Mini- Projet IoT : EDGE

0. Sipeed Maixduino

Key Features：

■ CPU: RISC-V Dual Core 64bit, with FPU，400Mhz-500Mhz, Neural network processor

■ Connector: Compatible with Arduino 24P LCD connector, 24P Camera connector, TF card slot, Speaker connector, Compatible with Arduino interface

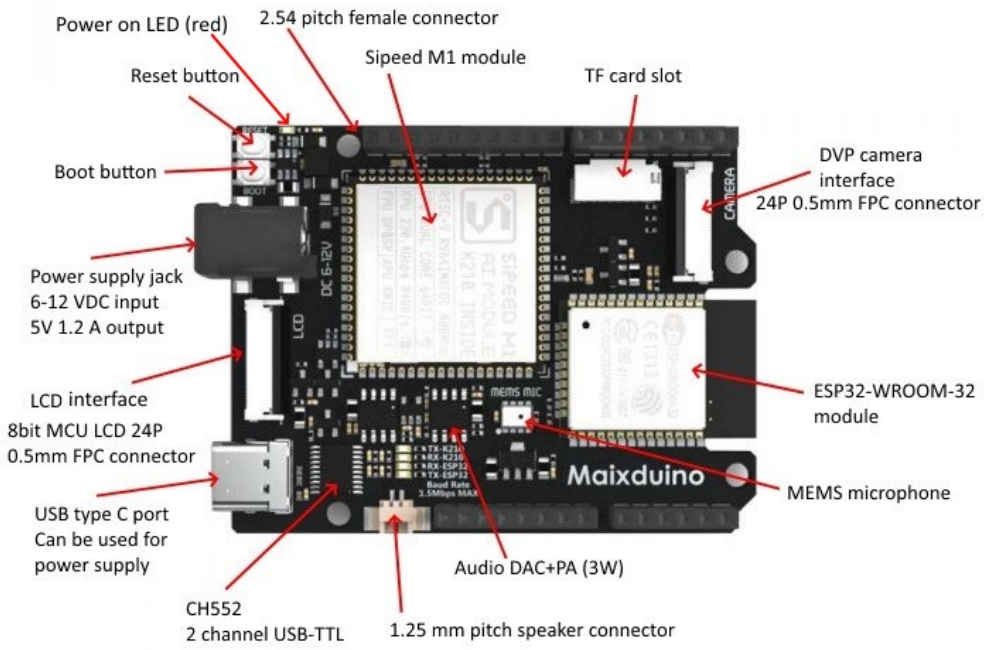
■ Development environment : PlatformIO, Arduino IDE

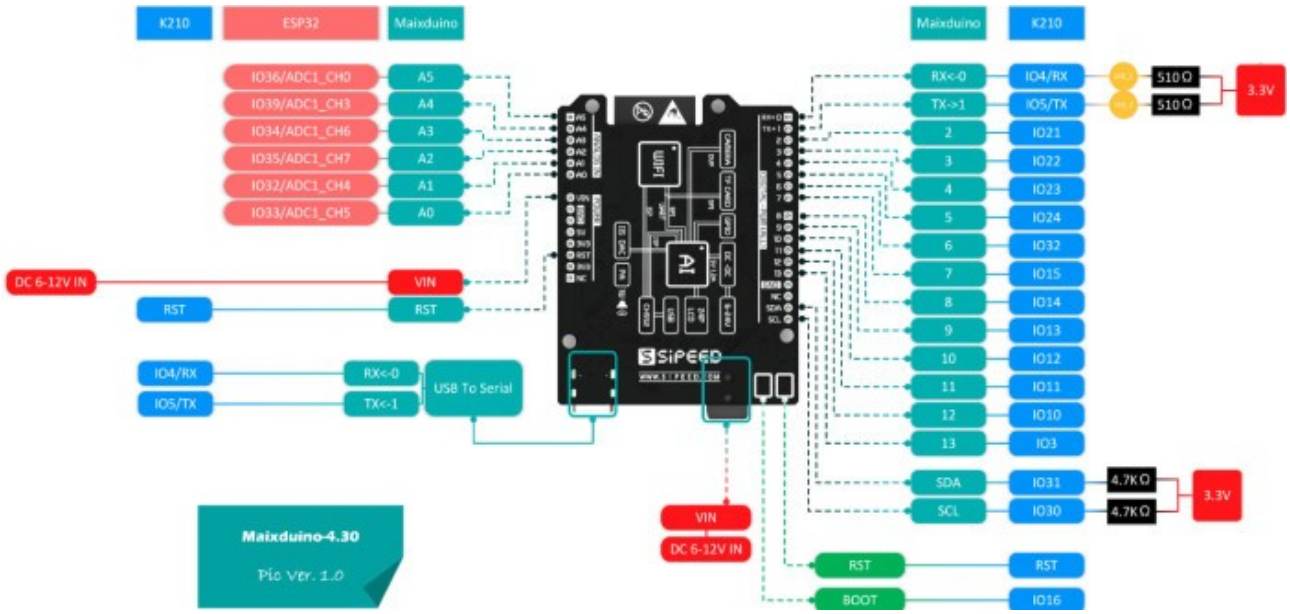
■ USB or DC connector(6-12V input;5V 1.2A output)

■ Download circuit: USB Type-C cable to complete the download

■ Wireless Function(Optional): Support 2.4G 802.11.b/g/n, Bluetooth 4.2

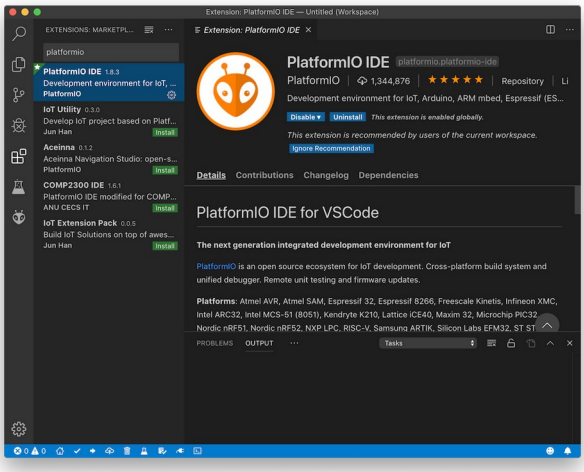
■ MEMS microphone and 3W speaker output

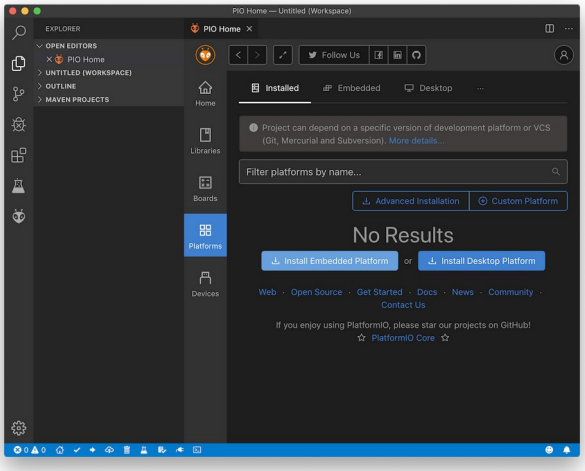




1. Set up PlatformIO with Visual Studio Code

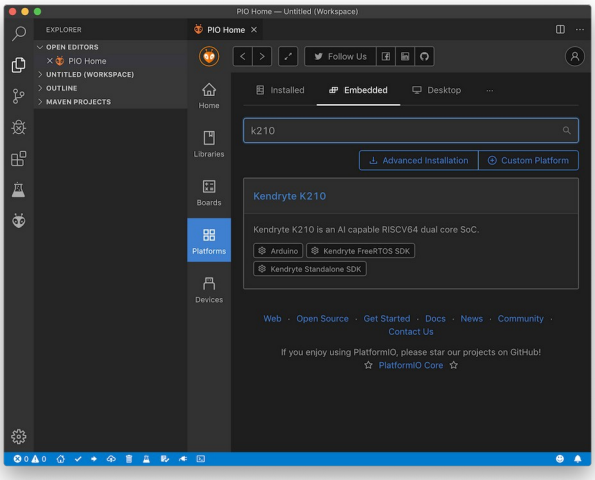
In Visual Studio Code, search for and install **Platform IO IDE** under the **Extension** tab.

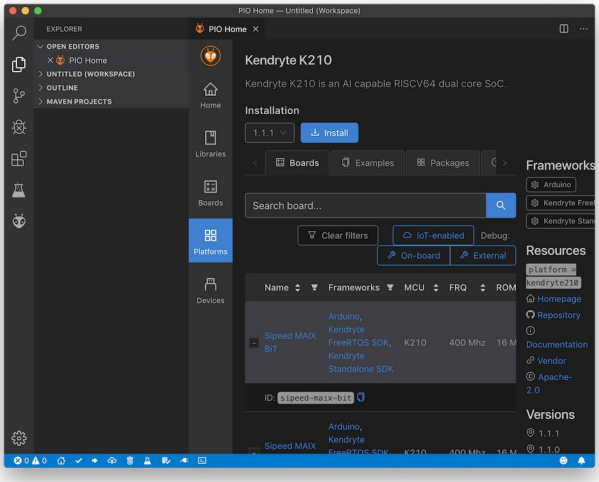




Click the **PlatformIO Home** button on the bottom left of the window. Then choose **Platforms** tab, and click **Install Embeded Platform.**

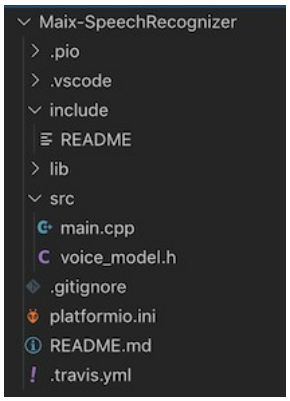
Search for "k210" and install the Kendryte K210 platform:





2. Run the voice recognition sample and make your own recordings

Navigate to PIO Home and open the **Maix-SpeechRecognizer** project. Look at the project structure on the left:

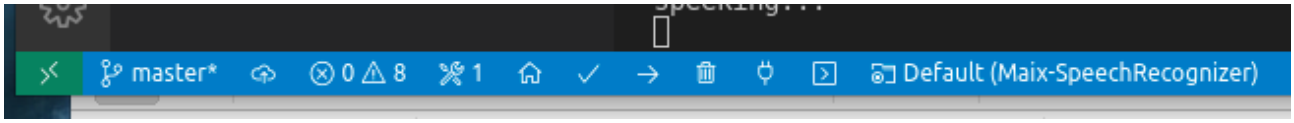


The source files are in src/ folder. Configure the project first by opening **platformio.ini**. Modify the configurations so that it looks like this:

```
[env:sipeed-maix-bit-mic]
platform = kendryte210
board = sipeed-maix-bit-mic
framework = arduino
upload_speed = 500000
monitor_speed = 115200
monitor_port = /dev/cu.usbserial-xel_sipeed0
upload_port = /dev/cu.usbserial-xel_sipeed0
```

If you are working with Linux (**`/dev/ttyUSB0`** or **`/dev/ttyUSB1 )`**
For board, we use the ID for the Maix Bit-Mic board. For upload_speed, we use 500000 (500 kbps). For monitor_port and upload_port, we use the same port that is selected in Arduino. You can find and copy the port name under PIO Home>Devices.

Now that Platform IO is configured, **compile** (**V**) and **upload** (→) the program to the Maix board. Click the Upload button on the bottom:



You should see output like this:

```
Use manually specified: /dev/cu.usbserial-xel_sipeed0
Uploading .pio/build/sipeed-maix-bit-mic/firmware.bin
[INFO] COM Port Selected Manually:  /dev/cu.usbserial-xel_sipeed0
[INFO] Default baudrate is 115200 , later it may be changed to the value you set.
[INFO] Trying to Enter the ISP Mode...
*
[INFO] Greeting Message Detected, Start Downloading ISP
Downloading ISP: |================================================================================| 100.0% 10kiB/s
[INFO] Booting From 0x80000000
[INFO] Wait For 0.1 second for ISP to Boot
[INFO] Boot to Flashmode Successfully
[INFO] Selected Baudrate:  500000
[INFO] Baudrate changed, greeting with ISP again ...
[INFO] Boot to Flashmode Successfully
[INFO] Selected Flash:  On-Board
[INFO] Initialization flash Successfully
Programming BIN: |================================================================================| 100.0% 31kiB/s
[INFO] Rebooting...
========================================= [SUCCESS] Took 20.88 seconds =========================================
```

3. Analyze and Modify the sample to recognize your own commands

0. Look at **voice_model.h**. These are sample MFCC frames for each command. Each sample is represented by its frame count (e.g. **fram_num_hey_friday_0**) and an array of its MFCC features (e.g. **hey_friday_0**). Notice there are 2628 elements in each array--the same length as each recording output from the serial port in 2.
1. Make your own recordings of two other commands (different from the two provided by the sample) from the serial port. Replace the sample **fram_num** and MFCC features in **voice_model.h** with yours.
2. In **main.cpp**, look for the strings that are displayed after each recognition. Modify them as necessary to reflect your commands in step 2.
3. Change **RECORD_MODE** back to 0, then re-run the program. Test your changes. **Demonstrate your changes to a TA.**

The beginning of the **voice_model.h** file :

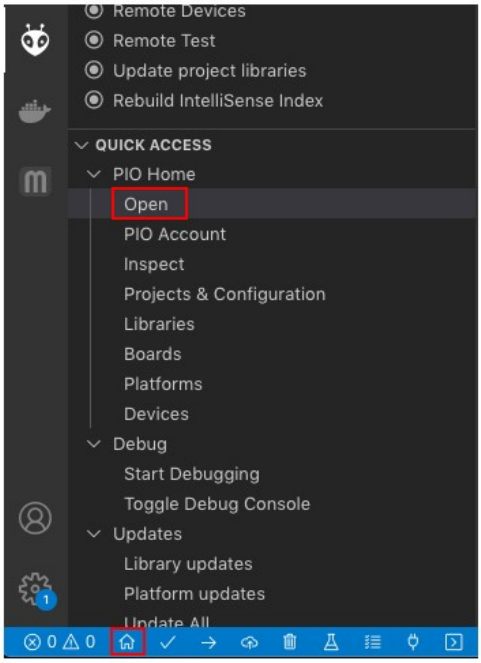
```
#ifndef __VOICE_MODEL_H
#define __VOICE_MODEL_H

#include <stdint.h>
#include "util/MFCC.h"

#ifdef __cplusplus
extern "C" {
#endif

const uint16_t fram_num_hey_friday_0 = 47;

const int16_t hey_friday_0[vv_frm_max*mfcc_num] = {
-96, 24, 12, 24, 18, 24, 21, 24, 23, 24, 20, 21, -562,
..
```

4. Import Arduino project to PlatformIO

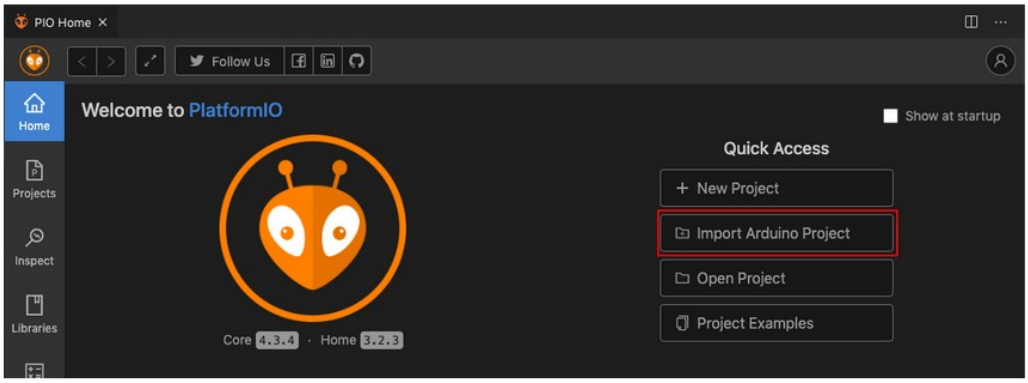
PlatformIO seems less accessible than the Arduino IDE when you start, but it is an infinitely more powerful code editor. We will see how to migrate (import) an existing project (ESP32,MaixDuino, ..)

4.1 Manually import an Arduino project on PlatformIO

Open the PIO home page from the toolbar on the left of the screen or from the house icon from the toolbar at the bottom of the screen.

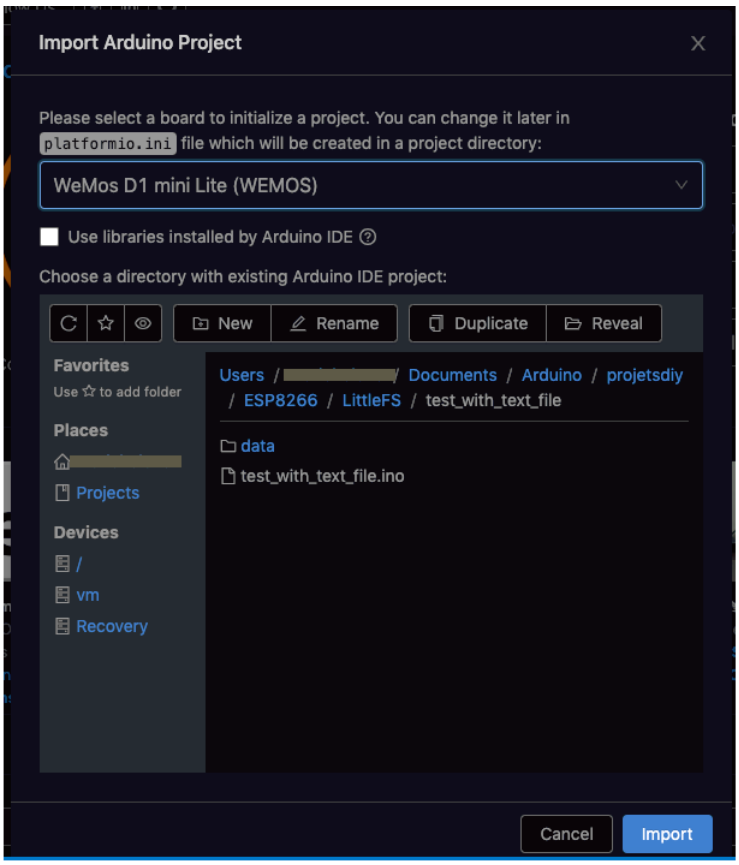


Click Import Arduino Projects to start importing an existing project.



Select the desired development board , for example **sipeed MaixDuino**

Launch the import / migration of the project to PIO



The folder created in the Projects directory takes as name the target date-time (name of the development board) and contains the following elements:

- **`lib`** private libraries are moved to the lib folder
- **`src`** will contain all the ino files (source code of the Arduino project)
- **`platformio.ini`** is the configuration file that allows you to define the environments. Each environment allows you to specify the type of development board (Arduino, ESP32, ...), the framework to use for compilation and many other parameters documented here.

```
; PlatformIO Project Configuration File
;
; Build options: build flags, source filter
; Upload options: custom upload port, speed and extra flags
; Library options: dependencies, extra library storages
; Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino
framework = arduino
lib_extra_dirs = ~/Documents/Arduino/libraries
monitor_port = /dev/ttyUSB0
upload_port = /dev/ttyUSB0
```

4.2 New structure of the Arduino project under PIO

With the Arduino IDE, libraries are stored in Documents in an Arduino -> Library subfolder.
Under PIO, it is advisable to manage the libraries directly in the project folder. This allows you to manage the versions of the libraries for each project. It's also possible to do this with the Arduino IDE, but it's less convenient.

When migrating an existing project, the main.ino file is automatically moved to the src folder.

When creating a new project, **PIO generates a cpp file** and adds a call to the **`Arduino.h`** library

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

You will have to manually move the data folder (if it exists) to the same level as the src folder

4.3 Simple example – maix.Blink.ino

```
// the setup function runs once when you press reset or power the board

#define LED_BUILTIN  12

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
// turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);
// turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```