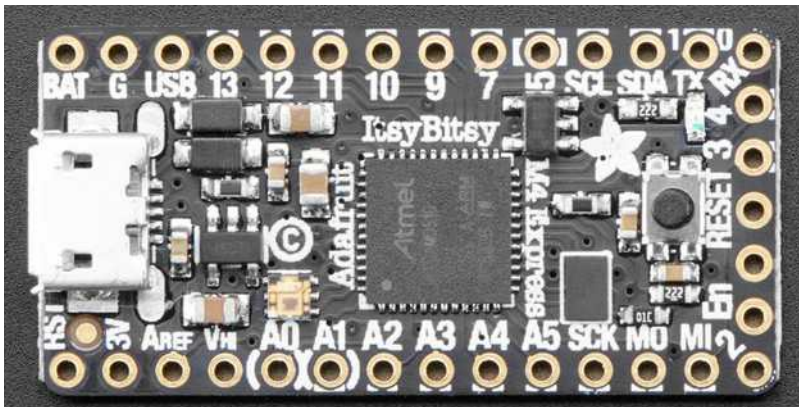
Cours sur les actionneurs

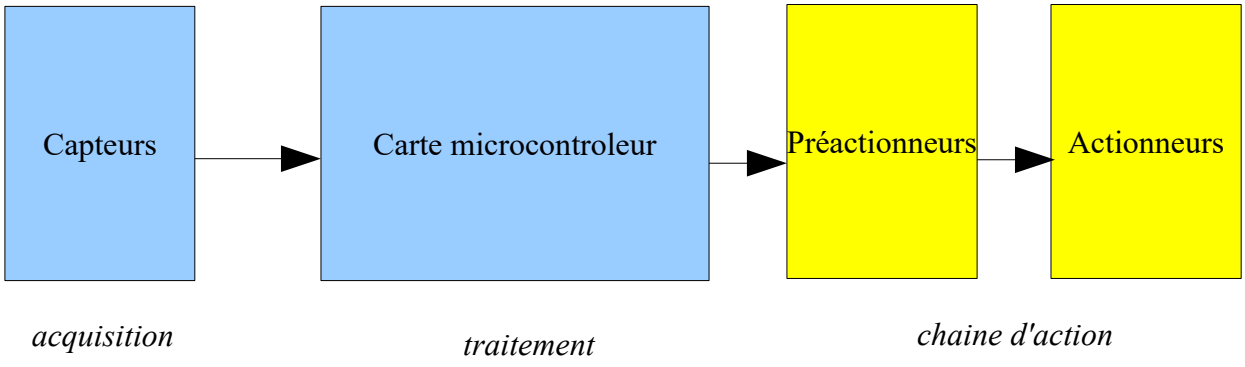
Christophe LIGERET et Benjamin MOUSSET
Juillet 2019

Introduction

Pour illustrer la séquence sur le pilotage d'actionneurs, nous allons utiliser un microtrôlleur ATSAMD51 : par exemple l'ItsyBitsy d'Adafruit : https://www.adafruit.com/product/3800



Cette carte électronique coûte environ 15 $ est contruite autour du microcontrôleur Microchip ATSAMD51 Cortex M4.
Cette carte se programme aussi bien en C avec Arduino qu'en Python avec Mu Editor par exemple.
Le processeur tourne à 120 MHz (il y a mieux : pour le même prix, une carte TeenSy 4.0 doté du microcontrolleur ARM Cortex-M7 fonctionne à 600 MHz et possède 1024K de RAM) ; possède une mémoire RAM de 192 KB et une mémoire Flask de 512 KB. Cette carte a 23 entrées / sorties (0 – 3,3V) dont 7 entrées analogiques ; 2 sorties analogiques et 18 configurables en mode PWM ; une sortie « haute tension » 0 – 5V pour piloter des leds Neopixel par exemple.
Ce microcontroleur est alimenté par une tension VCC = 3,3V (d'autres circuits fonctionnent avec du 5V : Arduino...). Dans ce document, on prendra Vcc = 3,3V.



Pilotage des actionneurs

Certains actionneurs (lampes ; résistances chauffantes ; moteur électrique...) se pilotent très simplement en « tout ou rien » c'est à dire qu'on peut les associer à une variable binaire qui ne prend que 2 états : une lampe sera allumée ou éteinte ; une résistance et un moteur alimentés ou hors tension.

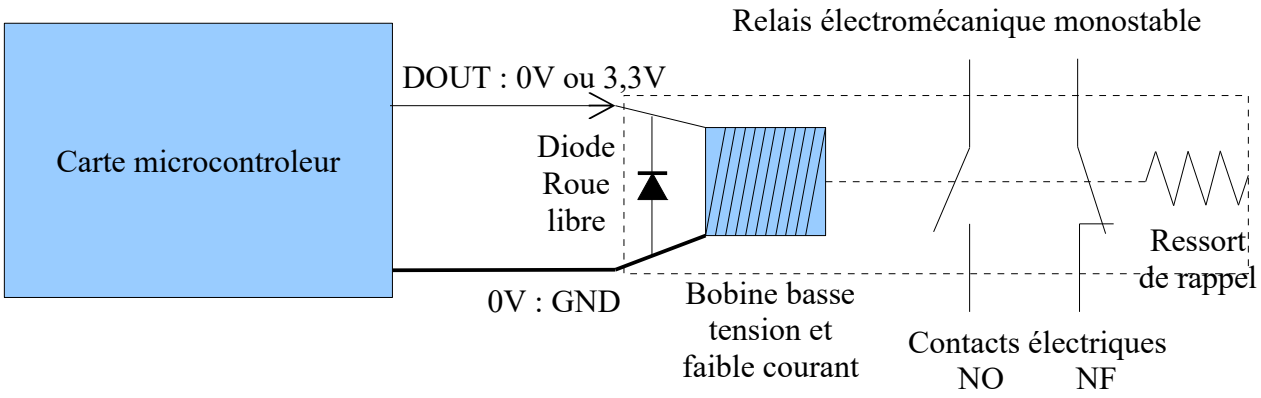
D'un point de vue technique, on ne peut pas brancher directement ces actionneurs sur une sortie de la carte électronique car les signaux électriques délivrés pas le microprocesseur sont très faibles (courants faibles) : courant maximums débités de quelques milliampères, alors que ces actionneurs peuvent nécessiter jusqu'à plusieurs dizaines d'ampères !

Pour cela, il faut placer entre le microcontrôleur et l'actionneur un amplificateur de signaux appelé préactionneur.

I / Actionneurs tout ou rien

Pour « amplifier » les signaux binaires : 0V ou Vcc (0V ou 3,3V) de la carte électronique, on peut utiliser comme préactionneur un relais : ça peut être un relais statique à base de transistors ou un relais électromécanique.

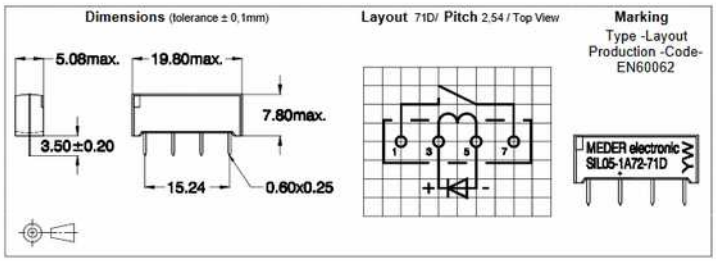
Exemple avec un relais électromécanique monostable à 2 contacts : 1 NO et 1 NF



Remarque : sur une telle installation, il faut s'assurer que la bobine du relais est équipée d'une diode de roue libre sinon il y a un risque de surtension lorsque le microcontrolleur passera la sortie DOUT de 3,3V → 0V, ce qui risque de la détruire (voire plus...). S'il n'y a pas de diode de roue libre, il faudra penser à la rajouter.
En effet, lorsque la bobine du relais passe de l'état alimentée (3,3V) à l'état non alimentée (0V), ella va générer une surtension (u = L di / dt). Cette surtention est redoutable car elle va se retrouver directement sur la patte DOUT du microcontrolleur et risquera de la détruire, voire de détruire le microcontrolleur (avec un effet retard parfois...).

Il faudra également veiller à ce que la bobine du relais soit compatible avec la sortie du microcontrôleur :

Ci-contre : documentation techique d'un relais à bobine « basse puissance » : relais REED référence SIL05-1A72-71D
On remarque la présence d'une diode de roue libre intégrée.
Par contre, la tension maximale d'activation du relais est de 3,5V, ce qui est un peu juste par rapport au 3,3V de la carte ItsyBitsy : il y a un risque que le contact ne se ferme pas même si DOUT = 3,3V : ce relais serait plutôt recommandé pour une sortie tout ou rien 0V ou 5V.
Il faut aussi vérifier le courant absorbé par la bobine : à 20°C, elle a une résistance de 450 Ohms minimum, ce qui fait un courant max de 7,4 mA (acceptable avec le microcontrolleur ATSAMD51)

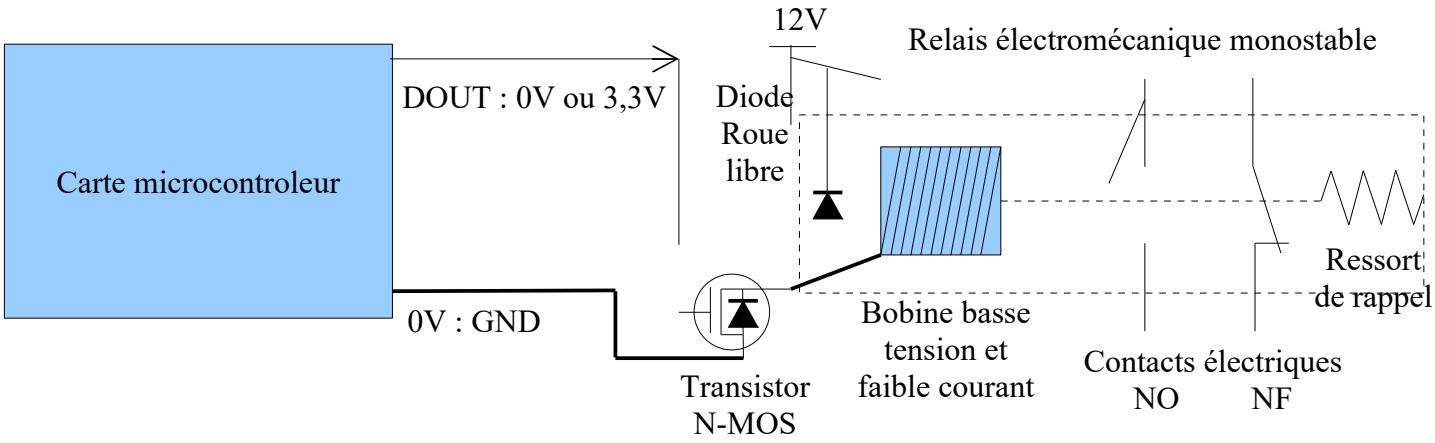


Coil/Relay Characteristics	Conditions at 20°C	Min.	Typ.	Max.	Units
Coil Resistance		450	500	550	Ω
Nominal Voltage			5,0		VDC
Nominal Rated Power			50		mW
Thermal Resistance			109		K / W
Operate Voltage				3,5	VDC
Release Voltage		0,75			VDC

Contact Data 72 (Form A/Dry)					
Contact Rating	Any combination of the switching voltage and current must not exceed the given rated power			15	W
Switching Voltage	DC or Peak AC			200	V
Switching Current	DC or Peak AC			1,0	A
Carry Current	DC or Peak AC			1,25	A
Static Contact Resistance (initial)	Measured with Nominal Voltage			150	mΩ
Insulation Resistance	RH 45%	10^{10}			Ω
Breakdown Voltage		250			VDC
Operate Time, including Bounce	Measured with Nominal Voltage			0,5	ms
Release Time	Measured with no coil suppression			0,1	ms
Capacitance			0,3		pF

Les données relatives au contact permettent de vérifier si on peut utiliser l'actionneur : le courant maximum est de 1A sur le contact la tension max est de 250V : aussi, une lampe fonctionnant sur du 12V / 0,5A sera parfaitement acceptable. Par contre, un moteur 6V / 3A ne conviendra pas.

Bien souvent, la bobine du relais fonctionne avec des tensions un peu plus élevées (6V ; 12V ; 24V) et necessite une électronique d'interface comme suit :



On peut rajouter un transistor mosfet canal N (par exemple un 2N7000) entre la sortie du microcontrolleur et la bobine : cet transistor permet d'amplifier les signaux du microcontrolleur et ainsi de pouvoir piloter des relais « plus puissants ».

Remarque : il est recommandé de placer au plus près de la bobine et du transistor N-MOS un condensateur de découplage (environ 100nF) entre la masse (GND) et le 12V pour réduire les surtensions destructrices apparaissant lorsque DOUT passe de 3,3V → 0V.

Ci-contre, la documentation technique du relais série VF4 40A / 12V : on remarque que la bobine se pilote avec une tension nominale de 12V (donc le transistor N-MOS sera indispensable car le microcontrolleur ATSAMD51 ne peut pas délivrer une tension si élevée).

ELECTRICAL CHARACTERISTICS: (ALL DATA APPLIES @ 23°C UNLESS OTHERWISE SPECIFIED)

COIL DATA:

NOMINAL VOLTAGE:	12 VDC
OPERATE VOLTAGE:	7.8 VDC MAXIMUM
RELEASE VOLTAGE:	1.2 VDC MINIMUM
COIL RESISTANCE:	90 OHMS +/- 10%
OPERATE TIME:	8 mSEC. MAXIMUM EXCLUDING BOUNCE
RELEASE TIME:	5 mSEC. MAXIMUM EXCLUDING BOUNCE
TEMPERATURE RANGE:	OPERATING -40°C TO +85°C

CONTACT DATA: (CONTACT DATA IS FORMATTED N.O./N.C.)

CONTACT ARRANGEMENT:	1 FORM C (SPDT)
CONTACT MATERIAL:	AgSnO (SILVER TIN-OXIDE)
CONTACT MILLIVOLT DROP:	200mv @ 35A ON N.O. CONTACTS (AFTER SWITCHING) 250mv @ 20A ON N.C. CONTACTS (AFTER SWITCHING)
MAXIMUM MAKE CURRENT:	90A/30A (LAMP) @ 16 VDC
MAXIMUM BREAK CURRENT:	40A/30A @ 16 VDC RESISTIVE
MAXIMUM CONTINUOUS CURRENT:	40A/30A @ 23°C , 35A/20A @ 85°C
INITIAL BREAKDOWN CURRENT	500V RMS CONTACTS TO COIL
EXPECTED LIFE:	100,000 OPERATIONS, 40 A, 14 VDC RESISTIVE ON NORMALLY OPEN CONTACT

MECHANICAL CHARACTERISTICS:

EXPECTED LIFE:	10 MILLION OPERATIONS, NO CONTACT LOAD
TERMINALS	BRASS, UNPLATED

https://learn.adafruit.com/circuitpython-essentials/circuitpython-digital-in-out


```
#Exemple de programme : clignoteur 1s
import time
import board
from digitalio import DigitalInOut, Direction
relay = DigitalInOut(board.D13) # la bobine du relais est câblée sur la patte 13
relay.direction = Direction.OUTPUT # La patte 13 est confinurée en sortie TOR
relay.value = True # On active la patte 13 : 3,3V → le relais s'active
time.sleep(1.0) # On attend 1s
relay.value = False # On désactive la patte 13 : 0V → le relais passe au repos
time.sleep(1.0) # On attend 1s
```

Exercices autour du pilotage d'un actionneur tout ou rien (TOR)

Exercice 1 : lire attentivement ce programme ci-dessous

```
import board
from digitalio import DigitalInOut, Direction, Pull
relay = DigitalInOut(board.D13) # la bobine du relais est câblée sur la patte 13
relay.direction = Direction.OUTPUT # la patte 13 est confinurée en sortie TOR
button = DigitalInOut(board.D2) # le bouton poussoir est cablé sur la patte 2
button.direction = Direction.INPUT # le bouton poussoir est une entrée TOR
button.pull = Pull.UP # le bouton est une entrée TOR PULLUP : 3,3V au repos

while True:
  if button.value == True: # le bouton poussoir est au repos (3,3V)
    relay.value = False # On désactive la bobine du relais
  else: # le bouton poussoir est au activé (0V)
    relay.value = True # On sactive la bobine du relais
```

1°) Que fait exactement le programme ci-dessus ?

2°) On remplace maintenant le code dans la boucle while par :

```
import time
while True:
  if button.value == True: # le bouton poussoir est au repos (3,3V)
    time.sleep(5.0)
    relay.value = False # On désactive la bobine du relais
  else: # le bouton poussoir est au activé (0V)
    relay.value = True # On active la bobine du relais
```

Que fait ce nouveau programme ? A quelle application de la vie de tous les jours pourrait-il être utile ?

Exercice 2 : lire attentivement ce programme ci-dessous

```
import board
from digitalio import DigitalInOut, Direction, Pull
relay = DigitalInOut(board.D13) # la bobine du relais est câblée sur la patte 13
relay.direction = Direction.OUTPUT # la patte 13 est confinurée en sortie TOR
button1 = DigitalInOut(board.D1) # le bouton poussoir 1 est cablé sur la patte 1
button1.direction = Direction.INPUT # le bouton poussoir 1 est une entrée TOR
button1.pull = Pull.UP # le bouton 1 est une entrée TOR PULLUP : 3,3V au repos
button2 = DigitalInOut(board.D2) # le bouton poussoir 2 est cablé sur la patte 2
button2.direction = Direction.INPUT # le bouton poussoir 2 est une entrée TOR
button2.pull = Pull.UP # le bouton 2 est une entrée TOR PULLUP : 3,3V au repos

while True:
  if button1.value == False: # le bouton poussoir 1 est activé (0V)
    relay.value = True # On active la bobine du relais
  if button2.value == False: # le bouton poussoir 2 est activé (0V)
    relay.value = False # On déactive la bobine du relais
```

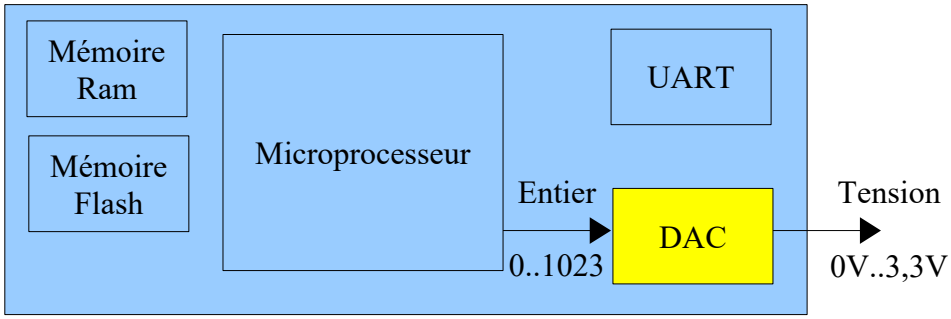
1°) Que fait exactement le programme ci-dessus ?

2° Faire une recherche sur Internet sur les bascules SET / RESET.

Exercice 3 : A partir des deux exercices-ci dessus, développer un programme en langage Python (puis implémentez et testez-le sur une carte microcontrolleur) qui réponde au cachier des charges ci-dessous
- Il y a deux boutons poussoirs : BP1 et BP2 et un relais : R.
- Lorsqu'on appuie brievement sur BP1, le relais se met à clignoter avec une période de 1s
- Lorsqu'on appuie brievement sur BP2, le relais passe au repos

II / Actionneurs analogique

Un DAC pour Digatal to Analog Converter est un convertisseur numérique analogique qui permet de convertir un nombre entier (généralement entre 0 et 1023) en une tension électrique (entre 0V et 3,3V).
On supposera que la tension électrique en sortie du convertisseur numérique analogique est linéaire en fonction de nombre en entrée



microcontrôleur

Cette sortie analogique est très souvent utilisée pour piloter des variateurs de vitesse de moteurs dans l'industrie (par exemple, 0V correspond à l'arrêt et 3,3V à la vitesse maximale) mais aussi plus généralement pour commander des hauts parleurs.

<u>Exercice 1 :</u>
On souhaite piloter le moteur électrique d'un petit véhicule électrique
(kart électrique ou bicyclette électrique) au moyen d'un variateur de vitesse
commandé par un signal analogique 0V – 3,3V : uVV

Exemple de variateur de vitesse piloté par un signal analogique Entre 0V et 3,3V

Les spécifications du variateur de vitesse sont les suivantes :
- lorsque la tension uVV est comprise entre 0,8V et 1V, le moteur est en roue libre
- lorsque la tension uVV est comprise entre 1V et 3V, le variateur applique un couple positif au moteur (le moteur travaille en mode « moteur » et la batterie se vide...)
1V : couple de 0 N.m ; 3V : couple moteur maximum
- lorsque la tension uVV est comprise entre 0V et 0,8V, le variateur applique un couple négatif au moteur (frein moteur et récupération d'une partie de l'énergie dans la batterie).
0,8V : couple de 0 N.m ; 0V : couple de freinage moteur maximum

On propose le programme ci-dessous

```
import board
from analogio import AnalogOut, AnalogIn
accelerateur = analog_in = AnalogIn(board.A1) # valeurs entre 0 et 1023
frein = analog_in = AnalogIn(board.A2) # valeurs comprises entre 0 et 1023
uVVN = AnalogOut(board.A0) # valeurs comprises entre 0 et 1023

while true:
  if frein > 0:
    uVVN.value = int( -0.242 * float(frein.value) + 248.0 )
  else:
    uVVN.value = int( 0.605 * float(accelerateur.value) + 310 )
```

1°) Commentez ce programme, notamment le test `if` et les formules :
`-0.242 * float(frein.value) + 248.0`
et `0.605 * float(accelerateur.value) + 310`

2°) On change de variateur de vitesse et il a pour caractéristiques :
commande frein moteur entre 0V et 0,9V ; commande moteur entre 1,1V et 3,3V

Modifier les deux lignes de code correspondant à ces nouvelles caractéristiques

Sources : https://learn.adafruit.com/circuitpython-essentials/circuitpython-analog-in
https://learn.adafruit.com/circuitpython-essentials/circuitpython-analog-out

Exercice 2 :

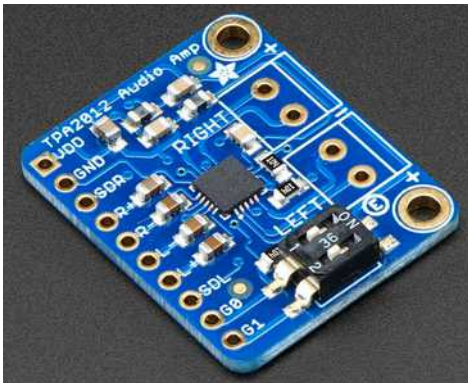
On souhaite jouer un morceau de musique avec un microcontrolleur et Python.
On propose le code ci-dessous :

```
import time
import array
import math
import board
import digitalio

try:
  from audioio import AudioOut
except ImportError:
  try:
   from audiopwmio import PWMAudioOut as AudioOut
  except ImportError:        pass  # not always supported by every board!

wave_file = open("StreetChicken.wav", "rb")
wave = WaveFile(wave_file)
audio = AudioOut(board.A0)
audio.play(wave)
```

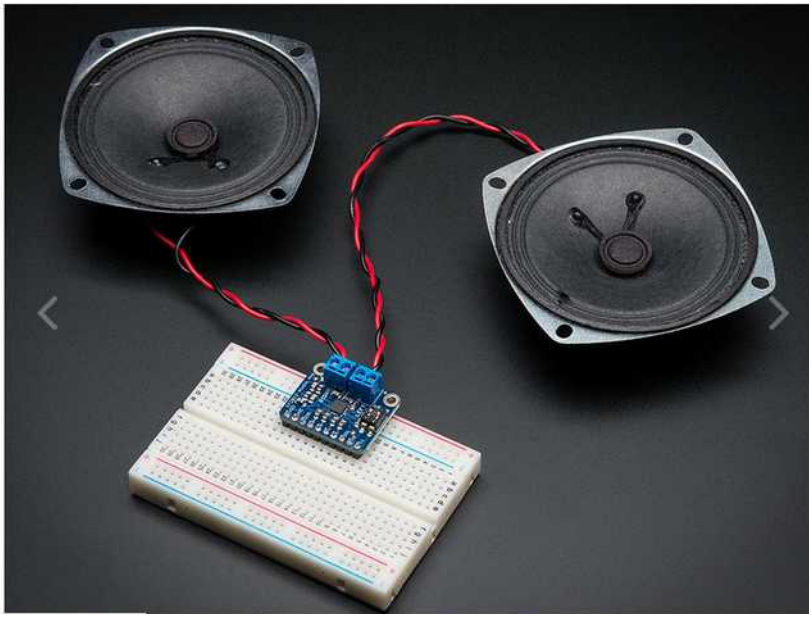
1°) Comment s'appelle le fichier audio qui est lu par le microcontroleur ? Quel est le type de format ? Où ce fichier est-il physiquement situé ?

2°) Sur quelle patte du microcontrôleur doit on câbler l'amplificateur stéréo ?

3°) Implémentez et testez ce code sur votre montage électronique



Amplificateur audio stéréo



Exemple de câblage de 2 hauts parleurs sur un amplificateur audio stéréo

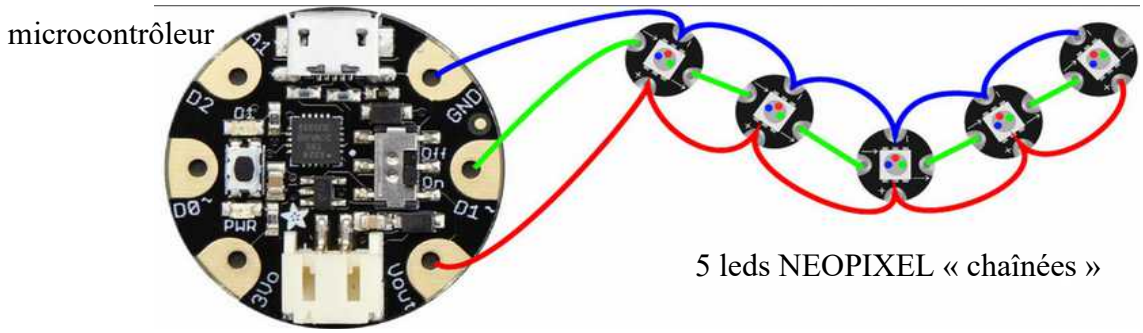
Exercice 3 :

Reprendre le code de l'exercice 2 de manière à ce qu'on puisse faire jouer deux morceaux de musique selon qu'on appuie sur un bouton poussoir BP1 ou sur un autre bouton poussoir : BP2

Sources : https://learn.adafruit.com/circuitpython-essentials/circuitpython-audio-out

III / Actionneurs numériques

Cette partie sera contruite autour des LEDs NEOPIXEL.



Les leds NEOPIXEL sont des leds rouges ; vertes ; bleues intégrant chacune une électronique permettant de piloter chaque composante rouge ; verte ou bleue sur 256 niveaux.
Elles sont commandées par un port série sur un fil : sur ce fil circule une série de 0 et de 1 que chaque led décode ; les leds sont chaînées entre elles : par exemple, lorsqu'une led reçoit un paquet :
START-R1-V1-B1-R2-V2-B2-R3-V3-B3-R4-V4-B4-R5-V5-B5
elle ne va prendre en compte que les composantes R1 ; V1 et B1 et reémettra pour la led suivante :
START-R2-V2-B2-R3-V3-B3-R4-V4-B4-R5-V5-B5
On peut ainsi considérer ces leds chaînées comme des pixels adressables : si on regarde le schéma ci-dessus, la led la plus à gauche (directement connectée au microcontrôleur) a l'adresse 0, puis en allnt vers la droite : adresse 1 ; adresse 2 ; … ; adresse 4
En fait, ces leds se comportent un peu comme des routeurs très simplifiés.

Exercice 1 :

```
# CircuitPython demo - NeoPixel
import time
import board
import neopixel
pixel_pin = board.A1
num_pixels = 5
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3,
auto_write=False)

RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

# On affiche un ruban rouge ; jaune ; vert ; cyan et bleu pendant 5s
color = RED
pixels[0] = color
color = YELLOW
pixels[1] = color
color = GREEN
pixels[2] = color
color = CYAN
pixels[3] = color
color = BLUE
pixels[4] = color
pixels.show()
time.sleep(5.0)
```

1°) Lire attentivement ce programme ; commenter chacune des lignes

2°) Décrire ce que fait ce programme

3°) Implémentez et testez-le sur votre montage électronique

Exercice 2 :

```
import time
import board
import neopixel
pixel_pin = board.A1
num_pixels = 5
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3,
auto_write=False)

RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

while True:
  for i in range(0,5):
    pixels[i] = (0 , 255 , 255) # MAGENTA
  pixels.show()
  time.sleep(1.0)
  for i in range(0,5):
    pixels[i] = (128 , 128 , 128) # YELLOW
  pixels.show()
  time.sleep(1.0)
```

1°) Lire attentivement ce programme ; commenter chacune des lignes

2°) Décrire ce que fait ce programme

3°) Implémentez et testez-le sur votre montage électronique

Exercice 3 :

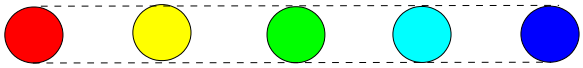
1°) Développez en langage Python un programme qui réalise un chenillard cyclique du type :
0 → 1 → 2 → 3 → 4 → 0 → 1 → 2 → 3 → 4 → 0 → 1 → … et ainsi de suite...

2°) Développez en langage Python un programme qui réalise un chenillard du type :
0 → 4 → 3 → 2 → 1 → 0 → 4 → 3 → 2 → 1 → 0 → 4 → … et ainsi de suite...

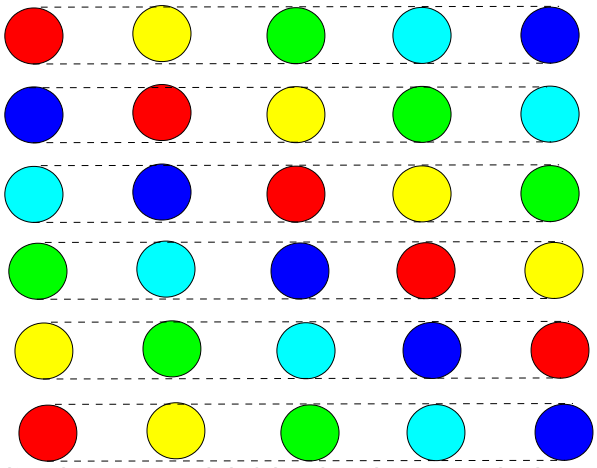
2°) Développez en langage Python un programme qui réalise un chenillard du type :
0 → 1 → 2 → 3 → 4 → 3 → 2 → 1 → 0 → 1 → 2 → 3 → 4 → 3 → 2 → 1 → 0 → 1 → … et ainsi de suite...

Exercice 4 :

1°) Développez en langage Python un programme qui affiche les couleurs du spectre sur ces 5 leds :



2°) Développez en langage Python un programme qui fait défiler les couleurs du spectre sur ces 5 leds :

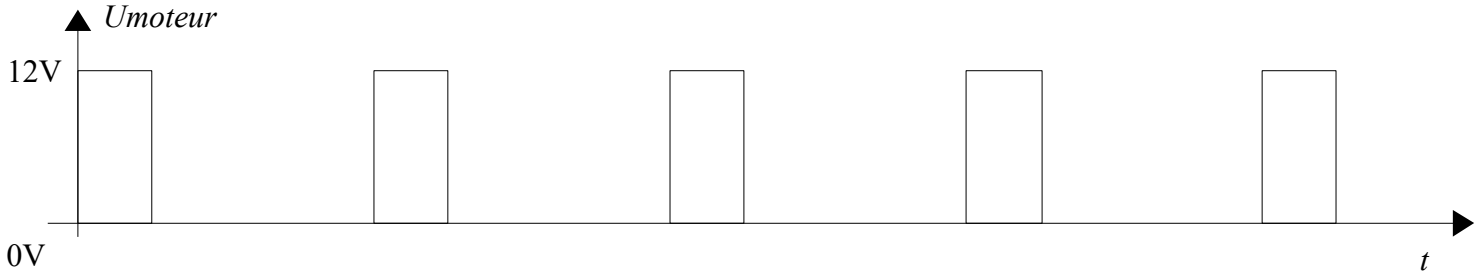


Sources :

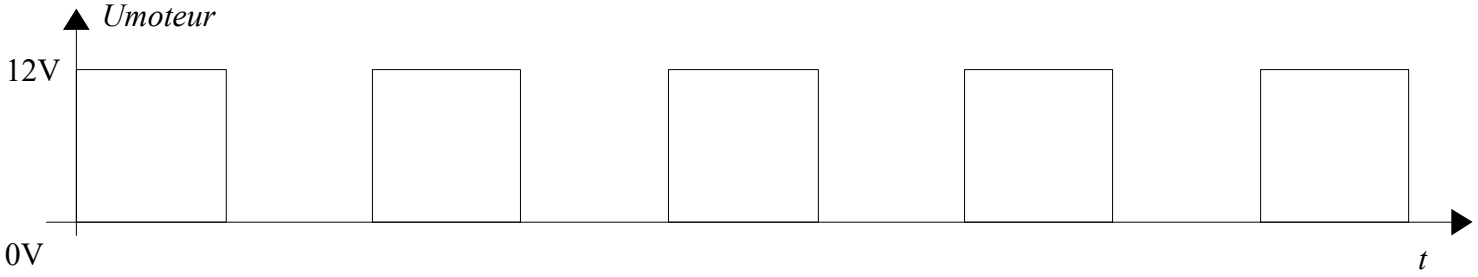
https://learn.adafruit.com/circuitpython-essentials/circuitpython-neopixel

IV / Commande en Modulation de Largeur d'Impulsion d'un actionneur

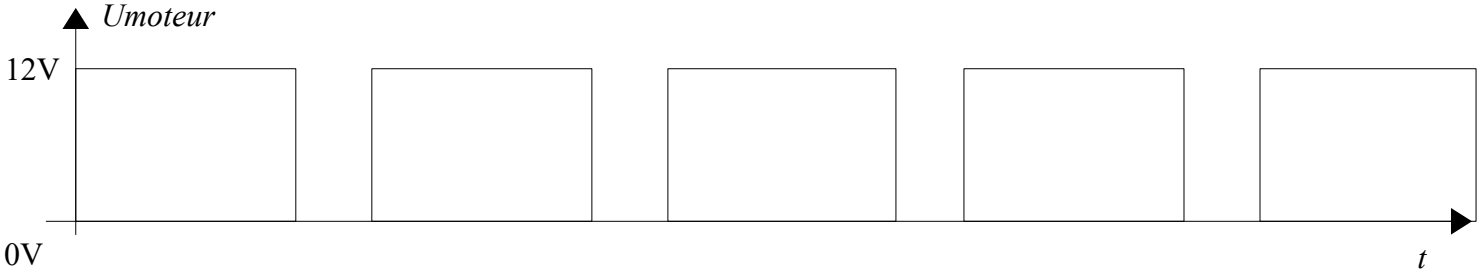
La plupart des moteurs électriques à courant continu sont pilotés par une alimentation à découpage : le principe est d'alimenter l'actionneur alternativement avec du 0V et une tension positive fixe (par exemple du 12V)



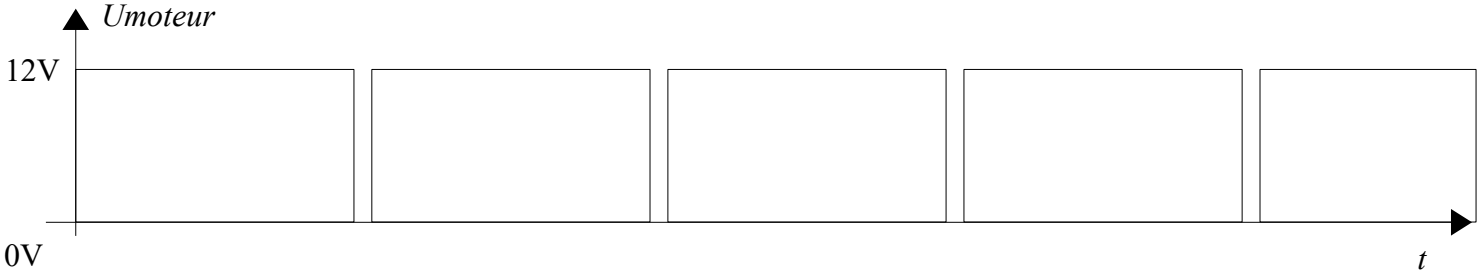
Ci-dessus : rapport cyclique d'environ 25 %, ce qui correspond à une tension moyenne de 3V



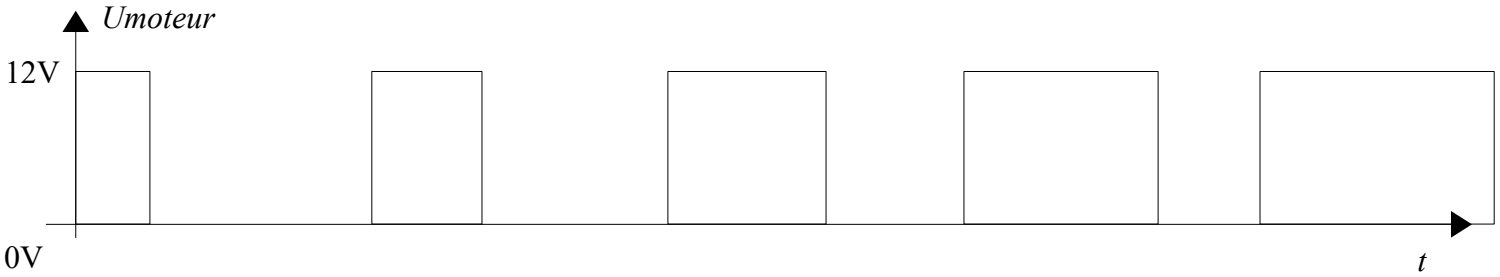
Ci-dessus : rapport cyclique d'environ 50 %, ce qui correspond à une tension moyenne de 6V



Ci-dessus : rapport cyclique d'environ 75 %, ce qui correspond à une tension moyenne de 9V



Ci-dessus : rapport cyclique d'environ 95 %, ce qui correspond à une tension moyenne de 11,4V



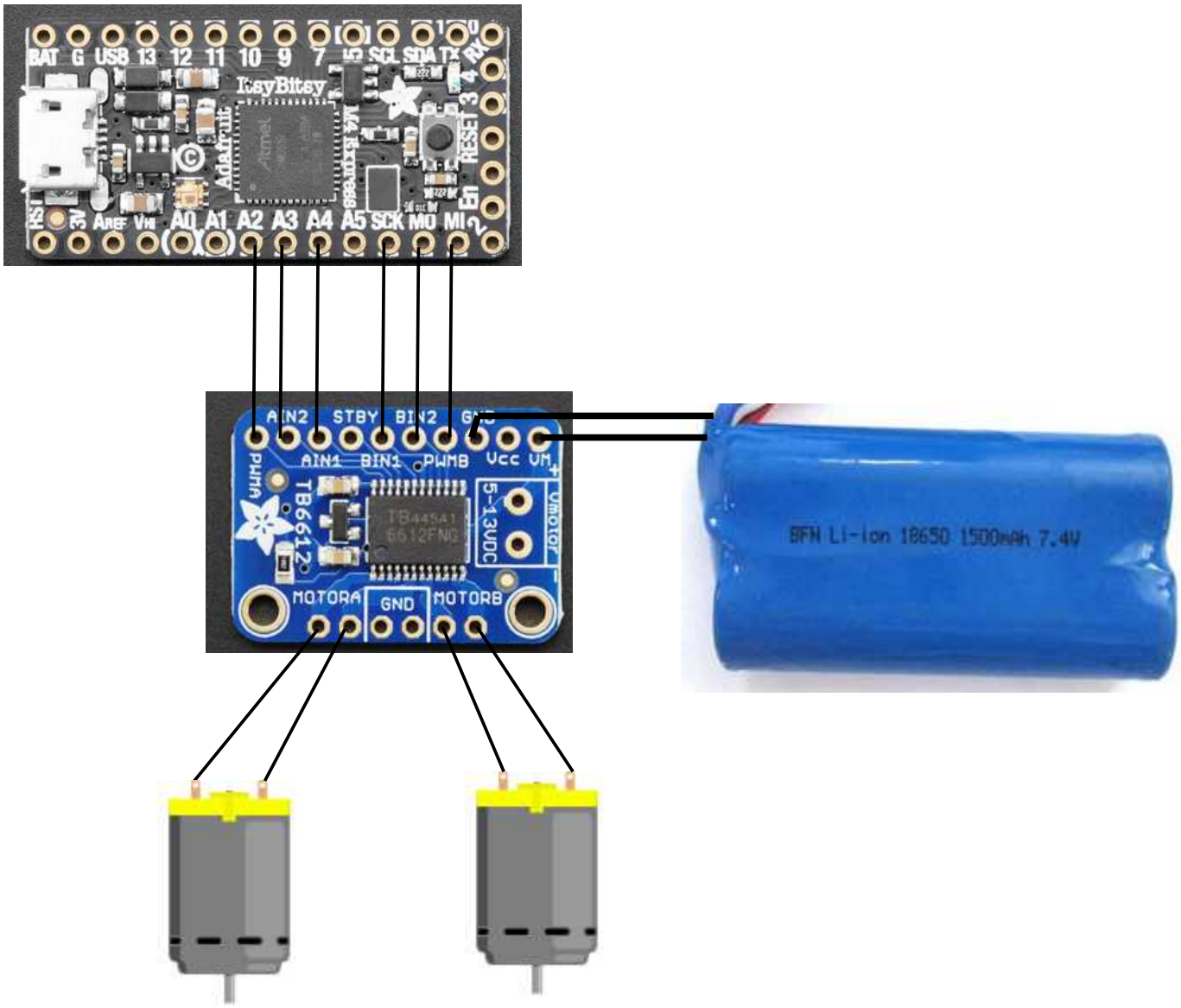
Ci-dessus : moteur électrique en phase d'accélération

Le microntrôleur va générer automatiquement ces signaux tout ou rien « modulés dans le temps ».
On dit que ce signal est du type MLI (modulé en largeur d'implusions) ou PWM (Pulse Width Modulation)
Il est nécessaire de lui configurer la fréquence du signal PWM ainsi que le rapport cyclique (pourcentage du temps pendant lequel le signal est actif sur une période).

Ce type de signal PWM généré par le microntrôleur varie entre 0V et 3,3V et est du type « courant faible » : il faut lui associer un amplificateur de puissance (bien souvent basé sur des transistors MOSFETs) pour pouvoir piloter les moteurs.

Il existe des circuits électroniques (TB6612) qui peuvent piloter jusqu'à 2 moteurs dans les 2 sens de marche (marche avant et marche arrière).

Ci-dessous : un exemple de branchements avec une carte microcontrôleur ; un amplificateur TB6612 ; deux moteurs électriques et une batterie.



Le type d'architecture électrique porposé ci-dessus est ce qui est le plus souvent utilisé en robotique : il y a deux moteurs pilotant respectivement la roue gauche et la roue droite.

Lorsque les 2 moteurs tournent à la même vitesse et dans le même sens, le véhicule avant en ligne droite.

Lorsque les 2 moteurs tournent à des vitesses différentes, le véhicule tourne (soit vers la droite, soit vers la gauche, selon la valeur de la vitesse différentielle entre le moteur gauche et le moteur droit).

Execice 1 : On considère le programme ci-dessous

```
import time
import board
import pulseio
# For the M0 boards:
motorRPWM = pulseio.PWMOut(board.A2,duty_cycle=0,frequency=500,variable_frequency=False)
# For the M4 boards:
# motorRPWM=pulseio.PWMOut(board.A1,duty_cycle=0,frequency=500,variable_frequency=False)
f = 400 # ..................
while True:   # ......................
  motorRPWM.duty_cycle = int(0.25 * 65535.0)  # On a un rapport cyclique de 25%
  time.sleep(1.0) # ............................
  motorRPWM.duty_cycle = int(0.5 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(0.75 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(1.0 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(0.75 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(0.5 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(0.25 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
  motorRPWM.duty_cycle = int(0.0 * 65535.0)  # On a un rapport cyclique de ...%
  time.sleep(1.0)
```

1°) Compléter les commentaires (pointillés)

2°) Décrire ce que fait ce programme

3°) Implémentez et testez-le sur votre montage électronique

Execice 2 : On considère le programme ci-dessous qui doit permettre de faire un démarrage et un arrêt en rampe.

```
import time
import board
import pulseio
# For the M0 boards:
motorRPWM = pulseio.PWMOut(board.A2,duty_cycle=0,frequency=500,variable_frequency=False)
# For the M4 boards:
# motorRPWM=pulseio.PWMOut(board.A1,duty_cycle=0,frequency=500,variable_frequency=False)
f = 400 # ..................
while True:   # ......................
  for(k in .................)     # On effectue l'accélération
    motorRPWM.duty_cycle = ….........
    time.sleep(0.001)
  time.sleep(5.0)     # Pendant 5 secondes, on est à vitesse maximale
  for(k in .................)
    motorRPWM.duty_cycle = ….........  # On a un rapport cyclique de 25%
    time.sleep(0.001)
  time.sleep(5.0)     # Pendant 5 secondes, on est à l'arrêt
```

1°) Compléter les commentaires (pointillés)

2°) Décrire ce que fait ce programme

3°) Implémentez et testez-le sur votre montage électronique

Execice 3 : A partir des programmes précédents, réaliser un noveau programme qui :
- Losqu'on appuie sur un bouton poussoir BP1, démarre le moteur avec un rampe de 2V / s
- Losqu'on appuie sur un bouton poussoir BP2, arrête le moteur avec un rampe de -4V / s

Commande en Modulation de Largeur d'Impulsion Aléatoire - RPWM

Un des problème de la commande d'actionneur en PWM est l'existance d'un siffement haute fréquence, très inconfortable pour les oreilles ! Ce sifflement est centré sur la fréquence de découpage : une solution pour atténuer ce bruit désagréable est « d'étaler le spectre » et d'utiliser un système de fréquences aléatoires.

Ce type de pilotage en MLI avec fréquence aléatoire est appelé Random Pulse Width Modulation : RPWM

Execice 4 : On considère les programme ci-dessous

```
# Programme PWM
import time
import board
import pulseio
# For the M0 boards:
motorRPWM = pulseio.PWMOut(board.A2,duty_cycle=0,frequency=500,variable_frequency=False)
# For the M4 boards:
# motorRPWM=pulseio.PWMOut(board.A1,duty_cycle=0,frequency=500,variable_frequency=False)
f = 500
while True:
  motorRPWM.duty_cycle = 65535 // 2  # On a un rapport cyclique de 50%
```

```
# Programme RPWM
import time
import board
import pulseio
from math import randint
# For the M0 boards:
motorRPWM = pulseio.PWMOut(board.A2,duty_cycle=0,frequency=500,variable_frequency=True)
# For the M4 boards:
# motorRPWM =pulseio.PWMOut(board.A1,duty_cycle=0,frequency=500,variable_frequency=True)
while True:
  for f in randint(400,600):
    motorRPWM.frequency = f
    motorRPWM.duty_cycle = 65535 // 2  # On 50%
    time.sleep(0.01)  # On change la fréquence toutes les 10 ms
```

1°) Analyser les similitudes et les différences entre ces 2 programmes.

2°) Commentez la boucle du programme RPWM

3°) Sur un chronogramme tracer un exemple de signal RPWM de rapport cyclique 50 %

3°) Implémentez et testez-le sur votre montage électronique : on doit remarquer qu'il n'y a plus un sifflement aïgu autour de 400 Hz, mais un étalement des fréquences entre 400 et 600 Hz, ce qui est moins douloureux pour nos oreilles !

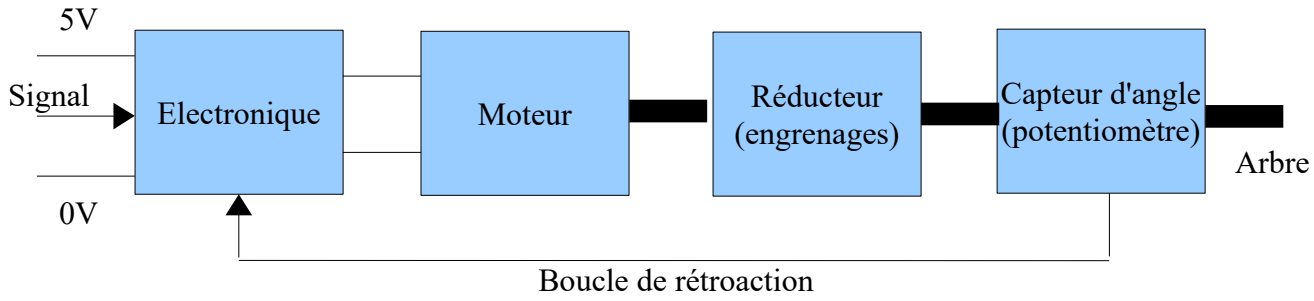
4°) Etudiez expérimentalement l'effet des bornes dans l'instruction `randint(400,600)`

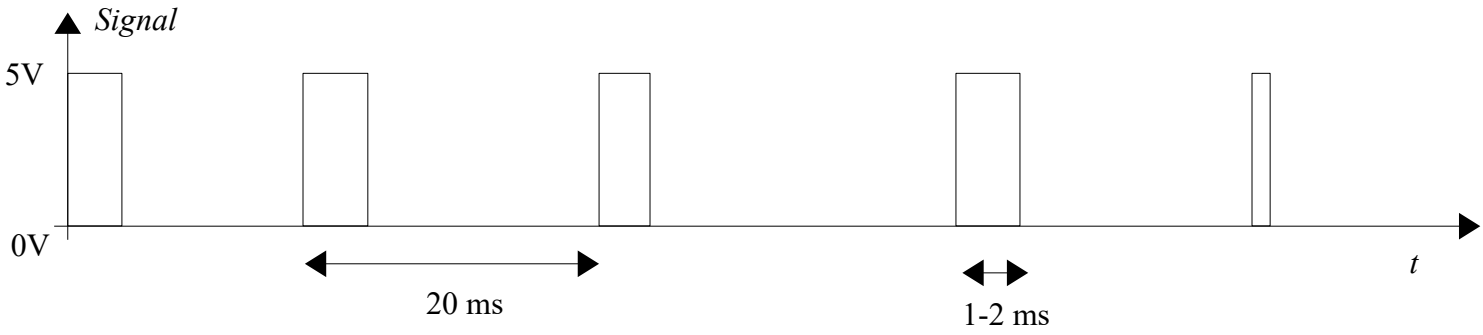
Execice 5 : A partir des programmes précédents, réaliser un nouveau programme avec de la RPWM qui :
- Losqu'on appuie sur un bouton poussoir BP1, démarre le moteur avec un rampe de 2V / s
- Losqu'on appuie sur un bouton poussoir BP2, arrête le moteur avec un rampe de -4V / s

Source : https://learn.adafruit.com/circuitpython-essentials/circuitpython-pwm

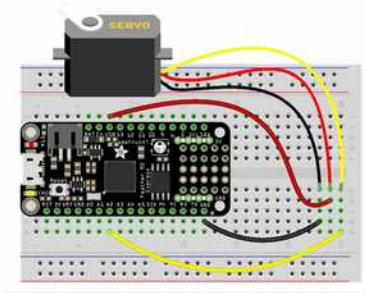
V / Servomoteurs

Les servomoteurs sont des moteurs électriques asservis en position : ces dispositifs intègrent un petit moteur à courant continu, un réducteur, un capteur de position et une électronique de gestion.





La partie asservissement de position est gérée par l'électronique interne.
L'angle de référence est codé par un signal modulé en largeur d'implusion, répété périodiquement (généralement toutes les 20 ms : 50Hz).
L'angle de l'arbre du servomoteur varie entre un angle minimum et un angle maximum. Le signal de commande correspondant varie entre 1ms et 2ms : la position neutre correspond à des impulsions de 1,5 ms.



Exercice 1 :

```
import time
import board
import pulseio
from adafruit_motor
import servo
# create a PWMOut object on Pin A2.
pwm = pulseio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
# Create a servo object,
my_servo.my_servo = servo.Servo(pwm)
while True:
  for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
    my_servo.angle = angle
    time.sleep(0.05)
  for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
    my_servo.angle = angle
    time.sleep(0.05)
```

1°) Commenter ce programme

2°) Décrire ce que fait ce programmme

Exercice 2 : Coder en langage Python un programme qui permette de piloter l'angle d'un servomoteur en fonction de la tension électrique appliquée sur l'entrée analogique A0.

Source : https://learn.adafruit.com/circuitpython-essentials/circuitpython-servo