Contrôle d'instruments scientifiques : les protocoles GPIB, VXI11 et USBTMC

J.-M Friedt

5 janvier 2010

1 Introduction

Le monde des instruments de mesure est suffisamment restreint pour qu'un petit nombre de protocoles aient émergé et que ces interfaces soient disponibles sur la majorité des équipements. Historiquement, le contrôle d'instrument s'est longtemps fait par une norme peu connue dans les milieux autres que l'instrumentation scientifique – GPIB, norme IEEE 488.2 – qui tend aujourd'hui a être remplacée par un protocole sur bus ethernet, VXI11. Nous allons voir que l'exploitation de ces deux protocoles de commande et d'acquisition de données est possible en utilisant exclusivement du logiciel libre. Nous achèverons ce tour d'horizon des protocoles de contrôle par l'exploitation du bus USB pour commander un instrument grâce à la classe USBTMC, introduite dans le noyau Linux depuis sa version 2.6.28. Ce protocole est notamment disponible sur des instruments bas de gamme (type oscilloscope numérique utilisé en enseignement) et permet, là encore, d'automatiser des séquences de mesures ou acquérir des données pour traitement ou illustrations de rapports d'expériences.

2 GPIB – IEEE 488

GPIB, initialement développé par Hewlett Packard et donc aussi nommé HPIB, est un bus dont la liaison se fait en parallèle sur un grand nombre de signaux. Les câbles, contenant un grand nombre de conducteurs, sont donc relativement coûteux et limités en longueur à 4 m entre instruments, et 20 m pour la longueur totale du bus. De plus, une carte de contrôle dédiée est nécessaire puisque ce protocole n'équipe pas en standard les ordinateurs usuellement disponibles. Ce protocole reste néanmoins incontournable, qu'il s'agisse par habitude des programmeurs ou disponibilité d'instruments coûteux trop anciens pour être équipés d'interface ethernet. Par ailleurs, le convertisseur USB-GPIB que nous exploitons dans ces exemples répond à une exigence de mobilité de l'ordinateur d'acquisition et contrôle des informations : tous les exemples proposés ici ont été testés au moyen d'une interface National Instruments GPIB-USB-HS et la version 3.2.14 de `linux-gpib` [1] (Fig. 1).

La philosophie qui domine les développements présentés ici est d'automatiser au maximum la phase d'acquisition des informations, sans imposer à l'utilisateur la phase de configuration de l'appareil afin de garder un maximum de souplesse. Il s'agit là probablement d'une approche d'ingénierie dédiée à la recherche, où il est rare que deux expériences soient strictement identiques, par rapport à l'approche industrielle qui a tendance à répéter un grand nombre de fois la même opération, incluant la configuration de l'instrument avec des paramètres pré-définis. Dans ces exemples, nous allons supposer que l'utilisateur a manuellement réglé les paramètres de mesure à sa convenance, a observé visuellement une courbe satisfaisante, et désire maintenant automatiser la phase d'acquisition pour enregistrer par exemple l'évolution temporelle d'un phénomène. Il peut néanmoins être utile de mémoriser une configuration afin de garantir qu'une série d'expériences étalées dans le temps sont effectuées avec les mêmes paramètres de mesure : GPIB permet aussi de recevoir et imposer la configuration, mais nous n'aborderons par ce point ici.

L'installation des pilotes et de la bibliothèque de gestion `linux-gpib` s'obtient trivialement, après rapatriement des sources à [1], par l'habituel `./configure && make && make install`. Lors du démarrage, le module approprié est inséré (dans notre cas lors de l'insertion à chaud du convertisseur GPIB-USB-HS). Le numéro de l'interface est alors défini par

```
gpib_config --minor 0 -t ni_usb_b
```

FIG. 1 – Interface USB-GPIB sur eeePC701 : ce type d'interface, tout comme l'interface ethernet pour communiquer avec les instruments supportant le protocole VXI11, permet d'automatiser l'acquisition de données quelque soit l'emplacement de l'instrument dans un laboratoire. Au-delà de l'automatisation des mesures, la mise en œuvre d'une expérience complexe regroupant plusieurs instruments impose des contraintes de distances telles que décrites dans le texte.

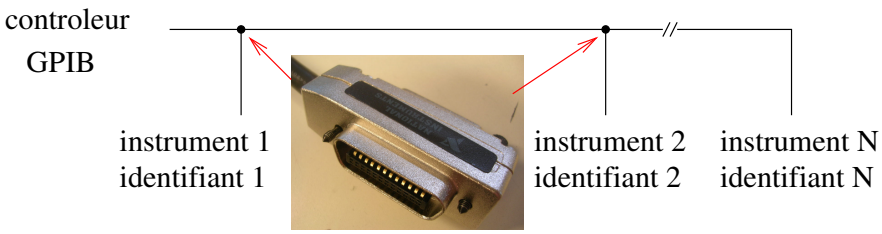

FIG. 2 – Un bus GPIB est contrôlé par un unique maître, généralement un PC équipé d'une carte de contrôle dédiée, et d'une multitude d'instruments esclaves chaînés : chaque connecteur GPIB s'enfiche d'un côté (mâle) sur un instrument et présente une interface femelle pour continuer la chaîne.

en prenant soin d'adapter le nom de son interface matérielle GPIB. Une fois le matériel reconnu, un programme C accède aux fonctions de communication sur bus GPIB en incluant l'entête `"gpib/ib.h"` puis en compilant avec l'option de liaison `-lgpib`. On prendra soin, en cas d'erreur à l'exécution, d'inclure le répertoire contenant la bibliothèque `libgpib.so` dans sa variable d'environnement `LD_LIBRARY_PATH`.

2.1 Description du bus : les adresses

Chaque instrument est défini, sur un bus GPIB, par un identifiant unique. Ce nombre, entre 0 et 30, est configuré sur chaque instrument soit au moyen de DIP switchs sur les instruments les plus anciens, ou programmé si un système d'exploitation est exécuté. La carte de contrôle est en général définie par l'identifiant 0 (un driver GPIB supporte en général jusqu'à 4 cartes de contrôle [2]). Deux instruments identifiés par le même nombre entreront en conflit lors d'une requête : on prendra donc soin de configurer chaque instrument avec un identifiant différent.

Les structures de données et fonctions proposées par `linux-gpib` sont identiques en terme de nomenclature et de prototype à celles fournies dans les bibliothèques propriétaires de National Instruments [2], le principal fournisseur aujourd'hui de cartes de contrôle.

La fonction `ibdev()`, équivalente à l'ouverture d'un descripteur de fichier, initialise la commu-

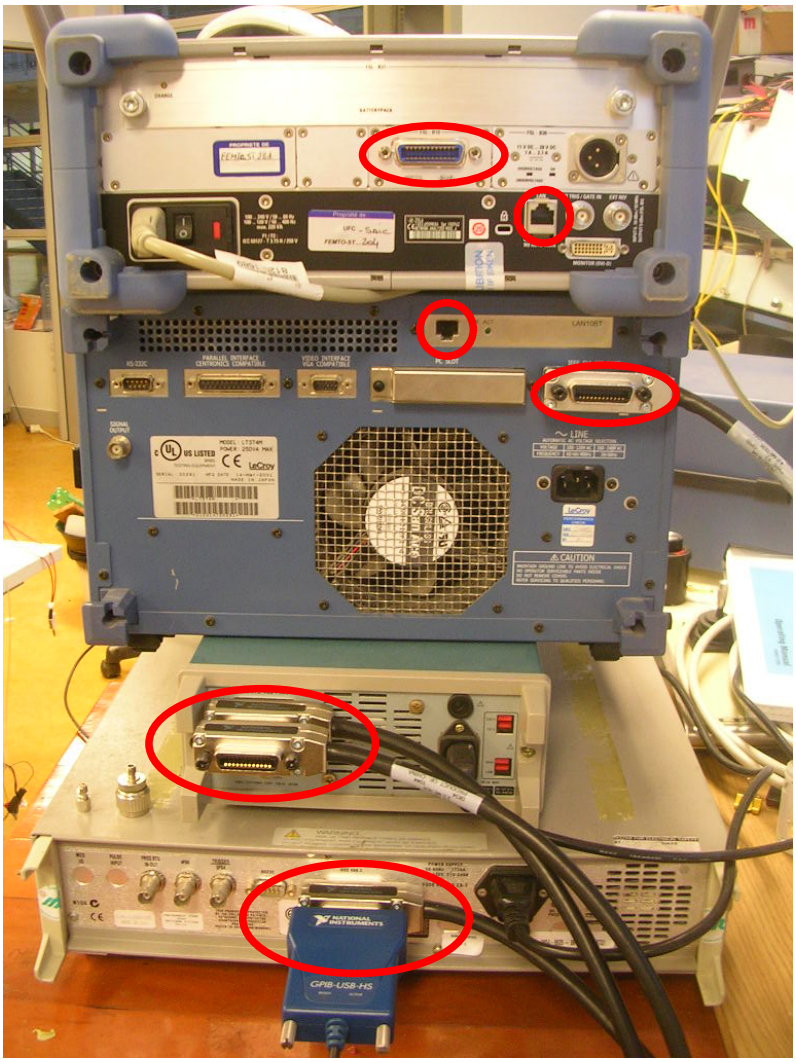

FIG. 3 – Exemple d'automatisation d'expérience pour la caractérisation automatique de dispositifs radiofréquences : divers instruments (ici un synthétiseur de fréquence, un générateur de fonctions et un oscilloscope numérique) sont connectés à un bus GPIB et contrôlés par un PC.

nication avec un instrument d'identifiant primaire `pad`. Cette fonction prend en argument, en plus de l'identifiant, une durée de timeout (typiquement `T1S` ou `T10S` pour 1 ou 10 secondes, définie dans `gpib_user.h`), pour renvoyer un index unique pour chaque instrument :

```
int dev,board_index=0,pad=10,sad=0,send_eoi=1,eos_mode=0 ;
dev=ibdev(board_index, pad, sad, T1s, send_eoi, eos_mode ) ;
```

Deux opérations permettent de valider l'initialisation de la transaction entre l'ordinateur de contrôle et l'instrument : d'une part `dev` doit être positif, et d'autre part nous allons émettre une commande standard qui demande à l'instrument de s'identifier.

2.2 Transactions : écritures

La première opération que nous désirons effectuer pour valider la communication entre le PC de contrôle et un instrument est d'émettre une requête standard de demande d'identification.

Bien que chaque fabriquant d'instrument ait une logique et des commandes qui lui sont propres, quelques commandes ont été imposées dans le standard – commençant par "*" – dont `*IDN?` qui est une requête de la chaîne de caractères contenant la marque, le modèle et la version de l'appareil. Toute commande GPIB est une chaîne de caractères ASCII, émise par la commande `ibwrt()` prenant pour argument le tableau de caractères et la longueur de la chaîne. Ainsi,

```
char buffer[7]="*IDN?\n";ibwrt(dev,buffer,strlen(buffer));
```

envoie sur le bus GPIB la trame `*IDN?` à destination de l'instrument identifié par `dev`.

Certaines commandes émises servent à configurer l'instrument et aucune réponse n'est attendue, d'autre sont des requêtes d'informations (généralement finissant par "?") et la transaction se poursuit de l'instrument vers le PC. La commande la plus standard, dont la syntaxe n'a jamais été modifiée depuis la création de la norme GPIB [1], est la commande `RST`, qui permet de *toujours* valider la liaison entre l'ordinateur et un instrument, et dont l'effet (réinitialiser l'instrument) est visible sur ce dernier.

2.3 Transactions : lectures

Afin de lire la réponse de l'appareil à la requête `*IDN?`, la commande `ibread()` permet de recevoir l'information en retour. Nous devons fournir en argument un nombre de caractères maximum dans la réponse, et un tableau de caractère contenant au moins autant d'octets. Si la taille de la réponse est supérieure au nombre de caractères requis, l'obtention de la réponse complète peut se faire en plusieurs requêtes (ce cas est par exemple courant pour récupérer une trace d'oscilloscope de plusieurs centaines de kB). Dans le cas qui nous intéresse ici,

```
ibrd(dev, buffer,buffer_length );
buffer[ThreadIbcntl()-1]=0;
```

avec `buffer` un tableau de caractères et `buffer_length` un entier contenant le nombre de caractères maximum requis, renvoie la chaîne de caractères identifiant l'instrument. La fonction `ThreadIbcntl()` renvoie le nombre de caractères effectivement lus et est exploitée pour former une chaîne compatible avec la norme du C (dernier caractère nul).

Un exemple de résponse aux requêtes `*IDN` lors du balayage du bus GPIB est :

```
MARCONI INSTRUMENTS,2024,112282/655,44533/446/04.08
LECROY,LT374M ,LT37400291,09.2.0
```

2.4 Exemple d'utilisation

Le programme suivant est un exemple d'utilisation concrète du protocole GPIB pour contrôler deux instruments, un synthétiseur de fréquences Marconi 2024 qui émet un signal dans la gamme FSTART à FSTOP. Pour chaque fréquence, les valeurs de 3 voies d'un oscilloscope numérique LeCroy LT374M sont mémorisées pour traitement ultérieur.

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>

#include "gpib/ib.h"
#define FSTART 433.44
#define FSTOP  433.64
#define FSTEP ((FSTOP-FSTART)/100)

void relit(int dev,uint8_t *buffer,int buffer_length)
{ibrd(dev, buffer,buffer_length );
 buffer[ThreadIbcntl()-1]=0;
}
```

[1] par exemple, `*IDN?` n'est pas supporté par l'analyseur de réseau HP4195A, qui réclame à la place `ID?`

```
20 void envoi(int dev, uint8_t *buffer)
   {if (ibwrt(dev,buffer,strlen(buffer))&ERR) printf("error writing\n");
22 }

24 int main(int argc, char *argv[] )
   {int dev1,dev2,board_index=0,pad1=10,pad2=5,sad=0,send_eoi=1,eos_mode=0;
26  float frequence;
    char *buffer;
28  static const unsigned long buffer_length = 10000000;
    time_t t1,t2;
30  FILE *f;

32  buffer = (char *)malloc( buffer_length );
    dev1 = ibdev( board_index, pad1, sad, T1s, send_eoi, eos_mode );
34  if (dev1 <0) {printf("error opening device1\n");return(0);}

36  dev2 = ibdev( board_index, pad2, sad, T1s, send_eoi, eos_mode );
    if (dev2 <0) {printf("error opening device2\n");return(0);}
38
    sprintf(buffer,"*IDN?\n");
40  envoi(dev1,buffer); relit(dev1,buffer,buffer_length); printf("%s\n",buffer);

42  sprintf(buffer,"*IDN?\n");
    envoi(dev2,buffer); relit(dev2,buffer,buffer_length); printf("%s\n",buffer);
44
    if (argc>1) sprintf(buffer,"osc%d.dat",atoi(argv[1]));
46        else  sprintf(buffer,"osc.dat");

48  f=fopen(buffer,"w");
    sprintf(buffer,"C1:INSPECT? 'WAVEDESC'\n");
50  envoi(dev2,buffer); relit(dev2,buffer,buffer_length);
    fprintf(f,"%s\n",buffer);
52
    for  (frequence=FSTART;frequence<FSTOP;frequence+=FSTEP)
54     {sprintf(buffer,"CFRQ:VALUE %fMHz\n",frequence);
       envoi(dev1,buffer);
56     printf("%s",buffer);

58     fprintf(f,"freq=%f\n",frequence);

60     sprintf(buffer,"C1:INSPECT? 'SIMPLE'\n");
       envoi(dev2,buffer); relit(dev2,buffer,buffer_length);
62     fprintf(f,"%s\n",buffer);
       sprintf(buffer,"C2:INSPECT? 'SIMPLE'\n");
64     envoi(dev2,buffer); relit(dev2,buffer,buffer_length);
       fprintf(f,"%s\n",buffer);
66     sprintf(buffer,"C3:INSPECT? 'SIMPLE'\n");
       envoi(dev2,buffer); relit(dev2,buffer,buffer_length);
68     fprintf(f,"%s\n",buffer);

70     sleep(1);
      }
72 }
```

Listing 1 – Deux instruments reliés au bus GPIB, un synthétiseur de fréquences d'adresse 10 et un oscilloscope d'adresse 5, sont interrogés avec un délai de 1 s entre chaque mesure. Le contenu de 3 voies de l'oscilloscope est stocké dans le fichier d'identifiant `f`.

Cet exemple est typique de l'automatisation d'acquisitions fastidieuses, pénibles pour un opérateur humain, et fournissant ainsi une somme d'informations impossibles à acquérir de façon manuelle dans un délai raisonnable. Nous nous contentons ici de démontrer l'acquisition des informations, mais le corollaire à un tel programme est évidemment l'asservissement de la fréquence générée sur le signal observé à l'oscilloscope afin de fermer une boucle de contrôle après cette première phase de caractérisation en boucle ouverte.

Nous retrouvons dans ce programme les principales fonctions décrites auparavant : au début (ligne 31 à 37) `ibdev()` pour initialiser la liaison, `ibwrt()` (lignes 39 et 40, enrobée dans la fonction `envoi()` pour envoyer une commande à l'instrument et `ibrd()` (enrobée dans la fonction `relit()` ligne 15) pour lire la réponse. Dans cet exemple, nous communiquons avec deux instruments – d'identifiants primaires 5 et 10 – et par conséquent dupliquons toutes ces commandes (les deux identifiants primaires sont définis lors de l'initialisation des variables à la ligne 25). Les ordres fournis à chaque instrument, correspondant généralement aux séquences de touches sur la face avant de l'appareil, sont spécifiques à chaque marque et décrites dans le manuel du programmeur associé aux instruments.

3 VXI11 sur ethernet

Avec des débits relativement faibles (1 à 8 Mb/s) et la nécessité d'acquérir une interface matérielle dédiée, GPIB tend à être remplacé dans les instruments récents par une interface ethernet visible sous forme d'une embase RJ45. La liaison ethernet supporte le protocole de communication TCP/IP.

Le protocole VXI11 [3] permet de contrôler un instrument équipé d'une interface ethernet et donc identifié par son adresse IP. L'avantage d'une telle stratégie est l'exploitation de l'infrastructure du réseau informatique existante, ou tout au moins d'exploiter des câbles à paires torsadées (*Unshielded Twister Pair* – UTP) peu coûteux pouvant mesurer jusqu'à 100 m de longueur. Basé sur la couche RPC [4], VXI11 est implémenté sous forme de bibliothèques simples à exploiter disponibles à [5]. Nous utilisons pour notre part un câble croisé pour acquérir périodiquement et automatiquement les données d'un instrument équipé d'une telle interface : l'analyseur de réseau Rohde & Schwartz ZVL. Cet instrument permet de caractériser en quelques secondes les propriétés en transmission et réflexion d'un quadipôle fonctionnant dans les gammes radiofréquences. Acquérir automatiquement les courbes d'un tel instrument nous permet de caractériser un capteur dont l'environnement évolue avec le temps. Le volume important d'informations transférées avec chaque courbe caractérisant les performances du capteur justifie l'exploitation de VXI11 au lieu de GPIB.

Un énorme intérêt de VXI11 (aussi nommé LXI – *LAN eXtension for Instrumentation* – dans la littérature) est l'exploitation des commandes GPIB : bien que le support matériel et le protocole logiciel de communication change, les instructions transmises aux instruments et les données reçues *sont les mêmes que dans le cas du GPIB*. Il est donc facile de porter les logiciels proposés auparavant de GPIB à VXI11, et nous proposons, pour les instruments munis des deux interfaces, les fonctions suivantes :

```
#ifdef gpib
void relit(int dev,uint8_t *buffer,int buffer_length)
#else
void relit(CLINK *clink,char *buffer,int buffer_length)
#endif
{
#ifdef gpib
        ibrd(dev, buffer,buffer_length );
        buffer[ThreadIbcntl()-1]=0;
#else
        int ret;
        ret=vxi11_receive(clink,buffer,buffer_length);
        buffer[ret-1]=0;
#endif
}
```

Cette unique fonction de lecture des données est compatible soit avec le GPIB (si cette constante est définie) soit VXI11 (dans le cas contraire). De la même façon, l'émission d'un ordre par

```
#ifdef gpib
void envoi(int dev, uint8_t *buffer)
#else
void envoi(CLINK *clink,char *buffer)
```

```
#endif
{
#ifdef gpib
  if (ibwrt(dev,buffer,strlen(buffer))&ERR) printf("error writing\n");
#else
  vxi11_send(clink, buffer);
#endif
}
```

permet de facilement passer d'un protocole à l'autre. L'initialisation du périphérique – cette fois identifié par son adresse IP – s'obtient par

```
CLINK    *dev;
char     device_ip[25];
strncpy(device_ip,"192.168.1.3",25);
dev = new CLINK;
if (vxi11_open_device(device_ip,dev)!=0) printf("erreur ouverture\n");
```

3.1 Tests de débit

Afin de comparer les performances du bus GPIB et VXI11 sur ethernet, nous effectuons 1000 acquisitions de courbes de 1601 points complexes (3202 points au total, ou 42914 octets/courbe) depuis un analyseur Rohde & Schwartz ZVL, sans effectuer de balayage des fréquences entre deux acquisitions. Ainsi, le temps mesuré est réellement le temps de transfert des données, sans tenir compte du temps que met l'instrument à acquérir chaque courbe. La fonction `time` donne le temps total d'exécution du programme : 16,8 secondes pour la liaison sur VXI11 avec un eeePC 701 relié à l'instrument par câble ethernet croisé, contre 75,8 s pour la liaison GPIB *via* une interface USB-GPIB-HS. Le gain en débit est donc significatif lors de l'exploitation de l'interface ethernet : 0,45 Mb/s pour le GPIB et 2,0 Mb/s pour VXI11 (Fig. 4).

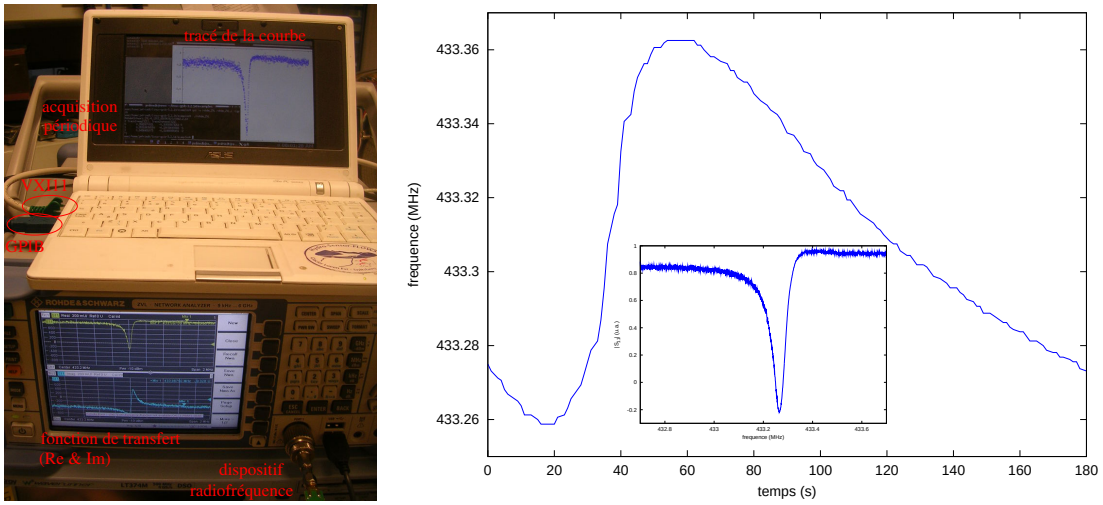

FIG. 4 – Gauche : prise de mesures automatiques à intervalles de temps réguliers pour observer le comportement d'un dipôle radiofréquence dont les caractéristiques évoluent avec la température. Droite : l'acquisition toutes les secondes des fonctions de transfert (en encart) permet, par identification de la fréquence de résonance (minimum du coefficient de réflexion), de retrouver la température du dispositif, et ainsi de qualifier le comportement du capteur.

Le format ASCII exploité ici, très défavorable en terme de temps de transfert, a l'avantage de fournir des fichiers trivialement exploitables avec les outils de traitement les plus courant (GNU/Octave, gnuplot). Dans cet exemple, les parties réelle et imaginaire de la courbe sont entrelacées, et nous retrouvons nos informations (sous GNU/Octave) par

```
load donnees2.dat
```

```
y=donnees2(k,2:2:3203)+i*donnees2(k,3:2:3203);
```

puisque la première colonne contient la date (en secondes depuis le début de l'expérience) de l'acquisition. Une telle série de données acquises automatiquement à intervalles de temps réguliers (Fig. 4, gauche), permet de remonter à la tempérture d'un capteur (Fig. 4, droite) par traitement de toutes les courbes pour en identifier le minimum de coefficient de réflexion (ou fréquence de résonance, encart de la Fig. 4 droite, qui est la donnée brute mesurée).

4 GPIB sur USB : *Test and Measurement Class* (TMC)

En observant les différents instruments disponibles dans un laboratoire, il est apparu qu'un certain nombre d'oscilloscopes achetés ces dernières années sont équipés de ports USB B. La norme USB définit le connecteur de type B comme esclave : il doit donc être possible de contrôler ces instruments par cette liaison. Une rapide recherche sur le web oriente nos recherches sur le module noyau `usbtmc` (USB *Test and Measurement Class*) dont l'archive est hébergée par Agilent [7]. Ce module se compile trivialement sous réserve d'avoir les entêtes du noyau dans son arborescence, pour générer un module noyau qu'on insèrera au moyen du script `usbtmc_load` qui se charge aussi de créer les entrées nécessaires dans `/dev`. En chargeant le module noyau `usbtmc.ko` avec un instrument connecté à un des ports, nous obtenons :

```
[148261.749905] usbcore: registered new interface driver USBTMC
[148293.924042] usb 2-2: new full speed USB device using uhci_hcd and address 8
[148294.102777] usb 2-2: configuration #1 chosen from 1 choice
[148294.108922] USBTMC: MKDEV
[148294.108936] USBTMC: CDEV_ADD
[148294.109299] usb 2-2: New USB device found, idVendor=0957, idProduct=0588
[148294.109308] usb 2-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[148294.109315] usb 2-2: Product: DSO1002A
[148294.109321] usb 2-2: Manufacturer: Agilent Technologies
[148294.109328] usb 2-2: SerialNumber: CN49251874
```

Ce module noyau nous donne accès dans un premier temps *via* `/dev/usbtmc0` à la liste des instruments connectés aux ports USB

```
$ cat /dev/usbtmc0
Minor Number    Manufacturer    Product Serial Number
001     Agilent Technologies    DSO1002A        CN49251874
002     Tektronix, Inc. Tektronix TDS2012B      C062033
```

Chaque instrument connecté à un port est accessible *depuis le shell* en écrivant et en lisant sur `/dev/usbtmc`i, $i > 0$:

```
# echo *IDN? > /dev/usbtmc1
# cat /dev/usbtmc1
Agilent Technologies,DSO1002A,CN49251874,00.04.02
```

Nous avons en particulier expérimenté avec le Tektronix TDS2012B et le Agilent DSO1002A (Fig. 5). Nous avons pu constater que depuis le shell, sans programmer une ligne de code en C, il nous est possible d'une part de récupérer les coordonnées des points affichés sur l'écran d'un oscilloscope, et d'autre part d'effectuer une capture d'écran (image bitmap). Cependant, quelques subtilités subsistent, et notamment la taille du tampon d'échange des données lors d'une lecture par `cat`. Le fichier d'entête `usbtmc.h` contient la définition de la taille du tampon dans la constante `USBTMC_SIZE_IOBUFFER` qui vaut 4096. En pratique, chaque périphérique exploite une partie du

FIG. 5 – Exemple d'utilisation de deux oscilloscopes par USBTMC. À droite, comparaison de la vue sur oscilloscope Agilent lorsqu'une entrée est connectée au signal crénau de test, et capture d'écran acquise au moyen de la commande `:DISP :DATA?` transférée sur bus USB.

tampon par un entête qui implique que chaque paquet transmis contient au maximum 4096 octets. Ainsi, récupérer une trace ou une image de capture d'écran nécessite un certain nombre d'appels depuis le shell pour obtenir l'information complète. Ainsi, pour obtenir une capture décran dans un fichier au format BMP sur un DSO1002A (noter l'appel à `sed` pour éliminer un préfixe à l'image au format BMP), on utilisera :

```
echo ":DISP:DATA?" > /dev/usbtmc1
for i in $(seq 1 20)
do
 cat /dev/usbtmc1 | sed 's/^#800077878//g' >> image.bmp
done
```

tandis qu'un simple

```
echo ":WAV:FORM ASCII" > /dev/usbtmc1
echo ":WAV:DATA?" > /dev/usbtmc1
cat /dev/usbtmc1 | tr ',' '\n'
```

permet d'obtenir en une seule transaction tous les *points* (coordonnées) de la trace affichée. Nous avons tenté diverses stratégies pour obtenir, avec un algorithme systématique, une copie d'écran de l'oscilloscope – en visant à nous affranchir de ce nombre prédéfini de lectures (`seq 1 20`) pour tester la fin de communication. Néanmoins, le stockage temporaire dans un variable du résultat de la lecture de `/dev/usbtmc1` ne peut pas fonctionner car `bash` ne sait pas manipuler de données binaire et tronque la lecture au premier caractère 0 rencontré [2]. De la même façon, nous

[2] http://mywiki.wooledge.org/BashFAQ/058

avons été incapables de faire varier la taille du tampon afin d'obtenir toute l'image en une unique transaction.

5 Conclusion

Tout comme l'automatisation du traitement des informations permet d'extraire des structures inaccessibles par des opérations manuelles, l'acquisition automatique de données permet d'engranger une masse d'information sur le comportement d'un dispositif pour en caractériser le comportement. Nous avons illustré dans ces exemples l'utilisation de trois bibliothèques libres pour exploiter les principales interfaces qui équipent les instruments de mesure scientifiques, GPIB, VXI11 et USBTMC.

Souhaitons que ces lignes dissuadent certains encadrants de stages d'employer des esclaves∧Wétudiants pour des tâches aussi répétitives et ingrates, pour leur laisser l'opportunité de s'épanouir sur l'aspect créatif de la mise en œuvre des protocoles de mesures.

Remerciements

Les instruments et interfaces exploités dans ce document sont la propriété du département temps-fréquence de l'institut FEMTO-ST de Besançon. Les commentaires lors de la relecture de X. Vacheret (FEMTO-ST, Besançon) ont amélioré la qualité du manuscrit.

Références

[1] http://linux-gpib.sourceforge.net

[2] http://www.ni.com/pdf/manuals/370963a.pdf

[3] http://www.lecroy.com/tm/library/AppNotes/LXI/LXI_Interfacing_AppNote.pdf

[4] J. Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc. (1992)

[5] S.D. Sharples, *VXI11 Ethernet Protocol for Linux*, disponible à http://optics.eee.nothingham.ac.uk/vxi11 (accessible 11/2009)

[6] Agilent propose une page web dédiée au contrôle de ses instruments sous GNU/Linux et des documents associés, par exemple *Using Linux to control LXI Instruments through VXI-11* à http://cp.literature.agilent.com/litweb/pdf/5989-671EN.pdf

[7] www.agilent.com/find/linux donne accès au module noyau USBTMC