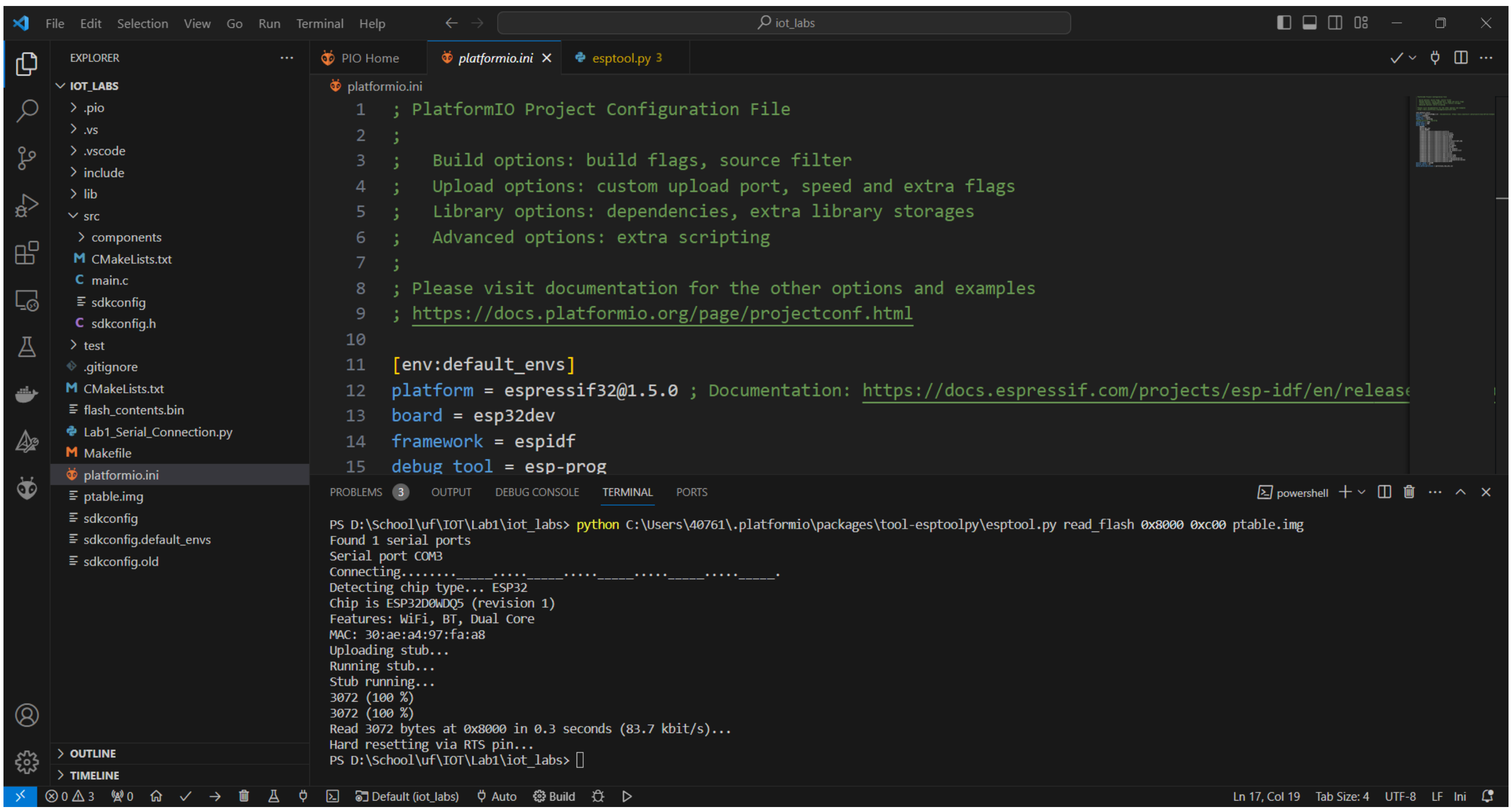
IoT Security and Privacy

Lab2 – ESP32 Flash Hack

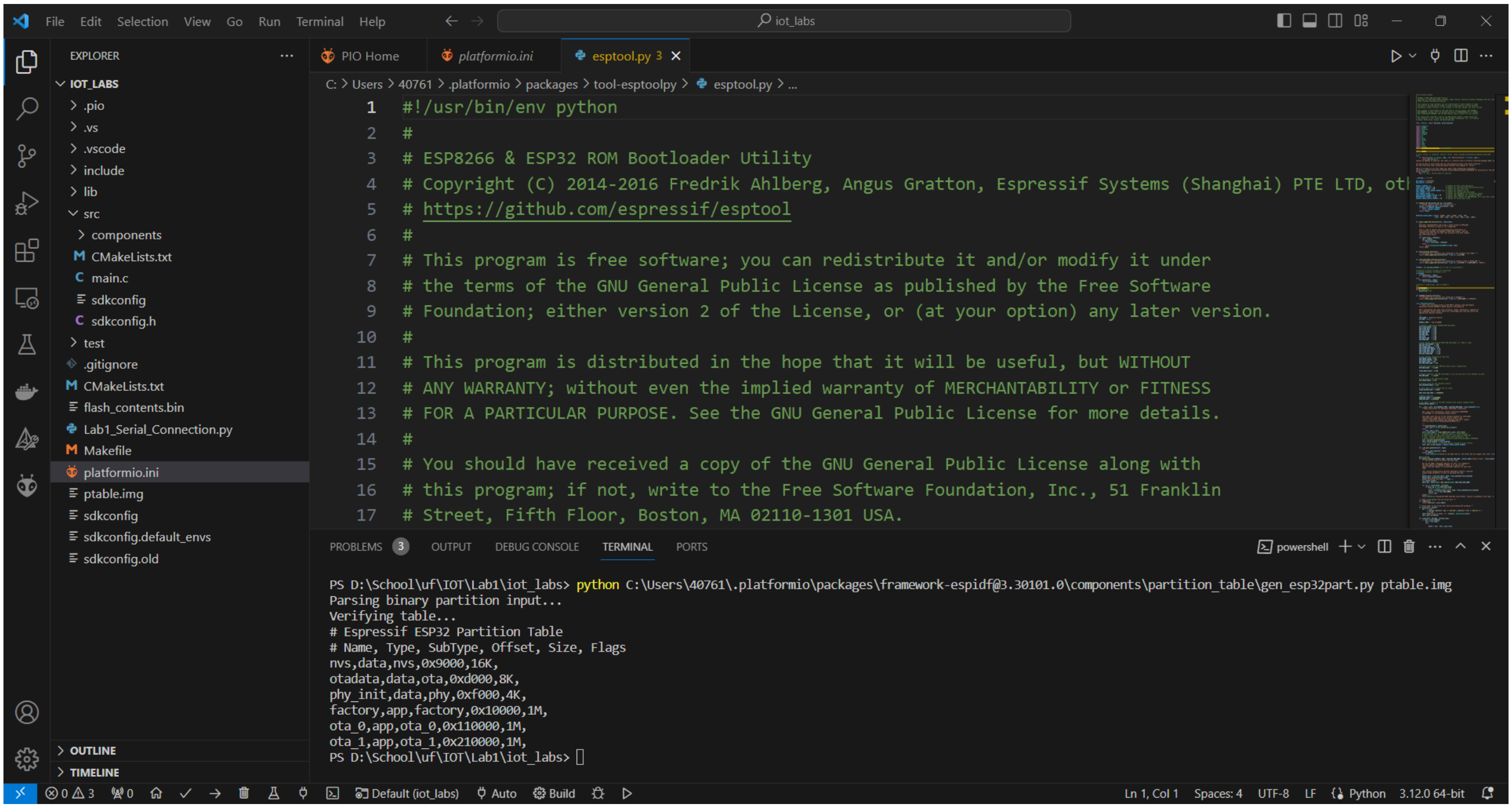
Questions

a. *python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0x8000 0xc00 ptable.img*
where 0x8000 is the start address of the partition table and 0xc00 is the length of the partition table. $USER is the username of the user that installed platformio. The binary partition table is saved in *ptable.img*. Please provide a screenshot of the command runninn. [10 point]



b. following command will print out the partition table of our IoT kit in the CSV (comma-separated values) format. The partition table shows how the flash is partitioned. *python C:\Users\40761\.platformio\packages\framework-espidf@3.30101.0\components\partition_table\gen_esp32part.py ptable.img*

Please provide a screenshot of the command running. [10 point]



c. Explain the partition table that is printed out. [10 point]

This partition table is for the flash memory layout of the ESP32 device. Each entry represents a partition in the flash memory.

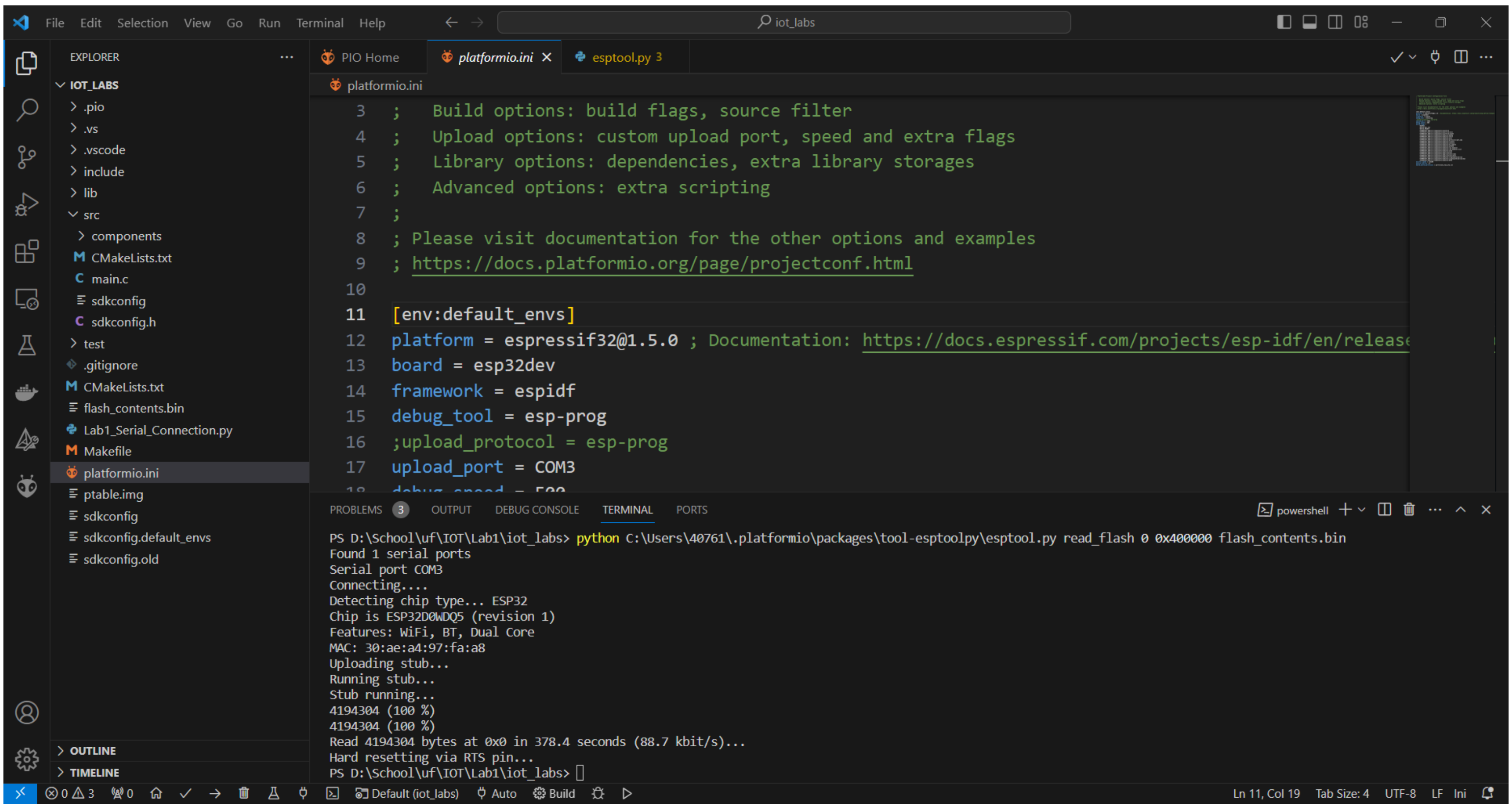
nvs: type data, subtype nvs. It starts at address 0x9000 and has a size of 16K. This partition is typically used to store Non-Volatile Storage data.

otadata: type data, subtype ota, which starts at address 0xd000 and has a size of 8K This partition is used to store OTA (Over-The-Air) update data.

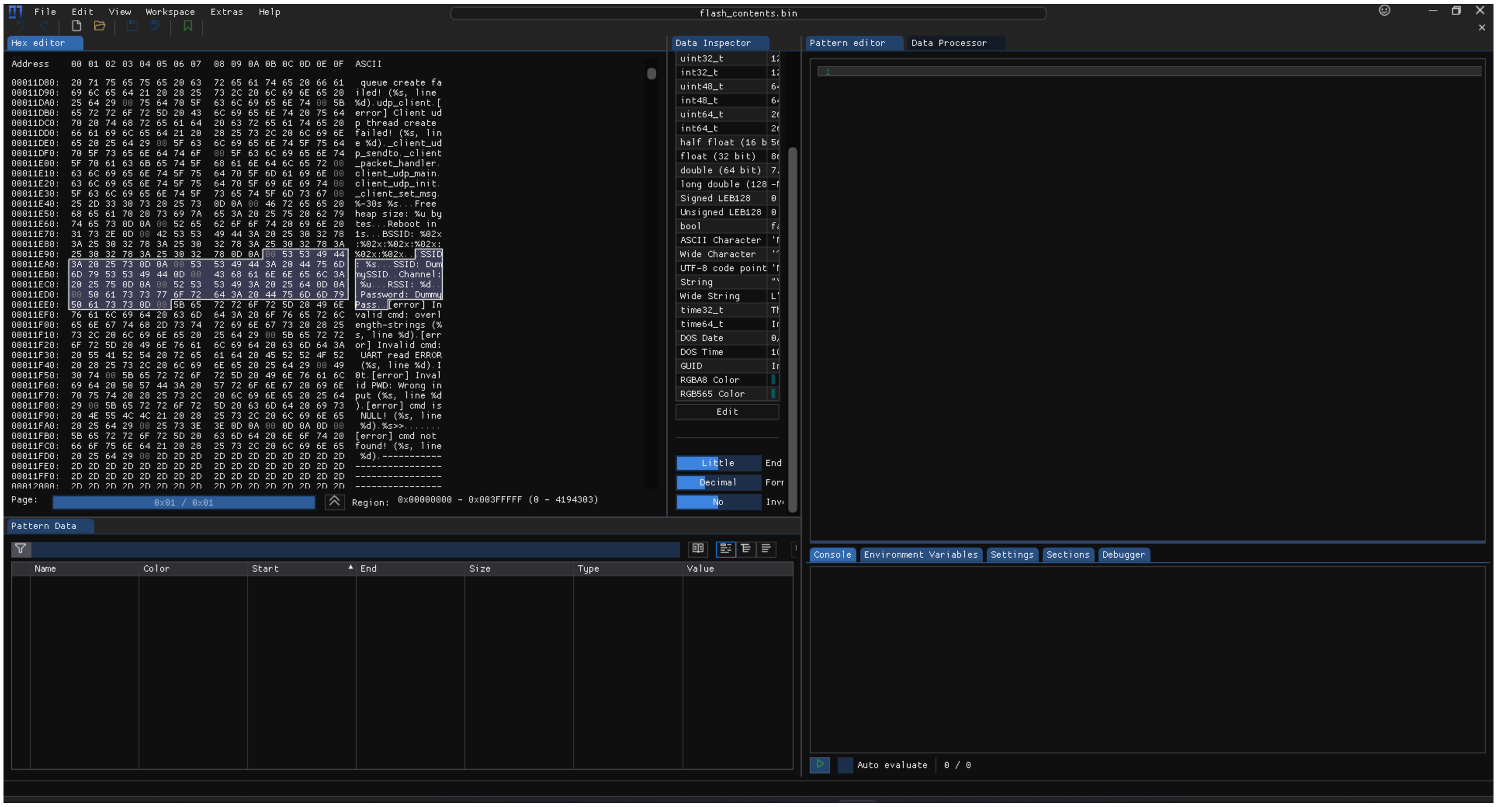
phy_init: type data, subtype phy, starts at address 0xf000 and is 4K in size. this partition is used to store ESP32 PHY initialization data.

factory: type app, subtype factory, it starts at address 0x10000 and is 1M in size. this is the factory firmware partition, which usually contains the original firmware of the device when it was shipped.
ota_0 and ota_1: These partitions are of type app and subtype ota_0 and ota_1, which start at address 0x110000 and 0x210000 respectively, and have a size of 1 M. These partitions are used to store OTA updated firmware. During the OTA update process, the new firmware is written to the unused OTA partition, and then the device switches to this new firmware at the next boot.

2. The following command retrieves the whole flash content although it is also possible to refer to the partition table and select only the occupied part of the flash.
   a. *python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py read_flash 0 0x400000 flash_contents.bin*
      where *0* is the starting address and *0x400000* is the length of the flash of the ESP32-WROOM-32 surface-mount module board that our IoT kit uses. The whole flash in the binary format is saved in *flash_contents.bin*. Please provide a screenshot of the command. [10 point]



   b. Students can use a hex editor (e.g. wxhexeditor, imhex) to search the WiFi credentials in the flash dump. Please provide a screenshot of found WiFi credentials (e.g. password or key) using a hex editor. [10 point]

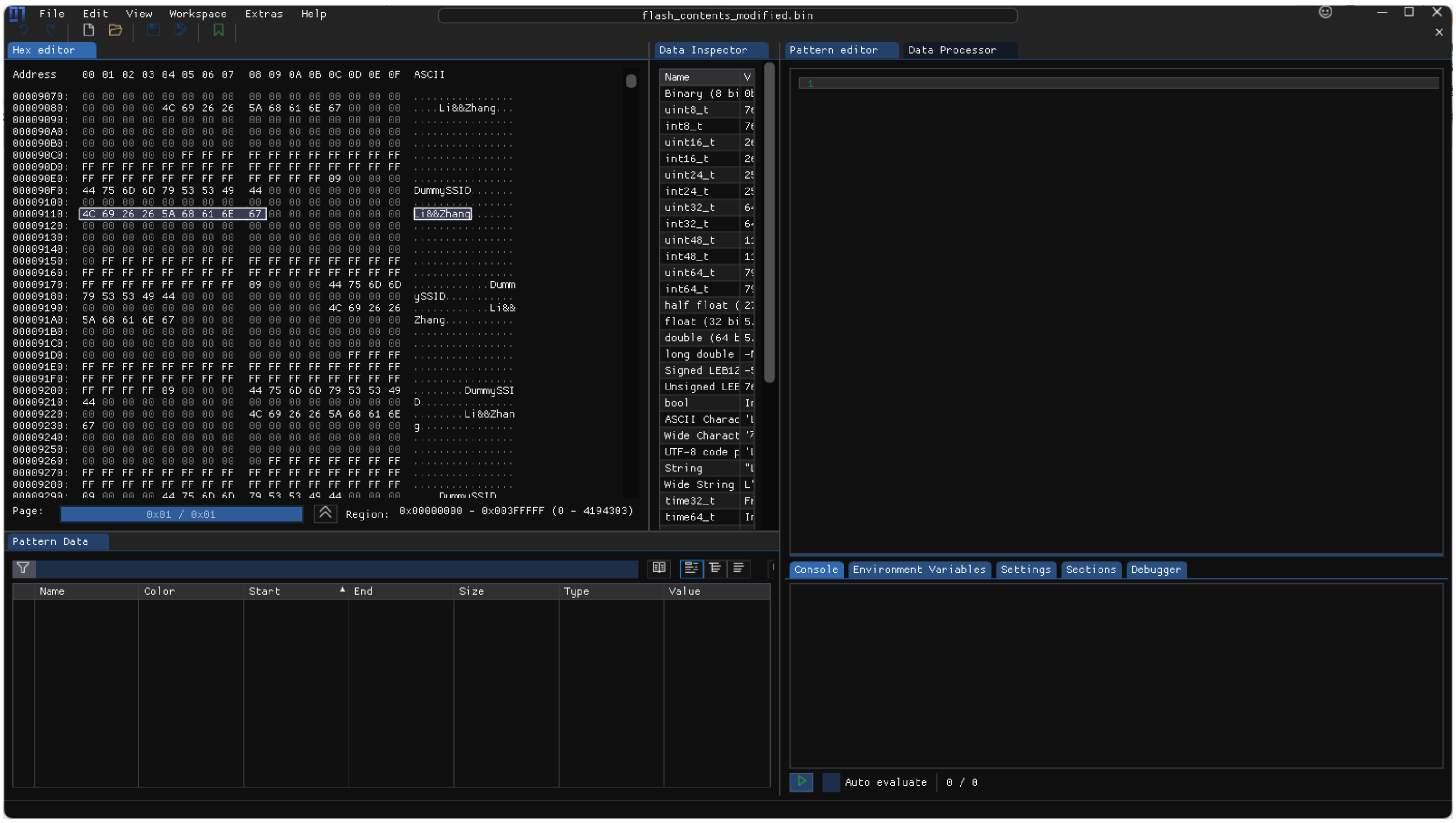


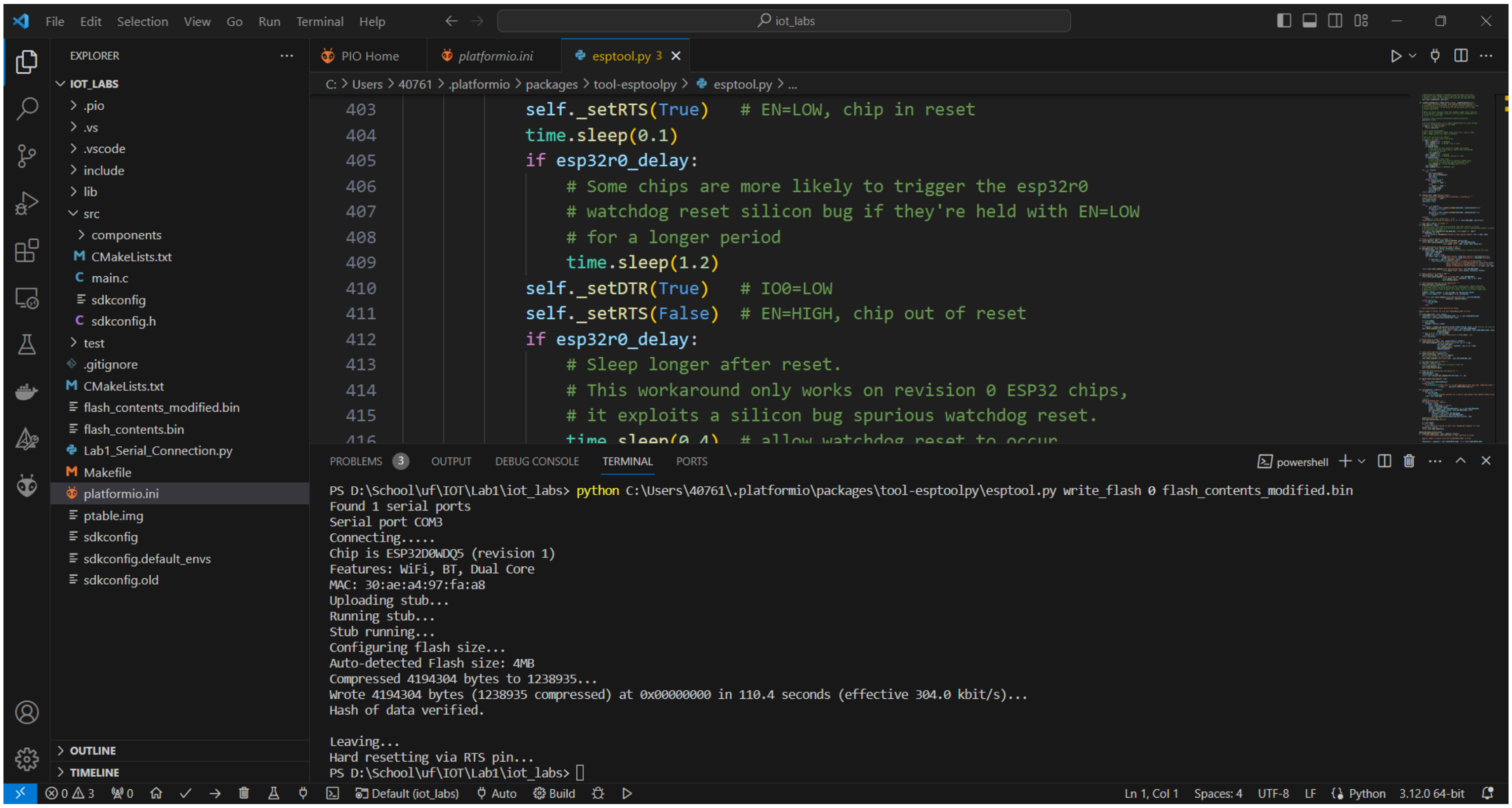
c. In an extreme case, an adversary may attempt to write another firmware to the IoT kit with esptool.py. Please demonstrate writing back to the IoT kit with esptool.py. Students should modify the downloaded flash to change the password to be some combination of all group members' names. Take care that the modified password is the same length as the original. Students must then try to write the changed flash dump to the IoT kit. Please provide a screenshot of the command. Describe the results and discuss potential reasons for what was observed. [20 points]

Hint: Here is a command writing the flash_contents_modified.bin to the IoT kit

python C:\Users\40761\.platformio\packages\tool-esptoolpy\esptool.py write_flash 0 flash_contents_modified.bin

change all the passwords:DummyPass into Li&&Zhang using ImHex





The system found a serial port (COM3) and successfully connected to the ESP32 chip. This indicates that the device is properly connected to the computer and that serial communication is working.

The system recognizes the chip as an ESP32D0WDQ5 (revision 1) with WiFi, Bluetooth, and Dual Core. This is a normal device recognition process.

The system uploads a small program (stub) to the ESP32 device to assist in the write process. This is part of the normal write process.

The system automatically detects a flash size of 4MB, which is one of the standard configurations for ESP32 devices.
The modified flash file is compressed and written to the device's flash memory. A total of 4,194,304 bytes were written in approximately 110.4 seconds. This indicates that the write operation completed successfully.

After the write was completed, the system performed data verification to ensure that the written data matched the original file." Hash of data verified" indicates that the data was verified successfully and no errors were found.

Finally, a hard reset was performed via the RTS pin, which is the normal procedure for rebooting the device. Then, we observe that the board will lose its original function and will no longer carry out its work and we won't be able to test with Lab1's scripts.