



Electronique pour les systèmes embarqués

## TP Raspberry PI pico

### 1 Introduction

Dans ce mini-TP, nous nous intéressons à la prise en main de la maquette Raspberry Pi Pico et plus particulièrement de son convertisseur analogique numérique (CAN). La Raspberry Pi Pico est une carte programmable construite autour du microcontrôleur RP2040. La carte intègre un ARM Cortex-M0+ Dual Core à 133 MHz. Elle dispose de 26 broches GPIO dont 3 entrées analogiques connectées à un CAN. La datasheet complète de la carte est disponible sur le lien suivant : <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf>.

**Question 1.1** *Parcourir la rapidement, notamment le chapitre 1 pour connaître les principales fonctionnalités de la carte.*

### 2 Programmation de la carte

La carte Raspberry Pi Pico se programme avec plusieurs langages. Nous vous invitons à utiliser dans le cadre de ce module le langage micropython qui permet de l'interactivité et une facilité d'utilisation. Vous pourrez bien évidemment si vous le souhaitez utiliser d'autres langages supportés par la carte notamment le C/C++. Dans la suite de ce mini-TP, nous utiliserons MicroPython <https://www.raspberrypi.com/documentation/microcontrollers/micropython.html>.

Pour programmer en MicroPython, il est nécessaire de l'installer sur la carte. Cette opération très simple décrite dans le lien ci-dessus, a été déjà réalisée sur les Raspberry Pi Pico utilisées dans ce TP. Une fois MicroPython installé, il est possible d'utiliser des environnements de développement comme Thonny <https://thonny.org/> ou d'interagir avec la carte avec Pyboard.py.

C'est un utilitaire qui permet d'interagir avec micropython à partir d'un interpréteur ou d'un code Python tournant sur le poste principal et permet notamment accéder au système de fichier de la carte pico. Nous utiliserons cette deuxième solution dans le cadre de ce mini-TP. Cependant l'utilisation de Thonny a également des intérêts en termes d'interactivité. N'hésitez pas à l'installer et l'utiliser pour le projet fil-rouge du module.

Sachez qu'il est possible d'avoir différents codes python sur la carte mais le code qui s'exécutera lors du démarrage est le code `main.py` qui bien évidemment peut faire appel à d'autres codes python sur la carte.

Pour tester la carte, nous allons, bien évidemment, faire clignoter une LED (une LED sous forme d'écran). What else!

```
from machine import Pin, Timer
led = Pin(25, Pin.OUT)
timer = Timer()

def blink(timer):
    led.toggle()

timer.init(freq=2.5, mode=Timer.PERIODIC, callback=blink)
```

Charger le code sur la carte en utilisant la commande

```
./pyboard.py -device /dev/ttyACM0 blink.py
```

Il est également possible de rendre le code par défaut à l'aide de la commande

```
./pyboard.py -device /dev/ttyACM0 -f cp ./blink.py :main.py
```

**Question 2.1** *Changer la valeur de la fréquence dans `blink.py` et vérifier que la fonctionnalité de la carte.*

### 3 Caractérisation du convertisseur

Nous allons à présent nous concentrer sur le CAN. La puce RP2040 contient un CAN à approximations successives, SAR en anglais. L'architecture du CAN ou ADC en anglais, est détaillé dans la datasheet de RP2040 <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#page=561>.

Le CAN a une dynamique d'entrée allant de 0 à 3.3 V et a une fréquence d'opération maximale de 48 MHz. Nous allons le tester avec une fréquence d'échantillonnage de 1 kHz. On appliquera un signal d'entrée de fréquence 101 Hz, d'amplitude 1 V<sub>peak</sub> ou 2 V<sub>peak-to-peak</sub> et un offset de 1.65 V pour se centrer au milieu de la dynamique d'entrée.

Nous allons tester le CAN avec 2 approches différentes. La première qu'on appellera l'approche *sleep* est décrite par le code suivant :

```
from machine import Pin, ADC
import utime
adc = ADC(Pin(26, mode=Pin.IN))

while True:
    start=utime.ticks_us()
```

```
print(adc.read_u16())
stop=utime.ticks_us()
utime.sleep_us(1000-stop+start)
```

La deuxième qu'on appellera l'approche *interruption* est décrite par le code suivant :

```
from machine import Timer, Pin, ADC

def interruption_handler(pin):
    print(adc.read_u16())

if __name__ == "__main__":
    adc = ADC(Pin(26, mode=Pin.IN))
    soft_timer = Timer(mode=Timer.PERIODIC, period=1,
        callback=interruption_handler)
```

Charger le code sur la carte en utilisant la commande :

```
./pyboard.py -device /dev/ttyACM0 ADC_sleep.py1
```

Nous utiliserons Matlab pour faire l'acquisition des données envoyées sur le port série et pour faire le post traitement (tracé de la FFT et calcul du SNR). Configurer l'entrée sur le logiciel *waveforms* et connecter la à la GPIO 26 (pince rouge sur la GPIO , pince noir à la masse).

**Question 3.1** Lancer le code script *rasberryPIpico.m*. Analyser le spectre. Quelle est la fréquence de la raie du signal ? Relever la valeur du SNDR.

Nous allons comparer à présent le résultat de l'acquisition à celle obtenue avec l'approche *interruption* Charger le code sur la carte en utilisant la commande

```
./pyboard.py -device /dev/ttyACM0 ADC_interrupt.py
```

**Question 3.2** Lancer le code script *rasberryPIpico.m*. Analyser le spectre et relever la valeur du SNDR. Comparer les 2 approches.

## 4 Affichage sur l'écran

Le raspberry PI pico intègre un écran LCD. Le fichier *ST7735.py* contient la bibliothèque nécessaire pour piloter l'écran. Copier la bibliothèque sur la Raspberry en utilisant la commande suivante :

```
./pyboard.py -device /dev/ttyACM0 -f cp ./ST7735.py :
```

Le script *LCDexample.py* contient un exemple d'utilisation de l'écran. Charger le sur la carte en utilisant la commande

```
./pyboard.py -device /dev/ttyACM0 LCDexample.py
```

---

1. Vous pouvez arreter l'affichage sur le terminal en appuyant sur Ctrl+c

Modifier le pour qu'il fasse un affichage de votre choix. Par exemple, vous pouvez changer le texte affiché, les couleurs, dessiner des dents de scie, afficher la sortie du convertisseur en équivalent analogique, ...

## 5 Acquisition d'un signal cardiaque

Pour pouvoir avancer sur la conception de l'algorithme de calcul du rythme cardiaque parallèle de la conception de la carte d'acquisition du capteur, nous allons utiliser un signal cardiaque pré-enregistré. Sur Waveforms, commencer par désactiver l'entrée. Puis appuyer sur la molette à droite de type puis `import to play` → `cardiac.txt`. Fixer la fréquence ou le nombre d'échantillons par seconde à 90 Hz, l'amplitude à 1 V et l'offset à 1.65 V.

**Question 5.1** *Relancer le code `ADC_interrupt.py` pour relancer l'acquisition. Après avoir patienter une dizaine de secondes afin que la séquence se charge sur la carte Digilent, lancer le code script `rasberryPIpico.m`. Analyser la sortie temporellement et fréquemment.*

**Question 5.2** *Proposer et implémenter un algorithme pour mesurer le rythme cardiaque. (Mais pour ça, vous avez jusqu'à fin juin).*