MicroPython

Contents

- Introduction to Micropython
- Features of Micropython
- Setting up micropython on microcontroller
 - Requirements
 - Setting up MicroPython Environment
 - Flashing MicroPython firmware on STM32 NUCLEO board
- Opening MicroPython interactive shell
- Resetting the board
- Toggling LED
- User push button
- UART
- Other Example codes

Introduction to MicroPython

- MicroPython is an efficient implementation of the Python 3 programming language that is designed to run on microcontrollers
- MicroPython provides built-in modules of the Python standard library (e.g. os, time etc), as well as MicroPython-specific modules which give programmer to access low-level hardware (e.g. pyb, machine etc).
- It brings the power and simplicity of Python to embedded systems, making it easier to develop, test, and deploy code on a wide range of hardware platforms.

Features of MicroPython

- **Python3 Syntax**: MicroPython supports Python 3 syntax and many of its features, making it easy to write code for embedded systems.
- Small Footprint: Designed to run on microcontrollers with as little as 256 KB of flash memory
- **REPL** (**Read-Eval-Print Loop**): Interactive prompt that allows for immediate feedback and quick testing of code.
- Cross-Platform: Runs on various microcontroller platforms, including STM32, ESP8266, ESP32, and more.
- Extensive Libraries: Includes libraries for handling hardware-specific functions like GPIO,
 UART etc.

Setting up MicroPython on Microcontroller

Requirements

- Computer with Linux installed, e.g. Ubuntu
- STM32 Nucleo F401RE microcontroller board
- STM32Cube Programmer
- Serial communication utility software such as PuTTY
- Micro-USB cable

Setting up MicroPython Environment

- In Linux machine, install git, make, gcc, and gcc-arm-none-eabi by going to the command terminal and executing the following command sudo apt-get install git make gcc gcc-arm-none-eabi
- Clone the MicroPython source repository by calling git clone https://github.com/micropython/
- Go to the MicroPython directory by calling cd micropython
- Go to the port directory and update STM32 submodules by calling cd ports/stm32
- Build the STM32 firmware for a specific board by calling Syntax: make BOARD={your-board-model-here}

make BOARD=NUCLEO_F401RE

- The Makefile is a text file that automates the process of compiling and linking the code into executable and binary files.
- The Makefile uses GNU toolchain utilities, including 'arm-none-eabi-gcc' for compiling C source files and converts the final ELF executable to HEX and binary formats.
- After compiling all source files, link them into an executable, and generate .hex and .bin files in the build/ directory.

build- NUCLEO F401RE/firmware.elf

GEN build-NUCLEO_F401RE/firmware0.bin

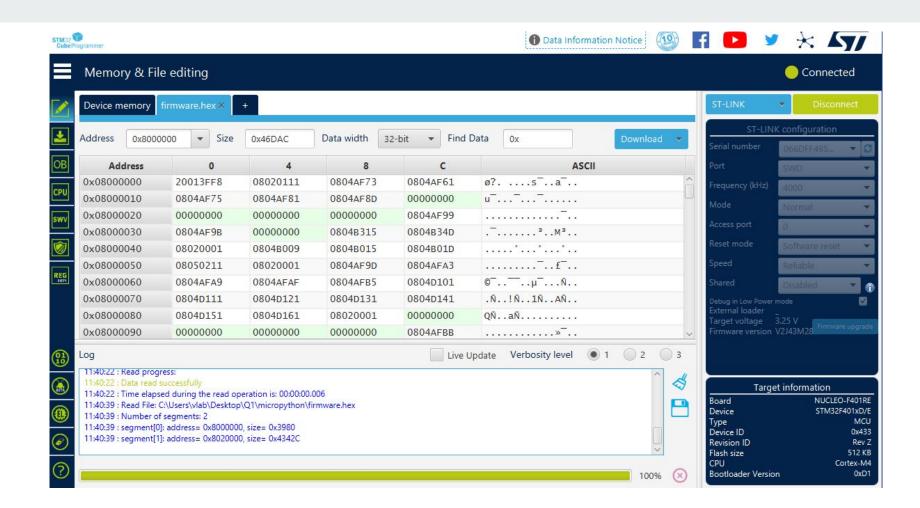
GEN build-NUCLEO_F401RE/firmware1.bin

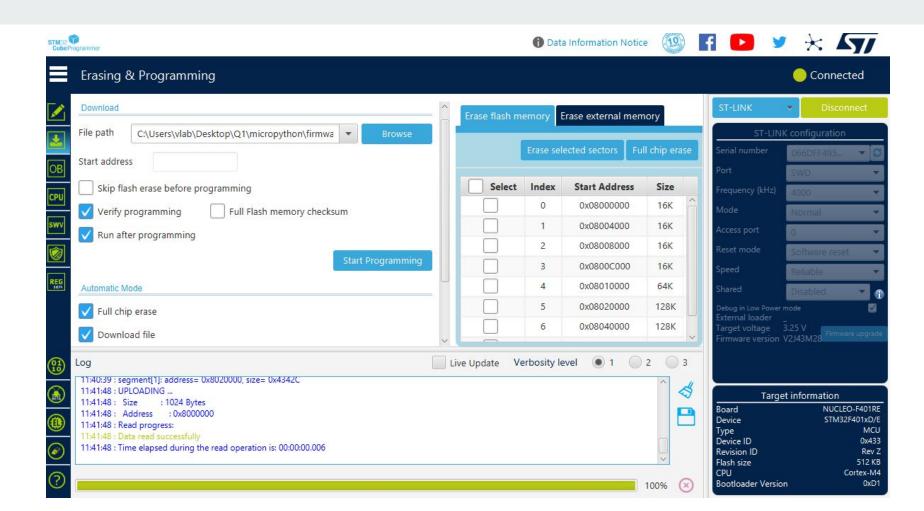
GEN build-NUCLEO_F401RE/firmware.dfu

GEN build-NUCLEO_F401RE/firmware.hex

Flashing MicroPython firmware on STM32 NUCLEO board

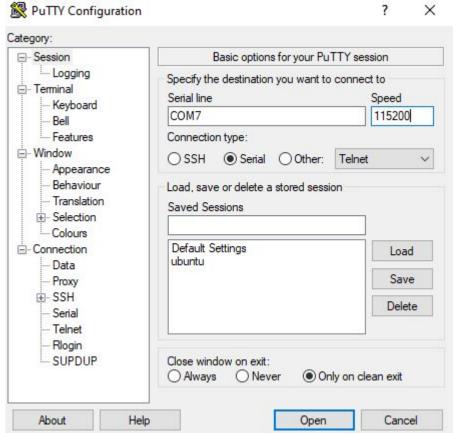
- Connect the board to the computer through a micro USB cable
- Open the STM32Cube Programmer → select the option ST-link near the top right corner to get details of our connected STM board → press Connect button
- Under Erase and programming, browse for firmware.hex file.(A .hex file is a common format used for firmware files that are ready to be flashed onto microcontrollers ,it includes both the binary data (firmware) and additional information like memory addresses.)
- Flash the file by pressing the Start Programming button.





Opening MicroPython Interactive Shell

- On a Windows Computer, open Windows Device Manager.
- Expand "Ports" → Look for the ST-Link Virtual COM Port entry and record the COM port. In my case, my COM port to the STM32 board connected is COM7.
- Open a serial communication utility software such as PuTTY.
- Select "Serial" under Connection Type → Enter the COM7 port you recorded into the
 Serial Line field → Enter 115200 for speed (baud rate) → Click Open
- Click reset (push button) to get the micropython REPL



Resetting the Board

If something goes wrong, you can reset the board in two ways.

• The first is to press CTRL-D at the MicroPython prompt, which performs a soft reset

```
>>>
MPY: sync filesystems
MPY: soft reboot
MicroPython v1.23.0-preview.379.gcfd5a8ea3 on 2024-05-23; NUCLEO-F401RE with STM
32F401xE
Type "help()" for more information.
>>>
```

• If that isn't working you can perform a hard reset by pressing the RST switch. This will end your session, disconnecting whatever program that you used to connect to the board

Toggling LED

Method-1: Using REPL shell

- The easiest way to turn on and off LED attached to the board is using REPL interactive shell.
- Connect the board, We will start by turning LED on and off using following commands

```
>>> myled = pyb.LED(1)
>>> myled.on()
>>> myled.off()
>>>
```

Method-2: Using Python files

- To run the code by using any ".py" file.
- Open the pycharm and write the required code to toggle the LED
- To run that script file on MicroPython-enabled boards we need adafruit-ampy which is a python package.
- It simplifies the process of interacting with the board's filesystem, uploading and downloading files, and executing scripts
- Install the ampy using "pip install adafruit-ampy" in the terminal of the venv of pycharm
- Syntax: "ampy --port <port> run <script.py>"
- "ampy –port COM7 run led.py"

```
Using on() and off() functions:
from machine import Pin
import time
# Define the GPIO pin connected to the LED
led pin = Pin('PA5', Pin.OUT)
# Main loop to toggle the LED
while True:
   led pin.on() # Turn the LED on
  time.sleep(1) # Delay for 1 second
   led pin.off() # Turn the LED off
   time.sleep(1) # Delay for 1 second
```

- The machine module contains specific functions related to the hardware on a particular board. Most functions in this module allow to achieve direct access and control of hardware blocks on a system (like CPU, timers, buses, etc.)
- A pin object is used to control I/O pins (also known as GPIO general-purpose input/output).
- The pin class has methods to set the mode of the pin (IN, OUT, etc) and methods to get and set the digital logic level.
- Pin.on(): Set pin to "1" output level.
- Pin.off(): Set pin to "0" output level.

```
Using high() and low() functions:
from machine import Pin
import time
# Define the GPIO pin connected to the LED
led pin = Pin('PA5', Pin.OUT)
# Main loop to toggle the LED
while True:
   led pin.high() # Turn the LED on
   time.sleep(1) # Delay for 1 second
   led pin.low() # Turn the LED off
```

time.sleep(1) # Delay for 1 second

Using toggle function:

```
led = pyb.LED(1)
while True:
    led.toggle()
    pyb.delay(1000)
```

Pyb module:

- The **pyb** module contains functions related to the specific board.
- **pyb.delay(ms)**: Delay for the given number of milliseconds.
- **class pyb.LED(id)**: Create an LED object associated with the given LED
 - **id** is the LED number \rightarrow (1 4).
 - As STM32F401re has only one LED the id by default corresponds to value 1.

User Push Button

Method-1: Using Pin class of machine module

```
import machine
import time

# Configure PA1 as an input
button = machine.Pin('PC13',
machine.Pin.IN,machine.Pin.PULL_DOWN)

while True:
   if button.value() == 0:
        print("Button pressed!")
   time.sleep(0.1) # Debounce delay
```

```
(.venv) PS C:\Users\vlab\PycharmProjects\MP>
ampy --port COM7 run push_button.py
Button pressed!
Button pressed!
Button pressed!
```

Method-2: Using Switch function()

class Switch – A Switch object of pyb constructor, used to control a push-button switch.

```
import pyb
# Initialize the switch
sw = pyb.Switch()
while True:
   if sw(): # Check if the switch is pressed
       print("Switch pressed")
   else:
       print("Switch not pressed")
   pyb.delay(1000)
                               (or)
sw = pyb.Switch()
while True:
    sw.callback(lambda:print('Switch pressed'))
    pyb.delay(1000)
```

```
(.venv) PS C:\Users\vlab\PycharmProjects\MP>
ampy --port COM7 run switch.py
Switch not pressed
Switch not pressed
Switch not pressed
```

UART

UART implements the standard UART/USART duplex serial communications protocol.

```
import machine
import time
# Initialize UART (use UART2 which is available on the
Nucleo-F401RE)
uart = machine.UART(2, baudrate=115200)
# Main loop to send "hello" over UART
while True:
  uart.write('hello\n') # Send the message "hello"
  time.sleep(1) # Wait for 1 second
   if uart.any(): # Check if there is any incoming data
      msg = uart.read() # Read the received data
      print(msq)
```

```
(.venv) PS C:\Users\vlab\PycharmProjects\MP>
ampy --port COM7 run uart.py
hello
hello
hello
hello
```

Interrupt Handling

Pin.irq(): It is a method of class Pin of the constructor machine, configure an interrupt handler to be called when the trigger source of the pin is active.

```
import machine
import time
# Define a function to be called when the interrupt occurs
def button pressed(b):
  print("Button pressed!")
# Initialize the button pin as an input with pull-down resistor
button = machine.Pin('PC13', machine.Pin.IN, machine.Pin.PULL DOWN)
uart = machine.UART(2, baudrate=115200)
# Attach an interrupt to the button pin
button.irg(trigger=machine.Pin.IRQ RISING, handler=button pressed)
# Main loop
while True:
   uart.write('hello\n') # Send the message "hello"
   time.sleep(1) # Wait for 1 second
   if uart.any(): # Check if there is any incoming data
       msg = uart.read() # Read the received data
       print(msg)
```

```
(.venv) PS C:\Users\vlab\PycharmProjects\MP>
ampy --port COM7 run interrupt.py
hello
hello
hello
Button pressed!
hello
hello
```

Timer

- Each timer consists of a counter that counts up at a certain rate. The rate at which it counts is the peripheral clock frequency (in Hz).
- When the counter reaches the timer period it triggers an event, and the counter resets back to zero. By using the callback method, the timer event can call a Python function.

```
import pyb
led = pyb.LED(1)
# Define the callback function to toggle the LED
def toggle_led(timer):
    led.toggle()
# Initialize Timer 1 with a frequency of 0.2 Hz (1 cycle every 5 seconds)
tim1 = pyb.Timer(1, freq=0.2)
# Set the callback function for Timer 1
tim1.callback(toggle led)
```

Toggle LED when push is button is pressed

```
import pyb
sw = pyb.Switch()
#method1
while True:
   sw.callback(lambda:pyb.LED(1).toggle())
#method 2
def led():
  pyb.LED(1).toggle()
while True:
   sw.callback(led)
```

Thank You