Interfaçage

nov. 21, 2018

1 Notions de base

L'objectif de l'interfaçage d'un instrument est d'envoyer des instructions depuis l'ordinateur afin que celui-ci effectue des opérations normalement faites manuellement.

1.1 Connexion

Il existe plusieurs type de connexion entre un ordinateur et un instrument. Citons le GPIB, le RS232, l'USB et l'ethernet.

La GPIB et le RS232 sont des interfaces de communications qui sont entre train de devenir obsolètes, mais se trouvent encore fréquemment sur des appareils. Ce sont des interfaces simples, puisque on échange des mots d'octets entre les deux appareils. Le RS232 est le port série (COM sous windows).

L'USB est beaucoup plus complexe car il ne définit par précisément ce qui peut être échangé. Pour l'USB, il est nécessaire d'avoir un driver pour chaque mode de communications. Comme mode de communication, citons l'émulation d'un port série (c'est par exemple ce que l'on retrouve sur les arduinos). Pour les instruments de mesure, une norme appelée USBTMC est utilisée par plusieurs fabricants (USB Test and Measurement Class). Certains fabricants peuvent cependant avoir développé leur propre interface USB.

Enfin, de plus en plus de d'instruments sont connecté par ethernet. Les choses sont en général beaucoup plus simple que l'USB car il n'y a pas besoin de driver spécifique. Des normes de communications existent pour les instrument, en particulier le VXI-11.

Le VXI-11 et l'USBTMC ont pour but de remplacer le GPIB. Ces protocoles permettent d'échanger des instructions selon une norme précise, le SCPI.

1.2 SCPI

Le SCPI (Standard Commands for Programmable Instruments) est en quelque sorte la grammaire que l'on va utiliser. Par exemple, pour changer l'échelle verticale d'un oscilloscope on peut envoyer le chaîne : "CH1 :SCALE 10". Pour connaître l'échelle verticale on va envoyer "CH1 :SCALE ?". L'instrument va alors renvoyer une chaîne correspondant à l'échelle. Dans cette exemple, les mots précis dépendent de la marque de l'instrument. Ce que le SCPI défini est la syntaxe et quelques instructions de base.

Les instructions sont regroupées sous forme d'un arbre. Chaque "branche" est séparée par un " :". Les arguments sont séparés de l'instruction par une espace et sont séparés entre eux par des virgules. Il est aussi possible de transférer des données binaires. Lorsque l'instruction attend une réponse, alors elle est suivie d'un " ?".

Pour connaître l'ensemble des commandes il faut se référer à la partie de la documentation de l'instrument appelée souvent Programmers' User Guide.

Les instructions de base d'un instrument sont entre autre :

— `*IDN?` qui renvoie une chaîne d'identification unique de l'appareil
— `*RST` qui fait un reset de l'appareil.

Un dernier point important : le SCPI n'est pas sensible à la casse. De plus chaque instruction possède une version courte et une version longue. Celles-ci sont en général distinguées par la casse. Par exemple "CHannel1 :SCAle" signifie que l'on peut utiliser indépendamment CH ou CHANNEL et SCA ou SCALE.

1.3 VISA

Sous Windows, il est possible d'utiliser un driver VISA qui fournit une interface commune, quelque soit le mode de connexion ou le protocole bas niveau de communication avec l'instrument. La librairie VISA est une librairie propriétaire gratuite fournie par National Instruments. Elle peut être utilisée sous Python avec le module pyvisa (qu'il faut installer en plus de VISA).

Le logiciel NI-MAX est installé en même temps que VISA. Il permet entre autre de lister l'ensemble des appareils connecté à l'ordinateur et d'envoyer des instructions directement à l'instrument.

2 Interfacer un instrument sous Python

Nous utiliserons la librairie pyvisa. Cette librairie donne un identifiant unique pour chaque appareil connecté. Il est possible de lister les appareils

```
import visa

rm = visa.ResourceManager()
print(rm.list_resources())
conn = rm.open_resource('GPIB0::1') # Ou USB ou TCPIP0::...
```

Un fois que l'appareil est connecté, on peut dialoguer avec lui à l'aide des méthodes write, read et ask

```
conn.ask('*IDN?')
conn.write('*RST')
```

3 Class SCPI

Nous allons écrire une classe qui a pour but d'implémenter des méthodes récurrentes en SCPI mais qui ne le sont pas dans le driver VISA.

Elle sera initialisée soit avec une connexion soit une chaîne de caractère

```
class SCPI(object):
    def __init__(self, conn):
        if isinstance(conn, str):
            conn = visa.ResourceManager().open_resource(conn)
        self._conn = conn

    def ask(self, *args):
        return self._conn.ask(*args)

    def write(self, *args):
        return self._conn.write(*args)
```

```
    def read(self, *args):
        return self._conn.read(*args)

    def scpi_ask(self, cmd):
        cmd = cmd if cmd.endswith('?') else cmd + '?'
        return self.ask(cmd)

    def scpi_write(self, cmd, *args):
        pass

    def scpi_ask_for_float(self, cmd):
        """Lit le résultat de l'instruction cmd et renvoie un float"""
        pass

    @property
    def idn(self):
        return self.scpi_ask('*IDN')

    def reset(self):
        pass

    def __repr__(self):
        return "{self.__class__.__name__}({self.idn})".format(self=self)
```

Question : Complétez et testez cette classe sur l'oscilloscope et le GBF

4 Générateur de fonction

Tant pour le générateur de fonction que pour l'oscilloscope, l'objectif est de créer une objet Python simple d'utilisation (et qui cache donc les détails). Les méthodes disponibles seront donc indépendantes du modèle utilisé.

Voici par exemple comment on souhaite utilisé un GBF

```
gbf = ClassOfTheGBF('....')
gbf.frequency = 10000
gbf.amplitude = 1
gbf.function = 'sin'
gbf.output_on()
```

On va créer une classe Agilent33200

```
class Agilent33200(SCPI):
    def _get_frequency(self):
        pass

    def _set_frequency(self, value):
        pass

    frequency = property(_get_frequency, _set_frequency)
```

Question : Complétez ces méthode et faire de même pour les propriétés `amplitude`, `offset`, `function`. Créer une méthode output_on et output_off.

5 Oscillocope

Pour l'oscilloscope, on va hiérarchiser les commandes afin d'avoir une utilisation du type

```
scope = ClassOfTheScope('...')
scope.auto_set()
scope.channel1.coupling = 'AC'
scope.channel1.offset = 0
scope.channel1.scale = 100E-3
scope.horizontal.scale = 100E-6
scope.horizontal.offset = 0
scope.trigger.source = 'CH1'
scope.trigger.slope = 'PositiveEdge'
scope.trigger.level = .5
```

L'objet de type Scope contient donc des méthodes ou properties et des attributs qui sont eux même des objets ayant des méthodes ou des properties.

Voici comment faire

```
class Horizontal(object):
    def __init__(self, parent):
        self._parent = parent

    @property
    def offset(self):
        return self._parent._get_horizontal_offset()

    @offset.setter
    def offset(self, val):
        return selt._parent._set_horizontal_offset(val)

    # Ou en plus compact !
    # offset = property(lambda self:self._parent._get_horizontal_offset,
    #                   lambda self, val:self._set_horizontal_offset(val))

class Channel(object):
    def __init__(self, parent, index):
        self._parent = parent
        self._index = index

    @property
    def scale(self):
        return self._parent._get_channel_scale(self._index)

class Scope(SCPI):
    @property
    def trigger(self):
        return Trigger(self)

    @property
    def channel1(self):
        return Channel(self, i)
```

Question : Complétez cet objet. On pourra dans un premier temps ne pas implémenter le trigger.

Le plus difficile pour l'oscillo est de récupérer les données. Elle arriverons sous forme binaires. Il y a donc deux problèmes : récupérer les données et les convertir en tableau numpy.

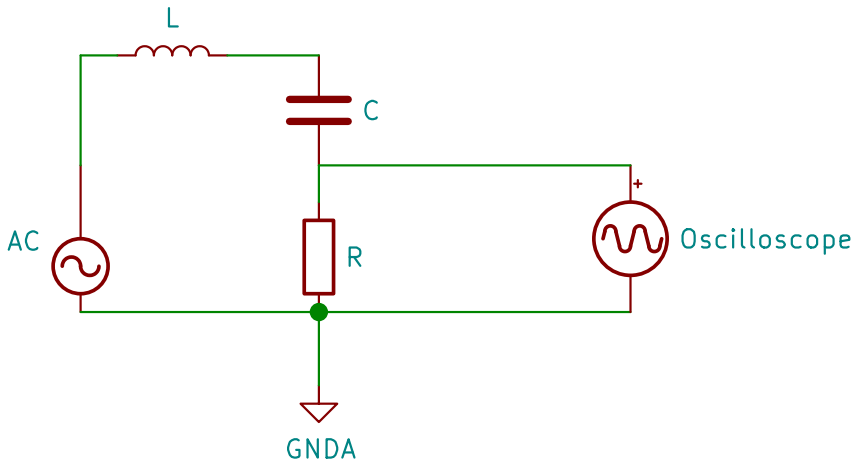
— Dans le norme SCPI, le données binaires sont enregistrée de la façon suivante : il y a un entête suivi des données. L'entête (en ASCII) est le suivant : `#nxxxxx` où `n` est le nombre de `x` et `xxxxx` est la taille des données. Par exemple un tableau de 1000 octets commencera par `#800001000`. Ce sera suivi donc des données et d'un caractère de fin de chaîne.
— La fonction frombuffer de numpy sera utilisée pour décoder directement les donnés.
— Il faut configurer correctement l'oscilloscope pour qu'il renvoie les données. Les données sont des entiers 8 bits (correspondant à la position sur l'écran) qu'il faut convertir. Enfin on a besoin de l'axe x. Voici ce que ça donne

```
class Keysight(Scope):
    def _get_channel_scale(self, channel):
        pass

    def _set_data_source(self, channel):
        self.write(':WAVeform:SOURce CHANnel{}'.format(channel))

    def _get_preamble(self):
        # Format, Type, Points, Count, XIncrement, XOrigin,
        # XReference, YIncrement, YOrigin, YReference
        out = self.ask(':WAV:PRE?')
        return list(map(eval, out.strip().split(',')))

    def ask_array(self, cmd):
        self.write(cmd)
        first = self._inst.read(1)
        header_size = eval(self._inst.read(1))
        size_str = self._inst.read(header_size)
        while size_str[0]==b'0': # remove leading 0
            size_str = size_str[1:]
        size = int(size_str)
        output = self._inst.read(size)
        self._inst.read(1) # \n
        return output

    def _get_channel_waveform(self, channel=1, **kwd):
        self._set_data_source(channel)
        self.write(':WAV:FORMAT BYTE')
        (Format, Type, Points, Count,
            XIncrement, XOrigin, XReference,
            YIncrement, YOrigin, YReference) = self._get_preamble()

        buff = self.ask_array(':WAV:DATA?')
        res = np.array(np.frombuffer(buff, dtype = np.dtype('uint8')),
                                                    dtype=int)
        data = (res - YReference)*YIncrement
        t0 = XOrigin
        dt = XIncrement
        return data, t0, dt
```

Question : En utilisant le code ci-dessus, faites en sorte que l'on puisse récupérer les données de la façon suivante

```
data, t0, dt = scope.channel1.waveform
```

6 Diagramme de Bode



Question : Réalisez le diagramme de Bode du circuit RLC.

Question : Ajustez la courbe par la formule théorique pour trouver la fréquence de résonance et le facteur de qualité du filtre.