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

Contd...

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/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

Contd...

- 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

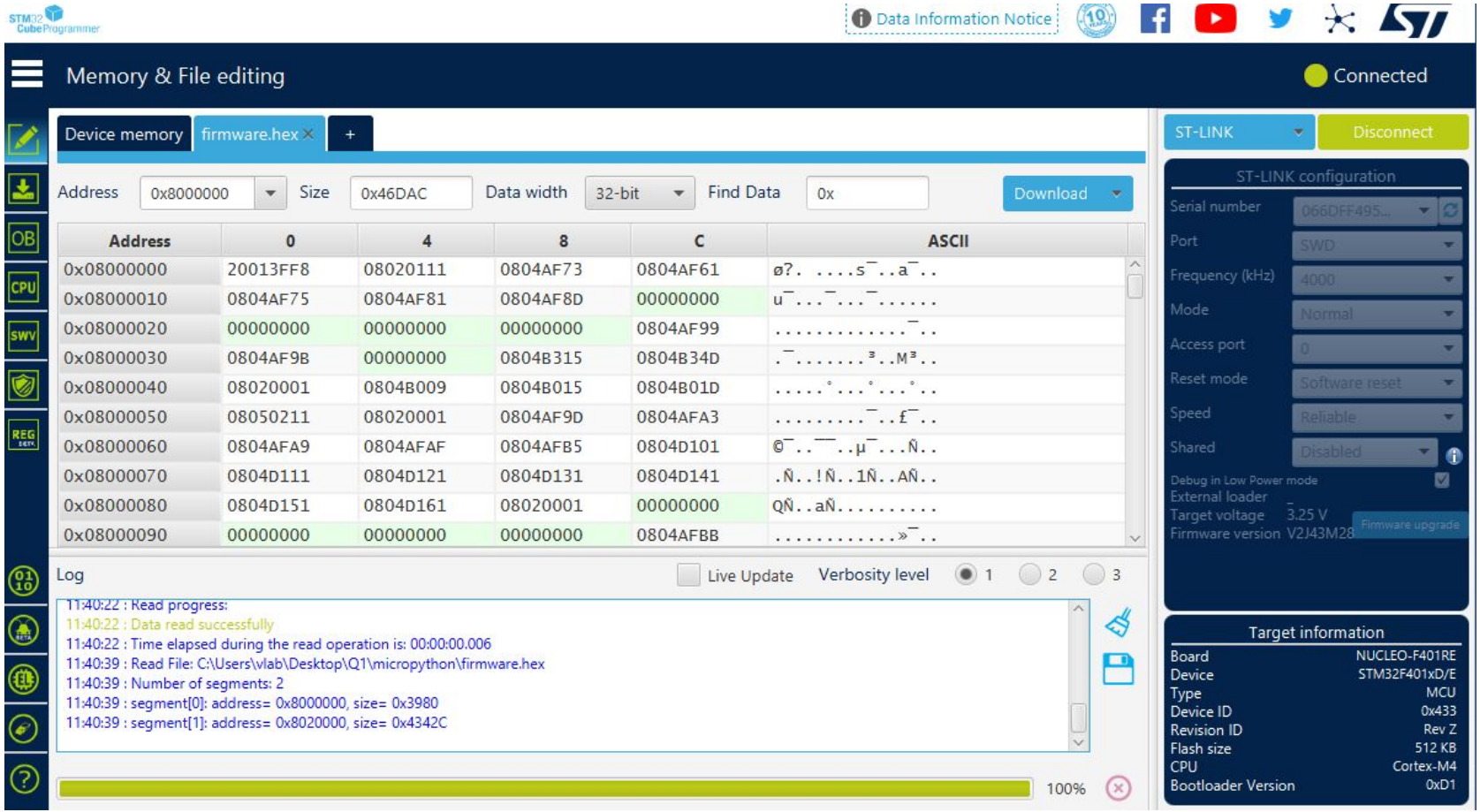
  GEN build-NUCLEO_F401RE/firmware.dfu

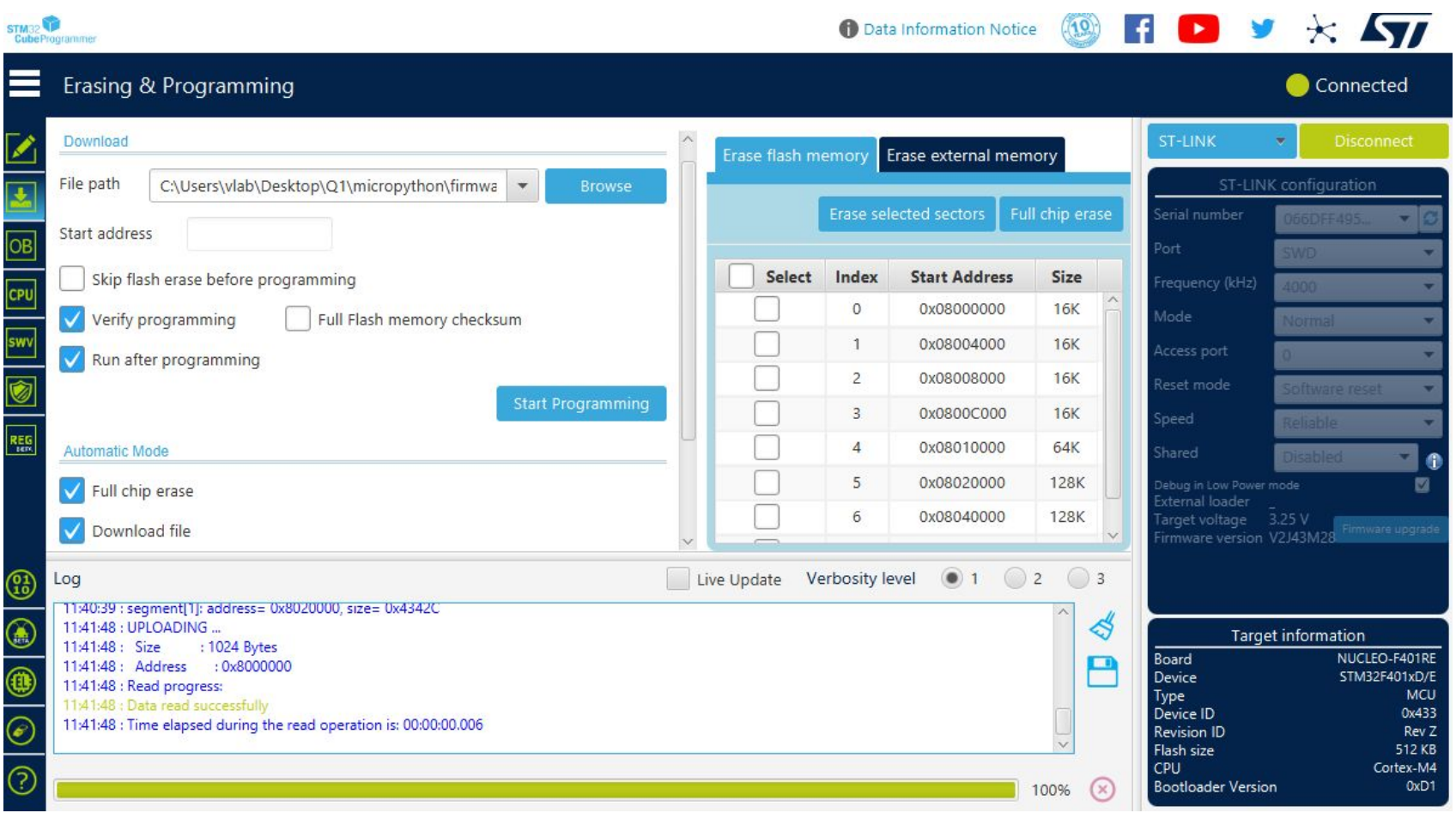
  GEN build-NUCLEO_F401RE/firmware.hex

Contd...

Flashing MicroPython firmware on STM32 NUCLEO board

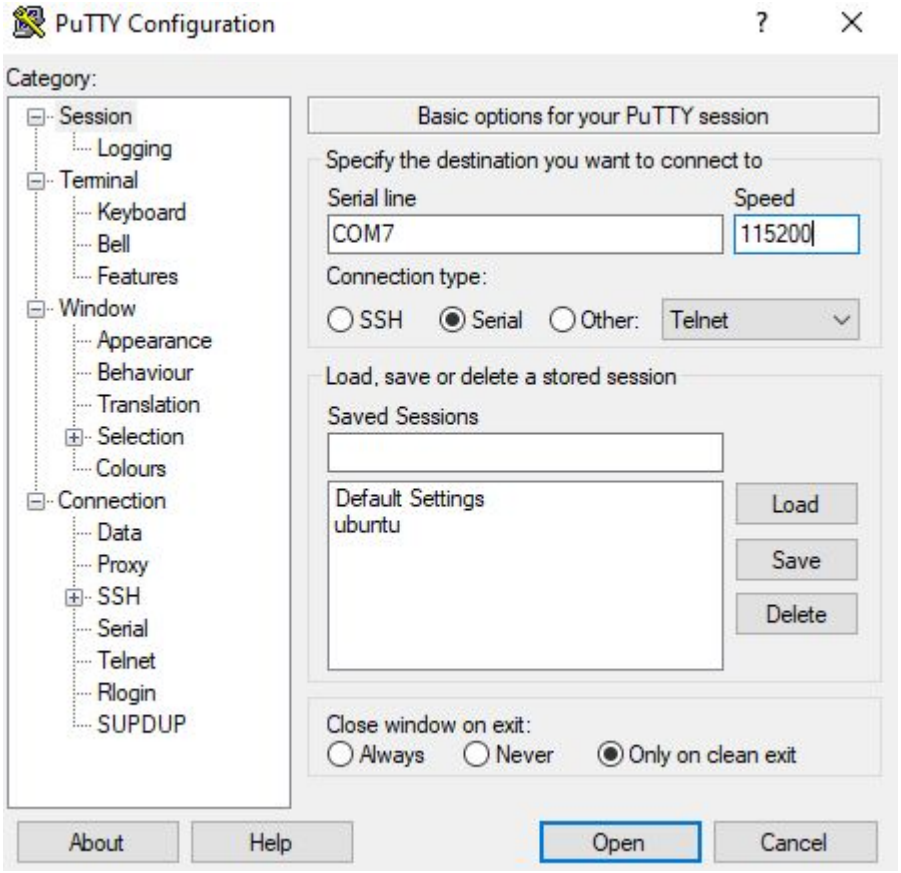
- Connect the board to the computer through a micro USB cable
- Open the STM32Cube Programmer → select the option ST-link near the top right corner to get details of our connected STM board → press Connect button
- Under Erase and programming, browse for firmware.hex file.(A .hex file is a common format 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.

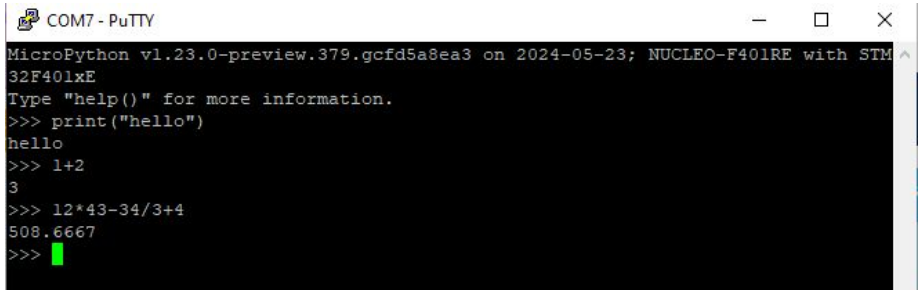

STM32 CubeProgrammer
Data Information Notice
Memory & File editing
Connected
Device memory
firmware.hex
+
Address
0x8000000
Size
0x46DAC
Data width
32-bit
Find Data
0x
Download
Address 0 4 8 C ASCII
0x08000000 20013FF8 08020111 0804AF73 0804AF61
0x08000010 0804AF75 0804AF81 0804AF8D 00000000
0x08000020 00000000 00000000 00000000 0804AF99
0x08000030 0804AF9B 00000000 0804B315 0804B34D
0x08000040 08020001 0804B009 0804B015 0804B01D
0x08000050 08050211 08020001 0804AF9D 0804AFA3
0x08000060 0804AFA9 0804AFAF 0804AFB5 0804D101
0x08000070 0804D111 0804D121 0804D131 0804D141
0x08000080 0804D151 0804D161 08020001 00000000
0x08000090 00000000 00000000 00000000 0804AFBB
Log
Live Update
Verbosity level
1
2
3
11:40:22 : Read progress:
11:40:22 : Data read successfully
11:40:22 : Time elapsed during the read operation is: 00:00:00.006
11:40:39 : Read File: C:\Users\vlab\Desktop\Q1\micropython\firmware.hex
11:40:39 : Number of segments: 2
11:40:39 : segment[0]: address= 0x8000000, size= 0x3980
11:40:39 : segment[1]: address= 0x8020000, size= 0x4342C
100%
ST-LINK
Disconnect
ST-LINK configuration
Serial number
Port
SWD
Frequency (kHz)
4000
Mode
Normal
Access port
0
Reset mode
Software reset
Speed
Reliable
Shared
Disabled
Debug in Low Power mode
External loader
Target voltage
3.25 V
Firmware version
V2J43M28
Firmware upgrade
Target information
Board NUCLEO-F401RE
Device STM32F401xD/E
Type MCU
Device ID 0x433
Revision ID Rev Z
Flash size 512 KB
CPU Cortex-M4
Bootloader Version 0xD1


STM32 CubeProgrammer
Data Information Notice
Erasing & Programming
Connected
Download
File path
C:\Users\vlab\Desktop\Q1\micropython\firmwa
Browse
Start address
Skip flash erase before programming
Verify programming
Full Flash memory checksum
Run after programming
Start Programming
Automatic Mode
Full chip erase
Download file
Erase flash memory
Erase external memory
Erase selected sectors
Full chip erase
Select
Index
Start Address
Size
0 0x08000000 16K
1 0x08004000 16K
2 0x08008000 16K
3 0x0800C000 16K
4 0x08010000 64K
5 0x08020000 128K
6 0x08040000 128K
ST-LINK
Disconnect
ST-LINK configuration
Serial number
Port
Frequency (kHz)
Mode
Access port
Reset mode
Speed
Shared
Debug in Low Power mode
External loader
Target voltage 3.25 V
Firmware version V2J43M28
Firmware upgrade
Log
Live Update
Verbosity level 1 2 3
11:40:39 : segment[1]: address= 0x8020000, size= 0x4342C
11:41:48 : UPLOADING ...
11:41:48 : Size : 1024 Bytes
11:41:48 : Address : 0x8000000
11:41:48 : Read progress:
11:41:48 : Data read successfully
11:41:48 : Time elapsed during the read operation is: 00:00:00.006
100%
Target information
Board NUCLEO-F401RE
Device STM32F401xD/E
Type MCU
Device ID 0x433
Revision ID Rev Z
Flash size 512 KB
CPU Cortex-M4
Bootloader Version 0xD1

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


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
SSH
Serial
Telnet
Rlogin
SUPDUP
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM7
115200
Connection type:
SSH
Serial
Other:
Telnet
Load, save or delete a stored session
Saved Sessions
Default Settings
ubuntu
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Help
Open
Cancel


COM7 - PuTTY
MicroPython v1.23.0-preview.379.gcfd5a8ea3 on 2024-05-23; NUCLEO-F401RE with STM32F401xE
Type "help()" for more information.
>>> print("hello")
hello
>>> 1+2
3
>>> 12*43-34/3+4
508.6667
>>>

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()
>>>
```

Contd...

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”**

Contd...

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
```

Contd...

- 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.

Contd…

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
```

Contd...

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 → (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!
```

Contd...

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)

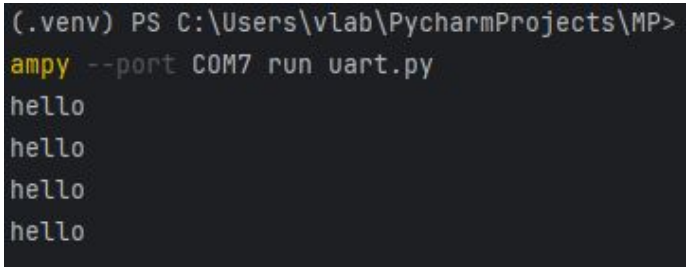
```
sw = pyb.Switch()
while True:
    sw.callback(lambda:print('Switch pressed'))
    pyb.delay(1000)
```

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



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.irq(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)
```

Control LED brightness using PWM

```
from pyb import Pin, Timer
import time

# Define the pin connected to the LED (PA5
for STM32F401)
led_pin = Pin('PA5')

# Create a Timer object using pyb module
tim = Timer(2, freq=1000)  # Timer 2, 1
kHz frequency

# Configure the Timer channel for PWM
using pyb module
ch = tim.channel(1, Timer.PWM,
pin=led_pin)
```

```
# Function to ramp up and down the PWM duty cycle
def ramp_pwm():
    try:
        for duty_cycle in range(0, 101, 5):
            # Ramp up from 0% to 100%
            ch.pulse_width_percent(duty_cycle)
            time.sleep(0.1)

        for duty_cycle in range(100, -1, -5):
            # Ramp down from 100% to 0%
            ch.pulse_width_percent(duty_cycle)
            time.sleep(0.1)

# Execute the ramp_pwm function continuously
while True:
    ramp_pwm()

# Cleanup (though it won't be reached in this
script)
tim.deinit()
```

Reading Internal Temperature Sensor value using ADC

```
from pyb import ADC
import time

# Initialize the ADC on the internal temperature sensor channel (typically ADC channel 16)
temp_sensor = ADC(16)

# Function to convert raw ADC value to temperature
def raw_to_temperature(raw_value):
    # STM32 internal temperature sensor calibration values
    V25 = 0.76  # Voltage at 25 degrees Celsius (in Volts)
    Avg_Slope = 2.5  # Average slope (in mV/degree Celsius)
    V_ref = 3.3  # Reference voltage (in Volts)

    # Convert the raw ADC value to a voltage
    voltage = (raw_value / 4095) * V_ref

    # Calculate temperature in Celsius
    temperature = (voltage - V25) / (Avg_Slope / 1000) + 25
    return temperature

# Continuously read and print the temperature
while True:
    raw_value = temp_sensor.read()
    temperature = raw_to_temperature(raw_value)
    print("Temperature: {:.2f}C".format(temperature))
    time.sleep(1)  # Delay for 1 second between readings
```

```
(venv) PS C:\Users\vlab\PycharmProjects\Micro
Python_codes> ampy --port COM7 run .\ADC_inte
rnal_Temp_Sensor.py
Temperature: 24.33C
Temperature: 24.33C
Temperature: 24.00C
Temperature: 23.68C
Temperature: 23.68C
Temperature: 23.68C
Temperature: 24.65C
Temperature: 24.00C
```

Thank You