to 8 fields in a channel, allowing
you to store up to 8 different
// pieces of information in a channel.
ThingSpeak.setField(1, ccsCO2);
ThingSpeak.setField(2, ccsTVOC);
ThingSpeak.setField(3, ccsTemp);
// write to the ThingSpeak channel
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if(x == 200)
{
Serial.println("Channel update successful.");
}
else
{
Serial.println("Problem updating channel. HTTP error code " +
String(x));
}
}
else
{
Serial.println("ERROR!");
while(1);
}
}
delay(20000); // Wait 20 seconds to update the channel again
}
```

Output

pen the serial monitor and verify you are connecting and the data has been successfully

Attempting to connect to SSID: .
Connected.
Channel update successful.
Channel update successful.
Channel update successful.
Channel update successful.
Channel update successful.

Lets look at our Thingspeak channel, all going well you should see data like the following channel, all going well you should see data like the following



In Review

We hope you have enjopyed this ebook on the ESP32

We have a variety of code examples at

https://github.com/getelectronics/ESP32-and-arduino-ebook