ECE 568: Embedded Systems

Spring 2022

Instructions for flashing MicroPython on your ESP32 board

1. Getting started with Python and MicroPython

The following sections describe various software installation procedures.

1.1. Install USB Driver on your own PC (Windows and Mac only)

Download and install the *CP210x USB to UART Bridge VCP Drivers* so that your machine identifies when you connect ESP32 board to your PC. You can download the files here: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers. Linux machines usually do not need this driver, however, if your Linux machine is unable to detect the board, you might want to install Linux version of this driver.

1.2. Flashing MicroPython on ESP32 (using Thonny)

The latest version of Thonny (3.3.13) comes preinstalled with the package 'esptool', which is a Python-based, open source, platform independent, utility to communicate with the ROM bootloader in ESP32. Esptool can be used to load MicroPython on your ESP32 board.

1.2.1. Acquiring MicroPython Firmware

Download the latest ESP32 firmware from http://micropython.org/download. As of today, the latest available firmware is 'esp32-20220117-v1.18.bin' (you can click this link directly to download the firmware).

1.2.2. Thonny Setup and Flashing

- Plug in your ESP32 board to your computer using a microUSB cable, launch Thonny and navigate to "Run > Select Interpreter".
- Under the "Which interpreter or device should Thonny use for running your code?" label, select "MicroPython (ESP32)".
- Select the COM port to which your ESP32 is connected under "Port" (see Figure 1).

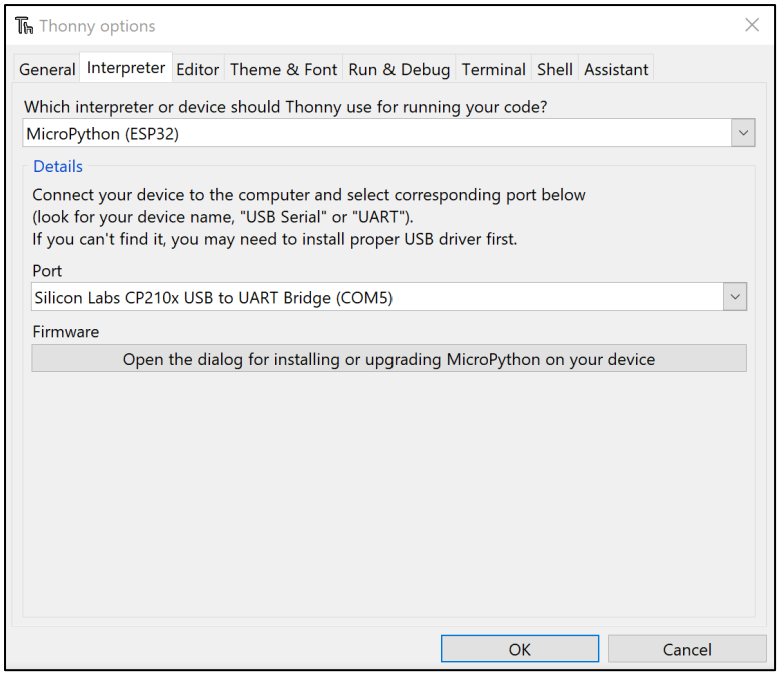

Figure 1. Set the interpreter and port

- After setting the interpreter and the port, click on the button: "Open the dialog for installing or upgrading MicroPython on your device". Select your device's port again and browse for the MicroPython firmware that you downloaded earlier for the "Firmware" field. If you are flashing MicroPython on your board for the first time, then make sure that "Erase flash before installing" is checked, then click "Install", as shown in Figure 2.

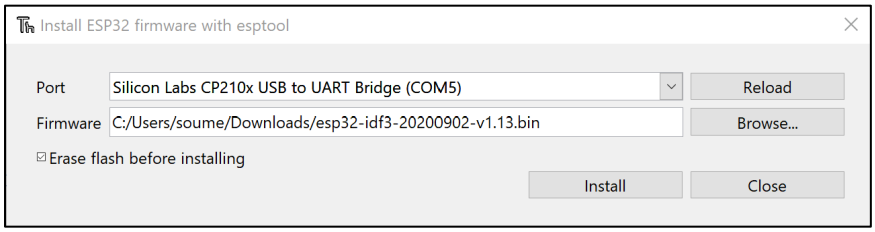

Figure 2. Select the port and downloaded MicroPython firmware

Thonny will now erase any existing firmware, and then flash the new MicroPython firmware on your ESP32, as you can see in Figures 3 and 4. This may take a few minutes. After the firmware has finished installing, you may close all the open dialogs.

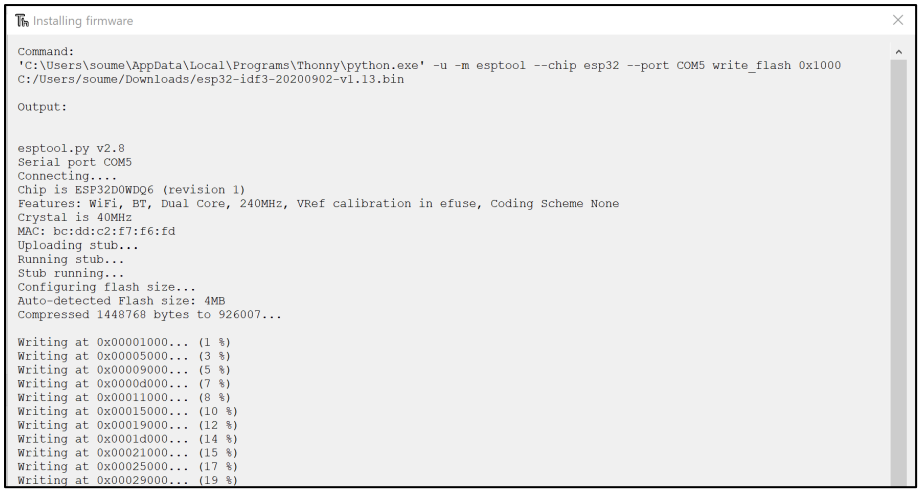

Figure 3. Flashing progress

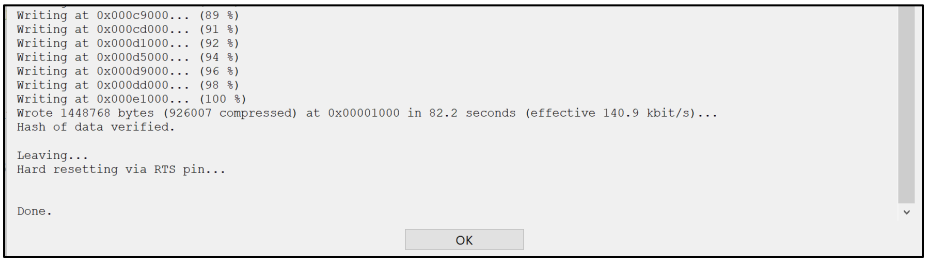

Figure 4. Thonny has completed flashing the firmware on the microcontroller.

- At the bottom of the Thonny window in the tab labeled “Shell”, you should see text that resembles Figure 5 (the version may be different).

Figure 5. MicroPython REPL ready for input

Your ESP32 board is now ready for programming.

NOTE: You should be able to flash MicroPython easily using the steps mentioned above. If you are unsuccessful for any reason, you can follow the alternative method outlined in the next section.

1.3. Alternative Method to flash MicroPython

1.3.1. Installing esptool on your PC

You can read further details on esptool here: https://github.com/espressif/esptool. You can install esptool using any one of the following methods:

1. Using python package installer, pip:

```
>> pip3 install esptool
```

2. Using Thonny’s package manager:

To access the package manager, open Thonny and go to the menu bar and select “Tools > Manage Packages”. This should pop open a new window with a search field. Type ‘esptool’

into that field and click the “Find package from PyPI” button. Go ahead and click “Install” to install this package. You will see a small window pop up showing the system’s logs while it installs the package. After the installation is finished, the plug-ins window can be closed. Then close and reopen Thonny. Now, you are ready to use esptool.

HELPFUL TIP: You can use this method in future labs to install any additional packages.

1.3.2. Flashing MicroPython on ESP32 Board

Program your board using the *esptool.py* program. If you are putting MicroPython on your board (identified as ***/dev/ttyUSBx****, x=0/1/2) for the first time, then erase the entire flash using:

```
>> esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
```

Then program the firmware starting at address 0x1000 using:

```
>> esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash -z 0x1000 <path to micropython .bin>
```

2. Pins on the ESP 32 Feather board

The ESP32-WROOM-32 module on the Feather Board has 38 pins, many of which are GPIO (general-purpose input/output) pins. However, the feather board has only 28 pins and has its own ad-hoc pin numbering (marked e.g. A0, A1...) with many peripheral pins (like TX, RX, SCL, SDA...) indicated on the board itself. You can explore the ESP32-WROOM-32 datasheet and Feather board manuals and Schematics to figure out the mapping between physical chip pins and board logical pins, and interface LEDs and Switches with the board pins.

2.1. Programming ESP32 using Thonny IDE

Before proceeding further, you should have completed installing any necessary software and flashed MicroPython on the ESP32 board. The implementation procedure explained in this section demonstrates a typical procedure to run any program on the board.

2.2. Connect the Board to PC

Connect the board to your PC using the Micro USB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V which will power the board. In addition, you can also connect the Lithium ion polymer (Lipo/Lipoly) battery to the JST jack (if you have ordered it), which will get continuously charged by the USB Power (OPTIONAL).

Lipoly JST Jack

Micro USB Jack

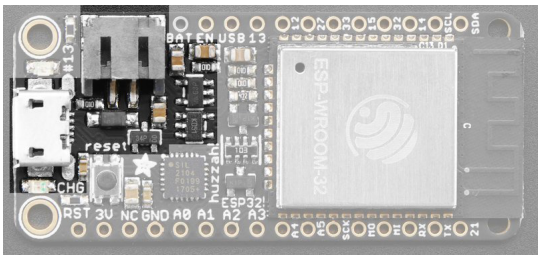

Figure 6. Board Power and PC Connections

You can get more information on starting MicroPython on your ESP32 board on this webpage: http://docs.micropython.org/en/latest/esp32/tutorial/intro.html#esp32-intro.

2.3. Testing Thonny IDE Installation

Important: Before testing the installation, your ESP32 board needs to be flashed with MicroPython firmware. Connect the board to your PC and open Thonny IDE. To test the installation, you need to tell Thonny that you want to run MicroPython Interpreter and select the board you are using.

1. Go to **Tools** > **Options** and select the **Interpreter** tab. Select **MicroPython (ESP32).**
2. Select the device serial port: **Silicon Labs CP210x USB to UART Bridge (COMPort)**

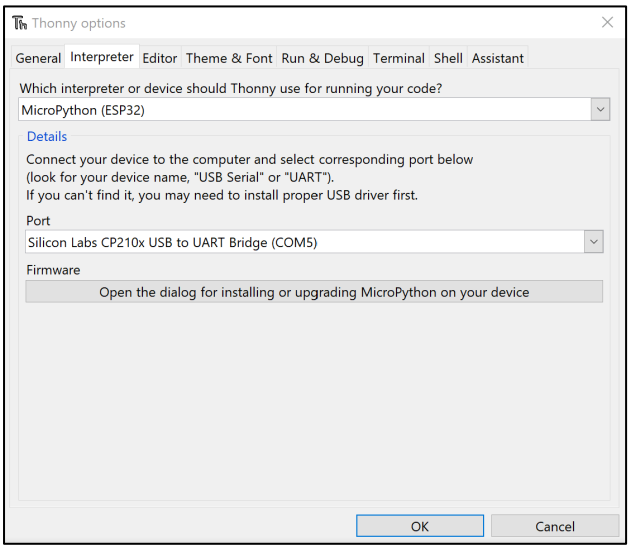

Figure 7. Thonny Interpreter and Port selection

Thonny IDE should now be connected to your board. The board will start and run the files '**boot.py**' and '**main.py**' (if any) automatically and provide you a MicroPython REPL shell. A read–eval–print-loop (REPL), also termed interactive top-level or language shell, is a simple, interactive computer programming environment that takes single user inputs (i.e., single expressions), evaluates (executes) them, and returns the result to the user, shown in Figure 8.

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:5008
ho 0 tail 12 room 4
load:0x40078000,len:10600
ho 0 tail 12 room 4
load:0x40080400,len:5684
entry 0x400806bc
I (538) cpu_start: Pro cpu up.
I (539) cpu_start: Application information:
I (539) cpu_start: Compile time:     Sep  2 2020 03:00:08
I (542) cpu_start: ELF file SHA256:  0000000000000000...
I (548) cpu_start: ESP-IDF:          v3.3.2
I (553) cpu_start: Starting app cpu, entry point is 0x40082f30
I (0) cpu_start: App cpu up.
I (563) heap_init: Initializing. RAM available for dynamic allocation:
I (570) heap_init: At 3FFAFF10 len 000000F0 (0 KiB): DRAM
I (576) heap_init: At 3FFB6388 len 00001C78 (7 KiB): DRAM
I (582) heap_init: At 3FFB9A20 len 00004108 (16 KiB): DRAM
I (588) heap_init: At 3FFBDB5C len 00000004 (0 KiB): DRAM
I (594) heap_init: At 3FFCA9E8 len 00015618 (85 KiB): DRAM
I (601) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (607) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (613) heap_init: At 4009DE28 len 000021D8 (8 KiB): IRAM
I (619) cpu_start: Pro cpu start user code
I (303) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
COMPLETED BOOT
Hello from main
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Figure 8. MicroPython startup on ESP32 in Thonny IDE

2.4. Using the REPL

Once you have a prompt, you can start experimenting! Anything you type at the prompt will be executed after you press the *Enter* key. MicroPython will run the code on the board and print the result (if there is one). If there is an error with the entered text, an error message is printed. For more info., see: http://docs.micropython.org/en/latest/esp8266/tutorial/repl.html (MicroPython execution is similar in ESP8266 and ESP32).

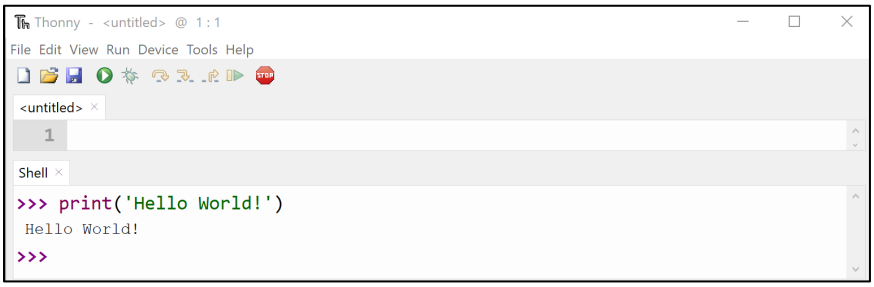

Figure 9. Experimenting with REPL

Type **help()** in the Shell and see if it responds. If it does, everything is working fine.

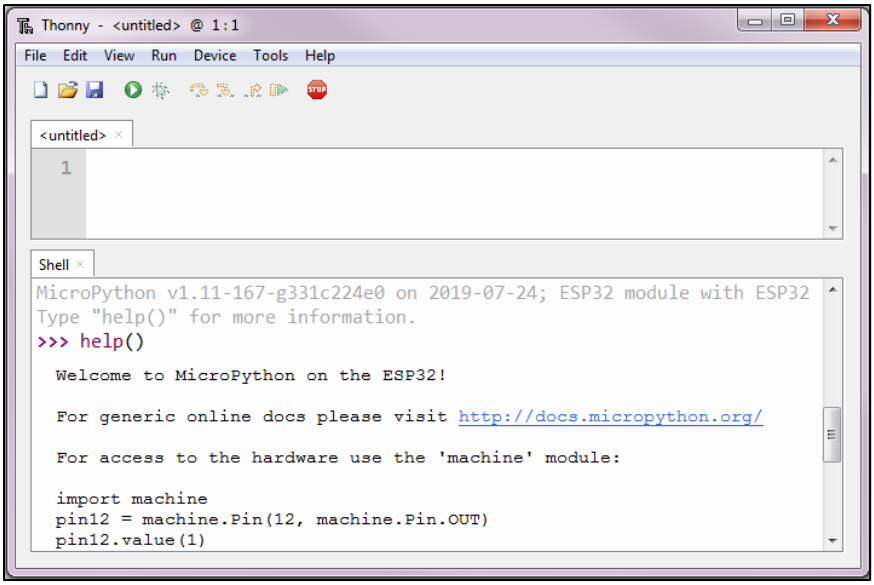

Figure 10. Testing the REPL

2.5. First Program on the ESP32

In the editor region of the Thonny IDE, write your first ***'Hello World'*** program and save it on the board as **main.py**. When the IDE asks for location to save the file, **Select** *MicroPython device* to save the file on the flash memory on the board, as shown in Figure 11. Pressing **Ctrl+D** or selecting Run > Send EOF/Soft Reboot in Thonny will perform a soft reboot of the board and as mentioned earlier, will run **main.py** and display outputs on the MicroPython shell. You can also run the program **main.py** from the shell by calling the following command, which will run the code inside **main.py** and give you output on the shell, both of which are shown in Figure 12. Similarly, any code you write for your labs, you can save them on the board and import your program module in MicroPython shell to run the code.

```
import main
```

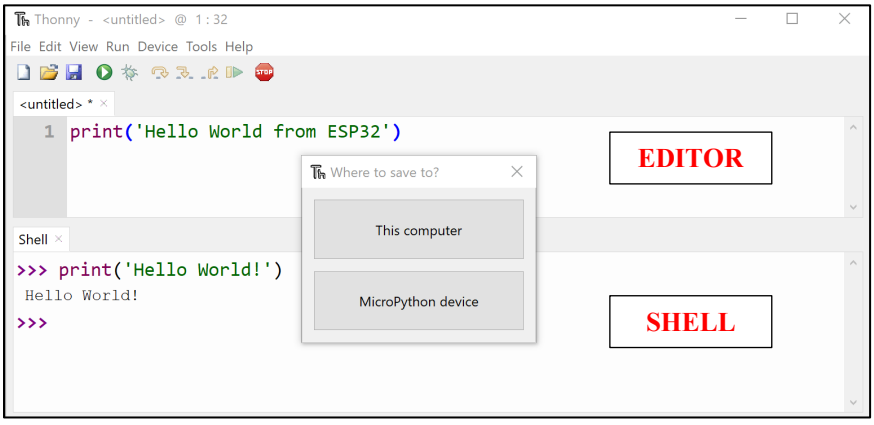

Figure 11. Saving a file on the ESP32 board

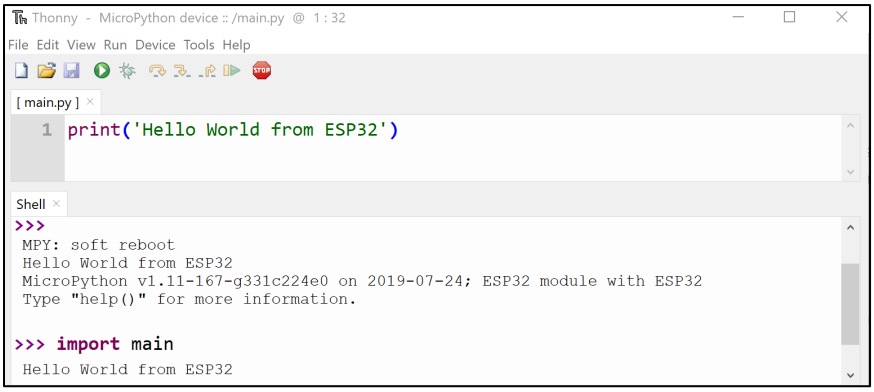

Figure 12. Rebooting and running main.py

2.6. The internal filesystem

Since ESP32 Feather has 4MB flash memory, a default filesystem will be generated upon first boot, which uses the FAT format and is stored in the flash after the MicroPython firmware. You can open any file already stored inside the board using Thonny, modify it and save it back on the board. For more details on working with the internal filesystem, please see this webpage: http://docs.micropython.org/en/latest/esp8266/tutorial/filesystem.html.

2.7. Simple Blink Program (main.py)

The example source code below blinks the Red LED on the ESP32 Feather board 5 times with the interval of roughly 0.5 seconds. To test this, you can save this code as **main.py** file on your board and perform a *soft reboot*. The pin ***X*** will vary depending on which PIN you connect to the red LED on the board (see the Feather board **schematics** for your options).

```
from machine import Pin
from time import sleep

# Onboard RED LED is connected to IO_X
# FIND OUT X FROM SCHEMATICS AND DATASHEET
# Create output pin on GPIO_X
led_board = Pin(X, Pin.OUT)

# Toggle LED 5 times
for i in range(10):
    # Change pin value from its current value, value can be 1/0
    led_board.value(not led_board.value())
    sleep(0.5)    # 0.5 seconds delay

print("Led blinked 5 times")
```

REFERENCES

[1] Getting started with MicroPython on the ESP32
https://docs.micropython.org/en/latest/esp32/tutorial/intro.html

[2] ESP32 WROOM-32 Datasheet
https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[3] ESP32 Technical Reference Manual https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[4] Adafruit HUZZAH32 – ESP32 Feather Online Manual https://learn.adafruit.com/adafruit-huzzah32-esp32-feather

[5] Adafruit ESP32 Feather Schematics https://cdn-learn.adafruit.com/assets/assets/000/041/630/original/feather_schem.png?1494449413

[6] MicroPython GitHub https://github.com/micropython/micropython

[7] REPL http://docs.micropython.org/en/latest/esp8266/tutorial/repl.html

[8] Thonny IDE https://thonny.org