English

Welcome to Petoi Doc Center

This is the GitBook hub for hosting the documentation of our products. We keep fast iteration on our models and codes to bring bionic robotic pets to the world. Please read the notes regarding versions carefully to configure your robot.

If you need any help, please write to support@petoi.com, or post on our forum at petoi.camp.

Products

Nybble Cat User Manual

Bittle Dog User Manual

Guide for the Petoi App

□□

Introduction

Thanks for choosing Petoi's robot Bittle or Nybble. This guide will help you set up your robot buddy and provide a simpler UI to calibrate, control, and program it. For advanced users, we recommend you keep the robot updated with the OpenCat firmware on Github for the best compatibility and newest features.

Download and installation

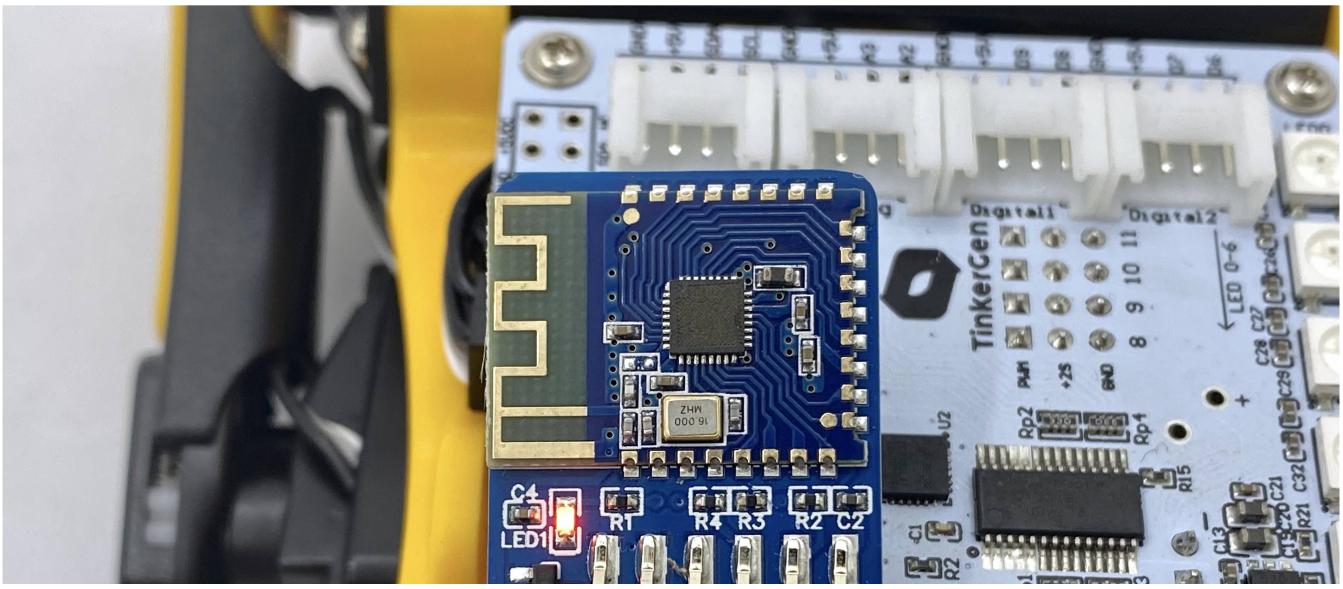
The app works on both Android and iOS devices.

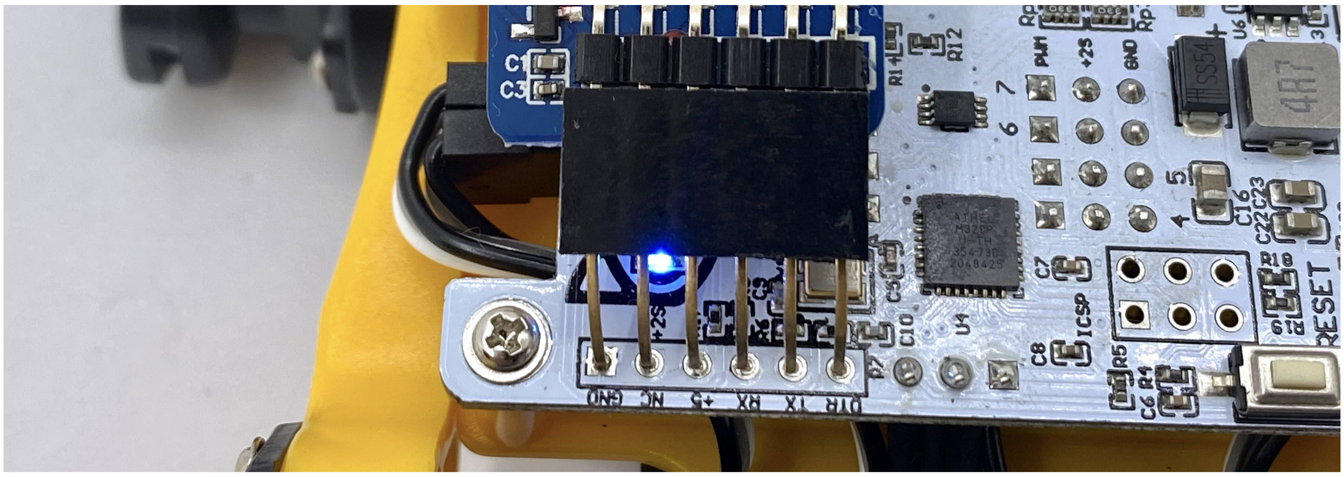
- Android 4.4+
- iOS 11+

Connect to your robot

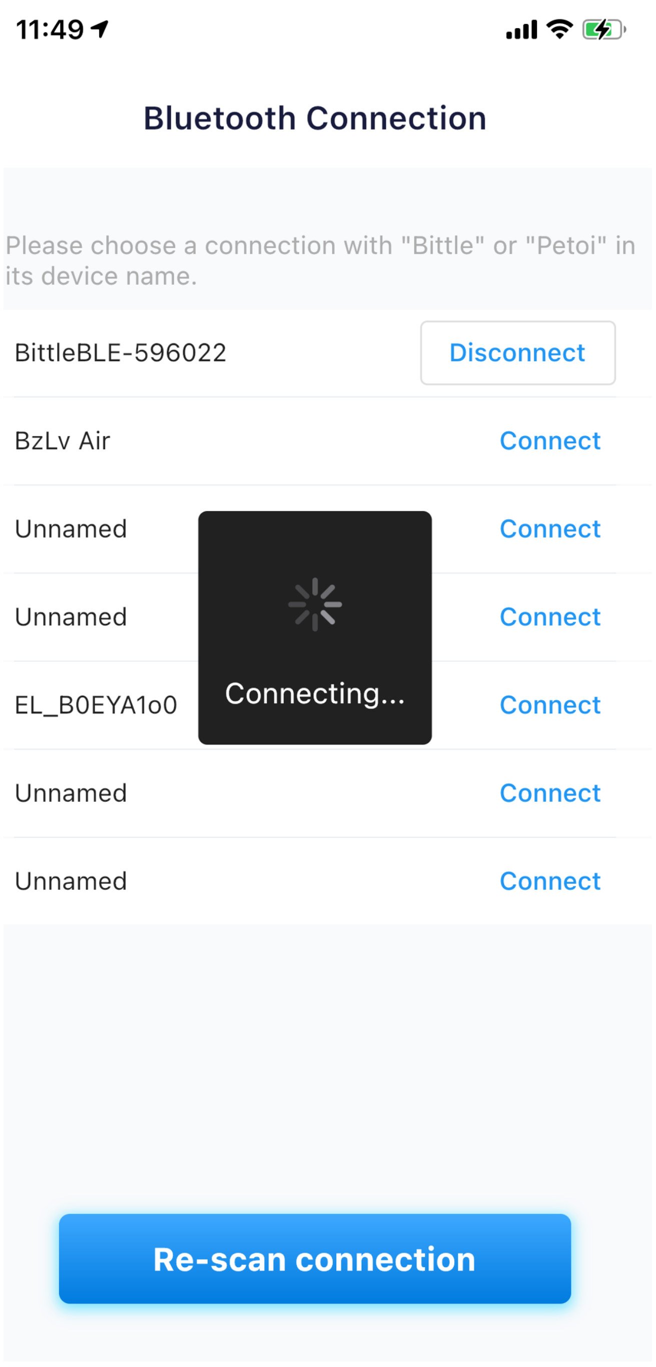
You need to plug the Bluetooth dongle into the 6-pin socket on the NyBoard. Pay attention to the Bluetooth dongle's pin order. Long-press the button on the battery to turn on the robot's power.

> ⓘ If the buzzer beeps three times (bi-bi-bi) repetitively after bootup or during use, it means the battery is low. Please charge it in time. The charging port is on one end of the battery.





The LED on the Bluetooth dongle should blink waiting for a connection. Open the app and scan to connect the device with the name Bittle, Petoi, or OpenCat. Remember to open the Bluetooth service and grant the app access to the service.



11:49

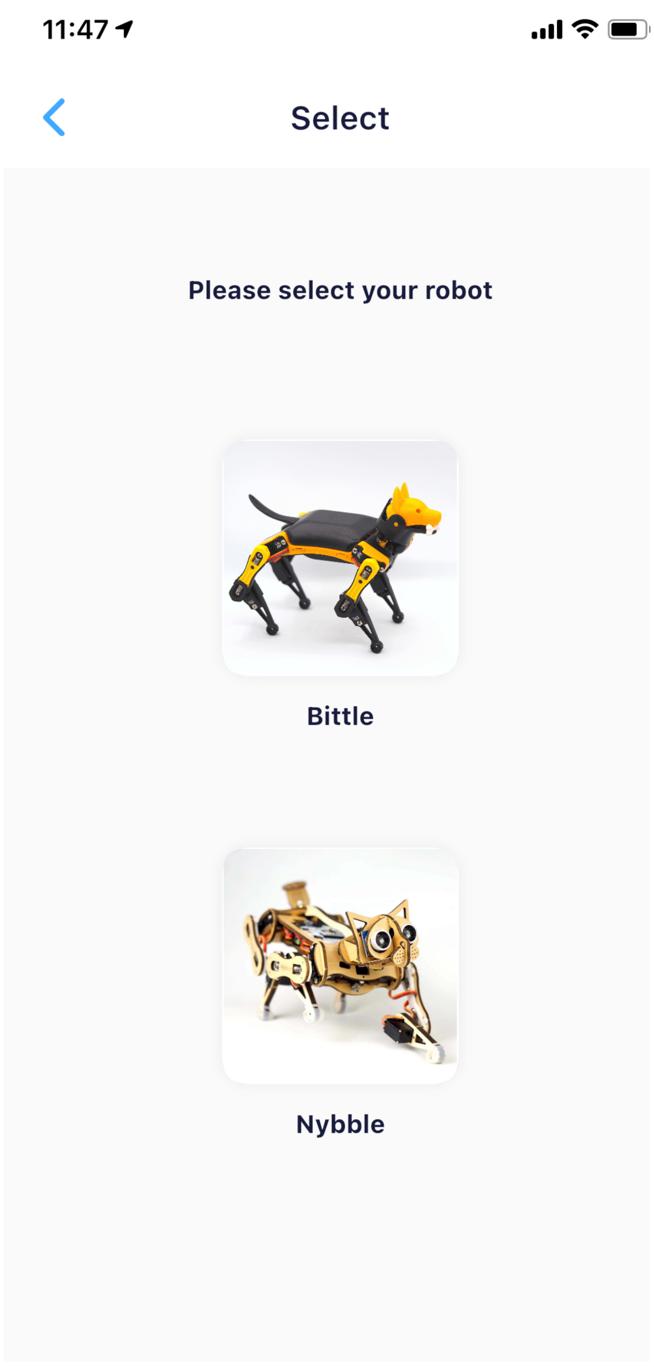
Bluetooth Connection

Please choose a connection with "Bittle" or "Petoi" in its device name.

BittleBLE-596022	Disconnect
BzLv Air	Connect
Unnamed	Connect
Unnamed	Connect
EL_B0EYA1o0	Connect
Unnamed	Connect
Unnamed	Connect

Connecting...

Re-scan connection



11:47

Select

Please select your robot

Bittle

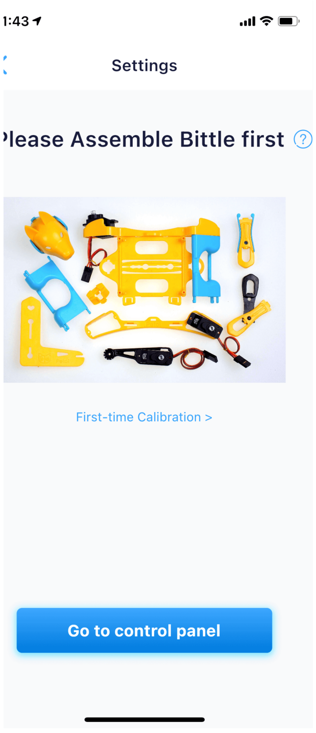
Nybble

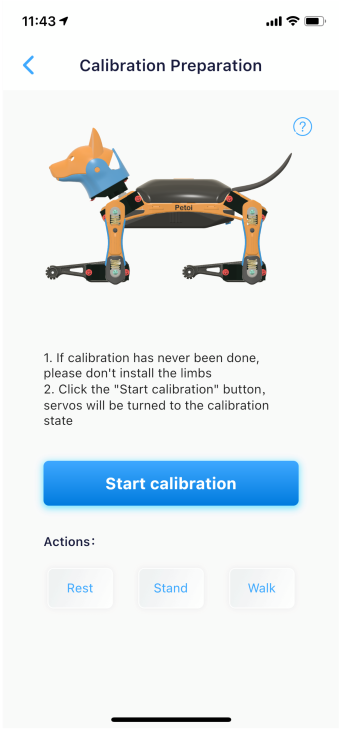
If the Bluetooth is connected, its LED will light steady. The robot will play a three-tone melody. If the robot doesn't respond or malfunction later, press the reset button on the NyBoard to restart the program on the NyBoard.

The App should automatically detect Nybble or Bittle with the latest OpenCat firmware. Otherwise, it will show the selections for Nybble or Bittle. The option can be re-visited in the control panel.

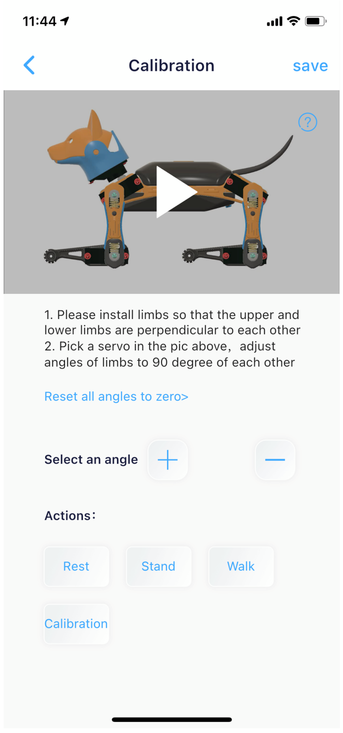
Calibrate the joints

The following screen will be calibration for first-time users. It can also be re-visited in the control panel.

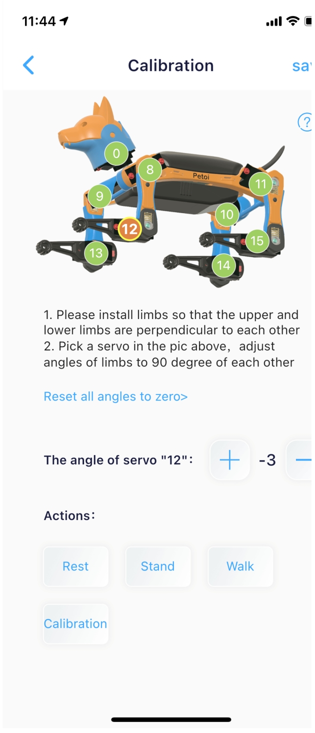




1. If calibration has never been done, please don't install the limbs
2. Click the "Start calibration" button, servos will be turned to the calibration state



1. Please install limbs so that the upper and lower limbs are perpendicular to each other
2. Pick a servo in the pic above, adjust angles of limbs to 90 degree of each other



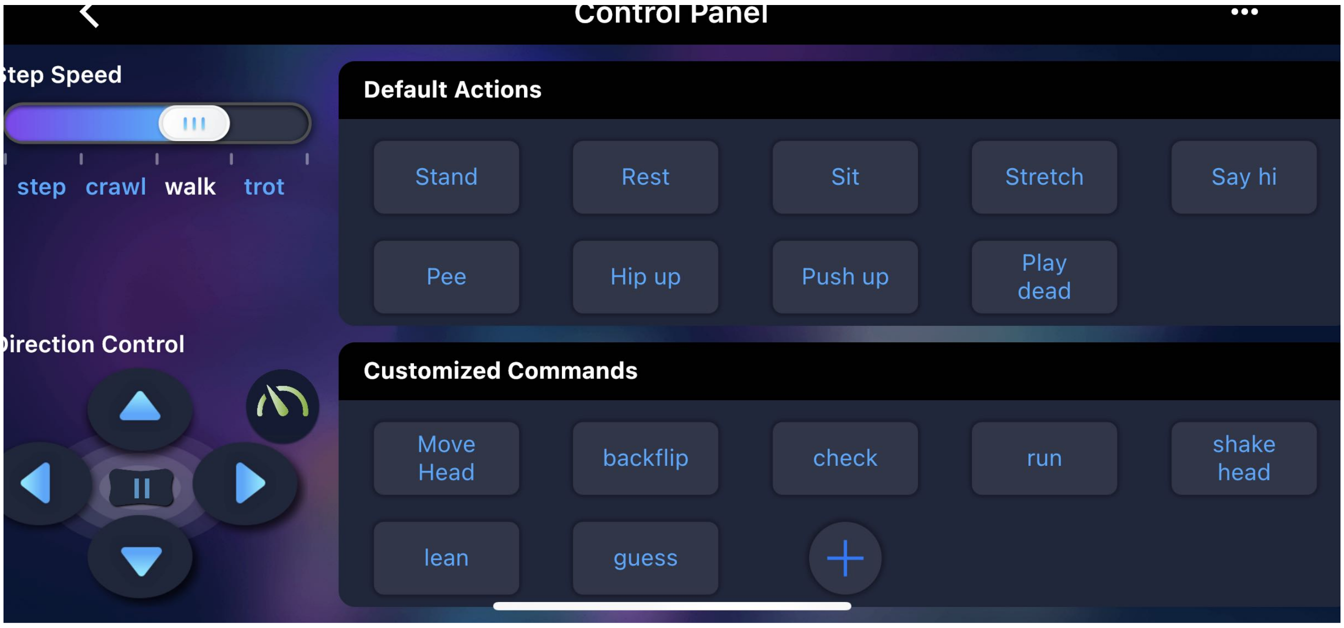
1. Please install limbs so that the upper and lower limbs are perpendicular to each other
2. Pick a servo in the pic above, adjust angles of limbs to 90 degree of each other

Select the joint index in the image on the calibration page, then click + or - to fine-tune the joints to right angles. **Use the included L-shape tool as a reference.** If the offset exceeds +/- 9 degrees, you need to take off the servo and reinstall it by one tooth. For example, what used to be +9 (out of bound) should become -3 or something.

You can switch between rest, stand, walk to test the effect of calibration. If it's good, remember to click "save" to save the calibration offsets. Otherwise, click return on the top-left corner to abandon the calibration.

Use the control panel

In the control panel, you can use the pe-set buttons to control the robot.



Gaits

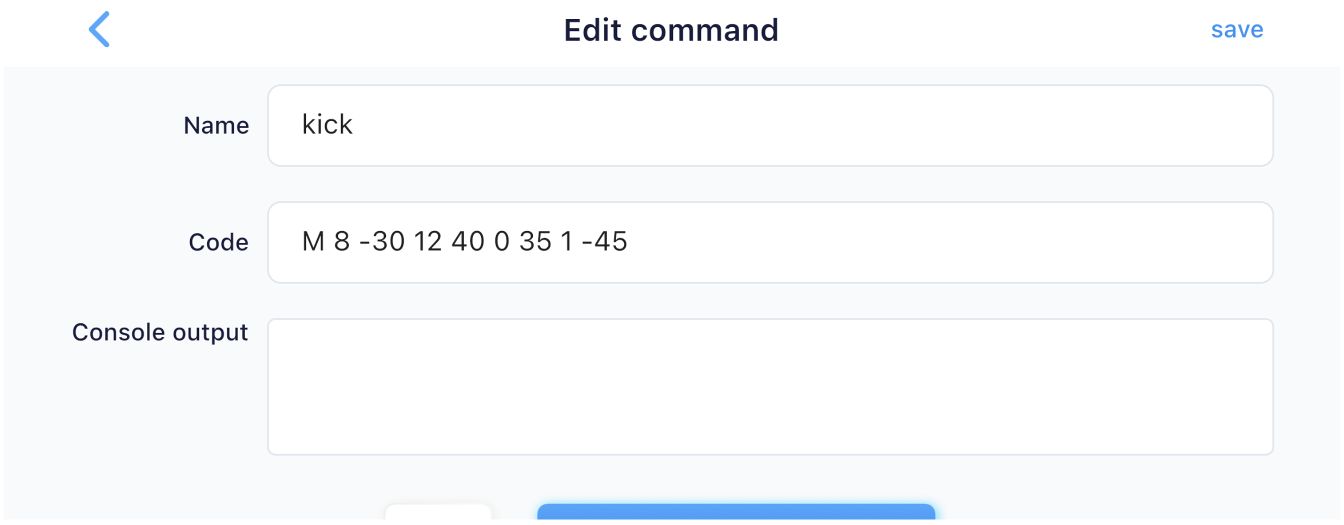
The left panel sets both the robot's gaits and directions and send combined command, such as "walk left" and "trot forward". The robot will only move if an initial gait and direction are selected. The "step" has no direction, and "backward" has left and right directions. The pause button "||" will pause the robot's motion and turn off the servos, so that you can rotate the joints to any angle. The "Turbo" button ( ) turns on/off the gyro, a sensor to detect the robot's body orientation. Turning it on will make the robot keep adjusting to body angles, and will know when it's upside down. Turning it off will reduce calculation and make it walk faster and more stable.

Postures and behaviors

The built-in postures and behaviors can be triggered by pressing the buttons. Don't press the button too frequently and repeatedly. Allow some time for the robot to finish its current tasks.

Customized buttons

You can also define customized commands by pressing the "+" button. Long-press a custom command button to edit it. There's a lite serial console to test the command and even configure the robot.



Name	kick
Code	M 8 -30 12 40 0 35 1 -45
Console output	

Custom commands to try:

* move head (move joint angle)

m0 45

* move head left and right (move joint1 angle1 joint2 angle2 .... The angle is -127~128)

m0 -70 0 70

* sit

ksit

* move joints one by one

m 0 -70 0 70 8 -30 8 45

* MOVE joints simutanuosly

M 0 -45 8 -30 12 -60

* show current joint angles

j

* long meow once

u0 1

* short meow three times

u2 20

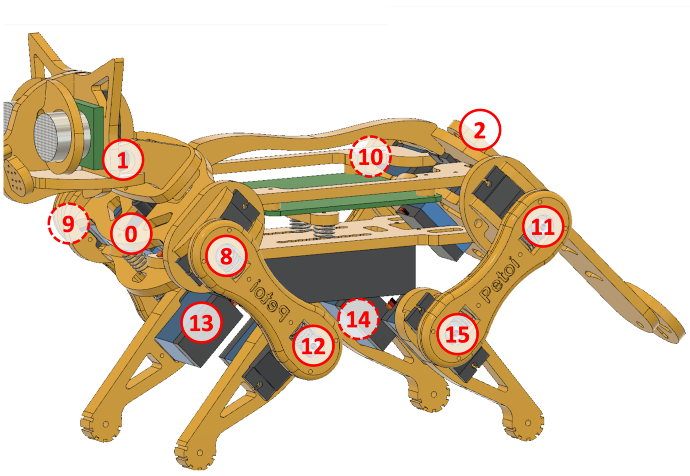
* play a short tone (beep tone duration, duration is 0~256)

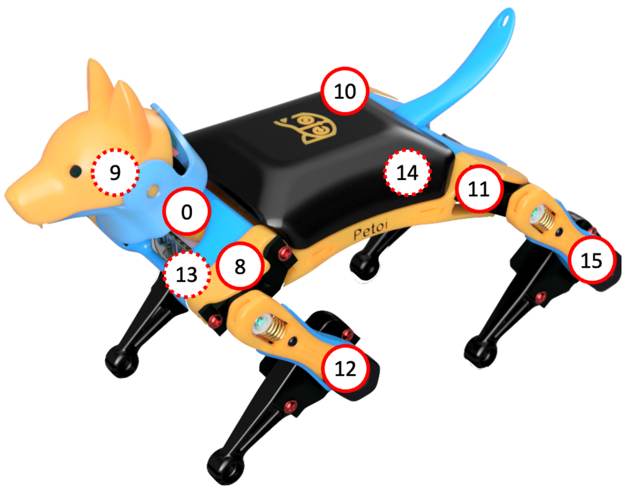
b12 100

* play a melody (beep tone1 duration1, tone2 duration2, tone3 duration3, ... only 64 characters are allowed)

b14 90 14 90 21 90 21 90 23 90 23 90 21 180

Below are the indexes of the joints for your reference. Observe the patterns of the ordering and try to remember them.





A more detailed table can be found in the user manual: https://bittle.petoi.com/7-play-with-bittle

Updates and support

We keep improving the app and will inform you of the updates when available. Please write to support@petoi.com if you have any questions about the app.

serialMaster User Guide

How to use python scripts to have fun with Nybble or Bittle

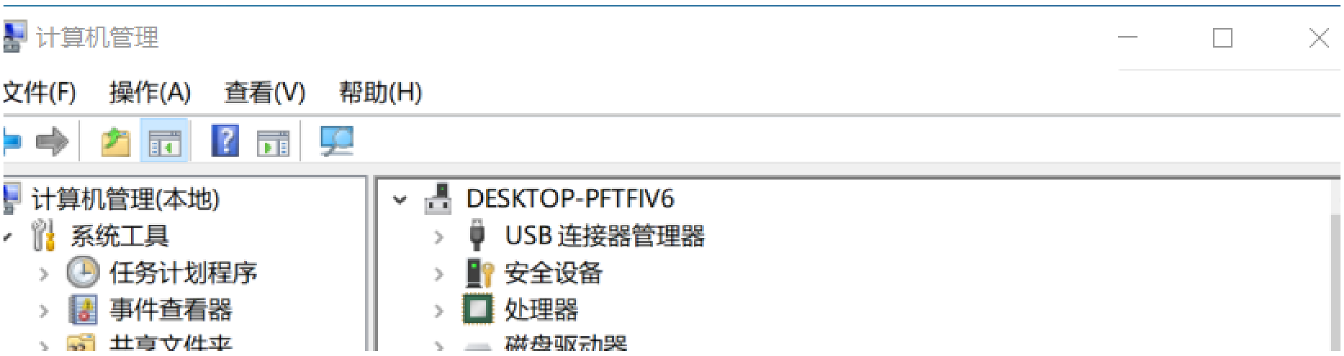
Preparation

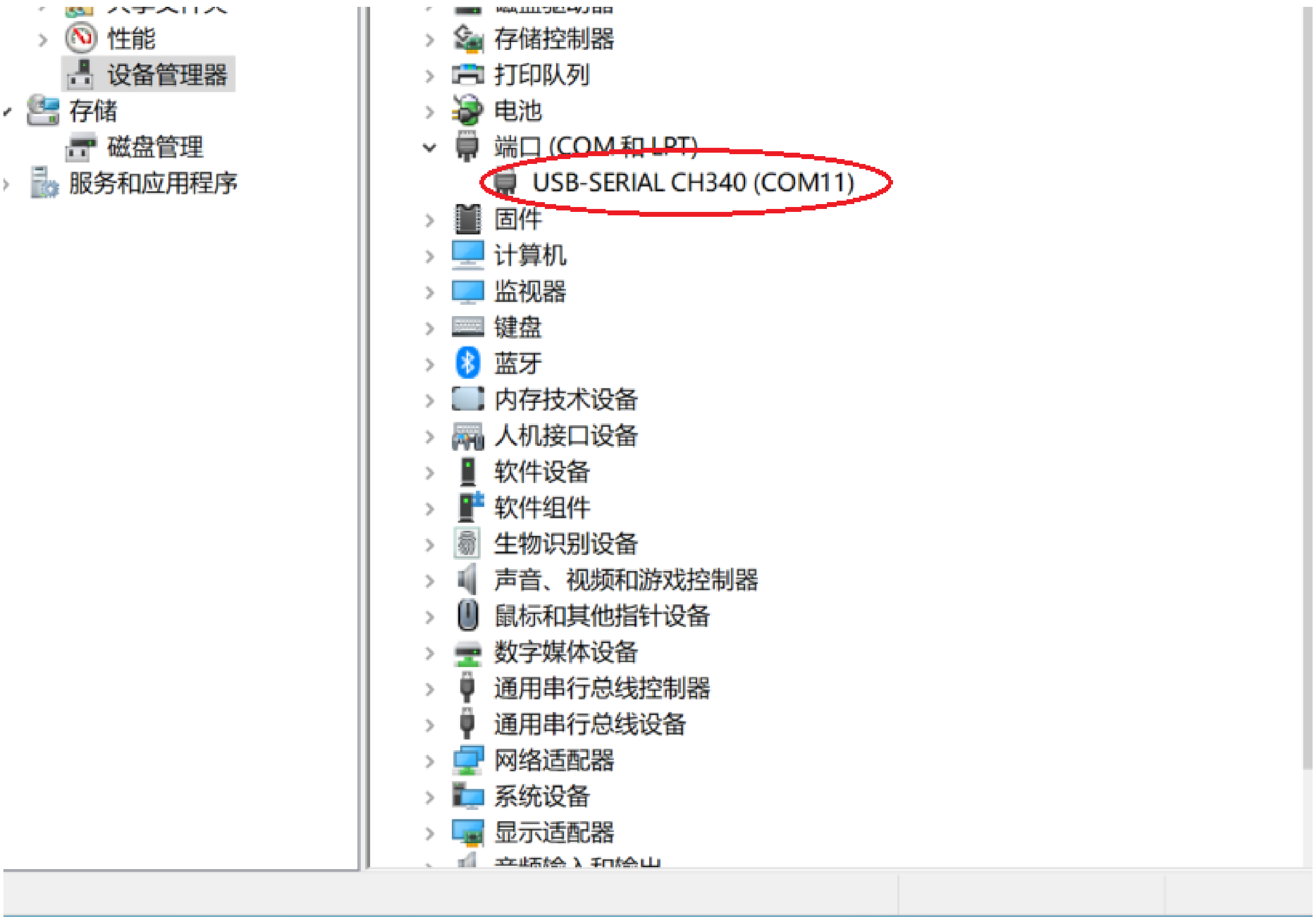
1. Install python (version≥ 3.6, such as Anaconda3-5.2.0-Windows-x86_64.exe)
2. Install pyserial library (version 3.5)

Run script

The script program will automatically identify the serial port number at the very beginning of the run, and complete the connection.

Generally, when using a USB adapter to connect to Nybble (or Bittle), there is only one serial port number:

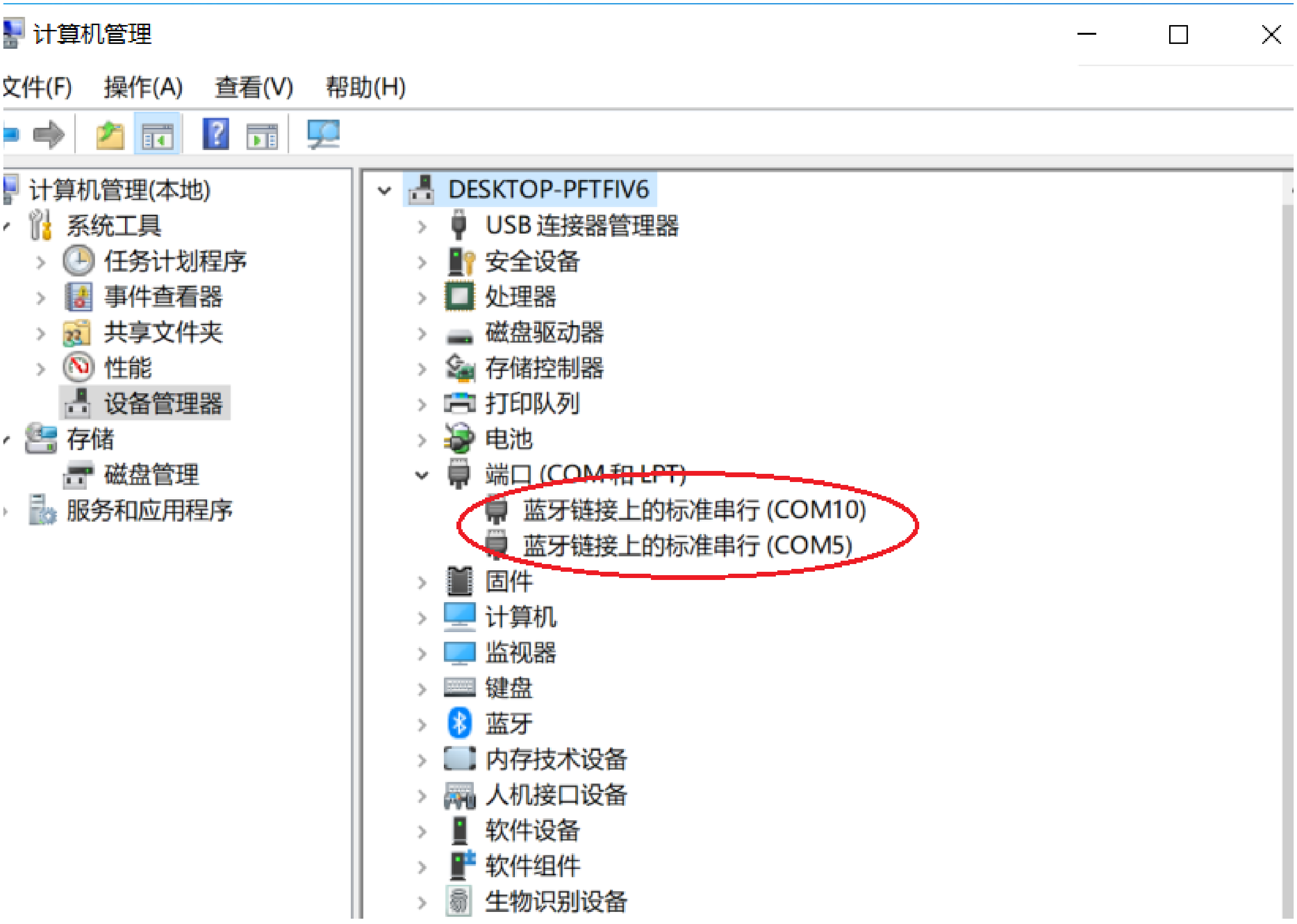



性能
设备管理器
存储
磁盘管理
服务和应用程序

存储控制器
打印队列
电池
端口 (COM 和 LPT)
USB-SERIAL CH340 (COM11)
固件
计算机
监视器
键盘
蓝牙
内存技术设备
人机接口设备
软件设备
软件组件
生物识别设备
声音、视频和游戏控制器
鼠标和其他指针设备
数字媒体设备
通用串行总线控制器
通用串行总线设备
网络适配器
系统设备
显示适配器

USB serial port number

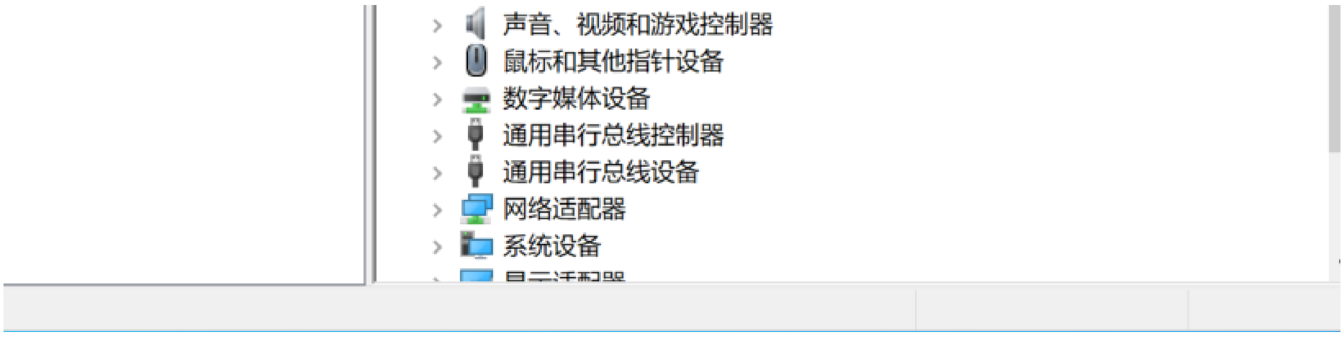
When using the Bluetooth module, there are two serial port numbers:


计算机管理

文件(F) 操作(A) 查看(V) 帮助(H)

计算机管理(本地)
系统工具
任务计划程序
事件查看器
共享文件夹
性能
设备管理器
存储
磁盘管理
服务和应用程序

DESKTOP-PFTFIV6
USB 连接器管理器
安全设备
处理器
磁盘驱动器
存储控制器
打印队列
电池
端口 (COM 和 LPT)
蓝牙链接上的标准串行 (COM10)
蓝牙链接上的标准串行 (COM5)
固件
计算机
监视器
键盘
蓝牙
内存技术设备
人机接口设备
软件设备
软件组件
生物识别设备



Bluetooth serial port number

Open Terminal (such as Anaconda Prompt), enter the path where the script is located (***\serialMaster), you can use the following command to run the script:

Method 1: Run the **ardSerial.py**

```
***\serialMaster>python3 ardSerial.py kbalance
```

Parameters: **kbalance** is a serial port command representing Nybble (or Bittle) skills.

Of course, you can also run this script without any parameters:

```
***\serialMaster>python3 ardSerial.py
```

When the system recognizes that there are multiple serial port numbers, the script will print out the following prompt message at the very beginning of the run:

If there is no response after you input the serial command in the terminal, you should close the terminal first, then change the value of "bluetoothPortIndex" in the ardSerial.py (line:128)to connect to another blue tooth serial port, then reopen the terminal and rerun the script.

When the script formally starts running, the following prompt information is printed out:

You can type 'quit' or 'q' to exit.

Next, you can enter serial port commands in Terminal to control Nybble (or Bittle) to do various interesting actions e.g.

```
Kbalance        # Command to control Nybble(or Bittle) to stand normally
m 0 -30 0 30    # Command to control Nybble(or Bittle) head to swing left and right
```

Method 2: Run a custom script, e.g **example.py**

```
***\serialMaster>python3 example.py
```

The list **testSchedule** in example.py is used to test various serial port commands. Run the following script code to see the execution effect of each serial port command in the list:

```
for task in testSchedule:
    wrapper(task)
```

You can also refer to the content of the **stepUpSchedule** list (in *****\serialMaster\demos\stepup.py**), write a list of behaviors according to your actual needs, and realize your creativity.

Note: When running the scripts under the path of ***\serialMaster\demos,*** you must first use the "**cd demos"** command to enter the path where the scripts are located (\serialMaster\demos), and then use the **python3** command to run the script (e.g. **"python3 stepup.py"**)

Explanation of the serial port commands in the list **testSchedule**:

['kbalance', 2]

- 'kbalance' indicates the command to control Bittle to stand normally
- 2 indicates the postponed time after finishing the command, in seconds

['m', [0, -20], 1.5]

- m indicates the command to control the rotation of the joint servo
- 0 indicates the index number of joint servo
- -20 indicates the rotation angle (this angle refers to the origin, rather than additive) the unit is degree
- 1.5 indicates the postponed time after finishing the command, in seconds. It can be a float number.

['m', ['m', '0', '45', '0', '-45', '0', '45', '0', '-45'], 2]

Using this format, multiple joint servo rotation commands can be issued at one time, and these joint servo rotation commands are executed **SEQUENTIALLY**, not at the same time.

The meaning of this example is: the joint servo with index number 0 is first rotated to the 45 degree position, and then rotated to the -45 degree position, and so on. After these motion commands are completed, the next command will be executed after a 2-second delay.

['M', ['M', '8', '-15', '9', '-20'], 2]

Using this format, multiple joint servo rotation commands can be issued at one time, and these joint servo rotation commands are executed **AT THE SAME TIME.**

The meaning of this example is the joint servos with index numbers 8, 9 are rotated to the -15, -20 degree positions at the same time. After these motion commands are completed, the next command will be executed after a 2-second delay.

['d', 2]

- d indicates the command to put the robot down and shut down the servos
- 2 indicates the postponed time after finishing the command, in seconds

['c', 2]

- c indicates the command to enter calibration mode

- 2 indicates the postponed time after finishing the command, in seconds. After these motion commands are completed, the next command will be executed after a 2-second delay.

['c', [0, -9], 2]

- c indicates the command to enter calibration mode
- 0 indicates the index number of joint servo
- -9 indicates the rotation angle, the unit is degree
- 2 indicates the postponed time after finishing the command, in seconds

Using this format, you can enter the calibration mode to calibrate the angle of a certain joint servo. **Note**: If you want the correction value in this command to take effect, you need to enter the "**s"** command after executing this command.

The meaning of this example: the joint servo with serial number 0 rotates -9 degrees. After these motion commands are completed, the next command will be executed after a 2-second delay.

['i', [8, 50, 9, 50, 10, 50, 11, 50, 0, 0], 3]

- i indicates the command to rotate multiple joint servos at the same time
- 8, 9, 10, 11, 0 indicate the index numbers of joint servos
- 50, 50, 50, 50, 0 indicate the rotation angle (this angle refers to the origin, rather than additive), the unit is degree
- 3 indicates the postponed time after finishing the command, in seconds

['I', [20, 0, 0, 0, 0, 0, 0, 0, 45, 45, 45, 45, 36, 36, 36, 36], 5]

- I indicates the command to control all joint servos to rotate at the same time (currently the command supports 16 degrees of freedom, that is, 16 servos)
- 20,0,0,0,0,0,0,0,45,45,45,45,36,36,36,36 indicate the rotation angle of each joint servo corresponding to 0-15 (this angle refers to the origin, rather than additive), the unit is degree
- 5 indicates the postponed time after finishing the command, in seconds

['b', [10, 255], 2]

- b indicates the command to control the buzzer to beep
- 10 indicates the music tone
- 255 indicates the lengths of duration, the value range is 0~255
- 2 indicates the postponed time after completing the pronunciation, in seconds

['b', ['b', '14', '90', '14', '90', '21', '90', '21', '90', ‘23', '90', '23', '90', '21', '180'], 5]

- b indicates the command to control the buzzer to beep
- '14', '14', '21', '21', ‘23', '23', '21' indicate the music tones
- '90', '90', '90', '90', '90', '90', '180' indicates the lengths of duration

- 5 indicates the postponed time after the music melody is played, in seconds

Using this format, multiple tone pronunciation commands can be issued at one time, and a simple melody can be played.

The meaning of this example is: play a simple melody, and delay 5 seconds after the music melody is played.

For the description of other serial port commands, please refer to Chapter 7 of the Petoi Bittle User Manuals:

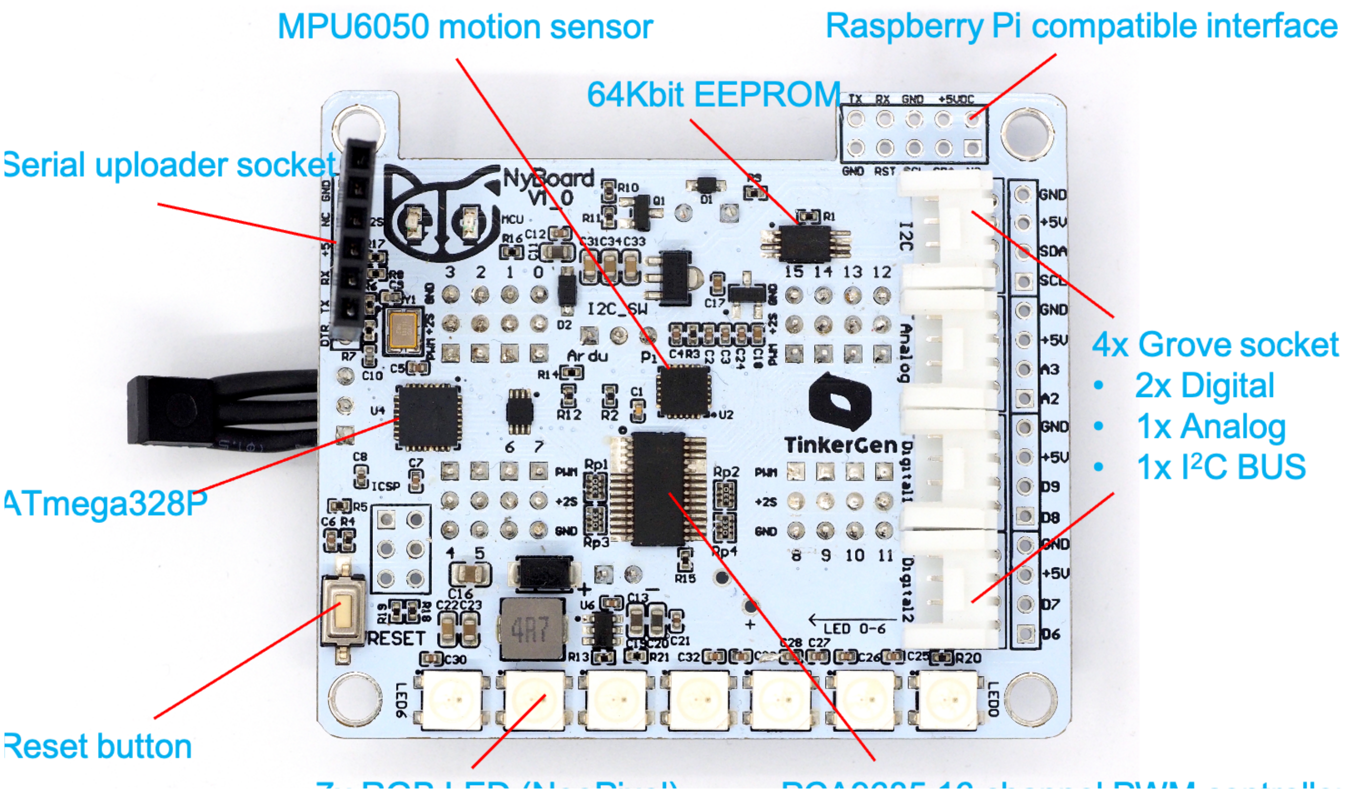
7 ▯ Play with Bittle
Petoi Bittle Manuals

Please help Nybble and Bittle find their sparks. Wish you have fun! ▯

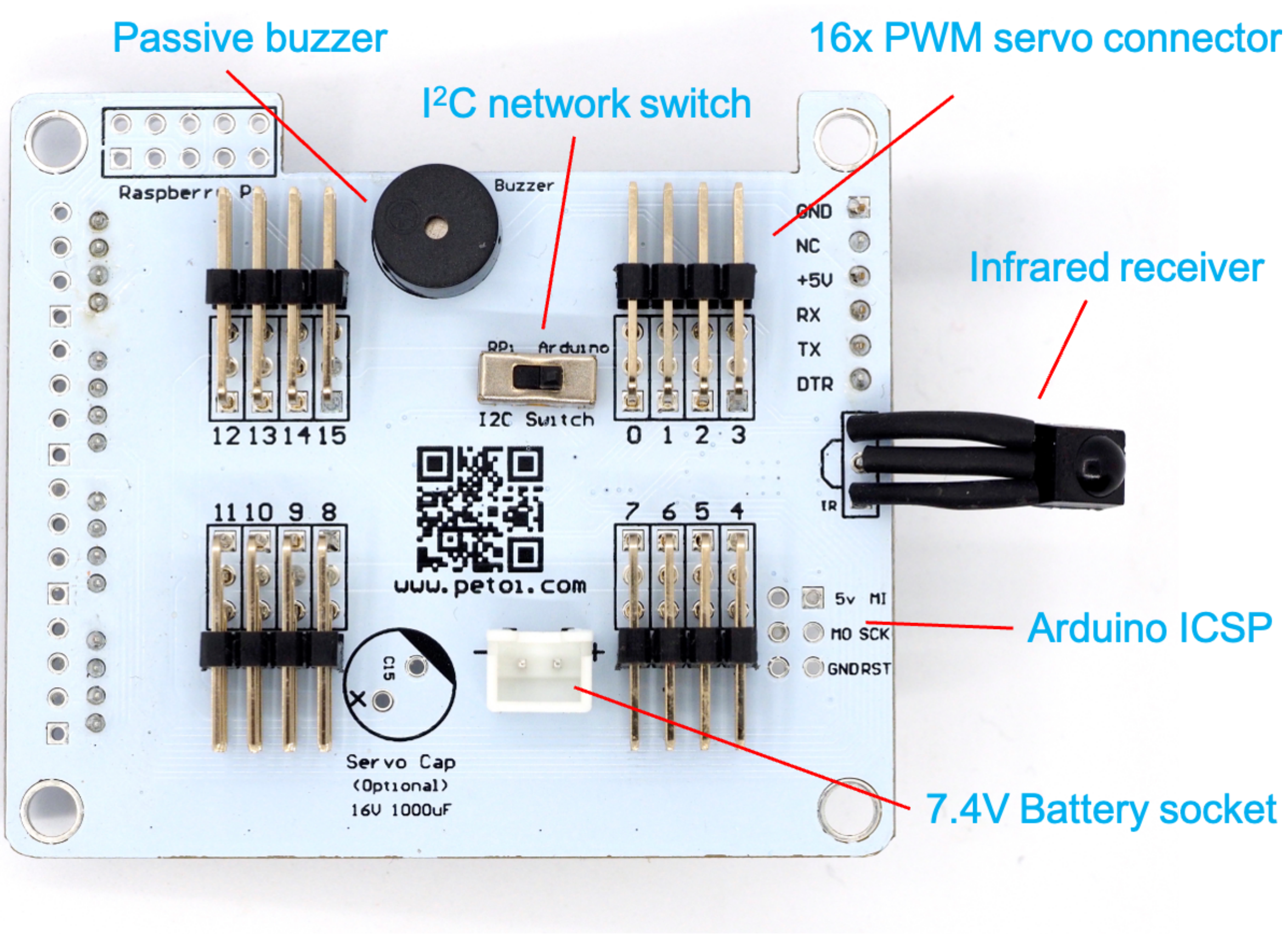
NYBOARD

NyBoard V1_0

Overview



Top side



Bottom side

NyBoard V1 is an upgraded version considering the users' feedback on NyBoard V0. It's compatible with previous versions, yet has some new design to make it easier to use.

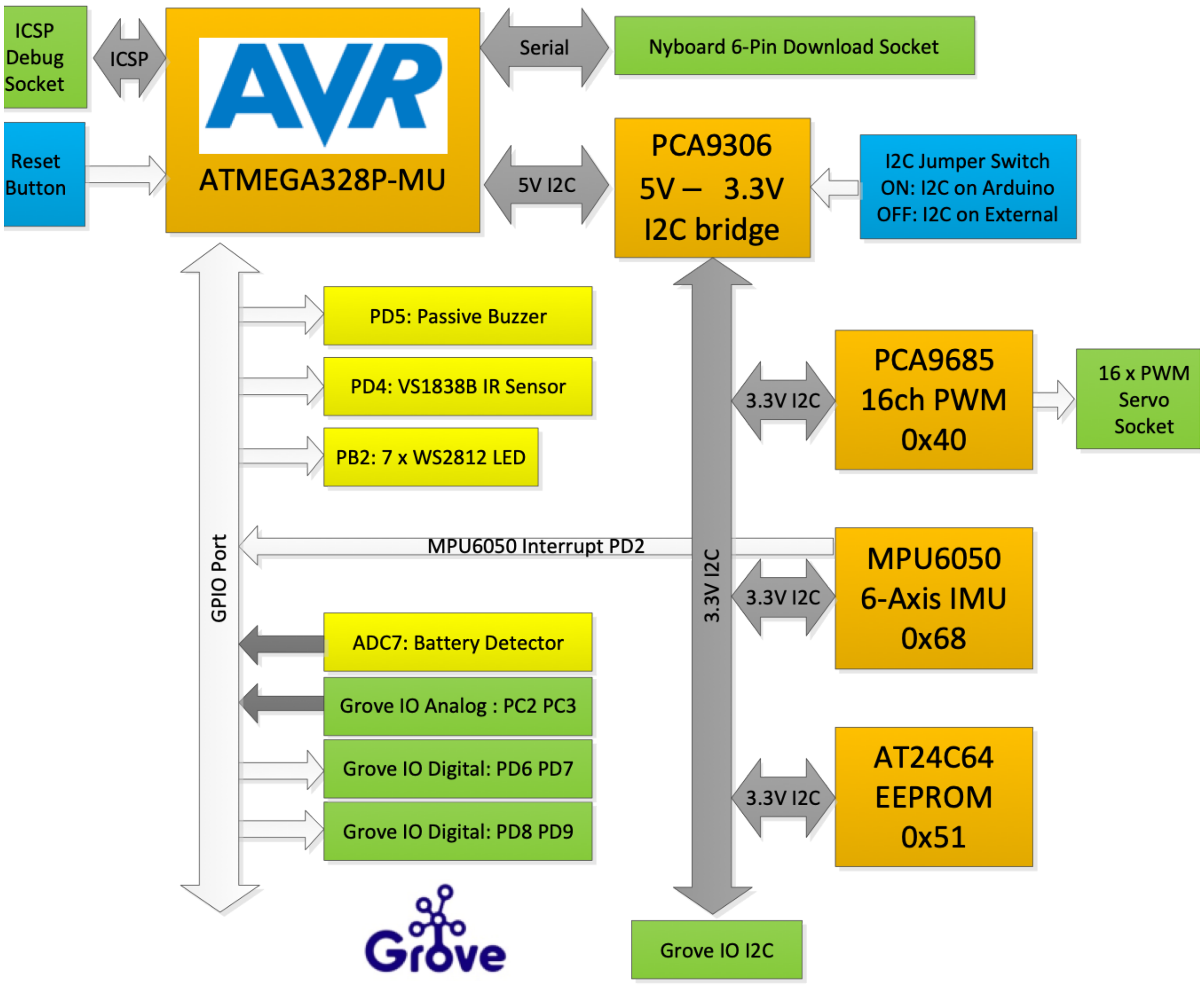
- It still uses Atmel ATMega328P as the main chip but adopts 16MHz without accelerating it to 20MHz. Now the board is fully compatible with Arduino Uno, much easier for new users to Arduino.
- It keeps driving 16 PWM channels with PCA9685. The pin order is altered, but you don't even need to read the indexes on the board, because the pin mapping is handled within the software.
- Now the 6-axis motion sensor MPU6050 is designed on the PCB, rather than a stand-alone module soldered above the board. It supports a built-in DMP (Digital Motion Processor) to calculate the motion data, as well as providing raw data for your own fusion and filtering algorithms.
- It continues to use an 8KB onboard I2C EEPROM to save constants for skills.
- The power system is redesigned to provide a more stable supply. The structure for peripherals is also optimized.
- From Jan 1st, 2021, We start to include an official Bluetooth dongle for wirelessly uploading and communication. The default baud rate for all the communication ports is set to be 115200.
- The reset button is more accessible on the back of the board.
- We added 4 Grove socket to plug-and-play Seeed Studio's extensible modules. We still provide

standard 2.54mm through-holes besides the socket.

- We added 7 WS2812 RGB LEDs on the board as another form of output and status indicator.
- The socket for the battery is now anti-reverse.

Logic diagram of the controller

The configuration of NyBoard V1_0 is shown as below:



Introduction to the onboard components

Main controller

NyBoard V1_0 uses Atmel ATMega328P-MUR as the main controller. We adopted its smaller version of QFN32 for better layout, and it's near-identical to regular TQFP32.

ATMega328P works at 16MHz with a 5V supply. It has 2KB SRAM, 32KB Flash, and 1KB on-chip

I2C switch

The main chip runs at 5V, while the other peripherals run at a 3.3V logic level. We use PCA9306 to convert the I2C bus of ATMega328P to 3.3V. We also added an I2C switch on the bus. By dialing it to "Arduino" or "Raspberry Pi", you can change the I2C master of the onboard peripherals.

6-Axis IMU MPU6050

MPU6050 is widely used in many DIY projects to acquire the motion state of devices. It detects the 3 acceleration and 3 angular motion states. It also includes a DMP to calculate the state directly, without using the main controller's computational resources.

On NyBoard V1_0, its I2C address is 0x68. The interrupt pin is connected to the PD2 port of ATMega328P (or the D2 pin of Arduino Uno).

There are a lot of available MPU6050 libraries and we are using I2CDev/6050DMP. You can also use other versions:

Name	Author	Feature
I2Cdev	jrowberg	built-in DMP
Adafruit MPU6050	Adafruit	standard MPU6050 library
Kalman Filter	TKJ Electronics	with Kalman filter

PCA9685 and the PWM servo ports

PCA9685 fans out 16 PWM 12-bit channels with instructions from the I2C port. Its address is set to 0x40. There are 16 PWM indexes printed on the PCB, but you don't really need to read them because the pin-mapping is done in the software. The physical wiring pattern is the same as the previous boards. You do need to check the direction of the servo pins. Regular servos have 3 pins for PWM, power(2S), and ground (GND). The ground should connect to the black wire of the servo.

On NyBoard V1_0, the servos' power connects to the 2S Li-ion battery. We designed our servos to be compatible with 8.4V input. Regular servos usually run at 6V. You should not connect regular 9g servos like the SG90 to the board directly.

We use Adafruit PWM Servo Driver Library for PCA9685.

EEPROM

We save the motion skills with an 8KB onboard I2C EEPROM AT24C64. Its I2C address is 0x54. The lookup table of skills is saved in the 1KB on-chip EEPROM of ATMega328P. It uses <EEPROM.h>. You need to pay attention to their differences when developing new codes.

Passive buzzer

The buzzer is driven by PD5 (or the D5 of Arduino UNO). The current is amplified by 2N7002 MOS.

Infrared receiver

We use VS1838B as the Infrared receiver, connected to PD4 (or D4 on Arduino Uno). It's driven by the IRremote library of Arduino, the corresponding remote is encoded in NEC format. You may disable the other protocols in IRremote.h to save Flash (about 10%!)

Voltage detector

The two LEDs in the Petoi logo indicates the powering state of the board. The left eye is blue for the logic chips. The right eye is yellow for the servos' power. When NyBoard is connected to the battery, both LEDs should lit up. When NyBoard is powered by the USB downloader, only the blue LED will lit up.

There's an anti-reverse socket for the battery. The battery's output is connected to ADC7 (or A7 of Arduino Uno) and is not threaded to an open pin. ADC7 collects the voltage over a voltage divider. The actual voltage is approximately 2x of the reading. A safe range of battery voltage is below 10V.

$$V oltage_{real} = \frac{ADC_{reading}}{1024} \times 5.0 \times 2$$

You should charge the battery in time when the battery is lower than 7.4V.

WS2812 RGB LED

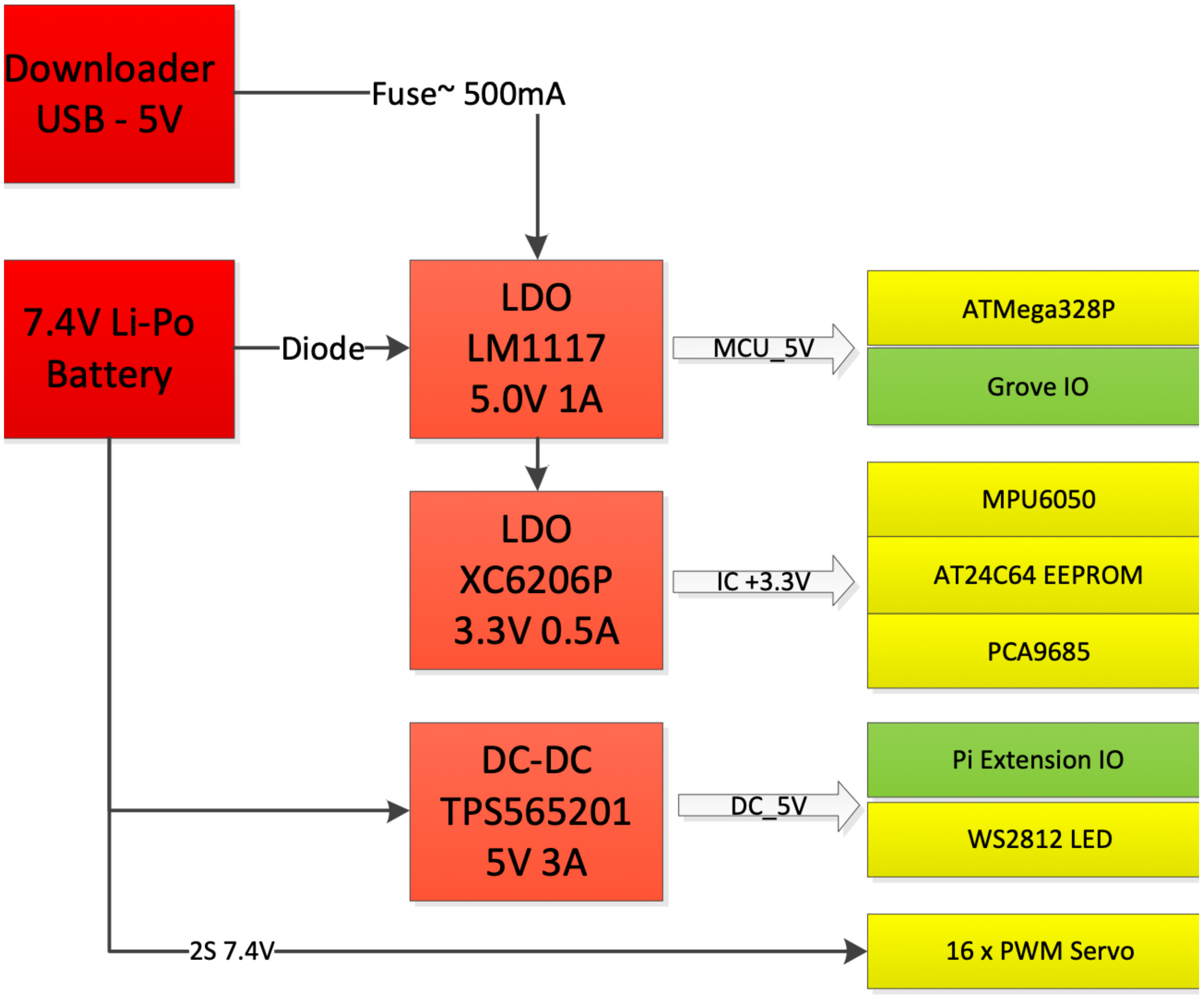
We added 7 WS2812 RGB LEDs (or the NeoPixel) on the NyBoard. The pin number is D10. They are powered by the 5V DC-DC power chip for Raspberry Pi and are independent of the 5V network of ATMega328P. So you need to plug in the battery to power the LEDs.

Grove sockets

We adopted the Grove sockets for convenient plug-and-play connections. There are three types of socket:

Grove Socket	Pin Number	Function
G1	I2C: SCL, SDA	I2C with 3.3V logic signal
G2	A2, A3	Analog input; 0-5V
G3	PD8, PD9	Digital I/O; 0-5V
G4	PD6, PD7	Digital I/O; 0-5V

Power system



The main chips are powered by a Low-dropout (LDO) linear regulators for noise removal and better stability. We use LM1117-5V and XC6206P-3.3V to power 5V and 3.3V chips. The 3.3V LDO is connected in serial after the 5V LDO for better efficiency.

There's a diode between the battery and LM1117-5V to prevent damage by the wrong connection. There's a self-recover fuse (6V 500mA) on the USB uploader to limit the current and protect the USB port.

The Raspberry Pi consumes much more power, so we choose TPS565201 DC-DC to provide a 5V 3A output. The peak output can be 5A and with high-temperature/current/voltage protection. It will cut off the power when the chip keeps outputting >4A and over 100 Celcius degrees until the temperature drops to normal. The WS2812 RGB LEDs are also powered by this DC-DC source.

The servos are powered by 2S Li-ion batteries directly. Pay attention not to short connect the power or any pins on the NyBoard.

Last updated: Jan 13, 2021

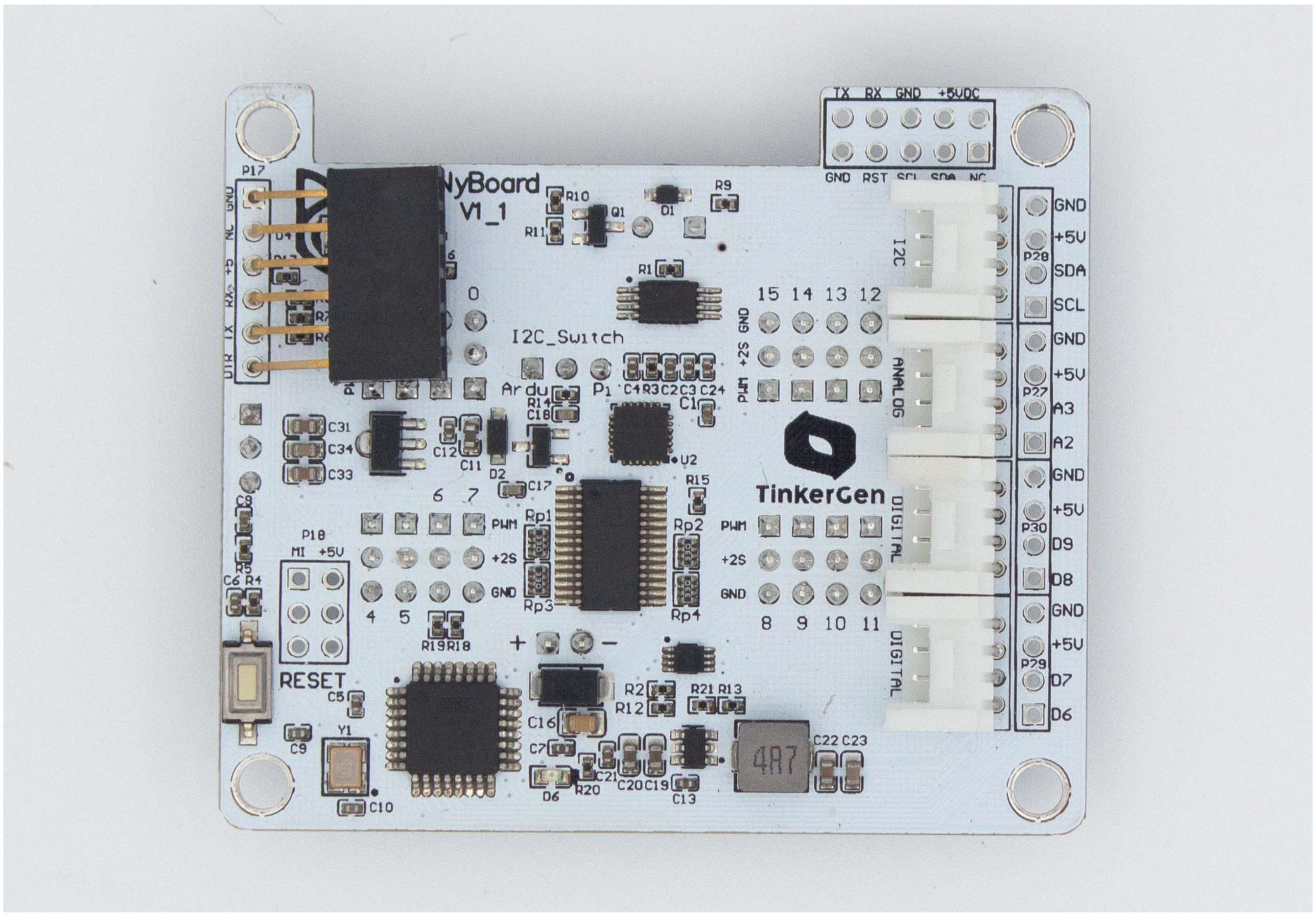
NyBoard V1_1

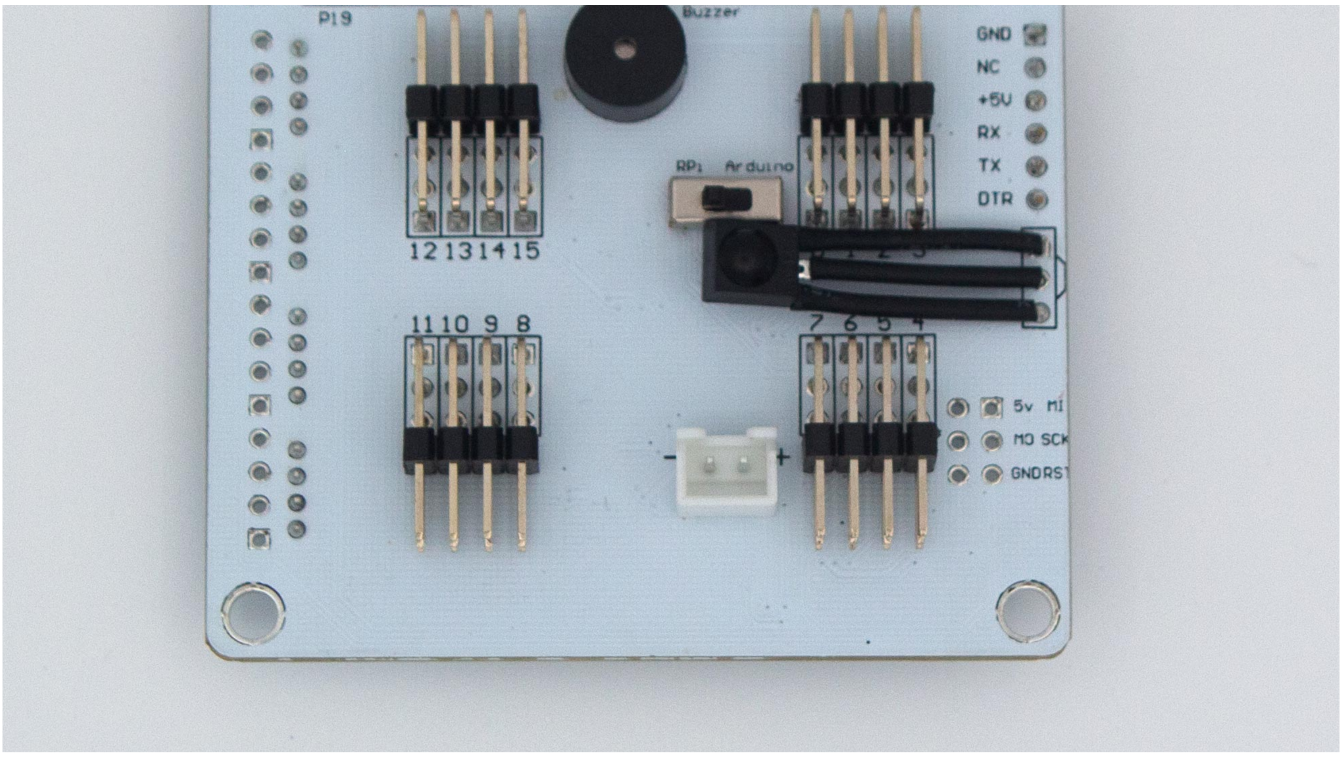
Update：

NyBoard V1_1 is a V1 refreshed version mainly focused on the shortage of the ATMEGA328P-MU in our supply chain.

1. Replace the ATMEGA328P-MU（QFN32=）with the ATMEGA328P-AU（TQFP32）
2. Removed 7 WS2812 LEDs to optimize the area.
3. A green LED is connected to the D10 port with PWM functions.
4. There's no changes of sockets and pin definitions from V1_0, the bootloader and the OpenCat sketchs is fully compatible.

Overview



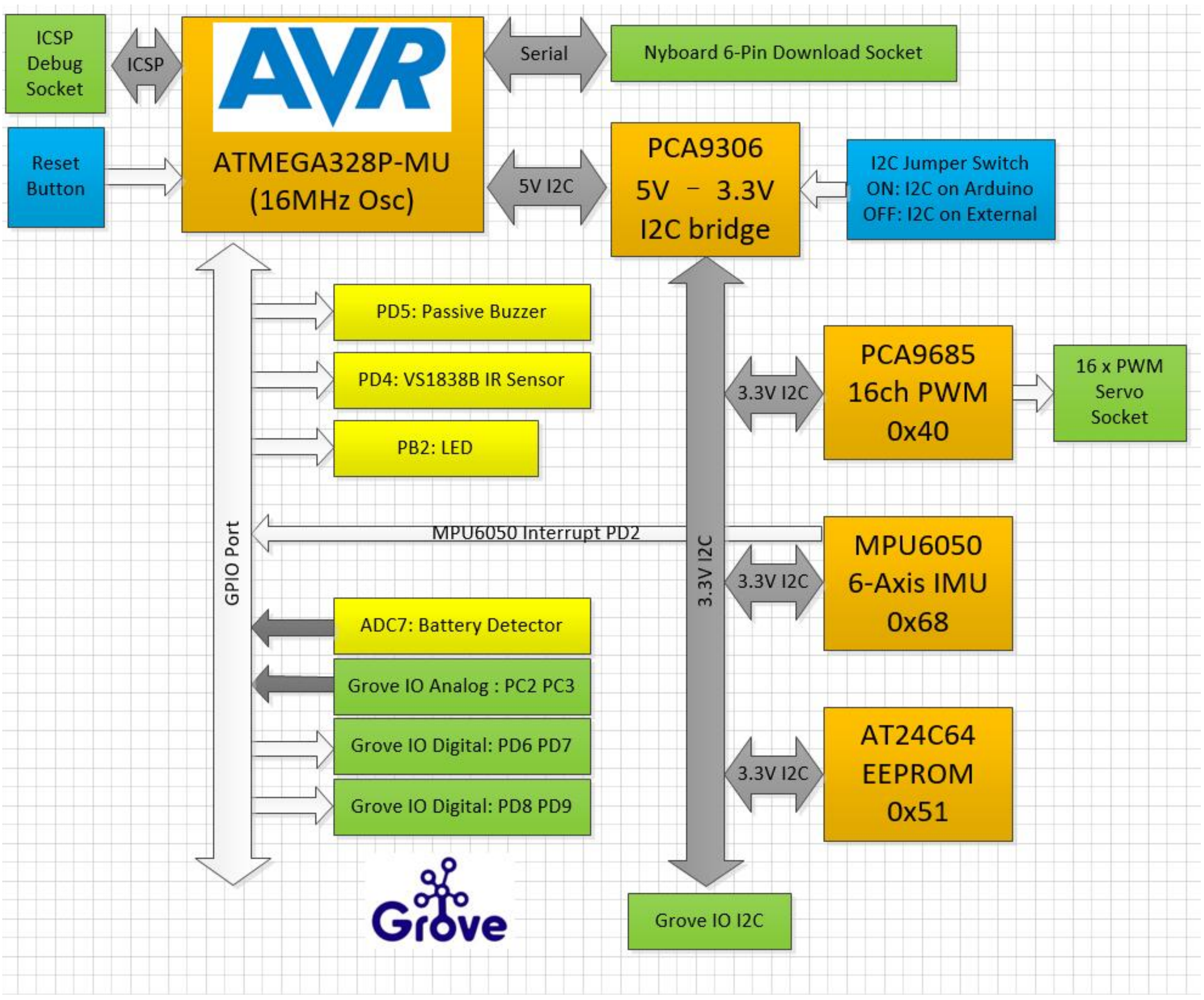


NyBoard V1 is an upgraded version considering the users' feedback on NyBoard V0. It's compatible with previous versions, yet has some new design to make it easier to use.

- It still uses Atmel ATMega328P as the main chip but adopts 16MHz without accelerating it to 20MHz. Now the board is fully compatible with Arduino Uno, much easier for new users to Arduino.
- It keeps driving 16 PWM channels with PCA9685. The pin order is altered, but you don't even need to read the indexes on the board, because the pin mapping is handled within the software.
- Now the 6-axis motion sensor MPU6050 is designed on the PCB, rather than a stand-alone module soldered above the board. It supports a built-in DMP (Digital Motion Processor) to calculate the motion data, as well as providing raw data for your own fusion and filtering algorithms.
- It continues to use an 8KB onboard I2C EEPROM to save constants for skills.
- The power system is redesigned to provide a more stable supply. The structure for peripherals is also optimized.
- From Jan 1st, 2021, We start to include an official Bluetooth dongle for wirelessly uploading and communication. The default baud rate for all the communication ports is set to be 115200.
- The reset button is more accessible on the back of the board.
- We added 4 Grove socket to plug-and-play Seeed Studio's extensible modules. We still provide standard 2.54mm through-holes besides the socket.
- The socket for the battery is now anti-reverse.

Logic diagram of the controller

The configuration of NyBoard V1_0 is shown as below:



Introduction to the onboard components

Main controller

NyBoard V1_0 uses Atmel ATMega328P-AU, the same MCU of the Arduino Nano (UNO Compatible).

The ATMega328P works at 16MHz with a 5V supply. It has 2KB SRAM, 32KB Flash, and 1KB on-chip EEPROM. With the same bootloader of Arduino Uno, you can upload sketches through the serial port.

LED (NEW!)

The WS2812 serial RGB LEDs are replaced by a single green LED. You can easily use it with standard Arduino GPIO control commands.

I2C switch

The main chip runs at 5V, while the other peripherals run at a 3.3V logic level. We use PCA9306 to convert the I2C bus of ATMega328P to 3.3V. We also added an I2C switch on the bus. By dialing it to "Arduino" or

"Raspberry Pi", you can change the I2C master of the onboard peripherals

6-Axis IMU MPU6050

MPU6050 is widely used in many DIY projects to acquire the motion state of devices. It detects the 3 acceleration and 3 angular motion states. It also includes a DMP to calculate the state directly, without using the main controller's computational resources.

On NyBoard V1_0, its I2C address is 0x68. The interrupt pin is connected to the PD2 port of ATMega328P (or the D2 pin of Arduino Uno).

There are a lot of available MPU6050 libraries and we are using I2CDev/6050DMP. You can also use other versions:

Name	Author	Feature
I2Cdev	jrowberg	built-in DMP
Adafruit MPU6050	Adafruit	standard MPU6050 library
Kalman Filter	TKJ Electronics	with Kalman filter

PCA9685 and the PWM servo ports

PCA9685 fans out 16 PWM 12-bit channels with instructions from the I2C port. Its address is set to 0x40. There are 16 PWM indexes printed on the PCB, but you don't really need to read them because the pin-mapping is done in the software. The physical wiring pattern is the same as the previous boards. You do need to check the direction of the servo pins. Regular servos have 3 pins for PWM, power(2S), and ground (GND). The ground should connect to the black wire of the servo.

On NyBoard V1_0, the servos' power connects to the 2S Li-ion battery. We designed our servos to be compatible with 8.4V input. Regular servos usually run at 6V. You should not connect regular 9g servos like the SG90 to the board directly.

We use Adafruit PWM Servo Driver Library for PCA9685.

EEPROM

We save the motion skills with an 8KB onboard I2C EEPROM AT24C64. Its I2C address is 0x54. The lookup table of skills is saved in the 1KB on-chip EEPROM of ATMega328P. It uses <EEPROM.h>. You need to pay attention to their differences when developing new codes.

Passive buzzer

The buzzer is driven by PD5 (or the D5 of Arduino UNO). The current is amplified by 2N7002 MOS.

Infrared receiver

We use VS1838B as the Infrared receiver, connected to PD4 (or D4 on Arduino Uno). It's driven by the IRremote library of Arduino, the corresponding remote is encoded in NEC format. You may disable the other protocols in IRremote.h to save Flash (about 10%!)

Voltage detector

The two LEDs in the Petoi logo indicates the powering state of the board. The left eye is blue for the logic chips. The right eye is yellow for the servos' power. When NyBoard is connected to the battery, both LEDs should lit up. When NyBoard is powered by the USB downloader, only the blue LED will lit up.

There's an anti-reverse socket for the battery. The battery's output is connected to ADC7 (or A7 of Arduino Uno) and is not threaded to an open pin. ADC7 collects the voltage over a voltage divider. The actual voltage is approximately 2x of the reading. A safe range of battery voltage is below 10V.

$$Voltage_{real} = \frac{ADC_{reading}}{1024} \times 5.0 \times 2$$

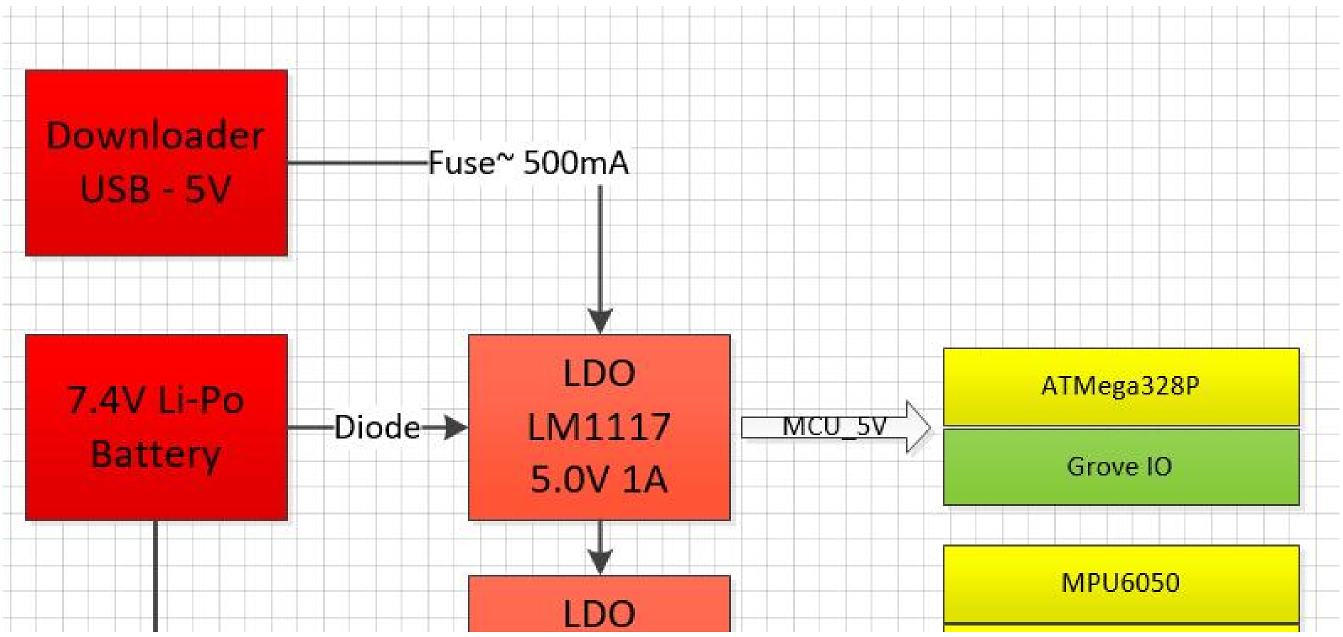
You should charge the battery in time when the battery is lower than 7.4V.

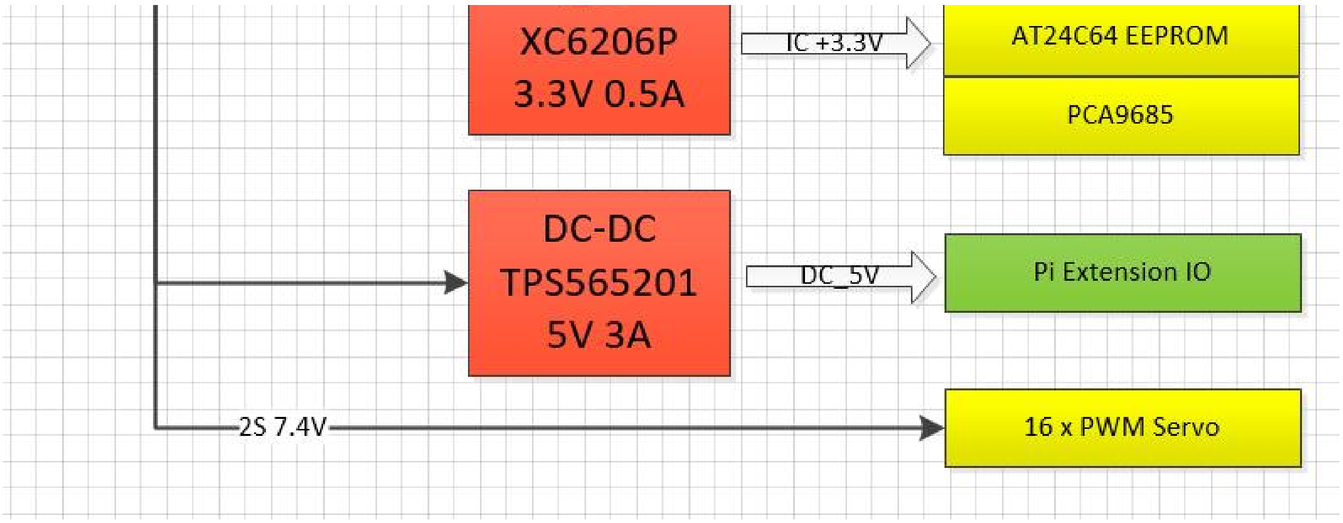
Grove sockets

We adopted the Grove sockets for convenient plug-and-play connections. There are three types of socket:

Grove Socket	Pin Number	Function
G1	I2C: SCL, SDA	I2C with 3.3V logic signal
G2	A2, A3	Analog input; 0-5V
G3	PD8, PD9	Digital I/O; 0-5V
G4	PD6, PD7	Digital I/O; 0-5V

Power system





The main chips are powered by a Low-dropout (LDO) linear regulator for noise removal and better stability. We use LM1117-5V and XC6206P-3.3V to power 5V and 3.3V chips. The 3.3V LDO is connected in serial after the 5V LDO for better efficiency.

There's a diode between the battery and LM1117-5V to prevent damage by the wrong connection. There's a self-recover fuse (6V 500mA) on the USB uploader to limit the current and protect the USB port.

The Raspberry Pi consumes much more power, so we choose TPS565201 DC-DC to provide a 5V 3A output. The peak output can be 5A and with high-temperature/current/voltage protection. It will cut off the power when the chip keeps outputting >4A and over 100 Celcius degrees until the temperature drops to normal.

The servos are powered by 2S Li-ion batteries directly. Pay attention not to short connect the power or any pins on the NyBoard.

BIBOARD

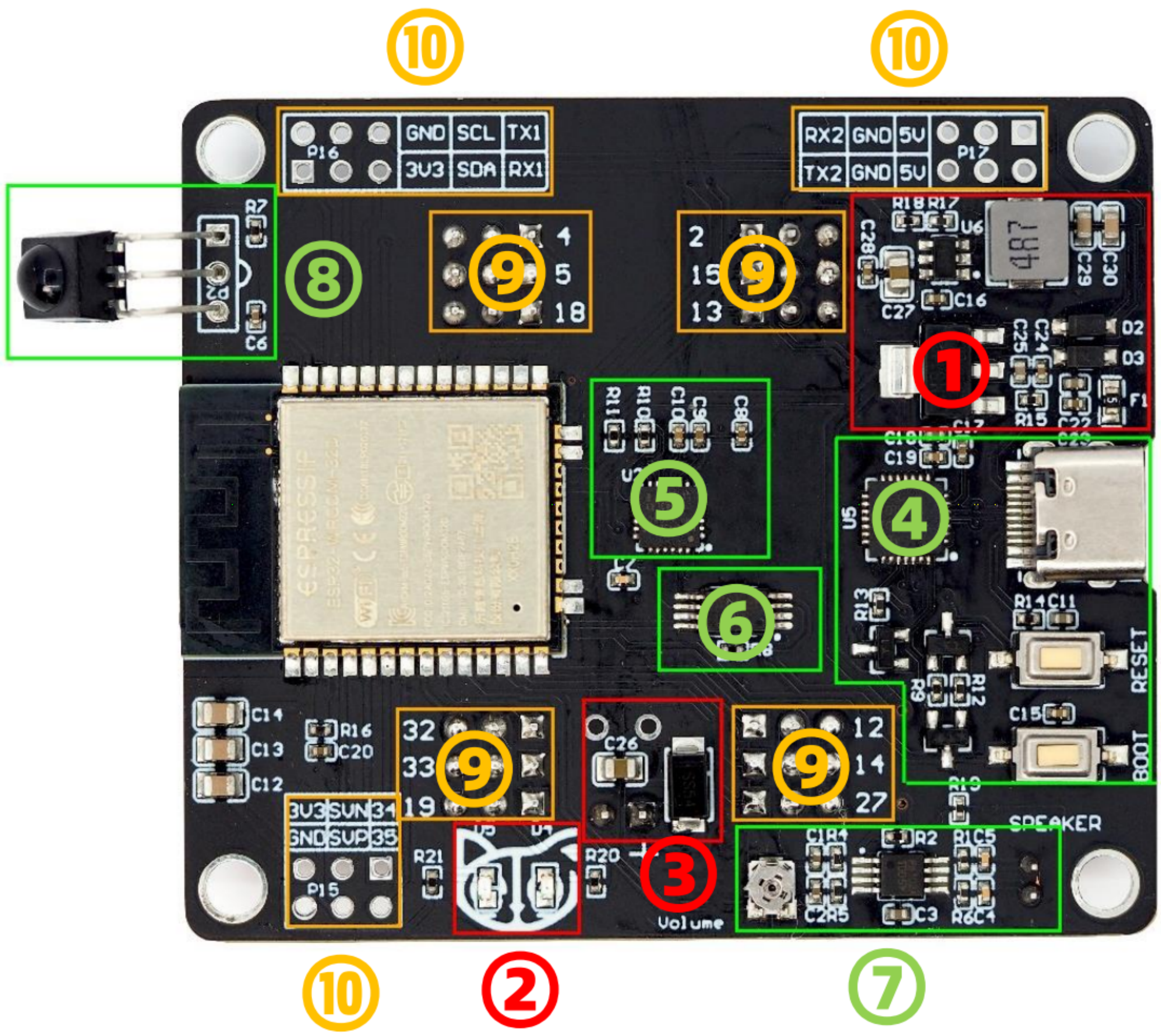
BiBoard Quick Start Guide

Introduction

BiBoard is a robot dog controller based on ESP32 developed by Petoi LLC. Unlike NyBoard for normal users and robot lovers, BiBoard is mainly facing developers and geeks. High-performance processors, larger memory and storage, wireless connections. Audio function is also included.

Modules and functions

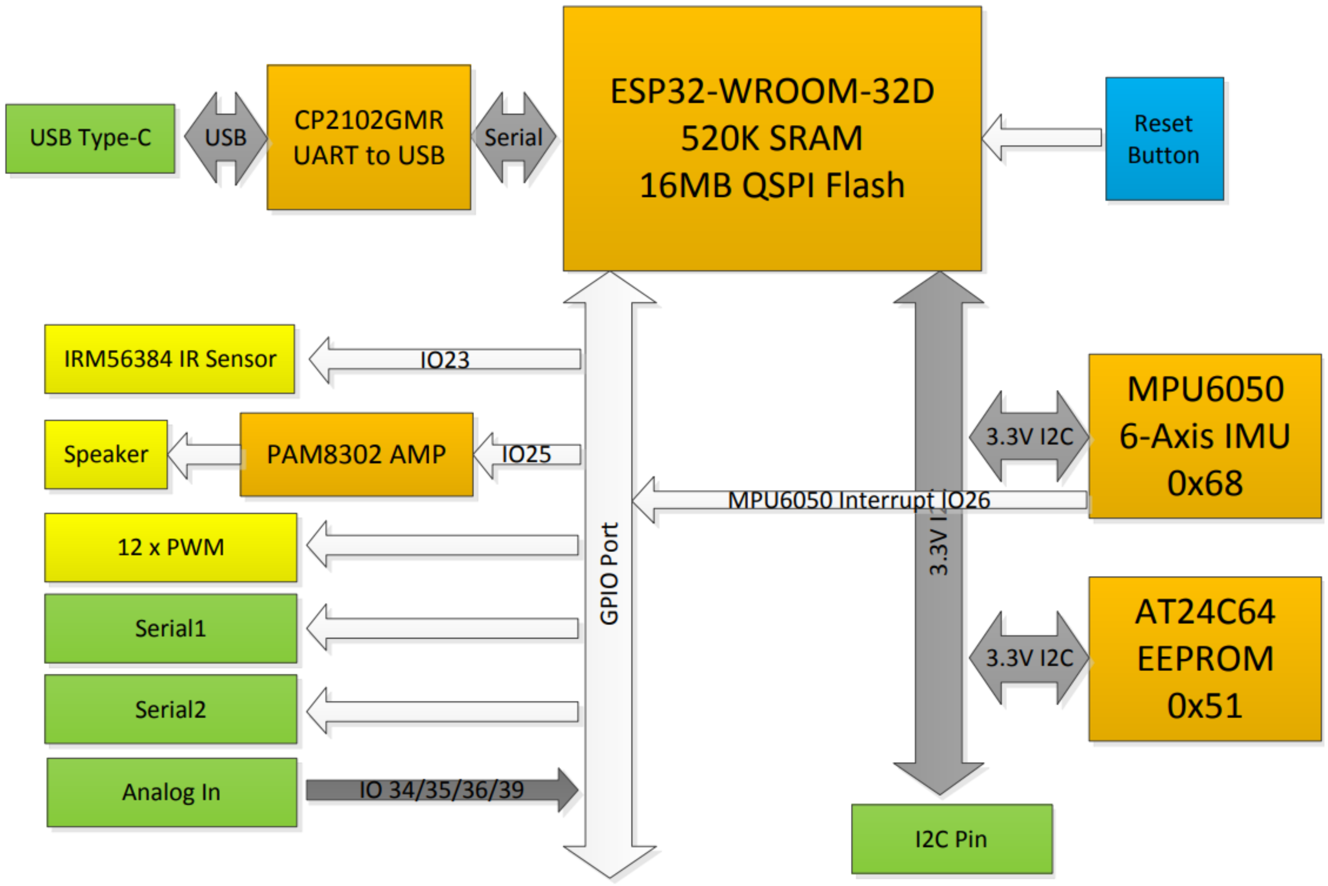
The function partition for BiBoard is shown below:



Part	No.	Module	Introduction
Power	1	Battery socket	Bittle battery or external 8.4V battery
	2	Power LED	Left: blue/5V Right:yellow/8.4V
	3	Power	5V DC-DC / 5V USB / 3.3V LDO
ICs	4	USB Downloader	Sketches download BiBoard automatic reset and boot
	5	IMU	MPU6050 6-Axis IMU with DMP

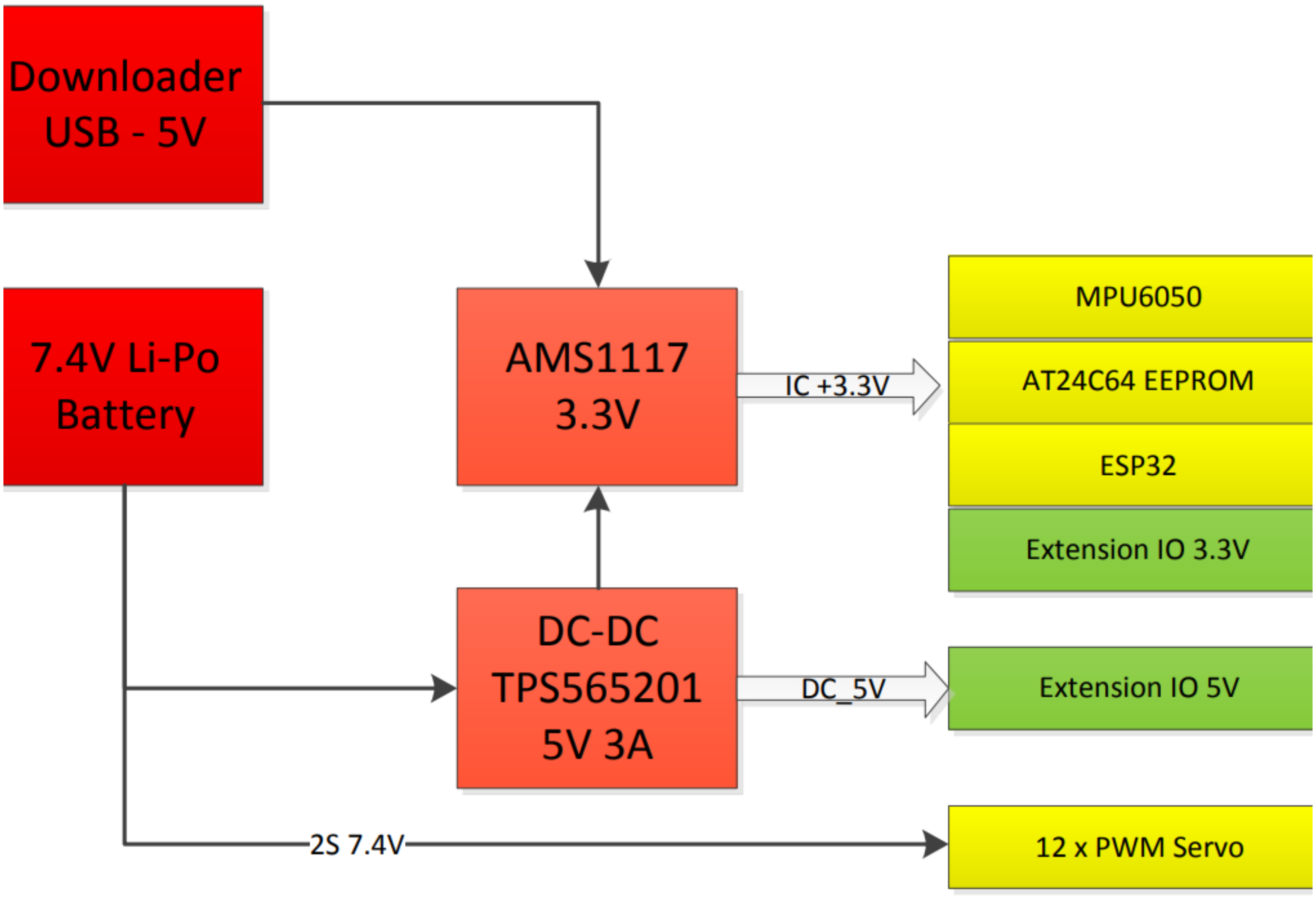
			I2C Addr:0x68, Interrupt pin: IO26
	6	EEPROM	64Kbit I2C EEPROM，I2C Addr: 0x51
	7	DAC Amp	Mono channel Amplifier，IO25
	8	IRDA receiver	IRDA Reciever, IO23
Extension	9	PWM Servo	12 PWM pins by ESP32
	10	3 extensions	4 Analog input, 2 Serials, 1 I2C, 5V DC-DC Power max 3A

Block diagram for BiBoard is shown below:



Module details:

3.1 Power



There're 2 ways to power the BiBoard: USB 5V and battery socket 7.4V.

When using USB power, there's no power output for DC-DC 5V extension and servo. So USB power mainly supplies ICs.

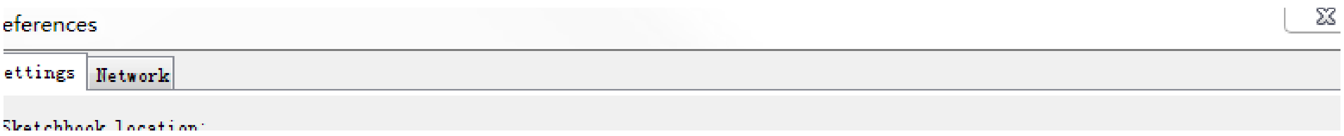
When using battery power at 7.4V (maximum: 8.4V). Both servos and 5V power will be supplied. You can use 5V powering the Raspberry Pi.

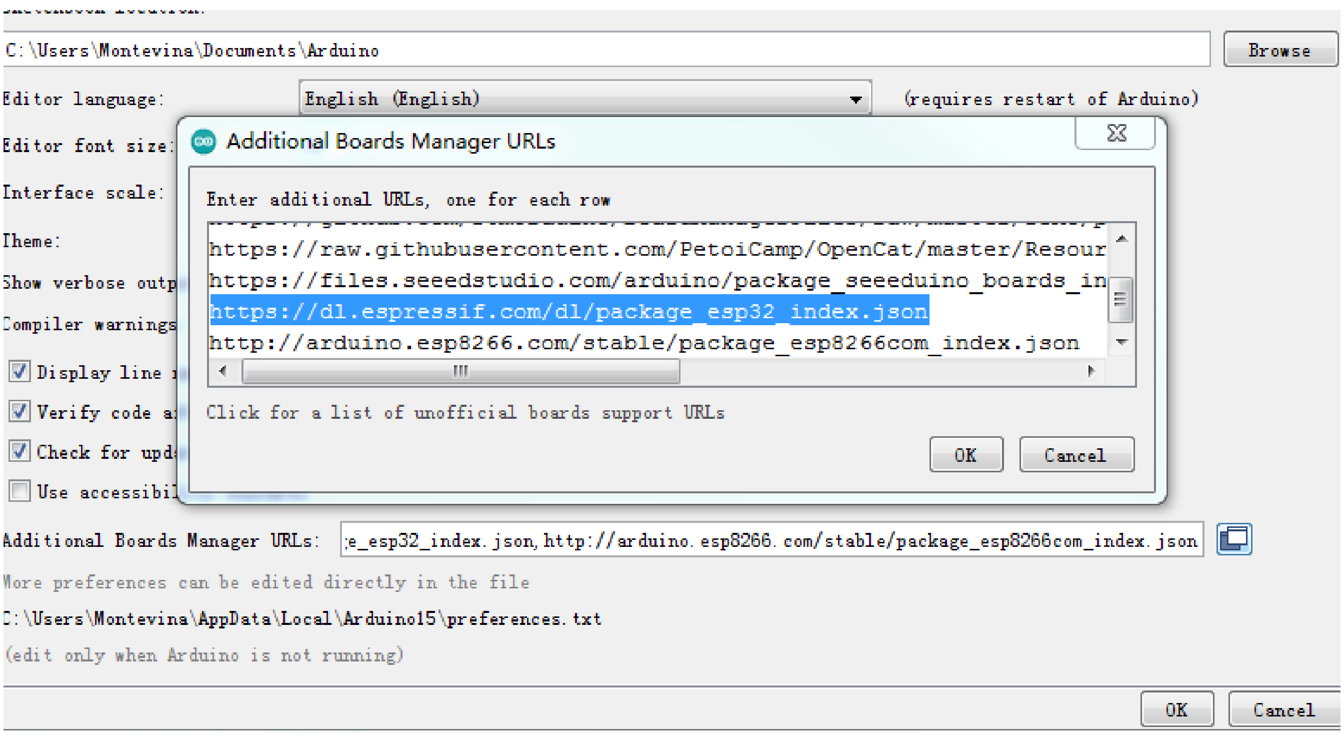
3.2 On board modules

3.2.0 ESP32 development environment set up

Open "Preferences" in Arduino, add ESP32 development board URL:

https://dl.espressif.com/dl/package_esp32_index.json




C:\Users\Montevina\Documents\Arduino

Editor language: English (English) (requires restart of Arduino)

Additional Boards Manager URLs

Enter additional URLs, one for each row

```
https://raw.githubusercontent.com/PetoiCamp/OpenCat/master/Resour
https://files.seeedstudio.com/arduino/package_seeeduino_boards_in
https://dl.espressif.com/dl/package_esp32_index.json
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Click for a list of unofficial boards support URLs

Additional Boards Manager URLs: ;e_esp32_index.json,http://arduino.esp8266.com/stable/package_esp8266com_index.json

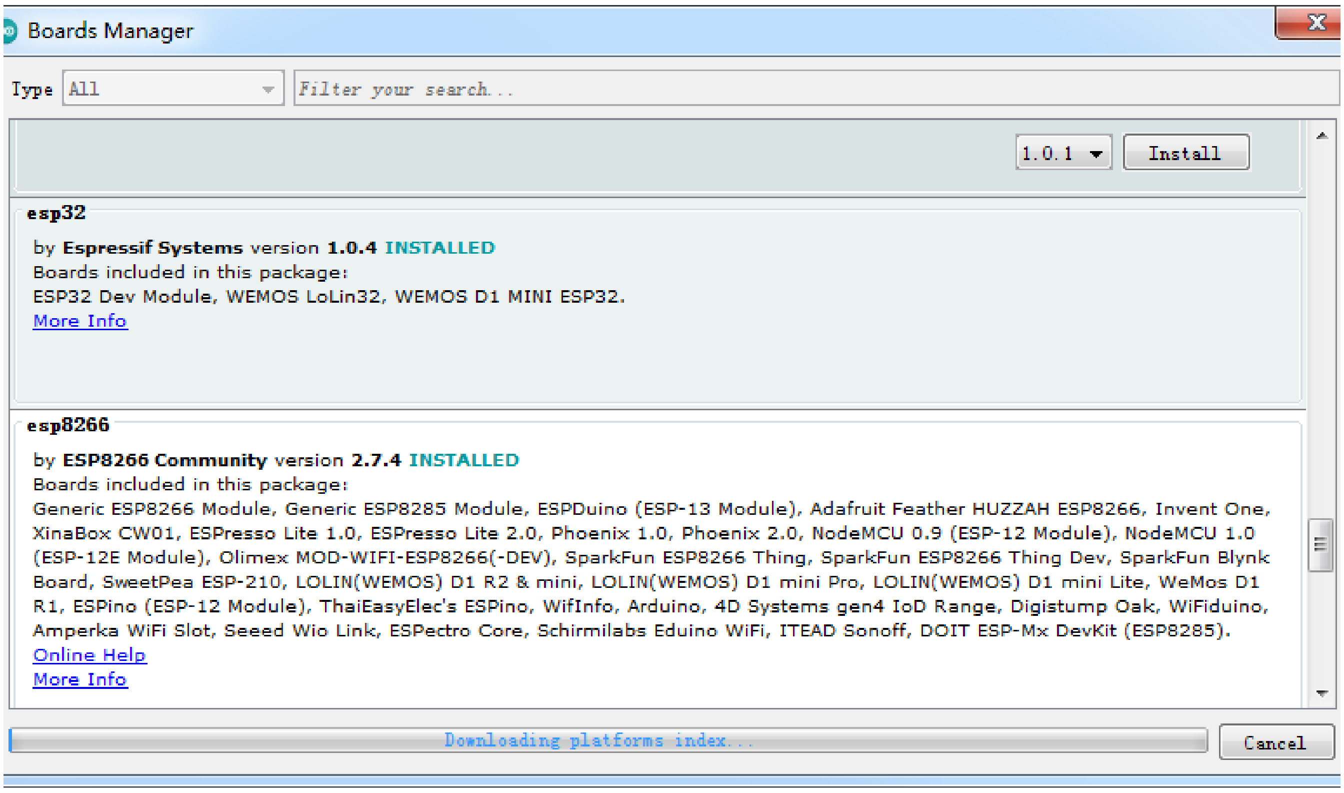
More preferences can be edited directly in the file

C:\Users\Montevina\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

Save it and then exit.

Open “Boards Manager” and wait for updates from external board support links. Search “ESP32” and download its board support package.


Boards Manager

Type All | Filter your search...

esp32
by **Espressif Systems** version **1.0.4** **INSTALLED**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32, WEMOS D1 MINI ESP32.
More Info

esp8266
by **ESP8266 Community** version **2.7.4** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmilabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).
Online Help
More Info

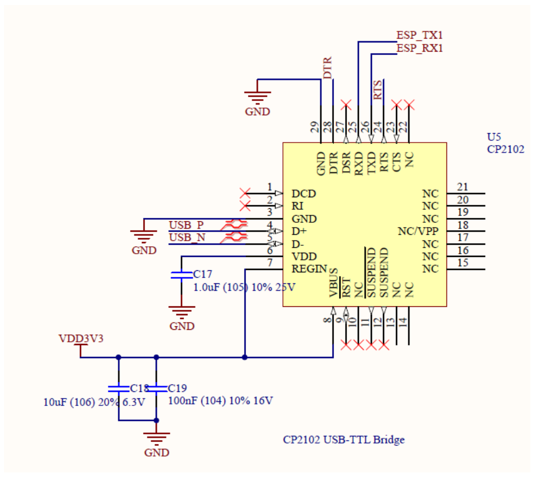
Downloading platforms index...

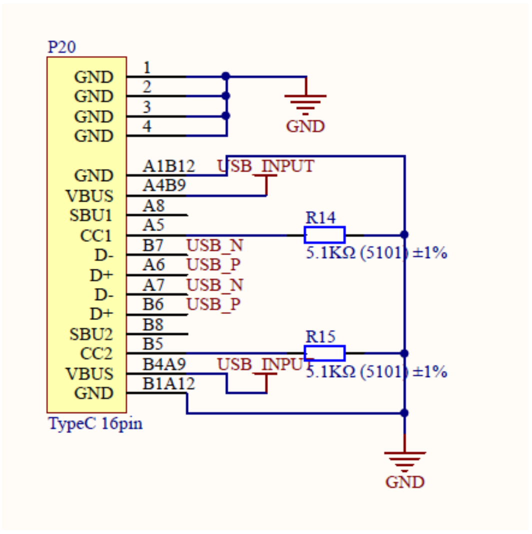
After shown “INSTALLED”, the BiBoard board support package is finished.

3.2.1 USB Downloader

There’s no USB circuit in the ESP32, so we use the CP2102 USB bridge as officially recommended. The maximum download baud is 921600. The bridge is connected to serial1 of the ESP32.

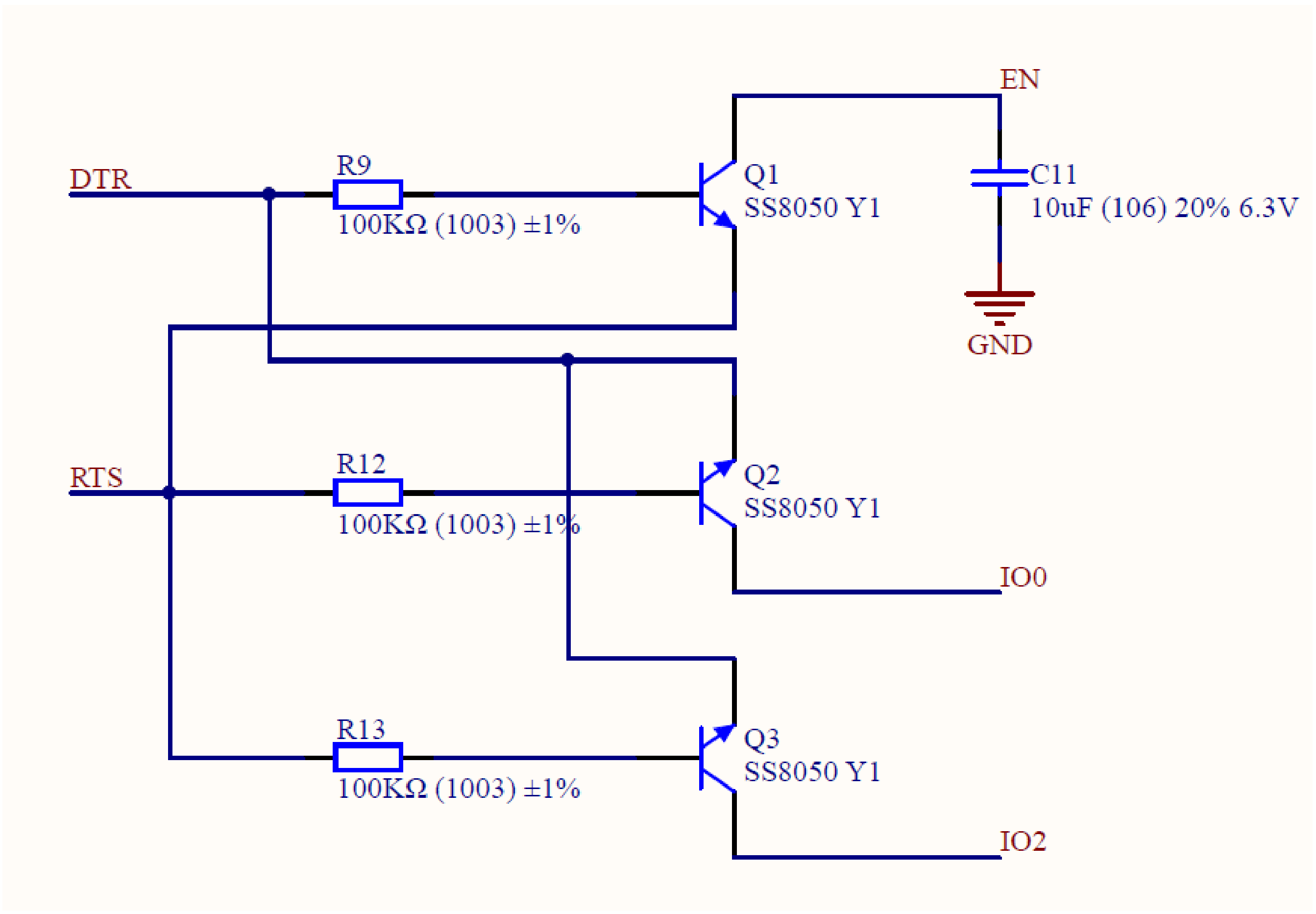
We use the USB Type-C port, 2 resistors CC1 and CC2 are added as the identifier.





We tried the automatic download circuit designed by ESP and the NodeMCU, but none of them works perfectly. So we modified the circuit by adding the third transistor and enlarger the capacitor.

The transistors receive standard serial modem signals DTR and RTS and trigger a unique timing-sequence forcing ESP32 into download mode and then reboot. The detail of the automatic download circuit is shown below.



3.2.2 IMU

We use Invensense MPU6050, the most widely used IMU. Its I2C address is 0x68, and DMP's interrupt is connected to IO26 of the ESP32.

With the help of Jrowberg's MPU6050 DMP library, you can easily get the motion status of the Bittle. The Jrowberg's MPU6050 library must be modified to adapt ESP32. The data types of "int8" and "PGMSpace" should be pre-defined instead of 8-bit AVR macros. We offer the modified library of MPU6050. You can replace the original library so that both AVR boards and ESP boards would be worked normally.

3.2.3 EEPROM

There is a 64Kbit EEPROM on the BiBoard. You can directly use the EEPROM read and write a program that is used on the Arduino UNO. You can use it to store calibration data.

There is also an example program named "EEPROM" in the ESP32 support package. This is not the demo code of the I2C EEPROM. That's the demo of the simulated EEPROM by ESP32's QSPI flash memory.

3.2.4 DAC and audio applications

We use DAC output and a class-D amplifier instead of a PWM buzzer to make Bittle more vivid. You can use 3 ways to drive the audio module:

1. Use Arduino "Tone()" function.
2. Use ESP32 "dacWrite()" function like "analogWrite()" in Arduino. The data quality produced by the DAC is better than the PWM.
3. Use ESP MP3 decode library developed by XTronical, you can play MP3 files. You should configure a file system like SPIFFS or FAT in the flash before you use this MP3 decoder.

URL：https://www.xtronical.com/basics/audio/dacs-on-esp32/

3.2.5 IR modules

The IR sensor on Nyboard and BiBoard are the same, so you can directly use the sketch from the Nyboard. The BiBoard's flash is large enough so that you don't have to disable macros in IRremote.h.

4. Servo sockets

There're 12 PWM servo sockets on the BiBoard, and the pin number is marked near the socket.

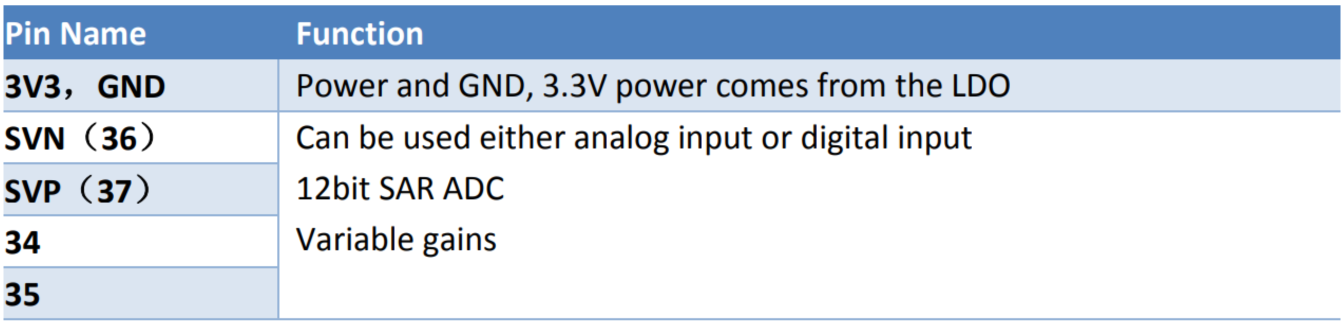
We transform the direction of the PWM servo socket by 90 degrees since the size of the ESP32 module. You should connect the wires first before you screw the BiBoard on the cage.

5. Extension sockets

There're 3 extension sockets on the BiBoard that marked with P15, P16 and P17.

5.1 Analog input sockets（P15）

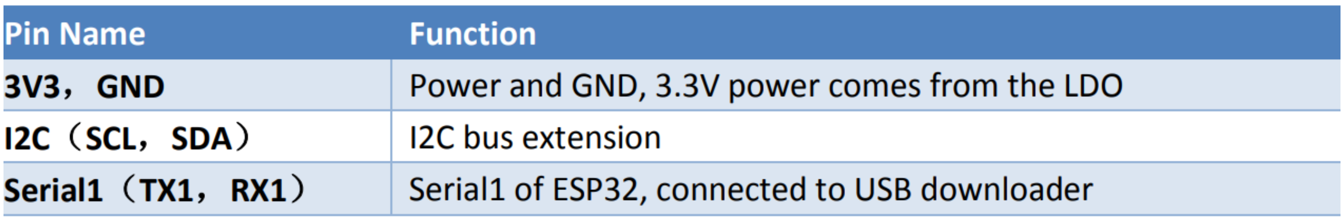
This socket is used for analog input extension, you can try to connect foot press sensors to this socket.



Pin Name	Function
3V3， GND	Power and GND, 3.3V power comes from the LDO
SVN（36）	Can be used either analog input or digital input 12bit SAR ADC Variable gains
SVP（37）	
34	
35	

5.2 Bus extension sockets（P16）

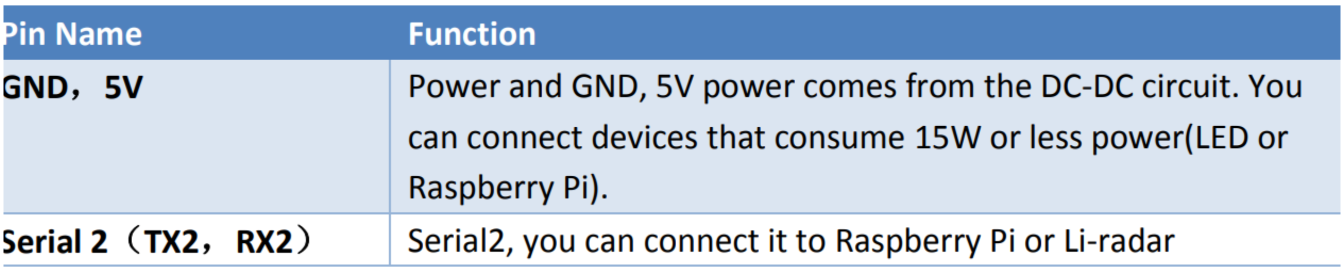
This socket is used for bus extension of the ESP32.



Pin Name	Function
3V3， GND	Power and GND, 3.3V power comes from the LDO
I2C（SCL， SDA）	I2C bus extension
Serial1（TX1， RX1）	Serial1 of ESP32, connected to USB downloader

5.3 树莓派接口 Raspberry Pi interface

You can use this interface to connect to the Raspberry Pi, but you cannot directly mount the Raspberry Pi above the BiBoard. Use wires or adapters instead.



Pin Name	Function
GND， 5V	Power and GND, 5V power comes from the DC-DC circuit. You can connect devices that consume 15W or less power(LED or Raspberry Pi).
Serial 2（TX2， RX2）	Serial2, you can connect it to Raspberry Pi or Li-radar

BiBoard Configuration

1. Read the user manual

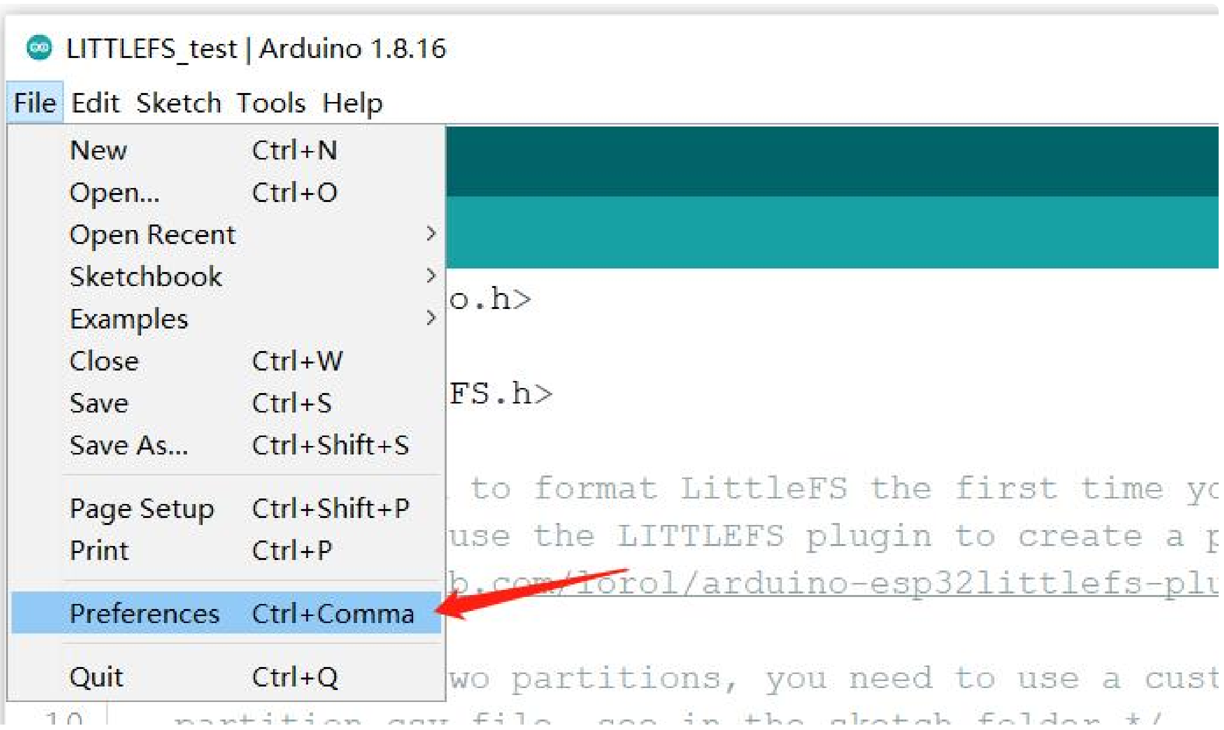
Read the user manual for BiBoard Quick Start Guide.

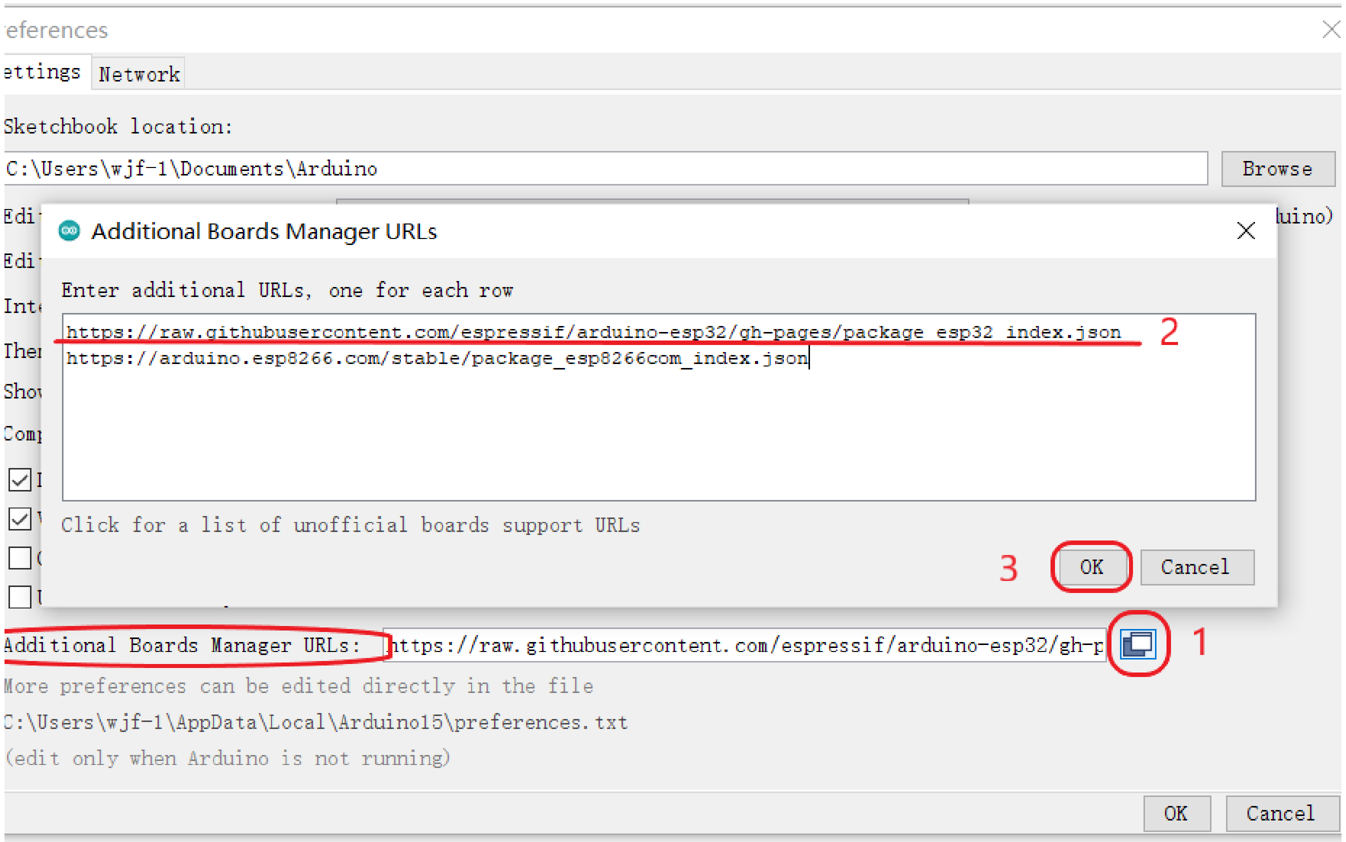
---

2. Set up BiBoard

2.1. Add URL in Arduino->Preferences->Additional Boards Manager URLs

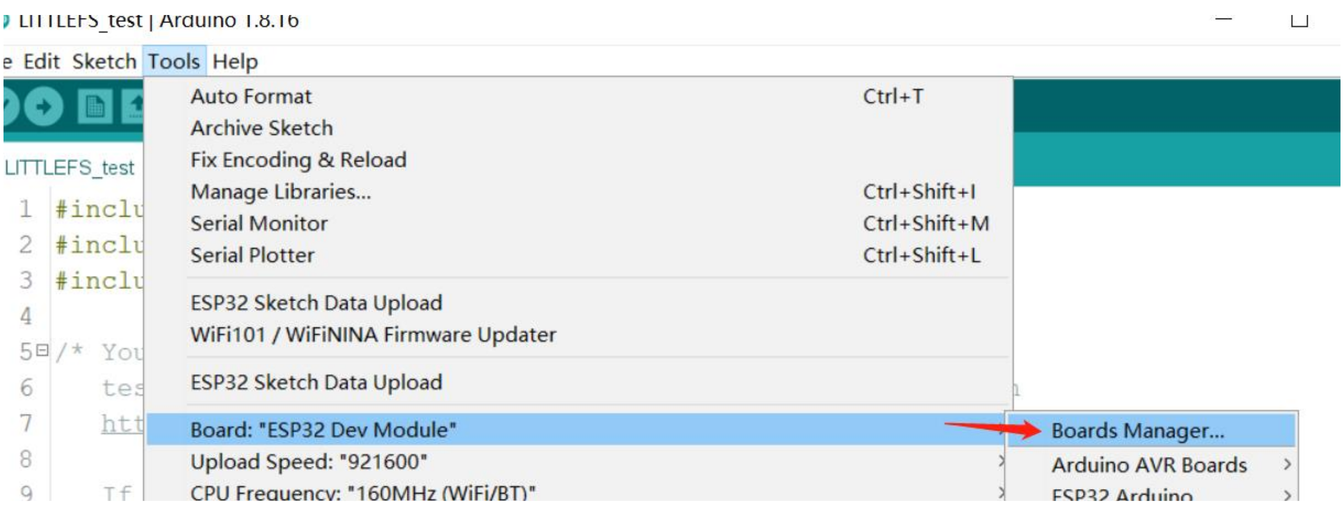
```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.js…
```



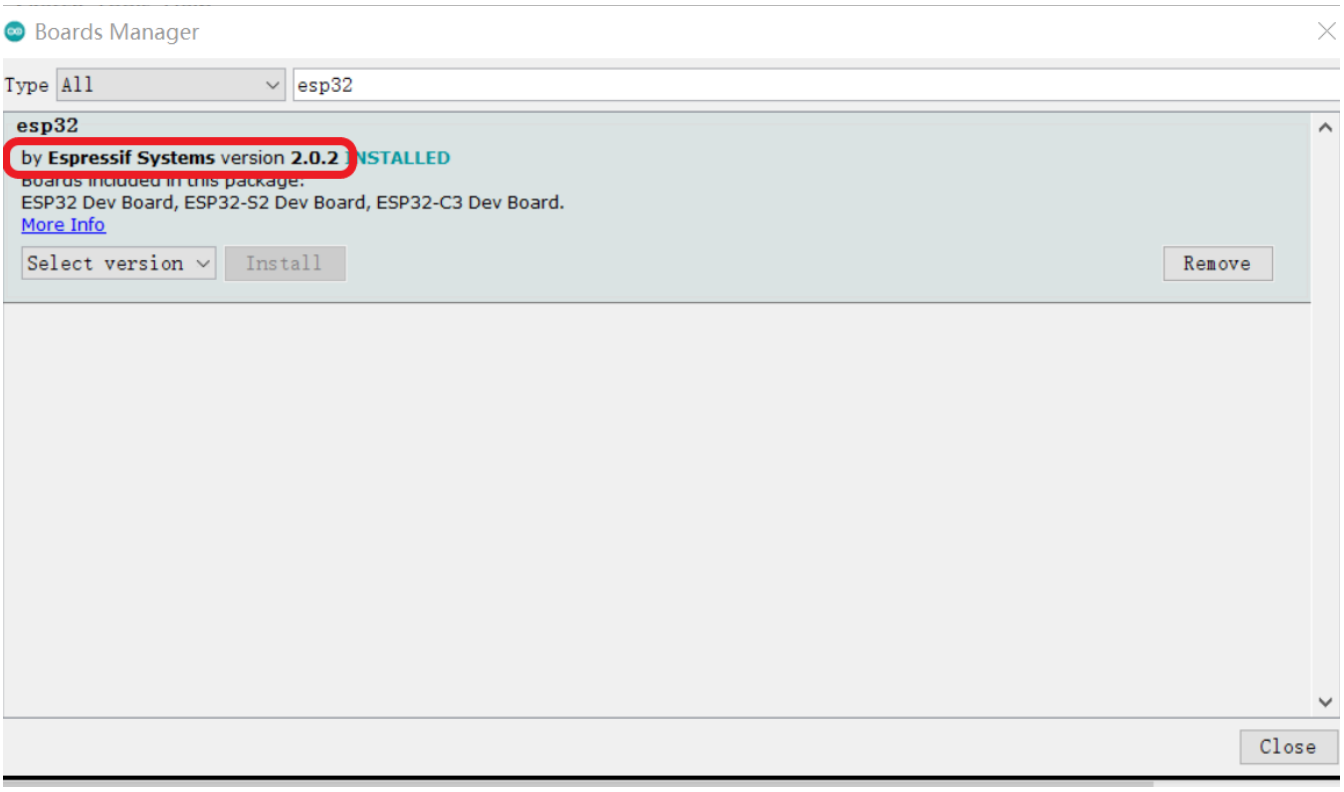

eferences
ettings
Network
Sketchbook location:
Browse
Additional Boards Manager URLs
Enter additional URLs, one for each row
Click for a list of unofficial boards support URLs
OK
Cancel
1
2
3
Additional Boards Manager URLs:
OK
Cancel

```
C:\Users\wjf-1\Documents\Arduino
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
https://arduino.esp8266.com/stable/package_esp8266com_index.json
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-p
More preferences can be edited directly in the file
C:\Users\wjf-1\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)
```

2.2. In Boards Manager, search ESP32 and install the 2.0.* version



Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "160MHz (WiFi/BT)"



esp32
by **Espressif Systems** version **2.0.2** INSTALLED
Boards included in this package:
ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-C3 Dev Board.

2.3. Modify code files

Modify the following files after downloading the BiBoard package:

sdkconfig.h

- For Windows:

  C:\Users\{username}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.2\tools\sdk\esp32\include\config\sdkconfig.h

- for Mac:

  /Users/{username}/Library/Arduino15/packages/esp32/hardware/esp32/2.0.2/tools/sdk/esp32/include/config/sdkconfig.h

Append a line of code at the end of the file:

```
#define CONFIG_DISABLE_HAL_LOCKS 1
```

Or replace with **BiBoard\ESP32config\sdkconfig.h**.

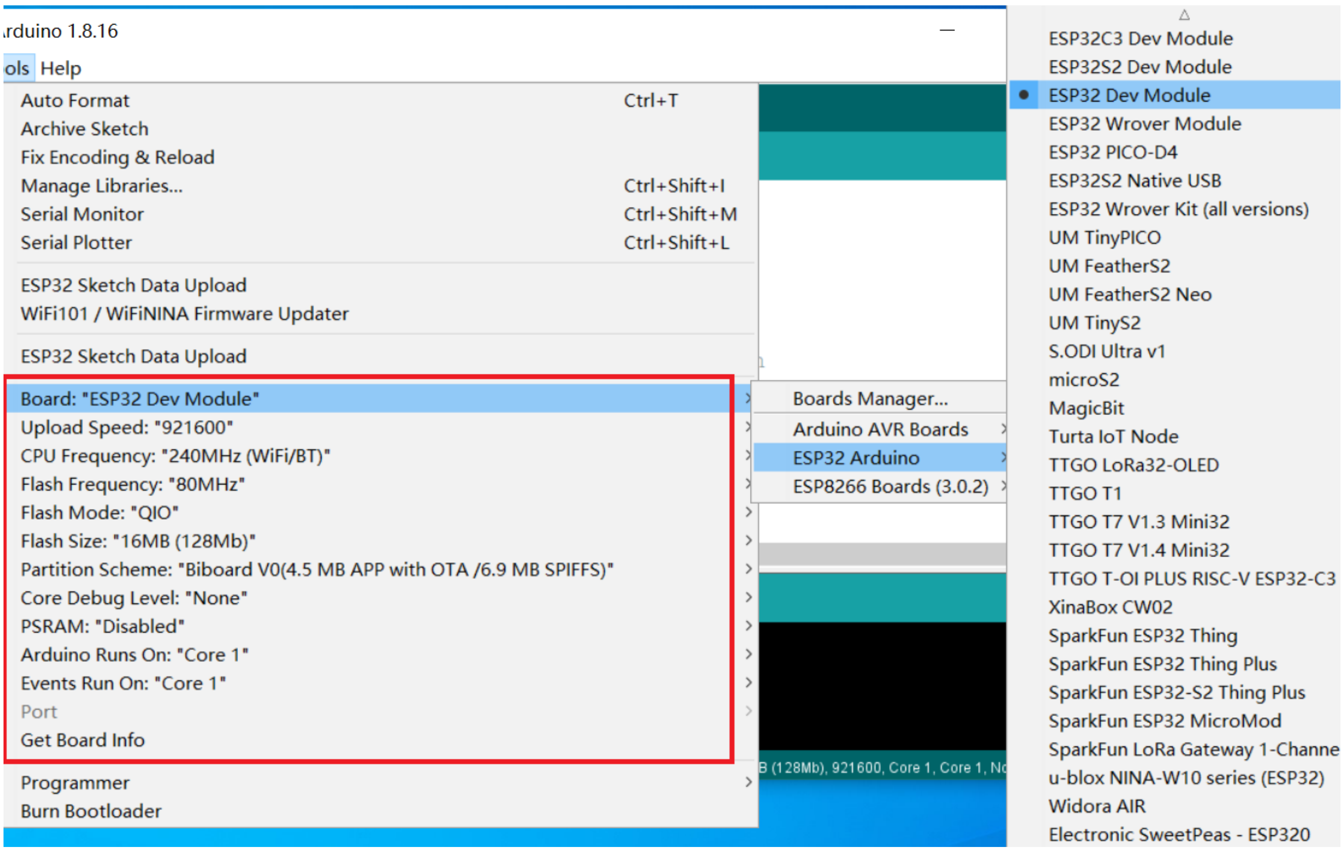
esp32-hal-i2c-slave.c

> - For Windows:
>
> C:\Users\{username}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.2\cores\esp32\esp32-hal-i2c-slave.c with the file in this folder (BiBoard\ESP32config\esp32-hal-i2c-slave.c).
>
> - For Mac:
>
> /Users/{username}/Library/Arduino15/packages/esp32/hardware/esp32/2.0.1/cores/esp32/esp32-hal-i2c-slave.c

Replace with **BiBoard\ESP32config\esp32-hal-i2c-slave.c.**

2.4. Add hardware partition

Read the user manual for Add hardware partition configuration option in Arduino IDE



Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "16MB (128Mb)"
Partition Scheme: "Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"
Arduino Runs On: "Core 1"
Events Run On: "Core 1"

2.5. Compile and upload the sketch

Modify the device type macro definition in **BiBoard.ino** according to the device type.

```
#define BITTLE    //Petoi 9 DOF robot dog: 1 on head + 8 on leg
//#define NYBBLE  //Petoi 11 DOF robot cat: 2 on head + 1 on tail + 8 on leg
//#define CUB
```

Modify the motherboard model macro definition in **BiBoard.ino** according to the motherboard model.

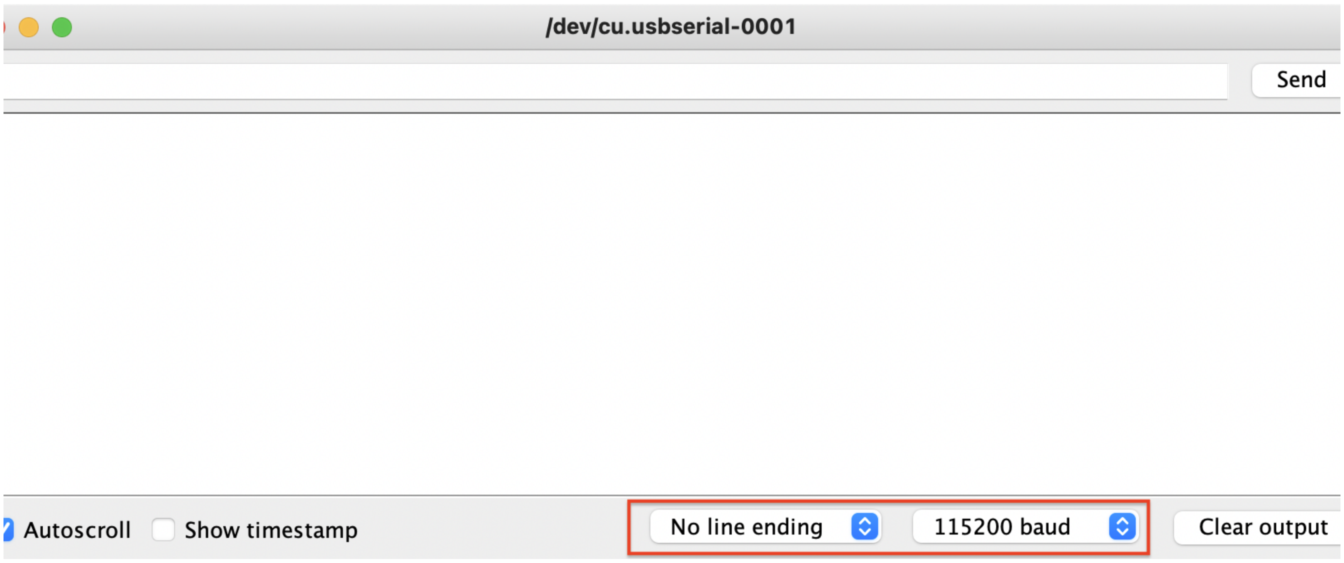
```
#define BiBoard     //ESP32 Board with 12 channels of built-in PWM for joints
//#define BiBoard2  //ESP32 Board with 16 channels of PCA9685 PWM for joints
```

After the modification is completed, you can click the **upload** button to upload BiBoard.ino, and the changes in the program will be automatically saved.

2.6. Initialization

When the newly set BiBoard is powered on, it needs to connect the serial port to the computer, and initialize the joint and gyroscope accelerometer (IMU) in the serial port monitor window.

> Make sure to set the serial monitor as **115200 baud rate and no line ending**.



No line ending
115200 baud

You will see several questions:

```
Reset the joints' calibration offsets? (Y/n):
```

Type ‘Y’ to the question, which means resetting all servo corrections to zero.

```
- Calibrate the Inertial Measurement Unit (IMU)? (Y/n):
```

Type 'Y' to the question, which means calibrating the MPU6050, i.e. the gyro/accelerometer sensor.

> Put the robot **FLAT** on the table and don't touch it during calibration.
>
> The program starts calibration after playing the melody 6 times.

After the calibration, the program will enter the regular **Power On** routine covered in the next section.

The details of serial port printing information are as follows：

```
* Start *
Scanning I2C network...
- I2C device found at address 0x54  !
- I2C device found at address 0x68  !
- done
Set up the new board...
// The device name to use when connecting with bluetooth
- Name the new robot as: BittleED
Reset the joints' calibration offsets? (Y/n):
Y

Initializing MPU...
- Testing MPU connections...attempt 0
- MPU6050 connection successful
- Initializing DMP...
- Calibrate the Inertial Measurement Unit (IMU)? (Y/n):
Y

Put the robot FLAT on the table and don't touch it during calibration.
- Calibrating the Inertial Measurement Unit (IMU)...
>..........>..........
MPU offsets:
//           X Accel  Y Accel  Z Accel   X Gyro   Y Gyro   Z Gyro
//OFFSETS     3752,    -968,     942,     170,      76,      21
- Enabling DMP...
- Enabling interrupt detection (Arduino external interrupt 26)...
- DMP ready! Waiting for the first interrupt...
Bluetooth name: BittleED
Waiting for a client connection to notify...
Setup ESP32 PWM servo driver...
Ready!

```

> The main program of Bittle judges whether it has been initialized by comparing the **BIRTHMARK** in the EEPROM, and will not enter the initialization process again when it is turned on next time.
>
> If you need to recalibrate the servo offsets or recalibrate the IMU (MPU6050), you can modify the following configuration macro definitions in the **BiBoard.ino** then recompile and upload the sketch.
>
> ```
> //change this mark to any other character if you want to reset
> ```

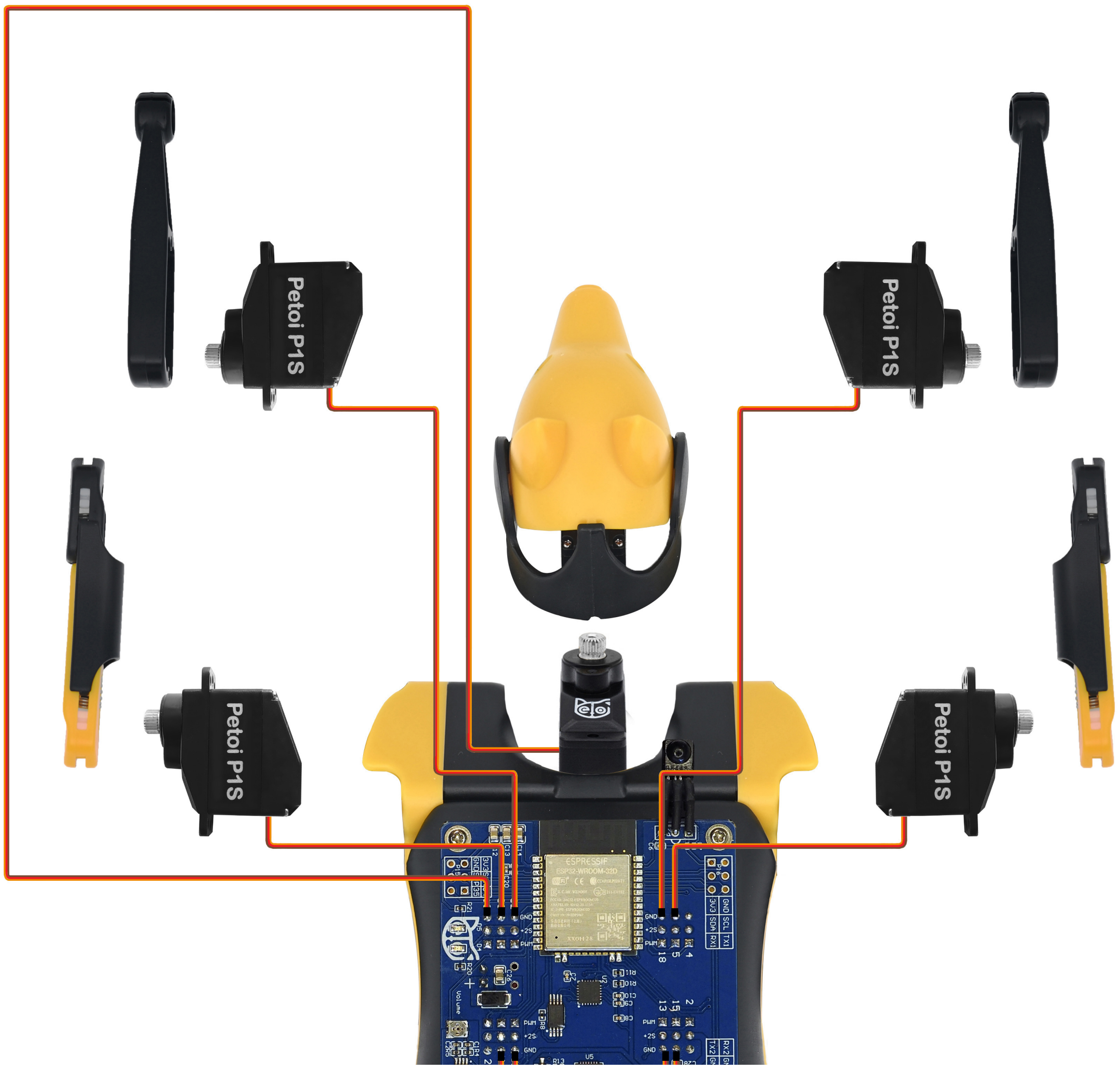
```
//the Bluetooth name and calibration values in the EEPROM
#define BIRTHMARK 'x'
```

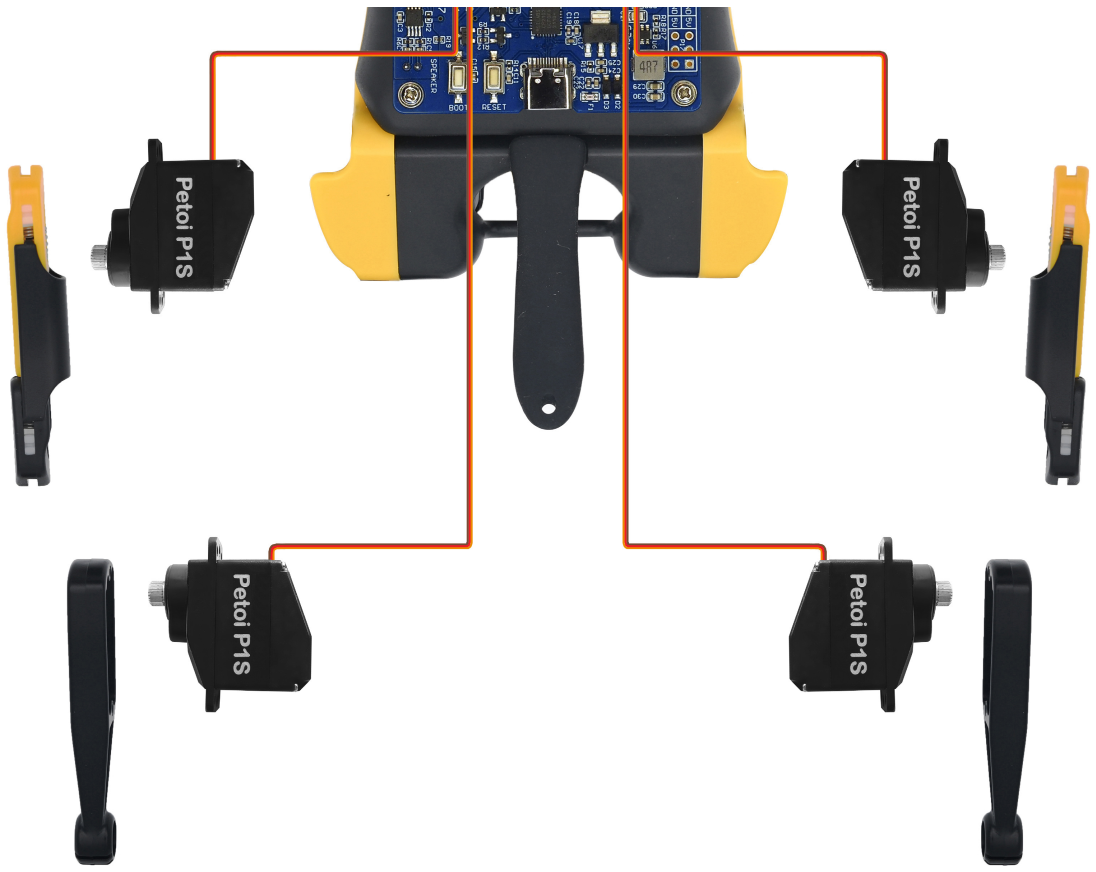
2.7. Power on

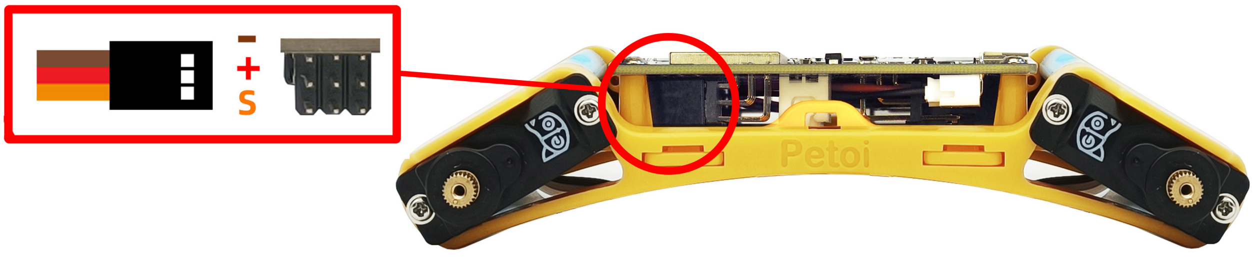
- If you power on the robot upside down, it will enter the calibration state:

The servos can point in any direction when the robot is turned off. After receiving the calibration signal, all the servos will rotate to their calibration positions (zero angles) and stop moving. We call this position Bittle's "calibration pose". The Bittle remains in the calibrated position until the first user input (infrared, Bluetooth or serial) is received.

Then you can attach body parts to the servos according to the calibrated position and use the ‘c’ commands to set the fine-calibration values. Although the BiBoard has only 12 pins, the joint index numbers are configured in the same order as the NyBoard. The connection between the joint servo and the pin is shown in the figure below：







Please refer to chapter 6 Calibration and Final Assembly for the complete calibration process.

- If you power on the robot and it is upright (with its back upward), the robot will start from the "rest" posture (fold the legs and unlock the servos).

2.8. Configuration with App

The BiBoard has built-in Bluetooth and you can connect it with the new Android app:

App for BiBoard (Android 4.4+).

You can check the update history and added features in **ChangeLog.md** (BiBoard\ChangeLog.md)

Demo Applications

1.GPIO port

Operate the GPIO port of BiBoard

There is no separate GPIO port on BiBoard, but the multiplexed serial port 2 (pin 16, 17) or the PWM pin of the unused PWM servo interface can be used as GPIO port. The GPIO port is also relatively simple to use. After configuring the input and output mode, the usage is exactly the same as that of Arduino UNO. You can use any IO control program of Arduino UNO, just change the number of IO .

```
/* In this demo, we use TX2, RX2 as general purpose IO
*   TX2 : IO17
*   RX2 : IO16
*/

void setup() {
  // initialize digital pin 16 & 17 as an output.
  pinMode(16, OUTPUT);
  pinMode(17, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {

  digitalWrite(16, HIGH);             // GPIO 16 & 17 HIGH
  digitalWrite(17, HIGH);
  delay(1000);                        // wait for a second

  digitalWrite(16, LOW);              // GPIO 16 & 17 LOW
  digitalWrite(17, LOW);
  delay(1000);                        // wait for a second
}
```

2.Serial port

There are 2 serial ports,which are separately located on 2 expansion sockets (P16, P17) ,on BiBoard.

The serial port 1 on the P16 can be connected to the USB downloader and the external serial device. Please do not use the downloader and the external serial device at the same time. The serial port voltage division will lead to communication errors.

In the Arduino demo, Serial represents the serial port 0, Serial1 represents the serial port 1.Serial and Serial1 send to each other.

```
/* In this demo, we use Serial and Serial1
*  Serial and Serial1 send to each other
*/

void setup() {
  // initialize both serial ports:
  Serial.begin(115200);
  Serial1.begin(115200);
}

void loop() {
  // read from port 1, send to port 0:
  if (Serial1.available()) {
    int inByte = Serial1.read();
    Serial.write(inByte);
  }

  // read from port 0, send to port 1:
  if (Serial.available()) {
    int inByte = Serial.read();
    Serial1.write(inByte);
  }
}
```

3.Analog-digital converter

Application of ADC which is variable gain on BiBoard (ESP32)

The instructions of ADC on BiBoard

The 34, 35, 36 and 39 pins of the ESP32 module support input only. We configure it as an analog input port on BiBoard, which makes it convenient for developers to connect 4 foot sensors.

The usage of analog input analog-to-digital converter (ADC) on BiBoard is the same as the basic Arduino UNO, but the accuracy is higher (12 bits, UNO is 10 bits), and a programmable gain amplifier is added to make the ADC work in the best range.

When a 1V voltage signal is input, if 12bit access is used according to the normal configuration, the reference voltage is equal to the power supply voltage (3.3V): the corresponding output is 0～1241, a large part of the ADC range will be wasted, resulting in inaccurate data. When we configure the programmable gain, we can make the 1V input signal fill almost the entire ADC range, and the accuracy and resolution are greatly improved.

This demo uses 4 inputs, respectively configured as: 0/2.5/6/11 decibel amplification gain, it should be noted that the default configuration of ESP32 Arduino is 11 decibel amplification gain.

We use "analogSetPinAttenuation(PIN_NAME, attenuation)" to configure the gain of a single input pin, or use "analogSetAttenuation(attenuation)" to configure the gain of all analog input pins.

```
// Ain 34 - 0dB Gain - ADC_0db
analogSetPinAttenuation(34, ADC_0db);

// Ain 35 - 2.5dB Gain - ADC_2_5db
analogSetPinAttenuation(35, ADC_2_5db);

// Ain 36 - 6dB Gain - ADC_6db
analogSetPinAttenuation(36, ADC_6db);

// Ain 39 - 11dB Gain - ADC_11db    (default)
analogSetPinAttenuation(39, ADC_11db);
```

In the actual test, when the 1V standard voltage is input, the ADC values are: 3850/2890/2025/1050. In future productions, the ADC range can be changed by changing the ADC gain without the replacement of the reference voltage source.

4.Digital-Analog Converter

The usage of DAC

The purpose of the DAC is the opposite of that of the ADC. The DAC converts a digital signal into an analog signal for output.

Remember the music when NyBoard is turned on? It is using PWM to make music sound which uses high-speed switching to adjust the duty cycle to output voltage.

Compared with PWM, the DAC will directly output the voltage without calculating the duty cycle. ESP32 integrates a 2-channel 8-bit DAC with a value of 0-255. The voltage range is 0-3.3V. Therefore, the formula for calculating the output voltage of the DAC is as follows:

$$DAC = (int)\ TargetV/3.3V * 255$$

The demo is as follows:

```
#define DAC1 25

void setup() {
}

void loop() {

  // 8bit DAC, 255 = 3.3V, 0 = 0.0V
  for(int i = 0; i < 255; i++){
    dacWrite(DAC1, i);
    delay(10);
  }
}
```

5.EEPROM (Electrically Erasable Programmable read only memory)

The usage of EEPROM is the same as Arduino UNO, there are two operations: read and write.

Read:

- I2C address of EEPROM
- The internal address of EEPROM (the address for storing data)
- Read data

Write:

- I2C address of EEPROM
- The internal address of EEPROM (the address for storing data)
- Write data

In the BiBoard demo, the address of EEPROM on the I2C bus is 0x54, and the capacity is 8192Bytes (64Kbit). We sequentially write a total of 16 values from 0 to 15 in the EEPROM from the first address, and then read them for comparison. Theoretically, the data written in EEPROM and the data read from the corresponding address should be the same.

In the NyBoard factory test, we also use this method, but it is more complicated. We will use a fixed list to fill the EEPROM and read it out for comparison.

```
#include <Wire.h>

#define EEPROM_ADDRESS 0x54
#define EEPROM_CAPACITY 8192       // 64Kbit
#define EEPROM_TESTBYTES 16


// write 1 byte EEPROM by address
void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
{
  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));    // MSB
  Wire.write((int)(eeaddress & 0xFF)); // LSB
  Wire.write(data);
  Wire.endTransmission();

  delay(5);
}

```

```
// read 1 byte EEPROM by address
byte readEEPROM(int deviceaddress, unsigned int eeaddress )
{
  byte rdata = 0xFF;

  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));   // MSB
  Wire.write((int)(eeaddress & 0xFF)); // LSB
  Wire.endTransmission();

  Wire.requestFrom(deviceaddress,1);

  if (Wire.available())
    rdata = Wire.read();
  return rdata;
}

void testI2CEEPROM(){

    byte tmpData = 0;

    Serial.println("EEPROM Testing...");

    // write EEPROM from 0 to EEPROM_TESTBYTES
    for(int i = 0; i < EEPROM_TESTBYTES; i++){
        writeEEPROM(EEPROM_ADDRESS, i, i % 256);
        delay(1);
    }

    Serial.println();

    // read from 0 to EEPROM_TESTBYTES
    for(int i = 0; i < EEPROM_TESTBYTES; i++){
        tmpData =  (int)readEEPROM(EEPROM_ADDRESS, i);
        Serial.print(tmpData);
        Serial.print("\t");
    }
}


void setup(){


    Serial.begin(115200);
    Wire.begin();

    testI2CEEPROM();
}

void loop(){

}
```

Note: the EEPROM operations, especially write operations, are generally not put into the loop() loop. Although the EEPROM is resistant to erasing (100,000 times), if a certain block is frequently written in the loop, It will cause the EEPROM to malfunction.

6.Gyro IMU（MPU6050）

MPU6050 is the most widely used 6-axis gyroscope, which can not only measure 3-axis angular velocity and 3-axis acceleration more accurately, but also use the built-in digital motion processor (DMP) for hardware based attitude fusion calculation. So novices can use it very conveniently. For this reason, we also use MPU6050 gyroscope.

There are many demos of MPU6050 on Arduino UNO, the most famous is jrowberg's I2Cdev and MPU6050DMP library:

i2cdevlib/Arduino/MPU6050 at master · jrowberg/i2cdevlib
GitHub

Unfortunately, this library cannot be run directly on BiBoard based on ESP32. We found the ported library on Github, which is easy to use. This library adds the definition of PGMSpace for the ARM and ESP series, adds the calibration function, and removes the FIFO overflow processing function (friends who are interested can use Beyond Compare for code comparison). The library contains I2Cdev and MPU6050, the address and compressed package are as follows:

mpu6050/src at master · ElectronicCats/mpu6050
GitHub

mpu6050-master.zip 121KB
Binary

mpu6050-master.zip

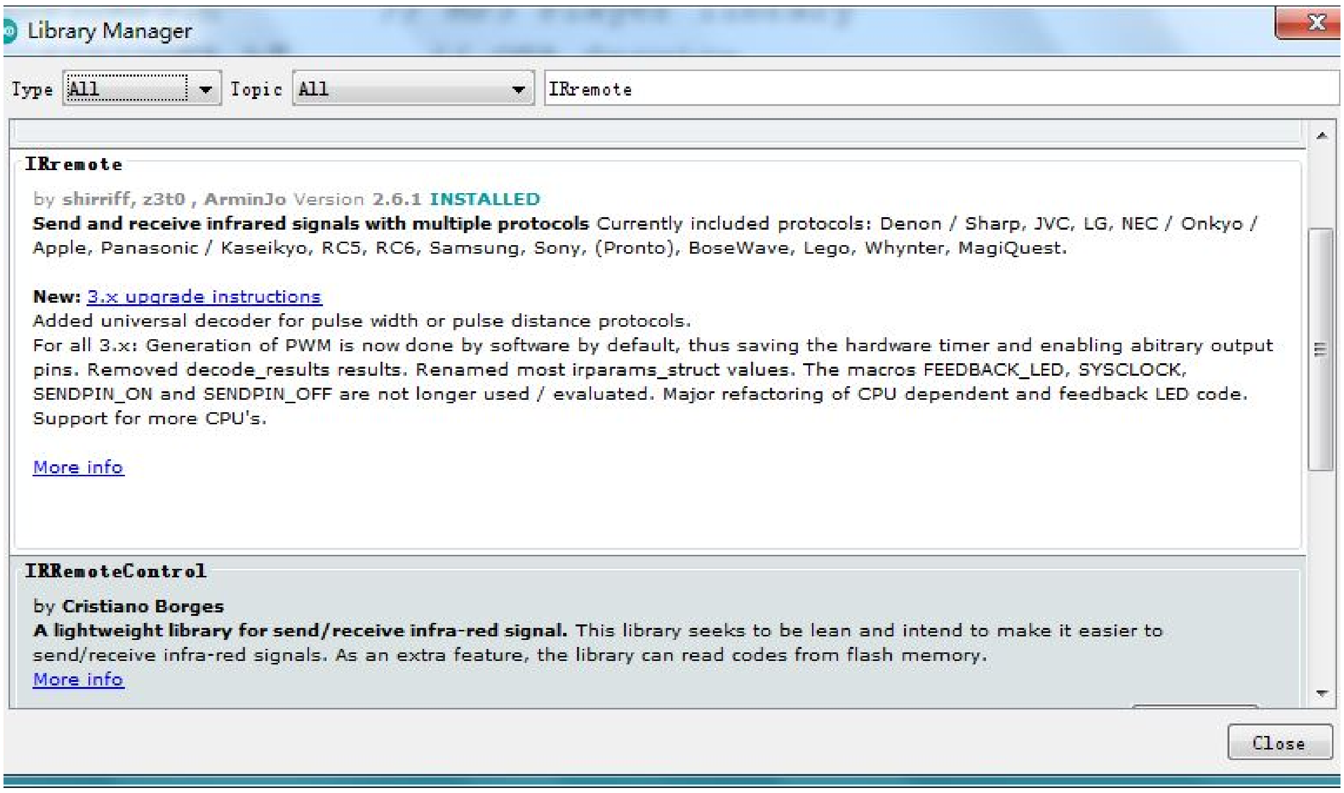
After the download is complete, create a MPU6050 folder under Documents/Arduino/library, and copy the library files in the compressed package into it. The library of this modified MPU6050 is also compatible with ARM and AVR, so if you have the original I2Cdev and MPU6050 libraries in your computer, you can delete them.

We can use the official MPU6050_DMP6 demo.

7.Infrared remote control

BiBoard is equipped with an infrared sensor, which is connected to the 23rd pin. The use of infrared is exactly the same as which is on Arduino UNO based on AVR

First download the 2.6.1 version of the IRremote library, you need to manually select the 2.6.1 version. Because the infrared-related codes have changed in later versions, if you use the 3.X version, the commands will not be translated. In order to be compatible with our previous products, we decided to use the 2.6.1 version after testing.



IRremote

by **shirriff, z3t0 , ArminJo** Version **2.6.1** **INSTALLED**
Send and receive infrared signals with multiple protocols Currently included protocols: Denon / Sharp, JVC, LG, NEC / Onkyo / Apple, Panasonic / Kaseikyo, RC5, RC6, Samsung, Sony, (Pronto), BoseWave, Lego, Whynter, MagiQuest.

New: 3.x upgrade instructions
Added universal decoder for pulse width or pulse distance protocols.
For all 3.x: Generation of PWM is now done by software by default, thus saving the hardware timer and enabling abitrary output pins. Removed decode_results results. Renamed most irparams_struct values. The macros FEEDBACK_LED, SYSCLOCK, SENDPIN_ON and SENDPIN_OFF are not longer used / evaluated. Major refactoring of CPU dependent and feedback LED code. Support for more CPU's.

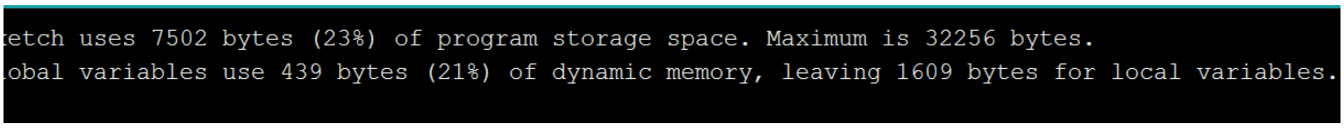
More info

IRRemoteControl

by **Cristiano Borges**
A lightweight library for send/receive infra-red signal. This library seeks to be lean and intend to make it easier to send/receive infra-red signals. As an extra feature, the library can read codes from flash memory.
More info

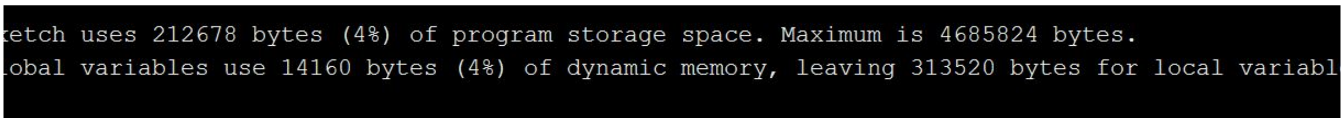
When using NyBoard, in order to ensure that the code can be compiled smoothly, we need to remove unnecessary code in the IRremote library, that is, remove the encoder/decoder that we don't use, and only keep the NEC_DECODER, which is the 38KHz signal decoder in NEC format.

Due to the flash memory capacity of BiBoard is “huge”, we don’t need to remove unnecessary code in the IRremote library.



```
etch uses 7502 bytes (23%) of program storage space. Maximum is 32256 bytes.
obal variables use 439 bytes (21%) of dynamic memory, leaving 1609 bytes for local variables.
```

23% of flash is used on UNO



```
etch uses 212678 bytes (4%) of program storage space. Maximum is 4685824 bytes.
obal variables use 14160 bytes (4%) of dynamic memory, leaving 313520 bytes for local variabl
```

Only 4% of flash is used on BiBoard

Finally, a demo is attached, which accepts infrared signals and prints via the serial port. You can also use official demo for testing.

```
#include <Arduino.h>
#include <IRremote.h>

```

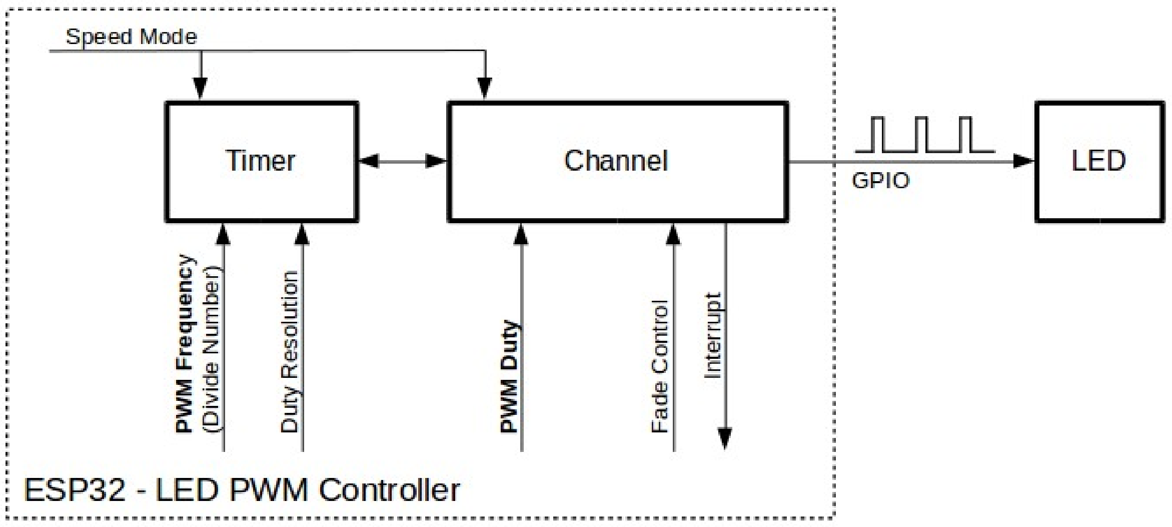
```
int RECV_PIN = 23;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn();
  Serial.println("IR Receiver ready");
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    Serial.print(" - ");
    irrecv.resume(); // Receive the next value
  }
  delay(300);
}
```

8.PWM(Pulse Width Modulation)

1. The introduction of PWM function on BiBoard (ESP32)

The ESP32 used by BiBoard is different from the 328P used by UNO. Because the PWM of ESP32 uses the matrix bus, it can be used on unspecified pins.

The PWM of ESP32 is called LED controller (LEDC). The LED PWM controller is mainly used to control LEDs, and it can also generate PWM signals for the control of other devices. The controller has 8 timers, corresponding to 8 high-speed channels and 8 low-speed channels, totaling 16 channels.



Key Settings of LED PWM Controller's API

Compared with UNO, directly use "analogWrite()" to input any duty ratio between 0-255. The PWM control of ESP32 on BiBoard is more troublesome. The parameters that need to be controlled are as follows:

1. Manual selection of PWM channels (0-15) also improves the flexibility of the use of pins
2. The number of bits of the PWM waveform determines the resolution of the duty cycle of the PWM waveform. The higher the number of bits, the higher the accuracy.
3. The frequency of the PWM waveform determines the speed of the PWM waveform, the higher the frequency, the faster the speed.

The frequency of the PWM waveform and the number of bits are relative, the higher the number of bits, the lower the frequency. The following example is quoted from the ESP32 programming manual:

For example, when the PWM frequency is 5 kHz, the maximum duty cycle resolution can be 13 bits. This means that the duty cycle can be any value between 0 and 100%, with a resolution of ~0.012% (2 ** 13 = 8192 discrete levels of LED brightness).

The LED PWM controller can be used to generate high-frequency signals, enough to clock other devices such as digital camera modules. Here the maximum frequency can be 40 MHz, and the duty cycle resolution is 1 bit. In other words, the duty cycle is fixed at 50% and cannot be adjusted.

The LED PWM controller API can report an error when the set frequency and duty cycle resolution exceed the hardware range of the LED PWM controller. For example, if you try to set the frequency to 20 MHz and the duty cycle resolution to 3 bits, an error will be reported on the serial port monitor.

2. Configure the PWM frequency on BiBoard in Arduinoin Arduino

As shown above, we need to configure the channel, frequency and number of bits, and select the output pin.

Step 1: Configure the PWM controller

```
const int freq = 5000; // PWM frequency
const int ledcChannel = 0; // ledc channel, 0-15
const int resolution = 8; // resolution of PWM，8bit (0～255)
ledcSetup(ledcChannel, freq, resolution);
```

Step 2: Configure the PWM output pins

```
ledcAttachPin(ledPin, ledcChannel);
```

Step 3: Output PWM waveform

```
ledcWrite(ledcChannel, dutyCycle);
```

In the demo, we choose IO2 as the output pin, connect IO2 to an LED, and you can observe the effect of the

3. Complete code:

```
/* In this demo, we show how to use PWM in BiBoard(ESP32)
* It's different from the Arduino UNO based on the ATMega328P
*/

// define the PWM pin
const int ledPin = 2;  // 16 corresponds to GPIO16

// setting PWM properties
const int freq = 5000;          // PWM frequency
const int ledcChannel = 0;      // ledc channel, in ESP32 there're 16 ledc(PWM) channels
const int resolution = 8;       // resolution of PWM

void setup(){
  // configure ledc functionalitites
  // channels 0-15, resolution 1-16 bits, freq limits depend on resolution
  // ledcSetup(uint8_t channel, uint32_t freq, uint8_t resolution_bits);
  ledcSetup(ledcChannel, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(ledPin, ledcChannel);
}

void loop(){
  // increase the LED brightness
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    ledcWrite(ledcChannel, dutyCycle);
    delay(15);
  }

  // decrease the LED brightness
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    ledcWrite(ledcChannel, dutyCycle);
    delay(15);
  }
}
```

9.Servo(under construction)

10.Classic Bluetooth serial port SPP

The sample code mainly demonstrates the mutual forwarding of information between the Bluetooth serial port and the serial port, which is derived from the official demo of ESP32, which is simple and easy to understand. So the description mainly explains the concepts that appear in the code.

1. Bluetooth protocol

At present, the main Bluetooth protocols are divided into two categories, traditional Bluetooth (HS/BR/EDR) based on RFCOMM and Bluetooth low energy (BLE) based on GATT.

Traditional Bluetooth is faster and has many specific application protocols, such as audio-oriented A2DP, Bluetooth serial port SPP, etc. However, the power consumption is high, and access to Apple devices requires MFi (Made For iOS) chips and certification.

Bluetooth Low Energy (BLE) can define various GATT profiles by itself, and it is also equipped with commonly used profiles (such as device information, battery, etc.). It has low power consumption and is widely used. It can be used on Apple devices. The disadvantages is that it is slower than traditional Bluetooth. Bluetooth low energy is mostly used on devices with low data volume but sensitive to power consumption, such as bracelets/smart watches/beacons.

2. Classic Bluetooth serial port (SPP)

This demo uses the SPP protocol based on traditional Bluetooth, which comes with all serial port protocols. When the computer or Android phone is connected and paired, a serial port number will be automatically generated in the system for communication, and the experience is not much different from that of a normal wired serial port.

```
//This example code is in the Public Domain (or CC0 licensed, at your option.)
//By Evandro Copercini - 2018
//

//This example creates a bridge between Serial and Classical Bluetooth (SPP)
//and also demonstrate that SerialBT have the same functionalities of a normal Serial

#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("BTSPP_Test");          //Bluetooth device name
  Serial.println("The device started,");
  Serial.println("Now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

The bluetooth low energy serial port will be demonstrated in the next chapter. In essence, it is a profile configured with a serial port and requires host software support.

11.Bluetooth low energy (BLE) serial port pass-through

Bluetooth Low Energy (BLE, Bluetooth Low Energy) serial port pass-through is widely used. On Apple's iOS platform, Classic Bluetooth requires MFi certification to connect with Apple's iOS devices. The Bluetooth low energy device does not have this restriction.

The protocol stack and principle of Bluetooth low energy will not be repeated here, there are many related articles and videos. In short, the bluetooth service is provided in the form of a profile, and there are N characters with independent IDs (UUID) under the profile of each service. Each character has different permissions (read, write, notify, indicate). After the user defines the character and combines it with the authority, a complete service can be provided.

What BLE pass-through is actually to establish a BLE Service, and there are 2 characters under this profile.

```
#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" // UART service UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

One for TX (transmit data) and one for RX (receive data). For this they have different permissions. The following code is to create a new service and character:

```
  // Create the BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // Create a BLE Characteristic
  pTxCharacteristic = pService->createCharacteristic(
                                                                              CHARACTERI
                                                                              BLECharact
                                                                      );

  pTxCharacteristic->addDescriptor(new BLE2902());

  BLECharacteristic * pRxCharacteristic = pService->createCharacteristic(
                                                                                       CI
                                                                                      BLI
                                                                              );
```

Next are two callback functions, which are performed when there is a connection and when there is a write RX character:

```
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();

      if (rxValue.length() > 0) {
        Serial.println("*********");
        Serial.print("Received Value: ");
        for (int i = 0; i < rxValue.length(); i++)
          Serial.print(rxValue[i]);

        Serial.println();
        Serial.println("*********");
      }
    }
};
```

Finally, the main loop is the control of the connection, which determines whether there is a connection and whether it is disconnected.

```
    if (deviceConnected) {
        pTxCharacteristic->setValue(&txValue, 1);
        pTxCharacteristic->notify();
        txValue++;
                    delay(10); // bluetooth stack will go into congestion, if too many pacl
          }

    // disconnecting
    if (!deviceConnected && oldDeviceConnected) {
        delay(500); // give the bluetooth stack the chance to get things ready
        pServer->startAdvertising(); // restart advertising
        Serial.println("start advertising");
        oldDeviceConnected = deviceConnected;
    }
    // connecting
    if (deviceConnected && !oldDeviceConnected) {
                // do stuff here on connecting
        oldDeviceConnected = deviceConnected;
    }
```

For the complete code, see the example of the official library: ble_uart, and the debugging tool can use LightBlue.

12.File system SPIFFS

ESP32 File System SPIFFS Configuration Guide

1. Why use a file system

On BiBoard (ESP32), in addition to the regular program area and boot area, we use the file system in the Flash partition.

The role of a file system with independent partitions is as follows:

- Save the data at the specified address and will not be deleted due to re-update (such as calibration data, gait data)
- No external SD card needed, saving hardware resources
- Save HTML and CSS files to build a web server
- Save images, audio and other files

Common file systems include Windows NTFS, exFAT, and Linux log file systems Ext and XFS. But in the embedded field, these large file systems are too large. We use the lightweight SPIFFS (SPI Flash File System), an embedded file system for SPI NOR flash devices, and support functions such as wear leveling and file system consistency checking.

Because of its light weight, the biggest feature of SPIFFS is that it does not support tree directories, that is, all files are stored in the same layer. SPIFFS provided by ESP32 has the following features:

- Currently, SPIFFS does not support directories, it produces a flat structure. If SPIFFS is mounted under /spiffs, then creating a file with the path /spiffs/tmp/myfile.txt will create a file called /tmp/myfile.txt in SPIFFS, instead of myfile.txt in the directory /spiffs/tmp.
- It is not a real-time stack. One write operation might take much longer than another.
- For now, it does not detect or handle bad blocks.

2. Install the Arduino ESP32 file system uploader

You can create/save and delete files with your own Arduino code, but the operation is cumbersome. You need to put data or even binary files into Arduino Sketch and create files by running the program.

However, there is a very useful tool that can directly upload files from the computer to the file system. Although it is slightly more troublesome than the "drag-and-drop" copy of the "removable storage", whether it is MP3 audio or HTML web files, all can be easily uploaded to flash memory. Let's learn how to use this plugin.

3. Install ESP32 file upload plugin

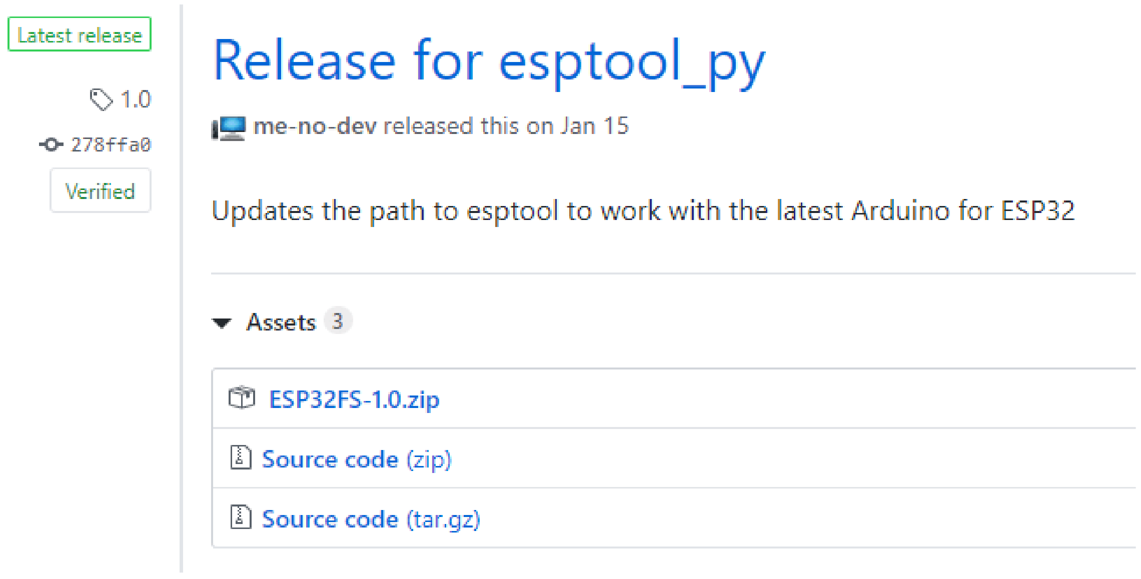
3.1 Preparing the Environment

Please install the latest Arduino IDE and the ESP32 support package of Arduino IDE (refer to the instructions).

3.2 Download the ESP32FS plugin

Download the compressed package of the ESP32FS plug-in at:

Releases · me-no-dev/arduino-esp32fs-plugin
GitHub



Latest release
1.0
278ffa0
Verified

Release for esptool_py

me-no-dev released this on Jan 15

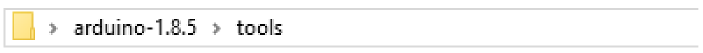
Updates the path to esptool to work with the latest Arduino for ESP32

Assets 3

- ESP32FS-1.0.zip
- Source code (zip)
- Source code (tar.gz)

Download ESP32 SPIFFS file system upload plug-in for Arduino IDE

Go to the Arduino IDE directory and open the "Tools" folder.

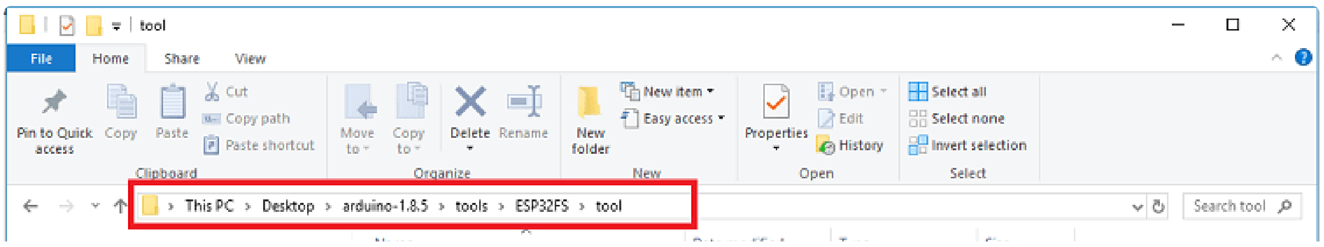


› arduino-1.8.5 › tools

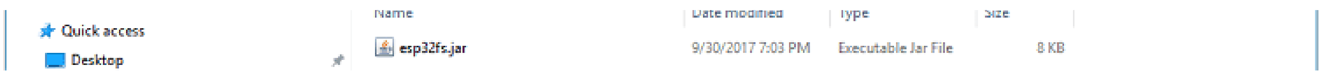
Use Arduino IDE to install ESP32 SPIFFS file system upload plug-in

Unzip the downloaded .zip folder to the Tools folder. You should have a similar folder structure :

<home_dir> / Arduino- <version> / tools / ESP32FS / tool / esp32fs.jar



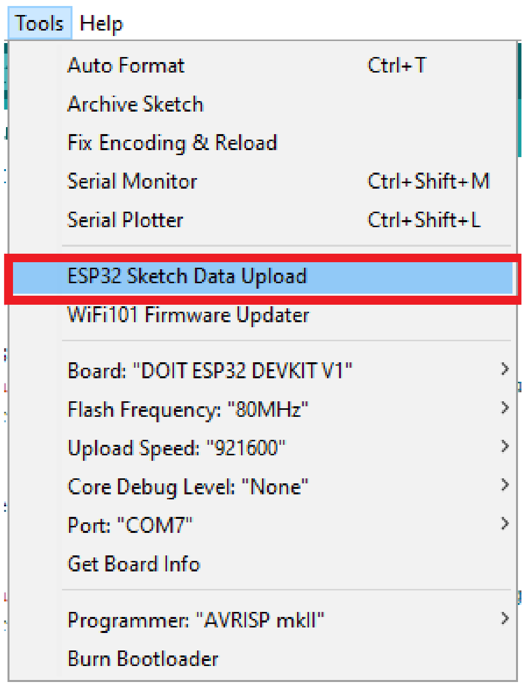
› This PC › Desktop › arduino-1.8.5 › tools › ESP32FS › tool



Use Arduino IDE to install ESP32 SPIFFS file system upload plug-in

Finally, restart the Arduino IDE.

To check whether the plug-in has been successfully installed, open the Arduino IDE. Select your ESP32 development board, go to "Tools", and then check if there is an "ESP32 Sketch Data Upload" option.

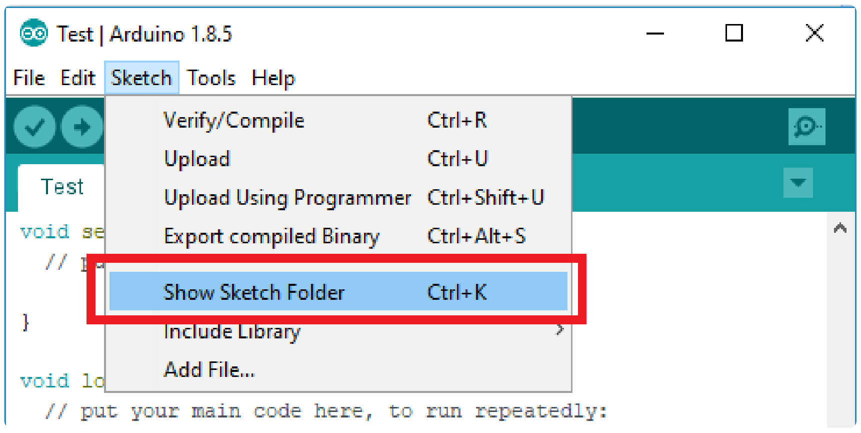


ESP32 Sketch Data Upload in Arduino IDE

4. Upload files using the file system uploader

To upload files to the ESP32 file system, follow the steps below:

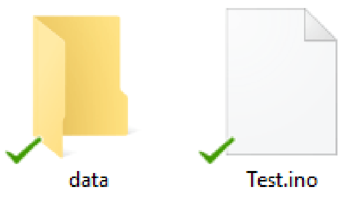
- Create an Arduino project and save
- To open the project directory, you can use the "Sketch-Show Sketch Folder" option



```
void se
  // p
}

void lo
  // put your main code here, to run repeatedly:
```

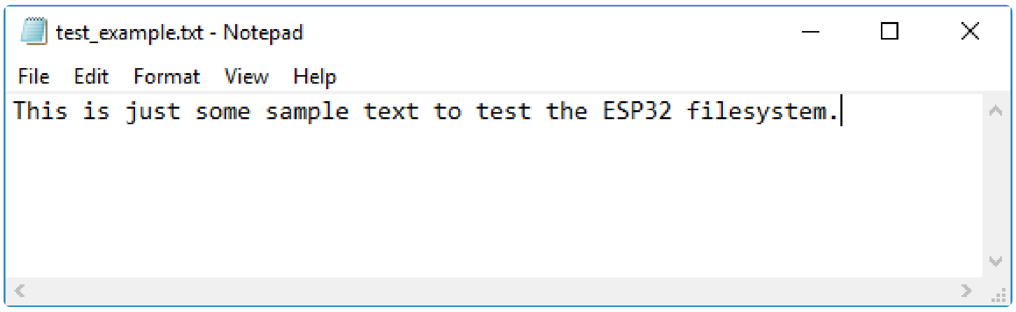
Arduino IDE displays Sketch folder to create data folder

- Inside this folder, create a new folder named "data"



"data" folder in the project directory

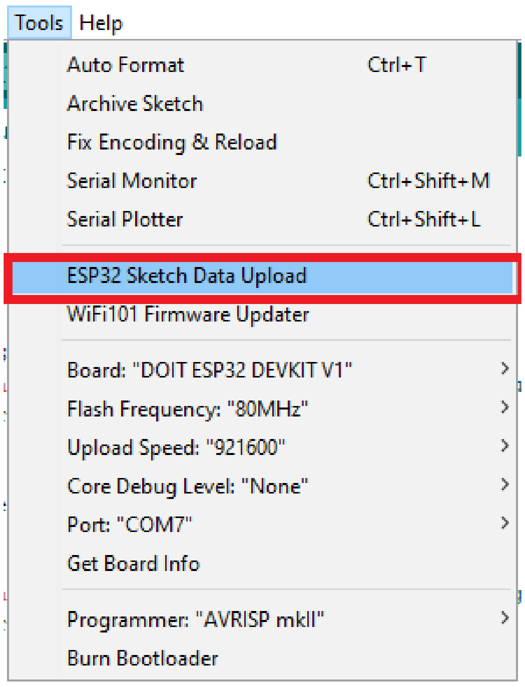
- In the "data" folder, you should put the file which you want to save into the ESP32 file system. For example, create a .txt file that contains some text named "test_example".



```
This is just some sample text to test the ESP32 filesystem.
```

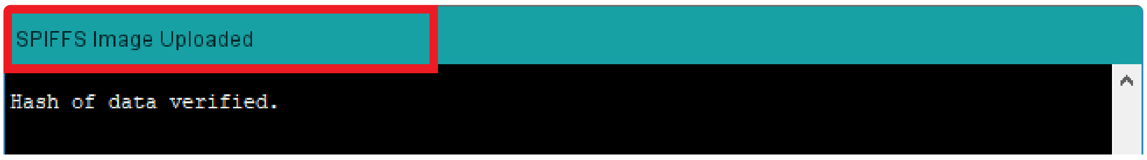
Create test sample file with Notepad

- Please go to Tools> ESP32 Sketch Data Upload in Arduino IDE



ESP32 Sketch Data Upload in Arduino IDE

When you see the "SPIFFS Image Uploaded" message, the file has been successfully uploaded to the ESP32 file system.



```
SPIFFS Image Uploaded

Hash of data verified.
```

```
Leaving...

Hard resetting...
```

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM7

The SPIFFS image has been uploaded to the ESP32 board

5. Demo of how to use the file system

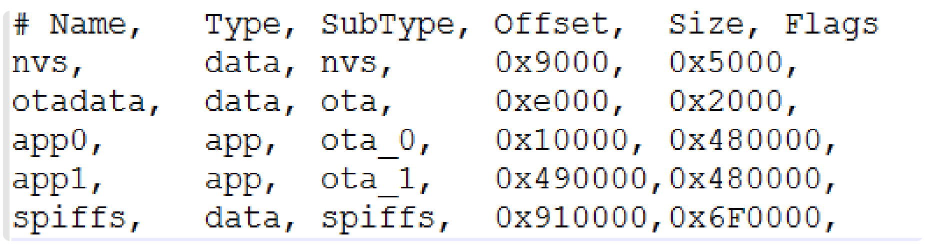
The demo of the file system comes from the official ESP32 without modification. The code implements the basic operation of "addition, deletion, modification, and check", and provides a SPI flash IO test program.

If necessary, it is recommended to directly use the code of the demo to operate ESP32 SPIFFS.

13. Add hardware partition configuration option in Arduino IDE

The flash memory of the ESP32 board has 16M, and the range of the storage address expressed in hexadecimal is: 0x0-0x01000000.

This is the partition table that has been configured by the system, as shown in the figure below:



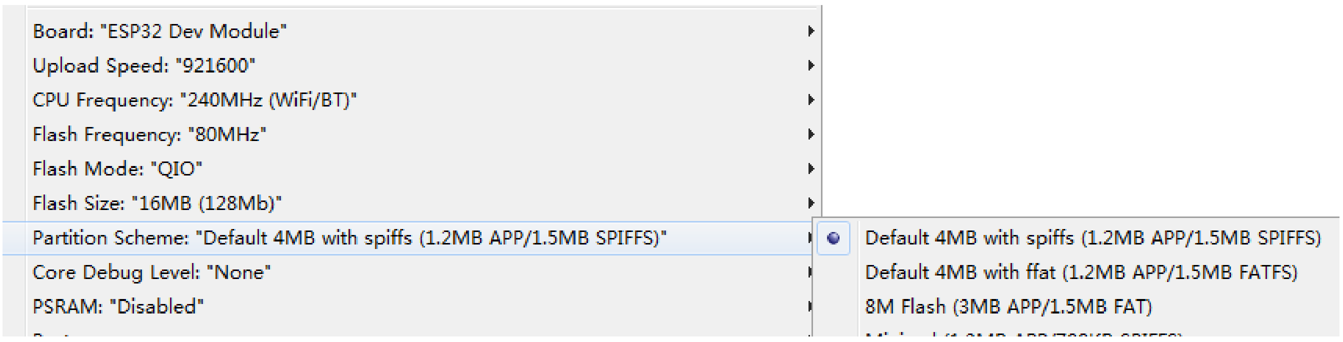
```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x480000,
app1,     app,  ota_1,   0x490000,0x480000,
spiffs,   data, spiffs,  0x910000,0x6F0000,
```

The storage location of this partition table file on the computer:

C:\Users\{YourUserName}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.*\tools\partitions\large_spiffs_16MB.csv

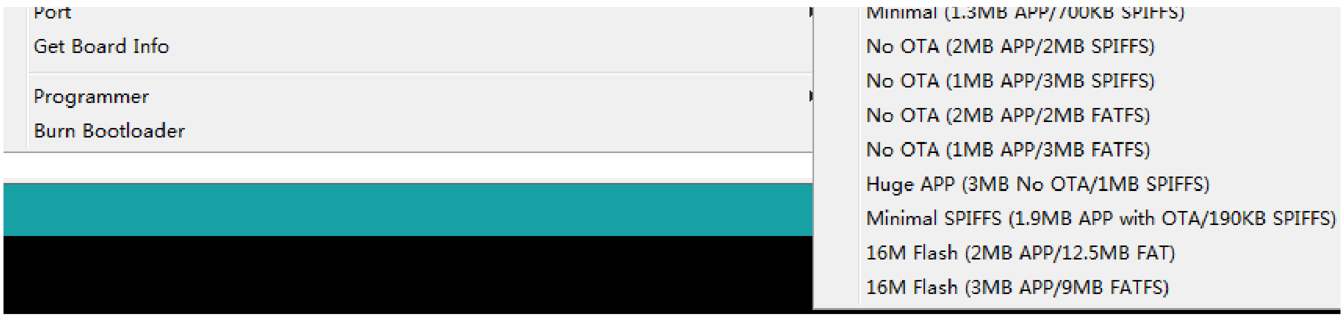
It can be seen from the above partition table: APP0 area and APP1 area are 4.5M each; the data area is SPIFFS, the size is 6.9M.

But in the Arduino IDE, this configuration is not included in the hardware partition configuration options of the ESP32 Dev Module:



Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "16MB (128Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"

● Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)
Default 4MB with ffat (1.2MB APP/1.5MB FATFS)
8M Flash (3MB APP/1.5MB FAT)

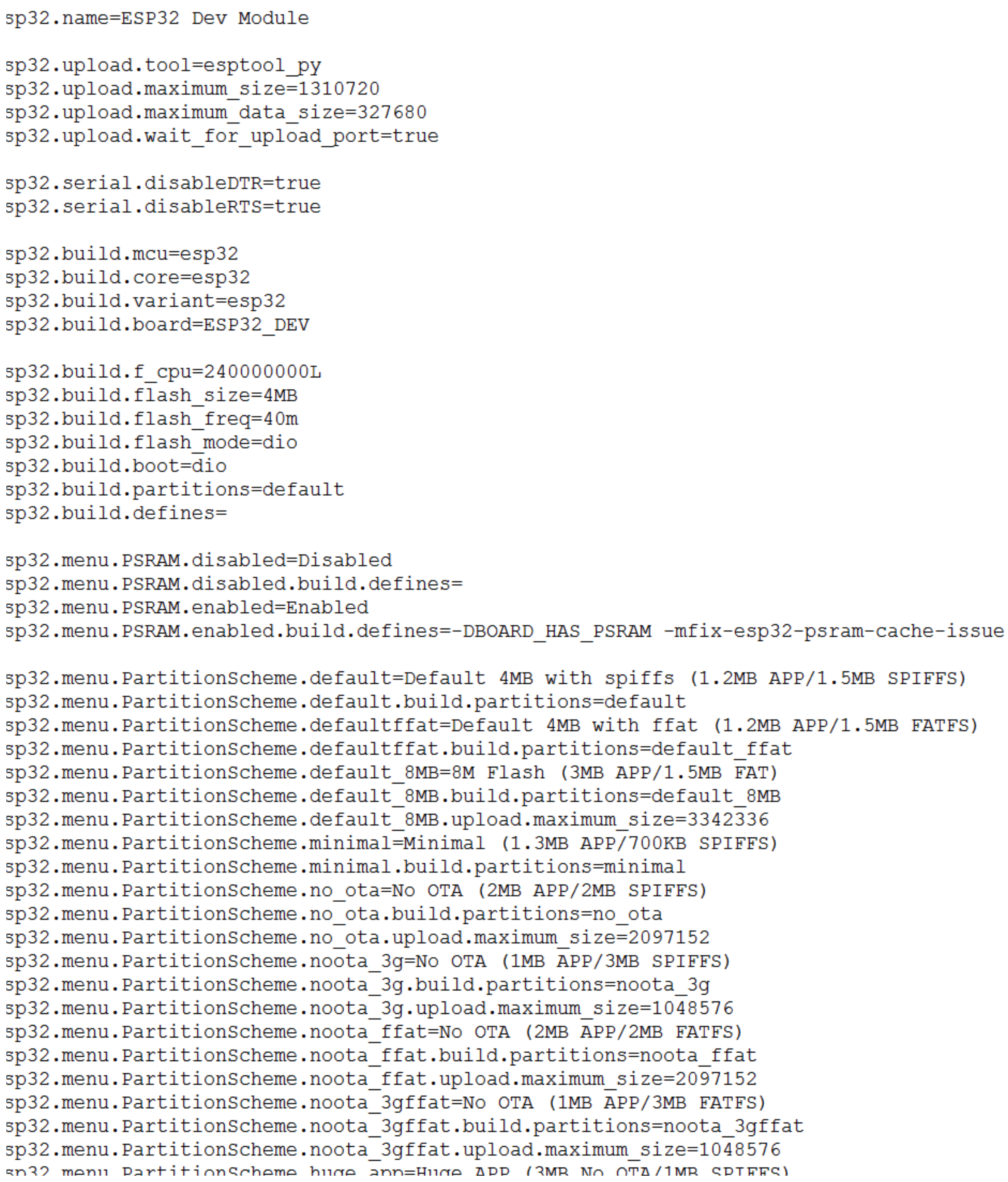

Port
Get Board Info
Programmer
Burn Bootloader
Minimal (1.3MB APP/700KB SPIFFS)
No OTA (2MB APP/2MB SPIFFS)
No OTA (1MB APP/3MB SPIFFS)
No OTA (2MB APP/2MB FATFS)
No OTA (1MB APP/3MB FATFS)
Huge APP (3MB No OTA/1MB SPIFFS)
Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
16M Flash (2MB APP/12.5MB FAT)
16M Flash (3MB APP/9MB FATFS)

We need to add this configuration to the ESP32 Dev Module.

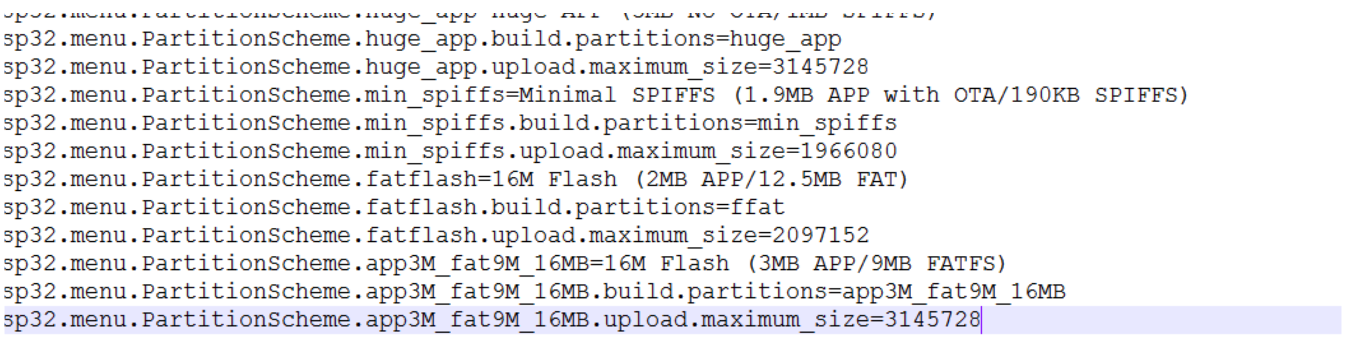
Open the development board configuration file:

C:\Users\{YourUserName}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.*\boards.txt

Locate the name of the development board: esp32.name=ESP32 Dev Module, as shown in the figure below:



```
sp32.name=ESP32 Dev Module

sp32.upload.tool=esptool_py
sp32.upload.maximum_size=1310720
sp32.upload.maximum_data_size=327680
sp32.upload.wait_for_upload_port=true

sp32.serial.disableDTR=true
sp32.serial.disableRTS=true

sp32.build.mcu=esp32
sp32.build.core=esp32
sp32.build.variant=esp32
sp32.build.board=ESP32_DEV

sp32.build.f_cpu=240000000L
sp32.build.flash_size=4MB
sp32.build.flash_freq=40m
sp32.build.flash_mode=dio
sp32.build.boot=dio
sp32.build.partitions=default
sp32.build.defines=

sp32.menu.PSRAM.disabled=Disabled
sp32.menu.PSRAM.disabled.build.defines=
sp32.menu.PSRAM.enabled=Enabled
sp32.menu.PSRAM.enabled.build.defines=-DBOARD_HAS_PSRAM -mfix-esp32-psram-cache-issue

sp32.menu.PartitionScheme.default=Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)
sp32.menu.PartitionScheme.default.build.partitions=default
sp32.menu.PartitionScheme.defaultffat=Default 4MB with ffat (1.2MB APP/1.5MB FATFS)
sp32.menu.PartitionScheme.defaultffat.build.partitions=default_ffat
sp32.menu.PartitionScheme.default_8MB=8M Flash (3MB APP/1.5MB FAT)
sp32.menu.PartitionScheme.default_8MB.build.partitions=default_8MB
sp32.menu.PartitionScheme.default_8MB.upload.maximum_size=3342336
sp32.menu.PartitionScheme.minimal=Minimal (1.3MB APP/700KB SPIFFS)
sp32.menu.PartitionScheme.minimal.build.partitions=minimal
sp32.menu.PartitionScheme.no_ota=No OTA (2MB APP/2MB SPIFFS)
sp32.menu.PartitionScheme.no_ota.build.partitions=no_ota
sp32.menu.PartitionScheme.no_ota.upload.maximum_size=2097152
sp32.menu.PartitionScheme.noota_3g=No OTA (1MB APP/3MB SPIFFS)
sp32.menu.PartitionScheme.noota_3g.build.partitions=noota_3g
sp32.menu.PartitionScheme.noota_3g.upload.maximum_size=1048576
sp32.menu.PartitionScheme.noota_ffat=No OTA (2MB APP/2MB FATFS)
sp32.menu.PartitionScheme.noota_ffat.build.partitions=noota_ffat
sp32.menu.PartitionScheme.noota_ffat.upload.maximum_size=2097152
sp32.menu.PartitionScheme.noota_3gffat=No OTA (1MB APP/3MB FATFS)
sp32.menu.PartitionScheme.noota_3gffat.build.partitions=noota_3gffat
sp32.menu.PartitionScheme.noota_3gffat.upload.maximum_size=1048576
sp32.menu.PartitionScheme.huge_app=Huge APP (3MB No OTA/1MB SPIFFS)
```



```
sp32.menu.PartitionScheme.huge_app=Huge APP (3MB No OTA/1MB SPIFFS)
sp32.menu.PartitionScheme.huge_app.build.partitions=huge_app
sp32.menu.PartitionScheme.huge_app.upload.maximum_size=3145728
sp32.menu.PartitionScheme.min_spiffs=Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
sp32.menu.PartitionScheme.min_spiffs.build.partitions=min_spiffs
sp32.menu.PartitionScheme.min_spiffs.upload.maximum_size=1966080
sp32.menu.PartitionScheme.fatflash=16M Flash (2MB APP/12.5MB FAT)
sp32.menu.PartitionScheme.fatflash.build.partitions=ffat
sp32.menu.PartitionScheme.fatflash.upload.maximum_size=2097152
sp32.menu.PartitionScheme.app3M_fat9M_16MB=16M Flash (3MB APP/9MB FATFS)
sp32.menu.PartitionScheme.app3M_fat9M_16MB.build.partitions=app3M_fat9M_16MB
sp32.menu.PartitionScheme.app3M_fat9M_16MB.upload.maximum_size=3145728
```

The last line of the ESP32 Dev Module partition configuration in this configuration file is:

```
esp32.menu.PartitionScheme.app3M_fat9M_16MB.upload.maximum_size=3145728
```

Add the following 3 lines of code below this line:

```
esp32.menu.PartitionScheme.large_spiffs=Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)
esp32.menu.PartitionScheme.large_spiffs.build.partitions=large_spiffs_16MB
esp32.menu.PartitionScheme.large_spiffs.upload.maximum_size=4685824
```

The following explains the meaning of the three lines of code:

```
esp32.menu.PartitionScheme.large_spiffs=Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)
```

The name of the ESP32 partition configuration, we named it Biboard V0 (4.5M APP with OTA /6.9 MB SPIFFS), or it can be replaced with other names you are familiar with.

```
esp32.menu.PartitionScheme.large_spiffs.build.partitions=large_spiffs_16MB
```

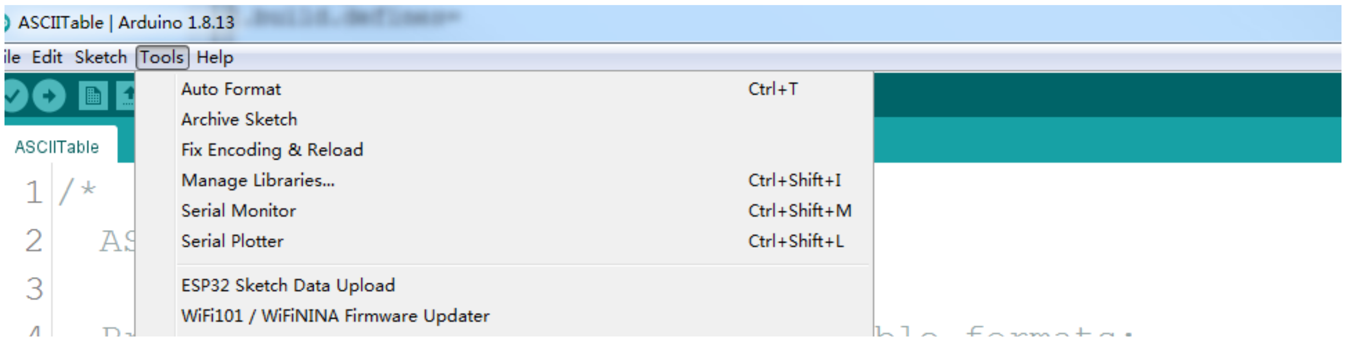
The partition configuration file information is the file large_spiffs_16MB.csv . You can also write a partition file to adjust the file size of the APP and data area.

```
esp32.menu.PartitionScheme.large_spiffs.upload.maximum_size=4685824
```

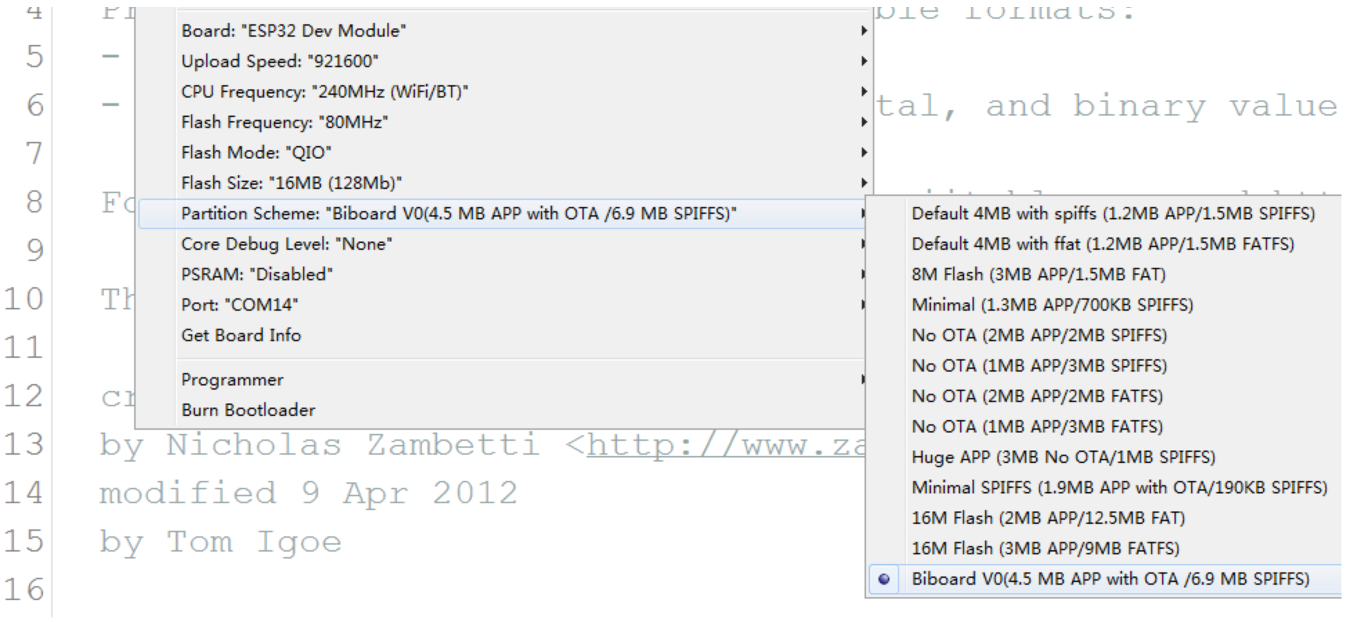
This line of code specifies that the maximum upload program size is 4685824 bytes.

Let's try to compile a simple program to test whether the above configuration is set successfully.

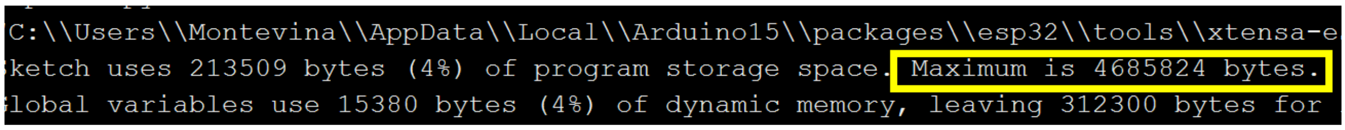
Reopen the Arduino IDE, we can see the Biboard just configured:



ASCIITable | Arduino 1.8.13
ile Edit Sketch Tools Help
ASCIITable
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP32 Sketch Data Upload
WiFi101 / WiFiNINA Firmware Updater


Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "16MB (128Mb)"
Partition Scheme: "Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"
Port: "COM14"
Get Board Info
Programmer
Burn Bootloader
Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)
Default 4MB with ffat (1.2MB APP/1.5MB FATFS)
8M Flash (3MB APP/1.5MB FAT)
Minimal (1.3MB APP/700KB SPIFFS)
No OTA (2MB APP/2MB SPIFFS)
No OTA (1MB APP/3MB SPIFFS)
No OTA (2MB APP/2MB FATFS)
No OTA (1MB APP/3MB FATFS)
Huge APP (3MB No OTA/1MB SPIFFS)
Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
16M Flash (2MB APP/12.5MB FAT)
16M Flash (3MB APP/9MB FATFS)
● Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)

After compiling the program, the result is as shown in the figure below:



```
C:\\Users\\Montevina\\AppData\\Local\\Arduino15\\packages\\esp32\\tools\\xtensa-e
ketch uses 213509 bytes (4%) of program storage space. Maximum is 4685824 bytes.
lobal variables use 15380 bytes (4%) of dynamic memory, leaving 312300 bytes for
```

Compilation is complete, using 213KB of Flash (4%), and the maximum usable size is 4,685,824 bytes.

In this passage，“4685824 bytes” is specified in the third line of code just added to the configuration file.

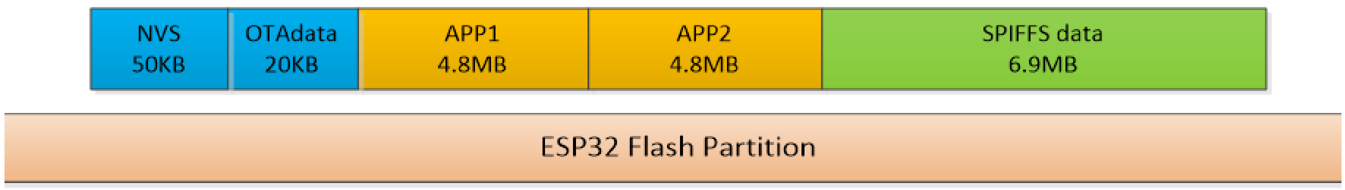
So far you have completed the configuration of the development board with the largest flash memory space in Arduino IDE.

14.The usage of Wi-Fi OTA(Over-The-Air)

The Arduino demo of ESP32 provides the function of OTA (updating/uploading a new program to ESP32 using Wi-Fi)

1. What is OTA?

The configuration of our BiBoard is 16MB Flash, and the specific partitions are as follows:



OTA mainly operates OTA data areas, namely APP1 and APP2 areas. The principle is:

- BiBoard runs the firmware with OTA function, at this time, the boot points to the APP1 area.

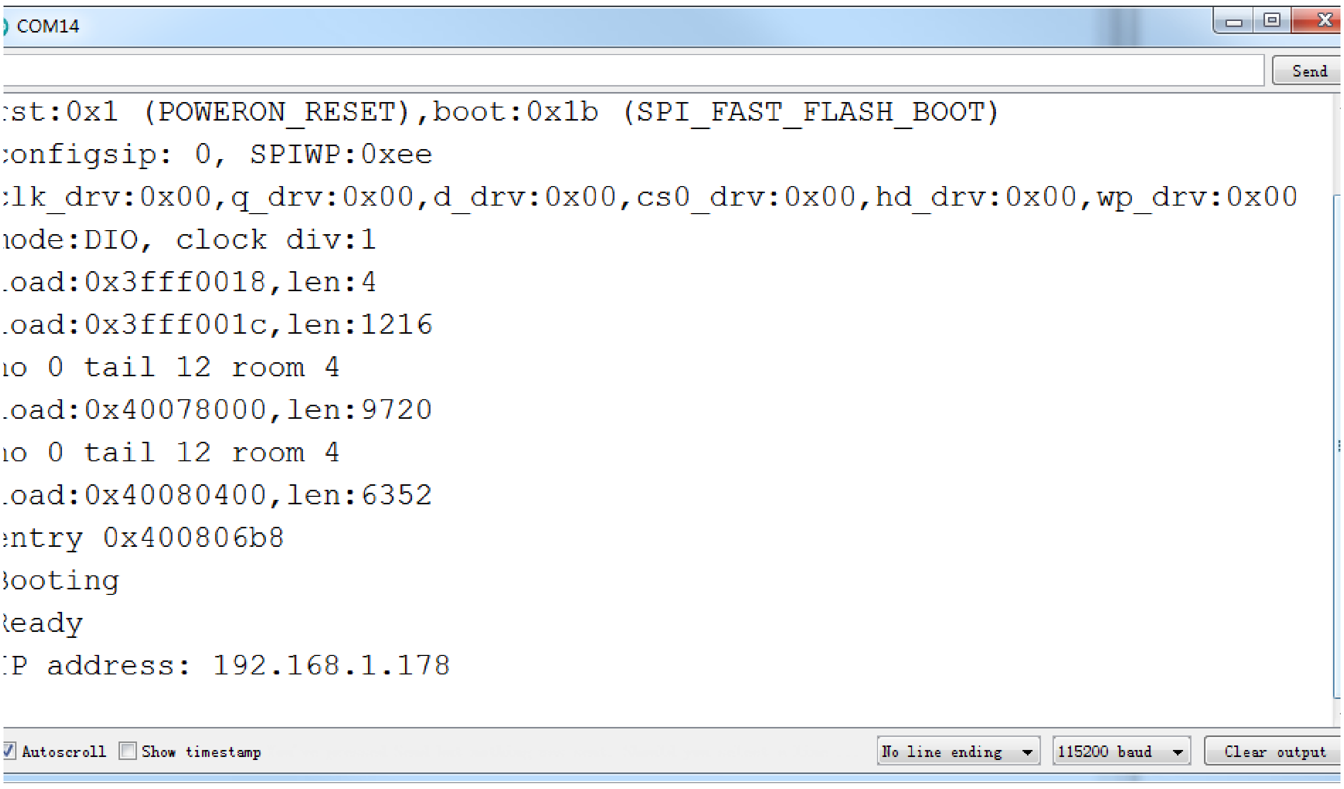
- The OTA command is sent to ESP32 via Wi-Fi, and the binary file of the upgrade program is transferred to the APP2 area.
- If the transmission of APP2 is completed and the verification is successful, OTAdata points to the APP2 area, and the next time it starts from the updated firmware area (APP2), the APP1 data is retained. Next time, OTA will write to APP1 area to overwrite the old firmware.
- If the transmission of APP2 is not completed due to a network transmission error, because APP2 has not passed the verification, OTAdata does not point to the APP2 area. The program in the APP1 area will still be executed after the reset is started, and the damaged APP2 area will be completely erased and overwritten during the next OTA.

OTA operation in Arduino program

In the demo, first configure WiFi, and configure the WiFi mode as STA (Station, base station mode). Enable the WiFi function and pass in the account password "WiFi.begin(ssid, password);"

```
Serial.println("Booting");
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.waitForConnectResult() != WL_CONNECTED) {
  Serial.println("Connection Failed! Rebooting...");
  delay(5000);
  ESP.restart();
```

When the Wi-Fi is successfully connected, the IP address will be printed via the serial port; if the connection is wrong, the ESP32 will restart.



```
:st:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
:onfigsip: 0, SPIWP:0xee
:lk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
iode:DIO, clock div:1
.oad:0x3fff0018,len:4
.oad:0x3fff001c,len:1216
io 0 tail 12 room 4
.oad:0x40078000,len:9720
io 0 tail 12 room 4
.oad:0x40080400,len:6352
:ntry 0x400806b8
3ooting
Ready
:P address: 192.168.1.178
```

In the demo, you can configure the port number, the OTA key or the hash of the key, and the area and type of the OTA (commented by default).

```
// Port defaults to 3232
// ArduinoOTA.setPort(3232);

// Hostname defaults to esp3232-[MAC]
// ArduinoOTA.setHostname("myesp32");

// No authentication by default
// ArduinoOTA.setPassword("admin");

// Password can be set with it's md5 value as well
// MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
// ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
```

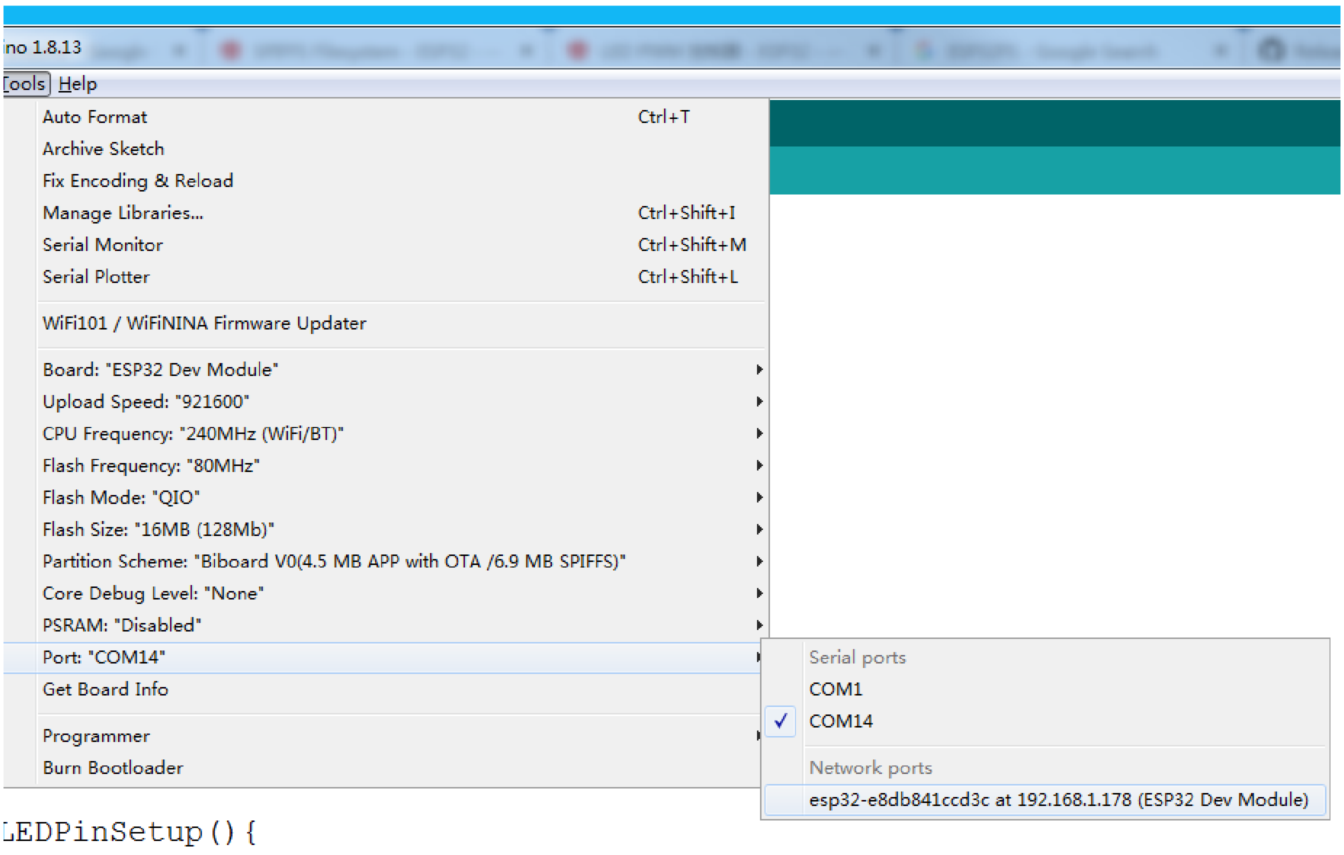
The following are several code snippets similar to callback functions, which are used to judge the state of each stage of OTA.

```
ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.e
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });

ArduinoOTA.begin();
```

After configuring according to the demo, call "ArduinoOTA.handle();" in the loop function. The following analogWrite function is to distinguish the effects of different firmware updates (by changing the value).

```
void loop() {
  ArduinoOTA.handle();
  analogWrite(2, 127);    // test OTA firmware
}
```

The first time you use the serial port to download, the python tool "esptool" is called. You can use OTA after the download is complete. In the port options, you will find an extra port based on the IP address, which is the OTA address.



ino 1.8.13

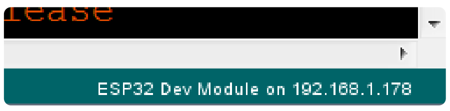
Tools Help

```
Auto Format                                              Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries...                                      Ctrl+Shift+I
Serial Monitor                                           Ctrl+Shift+M
Serial Plotter                                           Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "16MB (128Mb)"
Partition Scheme: "Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"
Port: "COM14"
Get Board Info
Programmer
Burn Bootloader
```

```
Serial ports
COM1
✓ COM14
Network ports
esp32-e8db841ccd3c at 192.168.1.178 (ESP32 Dev Module)
```

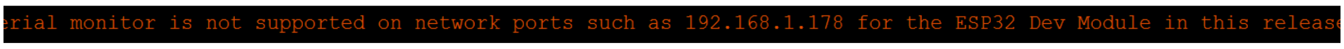
LEDPinSetup(){

Select this address, the lower right corner is the IP address of ESP32 Dev Module on your BiBoard (192.168.1.178)



ESP32 Dev Module on 192.168.1.178

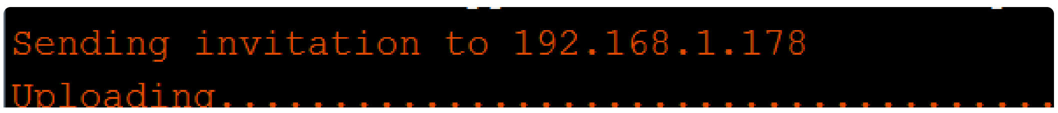
At the same time, a warning will pop up: "Serial monitor is not supported on network ports such as 192.168.1.178 for the ESP32 Dev Module in this release".



```
erial monitor is not supported on network ports such as 192.168.1.178 for the ESP32 Dev Module in this release
```

The ESP32 OTA of Arduino is only suitable for updating the program and cannot complete the serial port debugging work. If you need to debug BiBoard, please connect the USB-C interface.

Download the program, as shown in the figure.



```
Sending invitation to 192.168.1.178
Uploading........................................
```

Modules

USB Downloader (CH340C)

Introduction

The Nyboard V1 used by Bittle uses the Atmel ATMEGA328P controller, which only supports only one serial port. We separate the serial port of Nyboard to support more modules. The pins of the serial port are compatible with the 6-pin Arduino Pro Mini. Pin definitions are shown in the table below:

Pin No.	Name	Usage
1	DTR	Modem signal DTR，reset NyBoard after serial download finished.
2	RX	ATMEGA328P RX (receive)
3	TX	ATMEGA328P TX (send)
4	5V	5V power for MCU and chips
5	GND	Ground
6	GND	Gron

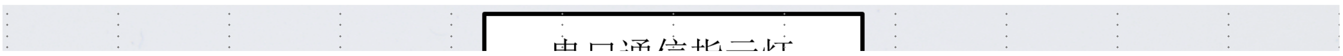
The default serial baud rate is **115200bps**.

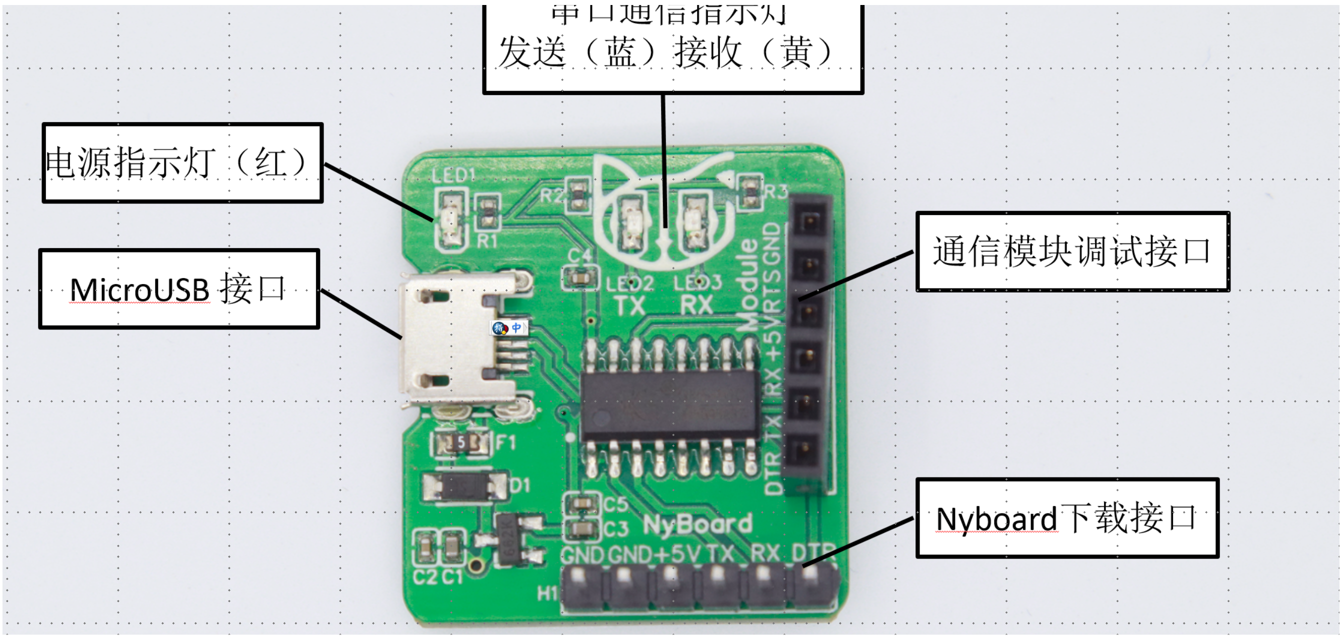
There're 3 communication modules for the NyBoard V1:

- USB downloader
- ESP8266 WiFi module
- Bluetooth dual mode（EDR & BLE）JDY-23

USB Downloader

The module uses CH340C USB bridge. Windows10, Linux and MacOS are all drive-free.





Insert the 6-pin(H1) to the NyBoard's downloader and then connect the module with the Micro-USB cable in the package to your PC. Open the Arduino IDE and select the corresponding COM port. You can download the programme and communicate with the NyBoard.

Another usage of the downloader is update the WiFi and bluetooth module's firmware. Plug the module in the female 6-pin socket. We changed the TX and RX pin and make one GND pin to the DTR to reset the WiFi module. The details are written in the chapter of the WiFi and bluetooth manual.

Do not plug the NyBoard and the module at the same time! That will make a mistake.

Bluetooth Dual Mode

Bluetooth Module

The Bluetooth module is a standard transparent transmission module, which sends serial port data to devices connected to Bluetooth.

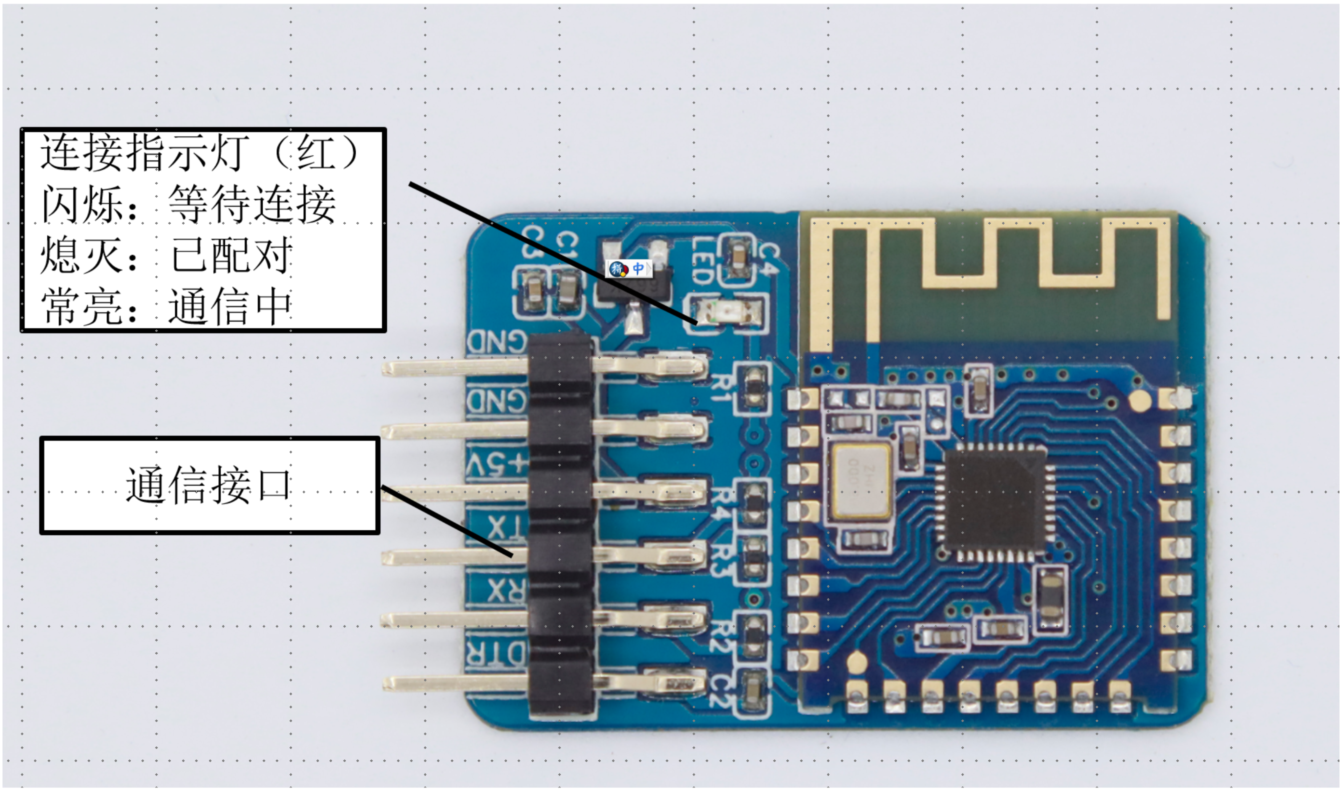
Insert the Bluetooth module into the Nyboard downloader socket and open the serial downloader to search for the Bluetooth signal of Bittle-xxxx (random number). The default password for pairing is “1234” or “0000” (for the convenience of connection, the default password for the newer version is set to “0000”). Please enter the password before pairing. After the pairing is successful, the system will assign a serial port number, and you can select the corresponding serial port number on the Arduino.

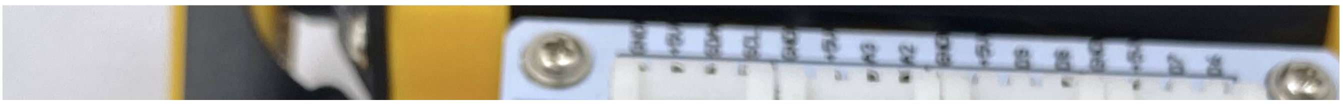
For Win10 users, the system will assign the "incoming" COM port and the "outgoing" COM port to Bluetooth, please use the "outgoing" COM port. For details, please check in the "Bluetooth Settings" of Win10.

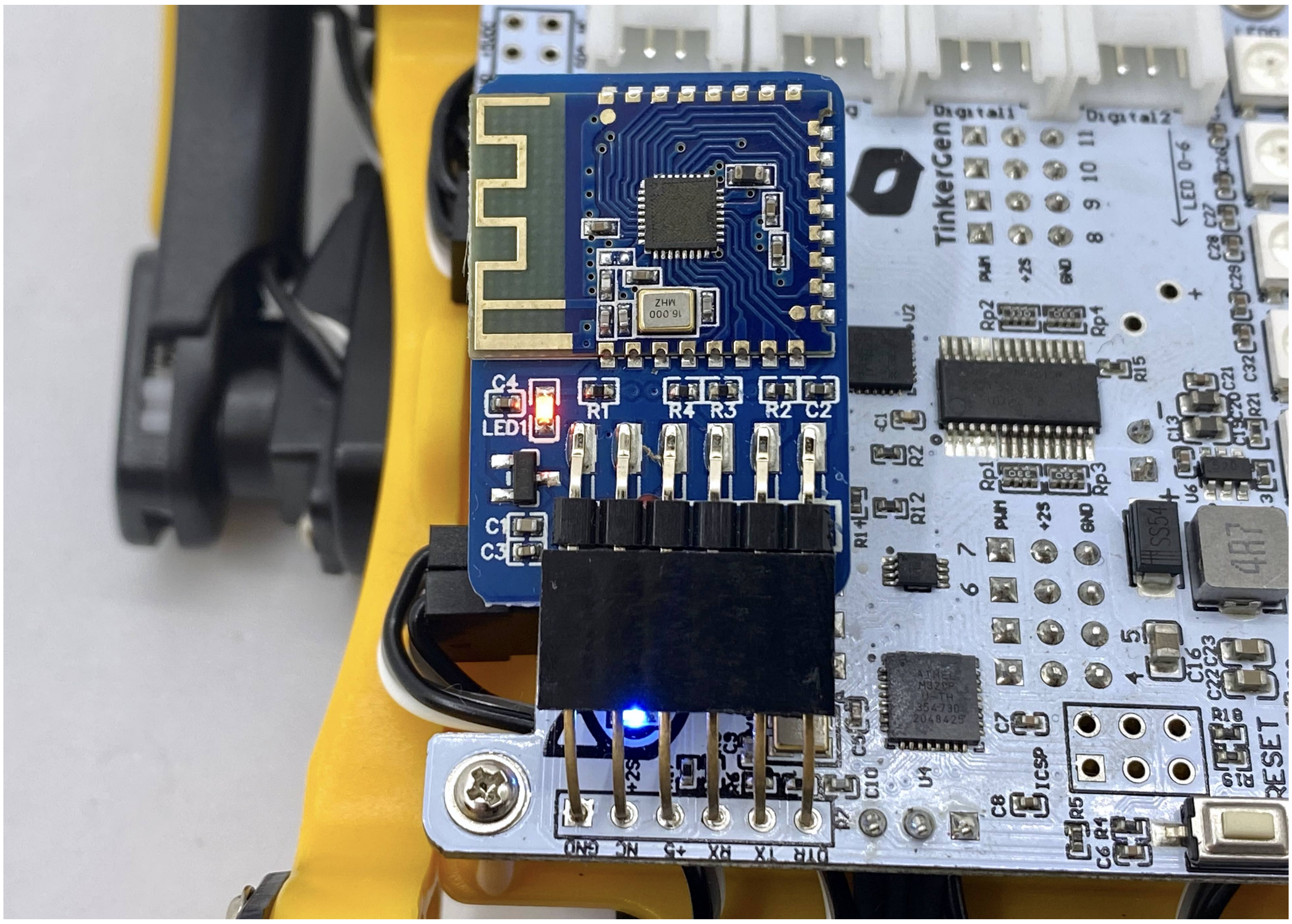
If you want to use the BLE connection, please scan and connect to BittleBLE-xxxx (random number), and you can use the Lightblue or other tools to communicate with the Bittle.

If you want to configure the Bluetooth module, please refer to "JDY-23 AT Command List". Plug the Bluetooth module into the USB communication module debugging interface. The commonly used commands are listed below:

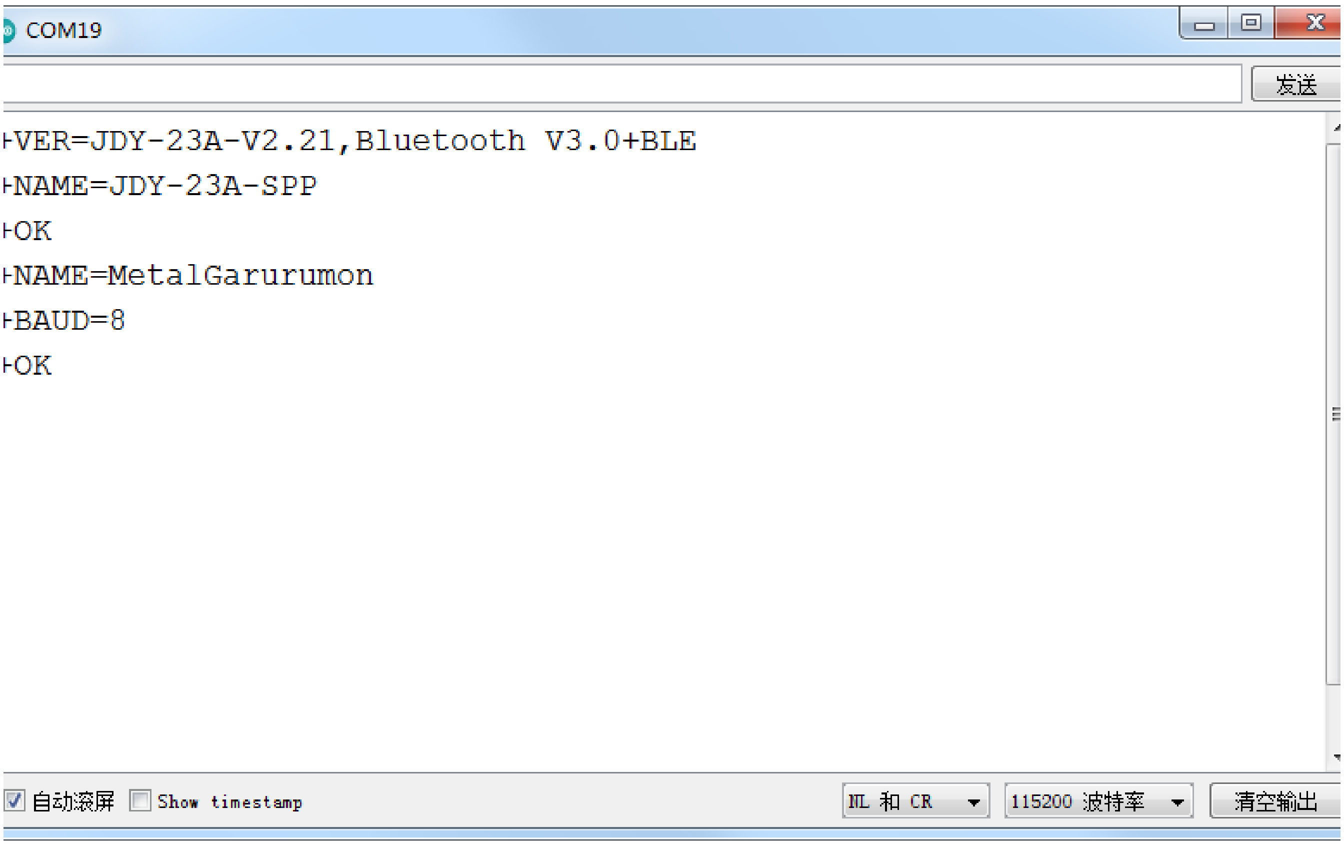
Usage	Command	Demo
Check BT module version	AT+VER	AT+VER >+VER=JDY-23A-V2.21,Bluetooth V3.0+BLE（BT module version infomation）
Check BT broadcast name	AT+NAME	AT+NAME >+NAME=BITTLE
Change BT broadcast name	AT+NAME(名字)	AT+NAMEPiggy >+OK AT+NAME >+NAME=Piggy
Check serial baud rate	AT+BAUD	AT+BAUD >+BAUD=8 （8 = 115200，7=57600）
Change serial baud rate	AT+BAUD	AT+BAUD7 >+OK （Set serial monitor to 57600） AT+BAUD >+BAUD=7







When you use serial terminal like "Arduino serial monitor" to set JDY-23 with AT commands, you must set "NL and CR", or JDY-23 module will not identify any AT command you send.



```
+VER=JDY-23A-V2.21,Bluetooth V3.0+BLE
+NAME=JDY-23A-SPP
+OK
+NAME=MetalGarurumon
+BAUD=8
+OK
```

NL 和 CR

115200 波特率

WiFi ESP8266

WiFi module ESP8266

Introduction

This module uses ESP8266EX's official model ESP-WROOM-02D, 4MB QSPI Flash. The module is certified by FCC in the United States, CE-RED in Europe, TELEC in Japan, and KC in South Korea.

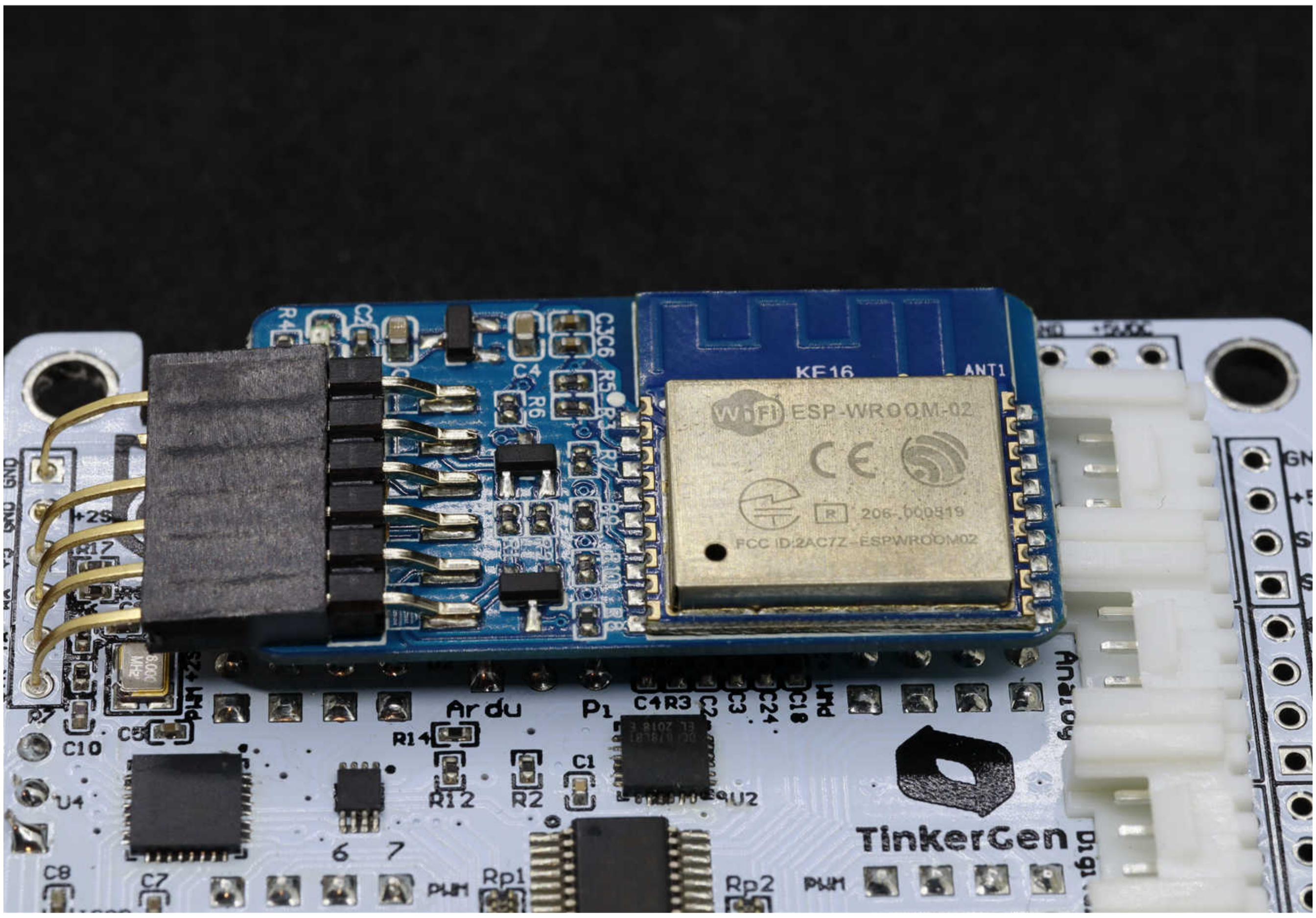
The module is fully opened, you can program it separately. This is not a simple transparent transmission module.

Module Functions

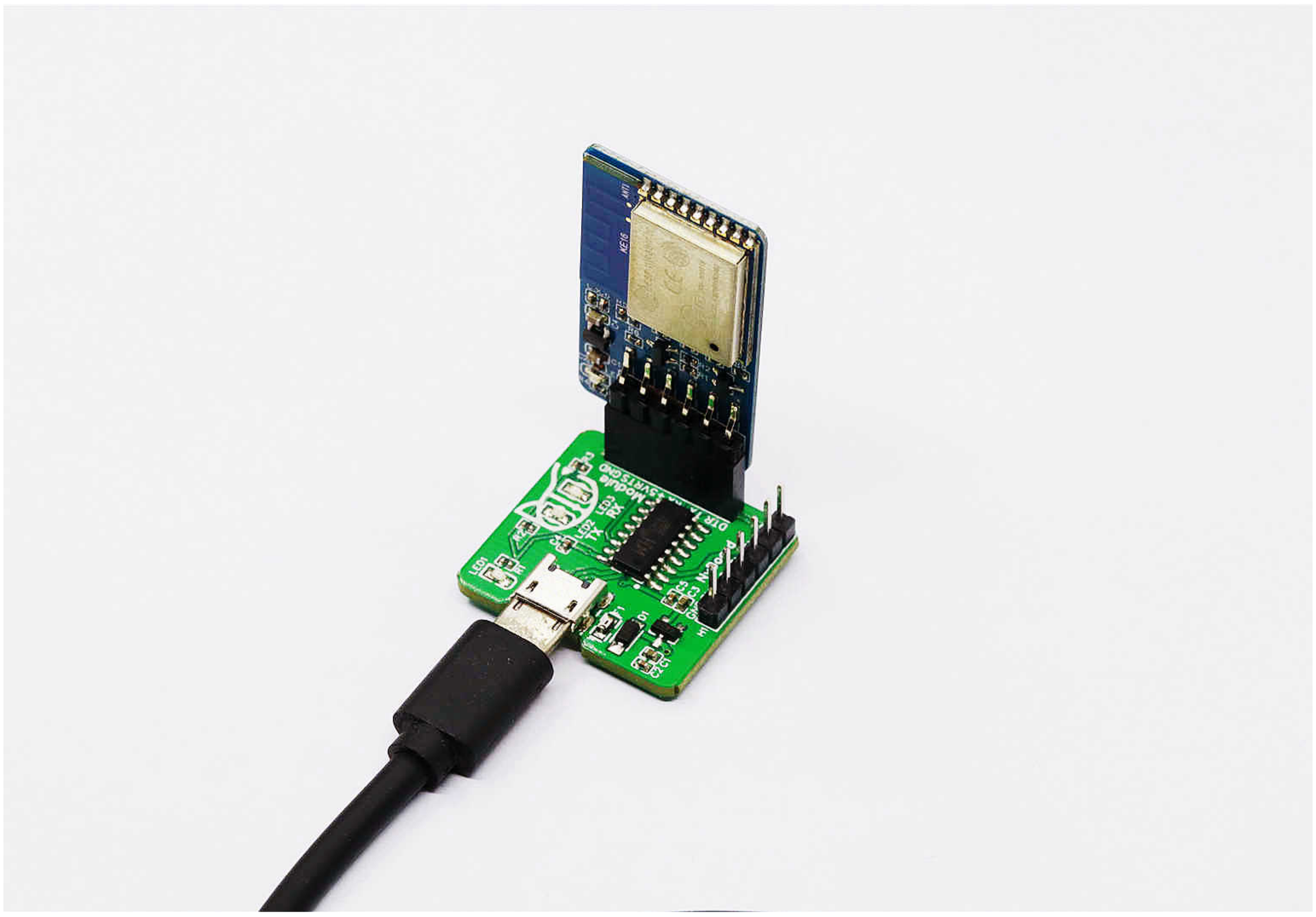
The module includes an automatic download circuit and a communication module. The automatic download circuit refers to the official recommendation to use 2 S8050 transistors to receive the RTS and DTR signals from the CH340C downloader and trigger the download sequence.

Assembly

Connect to the NyBoard：



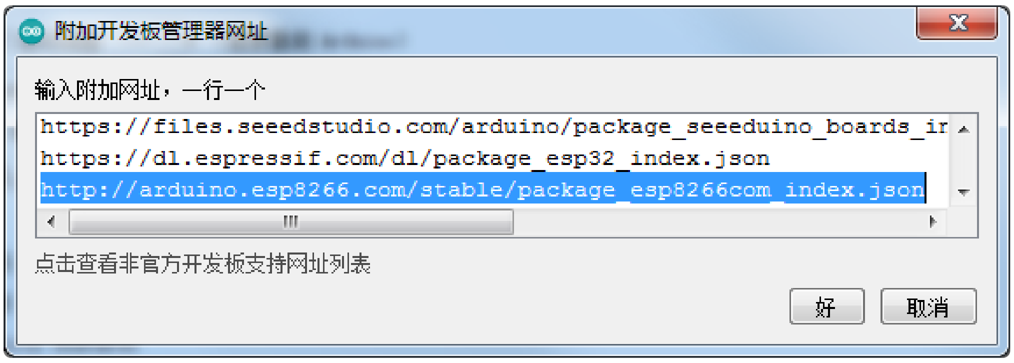
Update sketches through USB downloader：



Development Environment Settings

We use the Arduino as the development environment.

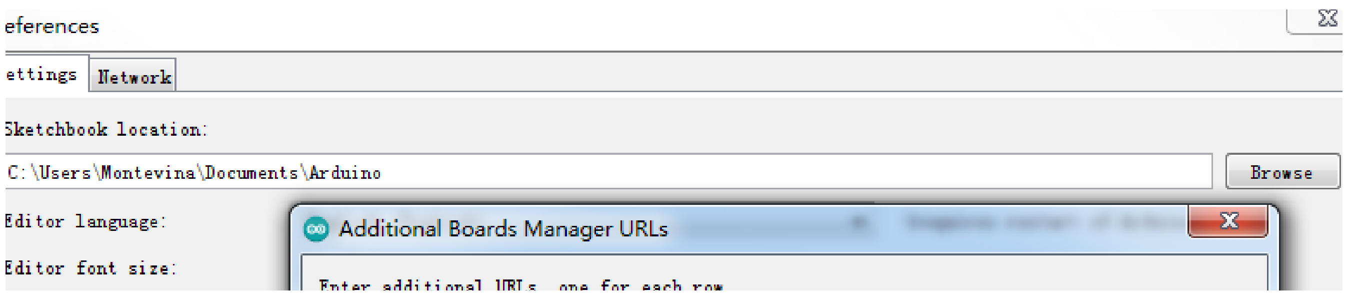
2.1 Add ESP8266 source to the board manager



输入附加网址，一行一个

```
https://files.seeedstudio.com/arduino/package_seeeduino_boards_ir
https://dl.espressif.com/dl/package_esp32_index.json
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

点击查看非官方开发板支持网址列表



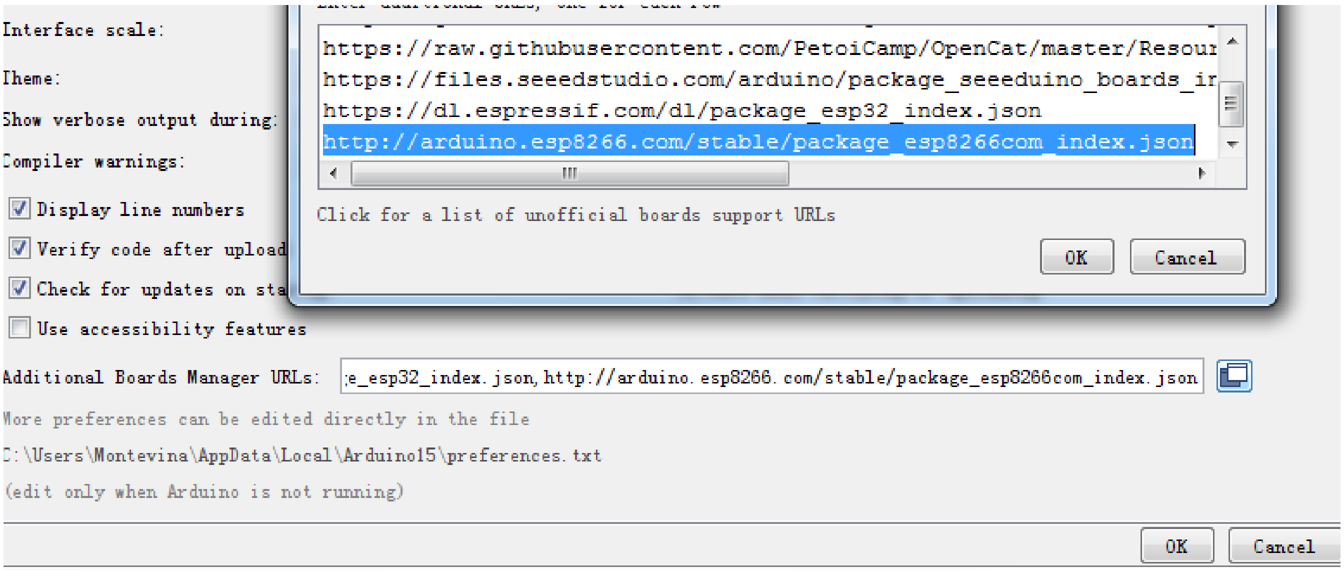
Sketchbook location:

C:\Users\Montevina\Documents\Arduino

Editor language:

Editor font size:

Enter additional URLs, one for each row


Interface scale:

Theme:

Show verbose output during:

Compiler warnings:

☑ Display line numbers

☑ Verify code after upload

☑ Check for updates on sta

☐ Use accessibility features

Additional Boards Manager URLs: ;e_esp32_index.json,http://arduino.esp8266.com/stable/package_esp8266com_index.json

More preferences can be edited directly in the file

C:\Users\Montevina\AppData\Local\Arduino15\preferences.txt

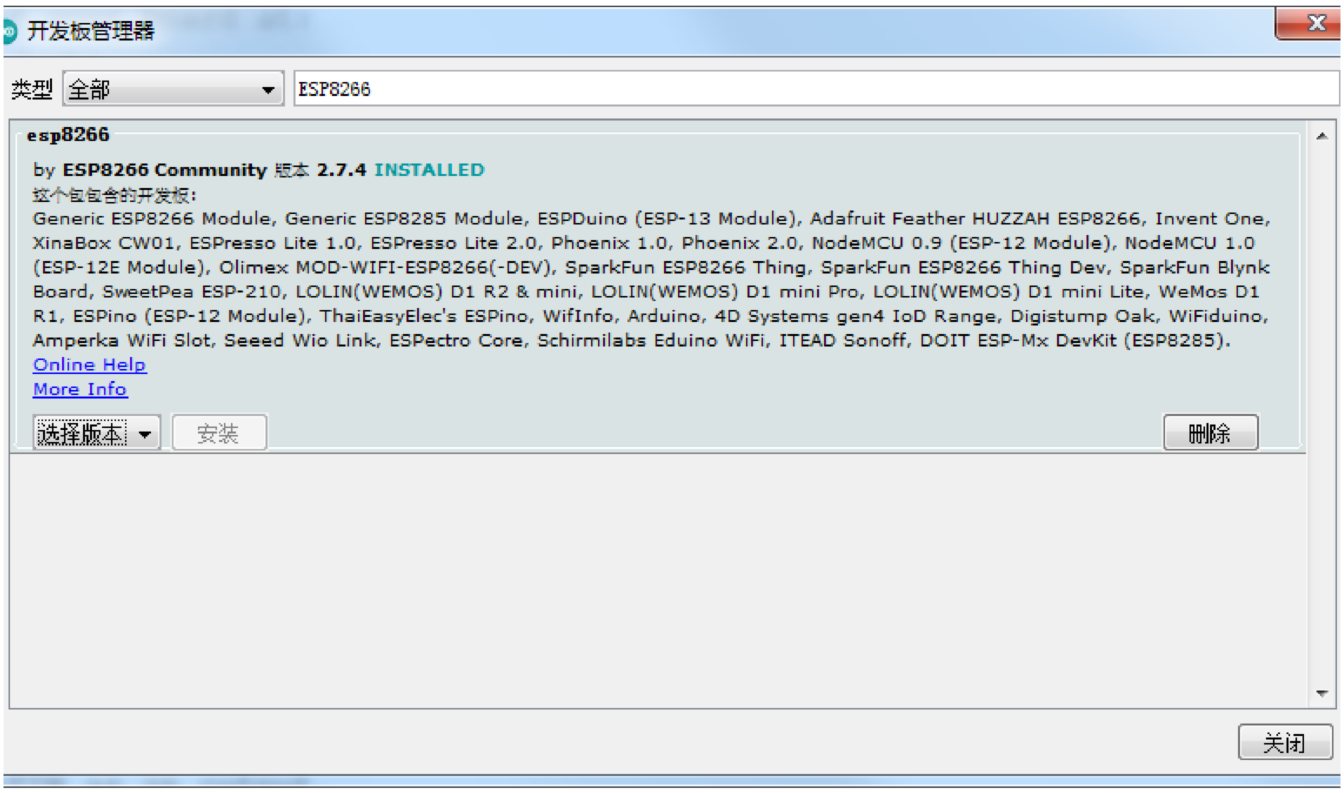
(edit only when Arduino is not running)

```
https://raw.githubusercontent.com/PetoiCamp/OpenCat/master/Resour
https://files.seeedstudio.com/arduino/package_seeeduino_boards_ir
https://dl.espressif.com/dl/package_esp32_index.json
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Click for a list of unofficial boards support URLs

URL：http://arduino.esp8266.com/stable/package_esp8266com_index.json. Paste it into the URL of the additional development board in the Arduino preferences.

Then open the "Board Manager" and enter ESP8266 or 8266 to search for the board support package:


开发板管理器

类型 全部 ESP8266

esp8266

by **ESP8266 Community** 版本 **2.7.4** INSTALLED

这个包包含的开发板:

Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmilabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).

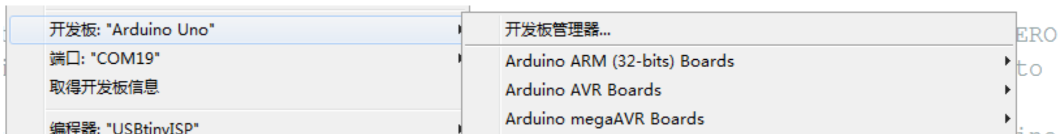
Online Help

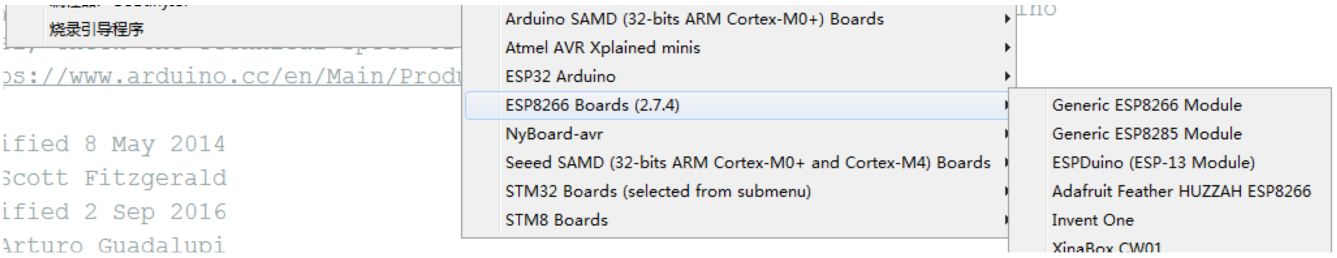
More Info

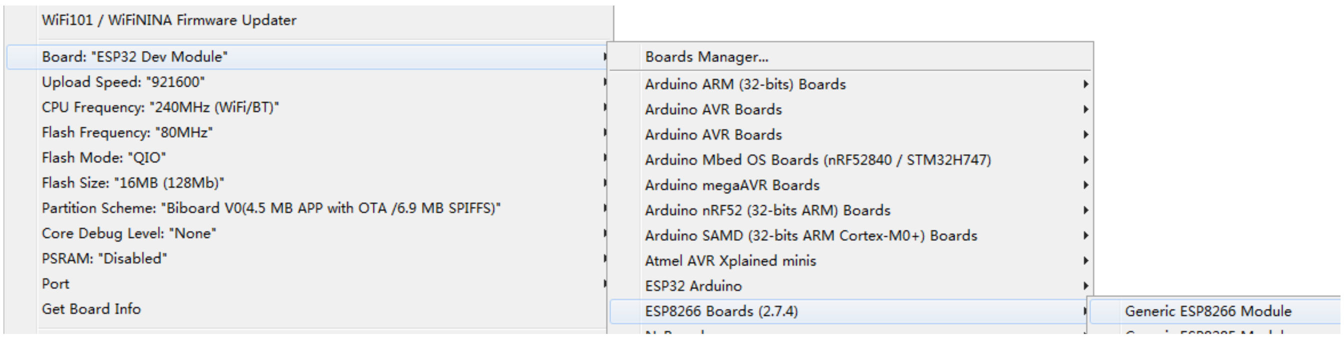
Download the package ESP8266 by ESP8266 Community.

Configuration of the Module

After the board support package downloaded, we select ESP8266 Board (current version: 2.74) -> Generic ESP8266 Module.



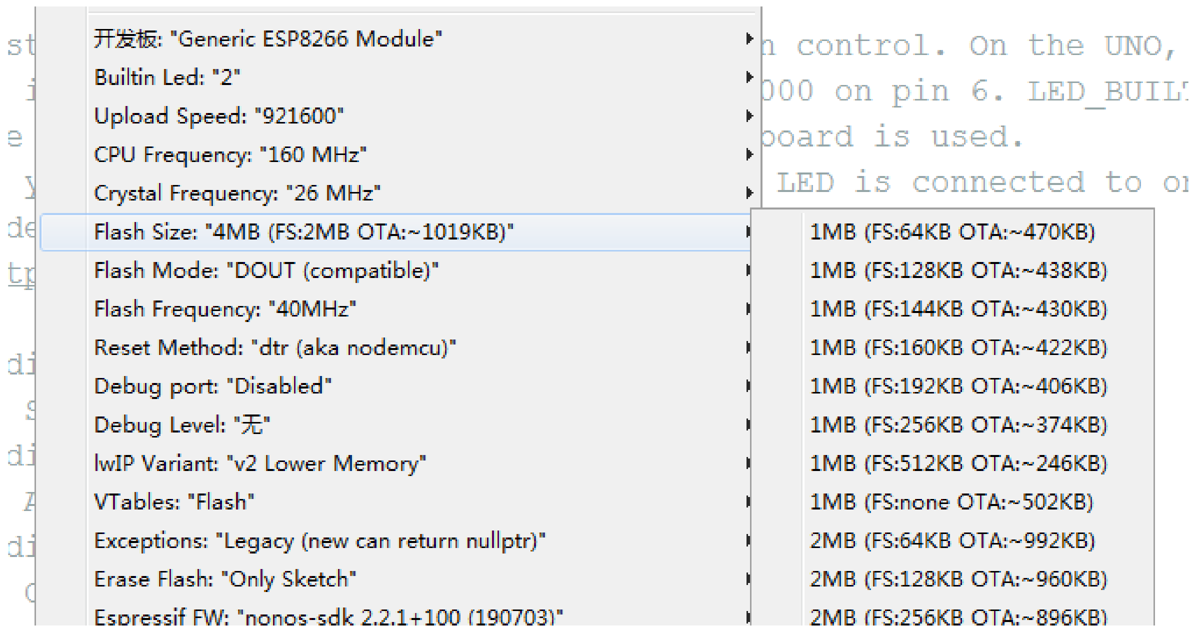



WiFi101 / WiFiNINA Firmware Updater
Port
Get Board Info
Boards Manager...
Arduino ARM (32-bits) Boards
Arduino AVR Boards
Arduino AVR Boards
Arduino Mbed OS Boards (nRF52840 / STM32H747)
Arduino megaAVR Boards
Arduino nRF52 (32-bits ARM) Boards
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
Atmel AVR Xplained minis
ESP32 Arduino
ESP8266 Boards (2.7.4)
Generic ESP8266 Module

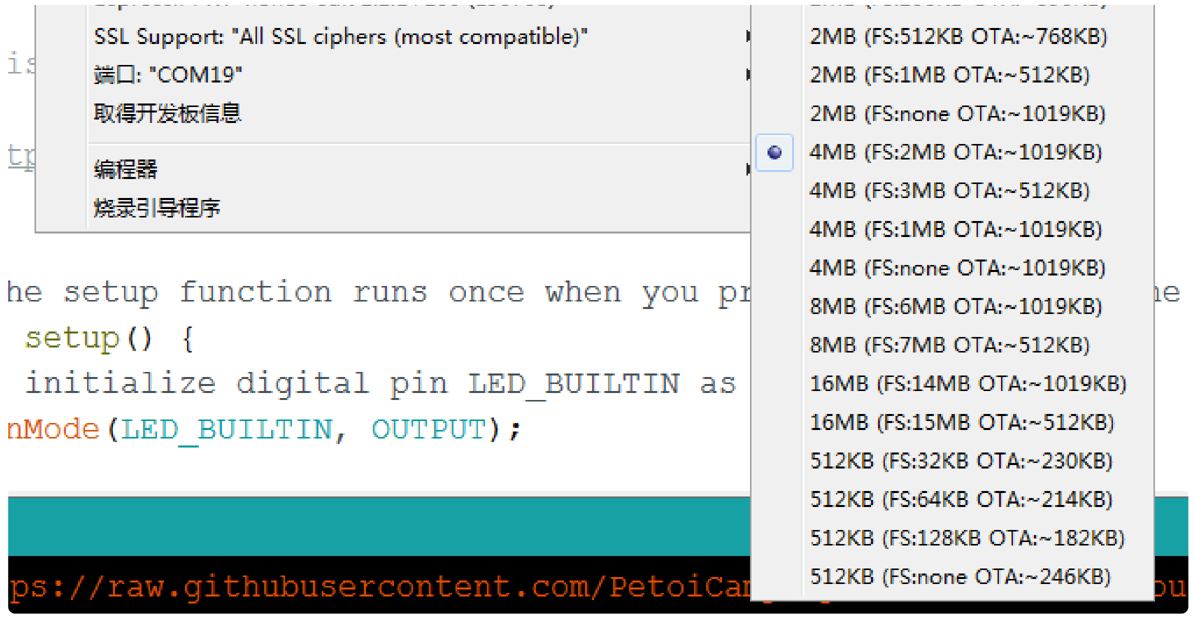
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "16MB (128Mb)"
Partition Scheme: "Biboard V0(4.5 MB APP with OTA /6.9 MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"

Then we set the parameters：

Parameters	Settings
Builtin Led	2
Upload Speed	921600（Auto-negotiation during downloading, 115200 is too slow）
CPU Frequency	160MHz
Flash Size	4MB
Reset Method	DTR reset
lwIP variant	V2 Lower memory
Erase Flash	Only sketch


开发板: "Generic ESP8266 Module"
Builtin Led: "2"
Upload Speed: "921600"
CPU Frequency: "160 MHz"
Crystal Frequency: "26 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Flash Mode: "DOUT (compatible)"
Flash Frequency: "40MHz"
Reset Method: "dtr (aka nodemcu)"
Debug port: "Disabled"
Debug Level: "无"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Legacy (new can return nullptr)"
Erase Flash: "Only Sketch"
Espressif FW: "nonos-sdk 2.2.1+100 (190703)"

1MB (FS:64KB OTA:~470KB)
1MB (FS:128KB OTA:~438KB)
1MB (FS:144KB OTA:~430KB)
1MB (FS:160KB OTA:~422KB)
1MB (FS:192KB OTA:~406KB)
1MB (FS:256KB OTA:~374KB)
1MB (FS:512KB OTA:~246KB)
1MB (FS:none OTA:~502KB)
2MB (FS:64KB OTA:~992KB)
2MB (FS:128KB OTA:~960KB)
2MB (FS:256KB OTA:~896KB)



```
is
tp
he setup function runs once when you pr                                    he
 setup() {
 initialize digital pin LED_BUILTIN as
nMode(LED_BUILTIN, OUTPUT);

ps://raw.githubusercontent.com/PetoiCar                                    ou
```

SSL Support: "All SSL ciphers (most compatible)"
端口: "COM19"
取得开发板信息
编程器
烧录引导程序

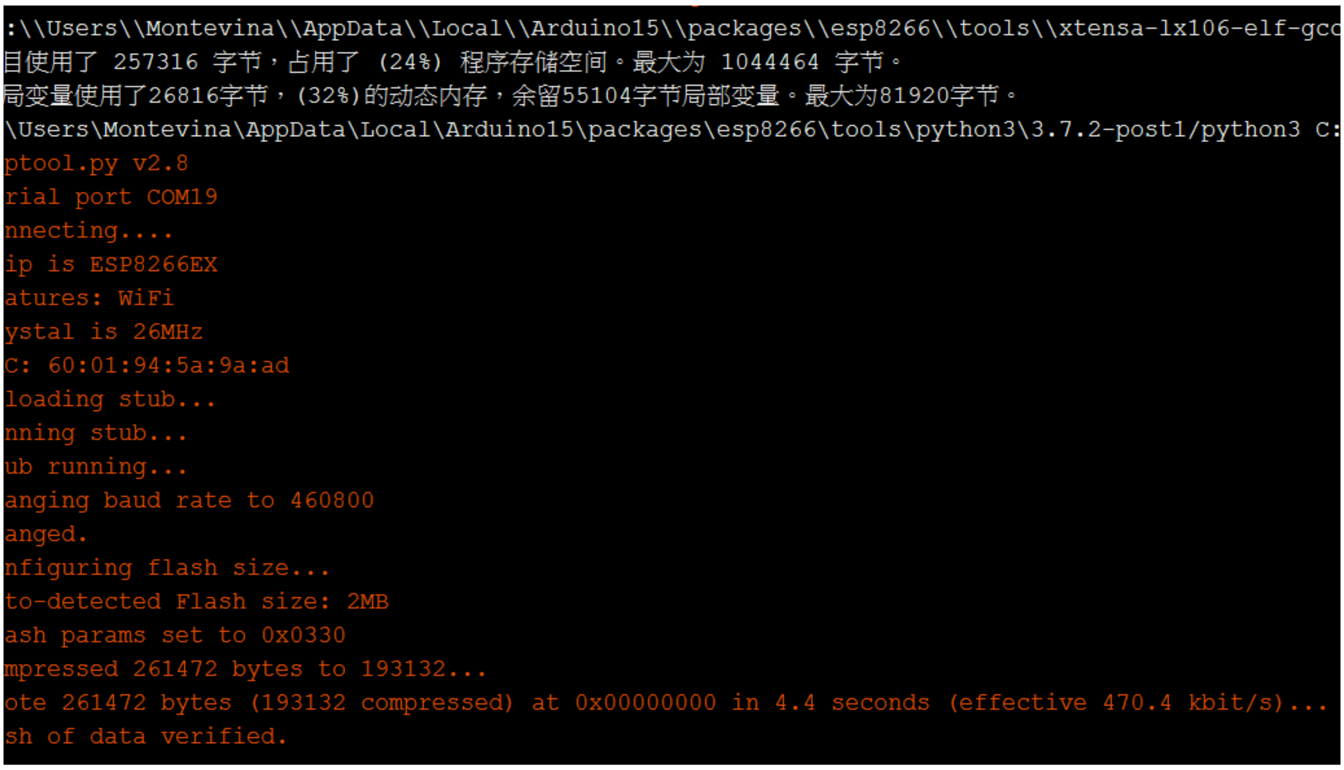
2MB (FS:512KB OTA:~768KB)
2MB (FS:1MB OTA:~512KB)
2MB (FS:none OTA:~1019KB)
● 4MB (FS:2MB OTA:~1019KB)
4MB (FS:3MB OTA:~512KB)
4MB (FS:1MB OTA:~1019KB)
4MB (FS:none OTA:~1019KB)
8MB (FS:6MB OTA:~1019KB)
8MB (FS:7MB OTA:~512KB)
16MB (FS:14MB OTA:~1019KB)
16MB (FS:15MB OTA:~512KB)
512KB (FS:32KB OTA:~230KB)
512KB (FS:64KB OTA:~214KB)
512KB (FS:128KB OTA:~182KB)
512KB (FS:none OTA:~246KB)

Download Test

After configuration, we use the Arduino classic "Blink" program to test the ESP8266 development board.

Open the Blink project, configure the development board, plug the module into the communication module debugging interface of the USB downloader, and download the Blink example.

Compared with the Arduino UNO, the compilation time is slightly longer, after Linking, the download progress will be displayed as a percentage:



```
:\\Users\\Montevina\\AppData\\Local\\Arduino15\\packages\\esp8266\\tools\\xtensa-lx106-elf-gcc
目使用了 257316 字节，占用了 (24%) 程序存储空间。最大为 1044464 字节。
局变量使用了26816字节，(32%)的动态内存，余留55104字节局部变量。最大为81920字节。
\Users\Montevina\AppData\Local\Arduino15\packages\esp8266\tools\python3\3.7.2-post1/python3 C:
ptool.py v2.8
rial port COM19
nnecting....
ip is ESP8266EX
atures: WiFi
ystal is 26MHz
C: 60:01:94:5a:9a:ad
loading stub...
nning stub...
ub running...
anging baud rate to 460800
anged.
nfiguring flash size...
to-detected Flash size: 2MB
ash params set to 0x0330
mpressed 261472 bytes to 193132...
ote 261472 bytes (193132 compressed) at 0x00000000 in 4.4 seconds (effective 470.4 kbit/s)...
sh of data verified.
```

The "Blink sketch" uses 257KB of flash and 26.8KB of the SRAM.

Download WiFi Firmware

Project URL：https://github.com/PetoiCamp/OpenCat/tree/main/ModuleTests/ESP8266WiFiController

There're 3 files in the project:

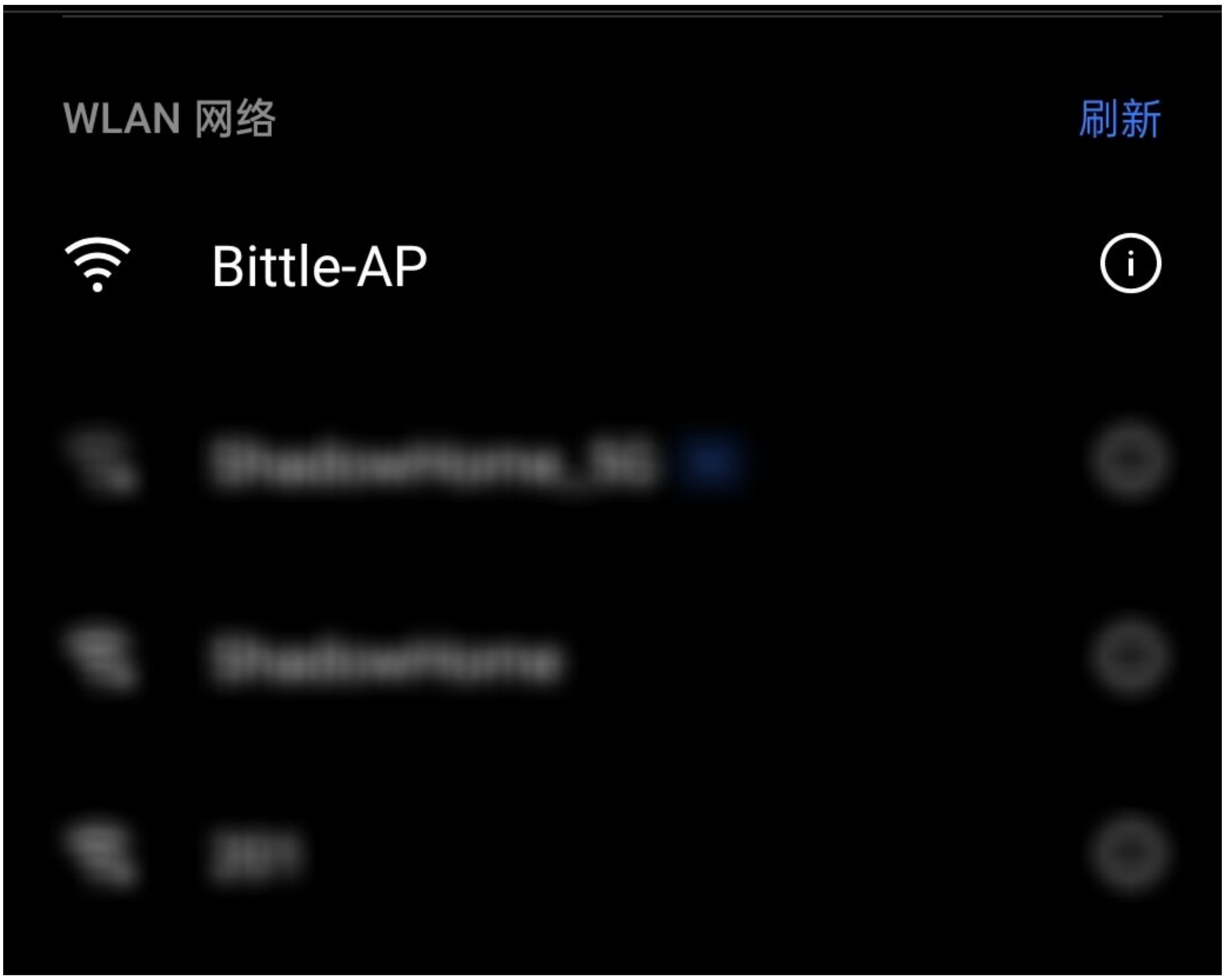
- ESP8266WiFiController.ino: Arduino sketch with server core code.
- mainpage.h：welcome page (html) in a char array.
- actionpage.h：action controller page (html) in a char array.

Please put them in the folder named "ESP8266WiFiController", then open the ino file and download it to the ESP8266 WiFi module.

How to Use

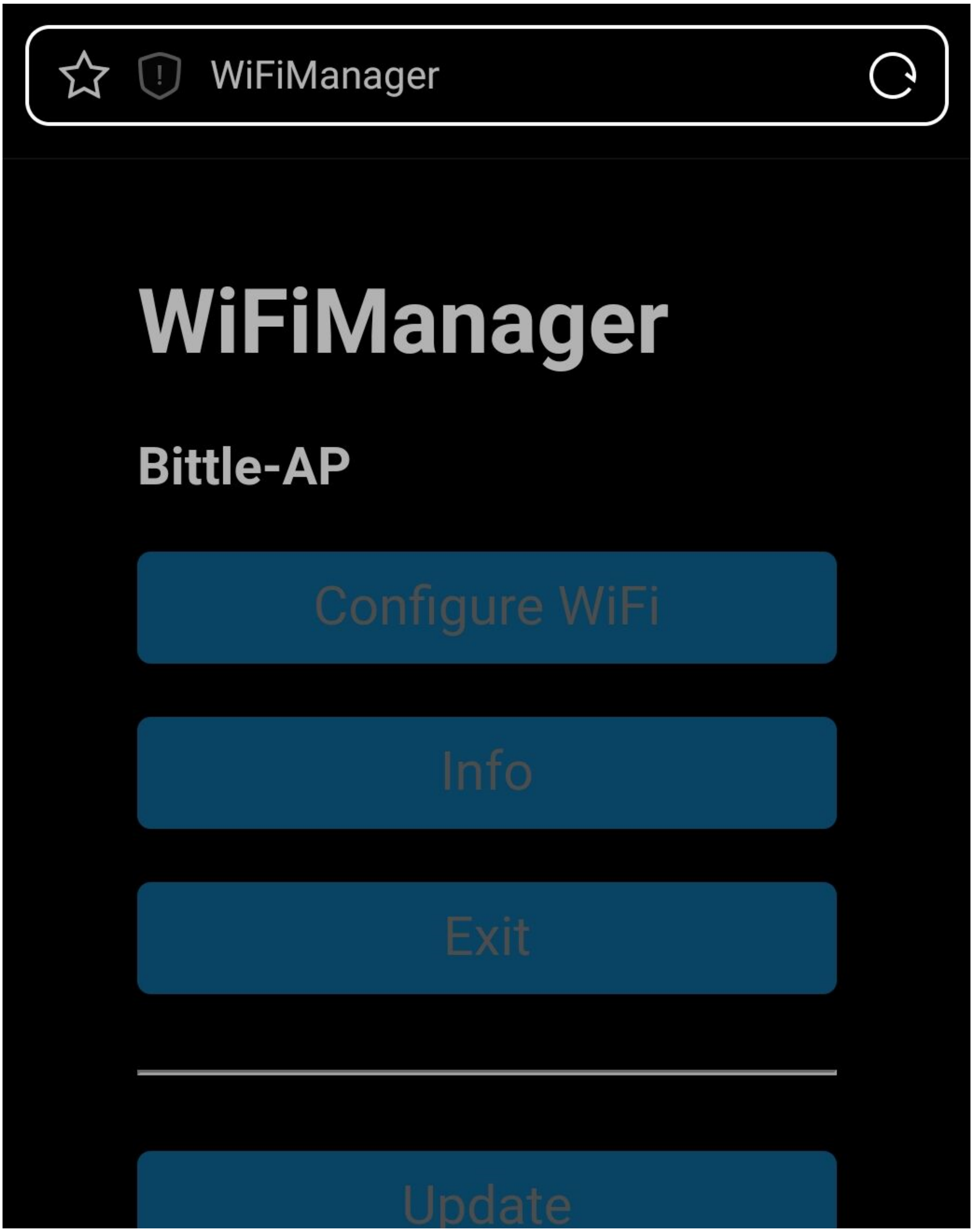
After the sketch downloader to the WiFi module, we strongly recommend run it with your USB downloader so you can get the serial output in the Arduino serial monitor.

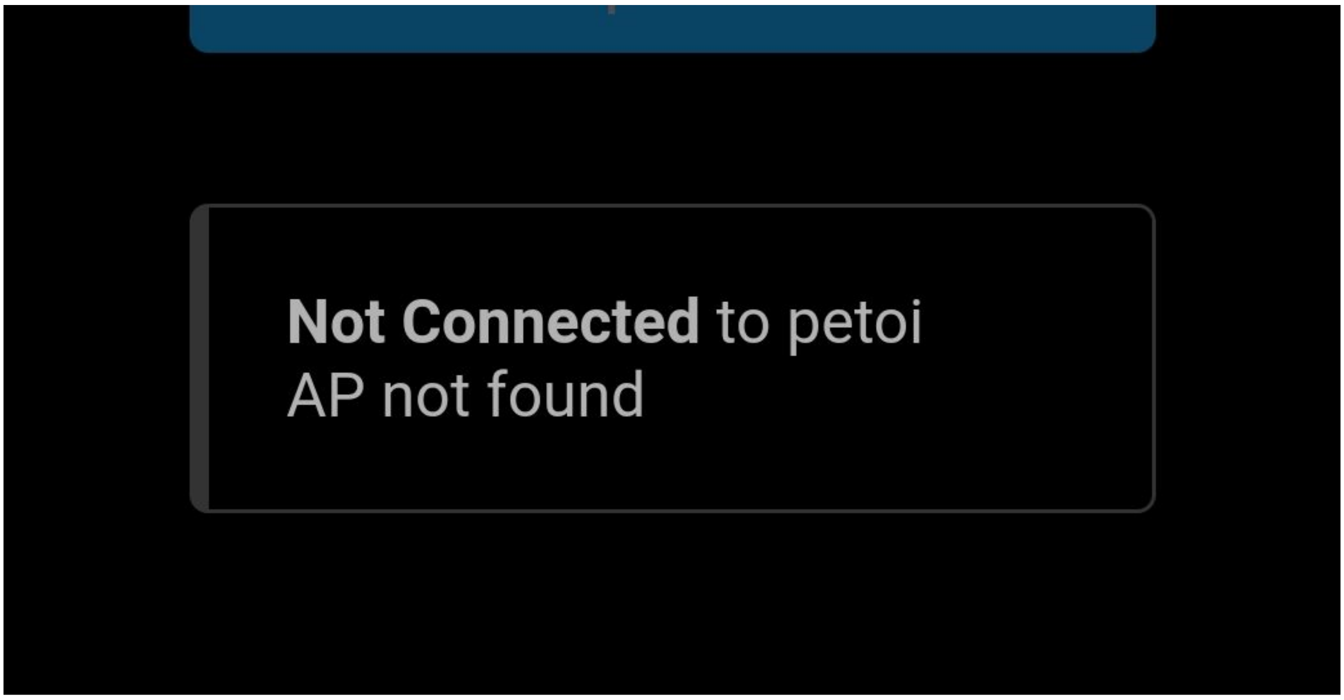
Open your smartphone WiFi scanner and find an access point named "Bittle-AP" that is not encrypted. Connect it.



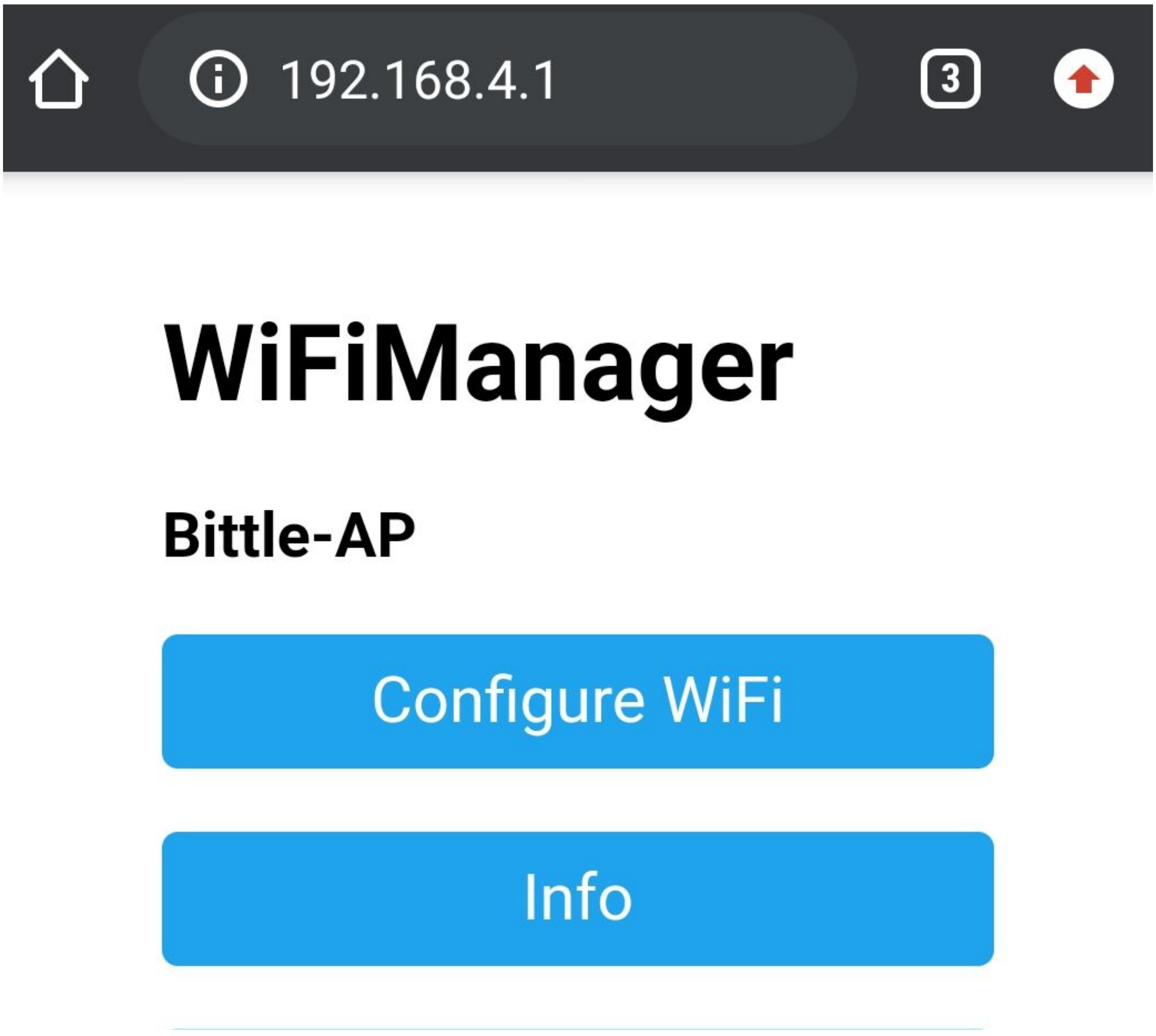
ⓘ If your smartphone automatically optimizes your network connection, it will cut the connection of "Bittle-AP" because there is no Internet connection. Your smartphone may connect to the WiFi with an Internet connection or even use your cellular data.

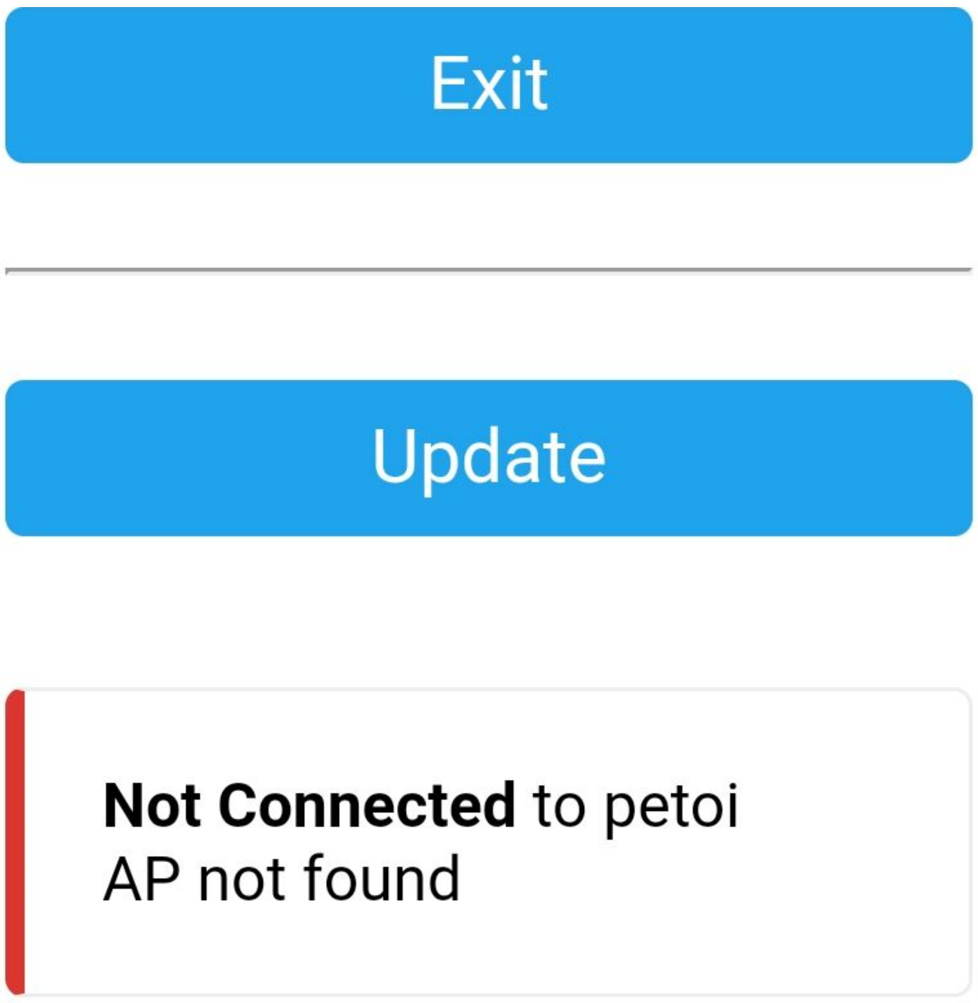
Your smartphone may auto jump to the "WiFiManager" page when connecting to "Bittle-AP".




Not Connected to petoi
AP not found

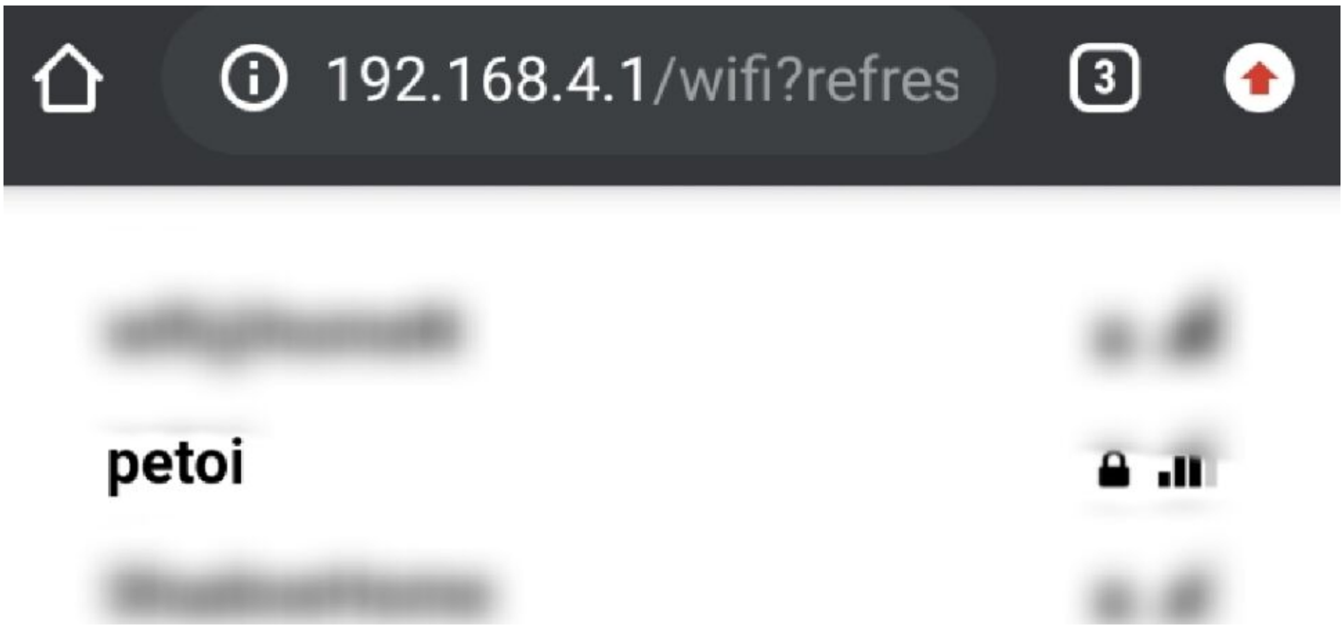
If not, please open your browser and enter **192.168.4.1** to enter the WiFi connection configuration page manually.



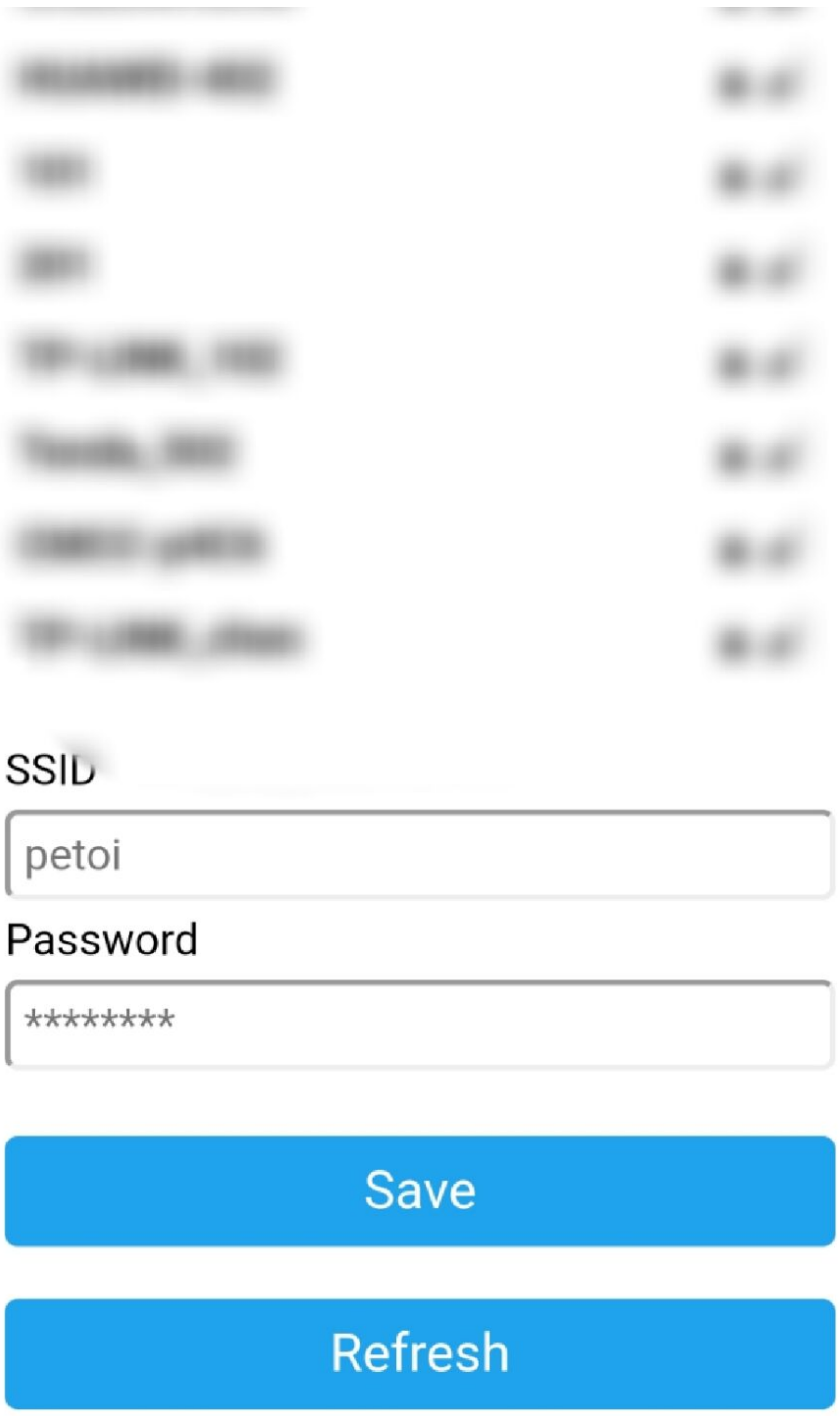


Not Connected to petoi
AP not found

On the WiFiManager page, Bittle's wireless module will automatically search for all nearby WiFi SSIDs and display them. After you click on your own WiFi SSID and enter the password, Bittle will connect to this network first.



192.168.4.1/wifi?refres

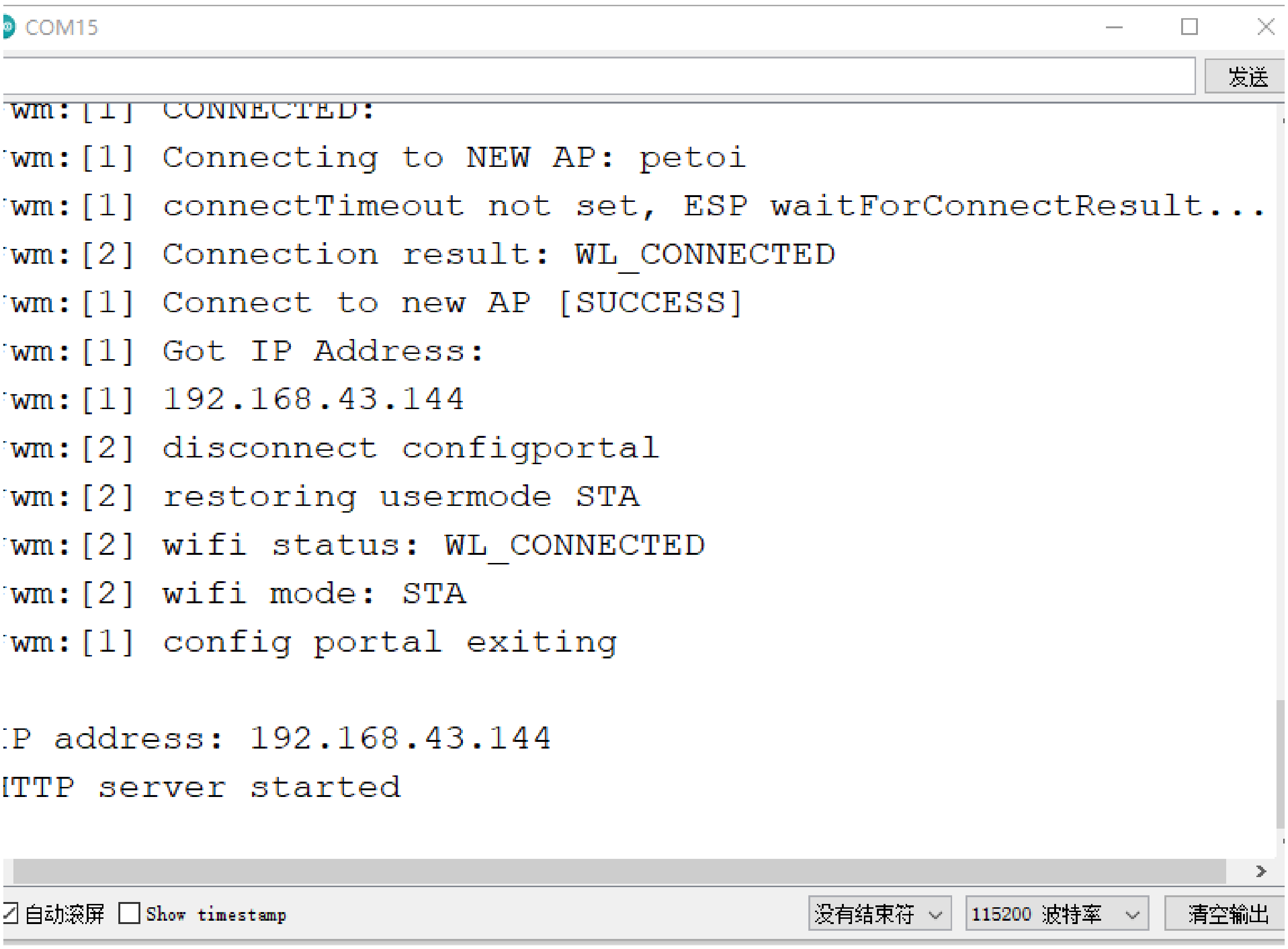

SSID

petoi

Password

****

Bittle will print out the IP address assigned by DHCP through the serial port after connecting to the WiFi successfully. You can also configure a fixed IP address in Arduino.



```
wm:[1] CONNECTED:
wm:[1] Connecting to NEW AP: petoi
wm:[1] connectTimeout not set, ESP waitForConnectResult...
wm:[2] Connection result: WL_CONNECTED
wm:[1] Connect to new AP [SUCCESS]
wm:[1] Got IP Address:
wm:[1] 192.168.43.144
wm:[2] disconnect configportal
wm:[2] restoring usermode STA
wm:[2] wifi status: WL_CONNECTED
wm:[2] wifi mode: STA
wm:[1] config portal exiting

P address: 192.168.43.144
TTP server started
```

Enter the IP address of the WiFi module, now you can control the Bittle through WiFi!

About the Sample Code

The sample code is a simple web server example, including 2 HTML pages. The two pages are stored in two header files in the form of string constants. The advantage is to avoid calling the client.print function constantly.

3.1 Set Up the WiFi Networks

Before we start our web server, we should configure the WiFi to connect to your local area network(LAN). We used to enter the WiFi SSID and password in the program, but it is very inconvenient while we need to change the network environment.

We use the WiFi manager library to configure the WiFi information through web.

```
// WiFiManager
WiFiManager wifiManager;
```

```
  // Start WiFi manager, default gateway IP is 192.168.4.1
  wifiManager.autoConnect("Bittle-AP");
```

3.2 Web server

Create a new web server and configure port 80 (commonly used HTTP server port)

```
ESP8266WebServer server(80);
```

3.3 Configure 3 HTTP service handler

The HTTP response function is to handle the incoming HTTP requests.

```
void handleMainPage() {
 //Serial.println("GET /");
 server.send(200, "text/html", mainpage);
}
void handleActionPage() {
 //Serial.println("GET /actionpage");
 server.send(200, "text/html", actionpage);
}
```

The handleMainPage and handleActionPage response 200 (OK) and corresponding web HTML code for your web browser (client).

```
void handleAction(){
  String argname = server.arg("name");

  if(argname == "gyro"){              // gyro switch
    Serial.print("g");
  }
…
```

The HandleAction function is slightly different. This is an HTTP request processing function with parameter passing. When the parameter is "gyro", the serial port of the WiFi module sends out the command ("g", switch IMU), so that our Bittle will execute the command.

So how is this "gyro" parameter generated and passed? Because we sent such an HTTP request with a value to the server:

```
http : //IP address or DomainName/action?name=gyro
```

The server parses the action parameter by the function and resolves that the name is "gyro".

We can directly enter this URL in the browser and execute it with the keyboard. The more common method is to add a link to the "Walk" button on the ActionPage web page. When the gyro button is pressed, the above URL will be sent to the host

The complete walk button configuration is as follows:

```
<button style="width: 25%" onclick="location.href='/action?name=gyro'">GyroOn/Off</button>
```

After parsing the "name" parameter, we send the actionpage again.

```
server.send(200, "text/html", actionpage);
```

We bond the handler method with the corresponding URLs.

```
server.on("/", handleMainPage);
server.on("/actionpage", handleActionPage);
server.on("/action", handleAction);
```

3.4 Start the Web Server

```
server.begin();
Serial.println("HTTP server started");
```

3.5 Handle Client Requests

```
void loop(void){
 server.handleClient();
}
```

More Ways Playing the WiFi Module

Compared to the ATMega328P on the NyBoard, there’re more hardware and software resources on the ESP8266, you can do more experiments with it.

Connect your Bittle to IoT platforms with HTTP restful APIs.

MQTT and node-red.

OTA with WiFi.

Make ESP8266 a strong co-processor for NyBoard for motion data fusion.

Useful Links