Acquisition et traitement parallèles

1. Introduction

Ce chapitre montre comment effectuer une acquisition de signaux en mode parallèle avec la carte SysamSP5. On verra comment faire un tracé des signaux simultanément à l'acquisition, avec éventuellement un filtrage des signaux.

2. Configuration des entrées

```
import pycanum.main as pycan
import math
from matplotlib.pyplot import *
import matplotlib.animation as animation
import numpy
```

La configuration des entrées se fait comme pour une acquisition normale. On configure ici les voies 0 et 1 avec le calibre 10 volts. La période d'échantillonnage est de $10\,\mathrm{ms}$. L'acquisition dure $100\,\mathrm{s}$.

```
sys=pycan.Sysam("SP5")
sys.config_entrees([0,1],[10,10])
te=0.01
ne=10000
duree=te*ne
sys.config_echantillon(te*1e6,ne)
```

3. Tracé des signaux en mode parallèle

Pour lancer l'acquisition en mode parallèle, on exécute :

```
sys.lancer()
```

Lorsqu'on effectue un traitement des données parallèlement à leur acquisition, il faut définir un délai d'attente, qui correspond au temps approximatif qui doit s'écouler entre deux traitements consécutifs. Un traitement consiste à lire un paquet de données fourni par l'interface et à les traiter (filtrage, représentation graphique, etc). On définit le délai en secondes :

```
delai=0.1
```

Pour ce délai de 0.1 secondes, il y aura environ 10 nouveaux échantillons à chaque traitement.

On peut aussi avoir besoin du nombre de lectures à faire pendant l'acquisition complète (avec une marge de sécurité) :

```
n_lec=int(math.floor(duree*1.1/delai))
```

Voyons comment programmer une animation pour afficher les signaux. On définit une fonction qui sera appelée pour chaque image de l'animation. Cette fonction récupère les données déjà acquise par la carte SysamSP5 depuis le début et met à jour les deux courbes.

```
fig,ax = subplots()
x = [0]
y = [0]

line0, = ax.plot(x,y)
line1, = ax.plot(x,y)
ax.grid()
ax.axis([0,duree,-10,10])

def animate(i):
    global sys,line0,line1,t0,t1,u0,u1
    data=sys.paquet(0) # paquet des données déjà acquises (temps et tension)
    t0=data[0]
    t1=data[1]
    u0=data[2]
    u1=data[3]
    line0.set_xdata(t0)
    line0.set_ydata(u0)
    line1.set_xdata(t1)
    line1.set_ydata(u1)
```

Voici comment lancer l'animation. Le délai entre deux images est donné en millisecondes.

```
ani = animation.FuncAnimation(fig,animate,n_lec,interval=delai*1000)
show()
```

4. Filtrage en mode parallèle

Il est aisé d'ajouter un traitement des signaux pendant l'animation, par exemple un filtrage par convolution :

```
P=30
fc = 0.1
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[fc],nyq=0.5,window='hamming')

fig,ax = subplots()
x = [0]
y = [0]

line0, = ax.plot(x,y)
line1, = ax.plot(x,y)
ax.grid()
```

```
ax.axis([0,duree,-10,10])

def animate(i):
    global sys,h,line0,line1,t0,t1,u0,u1
    data=sys.paquet(0) # paquet des données déjà acquises (temps et tension)
    t0=data[0]
    t1=data[1]
    u0=data[2]
    u1=data[3]
    u1_filtre = scipy.signal.convolve(u1,h,mode='same')
    line0.set_data(t0,u0)
    line1.set_data(t1,u1_filtre)

ani = animation.FuncAnimation(fig,animate,n_lec,interval=delai*1000)
show()
```