ESP32-Festival-Finder

Origin:

https://github.com/auryn31/festival_finder

Issues · platformio/platformio-core (github.com)

PlatformIO

Getting Started with PlatformIO - Unified IDE for Embedded Software Development - CIRCUITSTATE Electronics

pio pkg list

```
pio pkg list
```

Resolving esp32dev dependencies...
Platform espressif32 @ 5.0.0 (required: espressif32 @ ~5.0.0)
├── framework-arduinoespressif32 @ 3.20003.220626 (required: platformio/framework-arduinoespressif32 @ ~3.20003.0)
├── tool-cmake @ 3.16.4 (required: platformio/tool-cmake @ ~3.16.0)
├── tool-esptoolpy @ 1.30300.0 (required: platformio/tool-esptoolpy @ ~1.30300.0)
├── tool-idf @ 1.0.1 (required: platformio/tool-idf @ ~1.0.1)
├── tool-mconf @ 1.4060000.20190628 (required: platformio/tool-mconf @ ~1.4060000.0)
├── tool-mkfatfs @ 2.0.1 (required: platformio/tool-mkfatfs @ ~2.0.0)
├── tool-mklittlefs @ 1.203.210628 (required: platformio/tool-mklittlefs @ ~1.203.0)
├── tool-mkspiffs @ 2.230.0 (required: platformio/tool-mkspiffs @ ~2.230.0)
├── tool-ninja @ 1.9.0 (required: platformio/tool-ninja @ ^1.7.0)
├── tool-openocd-esp32 @ 2.1100.20220706 (required: platformio/tool-openocd-esp32 @ ~2.1100.0)
├── toolchain-esp32ulp @ 1.22851.191205 (required: platformio/toolchain-esp32ulp @ ~1.22851.0)
└── toolchain-xtensa-esp32 @ 8.4.0+2021r2-patch3 (required: espressif/toolchain-xtensa-esp32 @ 8.4.0+2021r2-patch3)

Libraries
├── LoRa @ 0.8.0 (required: sandeepmistry/LoRa @ ^0.8.0)
├── QMC5883LCompass @ 1.2.3 (required: mprograms/QMC5883LCompass @ ^1.2.3)
├── TFT_eSPI @ 2.5.43 (required: bodmer/TFT_eSPI @ ^2.5.43)
└── TinyGPSPlus-ESP32 @ 0.0.2 (required: tinyu-zhao/TinyGPSPlus-ESP32 @ ^0.0.2)

Linker Script Generation - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)

Application Level Tracing Library - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)

Error Codes Reference - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)

GCC-Macros

Just get compile date and time with:

const char compile_date[] = __DATE__ " " __TIME__;

then based on that string I get an (almost) unique ID, like checksum. I then read the first byte from EEPROM: if the value is not equal, I assume a newer version has been compiled, so update the EEPROM id value and increase the value of the second EEPROM byte (if you assume you don't have more than 255 versions, otherwise use two bytes).

I can't at the moment find my old code, but it was something like this:

```
const char compile_date[] = __DATE__ " " __TIME__;
...
void setup()
{
  unsigned int ver = 0;
  byte chk = 0;
  for(int i=0; i<strlen(compile_date); ++i)
    chk += compiledate[i];
  if (EEPROM.read(0) != chk) {
     EEPROM.get(1, ver);
     EEPROM.put(1, ++ver);
  }
}
```

Builtin Defines

https://rn-wissen.de/wiki/index.php/Avr-gcc/Interna

GCC

`__GNUC__`	X wenn GCC-Version X.Y.Z
`__GNUC_MINOR__`	Y wenn GCC-Version X.Y.Z
`__GNUC_PATCHLEVEL__`	Z wenn GCC-Version X.Y.Z
`__VERSION__`	"X.Y.Z" wenn GCC-Version X.Y.Z
`__GXX_ABI_VERSION`	Version der ABI (Application Binary Interface)
`__STDC__`	Ist 1, wenn Standard-C übersetzt wird
`__OPTIMIZE__`	Optimierung ist aktiviert
`__NO_INLINE__`	Ohne Schalter `-finline` resp. `-finline-all-functions` etc.
`__ASSEMBLER__`	Definiert, falls GCC die Eingabe als Assembler-Code betrachtet und nicht compiliert. Weiterleitung an den Assembler.
`__cplusplus`	Es wird C++ übersetzt (Quell-Endung `*.cpp`, `*.c++` oder Option `-x c++`).
`__FILE__`	Löst auf zum Dateinamen der Quelldatei, in der das `__FILE__` steht.
`__LINE__`	Löst auf zur Zeilennummer der Quelldatei, in der das `__LINE__` steht.
`__DATE__`	Löst auf zum Datum (precompile-date)
`__TIME__`	Löst auf zur Zeit (precompile-time)

avr-gcc

`__AVR`	Definiert für Target avr, d.h. avr-gcc ist am Werk
`__AVR__`	dito
`__AVR_ARCH__`	`c`odiert den AVR-Kern, für den Code erzeugt wird (Classic, Mega, ...).
`__AVR_XXXX__`	Gesetzt, wenn `-mmcu=xxxx`.

Getting started with ESP32

https://www.espressif.com/en/products/socs/esp32

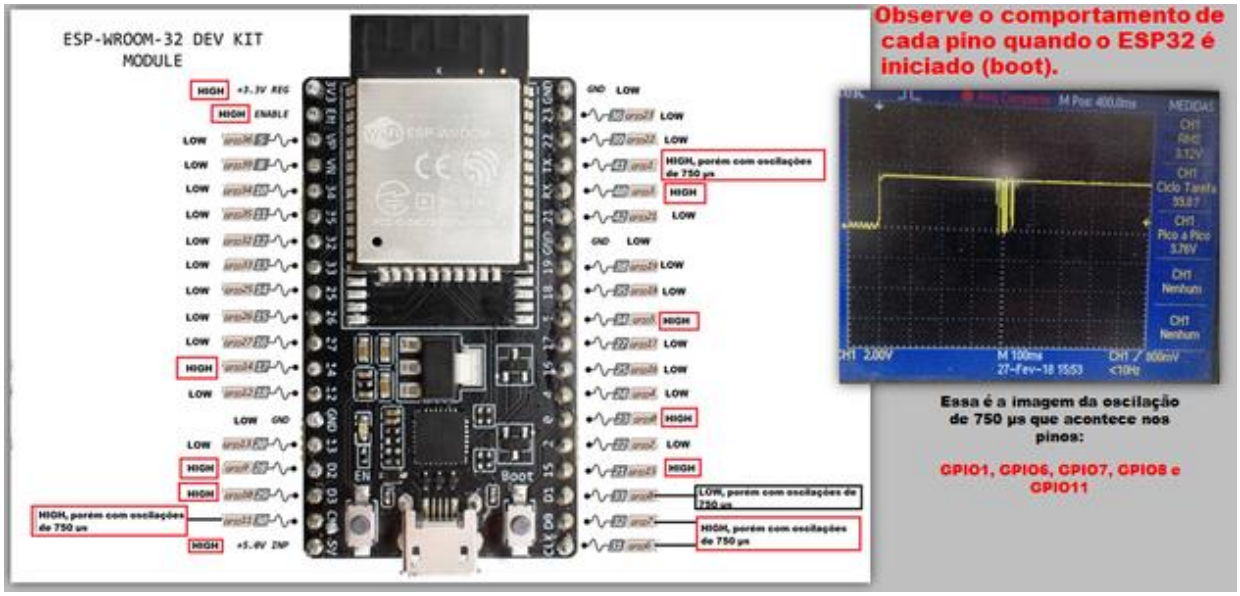
https://dronebotworkshop.com/esp32-intro/

https://randomnerdtutorials.com/esp32-troubleshooting-guide/

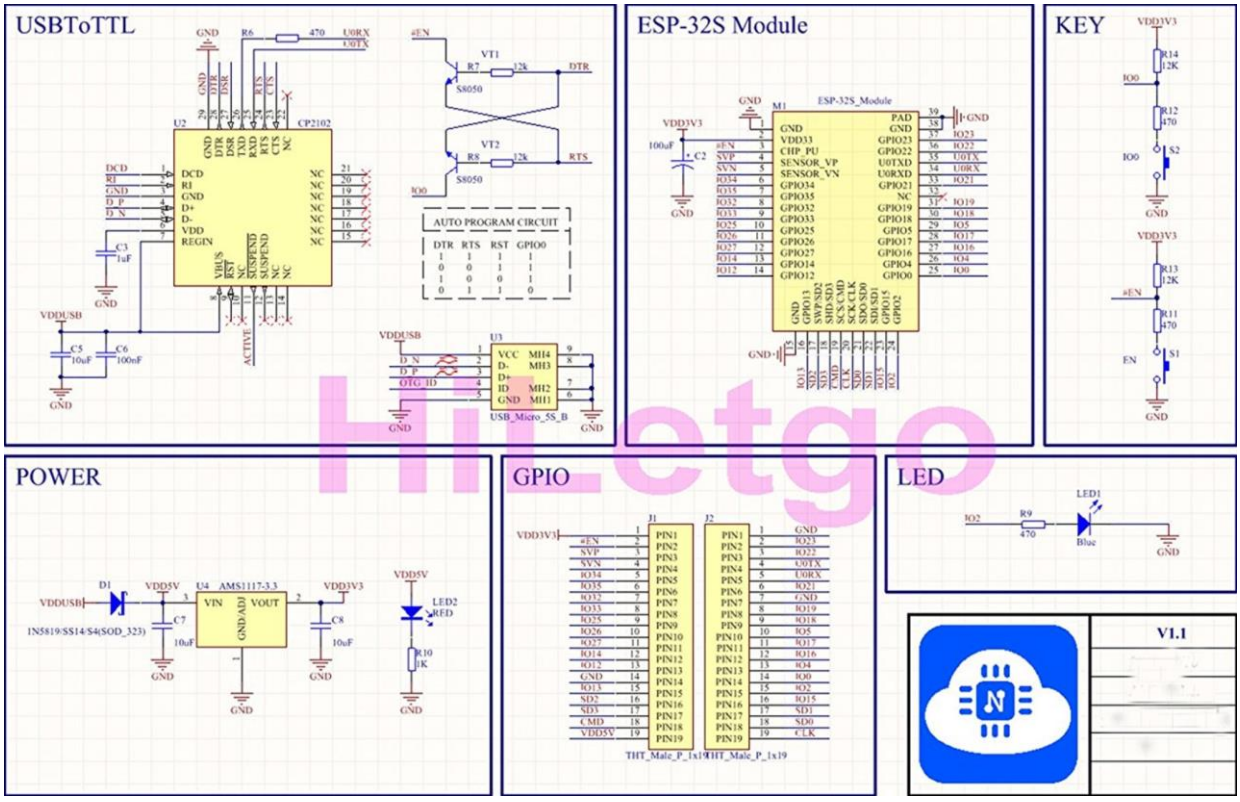
Why to press the BOOT-Button for firmware-upload?

https://randomnerdtutorials.com/solved-failed-to-connect-to-esp32-timed-out-waiting-for-packet-header/

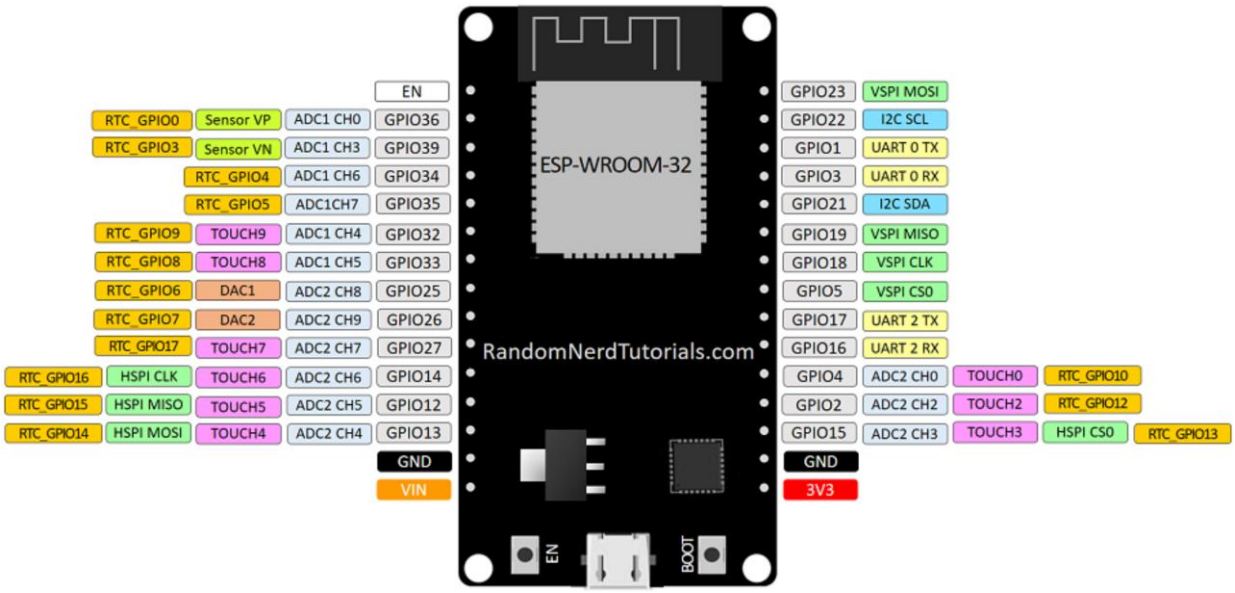
GPIO0 Input	Mode
Low/GND	ROM serial bootloader for esptool.py
High/VCC	Normal execution mode



[ESP32_Schaltplan.jpg (1500×993) (esp32-server.de)]

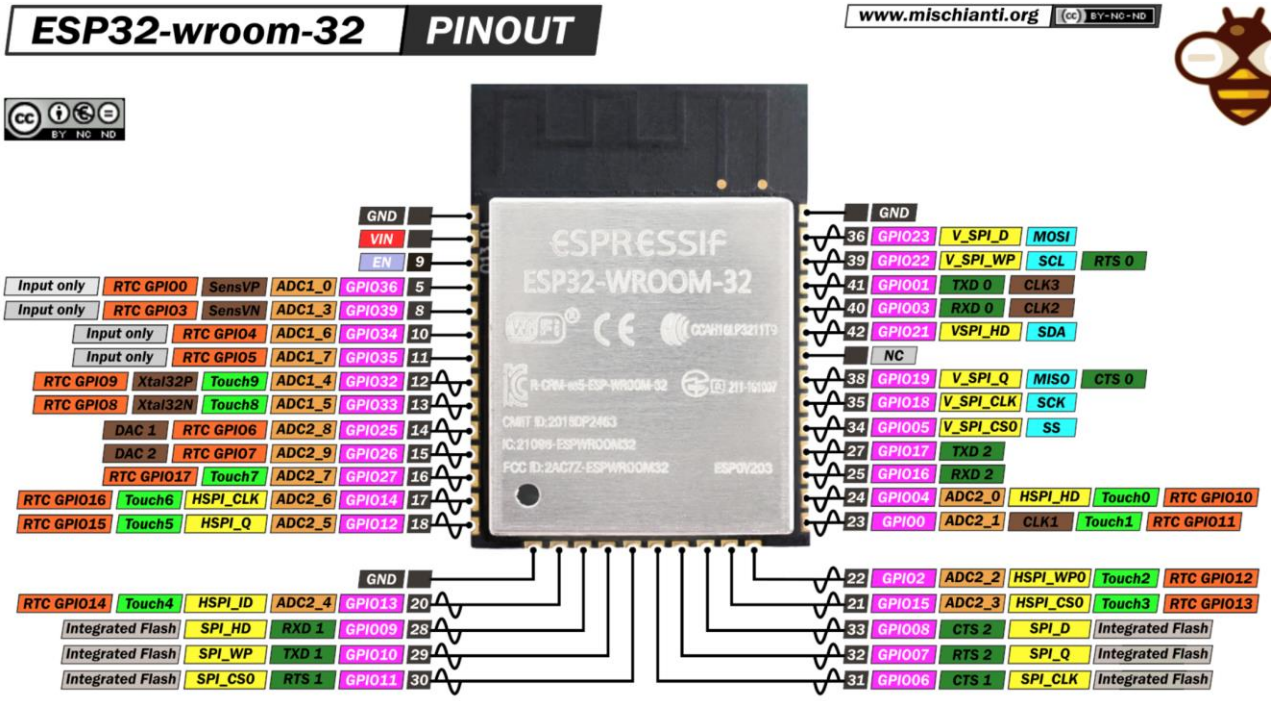


30Pin Dev-Kit-Version



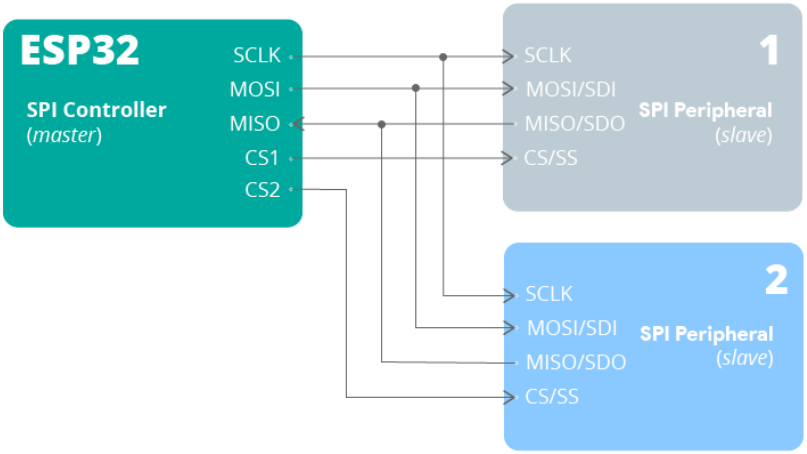
ESP32 SPI Communication: Set Pins, Multiple SPI Bus Interfaces, and Peripherals (Arduino IDE)

https://randomnerdtutorials.com/esp32-spi-communication-arduino/



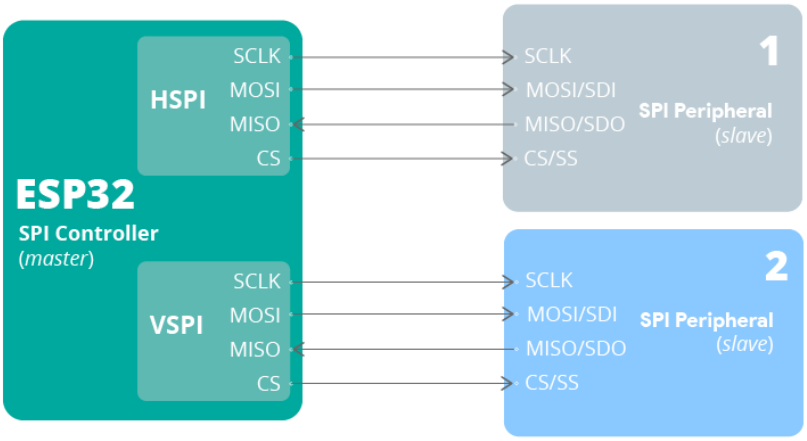
SPI-Variants

Multiple SPI Devices (same bus, different CS pin)



ESP32 Using Two SPI Bus Interfaces (Use HSPI and VSPI simultaneously)

To communicate with multiple SPI peripherals simultaneously, you can use the ESP32 two SPI buses (HSPI and VSPI). You can use the default HSPI and VSPI pins or use custom pins.



Briefly, to use HSPI and VSPI simultaneously, you just need to.

1) First, make sure you include the SPI library in your code.

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

2) Initialize two SPIClass objects with different names, one on the HSPI bus and another on the VSPI bus. For example:

```
vspi = new SPIClass(VSPI);
hspi = new SPIClass(HSPI);
```

3) Call the begin() method on those objects.

```
vspi.begin();
hspi.begin();
```

You can pass custom pins to the begin() method if needed.

```
vspi.begin(VSPI_CLK, VSPI_MISO, VSPI_MOSI, VSPI_SS);
hspi.begin(HSPI_CLK, HSPI_MISO, HSPI_MOSI, HSPI_SS);
```

4) Finally, you also need to set the SS pins as outputs. For example:

```
pinMode(VSPI_SS, OUTPUT);
pinMode(HSPI_SS, OUTPUT);
```

Then, use the usual commands to interact with the SPI devices, whether you're using a sensor library or the SPI library methods.

You can find an example of how to use multiple SPI buses on the arduino-esp32 SPI library.

See the example below:

```
/* The ESP32 has four SPi buses, however as of right now only two of
 * them are available to use, HSPI and VSPI. Simply using the SPI API
 * as illustrated in Arduino examples will use VSPI, leaving HSPI unused.
 *
 * However if we simply intialise two instance of the SPI class for both
 * of these buses both can be used. However when just using these the Arduino
 * way only will actually be outputting at a time.
 *
 * Logic analyser capture is in the same folder as this example as
 * "multiple_bus_output.png"
 *
 * created 30/04/2018 by Alistair Symonds
 */
#include <SPI.h>

// Define ALTERNATE_PINS to use non-standard GPIO pins for SPI bus

#ifdef ALTERNATE_PINS
  #define VSPI_MISO   2
  #define VSPI_MOSI   4
  #define VSPI_SCLK   0
  #define VSPI_SS     33

  #define HSPI_MISO   26
  #define HSPI_MOSI   27
  #define HSPI_SCLK   25
  #define HSPI_SS     32
#else
  #define VSPI_MISO   MISO
  #define VSPI_MOSI   MOSI
  #define VSPI_SCLK   SCK
  #define VSPI_SS     SS

  #define HSPI_MISO   12
  #define HSPI_MOSI   13
  #define HSPI_SCLK   14
  #define HSPI_SS     15
#endif

#if CONFIG_IDF_TARGET_ESP32S2 || CONFIG_IDF_TARGET_ESP32S3
#define VSPI FSPI
#endif

static const int spiClk = 1000000; // 1 MHz

//uninitalised pointers to SPI objects
SPIClass * vspi = NULL;
SPIClass * hspi = NULL;

void setup() {
  //initialise two instances of the SPIClass attached to VSPI and HSPI respectively
  vspi = new SPIClass(VSPI);
  hspi = new SPIClass(HSPI);

  //clock miso mosi ss

#ifndef ALTERNATE_PINS
  //initialise vspi with default pins
  //SCLK = 18, MISO = 19, MOSI = 23, SS = 5
  vspi->begin();
#else
  //alternatively route through GPIO pins of your choice
  vspi->begin(VSPI_SCLK, VSPI_MISO, VSPI_MOSI, VSPI_SS); //SCLK, MISO, MOSI, SS
#endif

#ifndef ALTERNATE_PINS
  //initialise hspi with default pins
  //SCLK = 14, MISO = 12, MOSI = 13, SS = 15
```

```
  hspi->begin();
#else
  //alternatively route through GPIO pins
  hspi->begin(HSPI_SCLK, HSPI_MISO, HSPI_MOSI, HSPI_SS); //SCLK, MISO, MOSI, SS
#endif

  //set up slave select pins as outputs as the Arduino API
  //doesn't handle automatically pulling SS low
  pinMode(vspi->pinSS(), OUTPUT); //VSPI SS
  pinMode(hspi->pinSS(), OUTPUT); //HSPI SS

}

// the loop function runs over and over again until power down or reset
void loop() {
  //use the SPI buses
  spiCommand(vspi, 0b01010101); // junk data to illustrate usage
  spiCommand(hspi, 0b11001100);
  delay(100);
}

void spiCommand(SPIClass *spi, byte data) {
  //use it as you would the regular arduino SPI API
  spi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
  digitalWrite(spi->pinSS(), LOW); //pull SS slow to prep other end for transfer
  spi->transfer(data);
  digitalWrite(spi->pinSS(), HIGH); //pull ss high to signify end of data transfer
  spi->endTransaction();
}
```

^^	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing mode
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED, must be left floating or LOW to enter flashing mode
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot, strapping pin
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fails if pulled high, strapping pin
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot, strapping pin
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Strapping Pins

ESP32 has 6 strapping pins:

• MTDI/GPIO12: internal pull-down
• GPIO0: internal pull-up
• GPIO2: internal pull-down
• GPIO4: internal pull-down
• MTDO/GPIO15: internal pull-up
• GPIO5: internal pull-up

Software can read the value of these 6 bits from the register ”GPIO_STRAPPING”.
During the chip power-on reset, the latches of the strapping pins sample the voltage level as strapping bits of ”0”
or ”1”, and hold these bits until the chip is powered down or shut down. The strapping bits configure the device
boot mode, the operating voltage of VDD_SDIO and other system initial settings.
Each strapping pin is connected with its internal pull-up/pull-down during the chip reset. Consequently, if a strapping
pin is unconnected or the connected external circuit is high-impendence, the internal weak pull-up/pull-down
will determine the default input level of the strapping pins.
To change the strapping bit values, users can apply the external pull-down/pull-up resistances, or apply the host
MCU’s GPIOs to control the voltage level of these pins when powering on ESP32.
After reset, the strapping pins work as the normal functions pins.
Refer to Table 2 for detailed boot modes configuration by strapping pins.

Capacitive touch GPIOs

The ESP32 has 10 internal capacitive touch sensors. These can sense variations in anything that holds an electrical charge, like the human skin. So they can detect variations induced when touching the GPIOs with a finger. These pins can be easily integrated into capacitive pads and replace mechanical buttons. The capacitive touch pins can also be used to wake up the ESP32 from deep sleep.

Those internal touch sensors are connected to these GPIOs:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

```
// ESP32 Touch Test
// Just test touch pin - Touch0 is T0 which is on GPIO 4.

void setup() {
  Serial.begin(115200);
  delay(1000); // give me time to bring up serial monitor
  Serial.println("ESP32 Touch Test");
}

void loop() {
  Serial.println(touchRead(4));  // get value of Touch 0 pin = GPIO 4
  delay(1000);
}
```

ESP32 Capacitive Touch Sensor Pins with Arduino IDE | Random Nerd Tutorials

I2C

The ESP32 has two I2C channels and any pin can be set as SDA or SCL. When using the ESP32 with the Arduino IDE, the default I2C pins are:

- GPIO 21 (SDA)
- GPIO 22 (SCL)

If you want to use other pins when using the wire library, you just need to call:

```
Wire.begin(SDA, SCL);
```

Learn more about I2C communication protocol with the ESP32 using Arduino IDE:
ESP32 I2C Communication (Set Pins, Multiple Bus Interfaces and Peripherals)

SPI

By default, the pin mapping for SPI is:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

Strapping Pins

The ESP32 chip has the following strapping pins:

- GPIO 0 (must be LOW to enter boot mode)
- GPIO 2 (must be floating or LOW during boot)
- GPIO 4
- GPIO 5 (must be HIGH during boot)
- GPIO 12 (must be LOW during boot)
- GPIO 15 (must be HIGH during boot)

Pins HIGH at Boot

Some GPIOs change their state to HIGH or output PWM signals at boot or reset. This means that if you have outputs connected to these GPIOs you may get unexpected results when the ESP32 resets or boots.

- GPIO 1
- GPIO 3
- GPIO 5
- GPIO 6 to GPIO 11 (connected to the ESP32 integrated SPI flash memory – not recommended to use).
- GPIO 14
- GPIO 15

Setting up the TFT_eSPI Library GC9A01

https://github.com/Bodmer/TFT_eSPI

https://www.waveshare.com/wiki/1.28inch_LCD_Module

https://dronebotworkshop.com/gc9a01/#GC9A01_with_ESP32

The **TFT_eSPI** library is ideal for this, and several other, displays. You can install it through your Arduino IDE Library Manager, just search for "TFT_eSPI".

You will need to make a couple of modifications in order to get the library working, as it was meant for several types of displays and processors.

Here is what you need to do:TFT_eSPI

Use your File Manager (Finder on a Mac) to navigate to your Arduino Libraries folder, which is inside your Arduino folder.

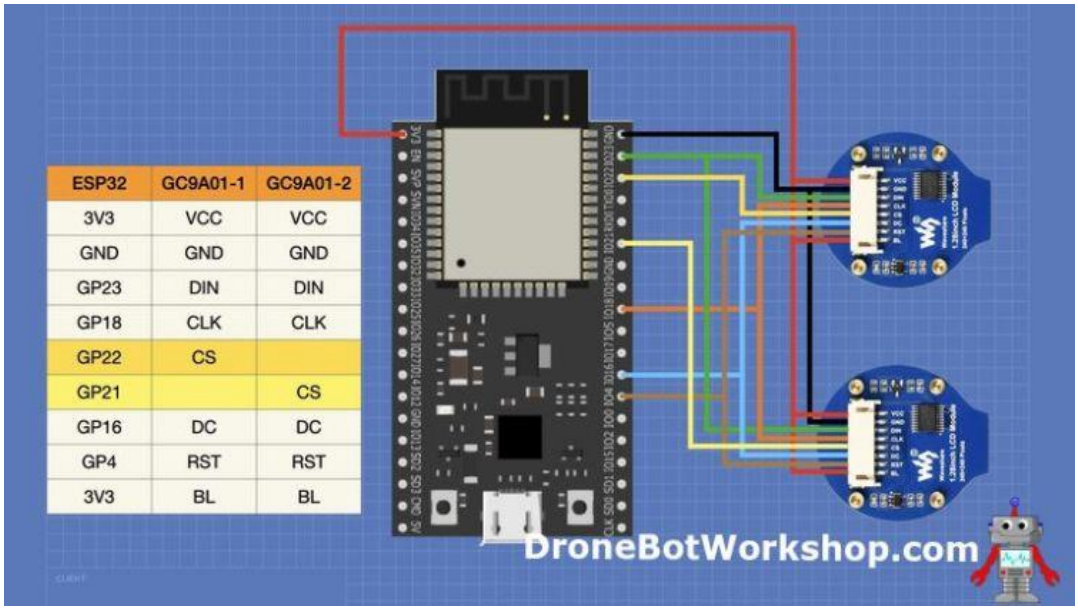
Look for the TFT_eSPI folder. Open it once you find it.

Inside the folder, you'll find several files. Look for the User_Setup.h file.

Open the User_Setup.h file with a text editor.

Once you have opened up the ***User_Setup.h*** file, you will need to make the following edits:

- On line 44 comment out (i.e. add "//" in front of the text) the line defining the ILI9341 driver.
- On line 64 uncomment (i.e. remove the "//") the line defining the GC9A01 driver.
- On lines 204 through 209 comment out all the SPI definitions for the ILI9341.
- On line 215 set MOSI to **23**
- On line 216 set SCLK to **18**
- On line 217 set CS to **22**
- On line 218 set DC to **16**
- On line 219 set RST to **4**



https://github.com/olikraus/u8g2/wiki/setup_tutorial

OLED 1306

u8g2setupcpp · olikraus/u8g2 Wiki (github.com) for u8g2 (Oled)

fntlist12 · olikraus/u8g2 Wiki (github.com)

Home · olikraus/u8g2 Wiki (github.com)

u8g2reference · olikraus/u8g2 Wiki (github.com)

GPS

Flexible GPS Parser:	http://www.technoblogy.com/show?1RB9
Minimal GPS Parser:	http://www.technoblogy.com/show?SJ0
TinyNav Simple GPS Navigator:	http://www.technoblogy.com/show?LNQ
	Technoblogy - Simple Compass Display

GPS routines

At first I tried using Mikal Hart's excellent TinyGPS library to parse the NMEA sentences from the GPS module [1], but the addition of the floating-point package needed for the routines **TinyGPS::distance_between()** and **TinyGPS::course_to()** took the size of my program over the 8192 maximum available on the ATtiny85.

However, the TinyGPS floating-point routines are overkill in this application where we're just showing one of eight compass directions over walking or cycling distances of a few kilometres at most, so I designed much simpler alternatives that use long arithmetic and are much less memory-intensive.

TinyNav is based on two routines to perform the latitude and longitude calculations, and these both use approximations to avoid the need for floating-point arithmetic or trig functions:

DistanceBetween() calculates the distance between two points, specified by their latitude and longitude. It ignores the curvature of the earth, a valid approximation for small distances.

CourseTo() calculates the course from one point to another, assuming the distance between them is small.

The routines are accurate for distances of up to several hundred kilometers. They work with the angular measures, latitude or longitude, in units of 1e-4 arc minutes.

Thus one degree is represented as 600,000 units:

```
const long DEGREE = 600000;
```

This is designed to allow the arithmetic to be done using long integers, and is ideal for parsing the values returned by the GPS module.

Diff and CosFix

The following routines **DistanceBetween()** and **CourseTo()** use a function **Diff()** that calculates the difference between two angular measures:

```
long Diff (long deg1, long deg2) {
  long result = deg2 - deg1;
  if (result > 180 * DEGREE) return result - 360 * DEGREE;
  else if (result < -180 * DEGREE) return result + 360 * DEGREE;
  else return result;
}
```

They also use a fixed-point approximation to cos, **CosFix()**:

```
unsigned int CosFix (long angle) {
  long u = labs(angle)>>16;
  u = (u * u * 6086)>>24;
  return 246 - u;
}
```

This returns a result scaled by 2^8.

DistanceBetween

This routine calculates the distance between two points, in metres, given their latitude and longitude:

```
unsigned int DistanceBetween (long lat1, long long1, long lat2, long long2) {
  long dx = (Diff(long2, long1) * CosFix((lat1 + lat2)/2)) / 256;
  long dy = Diff(lat2, lat1);
  unsigned long adx = labs(dx);
  unsigned long ady = labs(dy);
  unsigned long b = max(adx, ady);
  unsigned long a = min(adx, ady);
  if (b == 0) return 0;
  return 95 * (b + (110 * a / b * a + 128) / 256) / 512;
}
```

It returns the result in metres.

CourseTo

This routine calculates the course from one point to another, in degrees, given their latitude and longitude:

```
unsigned int CourseTo (long lat1, long long1, long lat2, long long2) {
  int c;
  long dx = (Diff(long2, long1) * CosFix((lat1 + lat2)/2)) / 256;
  long dy = Diff(lat2, lat1);
  long adx = labs(dx);
  long ady = labs(dy);
  if (adx == 0) c = 0;
  else if (adx < ady) c = (adx * (45 + (16 * (ady - adx))/ady))/ady;
  else c = 90 - (ady * (45 + (16 * (adx - ady))/adx))/adx;
  //
  if (dx <= 0 && dy < 0) return c;
  else if (dx < 0 && dy >= 0) return 180 - c;
  else if (dx >= 0 && dy >= 0) return 180 + c;
  else return 360 - c;
}
```

It returns the result in degrees, from 0 to 359.

Cardinal

Finally, we need a routine to give the cardinal direction from a direction in degrees:

```
int Cardinal (unsigned int dir) {
  return (((dir*2 + 45) / 90) & 7);
}
```

This returns 0=N, 1=NE, 2=E, 3=SE, 4=S, 5=SW, 6=W, or 7=NW.

Reading the GPS data

The ATtiny85 doesn't provide a USART, so I implemented a simple 9600 baud receive-only UART using the ATtiny85's USI, as described in an earlier article: Simple ATtiny USI UART.

The internal clock is only accurate to within 10%, which is not good enough for use with a UART, so I used an 8MHz crystal clock for the ATtiny85.

When a byte has been received by the USI a USI overflow interrupt is generated, and the interrupt service routine simply calls **ParseGPS()** to process the received character. This is described in my earlier article Minimal GPS Parser [Updated].

Displaying the direction

The correction, or direction between the course and destination, is calculated in the main loop using the following calculation:

```
  WayHome = CourseTo(Lat2, Long2, Lat1, Long1);
  if (WayHome >= Course2) Correction = WayHome-Course2;
    else Correction = WayHome+360-Course2;
```

where Course2 is your current course, Lat2, Long2 is your current position, and Lat1, Long1 is the destination.

The correction is displayed on four blue LEDs, which light up singly for the main directions left, right, forward, and back, or in pairs for the directions in between. Note that the TinyNav doesn't contain a compass, so it can only work out what direction you're pointing in when you're moving, using the course reading from the GPS module. The display when you're stationary is unreliable.

It uses the technique of driving two LEDs from a single output, as described in my earlier post Simple Compass Display. This relies on the fact that blue or white LEDs typically have a forward voltage of 3.4V, so connecting two in series with 220Ω resistors will result in no current flowing, with both LEDs off, if powered from 3.7V. Note that you must use blue or white LEDs; red LEDs will be dimly illuminated all the time. I used bright blue LEDs from Adafruit [2], available from The Pi Hut in the UK [3]. These are so bright that I increased the resistors to 1kΩ.

Here's the routine to display the compass bearing:

```
void DisplayLEDs (int i) {
    pinMode(LedsEW, INPUT);  pinMode(LedsNS, INPUT);
    if (i < 0) return;
    if (i%4 != 0) {digitalWrite(LedsEW,(i/4 == 1)); pinMode(LedsEW,OUTPUT);}
    if ((i+2)%4 != 0) {digitalWrite(LedsNS,((i+2)/4 == 1));
pinMode(LedsNS,OUTPUT);}
}
```

It takes the cardinal direction returned by **Cardinal()**, or -1 turns all the LEDs off.

Calculating the distance

The distance to the destination is calculated in the main loop using the following calculation:

```
  Distance = min(DistanceBetween(Lat2, Long2, Lat1, Long1), 1000);
```
This is then used to generate a gap of the corresponding number of milliseconds between flashes, up to a maximum gap of one second.

Storing your destination

Since there are no spare I/O pins, I used the reset pin as an analogue input to detect the pushbutton used to store your current destination. This is connected to the reset pin via a resistor divider to ensure that the voltage is never taken low enough to reset the chip, as described in the section *Using the reset pin on the ATtiny85* in my post Getting Extra Pins on ATtiny.

When you press the Store button all four LEDs are illuminated as confirmation that the location has been stored. The location is stored in EEPROM, so it won't be lost if you switch off TinyNav.

The following routines are used to write and read the latitude and longitude:

```
void EEPROMWritelong(int address, long value) {
  EEPROM.write(address, value & 0xFF);
  EEPROM.write(address+1, value>>8 & 0xFF);
  EEPROM.write(address+2, value>>16 & 0xFF);
  EEPROM.write(address+3, value>>24 & 0xFF);
}

long EEPROMReadlong(long address) {
  return EEPROM.read(address) | (long)EEPROM.read(address+1)<<8 |
    (long)EEPROM.read(address+2)<<16 | (long)EEPROM.read(address+3)<<24;
}
```

Alternatively, for a treasure hunt, you could look up the destination on Google Maps, and then program the values into the program by setting the values of Lat1 and Long1 in **setup()**. For example, to make your destination the Statue of Liberty:

Calculating the values in minutes*1000, the latitude (N-S position) is:

40.6892 x 60 x 10000 = 24413520 (north is positive).

and the longitude (E-W position) is:

-74.0444 x 60 x 10000 = -44426640 (west is negative).

so you would enter:

```
  Lat1 = 24413520;
```

```
  Long1 = -44426640;
```

Note that I haven't actually been able to test these!

TinyGPS Sample

```
#include<Arduino.h>
#include <TinyGPSPlus.h>
//#include <SoftwareSerial.h>
/*
   This sample sketch demonstrates the normal use of a TinyGPSPlus (TinyGPSPlus)
   object. It requires the use of SoftwareSerial, and assumes that you have a
   4800-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
*/
static const int RXPin = 17, TXPin = 16;
static const uint32_t GPSBaud = 9600;

// The TinyGPSPlus object
TinyGPSPlus gps;

// The serial connection to the GPS device
//SoftwareSerial ss(RXPin, TXPin);


void setup()
{
  Serial.begin(115200);
  Serial1.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPSPlus with an attached GPS
module"));
  Serial.print(F("Testing TinyGPSPlus library v. "));
Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
}

void loop()
{
    //--- for debugging
    //gps.updateSerial();

  // This sketch displays information every time a new sentence is correctly encoded.
  while (Serial1.available() > 0)
    if (gps.encode(Serial1.read()))
      displayInfo();


  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
}

void displayInfo()
{

  Serial.print(F("Location: "));

  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(","));
    Serial.print(gps.location.lng(), 6);
  }
  else
```

```
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F(" "));
  if (gps.time.isValid())
  {
    if (gps.time.hour() < 10) Serial.print(F("0"));
    Serial.print(gps.time.hour());
    Serial.print(F(":"));
    if (gps.time.minute() < 10) Serial.print(F("0"));
    Serial.print(gps.time.minute());
    Serial.print(F(":"));
    if (gps.time.second() < 10) Serial.print(F("0"));
    Serial.print(gps.time.second());
    Serial.print(F("."));
    if (gps.time.centisecond() < 10) Serial.print(F("0"));
    Serial.print(gps.time.centisecond());
  }
  else
  {
    Serial.print(F("INVALID"));
  }
  Serial.println();
}
```

I am amazed at how much you can do with such a small MCU. Some very clever stuff here. I have some comments on the code though:

1) In Diff(), if (result > 108000000) you should return result - 216000000 instead of 216000000 - result, but it only matters when you cross the 180° meridian.

2) In DistanceBetween() and CourseTo(), you should multiply dx by the cosine of the latitude. For your purpose, the approximation $\cos(\pi x/2) \approx 1 - x^2$ may be good enough (max error ≈ 5.6%). Dropping the cosine altogether carries an error close to 30% at mid latitudes.

3) In DistanceBetween(), you are calling sqrt(), which pulls lots of expensive floating point routines into your code. There are cheaper alternatives amenable to fixed-point implementations. C.f. http://stackoverflow.com/a/...
4) In CourseTo(), you use the approximation $\text{atan}(x) \approx 45x$ (in degrees). A far better, and only marginally more expensive approximation is $\text{atan}(x) \approx x(45+16(1-x))$.

Lora

Semtech SX1276, SX1277, SX1278, SX1279 Datasheets:

https://github.com/orgs/Lora-net/repositories?type=all

Arduino-Lib: https://github.com/sandeepmistry/arduino-LoRa/issues/647

Setup LoRa-Frequency:

```
if (!LoRa.begin(433700000)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
```

DIO0... DIO5

It is quite complex, and depends on many factors. First, the uses of DIOØ to 5 depends on whether the chip is in Continuous Mode or in Packet Mode. Then, it depends on the state the chip is in: Sleep, Standby, FSRxTx, Rx, Tx. And finally, it depends on which setting it is on: each pin has 4 states ØØ to 11. This is detailed in the Semtech SX1276, SX1277, SX1278, SX1279 Datasheet. Here are a couple of tables.

https://github.com/sandeepmistry/arduino-LoRa/issues/580

PlatformIO TFT_eSPI+SD slot. Problem duplicate func

[E][esp32-hal-cpu.c:110] addApbChangeCallback(): duplicate func=0x400d8098 arg=0x3ffbdc60

https://github.com/Bodmer/TFT_eSPI/issues/1509

https://github.com/greiman/SdFat/issues/462

Fundamentals on LoRaWAN LoRaWAN® | The Things Network

Name (Address)	Bits	Variable Name	Mode	Default value	FSK/OOK Description
RegDioMapping1 (0x40)	7-6	Dio0Mapping	rw	0x00	Mapping of pins DIO0 to DIO5 See Table 18 for mapping in LoRa mode See Table 29 for mapping in Continuous mode SeeTable 30 for mapping in Packet mode
	5-4	Dio1Mapping	rw	0x00	
	3-2	Dio2Mapping	rw	0x00	
	1-0	Dio3Mapping	rw	0x00	
RegDioMapping2 (0x41)	7-6	Dio4Mapping	rw	0x00	
	5-4	Dio5Mapping	rw	0x00	
	3-1	reserved	rw	0x00	reserved. Retain default value
	0	MapPreambleDetect	rw	0x00	Allows the mapping of either *Rssi* Or *PreambleDetect* to the DIO pins, as summarized on Table 29 and Table 30 0 → *Rssi* interrupt 1 → *PreambleDetect* interrupt

Table 30 DIO Mapping, Packet Mode

	DIOx Mapping	Sleep	Standby	FSRx/Tx	Rx	Tx
DIO0	00	-			PayloadReady	PacketSent
	01	-			CrcOk	-
	10	-				
	11	-	TempChange / LowBat		TempChange / LowBat	
DIO1	00	FifoLevel		FifoLevel	FifoLevel	
	01	FifoEmpty		FifoEmpty	FifoEmpty	
	10	FifoFull		FifoFull	FifoFull	
	11	-				
DIO2	00	FifoFull		FifoFull	FifoFull	
	01	-			RxReady	-
	10	FifoFull			TimeOut	FifoFull
	11	FifoFull			SyncAddress	FifoFull
DIO3	00	FifoEmpty		FifoEmpty	FifoEmpty	
	01	-				TxReady
	10	FifoEmpty		FifoEmpty	FifoEmpty	
	11	FifoEmpty		FifoEmpty	FifoEmpty	
DIO4	00	-	TempChange / LowBat		TempChange / LowBat	
	01	-		PllLock		
	10	-			TimeOut	-
	11	-			Rssi / PreambleDetect	-
DIO5	00	ClkOut if RC	ClkOut		ClkOut	
	01	-		PllLock		
	10	-			Data	
	11	-	ModeReady		ModeReady	

Table 29 DIO Mapping, Continuous Mode

	DIOx Mapping	Sleep	Standby	FSRx/Tx	Rx	Tx
DIO0	00	-			SyncAddress	TxReady
	01	-			Rssi / PreambleDetect	-
	10	-			RxReady	TxReady
	11	-				
DIO1	00	-			Dclk	
	01	-			Rssi / PreambleDetect	-
	10	-				
	11	-				
DIO2	00	-			Data	
	01	-			Data	
	10	-			Data	
	11	-			Data	
DIO3	00	-			Timeout	-
	01	-			Rssi / PreambleDetect	-
	10	-				
	11	-	TempChange / LowBat		TempChange / LowBat	
DIO4	00	-		TempChange / LowBat		
	01	-		PllLock		
	10	-			TimeOut	-
	11	-	ModeReady		ModeReady	
DIO5	00	ClkOut if RC	ClkOut		ClkOut	
	01	-		PllLock		
	10	-			Rssi / PreambleDetect	-
	11	-	ModeReady		ModeReady	

ESPTool

```
#!C:\Users\js\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.8
_qbz5n2kfra8p0\python.exe
# EASY-INSTALL-ENTRY-SCRIPT: 'esptool==2.8','console_scripts','esptool.py'
__requires__ = 'esptool==2.8'
import re
import sys
from pkg_resources import load_entry_point

if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw?|\.exe)?$', '', sys.argv[0])
    sys.exit(
        load_entry_point('esptool==2.8', 'console_scripts', 'esptool.py')()
    )
# esptool.py
```

A Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

[![Test esptool](https://github.com/espressif/esptool/actions/workflows/test_esptool.yml/badge.svg?branch=master)](https://github.com/espressif/esptool/actions/workflows/test_esptool.yml) [![Build esptool](https://github.com/espressif/esptool/actions/workflows/build_esptool.yml/badge.svg?branch=master)](https://github.com/espressif/esptool/actions/workflows/build_esptool.yml)

Documentation
Visit the [documentation](https://docs.espressif.com/projects/esptool/) or run `esptool.py -h`.

esptool.py -h

C:\Users\js\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\Scripts\esptool.py-script.py:6: DeprecationWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html

from pkg_resources import load_entry_point

usage: esptool [-h] [--chip {auto,esp8266,esp32}] [--port PORT] [--baud BAUD] [--before {default_reset,no_reset,no_reset_no_sync}] [--after {hard_reset,soft_reset,no_reset}] [--no-stub] [--trace] [--override-vddsdio [{1.8V,1.9V,OFF}]]

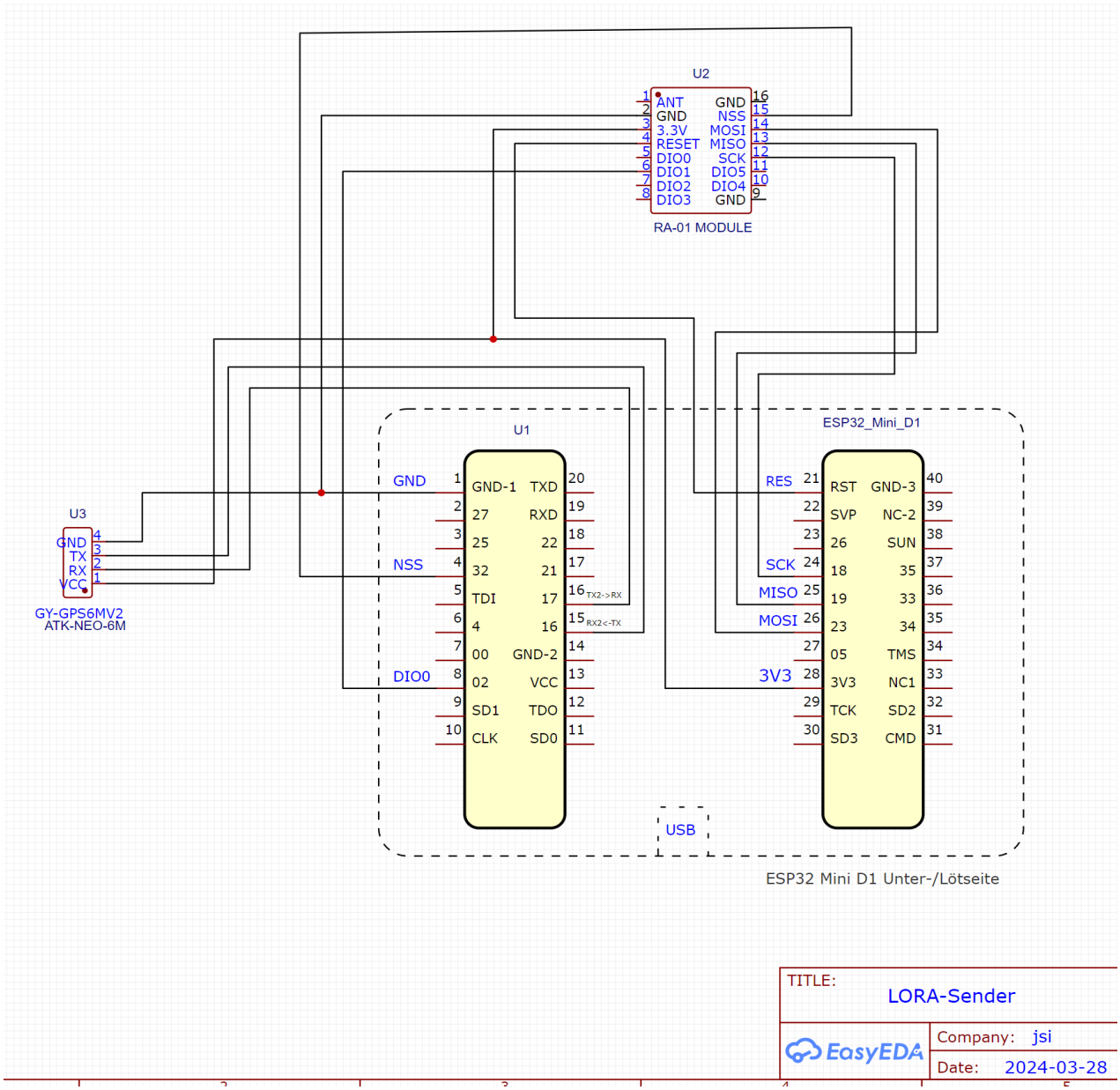
{load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read_mac,chip_id,flash_id,read_flash_status,write_flash_status,read_flash,verify_flash,erase_flash,erase_region,version} ...

```
esptool.py v2.8 - ESP8266 ROM Bootloader Utility

positional arguments:

{load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image
```

```
,read_mac,chip_id,flash_id,read_flash_status,write_flash_status,read_flash,verify_fla
sh,erase_flash,erase_region,version}

                        Run esptool {command} -h for additional help

    load_ram            Download an image to RAM and execute

    dump_mem            Dump arbitrary memory to disk

    read_mem            Read arbitrary memory location

    write_mem           Read-modify-write to arbitrary memory location

    write_flash         Write a binary blob to flash

    run                 Run application code in flash

    image_info          Dump headers from an application image

    make_image          Create an application image from binary files

    elf2image           Create an application image from ELF file

    read_mac            Read MAC address from OTP ROM

    chip_id             Read Chip ID from OTP ROM

    flash_id            Read SPI flash manufacturer and device ID

    read_flash_status   Read SPI flash status register

    write_flash_status  Write SPI flash status register

    read_flash          Read SPI flash content

    verify_flash        Verify a binary blob against flash

    erase_flash         Perform Chip Erase on SPI flash

    erase_region        Erase a region of the flash

    version             Print esptool version


optional arguments:

  -h, --help            show this help message and exit

  --chip {auto,esp8266,esp32}, -c {auto,esp8266,esp32}

                        Target chip type

  --port PORT, -p PORT  Serial port device

  --baud BAUD, -b BAUD  Serial port baud rate used when flashing/reading

  --before {default_reset,no_reset,no_reset_no_sync}

                        What to do before connecting to the chip

  --after {hard_reset,soft_reset,no_reset}, -a {hard_reset,soft_reset,no_reset}

                        What to do after esptool.py is finished

  --no-stub             Disable launching the flasher stub, only talk to ROM
bootloader. Some features will not be available.

  --trace, -t           Enable trace-level output of esptool.py interactions.

  --override-vddsdio [{1.8V,1.9V,OFF}]

                        Override ESP32 VDDSDIO internal voltage regulator (use with
care)
```

```
??? "C:\Users\js\.platformio\packages\tool-esptool\esptool.exe"
```

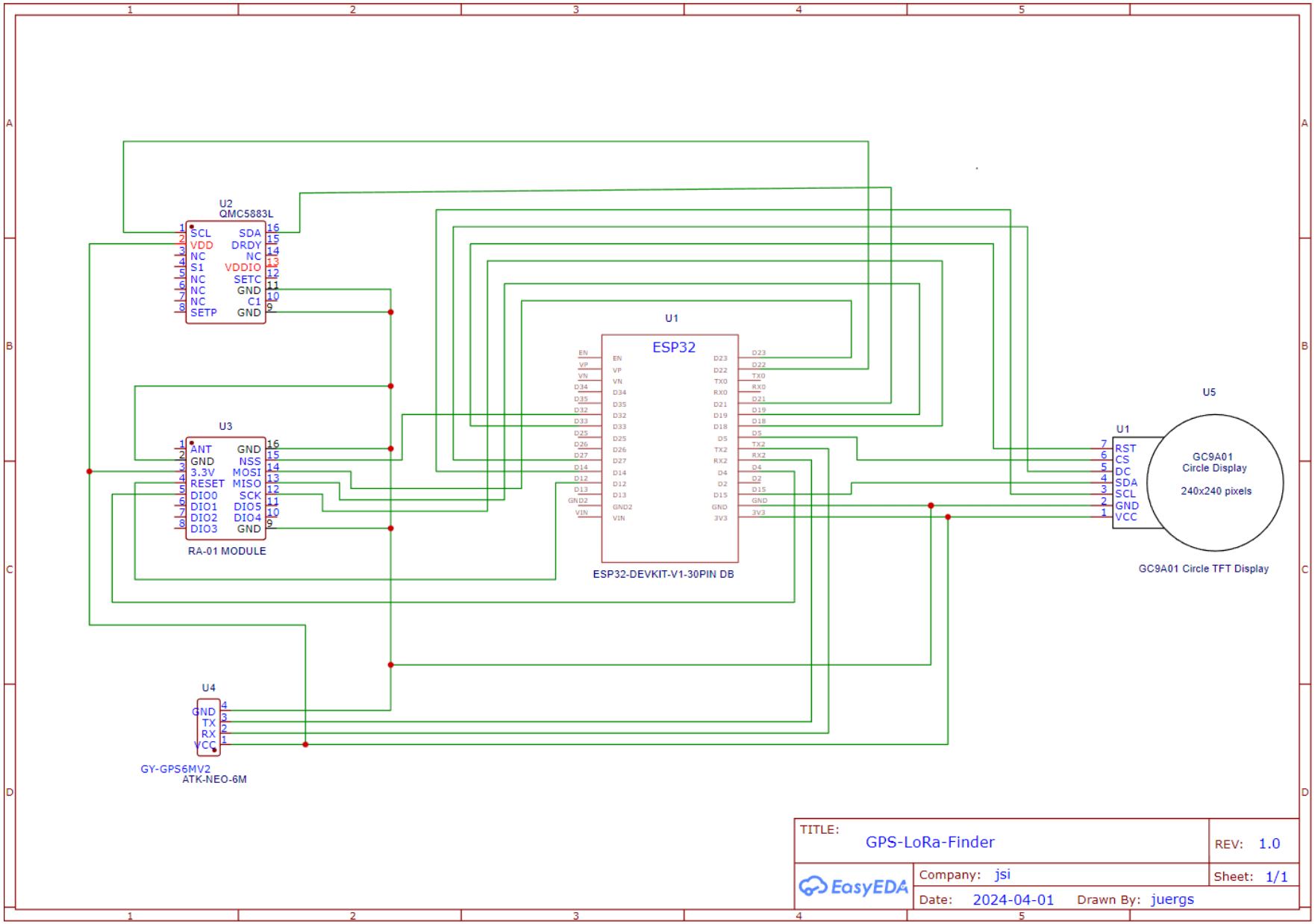
Lora-Sender-OLED



Using Wemos ESP32 Mini D1

DIO1 seems not to be necessary at all:

[LoRa stack and DIOs - End Devices (Nodes) - The Things Network]


U2
QMC5883L
SCL
VDD
NC
S1
NC
NC
NC
SETP
SDA
DRDY
NC
VDDIO
SETC
GND
C1
GND
U3
ANT
GND
3.3V
RESET
DIO0
DIO1
DIO2
DIO3
GND
NSS
MOSI
MISO
SCK
DIO5
DIO4
GND
RA-01 MODULE
U1
ESP32
ESP32-DEVKIT-V1-30PIN DB
U5
U1
RST
CS
DC
SDA
SCL
GND
VCC
GC9A01
Circle Display
240x240 pixels
GC9A01 Circle TFT Display
U4
GND
TX
RX
VCC
GY-GPS6MV2
ATK-NEO-6M
TITLE:
GPS-LoRa-Finder
REV: 1.0
EasyEDA
Company: jsi
Sheet: 1/1
Date: 2024-04-01
Drawn By: juergs

ESP32 – Debugging

Debugging NodeMCU Firmware over JTAG

May 30, 2017 esp8266, jtag

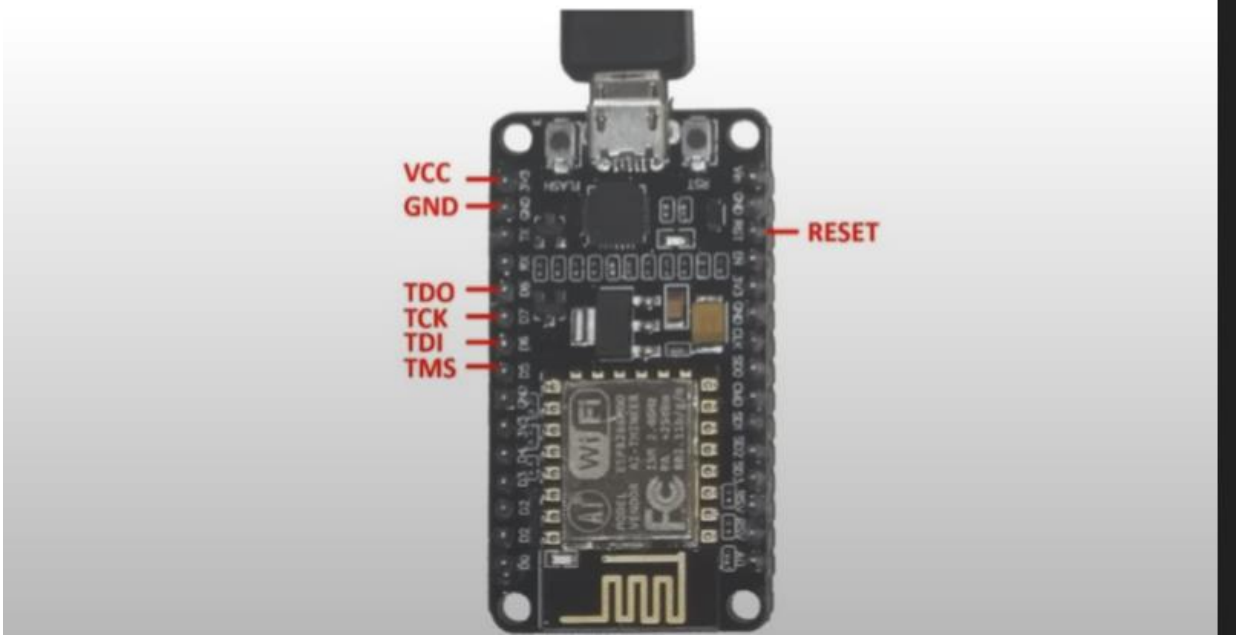
This tutorial shows how to debug ESP8266 firmware running on your NodeMCU board using JTAG. We will show how to setup the necessary connections and configure the software to automatically program the FLASH memory and debug the firmware.

In this tutorial we will use the Olimex ARM-USB-OCD-H JTAG debugger. Other JTAG debuggers may work as well, however they may need slightly different configuration.

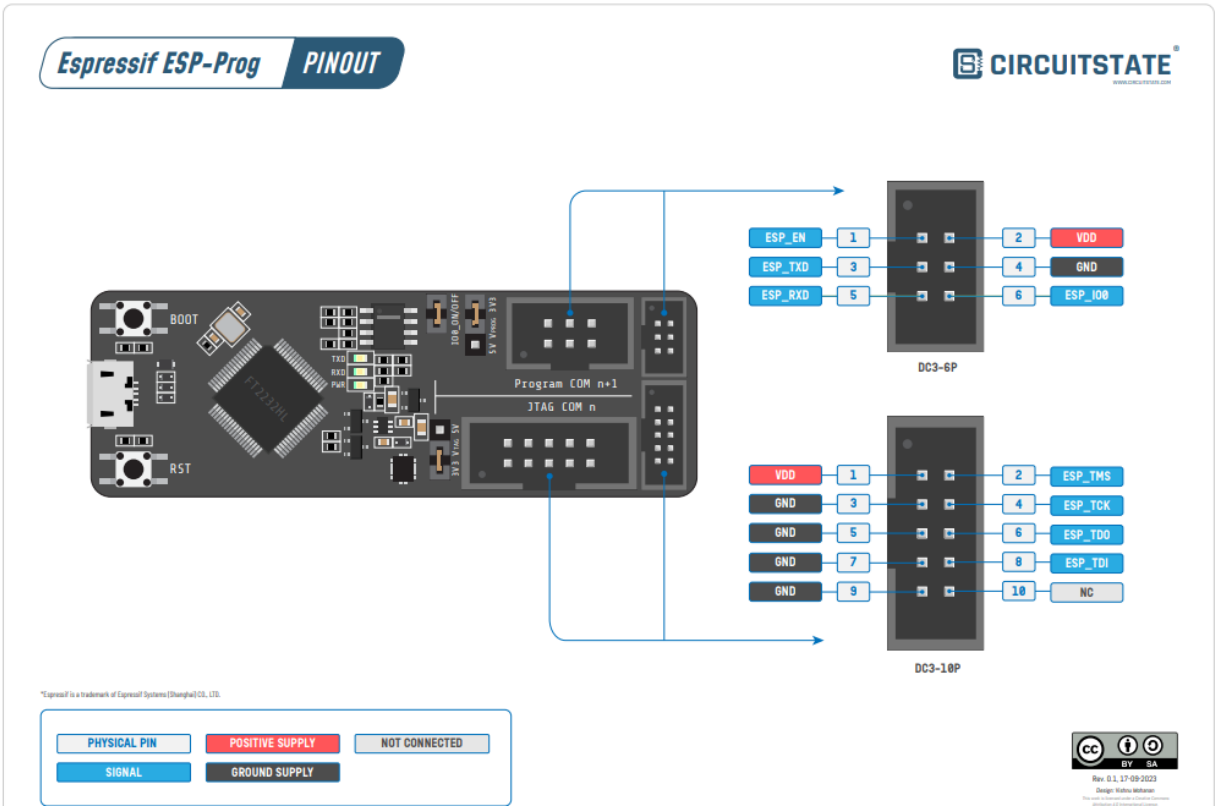
Before you begin, install VisualGDB 5.2 or later.

1. The first step will be to connect the JTAG pins of the ESP8266 chip on the NodeMCU to your JTAG debugger. The relevant pins are VCC, GND, TDI, TDO, TMS, TCK and RESET. For NodeMCU v1, they should be connected as follows (see NodeMCU schematic and ESP-12 schematic):

ESP8266 Signal	ESP8266 Pin	NodeMCU Signal	NodeMCU Pin	JTAG20 Signal	JTAG20 Pin
VDDPST	17	VDD3V	1 on J2	VDD	1
VDD	PAD	GND	2 on J2	GND	4
MTDI	10	GPIO12	7 on J2	TDI	5
MTMS	9	GPIO14	8 on J2	TMS	7
MTCK	12	GPIO13	6 on J2	TCK	9
MTDO	13	GPIO15	5 on J2	TDO	13
EXT_RSTB	32	RST	3 on J1	nTRST or nSRST	3 (Olimex) or 15 (Segger)



Espressif ESP-Prog ESP32 JTAG Debug Probe - Pinout Diagram - CIRCUITSTATE Electronics



Debugging — PlatformIO latest documentation

Tutorials

- Arduino In-circuit Debugging with PlatformIO
- Use the PlatformIO Debugger on the ESP32 Using an ESP-prog
- Get started with Arduino and ESP32-DevKitC: debugging and unit testing
- Get started with ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis
- Arduino and Nordic nRF52-DK: debugging and unit testing
- Zephyr and Nordic nRF52-DK: debugging, unit testing, project analysis
- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

Configuration

PlatformIO Debugging Solution can be configured using "platformio.ini" (Project Configuration File):

- Build Configurations
- Debugging options
    - `debug_tool`
    - `debug_build_flags`
    - `debug_init_break`
    - `debug_init_cmds`
    - `debug_extra_cmds`
    - `debug_load_cmds`
    - `debug_load_mode`
    - `debug_server`
    - `debug_port`
    - `debug_speed`
    - `debug_svd_path`
    - `debug_server_ready_pattern`
    - `debug_test`

CLI Guides

- pio debug
    - Usage
    - Description
    - Options
    - Examples

Espressif Platform

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESP32 Pico Kit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32S3 CAM LCD	Espressif 32	External	ESP32S3	240MHz	8MB	320KB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
Espressif ESP32 Dev Module	Espressif 32	External	ESP32	240MHz	4MB	320KB
Espressif ESP32-C3-DevKitC-02	Espressif 32	External	ESP32C3	160MHz	4MB	320KB
Espressif ESP32-C3-DevKitM-1	Espressif 32	External	ESP32C3	160MHz	4MB	320KB
Espressif ESP32-C6-DevKitM-1	Espressif 32	External	ESP32C6	160MHz	4MB	320KB
Espressif ESP32-S2-Kaluga-1 Kit	Espressif 32	On-board	ESP32S2	240MHz	4MB	320KB
Espressif ESP32-S2-Saola-1	Espressif 32	External	ESP32S2	240MHz	4MB	320KB
Espressif ESP32-S3-Box	Espressif 32	External	ESP32S3	240MHz	16MB	320KB
Espressif ESP32-S3-DevKitC-1-N8 (8 MB QD, No PSRAM)	Espressif 32	On-board	ESP32S3	240MHz	8MB	320KB
Espressif ESP32-S3-DevKitM-1	Espressif 32	On-board	ESP32S3	240MHz	8MB	320KB

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Espressif ESP32-S3-USB-OTG	Espressif 32	On-board	ESP32S3	240MHz	8MB	320KB