STMICROELECTRONICS

STM32WB55 GO FUTHER

NUCLEO STM32WB55

Christophe Priouzeau

8 février 2021

Chapitre 1

Introduction

MicroPython et **STM32WB55**

The goal of this document are to explain how to play with STM32WB55 board.

Note

The majority of examples and explanations are available on web site : https ://stm32python.gitlab.io/fr/

Chapitre 2

Warning

With the manipulation of NUCLEO-WB55 and MicroPython, you can encounter some issue of File System.

MicroPython provide a serial port and a FAT File system(PYBFLASH) on USB OTG.
The source sode executed are present on FAT File system, with the manipulation of file on your PC you can crash easly the file system.

Recommendation :

— When you edit some file on FAT file system (PYBFLASH), DON'T FORGOT TO SAVE FILE BEFORE TO REMOVE USB CABLE.

— When you would like to remove USB cable, DON'T FORGOT TO UMOUNT CLEANLY FAT file system (PYBFLASH).

— When you would like to test a new version of code, PLEASE USE CTRL+C and CTRL+D ON SERIAL PORT.
CTRL+C : to stop previous execution CTRL+D : ask to MicroPython to perform a UMOUNT and a RESET

— When you would like to test a new version of code and MicroPython repeat there is an error on code, YOU MUST UMOUNT FAT file system (PYBFLASH) AND UNPLUG/PLUG USB CACBLE.

Table des matières

Chapitre 3

Hardware

3.1 NUCLEO STM32WB55 Board

The STM32-NUCLEO board is designed with the components necessary to stat the microcontroller. Extension connectors (ARDUINO, MORPHO) allow the user to connect external electronic components to the STM32.
The electronic card also has a connector for CR2032 battery, 3 buttons (SW1, SW2, SW3), 3 LEDs (red, green and blue) and 2 micro-USB connectors .

Description of the Nucleo STM32BW55 board

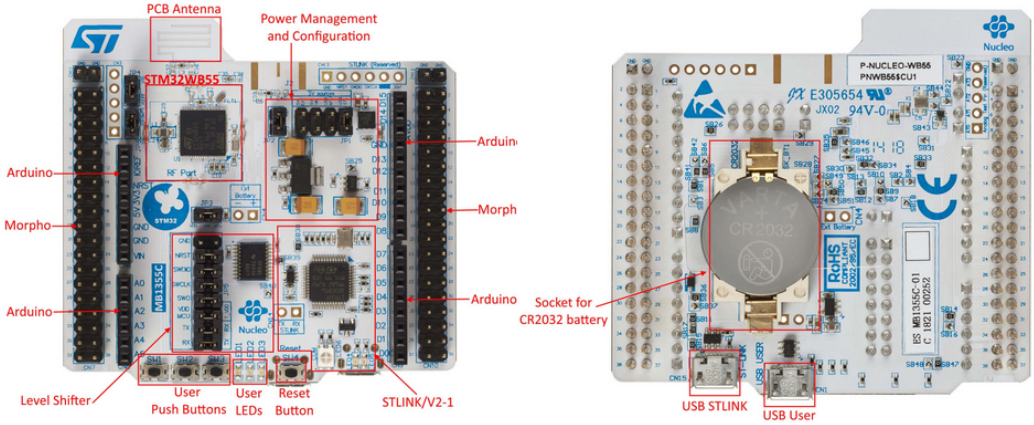


FIGURE 3.1 – Top and bottom view of the STM32BW55 Nucleo board

PCB Antenna

This is the bluetooth antenna of the STM32WB55 microcontroller. It is a so-called PCB antenna because it is integrated directly into the circuit of the board. Its complex shape is designed so as to optimize the reception and emission of radio waves for the bluetooth frequency, i.e. 2.4 GHz.

Connectors Arduino / Morpho

Extension kits can be simply added with these standardized connectors.
We can, for example, connect the Mems kit (X-NUCLEO-IKS01A3) to the card NUCLEO-WB5.

USB User

This is the USB port that will be used to :
— Communicate with the MicroPython interpreter ;
— Program the system ;
— Power on the development kit ;

Socket CR2032

Once the system has been programmed, it will be possible to supply the kit with a CR2032 battery, in order to make the system portable.

Note

The usage of CR2032 request to change a switch (put a 0 Ohm on switch SB29) on board but when the switch are enabled you don't have access to STLINK and USB OTG.
(CF user manual of NUCLEO-WB55 on www.st.com)

STLINK

This is the tool that allows you to program the microcontroller, for this use the "drag" to deposit the firmware on the USB device : WB55-NODE.

User LEDs

These LEDs are accessible to the developer and allow to be activated thanks to MicroPython.

3.2 USBDONGLE STM32WB55

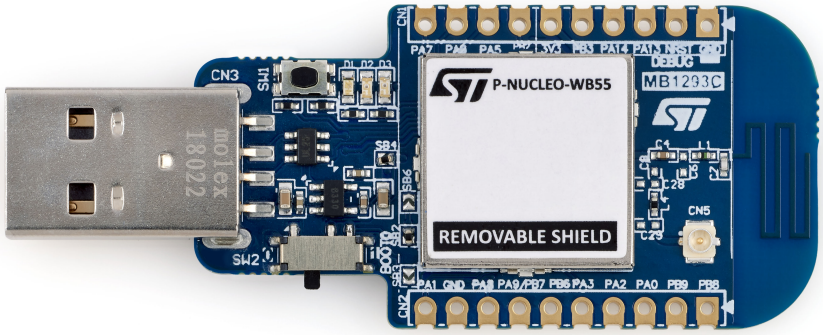


FIGURE 3.2 – USB DONGLE STM32BW55

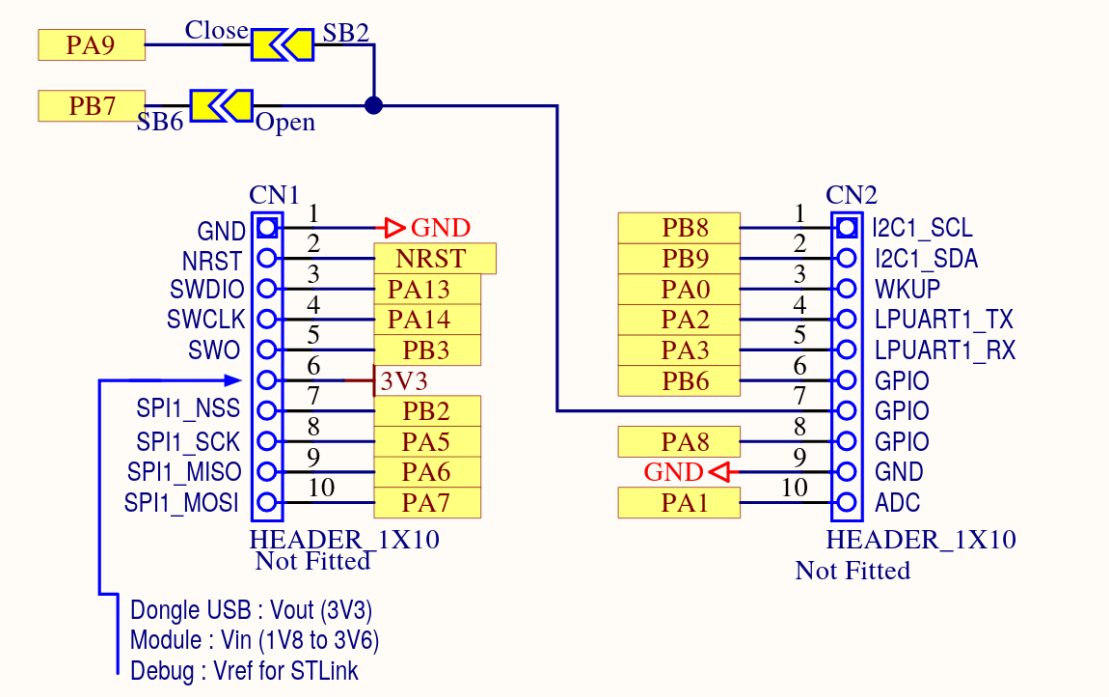


FIGURE 3.3 – USB DONGLE STM32BW55 schema

3.3 Extension Shield IKS01A3

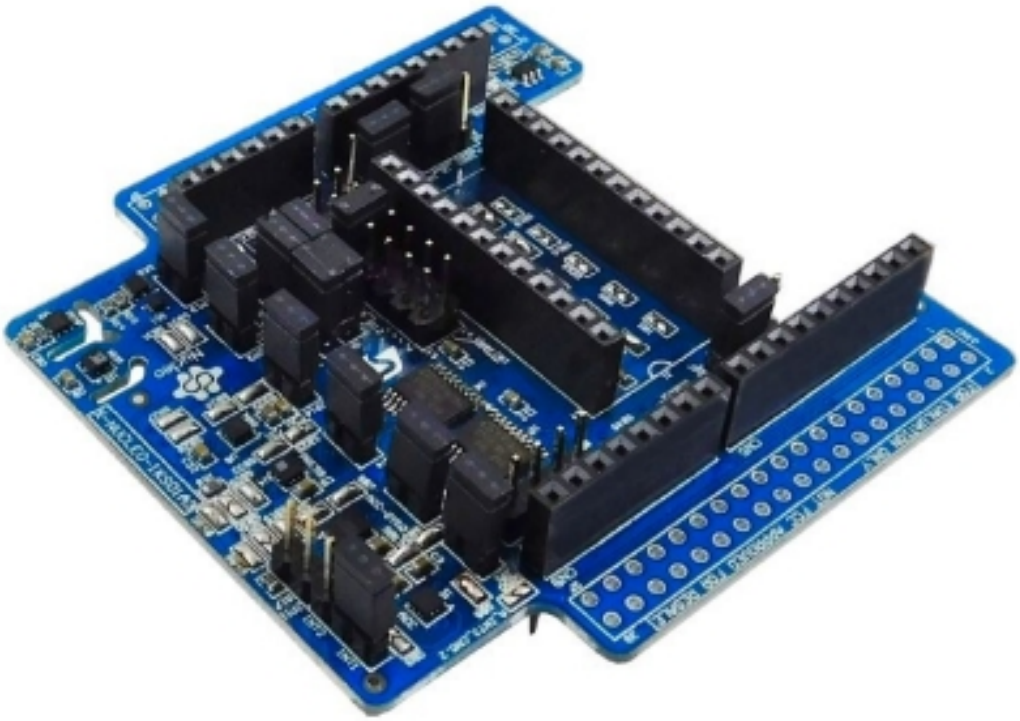


FIGURE 3.4 – Top view of the IKS01A3 expansion card

The IKS01A3 expansion board is a demonstration board of several MEMS sensors from ST Microelectronics.
Its A3 version contains the following sensors

- LSM6DSO : Accelerometer 3D + Gyroscope 3D ;
- LIS2MDL : Magnetometer 3D ;
- LIS2DW12 : Accelerometer 3D ;
- LPS22HH : Barometer (260-1260 hPa) ;
- HTS221 : Relative humidity sensor ;
- STTS751 : Relative temperature sensor (–40 °C to +125 °C) ;

These sensors are connected to the I2C bus of the NUCLEO-WB55 board.

I2C : Inter-Integrated Circuit (or computer communication bus). This bus allows you to connect several components on the same bus.

Each component which is connected on an I2C communication bus has a unique identifier :

Sensor	**Id** (on en Hexa)	**Id** (on decimal)
LSM6DSO	0x6B	107
LIS2MDL	0x1E	30
LIS2DW12	0x19	25
LPS22HH	0x5D	93
HTS221	0x5F	95
STTS751	0x4A	74

3.4 Breadboard or "plaque à bidouille"

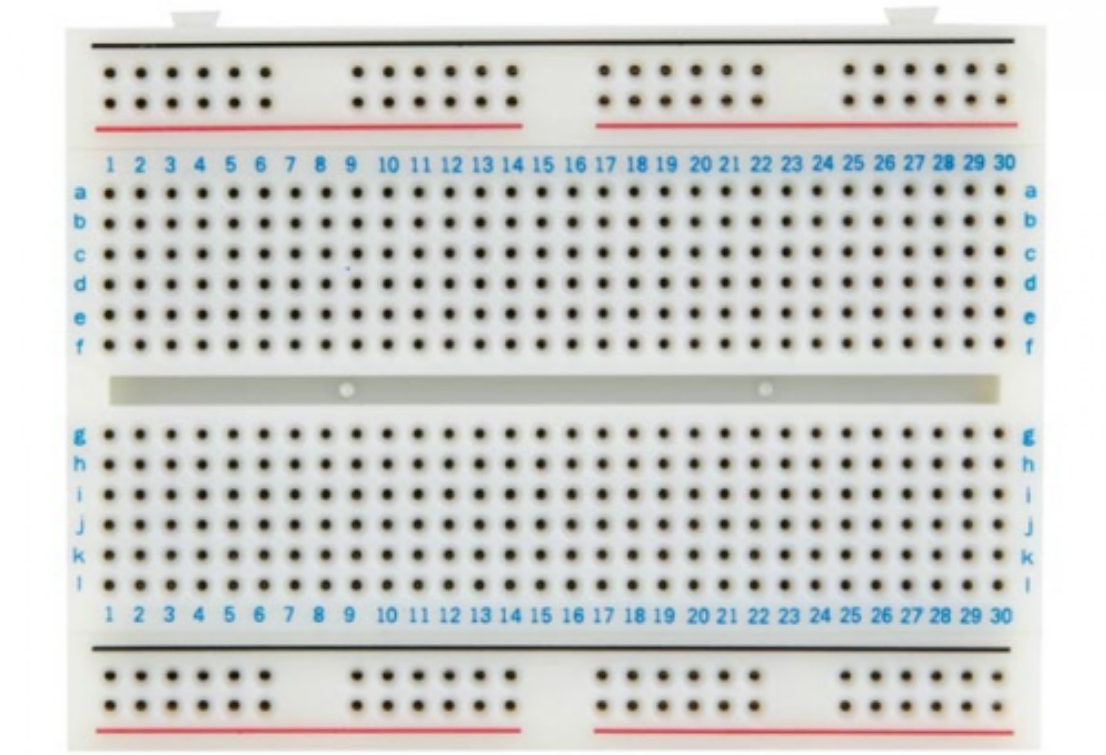


FIGURE 3.5 – Breadboard or "plaque à bidouille"

A "breadboard" allows you to easily make connections between components (resistors, LEDs, sensors, etc ...) without soldering. It allows you to easily test a temporary assembly.
A breadboard is a plate full of holes connected together according to a very specific scheme common to all the plates here :

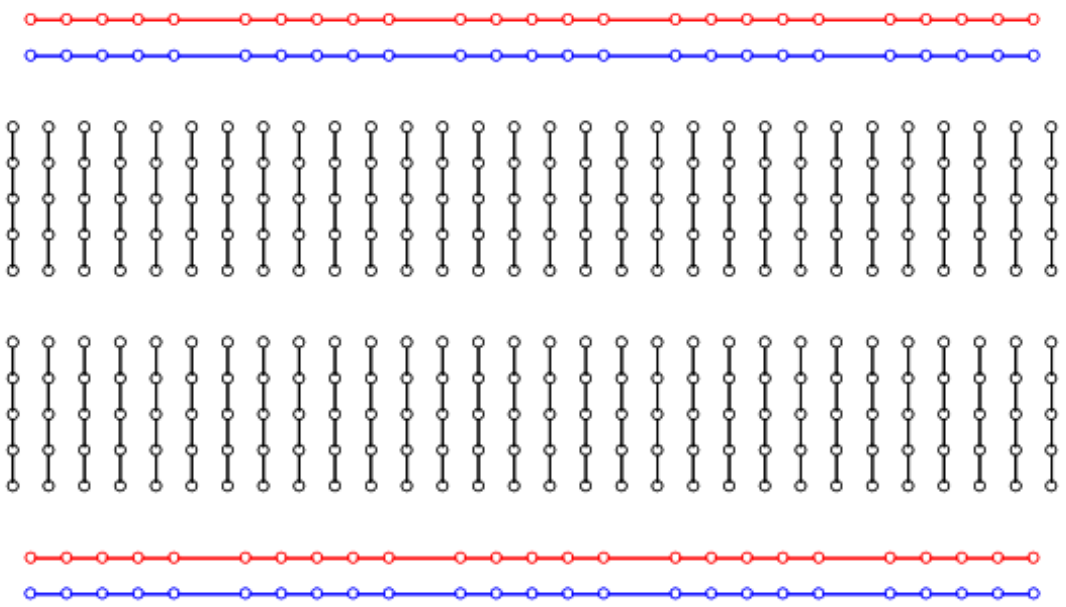


FIGURE 3.6 – Breadboard connection diagram

The holes of the two lines at the top and the bottom are electrically connected **horizontaly**.
The other holes located between the two lines at the top and at the bottom are connected **verticaly**.

3.5 I2C screen

FIGURE 3.7 – SSD1306 I2C screen

SSD1306 is an I2C OLED (Organic Light-Emitting Diode) display.
I2C : Inter-Integrated Circuit (or computer communication bus). This bus allows you to connect several components on the same bus .
Each component which is connected on an I2C communication bus has a unique identifier :

Screen	**Id** (on Hexa)	**Id** (on decimal)
SSD1306	0x3C	60

NUCLEO-WB55 Connection to the board

We can find two versions of this SSD1306 screen, the difference is on the order of the connectors :
— GND, VCC, SCL, SDA
— VCC, GND, SCL, SDA
Definition :
— **GND** : *ground*.
— **VCC** : *supply voltage*.
— **SCL** : *serial clock line input*.
— **SDA** : *serial data line*.

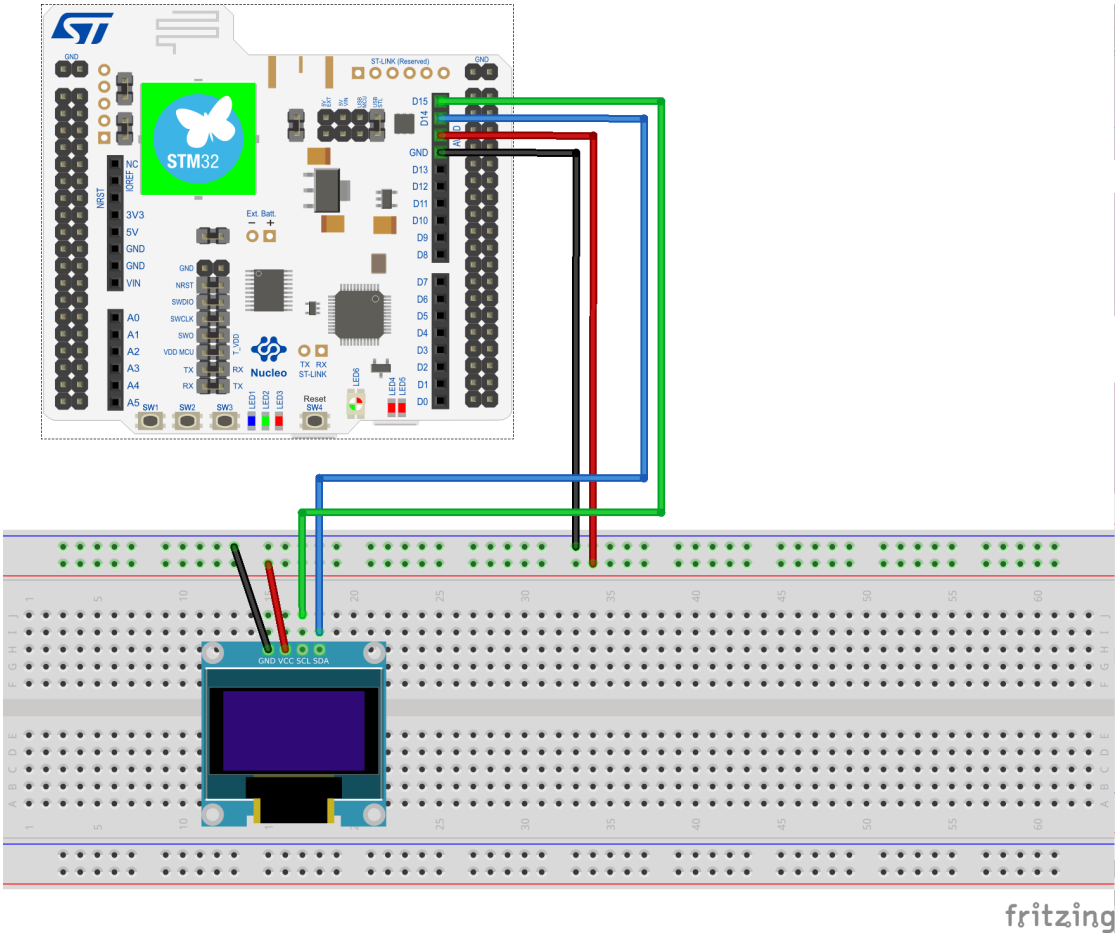


FIGURE 3.8 – Wiring GND VCC SCL SDA"

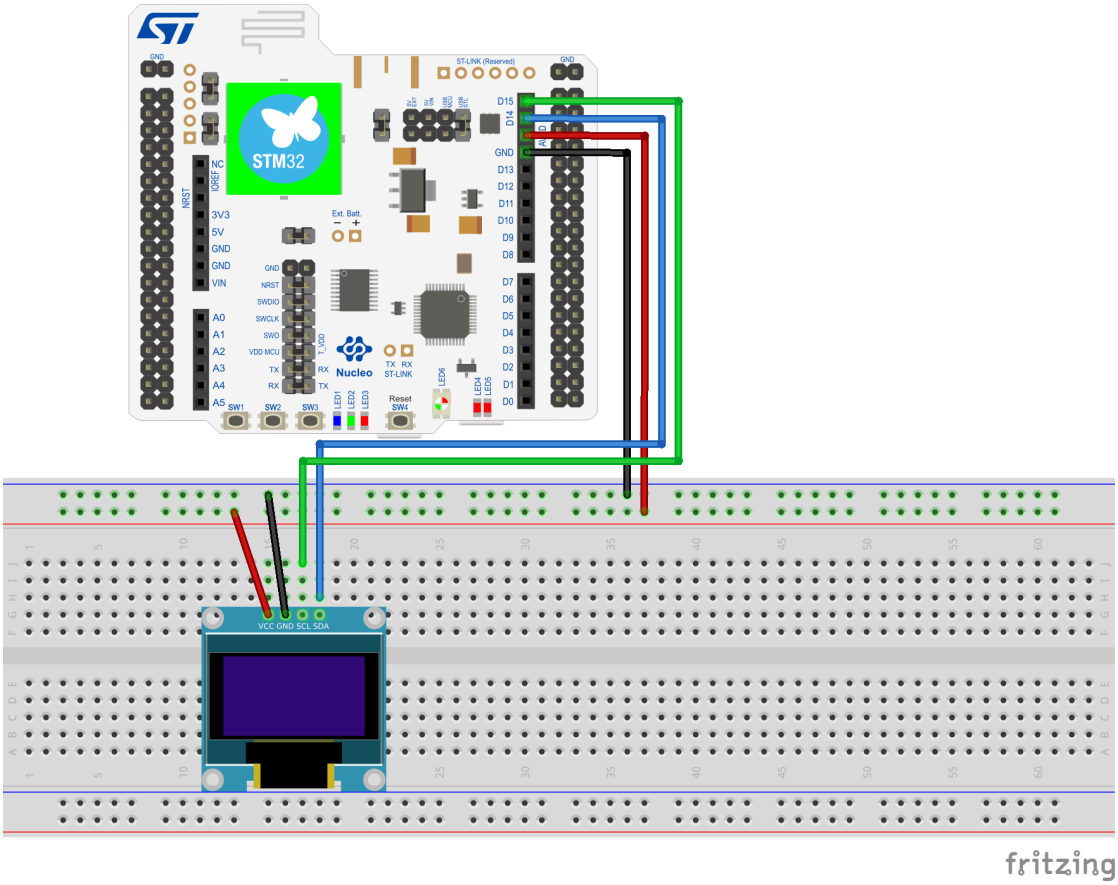


FIGURE 3.9 – Wiring VCC GND SCL SDA

Wiring :
— **GND** connect to arduino connector **GND**.
— **VCC** connect to arduino connector **AVDD**.
— **SCL** connect to arduino connector **D15**.
— **SDA** connect to arduino connector **D14**.

USBDONGLE-WB55 Connection to the board

We can find two versions of this SSD1306 screen, the difference is on the order of the connectors :
— GND, VCC, SCL, SDA
— VCC, GND, SCL, SDA
Definition :
— **GND** : *ground*.
— **VCC** : *supply voltage*.
— **SCL** : *serial clock line input*.
— **SDA** : *serial data line*.

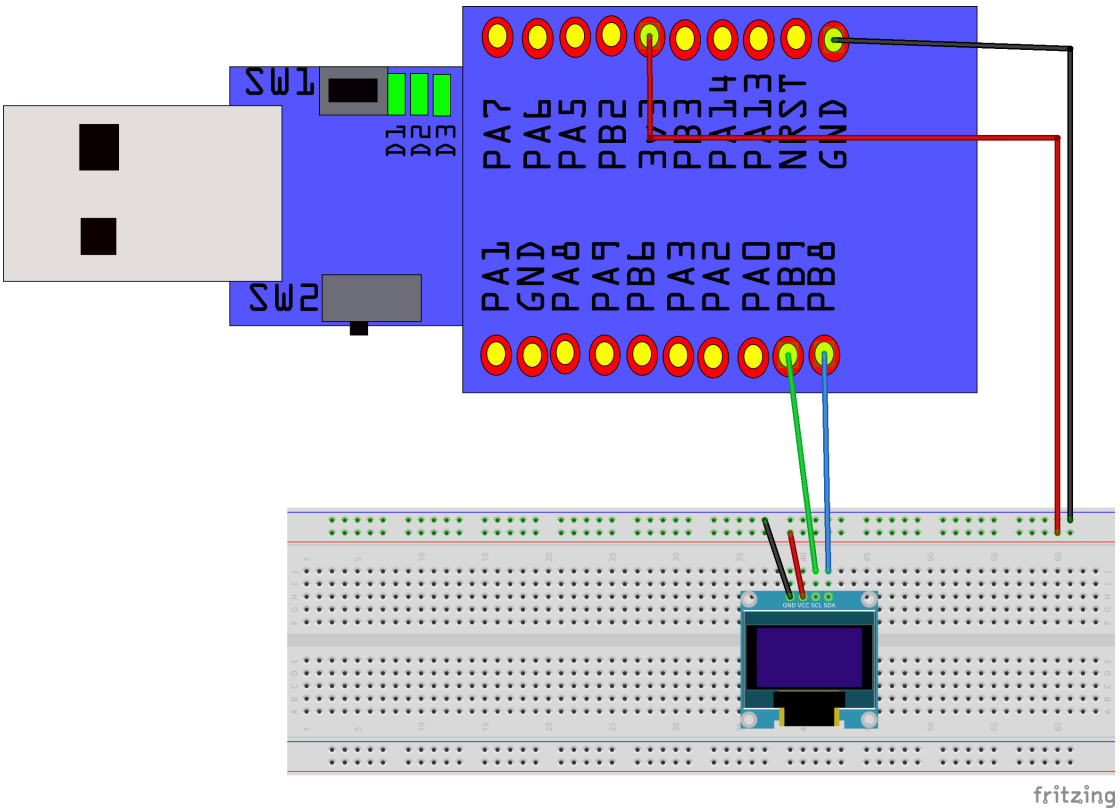


FIGURE 3.10 – Wiring GND VCC SCL SDA"

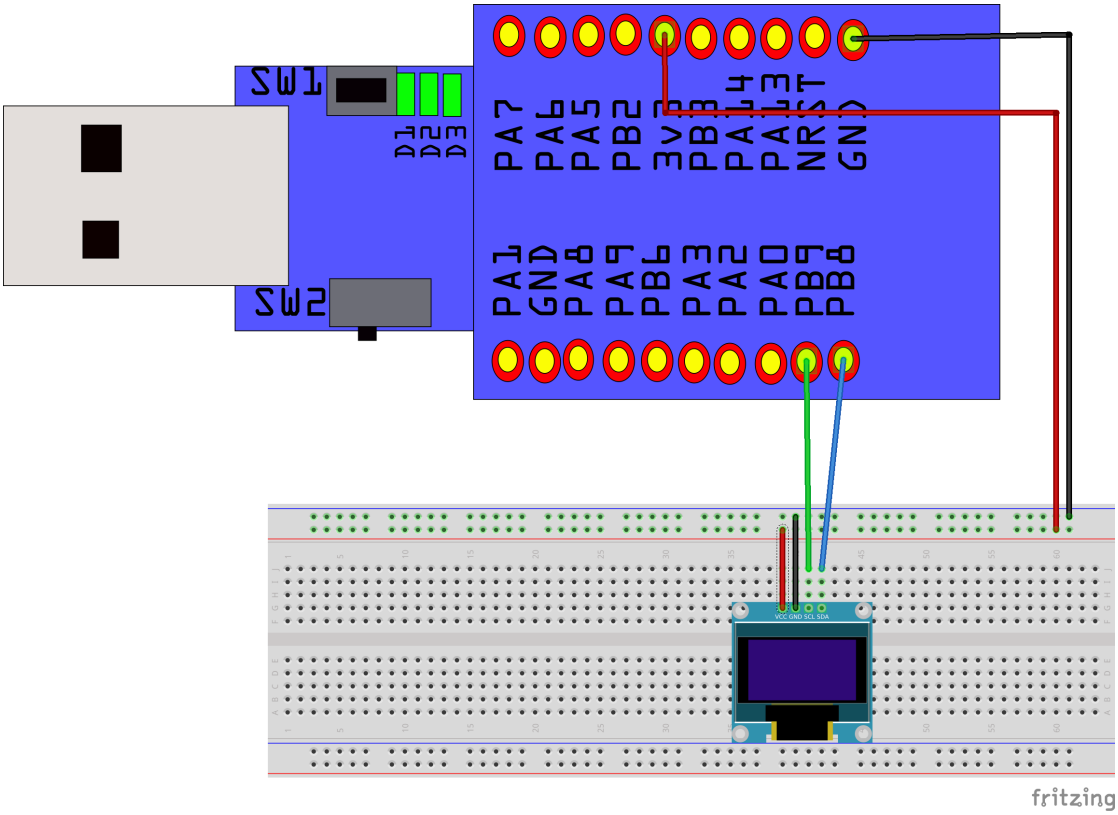


FIGURE 3.11 – Wiring VCC GND SCL SDA

Wiring :
— **GND** connect to arduino connector **GND**.
— **VCC** connect to arduino connector **AVDD**.
— **SCL** connect to arduino connector **D15**.
— **SDA** connect to arduino connector **D14**.

3.6 LED Matrices

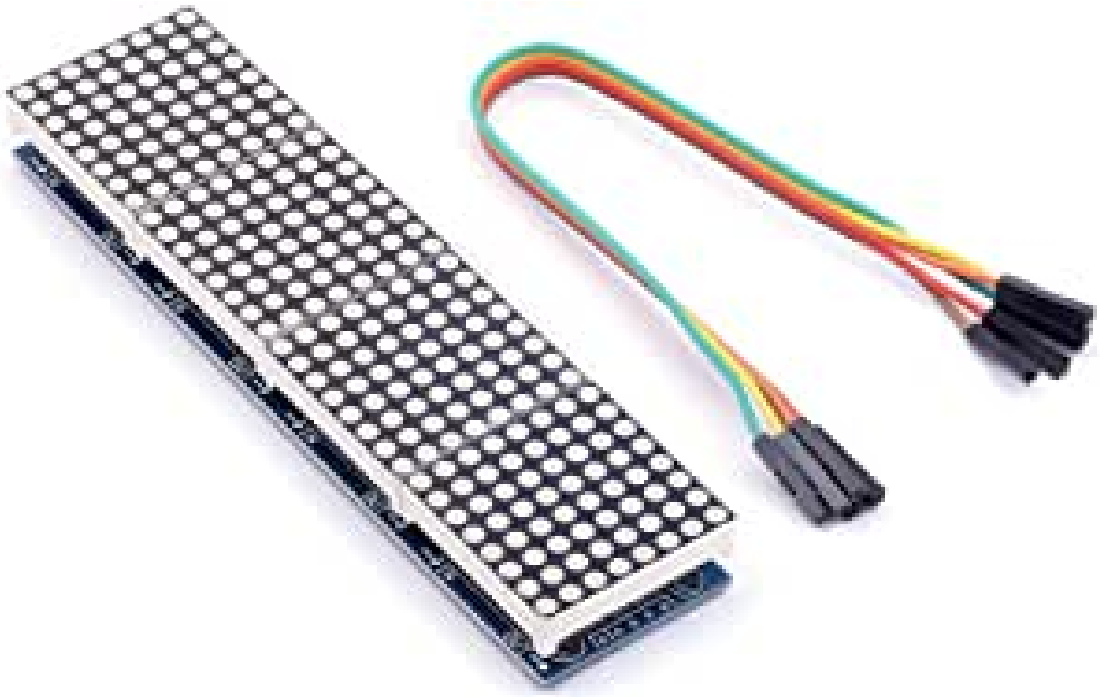


FIGURE 3.12 – LED Matrices with MAX7219 controler

The LED matrix with MAX7219 controller is connected to the arduino connector on the SPI (Serial Peripharal Interface) link.

Connection to the board

List of connectors :
— **GND** : *ground*.
— **VCC** : *supply voltage*.
— **DIN** (MOSI) : *Master output, slave input*.
— **CS** (SS) : *chip select*.
— **CLK** (SCLK) : *serial Clock*.

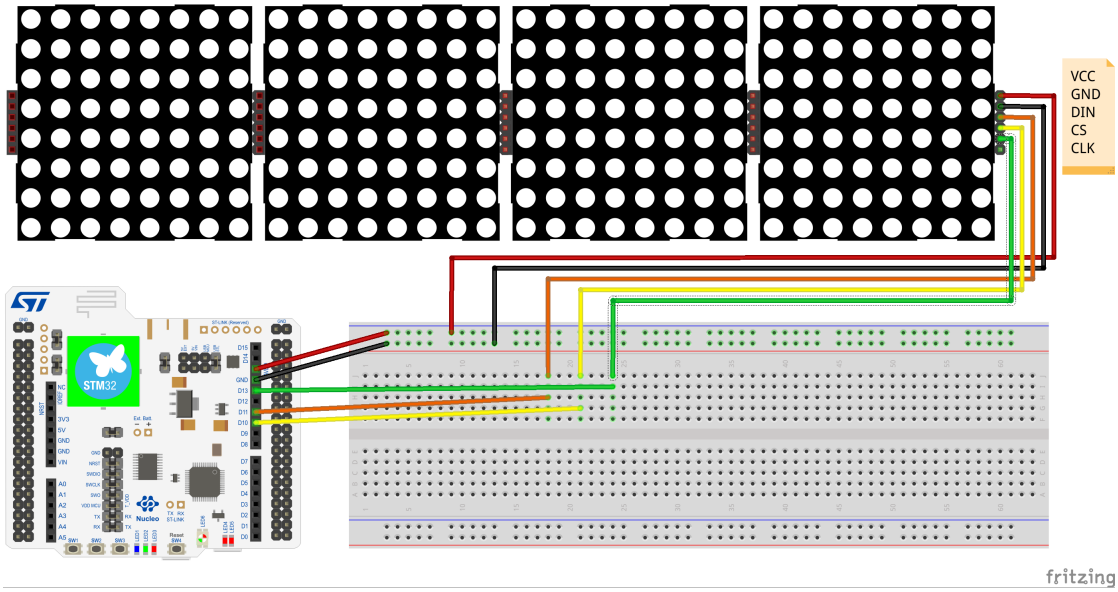


FIGURE 3.13 – Wiring MAX7219"

Câblage :

- **GND** connect to arduino connector **GND**.
- **VCC** connect to arduino connector **5V**.
- **DIN** connect to arduino connector **D11**.
- **CS** connect to arduino connector **D10**.
- **CLK** connect to arduino connector **D13**.

With USBDONGLE WB55

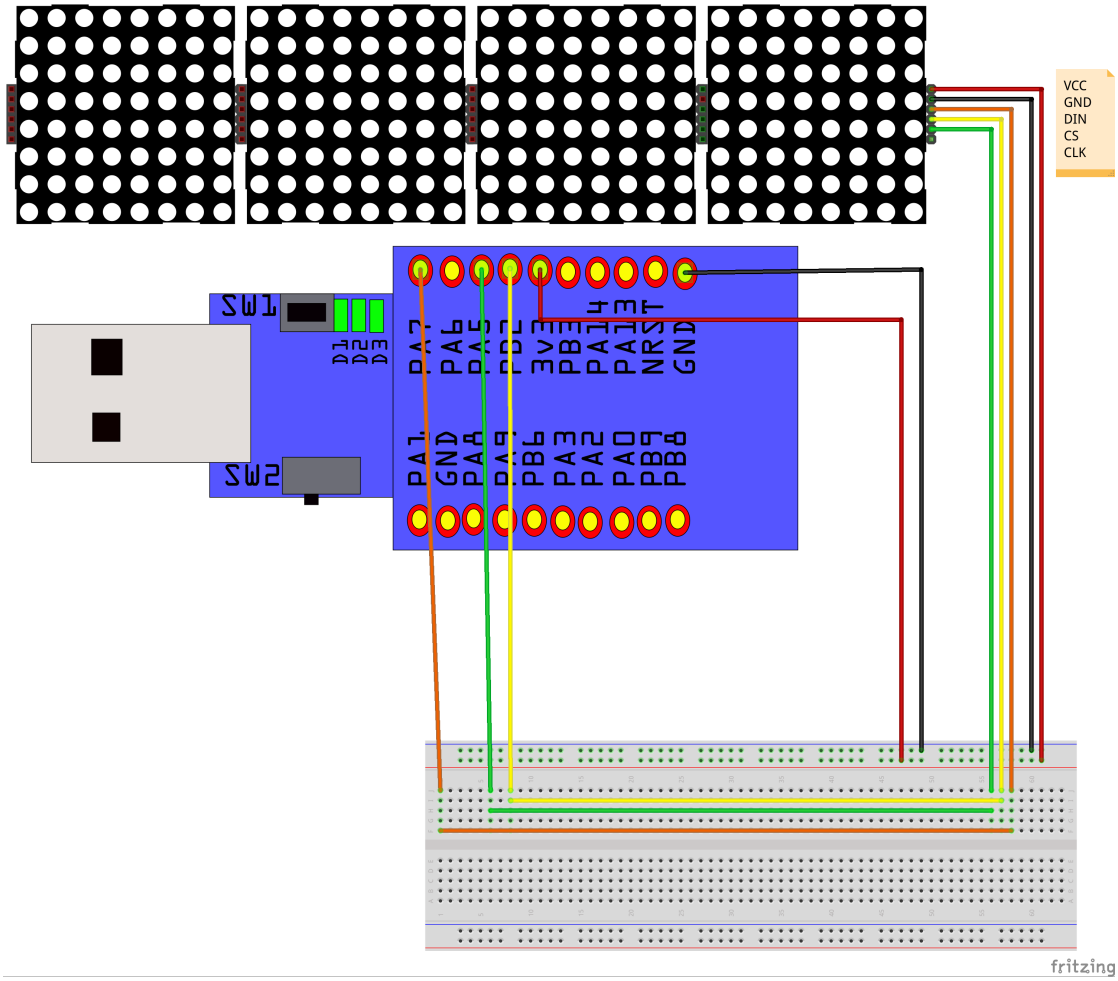


FIGURE 3.14 – Wiring MAX7219 with USBDONGLE"

Câblage :

- **GND** connect to arduino connector **GND**.
- **VCC** connect to arduino connector **3V3**.
- **DIN** connect to arduino connector **PA7**.
- **CS** connect to arduino connector **PB2**.
- **CLK** connect to arduino connector **PA5**.

Chapitre 4

Pre-requisite

4.1 Windows 10 Getting Started Guide

In this getting started guide we will :
— Program the board with the firmware.
— Use a UART communication terminal (TeraTerm or PUTTY).
— Program our first Python script on the Nucleo board.

4.1.1 Flashing NUCLEO-WB55 board

You will need a USB to micro USB cable.
This configuration can be summed up in :
— Move and make sure the jumper is in the position **USB_ST_LINK**
— Connect a micro USB cable to the ST LINK port below the LED4 and LED5
Here is how the kit should be configured :

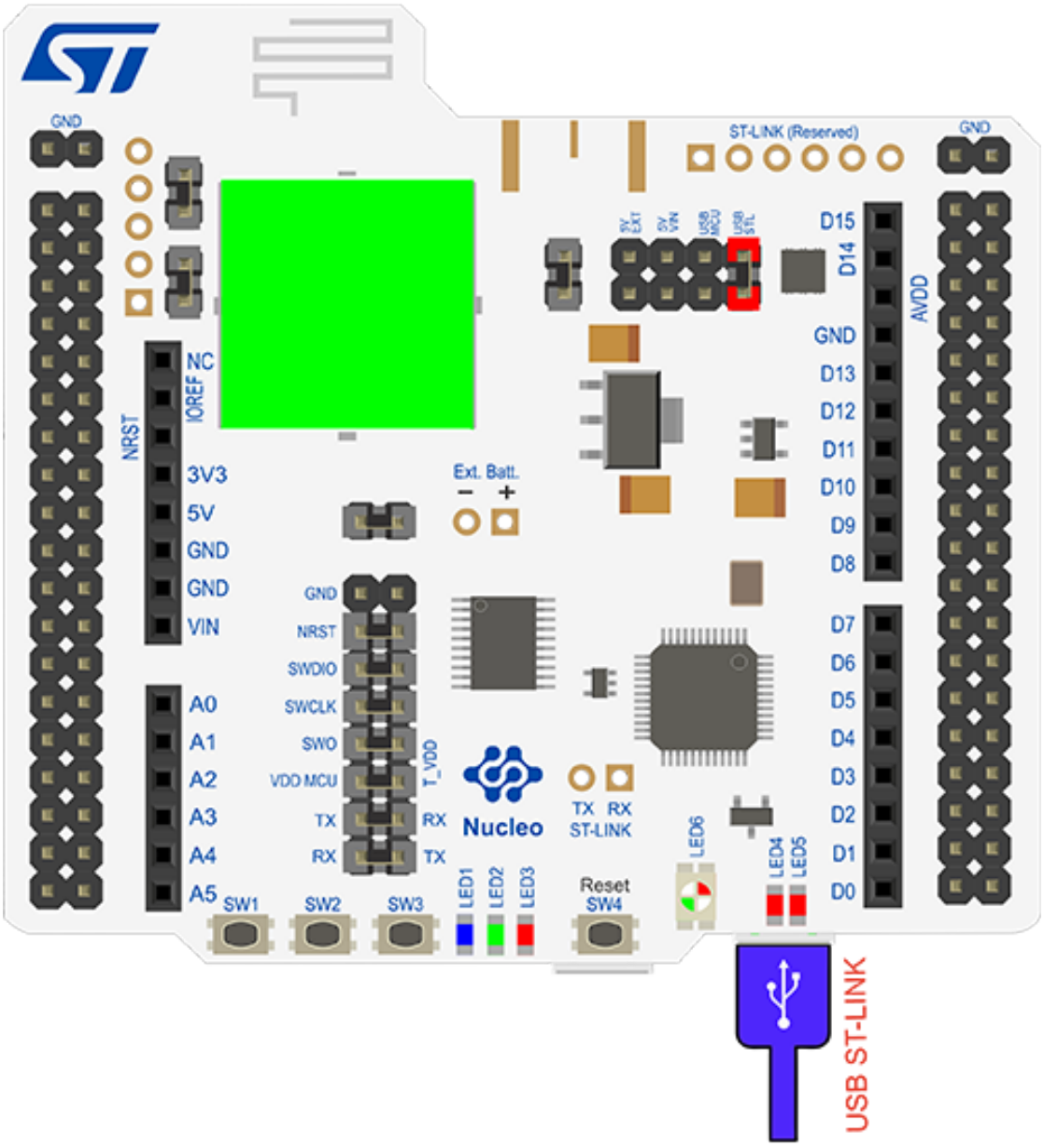


FIGURE 4.1 – Cablage de la carte NUCLEO pour la programmation

These modifications allow the STM32WB55 to be configured in ST Link programming mode.
We will be able to update the firmware (binary file) by **USB_ST_Link**.
Download and store the Firmware (.bin file) :

— via web site URL = https ://stm32python.gitlab.io/fr/docs/Micropython/Telechargement

— available on source codes directory : material/codes/firmware_micropython1.13-nucleo-wb55.bin

Connect the NUCLEO-WB55 board (USB STLINK) with the USB cable to your computer.

Open **NOD_WB55** mount point

Drag / drop the Firmware file (.bin file) to **NOD_WB55** mount point.
The firmware is now integrated (it is not visible in the NOD WB55 drive).
When LED6 has finished flashing (between red and green) and is green then the board is ready.

Place the jumper on **USB MCU** as well as the micro USB cable on the **USB_USER** connector under the RESET button image.

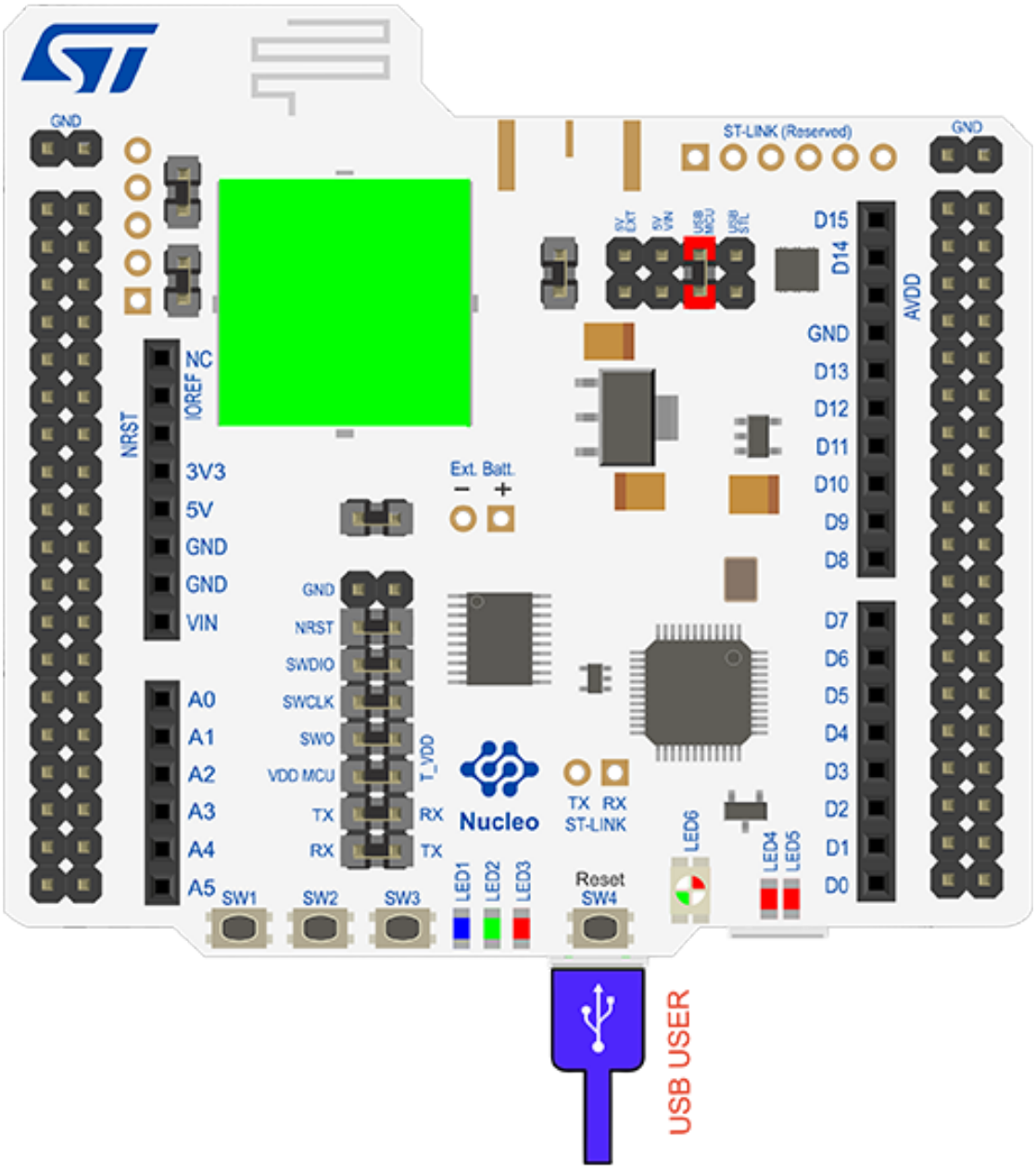


FIGURE 4.2 – Wiring the NUCLEO board for using MicroPython

Your Nucleo board is now able to interpret a .py file

Let's look at the files generated by the MicroPython system with Windows Explorer :

Open the PYBFLASH device with Windows Explorer.

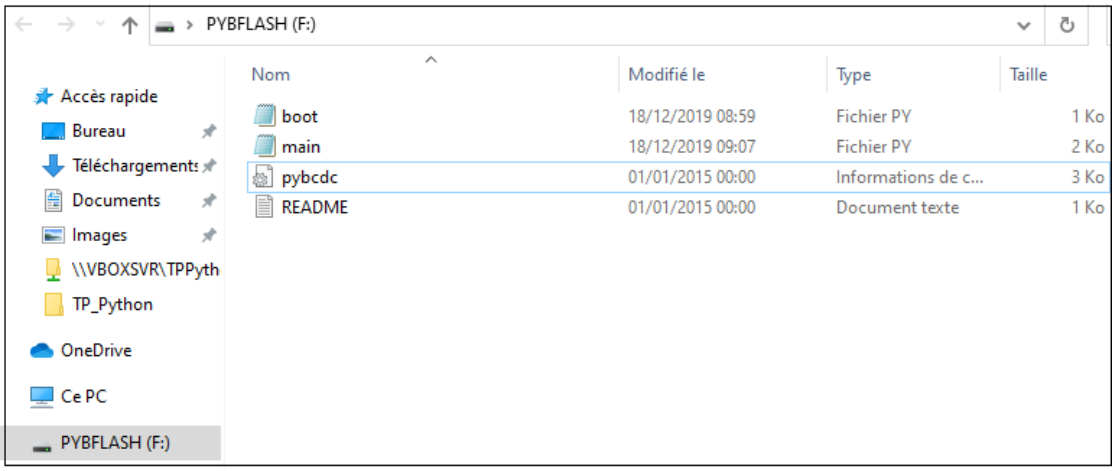


FIGURE 4.3 – Contents of PYBFLASH

We will see later how to edit the Python scripts available in the PYBFLASH filesystem.

Important notes : The boot.py script, can be modified by advanced users, it allows to initialize MicroPython and in particular to choose which script will be executed after the start. 'MicroPython boot, by default the main.py script .

4.1.2 Flashing USBDONGLE-WB55 board

This configuration can be summed up in :

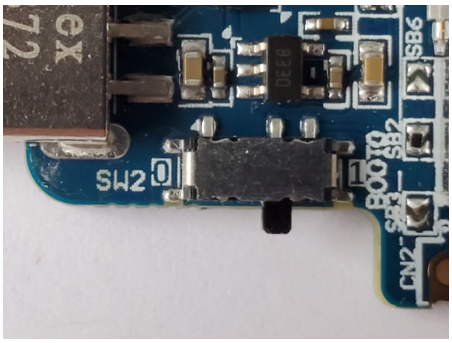
— Move and make sure the switch SW2 is in the position **1**

— Connect a USBDONGLE-WB55 board to PC

— Flash USBDONGLE-wb55 board with STM32CubeProgrammer

Here is how the kit should be configured :

For USB DONGLE WB55, please put the button SW2 on position "1" (second picture)



These modifications allow the USBDONGLE-WB55 to be configured in DFU mode.

We will be able to update the firmware (binary file) by **STM32CubeProgrammer**.

Download and store the Firmware (.hex file) : available on codes directory : material/codes/firmware_micropython1.14-usbdongle-wb55.bin

Open **STM32CubeProgrammer**

— 1. Put programmer mode on USB
— 2. Click on refresh button
— 3. Connect to the board
— 4. Open firmware file (hex file)
— 5. Download on board
— 6. Disconnect the board

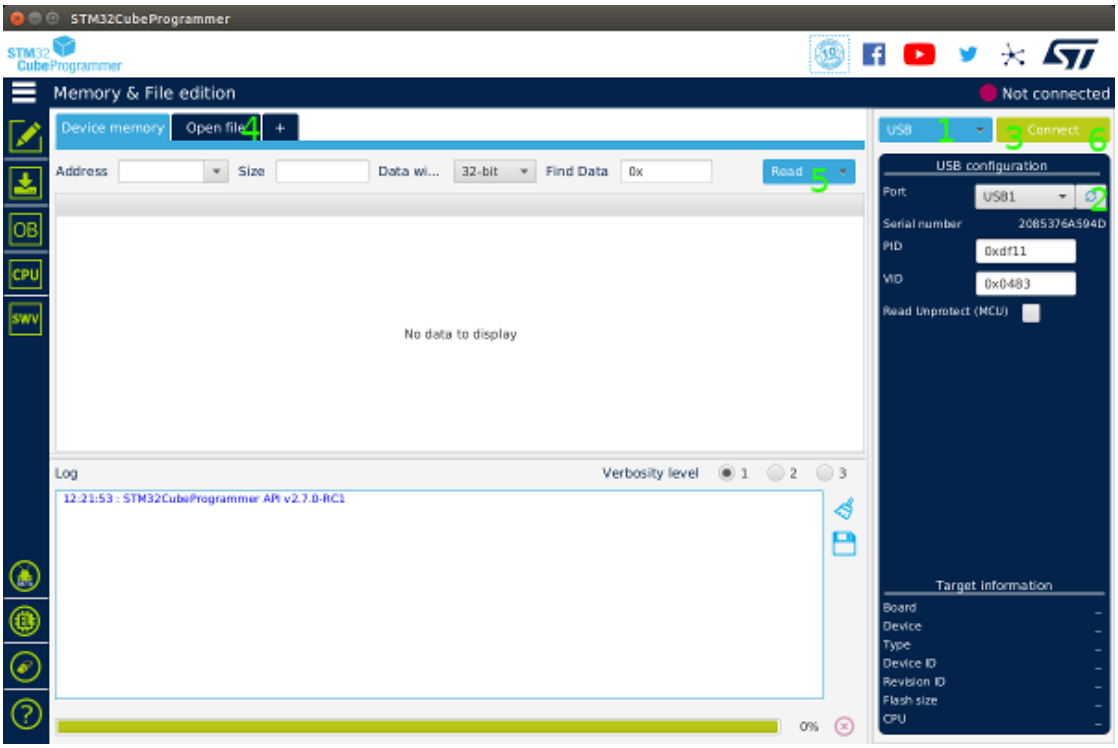


FIGURE 4.4 – STM32CubeProgrammer

Place the switch SW2 on **0**.

Your Nucleo board is now able to interpret a .py file

Let's look at the files generated by the MicroPython system with Windows Explorer :

Open the PYBFLASH device with Windows Explorer.

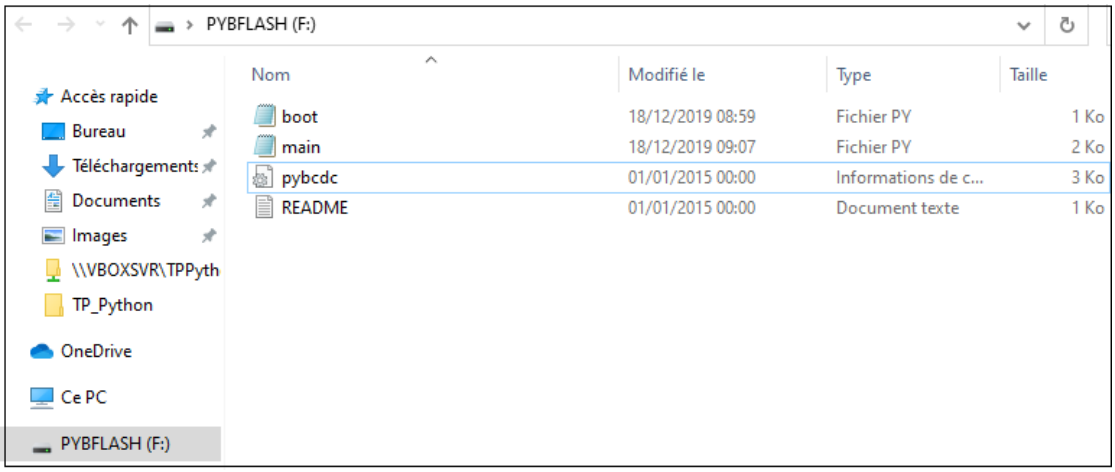


FIGURE 4.5 – Contents of PYBFLASH

We will see later how to edit the Python scripts available in the PYBFLASH filesystem.

Important notes : The boot.py script, can be modified by advanced users, it allows to initialize MicroPython and in particular to choose which script will be executed after the start. 'MicroPython boot, by default the main.py script .

4.1.3 Installation of the programming environment

Three solutions are available to you :
— Use the portable version or install (admin rights required) Notepad ++ with Python syntax highlighting and a serial terminal (PuTTY, TeraTerm. . .)
— Install (admin rights required) Thonny and its integrated terminal
— Use the portable version of PyScripter and a serial terminal (PuTTY, TeraTerm ...) .

4.1.4 Configuration of the communication terminal

Now that MicroPython is present on the NUCLEO-WB55 kit, we would like to test communication with Python. The test consists of sending a Python command and verifying that the execution takes place.

Communication is via USB through a serial port, so we need software that can send Python commands as text to the board and receive the r ' execution result.

Open TeraTerm after installing it (if you do not have admin rights on your PC, Puttytel is available in a portable version in the download area)

Select the Serial option and the COM port corresponding to your board (COMx : USB Serial Device) :

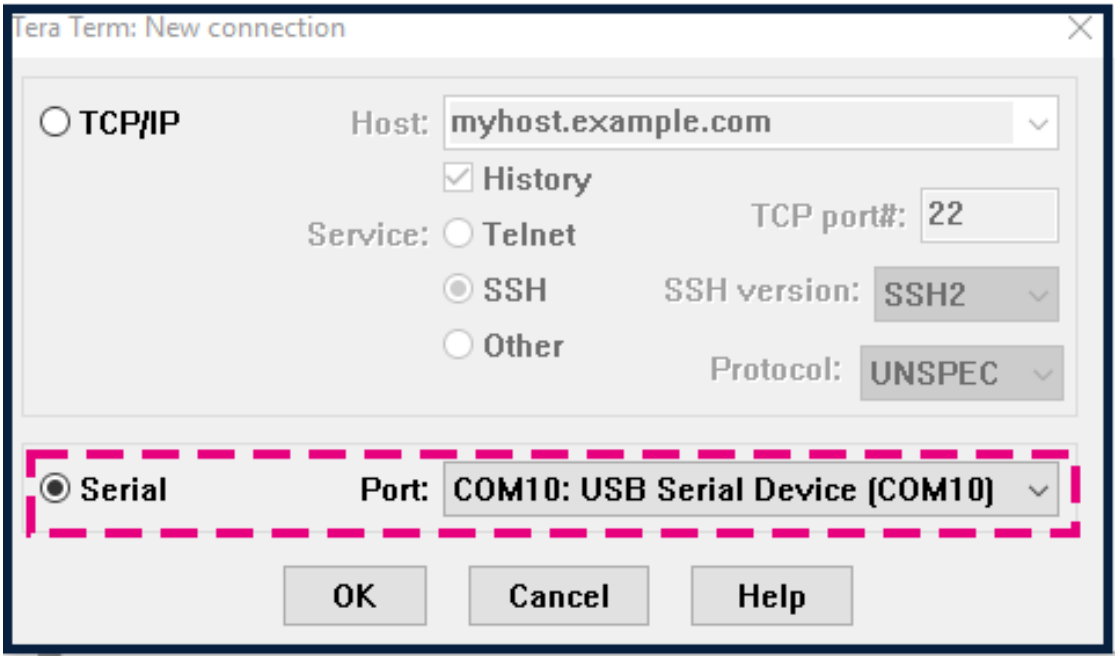


FIGURE 4.6 – TeraTerm

If the COM corresponding to the board is not available, check that the kit is listed on the correct COM port, see Checking the COM port allocation .

Configure the following fields in the Setup / Serial Port menu :

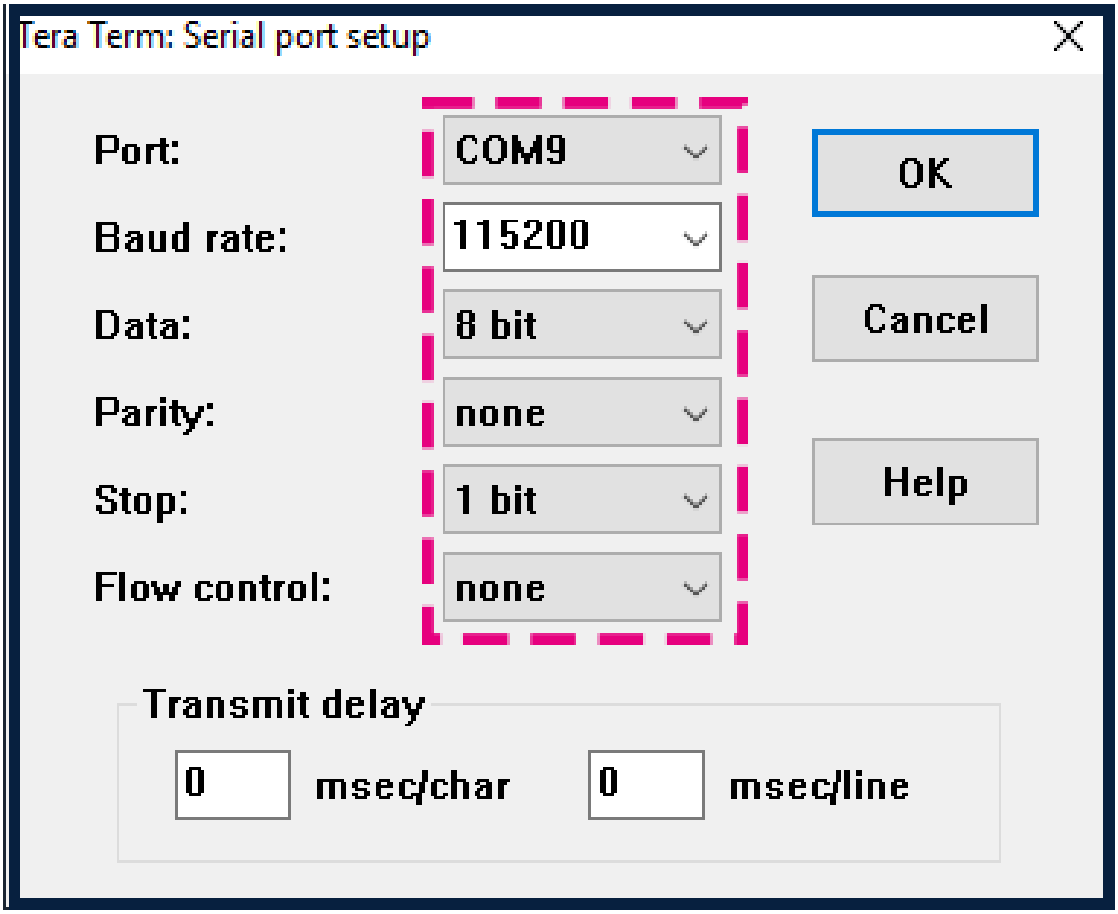


FIGURE 4.7 – TeraTerm configuration

A new window will appear :

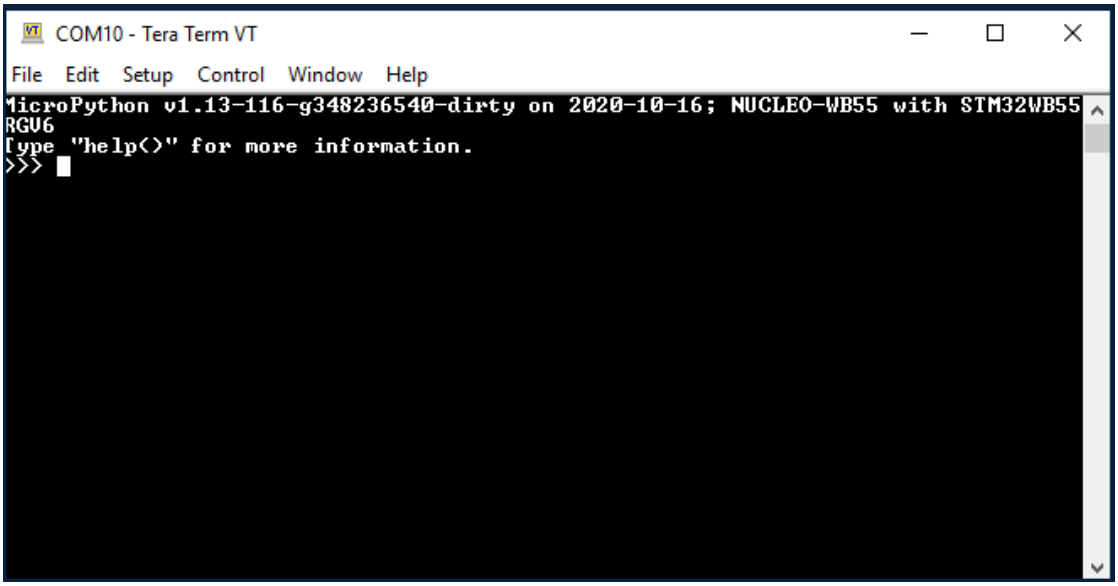


FIGURE 4.8 – TeraTerm console

If necessary, press CTRL + D to do a software reset.

You can now execute Python commands.

```
print("Hello World")
```

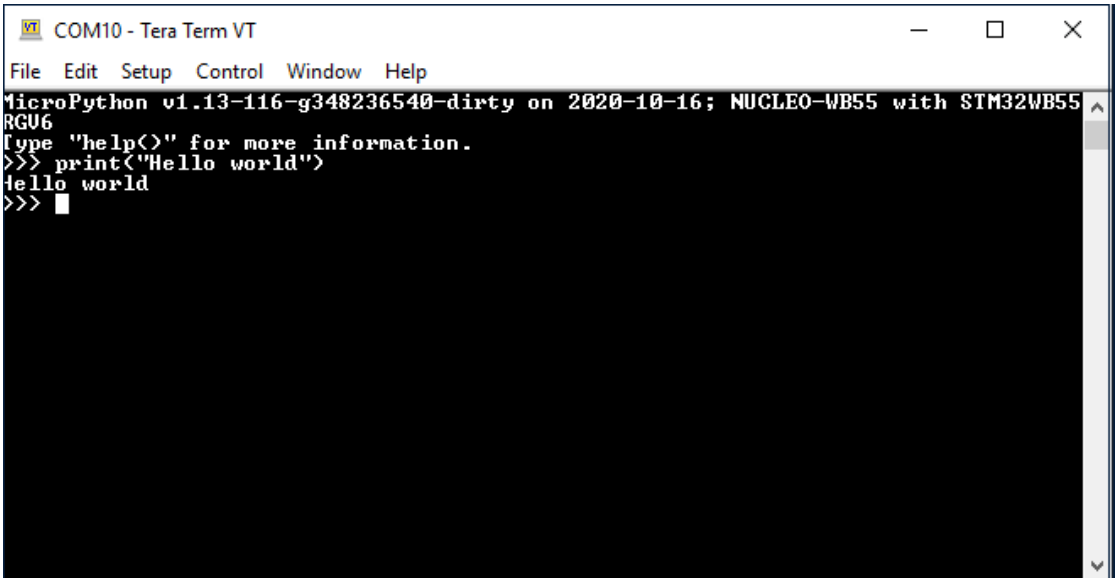


FIGURE 4.9 – TeraTerm console with Hello world

4.1.5 NotePad++ Settings

NotePad ++ defaults to configure to use spaces instead of tabs.
For coding in MicroPython, it is preferable to use a configuration using tabs, to avoid any Python syntax errors.
To do this, we will activate the display of all the symbols :

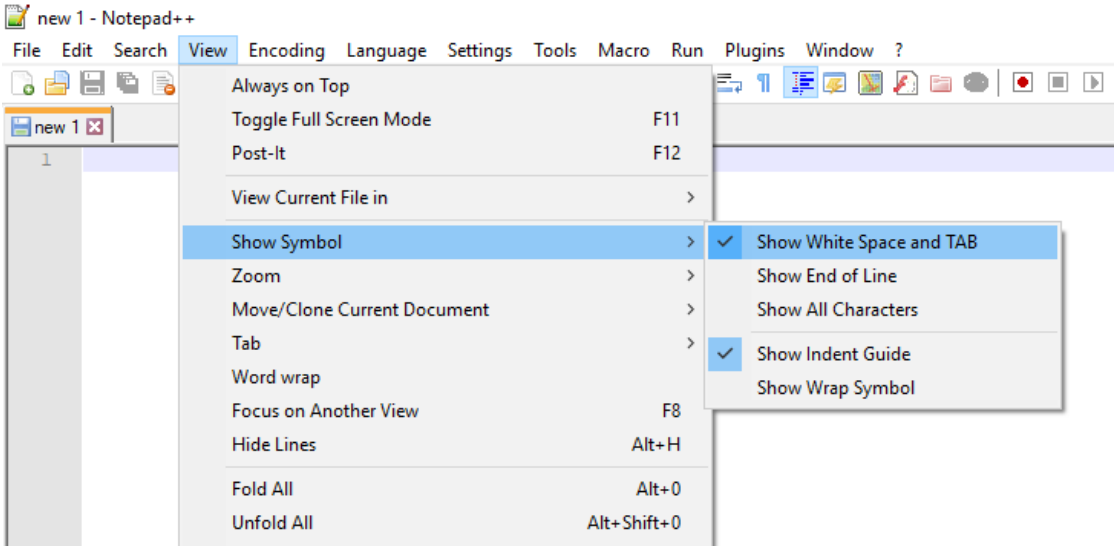


FIGURE 4.10 – Show symbols on NotePad++

And we will ask not to have a Tabulation in Space conversion :

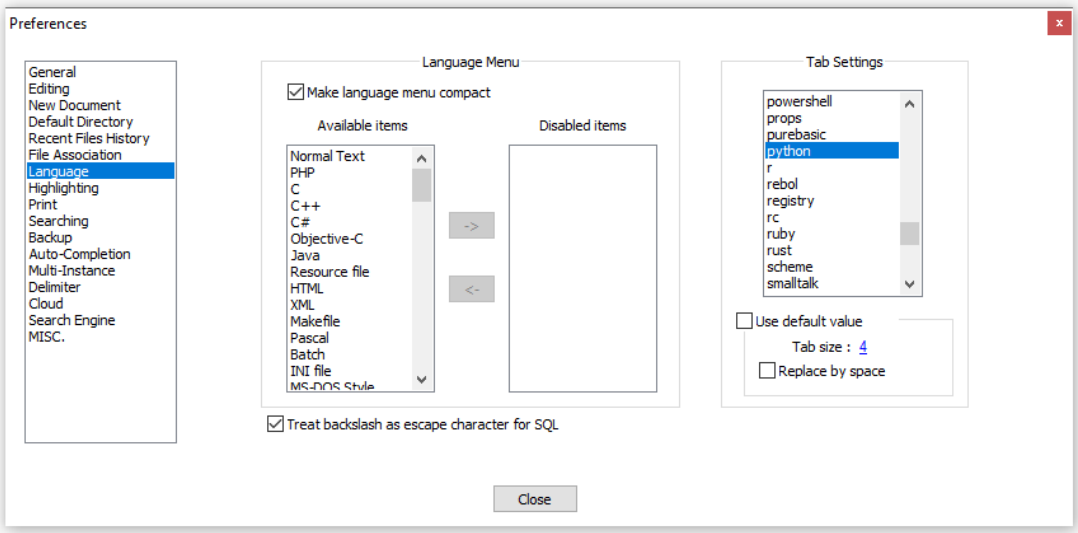


FIGURE 4.11 – Tab configuration

4.2 Linux Getting Started Guide

In this getting started guide we will :
— Program the board with the firmware.
— Use a UART communication terminal (minicom).
— Program our first Python script on the Nucleo board.

4.2.1 Flashing NUCLEO-WB55 board

You will need a USB to micro USB cable.
This configuration can be summed up in :
— Move and make sure the jumper is in the position **USB_ST_LINK**
— Connect a micro USB cable to the ST LINK port below the LED4 and LED5
— Flash board
Here is how the kit should be configured :

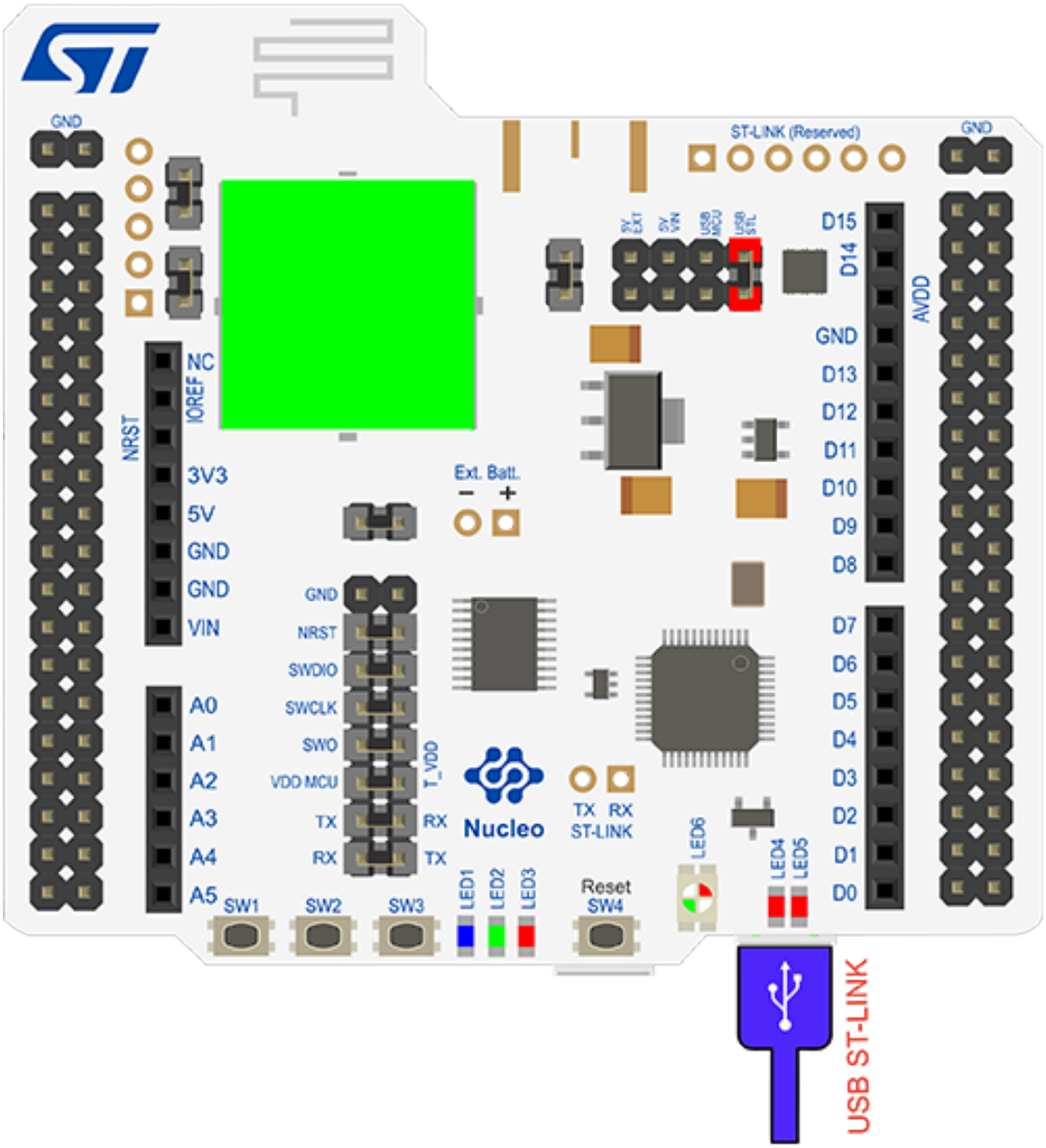


FIGURE 4.12 – Cablage de la carte NUCLEO pour la programmation

These modifications allow the STM32WB55 to be configured in ST Link programming mode.
We will be able to update the firmware (binary file) by **USB_ST_Link**.
Download and store the Firmware (.bin file) :

— via web site URL = https ://stm32python.gitlab.io/fr/docs/Micropython/Telechargement

— on source codes directory : material/codes/firmware_micropython1.13-nucleo-wb55.bin

Connect the NUCLEO-WB55 board (USB STLINK) with the USB cable to your computer.

Open **NOD_WB55** mount point

Drag / drop the Firmware file (.bin file) to **NOD_WB55** mount point.
The firmware is now integrated (it is not visible in the NOD WB55 drive).
When LED6 has finished flashing (between red and green) and is green then the board is ready.

Place the jumper on **USB MCU** as well as the micro USB cable on the **USB_USER** connector under the RESET button image.

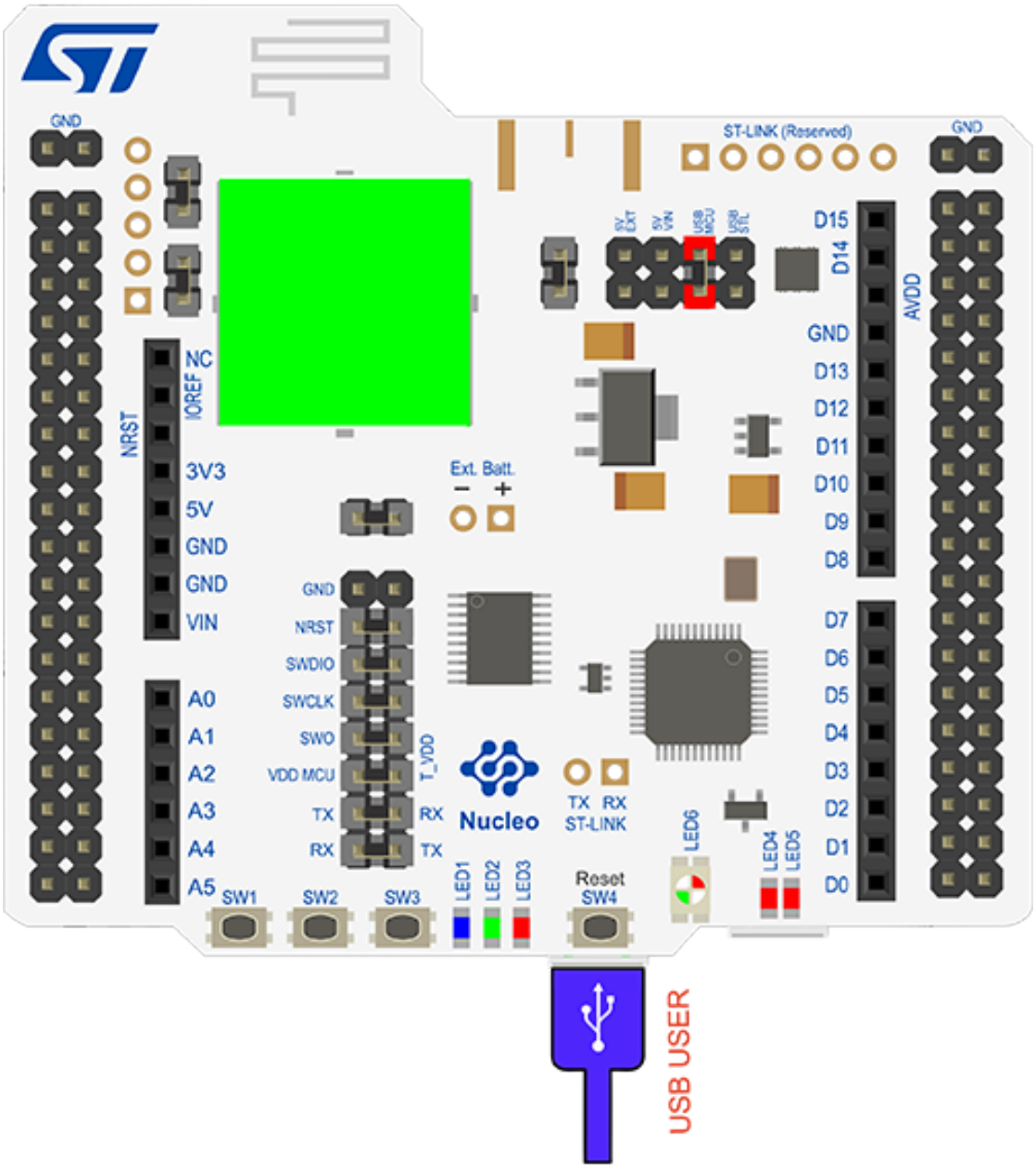


FIGURE 4.13 – Wiring the NUCLEO board for using MicroPython

Your Nucleo board is now able to interpret a .py file

4.2.2 Flashing USBDONGLE-WB55 board

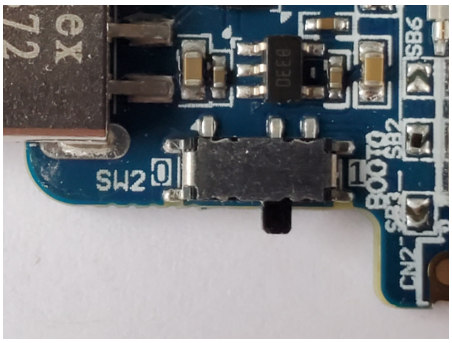
This configuration can be summed up in :

— Move and make sure the switch SW2 is in the position **1**

— Connect a USBDONGLE-WB55 board to PC

— Flashing board with STM32CubeProgrammer

Here is how the kit should be configured : For USB DONGLE WB55, please put the button SW2 on pistion "1" (second picture)



These modifications allow the USBDONGLE-WB55 to be configured in DFU mode.
We will be able to update the firmware (binary file) by **STM32CubeProgrammer**.
Download and store the Firmware (.hex file) : on codes directory : material/codes/firmware_micropython1.14-usbdongle-wb55.bin

Open **STM32CubeProgrammer**

— 1. Put programmer mode on USB
— 2. Click on refresh button
— 3. Connect to the board
— 4. Open firmware file (hex file)
— 5. Download on board
— 6. Disconnect the board

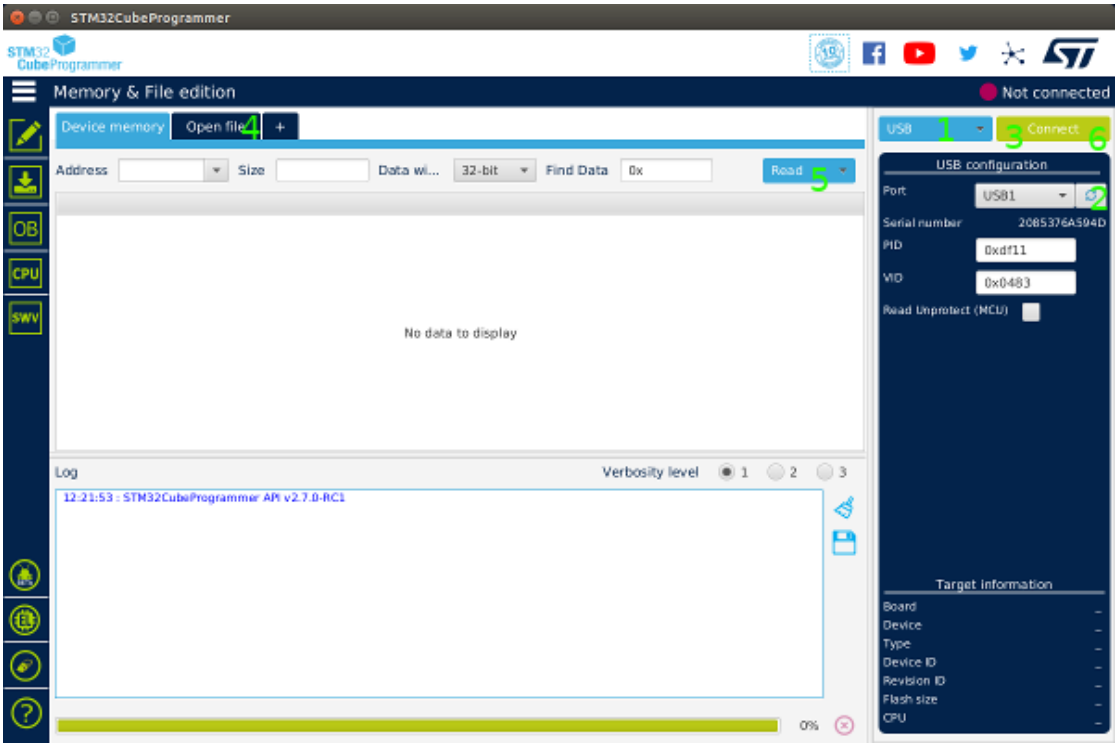


FIGURE 4.14 – STM32CubeProgrammer

Place the switch SW2 on **0**.

Your Nucleo board is now able to interpret a .py file

PYBFLASH content

Let's look at the files generated by the MicroPython system with a linux shell :

```
$> ls /media/$USER/PYBFLASH/
boot.py  main.py  pybcdc.inf  README.txt
```

We will see later how to edit the Python scripts available in the PYBFLASH filesystem.

Important notes : The boot.py script, can be modified by advanced users, it allows to initialize MicroPython and in particular to choose which script will be executed after the start. 'MicroPython boot, by default the main.py script.

4.2.3 Configuration of console terminal

Now that MicroPython is present on the NUCLEO-WB55 kit, we would like to test communication with Python. The test consists of sending a Python command and verifying that the execution takes place.

Communication is via USB through a serial port, so we need software that can send Python commands as text to the board and receive the r ' execution result.

Create a configuration file for minicom " to use with the serial port of the NUCLEO-WB55 board (normally / dev / ttyACM0) :

```
$> echo "pu port             /dev/ttyACM0" > $HOME/.minirc.dfl
$> echo "pu rtscts           No" > $HOME/.minirc.dfl
```

Open minicom on the serial port associated with the NUCLEO-WB55 board (normally / dev / ttyACM0) :

```
$> minicom -D /dev/ttyACM0
Bienvenue avec minicom 2.7.1

OPTIONS: I18n
Compilé le Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 17:35:15

Tapez CTRL-A Z pour voir l'aide concernant les touches spéciales

MicroPython v1.13-186-g5a7027915 on 2020-11-27; NUCLEO-WB55 with
↪  STM32WB55RGV6
Type "help()" for more information.
>>>
```

To Exit minicom, CTRL + A and X

You can now execute Python commands in minicom.

```
print("Hello World")
```

```
$> minicom -D /dev/ttyACM0
Bienvenue avec minicom 2.7.1

OPTIONS: I18n
Compilé le Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 17:35:15
```

```
Tapez CTRL-A Z pour voir l'aide concernant les touches spéciales

MicroPython v1.13-186-g5a7027915 on 2020-11-27; NUCLEO-WB55 with
↪  STM32WB55RGV6
Type "help()" for more information.
>>> print("Hello World")
Hello World
>>>
```

Chapitre 5

Program

5.1 First program via console

Type in the console (serial port of the NUCLEO-WB55 card) :

```
print("Hello World")
```

MicropPython will send on console : *Hello World*

Note

When you push the RESET button or you unplug/plug the usb cable, the program which print "Hello World" are not saved.

5.2 First program via main.py

Edit the file **main.py** which are located on **PYBFLASH** via an editor and copy/paste the following code :

```
if __name__ == '__main__':
  print("Hello World")
```

Warning

Python asks to have a tab as indentation for this block of code, here put a tab before print.

Warning

You must unmount the **PYBFLASH** disk before restarting the board, otherwise the files on the **PYBFLASH** disk will be corrupted .

noindent Restart the NUCLEO WB55 card by pressing the RESET or CTRL + D button in the console.
noindent When restarting the card, in the terminal, you should see "Hello Word".

5.3 Print several time a text

Edit the file **main.py** which is on the disk **PYBFLASH** via Notepad or another text editor and copy the following text :

```
if __name__ == '__main__':
        for i in range(1, 11):
                print("Hello World, MicroPython are awesome, ", i,
                 ↪  "times")
```

5.4 Blink LEDS

The goal now is to blink LEDs on the development kit :

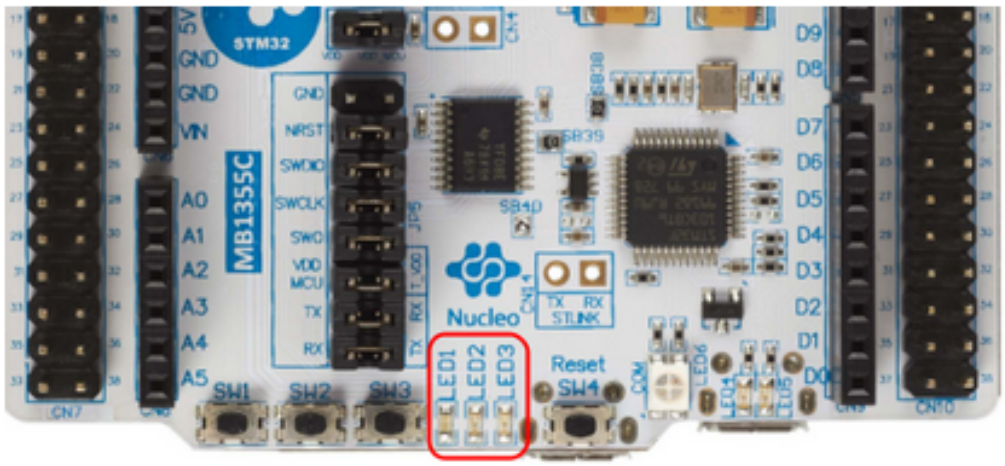


FIGURE 5.1 – position of the LEDs on the NUCLEO WB55 board

Here is the organization of LEDs by color and number :
— LED 1 : Red
— LED 2 : Green
— LED 3 : Blue
Under MicroPython, the **pyb.LED** module allows you to manage LEDs very simply.

In this part, we want to carry out a “chaser”, this exercise consists of turning the LEDs on and off, one after the other, in a cyclical fashion.
Using the previous method and the following script, perform the chase with MicroPython (main.py) :

Listing 1 – Leds.

```
import pyb
import time

```

```
print("Leds with MicroPython it's easy")
# Init of leds(LED_1, LED_2, LED_3)

led_red = pyb.LED(1)
led_green = pyb.LED(2)
led_blue = pyb.LED(3)

# Init of  du counter of LED
counter_led = 0

while 1: # Création d'une boucle infinie
    if counter_led == 0:
        led_rouge.on()
        led_green.off()
        led_blue.off()
    elif counter_led == 1:
        led_red.off()
        led_green.on()
        led_blue.off()
    else:
        led_red.off()
        led_green.off()
        led_blue.on()
    # we sould like to turn on next led at next iteration
    counter_led = counter_led + 1
    if counter_led > 2:
        counter_led = 0
    time.sleep_ms(500) # Système sleep during 500ms
```

(Files available on material/codes/examples/leds.py)

5.5 Button

The goal now is to get the state of Button :

Listing 2 – Buttons.

```
import pyb
import time

print("GPIO with MicroPython it's easy")

# Init entries Pin(SW1, SW2, SW3)
sw1 = pyb.Pin('SW1', pyb.Pin.IN)
sw1.init(pyb.Pin.IN, pyb.Pin.PULL_UP, af=-1)

sw2 = pyb.Pin('SW2', pyb.Pin.IN)
sw2.init(pyb.Pin.IN, pyb.Pin.PULL_UP, af=-1)
```

```
12
13 sw3 = pyb.Pin('SW3', pyb.Pin.IN)
14 sw3.init(pyb.Pin.IN, pyb.Pin.PULL_UP, af=-1)
15
16 # Init variables
17 old_state_sw1 = 0
18 old_state_sw2 = 0
19 old_state_sw3 = 0
20
21 while 1:
22     # System sleep during 300ms
23     time.sleep_ms(300)
24
25     # get state of button
26     state_sw1 = sw1.value()
27     state_sw2 = sw2.value()
28     state_sw3 = sw3.value()
29
30     #print("DEBUG STATE: ", [state_sw1, state_sw2, state_sw3])
31
32     if state_sw1 != old_state_sw1:
33         if state_sw1 == 0:
34             print("Buton 1 (SW1) is pressed")
35         else:
36             print("Buton 1 (SW1) is released")
37         old_state_sw1 = state_sw1
38     if state_sw2 != old_state_sw2:
39         if state_sw2 == 0:
40             print("Buton 2 (SW2) is pressed")
41         else:
42             print("Buton 2 (SW2) is released")
43         old_state_sw2 = state_sw2
44     if state_sw3 != old_state_sw3:
45         if state_sw3 == 0:
46             print("Buton 3 (SW3) is pressed")
47         else:
48             print("Buton 3 (SW3) is released")
49         old_state_sw3 = state_sw3
```

5.6 SSD1306 OLED SCREEN

Note

For wiring see section material.

Note

You need to copy the library material/codes/Display/ssd1306/ssd1306.py on mount point PYBFLASH.

Display text on several lines :

Listing 3 – SSD1306.

```
from machine import Pin, I2C
import ssd1306
from time import sleep

#Init I2C1
i2c = I2C(1)

# Screen settings
width_oled_screen = 128
height_oled_screen = 64

oled = ssd1306.SSD1306_I2C(width_oled_screen, height_oled_screen,
 ↪  i2c)

#Envoi du texte à afficher sur l'écran OLED
oled.text("MicroPython OLED!", 0, 0)
oled.text("    I2C    ", 0, 10)
oled.text("It's too easy !!!", 0, 20)
oled.show()
```

Note

For displaying pictures on board, you need to copy material/codes/examples/logo_st.pbm, material/codes/examples/climate.pbm and material/codes/examples/pressure.pbm

Display Text and pictures :

Listing 4 – SSD1306 and Pictures.

```
from machine import I2C, Pin
# display SSD1306 is connected on I2C1 of arduino connector
import framebuf
import ssd1306

# ----------------------------------------
# screen parameters
SCREEN_WIDTH = 128
SCREEN_HEIGHT = 64

# ----------------------------------------
```

```
# READ Picture from file
def load_image(filename):
    with open(filename, 'rb') as f:
        f.readline()
        width, height = [int(v) for v in f.readline().split()]
        data = bytearray(f.read())
    return framebuf.FrameBuffer(data, width, height,
    ↪ framebuf.MONO_HLSB)


# ----------------------------------------
# search if i2c id are present on i2c bus
def isPresentOnI2C(i2c, i2c2_list, id_tofound):
    if i2c2_list is None:
        local_i2clist = self.i2c.scan()
    else:
        local_i2clist = i2c2_list
    for i in local_i2clist:
        if i == id_tofound:
            return 1
    return 0

# ----------------------------------------
# display ST logo
def display_logo(screen):
    if screen is not None:
        screen.fill(0)

        logo_pbm = load_image("logo_st.pbm")
        screen.blit(logo_pbm, 0, 0)

        screen.show()
# ----------------------------------------
# display temperature and humidity
def display_temperature_humidity(screen, temp, hum):
    if screen is not None:
        screen.fill(0)

        temperature_pbm = load_image("climate.pbm")
        screen.blit(temperature_pbm, 0, 16)

        screen.text('{:14s}'.format('Temperature:'), 32, 0)
        screen.text('{:^14s}'.format(str(temp) + 'C'), 16, 16)

        screen.text('{:14s}'.format('Humidity:'), 32, 32)
        screen.text('{:^14s}'.format(str(hum) + '%'), 16, 48)

        screen.show()
```

```
# ----------------------------------------
# display pressure and altitude
def display_pressure_altitude(screen, press, alt):
    if screen is not None:
        screen.fill(0)

        pressure_pbm = load_image("pressure.pbm")
        screen.blit(pressure_pbm, 0, 16)

        screen.text('{:18s}'.format('Pressure:'), 36, 0)
        screen.text('{:^14s}'.format('{:.1f}'.format(press) +
        ↪  'hPa'), 18, 16)

        screen.text('{:18s}'.format('Altitude:'), 36, 32)
        screen.text('{:^14s}'.format('{:.1f}'.format(alt) + 'M'),
        ↪  18, 48)

        screen.show()

# ----------------------------------------
# display temperature and humidity

# ----------------------------------------------
#                  main
# ----------------------------------------------
i2c1 = I2C(1)
display = None
if __name__ == '__main__':
    # scan i2c bus to see which ip components are present
    print(i2c1.scan())
    # wait to be sure all the component are present
    time.sleep_ms(1000)
    # init display
    if isPresentOnI2C(i2c1, None, 60):
        display = ssd1306.SSD1306_I2C(SCREEN_WIDTH, SCREEN_HEIGHT,
        ↪  i2c1)

    # display log during 5 seconds
    display_logo(display)
    time.sleep_ms(5000)

    # display temperature
    display_temperature_humidity(display, 25.2, 45.6)
    time.sleep_ms(5000)

    # display pressure
    display_pressure_altitude(display, 1024.2, 122.1)
    time.sleep_ms(5000)
```

5.7 LED Matrices

Note

For wiring see section material.

Note

You need to copy the library material/codes/Display/max7219/max7219.py on mount point PYBFLASH.

Display text :

```
import max7219
from machine import Pin, SPI
spi = SPI(1)

display = max7219.Max7219(32, 8, spi, Pin('D10'))
display.text('1234',0,0,1)
display.show()
```

Display scroll text :

Listing 5 – LED Matric, scroll text

```
import max7219
from machine import Pin, SPI
import time


def display_text_scroll(text):
    # for 4 led blocks
    for p in range(4 * 8, len(text) * -8 - 1, -1):
        display.fill(False)
        display.text(text, p, 0, not False)
        display.show()
        time.sleep_ms(50)


spi = SPI(1)

time.sleep_ms(1000)

display = max7219.Max7219(32,8, spi, Pin('D10'))

display_text_scroll("STMicroelectronics")
```

5.8 IKS01A3

5.8.1 HTS221 : Temperature, humidity

HTS221 is a temperature sensor.
Function available :
— **temperature()** : get temperature ;
— **humidity()** : get humidity ;
— **get()** : return an array with [temperature, humidity] ;

Note

You need to copy file hts221.py on mount point PYBFLASH. (material/codes/Capteurs/HTS221/hts221.py)

```
from machine import I2C
import hts221
import time

i2c = I2C(1)
time.sleep_ms(1000) # wait sensor available on bus
sensor = hts221.HTS221(i2c)

print("Temperature: %.2f %s" % (sensor.temperature(), "°C"))  #
 ↪  result on °C
print("Humidity: %.2f %c" % (sensor.humidity(), "%")) # résult on %
```

5.8.2 LPS22HH : Temperature, Pressure

LPS22HH est un Barometer (260-1260 hPa) and temperature sensor.
Function available :
— **temperature()** : get temperature ;
— **pressure()** : get atmospheric pressure ;
— **get()** : return array with [temperature, pression] ;
— **altitude()** : get altitude ;

Note

You need to copy file LPS22.py on mount point PYBFLASH. (material/codes/Capteurs/LPS22/LPS22.py)

```
from machine import I2C
import LPS22
import time

i2c = I2C(1)
time.sleep_ms(1000) # wait sensor available on bus
sensor = LPS22.LPS22(i2c)
```

```
print("Temperature: %.2f %s" % (sensor.temperature(), "°C"))  #
↪  result on °C
print("Pressure: %.2f %s" % (sensor.pressure(), "hPa")) # résult on
↪  Hpa
print("Altitude: %.2f %s" % (sensor.altitude(), "M")) # résult on M
```

5.8.3 LSM6DSO : Accelerometer, gyroscope

LSM6DSO is an 3D accelerometer and Gyroscope sensor.
Function available :

— **ax_raw()** : get accelerometer on X axe value on raw ;
— **ay_raw()** : get accelerometer on Y axe value on raw ;
— **az_raw()** : get accelerometer on Z axe value on raw ;
— **gx_raw()** : get gyrsocope on X axe value on raw ;
— **gy_raw()** : get gyrsocope on Y axe value on raw ;
— **gz_raw()** : get gyrsocope on Z axe value on raw ;
— **ax()** : get accelerometer on X axe on mg (g intensity of gravity) ;
— **ay()** : get accelerometer on Y axe on mg (g intensity of gravity) ;
— **az()** : get accelerometer on Z axe on mg (g intensity of gravity) ;
— **gx()** : get gyroscope on X axe on rad/s (radian / seconds) ;
— **gy()** : get gyroscope on Y axe on rad/s (radian / seconds) ;
— **gz()** : get gyroscope on Z axe on rad/s (radian / seconds) ;
— **get_a_raw()** : get accelerometer raw value on array [X, Y, Z] ;
— **get_a()** : get accelerometer value on array [X, Y, Z] on mg (g intensity of gravity) ;
— **get_g_raw()** : get gyroscope raw value on array [X, Y, Z] ;
— **get_g()** : get gyroscope raw value on array [X, Y, Z] on rad/s (radian / seconds) ;
— **temperature()** : get temperature

Note

You need to copy file LSM6DSO.py on mount point PYBFLASH. (material/codes/Capteurs/LSM6DSO/LSM6DSO.py)

```
from machine import I2C
import LSM6DSO
import time

i2c = I2C(1)
time.sleep_ms(1000) # wait sensor available on bus
inertial_sensor = LSM6DSO.LSM6DSO(i2c)


print("Accelerometer: X ", intertial_sensor.ax()) # result on mg
print("Accelerometer: Y ", intertial_sensor.ay())
print("Accelerometer: Z ", intertial_sensor.az())
```

```
print("Gyroscope: X ", intertial_sensor.gx()) # result on rad/s
print("Gyroscope: X ", intertial_sensor.gy())
print("Gyroscope: X ", intertial_sensor.gz())

print("Accelerometer: ", intertial_sensor.get_a()) # result on mg
print("Gyroscope:     ", intertial_sensor.get_g()) # result on rad/s
```

Algorithme IMU (Inertial Measurement Unit)

pitch = round(math.atan(x / math.sqrt(y * y + z * z)) * 180.0 * math.pi)
roll = round(math.atan(y / math.sqrt(x * x + z * z)) * 180.0 * math.pi)
yaw = round(math.atan(z / math.sqrt(x * x + z * z)) * 180.0 * math.pi)

if abs(pitch) > 200 then
    if pitch is positif then the board is oriented to left
    else board is oriented to rigth
else if abs(roll) > 200 then
    if roll is positif then board is orientd to up
    else board is oriented to bottom
else
    board doesn't moveé

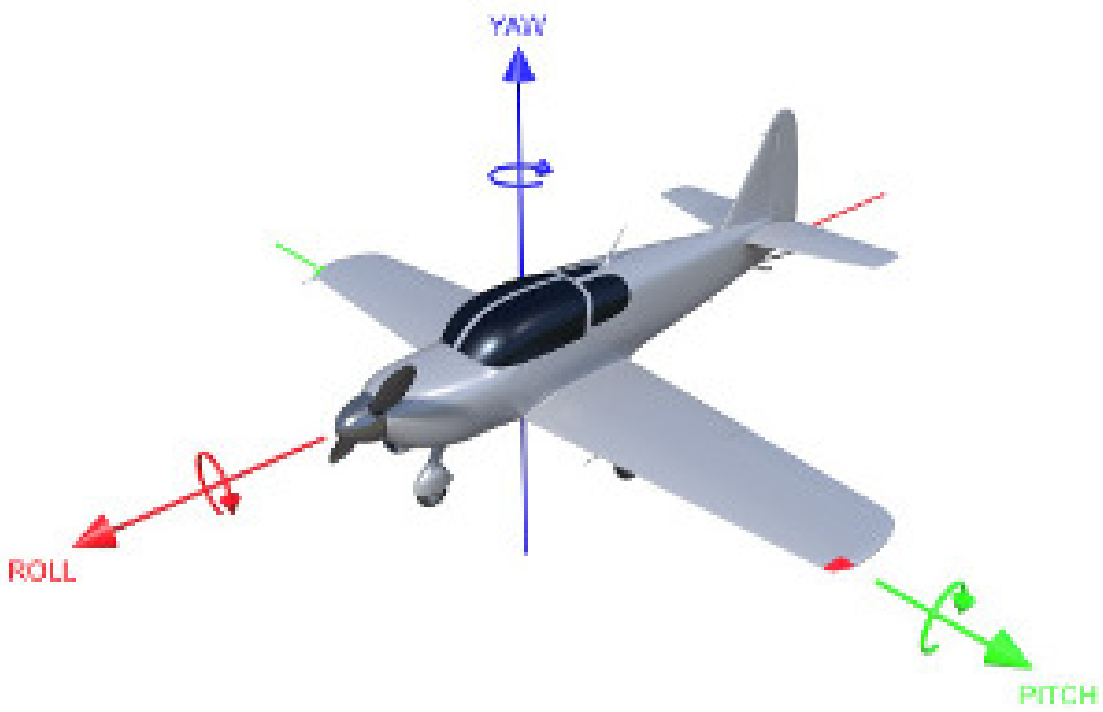


FIGURE 5.2 – Pitch, Roll, Yaw

Listing 6 – Orientation via IMU Algo.

```
from machine import I2C
import LSM6DSO
import math
import time

i2c = I2C(1)
time.sleep_ms(1000) # wait sensor available on bus
inertial_sensor = LSM6DSO.LSM6DSO(i2c)

while 1:
    accel = inertial_sensor.get_a_raw()
    x = accel[0]
    y = accel[1]
    z = accel[2]

    if x == 0 and y == 0 and z == 0:
        pitch = 0
        roll = 0
        yaw = 0
    else:
        # Calcul IMU
        pitch = round(math.atan(x / math.sqrt(y * y + z * z)) *
        ↪  180.0 * math.pi)
        roll  = round(math.atan(y / math.sqrt(x * x + z * z)) *
        ↪  180.0 * math.pi)
        yaw   = round(math.atan(z / math.sqrt(x * x + z * z)) *
        ↪  180.0 * math.pi)
        #print("[DEBUG]: Pitch: ", pitch)
        #print("[DEBUG]: Roll: ", roll)
        #print("[debug]: yaw: ", yaw)

        # display orientation
        if abs(pitch) > 200:
            ''' (pitch > 0) ? ORIENTATION_LEFT_UP :
            ↪  ORIENTATION_RIGHT_UP;'''
            if pitch > 0:
                print("Left")
            else:
                print("Right")
        elif abs(roll) > 200:
            '''ret = (roll < 0) ? ORIENTATION_VERTICAL_DOWN :
            ↪  ORIENTATION_VERTICAL_UP;'''
            if roll < 0:
                print("Bottom")
            else:
```

```
                print("UP")
        else:
            print("Not moving")
        time.sleep_ms(500)
```

(Fichier disponible dans material/codes/examples/meteo_orientation_algo.py)

5.9 WS2812b : led strip

The goal is to display french and italian flag on a LED STRIP wi ws2812b :
NUCLEO-WB55 :

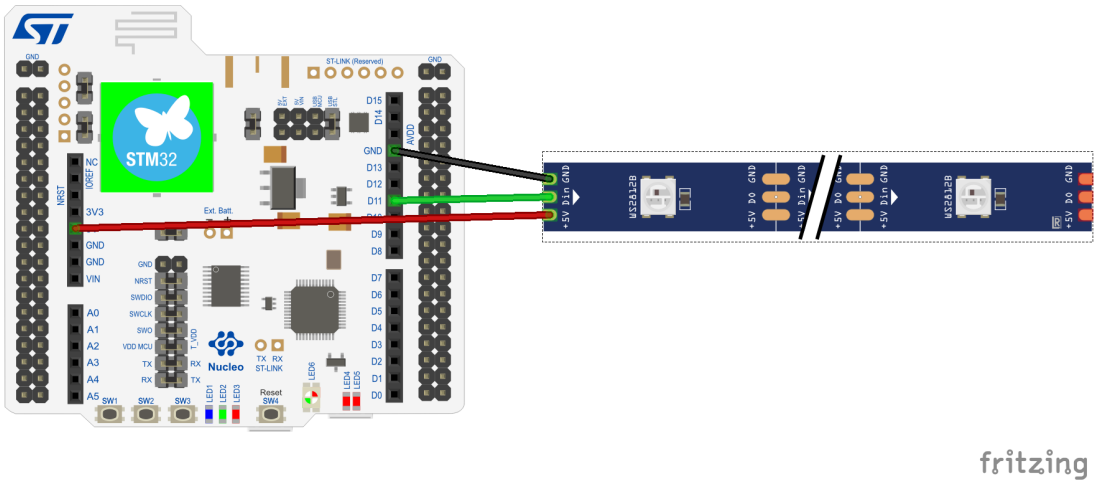


FIGURE 5.3 – Wiring with NUCLEO-WB55

USBDONGLE-WB55 :

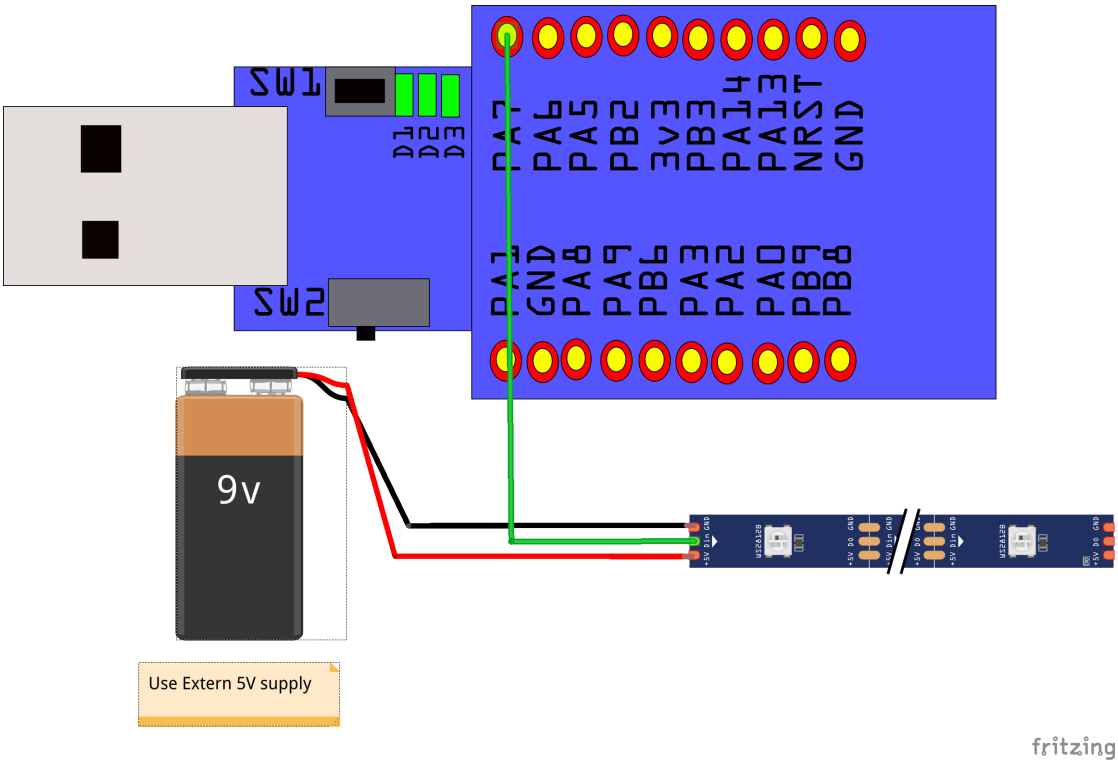


FIGURE 5.4 – Wiring with NUCLEO-WB55

Note

Put on the board the library ws2812bstm32 localted on material/codes/Display/ws2812b/ws2812bstm32.py

Listing 7 – SSD1306.

```
import math
import time
import ws2812bstm32

# --------------------------------
french_flag = [
    (0,0,250), (0,0,250), (0,0,250),
    (250,250,250), (250,250,250), (250,250,250),
    (250,0,0), (250,0,0), (250,0,0)]
italien_flag = [
    (0,250,0), (0,250,0), (0,250,0),
    (250,250,250), (250,250,250), (250,250,250),
    (250,0,0), (250,0,0), (250,0,0)]

# --------------------------------
def test_led(leds, led_count, color):
    for i in range(0, led_count):
        if i != 0:
            leds.clean(i-1)
        leds.put_pixel(i, color[0], color[1], color[2])
        leds.send_buf()
        time.sleep_ms(500)
    leds.clean(i)
    leds.send_buf()
    time.sleep_ms(500)
# --------------------------------
def flag(leds, data,display=True):
    i = 0
    for d in data:
        leds.put_pixel(i, d[0], d[1], d[2])
        if display:
            leds.send_buf()
            time.sleep_ms(1000)
        i+=1
def invflag(leds, data, display):
    i = len(data) - 1
    while i >= 0:
        d = data[i]
        leds.put_pixel(i, d[0], d[1], d[2])
        if display:
            leds.send_buf()
            time.sleep_ms(1000)
```

```
        i-=1
# --------------------------------
def dual_flag(leds, data0, data1, num_led, t, display):
    i = 0
    while i < num_led:
        if (i + len(data0)) < num_led:
            for d in data0:
                leds.put_pixel(i, d[0], d[1], d[2])
                if display:
                    leds.send_buf()
                    time.sleep_ms(t)
                i+=1
            if not display:
                leds.send_buf()
            i+=1
        else:
            break
        if (i + len(data1)) < num_led:
            for d in data1:
                leds.put_pixel(i, d[0], d[1], d[2])
                if display:
                    leds.send_buf()
                    time.sleep_ms(t)
                i+=1
            if not display:
                leds.send_buf()
            i+=1
        else:
            break
# --------------------------------
def flag_move(leds, data, num_led, t):
    i = 0
    while i < num_led:
        if (i + len(data)) < num_led:
            if i != 0:
                leds.clean(i-1)
            j = i
            for d in data:
                leds.put_pixel(j, d[0], d[1], d[2])
                j+=1
            leds.send_buf()
            time.sleep_ms(t)
            i+=1
        else:
            break

# init led strip
strip = ws2812bstm32.WS2812(spi_bus=1, ledNumber=30, intensity=0.5)
```

```
91
92  #test_led(30, [0,0,250])
93  #test_led(60, [250,250,0])
94
95  #clear strip
96  strip.clear()
97  strip.send_buf()
98
99  # display two flag
100 dual_flag(strip, french_flag, italien_flag, 30, 500, True)
101
102 while 1:
103     #clear strip
104     strip.clear()
105     strip.send_buf()
106
107     # flag move
108     flag_move(strip, french_flag, 30, 500)
109
110     #clear strip
111     strip.clear()
112     strip.send_buf()
113
114     # flag move
115     flag_move(strip, italien_flag, 30, 500)
```

Chapitre 6

BLE (Bluetooth Low Energy)

On this chapter, we will see how to communicate in Bluetooth Low Power with the STBLESensor application and the STM32WB55 development board.

Installation of ST BLE Sensor on your smartphone.

Install on your smartphone, the ST BLE Sensor application on Google Play or IOS Store

6.1 BLE communication with MicroPython

To communicate in Bluetooth Low Energy with micropython, you will need the following code :

Note For using this example of BLE you need to copy the files on :

- material/codes/Capteurs/HTS221/hts221.py
- material/codes/Capteurs/LPS22/LPS22.py
- material/codes/Capteurs/LSM6DSO/LSM6DSO.py
- material/codes/Display/ssd1306/ssd1306.py
- material/codes/Display/max7219/max7219.py

and copy material/codes/example/ble_sensor.py on main.c

Listing 8 – BLE Gadget example.

```
from micropython import const
import struct
from struct import *
import bluetooth
import pyb

# I2C and SPI screen
from machine import I2C, SPI, Pin

# IKS01a3
# HTS221: Temperature / Humidity sensor
import hts221
# LPS22 sensor is connected on I2C1 of arduino connector
import LPS22
# LSM6DSO: Accelerometer / Gyroscope sensor
import LSM6DSO

# display SSD1306 is connected on I2C1 of arduino connector
import framebuf
import ssd1306
# display mx7219 is connected on SPI1 of arduino connector

import max7219

import math
import time
import random


spi = None
display_oled = None
display_led = None

max7219_present = 0 # to change if it's present on bus
```

```
# ----------------------------------------
# Screen & READ Picture
SCREEN_WIDTH = 128
SCREEN_HEIGHT = 64
def load_image(filename):
    with open(filename, 'rb') as f:
        f.readline()
        width, height = [int(v) for v in f.readline().split()]
        data = bytearray(f.read())
    return framebuf.FrameBuffer(data, width, height,
    ↪  framebuf.MONO_HLSB)

# ----------------------------------------------
#                  advertising
# ----------------------------------------------
# Advertising payloads are repeated packets of the following form:
#   1 byte data length (N + 1)
#   1 byte type (see constants below)
#   N bytes type-specific data

_ADV_TYPE_FLAGS = const(0x01)
_ADV_TYPE_NAME = const(0x09)
_ADV_TYPE_UUID16_COMPLETE = const(0x3)
_ADV_TYPE_UUID32_COMPLETE = const(0x5)
_ADV_TYPE_UUID128_COMPLETE = const(0x7)
_ADV_TYPE_UUID16_MORE = const(0x2)
_ADV_TYPE_UUID32_MORE = const(0x4)
_ADV_TYPE_UUID128_MORE = const(0x6)
_ADV_TYPE_APPEARANCE = const(0x19)
_ADV_TYPE_MANUFACTURER = const(0xFF)

# Generate a payload to be passed to gap_advertise(adv_data=...).
def advertising_payload(limited_disc=False, br_edr=False, name=None,
↪  services=None, appearance=0, manufacturer=0):
    payload = bytearray()

    def _append(adv_type, value):
        nonlocal payload
        payload += struct.pack('BB', len(value) + 1, adv_type) +
        ↪  value

    _append(_ADV_TYPE_FLAGS, struct.pack('B', (0x01 if limited_disc
    ↪  else 0x02) + (0x00 if br_edr else 0x04)))

    if name:
        _append(_ADV_TYPE_NAME, name)

    if services:
```

```
        for uuid in services:
            b = bytes(uuid)
            if len(b) == 2:
                _append(_ADV_TYPE_UUID16_COMPLETE, b)
            elif len(b) == 4:
                _append(_ADV_TYPE_UUID32_COMPLETE, b)
            elif len(b) == 16:
                _append(_ADV_TYPE_UUID128_COMPLETE, b)

    if appearance:
        # See org.bluetooth.characteristic.gap.appearance.xml
        _append(_ADV_TYPE_APPEARANCE, struct.pack('<h', appearance))

    if manufacturer:
        _append(_ADV_TYPE_MANUFACTURER, manufacturer)

    return payload

# ----------------------------------------------
#            BLE sensor service
# ----------------------------------------------
_IRQ_CENTRAL_CONNECT        = const(1)
_IRQ_CENTRAL_DISCONNECT     = const(2)
_IRQ_GATTS_WRITE            = const(3)
_IRQ_GATTS_READ             = const(4)

_ST_APP_UUID =
 ↪  bluetooth.UUID('00000000-0001-11E1-AC36-0002A5D5C51B')
# Environment char: 1C = 0x04 (TEMPERATURE) | 0x08 (Humidity) | 0x10
 ↪  (Pressure)

# Temperature char: 0x04 (TEMPERATURE)
 ↪  00XX0000-0001-11E1-AC36-0002A5D5C51B)
_TEMPERATURE_UUID =
 ↪  (bluetooth.UUID('00040000-0001-11E1-AC36-0002A5D5C51B'),
 ↪  bluetooth.FLAG_NOTIFY) #Temperature Char
# Humidity char: 0x08 (HUMIDITY)
 ↪  00XX0000-0001-11E1-AC36-0002A5D5C51B)
_HUMIDITY_UUID =
 ↪  (bluetooth.UUID('00080000-0001-11E1-AC36-0002A5D5C51B'),
 ↪  bluetooth.FLAG_NOTIFY) #humidity Char
# Pressure char: 0x10 (Pressure)
 ↪  00XX0000-0001-11E1-AC36-0002A5D5C51B)
_PRESSURE_UUID =
 ↪  (bluetooth.UUID('00100000-0001-11E1-AC36-0002A5D5C51B'),
 ↪  bluetooth.FLAG_NOTIFY) #pressure Char

# 00XX0000-0001-11E1-AC36-0002A5D5C51B
```

```
_ENVIRONMENTAL_UUID =
↪  (bluetooth.UUID('001C0000-0001-11E1-AC36-0002A5D5C51B'),
↪  bluetooth.FLAG_NOTIFY)
# LED char: 0x20 (LED) XX000000-0001-11E1-AC36-0002A5D5C51B
_LED_UUID = (bluetooth.UUID('20000000-0001-11E1-AC36-0002A5D5C51B'),
↪  bluetooth.FLAG_WRITE|bluetooth.FLAG_NOTIFY) # LED Char

_DISPLAY_UUID =
↪  (bluetooth.UUID('5930b535-bc2a-4436-a9c1-d98dcbb2bb23'),
↪  bluetooth.FLAG_WRITE|bluetooth.FLAG_NOTIFY)

_ST_APP_SERVICE = (_ST_APP_UUID, ( _TEMPERATURE_UUID,
↪  _HUMIDITY_UUID, _PRESSURE_UUID,_LED_UUID, _DISPLAY_UUID))

_PROTOCOL_VERSION    = const(0x01)
_DEVICE_ID           = const(0x80)      # Generic Nucleo Board
_FEATURE_MASK        = const(0x201C0000)   # Temperature (2^18) and
↪  Humidity(2^19) and Pressure (2^20) and LED (2^29)
_DEVICE_MAC          = [0x10, 0xE7, 0x7A, 0x78, 0x9A, 0xBC]
_MANUFACTURER        = pack('>BBI6B', _PROTOCOL_VERSION, _DEVICE_ID,
↪  _FEATURE_MASK, *_DEVICE_MAC)

led_blue = pyb.LED(1)
led_red = pyb.LED(3)

class BLESensor:
    # NOTE: The name could be changed to be more easily recognize
    def __init__(self, ble, screen_oled, screen_matrice, sensors,
    ↪  name='WB55-MPY-XXX'):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        self.screen_oled = screen_oled
        self.screen_matrice = screen_matrice
        self.sensors = sensors

        ↪  ((self._temp_handle,self._humidity_handle,self._pressure_handle,self._le
        ↪  = self._ble.gatts_register_services((_ST_APP_SERVICE, ))

        self._connections = set()
        self._payload = advertising_payload(name=name,
        ↪  manufacturer=_MANUFACTURER)
        self._advertise()
        self._handler = None

        self.state_disconnected = 0
        self.state_connected = 1
        self.switch_connected = 1
```

```
        self.switch_disconnected = 0
        self.state = self.state_disconnected
        self.switch = self.switch_disconnected

    def get_state(self):
        return self.state
    def get_switch(self):
        return self.switch

    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
            conn_handle, _, _, = data
            self._connections.add(conn_handle)
            self.state = self.state_connected
            print("Connected added: ", conn_handle)
            led_blue.on()
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _, = data
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
            self.state = self.state_disconnected
            print("Disconnected")
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle, = data
            # ------------------------------------------------
            # Write Section
            # LED switch
            if conn_handle in self._connections and value_handle ==
            ↪  self._led_handle:
                data_received =
                ↪  self._ble.gatts_read(self._led_handle)
                print("LED receive: ", data_received)
                self._ble.gatts_write(self._led_handle,
                ↪  struct.pack('<HB', 1000, data_received[0]))
                self._ble.gatts_notify(conn_handle,
                ↪  self._led_handle)
                if data_received[0] == 1:
                    led_red.on()
                    self.switch_logo(fill=1)
                    self.switch = self.switch_connected
                else:
                    led_red.off()
                    self.switch_logo()
                    self.switch = self.switch_disconnected
            # Display
            if conn_handle in self._connections and value_handle ==
            ↪  self._display_handle:
```

```
                data_received =
                ↪  self._ble.gatts_read(self._display_handle)
                print("DISPLAY receive: ", data_received)
                self._ble.gatts_write(self._display_handle,
                ↪  struct.pack('<HB', 1000, 1))
                self._ble.gatts_notify(conn_handle,
                ↪  self._display_handle)
                if max7219_present:
                    display_text_scroll(data_received)
            # ------------------------------------------------

    def set_data_environment(self, timestamp, temperature, pressure,
    ↪  humidity, notify=False):
        self._ble.gatts_write(self._environment_handle,
                              struct.pack('<hihh', timestamp,
                              ↪  pressure * 100, humidity*10,
                              ↪  temperature * 10))
        if notify:
            for conn in self._connections:
                self._ble.gatts_notify(conn,
                ↪  self._environment_handle)

    def set_data_temperature(self, timestamp, temperature,
    ↪  notify=False):
        self._ble.gatts_write(self._temp_handle, struct.pack('<hh',
        ↪  timestamp, temperature * 10))
        if notify:
            for conn in self._connections:
                self._ble.gatts_notify(conn, self._temp_handle)
    def set_data_humidity(self, timestamp, humidity, notify=False):
        self._ble.gatts_write(self._humidity_handle,
        ↪  struct.pack('<hh', timestamp, humidity * 10))
        if notify:
            for conn in self._connections:
                self._ble.gatts_notify(conn, self._humidity_handle)
    def set_data_pressure(self, timestamp, pressure, notify=False):
        self._ble.gatts_write(self._pressure_handle,
        ↪  struct.pack('<hi', timestamp, pressure *100))
        if notify:
            for conn in self._connections:
                self._ble.gatts_notify(conn, self._pressure_handle)

    def _advertise(self, interval_us=500000):
        self._ble.gap_advertise(interval_us, adv_data=self._payload)
        led_blue.off()

    def switch_logo(self, fill=None):
        if self.screen_oled is not None:
```

```
            self.screen_oled.fill(0)
            if fill is not None:
                logo_pbm = load_image("logo_st.pbm")
                self.screen_oled.blit(logo_pbm, 0, 0)
            self.screen_oled.show()

# ------------------------------------------
#                 Sensor management
# ------------------------------------------
class Sensors:
    def __init__(self, i2c):
        self.i2c = i2c
        self.temperature_humidity = None
        self.pressure             = None
        self.accelerometer_sensor = None
        if self.isPresentOnI2C(None, 95):
            self.temperature_humidity = hts221.HTS221(i2c)
        if self.isPresentOnI2C(None, 93):
            self.pressure      = LPS22.LPS22(i2c)
        if self.isPresentOnI2C(None, 107):
            self.accelerometer_sensor = LSM6DSO.LSM6DSO(i2c)

        self.ORIENTATION_LEFT_UP        = 0
        self.ORIENTATION_RIGHT_UP       = 1
        self.ORIENTATION_VERTICAL_DOWN  = 2
        self.ORIENTATION_VERTICAL_UP    = 3
        self.ORIENTATION_NORMAL         = 4


    def get_temperature(self):
        if self.temperature_humidity is not None:
            return int(self.temperature_humidity.get()[0])
        else:
            return (random.randint(0, 1000)) # random value between
             ↪  0 and 100,0 °C
    def get_humidity(self):
        if self.temperature_humidity is not None:
            return int(self.temperature_humidity.get()[1])
        else:
            return (random.randint(0, 1000)) # random value between
             ↪  0 and 100,0 %
    def get_pressure(self):
        if self.pressure is not None:
            return int(self.pressure.pressure())
        else:
            return (random.randint(8000, 12000)) # random value
             ↪  between 800 and 1200,0 hPa
    def get_altitude(self):
```

```
        if self.pressure is not None:
            return int(self.pressure.altitude())
        else:
            return (random.randint(0, 5000)) # random value between
             ↪  0 and 500,0 m
    def get_accel_raw(self):
        if self.accelerometer_sensor is not None:
            return self.accelerometer_sensor.get_a_raw()
        else:
            return [(random.randint(-1000, 1000)),
             ↪  (random.randint(-1000, 1000)),
             ↪  (random.randint(-1000, 1000))]

    def isPresentOnI2C(self, i2c2_list, id_tofound):
        result = 0
        if i2c2_list is None:
            local_i2clist = self.i2c.scan()
        else:
            local_i2clist = i2c2_list
        for i in local_i2clist:
            if i == id_tofound:
                return 1

    # Acceleromter
    def calculate_imu(self, accel):
        local_accel = accel
        if accel is None:
            local_accel = self.get_accel_raw()
        x = local_accel[0]
        y = local_accel[1]
        z = local_accel[2]
        if x == 0 and y == 0 and z == 0:
            return [0, 0, 0]
        pitch = round(math.atan(x / math.sqrt(y * y + z * z)) *
         ↪  180.0 * math.pi)
        roll  = round(math.atan(y / math.sqrt(x * x + z * z)) *
         ↪  180.0 * math.pi)
        yaw   = round(math.atan(z / math.sqrt(x * x + z * z)) *
         ↪  180.0 * math.pi)
        return [pitch, roll, yaw]


    def get_imu_orientation(self, accel):
        ''' return value array:
            [ enum, string ]
        '''
        pitch, roll, yaw = self.calculate_imu(accel)
        if abs(pitch) > 200:
```

```
            ''' (pitch > 0) ? ORIENTATION_LEFT_UP :
            ↪  ORIENTATION_RIGHT_UP;'''
            if pitch > 0:
                return [self.ORIENTATION_LEFT_UP, pitch, roll,
                ↪  "left-up"]
            else:
                return [self.ORIENTATION_RIGHT_UP, pitch, roll,
                ↪  "right-up"]
        elif abs(roll) > 200:
            '''
                ret = (roll < 0) ? ORIENTATION_VERTICAL_DOWN :
                    ORIENTATION_VERTICAL_UP;'''
            if roll < 0:
                return [self.ORIENTATION_VERTICAL_DOWN, pitch, roll,
                ↪  "bottom-down"]
            else:
                return [self.ORIENTATION_VERTICAL_UP, pitch, roll,
                ↪  "top-up"]
        else:
            return [self.ORIENTATION_NORMAL, pitch, roll, "normal"]

# ----------------------------------------------
#                 Oled display
# ----------------------------------------------
class DisplayOnScreen:
    def __init__(self, display):
        self.display = display
        self.remaing_scren_display = 0
        self.DISPLAYED_TIME = 2000

    def dec(self):
        #print("[DEBUG]: request to dec remaining = %d" %
        ↪  self.remaing_scren_display)
        if self.remaing_scren_display < 0:
            self.remaing_scren_display = 0
        elif self.remaing_scren_display == 0:
            self.remaing_scren_display = 0
        else:
            self.remaing_scren_display -= 500
    def inc(self, t):
        self.remaing_scren_display += t

    def reset_remaining(self):
        self.remaing_scren_display = 0

    def display_logo(self, ble_force_display=False):
        if ble_force_display:
            return
```

```
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            logo_pbm = load_image("logo_st.pbm")
            self.display.blit(logo_pbm, 0, 0)

            self.display.show()

    def display_clear(self):
        #print("[DEBUG]: remaining = %d" %
        ↪  self.remaing_scren_display)
        self.dec()
        if self.remaing_scren_display > 0:
            return
        if self.display is not None:
            self.display.fill(0)
            self.display.show()


    def display_temperature_humidity(self, temp, hum,
    ↪  ble_force_display):
        if ble_force_display:
            return
        #print("[DEBUG]: remaining = %d" %
        ↪  self.remaing_scren_display)
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            temperature_pbm = load_image("climate.pbm")
            self.display.blit(temperature_pbm, 0, 16)

            self.display.text('{:14s}'.format('Temperature:'), 32,
            ↪  0)
            self.display.text('{:^14s}'.format(str(temp) + 'C'), 16,
            ↪  16)

            self.display.text('{:14s}'.format('Humidity:'), 32, 32)
            self.display.text('{:^14s}'.format(str(hum) + '%'), 16,
            ↪  48)

            self.display.show()
```

```
    def display_temperature(self, temp, ble_force_display):
        #print("[DEBUG]: remaining = %d" %
        ↪  self.remaing_scren_display, " Force BLE:",
        ↪  ble_force_display)
        if ble_force_display:
            return
        #print("[DEBUG]: remaining = %d" %
        ↪  self.remaing_scren_display)
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            temperature_pbm = load_image("climate.pbm")
            self.display.blit(temperature_pbm, 0, 16)

            self.display.text('{:14s}'.format('Temperature:'), 32,
            ↪  0)
            self.display.text('{:^14s}'.format(str(temp) + 'C'), 16,
            ↪  16)

            self.display.show()

    def display_humidity(self, hum, ble_force_display):
        #print("[DEBUG]: remaining = %d" %
        ↪  self.remaing_scren_display, " Force BLE:",
        ↪  ble_force_display)
        if ble_force_display:
            return
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            temperature_pbm = load_image("climate.pbm")
            self.display.blit(temperature_pbm, 0, 16)

            self.display.text('{:14s}'.format('Humidity:'), 32, 0)
            self.display.text('{:^14s}'.format(str(hum) + '%'), 16,
            ↪  16)

            self.display.show()

    def display_pressure_altitude(self, press, alt,
    ↪  ble_force_display):
        if ble_force_display:
```

```
            return
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            pressure_pbm = load_image("pressure.pbm")
            self.display.blit(pressure_pbm, 0, 16)

            self.display.text('{:18s}'.format('Pressure:'), 36, 0)

            ↪  self.display.text('{:^14s}'.format('{:.1f}'.format(press)
            ↪  + 'hPa'), 18, 16)

            self.display.text('{:18s}'.format('Altitude:'), 36, 32)
            self.display.text('{:^14s}'.format('{:.1f}'.format(alt)
            ↪  + 'M'), 18, 48)

            self.display.show()

    def display_pressure(self, press, ble_force_display):
        if ble_force_display:
            return
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)

            pressure_pbm = load_image("pressure.pbm")
            self.display.blit(pressure_pbm, 0, 16)

            self.display.text('{:18s}'.format('Pressure:'), 36, 0)

            ↪  self.display.text('{:^14s}'.format('{:.1f}'.format(press)
            ↪  + 'hPa'), 18, 16)

            self.display.show()

    def display_altitude(self, alt, ble_force_display):
        if ble_force_display:
            return
        if self.remaing_scren_display > 0:
            return
        self.inc(self.DISPLAYED_TIME)
        if self.display is not None:
            self.display.fill(0)
```

```
            pressure_pbm = load_image("pressure.pbm")
            self.display.blit(pressure_pbm, 0, 16)

            self.display.text('{:18s}'.format('Altitude:'), 36, 0)
            self.display.text('{:^14s}'.format('{:.1f}'.format(alt)
             ↪  + 'M'), 18, 16)

            self.display.show()
# ----------------------------------------
#                 screen max7219
# ----------------------------------------
def display_text_scroll(text):
    if max7219_present:
        if display_led is not None:
            # for 4 led blocks
            for p in range(4 * 8, len(text) * -8 - 1, -1):
                display_led.fill(False)
                display_led.text(text, p, 0, not False)
                display_led.show()
                time.sleep_ms(50)

# ----------------------------------------
#                 main
# ----------------------------------------
i2c1 = I2C(1)
i2c1_list = []
spi = SPI(1, baudrate=10000000)
sensors = None

if __name__ == '__main__':
   # scan i2c bus to see which ip components are present
    i2c1_list = i2c1.scan()
    print(i2c1_list)

    # wait to be sure all the component are present
    time.sleep_ms(1000)
    # init sensor
    sensors = Sensors(i2c1)

    # init display
    display_led = None
    if max7219_present:
        display_led = max7219.Max7219(32, 8, spi, Pin('B2'))
    display_oled = None
    if sensors.isPresentOnI2C(i2c1_list, 60):
        display_oled = ssd1306.SSD1306_I2C(SCREEN_WIDTH,
         ↪  SCREEN_HEIGHT, i2c1)
```

```
display_text_scroll("STMicroelectronics")

# init bluetooth
ble = bluetooth.BLE()
ble_device = BLESensor(ble, display_oled, display_led, sensors,
 ↪  name='WB55-GADGET')

display_text_scroll("BLE ready")
print("BLESensor initialized: ", ble.config("mac"))

screen_displayed = DisplayOnScreen(display_oled)
screen_displayed.display_logo()

while True:
    timestamp = time.time()
    humidity = sensors.get_humidity()
    temperature = sensors.get_temperature()
    pressure = sensors.get_pressure()
    altitude = sensors.get_altitude()

    # BLUETOOTH management
    if ble_device.get_state() == ble_device.state_connected:
        # via BLE: send timestamp and environment with
         ↪  notification
        print("send BLE environment")
        ble_device.set_data_temperature(timestamp, temperature,
         ↪  notify=1)
        ble_device.set_data_humidity(timestamp, humidity,
         ↪  notify=1)
        ble_device.set_data_pressure(timestamp, pressure,
         ↪  notify=1)


    ble_force_display = False
    if ble_device.get_switch() == ble_device.switch_connected:
        ble_force_display = True
    imu = sensors.get_imu_orientation(None)
    if imu[0] == sensors.ORIENTATION_LEFT_UP:
        # display humidity
        print("Humidite: %s %s" % (str(humidity), "%") )
        screen_displayed.display_humidity(humidity,
         ↪  ble_force_display)
        time.sleep_ms(1000)
    elif imu[0] == sensors.ORIENTATION_RIGHT_UP:
        # display temperature
        print("Temperature: %s C" % str(temperature) )
        screen_displayed.display_temperature(temperature,
         ↪  ble_force_display)
```

```
            time.sleep_ms(1000)
        elif imu[0] == sensors.ORIENTATION_VERTICAL_DOWN:
            # display altitude
            print("Altitude: %s M" % str(altitude) )
            screen_displayed.display_altitude(altitude,
             ↪  ble_force_display)
            time.sleep_ms(1000)
        elif imu[0] == sensors.ORIENTATION_VERTICAL_UP:
            # display pressure
            print("Pressure: %s hPa" % str(pressure) )
            screen_displayed.display_pressure(pressure,
             ↪  ble_force_display)
            time.sleep_ms(1000)
        if ble_device.get_state() == ble_device.state_connected:
            if ble_device.get_switch() ==
             ↪  ble_device.switch_connected:
                screen_displayed.reset_remaining()
            else:
                screen_displayed.display_clear()
        else:
            screen_displayed.display_clear()

        time.sleep_ms(1000)
```

(code available on material/codes/examples/ble_sensor_nucleo_wb55.py)

6.2 Play with STBLESensor application and MicroPython

Launch STBLESensor application on your smartphone.
Click on 'magnifying glass' icon to list all the BLE peripheral availables.

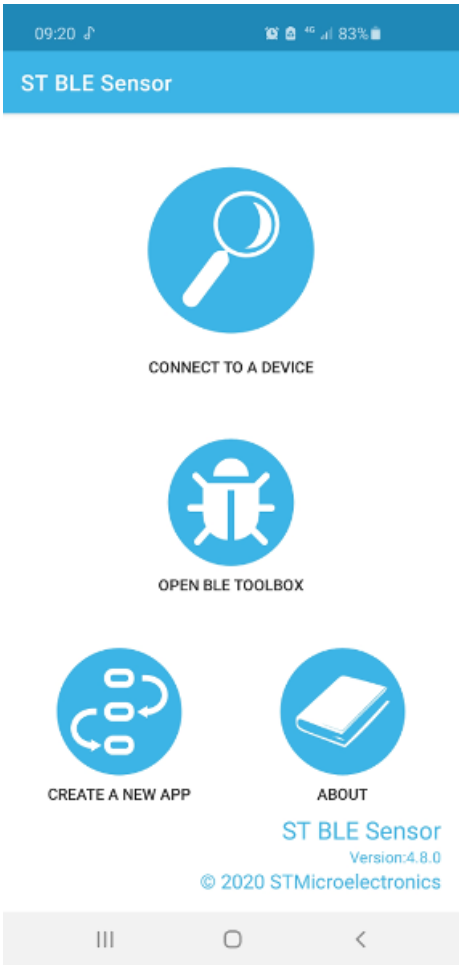


FIGURE 6.1 – STBLESensor

Select the peripheral corresponding to your MicroPython implementation.

FIGURE 6.2 – STBLESensor : search

You must show the temperature

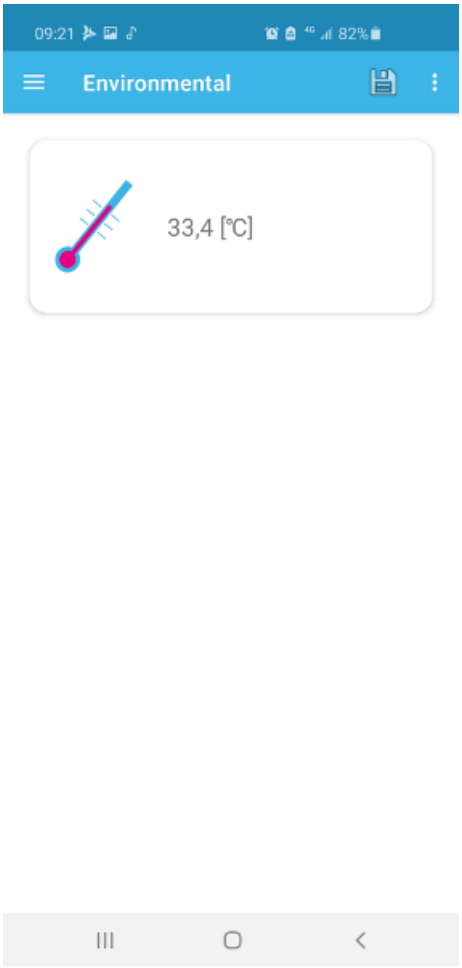


FIGURE 6.3 – STBLESensor : temperature

Or all the environment data (following your implementation)

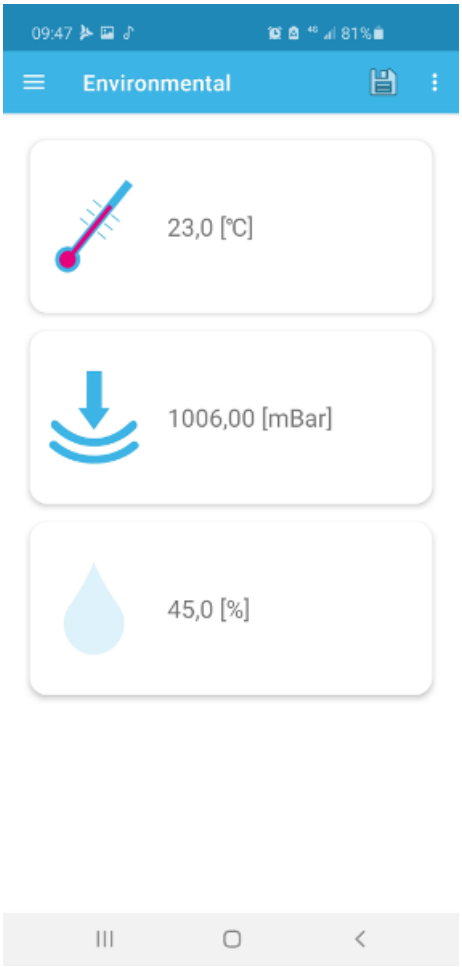


FIGURE 6.4 – STBLESensor : environmental

For turn on LED (switch), go on ’menu’ and select ’switch’

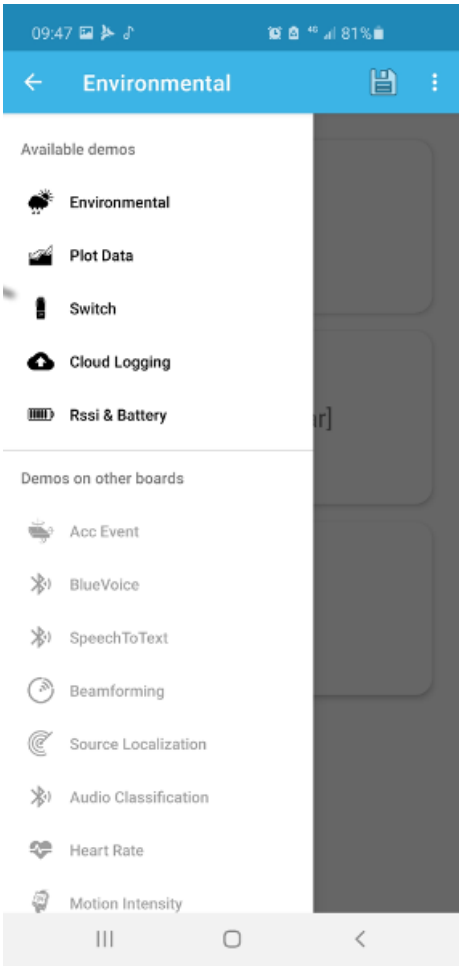
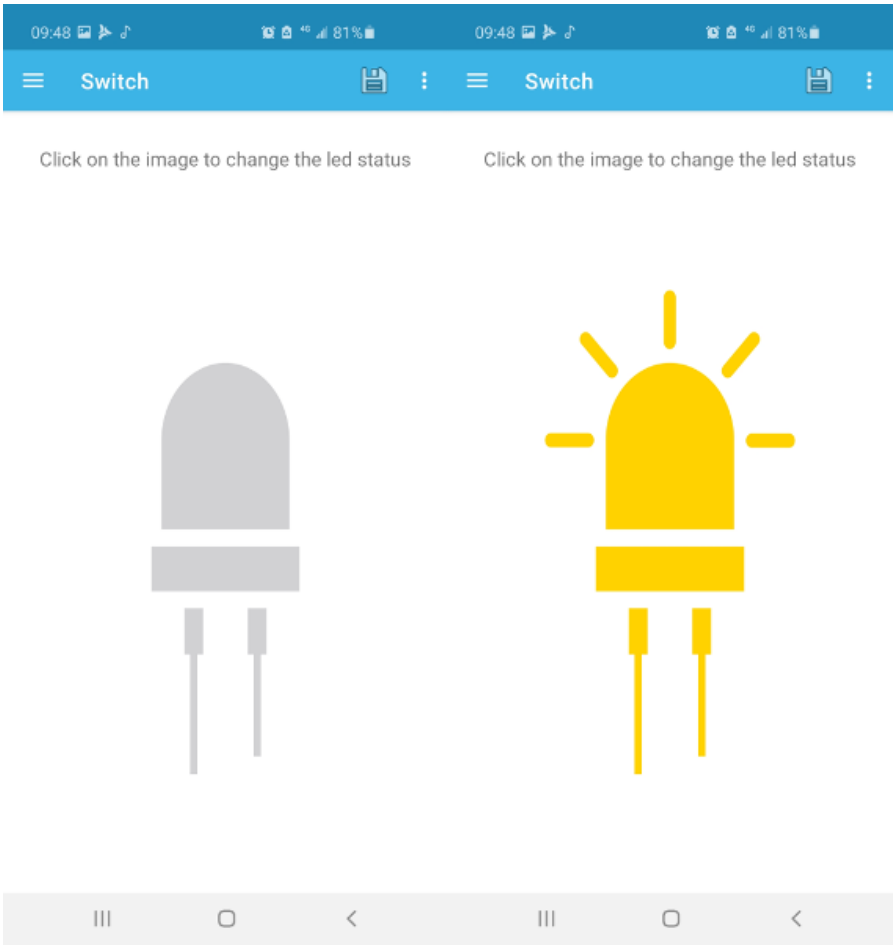


FIGURE 6.5 – STBLESensor : switch

Click on led icon to turn on or off the led



6.3 Play with Linux gatttool and MicroPython

Get list of VLE Sensor available :

```
$> sudo hcitool lescan
LE Scan ...
18:5F:5B:53:CE:63 (unknown)
59:94:75:98:B9:12 (unknown)
59:94:75:98:B9:12 (unknown)
43:F4:F2:02:68:71 (unknown)
43:F4:F2:02:68:71 (unknown)
03:FD:5F:C5:91:E9 (unknown)
24:9B:97:8C:72:76 (unknown)
0B:2A:85:8A:DF:3E (unknown)
02:02:27:4E:22:02 WB55-GADGET
02:04:73:9D:51:9E (unknown)
1F:7D:A6:A4:8F:DC (unknown)
42:AC:24:D5:31:79 (unknown)
42:AC:24:D5:31:79 (unknown)
4A:F6:3A:80:39:0C (unknown)
4A:F6:3A:80:39:0C (unknown)
```

(In this case the Mac address of our BLE peripheral is : **02 :02 :27 :4E :22 :02**)

List characteristics available

```
$> gatttool -b 02:02:27:4E:22:02 --characteristics
handle = 0x000b, char properties = 0x02, char value handle = 0x000c,
 ↪  uuid = 00002a00-0000-1000-8000-00805f9b34fb
handle = 0x000d, char properties = 0x02, char value handle = 0x000e,
 ↪  uuid = 00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0010, char properties = 0x20, char value handle = 0x0011,
 ↪  uuid = 00002a05-0000-1000-8000-00805f9b34fb
handle = 0x0014, char properties = 0x18, char value handle = 0x0015,
 ↪  uuid = 20000000-0001-11e1-ac36-0002a5d5c51b
handle = 0x0017, char properties = 0x18, char value handle = 0x0018,
 ↪  uuid = 5930b535-bc2a-4436-a9c1-d98dcbb2bb23
handle = 0x001a, char properties = 0x12, char value handle = 0x001b,
 ↪  uuid = 00040000-0001-11e1-ac36-0002a5d5c51b
handle = 0x001d, char properties = 0x12, char value handle = 0x001e,
 ↪  uuid = 00080000-0001-11e1-ac36-0002a5d5c51b
handle = 0x0020, char properties = 0x12, char value handle = 0x0021,
 ↪  uuid = 00100000-0001-11e1-ac36-0002a5d5c51b
```

The interesting charasteristics are :

- uuid = 20000000-0001-11e1-ac36-0002a5d5c51b char value handle = 0x0015 switch features (LED swtich)
- uuid = 00040000-0001-11e1-ac36-0002a5d5c51b char value handle = 0x001b Temperature feature
- uuid = 00080000-0001-11e1-ac36-0002a5d5c51b char value handle = 0x001e Humidity feature
- uuid = 00100000-0001-11e1-ac36-0002a5d5c51b char value handle = 0x0021 Pressure feature
- uuid = 5930b535-bc2a-4436-a9c1-d98dcbb2bb23 char value handle = 0x0018 Display text

Change Led Status (SWITCH)

```
$> echo "turn on led"
$> gatttool -b 02:02:27:4E:22:02 --char-write-req --handle=0x0015
 ↪  --value=01
$> echo "turn off led"
$> gatttool -b 02:02:27:4E:22:02 --char-write-req --handle=0x0015
 ↪  --value=00
```

Read Environmental values :

```
$> echo "Read value"
$> gatttool -b 02:02:27:4E:22:02 --char-read --handle=0x0015
 ↪  --listen
Notification handle = 0x0015 value: e0 4a f0 00
Notification handle = 0x0018 value: e0 4a cc 01
Notification handle = 0x001b value: e0 4a 30 88 01 00
Notification handle = 0x0015 value: e1 4a f0 00
Notification handle = 0x0018 value: e1 4a cc 01
Notification handle = 0x001b value: e1 4a 30 88 01 00
Notification handle = 0x0015 value: e2 4a f0 00
Notification handle = 0x0018 value: e2 4a cc 01
Notification handle = 0x001b value: e2 4a 30 88 01 00
Notification handle = 0x0015 value: e3 4a f0 00
Notification handle = 0x0018 value: e3 4a cc 01
Notification handle = 0x001b value: e3 4a 30 88 01 00
CTRL+C (to stop)
```

Notification handle = 0x0015 value : e0 4a f0 00

- handle = 0x0015 : Temperature feature

- e0 4a = time stamp
- f0 00 = value on BigEndian => 00f0 (temperature on hexa * 10)

Notification handle = 0x0018 value : e0 4a cc 01

- handle = 0x0015 : Humidity feature
- e0 4a = time stamp
- cc 01 = value on BigEndian => 01cc (humidity on hexa * 10)

Notification handle = 0x001b value : e0 4a 30 88 01 00

- handle = 0x0015 : Pressure feature

- e0 4a = time stamp
- 30 88 01 00 = value on BigEndian => 00018830 (pression on hexa * 100)

Display text on screen via 5930b535-bc2a-4436-a9c1-d98dcbb2bb23 service

```
$> gatttool -b 02:02:27:4E:22:02 --char-write-req --handle=0x0018
↪  --value=$(xxd -pu <<< "COVID-19" | sed "s/0a//");
```