|  |  |  |
| --- | --- | --- |
| **Sciences de**  **l’ingénieur** | **Compétences travaillées :**  - Caractériser les échanges d’informations  - Analyser le traitement de l’information  - Traduire le comportement attendu ou observé d’un objet  - Traduire un algorithme en un langage exécutable  - Documenter un programme informatique  - Conduire des essais en toute sécurité à partir d’un protocole expérimental fourni  - Proposer et justifier un protocole expérimental | **Connaissances associées :**  - Natures et caractéristiques des signaux, des données, des supports de communication  - Structures algorithmiques (variables, fonctions, structures : séquentielles, itératives, conditionnelles)  - Langage de programmation python  - Commentaires de programmes  - Règles de raccordement des appareils de mesure et des capteurs  - Capteurs, composants d’une chaîne d’acquisition  - Paramétrage d’une chaîne d’acquisition  - Carte microcontrôleur  - Programmation événementielle d’une interruption |
| **Durée :** 2 h |
| Référence :  Hydrao\_Prog1 |
| **Matériel :**  Carte Nucleo WB55  Divers composants. |
| **Logiciels :**  Py\_Scripter, version portable |
| **Documents :**  dossier technique du système HYDRAO DIDACT |
| **Prérequis :**  - Notions d’algorithme et de programme  - Syntaxe de base du langage python  - Eléments sur l’architecture et programmation d’un microcontrôleur  - Grandeurs électriques  - Notion de conversion analogique-numérique | |

**Remarque sur la séance au professeur :**

Deux heures devraient suffire pour une première prise en main de la carte STM32WB55. Il n’est pas nécessaire de mettre en œuvre toutes les briques logicielles dans un premier temps pour en appréhender le fonctionnement. Cela dépendra bien sûr des compétences déjà acquises en programmation des microcontrôleurs et langage interprété Python.

L'activité nécessite une carte STM32 Nucleo WB55 avec un interpréteur microPython préinstallé, cf Dossier\_technique\_et\_Mise\_en\_service\_du\_kit, partie Dossier\_professeur\_mise\_en\_service\_carte\_STM32WB55.

L'activité nécessite aussi quelques composants (potentiomètre, LED, résistances, …) et, si possible, un "shield" Grove.

# Problème scientifique et technologique

Afin de réduire la consommation d’eau et la consommation d’énergie (chauffage de l’eau) lors d’une douche, la startup Hydrao mise en partie sur une incitation ludique en fournissant une information instantanée sous forme lumineuse à l’utilisateur. Le système didactique HYDRAO DIDACT utilise la carte à microcontrôleur Nucleo STM32WB55, permettant un prototypage rapide de solution logicielle.

L'activité proposée ici, préalable à l'activité de programmation d'une solution de gestion du pommeau, permet de prendre en main cette carte en explorant des briques logicielles de base en microPython.

#### Présentation de la carte de développement STM32 Nucleo WB55

* La NUCLEO-WB55 est une carte à microcontrôleur Cortex M4 développée par STMicroelectronics. Elle rassemble sur la même puce en silicium deux microprocesseurs ARM Cortex ultra-basse consommation et offre une connectivité radio Bluetooth à basse consommation (BLE pour “*Bluetooth Low Energy*”).
* Les connecteurs de la carte sont compatibles avec Arduino et avec le connecteur ST standardisé (Morpho) qui donne accès à toutes les broches d’entrée/sortie disponibles du microcontrôleur.

Brochage de la carte : <https://os.mbed.com/platforms/ST-Nucleo-WB55RG/>

* La carte NUCLEO-WB55 permet de programmer aisément le microcontrôleur STM32WB55 qui l’équipe en donnant accès à toutes ses ressources.
* Les connecteurs d’extensions (Arduino, Morpho) permettent à l’utilisateur de connecter aisément des composants électroniques externes au STM32WB55.
* La carte dispose aussi d’un connecteur pour pile CR2032, de 3 boutons poussoirs (SW1, SW2, SW3), de 3 LED (rouge, verte et bleue) et de deux connecteurs micro-USB.

Une description plus détaillée est donnée dans la ressource « ***Guide\_présentation\_carte\_Nucleo\_STM32WB55*** ».

**Travail à effectuer :**

Nota : il conviendra de réaliser un dossier réponse documenté à l’aide de copies d’écran pertinentes si besoin.

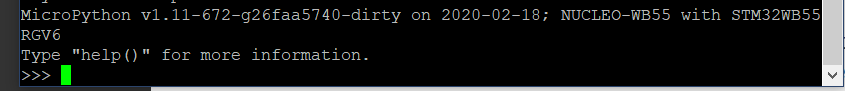
# Premières interactions en microPython avec la carte

En exploitant le guide d’utilisation de l’environnement de programmation, ressource : « ***Guide\_utilisation\_environnement\_de\_programmation\_micropython*** », nous pouvons aborder l'exploration de quelques briques logicielles de base.

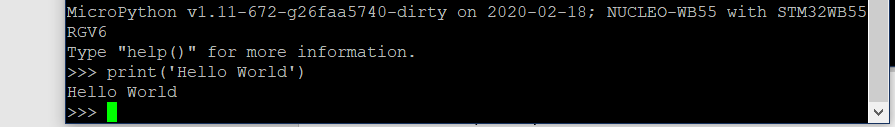
Lancez le terminal série PUTTY (on pourra utiliser également *TeraTerm* ou tout autre terminal série).

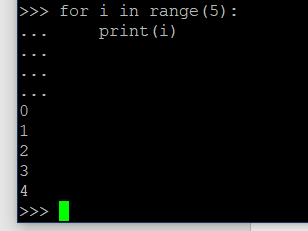
Choisissez le port série correspond à votre carte WB55 (à vérifier à l’aide du gestionnaire de périphériques).

Appuyez sur Ctrl-C pour interrompre un éventuel programme déjà en mémoire. Vous devez obtenir l’affichage suivant :



A ce stade, il est possible d’utiliser le terminal comme un interpréteur Python. Exemples de test :





Vous pouvez utiliser la fonction Python *help()* pour obtenir la documentation d'un module, d'une classe, d'une fonction, de variables, etc.



Nous savons à présent comment interagir en microPython avec une carte Nucleo-WB55 à partir d’un terminal série. Il est évident que cela deviendrait vite rédhibitoire s’il fallait écrire ne serait-ce qu’un programme de quelques lignes car il faudrait écrire ligne après ligne les différentes instructions.

Pour une efficacité satisfaisante nous utiliserons un IDE (environnement de développement intégré) open source comme *PyScripter portable* pour écrire un programme microPython.

# Exploration de briques logicielles

Nous allons dans ce paragraphe montrer comment développer quelques briques logicielles simples en microPython pour prendre en main cette carte Nucleo STM32. Nous aurons besoin d’interagir avec des composants externes à la carte, comme des LED ou des boutons poussoirs. Ces composants seront branchés aux broches de la carte STM32 et nous devrons les configurer. Nous allons donc nous intéresser au préalable aux différents modes de configuration des broches d’entrée / sortie.

### Configuration des ports d’entrée-sortie

**Quelques notions sur les GPIO :**

La famille des microcontrôleurs STM32 dispose de nombreuses **broches** (*pin* en anglais). Ces broches sont regroupées généralement par paquet de 16 ou 8 pour former ce qu’on appelle **des ports d’entrée/sortie** (*I/O ports*). Une broche peut être soit pilotée et configurée directement en entrée ou en sortie (on parle alors de *GPIO : General Purpose Input Output*), soit par le biais d’un périphérique dédié (on parle alors d’*Alternate function*).

Les configurations de ces broches se font par l’intermédiaire de registres dédiés (mémoires spécialisées) de configuration des étages électroniques associés.

**Configuration des broches :**

Les broches d’un même port sont configurables indépendamment les unes des autres. Une broche peut être configurée en entrée ou en sortie pour le microcontrôleur.

Une broche configurée **en entrée** permet au microcontrôleur de lire une information sous la forme d’un signal électrique imposé par des circuits ou composants externes à la carte STM32. De même, une broche **en sortie** permet au microcontrôleur de délivrer une information sous la forme d’un signal électrique imposé aux circuits ou composants externes à la carte STM32.

Les GPIO du microcontrôleur STM32 travaillent typiquement sous une tension maximale de 3,3 V et des intensités possibles jusqu'à environ 20 mA.

Les cellules d’entrée-sortie du WTM32 sont regroupées par 16 pour former des ports du microcontrôleur. Une cellule peut être configurée en entrée ou en sortie, ou encore peut être reliée à un périphérique interne au microcontrôleur comme SPI, I2C, USART, Timer, DMA, etc., pour assurer des fonctions alternatives (*alternate functions*).



Sortie logique ou PWM

Etage configuration de la broche en sortie

Broche E/S

Etage configuration de la broche en entrée

Sortie fonction alternative

Sortie logique

Entrée logique

Entrée fonction alternative

Entrée analogique

*Figure 1 : structure interne d’un GPIO – source ST Microelectronics -*

Dans le cas d’accès direct, la cellule étant configurée en conséquence, la lecture d’une entrée et l’écriture d’une sortie s’effectuent via des registres de donnée spécifiques. Un niveau haut de tension correspond à une valeur logique « 1 » et un niveau bas à une valeur logique « 0 ».

Dans le cas de l’utilisation d’une fonction alternative, l’entrée et la sortie sont reliées à un périphérique interne.

**Différentes configurations des broches d’entrée / sortie :**

Les informations ci-après visent à faciliter la compréhension de la syntaxe des instructions microPython.

* **Configuration** **en entrée :** c’est le circuit extérieur qui fixe la tension sur la broche (et le courant) compte tenu de la configuration du circuit interne. Cela permet par exemple de lire l’état (appuyé ou relâché) d’un bouton poussoir, ou la valeur d’une tension analogique.



*Figure 2 : une broche configurée en entrée*

En ouvrant ou fermant les « interrupteurs » (ce sont en réalité des circuits MOS) de la figure 2 ci-dessus, on configure l’entrée selon un des trois modes possibles de la figure 3.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Floating input :**  - Si le circuit extérieur ne fixe pas le niveau sur la broche dans certains cas (ou en l’absence de circuit extérieur), l’entrée est dite flottante. Cette situation est à proscrire.  - Ce mode est à utiliser lorsque le circuit extérieur fixe en permanence le niveau sur la broche, niveau qui peut être alternativement haut et bas. | **Pull-up resistor :**  - Une résistance (interne) de tirage vers le niveau haut permet de fixer le niveau haut en entrée lorsque le circuit extérieur fixe le niveau bas mais pas le niveau haut.  - Cela peut éviter l’ajout d’un élément externe au microcontrôleur pour fixer le niveau haut.  - Cela peut permettre de réaliser un « OU câblé » de plusieurs circuits extérieurs n’imposant pas le niveau haut. | **Pull-down resistor :**  - Une résistance (interne) de tirage vers le niveau bas permet de fixer le niveau bas en entrée lorsque le circuit extérieur fixe le niveau haut mais pas le niveau bas.  - Cela peut éviter l’ajout d’un élément externe au microcontrôleur pour fixer le niveau bas.  - Cela peut permettre de réaliser un « OU câblé » de plusieurs circuits extérieurs n’imposant pas le niveau bas. |

*Figure 3 : différentes configurations d’une broche en entrée*

* **Configuration** **en sortie :** c’est le circuit interne (transistors NMOS et PMOS) qui fixe la tension sur la broche (et le courant) compte tenu du circuit extérieur. Cela permet par exemple d’allumer ou d’éteindre une LED, ou de délivrer un signal plus complexe. Idéalement, on peut modéliser les transistors MOS par des interrupteurs lorsqu’ils travaillent en mode « on/off » (MOS fortement conducteur / MOS bloqué).



Commande

Commande

*Figure 4 : une broche configurée en sortie*



*Figure 5 : différentes configurations d’une broche en sortie*

* **Configuration** **en fonction alternative :** la broche est connectée à un périphérique interne du microcontrôleur tel qu’un bus USB, un contrôleur de port série (USART/UART), un TIMER (par exemple pour générer ou pour analyser des signaux de type PWM) ou toute autre fonction avancée.
* **Configuration** **en entrée analogique** (*analog input*) : l’entrée est connectée à un convertisseur analogique numérique – abrégé en CAN – (ADC, *Analog to Digital Converter*) interne au microcontrôleur. C’est le cas par défaut des broches reliées au connecteur Arduino A0 à A5.



*Figure 6 : une broche configurée en entrée analogique*

* **Configuration** **en sortie analogique** (*analog output*), la sortie est connectée à un convertisseur numérique analogique – abrégé en CNA – (DAC, *Digital to Analog Converter*) interne au microcontrôleur. Certains microcontrôleurs disposent d’un tel convertisseur mais pas celui utilisé ici. On peut cependant produire un signal analogique par modulation de largeur d’impulsion – MLI (ou PWM) – et filtrage (voir plus loin).

**Manipulation des broches en microPython :**

Vous trouverez sur la page : <http://docs.micropython.org/en/latest/library/pyb.Pin.html#pyb-pin> la description des méthodes de la classe « Pin »pour la carte « Pyboard », également valable pour la NUCLEO-WB55. Cette classe contient le nécessaire pour configurer et utiliser les broches de la carte NUCLEO-WB55.

Pour utiliser une broche particulière, on crée un objet de la classe Pin. On dispose alors de méthodes (fonctions) pour définir le mode souhaité (entrée, sortie, etc.) et des méthodes pour obtenir ou définir le niveau logique de la broche. Pour l’utilisation du mode analogique, on pourra utiliser les méthodes de la classe ADC : <https://docs.micropython.org/en/latest/library/pyb.ADC.html?highlight=adc>

Les broches sont accessibles en utilisant la syntaxe :

pyb.Pin.cpu.Nom

où Nom représente la lettre du port suivie du numéro de la broche (voir en annexe le document : *Guide\_presentation\_carte\_Nucleo\_STM32WB55.pdf*).

**Exemple :** pyb.Pin.cpu.C6 désigne l’entrée/ sortie n°6 du port C.

De plus, cette entrée/sortie est reliée à la broche D2 du connecteur Arduino, on pourra donc écrire :

broche\_D2 = pyb.Pin.cpu.C6

Pour créer un objet de type Pin, on utilise la syntaxe simplifiée (d’autres paramètres supplémentaires avec des valeurs par défaut peuvent être utilisés, notamment avec les fonctions alternatives) :

nom\_broche = pyb.Pin(id, mode, pull)

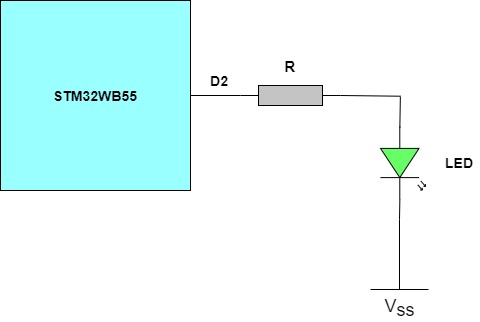
id : désigne l’identifiant (nom) de la broche, par exemple D2.

On aura bien entendu intérêt dans une application spécifique à donner des noms significatifs.

|  |  |  |
| --- | --- | --- |
| **Paramètre** | **Valeurs possibles** | **Signification** |
| mode | Pin.IN | Configure la broche en entrée |
| Pin.OUT\_PP | Configure la broche en sortie en mode push\_pull |
| Pin.OUT\_OD | Configure la broche en sortie en mode open-drain |
| Pin.AF\_PP | Configure la broche pour une fonction alternative, push\_pull |
| Pin.AF\_OD | Configure la broche pour une fonction alternative, open-drain |
| Pin.ANALOG | Configure la broche en analogique |
| pull | Pin.PULL\_NONE | Pas de résistance pull-up ou pull-down |
| Pin.PULL\_UP | Active la résistance pull-up |
| Pin.PULL\_DOWN | Active la résistance pull-down |

### Programme de configuration d’une broche en sortie

**Travail demandé :** la broche D2 sera configurée en sortie, reliée à une LED, et un programme de clignotement de la LED sera écrit. La LED permettra simplement ici de visualiser facilement l’état de la broche.



*Figure 7 : schéma de principe de la LED reliée à la broche D2 du STM32*

* Donnez la ligne de code microPython pour une configuration de la sortie en push-pull.
* Complétez le script ci-après pour changer l’état de la broche toutes les 500 ms.
* Testez le programme et visualisez ces changements d’état en branchant une LED sur la broche D2.

'''

Configuration d’une broche en sortie et visualisation de son état par une LED

'''

import pyb # importation du module pyb pour gérer les GPIO

import time # importation du module time permettant de faire des pauses

# Broche D2 configurée en sortie push-pull (OUT\_PP)

# Tirage PULL\_NONE : absence de résistance de tirage

# LED visualisant l'état de la broche D2

D2 = pyb.Pin.cpu.C6 # Equivalence broche ARDUINO broche STM32 WB55

etat\_broche = pyb.Pin(D2, pyb.Pin.OUT\_PP, pyb.Pin.PULL\_NONE)

etat\_broche.value(0) # Initialisation à 0 de la broche

# programme principal

while True : # boucle permanente

etat\_broche.value(1) # la broche passe à 1 : la LED s'allume

time.sleep\_ms(500) # temporisation de 500 ms

etat\_broche.value(0) # la broche passe à 0 : la LED s'éteint

time.sleep\_ms(500)

**Remarque :**

Les lignes

**import** pyb # importation du module pyb

**import** time # importation du module time

peuvent être remplacées ici par les lignes :

**from** pyb **import** Pin # importation de la classe Pin du module pyb

**from** time **import** sleep\_ms # importation de la classe sleep\_ms du module time

- La première forme d’importation donne potentiellement accès à tous les éléments du module et rend obligatoirement explicite la provenance des méthodes utilisées. Par exemple etat\_broche **=** pyb.Pin(…).

- La deuxième forme ne donne accès qu’aux éléments de la classe précisée et permet de simplifier les écritures. Par exemple etat\_broche **=** Pin(…).

- Sauf dans des cas très simples ou très particuliers, la première forme est conseillée pour rendre le code explicite et éviter d’éventuels conflits de nommage.

- Les deux formes peuvent être utilisées conjointement. Par exemple :

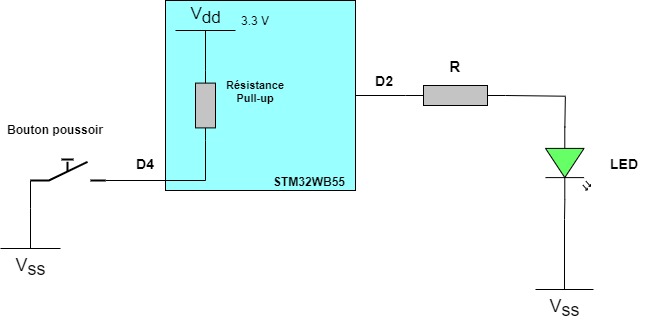
**import** pyb # importation du module pyb

**from** pyb **import** Pin # importation de la classe Pin du module pyb

pour avoir accès à l’ensemble des élément du module pyb et en écriture raccourcie pour la classe Pin.

### Programme de configuration d’une broche en entrée

**Travail demandé :** la broche D4 sera configurée en entrée, reliée à un bouton poussoir. La configuration de la broche D2 en sortie reliée à une LED sera reprise de l’exemple précédent, puis un programme de visualisation par la LED (allumée/éteinte) de l’état du bouton poussoir (on/off) sera écrit.



*Figure 8 : schéma de principe du bouton poussoir relié la broche D4 et de la LED reliée à la broche D2*

* Donnez la ligne de code microPython pour une configuration de la broche D4 en entrée avec une résistance de pull-up.
* Branchez un bouton-poussoir sur la broche D4.
* Complétez le script ci-après pour visualiser avec la LED (allumée/éteinte) l’état du bouton poussoir (on/off). On reprendra pour D2 la configuration de l’exemple précédent.

"""

Broche D4 en entrée avec pull-up et broche D2 en sortie

Bouton poussoir connecté à D4 et LED connectée à D2

Visualisation de l’état du bouton poussoir par la LED

Scrutation (polling) de l’état on/off du bouton poussoir

"""

**import** pyb # importation du module pyb pour gérer les GPIO

# Bouton poussoir branché sur broche D4 configurée en entrée

# Tirage PULL UP : résistance de tirage reliée à + 3.3 V

D4 **=** pyb**.**Pin**.**cpu**.**C10 # Equivalence broche ARDUINO broche STM32 WB55

bouton\_in **=** pyb**.**Pin**(**D4**,** pyb**.**Pin**.**IN**,** pyb**.**Pin**.**PULL\_UP**)**

# LED branchée sur broche D2 configurée en sortie push-pull.

# Tirage PULL\_NONE : absence de résistance de tirage.

D2 **=** pyb**.**Pin**.**cpu**.**C6 # Equivalence broche ARDUINO broche STM32 WB55

led\_out **=** pyb**.**Pin**(**D2**,** pyb**.**Pin**.**OUT\_PP**,** pyb**.**Pin**.**PULL\_NONE**)**

etat\_bp **=** 0 # Mémoire de l’état du bouton poussoir initialement off

led\_out**.**value**(**etat\_bp**)** # Affectation de la sortie

# Programme principal

**while** **True** **:** # Boucle permanente

etat\_bp **=** **not** bouton\_in**.**value**()** # lecture de l’entrée bouton poussoir et

# complémentation car l’appui sur bouton poussoir impose un 0 sur l’entrée

led\_out**.**value**(**etat\_bp**)** # recopie de l’état du bp sur la sortie led

**Travail demandé :** en modifiant l’exemple précédent, écrivez un programme provoquant un changement d’état de la LED (allumée ou éteinte) à chaque appui sur le bouton poussoir (passage de off à on).

"""

Broche D4 en entrée avec pull-up et broche D2 en sortie

Bouton poussoir connecté à D4 et LED connectée à D2

Changement d’état de la LED (allumée ou éteinte) à chaque appui sur

le bouton poussoir (passage de off à on).

Scrutation (polling) de l’état on/off du bouton poussoir

"""

**import** pyb # importation du module pyb pour gérer les GPIO

**import** time # module permettant de faire des pauses système

# Bouton poussoir branché sur broche D4 configurée en entrée (IN)

# Tirage PULL UP : résistance de tirage reliée à + 3.3 V

D4 **=** pyb**.**Pin**.**cpu**.**C10 # Equivalence broche ARDUINO broche STM32 WB55

bouton\_in **=** pyb**.**Pin**(**D4**,** pyb**.**Pin**.**IN**,** pyb**.**Pin**.**PULL\_UP**)**

# LED branchée sur broche D2 configurée en sortie push-pull.

# Tirage PULL\_NONE : absence de résistance de tirage.

D2 **=** pyb**.**Pin**.**cpu**.**C6 # Equivalence broche ARDUINO broche STM32 WB55

led\_out **=** pyb**.**Pin**(**D2**,** pyb**.**Pin**.**OUT\_PP**,** pyb**.**Pin**.**PULL\_NONE**)**

etat\_bp **=** **0** # Mémoire de l’état du bouton poussoir initialement off

etat\_bp\_precedent **=** **0** # Mémoire de l’état précédent du bouton poussoir

etat\_led **=** **0** # Mémoire de l’état de la LED initialement éteinte

led\_out**.**value**(**etat\_led**)** # Affectation de la sortie

# Programme principal

**while** **True** **:** # Boucle permanente

time**.**sleep\_ms**(**500**)** # temporisation 500ms

etat\_bp **=** **not** bouton\_in**.**value**()** # lecture de l’entrée bouton poussoir et

# complémentation car l’appui sur bouton poussoir impose un 0 sur l’entrée

**if** **(**etat\_bp **==** 1**)** **and** **(**etat\_bp\_precedent **==** 0**)** **:** # si front montant

etat\_led **=** **not** etat\_led # toggle

led\_out**.**value**(**etat\_led**)** # affectation de la sortie led

etat\_bp\_precedent **=** etat\_bp # actualisation etat\_bp\_precedent

**Remarque 1 :** ce code est un exemple de programmation par scrutation (polling en anglais). Le programme lit l’entrée en permanence (boucle infinie), ce qui occupe le processeur à 100 %.

**Remarque 2 :** sur la carte NUCLEO-WB55 on trouve trois boutons poussoirs SW1, SW2, SW3 et trois diodes électroluminescentes LED1, LED2, LED3 connectés à des broches du STM32WB55 nécessitant la configuration des entrées-sorties (GPIO) correspondantes.

|  |  |  |
| --- | --- | --- |
|  |  |  |

*Figure 9 : SW1, SW2, SW3, LED1, LED2, LED3 présents sur la carte Nucléo STM32WB55*

**Remarque 3 :** le module pyb de micro-Python intègre la classe LED permettant de gérer et piloter ces LED. Voir <https://docs.micropython.org/en/latest/library/pyb.LED.html>. On peut toutefois utiliser l'écriture habituelle vue précédemment en se référant au schéma de raccordement de ces LED. L’exemple qui suit montre l’utilisation de méthodes de la classe LED pour faire clignoter la LED verte toutes les 500 ms

"""

Clignotement de la LED verte toutes les 500 ms.

"""

**import** pyb

**import** time

led\_verte **=** pyb**.**LED**(**2**)** # configuration de la LED 2 verte

# ou : led\_verte = pyb.Pin(pyb.Pin.cpu.b0, pyb.Pin.OUT\_PP, pyb.Pin.PULL\_NONE)

#programme principal

**while** **True** **:** # Boucle permanente

led\_verte**.**on**()** # Allume la LED

time**.**sleep\_ms**(**500**)** # temporisation de 500 ms

led\_verte**.**off**()** # Eteint la LED

time**.**sleep\_ms**(**500**)** # temporisation de 500 ms

### Lecture d’une grandeur analogique

Une grandeur analogique – souvent une tension comme ici – doit être convertie en un nombre par un CAN pour Convertisseur Analogique Numérique (ou ADC : *Analog to Digital Converter*) afin de pouvoir être traitée par un processeur. Le microcontrôleur STM32WB55 intègre un CAN permettant de convertir une tension comprise entre 0 V et VDD = 3,3 V en un entier compris entre 0 et 4095 (le format étant de 12 bits). Il est possible d’adapter cette plage de tension mais nous ne verrons pas comment ici.

Pour le connecteur Arduino, seules les broches A0 à A5 sont reliées à des ADC.

Documentation sur la classe ADC : <https://docs.micropython.org/en/latest/library/pyb.ADC.html>

**Matériel requis :**

* La carte NUCLEO-WB55.
* Un potentiomètre de 10 KΩ par exemple.

|  |  |
| --- | --- |
| *Montage de test : le point milieu du potentiomètre linéaire de 10 kΩ est relié sur l’entrée A0.* | *Schéma électrique : la tension est proportionnelle à la position du curseur du potentiomètre linéaire* |
|  |  |

*Figure 10 : branchement du potentiomètre sur la carte Nucléo STM32WB55*

**Programme microPython :** le script suivant lit la tension présente sur l’entrée A0 et la convertit en un entier compris entre 0 et 4095 lorsque UA0 varie de 0 à 3,3 V.

"""

Lecture d'une valeur analogique sur A0

"""

**import** pyb

**import** time

A0 **=** pyb**.**Pin**.**cpu**.**C0 # Equivalence broche ARDUINO broche STM32 WB55

adc **=** pyb**.**ADC**(**A0**)** # Spécifie que adc est un ADC relié à la broche A0

# programme principal

**while** **True** **:**

# Acquisition de la valeur analogique, conversion analogique numérique,

# et affectation dans val\_numérique

val\_numerique **=** adc**.**read**()**

**print(**"La valeur numerique : "**,** val\_numerique**)** # Affichage valeur numérique

time**.**sleep\_ms**(**500**)** # temporisation de 500ms

**Travail demandé :**

- Éditez le script « main.py » contenu dans le répertoire du disque USB virtuel associé à la NUCLEO-WB55 : « PYBFLASH ». Analysez-le et observez la valeur numérique qui évolue lorsque vous tournez le potentiomètre.

- Complétez le programme pour afficher la valeur de la tension lue (entre 0 et 3,3 V) et pour allumer la LED rouge lorsque la tension est comprise entre 1 V et 1,5 V.

"""

Lecture d'une valeur analogique sur A0 et CAN

Affichage des valeurs analogique et numérique

LED rouge allumée si tension d'entrée comprise entre 1 V et 1.5 V

"""

**import** pyb

**import** time

# Equivalence broches ARDUINO broches STM32 WB55

A0 **=** pyb**.**Pin**.**cpu**.**C0

D2 **=** pyb**.**Pin**.**cpu**.**C6

# # Une LED est branchée sur la broche D2 configurée en sortie Push-pull.

led\_rouge **=** pyb**.**Pin**(**pyb**.**Pin**.**cpu**.**C6**,** pyb**.**Pin**.**OUT\_PP**,** pyb**.**Pin**.**PULL\_NONE**)**

adc **=** pyb**.**ADC**(**A0**)** # Spécifie que adc est un ADC relié à la broche A0

led\_rouge**.**value**(**0**)** # led éteinte au départ

# programme principal

**while** **True** **:**

# Acquisition de la valeur analogique, conversion analogique numérique,

# et affectation dans val\_numérique

val\_numerique **=** adc**.**read**()**

**print(**"valeur numerique : "**,** val\_numerique**)**

# Calcul de la tension correspondante à la valeur numérique pour affichage

# à titre informatif

tension\_correspondante **=** val\_numerique **\*** **(**3.3 **/** 4096**)**

**print(**"tension correspondante : "**,** round**(**tension\_correspondante**,** 2**),** " V"**)**

# led-rouge allumée si 1 V < tension\_correspondante < 1.5 V

**if** **(**val\_numerique **>** 1241**)** **and** **(**val\_numerique **<** 1861**)** **:**

led\_rouge**.**value**(**1**)** # allumer led

**else** **:**

led\_rouge**.**value**(**0**)** # eteindre led

time**.**sleep\_ms**(**500**)** # temporisation de 500 ms

### Gestion des interruptions

**Définition du problème :**

On ne connaît pas les instants d’apparition des évènements externes liés aux périphériques d’entrée/sortie (exemple : appui sur un bouton poussoir, fin de comptage, arrivée / détection d’un objet, …), et des évènements internes (dits exceptions) qui surviennent en cas d’erreurs lors de l’exécution du programme (division par zéro par exemple).

Sachant que l’exécution d’un programme est séquentielle (boucle infinie), se pose alors la question de la capture de ces évènements. Deux solutions sont envisageables :

* La boucle de scrutation (polling) : le processeur lit en permanence les entrées pour détecter un changement de valeur ; l’intervalle de temps entre deux lectures des entrées doit être suffisamment court pour ne pas "louper" d’événement, ce qui limite le temps disponible pour d’autres traitements.
* L’utilisation d’interruptions : la gestion des périphériques est confiée à des circuits spécialisés. Des commandes (requêtes) seront envoyées au processeur pour le prévenir par un signal d’interruption. Ce fonctionnement décharge ainsi le processeur de la surveillance des périphériques et il peut ainsi exécuter les autres tâches.

|  |  |
| --- | --- |
| **Qu’est-ce qu’une interruption ?**  Les deux microprocesseurs qui animent le STM32WB55 sont des Cortex M, conçus par la société ARM. L’architecture ARM Cortex-M est dominante dans les microcontrôleurs pour les applications embarquées, essentiellement en raison de son excellente efficacité énergétique et de sa gestion très performante des interruptions.  Une interruption est une suspension temporaire de l'exécution d'un programme par le processeur afin d'exécuter un programme plus prioritaire (appelé service d'interruption). | *Figure 11 : interruption de programme* |

Une interruption est déclenchée par l’apparition d’un évènement externe (ou interne par extension) qui va, dans l’ordre :

* interrompre le fonctionnement du programme en cours d’exécution ;
* sauvegarder le contexte d’exécution (état du programme en cours d’exécution : registres, valeur du compteur ordinal, …) ;
* exécuter un programme spécifique à l’interruption appelé routine de service d’interruption (ISR : *Interrupt Service Routine*) pour les événements externes ou gestionnaire d’exception (*Exception Handler*) pour les événements internes ;
* puis, le programme d’interruption étant achevé, restaurer le contexte précédent d’exécution ;
* ce qui permet de reprendre le fonctionnement du programme là où il avait été interrompu.

Un contrôleur d’interruptions (NVIC : *Nested Vectored Interrupt Controller*) interne aux microcontrôleurs de ce type est chargé de la gestion des interruptions. On remarquera notamment les interruptions en provenance des périphériques (NMI : *No-Maskable Interrupt*, IRQ : *Interrupt Request*).



*Figure 12 : schéma de principe du mécanisme d’interruptions des processeurs Cortex-M -*

*Extrait d’un article du journal Electronics weekly :*

<https://www.electronicsweekly.com/news/products/micros/arm-explains-good-interrupt-control-for-low-power-processors-2012-10/>

Programmer un microcontrôleur en utilisant des interruptions lui permet de traiter rapidement les apparitions d’événements provenant de périphériques en exécutant les programmes correspondants selon les priorités attribuées, et, en dehors des traitements des interruptions, laisser le temps au processeur pour les autres tâches.

Outre la bonne réactivité, les interruptions permettent également d’économiser l’énergie. Elles remplacent avantageusement les boucles permanentes (ou infinies) dans un programme principal par des ISR qui ne sont exécutées que lorsque les évènements apparaissent. En contrepartie, le temps d’exécution d’une ISR doit être suffisamment court afin de ne pas retarder l’exécution d’autres ISR ou suspendre trop longtemps le programme principal, qui, rappelons-le, est interrompu pendant l’exécution des ISR autorisées.

**Exemple : changement d’état d’une LED à chaque appui sur un bouton poussoir en mode interruptif**

L’exemple qui suit montre comment configurer les entrées et sorties (GPIO) pour commander une LED avec un bouton poussoir déclenchant une interruption. Lors d’un appui sur le bouton poussoir, l’ISR est exécutée et change l’état de la LED.

**Matériel requis :**

* La carte NUCLEO-WB55.
* Un Shield de base Grove (le commutateur d’alimentation doit être positionné sur 3,3 V).
* Un module Bouton Grove qui fait passer la sortie de l'état bas à l'état haut lors d'un appui.
* Un module LED Grove commandé par une sortie numérique.

|  |  |  |
| --- | --- | --- |
| Bouton poussoir (P) - Grove |  | Led verte 3 mm Grove 104030007 |

*Figure 13 : matériel requis*

**Travail demandé :**

- Connectez le module LED sur D2 et le module bouton sur D4.

- Analysez le programme ci-dessous et complétez le sous-programme d’interruption pour inverser l’état de la LED à chaque interruption sur la broche D4.

'''

LED connectée à D2

Bouton poussoir connecté à D4

Un appui sur le bouton poussoir déclenche une interruption.

La LED est alternativement allumée/éteinte à chaque appuie sur le bouton poussoir

'''

**import** pyb # importation du module pyb pour gérer les GPIO

# Bouton poussoir connecté broche D4 configurée en entrée

# Tirage PULL UP : résistance de tirage reliée à + 3.3 V

D4 **=** pyb**.**Pin**.**cpu**.**C10 # Equivalence broche ARDUINO broche STM32 WB55

bouton\_in **=** pyb**.**Pin**(**D4**,** pyb**.**Pin**.**IN**,** pyb**.**Pin**.**PULL\_UP**)**

# LED branchée sur broche D2 configurée en sortie push-pull.

# Tirage PULL\_NONE : absence de résistance de tirage.

D2 **=** pyb**.**Pin**.**cpu**.**C6 # Equivalence broche ARDUINO broche STM32 WB55

led\_out **=** pyb**.**Pin**(**D2**,** pyb**.**Pin**.**OUT\_PP**,** pyb**.**Pin**.**PULL\_NONE**)**

etat\_led **=** **0** # Mémoire de l’état de la LED initialement éteinte

led\_out**.**value**(**etat\_led**)** # Affectation de la sortie

'''

ISR\_appui\_bouton : sous-programme de traitement de l'interruption

déclenché par un appui sur le bouton poussoir

'''

**def** ISR\_appui\_bouton**(**pin**)** **:**

**global** etat\_led

etat\_led **=** **not** etat\_led # toggle

led\_out**.**value**(**etat\_led**)** # affectation de la sortie led

# Définition de l’interruption

# bouton\_in (bouton poussoir) est connecté à la ligne irq (interrupt request)

# Déclenchement de l’interruption : front montant

# Sous-programme de traitement de l'interruption : ISR\_appui\_bouton

bouton\_in**.**irq**(**trigger **=** bouton\_in**.**IRQ\_RISING**,** handler **=** ISR\_appui\_bouton**)**

#programme principal

**while** **True** **:**

**continue**

Le programme principal est une boucle permanente et ne fait qu’attendre l’apparition de l’événement « appui sur bouton-poussoir ». L’instruction [continue](https://courspython.com/boucles.html#continue) permet de passer prématurément à l’itération suivante de la boucle while.

**Problème des rebonds** : les boutons poussoirs mécaniques présentent des "rebonds" car un appui ou un relâchement vont souvent générer des fronts (montants / descendants) multiples. Ce problème sort du cadre de ce document. Voir par exemple : <https://my.eng.utah.edu/~cs5780/debouncing.pdf> pour une explication détaillée, ainsi que diverses techniques de résolution.

### Communication sans fil Bluetooth Low Energy (BLE)

**Présentation du Bluetooth Low Energy (BLE) :**

Le Bluetooth « Low Energy » ou « Bluetooth Smart » a vu le jour en 2006 sous le nom de projet « Wibree », réalisé dans le Centre de recherche de Nokia. La technologie a été adoptée par le Bluetooth SIG (Special Interest Group), qui l’a présentée comme une forme de Bluetooth à consommation énergétique ultrabasse lors de l’introduction de sa version 4.0 en 2010.

Le BLE permet des économies d’énergie en optimisant le temps de veille et en utilisant des connexions rapides et une faible puissance d’émission/réception de crête. Sa faible consommation énergétique réside principalement dans le fait que, contrairement au Bluetooth classique qui est une radio à intervalles de connexion fixes, le BLE présente généralement un état « déconnecté » éco énergétique, dans lequel chacune des deux extrémités d’une liaison est consciente de l’autre, mais ne se connecte qu’en cas de nécessité, et ce aussi brièvement que possible. En pratique la consommation du BLE est entre deux et dix fois plus faible que celle du Bluetooth lorsqu’on compare des fonctionnalités qui peuvent être implémentées avec ces deux protocoles.

En résumé :

* Fonctionne avec la bande ISM (Industrielle, Scientifique et Médicale) à 2.4 GHz.
* Faible complexité, faible coût.
* Faible bande passante (débit brut maximum inférieur à 2 Mbps en version 5.x).
* Faible consommation (courants de quelques microampères, alimentation sur pile bouton possible).
* Faible portée (de 20 à 50 mètres selon l’environnement).

Le SoC STM32WB55 qui anime la NUCLEO-WB55 est destiné aux objets connectés et implémente de ce fait les normes Bluetooth LE 5.2 et IEEE 802.15.4 (Bluetooth) mais il n’est pas dual mode (le dual mode permet une interopérabilité entre les deux protocoles en présence d’une radio BLE et d’une radio Bluetooth). Il ne peut donc pas « discuter » avec des modules Bluetooth standard (comme par exemple les HC-05 et HC-06).

**Communication sans fil (BLE) :** on va établir une communication en Bluetooth Low Energy entre l’application « **ST BLE Sensor »** et la carte de développement WB55.

* Installez, sur un SmartPhone, l’application « ST BLE Sensor » sur Google Play ou IOS Store.



Pour communiquer en Bluetooth Low Energy avec microPython, il faut inclure deux nouveaux fichiers dans le répertoire du disque USB « PYBFLASH » :

* **« ble\_advertising.py »** (fichier d’aide à la création de messages « advertissing » ou notifications)
* **« ble\_sensor.py »** (classe permettant la gestion de la connexion BLE)

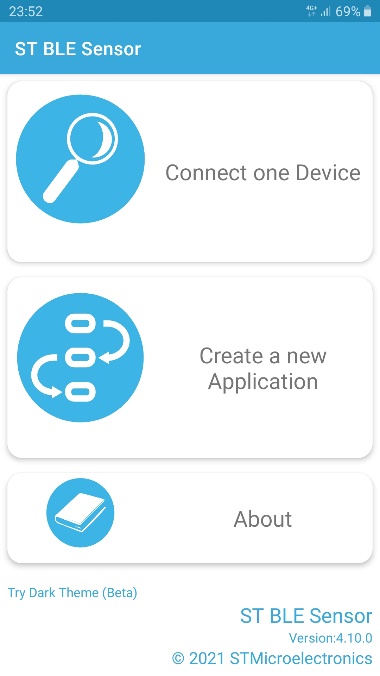
Il faudra télécharger les scripts nécessaires à partir du dossier *PackageBLE* disponible dans *Ressources\_eleves*.

Grâce au fichier **« ble\_sensor.py »**, nous allons pouvoir créer un objet BLE ayant 1 service et 2 caractéristiques. C’est ce fichier qu’il faudra modifier pour changer le profil BLE, si besoin.

Une fois le script lancé, le kit de développement WB55 se met à émettre des trames BLE, appelées « advertising ». Ces messages permettent d’identifier l’objet Bluetooth et de signifier que le périphérique est prêt à être connecté.

Le nom du périphérique est : « WB55-MPY », nous allons vérifier avec l’application installée sur le smartphone si la carte WB55 est en émission Bluetooth.

* Lancez l’application « STBLESensor » sur le SmartPhone :

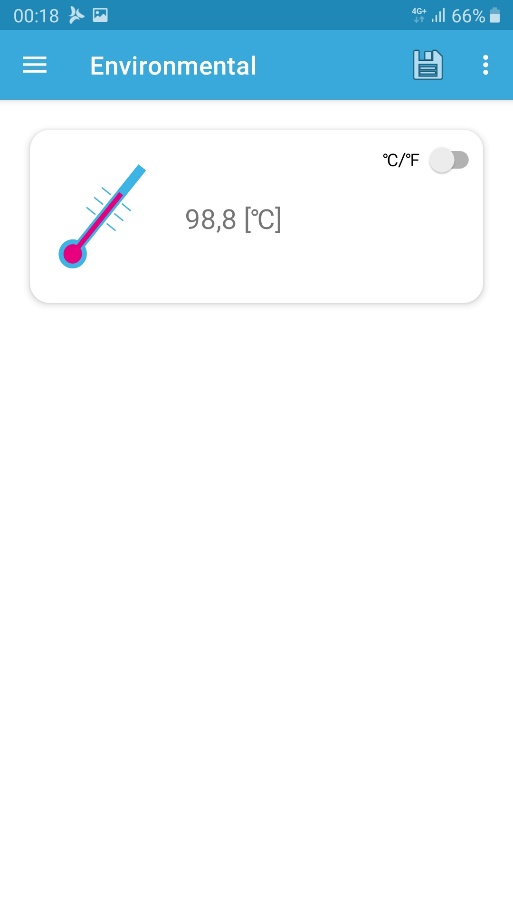


* Appuyez ensuite sur l’icône loupe pour afficher les périphériques BLE environnant :



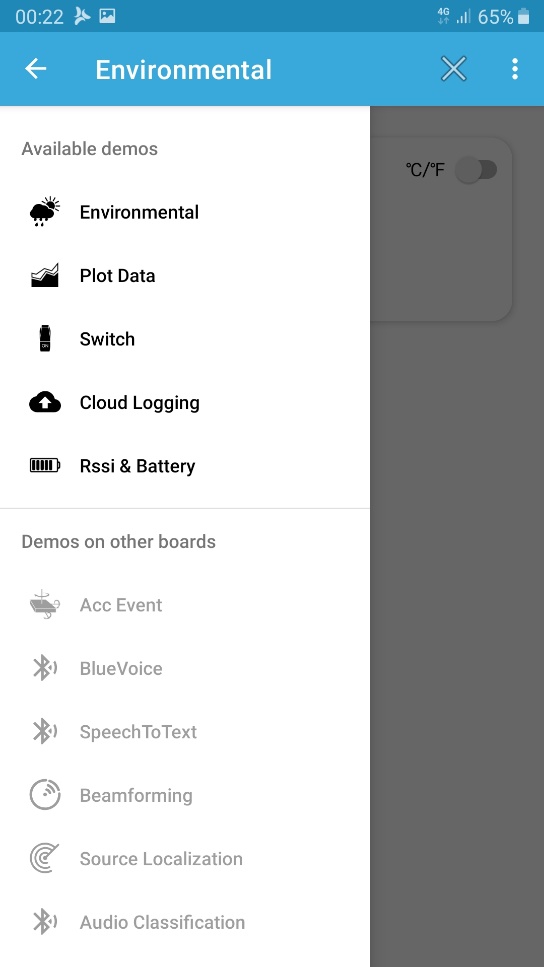
Dans cet exemple, le profil BLE choisi permet de simuler un thermomètre et d’allumer ou d’éteindre une LED. La valeur de la température est générée aléatoirement toutes les secondes.

* Connectez-vous à la carte de développement en appuyant sur « WB55-MPY » :



La LED bleue de la carte WB55 doit s’allumer lorsque cette dernière est connectée à l’application. On peut observer sur l’écran l’évolution aléatoire de la température entre 0 et 100°C.

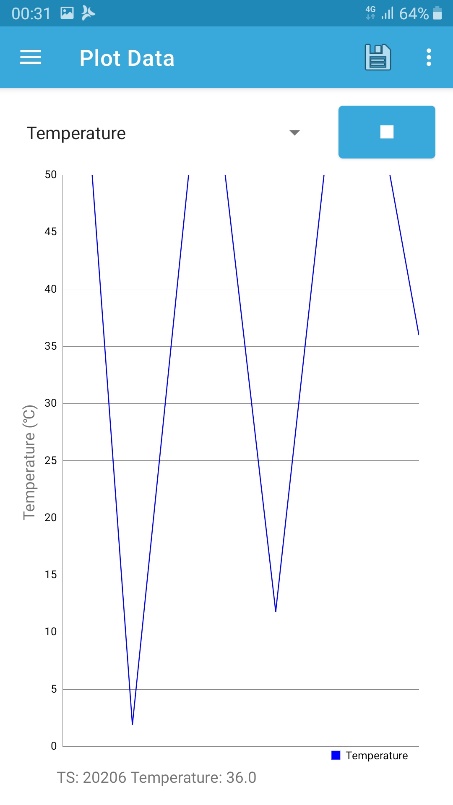
Il est possible d’afficher la température en mode graphique. Pour cela appuyez sur le bouton menu :



* Appuyez maintenant sur « Plot Data » :



* Pour afficher le graphique, appuyez sur « run » :

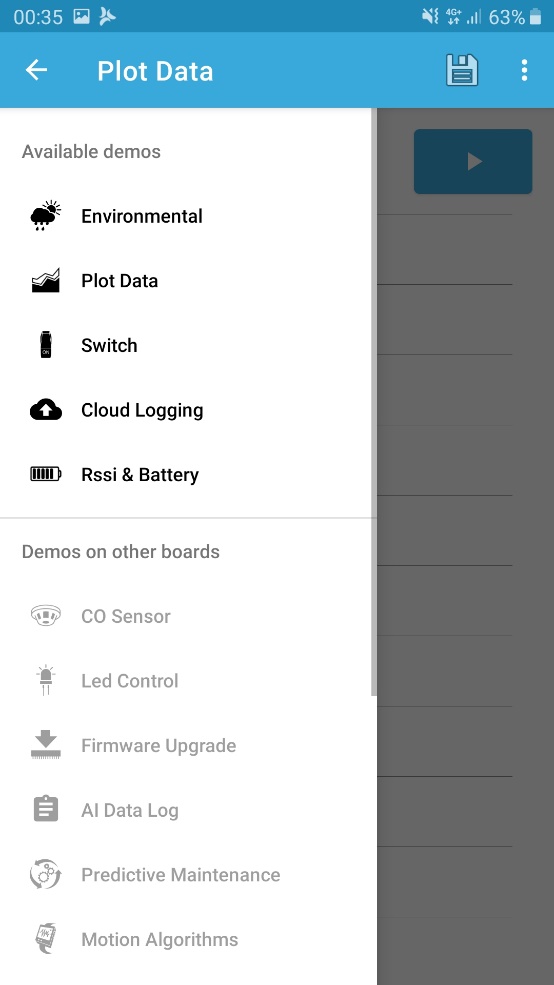


Remarque : vous pouvez modifier les options du graphique, comme la taille de l’axe X ou l’activation du changement automatique de l’échelle en Y.

**Travail demandé :**

Nous allons étudier l'envoi d'une information depuis le SmartPhone vers la plateforme WB55. On utilisera à cet effet l’application pour allumer ou éteindre la LED rouge du kit de développement.

* Pour cela appuyez sur le bouton menu :



* Choisissez l’option « Switch » : vous pouvez sur cet écran piloter la LED Rouge du kit de développement.

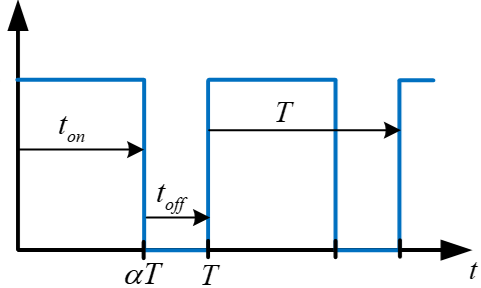
Mettez en œuvre l’application et donnez les copies d’écran obtenues :

|  |  |
| --- | --- |
|  |  |

### Utilisation de la PWM et des Timers

**Qu’est-ce que la PWM ?**

La MLI : modulation de largeur d’impulsion (ou PWM : *Pulse Width Modulation*) permet avec une sortie « tout ou rien » de faire varier la valeur moyenne d’une grandeur. Par exemple, avec une sortie « tout ou rien » comprise entre 0 V et VDD, on peut convertir un nombre N supposé sur 8 bits (donc compris entre 0 et 255) en une tension dont la valeur moyenne est égale à VDD x N/256.

La MLI est utilisée dans une grande variété d'applications, comme la variation de vitesse des moteurs à courant continu, de l’intensité d’éclairage des LED, la puissance sonore d’un buzzer, etc.

On définit le **rapport cyclique** (*duty cycle*) par le rapport entre la durée du signal à l'état haut divisée par la durée de la période :

La figure ci-après montre trois signaux PWM avec des rapports cycliques de 50, 10 et 90 % (de gauche à droite).



**Programmer des sorties PWM sur la carte NUCLEO-WB55 :**

En pratique, la génération de signaux PWM est l’une des fonctions assurées par les timers, circuits spécialisés intégrés dans le microcontrôleur (sortes de compteurs programmables). Le STM32WB55 contient plusieurs timers, et chacun d’eux pilote plusieurs canaux (channels). Certains canaux de certains timers sont connectables à des sorties du microcontrôleur. Ces sorties peuvent être utilisées pour générer des signaux PWM.

|  |  |
| --- | --- |
| Afin de programmer des sorties PWM, nous devons connaître :   * les broches de la NUCLEO-WB55 permettant des sorties PWM ; * les timers et canaux connectés à ces broches du STM32WB55.   La figure ci-contre fournit ces deux informations. | PWM WB55 |

**Mise en œuvre de la PWM :**

Le Buzzer (transducteur piézo-électrique) utilisé est de référence Velleman TV3 (fiche technique dans le dossier Ressource\_eleves). Le buzzer est un composant constitué essentiellement d’une lamelle réagissant à l’effet piézoélectrique. La piézoélectricité est la propriété que possèdent certains minéraux à se déformer lorsqu’ils sont soumis à un champ électrique. Ce phénomène est réversible ; si nous déformons ce minéral, il produit de l’énergie électrique.

Dans les applications courantes, le buzzer est principalement utilisé pour émettre un son.

**Travail demandé :**

Branchez le buzzer sur la broche D5. Le travail consiste à générer un signal rectangulaire dont la fréquence est l’image des notes de musique, pour cela nous utilisons un timer produisant un signal rectangulaire à la fréquence voulue. La puissance sonore est ajustée en appliquant une MLI au signal rectangulaire de façon à réduire, pour chaque période, sa durée à l'état haut par rapport à sa durée à l'état bas. Nous souhaitons jouer chaque note de la gamme proposée pendant 500 ms.

|  |  |
| --- | --- |
|  |  |

- Le script suivant lit en boucle une liste de fréquences correspondant à des notes de la gamme tempérée. Complétez le code pour délivrer un signal PWM avec le timer et le canal appropriés d’un rapport cyclique de 10 %. Voir documentation sur <https://docs.micropython.org/en/latest/pyboard/quickref.html#timers>

'''

Lecture de la liste des fréquences contenant une série de notes

Buzzer connecté broche D5

'''

**import** pyb

**import** time

# Liste des notes qui seront jouées par le buzzer

liste\_frequences **=** **[**261**,** 277**,** 293**,** 329**,** 349**,** 392**,** 440**,** 494**]**

D5 **=** pyb**.**Pin**.**cpu**.**A15 # D5 génère une PWM avec TIM2, CH1

Buzzer **=** pyb**.**Pin**(**D5**)**

# programme principal

**while** **True** **:**

# Parcours de la liste

**for** frequence **in** liste\_frequences **:**

# La fréquence est fixée dans l'itération

timer\_2 **=** pyb**.**Timer**(**2**,** freq **=** frequence**)**

# configuration : Timer 2, canal 1, rapport cyclique de 10 %

canal\_1**=**timer\_2**.**channel**(**1**,**pyb**.**Timer**.**PWM**,** pin**=**Buzzer**,** pulse\_width\_percent**=**10**)**

time**.**sleep\_ms**(**500**)** # attente de 500 ms entre chaque note

**Remarque 1 :** vous pouvez jouer votre propre mélodie en choisissant les notes dans la gamme tempérée ci-dessous.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Octave** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **do** | 32.70 | 65.41 | 130.8 | 261.6 | 523.3 | 1046.5 | 2093.0 | 4186.0 | 8372.0 | 16 744 |
| **do# réb** | 34.65 | 69.30 | 138.6 | 277.2 | 554.4 | 1108.7 | 2217.5 | 4434.9 | 8869.8 | 17 740 |
| **ré** | 36.71 | 73.42 | 146.8 | 293.7 | 587.3 | 1174.7 | 2349.3 | 4698.6 | 9397.3 | 18 795 |
| **ré# mib** | 38.89 | 77.78 | 155.6 | 311.1 | 622.3 | 1244.5 | 2489.0 | 4978.0 | 9956.1 | 19 912 |
| **mi** | 41.20 | 82.41 | 164.8 | 329.6 | 659.3 | 1318.5 | 2637.0 | 5274.0 | 10 548 | 21 096 |
| **fa** | 43.65 | 87.31 | 174.6 | 349.2 | 698.5 | 1396.9 | 2793.8 | 5587.7 | 11 175 | 22 351 |
| **fa# solb** | 46.25 | 92.50 | 185.0 | 370.0 | 740.0 | 1480.0 | 2960.0 | 5919.9 | 11 840 | 23 680 |
| **sol** | 49.00 | 98.00 | 196.0 | 392.0 | 784.0 | 1568.0 | 3136.0 | 6271.9 | 12 544 | 25 088 |
| **sol# lab** | 51.91 | 103.8 | 207.7 | 415.3 | 830.6 | 1661.2 | 3322.4 | 6644.9 | 13 290 | 26 580 |
| **la** | 55.00 | 110.0 | 220.0 | **440.0** | 880.0 | 1760.0 | 3520.0 | 7040.0 | 14 080 | 28 160 |
| **la# sib** | 58.27 | 116.5 | 233.1 | 466.2 | 932.3 | 1864.7 | 3729.3 | 7458.6 | 14 917 | 29 834 |
| **si** | 61.74 | 123.5 | 246.9 | 493.9 | 987.8 | 1975.5 | 3951.1 | 7902.1 | 15 804 | 31 609 |

**Remarque 2 :** sur lemême principe, on peut mettre en œuvre un servomoteur dont l’angle de rotation, compris entre 0 à 180°, est fonction du signal PWM de commande.

# Ce qu’il faut retenir

**A) Configurations des entrées et sorties des microcontrôleurs**

D’une façon générale, une broche physiquement accessible d’un port d’un microcontrôleur est directement reliée à une cellule configurable. Selon les possibilités de la cellule, plusieurs configurations typiques existent. Le tableau suivant présente les principales configurations rencontrées. On ne détaillera pas l’analyse des schémas électroniques qui dépassent le cadre de cette étude.

|  |  |
| --- | --- |
| ***Entrée logique*** | |
| *Schéma simplifié :*    • Le circuit « trigger de schmitt » permet la mise en forme du signal logique d’entrée (tension) toujours susceptible d’être perturbé (bruit) ou aux transitions niveau haut niveau bas insuffisamment franches.  • Une entrée (même inutilisée) ne doit jamais rester en « l’air » ; le niveau logique doit être garanti. | *Modèle idéalisé et son analyse :*    VLT : low threshold voltage (seuil bas)  VHT : high threshold voltage (seuil haut)  • IIN = 0  • si VIN < VLT alors VO = VSS  • si VIN > VHT alors VO = VDD  • si VLT < VIN < VHT alors  VO conserve sa valeur (VSS ou VDD) |
| ***Résistance de tirage interne ou externe*** | |
| *Schéma simplifié d’une résistance de tirage vers le niveau haut (pull-up resistor) :*    • Une résistance de tirage vers le niveau haut permet de fixer le niveau haut en entrée si besoin.  • Elle peut être interne au microcontrôleur ou dans le circuit extérieur. | *Schéma simplifié d’une résistance de tirage vers le niveau bas (pull-down resistor) :*    • Une résistance de tirage vers le niveau bas permet de fixer le niveau bas en entrée si besoin.  • Elle peut être interne au microcontrôleur ou dans le circuit extérieur. |
| ***Sortie logique à étage push-pull :*** | |
| *Schéma simplifié :*    • TN et TP sont des transistors respectivement NMOS (canal N) et PMOS (canal P).  • En fonctionnement en commutation un transistor MOS est soit bloqué (idéalement équivalent à un interrupteur ouvert), soit fortement conducteur (idéalement équivalent à un interrupteur fermé).  • En toute rigueur, le point de fonctionnement (VOUT , IOUT) dépend des caractéristiques des transistors, de leur commande, et du circuit extérieur. | *Modèle idéalisé et son analyse :*    • TN ouvert et TP ouvert : VOUT dépend uniquement du circuit extérieur ; IOUT = 0. La sortie est dite en haute impédance.  • TN fermé et TP ouvert : VOUT = VSS = 0 V ; IOUT = IDN  • TN ouvert et TP fermé : VOUT = VDD ; IOUT = IDP  • TN fermé et TP fermé : interdit, car court-circuit |
| ***Sortie logique à Drain Ouvert*** | |
| *Schéma simplifié :*    • En toute rigueur, le point de fonctionnement (VOUT , IOUT) dépend des caractéristiques du NMOS, de sa commande, et du circuit extérieur. | *Modèle idéalisé et son analyse :*    • TN ouvert : VOUT dépend du circuit extérieur ; IOUT = 0.  • TN fermé : VOUT = VSS = 0 V ; IOUT = IDN |
| *Réalisation d’un « OU câblé » :*  • La configuration en drain ouvert peut permettre de réaliser une fonction logique dite « OU câblé » de plusieurs sorties en adjoignant une résistance de tirage vers le niveau haut.  *Analyse :*  • S = 0 si l’interrupteur A est fermé ou si l’interrupteur B est fermé ou si l’interrupteur C est fermé, d’où : | *Modèle idéalisé :* |
| ***Entrée analogique*** | |
| • Afin de pouvoir être traitée par un processeur, une grandeur analogique –souvent une tension – doit être convertie en un nombre par un CAN (Convertisseur Analogique Numérique), ou ADC (Analog to Digital Converter). | |
| *Exemple :*  • Soit un CAN unipolaire en codage binaire sur 8 bits et une tension d’entrée possible entre 0 V et 3,3 V. On cherche la valeur numérique délivrée par le CAN pour une tension d’entrée de 1,5 V.        Avec le modèle idéalisé ci-contre, le changement de valeur numérique s’effectue pour . D’où :  pour | *Modèle idéalisé :*  • Un CAN convertit un signal analogique « e » en un signal numérique « A ». La transmittance idéale d'un CAN linéaire est :  et  où  est le quantum (plus petite variation détectable)  est la plage pleine échelle (Full Scale Range)  est le nombre de valeurs numériques possibles  est l'erreur de quantification  avec (erreur centrée) |
| ***Sortie analogique*** | |
| • La génération d’une grandeur analogique –souvent une tension – à partir d’un nombre nécessite un CNA (Convertisseur Numérique Analogique), ou DAC (Digital to Analog Converter). | |
| *Exemple :*  • Soit un CNA unipolaire en codage binaire sur 8 bits et une tension de sortie possible entre 0 V et 3,3 V. On cherche la valeur analogique délivrée par le CNA pour le nombre 116, puis pour 117.        ⇒  ⇒ | *Modèle idéalisé :*  • Un CNA convertit un signal numérique « A » en un signal analogique « s ». La transmittance idéale d'un CNA linéaire est :  et  où  est le quantum (plus petite variation détectable)  est la plage pleine échelle (Full Scale Range)  est le nombre de valeurs numériques possibles |
| *Modulation de Largeur d’Impulsion :*  • En l’absence de circuit spécialisé CNA, on peut cependant produire un signal analogique par MLI (Modulation de Largeur d’Impulsion) ou PWM (Pulse Width Modulation) – et filtrage pour obtenir la valeur moyenne.  où :  est le rapport cyclique (dutty cycle), | *Modèle idéalisé :* |
| ***Fonctions alternatives*** | |
| Divers circuits spécialisés sont intégrés aux microcontrôleurs pour assurer des fonctions particulières et décharger le processeur. Pour communiquer avec l’extérieur, ils utilisent les entrées et sorties dûment configurées des microcontrôleurs. | |

**B) Lecture des entrées**

Il existe deux principes de base pour lire des entrées :

* la scrutation (polling) des entrées dans une itération permanente ;

-------------------------------------------------------------------------------

// itération permanente

**itérer**

// acquisition des entrées (polling)

**fin\_itérer**

-------------------------------------------------------------------------------

* le déclenchement d’interruptions par des entrées.

-------------------------------------------------------------------------------

// procédure appelée à chaque demande d’interruption accordée

**procédure** interruption\_x()

**début**

…

**fin**

-------------------------------------------------------------------------------

|  |  |
| --- | --- |
| • Une interruption accordée provoque :   * la suspension de l'exécution du programme en cours ; * la sauvegarde du contexte d’exécution ; * l’exécution du sous-programme d’interruption ; * la restitution du contexte sauvegardé ; * la reprise de l’exécution du programme suspendu. |  |

# Documentation

Site officiel de microPython :

<https://micropython.org/>

<https://micropython.fr/>

Documentation générale sur microPython :

<http://docs.micropython.org/en/latest/>

Exemples d’utilisation de la bibliothèque pyb (très utile) :

<http://docs.micropython.org/en/latest/pyboard/quickref.html#general-board-control>

Code source du projet microPython :

<https://github.com/micropython/micropython>

Documentation technique de la NUCLEO-WB55 :

<https://www.st.com/en/evaluation-tools/p-nucleo-wb55.html>

Les bonnes pratiques de la programmation python :

<https://python.sdv.univ-paris-diderot.fr/15_bonnes_pratiques/>

Documentation de l’application Android BLE ST Sensor :

<https://github.com/STMicroelectronics/STBlueMS_Android>