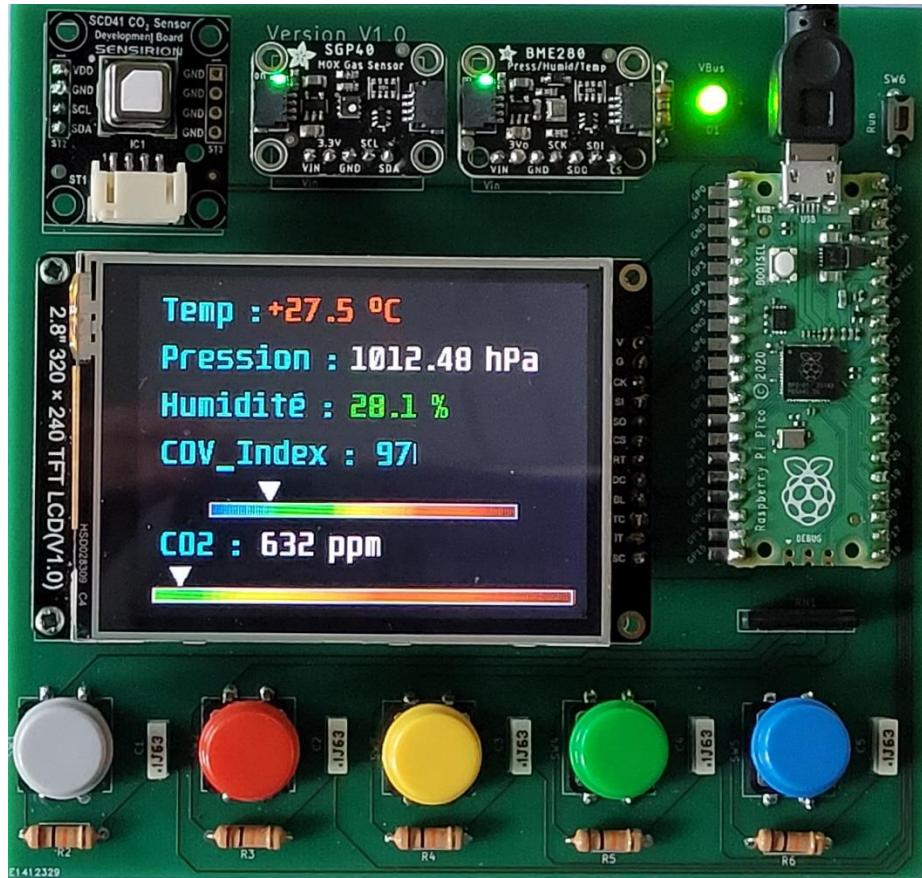


Station de mesures de paramètres environnementaux



Auteur : Pierre Gaucher (pierre.gaucher@univ-tours.fr)

Version 1.0. 30.09.2022

MàJ :

Ce document est en cours de rédaction et peut contenir des erreurs. Merci de bien vouloir les signaler afin d'en améliorer le contenu.

Reproduction et diffusion interdites sans autorisation préalable.

Table des matières

1.	Introduction.....	5
2.	Fonctionnalités attendues.....	5
3.	Moyens matériels.....	6
3.1.	Plateforme matérielle	6
3.2.	Carte microcontrôleur Raspberry Pi Pico	7
3.3.	Capteurs et périphériques utilisés.....	9
3.3.1.	Capteur de mesure de CO2 : SCD41	9
3.3.2.	Capteur index_COV : SGP40	9
3.3.3.	Capteur de mesure de température – pression et taux humidité : BME280	10
3.4.	Interface utilisateur : écran DFR0665	11
4.	Première mise en œuvre de la plateforme microcontrôleur Raspberry Pi Pico	12
4.1.	Structure générale d'un programme.....	12
4.2.	Faire clignoter une Led et gérer sa luminosité	13
4.2.1.	Initialisation des ressources	13
4.2.2.	Boucle infinie.....	14
4.2.3.	Exécution du programme sur la carte microcontrôleur	14
5.	Gérer la luminosité de la Led.....	23
5.1.	Principe de la PWM	23
5.2.	Mise en œuvre	23
6.	Gérer des événements temporels - timer.....	24
7.	Acquérir la température, l'humidité et la pression atmosphérique	27
7.1.	Connectique du shield BME280	27
7.2.	Mise en œuvre logicielle	29
7.2.1.	Mise en place de la librairie logicielle	29
7.2.2.	Initialisation de la connexion I2C	30
7.2.3.	Lectures des données du BME280 : acquisition des mesures	30
8.	Acquérir la mesure du taux de CO2.....	32
8.1.	Librairie logicielle	33
8.2.	Création de l'objet capteur SCD41 et initialisations	34
8.3.	Acquisition des mesures.....	35
8.4.	Gestion conjointe des capteurs BME280 et SCD41	37
8.4.1.	Approche 1 : utilisation de temporisation avec sleep (facultatif)	37
8.4.2.	Approche 2 : Gestion de la compensation de la mesure du taux de CO2 (facultatif).....	37
8.4.3.	Approche 3 : gestion par timers et alarmes uniquement (obligatoire).....	37
9.	Acquérir la valeur de l'index COV	39
9.1.	Librairies logicielles.....	40
9.2.	Mise en œuvre	40
9.3.	Validation du fonctionnement.....	41

10.	Synthèse1 : acquisition de l'ensemble des grandeurs environnementales.....	43
11.	Gestion d'un afficheur graphique	45
11.1.	Eléments de mise en œuvre	45
11.2.	Systèmes de coordonnées écran	46
11.3.	Première mise en œuvre de l'écran	47
11.3.1.	Prise en charge des librairies	48
11.3.2.	Prise en charge des polices de caractères	48
11.3.3.	Mise en œuvre logicielle	49
11.4.	Affichage des mesures sur l'écran TFT.....	50
11.4.1.	Définition des formats d'affichage.....	50
11.4.2.	Librairie Affichage_Graphique.py.....	51
11.4.3.	Mise en œuvre : Ecran 1	52
11.4.4.	Mise en œuvre : Ecran 2	53
11.5.	Simulation de la dalle tactile	55
11.5.1.	Mécanisme d'interruption : principe	56
11.5.2.	Gérer l'appui sur un bouton poussoir : mise en œuvre matérielle	57
11.5.3.	Gérer l'appui sur un bouton poussoir : mise en œuvre logicielle	57
11.5.4.	Gestion complète des boutons poussoir	60
11.6.	Utilisation de la dalle tactile	61
11.6.1.	Interface matérielle de la dalle tactile	61
11.6.2.	Référentiel écran TFT et dalle tactile du DFR0665	62
11.6.3.	Mise en œuvre logicielle 1 : calibration de la dalle tactile.....	63
11.6.4.	Mise en œuvre logicielle 2 : mapping écran TFT et dalle tactile	64
11.6.5.	Routine d'interruption associée à la dalle tactile	65
11.6.6.	Une première interface utilisateur : exemple	66
11.6.7.	Interface utilisateur : version finale.....	69
12.	Annexes	70
12.1.	Installation de l'IDE de programmation Thonny	70
12.2.	Configuration de l'IDE Thonny	75
12.3.	11.3 Interface de l'IDE Thonny.....	77
12.4.	Installation de microPython sur la carte Raspberry Pi Pico	77
12.4.1.	Mise à jour firmware depuis l'IDE Thonny	77
12.4.2.	Mise à jour firmware indépendante de l'IDE Thonny	79
12.5.	Diagramme d'E/S des broches de la carte Raspberry Pi Pico	82
12.6.	Le bus I2C	83
12.6.1.	Bus de communication.....	83
12.6.2.	Le bus I2C	83
12.6.3.	Le bus SPI	84
12.7.	Formatage de chaîne de caractères : <i>format</i>	84

12.9.	Calcul des composantes RGB d'une couleur en fonction de la longueur d'onde	87
12.9.1.	Spectre des couleurs visibles.....	87
12.9.2.	Système de coordonnées de l'espace colorimétrique	88
12.9.3.	Calcul des composantes RGB pour λ donnée	89
12.9.4.	Table des composantes chromatiques et interpolation linéaire	90
12.10.	Schéma électronique de la plateforme matérielle.....	91
12.11.	Tables des figures	93
12.12.	Table des tableaux	95

1. Introduction

L'objectif de ce projet est de sensibiliser les élèves ingénieurs à la mise en œuvre de systèmes nécessitant l'utilisation de ressources matérielles (capteurs, périphériques d'entrée – sortie, microcontrôleur) et la programmation de ce type de système. C'est donc un projet qui relève aussi bien de l'électronique que de l'informatique embarquée.

La finalité du projet est de programmer en Python une plateforme microcontrôleur destinée à effectuer des mesures environnementales telles que la température, la pression atmosphérique, le taux de CO₂, ... Au travers de cet exercice, plusieurs aspects seront abordés :

- La mise en œuvre de capteurs et de dispositifs d'entrée - sortie, aussi bien du point de vue matériel que logiciel ;
- La programmation de ressources du microcontrôleur afin de recueillir des mesures environnementales et de les afficher sur un écran utilisé comme interface utilisateur.
- La prise en compte d'événements particuliers au travers de la gestion du mécanisme d'interruption.

2. Fonctionnalités attendues

Il s'agit de mettre en œuvre une station de mesure de données environnementales. Les grandeurs mesurées seront :

- La température (en °C).
- La pression atmosphérique (en hPa).
- Le taux d'humidité relative (en %).
- La concentration des composés Organiques Volatils par la mesure d'un index COV (sans unité).
- Le taux de CO₂ (en ppm).

La mesure de ces différentes grandeurs sera rafraîchie selon une périodicité propre à chacun des capteurs utilisés.

L'affichage des mesures s'effectuera sur un écran TFT¹ couleur rétroéclairé, muni d'une dalle tactile. Le rafraîchissement de l'affichage des grandeurs mesurées sera effectué avec la même périodicité que celle des mesures réalisées.

En plus de l'affichage des mesures, l'utilisateur pourra visualiser la qualité de l'air à l'aide d'échelles graphiques couleur et d'un curseur qui rendra compte de la valeur de la mesure. Ces échelles graphiques seront uniquement associées aux mesures de l'index COV et du taux de CO₂.

Sur une période donnée, les valeurs min et max des grandeurs mesurées seront affichées à la demande de l'utilisateur. Il doit être possible d'effectuer un « reset » de ces valeurs min et max par la valeur des dernières mesures acquises. Les grandeurs retenues pour l'affichage des valeurs min et max sont :

- La température (en °C).
- Le taux d'humidité relative (en %).
- La concentration des composés Organiques Volatils par la mesure d'un index cov (sans unité).
- Le taux de CO₂ (en ppm).

L'écran utilisé, muni de sa dalle tactile, constitue l'interface utilisateur. La dalle tactile permettra :

- D'effectuer le passage de l'affichage des mesures courantes vers les mesures min et max.

¹ Thin-Film Transistor ou Transistor en couches minces (TCM). Type de transistor utilisé dans les écrans à cristaux liquides.

- De réaliser le « reset » des valeurs min et max.
- De gérer le rétroéclairage de l'écran sur une échelle de 5 niveaux.
- De provoquer le retour à l'écran d'affichage des mesures depuis l'affichage de l'écran des valeurs min et max.

Pour des raisons pédagogiques, il sera possible de simuler l'utilisation de la dalle tactile à l'aide de boutons poussoir.

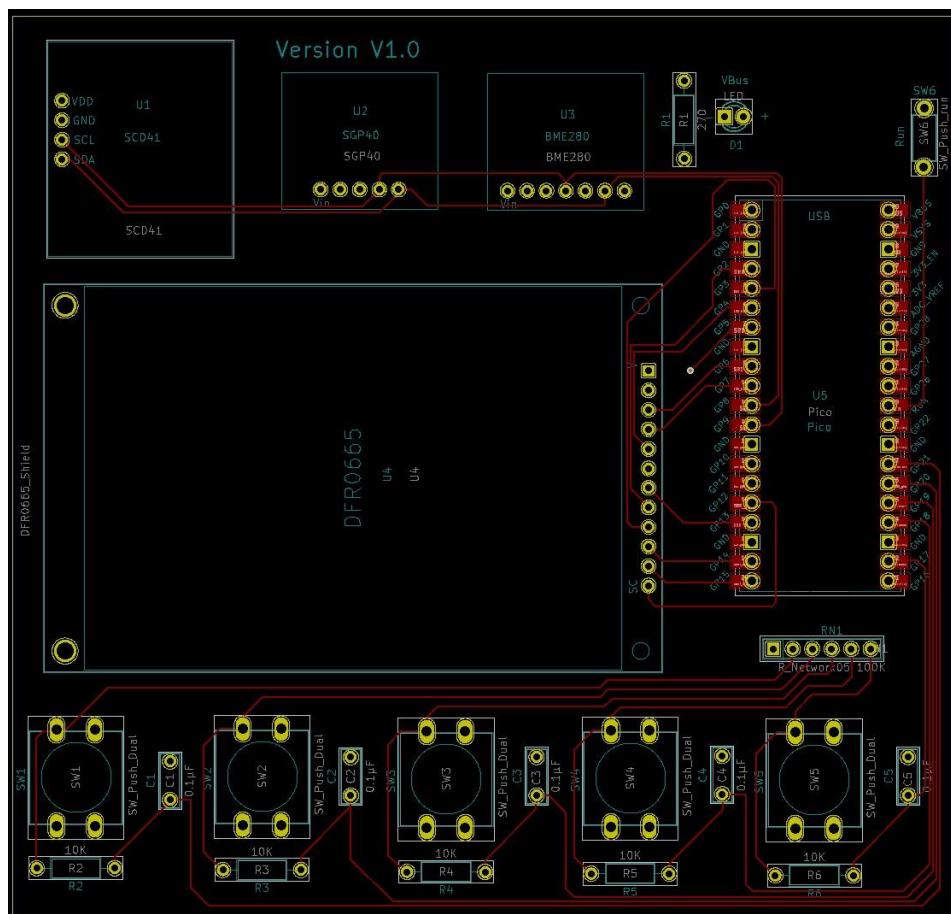
L'ensemble de ces fonctionnalités sera supporté par la mise en œuvre :

- De capteurs.
- D'un écran TFT couleur muni d'une dalle tactile.
- D'un microcontrôleur pour piloter l'ensemble des ressources.
- De boutons poussoir pour simuler l'utilisation de la dalle tactile (aspect pédagogique essentiellement).
- D'une plateforme sous la forme d'un circuit imprimé ou PCB² spécifique, afin de faciliter la mise en œuvre matérielle, et destiné à accueillir l'ensemble des ressources matérielles.

3. Moyens matériels

3.1. Plateforme matérielle

Elle se présente sous la forme d'un PCB destiné à accueillir l'ensemble des périphériques liés à la mise en œuvre du projet (cf. Figure 1)³.



² PCB : Printed Circuit Board.

³ Conception du PCB sous Kicad V5.1.10

Figure 1 : PCB plateforme matérielle - vue de dessus

Du fait du routage des pistes du PCB, les ressources de la carte microcontrôleur sont déjà pré affectées vis-à-vis des périphériques à piloter.

Les périphériques qui seront connectés sur le PCB sont :

- SCD41 : capteur de mesure de CO₂ (et aussi température et taux relatif d'humidité).
- SGP40 : capteur de mesure de l'index des Composants Organiques Volatils.
- BME280 : Capteur de mesure de la température, la pression atmosphérique et le taux d'humidité relative.
- DFR0665 : écran graphique couleur TFT + dalle tactile.
- Pico : carte microcontrôleur Raspberry Pi Pico.
- SW1 – SW5 : Boutons poussoir destinés à émuler les fonctionnalités de la dalle tactile.
- Run : Bouton poussoir de reset de la carte microcontrôleur.

3.2. Carte microcontrôleur Raspberry Pi Pico

La carte RaspberryPi Pico (cf. Figure 2) permet d'accéder à l'ensemble des ports d'E/S du microcontrôleur RP2040 au travers de 2 rangées de 20 broches. Elle possède une led pré câblée, un port USB dont les fonctions sont d'alimenter la carte et de gérer la communication série, un bouton BOOTSEL afin de faire basculer le microcontrôleur en mode mémoire de stockage USB.

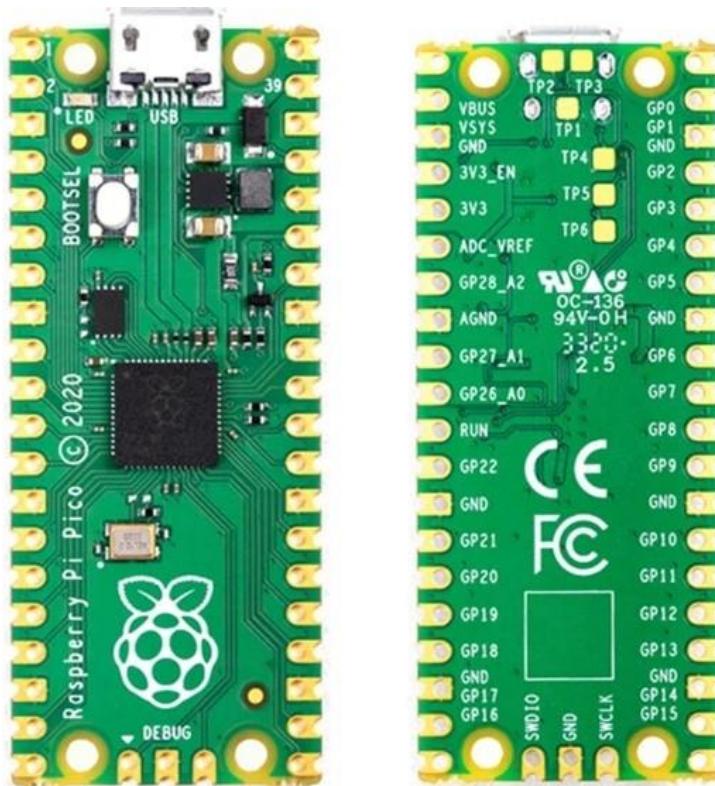


Figure 2: Carte Raspberry Pi pico - recto - verso

Les principales caractéristiques sont résumées ci-dessous :

- Dual ARM Cortex-M0+ @ 133MHz
- 264kB on-chip SRAM in six independent banks
- Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus

- 30 GPIO pins, 4 of which can be used as analogue inputs
- Peripherals
 - 2 UARTs
 - 2 SPI controllers
 - 2 I2C controllers
 - 16 PWM channels
 - USB 1.1 controller and PHY, with host and device support

L'ensemble des E/S (Entrées / Sorties) est décrit par la figure ci-dessous (cf. Figure 3). Les E/S sont regroupées par fonctionnalités. Une même broche peut assurer différentes fonctions.

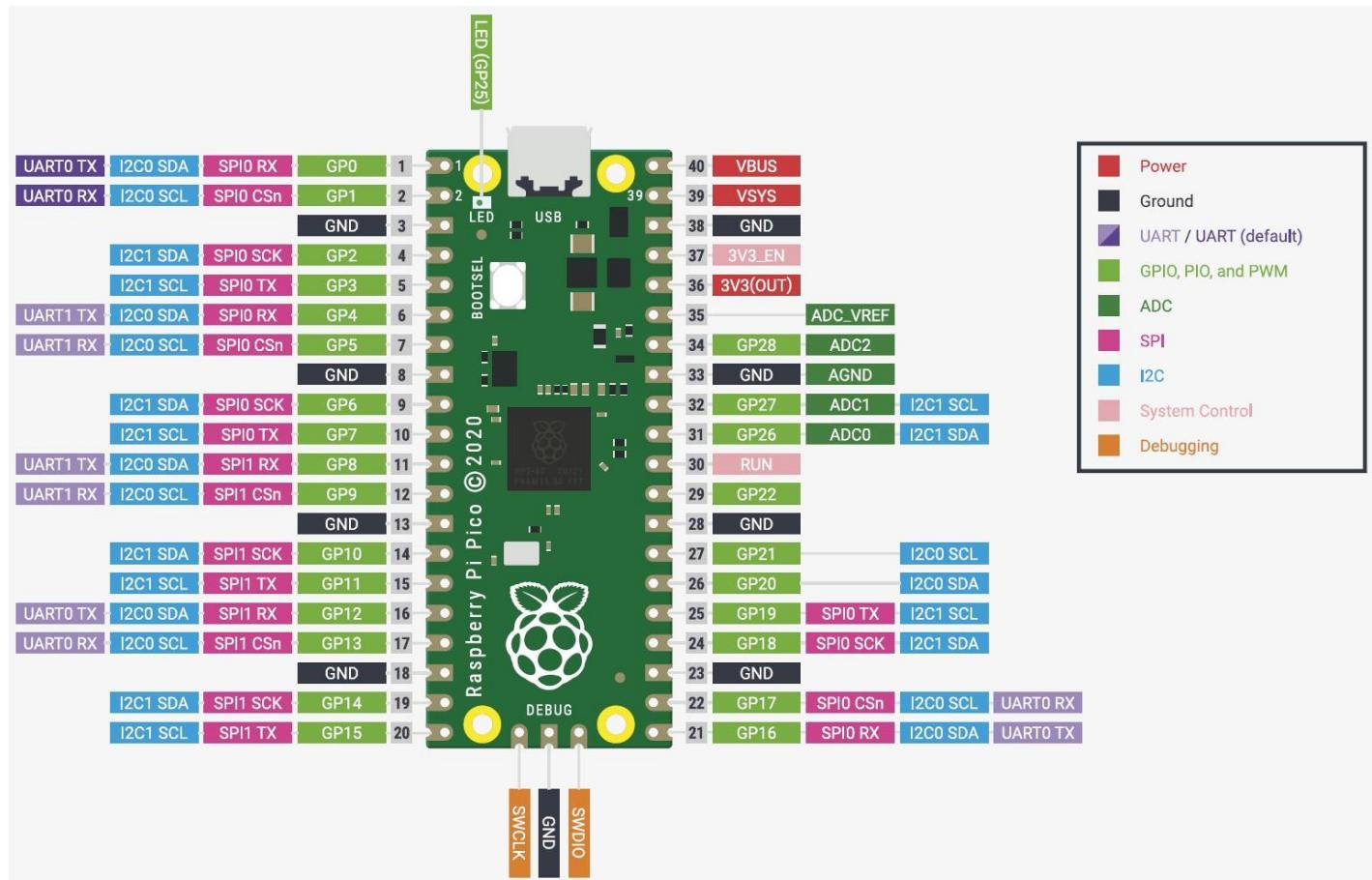


Figure 3: Fonctionnalités des ports GPIO Raspberry Pi pico

Par la suite, nous désignerons les broches de la carte microcontrôleur par leur numéro de broche sur le PCB de 1 à 40.

La dénomination GP_i, i dans {1, ..., 22, 26, 27, 28} permet de spécifier le port GPIO⁴ qui sera utilisé lors de la mise en œuvre de certaines ressources / fonctionnalités du microcontrôleur.

Pour approfondir :

<https://www.raspberrypi.com/products/raspberry-pi-pico/specifications/>

⁴ GPIO : General Purpose Input Output

3.3. Capteurs et périphériques utilisés

3.3.1. Capteur de mesure de CO₂ : SCD41

Le capteur SCD41 conçu par la société Sensirion permet d'effectuer des mesures du taux de CO₂. Sa mise en œuvre repose sur le kit de développement SCD41 -Devkit qui permet de gérer toute l'interface électronique du capteur SCD41 (cf. Figure 4). L'ensemble des ressources documentaires et logicielles sont disponibles via <https://sensirion.com/products/catalog/SEK-SCD41/>. Ses principales caractéristiques sont résumées ci-dessous :

- Mesure du taux de CO₂ dans la plage [400 ; 5000] ppm.
- Mesure de la température et du taux d'humidité.
- Auto calibration du capteur.
- Compensation de la mesure du taux de CO₂ en fonction de la pression atmosphérique.
- Mode périodique de mesure (période de 5s) ou single shot.
- Communication sur bus I2C et @I2C : 0x62
- Disponibilité de librairies logicielles pour environnement Arduino ou Python.



Figure 4: Capteur de CO₂ SCD41 sur dev. Board

3.3.2. Capteur index_COV : SGP40

La mesure de la concentration des Composants Organiques Volatils (COV) sera accessible sous la forme de la mesure d'un index_COV délivrée par le capteur SGP40 de chez Sensirion.. L'ensemble des ressources documentaires et logicielles sont disponibles via le lien <https://sensirion.com/products/catalog/SGP40/>. Ses principales caractéristiques sont résumées ci-dessous :

- Mesure d'un index_COV dans la plage [0 ; 500]
- Compensation de la mesure de l'index COV en fonction de la température et du taux d'humidité relative.
- Périodicité de la mesure : toutes les secondes.
- Communication sur bus I2C et @I2C : 0x59
- Disponibilité de librairies logicielles pour environnement Arduino ou Python.

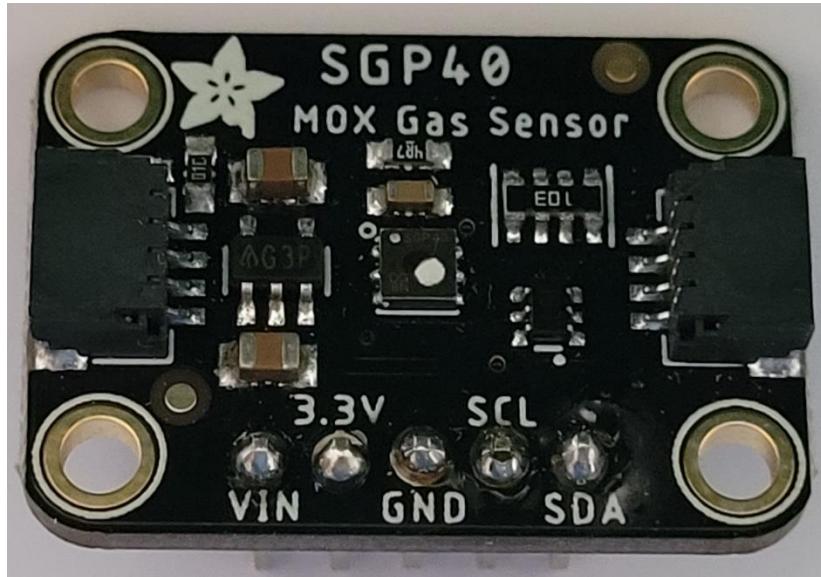


Figure 5: Shield Adafruit pour SGP40

Sa mise en œuvre sera réalisée en utilisant le shield Adafruit SGP40 ADA4829 (cf. Figure 5). La société Adafruit fourni également un guide de mise en œuvre accessible sur <https://learn.adafruit.com/adafruit-sgp40>.

La qualité de l'air vis-à-vis de la valeur de l'index COV peut être illustrée en utilisant une échelle couleur (cf. Figure 6).

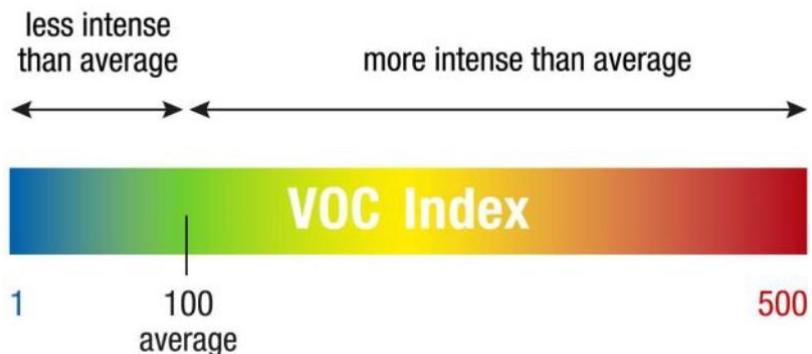


Figure 6: Echelle colorimétrique de mesure de l'index COV

3.3.3. Capteur de mesure de température – pression et taux humidité : BME280

Le capteur BME280 fabriqué par BOSCH permet de mesurer la température, le taux d'humidité relative et la pression atmosphérique. L'ensemble des ressources documentaires et logicielles sont disponibles via <https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/>.

Il permet d'effectuer la mesure des grandeurs suivantes :

- Pression atmosphérique en hPa
- Température en °C
- Taux d'humidité relative en %.
- Périodicité de la mesure : toutes les secondes.
- Communication sur bus I2C et @I2C : 0x76

Sa mise en œuvre s'effectuera en utilisant le shield Adafruit 2652 BME280 (cf. Figure 7). Un guide de mise en œuvre est disponible via le lien <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout.pdf>.



Figure 7: Shield Adafruit pour BME280

3.4. Interface utilisateur : écran DFR0665

L'interface utilisateur sera constituée par un écran TFT couleur muni d'une dalle tactile résistive Il s'agit de l'écran DFR0665 de la société DFRobot (cf. Figure 8) et (cf. <https://www.dfrobot.com/product-2106.html>).



Figure 8: Ecran TFT et dalle tactile DFR0665 – vue de dessus

Ses principales caractéristiques sont :

- Ecran LED couleur de diagonale de 2.8 inch (1 inch = 2.54cm).
- Résolution 320 x 240.
- Dalle tactile résistive.
- Slot pour carte µSD.

- Bus de communication : SPI⁵
- Disponibilité de librairies logicielles pour environnement Arduino ou Python.

Les principaux éléments concernant sa mise en œuvre sont disponibles sur https://wiki.dfrobot.com/2.8inches_320_240_TFT_LCD_Touchscreen_SKU_DFR0665.

Remarque : Une alternative à l'utilisation de la dalle tactile sera abordée en utilisant des boutons poussoirs dont les fonctions seront détaillées ultérieurement.

4. Première mise en œuvre de la plateforme microcontrôleur Raspberry Pi Pico

L'exercice de base consiste à faire clignoter une Led. Cela permet de comprendre comment on interagit avec une ressource matérielle simple ainsi que d'aborder la gestion du temps (pour clignoter). De plus, la structure générale du programme sera similaire pour tous les autres programmes. Donc en plus de vérifier que tout fonctionne, ce premier programme fournit un modèle pour tous les programmes qui vont suivre.

La programmation s'effectue en MicroPython⁶, un sous ensemble de Python, spécifique à la programmation Python sur cible microcontrôleur. La documentation de référence pour la version de MicroPython v1.18 est disponible sur <https://docs.micropython.org/en/v1.18/> (lien valide au 28.06.2022). Des mises à jour périodiques de MicroPython sont proposées régulièrement. L'accès à la documentation de la dernière version se fait par : <https://docs.micropython.org/en/latest/>. (au 28.06.2022, version de MicroPython : v1.19.1).

4.1. Structure générale d'un programme

Le code des programmes que nous produirons posséderont tous la même structure en 2 blocs de code :

1. Initialisations (variables, entrées-sorties...)
2. Boucle infinie (perception, analyse, ...)

L'ensemble du code est obligatoirement regroupé dans le fichier *main.py*.

De façon plus détaillée, chaque bloc contiendra (liste non exhaustive) :

- Bloc Initialisation :
 - o Inclusion des bibliothèques issues de MicroPython, ou spécifiques à l'architecture matérielle du microcontrôleur ou bien propres à l'utilisateur.
 - o Déclaration des variables globales.
 - o Définition de fonctions.
 - o Initialisation des ressources matérielles utilisées sur le microcontrôleur.
 - o Initialisation des ressources logicielles propres à l'application, ...
- Boucle infinie :
 - o Lecture des mesures issues des capteurs.
 - o Rafraîchissement de l'affichage des mesures.
 - o Prise en compte de la gestion de la dalle tactile.
 - o Gestion d'événements temporels,

⁵ https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface.

⁶ <https://fr.wikipedia.org/wiki/MicroPython>.

4.2. Faire clignoter une Led et gérer sa luminosité

La carte microcontrôleur possède une Led, connectée sur le port GPIO 25. Il s'agit de la faire clignoter.

4.2.1. Initialisation des ressources

L'utilisation de ressources matérielles, en particulier les broches d'E/S ou GPIO (*Pin*), repose sur l'utilisation de la librairie (ou bibliothèque) *machine* dont il faut importer le code. Le code python correspondant sera :

```
from machine import Pin
```

Pour la gestion du temps, il faut importer la librairie *time* par :

```
import time
```

Afin de gérer les différentes broches qui seront utilisées par la suite, le fichier utilisateur *ConfigMateriel_pico.py* contient des déclarations explicites de variables (au sens où le choix de l'identifiant est le plus explicite possible). Il faut donc l'inclure de la même façon que les librairies python :

```
from ConfigMateriel_pico import *
```

Notez que ces deux écritures permettent d'inclure une bibliothèque dans le programme : cela signifie que l'on peut ensuite faire appel à des fonctions qui sont écrites dans ces bibliothèques. La présentation différente de ces instructions (qui ont donc le même but) a des conséquences :

- La première : **from ... import** permet de ne retenir que l'élément *Pin* de la bibliothèque *machine* (il y a d'autres éléments dans cette librairie mais nous n'allons pas y faire appel). Cette façon d'inclure une partie de la librairie *machine* permet d'appeler, dans toute la suite du programme, directement *Pin* (sans avoir à préciser que cela fait partie de *machine*) ;
- La deuxième : **import ...** permet d'inclure toute la librairie *time*. Cependant, il faudra préciser que les éléments utilisés dans cette librairie y sont attachés.

La gestion de la broche (ou pin) GPIO25 nécessite de définir son mode d'utilisation. Comme on pilote la Led via cette broche, il faut indiquer que c'est une sortie.

Il faut créer la variable *Led_25* (qui sera un objet *Pin*) par :

```
Led_25 = Pin (Led_Pin_25, Pin.OUT)
```

Puis définir son état initial avec la méthode *value* :

```
Led_25.value(0)
```

Puis on attend 0.25s

```
time.sleep(0.25)
```

Cette instruction `sleep()` est disponible depuis la librairie `time` incluse plus haut. On remarque que la bibliothèque ayant été incluse par la forme `import ...`, il est nécessaire, pour atteindre l'instruction, d'écrire `time.sleep()`.

4.2.2. Boucle infinie

Une fois les initialisations terminées, les variables et données nécessaires sont prêtes. Le microcontrôleur rentre alors dans un cycle (ou une boucle) qui ne s'arrêtera que lorsque qu'il ne sera plus alimenté. Un appui sur le bouton poussoir « Run » relance également l'exécution du programme. Pour obtenir ce comportement, une boucle qui ne s'arrête jamais, on peut utiliser l'instruction `while` en lui donnant une condition d'arrêt qui ne sera jamais fausse. Par exemple, on peut utiliser `True` qui, sans surprise, prend toujours la valeur Vrai :

```
while True:  
    # instructions
```

Pour faire clignoter une led, il faut l'allumer, attendre, l'éteindre, attendre, l'allumer, attendre, l'éteindre, ...

Cela peut se faire à l'aide de la suite d'instructions suivantes :

```
Led_25.value(1) # Allumer la led  
time.sleep (1)  
Led_25.value(0) # Eteindre la led  
time.sleep (0.5)
```

En rajoutant ce bloc de code dans la boucle infinie, tout en respectant l'indentation, alors le cycle de gestion de la Led sera exécuté indéfiniment.

```
while True:  
    Led_25.value(1) # Allumer la led  
    time.sleep (1)  
    Led_25.value(0) # Eteindre la led  
    time.sleep (0.5)
```

Le programme final, augmenté de commentaires et contenu dans le fichier `main.py`, est disponible ci-après.

4.2.3. Exécution du programme sur la carte microcontrôleur

L'écriture du code source, le chargement des librairies utilisateur et l'exécution du programme sont gérés depuis l'IDE Thonny. La gestion de la mémoire de programme du microcontrôleur obéit à des règles qu'il est impératif de respecter :

- Les librairies utilisateur sont placées dans le répertoire `lib`.
- Le fichier `main.py` est placé en dehors de tout répertoire, à la racine.

Etape 1 : Ecrire le code source dans l'IDE Thonny (cf. Figure 9)

The screenshot shows the Thonny IDE interface. At the top, there's a toolbar with icons for file operations like new, open, save, and run. Below the toolbar is a menu bar with 'Fichier', 'Édition', 'Affichage', 'Exécuter', 'Outils', and 'Aide'. On the left is a 'Fichiers' (Files) sidebar showing the directory structure of the current project. In the center is the main workspace divided into three panes: 'main.py' (code editor), 'Console' (terminal output), and a status bar at the bottom. The code editor pane contains the following Python script:

```

1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

```

The 'Console' pane shows the message: '>>> Backend terminated or disconnected. Use 'Stop/Restart' to restart.'

Figure 9 : Ecriture du code source dans l'éditeur de Thonny

Etape 2 : Le sauver sur disque au sein du fichier *main.py*.

Fichier → Enregistrer sous...

Etape 3 : Connecter la carte microcontrôleur sur un port USB. Activer la connexion avec l'interpréteur de commande. Cela permet d'ouvrir le terminal série en activant le port COM dédié à la communication avec la carte Raspberry Pi Pico. Cela se traduit dans la console par l'apparition de >>> (cf. Figure 10). De plus, dans la fenêtre « Fichiers » de l'IDE Thonny, la carte microcontrôleur apparaît.

The screenshot shows the Thonny IDE interface. At the top, there is a toolbar with icons for file operations like new, save, and run. Below the toolbar, there are two tabs: "Fichiers" and "main.py". The "main.py" tab is active, displaying the following Python code:

```

1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

```

Below the code editor is a "Console" tab which shows the MicroPython prompt and the message: "Backend terminated or disconnected. Use 'Stop/Restart' to restart." The console also displays the MicroPython version and information about the connection.

Figure 10 : Activer le terminal série de l'IDE Thonny

Etape 4 : Créer dans la zone mémoire du microcontrôleur le répertoire *lib*. Positionner le curseur souris dans la fenêtre Raspberry Pi pico puis clic droit souris (cf. Figure 11). Après validation, le répertoire *lib* apparaît dans la zone Raspberry Pi Pico (cf. Figure 12).

The screenshot shows the Thonny IDE interface. At the top, there's a toolbar with icons for file operations like Open, Save, and Run. Below the toolbar are two tabs: "Fichiers" and "main.py". The "Fichiers" tab shows a tree view of the local drive, with "Cet ordinateur" expanded to show various folders like .atom, .dbus-keyrings, jsc, matplotlib, oracle_jre_usage, .vscode, AppData, Application Data, Contacts, Cookies, Favorites, and IntelGraphicsProfiles. The "Raspberry Pi Pico" tab is also visible. The main code editor window contains the following Python script:

```

1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

```

Below the code editor is a "Console" window showing MicroPython output:

```

>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>>

```

A modal dialog box titled "Nouveau dossier" (New folder) is open in the foreground, prompting the user to enter a name for a new folder in the root directory. The input field contains "lib". There are "OK" and "Annuler" (Cancel) buttons at the bottom.

Figure 11 : Création du répertoire lib (1)

Thonny - D:\Documents\PEIP2_S3_Option_Info\Programme\Version finale\Led_25\main.py @ 21 : 1

Fichier Édition Affichage Exécuter Outils Aide

```

Fichiers ✎
Cet ordinateur
C:\Users\gaucher
Raspberry Pi Pico
    lib

main.py ✎
1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

Console ✎
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>>

```

Figure 12 : Création du répertoire lib (2)

Etape 5 : Ouvrir dans l'IDE le fichier *ConfigMateriel_pico.py* (cf. Figure 13) et (cf. Figure 14).

Fichier → Ouvrir et sélectionner « Cet Ordinateur » puis naviguer dans l'explorateur Windows et sélectionner le fichier *ConfigMateriel_pico.py*.

Thonny - D:\Documents\PEIP2_S3_Option_Info\Programme\Version finale\Led_25\main.py @ 21 : 1

Fichier Édition Affichage Exécuter Outils Aide

```

Fichiers ✎
Cet ordinateur
Raspberry Pi Pico
    lib

main.py ✎
1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

Console ✎
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>>

Where to open from? ✎
Cet ordinateur
Raspberry Pi Pico

```

Figure 13 : Ouvrir *ConfigMateriel_pico.py* (1)

Thonny - D:\Documents\PEIP2_S3_Option_Info\lib\ConfigMateriel_pico.py @ 37 : 1

Fichier Édition Affichage Exécuter Outils Aide

```

main.py ConfigMateriel_pico.py
1 # Ressources GPIO du microcontrôleur Raspberry Pi pico
2
3 from machine import Pin
4
5 Led_Pin_25 = 25 # Variable Led_Pin_25 pour adresser la broche GP25 du µC RP2040
6
7 # Ressources pour communication I2C
8 #   - SCL : GP9 soit broche 12 PCB
9 #   - SDA : GP8 soit broche 11 PCB
10 #   - Fréquence : 400KHz
11 # Correspond à I2C0
12 SDA_pin = Pin(8)
13 SCL_pin = Pin(9)
14 Freq_i2c = 400000
15
16 # Ressources pour gestion écran TFT : SPI0
17 # GPIO par défaut : bus SPI
18 #   SPI_SCK_pin = 6 # GP6 soit broche 9 du PCB; CLK
19 #   SPI_TX_pin = 7 # GP7 soit broche 10 du PCB; MOSI
20 #   SPI_RX_pin = 4 # GP4 soit broche 6 du PCB; MISO
21
22 # Spécifique à l'écran TFT Touchscreen DFR0665
23 SPI_CS_pin = Pin(5) # GP5 soit broche 7 du PCB; A gérer explicitement; A mettre à 1
24 TFT_DC_pin = Pin(2) # GP2 soit broche 4 du PCB. Permet de gérer envoi data ou commande vers écran TFT
25 TFT_RESET_pin = Pin(13) # GP13 soit broche 17 du PCB
26 TFT_Backlite_pin = Pin(3) # GP3 soit broche 5 du PCB

```

Console

```

>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /

```

Figure 14 : Ouvrir ConfigMateriel_pico.py (2)

Etape 6 : Sauver le fichier *ConfigMateriel_pico.py* dans le répertoire *lib* (cf. Figure 15), (cf. Figure 16) et (cf. Figure 17).

Thonny - D:\Documents\PEIP2_S3_Option_Info\lib\ConfigMateriel_pico.py @ 37 : 1

Fichier Édition Affichage Exécuter Outils Aide

Console

```

>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>>

```

Where to save to?

Cet ordinateur

Raspberry Pi Pico

Figure 15 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (1)

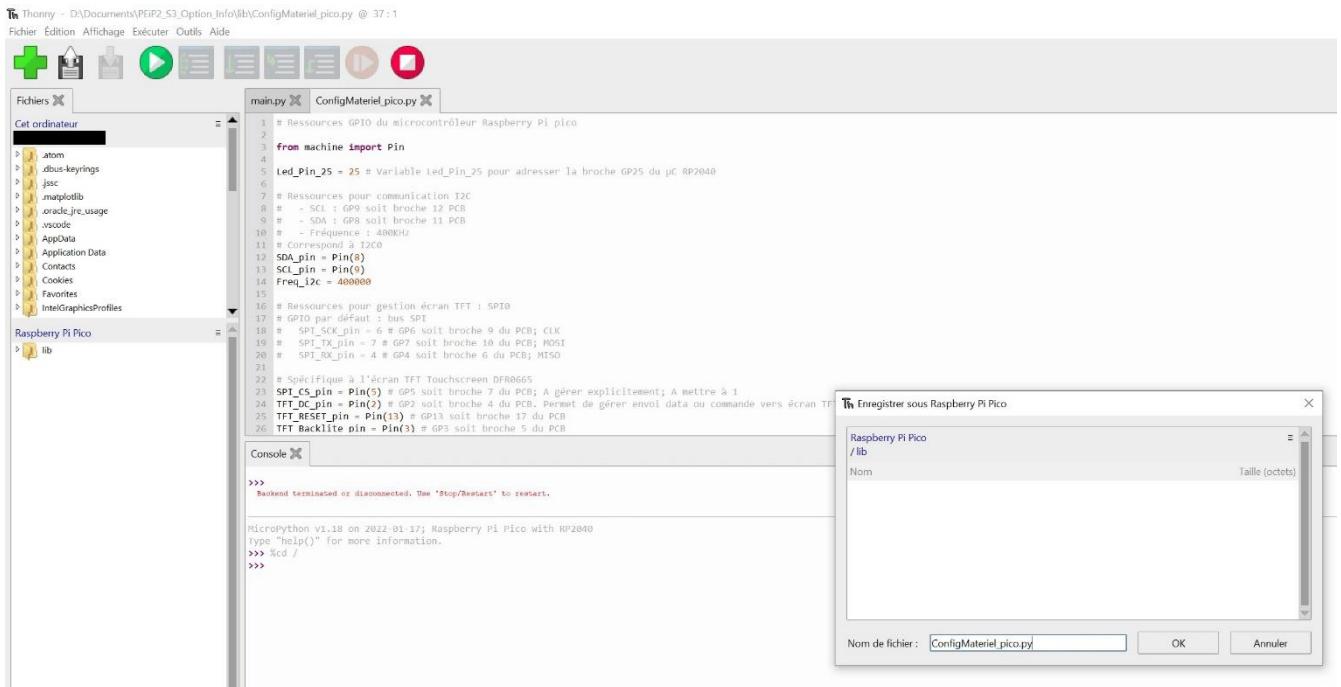


Figure 16 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (2)

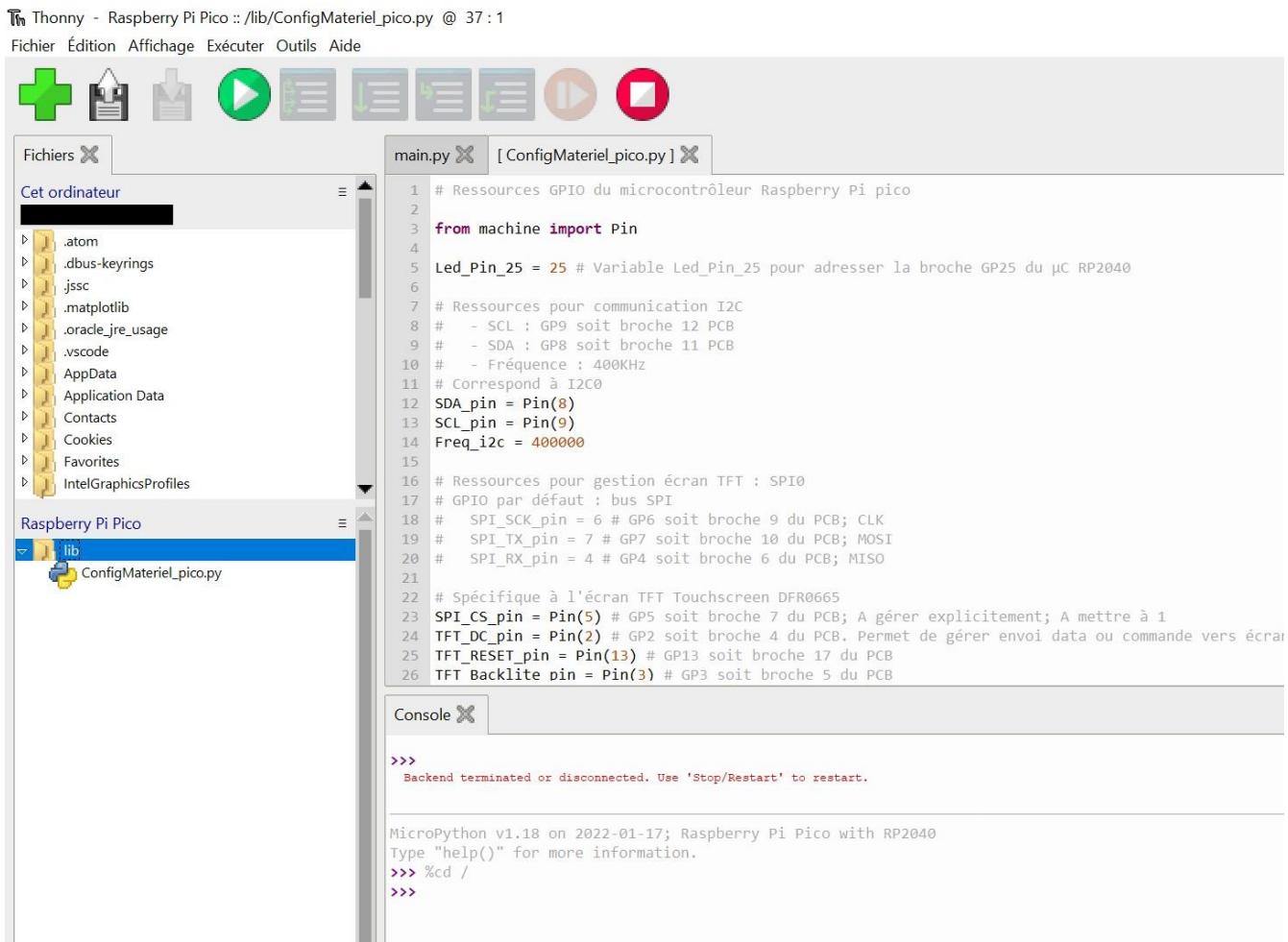
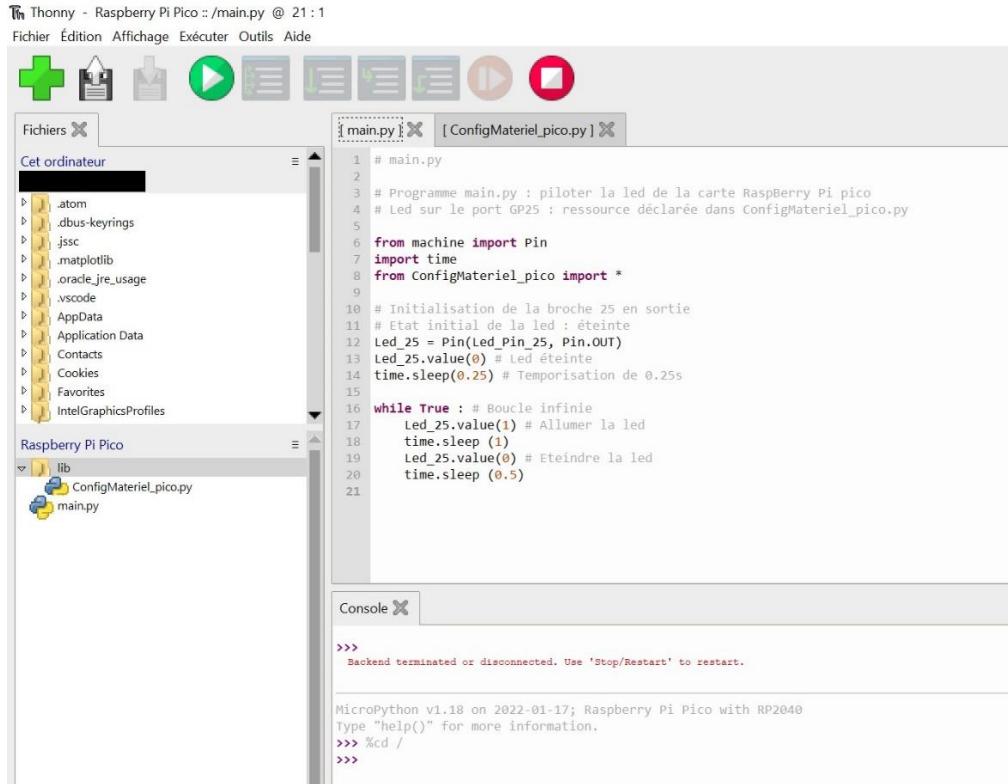


Figure 17 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (3)

Etape 7 : Sauver le fichier *main.py* dans la mémoire du microcontrôleur, en dehors du répertoire *lib*.
 Sélectionner au préalable le fichier *main.py* avec son onglet dans la zone de l'éditeur (cf. Figure 18) et (cf. Figure 19).



The screenshot shows the Thonny IDE interface. The top menu bar includes Fichier, Édition, Affichage, Exécuter, Outils, and Aide. Below the menu is a toolbar with icons for new file, open file, save file, run, stop, and exit. The left sidebar shows a file tree for 'Cet ordinateur' and 'Raspberry Pi Pico'. Under 'Raspberry Pi Pico', the 'lib' folder is expanded, showing 'ConfigMateriel_pico.py' and 'main.py'. The main area contains two tabs: '[main.py]' and '[ConfigMateriel_pico.py]'. The '[main.py]' tab is active, displaying the following Python code:

```

1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
21

```

Below the editor is a 'Console' tab showing MicroPython v1.18 output:

```

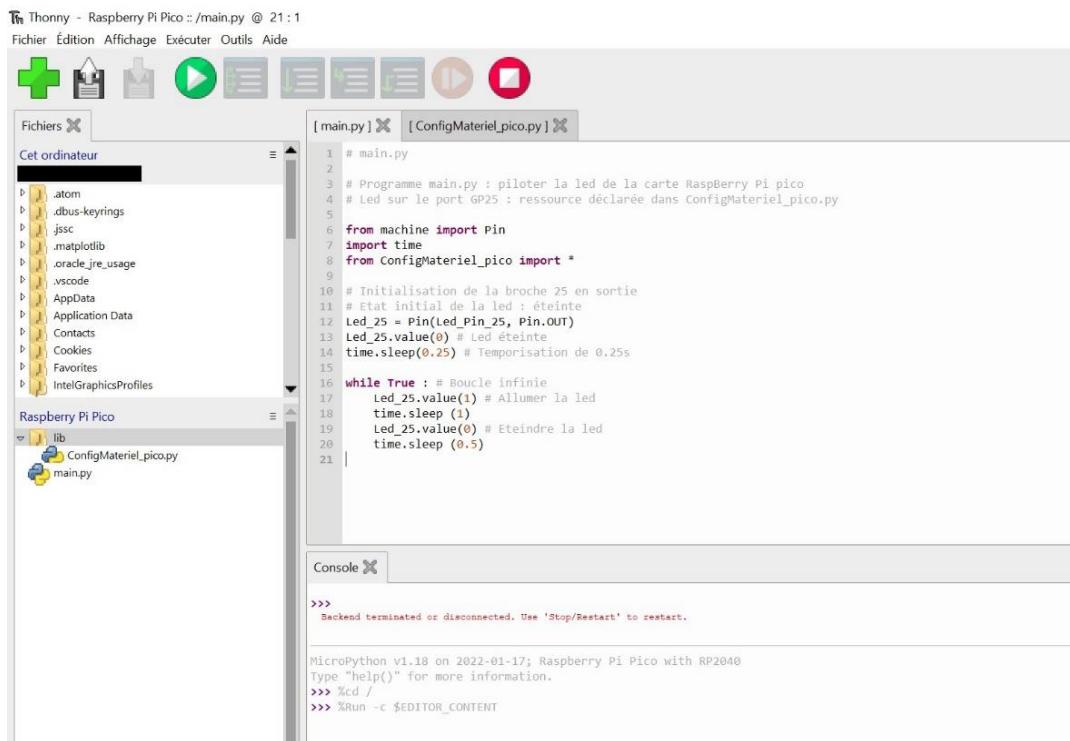
>>> Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>>

```

Figure 18 : Charger *main.py* dans la mémoire du microcontrôleur

Etape 8 : Lancer l'exécution du programme correspondant au fichier *main.py* en exécutant le script courant (cf. Figure 19).



This screenshot is identical to Figure 18, showing the Thonny IDE interface with the 'main.py' tab active in the editor. The code is the same as in Figure 18. The console output at the bottom is also identical, showing the MicroPython version and a command prompt.

Figure 19 : Lancement de *main.py*

Vérifier que l'exécution de votre programme est correcte sur la carte microcontrôleur.

Le fichier *main.py* ayant été sauvegardé dans la mémoire du microcontrôleur, il peut alors s'exécuter indépendamment de l'IDE Thonny en appuyant sur le bouton « Run » du PCB. Cela provoque néanmoins la perte de la connexion USB gérée sous l'IDE Thonny.

Remarque : pour stopper l'exécution du programme, cliquer dans la console puis appuyer sur Ctrl + C. Un message s'affiche pour signaler une interruption du programme via le clavier et l'on retrouve le prompt >>> de la console (cf. Figure 20). Vous devez constater que la led ne clignote plus.

The screenshot shows the Thonny IDE interface. The top menu bar includes Fichier, Édition, Affichage, Exécuter, Outils, and Aide. The toolbar features icons for file operations like new, open, save, and run. The left sidebar shows a file tree with 'Cet ordinateur' and 'Raspberry Pi Pico' sections. In the 'Raspberry Pi Pico' section, 'lib' contains 'ConfigMateriel_pico.py' and 'main.py'. The main workspace displays the code for 'main.py':

```
1 # main.py
2
3 # Programme main.py : piloter la led de la carte Raspberry Pi pico
4 # Led sur le port GP25 : ressource déclarée dans ConfigMateriel_pico.py
5
6 from machine import Pin
7 import time
8 from ConfigMateriel_pico import *
9
10 # Initialisation de la broche 25 en sortie
11 # Etat initial de la led : éteinte
12 Led_25 = Pin(Led_Pin_25, Pin.OUT)
13 Led_25.value(0) # Led éteinte
14 time.sleep(0.25) # Temporisation de 0.25s
15
16 while True : # Boucle infinie
17     Led_25.value(1) # Allumer la led
18     time.sleep (1)
19     Led_25.value(0) # Eteindre la led
20     time.sleep (0.5)
```

The bottom pane is the 'Console' window, which shows the command line interface:

```
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 18, in <module>
KeyboardInterrupt:
>>> |
```

Figure 20 : Arrêt exécution programme par Ctrl + C

Remarque : Gestion des modifications de code et stockage

- Lorsqu'un fichier a été sauvé dans la mémoire du microcontrôleur, toute modification du code source s'effectue vers la mémoire du µC.
- Si le microcontrôleur n'est plus alimenté, le contenu des fichiers sauvés dans sa mémoire est conservé.
- Il faut toujours sauver le contenu des fichiers de la mémoire du µC sur disque de façon explicite.

5. Gérer la luminosité de la Led

Dans le programme précédent, la led est pilotée de façon binaire : éteinte ou allumée. Nous allons maintenant gérer sa luminosité en utilisant un signal de commande PWM.

5.1. Principe de la PWM

Un signal PWM (Pulse Width Modulation) est un **signal périodique** dont le **rapport cyclique varie**. Cela permet alors de récupérer une grandeur analogique correspondant à la valeur moyenne du signal sur une période. Le rapport cyclique correspond au rapport T_h/T (T_h : durée pendant laquelle le signal est à l'état haut, T : période du signal PWM), et la valeur moyenne correspondant du signal PWM est alors égale à l'amplitude maximale du signal divisés par le rapport cyclique (cf. Figure 21) (Source : https://en.wikipedia.org/wiki/Pulse-width_modulation#/media/File:Duty_Cycle_Examples.png).

La valeur du rapport cyclique évolue dans l'intervalle de valeurs [0.0 , 1.0] (il est parfois exprimé en % soit [0.0 , 100.0] %).

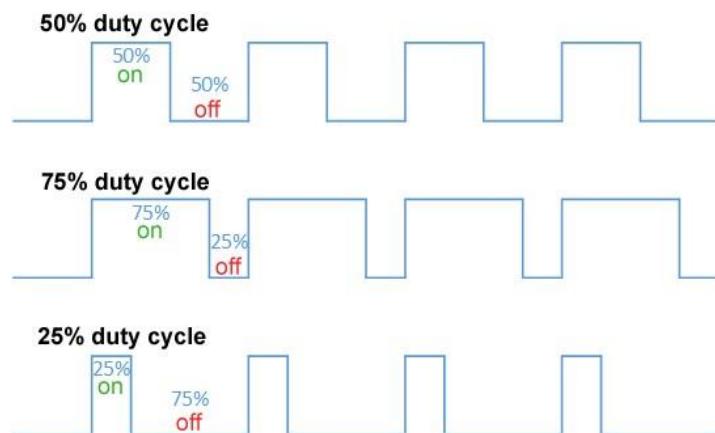


Figure 21 : Signal PWM - principe

C'est ce principe qui permet de faire varier la tension de commande de la led, et donc sa luminosité.

5.2. Mise en œuvre

La documentation MicroPython concernant la mise en œuvre de la PWM est accessible sur <https://docs.micropython.org/en/latest/rp2/quickref.html#pwm-pulse-width-modulation>.

Nous disposons de la librairie *PWM*, au sein de la bibliothèque *machine*, qui permet de gérer un signal PWM. Il est donc nécessaire d'inclure cette librairie :

```
from machine import Pin, PWM
```

Il faut ensuite spécifier sur quelle GPIO (ou broche) de la carte microcontrôleur le signal PWM sera généré. Ici, il s'agit toujours de la broche associée à la led connectée sur le port GPIO 25.

```
pwm_Led_25 = PWM(Pin(Led_Pin_25))
```

Une fois cette initialisation effectuée, il faut maintenant définir la fréquence du signal PWM, soit ici 1000Hz.

```
pwm_Led_25.freq(1000) # Fréquence du signal pwm en Hz
```

Enfin, l'état initial de la led est défini en spécifiant la valeur du rapport cyclique, ici 0, ce qui entraîne que la led est éteinte.

```
pwm_Led_25.duty_u16(0) # Led éteinte, rapport cyclique = 0.
```

Remarque : la valeur du rapport cyclique est définie sous forme numérique par un paramètre de type entier non signé sur 2 octets. Aussi ce paramètre peut prendre toute valeur dans l'ensemble {0, 1, ..., 65535} (soit $2^{16}-1$ pour la valeur max) :

- 0 : PWM = 0
- 65535 : PWM = 1.0

Lorsque l'ensemble des initialisations sont achevées, il est alors possible d'écrire la portion de code permettant de faire varier l'intensité lumineuse de la led au sein de la boucle **while True** :

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De faire varier l'intensité lumineuse de la led depuis l'état éteint jusqu'à ce qu'elle soit allumée avec l'intensité lumineuse max.
- Faire ensuite décroître l'intensité lumineuse de la led jusqu'à son extinction.

Remarque : le principe de PWM mise en œuvre ici permettra par la suite de contrôler le niveau du rétroéclairage de l'écran.

6. Gérer des événements temporels - timer

On souhaite gérer des événements, selon des périodicités qui peuvent être différente d'un événement à l'autre. Par exemple, on veut acquérir une mesure de la température toutes les secondes et une mesure du taux de CO₂ toutes les 5 secondes. Dans ce cas, l'utilisation de la méthode *sleep* de la librairie *time* n'est pas adaptée. En effet, lorsque *sleep* est appelée, la suite des instructions placées après ne peut s'exécuter que lorsque la durée passée en paramètre de la fonction *sleep* s'est écoulée. Cela peut donc retarder l'exécution de certaines portions de code.

Afin de s'affranchir de ces limites, nous pouvons utiliser une ressource matérielle particulière du microcontrôleur : les **timers**. Associé aux timers, nous utiliserons un mécanisme d'**interrupt** lié au temps, en manipulant des **alarmes** associées à nos timers.

La documentation MicroPython associée aux timers est accessible sur <https://docs.micropython.org/en/latest/library/machine.Timer.html#machine.Timer>.

Le programme ci-dessous illustre ces mécanismes. Il s'agit de gérer 2 événements distincts selon des périodicités différentes. La première partie du programme contient l'ensemble des initialisations (cf. Figure 22). Quant à la seconde, elle montre comment, au sein de la boucle **while... True**, par la gestion de 2 flags, chaque événement est géré selon une périodicité qui lui est propre (cf. Figure 23). Ainsi, l'événement 1 est géré toutes les secondes tandis que l'événement 2 l'est toutes les 5 secondes.

L'exécution du code conduit à des sorties au sein de la console permettant de vérifier la périodicité de gestion de chacun des événements (cf. Figure 24)

Vous pouvez récupérer le code correspondant à l'illustration de la mise en œuvre des alarmes dans le fichier *main.py* au sein du répertoire *Gestion_Timer_Alarme* de l'archive logicielle mise à votre disposition.

```
1  # Programme d'exemple de gestion temporelle d'événements
2  # Ce programme illustre l'utilisation des timers, associée à la notion d'alarme
3  # Notions complémentaires :
4  #   - Interruption timer;
5  #   - Routine d'interruption
6  #   - Gestion de flags
7  # Événement 1 : périodicité de prise en compte : 1s
8  # Événement 2 : périodicité de prise en compte : 5s
9
10 # Cible Raspberry Pi pico
11 # Micropython V1.18
12
13 from machine import Timer
14 from micropython import const
15
16 Evenement1_flag = False
17 Evenement2_flag = False
18
19 _1s = const(1000) # Correspond à 1000 ms, soit 1s
20 _5s = const(5000) # Correspond à 5000 ms, soit 5s
21
22 # Fonction callbak
23 def Evenement1_callback (self):
24     global Evenement1_flag
25     Evenement1_flag = True
26 def Evenement2_callback (self):
27     global Evenement2_flag
28     Evenement2_flag = True
29
30 # Init des timers
31 Evenement1_timer = Timer()
32 Evenement2_timer = Timer()
33 Evenement1_timer.init(period = _1s, mode = Timer.PERIODIC, callback = Evenement1_callback)
34 Evenement2_timer.init(period = _5s, mode = Timer.PERIODIC, callback = Evenement2_callback)
35
36 Index_evenement_1 = 1
37 Index_evenement_2 = 1
```

Figure 22: Interruption Timer (1)

```
36
37 while True :
38     # Gestion de l'événement 1
39     if Evenement1_flag == True :
40         # Traitement de l'événement 1
41         print ('-----')
42         print (" Index_evenement_1 : ", Index_evenement_1)
43         print (" Evenement 1 en cours de traitement")
44         Evenement1_flag = False
45         Index_evenement_1 += 1
46
47     # Gestion de l'événement 2
48     if Evenement2_flag == True :
49         # Traitement de l'événement 2
50         print ('-----')
51         print (" Index_evenement_2 : ", Index_evenement_2)
52         print (" Evenement 2 en cours de traitement")
53         Evenement2_flag = False
54         Index_evenement_2 += 1
```

Figure 23 : Interruption Timer (2)

```

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

-----
Index_evenement_1 : 1
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 2
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 3
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 4
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 5
Evenement 1 en cours de traitement
-----
Index_evenement_2 : 1
Evenement 2 en cours de traitement
-----
Index_evenement_1 : 6
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 7
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 8
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 9
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 10
Evenement 1 en cours de traitement
-----
Index_evenement_2 : 2
Evenement 2 en cours de traitement
-----
Index_evenement_1 : 11
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 12
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 13
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 14
Evenement 1 en cours de traitement
-----
Index_evenement_1 : 15
Evenement 1 en cours de traitement
-----
Index_evenement_2 : 3
Evenement 2 en cours de traitement
-----
Index_evenement_1 : 16
Evenement 1 en cours de traitement
-----
```

Figure 24 : Interruption Timer (3)

7. Acquérir la température, l'humidité et la pression atmosphérique

L'acquisition des mesures de température, pression atmosphérique et taux d'humidité s'effectuent à l'aide du capteur Bosch BME280, monté sur un shield Adafruit 2652. Par la suite, nous désignerons indifféremment le capteur ou le shield par BME280.

L'échange des données entre le capteur BME280 et le microcontrôleur s'appuiera sur l'utilisation du bus de communication I2C

7.1. Connectique du shield BME280

Le shield BME280 (cf. Figure 25) possède une connectique de 7 broches, repérées comme suit :



Figure 25 : Shield BME280 : broches

- Alimentation électrique :
 - o Vin : alimentation de la carte. La carte accepte des tensions comprises entre 3V et 5V grâce à la présence d'un régulateur (qui régule à 3.3V pour le capteur).
 - o 3.3V : sortie régulée à 3.3V.
 - o GND : masse.
- Communication I2C :
 - o SCK : horloge de synchronisation du bus I2C
 - o SDO : non utilisée en I2C
 - o SDI : ligne de données, correspondant aussi à SDA.
 - o CS : non utilisée en I2C

Les broches SDO et CS sont utilisées lorsque le capteur utilise un autre protocole de communication : SPI (l'usage de ce protocole sera détaillé lors de la mise en œuvre de l'écran). Donc pour utiliser le capteur avec le protocole de communication I2C, nous avons besoin de connecter les broches Vin, GND, SCK et SDI.

Pour connecter physiquement le capteur BME280 à la carte microcontrôleur, il suffit d'insérer le shield à l'emplacement prévu à cet effet sur le circuit imprimé (cf. Figure 26).



Figure 26 : Mise en place capteur BME280 sur PCB

7.2. Mise en œuvre logicielle

7.2.1. Mise en place de la librairie logicielle

Du fait de la complexité de mise en œuvre du capteur BME280, nous utiliserons une bibliothèque qui permettra :

- D'initialiser les ressources matérielles du capteur.
- De définir son mode d'utilisation.
- D'accéder aux méthodes permettant d'effectuer l'acquisition des mesures.

Cette bibliothèque est disponible au sein du fichier **BME280.py** qu'il faut enregistrer dans le répertoire *lib* de la mémoire du microcontrôleur.

- Etape 1 : Ouvrir le fichier BME280.py dans l'IDE Thonny (cf . Figure 27).
- Etape 2 : Sauver le fichier BME280 dans le répertoire lib de la mémoire du microcontrôleur (cf. Figure 28).

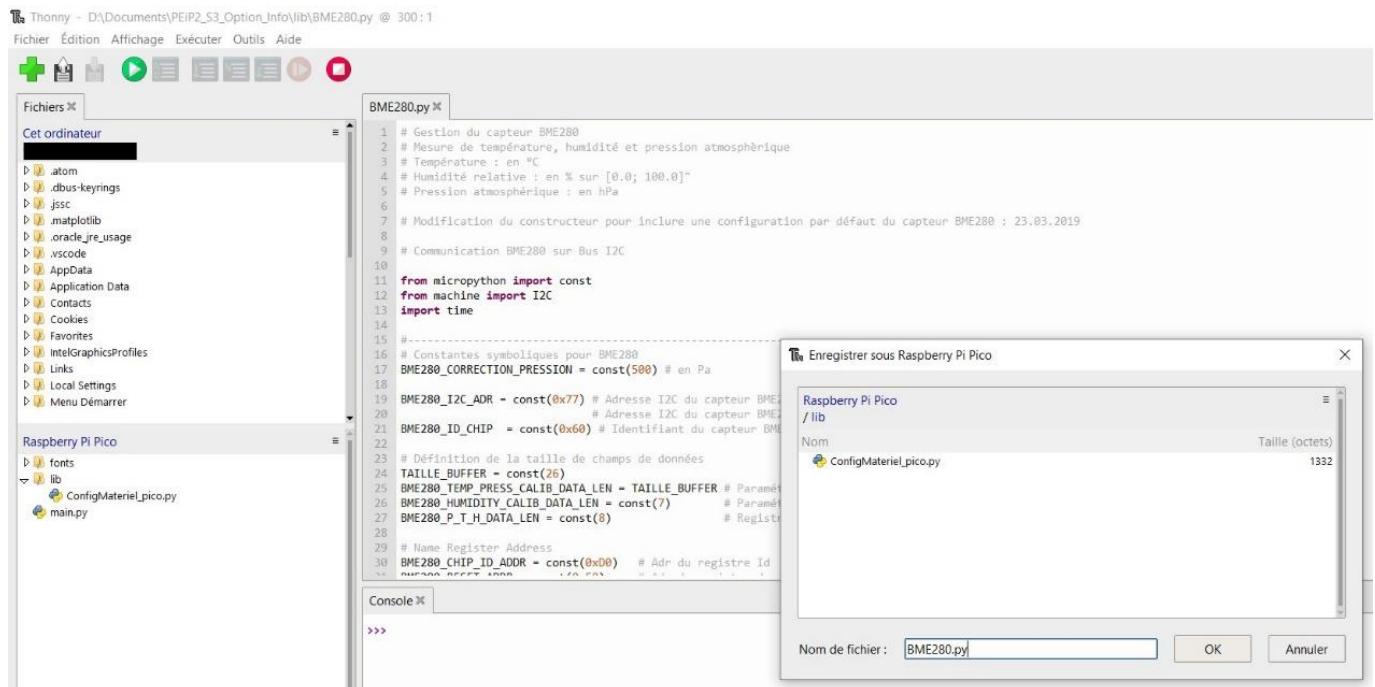


Figure 27: Ouverture du fichier BME280.py dans IDE Thonny



Figure 28 : Configuration du répertoire lib après chargement de la librairie BME280.py

7.2.2. Initialisation de la connexion I2C

Nous utiliserons les ressources matérielles par défaut de la carte microcontrôleur pour la gestion du bus I2C, soient :

- SCL : GP9 soit broche 12 PCB de la carte microcontrôleur Raspberry Pi pico.
- SDA : GP8 soit broche 11 PCB PCB de la carte microcontrôleur Raspberry Pi pico.

Elles sont déjà décrites au sein du fichier *ConfigMateriel_pico.py*.

La valeur de la fréquence d'horloge du bus I2C est fixée à 400KHz.

La bibliothèque BME280 fait elle-même appel à la librairie *machine* qui contient la définition de la classe I2C. La documentation de la prise en charge de l'I2C sur la carte microcontrôleur Raspberry Pi Pico est décrite sur <https://docs.micropython.org/en/latest/rp2/quickref.html#hardware-i2c-bus>.

La classe I2C permet de créer un objet qui représente le bus de communication I2C. Afin de le créer, tout en initialisant les ressources matérielles du microcontrôleur, on fait appel au constructeur de la classe I2C :

```
i2c = I2C(0, sda = SDA_pin, scl = SCL_pin, freq=Freq_i2c)
```

Il devient alors possible d'utiliser la méthode *scan* qui permet de savoir quels sont les périphériques I2C connectés sur le bus, et d'afficher la liste des adresses I2C des périphériques connectés :

```
adr = i2c.scan()  
print('Adresse peripherique I2C :', adr)
```

Validation : Ecrire le fichier *main.py* qui permet d'afficher dans la console l'adresse I2C du capteur BME280.

7.2.3. Lectures des données du BME280 : acquisition des mesures

La première étape consiste à charger dans le répertoire *lib* situé dans la mémoire du microcontrôleur le fichier *BME280.py*.

Au sein du fichier *main.py* précédent, il faut maintenant importer la librairie *BME280.py* :

```
from BME280 import *
```

L'étape suivante consiste à lire sur le bus I2C l'identifiant du capteur BME280. Cela se fait par la ligne suivante :

```
# Id du capteur BME280 : normalement 0x60 (soit 96 en decimal)  
Id_BME280 = i2c.readfrom_mem(BME280_I2C_ADR, BME280_CHIP_ID_ADDR, 1)
```

Cette commande lit 1 octet de la mémoire du périphérique repéré par l'adresse BME280_I2C_ADR qui est une constante (0x77) définie dans la bibliothèque BME280 pour correspondre à l'adresse I2C du capteur BME280 (119 en decimal pour le shield Adafruit BME280). La lecture commence à l'adresse BME280_CHIP_ID_ADDR (qui est aussi une constante définie dans la bibliothèque BME280 : 0xD0). Cette constante correspond à l'adresse du registre contenant l'identifiant du capteur.

Le retour de la fonction `readfrom_mem` est stocké dans la variable `Id_BME280`. Si on veut en afficher le contenu (pour vérifier que tout va bien, que notre capteur est bien celui que l'on a branché...), on peut écrire :

```
print ('Valeur Id_BME280 : ', hex (Id_BME280[0]))
```

Il reste une chose à faire avant de pouvoir lire des informations de température, pression et humidité avec le capteur : il faut récupérer ses paramètres de calibration.

Pour ce faire, nous allons d'abord créer un objet capteur associé au BME280, `capteur_BME280` qui représentera le capteur :

```
capteur_BME280 = BME280 (BME280_I2C_ADR, i2c)
```

Il est à noter que la bibliothèque masque à l'utilisateur une partie des opérations nécessaires à l'initialisation du capteur. Par exemple, les paramètres suivants permettent de configurer le mode de fonctionnement du capteur :

```
# Init du mode d'utilisation du capteur BMP280
# config_osr_p = BME280_OVERSAMPLING_16X      # Valeur de sur échantillonnage pression
# config_osr_t = BME280_OVERSAMPLING_16X      # Valeur de sur échantillonnage température
# config_osr_h = BME280_OVERSAMPLING_16X      # Valeur de sur échantillonnage humidité
# config_filter = BME280_FILTER_COEFF_2        # Coefficients filtre
# config_standby_time = BME280_STANDBY_TIME_125_MS # Délai d'inactivité
# config_mode = BME280_NORMAL_MODE            # Choix du mode de fonctionnement du capteur
(sommeil / forcé / normal)
```

L'utilisation du capteur nécessite de récupérer les paramètres de calibration « usine » par la commande :

```
capteur_BME280.Calibration_Param_Load()
```

Il est désormais possible d'effectuer la lecture des mesures de température, pression atmosphérique et taux d'humidité en utilisant les méthodes :

- Température : `read_temp()` en °C
- Température : `read_pressure()` en hPa
- Humidité : `read_humidity()` en %

Synthèse : écrire le programme (fichier `main.py`) qui permet :

- De récupérer les adresses I2C des périphériques I2C actifs et de les afficher sur la console ;
- De lire l'Id du capteur BME280 et d'afficher sa valeur sur la console ;
- D'afficher sur la console (ou le terminal série) les mesures de température, pression et taux d'humidité, avec une périodicité d'acquisition de 1s.

Vérifier que l'exécution de votre programme vous permet d'obtenir l'affichage des mesures du BME280 sur la console, après formatage des valeurs des mesures, proche de l'illustration ci-après (cf. Figure 29).

Remarque1 : bien que le capteur BME280 soit calibré en usine, les mesures fournies d'un capteur à l'autre peuvent présenter des différences de valeurs.

Remarque2 : lors de l'acquisition des mesures, il faut toujours réaliser en premier l'acquisition de la température, avant d'effectuer celle de la pression et de l'humidité.

Remarque 3 : pour cette première version, vous êtes libre de gérer la périodicité d'acquisition soit à l'aide de la méthode sleep, soit en utilisant les timers avec une alarme.

```
Console >>> %Run -c $EDITOR_CONTENT
Configuration bus I2C : begin
Configuration bus I2C : done
Adresse peripherique I2C : [89, 98, 119]
Valeur Id_BME280 : 0x60
Chargement parametres de calibration : begin
Chargement parametres de calibration : done
Index : 0
Temperature : 30.3 DegC
Pression : 1015.99 hPa
humidity : 36.8 RH
-----
Index : 1
Temperature : 30.3 DegC
Pression : 1016.01 hPa
humidity : 36.4 RH
-----
Index : 2
Temperature : 30.3 DegC
Pression : 1016.00 hPa
humidity : 36.3 RH
-----
Index : 3
Temperature : 30.3 DegC
Pression : 1016.01 hPa
humidity : 36.2 RH
-----
Index : 4
Temperature : 30.2 DegC
Pression : 1015.99 hPa
humidity : 36.1 RH
-----
Index : 5
Temperature : 30.2 DegC
Pression : 1016.00 hPa
humidity : 36.1 RH
-----
Traceback (most recent call last):
  File "<stdin>", line 57, in <module>
KeyboardInterrupt:
```

Figure 29 : Données du capteur BME280

8. Acquérir la mesure du taux de CO₂

Tout comme le capteur BME280, le capteur SCD41 de mesure du taux de CO₂ nécessite d'être initialisé avant de pouvoir accéder à ses mesures. Comme ce capteur communique sur le bus I2C, l'initialisation du bus I2C réalisée pour le capteur BME280 sera commune au capteur SDC41. Le capteur SCD41, via le shield utilisé, permet d'accéder à l'ensemble des broches nécessaires à son utilisation :

- Alimentation électrique :
 - o VDD : alimentation de la carte. La carte accepte des tensions comprises entre 3V et 5V grâce à la présence d'un régulateur (qui régule à 3.3V pour le capteur).
 - o GND : masse.
- Communication I2C :
 - o SCL : horloge de synchronisation du bus I2C
 - o SDA : ligne de données.

Sur le circuit imprimé de la plateforme matérielle, un emplacement spécifique lui est dédié (cf. Figure 30).

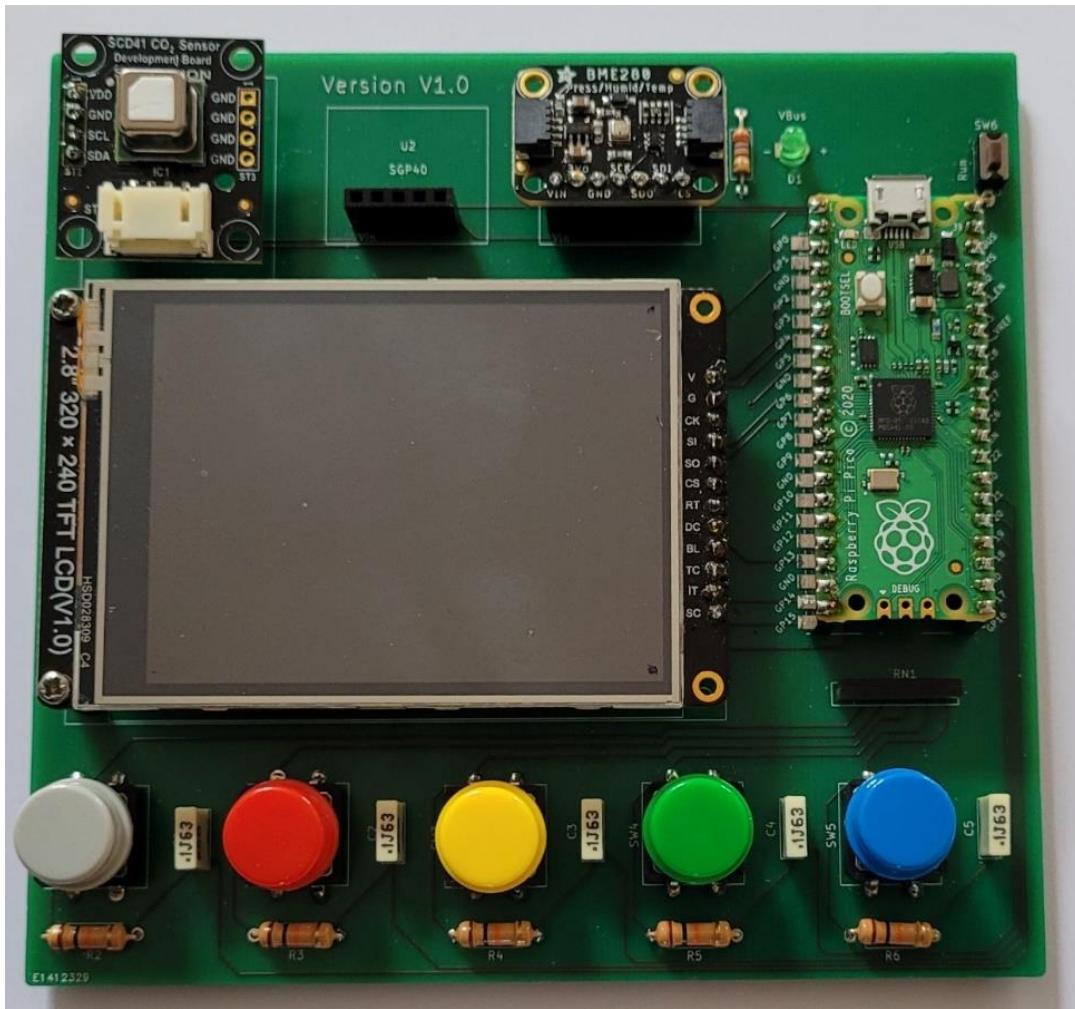


Figure 30 : Mise en place du capteur de CO₂ SCD41

8.1. Librairie logicielle

Afin de faciliter l'acquisition des mesures, nous disposons de la bibliothèque *SCD41.py*⁷ qui permet d'accéder à l'ensemble des fonctionnalités du capteur :

- Création de l'objet *scd41_capteur_co2* et initialisation de la communication I2C.
- Lecture de l'Id du capteur.
- Initialisation du mode de fonctionnement du capteur.
- Compensation de la mesure de CO₂ avec la valeur de la pression atmosphérique.
- Acquisition des mesures :
 - o Taux de CO₂
 - o Température
 - o Taux d'humidité

Comme précédemment, il faut enregistrer le fichier *SCD41.py* dans le répertoire *lib* du microcontrôleur (cf. Figure 31) et (cf. Figure 32).

⁷ La librairie *SCD41.py* fournie est dérivée de celle mise à disposition par Adafruit *adafruit_scd4x.py* https://github.com/adafruit/Adafruit_CircuitPython_SCD4X

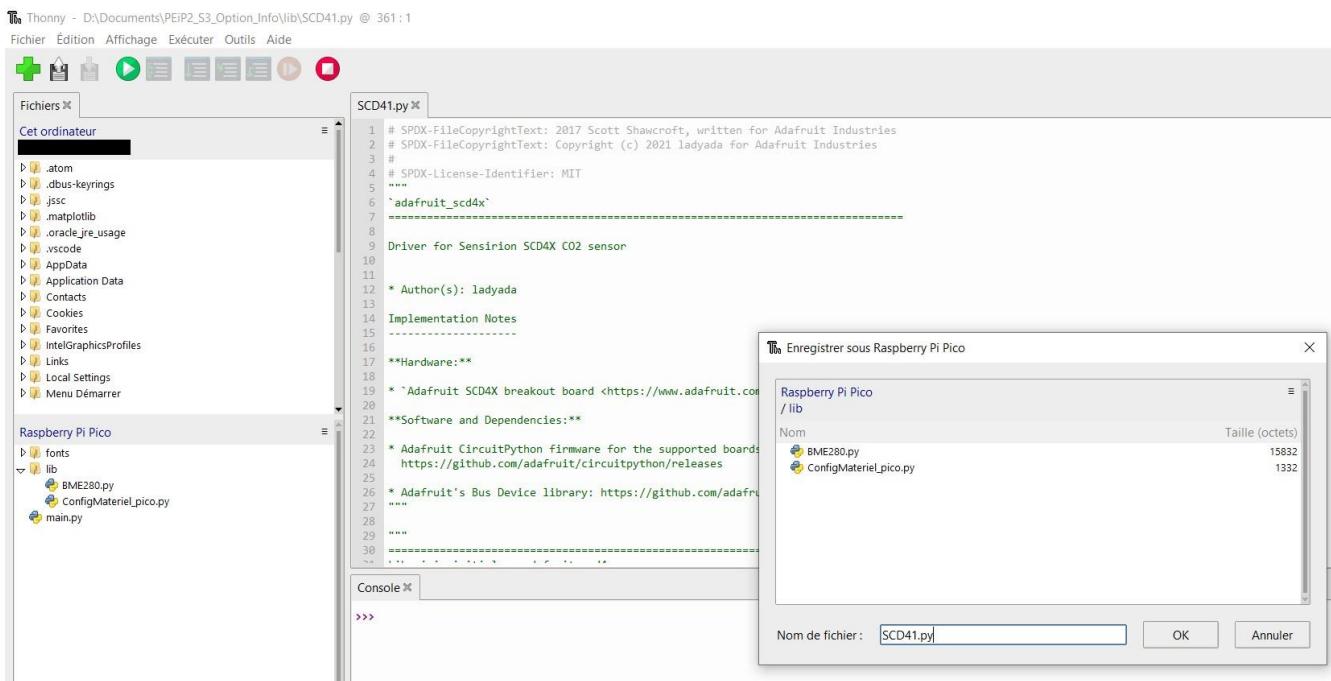


Figure 31 : enregistrement de la librairie SCD41.py dans le répertoire lib



Figure 32 : Contenu du répertoire lib après chargement de la librairie SCD41.py

En reprenant le squelette du fichier *main.py* utilisé pour valider l'utilisation du capteur BME280, nous allons écrire le programme permettant de valider celui du capteur SCD41.

Il faut importer la librairie associée au capteur SCD41 dans le fichier *main.py*:

```
from SCD41 import *
```

L'initialisation du bus de communication I2C est similaire à celle effectuée pour le capteur BME280.

8.2. Crédation de l'objet capteur SCD41 et initialisations

Nous allons créer l'objet *capteur_SCD41*, en lui associant les ressources du bus de communication I2C, et en spécifiant l'adresse I2C du capteur. Cette adresse est définie par la constante symbolique *SCD4X_DEFAULT_ADDR* au sein de la bibliothèque associée au capteur. Immédiatement après, il faut mettre le capteur en mode « arrêt ». Cela s'effectue avec la fonction *stop_periodic_measurement()*.

```
capteur_SCD41 = SCD4X (i2c, SCD4X_DEFAULT_ADDR)
capteur_SCD41.stop_periodic_measurement()
```

Afin de vérifier que la communication avec le capteur s'effectue correctement, il est possible de récupérer son numéro de série sous la forme d'une liste de 6 entiers et d'afficher leur valeur respective :

```
# Acquérir le numéro de série du capteur SCD41
print("Serial number :", [hex(i) for i in capteur_SCD41.serial_number])
```

Puis on initialise la compensation de la mesure de CO₂ avec la valeur de la pression atmosphérique exprimée en hPa :

```
# Compensation mesure taux CO2 en fonction de la pression atmosphérique
capteur_SCD41.set_ambient_pressure(int(pression_BME280))
```

Remarque : pression_BME280 est une variable initialisée avec la valeur courante de la pression atmosphérique. Il est possible de récupérer cette valeur en se connectant sur un site de prévision météo tel que : <https://www.infoclimat.fr/observations-meteo/temps-reel/tours-parcay-meslay/07240.html>

Par la suite, la compensation avec la valeur de pression atmosphérique sera réalisée à partir des données du capteur BME280, avec une périodicité d'1 heure. Cette partie sera gérée lorsque les capteurs BME280 et SCD41 seront utilisés conjointement au sein du même programme.

Enfin, on fait passer le capteur en mode d'acquisition périodique (période d'acquisition de 5s), et on attend 5s :

```
# Mode de mesure périodique
capteur_SCD41.start_periodic_measurement()
time.sleep(5)
```

8.3. Acquisition des mesures

L'acquisition de la valeur du taux de CO₂, exprimée en ppm, s'effectue au sein d'une boucle infinie, avec une périodicité de 5s. Par précaution, avant d'effectuer la lecture des mesures, nous devons nous assurer qu'une nouvelle mesure est effectivement disponible. Si c'est le cas, alors lire la mesure du taux de CO₂.

Remarque : Le capteur SCD41 permet également de mesurer la température ainsi que le taux d'humidité.

Le code implémente les fonctionnalités décrites ci-dessus :

```
# Acquisition taux CO2, température et humidité relative du SCD41
if capteur_SCD41.data_ready: # attendre qu'une mesure soit disponible
    co2 = capteur_SCD41.CO2
    temperature_SCD41 = capteur_SCD41.temperature
    humidity_SCD41 = capteur_SCD41.relative_humidity
    # Affichage des mesures SCD41

time.sleep(5000) # Pour garantir la périodicité de 5s
```

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De récupérer les adresses I2C des périphérique I2C actifs et de les afficher sur la console ;
- De lire l'Id du capteur SCD41 et d'afficher sa valeur sur la console ;
- De gérer la compensation de la mesure avec la valeur de la pression atmosphérique ;
- D'afficher sur la console (ou le terminal série) les mesures du taux de co₂, ainsi que de la température et l'humidité du capteur SCD41, avec une périodicité de 5s.

Remarque : pour cette première version, vous êtes libre de gérer la périodicité d'acquisition soit à l'aide de la méthode sleep, soit en utilisant les timers avec une alarme.

Vérifier que lors de l'exécution de votre programme, vous obtenez un affichage sur la console similaire à la figure ci-dessous (cf. Figure 33).

```
Console X

>>> %Run -c $EDITOR_CONTENT

Configuration bus I2C : begin
Configuration bus I2C : done
Adresse peripherique I2C : [89, 98, 119]
Serial number : ['0xcb', '0x41', '0xbf', '0x7', '0x3b', '0xce']
Index : 1
CO2: 702 ppm
Temperature_scd41: 32.2 °C
Humidity_scd41: 30.6 %

-----
Index : 2
CO2: 679 ppm
Temperature_scd41: 31.9 °C
Humidity_scd41: 31.1 %

-----
Index : 3
CO2: 678 ppm
Temperature_scd41: 31.6 °C
Humidity_scd41: 31.4 %

-----
Index : 4
CO2: 702 ppm
Temperature_scd41: 31.4 °C
Humidity_scd41: 31.9 %

-----
Traceback (most recent call last):
  File "<stdin>", line 58, in <module>
KeyboardInterrupt:

>>>
```

Figure 33 : Mesures du taux de CO₂ avec capteur SCD41

8.4. Gestion conjointe des capteurs BME280 et SCD41

On souhaite utiliser en même temps les capteur BME280 et SCD41. Les périodicités d'acquisition des mesures de chaque capteur restent identiques à celles utilisées précédemment :

- BME280 : toutes les 1s.
 - o Température
 - o Pression atmosphérique
 - o Taux d'humidité
- SCD41 : toutes les 5s.
 - o Taux de CO2
 - o Température (**optionnel**)
 - o Taux d'humidité (**optionnel**)

La compensation de la mesure du taux de CO2 sera réalisée à l'aide de la valeur de la pression atmosphérique acquise depuis le capteur BME280. Il sera nécessaire de tenir compte de la variation de la pression atmosphérique au cours du temps.

8.4.1. Approche 1 : utilisation de temporisation avec sleep (**facultatif**)

La compensation de la mesure du taux de CO2 avec la pression atmosphérique est effectuée en utilisant uniquement la 1^{ère} mesure de la température, réalisée lors du lancement de l'application.

Les périodicités d'acquisition propre à chaque capteur peuvent être gérées en utilisant la fonction *sleep* de la librairie *time*.

8.4.2. Approche 2 : Gestion de la compensation de la mesure du taux de CO2 (**facultatif**)

La compensation de la mesure du taux de CO2 avec la pression atmosphérique est réalisée à partir de la mesure de la pression effectuée toutes les heures. Cette périodicité d'une heure sera gérée en utilisant un timer et une alarme (cf. [6 Gérer des événements temporels - timer](#)) qui permettra de déclencher la mise à jour de la compensation de la pression pour le capteur SCD41.

Les périodicités d'acquisition propre à chaque capteur peuvent être gérées en utilisant la fonction *sleep* de la librairie *time*.

8.4.3. Approche 3 : gestion par timers et alarmes uniquement (**obligatoire**)

La compensation de la mesure du taux de CO2 avec la pression atmosphérique est réalisée à partir de la mesure de la pression effectuée toutes les heures. Cette périodicité d'une heure sera gérée en utilisant un timer et une alarme (cf. [6 Gérer des événements temporels - timer](#)) qui permettra de déclencher la mise à jour de la compensation de la pression pour le capteur SCD41.

Les périodicités d'acquisition propre à chaque capteur sont également gérées en utilisant les timers et les alarmes.

A l'issue de l'exécution de votre application, vous devez obtenir un résultat proche de la sortie sur la console, telle que présenté ci-après (cf. Figure 34).

```

Console X

>>> %Run -c $EDITOR_CONTENT

Configuration bus I2C : begin
Configuration bus I2C : done
Adresse périphérique(s) I2C : ['0x59', '0x62', '0x77']
Valeur Id_BME280 : 0x60
Chargement paramètres de calibration : begin
Chargement paramètres de calibration : done
Serial number SCD41 : ['0xcb', '0x41', '0xbf', '0x71', '0x3b', '0xce']
Index : 1
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.90 hPa
humidity : 35.3 RH
-----
Index : 2
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.90 hPa
humidity : 35.3 RH
-----
Index : 3
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.90 hPa
humidity : 35.3 RH
-----
Index : 4
Mesures du BME280
Temperature : 30.4 °C
Pression : 1015.90 hPa
humidity : 35.3 RH
-----
Index : 5
Mesures du BME280
Temperature : 30.4 °C
Pression : 1015.90 hPa
humidity : 35.2 RH
-----
Index : 6
Mesures du SCD41
CO2: 687 ppm
Temperature_scd41: 32.2 °C
Humidity_scd41: 30.3 %
-----
Index : 7
Mesures du BME280
Temperature : 30.4 °C
Pression : 1015.88 hPa
humidity : 35.2 RH
-----
```

Figure 34 : Mesures conjointes capteur BME280 et SCD41

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De récupérer les adresses I2C des périphériques I2C actifs et de les afficher sur la console ;
- De lire l'Id du capteur BME280 et d'afficher sa valeur sur la console ;
- De lire l'Id du capteur SCD41 et d'afficher sa valeur sur la console ;
- De gérer la compensation de la mesure du taux de CO₂ avec la valeur de la pression atmosphérique acquise avec le capteur BME280, toutes les heures ;
- D'afficher sur la console (ou le terminal série) les mesures de température, pression et taux d'humidité du capteur BME280, avec une périodicité d'acquisition de 1s.
- D'afficher sur la console (ou le terminal série) les mesures de taux de CO₂ du capteur SCD41, éventuellement température et taux d'humidité du capteur SCD41, avec une périodicité d'acquisition de 5s.
- L'ensemble des périodicités sera géré en utilisant timers et alarmes.

9. Acquérir la valeur de l'index COV

Le capteur SGP40⁸ ne permet pas d'accéder à une mesure directe de la concentration des Composés Organiques Volatils (COV), mais permet d'acquérir la mesure d'un index _COV (valeur dans {0,..., 500}), avec une périodicité d'1s. Plus la valeur de l'index calculée est faible, meilleure est la qualité de l'air vis-à-vis de l'index utilisé. Ce capteur nécessite 2 librairies, fournies par Adafruit⁹. Elles sont disponibles sur le Git au sein du répertoire adafruit_sgp40. Les versions disponibles ont été modifiées afin d'être compatibles avec l'utilisation de la carte microcontrôleur Raspberry Pi Pico et le firmware MicroPython installé sur la carte microcontrôleur.

Du point de vue de la mise en œuvre matérielle, le connecteur du shield Adafruit qui supporte le capteur possède les broches suivantes :

- Alimentation électrique :
 - o Vin : alimentation de la carte. La carte accepte des tensions comprises entre 3V et 5V grâce à la présence d'un régulateur (qui régule à 3.3V pour le capteur).
 - o 3.3V : sortie régulée à 3.3V.
 - o GND : masse.
- Communication I2C :
 - o SCL : horloge de synchronisation du bus I2C
 - o SDA : ligne de données.

Un emplacement dédié sur le PCB est disponible afin d'utiliser le capteur SGP40 (cf. Figure 35).

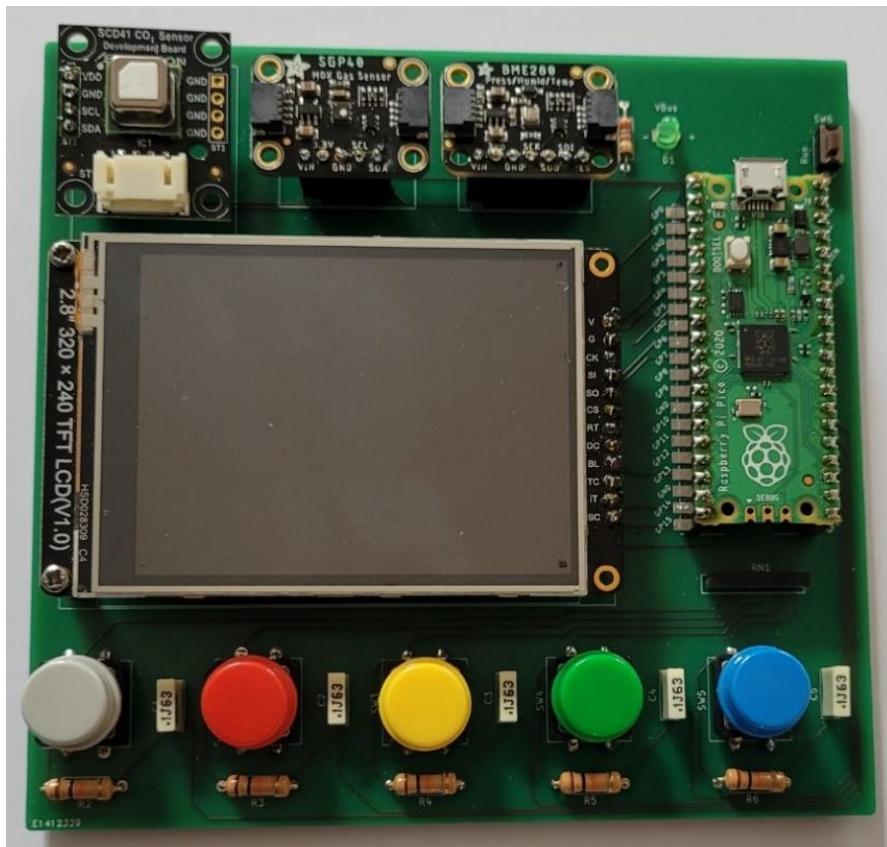


Figure 35 : Mise en place du capteur SGP40

⁸ Documentation Adafruit : <https://learn.adafruit.com/adafruit-sgp40/overview>

⁹ Librairie python pour SGP40 : https://github.com/adafruit/Adafruit_CircuitPython_SGP40

9.1. Librairies logicielles

L'utilisation du capteur SGP40 s'appuie sur 2 librairies regroupées au sein du répertoire *adafruit_sgp40* :

- *__init__.py* : elle permet d'initialiser la communication I2C sur le bus de communication, ainsi que les ressources du capteur.
- *voc_algorithm.py* : elle contient la méthode de calcul de l'*index_VOC*, selon les spécifications logicielles de la société Sensirion.

Afin d'accéder à l'ensemble du code de ces bibliothèques, il faut suivre les étapes suivantes, sous l'IDE Thonny :

- 1. Créer dans le répertoire *lib* de la mémoire du microcontrôleur le répertoire *adafruit_sgp40*.
- 2. Sauver les fichiers *__init__.py* et *voc_algorithm.py* dans ce même répertoire.

Le contenu de la mémoire du microcontrôleur doit correspondre à celui de la figure ci-dessous (cf. Figure 36).

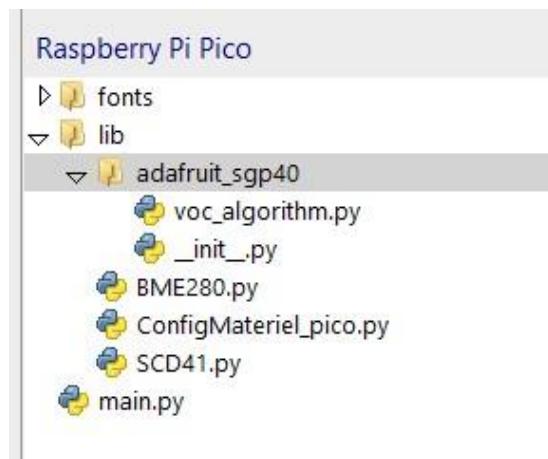


Figure 36 : Insertion des librairies associées au capteur SGP40

9.2. Mise en œuvre

En reprenant l'un des fichiers *main.py* utilisé pour valider le fonctionnement soit du capteur BME280, soit du capteur SCD41, le modifier afin de ne tester que le capteur SGP40.

Le préalable est d'importer les librairies par :

```
from adafruit_sgp40 import *
```

Concernant l'initialisation du bus de communication I2C, elle est commune aux 2 autres capteurs déjà utilisés.

La création de l'objet *SGP40_capteur* fait appel au constructeur de la classe SGP40 au sein de la librairie *__init__.py*. Cette initialisation provoque également un auto test et un reset du capteur.

```
# Initialisation du capteur COV SGP40
# Vérifier le n° de série
# Vérifier les paramètres du capteur
# Effectuer un autotest
```

```
# Effectuer un reset du capteur  
sgp40_capteur_cov = SGP40 (i2c)
```

Afin de vérifier que le capteur est correctement initialisé, nous pouvons afficher son numéro de série :

```
# Afficher le numéro de série du capteur SGP40  
print("SGP40 Serial number :", [hex(i) for i in sgp40_capteur_cov._serial_number])
```

9.3. Validation du fonctionnement

Pour valider le fonctionnement du capteur, nous voulons afficher toutes les secondes la valeur de l'index_COV. Pour cela, nous disposons de la méthode *measure_index* qui admet 2 paramètres permettant de compenser la mesure effectuée :

- La température.
- Le taux d'humidité.

et renvoie la valeur de l'index.

```
voc_index = sgp40_capteur_cov.measure_index(temperature, relative_humidity)
```

Les paramètres de la méthode *measure_index* sont exprimés dans les unités suivantes :

- temperature : °C (valeur réelle)
- relative_humidity : % (valeur réelle)

Remarque : dans un premier temps, le capteur SGP40 sera testé en utilisant des valeurs de température et du taux d'humidité stockées au sein de variables. Par la suite, ces mêmes valeurs seront définies à partir des mesures du BME280.

Remarque : au démarrage, la valeur de *voc_index* est nulle. Il est nécessaire d'attendre plusieurs dizaines d'acquisition avant de voir évoluer cette valeur. Au bout de plusieurs minutes, elle se stabilise.

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De récupérer les adresses I2C des périphérique I2C actifs et de les afficher sur la console ;
- De lire l'Id du capteur SGP40 et d'afficher sa valeur sur la console ;
- De gérer la compensation de la mesure avec la valeur de la température et du taux d'humidité ;
- D'afficher sur la console (ou le terminal série) la valeur des mesures de l'index COV, avec une périodicité de 1s.

Un exemple de sortie sur le terminal série de l'IDE Thonny est présenté ci-dessous (cf. Figure 37) et (cf. Figure 38) :

Console

```
>>> %Run -c $EDITOR_CONTENT

Configuration bus I2C : begin
Configuration bus I2C : done
Adresse périphérique(s) I2C : ['0x59', '0x62', '0x77']
Serial number
Feature set
Self test
Reset
SGP40 Serial number : ['0x0', '0x2d2', '0x9abd']
Id : 1
VOC Index: 0

Id : 2
VOC Index: 0

Id : 3
VOC Index: 0

Id : 4
VOC Index: 0

Id : 5
VOC Index: 0

Id : 6
VOC Index: 0

Traceback (most recent call last):
  File "<stdin>", line 51, in <module>
KeyboardInterrupt:
>>> |
```

Figure 37: Affichage des mesures du capteur SGP40 VOC_index (1)

```
Id : 80
VOC Index: 71

Id : 81
VOC Index: 73

Id : 82
VOC Index: 74

Id : 83
VOC Index: 75

Id : 84
VOC Index: 76

Id : 85
VOC Index: 77

Traceback (most recent call last):
  File "<stdin>", line 51, in <module>
```

Figure 38 : Affichage des mesures du capteur SGP40 VOC_index (2)

10. Synthèse1 : acquisition de l'ensemble des grandeurs environnementales

Il s'agit de regrouper au sein d'un même programme le code associé à la gestion de chacun des capteurs, en gérant distinctement la période d'acquisition des mesures de chaque capteur, ainsi que la compensation des mesures du taux de CO₂ et de l'index COV. Cette gestion fait appel au mécanisme de gestion des timers décrit plus tôt (cf. 6 Gérer des événements temporels - timer).

Fonctionnalités attendues :

- BME280 :
 - o Acquisition des mesures de température, pression atmosphérique et taux d'humidité.
 - o Périodicité : 1s.
- SCD41 :
 - o Acquisition de la mesure du taux de CO₂.
 - o Périodicité : 5s
 - o Compensation avec la mesure de la pression atmosphérique (BME280), acquise toutes les heures (on peut aussi considérer la valeur moyenne de la pression atmosphérique sur un intervalle de temps d'une heure (facultatif)).
- SGP40 :
 - o Mesure de l'index_cov.
 - o Périodicité : 1s.
 - o Compensation de la mesure de l'index_cov avec les dernières mesures de la température et du taux d'humidité (BME280).

Toutes les périodicités seront gérées en faisant appel au mécanisme de gestion des timers déjà mis en œuvre précédemment.

L'ensemble des mesures sera affiché dans la console, en tenant compte des périodicités précisées ci-dessus.

L'ensemble du code source sera placé dans le fichier *main.py*. Les mesures seront affichées dans le terminal série, selon le formatage de votre choix (cf. Figure 39 à titre d'illustration).

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De récupérer les adresses I2C des périphérique I2C actifs et de les afficher sur la console ;
- De lire les Id des capteurs BME280, SCD41 et SGP40, et d'afficher leur valeur sur la console ;
- D'afficher sur la console (ou le terminal série) les mesures de température, pression et taux d'humidité du BME280, avec une périodicité d'acquisition de 1s.
- D'afficher sur la console (ou le terminal série) les mesures de taux de CO₂ du capteur SCD41, avec une périodicité d'acquisition de 5s.
- D'afficher sur la console (ou le terminal série) la valeur des mesures de l'index COV, avec une périodicité de 1s.
- La totalité des périodicités d'acquisition de chaque capteur sera gérée en utilisant les timers ;
- La compensation de la mesure du taux de CO₂ par la pression atmosphérique sera aussi gérée par timer ;
- La mesure de l'index COV sera systématiquement compensée par la dernière mesure de la température et du taux d'humidité du capteur BME280.

Console

```
>>> %Run -c $EDITOR_CONTENT

Configuration bus I2C : begin
Configuration bus I2C : done
Adresse périphérique(s) I2C : ['0x59', '0x62', '0x77']
Valeur Id_BME280 : 0x60
Serial number SCD41 : ['0xcb', '0x41', '0xbff', '0x7', '0x3b', '0xce']
Serial number
Feature set
Self test
Reset
SGP40 Serial number : ['0x0', '0x2d2', '0x9abd']
Fin initialisation
Index : 1
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.57 hPa
humidity : 36.3 RH
-----
Index : 2
Mesures du SGP40
VOC Index: 0
-----
Index : 3
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.56 hPa
humidity : 36.4 RH
-----
Index : 4
Mesures du SGP40
VOC Index: 0
-----
Index : 5
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.56 hPa
humidity : 36.2 RH
-----
Index : 6
Mesures du SGP40
VOC Index: 0
-----
Index : 7
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.57 hPa
humidity : 36.2 RH
-----
Index : 8
Mesures du SGP40
VOC Index: 0
-----
Index : 9
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.55 hPa
humidity : 36.4 RH
-----
Index : 10
Mesures du SCD41
CO2: 935 ppm
Temperature_scd41: 32.5 °C
Humidity_scd41: 30.7 %
-----
Index : 11
Mesures du SGP40
VOC Index: 0
-----
Index : 12
Mesures du BME280
Temperature : 30.3 °C
Pression : 1015.57 hPa
```

Figure 39: Mesures conjointes BME280 - SCD41 et SGP40

11. Gestion d'un afficheur graphique

11.1. Eléments de mise en œuvre

Plutôt que d'afficher les valeurs des mesures effectuées sur le terminal série, nous allons maintenant les afficher sur un écran graphique couleur de 320 lignes et 240 colonnes. L'écran utilisé a été brièvement présenté auparavant (cf. 3.4 Interface utilisateur : écran DFR0665). Il est constitué d'une dalle graphique, contrôlée par un circuit intégré spécialisé : le contrôleur ili9341. C'est ce contrôleur qui permet de gérer la couleur d'un pixel, ainsi que sa position sur la dalle graphique (cf. Figure 40) et (cf. Figure 41).

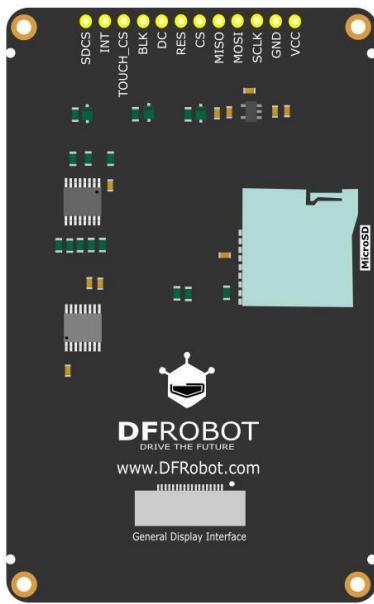


Figure 40 : Ecran DFR0665 (1)



Figure 41 : Ecran DFR0665 (2)

Les couleurs des éléments affichés sur l'écran sont définies sur 2 octets selon un codage RGB – Red Green Blue – dit RGB565 (cf. Figure 42).

- La composant R est codée sur 5 bits.
- La composante G est codée sur 6 bits.
- La composante B est codée sur 5 bits

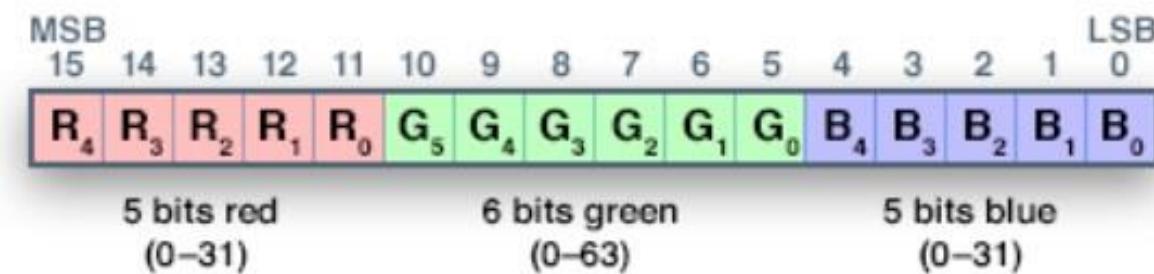


Figure 42 : Codage RGB565

Il est possible de coder des couleurs usuelles sous la forme de constantes, comme ci-dessous :

```
# Définition de couleur de base
BLANC = const (0xFFFF)
NOIR = const (0x0000)
ROUGE = const (0xF800)
VERT = const (0x07E0)
CYAN = const (0x07FF)
BLEU = const (0x001F)
LIGHTGREY = const (0xAD75)
MEDIUMGREY = const (0x8410)
DARKGREY = const (0x4A49)
```

Quant à la dalle tactile, elle est pilotée par le contrôleur XPT2046, dont le rôle est de convertir les données tactiles et de les transmettre sur le bus SPI.

Le shield écran dispose d'un emplacement pour gérer une carte µSD, mais cette fonctionnalité ne sera pas utilisée.

Afin de pouvoir communiquer avec l'écran, dans le but d'afficher des données, il faut disposer d'un bus de communication entre l'écran et le microcontrôleur. Cette communication s'effectuera au travers d'un bus SPI.

La nomenclature des broches de l'écran est définie dans le tableau ci-dessous (cf. Tableau 1) :

Numéro broche	Désignation	Fonction
1	Vcc	Alimentation électrique : 5V
2	Gnd	Masse
3	SCLK	Horloge bus SPI
4	MOSI	Data (microcontrôleur vers écran)
5	MISO	Data (écran vers microcontrôleur)
6	CS	Chip Select
7	RES	Reset
8	DC	Data / Command
9	BLK	Backlite (rétroéclairage)
10	Touch_CS	Chip select dalle tactile
11	INT	INTerrupt
12	SDCS	µSD card Chip Select

Tableau 1 : Broches écran TFT DFR0665

Afin de connecter l'écran aux ressources du microcontrôleur, un connecteur spécifique est disponible sur le circuit imprimé. L'écran est déjà installé sur le PCB. Les ressources (broches) du microcontrôleur nécessaire à l'utilisation de l'écran sont également définies. Il faut se reporter au contenu du fichier *ConfigMateriel_pico.py*.

11.2. Systèmes de coordonnées écran

La position de chaque pixel est définie par ses coordonnées écran (X_p , Y_p) au sein du référentiel (X_e , Y_e) de l'écran (cf. Figure 43). Les principaux paramètres de définition de ce référentiel de coordonnées sont :

- Largeur (width) : 240 pixels
- Hauteur (height) : 320 pixels

- Mode portrait par défaut
- Rotation : 0
- Coordonnées pixels : (Xp, Yp)
 - o Xp dans {0, ..., 239}
 - o Yp dans {0, ..., 319}

La figure ci-dessous décrit le système de coordonnées écran pour les valeur de rotation respectivement égale à 0, 90, 180 et 270 (cf. Figure 43).

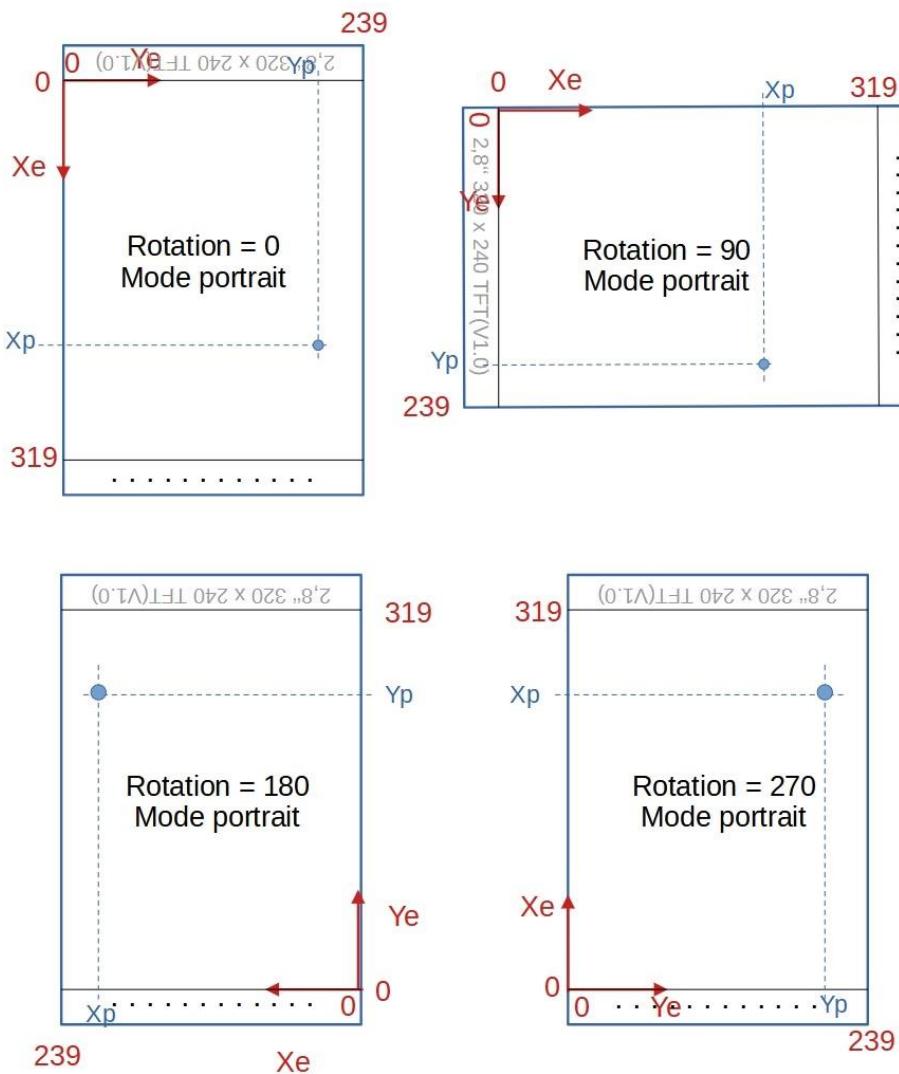


Figure 43 : Référentiel de coordonnées dalle graphique TFT écran DFR0665

Par la suite, nous utiliserons l'écran avec une rotation de 90°.

11.3. Première mise en œuvre de l'écran

Il s'agit de valider :

- L'utilisation des bibliothèques logicielles associées à l'écran.
- La gestion de polices de caractères.
- L'utilisation de quelques routines graphiques définies au sein de la librairie *ili9341.py*.

11.3.1. Prise en charge des librairies

Il est nécessaire d'inclure au sein du répertoire *lib* les librairies qui permettent d'initialiser les ressources de l'écran et celles du microcontrôleur, ainsi que celle associée à la gestion des polices de caractères. Les 2 librairies concernées sont disponibles au sein des fichiers :

- *ili9341.py*
- *xglcd_font.py*

Ces 2 fichiers seront placés au sein du répertoire *ILI9341*, lui-même situé à la racine du répertoire *lib*.

11.3.2. Prise en charge des polices de caractères

L'utilisation de polices de caractères demande de disposer de la définition graphique de chaque caractère alphanumérique au sein d'un format de fichier possédant l'extension *.c*. Vous disposez des polices de caractères suivantes :

- Unispace :
 - o Dimension caractère : largeur 12 pixels, hauteur 24 pixels
 - o Nombre de caractères (par défaut) : 96
- UnispaceExt :
 - o Dimension caractère : largeur 12 pixels, hauteur 24 pixels
 - o Nombre de caractères : 244

Les fichiers associés à ces 2 polices seront placés au sein du répertoire *fonts* à la racine de la zone mémoire du microcontrôleur (cf. Figure 44).

- Unispace12x24.c
- UnispaceExt12x24.c

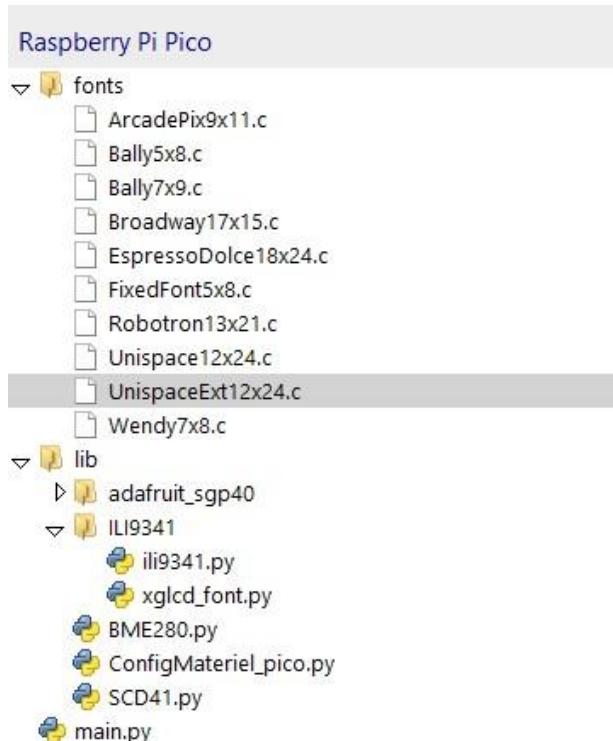


Figure 44 : Prise en compte de polices de caractères

11.3.3. Mise en œuvre logicielle

La première étape consiste à importer les bibliothèques nécessaires à la mise en œuvre des ressources matérielles et logicielles du microcontrôleur :

```
from ConfigMateriel_pico import *
import machine
import time
from ILI9341 import ili9341
from ILI9341 import xglcd_font
```

Il faut ensuite créer un objet correspondant à la liaison SPI qui servira à l'échange des données entre le microcontrôleur et l'écran TFT. Nous utiliserons les broches du microcontrôleur définies par défaut pour cette liaison SPI.

```
spi_tft = machine.SPI(0)
```

Puis il faut réaliser l'initialisation du mode d'utilisation de l'écran en appelant le constructeur de la librairie *ili9341.py* :

```
:
def __init__(self, spi, cs, dc, rst,
             width=240, height=320, rotation=0):
    """Initialize OLED.

Args:
    spi (Class Spi): SPI interface for OLED
    cs (Class Pin): Chip select pin
    dc (Class Pin): Data/Command pin
    rst (Class Pin): Reset pin
    width (Optional int): Screen width (default 240)
    height (Optional int): Screen height (default 320)
    rotation (Optional int): Rotation must be 0 default, 90, 180 or 270
"""

```

Nous allons donc créer l'objet *tft*, en spécifiant en paramètres :

- L'objet *spi*.
- Les broches de contrôle CS, DC et RST.
- Les paramètres de résolution restent inchangés.
- Le paramètre de rotation sera égal à 90.

```
tft = ili9341.Display(spi_tft, dc = TFT_DC_pin, cs = SPI_CS_pin, rst = TFT_RESET_pin, rotation = 90)
```

Nous pouvons désormais appeler toutes les méthodes définies au sein de la librairie *ili9341* et associées à l'objet *tft*. Par exemple, pour effacer la totalité de l'écran :

```
tft.clear()
```

Remarque : vous trouverez en annexe la description des en tête de quelques fonctions graphiques de la librairie graphique *ili9341*

L'étape suivante consiste à charger les polices de caractères :

```

# Importer police de caractères
print('Loading fonts...')
print('Loading unispace')
unispace = xglcd_font.XglcdFont('fonts/Unispace12x24.c', 12, 24)
print('Loading unispaceExt')
unispaceExt = xglcd_font.XglcdFont('fonts/UnispaceExt12x24.c', 12, 24, letter_count=224)
print('Fonts loaded.')

```

A partir de ce moment, il est possible d'afficher du texte ainsi que des éléments graphiques. L'examen des méthodes au sein du fichier *ili9341.py* permet d'accéder à la syntaxe de l'ensemble des fonctions supportées. Vous trouverez en annexe un résumé de quelques fonctions disponibles au sein de cette bibliothèque.

Un exemple complet de code est disponible au sein du fichier *main.py* dans le répertoire *TFT_DFR0665_V0* de l'archive fournie. Le résultat de l'exécution du programme doit correspondre à la figure ci-après (cf. Figure 45).



Figure 45 : Test affichage graphique sur l'écran DFR0665

11.4. Affichage des mesures sur l'écran TFT

L'objectif est de spécifier et concevoir un premier écran qui va permettre l'affichage de l'ensemble des mesures effectuées. L'affichage sera rafraîchi pour chaque grandeur selon la périodicité d'acquisition de chacune des mesures.

Pour des questions de structuration de code, l'ensembles des fonctions de gestion de l'affichage graphique sur l'écran TFT couleur seront définies au sein du module logiciel ou librairie *Affichage_Graphique.py*.

11.4.1. Définition des formats d'affichage

Le tableau ci-dessous (cf. Tableau 2) résume le format d'affichage de chacune des mesures, considéré sous la forme d'une **chaîne de caractères**. La colonne format spécifie le format de chacune des chaînes associées aux mesures. Il est impératif de respecter ce format

	Intervalle de validité pour affichage	Mesure	Unité	Affichage	Format
Température	[-80.0 ; +99.9]	+23.5	"°C"	+23.5°C	{:+5.1f}
Pression	[800.0 ;1100]	1023.48	"hPa"	1023.48hPa	{:7.2f}
Humidité	[0.0 ; 100.0]	42.3	"%"	42.3%	{:.1f}
CO2	{0,..., 4000}	1234	"ppm"	1234 ppm	{:4d}
COV_Index	{0,..., 500}	123	""	123	{:3d}

Tableau 2 : Formatage associé aux mesures à afficher

Pour chaque type de mesure (température, pression, ...), la chaîne de caractères générées doit posséder **le même nombre de caractères**, quelle que soit la valeur de la mesure. Ainsi :

- Mesure température : 23.5 et chaîne de caractères générée : s_t = "+23.5", soit 5 caractères au total dont un pour les 1/10^{ème} de température.
- Mesure température : 9.4 et chaîne de caractères générée : s_t = " +9.4", soit 5 caractères au total dont un pour les 1/10^{ème} de température.

En utilisant les possibilités de formatage d'une chaîne de caractères en python, il sera donc possible de gérer l'affichage graphique de chaque mesure et son rafraîchissement. Des éléments documentaires sur la fonction *format* sont fournis en annexe, ainsi que des exemples de syntaxe de mise en œuvre.

11.4.2. Librairie Affichage_Graphique.py

Vous disposez du squelette du module *Affichage_Graphique.py* qui contiendra l'ensemble des fonctions nécessaires à la gestion des affichages sur l'écran TFT. Afin de pouvoir utiliser les routines graphiques définies au sein de la bibliothèque *ili9341.py*, il faudra créer un objet de type **TFT_AFFICHAGE** en appelant le constructeur de la classe (cf. Figure 46).

```
class TFT_affichage :

    def __init__(self, tft = None) :
        """
        Args:
            TFT_affichage (class ili9341) : tft interface pour écran TFT
        """
        if tft is None:
            raise ValueError('An tft object is required.')
        self.tft = tft
```

Figure 46 : Constructeur du module *Affichage_Graphique.py*

L'utilisation de ce module nécessite de le charger dans le répertoire *lib* dans la mémoire du microcontrôleur.

11.4.3. Mise en œuvre : Ecran 1

Avant de commencer la programmation des fonctions d'affichage pour chaque mesure, il est nécessaire de définir au préalable dans l'espace de l'écran la position de chaque élément. Cela signifie qu'il faut définir les coordonnées pixels de la position chaque élément dans le référentiel (Xe, Ye).

Dans cette structuration spatiale des éléments de l'écran, certains sont fixes, et d'autres variables. Les **éléments fixes** constituent le **fond de l'écran**, et n'ont besoin d'être affichés qu'une seule fois. Les éléments liés au mesures des capteurs sont eux considérés comme variables. Par exemple, pour la température :

"Temp : +21.3 °C" où :

- "Temp :" est un élément fixe
- "+21.3 °C" est un élément variable

L'affichage des éléments du fond d'écran sera réalisé par la fonction *Fond_ecran_1* définie au sein de la librairie *Affichage_Graphique.py*.

Afin de gérer sous une forme générique l'affichage de la valeur de chaque mesure, vous devez implémenter le code de la fonction *Affiche_mesure* dont l'en-tête est fournie (cf. Figure 47)

```
# Pour afficher la valeur des mesures de t, p, h, cov ou co2 sur écran1

def Affiche_mesure (self, x, y, mesure, mesure_prec, unite_mesure, format_str, couleur, police = None) :
    # x , y : début position affichage
    # mesure = temp ou pression ou h ou cov ou co2. Valeur scalaire réelle ou entière
    # mesure_prec = temp_prec ou pression_prec ou h_prec ou cov_prec ou co2_prec. Valeur scalaire réelle ou entière
    # unité de mesure : string "°C" ou "%" ou "" ou "ppm"
    # format_str : string
    #   "{:+5.1f}" : pour la température
    #   "{:+7.2f}" : pour la pression atmosphérique
    #   "{:+4.1f}" : pour l'humidité
    #   "{:+4d}" : pour la concentration de co2
    #   "{:+3d}" : pour l'index cov'

    if police == None :
        raise ValueError ('Une police de caractères est requise')
```

Figure 47 : En tête fonction affichage d'une mesure capteur

Remarque : *mesure_prec* désigne la valeur d'une mesure acquise précédemment avant le rafraîchissement de l'affichage.

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- D'afficher les éléments du fond de l'écran 1 et de valider leur mise en forme ;
- De vérifier que l'affichage des différentes mesures respecte les formats attendus ;

Compléter le code du module *Affichage_Graphique.py* avec l'écriture des fonctions :

- *Fond_ecran_1*
- *Affiche_mesure*
- Et toutes autres fonctions liées à la gestion de l'affichage d'éléments graphiques que vous jugerez utiles.

Vous trouverez ci-après un exemple de conception et d'affichage de l'écran 1 (cf. Figure 48).

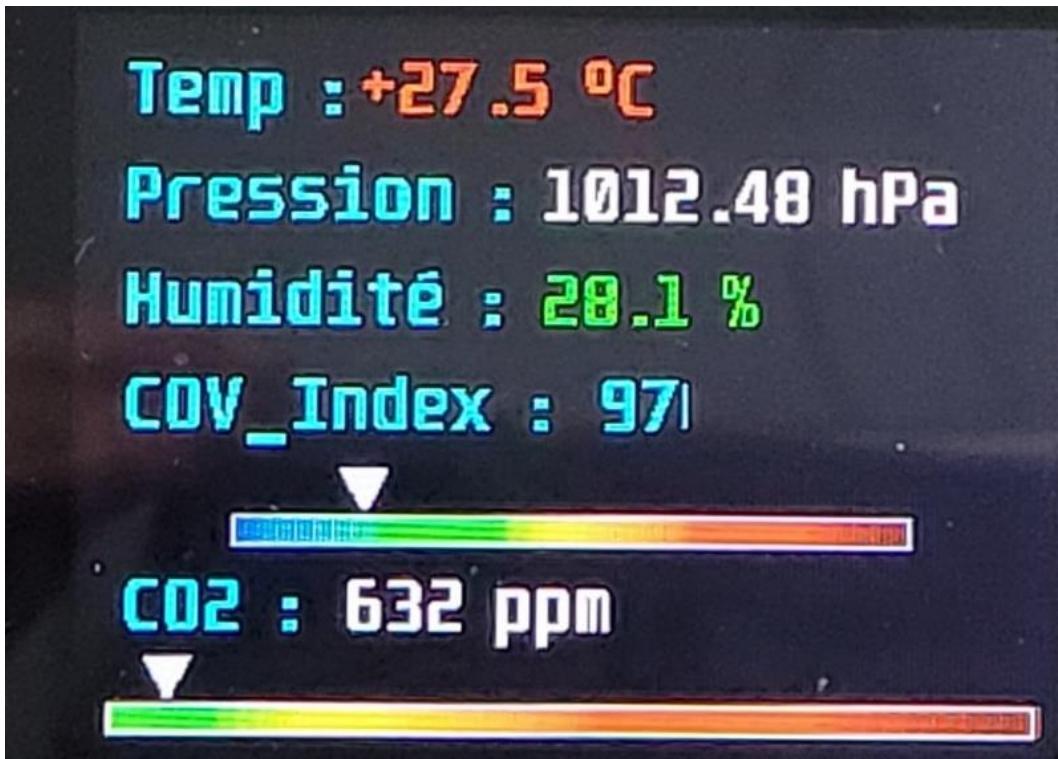


Figure 48 : Exemple d'affichage des mesures capteurs - Ecran1

Par la suite, vous devrez également respecter la périodicité de mise à jour des valeurs courantes (la dernière) et précédentes (l'avant dernière) de chaque mesure. Le rafraîchissement du contenu de l'écran1 sera aligné sur ces périodicités.

11.4.4. Mise en œuvre : Ecran 2

On souhaite définir un second écran (écran 2) qui permettra d'afficher les valeurs min et max des mesures de température, taux d'humidité, index_Cov et taux de CO2.

Sur ce second écran, on affichera également un bouton *Reset* qui aura pour fonction de mettre à jour les valeurs de ce tableau avec les dernières mesures effectuées, ainsi que de gérer le niveau de rétroéclairage sur une échelle de 5 niveaux.

Afin d'indiquer à l'utilisateur quel est le niveau de rétroéclairage courant de l'écran, un indicateur (ici un disque vert) est superposé à l'échelle des niveaux de rétroéclairage. Un exemple de conception de l'écran 2 est présenté sur la figure ci-après (cf. Figure 49).

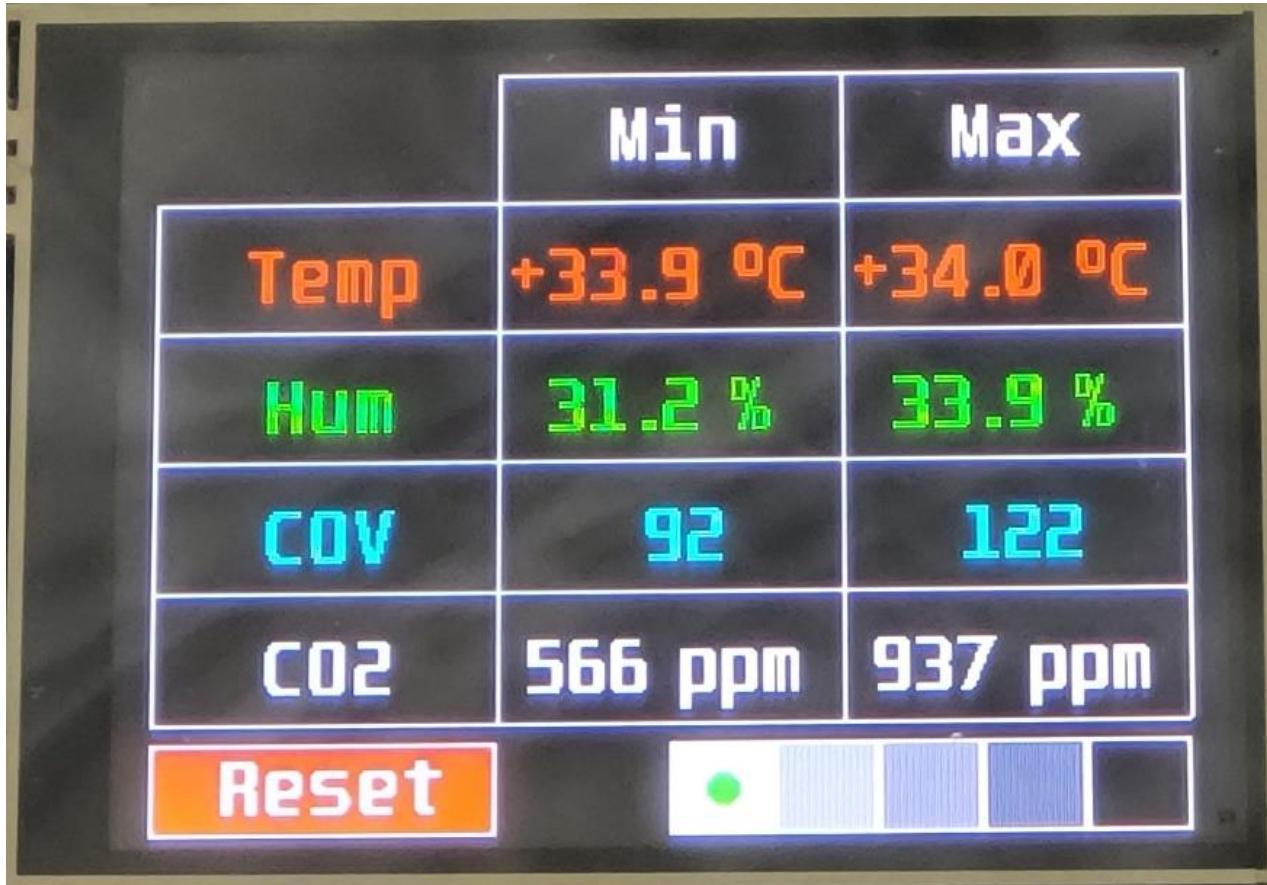


Figure 49 : Exemple de mise en forme Ecran2

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De valider l'affichage des différents éléments de l'écran 2.

Compléter le code du module *Affichage_Graphique.py* avec l'écriture des fonctions :

- *Tableau_ecran_2* : pour afficher le tableau des valeurs min et max de température, humidité, Index Cov et taux de CO₂, ainsi que l'échelle du rétroéclairage avec le niveau courant, et un bouton Reset pour provoquer ultérieurement la mise à jour des valeurs du tableau ;
- *Bouton* : pour afficher un bouton avec une légende et une couleur de fond ;
- *Echelle_choix_RetroEclairage* : pour afficher sur l'écran TFT l'échelle des niveaux de luminosité du rétroéclairage et visualiser le niveau courant.

11.5. Simulation de la dalle tactile

La plateforme matérielle dispose de 5 boutons poussoir SW_i, i = 1 à 5 (de gauche à droite) (cf. Figure 50). Ils vont nous permettre de simuler l'utilisation de la dalle tactile présente sur l'écran.



Figure 50 : Boutons poussoir carte PCB

Les fonctions associées à chaque bouton poussoir sont résumées ci-dessous (cf. Tableau 3).

Bouton poussoir sur PCB	Couleur	Fonctionnalité 1	Fonctionnalité 2
SW1	Gris	Reset des valeurs du tableau récapitulatif des mesures min et max (écran 2) avec les dernières valeurs acquises.	
SW2	Rouge	Provoquer l'affichage du tableau récapitulatif des valeurs min et max pour la température, le taux d'humidité, la valeur de l'index COV et le taux de CO ₂ (écran 2).	
SW3	Jaune	Gestion du niveau de rétroéclairage de l'écran TFT : incrément.	
SW4	Vert	Validation du niveau de rétroéclairage de l'écran TFT.	Forcer le retour à l'écran d'affichage des mesures (écran 1) depuis écran du tableau des valeurs min et max (écran 2).
SW5	Bleu	Gestion du niveau de rétroéclairage de l'écran TFT : décrément.	

Tableau 3 : Fonctionnalités associées aux boutons poussoir

L'appui sur un des 5 boutons poussoir doit être pris en compte aussi rapidement que possible, quel que soit l'instruction qui est exécutée au sein de la boucle `while ... True`. Afin de garantir que l'appui sur un bouton poussoir est pris en compte « immédiatement », nous allons de nouveau faire appel au mécanisme de gestion des interruptions.

11.5.1. Mécanisme d'interruption : principe

Le microcontrôleur possède des ressources matérielles internes comme les timers, qui peuvent générer des interruptions. Ce mécanisme a déjà été utilisé pour gérer la périodicité d'acquisition de chaque capteur.

De façon plus générale, une interruption (IT) est constituée par un événement externe au programme en cours d'exécution. Lorsqu'une interruption survient, si elle est prise en compte, le programme en cours d'exécution est interrompu et le microcontrôleur exécute alors un programme particulier appelée routine d'interruption, avant de redonner la main au programme interrompu (cf. Figure 51).

Pour un microcontrôleur, plusieurs sources d'interruptions peuvent être considérées :

- Interruptions externes :
 - o Changement de l'état d'une broche ;
 - o Détection d'un front montant et / ou descendant sur une broche ;
- Interruptions internes :
 - o Sur un événement particulier associé à un timer ;
 - o Lors d'une Conversion Analogique Numérique ;
 - o Associée à la gestion d'une liaison série, ...

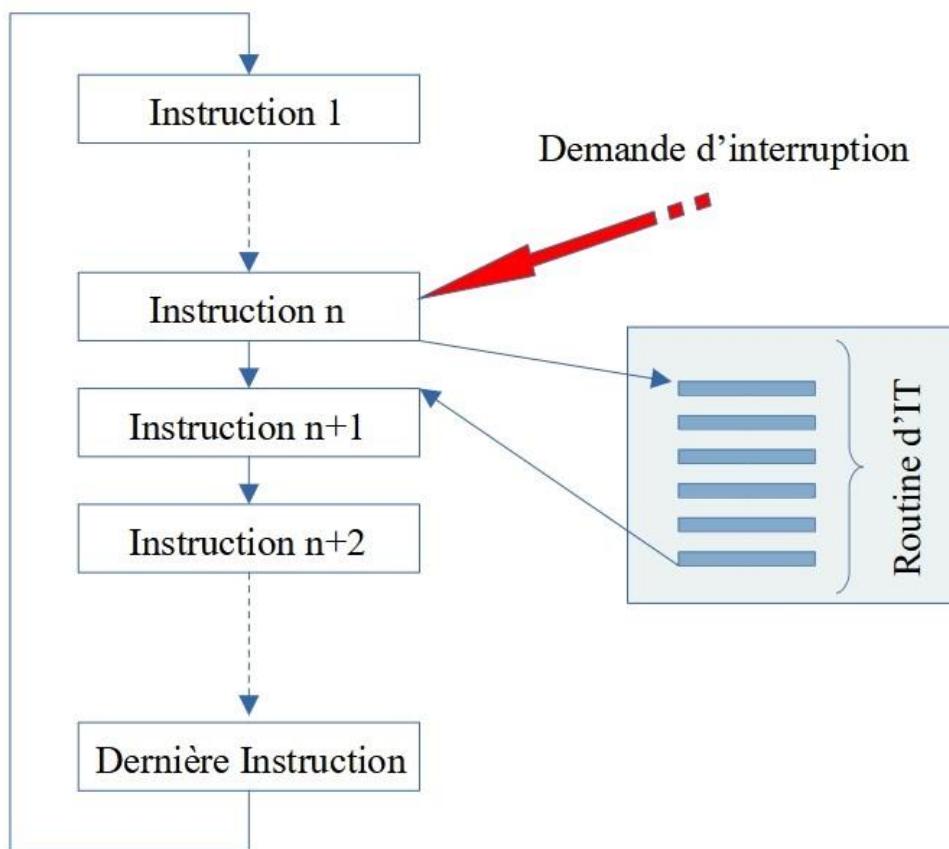


Figure 51 : Mécanisme d'interruption

Ici, nous voulons gérer une ressource matérielle, un bouton poussoir, et provoquer une interruption afin de prendre en compte l'événement « appui » sur un des 5 boutons poussoir. La librairie *Pin* permet d'accéder à la méthode *irq* qui implémente le mécanisme que nous souhaitons mettre en œuvre.

11.5.2. Gérer l'appui sur un bouton poussoir : mise en œuvre matérielle

Chaque bouton poussoir est connecté à un des port d'E/S du microcontrôleur (voir fichier *ConfigMateriel_pico.py*). Pour chaque bouton poussoir, un circuit anti rebond est mis en place. Il permet de filtrer les oscillations du signal disponible sur la broche d'E/S lors d'un appui. Au repos (aucun appui sur un bouton poussoir), l'entrée de la broche d'E/S est à l'état haut. Lorsque l'on appuie sur un des boutons poussoir, l'état de la broche d'E/S correspondant passe à l'état bas (cf. Figure 52)¹⁰. La transition état haut – état bas correspond à un front descendant que l'on observe sur la sortie CLOCK.

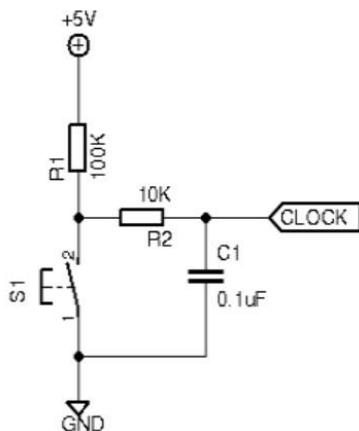


Figure 52 : Circuit anti rebond associé à un interrupteur

C'est ce changement d'état, défini par un front descendant (cf. Figure 53) qui provoque l'interruption.

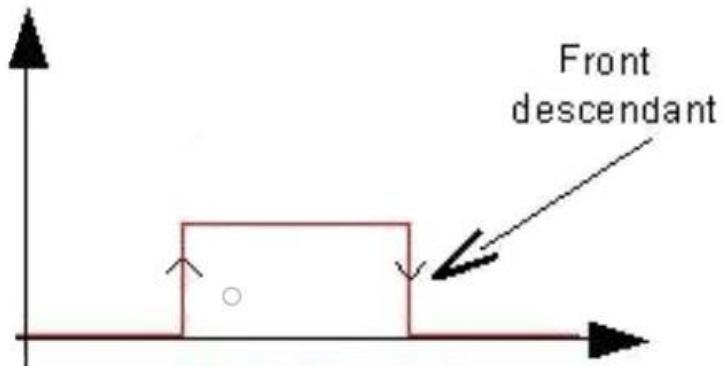


Figure 53: Front descendant

11.5.3. Gérer l'appui sur un bouton poussoir : mise en œuvre logicielle

La librairie *Pin* de Micropython possède la méthode *irq* (cf. <https://docs.micropython.org/en/latest/library/machine.Pin.html?highlight=irq>) qui permet de traiter une interruption matérielle sur une broche d'E/S. Au sein de cette méthode, il faut juste préciser :

- *handler* : le nom de la fonction qui sera appelée lors du traitement de l'interruption.
- *trigger* : définit le type d'événement qui provoque l'interruption. En ce qui nous concerne, ce sera un **front descendant**, qui peut être spécifié avec la constante *IRQ_FALLING* définie au sein de la classe *Pin*.

¹⁰ Source : <https://forums.cnetfrance.fr/materiel-informatique-composants-et-peripheriques/6583029-gestion-des-anti-rebonds-d-un-bouton-poussoir>.

Ainsi, pour gérer l'appui sur n'importe lequel des 5 boutons poussoir, nous appellerons la fonction *BP_callback (pin)* (la routine d'interruption) qui admet un seul paramètre *pin* dont l'instance correspond à l'objet *Pin* à l'origine de l'interruption. Cela signifie qu'au sein de la fonction *BP_callback (pin)*, nous pourrons savoir quel bouton poussoir est à l'origine de l'IT, et donc induire la fonctionnalité à traiter.

L'exemple ci-après détaille la gestion de la transition de l'écran 1 à l'écran 2 lorsque l'on appuie sur le bouton SW2 et le retour de l'écran 2 vers l'écran 1 lorsque l'on appuie sur le bouton SW4.

On déclare les variables qui vont permettre de gérer les différents événements :

```
# Pour gestion des actions selon les écrans
BP_press_flag = False          # Permet de savoir si appui sur un des BP
Affiche_Ecran1_flag = True      # Ecran 1 : affichage par défaut
Affiche_Ecran2_flag = False     # Ecran 2 pas affiché
Interrupt_pin = None            # Pour récupérer le BP à l'origine de l'IT dans boucle while
```

Puis on écrit la fonction d'interruption *BP_callback (pin)* (cf. Figure 54).

Remarque : la variable globale *interrupt_pin* permet de récupérer au sein du programme *main.py* quel est le bouton poussoir à l'origine de l'interruption et d'adapter le traitement à effectuer.

Remarques :

L'écriture d'une routine d'interruption est régie par des règles « simples » (cf. https://docs.micropython.org/en/latest/reference/isr_rules.html#isr-rules) dont vous trouverez ci-dessous un extrait :

[...]

- *Keep the code as short and simple as possible.*
- *Avoid memory allocation : no appending to lists or insertion into dictionaries, no floating point.*
- *Where data is shared between the main program and an ISR, consider disabling interrupts prior to accessing the data in the main program and re-enabling them immediately afterwards (see Critical Sections).*

[...]

Ces règles d'écriture conduisent à dire qu'au sein d'une routine d'IT, on ne trouvera jamais d'appel à la fonction *print* par exemple.

```

96 # Fonction callback associée aux boutons poussoir
97 def BP_callback (pin) :
98     global BP_press_flag, interrupt_pin
99     global Affiche_Ecran1_flag, Affiche_Ecran2_flag, Time_Out_screen2
100
101    interrupt_pin = pin
102
103    if Affiche_Ecran1_flag == True :
104        if pin == BP_ECRAN2_pin: # Aller à L'écran 2
105            Affiche_Ecran1_flag = False
106            Affiche_Ecran2_flag = True
107            Time_Out_screen2 = True
108
109    elif Affiche_Ecran2_flag == True :
110        if pin == BP_BACKLITE_VALIDE_pin : # Retour à L'écran 1
111            Affiche_Ecran1_flag = True
112            Affiche_Ecran2_flag = False
113
114    BP_press_flag = True # Permet de savoir qu'un appui sur un BP a eu lieu

```

Figure 54: Routine d'interruption associée aux boutons poussoir

```

# Gestion des Boutons Poussoir : gestion par IT matérielle de chaque BP
if BP_press_flag == True : # Si un appui sur un des BP est détecté

    if Affiche_Ecran1_flag == True: # L'écran 1 est affiché
        pass

    elif Affiche_Ecran2_flag == True: # Afficher écran 2

        tft.clear()
        tft_affichage.Tableau_ecran_2 (mesures_min_max, index_retroclairage, unispaceExt)
        TimeOut_Affichage_Ecran_2_Timer.init(period = _15s, mode = Timer.ONE_SHOT, callback = TimeOut_Ctrl_affichage_Ecran2)

        while (Time_Out_screen2 == True and Affiche_Ecran2_flag == True) : # Gestion du temps d'affichage de l'écran 2
            pass

        Affiche_Ecran1_flag = True
        Affiche_Ecran2_flag = False

        # Restaurer Le fond de L'écran 1
        tft.clear()
        tft_affichage.Fond_ecran_1(unispaceExt)
        tft_affichage.Affiche curseur_ecelle_couleur (voc_index, voc_index_prec, COV_MIN, COV_MAX, X_ECHELLE_COULEUR_COV,
            LARGEUR_ECHELLE_COULEUR_COV, Y_TGLE_INDEX_COV) # cov
        tft_affichage.Affiche curseur_ecelle_couleur (co2, co2_prec, CO2_MIN, CO2_MAX, X_ECHELLE_COULEUR_CO2,
            LARGEUR_ECHELLE_COULEUR_CO2, Y_TGLE_INDEX_CO2) # co2

        BP_press_flag = False

```

Figure 55 : Gestion de la transition de l'écran 1 vers l'écran 2

Afin d'avoir une exécution aussi rapide que possible, le code de la routine *BP_callback (pin)* se limite à la mise à jour de variable booléennes qui seront traitées ensuite au sein de la boucle *while ... True* :

Enfin, il faut associer la routine d'interruption *BP_callback* aux broches d'E/S du microcontrôleur qui seront à l'origine de l'IT. Ici, ce seront les BP SW2 et SW4. On aura donc les initialisations suivantes associées à chaque broche du microcontrôleur au sein du fichier *main.py* :

```
BP_ECRAN2_pin.irq(handler = BP_callback, trigger = Pin.IRQ_FALLING)  
BP_BACKLITE_VALIDE_pin.irq(handler = BP_callback, trigger = Pin.IRQ_FALLING)
```

A ce stade, l'ensemble des initialisations sont effectuées. Il faut maintenant gérer les différents événements liés à l'appui sur les BP dans la boucle *while ... True*. Cela se traduit par la portion de code suivante (cf. Figure 55).

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De gérer l'affichage de l'écran 1 et la mise à jour de l'affichage des mesures issues des capteurs.
- Lorsqu'un appui sur le BP SW2 est effectué, l'écran 2 s'affiche.
- Le retour à l'écran 1 s'effectue automatiquement en contrôlant la durée d'affichage de l'écran 2 par un timer ou bien par un appui sur le BP SW4.

11.5.4. Gestion complète des boutons poussoir

A partir de l'exemple précédent, il est possible d'étendre les fonctionnalités de notre système par la gestion complète de l'ensemble des boutons poussoir selon les spécifications du Tableau 2.

Cela revient à implémenter les fonctionnalités suivantes :

- Lorsque l'écran 1 est affiché :
 - o Si appui sur SW2, affichage de l'écran 2.
 - o Si appui sur SW3, SW4 ou SW5, gestion du niveau de rétroéclairage avec mise à jour du niveau de rétroéclairage de l'écran TFT (fonctionnalité optionnelle).
- Lorsque l'écran 2 est affiché :
 - o Si appui sur SW1 : reset des valeurs du tableau des valeurs min et max et affichage de l'écran 2 mis à jour.
 - o Si appui sur SW3, SW4 ou SW5, gestion du niveau de rétroéclairage avec mise à jour du niveau de rétroéclairage de l'écran TFT.
 - o Si appui sur SW4, retour à l'écran 1.
 - o Ou retour automatique à l'écran 1 après une durée donnée.

La figure ci-après (cf. Figure 56) représente le diagramme de transition entre les 2 écrans ainsi que les actions qui peuvent être réalisées selon l'écran affiché (écran 1 ou écran 2).

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- De gérer l'affichage de l'écran 1 et la mise à jour de l'affichage des mesures issues des capteurs.
- De gérer l'ensemble des fonctionnalités associées aux boutons poussoir.

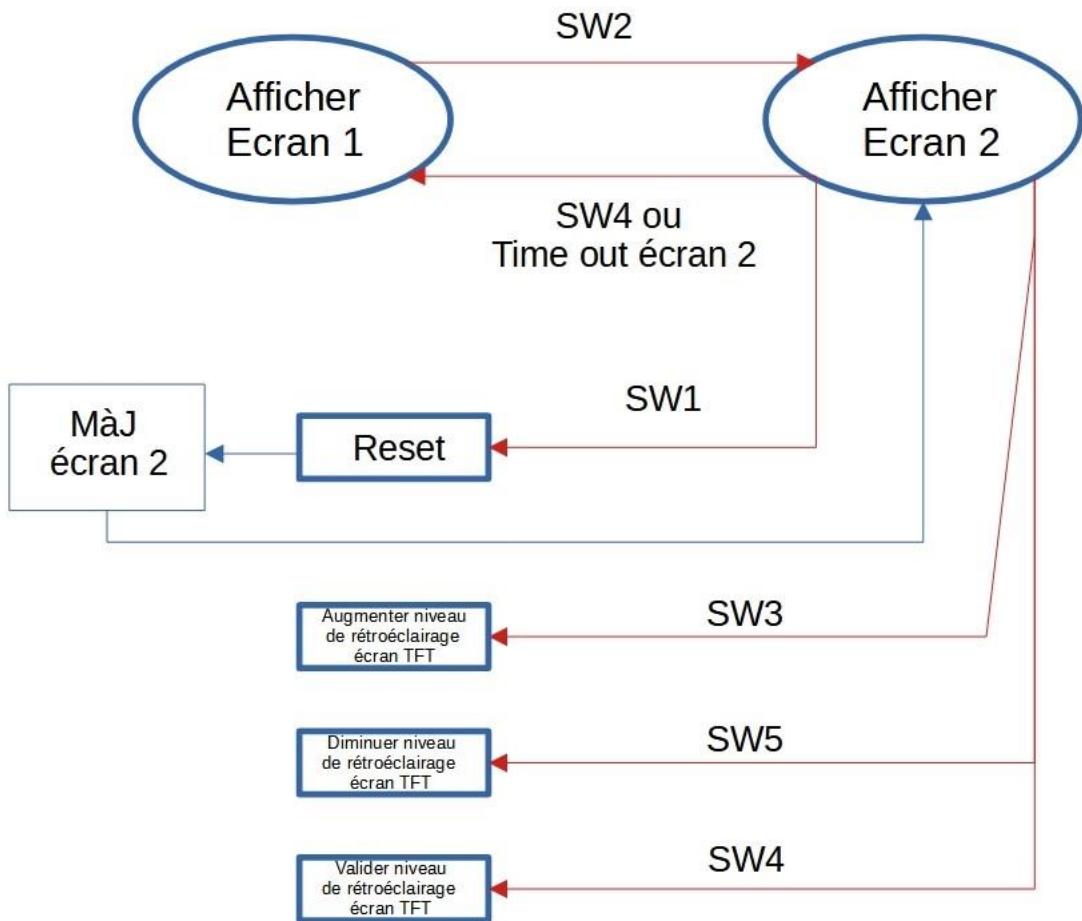


Figure 56 : Fonctionnalités gérées depuis les BP

11.6. Utilisation de la dalle tactile

Plutôt que d'utiliser les boutons poussoir de la plateforme matérielle, nous allons maintenant tirer parti de la dalle tactile. Ainsi, l'écran et sa dalle tactile constitueront l'unique interface utilisateur.

La dalle tactile est résistive. Elle se situe au-dessus de la dalle graphique. Elle est pilotée par le contrôleur XPT2046, dont le rôle est de convertir les données tactiles et de les transmettre sur le bus SPI. Ces données tactiles « brutes » doivent ensuite être converties en coordonnées écran. Pour cela, nous devrons procéder à la calibration de la dalle tactile, puis vérifier qu'à partir de l'appui sur un point de la dalle tactile, nous récupérons les coordonnées écran du même point d'appui.

11.6.1. Interface matérielle de la dalle tactile

Au travers du contrôleur xpt2046, les données de la dalle tactile sont transmises via le bus de communication SPI, qui est donc commun avec l'écran TFT.

Concernant le contrôle de la dalle tactile, il faut définir sur quel port d'E/S du microcontrôleur sera connectée la broche Touch_CS (ou encore TC). Cela correspond à la variable `Touch_Screen_CS_pin` au sein du module `ConfigMateriel_pico.py`.

Enfin, lorsque l'on exerce une pression suffisante sur un point de la dalle tactile, un signal est généré sur la broche `INT` du shield écran. Ce signal est ensuite utilisé comme source d'interruption matérielle vers

une broche particulière du microcontrôleur. Cette broche est aussi définie au sein du module *ConfigMateriel_pico.py* à l'aide de la variable *Touch_Interrupt_pin*.

11.6.2. Référentiel écran TFT et dalle tactile du DFR0665

La dalle tactile possède le référentiel (X_t , Y_t). Ce référentiel est absolu et ne dépend que de la dalle tactile. Il est invariant si l'on fait pivoter l'écran TFT de 90° par exemple. Lorsque l'on appuie sur la dalle tactile, on récupère le couple des coordonnées tactiles (Raw_x , Raw_y) dans le repère (X_t , Y_t). Il faut ensuite faire le lien (ou le mapping) entre ces coordonnées (Raw_x , Raw_y) et les coordonnées écran (X_e , Y_e) du même point d'appui.

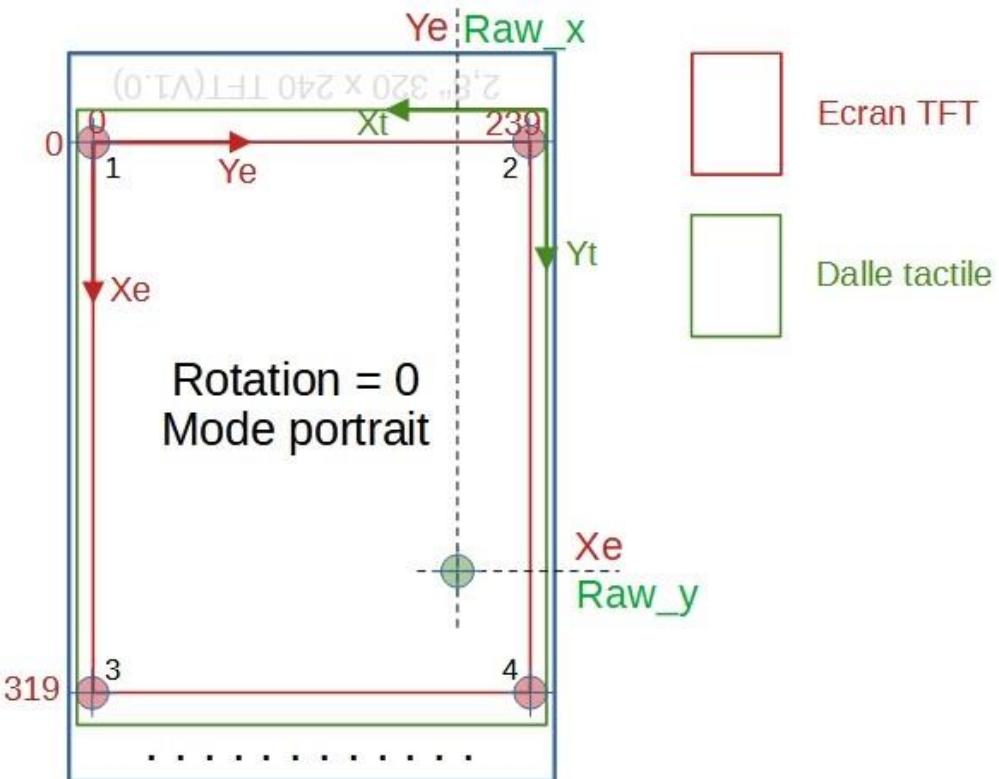


Figure 57 : Référentiels écran TFT et dalle tactile

Les zones d'affichage de l'écran TFT et celle de la dalle tactile ne sont pas tout à fait superposées. Celle de la dalle tactile possède une surface plus importante que celle de l'écran TFT. Pour que le mapping entre les coordonnées écran et celles de la dalle tactile soit le plus exact possible, il faut caractériser 4 points particuliers qui sont les coins de la zone de l'écran TFT (point 1, 2, 3 et 4) (cf. Figure 57). Le tableau ci-dessous fait le lien entre les coordonnées écran TFT et les coordonnées de la dalle tactile, pour l'écran TFT en mode portrait et rotation = 0 (cf. Tableau 4).

	X_e	Y_e	Raw_x	Raw_y
Point 1	0	0	Raw_x_Max1	Raw_y_min1
Point 2	0	239	Raw_x_min1	Raw_y_min2
Point 3	319	0	Raw_x_Max2	Raw_y_Max1
Point 4	319	239	Raw_x_min2	Raw_y_Max2

Tableau 4 : Correspondance coins écran TFT et données de la dalle tactile

A partir des données du tableau, on extrait les 4 paramètres x_min, x_max, y_min et y_max tels que :

- x_min = min (Raw_x_min1, Raw_x_min2)
- x_max = max (Raw_x_max1, Raw_x_max2)
- y_min = min (Raw_y_min1, Raw_y_min2)
- y_max = max (Raw_y_max1, Raw_y_max2)

Ce sont ces 4 paramètres qui permettront le mapping le plus fidèle entre la dalle tactile et l'écran TFT, pour un écran DFR0665 donné.

11.6.3. Mise en œuvre logicielle 1 : calibration de la dalle tactile

Dans cette première mise en œuvre, nous allons déterminer la valeur des paramètres x_min, x_max, y_min et y_max, puis vérifier que le mapping entre la dalle tactile et l'écran TFT est correct. Pour cela, vous disposez de la librairie *xpt2046.py*¹¹ qu'il faut inclure dans le répertoire *lib* de la carte microcontrôleur (cf. Figure 58).

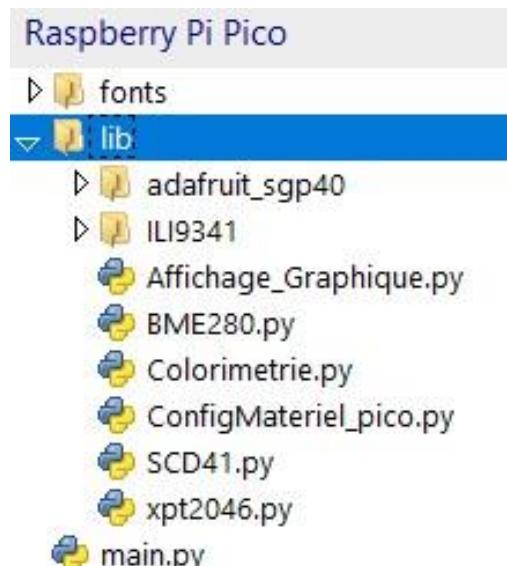


Figure 58 : Prise en compte de la librairie dédié à la gestion de la dalle tactimle

La détermination des valeurs x_min, x_max, y_min et y_max s'appuie sur le programme (fichier *main.py*) disponible dans le répertoire *TFT_DFR0665_TouchScreen_Coord*¹² de l'archive logicielle fournie. Le charger dans la mémoire du microcontrôleur et lancer son exécution. Vous devez voir s'afficher l'écran ci-après (cf. Figure 59) :

¹¹ Source : <https://github.com/rdagger/micropython-ili9341/blob/master/xpt2046.py>.

Elle a été modifié afin de prendre en compte la rotation de l'écran TFT.

¹² Le programme fourni est dérivé de celui présenté dans : <http://electroniqueamateur.blogspot.com/2021/06/utilisation-dun-ecran-tactile-tft-spi.html>



Figure 59 : Calibration de la dalle tactile (1)

Appuyer avec un stylet sur la pointe de la flèche, et répéter l'opération en suivant les instructions affichées sur l'écran. Après appui sur la pointe de la 4^{ème} flèche, vous obtenez l'affichage sur l'écran TFT des valeurs de x_min, x_max, y_min et y_max (cf. Figure 61) que vous devez noter (ces mêmes valeurs s'affichent également dans le terminal série de l'IDE Thonny) (cf. Figure 60).

```
Console X
>>> %Run -c $EDITOR_CONTENT
Loading fonts Unispace12x24 ...
Fonts loaded.
Etape 1 : 1730,225
Etape 2 : 154,198
Etape 3 : 1821,1924
Etape 4 : 149,1926
x_min: 149 , x_max: 1821 , y_min: 198 , y_max: 1926
```

Figure 60: : Calibration de la dalle tactile (2)



Figure 61 : : Calibration de la dalle tactile (3)

11.6.4. Mise en œuvre logicielle 2 : mapping écran TFT et dalle tactile

Nous pouvons maintenant vérifier que le mapping entre la dalle tactile et l'écran TFT est correct. Vous pouvez utiliser le fichier *main.py*¹³ fourni dans le répertoire *TFT_DFR0665_Verif_Mapping_TFT_TouchScreen* de l'archive logicielle fournie. Lancer son exécution et vérifier de les disques blancs qui s'affichent sont centrés sur le point d'appui de la dalle tactile (cf. Figure 62).

¹³ Source : <http://electroniqueamateur.blogspot.com/2021/06/utilisation-dun-écran-tactile-tft-spi.html>



Figure 62: Vérification du mapping écran TFT et dalle tactile

Si vous constatez un écart entre la position centrale du disque affiché et le point d'appui sur la dalle tactile, vous pouvez affiner le mapping en réitérant la procédure de recherche des valeurs de `x_min`, `x_max`, `y_min` et `y_max` vue précédemment.

11.6.5. Routine d'interruption associée à la dalle tactile

Dans l'exemple précédent, la routine d'interruption est exécuté lorsqu'un appui sur la dalle tactile est détecté. Cela permet de récupérer directement les coordonnées écran du point d'appui dans le repère (X_e , Y_e) de l'écran TFT. C'est ce qui permet ensuite de tracer des disques blancs (cf. Figure 63).

```

25  #-----
26  def routine_touch(x, y):
27      """
28          Routine d'interruption appelée lors d'un appui sur la dalle tactile
29          Args : (x,y)
30              Coordonnées écran TFT retournée après mapping avec
31              les coordonnées du point d'appui sur la dalle tactile
32      """
33
34      tft.fill_circle(x, y, 2, ili9341.color565(255, 255, 255))
35
36  #-----
```

Figure 63 : Routine d'interruption associée à la dalle tactile

Pour que le mapping entre les coordonnées de l'écran TFT et celles de la dalle tactile soit correct, il faut veiller aux points suivants (cf. Figure 64) :

- Les paramètres de largeur et hauteur de l'écran TFT doivent être identiques lors de l'appel aux constructeurs destinés à créer les objets écran TFT et dalle tactile.
- Le paramètre de rotation doit être identique lors de l'appel aux constructeurs destinés à créer les objets écran TFT et dalle tactile.
- Lors de l'appel au constructeur de la dalle tactile, il faut veiller à instancier les paramètres `x_min`, `x_max`, `y_min` et `y_max` avec ceux obtenus lors de la phase de calibration de la dalle tactile.

```

1 tft = ili9341.Display(spi_tft, dc=TFT_DC_pin, cs=SPI_CS_pin, rst=TFT_RESET_pin, rotation = 0)
2
3 Touchscreen = Touch (spi_tft, Touch_Screen_CS_pin, Touch_Interrupt_pin, int_handler = routine_touch,
4                         width = 240, height = 320, rotation = 0,
5                         x_min = 157, x_max = 1841, y_min = 200, y_max = 1947)

```

Figure 64 : Création des objets tft et Touchscreen

Afin de vérifier l'effet du paramètre de rotation, vous pouvez modifier le programme précédent en utilisant rotation = 90 par exemple.

Maintenant que nous pouvons associer l'utilisation de la dalle tactile et de l'écran TFT, nous sommes en capacité de mettre en place des traitements particuliers selon la zone de l'écran sur laquelle on effectue un appui.

11.6.6. Une première interface utilisateur : exemple

On suppose que l'écran TFT est paramétré avec une rotation de 90. Dans ce cas, les coordonnées de chaque pixels de l'écran TFT (X_p , Y_p) prennent leurs valeurs dans $\{0, \dots, 319\}$, $\{0, \dots, 239\}$ (cf. Figure 65).

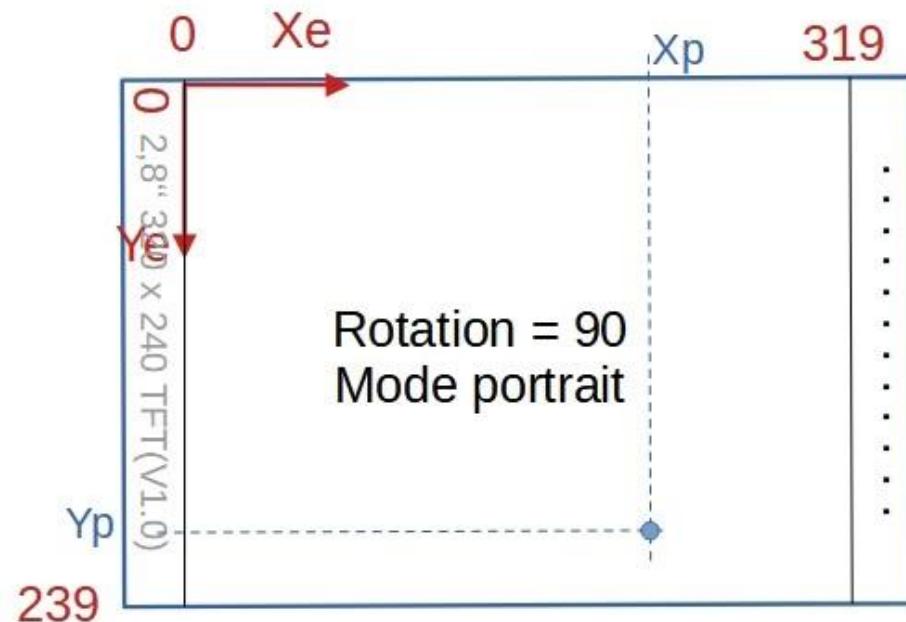


Figure 65 : Ecran TFT en mode paysage (rotation = 90)

On peut alors définir des actions que l'utilisateur peut réaliser au travers de la dalle tactile, par exemple (cf. Figure 66):

Lorsque l'écran 1 est affiché (écran d'affichage de l'ensemble des mesures) :

- Si appui au sein de la zone 1, alors afficher l'écran 2 des valeurs min et max.

Lorsque l'écran 2 est affiché (écran d'affichage des valeurs min et max) :

- Si appui au sein de la zone 2, alors revenir à l'affichage de l'écran 1.

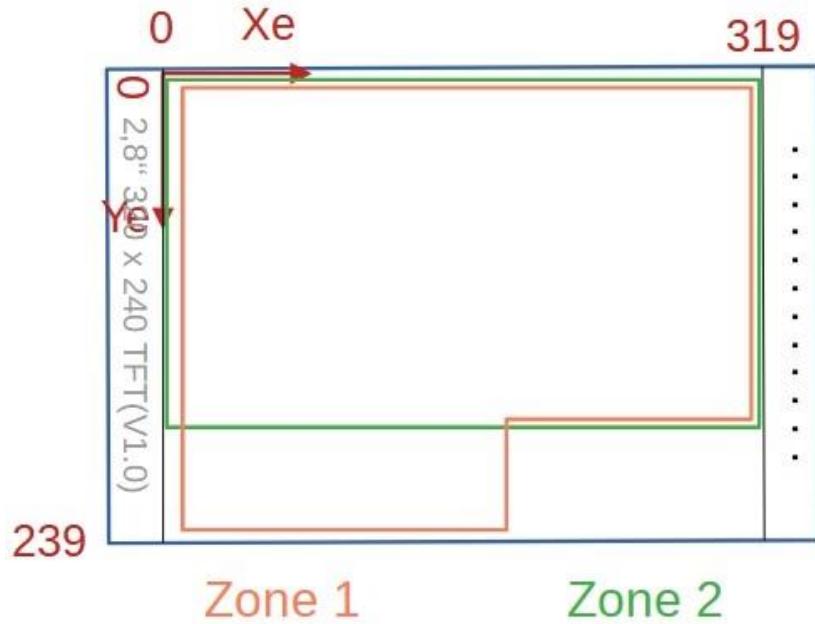


Figure 66 : Définition de zones écran TFT

Il devient nécessaire de caractériser précisément chacune des zones de l'écran TFT afin de lancer correctement l'action souhaitée suite à un appui sur la dalle tactile. Dans le repère de l'écran TFT, les zones 1 et 2 sont caractérisés comme suit (cf. Figure 67) .

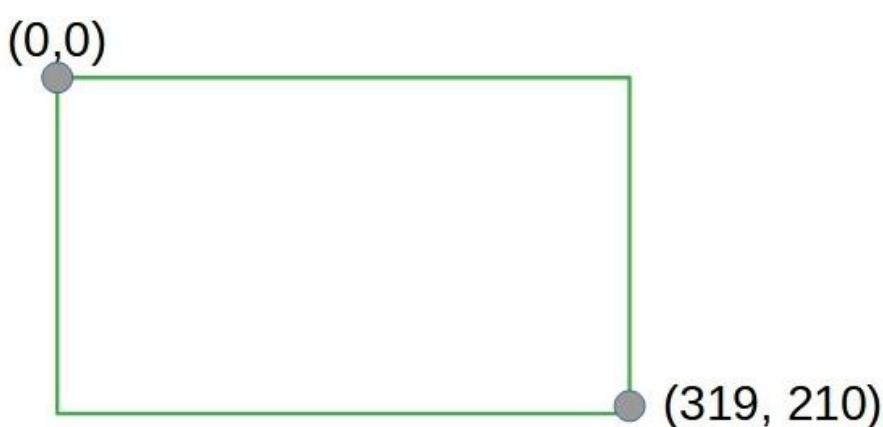
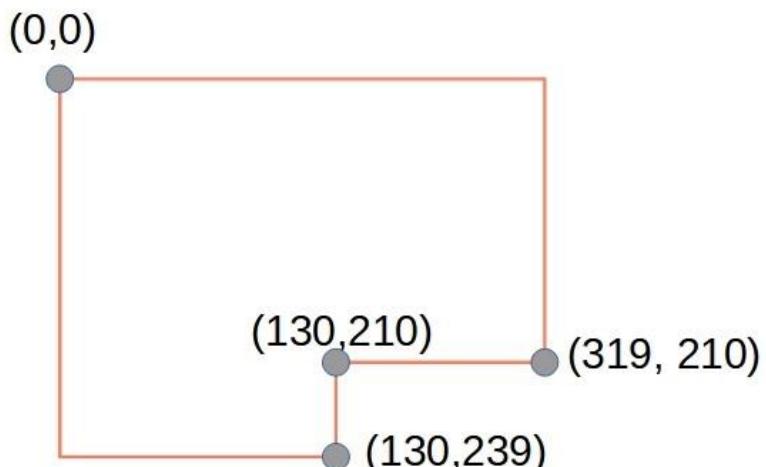


Figure 67: Définition zones écran TFT

Il convient maintenant de définir au sein de la fonction *routine_touch()* la gestion des actions qui seront effectuées ensuite dans la boucle `while ... True` : Le mécanisme de gestion des différentes actions reposera également sur l'utilisation de **flags** mis à jour au sein de la routine d'interruption associée à la dalle tactile, à l'identique de ce qui a été réalisé avec les boutons poussoir.

A titre d'illustration, le code de la fonction *routine_touch()* est fournie ci-dessous (cf. Figure 68).

Au sein de la boucle `while ... True` : la gestion et le traitement des actions à réaliser peut s'effectuer selon l'implémentation ci-après (cf. Figure 69).

Remarque : les exemples de code fournis ci-après ne constituent que des illustrations des mécanismes à mettre en œuvre pour gérer depuis la dalle tactile le swap entre 2 écrans affichés sur l'écran TFT.

```

120  #-----
121  # Gestion de La dalle tactile : fonction appelée lors d'une IT générée
122  # suite à appui sur la dalle tactile
123  # Gestion uniquement du swap entre écran 1 et écran 2
124  #-----
125  def routine_touch(x, y):
126      # x,y : coordonnées écran du point d'appui sur la dalle tactile
127      global Touch_press_flag
128      global Affiche_Ecran1_flag
129      global Affiche_Ecran2_flag
130      global Time_Out_screen2
131
132      Xe_touch = x
133      Ye_touch = y
134
135      if Affiche_Ecran1_flag == True :
136          if ((Xe_touch > 0 and Ye_touch < 319 and Ye_touch < BOUTON_RESET_DEB_Y) # Zone 1.1
137              or (Ye_touch >= BOUTON_RESET_DEB_Y and Ye_touch < (BOUTON_RESET_DEB_X + LARGEUR_BOUTON_RESET))): # Zone 1.2
138              # Aller à l'écran 2
139              Affiche_Ecran1_flag = False
140              Affiche_Ecran2_flag = True
141              Time_Out_screen2 = True
142
143      elif Affiche_Ecran2_flag == True :
144          if (Ye_touch < BOUTON_RESET_DEB_Y) : # Retour à l'écran 1
145              Affiche_Ecran1_flag = True
146              Affiche_Ecran2_flag = False
147
148      Touch_press_flag = True
149  #-----
```

Figure 68: Gestion de la dalle tactile - routine d'interruption

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- La gestion complète de l'écran 1.
- Le swap entre l'écran 1 et l'écran 2

```

391     # Gestion de la dalle tactile : Après appel routine d'IT liée à la dalle tactile
392     if Touch_press_flag == True : # Si un appui sur la dalle tactile est détecté
393
394         if Affiche_Ecran1_flag == True:
395             pass
396
397         elif Affiche_Ecran2_flag == True: # Afficher écran 2
398
399             tft.clear()
400             tft_affichage.Tableau_ecran_2 (mesures_min_max, index_retroeclairage, unispaceExt)
401             TimeOut_Affichage_Ecran_2_Timer.init(period = _15s, mode = Timer.ONE_SHOT, callback = TimeOut_Ctrl_affichage_Ecran2)
402
403             while (Time_Out_screen2 == True and Affiche_Ecran2_flag == True) :
404                 pass
405
406             Affiche_Ecran1_flag = True
407             Affiche_Ecran2_flag = False
408
409             # Restaurer l'écran 1
410             tft.clear()
411             tft_affichage.Fond_ecran_1(unispaceExt)
412             tft_affichage.Affiche curseur_ecelle_couleur (voc_index, voc_index_prec, COV_MIN, COV_MAX, X_ECHELLE_COULEUR_COV,
413                                         LARGEUR_ECHELLE_COULEUR_COV, Y_TGLE_INDEX_COV) # cov
414             tft_affichage.Affiche curseur_ecelle_couleur (co2, co2_prec, CO2_MIN, CO2_MAX, X_ECHELLE_COULEUR_CO2,
415                                         LARGEUR_ECHELLE_COULEUR_CO2, Y_TGLE_INDEX_CO2) # co2
416
417             Touch_press_flag = False

```

Figure 69 : Gestion des fonctionnalités utilisateurs swap écran 1 – écran 2

11.6.7. Interface utilisateur : version finale

Il s'agit d'étendre les mécanismes mis en œuvre précédemment à l'ensemble des fonctionnalités rappelées ci-dessous, mais en utilisant la dalle tactile. Pour cela, il est nécessaire de définir au préalable des zones de l'écran TFT et de spécifier l'action qui sera réalisée lors d'un appui sur chacune des zones.

- Lorsque l'écran 1 est affiché :
 - o Si appui sur **zone1** de l'écran TFT, affichage de l'écran 2.
 - o Si appui sur **zone2** de l'écran TFT, gestion du niveau de rétroéclairage avec mise à jour du niveau de rétroéclairage de l'écran TFT (fonctionnalité optionnelle).
- Lorsque l'écran 2 est affiché :
 - o Si appui sur **zone3** de l'écran TFT : reset des valeurs du tableau des valeurs min et max et affichage de l'écran 2 mis à jour.
 - o Si appui sur **zone2** de l'écran TFT, gestion du niveau de rétroéclairage avec mise à jour du niveau de rétroéclairage de l'écran TFT.
 - o Si appui sur **zone1** de l'écran TFT, retour à l'écran 1.
 - o Ou retour automatique à l'écran 1 après une durée donnée.

Synthèse : écrire le programme (fichier *main.py*) qui permet :

- La gestion complète de l'écran 1.
- Le swap entre l'écran 1 et l'écran 2
- La gestion complète des éléments de l'écran 2.
- La gestion de l'ensemble des fonctionnalités décrites ci-dessus.

12. Annexes

La programmation du microcontrôleur qui héberge le code qui sera exécuté nécessite de disposer d'un environnement de développement intégré ou IDE¹⁴ qui offre les fonctionnalités suivantes :

- Accès à un éditeur de code source.
- Outil de configuration de l'IDE destiné à prendre en compte la cible microcontrôleur sur laquelle le code sera exécuté.
- Gestionnaire de la mémoire de programme du microcontrôleur, aussi bien pour les librairies que pour le programme principal.
- Communication avec le microcontrôleur via un terminal série.

L'IDE utilisé est Thonny (cf. www.thonny.org). La procédure d'installation présentée ci-après est réalisée sous environnement Windows 10, 64 bits. D'autres OS sont également supportés (MAC OS, Linux).

12.1. Installation de l'IDE de programmation Thonny

La première étape consiste à se connecter sur le site www.thonny.org (cf. Figure 70).

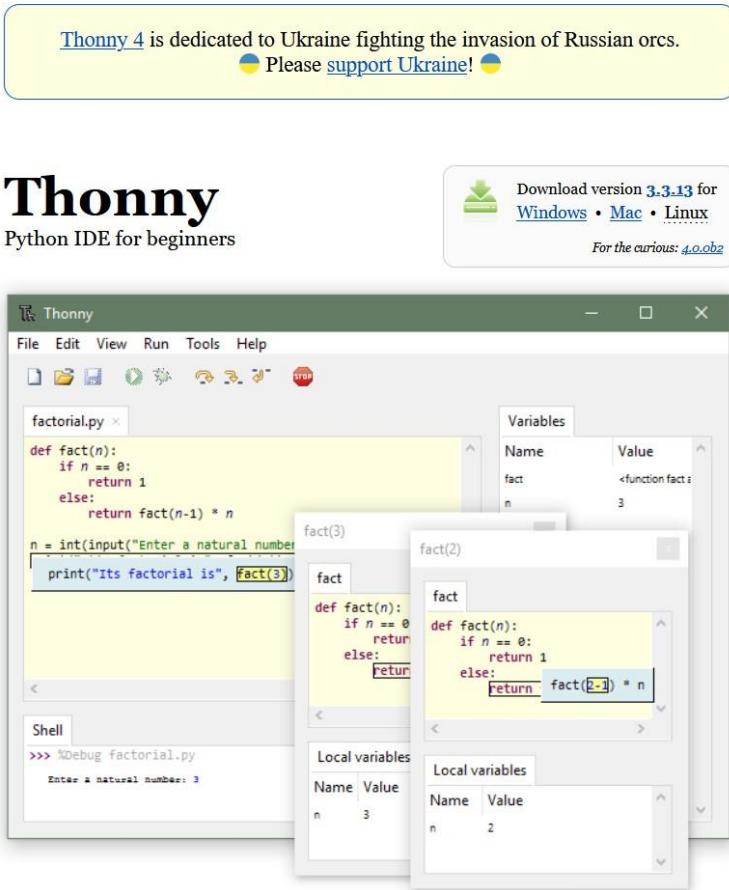


Figure 70 : Connexion sur le site thonny.org

puis à sélectionner l'OS de la machine sur laquelle sera installé l'IDE (ici IDE version 3.3.13 pour Windows) (cf. Figure 71).

¹⁴ IDE : Integrated Development Environment.

Thonny 4 is dedicated to Ukraine fighting the invasion of Russian orcs.

Please support Ukraine! 🇺🇦

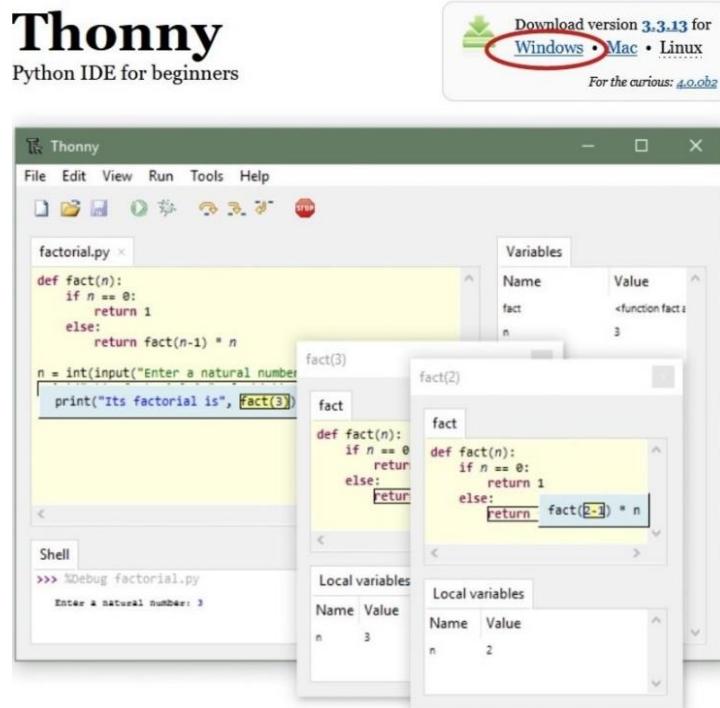


Figure 71 : Thonny - téléchargement de l'installateur Windows (1)

Une fenêtre de téléchargement de l'installateur s'ouvre alors afin d'enregistrer le fichier `thonny-3.3.13.exe` (cf. Figure 72).

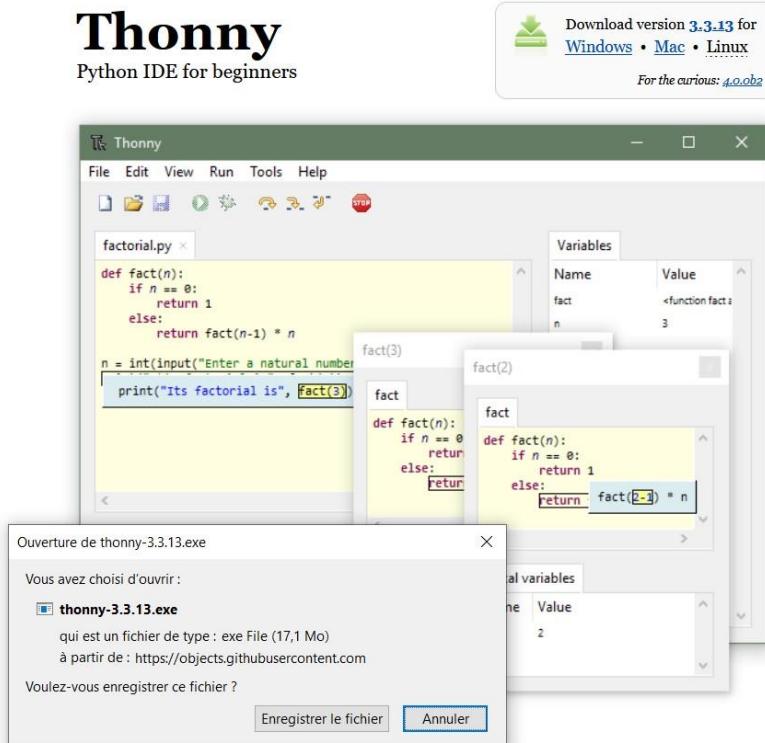


Figure 72 : Thonny - téléchargement de l'installateur Windows (2)

Une fois le fichier *thonny-3.3.13.exe* enregistré, il faut lancer son exécution (cf. Figure 73) ce qui conduit à l’ouverture de fenêtres successives (cf. Figure 74, Figure 75, Figure 76, Figure 77 et Figure 78) :

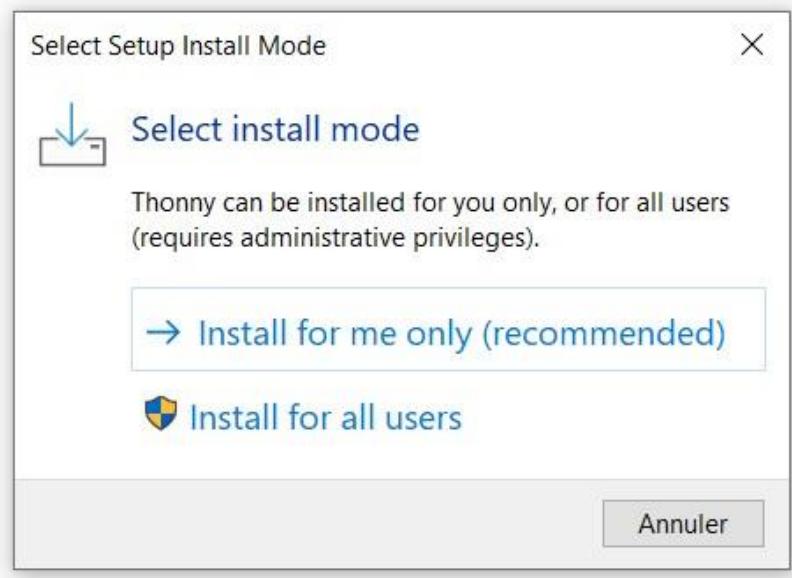


Figure 73 : Lancement de l’installateur Windows de Thonny

Une fois le mode d’installation sélectionné, la fenêtre suivante s’ouvre (cf. Figure 74) :



Figure 74 : Procédure d’installation de thonny sous Windows (1)

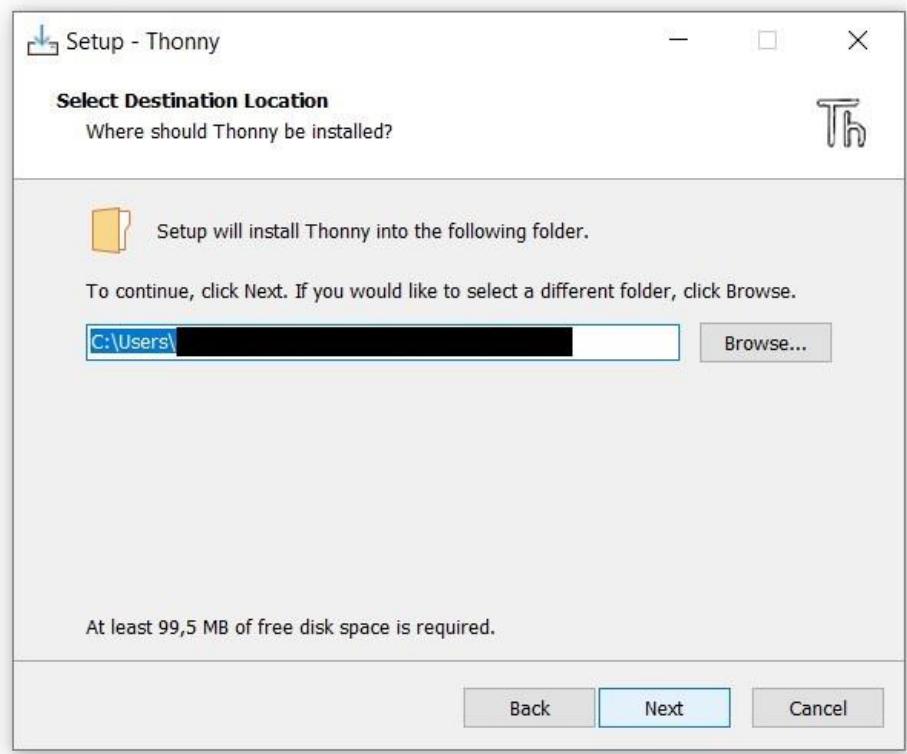


Figure 75 : Procédure d'installation de thonny sous Windows (2)

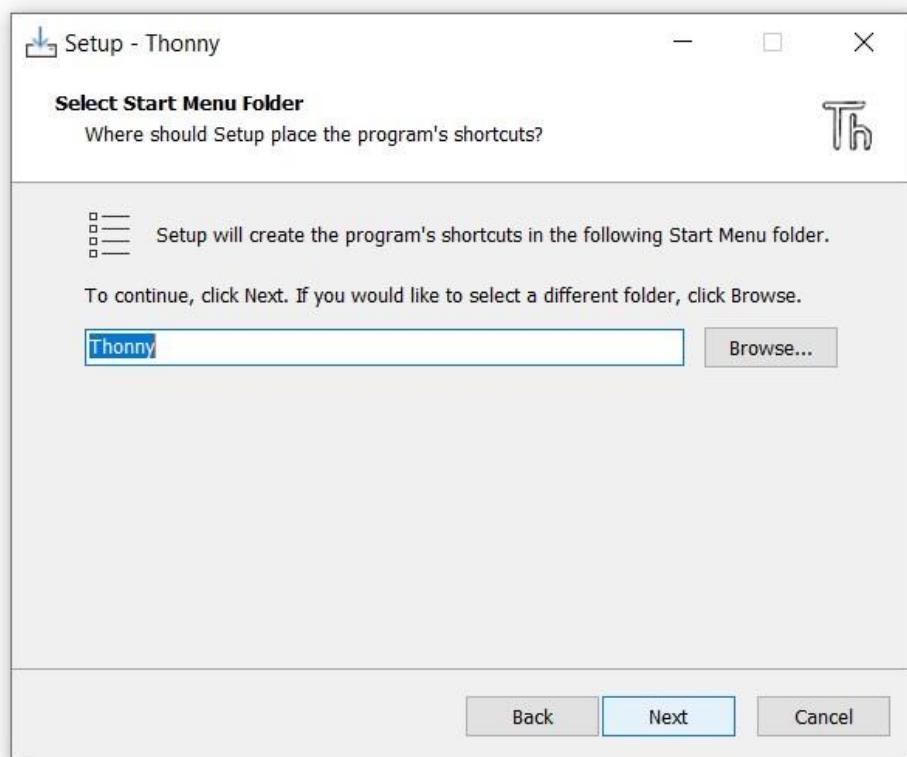


Figure 76 : Procédure d'installation de thonny sous Windows (3)

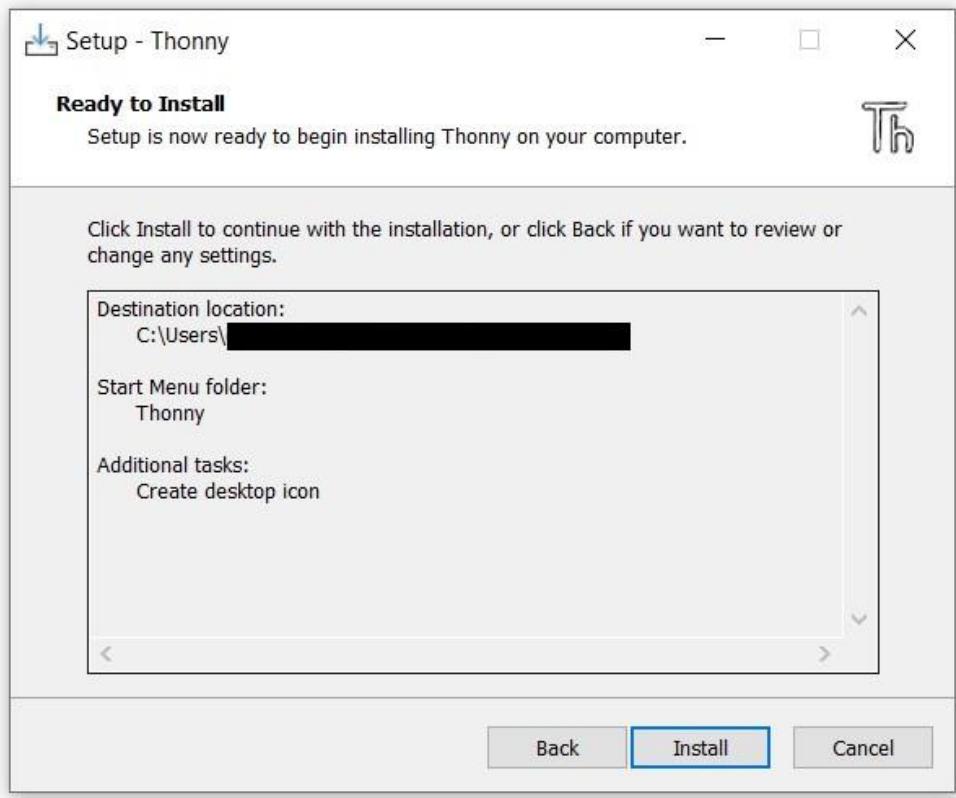


Figure 77 : Procédure d'installation de thonny sous Windows (4)

Après validation sur *Install*, attendre la fin du processus d'installation (cf. Figure 78).



Figure 78 : Procédure d'installation de thonny sous Windows (5)

L'appui sur *Finish* provoque l'ouverture de l'IDE (cf . Figure 79). Il est nécessaire de le paramétriser afin de pouvoir gérer nos programmes sur la cible microcontrôleur Raspberry Pi Pico.

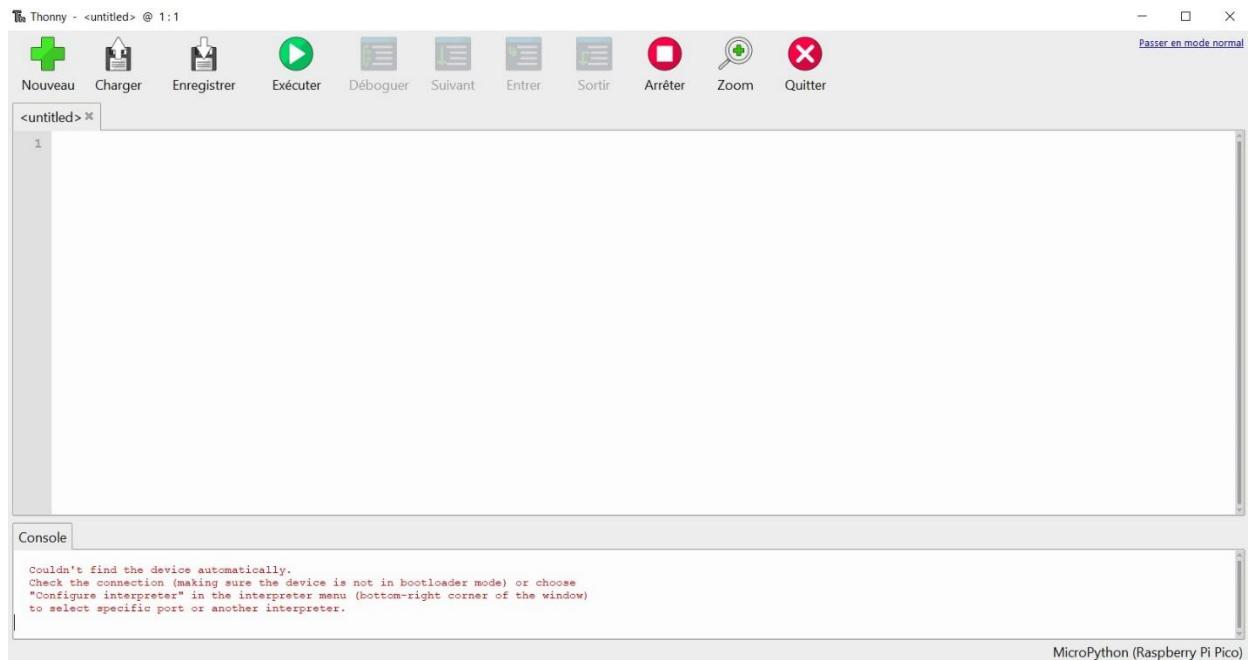


Figure 79 : Ouverture de l'IDE Thonny

12.2. Configuration de l'IDE Thonny

La configuration de l'IDE nécessite de paramétriser l'interpréteur Python ou bien de préciser la cible matérielle sur laquelle le code sera exécuter. Pour cela, il faut sélectionner le mode normal en cliquant sur le lien situé en haut à droite de la fenêtre de l'IDE, pour aboutir à l'affichage ci-dessous (cf. Figure 80).

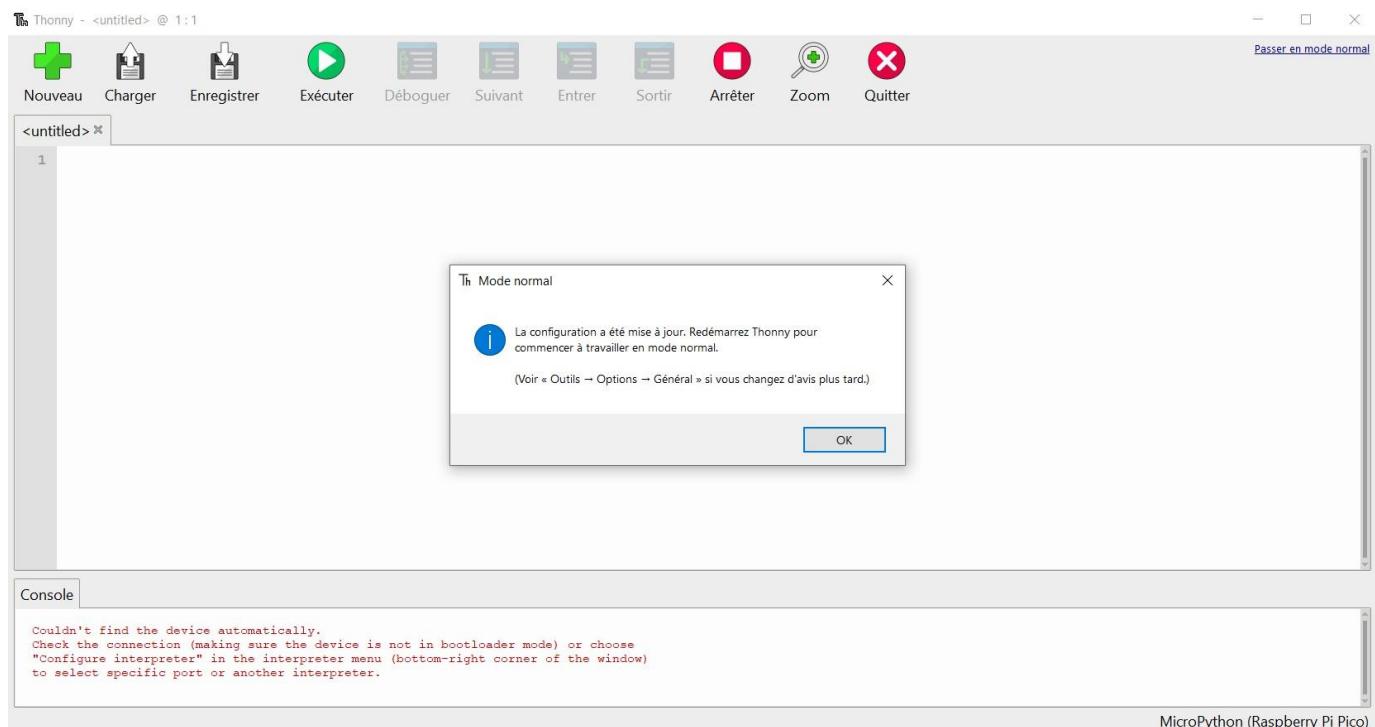


Figure 80 : IDE Thonny - mode normal

Après avoir redémarrer l'IDE, il devient possible d'accéder aux paramétrages associé au choix de la cible en sélectionnant dans *Outils* → *Options* → *Interpréteur* puis de choisir MicroPython (Raspberry Pi Pico) (cf. Figure 81). Cliquer sur *Ok* pour valider le choix.

Il est aussi nécessaire de modifier les paramètres d'affichage de l'IDE afin de faire apparaître l'ensemble des fichiers. Cela permet par la suite de visualiser les fichiers stockés dans la mémoire du microcontrôleur ainsi que l'arborescence des répertoires. Pour cela *Affichage* → *Console* (cf. Figure 82).

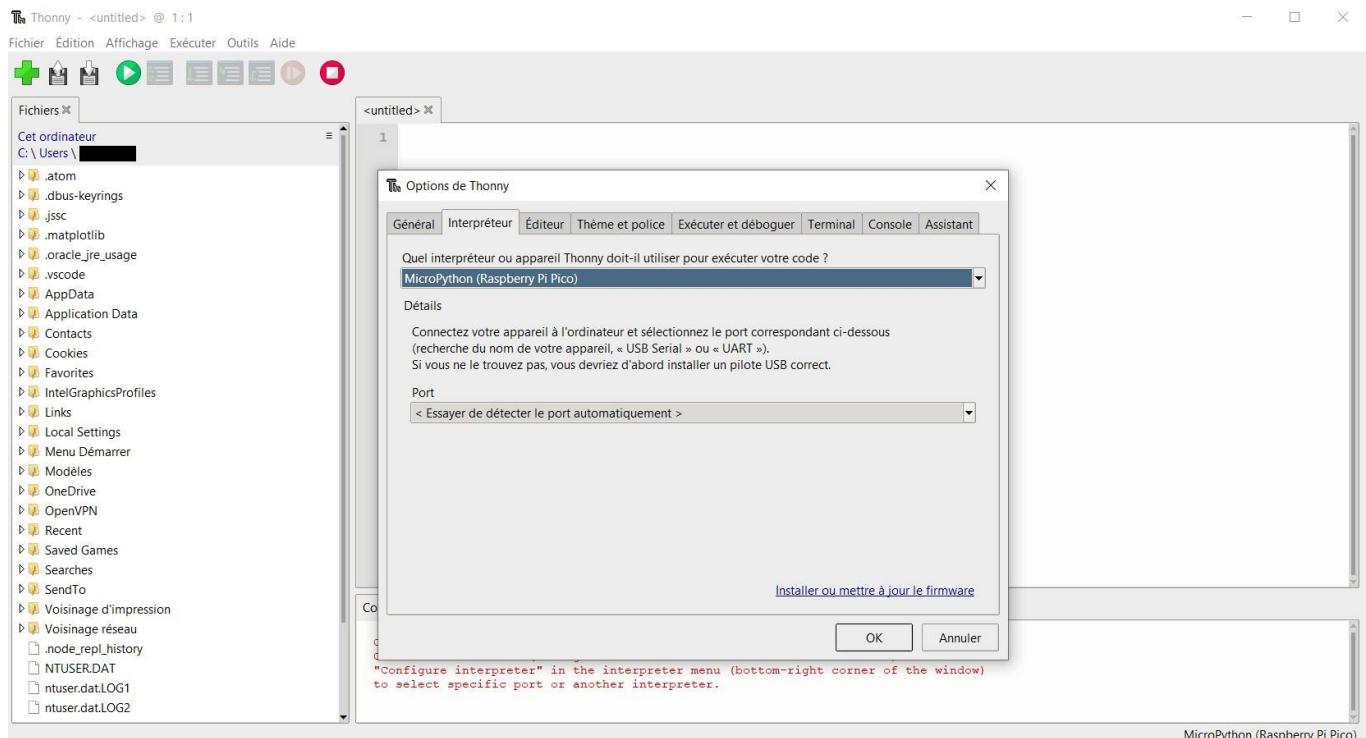


Figure 81 : IDE Thonny : paramétrage cible Raspberry Pi pico

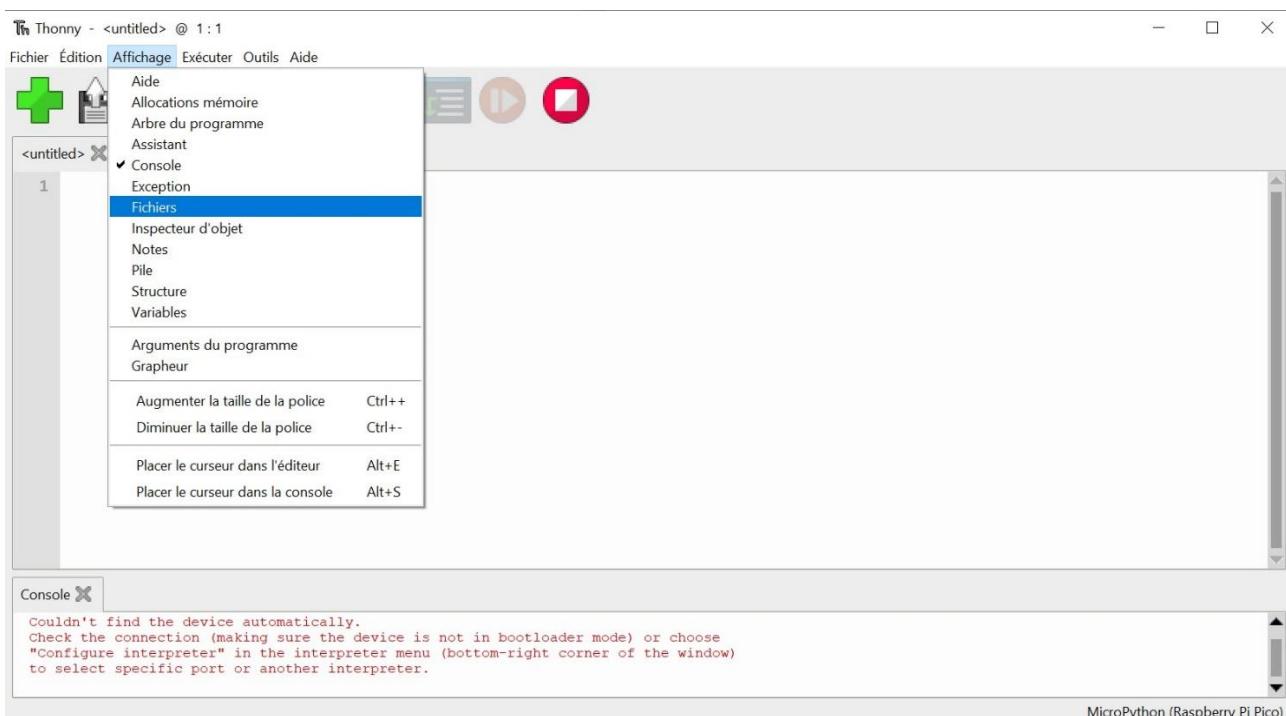


Figure 82 : IDE Thonny : paramétrage - Affichage fichiers

L’IDE est alors configuré et prêt à être utilisé (cf. Figure 83).

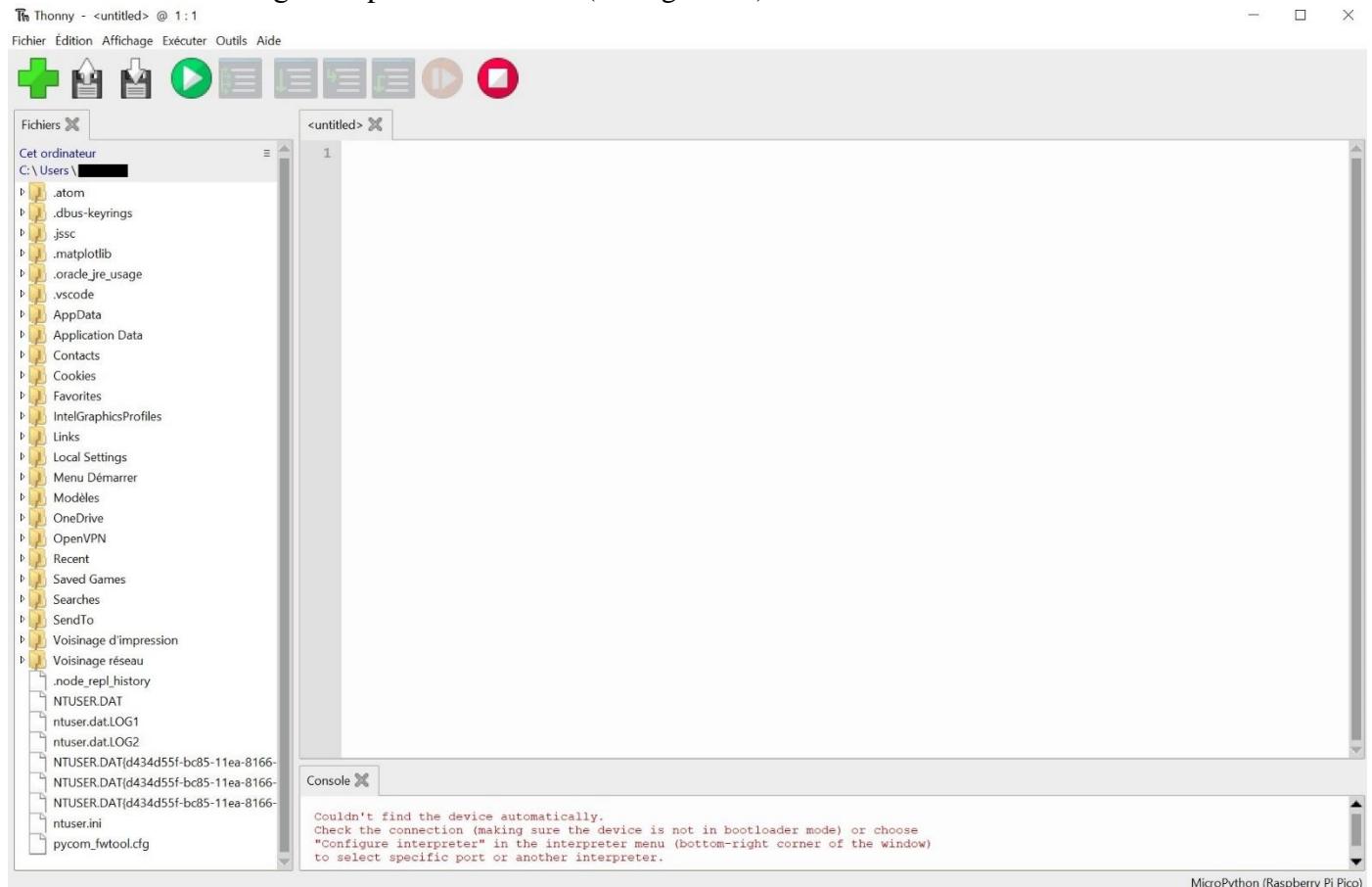


Figure 83 : IDE Thonny : paramétrage - Fin

12.3. 11.3 Interface de l’IDE Thonny

L’interface de l’IDE Thonny comporte 4 zones

12.4. Installation de microPython sur la carte Raspberry Pi Pico

La seconde étape consiste à charger sur le microcontrôleur Raspberry Pi Pico l’interpréteur Python sous la forme d’un firmware¹⁵ ou micro logiciel. Les procédures décrites ci-dessous permettent de charger le firmware pour la première fois ou bien d’effectuer une mise à jour de version. Nous pouvons utiliser l’une des 2 procédures décrites ci-dessous.

12.4.1. Mise à jour firmware depuis l’IDE Thonny

En retournant dans les options de l’IDE Thonny associés à l’interpréteur (cf. Figure 84) et en activant le lien Installer ou mettre à jour le firmware, une nouvelle fenêtre s’ouvre (cf. Figure 85). Son contenu décrit :

- La procédure à mettre en œuvre pour effectuer l’installation ou la mise à jour du firmware.
- La version de MicroPython qui sera installée, ici la v1.18.

¹⁵ Cf. <https://fr.wikipedia.org/wiki/Firmware>.

Il faut connecter la carte Raspberry Pi Pico sur un port USB en appuyant au préalable sur le bouton BOOTSEL. Lorsque le câble USB est branché, relâcher le bouton BOOTSEL. L'onglet *Installer* devient actif et un dossier RPI-RP2 s'ouvre dans l'explorateur Windows (cf . Figure 86). C'est dans ce dossier que le firmware sera chargé. Cliquer sur *Installer* et attendre la fin du chargement du firmware. Le dossier RPI-RP2 se ferme alors automatiquement et la fin du téléchargement conduit à la fenêtre ci-dessous (cf. Figure 87). La carte est prête pour accueillir nos programmes.

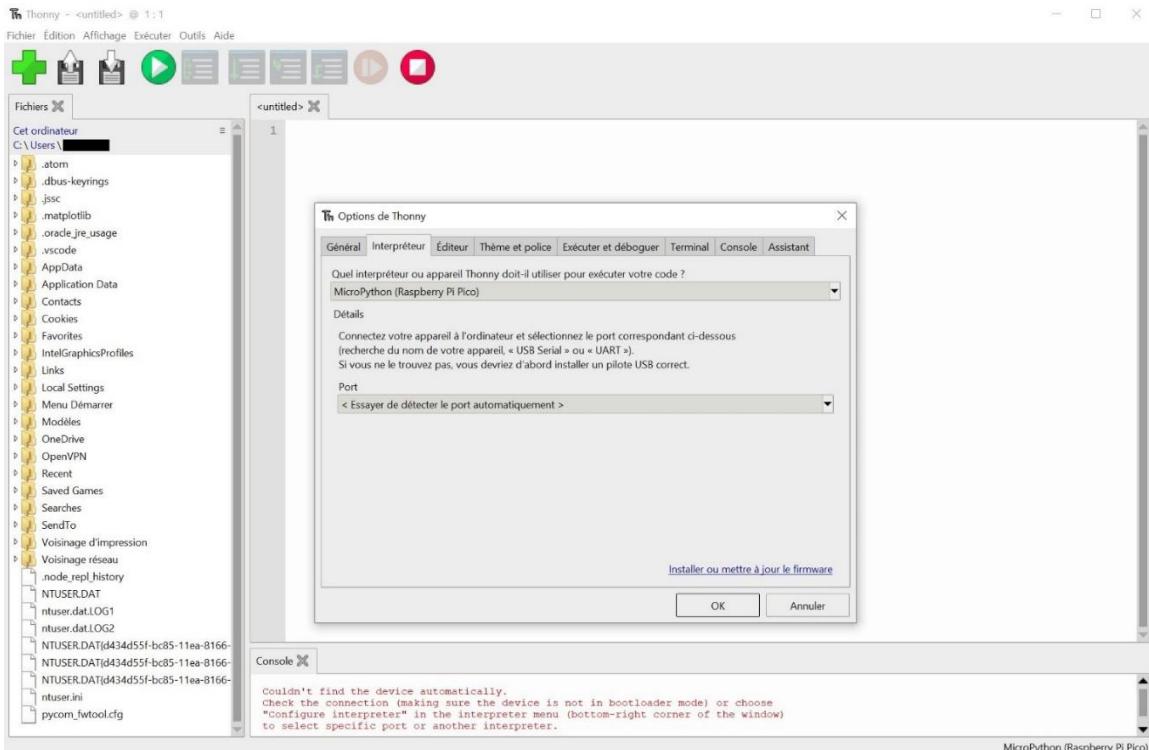


Figure 84: Chargement firmware micropython (1)

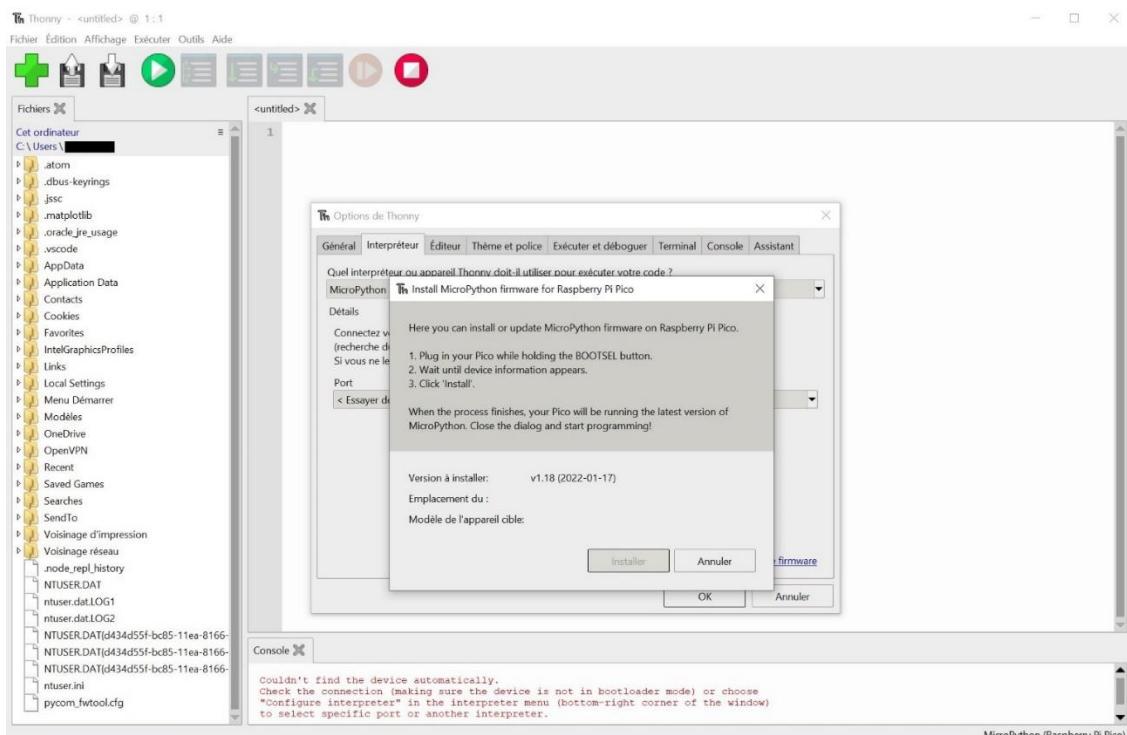


Figure 85 : Chargement firmware micropython (2)

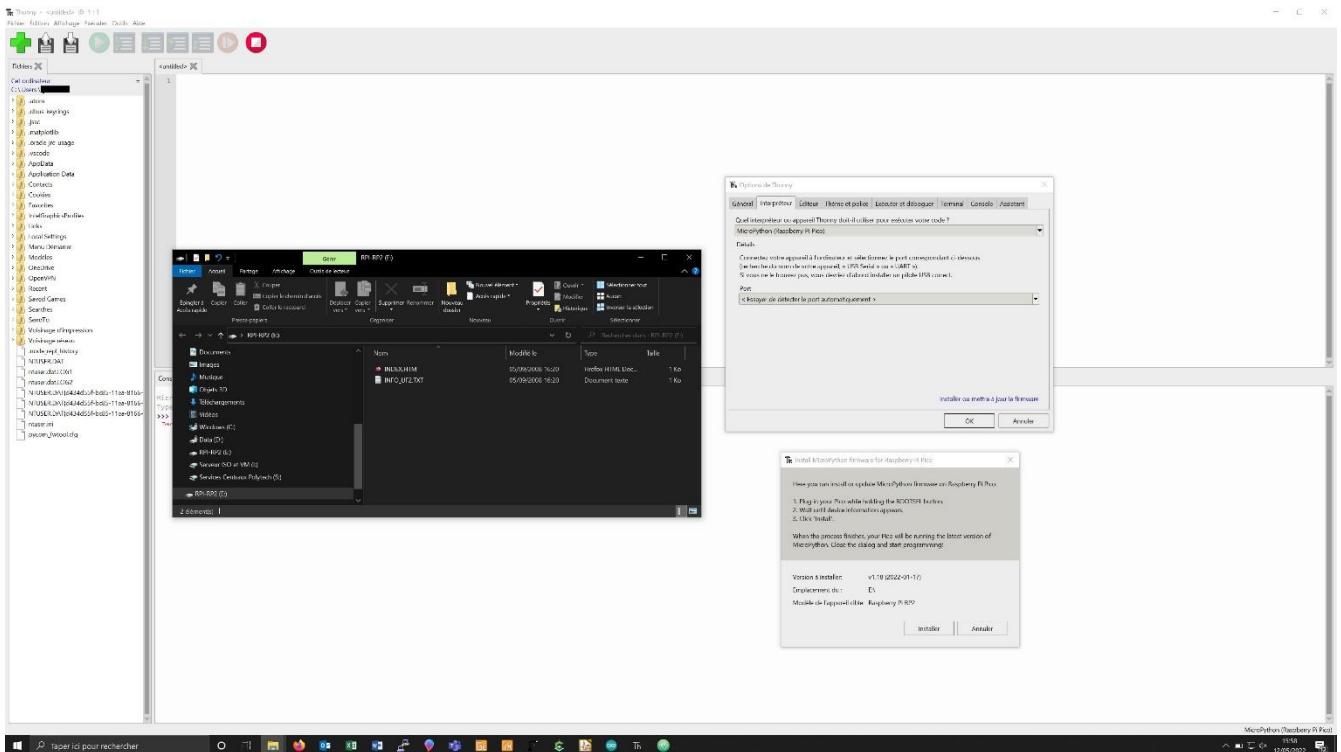


Figure 86 : Chargement firmware micropython (3)

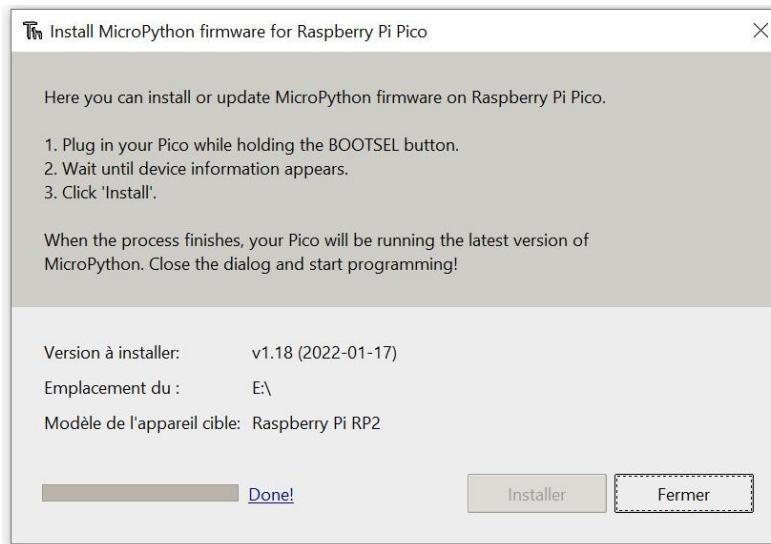


Figure 87 : Chargement firmware micropython (4)

12.4.2. Mise à jour firmware indépendante de l'IDE Thonny

Il est également possible de charger le firmware MicroPython par une procédure indépendante de l'IDE Thonny. Pour cela, il faut récupérer le fichier via le lien <https://micropython.org/download/rp2-pico/> qui amène sur la page ci-dessous (cf. Figure 88). Télécharger le fichier .uf2 correspondant à la dernière version du firmware (version stable) et l'enregistrer (cf. Figure 89)



Vendor: Raspberry Pi
Features: Breadboard friendly, Castellated Pads, Micro USB
Source on GitHub: [rp2/PICO](#)
More info: [Website](#)

Installation instructions

Flashing via UF2 bootloader

To get the board in bootloader mode ready for the firmware update, execute `machine.bootloader()` at the MicroPython REPL. Alternatively, hold down the BOOTSEL button while plugging the board into USB. The uf2 file below should then be copied to the USB mass storage device that appears. Once programming of the new firmware is complete the device will automatically reset and be ready for use.

Firmware

Releases

[v1.18 \(2022-01-17\) .uf2 \[Release notes\] \(latest\)](#)
 v1.17 (2021-09-02) .uf2 [Release notes]
 v1.16 (2021-06-18) .uf2 [Release notes]
 v1.15 (2021-04-18) .uf2 [Release notes]
 v1.14 (2021-02-02) .uf2 [Release notes]

Nightly builds

[v1.18-421-gc3c880e56 \(2022-05-06\) .uf2](#)
[v1.18-420-gf1e72580fd \(2022-05-05\) .uf2](#)
[v1.18-416-g1216c9ff \(2022-05-05\) .uf2](#)
[v1.18-415-gda31adfaa \(2022-05-05\) .uf2](#)

Figure 88: :Chargement firmware micropython (5)

Ouverture de rp2-pico-20220117-v1.18.uf2 X

Vous avez choisi d'ouvrir :

 **rp2-pico-20220117-v1.18.uf2**

qui est un fichier de type : Fichier UF2 (569 Ko)
 à partir de : <https://micropython.org>

Que doit faire Firefox avec ce fichier ?

Ouvrir avec Applications\wordpad.exe (par défaut) ▼
 Enregistrer le fichier

OK Annuler

Figure 89 : Chargement firmware micropython (6)

De nouveau, faut connecter la carte Raspberry Pi Pico sur un port USB en appuyant au préalable sur le bouton BOOTSEL. Lorsque le câble USB est branché, relâcher le bouton BOOTSEL. Le répertoire RPI-RP2 s'ouvre. (cf. Figure 90).

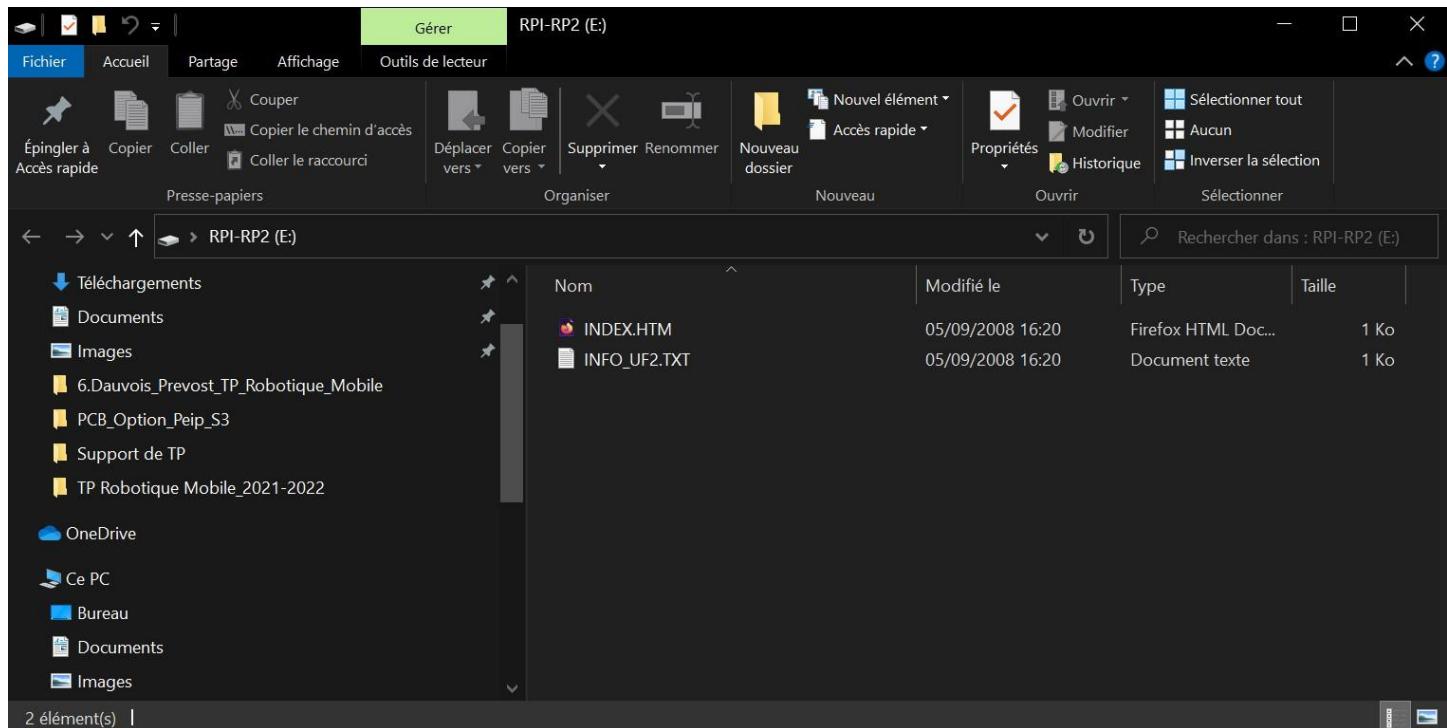


Figure 90 : Chargement firmware micropython (7)

Faire alors glisser le fichier rp2-pico-20220117-v1.18.uf2 pour le copier dans le répertoire RPI-RP2 (ou bien Copier – Coller).

Compléments :

A la racine du lecteur logique RPI-RP2, 2 fichiers sont présents :

- INDEX.HTM
- INFO_UF2.TXT

Le fichier INFO_UF2.TXT contient les informations suivantes :

UF2 Bootloader v1.0

Model: Raspberry Pi RP2

Board-ID: RPI-RP2

Rq : format de BOARD-ID : <CPU type> - <Board type>

Le fichier INDEX.HTM permet d'accéder au contenu de la page de démarrage sur la mise en œuvre de la carte au travers du lien <https://www.raspberrypi.org/documentation/rp2040/getting-started/>

12.5. Diagramme d'E/S des broches de la carte Raspberry Pi Pico

La figure ci-après (cf. Figure 91) représente l'ensemble des fonctionnalités que chaque broche du microcontrôleur peut assurer. Une même broche peut assurer plusieurs fonctions. Le choix de la fonctionnalité assurée dépend de la configuration des ressources du microcontrôleur.

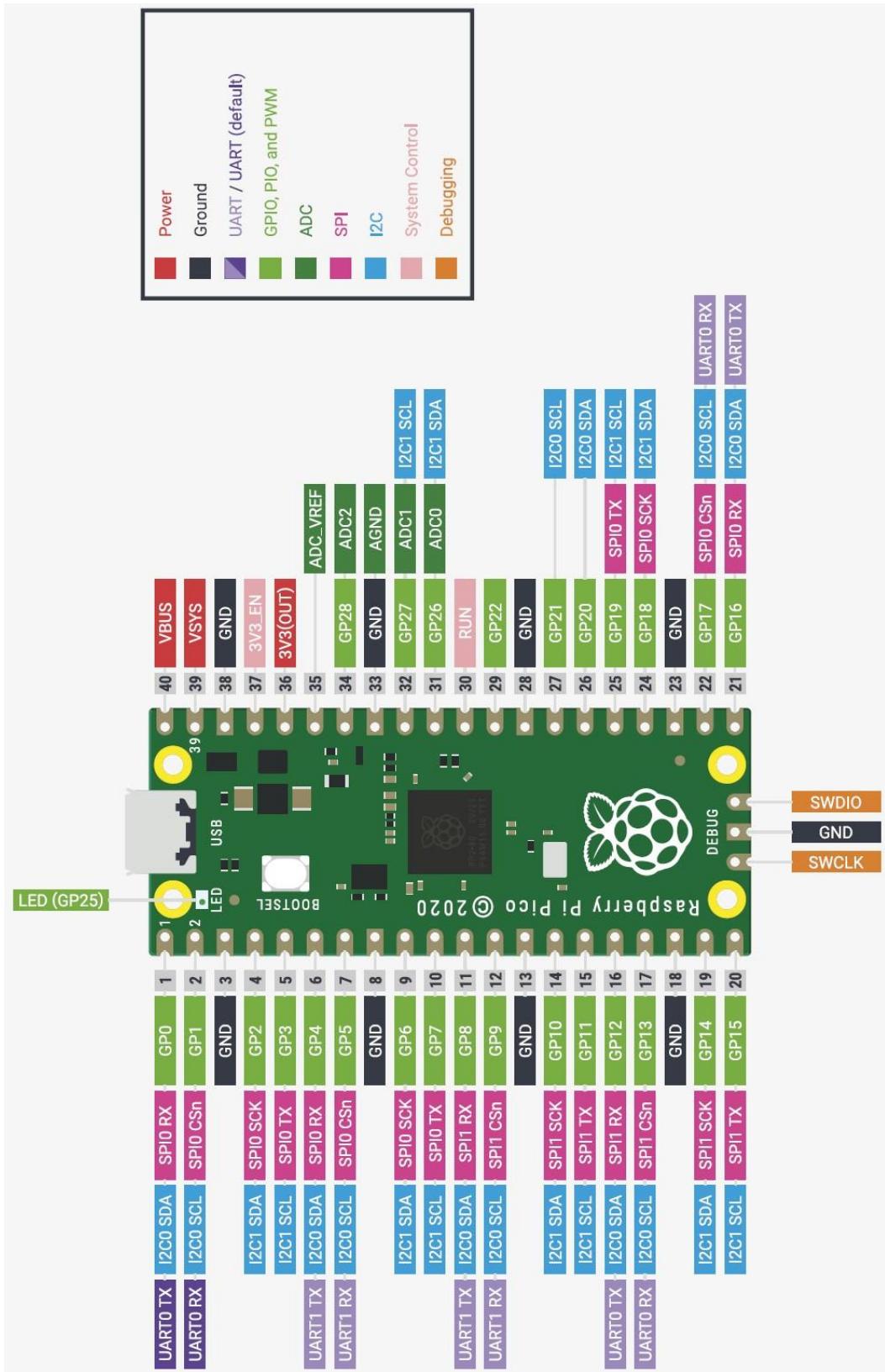


Figure 91 : Diagramme d'E/S des broches de la carte Raspberry Pi pico

12.6. Le bus I2C

12.6.1. Bus de communication

Un bus informatique est un dispositif de transmission de données partagé entre plusieurs composants d'un système numérique. Le terme dérive du latin *omnibus* (à tous) ; c'est le sens, d'un usage plus ancien, du terme bus en électronique. Le bus informatique est la réunion des parties matérielles et immatérielles qui permet la transmission de données entre les composants participants¹⁶.

Afin de simplifier la mise en œuvre aussi bien matérielle que logicielle, la communication entre les capteurs et la carte microcontrôleur sera réalisé uniquement en utilisant un bus particulier : le bus I2C.

Un autre bus de communication, le bus SPI, sera utilisé notamment pour gérer l'affichage sur un écran.

12.6.2. Le bus I2C

Le bus I2C¹⁷ ¹⁸a été proposé par Philips en 1982. La version 1.0 a été publiée en 1992, la version 2.0 en 1998, la version 2.1 en 2000, la version 3.0 en 2007, la version 4 en février 2012, la version 5 en octobre 2012 et la 6^e version en avril 2014. Le document de référence d'écrivant à la fois les aspects matériels et logiciels de la mise en œuvre d'une communication I2C se trouve au sein du document de référence : « *The I2C bus specification, version 2.1*, janvier 2000 (number 9398-393-40011) », Phillips.

Pour établir une connexion I2C, il faut 2 fils (et une masse), dont les noms sont :

- SDA (Serial Data Line) : ligne de données bidirectionnelle,
- SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Chaque ligne peut transmettre un niveau haut ou bas électrique, ce qui se traduit par (cf. Figure 92):

- La donnée 1 ou 0 sur la ligne SDA
- Une impulsion d'horloge sur la ligne SCL

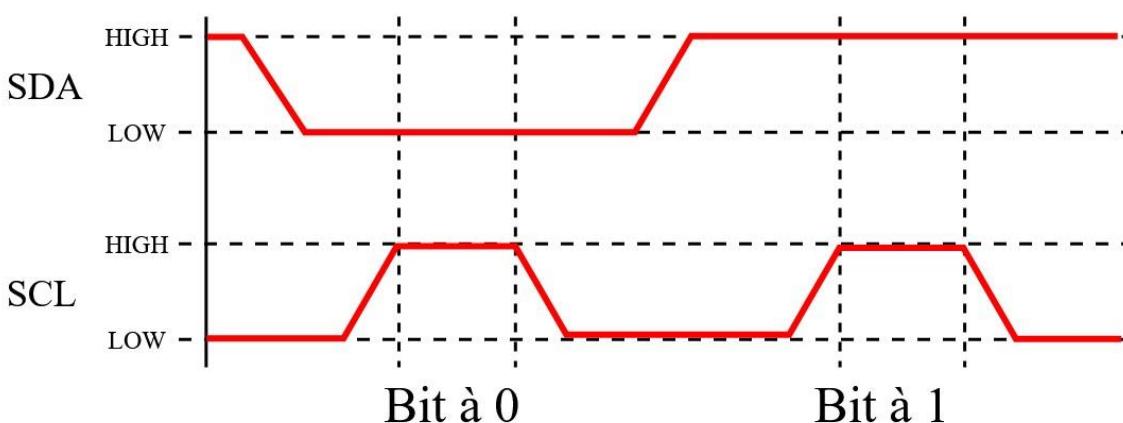


Figure 92 : Communication sur le bus I2C

¹⁶ https://fr.wikipedia.org/wiki/Bus_informatique.

¹⁷ I2C : Inter-Integrated Circuit

¹⁸ <https://fr.wikipedia.org/wiki/I2C>.

12.6.3. Le bus SPI

Une liaison **SPI** (pour *Serial Peripheral Interface*) est un bus de données série synchrone baptisé ainsi par Motorola, au milieu des années 1980¹ qui opère en mode full-duplex. Les circuits communiquent selon un schéma maître-esclave, où le maître contrôle la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée « *Slave Select (SS)* »¹⁹.

Le bus SPI utilise quatre signaux logiques (cf. Figure 93):

- SCLK : Serial Clock, Horloge (généré par le maître).
- MOSI : Master Output, Slave Input (généré par le maître).
- MISO : Master Input, Slave Output (généré par l'esclave).
- SS : Slave Select, Actif à l'état bas (généré par le maître).

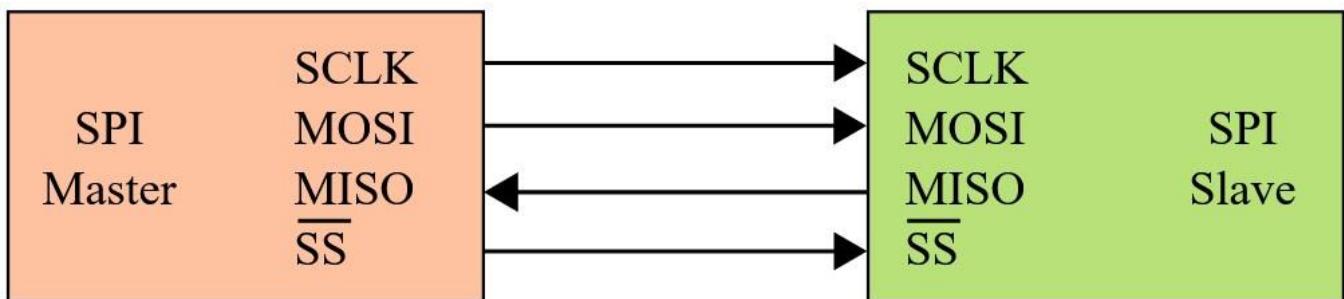


Figure 93 : Principe de la communication sur le bus SPI

12.7. Formatage de chaîne de caractères : *.format*

Micropython supporte le formatage de chaîne de caractères. Les liens ci-dessous renvoient vers des ressources documentaires.

<https://www.programiz.com/python-programming/methods/string/format>
https://www.w3schools.com/python/ref_string_format.asp

Afin de comprendre la syntaxe à mettre en œuvre, il est possible dans la console de l'éditeur Thonny de faire tourner l'exemple suivant (cf. Figure 94) :

¹⁹ Référence bus SPI : Wikipédia : https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface.

```

Console ✘

>>> format_str = "{:+5.1f}"
>>> mesure = 23.4
>>> s = format_str.format(mesure)
>>> print (s)
+23.4

>>> mesure = 9.5
>>> s = format_str.format(mesure)
>>> print (s)
+9.5

>>> mesure = 0.5
>>> s = format_str.format(mesure)
>>> print (s)
+0.5

>>> mesure = -5.4
>>> s = format_str.format(mesure)
>>> print (s)
-5.4

>>>

```

Figure 94 : Micropython - formatage de chaînes de caractères

12.8. Librairie graphique ili9341 : fonctions graphiques de base

Ci-dessous sont décrites les entêtes de quelques fonctions graphiques de base de la librairie graphique fournie. Pour les autres fonctions accessibles, se référer au contenu du fichier *ili9341.py*.

- **Tracer un cercle :**

```
draw_circle(self, x0, y0, r, color):
    """Draw a circle.
```

Args:

x0 (int): X coordinate of center point.
y0 (int): Y coordinate of center point.
(r): Radius.
color (int): RGB565 color value.
"""

- **Tracer une ligne - segment de droite horizontale**

```
draw_hline(self, x, y, w, color):
    """Draw a horizontal line.
```

Args:

x (int): Starting X position.
y (int): Starting Y position.
w (int): Width of line.
color (int): RGB565 color value.
"""

- **Tracer un rectangle**

```
draw_rectangle(self, x, y, w, h, color):
    """Draw a rectangle.
```

Args:

x (int): Starting X position.
y (int): Starting Y position.
w (int): Width²⁰ of rectangle.
h (int): Height²¹ of rectangle.
color (int): RGB565 color value.

"""

- Afficher du texte

```
draw_text(self, x, y, text, font, color, background_flag = True, background=0,
          landscape = False, spacing = 1):
    """Draw text.
```

Args:

x (int): Starting X position.
y (int): Starting Y position.
text (string): Text to draw.
font (XglcdFont object): Font.
color (int): RGB565 color value.
background (int): RGB565 background color (default: black).
landscape (bool): Orientation (default: False = portrait)
spacing (int): Pixels between letters (default: 1)

"""

"""

Modif : afficher texte avec gestion explicite de la couleur de fond d'écran

Args :

background_flag : contrôle affichage de la couleur du fond d'écran :
 : True : valeur par défaut
 : False : inhiber la gestion de la couleur de fond d'écran

"""

Remarque : variante de *draw_text* : *draw_text2* qui possède les mêmes fonctionnalités que *draw_text* mais gestion optimisée de l'affichage des caractères

- Tracer une ligne - segment de droite vertical

```
draw_vline(self, x, y, h, color):
    """Draw a vertical line.
```

Args:

x (int): Starting X position.
y (int): Starting Y position.
h (int): Height of line.

²⁰ Width : largeur

²¹ Height : longueur

color (int): RGB565 color value.

"""

- **Remplir un disque avec une couleur**

`fill_circle(self, x0, y0, r, color):`

"""Draw a filled circle.

Args:

x0 (int): X coordinate of center point.

y0 (int): Y coordinate of center point.

r (int): Radius.

color (int): RGB565 color value.

"""

- **Remplir un rectangle avec une couleur**

`fill_rectangle(self, x, y, w, h, color):`

"""Draw a filled rectangle.

Args:

x (int): Starting X position.

y (int): Starting Y position.

w (int): Width of rectangle.

h (int): Height of rectangle.

color (int): RGB565 color value.

"""

12.9. Calcul des composantes RGB d'une couleur en fonction de la longueur d'onde

Cette partie détaille la procédure pour calculer les composantes RGB d'une couleur définie par la valeur de sa longueur d'onde λ .

12.9.1. Spectre des couleurs visibles

Le spectre des couleurs monochromatiques visibles, par l'œil humain est une portion du spectre électromagnétique sur la plage de longueur d'onde λ située dans l'intervalle [380 , 780] nm (cf. Figure 95)²².

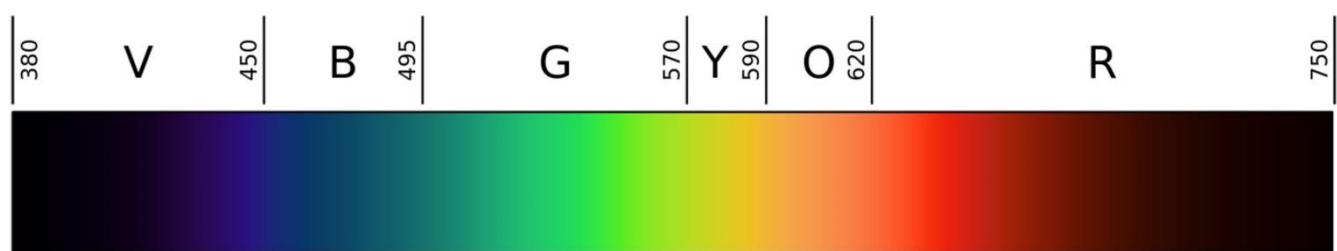


Figure 95 : Spectre des couleurs monochromatiques visibles

Pour $\lambda = 380$ nm : couleur violette.

Pour $\lambda = 780$ nm : couleur rouge.

²² Source : https://fr.wikipedia.org/wiki/Spectre_visible.

La figure ci-dessous (cf. Figure 96) détaille des plages de couleurs monochromatiques en fonction de leur plage correspondante de longueur λ .

Longueur d'onde (nm)	Champ chromatique	Couleur	Commentaire
380 — 449	Violet	445	primaire CIE 1931 435,8
449 — 466	Violet-bleu	455	primaire sRGB : 464
466 — 478	Bleu-violet	470	<i>indigo</i> entre le bleu et le violet (Newton)
478 — 483	Bleu	480	
483 — 490	Bleu-vert	485	
490 — 510	Vert-bleu	500	
510 — 541	Vert	525	
541 — 573	Vert-jaune	555	CIE 1931 : 546,1 ; primaire sRGB : 549.
573 — 575	Jaune-vert	574	
575 — 579	Jaune	577	
579 — 584	Jaune-orangé	582	
584 — 588	Orangé-jaune	586	
588 — 593	Orangé	590	
593 — 605	Orangé-rouge	600	
605 — 622	Rouge-orangé	615	primaire sRGB : 611
622 — 780	Rouge	650	primaire CIE 1931 : 700

Figure 96 : Table des plages de couleurs monochromatiques

12.9.2. Système de coordonnées de l'espace colorimétrique

Nous allons utiliser le système de couleur sRGB²³ – standard RGB – (cf.) qui est l'espace de couleur par défaut proposé pour les dispositifs numériques, notamment les écrans d'ordinateurs. Une norme définie par la CIE - Commission Internationale de l'Eclairage - - [CIE 61966-2-1 \(1999\)](#) définit l'espace de couleurs sRGB comme « un espace chromatique commun pour le stockage », de façon que les équipements d'origine et de système d'exploitation divers puissent communiquer simplement. Les valeurs des composantes dans l'espace sRGB sont comprises entre 0 et 1. L'espace des couleurs est représenté par un diagramme de chromacité qui permet de définir l'espace des couleurs qui pourra être affiché (cf. Figure 97).

²³ Cf. https://fr.wikipedia.org/wiki/SRGB#Transformation_de_CIE_XYZ_vers_sRGB.

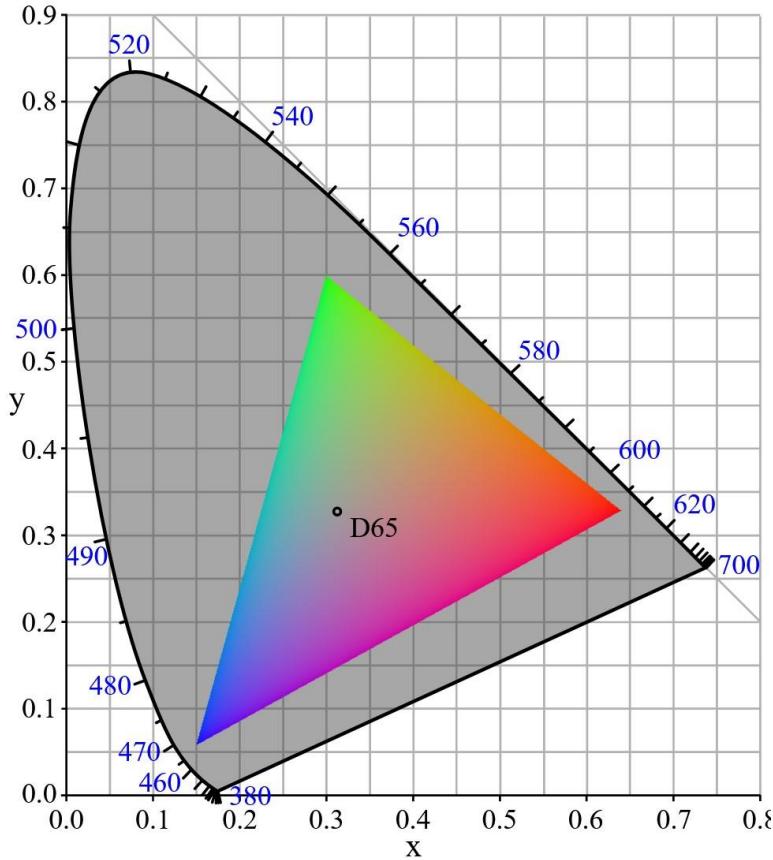


Figure 97 : Diagramme CIE sRGB

Pour une couleur monochromatique, définie par la valeur de sa longueur d'onde λ , on lui associe ses composantes chromatiques $x(\lambda)$, $y(\lambda)$ et $z(\lambda)$, qui représente respectivement la proportion de Rouge, Vert et Bleu qui contribueront à définir la couleur à afficher. Les valeurs des composantes chromatiques sont tabulées (voir ci-après).

12.9.3. Calcul des composantes RGB pour λ donnée

Dans un premier temps, nous supposons que les valeurs R, G et B sont codées sur 8 bits, pour des valeurs allant de 0x00 à 0xFF (255). Cela signifie qu'une couleur est codé sur 3 octets ou 24bits. A partir de la donnée d'une valeur de λ , on veut calculer la valeur de chaque composante R, G et B. Pour le système sRGB, nous disposons d'une matrice de transformation telle que :

$$\begin{bmatrix} R_l \\ G_l \\ B_l \end{bmatrix} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix} \begin{bmatrix} x(\lambda) \\ y(\lambda) \\ z(\lambda) \end{bmatrix}$$

Où :

- $x(\lambda)$, $y(\lambda)$ et $z(\lambda)$, composantes chromatiques interpolées depuis la table CIE 1964.
- R_l , G_l et B_l : valeur linéarisée de Rouge, Green et Blue. Ces valeurs peuvent être en dehors de la plage [0.0 ; 1.0] ce qui signifie dans ce cas qu'elles ne définissent pas une couleur que l'on peut afficher. Il faut appliquer une seconde transformation qui est la correction Gamma définie par :

$$C_{sRGBl} = \begin{cases} 12.92 * C_l \text{ si } C_l \leq 0.0031308 \\ (1 + a) * C_l^{\frac{1}{2.4}} - a \text{ si } C_l > 0.0031308 \end{cases}$$

Avec $a = 0.055$

où C_l représente R_l ou G_l ou B_l

Puis au final, on tronque chaque valeur corrigée C_l pour la ramener dans l'intervalle [0.0 ; 1.0].

Enfin, en fonction de la profondeur de codage des composantes R, G et B (ici 8 bits donc valeur max = 255=, on calcule le niveau de R, G ou B par :

$R, G, B = C_l * 255$ arrondi à l'entier le plus proche avec C_l dans [0.0 ; 1.0].

Remarque : l'écran utilisé utilise un codage RGB des couleurs dit RGB565. Cela signifie que :

- La composante R est codée sur 5 bits ;
- La composante G est codée sur 6 bits ;
- La composante B est codée sur 5 bits ;

afin de former un entier sur 2 octets (cf. [partie 11](#) Gestion d'un afficheur graphique) selon la transformation :

$$(R \& 0xf8) << 8 | (G \& 0xfc) << 3 | B >> 3$$

12.9.4. Table des composantes chromatiques et interpolation linéaire

Afin de calculer la valeur des composantes RGB depuis la donnée de sa longueur d'onde λ , il faut accéder à la valeur des composantes chromatiques $x(\lambda)$, $y(\lambda)$ et $z(\lambda)$, pour une valeur de longueur λ . Ces valeurs sont accessibles au sein de tables. Nous utiliserons la table CIE1964 pour λ dans [380, 780], avec un pas de 5nm. Cette table contient 81 éléments pour chaque composante chromatique, la première valeur étant associée à la valeur de longueur d'onde $\lambda = 380$ nm et la dernière à la valeur de longueur d'onde $\lambda = 780$ nm.

Pour des valeurs de longueur λ qui ne sont pas présentes dans la table CIE, la valeur des composantes chromatiques $x(\lambda)$, $y(\lambda)$ et $z(\lambda)$ sera calculée en procédant par une interpolation linéaire.

Soit une valeur de λ dont la valeur est située dans l'intervalle [380, 780] nm. Plus précisément, au sein de la table CIE1964, la valeur de λ est encadrée par les valeur λ_1 et λ_2 , auxquelles sont associées les valeurs de la composante chromatique $x(\lambda_1)$ et $x(\lambda_2)$ (idem pour les composantes $y(\lambda)$ et $z(\lambda)$) (cf. Figure 98).

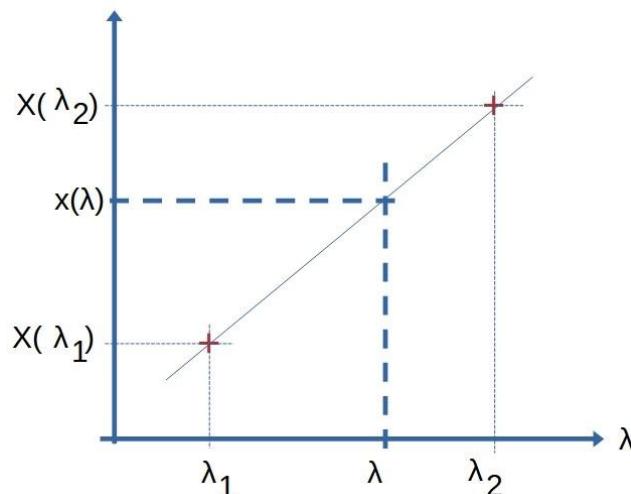


Figure 98 : Interpolation linéaire

Par interpolation linéaire, à l'aide d'une fonction affine $y = ax + b$, il est possible de calculer la valeur de la composante $x(\lambda)$ (respectivement $y(\lambda)$ et $z(\lambda)$), connaissant les valeurs de longueur d'onde les valeur λ_1 et λ_2 ainsi que les 2 valeurs de la composante chromatique $x(\lambda)$, soient $x(\lambda_1)$ et $x(\lambda_2)$.

Par identification, on a alors :

$$x(\lambda) = x(\lambda_1) + (\lambda - \lambda_1) \frac{x(\lambda_2) - x(\lambda_1)}{\lambda_2 - \lambda_1}$$

12.10. Schéma électronique de la plateforme matérielle

Le schéma électronique de la plateforme matérielle est disponible ci-après (cf. Figure 99) (réalisé sous Kicad 5.1.10). Il vous permettra de retrouver notamment les éléments de configuration matérielle contenu dans le fichier *ConfigMateriel_pico.py*.

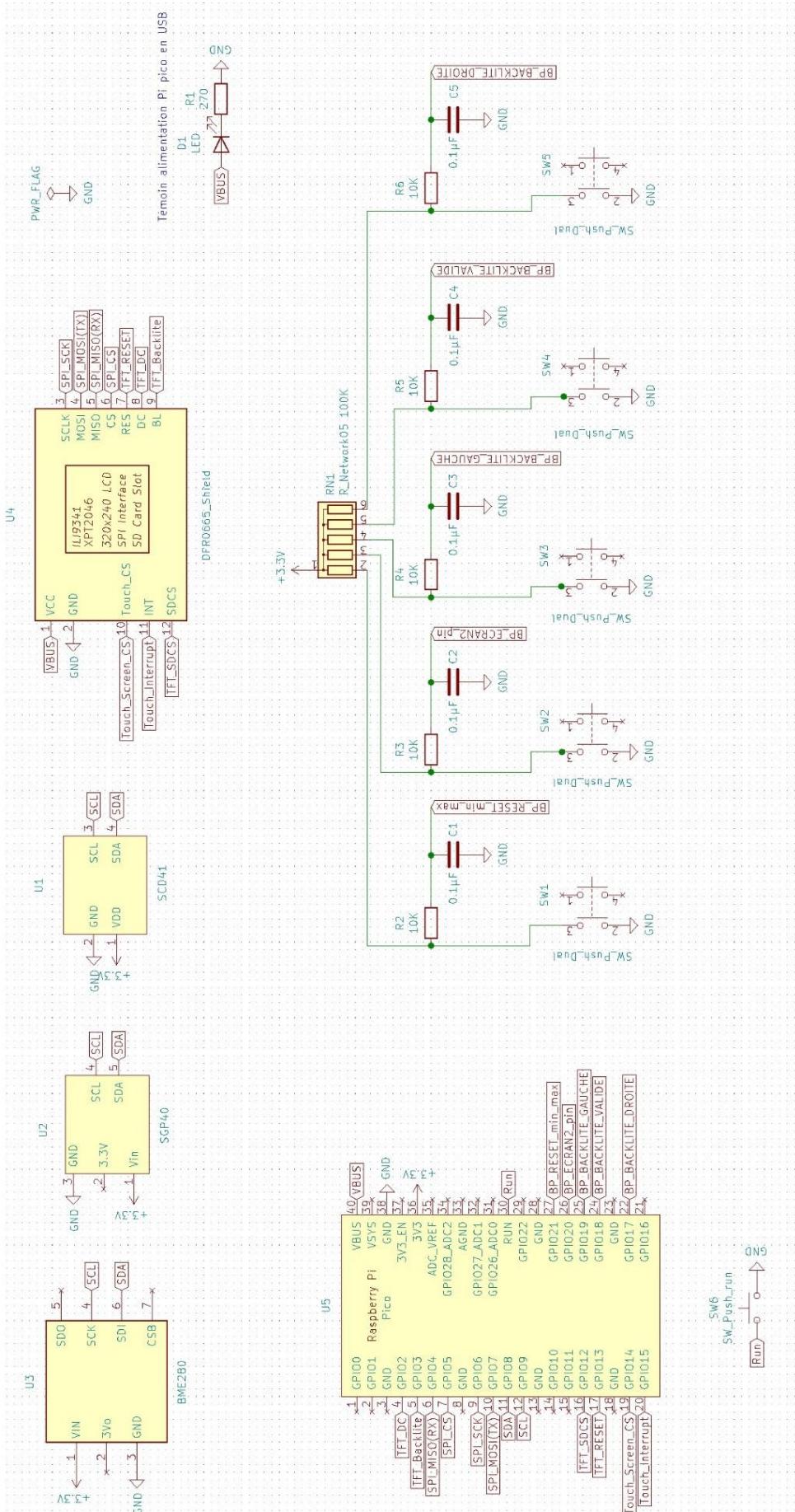


Figure 99 : Schéma de la plateforme matérielle "Mesures environnementales"

12.11. Tables des figures

Figure 1 : PCB plateforme matérielle - vue de dessus -----	7
Figure 2: Carte Raspberry Pi pico - recto - verso -----	7
Figure 3: Fonctionnalités des ports GPIO Raspberry Pi pico -----	8
Figure 4: Capteur de CO ₂ SCD41 sur dev. Board-----	9
Figure 5: Shield Adafruit pour SGP40 -----	10
Figure 6: Echelle colorimétrique de mesure de l'index COV -----	10
Figure 7: Shield Adafruit pour BME280-----	11
Figure 8: Ecran TFT et dalle tactile DFR0665 – vue de dessus -----	11
Figure 9 : Ecriture du code source dans l'éditeur de Thonny -----	15
Figure 10 : Activer le terminal série de l'IDE Thonny -----	16
Figure 11 : Création du répertoire lib (1) -----	17
Figure 12 : Création du répertoire lib (2) -----	18
Figure 13 : Ouvrir ConfigMateriel_pico.py (1) -----	18
Figure 14 : Ouvrir ConfigMateriel_pico.py (2) -----	19
Figure 15 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (1) -----	19
Figure 16 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (2) -----	20
Figure 17 : Charger librairie ConfigMateriel_pico.py dans répertoire lib (3) -----	20
Figure 18 : Charger main.py dans la mémoire du microcontrôleur -----	21
Figure 19 : Lancement de main.py-----	21
Figure 20 : Arrêt exécution programme par Ctrl + C -----	22
Figure 21 : Signal PWM - principe -----	23
Figure 22: Interruption Timer (1) -----	25
Figure 23 : Interruption Timer (2) -----	26
Figure 24 : Interruption Timer (3) -----	27
Figure 25 : Shield BME280 : broches -----	28
Figure 26 : Mise en place capteur BME280 sur PCB -----	28
Figure 27: Ouverture du fichier BME280.py dans IDE Thonny -----	29
Figure 28 : Configuration du répertoire lib après chargement de la librairie BME280.py -----	29
Figure 29 : Données du capteur BME280 -----	32
Figure 30 : Mise en place du capteur de CO ₂ SCD41 -----	33
Figure 31 : enregistrement de la librairie SCD41.py dans le répertoire lib -----	34
Figure 32 : Contenu du répertoire lib après chargement de la librairie SCD41.py -----	34
Figure 33 : Mesures du taux de CO ₂ avec capteur SCD41 -----	36
Figure 34 : Mesures conjointes capteur BME280 et SCD41-----	38
Figure 35 : Mise en place du capteur SGP40-----	39
Figure 36 : Insertion des librairies associées au capteur SGP40 -----	40
Figure 37: Affichage des mesures du capteur SGP40 VOC_index (1)-----	42
Figure 38 : Affichage des mesures du capteur SGP40 VOC_index (2) -----	42
Figure 39: Mesures conjointes BME280 - SCD41 et SGP40-----	44
Figure 40 : Ecran DFR0665 (1) -----	45
Figure 41 : Ecran DFR0665 (2) -----	45
Figure 42 : Codage RGB565 -----	45
Figure 43 : Référentiel de coordonnées dalle graphique TFT écran DFR0665 -----	47
Figure 44 : Prise en compte de polices de caractères -----	48
Figure 45 : Test affichage graphique sur l'écran DFR0665 -----	50
Figure 46 : Constructeur du module Affichage_Graphique.py -----	51
Figure 47 : En tête fonction affichage d'une mesure capteur -----	52
Figure 48 : Exemple d'affichage des mesures capteurs - Ecran1 -----	53
Figure 49 : Exemple de mise en forme Ecran2 -----	54
Figure 50 : Boutons poussoir carte PCB -----	55

Figure 51 : Mécanisme d'interruption -----	56
Figure 52 : Circuit anti rebond associé à un interrupteur -----	57
Figure 53: Front descendant -----	57
Figure 54: Routine d'interruption associée aux boutons pousoir -----	59
Figure 55 : Gestion de la transition de l'écran 1 vers l'écran 2-----	59
Figure 56 : Fonctionnalités gérées depuis les BP-----	61
Figure 57 : Référentiels écran TFT et dalle tactile -----	62
Figure 58 : Prise en compte de la librairie dédié à la gestion de la dalle tactimle -----	63
Figure 59 : Calibration de la dalle tactile (1) -----	64
Figure 60: : Calibration de la dalle tactile (2) -----	64
Figure 61 : : Calibration de la dalle tactile (3) -----	64
Figure 62: Vérification du mapping écran TFT et dalle tactile -----	65
Figure 63 : Routine d'interruption associée à la dalle tactile-----	65
Figure 64 : Création des objet tft et Touchsreen -----	66
Figure 65 : Ecran TFT en mode paysage (rotation = 90) -----	66
Figure 66 : Définition de zones écran TFT -----	67
Figure 67: Définition zones écran TFT -----	67
Figure 68: Gestion de la dalle tactile - routine d'interruption -----	68
Figure 69 : Gestion des fonctionnalités utilisateurs swap écran 1 – écran 2-----	69
Figure 70 : Connexion sur le site thonny.org-----	70
Figure 71 : Thonny - téléchargement de l'installer Windows (1)-----	71
Figure 72 : Thonny - téléchargement de l'installer Windows (2)-----	71
Figure 73 : Lancement de l'installer Windows de Thonny -----	72
Figure 74 : Procédure d'installation de thonny sous Windows (1)-----	72
Figure 75 : Procédure d'installation de thonny sous Windows (2)-----	73
Figure 76 : Procédure d'installation de thonny sous Windows (3)-----	73
Figure 77 : Procédure d'installation de thonny sous Windows (4)-----	74
Figure 78 : Procédure d'installation de thonny sous Windows (5)-----	74
Figure 79 : Ouverture de l'IDE Thonny -----	75
Figure 80 : IDE Thonny - mode normal -----	75
Figure 81 : IDE Thonny : paramétrage cible Raspberry Pi pico-----	76
Figure 82 : : IDE Thonny : paramétrage - Affichage fichiers-----	76
Figure 83 : : IDE Thonny : paramétrage - Fin-----	77
Figure 84: Chargement firmware micropython (1)-----	78
Figure 85 : Chargement firmware micropython (2)-----	78
Figure 86 : Chargement firmware micropython (3)-----	79
Figure 87 : Chargement firmware micropython (4)-----	79
Figure 88: :Chargement firmware micropython (5)-----	80
Figure 89 : Chargement firmware micropython (6)-----	80
Figure 90 : Chargement firmware micropython (7)-----	81
Figure 91 : Diagramme d'E/S des broches de la carte Raspberry Pi pico -----	82
Figure 92 : Communication sur le bus I2C -----	83
Figure 93 : Principe de la communication sur le bus SPI -----	84
Figure 94 : Micropython - formatage de chaines de caractères-----	85
Figure 95 : Spectre des couleurs monochromatiques visibles -----	87
Figure 96 : Table des plages de couleurs monochromatiques -----	88
Figure 97 : Diagramme CIE sRGB -----	89
Figure 98 : Interpolation linéaire-----	90
Figure 99 : Schéma de la plateforme matérielle "Mesures environnementales" -----	92

12.12. Table des tableaux

Tableau 1 : Broches écran TFT DFR0665	46
Tableau 2 : Formatage associé aux mesures à afficher	51
Tableau 3 : Fonctionnalités associées aux boutons poussoir	55
Tableau 4 : Correspondance coins écran TFT et données de la dalle tactile	62