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

Sparrow

Programmation et Systèmes embarqués

Florian Topeza
28 novembre 2024

Table des matières

Table des figures

Liste des tableaux

Introduction

Ce polycopié est le support du cours de *Programmation et Systèmes embarqués* enseigné à l'ISAE-SUPAERO dans le cadre du projet de formation Sparrow 2024 de la Supaéro Space Section.

Ce document reprend tout le contenu du cours magistral, avec quelques approfondissements. Ces approfondissements, signalés par un titre en gras et une ligne verticale à gauche, ouvrent quelques pistes de réflexion sur des sujets abordés par le cours mais ne sont pas nécessaires au projet Sparrow.

Les ressources complémentaires au cours, et notamment les codes en MicroPython, sont disponible dans ce répertoire GitHub.

Pour toute remarque, suggestion ou correction concernant ce document, merci de me contacter pour que je puisse modifier et corriger ce polycopié.

Première partie

Microcontrôleur Raspberry Pi Pico

I.1 Introduction

Pour voler, une fusée Sparrow doit embarquer des éléments électroniques : un servo-moteur pour ouvrir et fermer la trappe du parachute, un buzzer dont le son indique l'état de la fusée (prête à déoller, en vol...) et des capteurs (IMU et baromètre) pour récupérer les données de vol. Ces différents éléments nécessitent d'être pilotés pour fonctionner.

Quoi utiliser pour contrôler l'électronique de la fusée ? Il nous faut quelque chose de compact pour rentrer dans la fusée et qui consomme peu d'énergie. Un ordinateur est donc bien évidemment exclu. Un micro-ordinateur (comme ceux de Raspberry) est déjà plus compact, mais il consomme trop d'énergie ! En effet, même sans programme actif pour contrôler la fusée, le système d'exploitation du micro-ordinateur (OS) consomme de l'énergie. C'est pourquoi nous allons utiliser un microcontrôleur.

Un microcontrôleur, contrairement à un micro-ordinateur, n'a pas pas de système d'exploitation et peut être programmé en ne téléversant dessus que les programmes utiles. Bien souvent, un microcontrôleur peut être programmé de plusieurs façons différentes, plus ou moins complexes. Dans le cas de Sparrow, nous nous contenterons d'utiliser un environnement de développement intégré (IDE) qui permet d'écrire et téléverser simplement du code sur notre carte.

Reste à choisir le modèle de microcontrôleur à utiliser. Il nous faut un microcontrôleur avec suffisamment de mémoire pour stocker les programmes que l'on va téléverser dessus (pour Sparrow, cela représente quelques kilo-octets) mais surtout qui puisse stocker les données de vol enregistrées par les capteurs. Il nous faut également un processeur apte à effectuer rapidement et simultanément les tâches qu'on lui demande. Enfin, il faudrait un microcontrôleur simple d'utilisation, avec une bonne documentation existante. Ce dernier point peut sembler anecdotique, mais il est en réalité crucial : dans tout projet, mieux vaut utiliser du matériel éprouvé avec des ressources pour nous aider plutôt que du matériel inconnu au fonctionnement incertain (dans le domaine du spatial, cela est même formalisé par le concept de Technology Readiness Level ou TRL).

Avec tous ces éléments en tête, le microcontrôleur que nous allons utiliser est le Raspberry Pi Pico.

I.2 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 : plus de broches sur lequelles connecter des composants, son processeur à 2 coeurs est plus rapide et puissant que le processeur de la Nano, sa mémoire Flash de 2Mo permet de téléverser des scripts assez longs et d'écrire des données dans un fichier sur la carte, ce qui est impossible sur la Nano et la Pico offre beaucoup plus de possibilités en terme de programmation, pouvant être programmé en MicroPython, C/C++. Et tout ceci en étant moins chère que la Nano !

Dis Jamy, c'est quoi un coeur dans un processeur ?

> Utilisons une analogie. Imaginons que votre processeur soit une cuisine, et chaque coeur de votre processeur, un cuisinier. Si votre processeur n'a qu'un seul coeur, i.e. il n'y a qu'un seul cuisinier dans la cuisine, et que vous avez plusieurs plats à préparer, votre cuissinier devra alterner entre ces plats, ils ne pourra pas s'occuper de tous les plats en même temps. Par contre, si vous avez plusieurs cuisiniers, vous pourrez préparer plusieurs plats en même temps, sans risquer d'en brûler un pendant que vous vous occupiez d'un autre !
> C'est exactement ce qui se passe avec les coeurs d'un processeur. S'il n'y a qu'un seul coeur, il devra alterner entre les tâches en les avançant chacune par petits bouts, c'est ce qu'on appelle du multi-threading. Le risque, en faisant cela, est de se retrouver avec une tâche gourmande en ressources de calcul qui accapare le coeur, ce qui fait que les autres tâches n'avancent plus. Avec plusieurs coeurs, une tâche gourmande en ressources peut être effectuée par un coeur en particulier, sans gêner les autres tâches qui elles sont effectuées par les autres coeurs.

Pour les curieux...

> Je vous conseille cette vidéo YouTube (et plus généralement cette chaîne) si vous aimez l'électronique et que vous voulez aller plus loin dans la compréhension du fonctionnement du microcontrôleur. Vous trouverez également la documentation du RP2040 ici.

I.3 Fonctionnement des Pins de la Raspberry Pi Pico

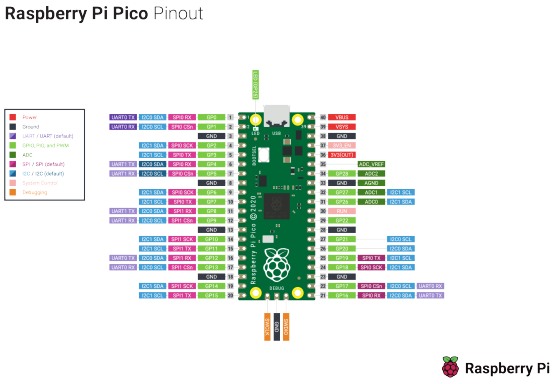


FIGURE 2 – Pinout de la Raspberry Pi Pico

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 de ce cours.

I.3.1 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.3.2 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.3.3 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.3.4 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 SPIO0 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.3.5 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 4.7kΩ) 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.3.6 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.

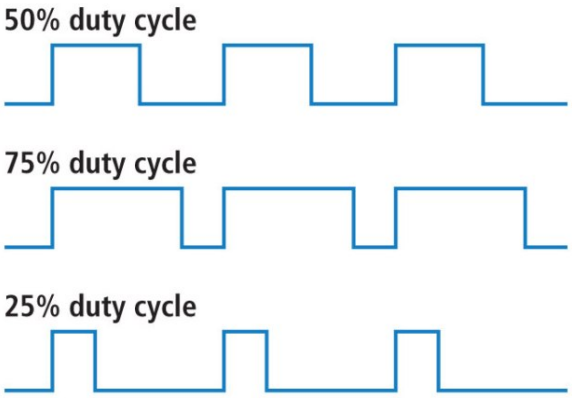
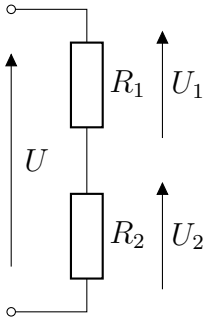


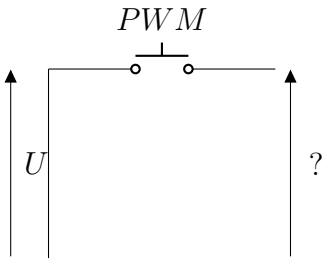
FIGURE 3 – Signaux PWM avec différents duty cycles

Application à la régulation de tension

Supposons que l'on dispose d'une alimentation qui fournit une tension U et que l'on veut diviser cette tension par deux pour alimenter un système. Une première idée, assez simple, serait d'utiliser un pont diviseur de tension.

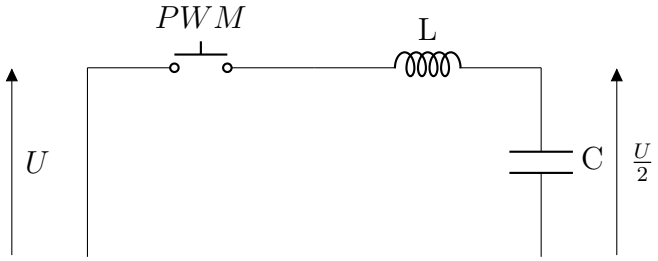


Comme $U_2 = \dfrac{R_2}{R_1 + R_2} U$, en prenant $R_1 = R_2$, on obtient bien $U_2 = \dfrac{U}{2}$.
Cependant, un tel dispositif n'est pas efficace du fait de l'effet Joule. Beaucoup d'énergie est perdue par dissipation thermique. Une stratégie bien plus efficace, mais plus subtile consiste à utiliser un interrupteur commandé en PWM, avec un duty cyle de 50%.



La tension obtenue en sortie est alors un signal créneau, dont la valeur moyenne (intégrale du signal sur une période, divisée par la période) vaut la moitié de le tension d'entrée. On veut alors traiter ce signal créneau pour ne garder que sa valeur moyenne, aussi appelée composante continue du signal. On va donc filtrer toutes les harmoniques du signal. Pour cela, on utilise un filtre passe-bas, dont la fréquence de coupure est au moins 10 fois plus petite que la fréquence du fondamental du signal carré, qui est $f_0 = \frac{1}{T}$ où T est la période du signal PWM.

On choisit un filtre LC plutôt qu'un filtre RC, afin d'éviter les pertes par effet Joule. En traitement du signal, on aurait au contraire privilégié un filtre RC pour éviter les phénomènes magnétiques dus à l'inductance. Pour dimensionner L et C, il faut alors $f_c = \frac{1}{2\pi\sqrt{LC}} \leq \frac{f_0}{10}$.



On obtient alors une tension de sortie égale à la moitié de la tension d'entrée, avec un rendement bien mieux qu'un pont diviseur de tension. C'est sur ce principe que fonctionne les réulateurs de tension à découpage, dont le rendement peut atteindre 95% !

I.3.7 Power

VBUS (USB Power Input) 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, c'est un court-circuit et la carte ne fonctionnera pas !.

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 la broche d'alimentation principale 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). Elle ne débite pas de courant.

3V3(OUT) (3.3V Output) est une sortie de 3.3V régulée, utilisée pour alimenter des composants externes.

Enfin, les broches de masse **GND** (Ground) sont essentielles pour compléter les circuits électriques et fournir une référence de tension commune. Les broches de masse des composants du circuits doivent être connectés aux broches de masse de la Pico

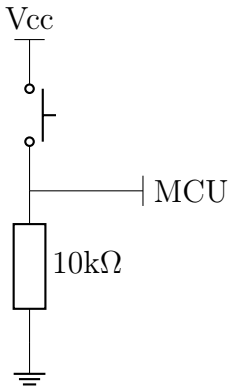
I.3.8 System control

3V3_EN (Enable 3.3V Regulator) est une entrée qui contrôle le régulateur de 3.3V de la Pico. 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 cette broche est mise à la masse, la carte cesse de fonctionner.

Résistance de pull-up, résistance de pull-down

> Les résistances de pull-up et de pull-down (respectivement appelées résistances de tirage et de rappel dans la langue de Molière) sont des résistances placées pour éviter que des broches ne soient flottantes, ce qui signifie que la tension de la broche varie aléatoirement quand rien n'y est connecté. Cela peut se produire par exempl si une broche est connectée à un bouton poussoir. Lorsque le bouton est pressé, la tension de la broche sera la tension d'alimentation du bouton, mais lorsque le bouton est relâché, la tension de la broche ne revient pas nécessairement à zéro, ce qui peut occasionner de fausses lectures de la valeur du signal sur la broche ! Une résistance de pull-down permet alors de mettre la broche à la masse lorsque celle-ci ne reçoit aucun signal, tandis qu'un résistance de pull-up maintient une tension sur la broche lorsque celle-ci n'est pas connectée.

> Voici un exemple de résistance de pull-down. Lorsque l'interrupteur est fermé, le courant passe par le chemin de moindre résistance et la broche du microcontrôleur reçoit donc la tension Vcc, tandis que lorsque l'interrupteur est ouvert, la broche est mise à la masse
>
> 
>
>
> Les broches de la Raspberry Pi Pico sont équipées de résistances de pull-up ou pull-down internes, qui peuvent être activées lorsqu'on programme le microcontrôleur, comme expliqué ici.

RUN (Reset) est une entrée qui permet de contrôler l'alimentation de la carte. Lorsqu'elle est tirée à la masse (GND), elle désactive le régulateur 3.3V, coupant ainsi l'alimentation de la carte et mettant le RP2040 en mode "off". Lâcher cette broche (la laissant flotter ou connectée à VSYS via une résistance) redémarre le régulateur, alimentant à nouveau la carte. Cette broche est utilisée 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 cette broche est mise à la masse, la carte cesse de fonctionner.

I.3.9 SWD (Serial Wire Debug)

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

I.3.10 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.

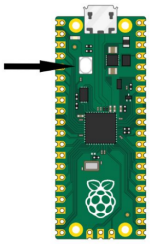


FIGURE 4 – Emplacement du bouton BOOTSEL sur la carte

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.

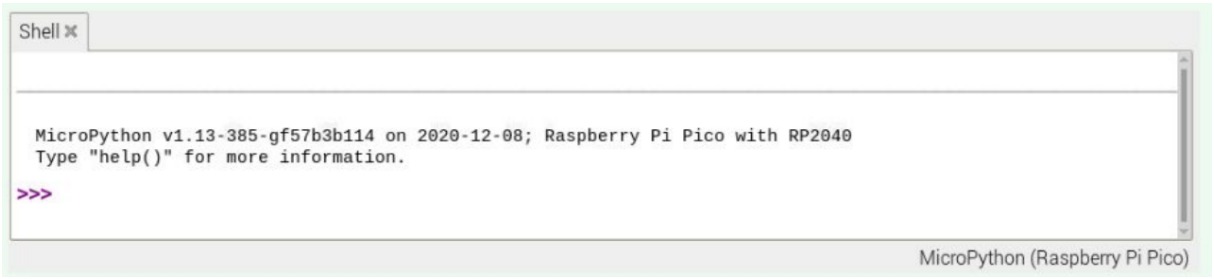


FIGURE 5 – Shell Thonny

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.

```
import math  # Importe toute la librairie math
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_librairie.nom_fonction()`. Par exemple :

```
import math

resultat = math.sqrt(16)  # Utilise la fonction sqrt de la
# librairie math pour calculer la racine carree de 16

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 :

```
from math import sqrt  # Importe uniquement la fonction sqrt
                       # du module math

resultat = sqrt(25)
print(resultat)  # Affiche 5.0
```

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

```
import math as m  # Renomme math en m

resultat = m.sqrt(9)
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.

```
from machine import Pin

led = Pin(25, Pin.OUT)  # Cree un objet Pin pour controler la
                        # LED integree

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 chronométrer des événements.

```
import time

time.sleep(2)  # Pause de 2 secondes
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.

```
import utime

start = utime.ticks_ms()  # Obtenir le temps actuel en
                          # millisecondes
utime.sleep_ms(100)       # Pause de 100 millisecondes
end = utime.ticks_ms()

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.

```
import uos

uos.listdir()  # Liste les fichiers et repertoires dans le
               # repertoire 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 :

```
class Voiture:
    def __init__(self, marque, modele, annee):
        self.marque = marque
        self.modele = modele
        self.annee = annee

    def afficher_details(self):
        print(f"Voiture: {self.marque} {self.modele},
        Annee: {self.annee}")
```

- **Déclaration de la classe** : La classe `Voiture` est créée avec trois attributs : `marque`, `modele`, et `annee`.
- **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 :

```
# Creation d'une instance de la classe Voiture
ma_voiture = Voiture("Toyota", "Corolla", 2020)

# Appel de la methode afficher_details pour afficher les
# 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.

```
from machine import Pin

# Configuration d'une broche en sortie
led = Pin(25, Pin.OUT)
led.value(1)  # Allume la LED connectee a la broche 25

# Configuration d'une broche en entree
bouton = Pin(14, Pin.IN, Pin.PULL_DOWN)
etat_bouton = bouton.value()  # Lit l'etat 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 la résistance de pull-down ou de pull-up interne à la broche. Dans le cas d'une broche configurée en entrée, par exemple pour lire la valeur d'un bouton poussoir (LOW si le bouton est relâché et HIGH si le bouton est pressé), activer la résistance de pull-down évite que la broche soit laissée flottante quand le bouton est relâché.

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.

```
from machine import ADC

# Creation d'un objet ADC sur la broche 26 (GP26)
potentiometre = ADC(26)

# Lecture de la valeur analogique (entre 0 et 65535)
valeur = potentiometre.read_u16()
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.

```
from machine import Pin, PWM

# Configuration de la broche 15 en PWM
led_pwm = PWM(Pin(15))

# Reglage de la frequence du PWM
led_pwm.freq(1000)  # 1 kHz

# Reglage du rapport cyclique (duty cycle) entre 0 (eteint) et
# 65535 (100% allume)
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.

```
from machine import Pin, I2C
```

```
# Configuration de l'I2C broches GPIO8 (SDA) et GPIO9 (SCL)
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)

# Scanner les peripheriques I2C connectes
devices = i2c.scan()
print("Peripheriques I2C trouves :", devices)

# Lire et ecrire des donnees sur un peripherique I2C
# Exemple : lire 2 octets a l'adresse 0x3C
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.

```
from machine import Pin, SPI

# Configuration du SPI sur les broches GPIO10 (SCK), GPIO11
# (MOSI), GPIO12 (MISO)
spi = SPI(0, baudrate=1000000, polarity=0, phase=0,
        sck=Pin(10), mosi=Pin(11), miso=Pin(12))

# Ecriture et lecture de donnees SPI
# Exemple : ecrire [0x01, 0x02] et lire 2 octets
spi.write([0x01, 0x02])
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.

```
from machine import UART
```

```
# Configuration du UART sur le port UART 0, avec les broches
# GPIO0 (TX) et GPIO1 (RX)
uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

# Envoyer des donnees
uart.write('Hello, UART!')

# Lire des donnees
data = uart.read(10)  # Lit 10 octets de donnees
```

— Comme pour la classe SPI, le baudrate choisit 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.

```
from machine import Timer

# Creation d'une minuterie qui execute une fonction toute
# les 2 secondes
def clignotement(t):
    print("Timer declenche !")

timer = Timer()
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.

```
# Ouvrir (ou creer) un fichier nomme "donnees.txt" en mode
# ecriture
with open("donnees.txt", "w") as fichier:
    # Ecrire des donnees dans le fichier
    fichier.write("Bonjour, Raspberry Pi Pico !\n")
    fichier.write("Ceci est un exemple d'ecriture dans un
    fichier.\n")

# Le fichier est automatiquement ferme a la fin du bloc with
```

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

```
# Ouvrir le fichier en mode ajout
with open("donnees.txt", "a") as fichier:
    # Ajouter des donnees au fichier
    fichier.write("Ajout de nouvelles donnees.\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.

```
# Ouvrir un fichier en mode ecriture
fichier = open("donnees.txt", "w")

# Ecrire des donnees dans le fichier
fichier.write("Ecriture de donnees importantes.\n")

# Forcer l'enregistrement des donnees sur le disque
fichier.flush()

# Fermer le fichier
fichier.close()
```

Ecriture de données sur une carte SD

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 `sdcard` 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. La librairie `sdcard` ainsi qu'un exemple se trouvent dans les ressources du cours.

Au secours j'ai supprimé les données de ma carte !

Si vous avez par mégarde effacé les données de vol de votre carte, par exemple en ayant supprimé le fichier avec Thonny ou bien si vous avez ouvert le fichier avec le programme Python en mode write et pas append, il est toujours possible de récupérer les données. En effet, comme sur un ordinateur, lorsqu'un fichier est supprimé de la Pico, les données du fichier sont encore présentes en mémoire, seul l'index du fichier est supprimé. Si la carte n'a pas été réutilisé après la suppression des données, il y a de grandes chances que celles-ci n'aient pas été écrasées par d'autres données et soient récupérables.
Pour ce faire, il faut un ordinateur sur Linux (ou une simple machine virtuelle comme Lubuntu) et une seconde Raspberry Pi Pico qui va servir de debugger, ainsi que quelques câbles pour brancher la Pico dont on veut récupérer les données au debugger et le fichier picoprobe.uf2.
Pour configurer le debugger, appuyer sur le bouton BOOTSEL de la Pico qui sert de debugger et la brancher à l'ordinateur en maintenant le bouton appuyé. Relâcher le bouton. Un explorateur de fichiers pour la Pico s'ouvre, y glisser le fichier picoprobe.uf2. La Pico est maintenant configurée en debugger.
Pour récupérer les données avec Linux, ouvrir un terminal et installer openocd avec la commande suivante.

```
sudo ap install openocd
```

Se placer dans le répertoire où vous souhaitez récupérer la mémoire de la carte à debugger. Si besoin, créer un répertoire avec la commande mkdir, puis y aller avec la commande cd. Brancher le debugger à la carte à debugger comme suit.

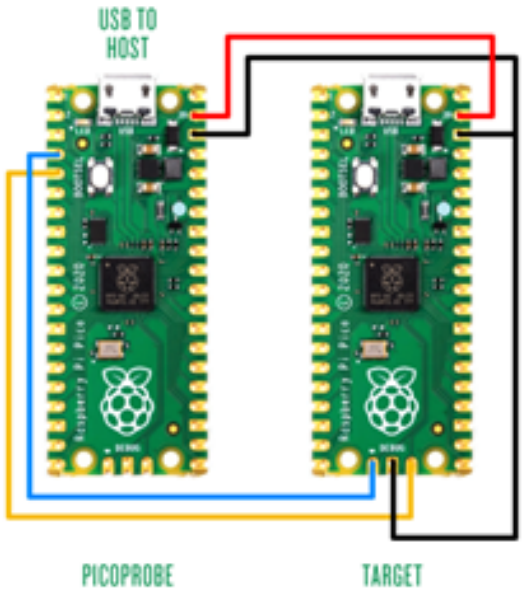


FIGURE 6 – Branchement du debugger à la carte à debugger

Tout en maintenant bien les cartes branchées sans bouger, éxécuter la commande suivante.

```
sudo openocd -f interface/cmsis-dap.cfg -f target/
rp2040.cfg -c "adapter speed 5000" -c "init" -c
"halt" -c "flash read_bank 0 dump.bin" -c "exit"
```

La mémoire de la Raspberry à debugger est copiée dans le fichier dump.bin. Quitter le terminal et changer renommer le fichier dump.bin en dump.txt. Plus qu'à repérer les données recherchées dans le fichier dump.txt !