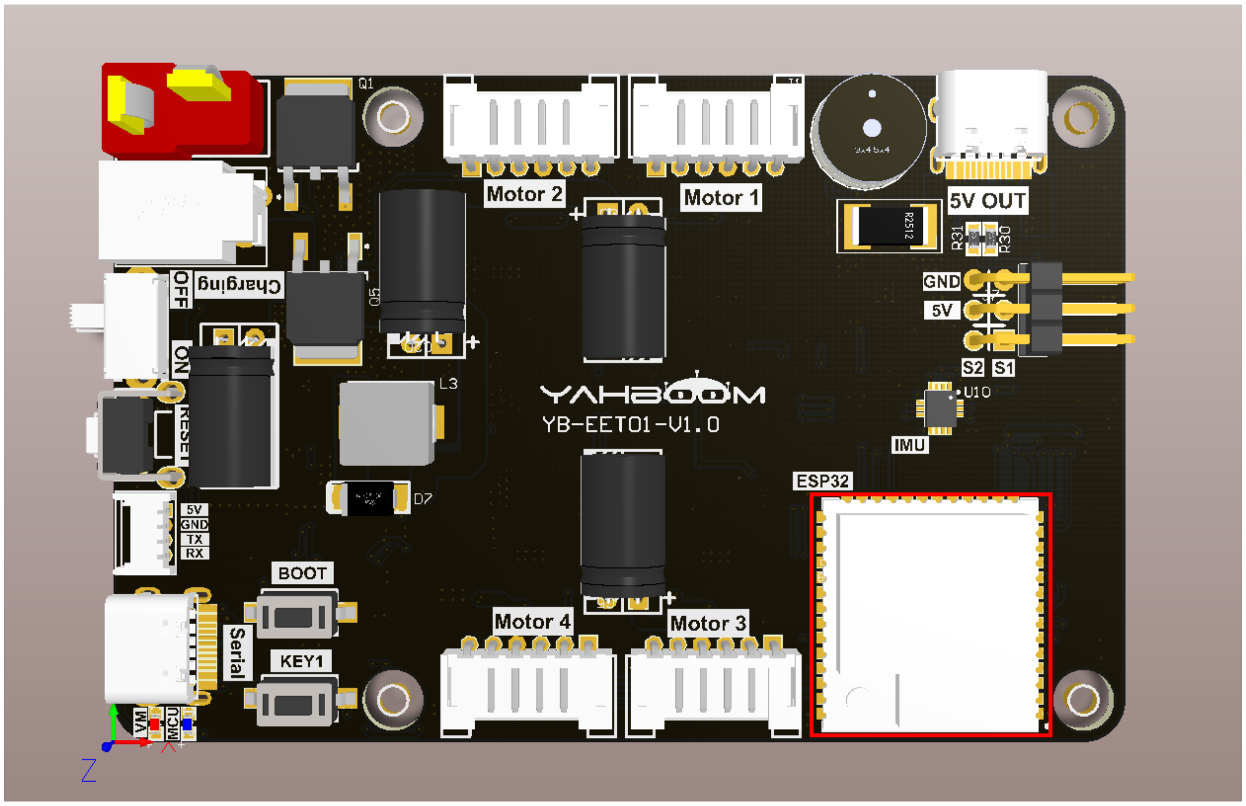
Partition table and memory

1. Experimental purpose

Use the ESP32S3 core module of the microROS control board to learn the function of ESP32 custom partition table.

2. Hardware connection

As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM-1U-N4R2 core module. It not only has internal space, but also has an additional 4MB FLASH program space and 2MB PSRAM memory space. You only need to connect the type-C data cable Connect the computer to the microROS control board as a firmware burning function.



3. Core code analysis

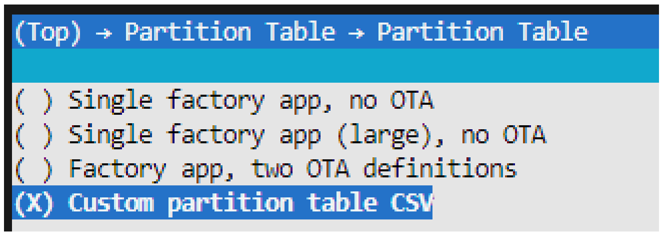
The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/esp32_samples/partition_table
```

Create a new partitions.csv file in the project root directory and add the following content.

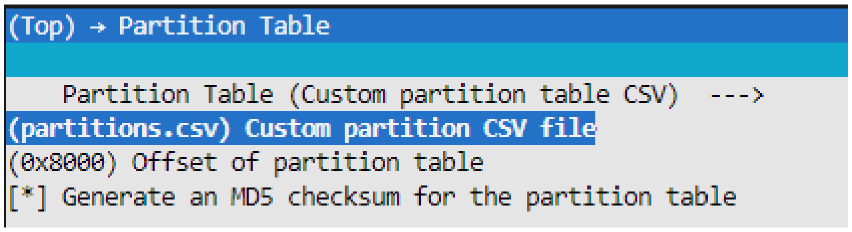
```
# Name,   Type, SubType, Offset,  Size,   Flags
# Note: if you have increased the bootloader size, make sure to update the offsets
to avoid overlap
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 3M,
```

Then open the IDF configuration tool and specify the partition table as partitions.csv.



```
(Top) → Partition Table → Partition Table

( ) Single factory app, no OTA
( ) Single factory app (large), no OTA
( ) Factory app, two OTA definitions
(X) Custom partition table CSV
```



```
(Top) → Partition Table

    Partition Table (Custom partition table CSV)  --->
(partitions.csv) Custom partition CSV file
(0x8000) Offset of partition table
[*] Generate an MD5 checksum for the partition table
```

4. Compile, download and flash firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

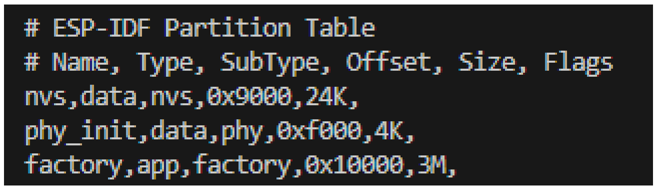
Enter the project directory

```
cd ~/esp/Samples/esp32_samples/partition_table
```

Compile project

```
idf.py build
```

You can see that the partition table printed during compilation is consistent with the partitions.csv file.



```
# ESP-IDF Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,data,nvs,0x9000,24K,
phy_init,data,phy,0xf000,4K,
factory,app,factory,0x10000,3M,
```

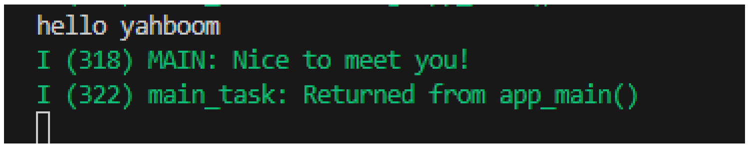
Flash and open the serial port simulator

```
idf.py flash monitor
```

If you need to exit the serial port simulator, press **Ctrl+]**.

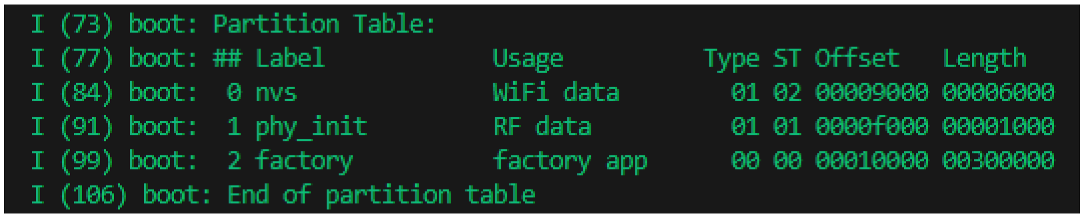
5. Experimental results

The serial port simulator prints the "hello yahboom" greeting.



```
hello yahboom
I (318) MAIN: Nice to meet you!
I (322) main_task: Returned from app_main()
```

Then slide the mouse wheel upward to view the printed system information. You can see the partition table content displayed in the boot: Partition Table column, which is consistent with the content of partitions.csv in the project root directory.



```
I (73) boot: Partition Table:
I (77) boot: ## Label            Usage          Type ST Offset   Length
I (84) boot:  0 nvs              WiFi data        01 02 00009000 00006000
I (91) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (99) boot:  2 factory          factory app      00 00 00010000 00300000
I (106) boot: End of partition table
```