Board Bring Up Software

Edit New page

Jump to bottom

andy diller edited this page on Feb 11 · 11 revisions

Bringing Up the Software for FujiNet

This document will take you thru the steps needed to build, modify and deploy the FujiNet project onto the ESP32 system that is the heart of the FujiNet Hardware Device. Even if you are builing a raw devkit with an ESP32 dev board Board Bring Up Hardware or if you are using a retail FuijNet purcahsed from one of the many vendors selling them, you will have to build the same project and upload it to your device.

The production version of the #FujiNet firmware is being written in Platform.IO. If you want to help work on it, you'll need to bring up this version of the code on your hardware.

Install USB to UART Driver

You may need to install the USB to UART driver for your operating system available from Silicon Labs

Install git

If you haven't already done so. Please install git, which is required to check out the code from the source code repository.

Linux

Debian/Ubuntu/Mint/etc.

```
apt-get install git-core
```

or

```
apt-get install build-essential
```

Which also adds a bunch of useful things.

For Ubuntu users

NOTE: You need to add yourself to the dialout group and reboot

```
sudo adduser $USER dialout
sudo reboot
```

Windows

Under Windows, git can simply be installed from here: https://git-scm.com/download/win

macOS

Installing XCode from the App Store will give you 'git'. Install XCode via the command line with:

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

Brew

If you have brew installed you can install Visual Studio Code via brew:

```
brew install visual-studio-code
```

Brew also contains a better version of git for MacOS:

```
brew install git
```

Install Visual Studio Code

Microsoft Visual Studio Code is used as the IDE for Platform.IO. You can get a copy of it for Windows, Linux, or Mac.

- Download a copy of Visual Studio Code for Windows, Mac, or Linux

Note for Linux users:

As I was testing these instructions for Linux, I found that I had to install the following packages after installing Visual Studio Code, due to only a limited Python environment being installed by default under Ubuntu based Linux distributions:

```
apt-get install python3-venv
apt-get install python3-distutils
```

Install Platform.IO

Once you've installed Visual Studio Code you will need to run Visual Studio Code and install the Platform.IO extension from the following Link:

- Install PlatformIO for Visual Studio Code

Restart Visual Studio Code

Let the resulting terminal window finish installing Platform.IO

Once the `Reload Now` button appears, press it.

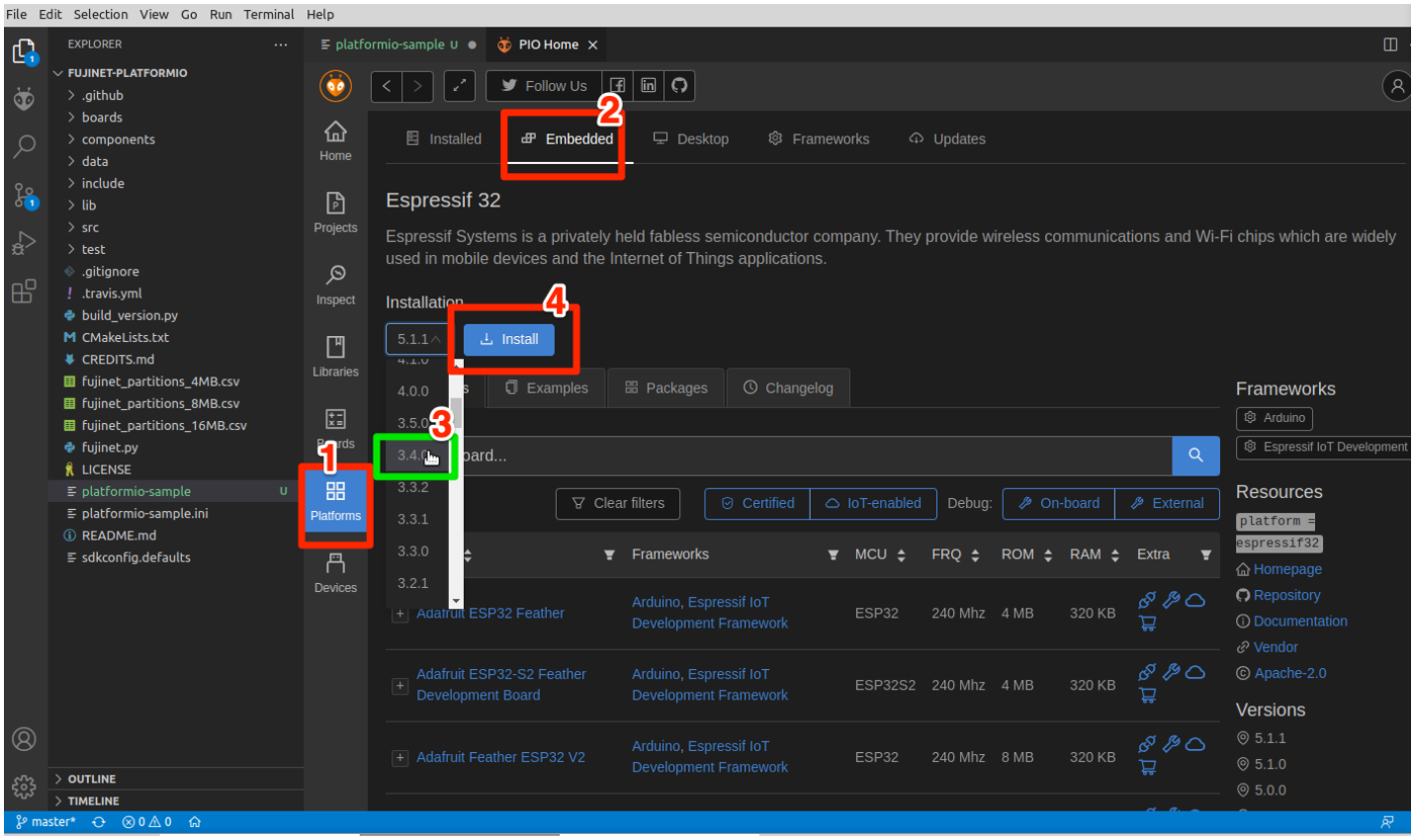
Install support for your board

Once Platform.IO is installed, a new tab should appear with PIO Home, with icons down its left side. Select **Platforms**[1]. You should then select **Embedded**[2]. You should then select the **Espressif 32** ESP32 platform for #FujiNet.

On the pull-down ensure you choose version **3.4.0**

Then click the install button[4].

- Espressif 32 - for ESP32 based boards



You may need to quit and restart for the platforms to be loaded in the left-hand pane (Linux seems require a restart).

Clone the FujiNet repository

You should then press CTRL-` (the backtick), to open a terminal and move to where you want your repository to be. If you haven't created a space for projects, you should make one. e.g.

```
mkdir Projects
cd Projects
```

Anonymously cloning the repository

If you are only planning to test, you should then clone the repository with the following git command:

```
git clone https://github.com/FujiNetWIFI/fujinet-platformio.git
```

This command will download the project to a folder named `fujinet-platformio` in the current directory.

Using your GitHub account

If you intend to contribute to this code, you should have an account on github.com. It is easy enough to make one, and it's free.

You should then have an SSH public key associated with your account that matches the one used by Visual Studio Code. If there isn't one yet, you can run the following program to generate one:

```
ssh-keygen
```

And then view the contents of the file it generated:

```
cat ~/.ssh/id_rsa.pub
```

Copy the contents to the clipboard. Go to your account settings, select SSH Keys, and add the key you just copied into the clipboard there.

You can then check out the code with the following command:

```
git clone git@github.com:FujiNetWIFI/fujinet-platformio.git
```

This command will download the project to a folder named `fujinet-platformio` in the current directory.

Don't like SSH?

If you are not comfortable using SSH keys you can also use Personal Access Tokens to checkout and push code using HTTPS (not ssh).

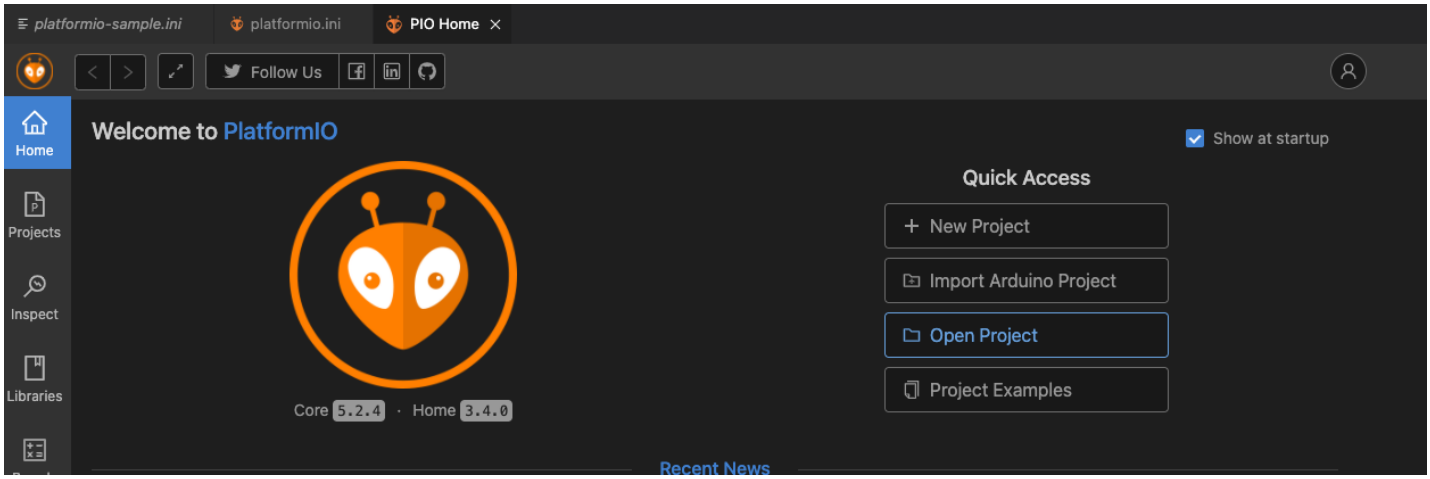
- Using Personal Access Tokens in GitHub Again, this is a requirement if you are adding and fixing code and making PR requests to help FujiNet. It is not required to simply pull the latest code, build it and flash your FujiNet.

Create a platformio.ini file from the sample before opening the project

Before opening the project, copy the file in the project folder you just cloned (`fujinet-platformio`) named `platformio-sample.ini` to `platformio.ini` . This file both tells the PlatformIO extension in Visual Studio Code that this is a PlatformIO project, and is the place you'll adjust settings for your particular hardware configuration (see below). For example:

```
cp platformio-sample.ini platformio.ini
```

Now open the project in Visual Studio Code by using the **Open Folder** option in the **File** menu and navigating to open the fujinet-platformio folder you cloned.



You should now see "FUJINET-PLATFORMIO" in the Explorer menu bar (click on the top-left icon that looks like a couple of stacked pages) with various folders including `data`, `lib`, and `src`.

Modify platformio.ini for your hardware configuration

Select the Explorer tab (upper-left-most icon) and open the `platformio.ini` file.

In the top `[fujinet]` section, uncomment the `build_platform` and `build_bus` variable that targets the system you are building for. Also, uncomment the `build_board` variable that targets the physical hardware you are building for. Multiple `build_board` examples are provided in `platformio-sample.ini` but only one should be uncommented.

Setting USB Speeds

We were using 921600 for both upload and monitor speeds; however, we ran into a problem with one devit model that couldn't support that high rate, so dropped back to 460800. You can change the speeds back if your devkit works OK. Also have to change DEBUG_SPEED in build_flags to match the monitor_speed.

Unless you know exactly what you are doing, use 460800 as all speeds.

Example:

```
[fujinet]
; Espressif32 PlatformIO Version to use for building
esp32_platform_version = 3.4.0
;esp32_platform_version = 3.2.0 ; For Bluetooth support

; Uncomment the platform, bus and board type you are building for below
;build_platform = BUILD_ATARI
;build_bus      = SIO
;build_board    = fujinet-atari-v1       ; FujiNet for Atari v1.0 and up
;build_board    = fujinet-v1-8mb         ; Atari generic/custom with 8mb flash
;build_board    = fujinet-v1-4mb         ; Atari generic/custom with 4mb flash

;build_platform = BUILD_ADAM
```

```
;build_bus      = ADAMNET
;build_board    = fujinet-adam-v1       ; FujiNet for Coleco ADAM v1.0

;build_platform = BUILD_APPLE
;build_bus      = IWM
;build_board    = fujiapple-rev0         ; FujiApple Rev 0 Prototype
;build_board    = fujiapple-fujinetv1    ; FujiApple using Atari FujiNet v1.0

;build_platform = BUILD_CBM
;build_bus      = IEC
;build_board    = fujinet-cbm            ; Commodore IEC using Atari FujiNet

build_platform = BUILD_LYNX
build_bus      = comlynx
;build_board    = fujinet-lynx-prototype ; Lynx Prototype PCB
build_board    = fujinet-lynx-devkitc   ; Lynx with basic DEVKITC

;build_platform = BUILD_S100
;build_bus      = s100Bus
;build_board    = fujinet-v1-8mb         ; ESP32-DEVKITC-VE
```

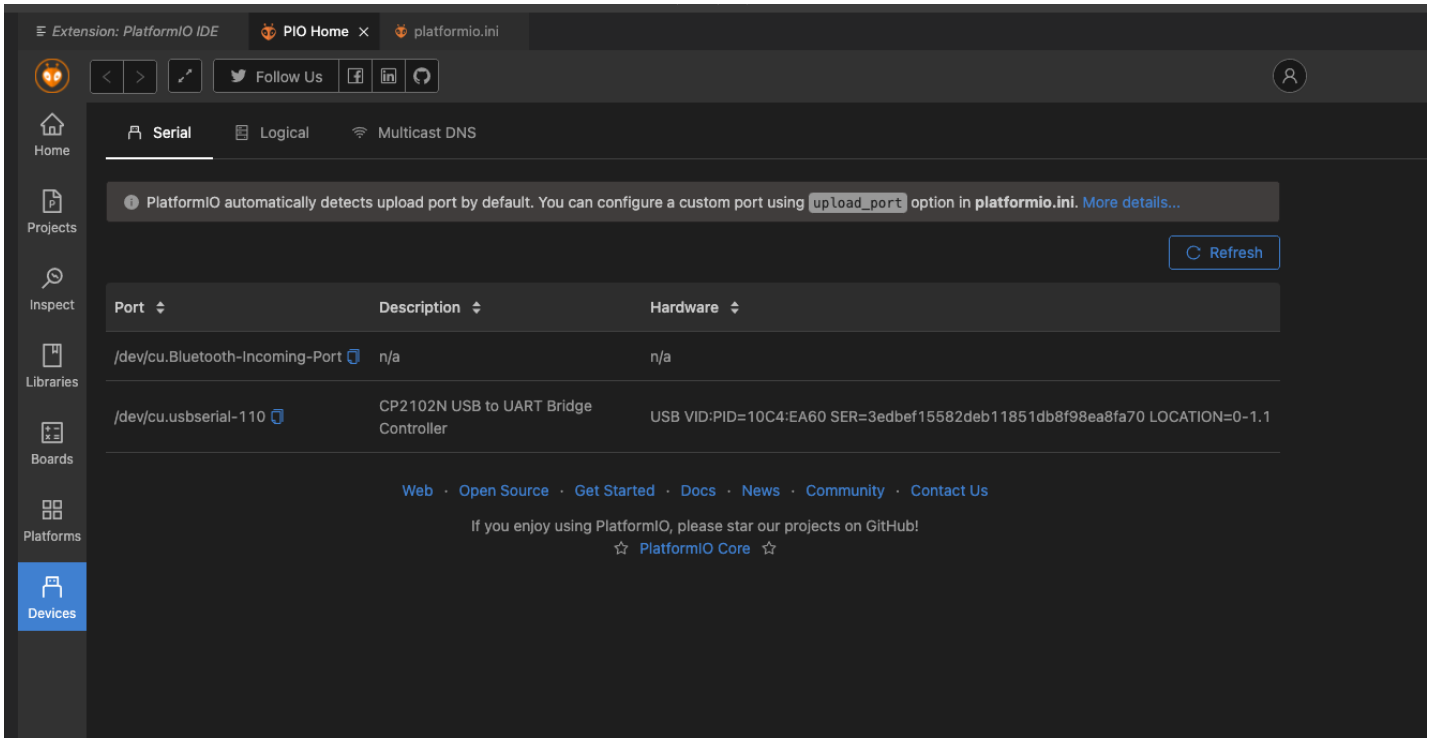
In the above example, I've commented out the LYNX platform and the devkitc. Ensure only one platform is ever enabled at build time. Simply uncomment the platform, bus and board you want to build and ensure any others are commented out.

In the `[env]` section, you need to set the communications port your board is connected to. This port is used for both loading the FujiNet firmware on the board and viewing its debugging output. The speeds defined in the sample configuration file should work in most configurations, but you'll very likely need to change the port number. `upload_port` and `monitor_port` are usually set to the same value, and `upload_speed` and `monitor_speed` are usually set to the same value.

Example:

```
[env]
upload_port = /dev/ttyUSB0
upload_speed = 921600
monitor_port = /dev/ttyUSB0
monitor_speed = 921600
```

You can see what your local USB device is named by clicking on the Devices tab, all connected USB serial devices should be listed. If not ensure your USB cable is plugged in and then click the Refresh button.



If you'd like to change build settings, you should change them in the `[env:]` section. The main setting that is of interest here is the `build_flags` which makes certain changes when building the firmware code. Build flags of interest are:

- JTAG: Enables support for JTAG debugging.
- DEBUG_SPEED: Set's the FujiNet's debug output log speed. This should match the `monitor_speed` setting in the `[env]` section.
- BLUETOOTH_SUPPORT: Enables FujiNet's Bluetooth support. As the Bluetooth library is relatively large, disabling this can reduce required system resources.
- FN_HISPEED_INDEX: This is the "POKEY divisor" value FujiNet uses when communicating over the Atari SIO bus. The value is '6' by default, which is compatible with SpartDOS's high-speed mode. You can set this to '0' if you have an Atari OS that supports it.
- NO_BUTTONS: disables button activity within the code for custom boards that do not have them
- VERBOSE_xxx: Several VERBOSE flags that give more debugging information from FujiNet

Note that a semicolon before a value makes it disabled or use the default value.

This example has JTAG disabled but Bluetooth enabled and debug monitor speed set to 921,600 baud:

```
[env]
build_flags =
   ; -D JTAG
   -D BLUETOOTH_SUPPORT
   -D DEBUG_SPEED=921600
```

The `build_board` variable chosen above selects a more specific `[env]` targeting specific hardware. There are additional `build_flags` in these sections that relate specifically to the platform selected. You may (un)comment `build_flags` in this section as needed BUT do NOT uncomment the `PINMAP_XXXX` line as this is needed to define which pins are used by the ESP32 (these are defined in `include/pinmap.h`).

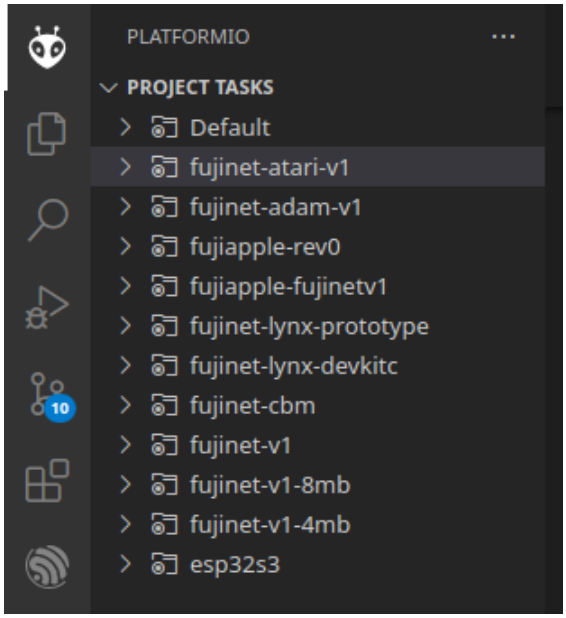
Atari Example:

```
; FujiNet for Atari v1.0 and up (ESP32 WROVER 16MB Flash, 8MB PSRAM)
[env:fujinet-atari-v1]
platform = espressif32@${fujinet.esp32_platform_version}
board = fujinet-v1
build_type = debug
build_flags =
    ${env.build_flags}
    -D PINMAP_ATARIV1
    ;-D DEBUG_UDPSTREAM     ; enable UDP to display IN/OUT packets
    ;-D VERBOSE_SIO         ; Debug SIO
    ;-D VERBOSE_ATX         ; Debug ATX files
    ;-D FN_HISPEED_INDEX=0  ; Set SIO High Speed Index
```

Attempt to compile

Select the Platform.IO tab (it has an "alien head" icon).

Expand the project environment for the `build_board` you have chosen in `platformio.ini` . This will on-demand load required libraries/frameworks (this can take some time, the first time you do this).



Carefully Read this part

As of November 2021, there are several bugs in the Espressif 32 IDF that need manual fixing until PlatformIO pulls in a newer version of ESP-IDF. These bugs affect the Espressif 32 PlatformIO version 3.4.0

1. ~~Edit .platformio/packages/framework-espidf/components/mbedtls/esp_crt_bundle/cacrt_all.pem 1. remove the EC-ACC certificate~~
   2022-01-22 — The certificate appears to be fixed... "deprecating" this instruction until we're comfortable it's not an issue.
2. Edit .platformio/packages/framework-espidf/components/newlib/platform_include/sys/dirent.h
   - Add after the #includes:

     ```
     #ifdef __cplusplus
     extern "C" {
     #endif
     ```

   - and at the very bottom:

     ```
     #ifdef __cplusplus
     }
     #endif
     ```

If you are comfortable with a terminal, there are some UNIX commands that can make these changes for you. They are listed below for you to use.

This command will open the ifdef at the top of the dirent.h file:

```
gawk -i inplace '1;/#include\ <stdint.h>/{print "\n\n#ifdef __cplusplus\nextern \"C\"
{\n#endif\n\n"}' ~/.platformio/packages/framework-
espidf/components/newlib/platform_include/sys/dirent.h
```

This command will close the ifdef in the dirent.h file:

```
printf "\n\n\n#ifdef __cplusplus\n}\n#endif\n\n" >> ~/.platformio/packages/framework-
espidf/components/newlib/platform_include/sys/dirent.h
```

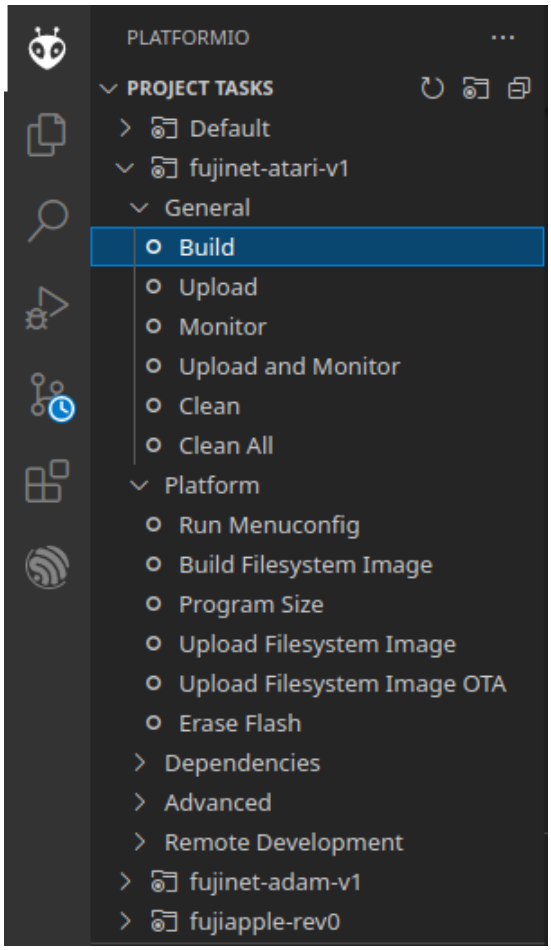
You can verify the changes have been made to the file:

```
tail -n 10 ~/.platformio/packages/framework-
espidf/components/newlib/platform_include/sys/dirent.h

head -n 30 ~/.platformio/packages/framework-
espidf/components/newlib/platform_include/sys/dirent.h
```

Of course you can always just open up the file ~/.platformio/packages/framework-espidf/components/newlib/platform_include/sys/dirent.h in your favorite text editor and make the appropriate changes.

Return to the Platform.IO tab, and choose `Build`.



The TERMINAL section of Visual Studio Code will show the progress of the build. You know it's done right when you get the output:

```
[... bunch of text ...]

Retrieving maximum program size .pio/build/fujinet-v1/firmware.elf
Checking size .pio/build/fujinet-v1/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
```

```
RAM:   [          ]   1.1% (used 50036 bytes from 4521984 bytes)
Flash: [=         ]  14.4% (used 1363069 bytes from 9437184 bytes)
Building .pio/build/fujinet-v1/firmware.bin
esptool.py v3.1
Merged 2 ELF sections
========================================================================= [SUCCESS] Took 104.31
seconds

Environment    Status    Duration
-------------  --------  ------------
fujinet-v1     SUCCESS   00:01:44.307
========================================================================= 1 succeeded in
00:01:44.307

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

If you see output that indicates error conditions you won't have a firmware.bin file to upload. Make the corrections to ensure that the build completes and you have a firmware.bin create. This will be used in the next steps to flash the fujinet with the new functionality.

Upload the firmware to your ESP32 board

There are two blocks of data to upload to your board: the file system image and the firmware. The file system image consists of files that the firmware makes use of while operating (e.g. printer emulation fonts). These are stored in the project's `data` directory and don't change very often. The firmware is the actual code the board runs, and this changes with every new build.

Select the `Upload File System Image` task in the Platform.IO tab to upload the contents of the `data` directory.

Select the `Upload` task in the Platform.IO tab to upload the firmware to your board. This will also cause the firmware to be re-compiled if any changes have been made to any of the source code files.

If everything works, the FujiNet board should automatically reboot and start running the newly uploaded firmware image. You can use the `Monitor` option in the Platform.IO menu to view any debug logs the FujiNet generates as it operates.

To recap:

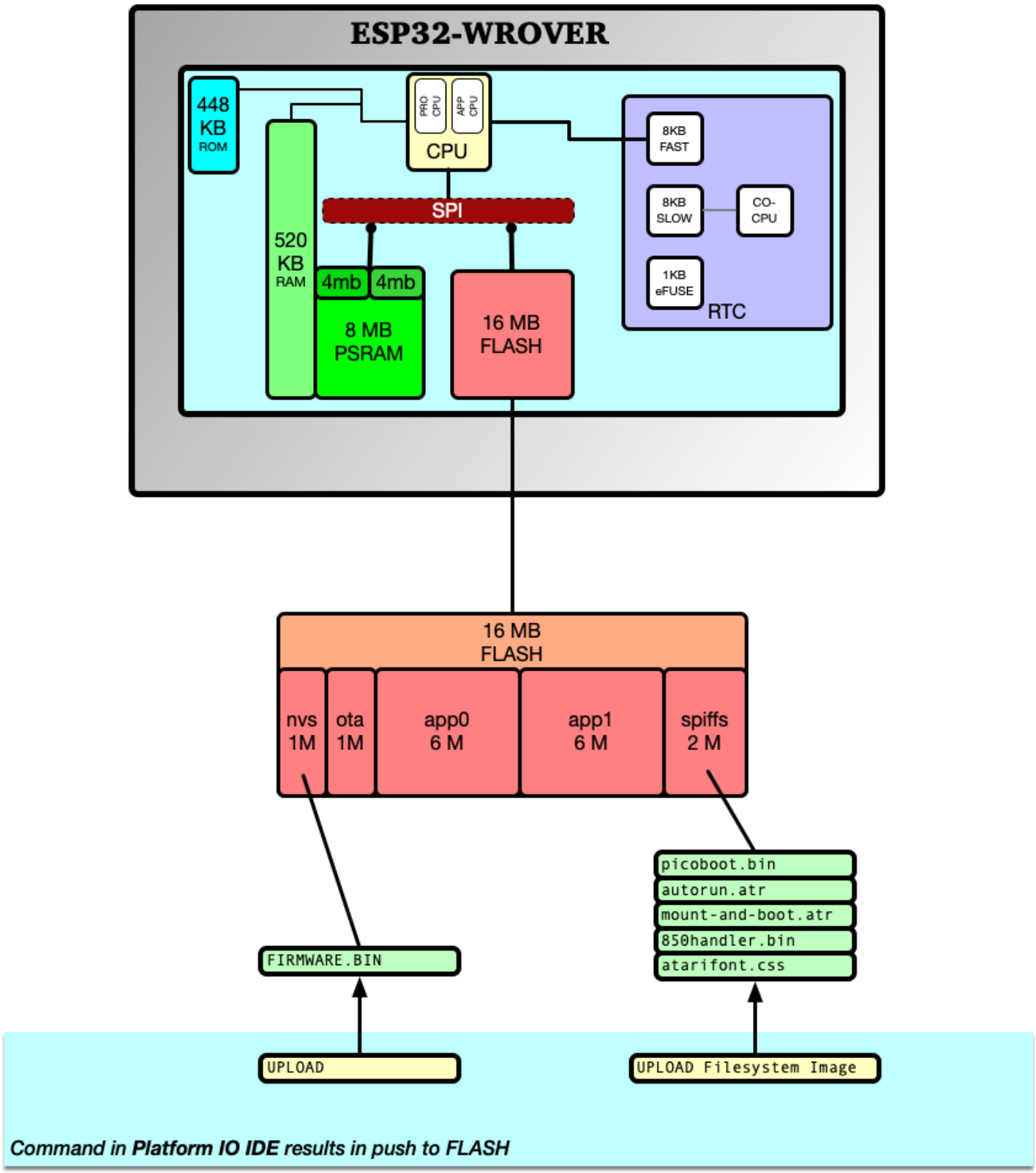
- Be in the correct git branch for the work you doing
- Update code as required, saving when necessary
- `Build` - will recomplie what is necessary given the changed files
- `Upload File System Image` - use only if you have changed items in the `data` directory - like web or a CONFIG app for the platform. *It's not necessary to do this step if you have just made changed to the firmware code.*
- `Upload and Monitor` - will upload the Firmware, reset the ESP32 and show debug output of the system in the VSC terminal area

Example debug output:

```
12:54:57.236 > --~--~--~--
12:54:57.236 > FujiNet 0.5.8d6b340f 2022-01-27 04:16:40 Started @ 5
12:54:57.237 > Starting heap: 4396479
12:54:57.272 > Detected Hardware Version: 1.0
12:54:57.423 > SPIFFS mounted.
12:54:57.505 > SD mounted.
12:54:57.505 > fnConfig::load
12:54:57.505 > Load fnconfig.ini from SD
12:54:57.505 > fopen = ok
12:54:57.509 > fnConfig::load read 631 bytes from config file
12:54:57.569 > SPIFFS Config Storage: Enabled
12:54:57.571 > fnConfig::load read 631 bytes from SPIFFS config file
12:54:57.679 > WIFI_EVENT_STA_START
12:54:57.679 > WiFi connect attempt to SSID "_irisindigo_"
12:54:57.681 > esp_wifi_connect returned 0
12:54:57.681 >
12:54:57.681 > IWM FujiNet based on SmartportSD v1.15
12:54:57.683 >
12:54:57.683 > IWM GPIO configured
12:54:57.683 > IWM timer started
12:54:57.683 > mounting serverResolving hostname "3.211.118.17"
12:54:57.683 > Resolved to address 3.211.118.17
12:54:57.683 > TNFS mount 3.211.118.17[3.211.118.17]:16384
12:54:57.683 > could not send data: 118Failed to send packet - retrying
12:54:58.425 > WIFI_EVENT_STA_CONNECTED
12:54:58.677 > could not send data: 118Failed to send packet - retrying
12:54:59.566 > IP_EVENT_STA_GOT_IP
12:54:59.566 > Obtained IP address: 10.0.0.151
12:54:59.566 > SNTP client start
12:54:59.568 > Starting web server on port 80
12:54:59.742 > TNFS mount successful. session: 0x18f8, version: 0x0102, min_retry: 1000ms
12:54:59.743 > vfs_tnfs_register "/tnfs0x3ffc367c" @ 0x3ffc367c = 0 "ESP_OK"
12:54:59.745 >
12:54:59.745 > opening fileTNFS open file: "/APPLE/A2OSX.PO" (0x0001, 0x01b6)
12:54:59.850 > File opened, handle ID: 0, size: 33553920, pos: 0
12:54:59.850 >
12:54:59.850 > Testing file tnfs_lseek currpos=0, pos=0, typ=1
12:55:00.383 > SNTP time sync event: Sat Jan 29, 12:55:00 2022 -0500
12:55:00.394 >
12:55:00.394 > file open good
12:55:00.396 > Demo TNFS file open complete - remember to remove this codeAvailable heap:
4166023
12:55:05.386 > Setup complete @ 8154 (8149ms)
```

What is going where?

Loading builds into the ESP32. This diagram shows what parts are pushed into the ESP32




```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
```

```
app0,     app,  ota_0,   0x10000, 6M,
app1,     app,  ota_1,     ,6M,
```


Join us on Discord: https://discord.gg/7MfFTvD

Pages 216

https://github.com/pellepl/spiffs/blob/master/docs/TECH_SPEC

        - NQUERY
        - NTIME
    - Help me get written!
- C64 Programming
    - Help me get written!
- ADAM Programming
    - Help me get written!
- Testing Plan
- Hacker List

Clone this wiki locally

```
https://github.com/FujiNetWIFI/fujinet-platformio.wiki.git
```