Customising Ringo firmware

Setting it up

Introduction

Hello! Welcome to this little tutorial on how to modify your own Ringo and how to create your own versions of the firmware!

In this tutorial, you won't learn how to actually program, but how to set up the whole firmware for editing and compiling after you made your edits.

If you already have some experience in coding and compiling in different IDEs, you shouldn't have any problems. However, if you're still familiar with the language and never compiled your own firmware, don't worry, we'll guide you through this tutorial.

On the other hand, if you don't have any C/C++ experience, it's best for you not to go on with this tutorial and try to first learn the language.
All good? Okay, let's start!

IDE

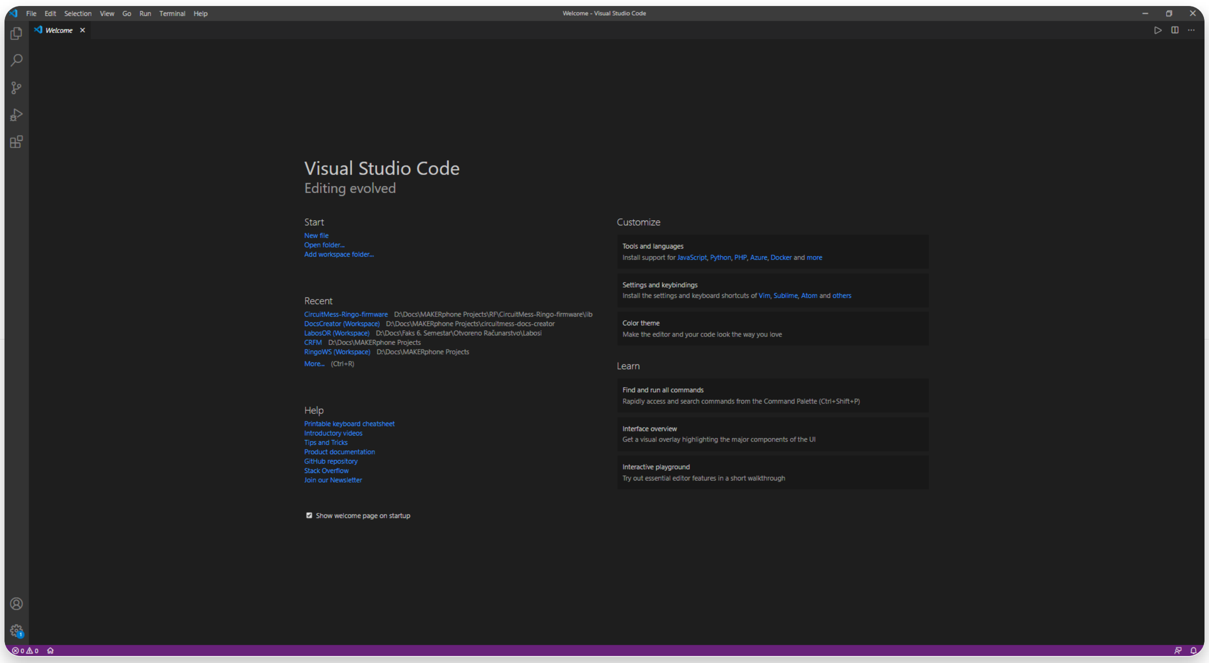
The IDE or integrated development environment is a piece of software that is used to write, edit, modify, and compile your code in order to run it either via your computer or some other device.
In this tutorial, we're going to use **Visual Studio Code** or simply **VS Code**, a very popular IDE by Microsoft, which is used by developers at CircuitMess.

It is really light and easy to use, whilst offering wide support of additional plugins that can be used if necessary. It is a "younger brother" of another very popular IDE, Microsoft Visual Studio.

VS Code logo

You can go to the **VS Code official page** to download and install the software. There are versions available for Windows, Linux, and Mac OS, and the installation itself is pretty straightforward.

Once installed, you should get something like this.


File Edit Selection View Go Run Terminal Help
Welcome - Visual Studio Code
Welcome

Visual Studio Code
Editing evolved

Start
New file
Open folder...
Add workspace folder...

Recent
CircuitMess-Ringo-firmware D:\Docs\MAKERphone Projects\RF\CircuitMess-Ringo-firmware\lib
DocsCreator (Workspace) D:\Docs\MAKERphone Projects\circuitmess-docs-creator
LabosOR (Workspace) D:\Docs\Faks 6. Semestar\Otvoreno Računarstvo\Labosi
CRFM D:\Docs\MAKERphone Projects
RingoWS (Workspace) D:\Docs\MAKERphone Projects
More... (Ctrl+R)

Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter

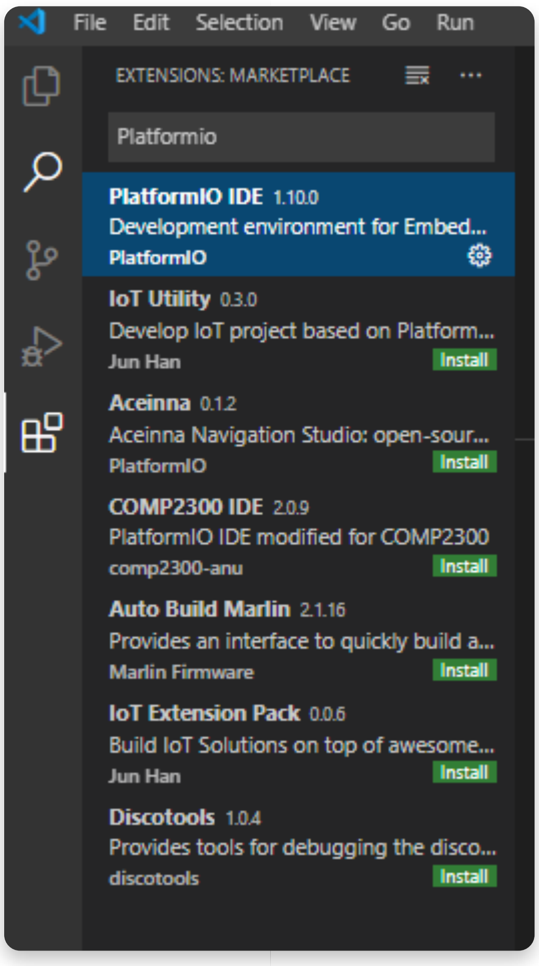
Show welcome page on startup

Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others
Color theme
Make the editor and your code look the way you love

Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try out essential editor features in a short walkthrough

VS Code main screen

Before starting the development, there are a few plugins you need to install.



PlatformIO IDE 1.10.0
Development environment for Embed...
PlatformIO

IoT Utility 0.3.0
Develop IoT project based on Platform...
Jun Han Install

Aceinna 0.1.2
Aceinna Navigation Studio: open-sour...
PlatformIO Install

COMP2300 IDE 2.0.9
PlatformIO IDE modified for COMP2300
comp2300-anu Install

Auto Build Marlin 2.1.16
Provides an interface to quickly build a...
Marlin Firmware Install

IoT Extension Pack 0.0.6
Build IoT Solutions on top of awesome...
Jun Han Install

Discotools 1.0.4
Provides tools for debugging the disco...
discotools Install

Click on the **Extensions** icon and type in **'platformio'**. Select the first result **PlatformIO IDE**.

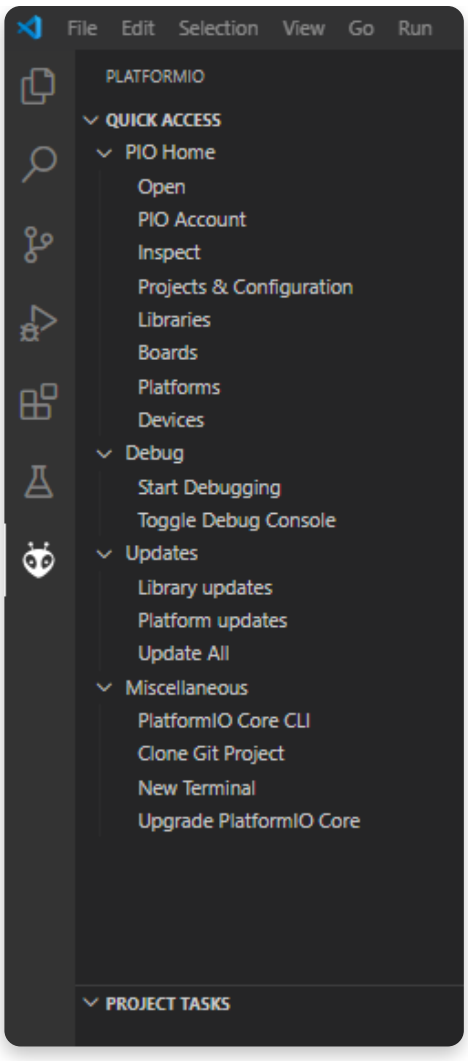
This one is used to connect your firmware to the specific hardware used by your device.

The installation should be quick and the files itself shouldn't take up too much space.

You can see a change in your VS Code.

Once that's done you'll get a new icon on the left-hand sidebar.

That should mean that the installation has been completed.



PlatformIO menu

Now that this part is done, our IDE is ready.

We can move on to downloading the code and creating the project in which we'll modify the firmware.

Setting up the project

First, let's create a new PlatformIO project.

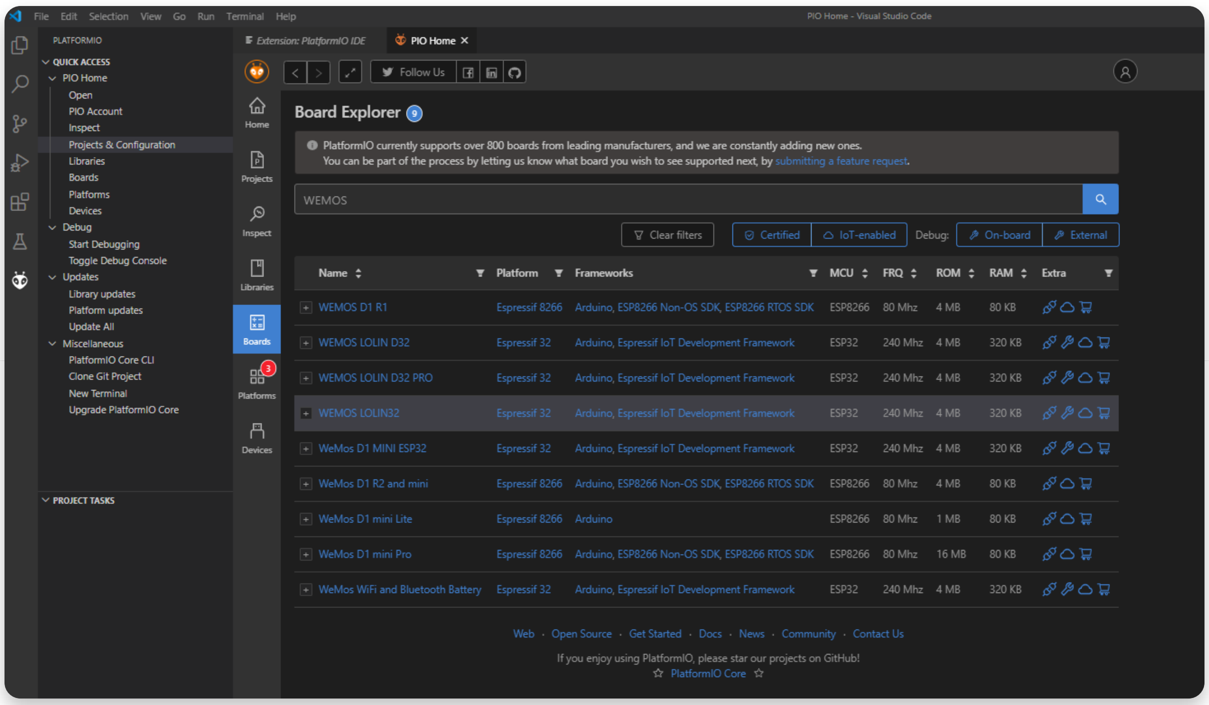
It's a good practice to create a new folder somewhere on your disk and to put everything you're going to use in this project inside it.

That will definitely keep you organized and help you with the search for your files.

It would also be good to check if the board files that we need are installed properly.

Click on the **Boards** icon and search for **WEMOS LOLIN32**. Since the processor inside Ringo is **ESP32** by **Espressif**, we're going to need to use one of the boards that have the same pinout as our board, which is this one. You can also use some of the other boards as well and they should work fine too, although that's not the case for all of them.

If you find it in the list, you're good to go. However, if it's not there, you need to re-install the whole PlatformIO plugin.



Board Explorer 9

PlatformIO currently supports over 800 boards from leading manufacturers, and we are constantly adding new ones.
You can be part of the process by letting us know what board you wish to see supported next, by submitting a feature request.

WEMOS

Clear filters | Certified | IoT-enabled | Debug: On-board | External

Name	Platform	Frameworks	MCU	FRQ	ROM	RAM
WEMOS D1 R1	Espressif 8266	Arduino, ESP8266 Non-OS SDK, ESP8266 RTOS SDK	ESP8266	80 Mhz	4 MB	80 KB
WEMOS LOLIN D32	Espressif 32	Arduino, Espressif IoT Development Framework	ESP32	240 Mhz	4 MB	320 KB
WEMOS LOLIN D32 PRO	Espressif 32	Arduino, Espressif IoT Development Framework	ESP32	240 Mhz	4 MB	320 KB
WEMOS LOLIN32	Espressif 32	Arduino, Espressif IoT Development Framework	ESP32	240 Mhz	4 MB	320 KB
WeMos D1 MINI ESP32	Espressif 32	Arduino, Espressif IoT Development Framework	ESP32	240 Mhz	4 MB	320 KB
WeMos D1 R2 and mini	Espressif 8266	Arduino, ESP8266 Non-OS SDK, ESP8266 RTOS SDK	ESP8266	80 Mhz	4 MB	80 KB
WeMos D1 mini Lite	Espressif 8266	Arduino	ESP8266	80 Mhz	1 MB	80 KB
WeMos D1 mini Pro	Espressif 8266	Arduino, ESP8266 Non-OS SDK, ESP8266 RTOS SDK	ESP8266	80 Mhz	16 MB	80 KB
WeMos WiFi and Bluetooth Battery	Espressif 32	Arduino, Espressif IoT Development Framework	ESP32	240 Mhz	4 MB	320 KB

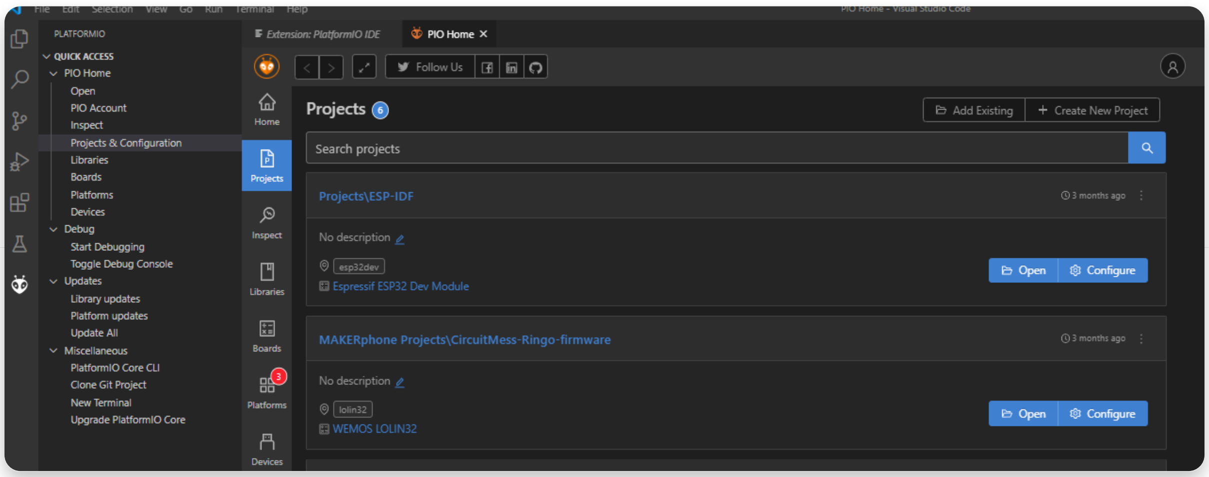
Web · Open Source · Get Started · Docs · News · Community · Contact Us

If you enjoy using PlatformIO, please star our projects on GitHub!
PlatformIO Core

List of supported ESP32 boards is pretty big

Now, let's create a new project.

Click on the Project tab and select '**Create New Project**'.

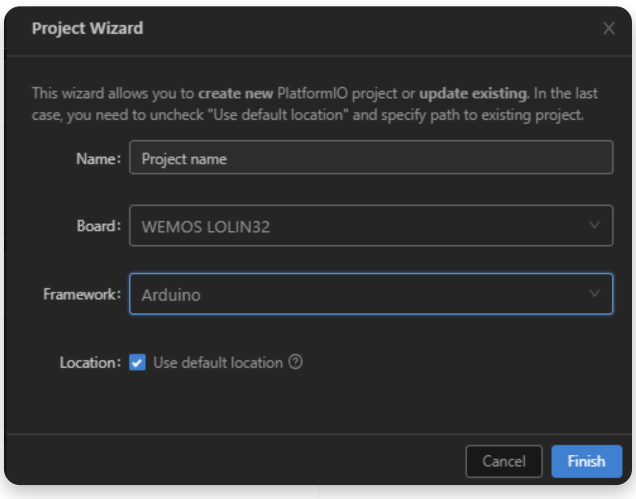


Projects 6

Projects\ESP-IDF — 3 months ago
No description
esp32dev
Espressif ESP32 Dev Module

MAKERphone Projects\CircuitMess-Ringo-firmware — 3 months ago
No description
lolin32
WEMOS LOLIN32

Creating a new project



Project Wizard

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name: Project name

Board: WEMOS LOLIN32

Framework: Arduino

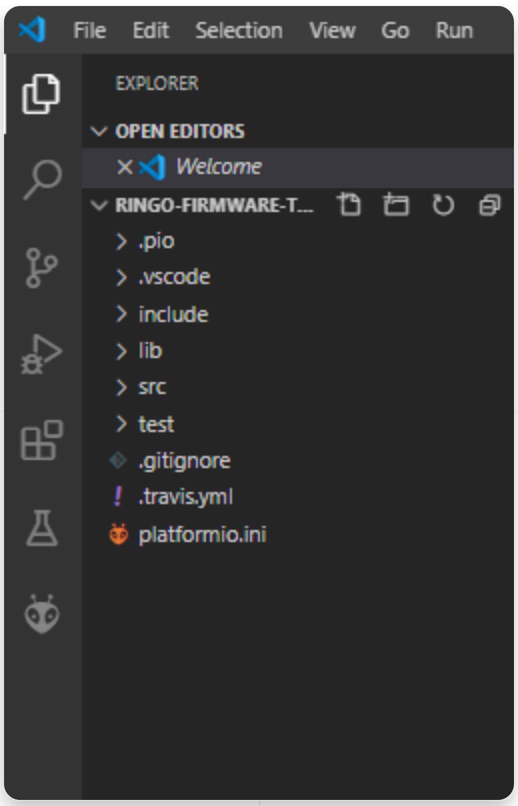
Location: [✓] Use default location

Cancel | Finish

A new screen should pop-up.

Here you can set the project name, location, framework, and most importantly, select the board.

Select the already mentioned **WEMOS LOLIN32**.



The project is now created and your folder will contain some additional folders and files.

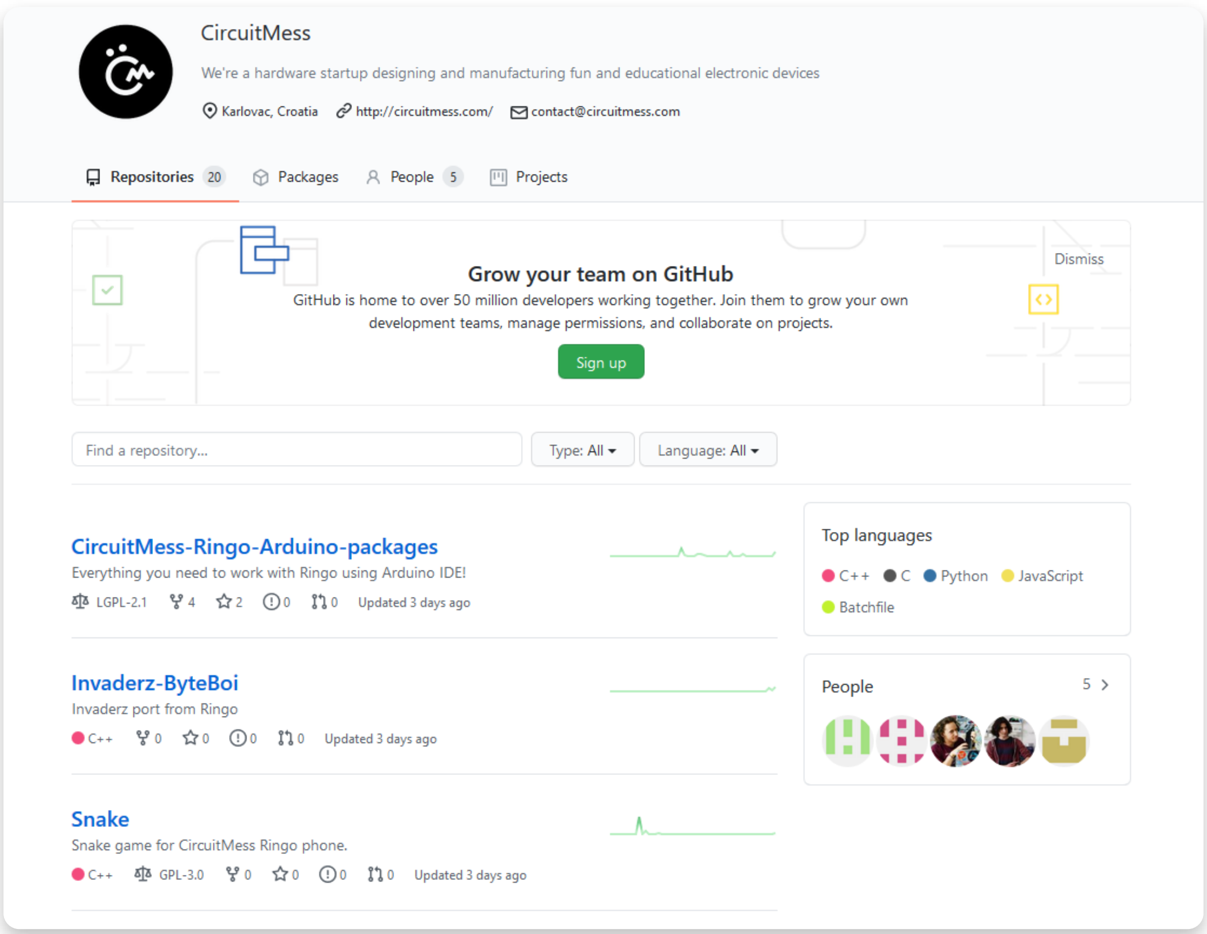
We'll explain what does this all means in a bit.

Now that the project has been set up, it's important to download the files we're going to work with.

For that, we're heading to the **GitHub**.

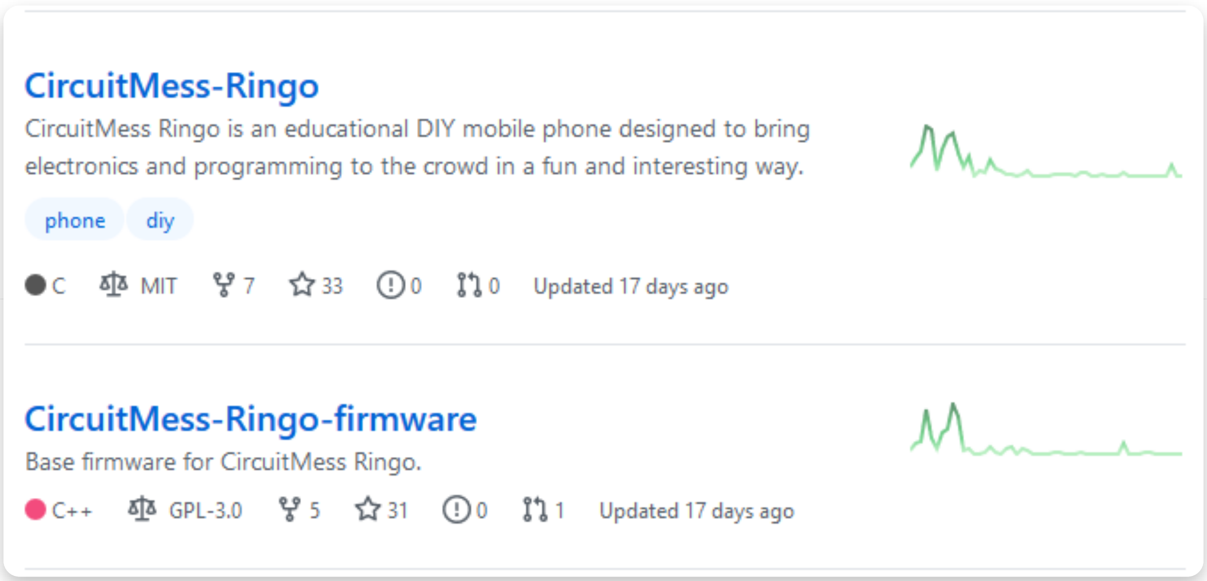
Downloading the source files

First things first - open a new tab on this page - https://github.com/CircuitMess. You're going to need it a lot while modifying your firmware since there are a lot of materials to learn from.


CircuitMess
We're a hardware startup designing and manufacturing fun and educational electronic devices
Karlovac, Croatia
http://circuitmess.com/
contact@circuitmess.com
Repositories 20
Packages
People 5
Projects
Grow your team on GitHub
GitHub is home to over 50 million developers working together. Join them to grow your own development teams, manage permissions, and collaborate on projects.
Sign up
Dismiss
Find a repository...
Type: All
Language: All
CircuitMess-Ringo-Arduino-packages
Everything you need to work with Ringo using Arduino IDE!
LGPL-2.1 4 2 0 0 Updated 3 days ago
Invaderz-ByteBoi
Invaderz port from Ringo
C++ 0 0 0 0 Updated 3 days ago
Snake
Snake game for CircuitMess Ringo phone.
C++ GPL-3.0 0 0 0 0 Updated 3 days ago
Top languages
C++ C Python JavaScript Batchfile
People 5

CircuitMess GitHub repository - loads of cool stuff!

For start, we'll be looking at these two repositories - **CircuitMess-Ringo** and **CircuitMess-Ringo-firmware**.



CircuitMess-Ringo

CircuitMess Ringo is an educational DIY mobile phone designed to bring electronics and programming to the crowd in a fun and interesting way.

phone diy

C MIT 7 33 0 0 Updated 17 days ago

CircuitMess-Ringo-firmware

Base firmware for CircuitMess Ringo.

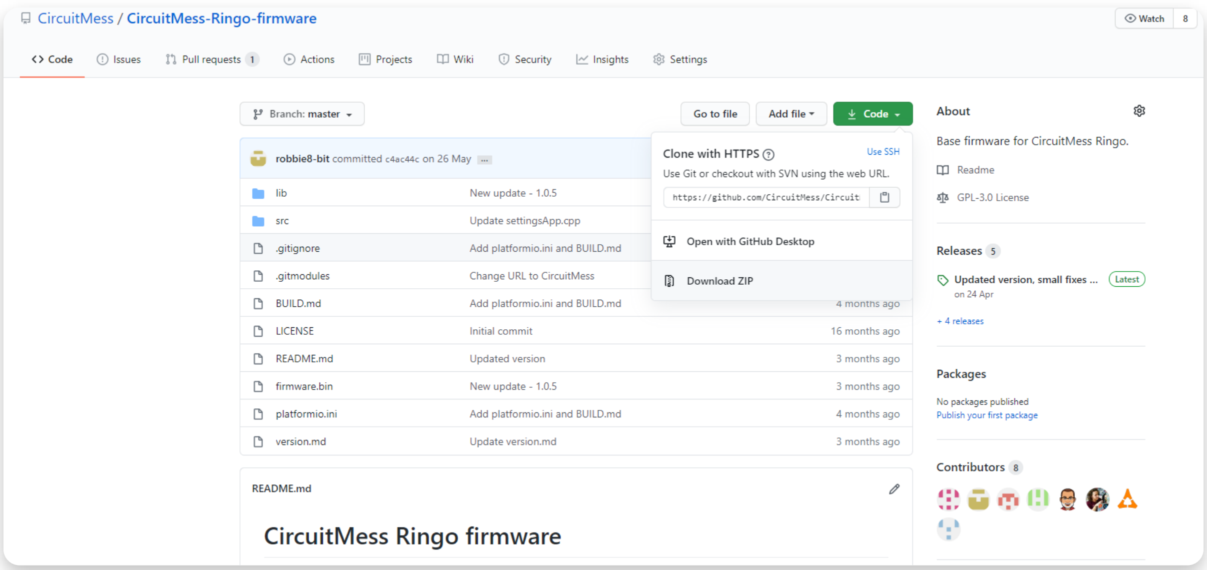
C++ GPL-3.0 5 31 0 1 Updated 17 days ago

Two main repositories we're going to use

First, you're going to open CircuitMess-Ringo-firmware.

It is where pretty much all of the firmware files are located and you'll be editing those.

When you open the repository, it looks something like this.



robbie8-bit committed c4ac44c on 26 May

lib	New update - 1.0.5	
src	Update settingsApp.cpp	
.gitignore	Add platformio.ini and BUILD.md	
.gitmodules	Change URL to CircuitMess	
BUILD.md	Add platformio.ini and BUILD.md	4 months ago
LICENSE	Initial commit	16 months ago
README.md	Updated version	3 months ago
firmware.bin	New update - 1.0.5	3 months ago
platformio.ini	Add platformio.ini and BUILD.md	4 months ago
version.md	Update version.md	3 months ago

Clone with HTTPS

Use SSH

Use Git or checkout with SVN using the web URL.

`https://github.com/CircuitMess/Circuit`

Open with GitHub Desktop

Download ZIP

README.md

CircuitMess Ringo firmware

About

Base firmware for CircuitMess Ringo.

Readme

GPL-3.0 License

Releases 5

Updated version, small fixes ... Latest
on 24 Apr

+ 4 releases

Packages

No packages published
Publish your first package

Contributors 8

CircuitMess Ringo firmware repository

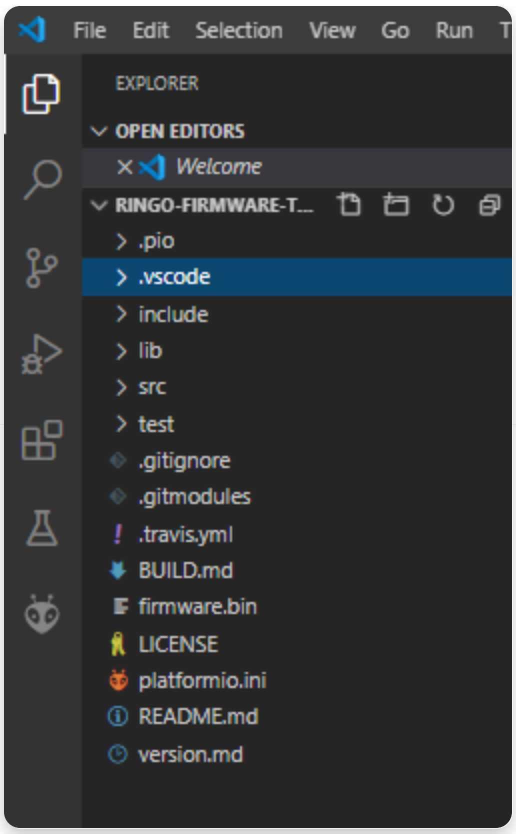
Download the ZIP containing all of the files. You can also use GitHub Desktop if you're familiar with that piece of software.

GitHub Dekstop basically allows you to manipulate with GitHub repositories much easier and faster, but we'll not be covering it in this tutorial.

Once you've downloaded the ZIP, unpack all of the files into the project folder.

Replace all of the files that are matching.

Your project folder should look something like this.



.pio
.vscode
include
lib
src
test
.gitignore
.gitmodules
.travis.yml
BUILD.md
firmware.bin
LICENSE
platformio.ini
README.md
version.md

VS Code project folder after the insertion of Ringo firmware files

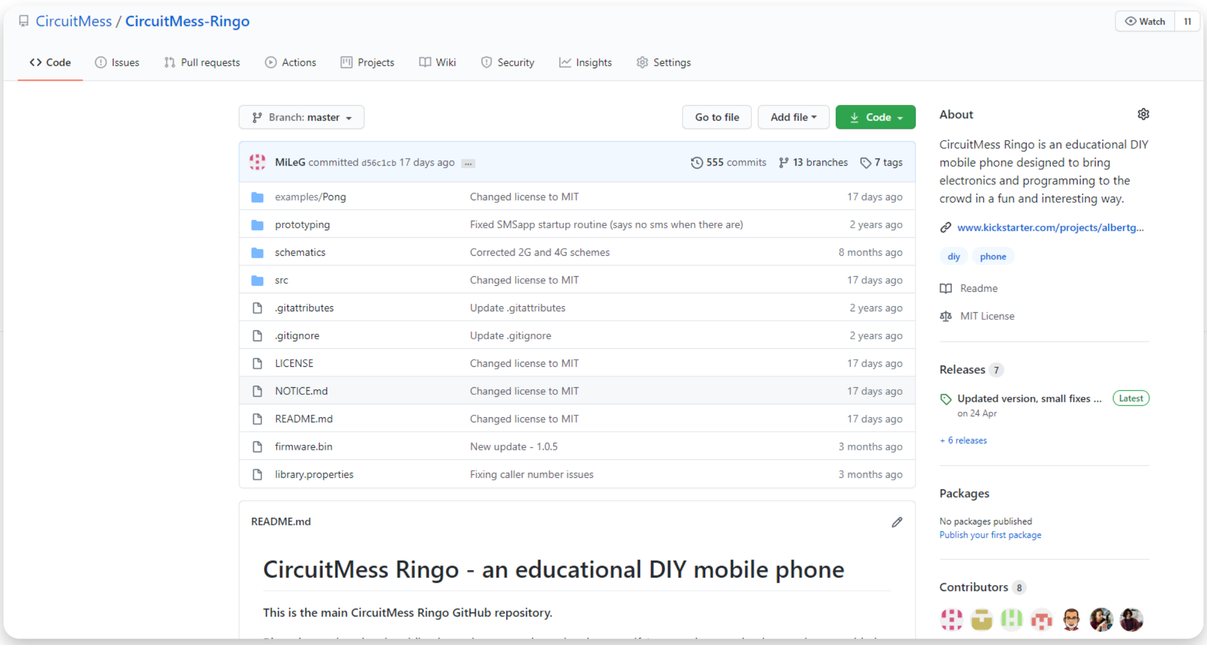
One of the most important files in this folder is **platformio.ini**, which contains the settings for the board.

Open it up and make sure that the parameters inside are set to the following:

```
[env:lolin32]
platform = espressif32
board = lolin32
framework = arduino
board_build.partitions = min_spiffs.csv
monitor_speed = 115200
upload_speed = 921600
```

If not, copy the lines above and paste it inside the file.

The next step is to download the **CircuitMess-Ringo** repository.



MiLeG committed d56c1cb 17 days ago · 555 commits · 13 branches · 7 tags

examples/Pong	Changed license to MIT	17 days ago
prototyping	Fixed SMSapp startup routine (says no sms when there are)	2 years ago
schematics	Corrected 2G and 4G schemes	8 months ago
src	Changed license to MIT	17 days ago
.gitattributes	Update .gitattributes	2 years ago
.gitignore	Update .gitignore	2 years ago
LICENSE	Changed license to MIT	17 days ago
NOTICE.md	Changed license to MIT	17 days ago
README.md	Changed license to MIT	17 days ago
firmware.bin	New update - 1.0.5	3 months ago
library.properties	Fixing caller number issues	3 months ago

README.md

CircuitMess Ringo - an educational DIY mobile phone

This is the main CircuitMess Ringo GitHub repository.

About

CircuitMess Ringo is an educational DIY mobile phone designed to bring electronics and programming to the crowd in a fun and interesting way.

www.kickstarter.com/projects/albertg...

diy phone

Readme

MIT License

Releases 7

Updated version, small fixes ... Latest
on 24 Apr

+ 6 releases

Packages

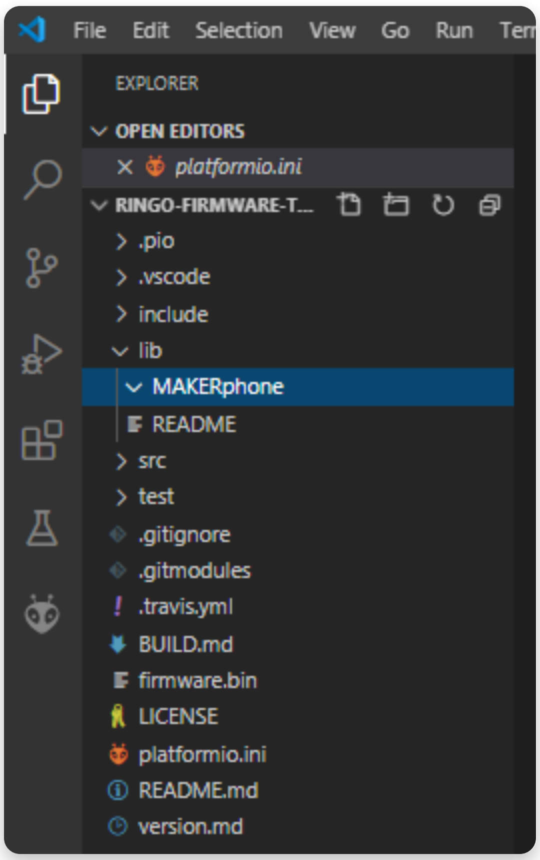
No packages published
Publish your first package

Contributors 8

CircuitMess Ringo repository

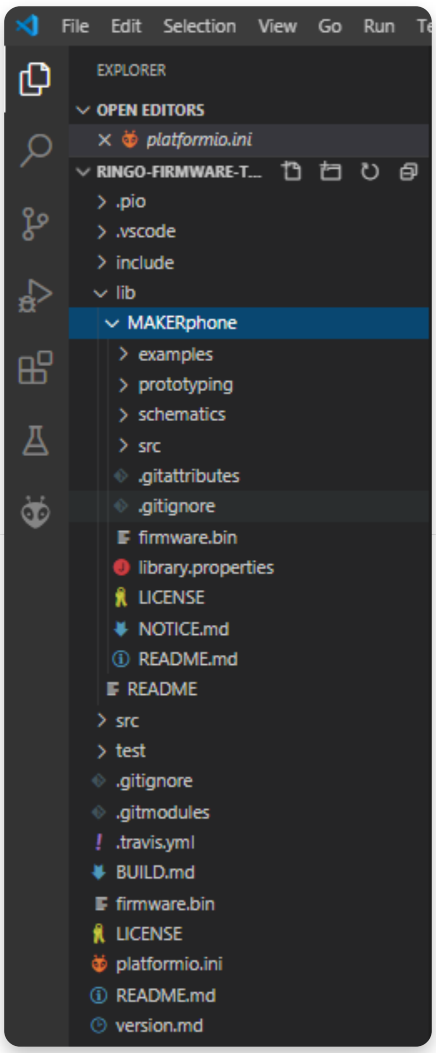
Download it as a ZIP (or by using the GitHub Desktop).

When unzipping it, make sure you place it inside the **lib > MAKERphone folder** of the main project folder.



Project folder before copying CircuitMess Ringo repository

Your project folder should now look something like this.



Project folder after copying CircuitMess Ringo repository

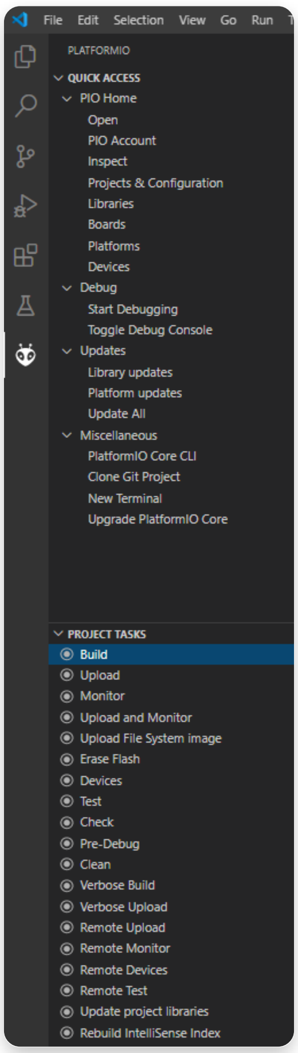
Now that everything is placed in the proper directories, it's time to compile and upload the firmware!

Fiddling with the firmware

Compilation test

If you've set everything up correctly, you should have no issues compiling the firmware as it is.

Select the PlatformIO icon on the left-hand sidebar and select **Build** under **Project tasks.**

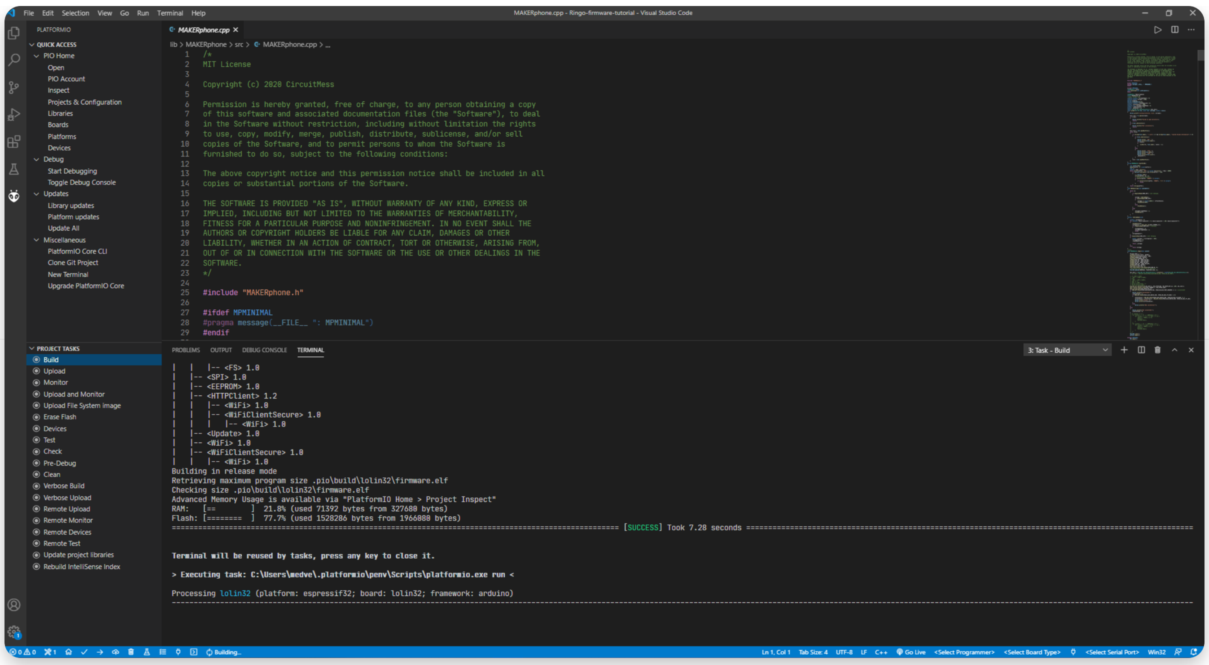


Building the firmware

The terminal should open up and show you the current status of the action.

The whole building of the firmware shouldn't take too long and it depends on your computer speed.

If you see a green **SUCCESS**, that means everything is up and running.



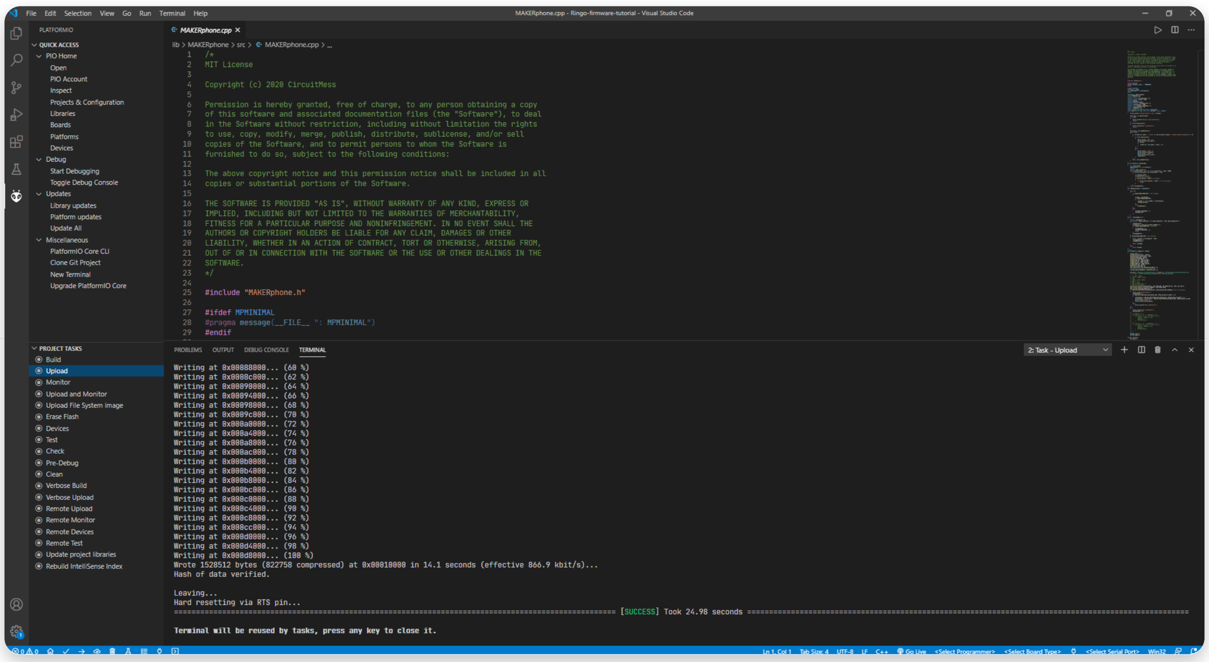
```
/*
MIT License

Copyright (c) 2020 CircuitMess

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "MAKERphone.h"

#ifdef MPMINIMAL
#pragma message(__FILE__ ": MPMINIMAL")
#endif
```

```
|   |   |-- <FS> 1.0
|   |-- <SPI> 1.0
|   |-- <EEPROM> 1.0
|   |-- <HTTPClient> 1.2
|   |   |-- <WiFi> 1.0
|   |   |-- <WiFiClientSecure> 1.0
|   |   |   |-- <WiFi> 1.0
|   |-- <Update> 1.0
|   |-- <WiFi> 1.0
|   |-- <WiFiClientSecure> 1.0
|   |   |-- <WiFi> 1.0
Building in release mode
Retrieving maximum program size .pio\build\lolin32\firmware.elf
Checking size .pio\build\lolin32\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [==        ]  21.8% (used 71392 bytes from 327680 bytes)
Flash: [========  ]  77.7% (used 1528286 bytes from 1966080 bytes)
=============================================================== [SUCCESS] Took 7.28 seconds ===============================================================

Terminal will be reused by tasks, press any key to close it.

> Executing task: C:\Users\medve\.platformio\penv\Scripts\platformio.exe run <

Processing lolin32 (platform: espressif32; board: lolin32; framework: arduino)
------------------------------------------------------------------------------------------------------------------------------
```

Terminal shows exactly what is happening during each task

Now when you connect your **Ringo** to the computer via the **USB cable** and press **Upload**, the compiled firmware should appear on your phone in a minute or so.

The phone itself will restart and you will get another green **SUCCESS** text.



```
Writing at 0x00088000... (60 %)
Writing at 0x0008c000... (62 %)
Writing at 0x00090000... (64 %)
Writing at 0x00094000... (66 %)
Writing at 0x00098000... (68 %)
Writing at 0x0009c000... (70 %)
Writing at 0x000a0000... (72 %)
Writing at 0x000a4000... (74 %)
Writing at 0x000a8000... (76 %)
Writing at 0x000ac000... (78 %)
Writing at 0x000b0000... (80 %)
Writing at 0x000b4000... (82 %)
Writing at 0x000b8000... (84 %)
Writing at 0x000bc000... (86 %)
Writing at 0x000c0000... (88 %)
Writing at 0x000c4000... (90 %)
Writing at 0x000c8000... (92 %)
Writing at 0x000cc000... (94 %)
Writing at 0x000d0000... (96 %)
Writing at 0x000d4000... (98 %)
Writing at 0x000d8000... (100 %)
Wrote 1528512 bytes (822758 compressed) at 0x00010000 in 14.1 seconds (effective 866.9 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
=============================================================== [SUCCESS] Took 24.98 seconds ===============================================================

Terminal will be reused by tasks, press any key to close it.
```

The uploading process usually takes a bit longer

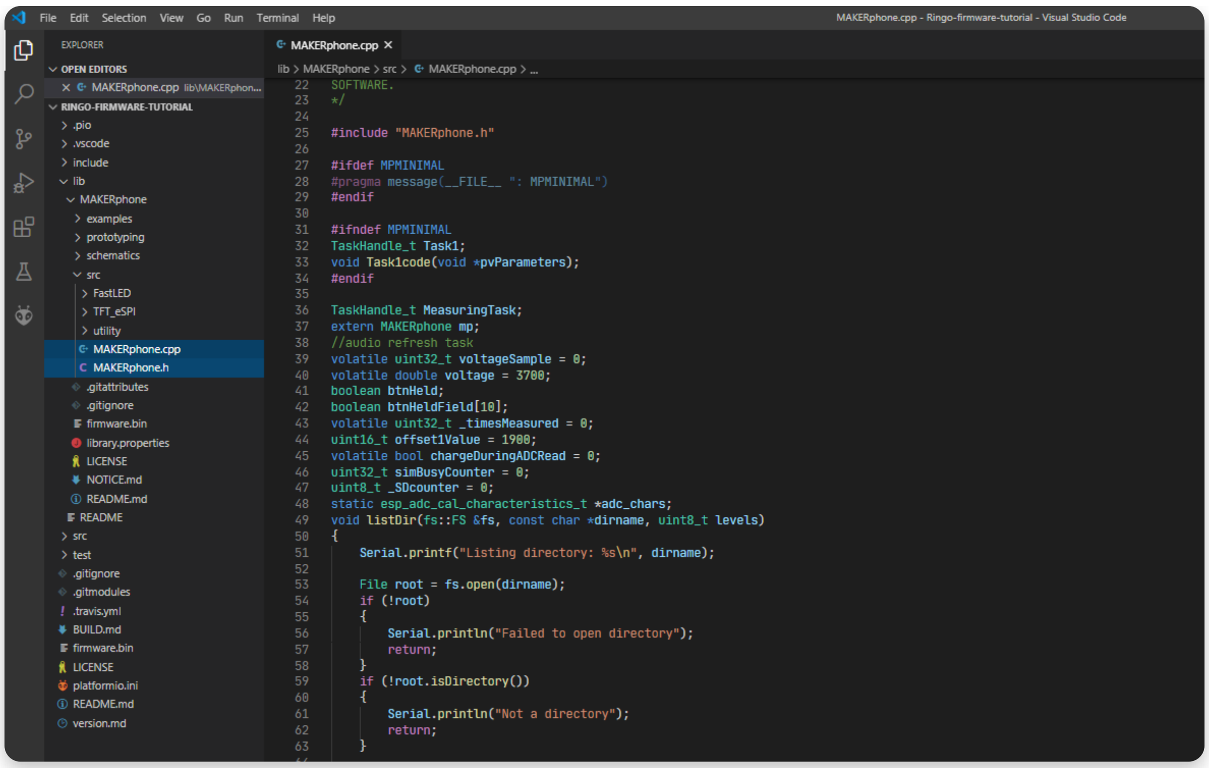
You've successfully compiled and uploaded the firmware for Ringo! Bravo!

Modification examples

Now that everything's set up, it's time to do something cool.

If you're wondering what are the things that you can change on the phone, the answer is - pretty much everything!

Two main files out of the bunch are **MAKERphone.cpp** and **MAKERphone.h.**



```
SOFTWARE.
*/

#include "MAKERphone.h"

#ifdef MPMINIMAL
#pragma message(__FILE__ ": MPMINIMAL")
#endif

#ifndef MPMINIMAL
TaskHandle_t Task1;
void Task1code(void *pvParameters);
#endif

TaskHandle_t MeasuringTask;
extern MAKERphone mp;
//audio refresh task
volatile uint32_t voltageSample = 0;
volatile double voltage = 3700;
boolean btnHeld;
boolean btnHeldField[10];
volatile uint32_t _timesMeasured = 0;
uint16_t offset1Value = 1900;
volatile bool chargeDuringADCRead = 0;
uint32_t simBusyCounter = 0;
uint8_t _SDcounter = 0;
static esp_adc_cal_characteristics_t *adc_chars;
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;
    }
```

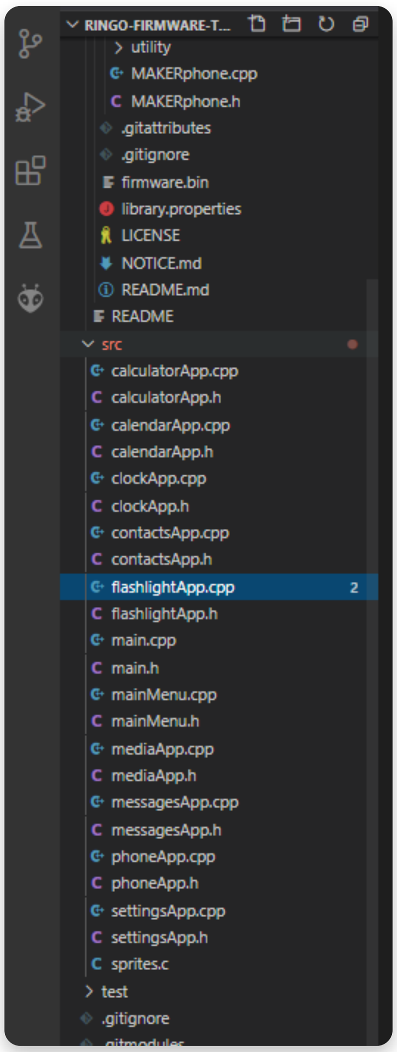
MAKERphone.cpp

If you're not familiar with the .cpp and .h extensions, it's time for you to use your friend Google and get acquainted.

Basically, .h files are header files and you don't want to mess around with them since they don't really contain any functionalities.

On the other hand, .cpp and .c files are the ones you want to edit.

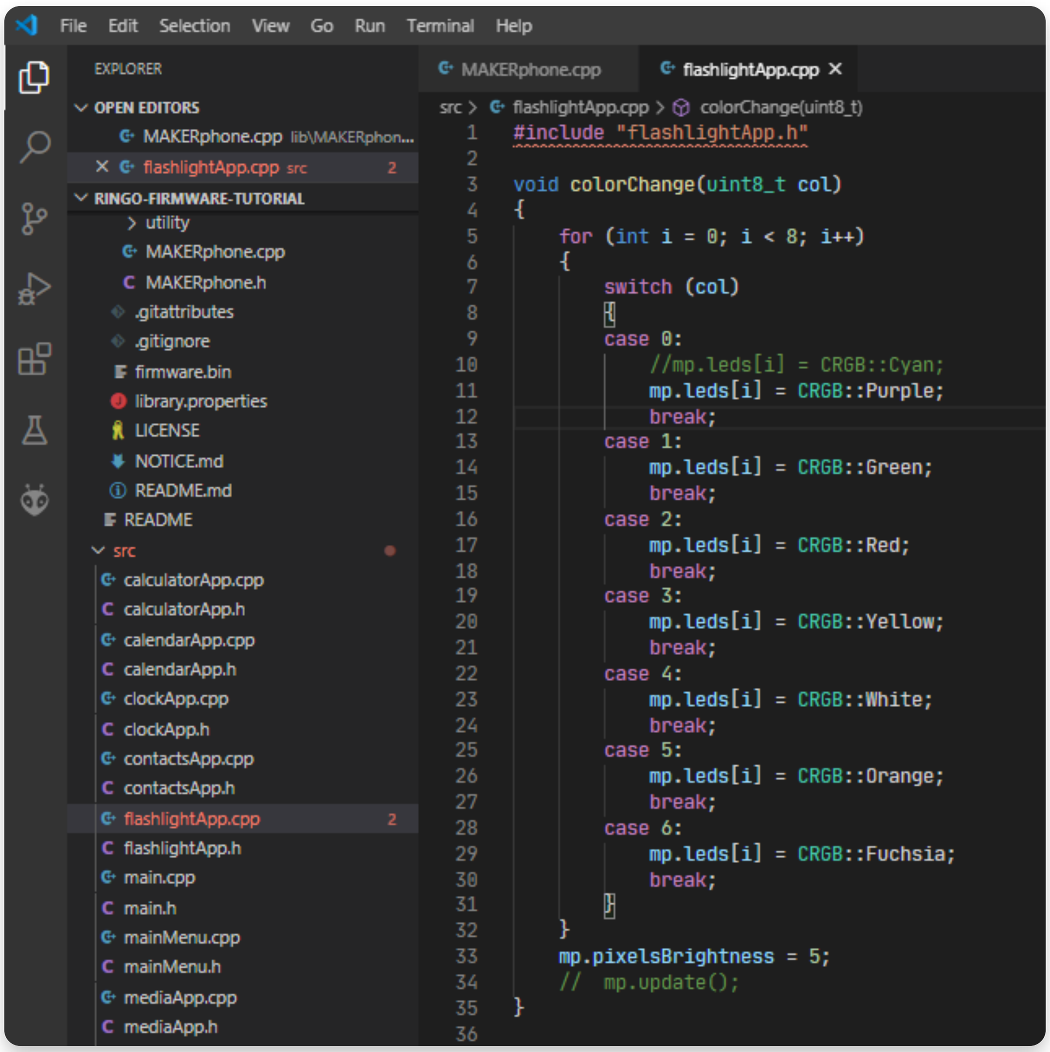
The place where you should be spending most of your time is in the **other src folder**, the one containing all the default apps.



Src folder containing all the default apps

In this folder, you can find and edit the files that define all the default menus, apps, and other main functionalities.

For example, you can change the color of LEDs in the flashlight app like this.



```
1   #include "flashlightApp.h"
2
3   void colorChange(uint8_t col)
4   {
5       for (int i = 0; i < 8; i++)
6       {
7           switch (col)
8           {
9           case 0:
10              //mp.leds[i] = CRGB::Cyan;
11              mp.leds[i] = CRGB::Purple;
12              break;
13          case 1:
14              mp.leds[i] = CRGB::Green;
15              break;
16          case 2:
17              mp.leds[i] = CRGB::Red;
18              break;
19          case 3:
20              mp.leds[i] = CRGB::Yellow;
21              break;
22          case 4:
23              mp.leds[i] = CRGB::White;
24              break;
25          case 5:
26              mp.leds[i] = CRGB::Orange;
27              break;
28          case 6:
29              mp.leds[i] = CRGB::Fuchsia;
30              break;
31          }
32      }
33      mp.pixelsBrightness = 5;
34      //  mp.update();
35  }
36
```

Changed lines are 10 and 11

Now when you build and upload the firmware, the first color of LEDs in the list inside the flashlight app will be purple instead of cyan.

However, this will only change the color of the LEDs, not the color shown on the screen, which is something you're going to have to change additionally in the flashlight app.

Everything is ready for your creations now!

You can scroll the forum for some ideas and already made user creations.

In our GitHub you can find many examples on how to use different functions and how to create your own apps.

Have fun and most importantly - **keep making**!