



Institut Supérieur de l'Aéronautique et de l'Espace ISAE-SUPAERO
Supaéro Space Section

Sparrow

Cours de programmation

Florian Topeza
28/11/2024



Table des matières

I Microcontrôleur Raspberry Pi Pico	5
I.1 Présentation du microcontrôleur	6
I.2 Fonctionnement des Pins de la Raspberry Pi Pico	7
I.2.1 GPIO (General-Purpose Input/Output)	7
I.2.2 ADC (Analog-to-Digital Converter)	7
I.2.3 UART (Universal Asynchronous Receiver/Transmitter)	8
I.2.4 SPI (Serial Peripheral Interface)	8
I.2.5 I2C (Inter-Integrated Circuit)	8
I.2.6 PWM (Pulse Width Modulation)	9
I.2.7 Power	9
I.2.8 System control	10
I.2.9 SWD (Serial Wire Debug)	10
I.2.10 LED interne	10
II Programmation en MicroPython	11
II.1 Thonny	12
II.2 Librairies	13
II.2.1 Rappels généraux	13
II.2.2 Librairies importantes pré-installées sur une Raspberry Pi Pico avec MicroPython	14
II.3 Classes	15
II.3.1 Rappels généraux	15
II.3.2 Classes importantes sur une Raspberry Pi Pico	16
II.4 Écriture de fichiers en MicroPython sur Raspberry Pi Pico	19

Table des figures

1	Raspberry Pi Pico	6
2	Pinout de la Raspberry Pi Pico	7
3	Emplacement du bouton BOOTSEL sur la carte	12
4	Shell Thonny	12

Liste des tableaux

1	Comparaison des performances du Raspberry Pi Pico par rapport à l'Arduino Nano et un ordinateur de moyenne gamme	6
---	--	---

Première partie

Microcontrôleur Raspberry Pi Pico

I.1 Présentation du microcontrôleur



FIGURE 1 – Raspberry Pi Pico

Le Raspberry Pi Pico utilise un processeur RP2040 développé par Raspberry Pi. Il s'agit du carré noir visible au centre de la carte. Pour bien comprendre les capacités du Raspberry Pi Pico et de son microcontrôleur RP2040, il est utile de le comparer à un ordinateur de moyenne gamme et à une Arduino Nano, microcontrôleur de la famille Arduino équivalent en terme de taille au Pico. Voici un tableau comparatif qui permet de visualiser les différences majeures entre ces trois systèmes.

Caractéristiques	Raspberry Pi Pico (RP2040)	Arduino Nano	Ordinateur Moyenne Gamme
Processeur (CPU)	2 cœurs ARM Cortex-M0+ à 133 MHz	1 cœur ATmega328P à 16 MHz	Processeur x86_64 (ex : Intel Core i5) à 2,5-4 GHz
Architecture	ARM Cortex-M0+ 32 bits	AVR 8 bits	x86_64 64 bits
Mémoire RAM	264 Ko de SRAM	2 Ko de SRAM	8 Go - 16 Go DDR4
Mémoire Flash	2 Mo de Flash	32 Ko de Flash	256 Go - 1 To de stockage (SSD/HDD)
Vitesse du bus	Jusqu'à 133 MHz	16 MHz (fréquence du MCU)	Plusieurs GHz
Entrées/Sorties (I/O)	26 GPIO, 2 × UART, 2 × SPI, 2 × I2C, 3 × ADC	14 GPIO (6 PWM), 1 × UART, 1 × SPI, 1 × I2C, 8 ADC	USB 3.0, Ethernet, HDMI, etc.
Consommation électrique	1,8 V - 3,3 V	5 V (via USB ou Vin)	100 W - 400 W (selon usage)
Systèmes d'exploitation	Bare metal, MicroPython, FreeRTOS	Bare metal (Arduino IDE)	Windows, macOS, Linux
Coût	4 USD	20 USD	500 - 1500 USD

TABLE 1 – Comparaison des performances du Raspberry Pi Pico par rapport à l'Arduino Nano et un ordinateur de moyenne gamme

Le Raspberry Pi Pico offre donc de bien meilleures performances et plus de possibilités d'usage qu'une Arduino Nano. En particulier, son processeur à 2 coeurs est nettement plus puissant que le processeur de la Nano, et sa mémoire Flash de 2Mo permet de téléverser des scripts assez longs et d'écrire des données dans un fichier sur la carte, tandis que sur la Nano un script un peu long est suffisant pour excéder la capacité de sa mémoire !

I.2 Fonctionnement des Pins de la Raspberry Pi Pico

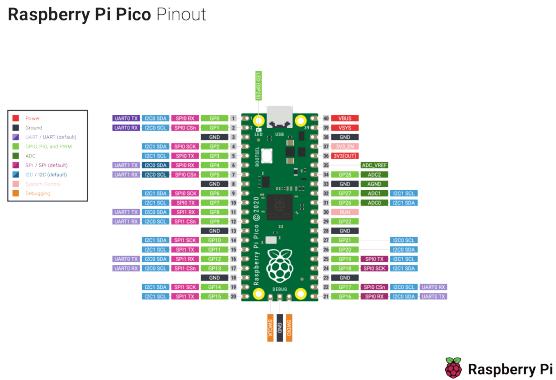


FIGURE 2 – Pinout de la Raspberry Pi Pico

La Raspberry Pi Pico est équipée d'un ensemble de broches (pins) qui offrent une grande variété de fonctionnalités pour le développement de projets électroniques. Ces pins permettent de connecter divers capteurs, modules, et autres périphériques pour interagir avec le microcontrôleur RP2040. Voici une explication des différentes fonctions des pins de la Raspberry Pi Pico. **La programmation des protocoles UART, SPI, I2C et PWM est expliquée dans la section Classes de la partie Programmation en MicroPython du présent document.**

I.2.1 GPIO (General-Purpose Input/Output)

Les pins GPIO peuvent être configurés comme des entrées ou des sorties numériques. Cela permet de lire l'état d'un bouton, d'allumer une LED, ou de contrôler d'autres composants électroniques. La Raspberry Pi Pico dispose de 26 GPIO numérotés de 0 à 25. Chaque GPIO peut être configuré en mode d'entrée, sortie, ou dans certains cas, en entrée/sortie analogique.

I.2.2 ADC (Analog-to-Digital Converter)

Les pins ADC permettent de lire des signaux analogiques, c'est-à-dire des valeurs variables comme la tension d'un capteur analogique. Les GPIO 26, 27 et 28 sont les pins ADC, capables de lire des signaux analogiques avec une résolution de 12 bits. Lecture de capteurs analogiques tels que des potentiomètres, des capteurs de température, ou des capteurs de lumière.

I.2.3 UART (Universal Asynchronous Receiver/Transmitter)

Les pins UART sont utilisés pour la communication série avec d'autres microcontrôleurs, ordinateurs ou périphériques série avec des modules GPS ou Bluetooth.

Il n'y a pas de signal d'horloge partagé entre les dispositifs, ce qui signifie que la synchronisation est gérée par le débit en bauds, que les deux parties doivent configurer de manière identique.

Le pin TX de la Pico doit être connecté au pin RX du périphérique et vice versa. Par exemple, si vous connectez un module GPS à la Pico, le TX du GPS doit aller au RX de la Pico et le RX du GPS doit aller au TX de la Pico. Sur certains dispositifs cependant, les pins doivent être connectés non pas de façon croisée, mais avec TX sur TX et RX sur RX. Ce cas est assez rare.

La Pico fonctionne à 3,3V, donc assurez-vous que les périphériques que vous connectez sont également compatibles avec 3,3V ou utilisez des convertisseurs de niveau de tension si nécessaire. Certains périphériques utilisent en effet du UART sur 5V par exemple.

La Pico dispose de deux contrôleurs UART. Les GPIO 0 (TX) et 1 (RX) pour le UART0, et GPIO 4 (TX) et 5 (RX) pour le UART1.

I.2.4 SPI (Serial Peripheral Interface)

Le protocole SPI est utilisé pour la communication rapide avec des périphériques comme des écrans, des mémoires flash, ou des capteurs. C'est un protocole de communication série synchrone permettant une communication rapide entre un maître (comme la Pico) et un ou plusieurs périphériques esclave. SPI utilise quatre fils : TX/MOSI, RX/-MISO, SCK/CLK, et CS/SS.

MOSI (Master Out Slave In) est la ligne où le maître envoie des données, MISO (Master In Slave Out) est la ligne où le maître reçoit des données, SCK (Serial Clock) est la ligne d'horloge qui synchronise le transfert de données et CS (Chip Select) est la ligne qui sélectionne l'esclave actif.

Le pin MOSI du maître doit être connecté au pin MOSI de chaque esclave, et le pin MISO du maître doit être connecté au pin MISO de chaque esclave. Le pin SCK doit être partagé entre tous les périphériques, et chaque esclave doit avoir son propre pin SS relié au maître.

Le pin CS (Chip Select) doit être utilisé pour sélectionner l'esclave avec lequel le maître souhaite communiquer. Chaque esclave doit être configuré pour être sélectionné en mettant son pin CS à LOW.

La Pico dispose de deux contrôleurs SPI. Les Pins disponibles pour SPI0 sont les pins GPIO 16 (TX/MOSI), 17 (RX/MISO), 18 (SCK/CLK), 19 (CS/SS), tandis que les pins disponibles pour le SPI1 sont les pins GPIO 12 (TX/MOSI), 13 (RX/MISO), 14 (SCK/CLK), 15 (CS/SS).

I.2.5 I2C (Inter-Integrated Circuit)

I2C est un protocole de communication permettant la connexion de multiples périphériques avec un minimum de fils. C'est idéal pour des capteurs et modules nécessitant une communication bidirectionnelle. Ce protocole est généralement utilisé pour la communication avec des modules RTC, des capteurs de température, ou des écrans LCD.

SDA (Serial Data Line) est la ligne de données sur laquelle les informations sont transmises. SCL (Serial Clock Line) est la ligne d'horloge qui synchronise les transferts

de données.

Les lignes SDA et SCL doivent être connectées aux lignes SDA et SCL du périphérique respectivement. Par exemple, si vous connectez un écran LCD I2C, le SDA du Pico va au SDA de l'écran, et le SCL du Pico va au SCL de l'écran.

Les lignes SDA et SCL nécessitent des résistances de pull-up pour fonctionner correctement. La Pico intègre des résistances de pull-up internes, mais pour des longueurs de bus plus importantes ou des vitesses élevées, des résistances externes (typiquement $4.7\text{k}\Omega$) sont recommandées.

La Pico dispose de deux contrôleurs I2C. Les pins disponibles pour I2C0 sont les pins GPIO 4 (SDA), 5 (SCL). Pour I2C1, ce sont GPIO 6 (SDA), 7 (SCL).

I.2.6 PWM (Pulse Width Modulation)

Le PWM (Pulse Width Modulation) est une technique utilisée pour simuler une tension analogique à l'aide d'un signal numérique. Le principe du PWM repose sur la génération d'un signal carré dont la fréquence est fixe, mais dont la largeur des impulsions (c'est-à-dire la durée pendant laquelle le signal est à un niveau haut) peut varier.

La proportion du temps pendant lequel le signal est à l'état haut par rapport à la période totale du signal est appelée le "duty cycle". Par exemple, un duty cycle de 50 % signifie que le signal est à l'état haut pendant 50 % du temps et à l'état bas pendant les 50 % restants. En modifiant ce rapport, le PWM permet de contrôler l'intensité d'un courant ou la vitesse d'un moteur, la luminosité d'une LED, etc.

Tous les GPIO peuvent être utilisés pour le PWM (jusqu'à 16 canaux PWM en tout). Le PWM permet le contrôle de la vitesse des moteurs, de la position des servos, ou de l'intensité lumineuse.

I.2.7 Power

- **VBUS (USB Power Input)** : Ce pin est relié au pin VSYS via une diode Schottky, qui permet le passage du courant depuis le pin VBUS vers le pin VSYS, mais bloque le courant dans le sens opposé. Lorsque la carte est alimentée via son connecteur micro USB, le pin VBUS alimente la carte et débite également une tension de 5V qui peut servir à alimenter d'autres composants. **En résumé, VBUS fournit 5V uniquement lorsque la carte est connectée à l'USB, et cette tension peut être utilisée pour alimenter d'autres composants externes.** **Attention, si la carte est alimentée par USB et que le pin VBUS est connecté à la masse, la carte refusera de fonctionner.**
- **VSYS (System Power In)** : Alimente la Raspberry Pi Pico. Lorsque la carte est alimentée par son port micro USB, le courant qui arrive en VBUS accède à VSYS via la diode Shottky ce qui alimente la carte. Lorsque la carte est alimentée autrement, par exemple par une batterie, l'alimentation doit être connectée au pin VSYS, qui alimente directement la carte. **En résumé, VSYS est le pin d'alimentation principal de la Raspberry Pi Pico, et il doit recevoir une alimentation externe comprise entre 1,8V et 5,5V, ou via VBUS (quand l'USB est branché). Il ne débite pas de courant.**

- **3V3(OUT) (3.3V Output)** : Sortie de 3,3V régulée, utilisée pour alimenter des composants externes.
- **GND** : Les pins de masse (Ground) sont essentiels pour compléter les circuits électriques et fournir une référence de tension commune.

I.2.8 System control

- **3V3_EN (Enable 3.3V Regulator)** : Le pin 3V3_EN est une entrée active basse qui contrôle le régulateur de 3,3V. Lorsqu'il est tiré à la masse (GND), il désactive le régulateur 3,3V, coupant l'alimentation à la fois au RP2040 et aux périphériques connectés au pin 3V3. Par défaut, ce pin est connecté en interne à VSYS via une résistance de pull-up, ce qui maintient le régulateur actif. Ce pin est utilisé pour désactiver l'alimentation 3,3V dans des scénarios où la consommation d'énergie doit être minimisée, comme dans les applications à très faible puissance ou lorsque la carte doit être mise en veille prolongée. **En résumé, si ce pin est mis à la masse, la carte cesse de fonctionner.**
- **RUN (Reset)** : Le pin RUN est une entrée active basse qui permet de contrôler l'alimentation de la carte. Lorsqu'il est tiré à la masse (GND), il désactive le régulateur 3,3V, coupant ainsi l'alimentation de la carte et mettant le RP2040 en mode "off". Lâcher ce pin (le laissant flotter ou connecté à VSYS via une résistance) redémarre le régulateur, alimentant à nouveau la carte. Ce pin est utilisé pour effectuer un redémarrage matériel (reset) de la carte ou pour mettre la carte en veille (consommation minimale) en désactivant l'alimentation. **En résumé, là encore, si ce pin est mis à la masse, la carte cesse de fonctionner.**

I.2.9 SWD (Serial Wire Debug)

Les pins SWD sont utilisés pour le débogage du code en temps réel et la programmation avancée de la Raspberry Pi Pico. Ils se situent aux pins GPIO 24 (SWDIO) et GPIO 25 (SWCLK).

I.2.10 LED interne

La Raspberry Pi Pico possède une LED intégrée connectée au GPIO 25. Cette LED peut être utilisée pour des indicateurs d'état ou des tests de code. Cette LED peut servir à indiquer le bon fonctionnement du microcontrôleur ou effectuer des tests rapides de code.

Deuxième partie

Programmation en MicroPython

II.1 Thonny

Pour téléverser du code sur la Raspberry Pi Pico, nous allons utiliser l'IDE Thonny, téléchargeable [ici](#). Pour connecter la Raspberry Pi Pico à l'ordinateur, il faut un câble USB-A vers micro USB qui puisse transmettre des données. Attention, certains câbles ne permettent que de recharger sans pouvoir transmettre de données.

Une fois Thonny ouvert et le câble branché à l'ordinateur, il faut maintenir le bouton BOOTSEL de la Pico appuyé tout en branchant la carte au câble. Cela met la carte en mode dispositif de stockage de masse USB et une fenêtre de l'explorateur de fichiers s'ouvre. Cette fenêtre peut être utile dans des cas de débogage de la carte, mais il n'y en n'a pas besoin ici.

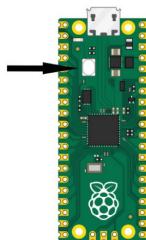


FIGURE 3 – Emplacement du bouton BOOTSEL sur la carte

Dans le coin en bas à droite de la fenêtre de Thonny, on peut voir la version de Python utilisée. Cliquez sur la version de Python et choisissez MicroPython (Raspberry Pi Pico). Si cette option n'apparaît pas, vérifiez le branchement de la Pico.

Observez maintenant la console en bas de l'éditeur Thonny, vous devriez voir quelque chose comme sur la capture d'écran suivante.

FIGURE 4 – Shell Thonny

Thonny est maintenant apte à communiquer avec la carte. Pour facilement naviguer dans les fichiers de l'ordinateur et de la carte, vérifier que l'option Fichiers est cochée dans le panneau Affichage situé en haut de Thonny.

Pour téléverser un fichier Python de l'ordinateur vers la carte, vous pouvez l'ouvrir dans Thonny puis aller dans **Fichiers** et **Enregistrer sous** puis enregistrez le fichier sur la carte. Si le fichier est accessible depuis l'arborescence des fichiers de l'ordinateur dans le panneau sur la gauche de l'éditeur Thonny, vous pouvez aller dessus, faire clic-droit et **Téléverser vers**.

II.2 Librairies

II.2.1 Rappels généraux

Une librairie en Python est un ensemble de modules, c'est-à-dire des fichiers Python contenant des fonctions, des classes et des variables, qui permettent d'accomplir des tâches spécifiques. Les librairies sont utilisées pour éviter de réinventer la roue en offrant du code réutilisable pour des fonctionnalités courantes, comme la manipulation de fichiers, la gestion du temps, l'interaction avec du matériel,...

Pour utiliser une librairie en Python, vous devez l'importer dans votre script. L'importation d'une librairie se fait avec le mot-clé import. Voici quelques exemples de base.

```
1 import math # Importe toute la librairie math
2 import os   # Importe toute la librairie os
```

Une fois importée, vous pouvez accéder aux fonctions et classes de la librairie en utilisant la syntaxe `nom_libreria.nom_fonction()`. Par exemple :

```
1 import math
2
3 resultat = math.sqrt(16) # Utilise la fonction sqrt de la
4 # librairie math pour calculer la racine carree de 16
5
6 print(resultat) # Affiche 4.0
```

Si vous ne souhaitez importer qu'une partie d'une librairie, vous pouvez spécifier un module ou une fonction particulière :

```
1 from math import sqrt # Importe uniquement la fonction sqrt
2                         # du module math
3
4 resultat = sqrt(25)
5 print(resultat) # Affiche 5.0
```

Vous pouvez aussi renommer une librairie ou une fonction lors de l'importation :

```
1 import math as m # Renomme math en m
2
3 resultat = m.sqrt(9)
4 print(resultat) # Affiche 3.0
```

II.2.2 Librairies importantes pré-installées sur une Raspberry Pi Pico avec MicroPython

La Raspberry Pi Pico, lorsqu'elle est utilisée avec MicroPython, est équipée de plusieurs librairies essentielles qui permettent d'interagir avec le matériel et de gérer des tâches de base. Voici quelques-unes des plus importantes :

machine

La librairie `machine` est essentielle pour interagir avec le matériel de la Raspberry Pi Pico. Elle permet de contrôler les GPIO (General Purpose Input/Output), les interfaces comme I2C, SPI, UART, les timers, les PWM, etc.

```
1 from machine import Pin
2
3 led = Pin(25, Pin.OUT) # Cree un objet Pin pour controler la
4 # LED integree
5
6 led.value(1) # Allume la LED
```

time

La librairie `time` fournit des fonctions pour gérer le temps, comme les délais, le temps écoulé, etc. Cette librairie est souvent utilisée pour créer des pauses ou pour chronométrier des événements.

```
1 import time
2
3 time.sleep(2) # Pause de 2 secondes
4 print("2 secondes se sont ecoulees")
```

utime

`utime` est une version spécifique à MicroPython de la librairie `time`, offrant des fonctionnalités similaires mais optimisées pour les microcontrôleurs.

```
1 import utime
2
3 start = utime.ticks_ms() # Obtenir le temps actuel en
4 # millisecondes
5 utime.sleep_ms(100) # Pause de 100 millisecondes
6 end = utime.ticks_ms()
7
8 print("Temps ecoule : ", utime.ticks_diff(end, start), "ms")
```

uos

La librairie `uos` (semblable à `os` en Python standard) est utilisée pour interagir avec le système de fichiers, accéder aux fichiers et répertoires, etc.

```

1 import uos
2
3 uos.listdir() # Liste les fichiers et répertoires dans le
4     # répertoire actuel

```

II.3 Classes

II.3.1 Rappels généraux

Une **classe** en Python est un modèle qui permet de créer des objets. Une classe regroupe des données sous forme d'attributs (variables) et des comportements sous forme de méthodes (fonctions) associées à ces données. Les classes permettent de définir des types d'objets personnalisés en regroupant à la fois les données et les fonctionnalités qui opèrent sur ces données.

Pour créer une classe en Python, on utilise le mot-clé `class`, suivi du nom de la classe et des deux points (:). À l'intérieur du bloc de la classe, on définit les méthodes, dont la première est généralement `__init__`. Cette méthode est un constructeur qui initialise les objets créés à partir de la classe. Voici un exemple simple :

```

1 class Voiture:
2     def __init__(self, marque, modèle, année):
3         self.marque = marque
4         self.modèle = modèle
5         self.année = année
6
7     def afficher_details(self):
8         print(f"Voiture: {self.marque} {self.modèle},"
9             f"Année: {self.année}")

```

- **Déclaration de la classe** : La classe `Voiture` est créée avec trois attributs : `marque`, `modèle`, et `année`.
- **Méthode `__init__`** :
 - Cette méthode spéciale est appelée automatiquement lors de la création d'un nouvel objet.
 - Elle initialise les attributs de l'objet avec les valeurs fournies.
 - Le paramètre `self` fait référence à l'instance courante de la classe, c'est-à-dire à l'objet lui-même.
- **Méthode `afficher_details`** :
 - Cette méthode est une méthode normale de la classe.
 - Elle affiche les détails de la voiture.
 - `self` est utilisé pour accéder aux attributs de l'objet.

Pour utiliser une classe, vous devez d'abord créer un objet (une instance) de cette classe. Vous pouvez ensuite accéder aux attributs et appeler les méthodes de l'objet. Exemple d'utilisation :

```
1 # Creation d'une instance de la classe Voiture
2 ma_voiture = Voiture("Toyota", "Corolla", 2020)
3
4 # Appel de la methode afficher_details pour afficher les
5 # informations de la voiture ma_voiture.afficher_details()
```

- **Création de l'objet** : `ma_voiture = Voiture("Toyota", "Corolla", 2020)`
 - Ici, un objet `ma_voiture` est créé à partir de la classe `Voiture`.
 - Les valeurs "Toyota", "Corolla", et 2020 sont passées au constructeur `__init__` pour initialiser l'objet.
- **Appel d'une méthode** : `ma_voiture.afficher_details()`
 - Cette ligne appelle la méthode `afficher_details` de l'objet `ma_voiture`, qui affiche les informations de la voiture.

II.3.2 Classes importantes sur une Raspberry Pi Pico

Sur une Raspberry Pi Pico, plusieurs classes importantes sont disponibles, surtout lorsque l'on utilise MicroPython. Ces classes sont essentielles pour interagir avec le matériel de la carte, contrôler les périphériques et gérer les entrées/sorties. Voici les principales classes à connaître.

Pin

La classe `Pin` est utilisée pour contrôler les broches GPIO (General Purpose Input/Output) de la Raspberry Pi Pico. Vous pouvez configurer chaque broche comme une entrée ou une sortie numérique, et lire ou écrire des valeurs sur ces broches.

```
1 from machine import Pin
2
3 # Configuration d'une broche en sortie
4 led = Pin(25, Pin.OUT)
5 led.value(1) # Allume la LED connectee a la broche 25
6
7 # Configuration d'une broche en entree
8 bouton = Pin(14, Pin.IN, Pin.PULL_DOWN)
9 etat_bouton = bouton.value() # Lit l'état du bouton (0 ou 1)
```

- `Pin.OUT` : Configure la broche en mode sortie.
- `Pin.IN` : Configure la broche en mode entrée.
- `Pin.PULL_DOWN / Pin.PULL_UP` : Active une résistance de pull-down ou de pull-up interne.

ADC

La classe `ADC` (Analog-to-Digital Converter) est utilisée pour lire des valeurs analogiques sur certaines broches de la Raspberry Pi Pico. Cela permet de lire des capteurs analogiques comme des potentiomètres ou des capteurs de température.

```
1 from machine import ADC
2
3 # Creation d'un objet ADC sur la broche 26 (GP26)
4 potentiometre = ADC(26)
5
6 # Lecture de la valeur analogique (entre 0 et 65535)
7 valeur = potentiometre.read_u16()
8 print("Valeur lue :", valeur)
```

— ADC.read_u16() : Retourne une valeur entre 0 et 65535 correspondant à la tension lue.

PWM

La classe PWM (Pulse Width Modulation) est utilisée pour générer des signaux PWM sur les broches GPIO. Le PWM est souvent utilisé pour contrôler la luminosité des LEDs, la vitesse des moteurs, etc.

```
1 from machine import Pin, PWM
2
3 # Configuration de la broche 15 en PWM
4 led_pwm = PWM(Pin(15))
5
6 # Reglage de la frequence du PWM
7 led_pwm.freq(1000) # 1 kHz
8
9 # Reglage du rapport cyclique (duty cycle) entre 0 (eteint) et
10 # 65535 (100% allume)
11 led_pwm.duty_u16(32768) # 50% de luminosite
```

— PWM.freq() : Définit la fréquence du signal PWM.
— PWM.duty_u16() : Définit le rapport cyclique en utilisant une valeur comprise entre 0 et 65535.

I2C

La classe I2C permet de communiquer avec des périphériques utilisant le bus I2C (Inter-Integrated Circuit), un protocole de communication série très répandu pour les capteurs et autres modules.

```
1 from machine import Pin, I2C
2
3 # Configuration de l'I2C broches GPIO8 (SDA) et GPIO9 (SCL)
4 i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
5
6 # Scanner les peripheriques I2C connectes
7 devices = i2c.scan()
```

```

8 print("Peripheriques I2C trouves : ", devices)
9
10 # Lire et ecrire des donnees sur un peripherique I2C
11 # Exemple : lire 2 octets a l'adresse 0x3C
12 data = i2c.readfrom(0x3C, 2)

```

- `I2C.scan()` : Scanne le bus I2C et retourne une liste des adresses des périphériques connectés.
- `I2C.readfrom()` : Lit des données d'un périphérique I2C.

SPI

La classe SPI est utilisée pour communiquer avec des périphériques utilisant le bus SPI (Serial Peripheral Interface), un autre protocole de communication série utilisé pour les écrans, capteurs, mémoires, etc.

```

1 from machine import Pin, SPI
2
3 # Configuration du SPI sur les broches GPIO10 (SCK), GPIO11 # (MOSI), G
4 spi = SPI(0, baudrate=1000000, polarity=0, phase=0,
5           sck=Pin(10), mosi=Pin(11), miso=Pin(12))
6
7 # Ecriture et lecture de donnees SPI
8 # Exemple : ecrire [0x01, 0x02] et lire 2 octets
9 spi.write([0x01, 0x02])
10 data = spi.read(2)

```

- Dans l'initialisation du SPI, le `baudrate` définit le débit de données entre le microcontrôleur et le périphérique. Le baudrate lors de l'initialisation doit être le baudrate utilisé par le périphérique.
- `SPI.write()` : Envoie des données sur le bus SPI.
- `SPI.read()` : Lit des données du bus SPI.

UART

La classe UART est utilisée pour la communication série via les broches UART. Elle est souvent utilisée pour communiquer avec des modules série, comme des GPS, des modems ou des modules Bluetooth.

```

1 from machine import UART
2
3 # Configuration du UART sur les broches GPIO0 (TX) et GPIO1 #
4 # (RX)
5 uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))
6
7 # Envoyer des donnees
8 uart.write('Hello, UART!')

```

```
9  
10 # Lire des données  
11 data = uart.read(10) # Lit 10 octets de données
```

- Comme pour la classe SPI, le baudrate choisi doit être celui utilisé par le périphérique connecté au microcontrôleur.
- `UART.write()` : Envoie des données sur le bus UART.
- `UART.read()` : Lit les données reçues par le bus UART.

Timer

La classe `Timer` permet de créer des minuteries matérielles, souvent utilisées pour exécuter des fonctions à des intervalles réguliers sans bloquer le programme principal.

```
1 from machine import Timer  
2  
3 # Creation d'une minuterie qui execute une fonction toutes les  
4 # 2 secondes  
5 def clignotement(t):  
6     print("Timer declenche !")  
7  
8 timer = Timer()  
9 timer.init(period=2000, mode=Timer.PERIODIC, callback=clignotement)
```

- `Timer.PERIODIC` : Mode périodique où la fonction est appelée à intervalles réguliers.
- `Timer.ONE_SHOT` : Mode où la fonction est appelée une seule fois après le délai spécifié.

II.4 Écriture de fichiers en MicroPython sur Raspberry Pi Pico

En MicroPython, il est possible d'écrire des données dans un fichier sur la mémoire interne de la Raspberry Pi Pico. Cette opération est utile pour stocker des données de manière persistante, comme des journaux, des configurations ou des résultats de capteurs.

Pour écrire des données dans un fichier en MicroPython, on utilise la fonction `open()` pour ouvrir (ou créer) un fichier. Ensuite, on utilise la méthode `write()` pour écrire des données dans le fichier. Enfin, il est important de fermer le fichier avec `close()` pour s'assurer que toutes les données sont correctement enregistrées.

La commande `file.flush()` permet de vider le tampon d'écriture d'un fichier et forcer l'écriture immédiate des données sur le disque. Normalement, les données écrites dans un fichier peuvent être temporairement stockées dans un tampon avant d'être enregistrées sur le disque. Utiliser `flush()` est particulièrement utile pour s'assurer que les données sont enregistrées immédiatement, par exemple, si vous craignez une coupure d'alimentation ou un arrêt soudain du programme. Cela permet de minimiser les risques de perte de données.

```

1 # Ouvrir (ou créer) un fichier nommé "donnees.txt" en mode
2 # écriture
3 with open("donnees.txt", "w") as fichier:
4     # Écrire des données dans le fichier
5     fichier.write("Bonjour, Raspberry Pi Pico !\n")
6     fichier.write("Ceci est un exemple d'écriture dans un
7     fichier.\n")
8
9 # Le fichier est automatiquement fermé à la fin du bloc with

```

Si vous voulez ajouter des données à un fichier existant sans le remplacer, utilisez le mode 'a' :

```

1 # Ouvrir le fichier en mode ajout
2 with open("donnees.txt", "a") as fichier:
3     # Ajouter des données au fichier
4     fichier.write("Ajout de nouvelles données.\n")

```

- 'w' : Ouvre un fichier en mode écriture. Si le fichier existe, son contenu sera supprimé avant que le nouvel écrit ne commence.
- 'a' : Ouvre un fichier en mode ajout. Les nouvelles données seront ajoutées à la fin du fichier sans supprimer le contenu existant.
- 'r' : Ouvre un fichier en mode lecture (sans modification possible).

Dans cet exemple, la méthode `flush()` garantit que la ligne écrite est immédiatement sauvegardée sur le disque, même si le fichier n'est pas encore fermé.

```

1 # Ouvrir un fichier en mode écriture
2 fichier = open("donnees.txt", "w")
3
4 # Écrire des données dans le fichier
5 fichier.write("Écriture de données importantes.\n")
6
7 # Forcer l'enregistrement des données sur le disque
8 fichier.flush()
9
10 # Fermer le fichier
11 fichier.close()

```

Il est aussi possible d'écrire des données sur une carte SD séparée, connectée à la Raspberry Pi Pico en SPI. Cela nécessite de téléverser la librairie `sdcards` sur la carte et de l'importer au début du programme, ainsi que d'importer la librairie `os` et les classes `Pin` et `SPI` de la librairie `machine`. Il faut ensuite initialiser la communication avec la carte et la monter sur le système de la Raspberry Pi Pico comme suit.

```

1 # Importations pour la carte SD
2 import os

```

```
3 from machine import Pin, SPI
4 import sdcard
5
6 # Configuration de la carte SD
7 SD_CS_PIN = 5
8 SD_SCK_PIN = 6
9 SD_MOSI_PIN = 7
10 SD_MISO_PIN = 4
11 spi = SPI(0, baudrate=100000, sck=Pin(SD_SCK_PIN),
12             mosi=Pin(SD_MOSI_PIN), miso=Pin(SD_MISO_PIN))
13
14 sd = sdcard.SDCard(spi, Pin(SD_CS_PIN))
15 vfs = os.VfsFat(sd)
16 os.mount(vfs, '/sd')
17
18 # Ouverture d'un fichier data.csv sur la carte SD
19 file = open('/sd/data.csv', 'a')
20 file.write("test\n")
21 file.flush()
22
23 # On ferme le fichier et on demonte la carte SD
24 os.umount('/sd')
```