MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE AKLI MOHAND OULHADJ -BOUIRA

Faculté des Sciences et des Sciences Appliquées
Département Génie Electrique

Mémoire de fin d'études pour l'obtention du diplôme de
Master en électronique
Option : Electronique des systèmes embarqués

Thème

Acquisition de données et contrôle d'instruments de laboratoire via le bus de communication « GPIB »

Réalisé par :
M. GUIDDIR Amirouche
M.FOUDI Aghiles

Encadré par :
M. MESSAI Adnane
M.MOUDACHE Said

Jury composé de :
Mr. BENZIANE Mourad
Mr. BOUGHEROUATE Ali
Mr. SAOUDI Kamel

Année universitaire :
2017 - 2018

Remerciements

Nous tenons d'abord à remercier Dieu le tout puissant et miséricordieux qui nous a donné la force et la patience d'accomplir ce de travail.

Nous voudrions ensuite présenter nos chaleureux remerciements à notre encadreur M. MESSAI Adnane et notre co-encadreur M. MOUDACHE Said. Nous voudrions leur témoigner notre gratitude pour leurs soutien, leurs disponibilité et leurs orientations qui mous nous ont permis de mener notre travail à bon port.

Nous souhaitons à l'occasion remercier toutes les personnes qui nous ont porté aide au sein du Centre Nucléaire de Recherche Berine , pendant notre stage pratique.

Nos remerciements vont aussi aux membres du Jury pour l'intérêt qu'ils ont porté notre travail en acceptant de l'examiner et de l'enrichir par leurs remarques.

Nous tenons à saisir cette occasion pour adresser nos remerciements aussi à nos enseignants et à tout le personnel du département génie électrique.

En fin, nous remercions chaleureusement nos familles et nos amis pour leurs encouragements qui nous ont permis de surmonter tous les obstacles.

Amirouche et Aghiles

Dédicaces

Nous dédions ce travail à nos deux familles,

A KhoukhaGuiddir,

A Imad Fettache,

A tous nos amis,

Amirouche et Aghiles

Table de matières

Table de matières

Table de matières

I.Partie théorique

Introduction

L'établissement de la communication entre les différents matériels utilisés est incontournable pour l'accomplissement de toute recherche ou travail en laboratoire. Les systèmes de communication ont connu un grand développement depuis la création du premier ordinateur.

Dans ce chapitre, on se propose dans un premier temps, de donner un aperçu sur les systèmes de communication (ISA, VLB VESA, PCI), ensuite présenter la carte GPIB Quancom et décrire quelque fonctions fournies par cette interface, et enfin présenter le bus de communication GPIB.

Section 1 : Ordinateurs et bus d'interface

Un bus informatique est un système de communication soit entre les composants d'un même ordinateur, ou entre un ordinateur et un autre équipement externe. On distingue de ce fait sur un ordinateur deux principaux bus :

- **Le bus système :** (appelé aussi ***bus interne***, en anglais internal bus ou front-side bus, noté FSB). Il permet au processeur de communiquer avec la mémoire centrale du système (mémoire vive ou RAM).

- **Le bus d'extension :** (parfois appelé bus ***d'entrée/sortie*** ou ***bus externe***) c'est le bus de second niveau d'un ordinateur. Comme son nom l'indique, il est conçu pour permettre l'ajout à un ordinateur de cartes d'extension qui le dotent de fonctions spécialisées absentes de la configuration initiale. Notons que ces cartes spécialisées peuvent être des contrôleurs de bus externe (SCSI, FireWire, Ethernet, GPIB notamment) qui introduisent un troisième niveau de connectivité pour l'ordinateur et relient ce dernier à des périphériques externes tel notre cas d'étude.

Lors du travail qu'on aura à décrire tout au long de ce manuscrit, on aura souvent à faire avec le deuxième type de bus, i. e. le bus d'extension ou le bus externe. C'est pour cette raison qu'on a jugé utile de donner un petit aperçu sur ce type de bus, ses variantes et ses caractéristiques.

1. 2 Bus d'extension

Il existe différents types de bus d'extension normalisés par : Leurs formes, le nombre de broches de connexion et le type de signaux (*fréquence, données, ... etc.*). Parmi les bus d'extension les plus utilisée on distingue :

1. 2. 1 Le bus ISA (Industry Standard Architecture)

Le bus ISA date des premiers PC XT d'IBM (International Business Machine). Au début, le nombre de bits de données était de 8 bits, et la fréquence atteint les 4.77 MHz, avec le microprocesseur 486, l'IBM a sorti le même bus ISA en évoluant des caractéristiques puisqu'il (le bus) va passer en version 16 bits avec une vitesse qui atteint 8 MHz. Cette vitesse est toujours utilisée même dans les derniers PC. La bande passante de bus ISA 16 bits est de 16 MB/s, ainsi la carte de 8 bits a la possibilité de s'insérer dans un bus ISA de 16 bits. A la fin des années 90, ce type du bus était en voie de disparition vu son inutilité de la part des concepteurs des cartes mères, en incluant un nouveau bus appelé PCI au lieu de ISA. [2]

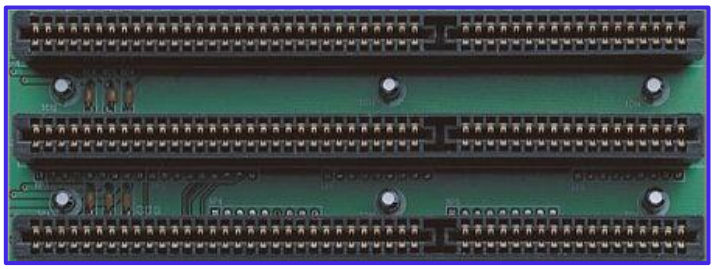


Figure 1: Ports ISA sur une carte mère. [2]

1. 2. 2 Le bus VLB (VESA Local Bus)

Développé par l'association VESA (Video Electronics Standards Association) en 1992, le bus VESA est consacré aux cartes graphiques. Il était aussi utilisé également par la carte d'entrée-sortie (connexion E-IDE, lecteur de disquette, ports, séries et parallèles). Ce type de bus est spécifique pour les microprocesseurs 486. Le bus de données VLB est sur 32 bits, le rôle de connecteur rajouté sert à reprendre directement les signaux de sortie d'un microprocesseur 486. La vitesse de l'extension est au début 33 MHz (la partie ISA reste 8 MHz), certaines versions sont venues et ont évolué jusqu'à 40 et 50 MHz. Le VLB a été remplacé par un nouveau type de bus appelé PCI pour les pentium utilisant un bus de données sur 64 bits. [3]

Figure2 : Ports VLB VESA sur une carte mère. [3]

1. 2. 3. Le bus PCI (Peripheral Component Interconnect)

On note ici que ce type de bus est utilisé lors de ce travail pour la connexion de la carte d'interface « ***QuanCom GPIB-PCI*** » avec un PC bureautique et c'est pour cela qu'on s'attardera plus sur les détails et le mode de fonctionnement de ce bus.

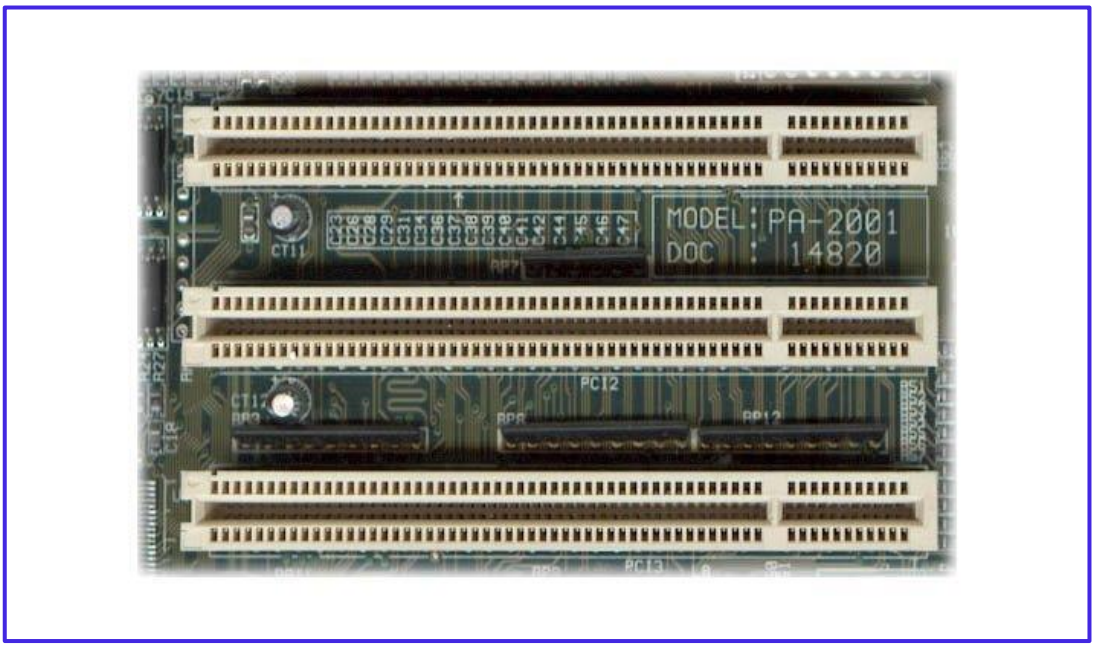


Figure3 : ports PCI sur une carte mère. [4]

Commençant d'abord par la structure du bus PCI :

1. 2. 3. 1 Structure du bus PCI

Le bus PCI est apparu au départ pour remplacer les vieux bus ISA et VLB. Il utilise une fréquence d'horloge de 33.33 MHz sur 32 bits de données (4 octets), donc sa bande passante est au maximum de 133 MB/s (33x4) (8 fois plus rapide que celle d'un ISA). On note aussi que le chipset 430 d'Intel est le premier à supporter ce bus. Le bus PCI est utilisé au début par la carte graphique (replacé depuis l'AGP vers le PCI et récemment par le PCI-Express) maintenant, il ne reste que les modem RTC (Réseaux Téléphoniques Commuté), cartes audio, réseaux… qui l'utilisent.

Le bus PCI utilise le DMA (Direct Memory Access) pour la transmission des données vers la RAM ou entre les cartes, sauf qu'il y a quelques limitations pour les chipsets vers le port AGP. Ce type de bus (PCI) a connu plusieurs versions qui ont essayé de le développer de plus en plus, ce sont : PCI-X 1.0 (sortie en 1999) ; PCI-X 2.0 (sortie en 2002) ; PCI-Express (1.1, 2.0 et 3.0).

La figure 4 représente un schéma électronique d'une structure du bus PCI, l'information (la fonction) représentée entre le processeur et le bus s'appelle le chipset gère le bus PCI et la mémoire, les autres bus comme ISA, SCSI, LAN … sont liés au bus PCI via un circuit qui sert de pont, les transactions qui se passent actuellement entre ISA et le processeur passent par le pont PCI.[4]

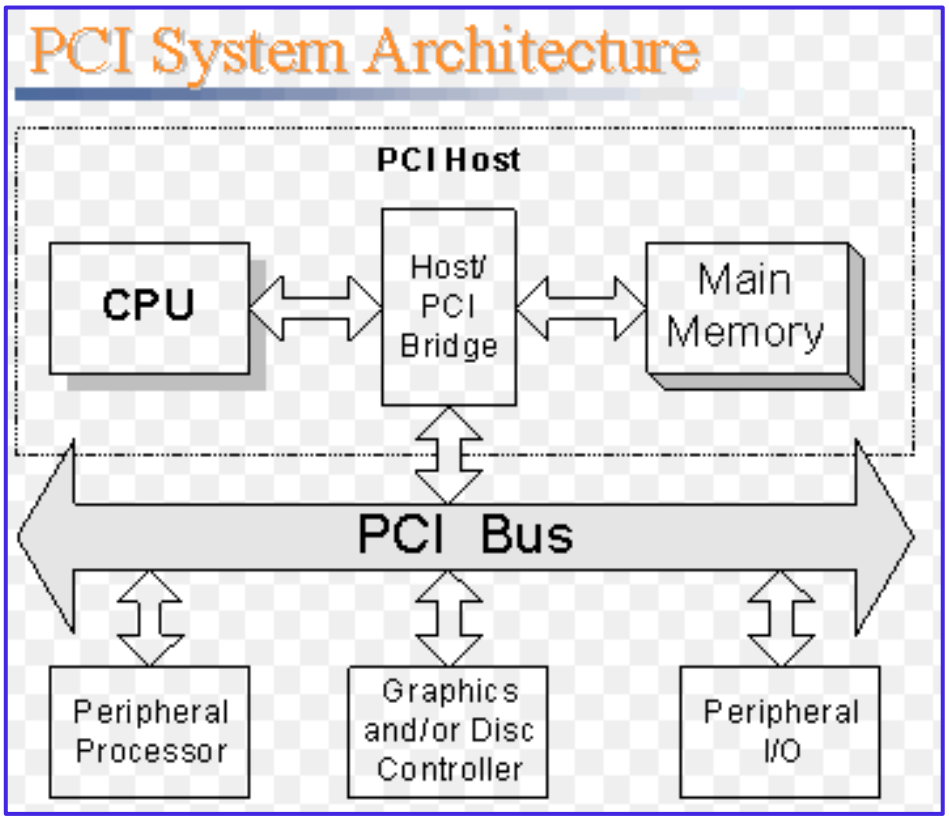


Figure4 : architecture du bus PCI. [5]

Le multiplexage des bus de données et d'adresse permet de réduire le nombre de broches sur le connecteur PCI, ce qui réduit les coûts et la taille des composants PCI. Les cartes d'extension PCI 32 bits classiques utilisent seulement environ 50 broches de signaux sur le connecteur PCI, dont 32 sont des bus d'adresses et de données multiplexées. [5]

Concernant le protocole de communication que le bus PCI utilise, on signale ici que ce protocole est très complexe et sort du cadre de notre travail et on se contente juste de signaler qu'un transfert de bus PCI consiste en une phase d'adresse et en un nombre quelconque de phases de données. Les opérations d'E / S qui accèdent aux registres dans les cibles PCI ne comportent qu'une seule phase de données dans la majorité des cas. Les transferts de mémoire qui déplacent des blocs de données consistent en plusieurs phases de données qui lisent ou écrivent plusieurs emplacements de mémoire consécutifs. L'initiateur et la cible peuvent « les deux » terminer une séquence de transfert de bus à tout moment. La figure suivante représente un diagramme de temps pour une transaction de lecture typique.

Pour ce qui est de la nature des signaux manipulés, on souligne le fait qu'Il existe 4 types des de signaux : on distingue ceux qui sont du système, d'adresses et de données, d'interface de contrôle, d'arbitration des conflits du bus, ainsi ceux d'interruptions. [6]

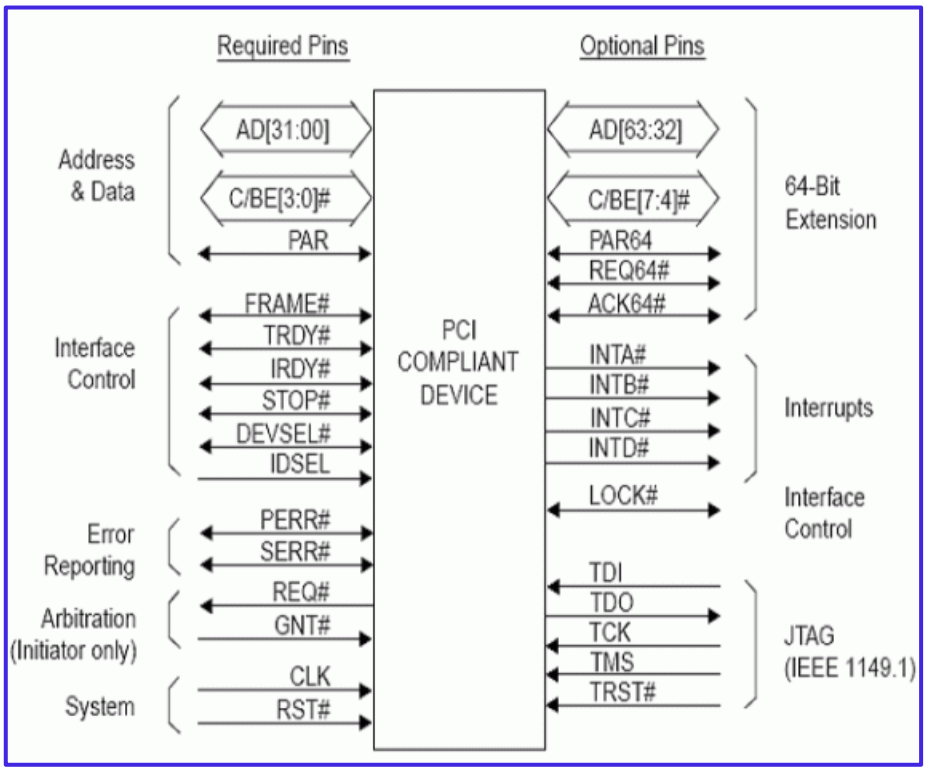


Figure 5: signaux définis dans le standard PCI. [6]

Au cours de cette section, nous avons passé en relief les différents bus d'extension d'un ordinateur bureautique. L'accent a été mise sur le bus PCI, mais nous rapportons ici que pour cerner tous les détails techniques nécessaires à la compréhension du mode fonctionnement de ce bus il nous aura fallu plusieurs pages d'explications rien que pour le protocole de communication. Et vu le fait que ça sort du cadre de notre travail, nous nous sommes volontairement limité à quelques idées principales et concepts essentiels directement impliqués dans le travail en cours.

Section 2 : Description de la carte d'interface « *QuanCom GPIB-PCI* »

La présente section sera consacrée à la description de la carte d'interface utilisée dans le travail décrit dans ce mémoire : **« *QuanCom GPIB-PCI* »**. On examinera plus particulièrement son mode d'insertion dans le PC, la description de son câble dédié et on finira par une description de quelques fonctions software utiles fournies avec la carte en tant que package software sous forme de CD ou software téléchargeable sur internet.

2.2 La carte d'interface (QUANCOM GPIB-PCI)

La carte standard QUANCOM GPIB a pour but essentiel de connecter des instruments de table avec l'interface connue IEEE 488 à l'ordinateur. Ce type de carte peut connecter jusqu'à 15 appareils par l'interface IEEE 488 au PC. L'échange des données est vérifié par un contrôleur intelligent uPD7210 de NEC, ce qui rend la fonction de la carte compatible avec la majorité (presque tous) des logiciels-pilotes IEEE courants.

Le type de la carte utilisée dans notre travail est : PCIGPIB, nous allons l'examiner de près et mettre l'accent sur le contrôleur GPIB qu'elle porte. [7]

2.3 PCIGPIB (Rev. 4.xx)

Figure 6.1: aperçu de la carte PCIGPIB. [13]

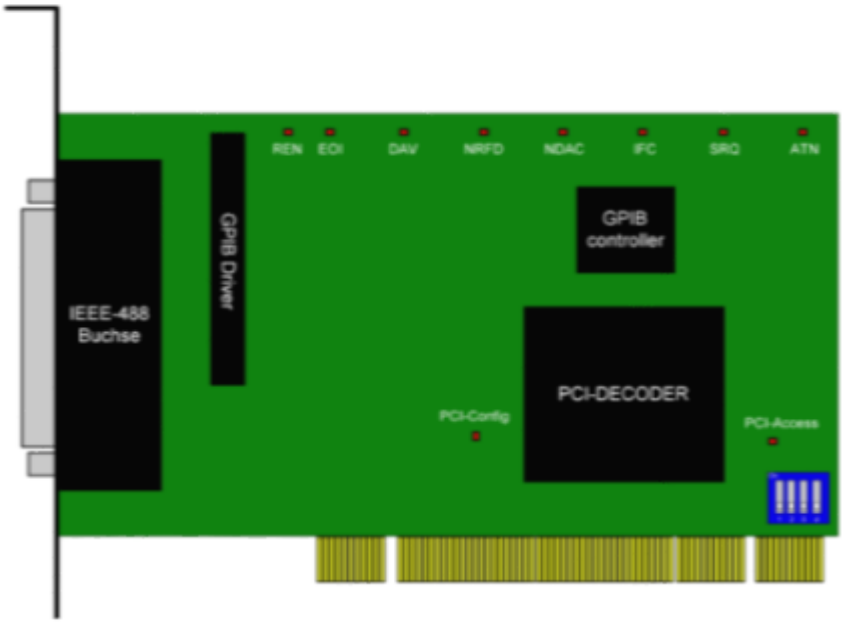


Figure 6.2: schéma représentant les composants de la carte PCIGPIB. [7]

Un extrait de la fiche technique de la carte PCIGPIB est fourni dans ce qui suit :

- **Contrôleur :** µPD7210 compatible avec HW FIFO
- **Connecteur** :24 pins. Prise IEEE-488 avec fil M3.5.
- **Diodes électroluminescentes** : 8 LED sont sur la carte pour l'annonce des lignes de contrôle ;ATN, SRQ, NDAC, NRFD, DAV, IFC, REN, EOI. 2 LED indiquent aussi l'état du PCI (configuration status) et PCI-Accès (access).
- **Programmation** : souvent en C / C ++, Visual Basic, Lotus NotesLabWindows et Delphi.

D'un point de vue hardware, la carte est principalement constituée de deux circuits intégrés sous forme d'ASIC. Un premier circuit intégré qui s'occupe de d'interface du bus PCI entre la carte et le PC et un autre qui, le µPD7210, qui se charge de l'interface GPIB entre carte et l'ensemble des instruments externes. Dans ce qui suit, nous allons nous pencher sur les caractéristiques techniques de ce dernier et ce vu son importance pour le travail à réaliser.

2.3.1 Le contrôleur de la carte PCIGPIB

Le **NAT7210** de National Instruments est un composant de remplacement DIP à 40 broches pour le NEC µPD7210. Le NAT7210 est compatible à 100% avec le NEC **µPD7210** à la mise sous tension et possède des fonctionnalités supplémentaires sur la puce du contrôleur NAT4882 IEEE 488.2. Ainsi, le NAT7210 peut exécuter toutes les fonctions d'interface définies par la norme ANSI / IEEE 488.1-1987 et répond aux exigences et recommandations de la norme ANSI

/ IEEE 488.2-1987. Le NAT7210 exécute des fonctions complètes du parleur, de d'écouteur et de contrôleur IEEE 488. [8]

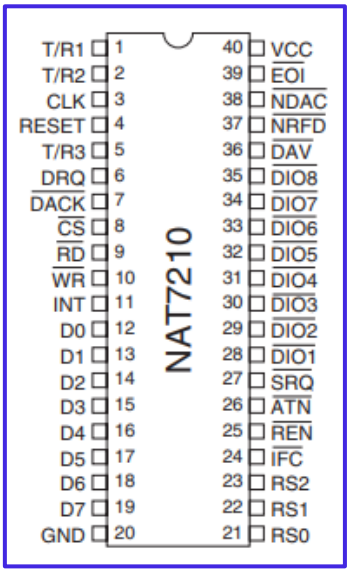


Figure 7 : Pins du NAT7210. [6]

Le tableau suivant décrit les broches NAT7210.

Numéro du pin	Code (mnémonique)	Type	Description
19, 18, 17, 16, 15, 14, 13, 12	D(7-0)	I/O†	Le bus de données bidirectionnel à 3 états transfère les commandes, les données et l'état entre 15, 14, 13, 12 NAT7210 et la CPU
23,22,21	RS(2-0)	I†	Le registre sélectionne le registre auquel accéder lors d'une opération de lecture ou d'écriture
8	CS*	I	Chip select donne accès au registre sélectionné par une opération de lecture ou d'écriture et le registre sélectionne RS (2-0)
9	RD*	I	Avec Read Input, vous pouvez placer le contenu du registre sélectionné par RS (2-0) et CS * sur le bus de données D (7-0)
10	WR	I †	L'entrée d'écriture verrouille le contenu du bus de données D (7-0) dans le registre que RS (2-0) sélectionne
7	DACK*	I †i	Le signal d'accusé de réception DMA sélectionne le DIR ou le CDOR pour le cycle de lecture ou d'écriture en cours.

6	DRQ	O	La sortie de la demande DMA s'affirme pour demander un cycle de knowledge DMA.
3	CLK	I	L'entrée CLK peut aller jusqu'à 20 MHz.
4	RESET	I	L'affectation de l'entrée de réinitialisation place le NAT7210 dans un état initial d'inactivité
11	INT	O†	La sortie d'interruption indique que l'une des conditions d'interruption de masquage est **vraie**
24	IFC*	I/O†	La ligne de contrôle bidirectionnelle initialise les fonctions de l'interface IEEE 488
25	REN*	I/O†	La ligne de contrôle bidirectionnelle sélectionne le contrôle à distance ou local des périphériques
26	ATN*	I/O†	La ligne de contrôle bidirectionnelle indique si les données sur les lignes DIO sont une interface ou un message dépendant du périphérique
27	SRQ*	I/O†	Ligne de contrôle bidirectionnelle demande le service du contrôleur
15, 14, 13, 12 31, 30, 29, 28	DIO(8-1)*	I/O†	Bus de données IEEE 488 bidirectionnel 8 bits
36	DAV*	I/O†	La ligne de poignée de main (handshake) indique que les données sur les lignes DIO (8-1) * sont valides
37	NRFD*	I/O†	La ligne de poignée de main indique que l'appareil est prêt à recevoir des données.
38	NDAC*	I/O†	La ligne de poignée de main indique la fin de la réception du message.
39	EOI*	I/O†	La ligne de contrôle bidirectionnelle indique le dernier octet d'un message de données ou exécute un polling parallèle.
1	T/R1	O	Talk Enable contrôle la direction de l'émetteur-récepteur de données IEEE 488
5	T/R3	O	Ces broches sont pour le contrôle d'entrée / sortie pour les émetteurs-récepteurs IEEE 488
2	T/R2		
40	VCC	-	Broche d'alimentation +5 V (± 5%)
20	GND	-	Broche de terre - 0 V

Tableau 1 : paramètres des broches du NAT7210. [12]

2.3.2 Configuration d'identifiant de la carte via les DIP- switch

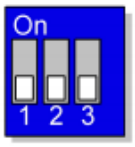
On sait que pour être connecté à un bus GPIB, Chaque équipement (*Talker* ou *listner*) faisant partie du réseau doit posséder une adresse propre à lui comprise entre (0-15). Et pour ce fait la carte est munie d'un outil de fixation d'adresse qui se résume en un ensemble de DIP Switch configurable manuellement

SW1	SW2	SW3	Card ID
Off	Off	Off	0 (default)
On	Off	Off	1
Off	On	Off	2
On	On	Off	3
Off	Off	On	4
On	Off	On	5
Off	On	On	6
On	On	On	7

Figure 8 : configuration d'identifiant de la carte [9]

Le tableau ci-dessus, montre les combinaisons possibles qu'on peut effectuer pour le choix de l'adresse « l'ID » de la carte à travers les « Dip-switch » prévus à cet effet sur la carte.

L'exemple suivant illustre l'ID de carte 0 et 2 : [9]



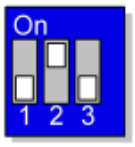


Figure 9.a : Adresse 0[9]

Figure 9.b : Adresse 2 [9]

La configuration de l'E/S et du mode mémoire est avec le Dip-switch 4 :

Un Dip-Switch est également disponible sur la carte et peut être utilisé pour le choix entre les modes E/S ou mode mémoire comme suit :

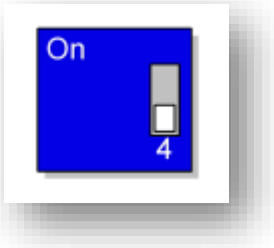


Figure 9.c: réglage du dip-switch 4 [9]

ON = Mode entrées/sorties

Off = Mode mémoire

2.4 Description et affectation des broches du connecteur GPIB

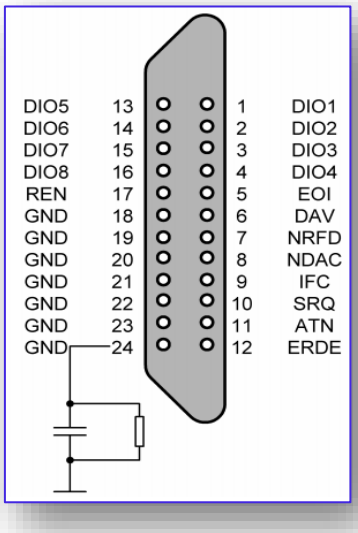


Figure 10.a : photo réelle du connecteur [13]. **Figure10.b**: affectation des broches du connecteur [8]

2.5 Procédures de montage et d'installation de la carte PCIGPIB dans un PC bureautique

- **Exigences du système**

Ordinateur personnel (PC), Les cartes QUANCOM sont destinées à fonctionner dans des ordinateurs IBM-AT compatibles avec 80X86 (c'est-à-dire 80386/80486 / Pentium et plus)

L'ordinateur doit être muni d'un bus d'extension de type (PCI).

- **Montage de la carte PCI / GPIB**

Pour réussir le montage de la carte, nous avons suivi les étapes suivantes :

1. En premier lieu, nous avons éteint l'ordinateur et les périphériques connectés du fait que l'électricité statique peut détruire soit ordinateur soit la carte ou les deux à la fois
2. Par la suite, on a ouvert le PC et on a détaché les quatre vis à l'arrière de du boîtier de l'unité centrale, ensuite, on a tiré la couverture vers l'avant et retiré les câbles gênants.
3. Les fentes d'emplacement de la carte sont positionnées à l'arrière de l'ordinateur. Les fentes non utilisées sont recouvertes d'un petit métal assiette. Nous avons recherché une fente libre, détaché sa vis de fixation et retiré la petite plaque de métal qui lui appartient.

4. Une fois arrivé à ce stade, nous avons positionné la carte d'extension, i. e. la carte GPIB-PCI dans une fente libre et Vérifié que la carte est fermement insérée dans la fente.
5. Comme mesure de sécurité, on a Fixé la carte avec la vis de la petite plaque métallique sur la paroi arrière.
6. Après, on a fermé le couvercle de notre unité centrale (ordinateur), reconnecté les câbles qu'on a détachés pendant l'installation.
7. A l'étape finale, on Connecte le câble GPIB de la carte dans l'emplacement correspondant.

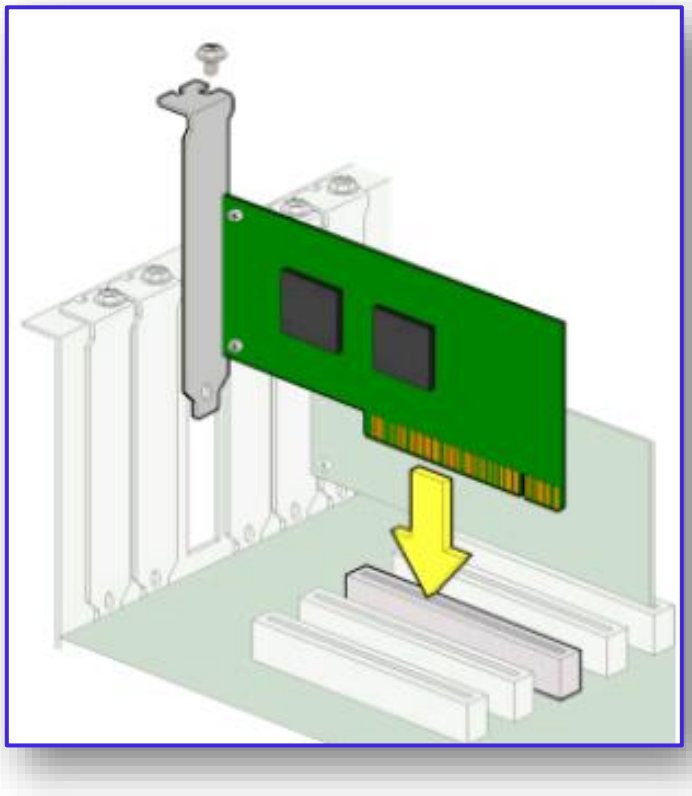


Figure 11.a: placement de la carte ans un bus PCIGPIB [9]

Figure 11.b: placement de la carte ans un bus PCIGPIB. [13]

Figure11.c : placement de la carte ans un bus PCIGPIB. [13]

2.5.3 Outils de programmation et packagea software de la carte d'interface GPIB-PCI

La carte « *QuanCom GPIB-PCI* » est fournie avec tant d'accessoires tel que le câble et une le package software sous forme de bibliothèques contenant la déclaration et les fonctions utilisées pour la programmation et la configuration de la carte. Le package en question : « Le QLIB » (*abréviation de QUANCOM LIBrary*) donne la possibilité d'adresser toutes les cartes QUANCOM sous les systèmes d'exploitation Windows XP / 2000 / NT et ME / 98/95 et les langages de programmation C / C ++ / Delphi / Visual Basic. Il est fourni et autorisé à toutes les cartes QUANCOM par la simplicité des instructions à l'utilisateur de fusionner le QLIB dans ses propres applications. Les instructions et les fonctions s'appliquent à tous les systèmes d'exploitation.

- **Systèmes d'exploitation adaptés :**

Le QLIB est compatible avec : Microsoft Windows Vista / XP / 2000 / NT 4.0 / ME / 98 / 95 und Linux. Des versions récemment sorties sont compatibles avec Microsoft Windows 7, 8, 8.1, 10.

- **Compilateurs adaptés :**

C / C++ : Borland C++ 3.1, 4.x, 5.x, Microsoft® Visual C++ 1.x, 2.x, 4.x, 5.x, 6.x

- **Pascal**: Borland Turbo Pascal.
- **Delphi**: Borland Delphi.
- **Basic**: Microsoft® Visual Basic 3.x, 4.x, 5.x; 6.x.
- **Langages de programmation graphique:** Agilent-VEE de Agilent, LabView® de National Instruments.

2.6 Fonctions fournies dans la bibliothèque

- **QAPIExtOpenCard()**

La fonction *QAPIExtOpenCard()* ouvre la communication avec la carte et récupére le descripteur de la carte (*handle*).

Syntaxe : ULONG *QAPIExtOpenCard* (ULONG cardid, ULONG devnum);

Paramètre 1 : cardid : définit l'identifiant de la carte à partir duquel les informations pourraient être reçues.

Paramètre 2 : devnum : définit le numéro de la carte qui doit être ouverte.

- **QAPIEXTCloseCard()**

Cette fonction réalise la tâche inverse réalisée par la fonction précédente. La communication avec la carte sera fermée après l'exécution de cette fonction.

Syntaxe:void *QAPIExtCloseCard*(ULONGcardhandle);

Paramètre 1: cardhandle: définit le handle de la carte ouverte.

- **QAPINumOfCards()**

Avec la fonction QAPINumOfCards, il est possible d'avoir le nombre des cartes utilisées etsupportées par la QLIB.

Syntaxe : ULONG *QAPINumOfCards* (void);

La valeur renvoyée est le nombre de cartes supportées par le QLIB.

- **QAPIExtSpecial()**

Avec cette fonction, il est possible d'exécuter des opérations spécifiques à la carte.

Syntaxe : ULONG *QAPIExtSpecial*(ULONG cardhandle, ULONG jobcode, ULONG para1, ULONG para2);

Paramètre 1 : cardhandle: définit le handle de la carte.

Paramètre 2 :jobcode: définit l'action qui s'exécute sur la carte.

Paramètre 3 : para1 : définit le canal à partir duquel les fichiers doivent être lus.

Paramètre 4 : para2 : Définit le canal à partir duquel les fichiers doivent être lus.

Il existe 13 codes spéciaux pour le QLIB. Chaque code a un effet sur le fonctionnement de la carte, parmi ces codes les plus utilisés :

- JOB_RESET : réinitialise la carte.
- JOB_REN : met la carte e mode remote.
- JOB_DCL : met tous les périphériques branchés au bus à un état prédéfini sauf les périphériques en mode LLO.
- JOB_SDC : met le périphérique adressé à un état prédéfini (par default).
- JOB_GET : exécute une action en mode trigger.
- JOB_LLO : verrouille le panneau avant de l'appareil, après l'envoi de ce code, l'utilisateur ne peut plus modifier les paramètres de l'instrument de son panneau avant.

Les détails de tous ces codes seront mis dans la troisième section de la partie théorique (*Caractéristiques du bus GPIB*).

- **QAPIExtWriteString()**

La fonction QAPIExtWriteString envoie une chaîne de caractère à un périphérique GPIB.

Syntaxe:

ULONG QAPIExtWriteString (ULONG cardhandle, ULONG device, char* buffer, ULONG chars, ULONG reserved);

Paramètre 1 : cardhandle : un descripteur précédement ouvert pour la carte.

Paramètre 2 : device : ce paramètre contient l'adresse de l'écouteur d'un périphérique sur le bus GPIB.

Paramètre 3 : buffer : un pointeur vers un buffer qui contient la chaîne (string) qui doit être envoyée au périphérique.

Paramètre 4 : char :ce paramètre indique la taille en octets du buffer.

Paramètre 5 : reserved : paramètre pour une utilisation future. Ce paramètre devrait être zéro.

La valeur renvoyée après l'exécution de cette fonction est : 1 si elle est réussie, 0 si elle échoue.

- **QAPIExtReadString()**

La fonction *QAPIExtReadString* lit ou reçoit une chaîne de caractère à partir d'un périphérique GPIB.

Syntaxe:

ULONG *QAPIExtReadString*(*ULONG cardhandle, ULONG device, char* buffer, ULONG chars, ULONG reserved*);

Paramètre 1 : cardhandle : un descripteur précédement ouvert pour la carte.

Paramètre 2 : device : ce paramètre contient l'adresse de l'écouteur d'un périphérique sur le bus GPIB.

Paramètre 3 : buffer : un pointeur vers un buffer qui contient la chaîne (string) qui doit être envoyée au périphérique.

Paramètre 4 : char :ce paramètre indique la taille en octets du buffer.

Paramètre 5 : reserved : paramètre pour une utilisation future. Ce paramètre devrait être zéro.

La valeur renvoyée après l'exécution de cette fonction est : 1 si elle est réussie, 0 si elle échoue.

- **QAPIGetLastError()**

La fonction *QAPIGetLastError()* récupère la valeur du dernier code d'erreur du thread appelant. Le code de la dernière erreur est maintenu sur une base par le thread. Plusieurs threads ne remplacent pas le dernier code d'erreur de l'autre.

Syntaxe : ULONG *QAPIGetLastError(void);*

Dans cette section, nous avons fait une présentation plus au moins détaillée de la carte d'interface GPIB. Nous avons essayé d'examiner les deux aspects hardware et software à la fois. Arrivés à un tel stade, il parait que nous avons acquis le lot d'informations nécessaire pour entamer le travail pratique qui fera le sujet de la prochaine partie.

Section 3: Le bus de communication GPIB :

La troisième section de la partie théorique contient un bref historique du bus GPIB, sa structure générale, ses fonctions d'interface et ses caractéristiques. Nous essayerons également de présenter le protocole de communication et le mode de fonctionnement de la norme IEEE 488.1 ainsi que ses différentes commandes.

3.1 Historique

Le bus IEEE-488 a été développé par la société HEWLETT PACKARD en 1965 sous le nom de bus HPIB (HEWLETT PACKARD I NTERFACE BUS). Ce bus avait pour but d'interconnecter les différents instruments de mesures (calculateurs…) pour réaliser des bancs de tests ou de mesures automatiques. En 1972, l'idée de réalisation d'un bus normalisé HPIB est venue de la part de l'IEC (Institut of Electrotechnical Commission). En 1975, le bus a été normalisé (IEEE 488). Cette norme a été révisée en 1978 puis approuvée en 1987 sous la référence IEEE 488-1 par l'IEEE (INSTITUTE of ELECTRICAL and ElectronicEngineers). La même année la version IEEE 488-2 était également adoptée.

Cette norme (IEEE 488.1) est connue aussi sous les noms : ***GPIB*** (**G**eneral **P**urpose**I**nterface **B**us). -IEC 625 (International Electric Commission). En 1982, la norme a été révisée (CEI-625-2) ensuite en 1988 (IEEE-488.2) minimise le manque de précision dans le but de standardiser le format des données. Ceci pour garantir une bonne transmission des messages mais n'implique pas qu'ils soient parfaitement compris par tous les appareils. Beaucoup d'applications d'instrumentation automatisée utilisent encore le bus IEEE.488 malgré l'apparition d'une nouvelle instrumentation « *low-cost* » à base de bus USB ou de réseau Ethernet, mais ces derniers ne sont pas déterministes.

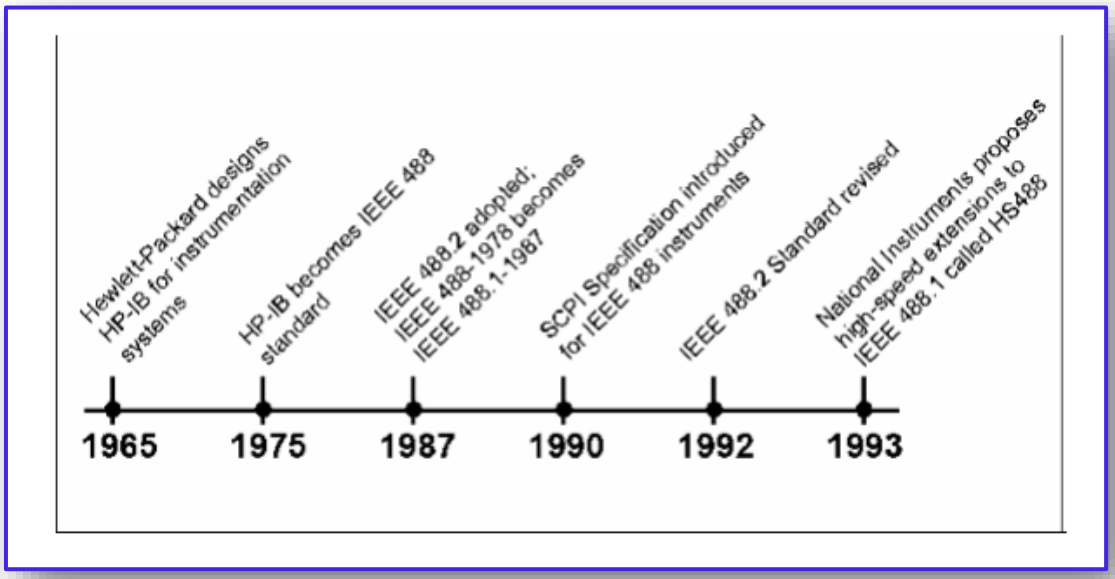


Figure 12 : Evolution du bus GPIB. [8]

2.3 Structure générale du bus GPIB

3.3.1 Câbles de liaison

Le bus se matérialise par un ensemble de protocoles et de câbles qui relient les appareils entre eux. Chaque câble est lié à chacune de ses extrémités d'un connecteur mâle et d'un connecteur femelle emboités et équipés de vis imperdables.

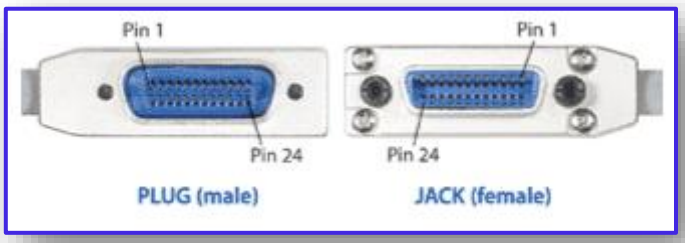


Figure 13.a : types de connecteurs du câble GPIB. [8]

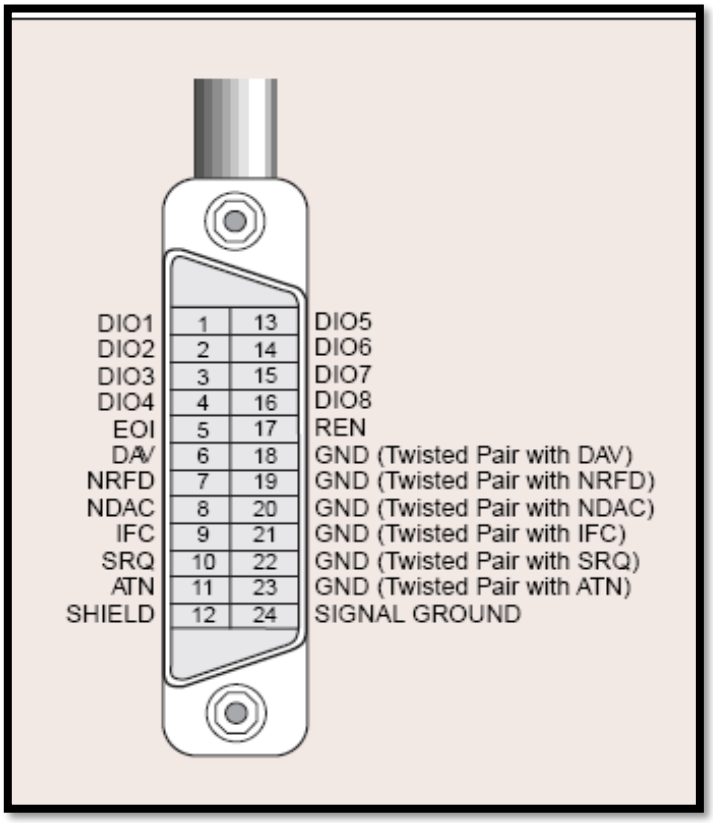


Figure 13.b : affectation des broches du connecteur GPIB. [8]

Figure 13.c : photo réelle des connecteurs du câble GPIB. [13]

3.3.2 Interconnexion des instruments sur le bus

L'utilisateur peut interconnecter ses appareils en **trois** manières différentes : étoile, chaine (lineaire) ou réaliser une combinaison des deux. Le nombre maximum d'appareils sur le bus ne doit pas dépasser **15**, la longueur totale du câble ne doit pas dépasser **2mètres** multipliés fois le

nombre d'appareils chainés, avec une longueur maximale de **20mètres**. Il ne faut jamais enficher plus de **quatre** câbles sur la même prise de châssis.

Le transfert des informations peut arriver théoriquement à une vitesse de 1,5 Mo/s (IEEE 488.1) et 8 Mo en mode HS 488.

Les figures suivantes expliquent les types du câblage des appareils dur le bus.

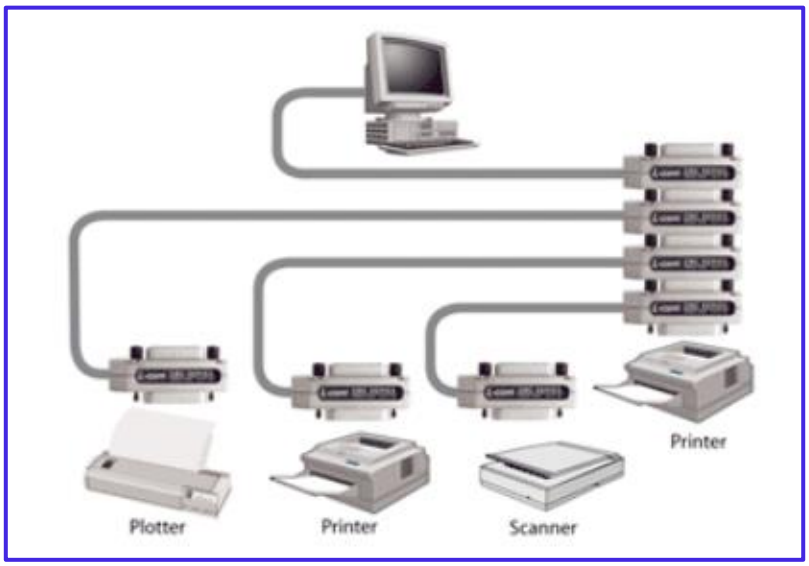


Figure 14.a : schéma explicatif du branchement des appareils sur le bus GPIB. [8]

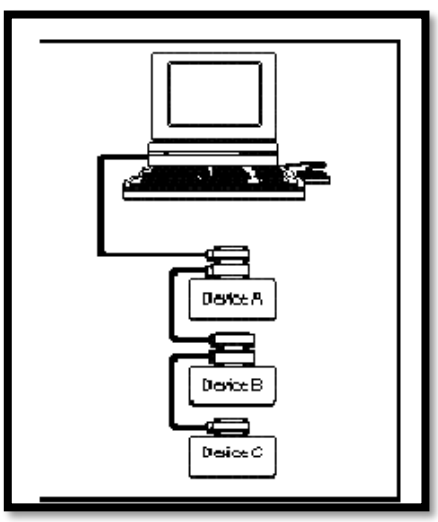


Figure 14.b : câblage linéaire. [8]

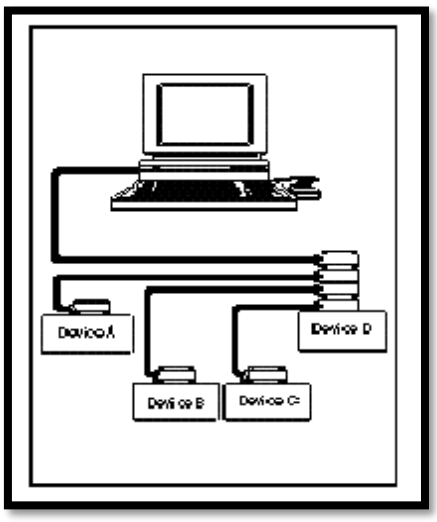


Figure14.c : câblage étoile. [8]

3.4 Fonctions d'interface

Chaque instrument connecté au buscorrespondant à la norme IEEE 488.1 possède une interface qui satisfait une ou plusieurs spécifications suivantes :

- **Récepteur (LISTENER ou écouteur) :** c'est un dispositif capable de recevoir des informations quand il est adressé par le système. Il est capable de lire les données qui circulent sur le bus de données lorsque le contrôleur lui en donne l'autorisation.
- Exemples de ce type des appareils : les imprimantes, systèmes d'affichage, alimentation programmable et tout instrument programmable.
- **Emetteur (TALKER ou parleur) :** c'est un dispositif capable d'envoyer des informations lorsque le système l'adresse. Il est capable d'envoyer des données sur le bus lorsque le contrôleur lui en donne l'ordre.
- Exemples de ce type d'appareils : les voltmètres, les systèmes d'acquisitions et tout instrument programmable.
- **Contrôleur (controller) :** c'est un appareil qui assure le contrôle du bus, il peut définir parmi les appareils liés au bus ceux qui écoutent et ceux qui parlent**,** souvent c'est un ordinateur équipé d'une interface IEEE.
- Ils se peut retrouver plusieurs contrôleurs dans un bus mais un seul qui en a la gestion.

3.5 Caractéristiques du bus GPIB

Les fils utilisés par le bus sont classifiés en 3 catégories de lignes :

- **Les lignes de données : (data lines) 8** lignes.
- **Les lignes de contrôle du flot d'informations : (handshake lines) 3** lignes.
- **Les lignes de gestion de bus : (general bus management) 5** lignes.

Le bus (les fils du bus) transporte ces types de signaux vers les appareils adressés.

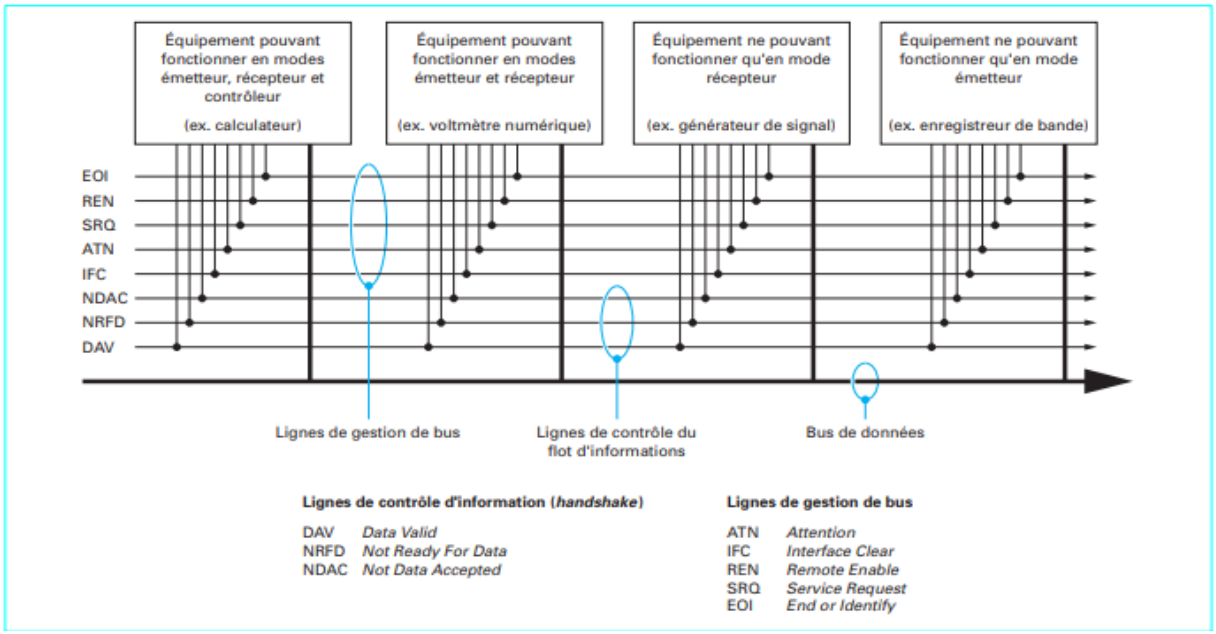


Figure15 : Fils utilisés par le bus GPIB. [11]

Le bus IEEE 488.1 utilise une logique négative (vraie à l'état bas), ce qui permet aux constructeurs de mettre en œuvre la technique du « OU câblé » nécessaire à la gestion des signaux NRFD et NDAC.

3.5.1 Les signaux de gestion de bus (commandes)

- **ATN(ATtentioN)** : Activé par le contrôleur, il indique que les données sur le bus correspondent à une adresse ou à une commande. ATN est actif (true) à l'état bas.
- **Lorsque ATN=0 (true)** : tous les équipements deviennent récepteurs et participent à la communication, il informe qu'une donnée se trouve sur le bus.

  Lorsque ATN=1 (false) : il informe qu'une commande IEEE se trouve sur le bus.
- **IFC (InterFace Clear)** : activé par le contrôleur. Il permet de réinitialiser tous les appareils connectés au bus, permet aussi au contrôleur d'arrêter à tout instant, de manière asynchrone, l'opération en cours sur le bus.
- **REN (Remote Enable)** : activé par le contrôleur, il place les appareils "écouteurs" en mode télécommandé. Ce signal est géré par le contrôleur, il force chaque appareil concerné à être piloté dans le bus. A l'état bas (true) : chaque récepteur adressé et a la possibilité d'être piloté passe en mode « remote », à l'état haut (false) :l'appareil passe en mode local et valide les commandes de son panneau frontal.
- **SRQ (Service ReQuest)** : activé par n'importe quel appareil sur le bus écouteur ou parleur (sauf le contrôleur). Il se comporte comme une demande d'interruption.

  L'utilisation de ce signal est pour prévenir le contrôleur d'une disponibilité de donnée ou bien d'avertir d'un problème quelconque sur l'appareil (décalibration, lecture erronée …).
- **EOI (End Or Identify) :** lorsque EOI activé, seul il signifie une fin de message et est positionné en même temps que le dernier octet du message

  Le signal de cette ligne est combiné avec le signal ATN, il signifie un début d'identification.

3.5.2 Les signaux de contrôle du flot d'informations (synchronisation)

- **DAV (DAtaValid) :** activé par un parleur, si DAV est à l'état bas (true), les données sur le bus peuvent être acceptées par tous les appareils récepteurs, tous les appareils parleurs peuvent agir sur ce signal,le parleur met DAV à l'état haut (false) lorsque tous les écouteurs concernés ont accepté la donnée.
- **NRFD (Not Ready For Data) :** piloté (activé) par les écouteurs, il informe le parleur e si l'appareil peut ou ne peut pas accepter la donnée.

Un appareil qui refuse les informations met le signal NRFD à l'état bas (true) ensuite, il autorise sa remontée à l'état haut (false) lorsqu'il peut accepter des données.Le signal sera à l'état haut dès que tous les récepteurs peuvent effectivement recevoir des informations.

- **NDAC (Not Data ACcepted) :** piloté (activé) par les écouteurs, il indique si l'appareil accepte ou refuse la donnée reçue.La ligne NDAC remonte à l'état haut après l'acceptation des données du dernier et plus lent des récepteurs.

3.5.3 Les signaux de données (Data lines)

8 lignes de DI/O1 à DI/O8, ils suivent l'état de signal ATN les données transitant sur le bus peuvent être :

- des adresses ou des commandes multilignes (ATN activé) : caractères ASCII

- des données ou des octets d'états (status byte).

3.6 Protocole de communication

La figure 16 représente le contrôle des données sur le bus et la gestion de ce dernier via les lignes de synchronisation (*handshake lines*).

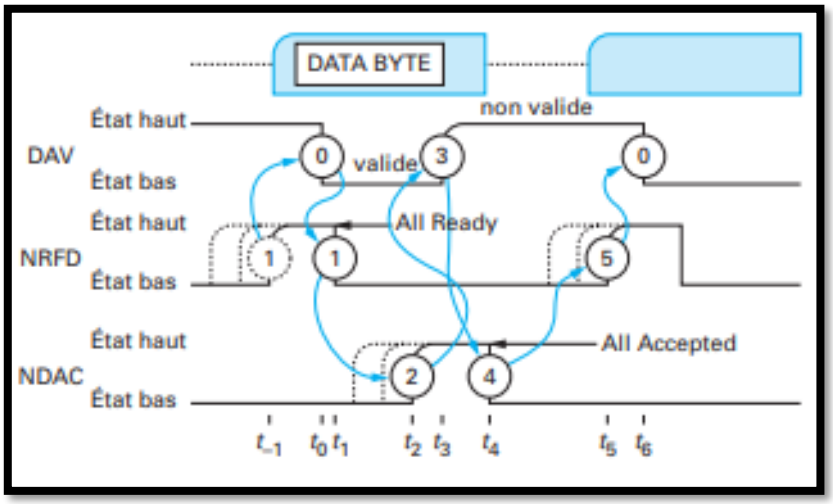


Figure 16 : protocole de communication (le handshake)[11]

t_{-1} : tous les récepteurs sont prêts. NRFD passe à l'état haut avec l'appareil le plus lent.

t_0 : la source valide ses données, DAV à l'état bas.

t_1 : le premier récepteur qui accepte l'octet signale, met NRFD à l'état bas, qu'il ne peut assimiler de nouvelles données.

t_2: NDAC monte au niveau haut avec le récepteur le plus lent et indique que tous les récepteurs ont accepté la donnée.

t_3 : la source positionne la ligne DAV à l'état haut, indiquant que la donnée n'est plus valide.

t_4: le premier récepteur passe la ligne NDAC au niveau bas en préparation du cycle suivant.

t_5 : retour à t –1 avec un nouvel octet sur le bus (nouveau cycle).[11]

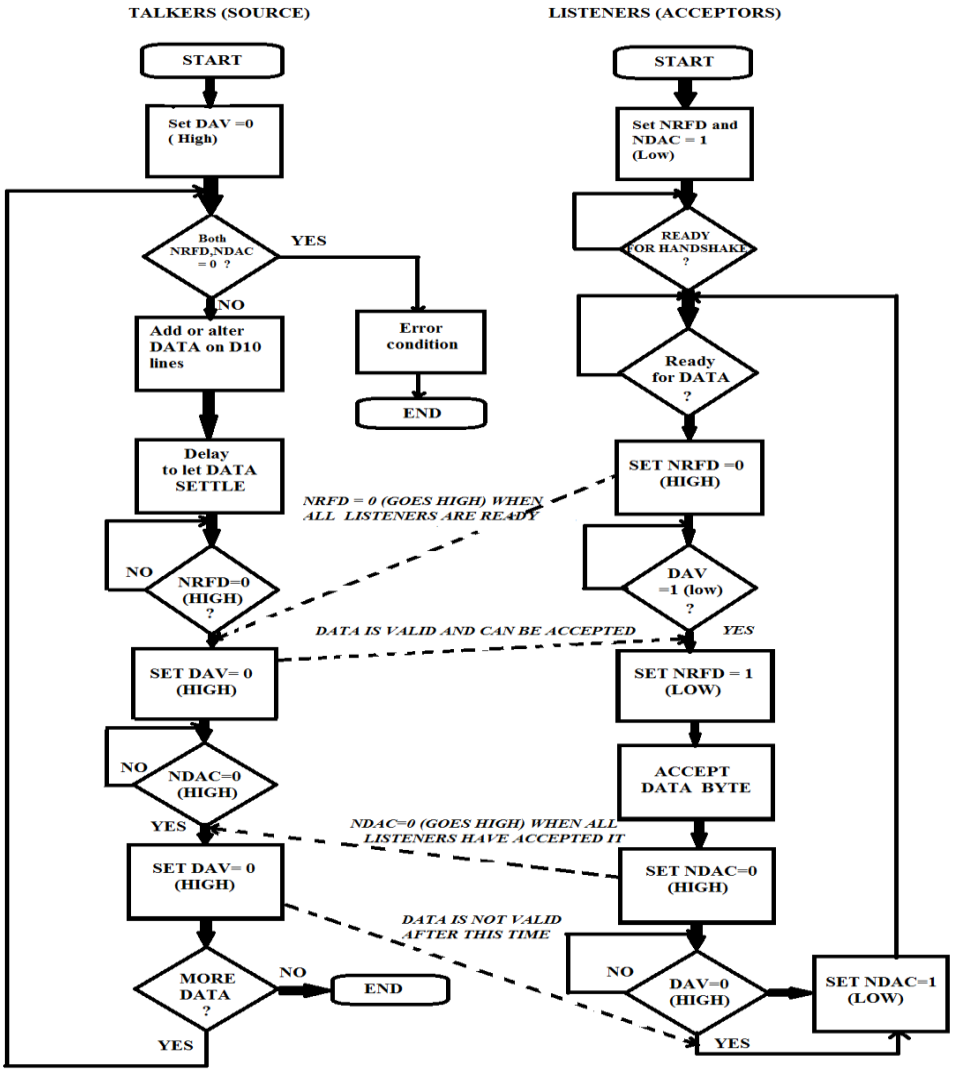


Figure17 : protocole de communication.[10]

La communication entre le parleur et l'écouteur se fait comme suit :

Au début : le parleur met la ligne DAV au niveau haut, et les écouteurs mettent NRFD et NDAC à l'état bas, ensuite le parleur vérifie la condition si NRFD et NDAC sont à l'état haut, si OUI → ERREUR→ Fin, SI NON il met les données sur les lignes.

Pour les écouteurs, ils doivent vérifier est ce qu'ils sont prêts à accepter les données, si oui, ils mettent NRFD au niveau haut, sinon ils attendent jusqu'à ce qu'ils soient prêts.

Revenant au parleur, il vérifie si le signal NRFD est au niveau haut ? Si oui, il met DAV en état bas (donnée valide et peut être acceptée par tous les écouteurs), sinon il attend le remis en haut du signal NRFD par les écouteurs.

Pour les écouteurs : ils vérifient si le signal DAV est au niveau bas ? Si oui, ils acceptent la donnée ce qui expliquent que le parleur a validé la donnée et peut être acceptées en toute sécurité par les écouteurs.

Si non, ils attendent le remis en bas du signal DAV (la validation de la donnée) par le parleur.

Revenant au parleur, il va vérifier si le signal de la ligne NDAC est en état haut ? Si oui, il met le signal DAV en état haut (les données sur le bus ne sont pas prises en charge), si Non, il attend le remis en haut par les écouteurs pour que les écouteurs puissent accepter la donnée.

Pour les écouteurs, ils mettent les signaux NRFD au niveau bas et NDAC au niveau haut, sachant que le signal NDAC reste en état haut jusqu'à ce que tous les écouteurs ont accepté la donnée.

Enfin, le parleur vérifie s'il y a encore d'autres données sur le bus, si oui, il refait le cycle, sinon il met fin. Pour les écouteurs, ils vérifient si le signal de la ligne DAV est remis en haut par le parleur, si non : ils attendent le parleur met DAV haut, si oui ils mettent NDAC au niveau bas et vont refaire le cycle. Après ce tems les données ne sont pas prises en considération.

3.7 Modes de fonctionnement du bus

Le bus IEEE 488.1 peut fonctionner en deux modes différents : commandes et données.

3.7.1 Mode commandes

ATN doit être réglé au niveau bas (true). Dans ce mode, le parleur envoie les ordres à tous les appareils récepteurs, ces commandes ont des particularités :

— les adresses d'émetteurs ou de récepteurs sélectionnent les appareils qui vont recevoir ou envoyer les informations sur le bus. Ces commandes s'appellent « multilignes »elles transitent via le bus de donnée, contrairement aux commandes « unilignes » elles n'utilisent un qu'un fil (IFC, REN,…).il existe 4 types commandes du bus IEEE488.1 : multilignes, unilignes, adressées et secondaires.

3.7.2 Mode données

ATN doit être réglé au niveau haut (false). Dans ce mode, les informations se transmettent via le bus d'un émetteur à un récepteur. Les appareils concernés seuls qui font le « handshake ».

3.8 Adressage du bus IEEE 488.1

Chaque appareil IEEE 488.1 possède au minimum une adresse d'émetteur ou de récepteur : MLA (MyListenerAddress) et MTA (MyTalkerAddress). Le contrôleur envoie ces mnémoniques (codes) suivies d'un chiffre de 0 à 30 dans le mode commandes dans le but de spécifier les appareils parleur et les appareils écouteur.

La sélection de l'adresse des appareils se fait sur 5 bits par le basculement des interrupteurs qui sont placées à l'arrière de l'instrument, ou bien en entrant l'adresse via les touches du panneau frontal.

Les 6ème et 7 ème bits font la distinction entre une adresse réceptrice ou émettrice. Le code d'adresse 31 est réservé, il est utilisé pour demander à chaque instrument de ne plus parler ou de ne plus écouter (UNTALK, caractère « – ») (UNLISTEN : caractère « ? »).

L'adresse "parleur" est obtenue en ajoutant 64 à l'adresse de base, tandis que l'adresse "écouteur" est obtenue en ajoutant 32 à l'adresse de base.

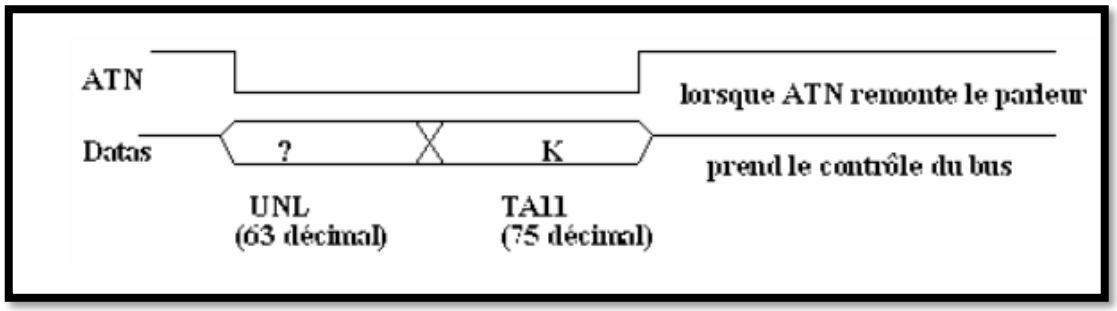


Figure18 : envoie de l'adresse 11 du parleur sur le bus. [8]

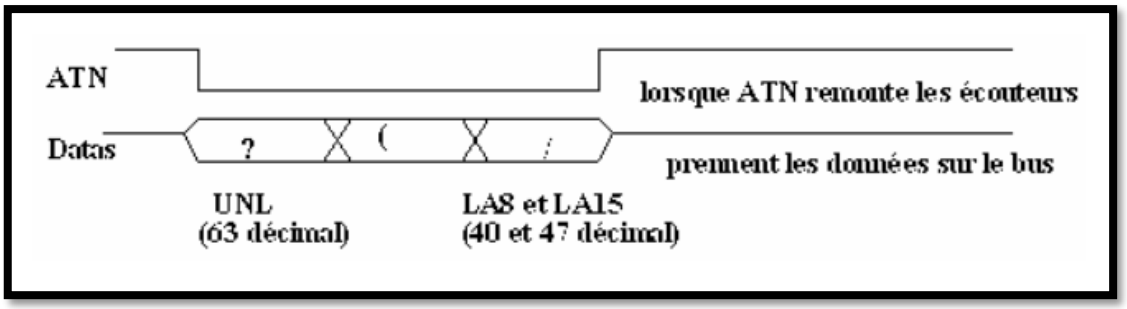


Figure 19 : les instruments écouteur aux adresses 8 et 15. [8]

La figure 22 contient la liste des adresses primaires du parleur MTA et écouteur MLA.

Bit Position 7 6 5 4 3 2 1 0
Meaning 0 TA LA GPIB Primary Address (range 0–30)

20	32	SP	MLA0	30	48	0	MLA16	40	64	@	MTA0	50	80	P	MTA16
21	33	!	MLA1	31	49	1	MLA17	41	65	A	MTA1	51	81	Q	MTA17
22	34	"	MLA2	32	50	2	MLA18	42	66	B	MTA2	52	82	R	MTA18
23	35	#	MLA3	33	51	3	MLA19	43	67	C	MTA3	53	83	S	MTA19
24	36	$	MLA4	34	52	4	MLA20	44	68	D	MTA4	54	84	T	MTA20
25	37	%	MLA5	35	53	5	MLA21	45	69	E	MTA5	55	85	U	MTA21
26	38	&	MLA6	36	54	6	MLA22	46	70	F	MTA6	56	86	V	MTA22
27	39	'	MLA7	37	55	7	MLA23	47	71	G	MTA7	57	87	W	MTA23
28	40	(	MLA8	38	56	8	MLA24	48	72	H	MTA8	58	88	X	MTA24
29	41	)	MLA9	39	57	9	MLA25	49	73	I	MTA9	59	89	Y	MTA25
2A	42	*	MLA10	3A	58	:	MLA26	4A	74	J	MTA10	5A	90	Z	MTA26
2B	43	+	MLA11	3B	59	;	MLA27	4B	75	K	MTA11	5B	91	[	MTA27
2C	44	,	MLA12	3C	60	<	MLA28	4C	76	L	MTA12	5C	92	\	MTA28
2D	45	-	MLA13	3D	61	=	MLA29	4D	77	M	MTA13	5D	93	]	MTA29
2E	46	.	MLA14	3E	62	>	MLA30	4E	78	N	MTA14	5E	94	^	MTA30
2F	47	/	MLA15					4F	79	O	MTA15				

Figure 20 : listes des adresses primaires MLA et MTA. [8]

3.9 Commandes du bus

Quatre types de commande du bus : les commandes universelles multilignes, unilignes, adressées et secondaires :

3.9.1 Commandes universelles multilignes

- **UNTalk(UNT)** ne parle pas : cette commande cesse d'adresser l'émetteur actuel, elle met les appareils dans le mode écoute. Sachant que le fait d'adresser un parleur repasse automatiquement tous les appareils en écouteur.
- **UNListen (UNL)** n'écoute pas : cette commande stoppe l'adressage des récepteurs à l'instant actuel. Un récepteur ne peut seul être invalidé, tous les appareils sont arrêtés.
- **DeviceCLear (DCL)** réinitialisation d'un instrument : tous les appareils qui acceptent cette commande retournent dans un état prédéfini (paramètres initiaux).

- **Local LockOut (LLO)** mode local verrouillé : le pilotage de l'instrument dépend exclusivement du bus IEEE, dans ce cas, le panneau frontal de l'appareil est verrouillé et ne peut plus reconfigurer ses paramètres à partir de son panneau frontal (avant).
- **Serial Poll Enable (SPE)** polling série validé : cette commande consiste à interroger un instrument par le biais d'un mot de huit bits, représentatif de son état. Lorsqu'un instrument reçoit l'ordre SPE, il retourne son mot d'état (status byte) au parleur.
- **Serial PollDisable (SPD**) polling série invalidé : c'est une suite logique de SPE, SPD termine l'interrogation d'un instrument en le repassant dans son état précédant le polling.
- **ParallelPollUnconfigure (PPU)** initialise tous les instruments qui ont subi la programmation pour un polling parallèle, dans leur état normal.

3.9.2 Commandes unilignes

Elles ont été abordées précédemment au paragraphe II.3.4.1.

3.9.3 Commandes adressées

- **Groupe Execute Trigger (GET)** déclenche une action : cette commande force l'instrument adressé à exécuter une séquence, exemple : actionner le trigger (déclencheur) interne d'un voltmètre, le balayage horizontal d'un oscilloscope…
- **Selecteddevice Clear (SDC)** réinitialisation de l'appareil adressé, elle retourne dans un mode prédéfini correspondant souvent à celui qui suit la mise sous tension.
- **Go To Local (GTL)** retourne en mode local : l'appareil se déconnecte du bus et autorise sa commande en déverrouillant son panneau frontal.
- **ParallelPoll Configure (PPC)** configure en polling parallèle : cette commande demande au récepteur adressé de se configurer pour un polling parallèle.
- **Take Control Talker (TCT)** prend le contrôle du bus : l'appareil adressé qui reçoit cette commande prend le contrôle du bus IEEE.

3.9.4 Commandes secondaires

- **ParallelPoll Enable (PPE)** polling parallèle validé : cette commande complète la commande PPC. Elle demande à l'instrument concerné de répondre au polling parallèle sur une ligne particulière du bus de données.
- **ParallelPollDisable (PPD)** polling parallèle invalidé : cette commande invalide l'appareil précédemment configuré en PPC.

3.10 Le polling série et le polling parallèle

- **Le polling série (serial poll)**

Le parleur est renseigné sur l'état qui se trouve un appareil de mesure (erreur de fonctionnement ou disponibilité de données), à travers ce polling série. L'appareil envoie un mot d'octets qui sera lu durant le polling. L'utilisation liée à la ligne SRQ (Service ReQuest) avec ce mot d'état sera effectuée par un appareil prêt, donnant l'exemple de transmission des données. Le parleur lance alors une recherche auprès de tous les appareils pour déterminer qui réclame la parole. Une fois le bon équipement détecté, son statut récupéré via le polling série informe le contrôleur que la lecture des données peut commencer.

- **Le polling parallèle**

Très utilisé dans des systèmes où acceptable de minimiser les temps morts, en identifiant d'une façon très rapide les appareils prêts à envoyer ou accepter des données. L'interrogation se fait de manière parallèle ce qui augmente la rapidité puisque, le protocole est simplifié, tous les instruments répondent en même temps. La réponse obtenue depend alors d'un seul et unique bit transféré par l'une des lignes du bus de données. Le parleur détermine rapidement qui réclame l'attention, mais ne peut obtenir le mot d'état qui motive la SRQ (demande de service) de l'équipement concerné. Les équipements qui supportent le polling parallèle offrent la possibilité (via des interrupteurs ou des cavaliers) de sélectionner la ligne ainsi que son niveau associé pour témoigner d'une demande de service. Le parleur peut faire cette tâche de polling par le biais de la commande PPC (ParallelPoll Configure).

3.11 Norme IEEE 488.2

Il existe une certaine perturbation entre les normes IEEE 488.1 et IEEE 488.2. En réalité, les deux normes visent des objectifs différents.

La norme IEEE 488.1 définit les caractéristiques électriques et mécaniques du GPIB (*General Purpse Interface Bus*). Des protocoles sont fournis pour mettre une communication entre un émetteur (talker) et un ou plusieurs récepteurs (*listeners*) et pour synchroniser le transfert des données (*handshake*).

Bien que le standard IEEE 488.2 définisse comment le transfert des données, il ne choisit pas quelles données doivent être envoyées. Le standard IEEE 488.2 définit le protocole de communication au niveau de l'échange des messages entre un parleur et un appareil.

En fait IEEE 488.2 a remplacé l'IEEE 728, ajoute une couche au-dessus de la précédente en décrivant un ensemble de codes et formats communs, un protocole de communication, et enfin un groupe de commandes utilisées couramment.

La norme IEEE-488.2 adopte donc des commandes communes : les appareils 488.2 supportent obligatoirement certaines d'entre elles. Cependant, cette définition laisse le choix libre aux fabricants de développer leur propre langage de programmation, qui freine ainsi les efforts fournis pour la compatibilité des instruments.

La norme SCPI (Standard Commands for Programmable Instruments) a résolu ce problème, en offrant une syntaxe performante, aisément transportable d'un équipement à un autre.

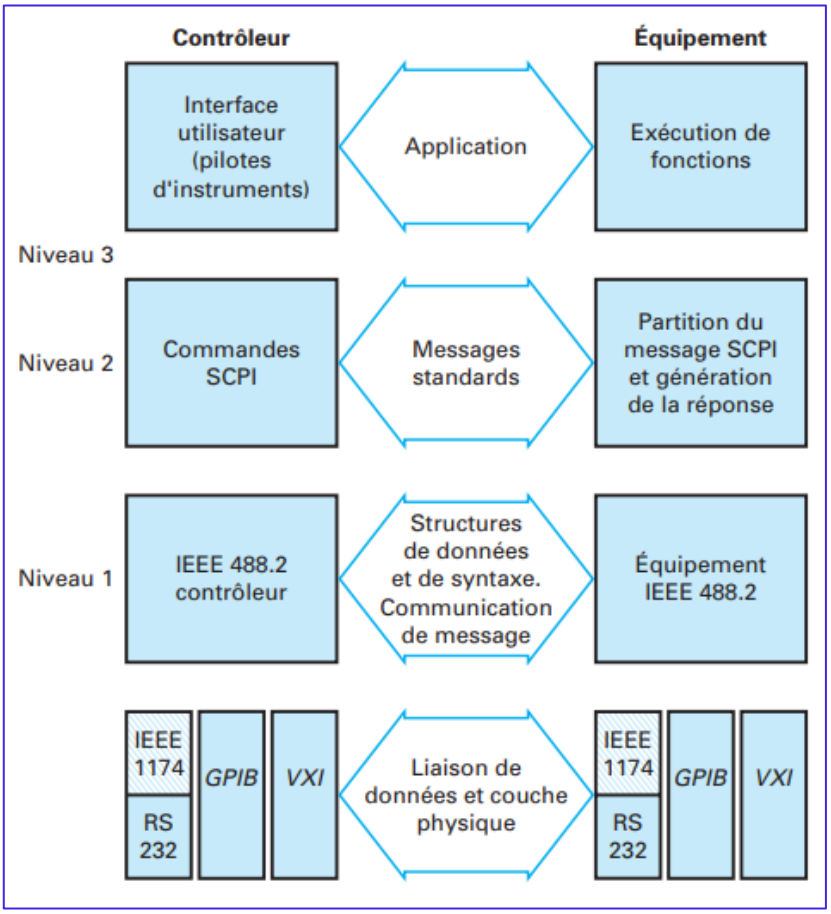


Figure 21 : relation entre les différents standards. [11]

Au cours de cette section, nous avons décortiqué le bus GPIB en termes de structure, caractéristiques, le protocole de communication, et les commandes qu'il supporte. La partie qui va suivreest consacrée au côté pratique.

Conclusion

L'objet de notre mémoire portant sur la communication entre un ordinateur et un instrument de laboratoire, dans ce premier chapitre, nous avons d'abord défini les bus d'interface (ISA, VESA, PCI) qui sont des bus informatiques implantés à l'intérieur de l'ordinateur permettant la communication entre les composants d'un même ordinateur ou entre différents ordinateurs. Nous nous sommes intéressés plus particulièrement au bus PCI qui va nous servir pour installer la carte GPIB.

Ensuite, nous avons décri la carte Quancom GPIB, modèle PCI qui a pour but essentiel de connecter des instruments de table avec l'interface connue IEEE 488 à l'ordinateur, et donné quelque fonctions supportée par cette interface de communication.

Enfin, nous avons présenté le bus de communication GPIB, qui se matérialise par un ensemble de protocoles et de câbles pour relier un pc bureautique avec des instruments.

II.Partie pratique

Introduction

L'objectif de notre travail étant d'établir la communication et le contrôle entre un PC bureautique et un ensemble d'instruments de laboratoire donnés. Dans notre cas d'application, nous allons interfacer une source de courant « ***Keitheley 220*** » à un ordinateur en utilisant la carte d'interface GPIB « ***Quancom GPIB/PCI*** » insérée dans l'ordinateur et le langage de programmation Visual C^{++} qui va nous permettre d'écrire les sub-routines requises pour atteindre notre objectif.

Par ailleurs, vu que la quasi-totalité du travail réalisé s'appuie sur le langage de programmation Visual C^{++}, il serait judicieux d'entamer cette partie avec une brève présentation de cet outil tout en mettant l'accent sur les partie que nous allons utiliser par la suite. Après ce passage, nous décrirons l'instrument de laboratoire « ***Keitheley 220*** », et en dernier lieu, nous allons détailler chaque fonction du programme utilisé pour interfacer cet instrument.

Section 01 : Brève présentation du langage de programmation « Visual *C*$^{++}$ »

Selon les instruction du constructeur de la carte d'interface « ***QuanCom GPIB-PCI*** », celle-ci peut être programmée en utilisant un nombre important de langages de programmation : Visual *C*$^{++}$, Visual Basic, Visual C, JAVA , Lab View …..). Pour ce qui est de notre cas, nous utiliserons le Visual *C*$^{++}$. Mais comme on vient de l'annoncer en haut, avant de passer à description du travail réalisé, nous donnerons un bref aperçu sur celui-ci.

1.1 Définition

Le « Visual *C*$^{++}$ » est un logiciel qui regroupe au sein d'une même interface tous les outils nécessaires au développement de programmes en *C*$^{++}$. C'est un environnement de développement intégré (ou IDE pour *Integrated Development Environment*) qui regroupe au minimum les fonctionnalités suivantes : un éditeur de texte, un compilateur, des outils d'aide à la programmation et un débogueur. Il fournit aussi des outils et des bibliothèques propres permettant de créer des interfaces graphiques (IHM : Interfaces Hommes Machines) ou GUI (Graphical User Interface).

1.2 Création d'un nouveau projet avec Visual *C*$^{++}$

Il existe deux grands types d'applications informatiques : les applications « **fenêtrées** » constituées d'une interface avec des boutons, des cases à cocher, des listes déroulantes… et des applications **« consoles »** ; lancées dans une interface texte et n'utilisant pas la souris ni les graphismes. Sous Visual, on retrouve ces deux choix. On utilise les « **applications Windows Forms** « pour les applications fenêtrées et les « **applications console** » pour les applications consoles.

La création d'un projet en Visual *C*$^{++}$ passe par les étapes suivantes :

1. Dans le menu Fichier, sélectionnez **Nouveau** puis la commande **Projet.**
2. Choisir le type de projet convenant parmi les modèles proposés. Ils sont regroupés par catégories. Dans la partie inférieure de la boîte de dialogue, saisir le nom du projet dans le champ Nom, et spécifier un emplacement.
3. Une nouvelle solution va être créée dans l'emplacement choisi pour le projet. Nommer la solution dans le champ Nom de la solution (par défaut, le nom du projet est utilisé). Cocher la case Créer le répertoire pour la solution, et cliquez sur OK.
4. La boîte de dialogue Assistant Application Win32 apparaît. Cliquer sur Suivant
5. Dans le nouvel écran, laisser l'option Application console sélectionnée, cocher la case Projet vide et cliquer sur Terminer pour terminer la création du projet et de sa solution.

Les fichiers de programmation sont des fichiers texte, appartenant à deux familles : **les fichiers sources**, dont l'extension est traditionnellement .cpp, et **les fichiers d'en-tête,** d'extension.

Les fichiers nécessaires à la création d'un programme sont regroupés dans un projet. Ce dernier contient également les éléments de configuration nécessaires à la création du programme. **Le fichier associé au projet porte l'extension .vcxproj.** Enfin, plusieurs projets peuvent être regroupés dans un même espace de travail, appelé "solution". **Le fichier associé à une solution est au format .sln.**

1.3 Génération et exécution d'un programme

Un programme C^{++}, tel que créé est compréhensible par le programmeur, mais n'est pas interprétable par son ordinateur. Deux étapes permettent de créer un programme exécutable par la machine.

Les étapes de la création d'un exécutable se présentent comme suivant :

- **L'étape 1 « la compilation » :** chacun des fichiers sources (.cpp) contenus dans le projet est compilé séparément. La compilation est le processus qui vérifie la correction de la syntaxe, puis qui traduit en langage machine les instructions d'un fichier source. En sortie, le résultat de la compilation d'un fichier source est stocké dans un fichier objet, qui porte le nom du fichier source correspondant, avec l'extension .obj.
- **L'étape 2 « l'édition de liens » :** fusionnement du résultat de la compilation pour créer un exécutable. Les fichiers objets sont concaténés. Les fonctionnalités qui ne sont pas programmées sont liées au programme durant cette deuxième étape, appelée "édition de liens". À l'issue de cette étape, le programme exécutable est généré ; il porte le nom du projet, et l'extension (exe).

Il existe deux types de compilation dans les environnements de développement : **« debug » et « release ».**

- **Le mode Debug** permet d'accéder aux fonctionnalités de débogage.
- **Le mode Release :** correspond au mode utilisé pour produire le programme distribué aux utilisateurs.

En mode Debug lorsqu'un bug majeur stoppe le programme, le développeur a la possibilité de reprendre la main et de diagnostiquer les variables produisant l'erreur. Mais en mode release, le programme s'arrête brutalement. Et l'environnement de développement peut aussi subir

quelques dégâts (blocage des menus, impossibilité de relancer le programme, ….). Les deux modes coexistent et le développeur a le choix entre les deux ; même si le mode Debug est le plus majoritairement utilisé.

Dans Visual C^{++}, tout ce qui se rapporte à la génération d'un programme exécutable se trouve dans le menu **Générer**.

Le programme étant généré, il peut alors être testé, on lançant l'exécutable en double-cliquant sur le fichier créé sur le disque dur, mais il y a plus rapide : on sélectionnant le menu Déboguer de Visual C^{++}, puis la commande Exécuter sans débogage, ou bien on utilisant les touches [Ctrl]+[F5].

En suivant pas-à-pas toutes ces informations et ces étapes, nous avons pu interfacer notre instrument sans grands problèmes méritant d'être cités dans ce rapport.

Section 02 : Présentation de l'instrument « Keithley 220»

Comme nous l'avons déjà annoncé en haut, l'instrument de laboratoire à interfacer avec les PC est la source de courant haute précision : « ***Keithley 220***». Dans ce qui suit, nous allons décrire cet instrument en fournissant ses caractéristiques, ses fonctions et commandes.

La source de courant : « ***Keithley 220***» est un instrument de laboratoire programmable qui sert à générer un courant continu de précision, servant à alimenter des charges formées par des matériaux spéciaux en cours d'étude. Le modèle 220 est une source de courant de 1 mA à 100 mA sous des tensions allant de 1 à 105 V avec la possibilité d'incrémentation par programmation de 1V par pas.

La source de courant programmable modèle 220 peut être interfacée à un PC grâce à une interface standard : IEEE-488. Cette interface permet aussi la transmission de données et de commandes sur le bus IEEE-488. Cette interface intégrée fournit toute la logique nécessaire pour relier le modèle 220 au bus en utilisant le protocole standard IEEE-488-1978. De plus, l'interface intègre un port E/S numérique distinct pouvant être utilisé pour relier le modèle 220 à d'autres instruments numériques.

2. 1 Les fonctions et les commandes supportées par l'interface GPIB du model 220

La norme IEEE 488-1 prévoit une grande variété de fonctions et de commandes. Tous les appareils du marché ne les supportent pas pour autant. Chaque constructeur indique clairement les fonctions supportées par son interface.

Les codes des fonctions d'interface font partie des normes IEEE-488/1978. Ces codes définissent la capacité d'un instrument à soutenir diverses fonctions d'interface, ils ne doivent pas être confondus avec les commandes de programmation. Le tableau 2 répertorie les codes pour le modèle 220.

Ces codes sont également répertoriés pour plus de commodité sur le panneau arrière de l'instrument immédiatement au-dessus du connecteur IEEE :

1. **SH (Source Handshake Function)** : La capacité du Modèle 220/230 pour initier le transfert de message / données sur le bus de données est fournie par la fonction SH.
2. **AH (Acceptor Handshake Function)** : la capacité du Modèle 220/230 pour garantir une réception correcte des messages / les données sur le bus de données sont fournis par l'AH fonction.

3. **T (Talker Function)** : La capacité du modèle 220/230 à envoyer des données dépendantes de l'instrument via le bus (à d'autres dispositifs) est permise par la fonction T. les capacités de conversation du Modèle 220/230 existent si seulement l'instrument a été adressé pour parler.
4. **L (Listener Function)** : La capacité du modèle 220/230 à recevoir des données dépendantes d'un instrument à travers le bus (à partir de autres appareils) est fournie par la fonction L. les capacités des fonctions d'écoute du modèle 220 ou 230 sont possibles uniquement s'il a été adressé pour écouter.
5. **SR (service Request Function)** : La capacité du modèle 220/230 pour demander le service du contrôleur est fournie par la fonction SR.
6. **RL (Remote-Local Function)** : La possibilité au modèle 220/230 à se placer en mode distant ou local est fournie par la fonction RL.
7. **PP (Paralle Poll Function)** : Le modèle 220/230 n'a pas les capacités d'interrogation en parallèle.
8. **DC (Device Clear Function)** : Capacité du modèle 220/230 à effacer (initialiser) est fournie par la fonction DC.
9. **DT (Device Trigger Function)** : La capacité du modèle 220/230 pour que son fonctionnement de base commence (début du fonctionnement du programme) est fournie par la fonction DT.
10. **C (Controller Function) :** Le modèle 220 n'a pas des capacités de contrôleur.
11. **TE (Extended Talker Capabilities) :** Le modèle 220 n'a pas des capacités étendues de parleur.
12. **LE (Extended Listener Capabilities) :** Le modèle 220 n'a pas des capacités étendues d'écoute.

La valeur numérique suivant chaque code à une ou deux lettres définie les capacités du modèle 220/230 comme suivant:

code	Fonction d'interface
SH1	Source Handshake Capability
AH1	Acceptor Handshake Capability
T6	Talker (Basic Talker, Serial Poll,Unaddressed to Talker On LAG)
L4	Listener(Basic Listener, Unaddressed To Listen On TAG)
SR1	Service Request Capability
RL1	Remote/local Capability
PP0	No Parallel Poll Capability
DC1	Device Clear Capability
DT1	Device Trigger Capability
C0	No Controller Bus Drivers
E1	Open Collector Bus Drivers
TE0	No Extended Talker Capabiliities
LE0	No Extended Listener Capabilities

Tableau 2 : les codes des fonctions d'interface du Modèle 220 *[12]*

2.2 Les modes de programmation local et à distance de l'instrument *Keithley 220*

Il existe deux modes de programmation de l'instrument keithley :

- Le mode local en utilisant les boutons de la face avant,
- Le mode à distance : à travers la programmation.

2.2.1 Le mode local

Dans ce mode, on utilise les boutons de la face avant (figure ci-dessous), pour gérer l'instrument : allumer ou éteindre l'instrument, et le choix du mode dont nous avons besoin et rentrer les données.

Figure 22 : La face avant de l'instrument Keithley 220. [13]

La face avant de l'instrument nous permet de contrôler l'instrument à travers 06 modes :

1. Mode Display,
2. Mode program,
3. Mode program contrôle,
4. Mode output,
5. Mode Data Entry,
6. Mode Data.

Chaque mode est géré via un ensemble de boutons, comme illustrés dans la figure supra.

2.2.2 Le mode à distance

L'instrument est piloté via un bus de communication « IEEE488 bus ». Avant que le modèle 220 puisse être utilisé avec ce bus, l'instrument doit être connecté au bus avec un connecteur approprié. En outre, l'adresse principale doit être correctement sélectionnée. Grace à ce mécanisme l'instrument est contrôlé via un programme intégrant les commandes suivantes :

2.2.2.1 Les commandes de bus

Alors que les aspects matériels du bus sont essentiels, l'interface serait essentiellement sans valeur sans les commandes appropriées pour permettre la communication avec l'instrument. Les commandes de bus sont regroupées dans les trois catégories suivantes, nous allons les définir brièvement dans ce qui suit :

- **Les commandes Uniline** : (ATN, IFC, REN, SRQ, EOI) Ces commandes sont envoyées sur une ligne du bus associé à l'état bas. Les signaux : ATN, IFC et REN sont validés uniquement par le contrôleur. La commande SRQ est envoyée par un l'instrument externe. La commande EOI peut être envoyée soit par le contrôleur ou un l'instrument externe en fonction de la direction de transfert de données.

- **Les Commandes multilignes** : sont des commandes envoyées sur les lignes du bus de données avec ATN à l'état bas :

  - **Les commandes universelles : (**LLO, DCL, SPE, SPD). Celles-ci sont des commandes multilignes qui ne nécessitent pas d'adressage. Tous les instruments équipés pour les mettre en œuvre le feront simultanément lorsque la commande est transmise sur le bus.
  - **Les commandes adressées :** (SDC, GTL, GET) Ce sont des commandes multilignes qui doivent être précédés d'une commande d'écoute dérivée de l'adresse primaire avant que l'instrument ne réponde. Seulement l'appareil adressé répondra à chacune de ces commandes.
  - **Les commandes non adressées :** (UNL, UNT) elles ne sont pas adressées et elles sont utilisées par le contrôleur pour retirer tous les parleurs et les écouteurs du bus simultanément.

Toutes les commandes de bus sont résumées dans le tableau ci-dessous :

Type de commande	**commande**	**Etat de la ligne ATN**	**description**
Uniligne	**REN**	**X**	Configure le fonctionnement à distance.
	EOI	**X**	Envoyé en définissant EOI faible.
	IFC	**X**	Efface l'interface
	ATN	**Bas**	Définit le contenu du bus de données.
	SRQ	**X**	Contrôlée par un périphérique externe.
Multiligne Universelles	**LLO**	**Bas**	Verrouille le contrôle via la face avant.
	DCL	**Bas**	Remet le l'instrument aux conditions par défaut.
	SPE	**Bas**	Active l'interrogation en série.
	SPD	**Bas**	Désactive l'interrogation en série.
Adressées	**SDC**	**Bas**	Remet l'unité aux conditions par défaut.
	GTL	**Bas**	Retour au contrôle local.
	GET	**Bas**	Déclenchement de l'instrument.

Non adressées	**UNL** **UNT**	**Bas** **Bas**	Retire tous les écouteurs du bus. Retire tous les parleurs du bus.
Dépendante du l'instrument		**haut**	**Programme le Modèle 220 pour divers modes.**

Tableau 3 : les types de commandes de bus [12]

2.2.2.2 Commandes dépendantes de l'instrument

Ces commandes dépendant de la configuration de l'instrument, elles sont envoyées sur les lignes de données avec un signal ATN à l'état haut.

Généralement, ces commandes sont envoyés sous forme d'un ou plus caractères ASCII, indiquant à l'instrument d'exécuter une fonction spécifique. Par exemple, FO est envoyée au modèle 220 pour mettre l'instrument en mode veille.

Les commandes dépendantes du périphérique sont présentées dans le tableau suivant :

Type de commande	Commandes	Description
Les commandes du mode trigger	T0 T1 T2 T3 T4 T5 T6 T7	Start On Talk Stop On Talk Start On GET Stop On GET Start On X Stop On X Start On External Trigger Stop On External Trigger
Les commandes du mode program	P0 P1 P2	Single Continuous Step
Les commandes du mode Display(D) :	D0 D1 D2 D3	Current Source Voltage LImit Dwell Time Memory Location
Les commandes data format (G) :	G1 G2 G3 G4 G5	Location With Prefix Location Without Prefix Buffer Address With Prefix Full Buffer With Prefix Full Buffer Without Prefix
Les commandes de mode fonction (F) :	F0 F1	Standby (Voltage Limit <32V) Operate (Set to Programmed Value)

Les commandes d'Exécute (X) :	X	Execute Other Device-Dependent Commands
Les commandes du mode range	R0	Auto
	R1	1nA
	R2	10nA
	R3	100nA
	R4	1µA
	R5	10µA
	R6	100µA
	R7	1mA
	R8	10mA
	R9	100mA
Les commandes du mode programmable terminaton (Y) :	Y(ASCII)	Any ASCII except capitals, numbers, + - /, e
	Y(LF)	CR LF
	Y(CR)	LF CR
	Y(DEL)	None
Les commandes du mode digital self test (J)	J0	Test Instrument, Set J Byte in Status Commands
Les commandes du mode status word (U)	U0	Send Status Word
	U1	Send I/O Port Status
Les commandes I/P port (O) : (0,15)	O(0-15)	Set Ouput Control Bits
Les commandes Inputs :	I	Current Source
	V	Voltage limit
	W	Dwell Time
	B	Buffer Address
	L	Memory Location
Les commandes SRQ mode (M) and Status Byte Format	M0	Disabled
	M1	IDDC, IDDCO, No Remote
	M2	Over Voltage Limit
	M4	End Of Buffer
	M8	End Of Dwell Time
	M16	Input Port Change
Les commandes EOI Programming (K)	K0	Send EOI
	K1	Send No EOI

Tableau 4 : les types de commandes dépendantes de l'instrument [12]

Section 03 : Présentation du programme d'interfaçage (La conception et le déroulement du programme)

Afin de contrôler l'instrument de laboratoire « *Keitheley 220* » via la carte GPIB Quancom /PCI, nous avons conçu un programme à l'aide du langage de programmation « Visual C^{++}» que nous allons décrire en détail dans ce qui suit.

Notre programme est constitué d'un ensemble de fonctions déclarées et définies dans une bibliothèque pré-élaboré par le constructeur de la carte d'interface : « QLIB 32 ». Ces fonctions ont été intégrées dans nos programmes écrits en Visual C^{++}. Il est également à noter que l'utilisation de ces fonctions facilite énormément l'utilisation de la carte GPIB/ PCI, ainsi que l'envoi des commandes dépendantes de l'instrument « Keithley 220».

Pour comprendre la logique de développement de nos programmes, nous allons décrire les différentes fonctions et commandes. Nous allons ainsi dans ce qui suit expliquer en détail le rôle des principales instructions qui le composent et illustrer leurs résultats.

3.1 Les fonctions liées à la carte GPIB-PCI

Pour commander et configurer la carte d'interface « QuanCom, GPIB-PCI », Nous avons utilisé l'ensemble des fonctions suivantes :

3.1.1 La fonction QAPIExtOpenCard()

Cette fonction permet d'ouvrir une session de communication entre le PC (Le processeur) et la carte d'interface GPIB-PCI.

```
ULONG handle = QAPIExtOpenCard(PCIGPIB,0);
	if ( handle == NULL )
		{
			handle = QAPIExtOpenCard(USBGPIB,0);

			if ( handle == NULL )
				{
					handle = QAPIExtOpenCard(GPIB,0);
				}

		}

	printf("Handle %u\n",handle);


	if ( handle == NULL )
		{
			MessageBox(NULL, "Il n'y a aucun controlleur GPIB connecté à
ce PC !", "Error", MB_OK)   }
```

Figure 23 : la syntaxe de la fonction *QAPIExtOpenCard()*[13]

Grace à cette fonction, la carte est recherchée consécutivement dans les bus d'extension du PC : PCI, USB, PCI Express, …, jusqu'à sa détection. Dans le cas contraire, son absence est indiquée à travers un message : ("Il n'y a aucun contrôleur GPIB connecté à ce PC !")

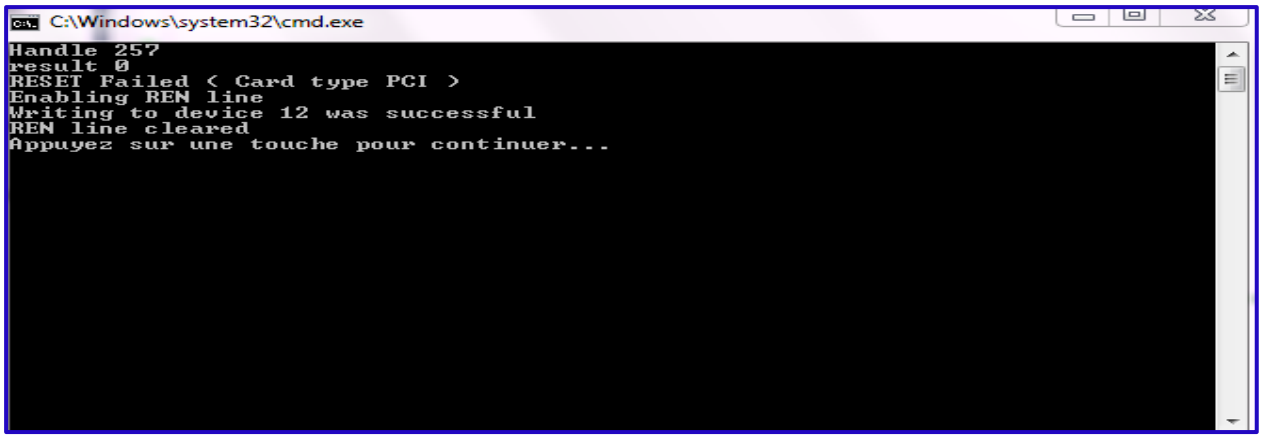


Figure 24 : résultat de l'exécution de la fonction *QAPIExtOpenCard()*[13]

3.1.2 La fonction *QAPIExtSpecial()*

Cette fonction permet d'envoyer des commandes spécifiques à l'instrument de table connecté au PC via le bus GPIB.

```
QAPIExtSpecial(unlog cardid, unlog code du travail, unlog para1, unlog para 2)
```

Figure 25 : La syntaxe de la fonction *QAPIExtSpecial()*.[14]

Dans notre cas nous avons utilisé cette fonction pour envoyer plusieurs commandes à la carte d'interface. Celles-ci sont énoncées juste en dessous :

3.1.2.1 La commande job_reset

Dans ce cas, la commande JOB RESET est appelée par la fonction *QAPIExtSpecial()* et exécutée pour réinitialiser la carte GPIB.

```
1.  result = QAPIExtSpecial(handle, JOB_RESET,1,0);
2.      printf("result %u\n",result);
3.      if (result)
4.
5.              {
6.                      printf("RESET Controller\n");
7.              }
8.      else
9.              {
10.
11.                     printf("RESET Failed ( Card type PCI )\n");
12.             }
```

Figure 26 : syntaxe de la commande JOB_RESET. [14]

Dans notre cas d'application, la carte GPIB est connectée sur un bus PCI ; c'est le seul cas où la carte GPIB n'est pas réinitialisée par le programme car, elle l'est toujours par défaut.

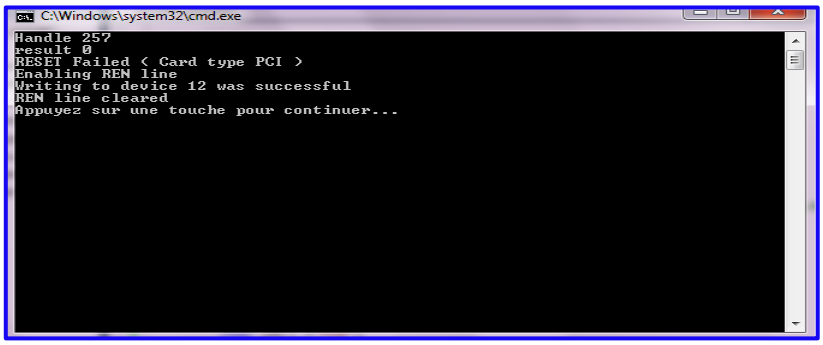


Figure 27 : résultat de l'exécution de la commande job_reset. [13]

3.1.2.2 La commande JOB_REN

Cette commande a pour rôle d'activer ou désactiver le mode à distance. Généralement, la commande REN doit être envoyée avant d'essayer de contrôler les instruments par le bus de communication.

```
13. QAPIExtSpecial(handle, JOB_REN, TRUE, 0);
14.
15.     if (result)
16.             {
17.                     printf("Enabling REN line \n");
18.             }
19.     else
20.             {
21.                     printf("Enabling REN line failed\n");
22.             }
```

Figure 28 : La syntaxe de la commande JOB_REN. [14]

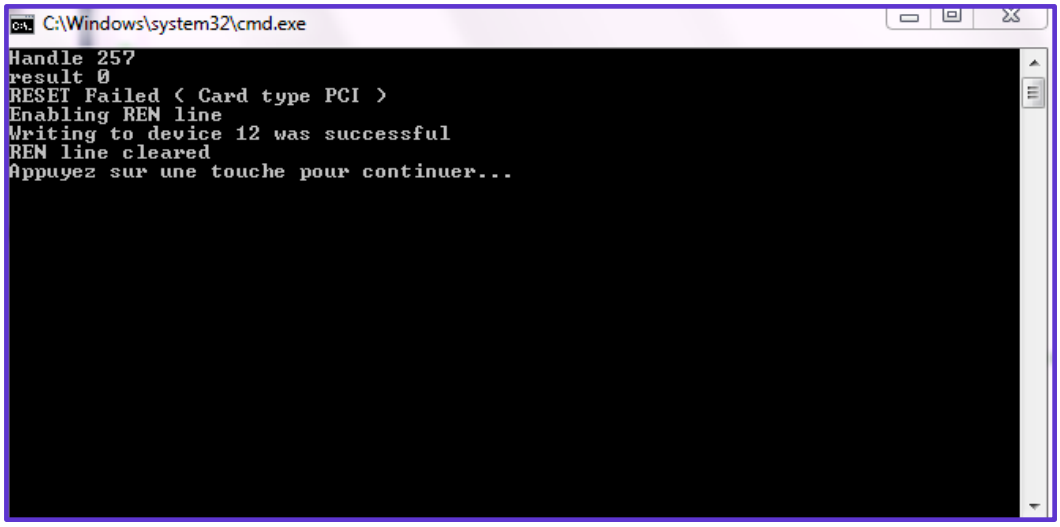


Figure 29 : Résultat de l'exécution de la commande JOB_REN. [13]

3.1.2.3 La commande JOB_SDC (selected device clear)

Cette commande réinitialise l'instrument adressé. Elle effectue essentiellement la même fonction que la commande DCL sauf que seul l'instrument adressé répondra. Les instruments reviennent généralement à leurs conditions par défaut lorsque la commande SDC est envoyée.

```
23. result = QAPIExtSpecial(handle, JOB_SDC, 12, NULL);
24.
25.    if ( !result )
26.           {
27.                  printf("Command SDC failed\n");
28.           }
29.    else {   printf("command SDC\n");
30.    }
```

Figure 30 : la syntaxe de la commande JOB_SDC. [14]

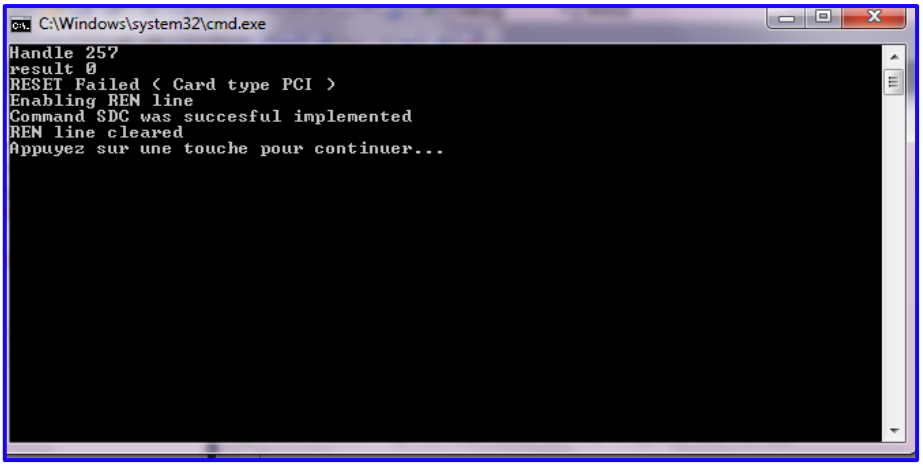


Figure 31 : résultat de la commande JOB_SDC. [13]

3.1.2.4 La commande JOB_GET (Groupe Execute Trigger)

Cette commande force l'instrument adressé à déclencher une action. Bien que GET soit considéré comme une commande adressée, de nombreux instruments répondent à GET sans être adressés.

```
result = QAPIExtSpecial(handle, JOB_GET, 12, NULL);

    if ( !result  )
           {
                  printf("Command GET failed  \n");
           }
      else {printf("Command GET \n");
     }
```

Figure 32 : la syntaxe de la commande JOB_GET. [14]

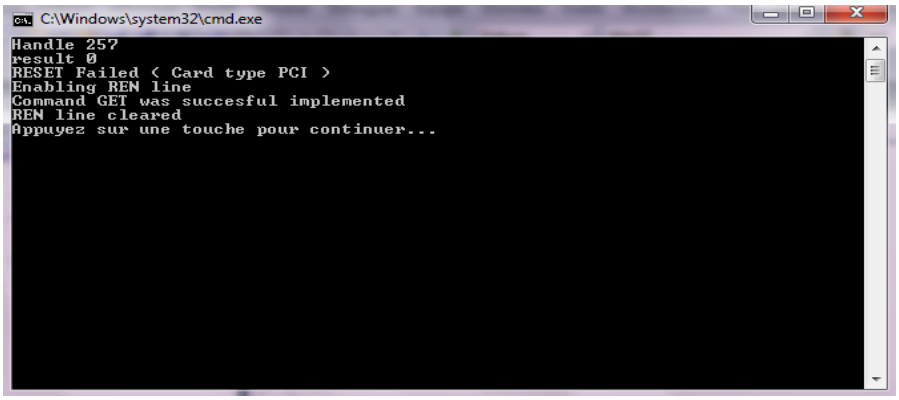


Figure 33 : résultat de l'exécution de la commande JOB_GET. [13]

3.1.2.5 La commande JOB_DCL (Device Clear)

Cette commande permet de réinitialiser les instruments adressés. Ceux-ci retournent dans leur état prédéfini.

```
      result = QAPIExtSpecial(handle, JOB_DCL, 1, NULL);

     if ( !result )
          {
                 printf("Command DCL failed\n");
          }
        else { printf("Command DCL was succesfully implemented\n"); }
```

Figure 34 : la syntaxe de la commande JOB_DCL. [14]

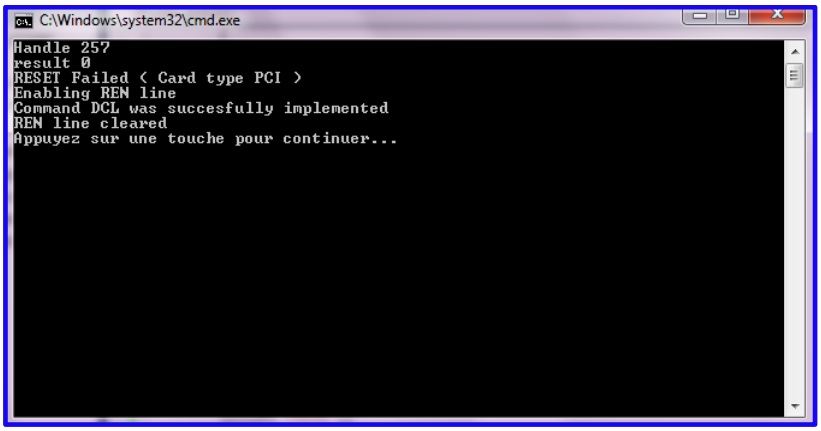


Figure 35 : résultat de l'exécution de la commande JOB_DCL. [13]

3.1.2.6 La commande JOB_LLO (Mode Local Verouillé)

Le mode local est verrouillé par cette commande, le pilotage de l'instrument dépend exclusivement du bus IEEE.

```
45.    result = QAPIExtSpecial(handle, JOB_LLO, 1, NULL);
46.
47.    if ( !result )
48.           {
49.                  printf("Command LLO failed\n");
50.           }
51.    else { printf("Command LLO was succesfully implemented\n");
52.                }
```

Figure 36 : la syntaxe de la commande JOB_LLO. [14]

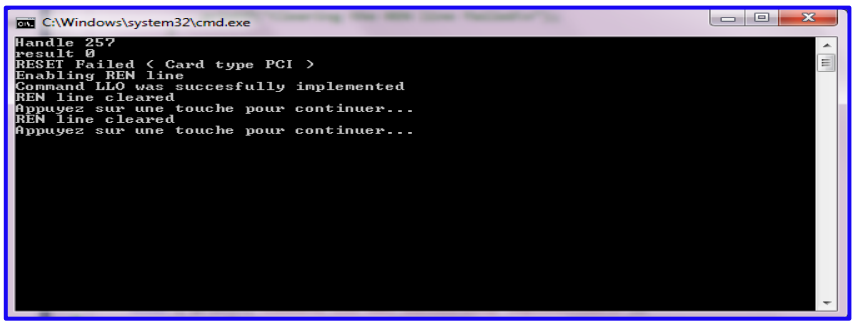


Figure 37 : résultat de l'exécution de la commande JOB_LLO. [13]

3.1.2.7 La commande JOB_GTL (Go To Local)

La commande en question permet le retour au mode local ; l'appareil se déconnecte du bus et autorise sa commande par le panneau frontal. Aussi, le contrôle par le panneau avant sera généralement restauré si la commande LLO a été précédemment envoyée.

```
result = QAPIExtSpecial(handle, JOB_GTL, 12, NULL);

  if ( !result  )
              {
                    printf("Command GTL failed\n");
              }
         else { printf("Command GTL was succesful implemented\n"); }
```

Figure 38 : la syntaxe de la commande JOB_GTL. [14]

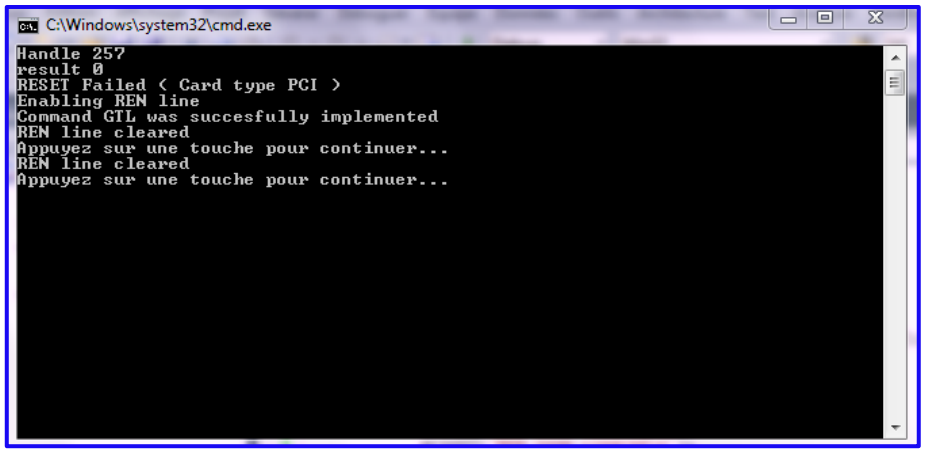


Figure 39 : résultat de l'exécution de la commande JOB_GTL. [13]

3.1.3 La fonction QAPIExtWriteString()

Avec cette fonction on peut envoyer n'importe quelle commande à l'instrument de table connecté au PC via l'interface GPIB.

```
       chars4[]="I+101.00E-3,V+10.00E+0,W+4.000E-2,B+1.0E+0XI+098.00E-
       3,V+10.00E+0,W+4.000E-2,B+2.00E+0XI+058.00E-3,V+10.00E+0,W+4.000E-
       2,B+3.00E+0X";
       if ( QAPIExtWriteString(handle, listener, (char*)&s1, strlen(s1),0))
53.           {
54.                 printf("Writing to device %u was successful\n", listener);
55.    }
56.    else
57.           {
58.                 printf("Writing to device %u failed\n", listener);
59.           }
```

Figure 40 : syntaxe de la fonction *QAPIExtWriteString()*.[14]

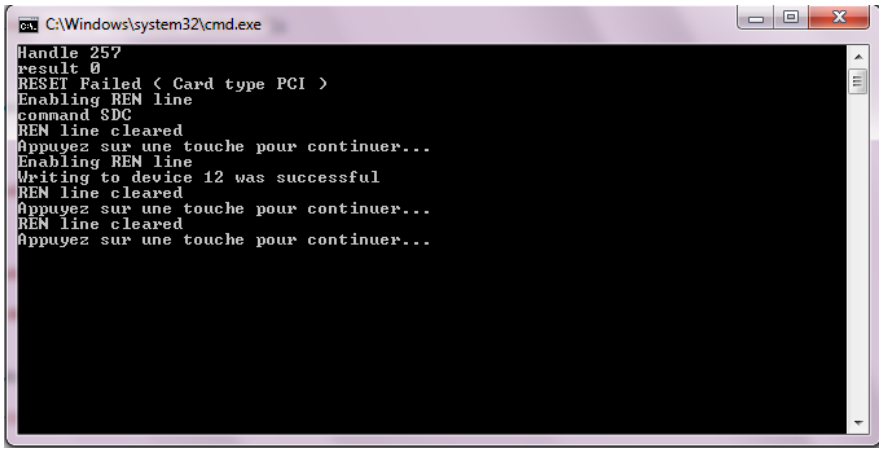


Figure 41 : résultat de l'exécution de la fonction *QAPIExtWriteString()*.[13]

3.1.4 La fonction *QAPIExtReadString()*

A travers cette fonction, on peut lire une chaine de caractères ou une commande une chaine reçue à partie d'un appareil connecté au PC.

```
60. result = QAPIExtReadString(handle, talker, (char*)&buffer, sizeof(buffer), 0);
61.
62.    if (result)
63.           {
64.                  printf("Reading from device %u was: %s\n",talker, buffer);
65.           }
66.    else
67.           {
68.                  printf("Reading from device %u failed\n", talker);
69.           }
```

Figure 42 : syntaxe de la fonction *QAPIExtReadString()*.[14]

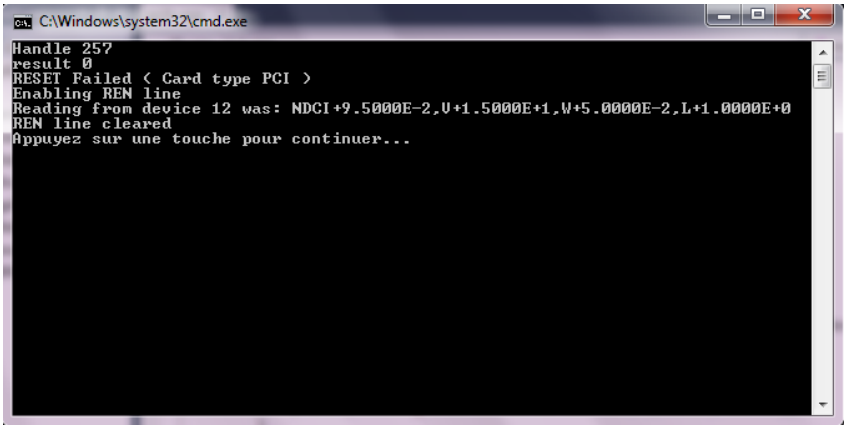


Figure 43 : résultat de l'exécution de la commande QAPIExtReadString. [13]

3.1.5 La fonction *QAPIExtCloseCard(handle)*

C'est une fonction qui est utilisée pour fermer la session de communication entre la carte GPIB et le PC.

```
QAPIExtCloseCard(handle);
```

Figure 44 : syntaxe de la commande QAPIExtCloseCard(handle). [14]

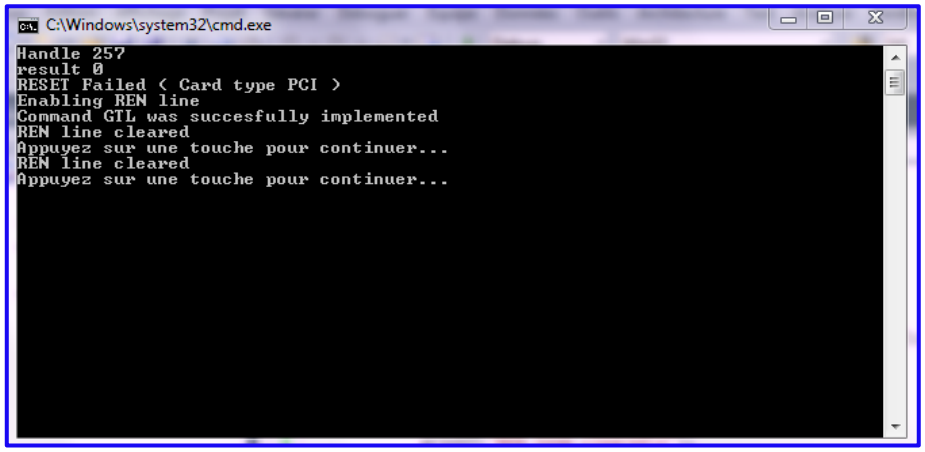


Figure 45 : résultat de l'exécution de la commande QAPIExtCloseCard(handle). [13]

3.2 Commandes dépendantes du périphérique

Dans cette catégorie nous avons utilisé les commandes suivantes :

3.2.1 Les commandes du mode trigger

A la base les commandes du mode trigger contrôlent les opérations de l'instrument tel que le bouton START/STOP de la face avant, sauf qu'elles ne peuvent contrôler que l'un ou l'autre (Start ou stop). Pour notre cas, nous avons utilisé les commandes : ***T1, T2, T3, T4 et T5.***

3.2.1.1 La commande *T0*

T0 permet de déclencher l'instrument en mode parleur (Start On Talk). Le témoin START/STOP de la face avant s'allume, indiquant que l'instrument a reçu le signal.

```
70. char s1[] ="T0X";
71.
72.    if ( QAPIExtWriteString(handle, listener, (char*)&s1, strlen(s1),0))
73.          {
74.                printf("Writing to device %u was successful\n", listener);
75.    }
76.
77.    else
78.          {
79.                printf("Writing to device %u failed\n", listener);
80.          }
```

Figure 46 : syntaxe de la commande T0. [14]

Figure 47 : résultat de l'exécution de la commande T0. [13]

3.2.1.2 La commande T1

T1 permet d'arrêter le mode parleur (Stop On Talk). Le témoin START/STOP du panneau avant s'teint, indiquant que l'instrument a reçu le signal.

```
81. char s1[] ="T1X";
82.
83.     if (QAPIExtWriteString(handle, listener, (char*)&s1, strlen(s1),0))
84.             {
85.                     printf ("Writing to device %u was successful\n", listener);
86.     }
87.
88.     else
89.             {
90.                     Printf ("Writing to device %u failed\n", listener);
91.             }
```

Figure 48 : syntaxe de la commande T1. [14]

Figure 49 : résultat de l'exécution de la commande T1. [13]

3.2.1.3 La commande T2

Elle Permet d'activer l'instrument en mode GET (Start On GET). Le témoin START/STOP de la face avant s'allume, indiquant que l'instrument a reçu le signal.

```
result = QAPIExtSpecial(handle, JOB_GET, 12, NULL);
       if ( !result  ) { printf("Command GET failed  \n");}
         else {printf("Command GET \n"); }
         if (result) { printf("REN line cleared\n"); }
               else  { printf("Clearing the REN line failed\n");}
         system("PAUSE");
        char s2[] ="T2X";
       if ( QAPIExtWriteString(handle, listener, (char*)&s2, strlen(s2),0))
             {printf("Writing to device %u was successful\n", listener);}
       else { printf("Writing to device %u failed\n", listener);}
```

Figure 50 : syntaxe de la commande T2. [14]

Figure 51 : résultat de l'exécution de la commande T2. [13]

3.2.1.4 La commande T3

Permet d'arrêter l'instrument (Stop On GET). Le témoin START/STOP du panneau avant s'teint, indiquant que l'instrument a reçu le signal.

```
       result = QAPIExtSpecial(handle, JOB_GET, 12, NULL);
       if ( !result  ) { printf("Command GET failed  \n");}
         else {printf("Command GET \n"); }
         if (result) { printf("REN line cleared\n"); }
               else  { printf("Clearing the REN line failed\n");}
         system("PAUSE");
        char s2[] ="T3X";
       if ( QAPIExtWriteString(handle, listener, (char*)&s2, strlen(s2),0))
             {printf("Writing to device %u was successful\n", listener);}
       else { printf("Writing to device %u failed\n", listener);}
```

Figure 52 : syntaxe de la commande T3. [14]

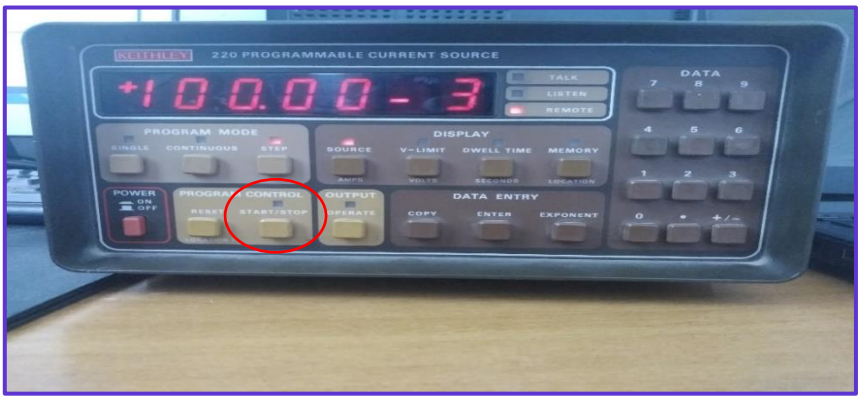


Figure 53 : résultat de l'exécution de la commande T3. [13]

3.2.1.4 La commande T4

La commande T4 déclenche l'instrument en mode X (Start On X). Le voyant du bouton Start/stop s'allume.

```
char s4[] ="T4X";

  if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
        {
  printf("Writing to device %u was successful\n", listener);
  }
  else
        {printf("Writing to device %u failed\n", listener); }
```

Figure 54 : syntaxe de la commande T4. [14]

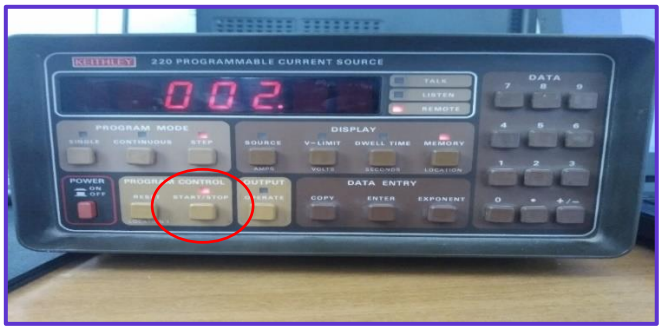


Figure 55 : résultat de l'exécution de la commande T4. [13]

3.2.1.5 La commande T5

La commande T5 arrête l'instrument en mode X (Stop On X). Le voyant du bouton Start/stop s'éteint.

```
char s4[] ="T5X";
if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
        {
printf("Writing to device %u was successful\n",listener)}
else
        {
                printf("Writing to device %u failed\n", listener);
        }
```

Figure 56 : syntaxe de la commande T5. [14]

Figure 57 : résultat de l'exécution de la commande T5. [13]

3.2.2 Les commandes du mode Display

Elles jouent le même rôle que les quatre boutons d'affichage (display) de la face avant de l'instrument. Elles permettent de régler l'affichage des valeurs de la source, limite de courant) ou de tension, le temps de pause et l'emplacement de la mémoire.

Nous retrouvons quatre commandes de mode affichage : ***D0X, D1X, D2X et D3X.***

3.2.2.1 La commande D0

Elle allume le voyant concernant le courant sur le panneau avant de l'instrument et affiche la valeur du courant sur l'afficheur de ce dernier.

```
char s4[] ="D0X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
                {
                        printf("Writing to device %u was successful\n", listener);
        }
        else
                {
                        printf("Writing to device %u failed\n", listener);
                }
```

Figure 58 : syntaxe de la commande D0. [14]

Figure 59 : Résultat de l'exécution de la commande D0. [13]

3.2.2.2 La commande D1

Elle allume le voyant concernant la tension sur la face avant de l'instrument et affiche sa valeur sur l'afficheur de ce dernier.

```
if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
		{
			printf("Writing to device %u was successful\n", listener);
	}
	else
		{
			printf("Writing to device %u failed\n", listener);
		}
```

Figure 60 : syntaxe de la commande D1. [14]

Figure 61 : Résultat de l'exécution de la commande D1. [13]

3.2.2.3 La commande D2

Elle allume le voyant concernant le temps de pause sur la face avant de l'instrument et affiche sa valeur sur l'afficheur de ce dernier.

```
char s4[] ="D2X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
		{
			printf("Writing to device %u was successful\n",
listener);
	}
	else
		{
			printf("Writing to device %u failed\n", listener);
		}
```

Figure 62 : syntaxe de la commande D2. [14]

Figure 63 : Résultat de l'exécution de la commande D2. [13]

3.2.2.3 La commande D3

Elle allume le voyant concernant l'emplacement en mémoire sur la face avant de l'instrument et affiche la valeur de la case sur l'afficheur de ce dernier.

```
char s4[] ="D3X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
              {
                     printf("Writing to device %u was successful\n",
listener);
       }
       else
              {
                     printf("Writing to device %u failed\n", listener);
              }
```

Figure 64 : syntaxe de la commande D3. [14]

Figure 65 : résultat de l'exécution de la commande D3. [13]

3.2.3 Les commandes du mode program

Elles réalisent les mêmes opérations que les boutons du panneau avant de l'instrument en mode program.

3.2.3.1 La commande P0

Elle sert à mettre l'instrument au mode program SINGLE, ce qui allume le bouton correspondant sur le panneau avant de l'instrument. L'instrument passe dans le tampon une seule fois, jusqu'à ce qu'il rencontre un temps de pause de zéro dans une case mémoire, il s'arrête.

```
    char s4[] ="P0X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
              {
                     printf("Writing to device %u was successful\n",
listener);
       }
       else
              {
                     printf("Writing to device %u failed\n", listener);
              }
```

Figure 66 : Syntaxe de la commande P0. [14]

Figure 67 : Résultat de l'exécution de la commande P0. [13]

3.2.3.2 La commande P1

L'instrument est mis en mode CONTINU, le bouton correspondant sur son panneau avant s'allume. L'instrument passe en continu dans le tampon jusqu'à ce qu'il soit appelé à s'arreter. Lorsque le temps de repos de zéro est rencontré, l'instrument retourne au premier emplacement de mémoire et répète le cycle.

```
char s4[] ="P0X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
              {
                     printf("Writing to device %u was successful\n",
listener);
       }
       else
              {
                     printf("Writing to device %u failed\n", listener);
              }
```

Figure 68 : syntaxe de la commande P1. [14]

Figure 69 : résultat de l'exécution de la commande. [13]

3.2.3.3 La commande P2

L'instrument est mis en mode STEP, le bouton correspondant sur le panneau avant s'allume. L'instrument incrémente l'emplacement mémoire une seule fois, chaque fois un stimulateur déclencheur est reçu.

Le stimulateur déclencheur peut provenir de la face avant du bouton START/STOP, ou d'une d'un bus de commandes… si Lorsque le temps de repos de zéro est rencontré, l'instrument pace à l'emplacement mémoire suivant avec le stimulateur déclencheur suivant.

```
char s4[] ="P2X";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
              {
                     printf("Writing to device %u was successful\n",
listener);
       }
       else
              {
                     printf("Writing to device %u failed\n", listener);
              }
```

Figure 70 : Syntaxe de la commande P2. [14]

Figure 71 : Résultat de l'exécution de la commande P2. [13]

3.2.4 Les commandes data format (format de données) G

Lorsque l'instrument est adressé pour parler, il envoie une chaine de données contenant des informations sur le courant programmé, la tension, le temps de pause et la mémoire ou les emplacements des tampons.

Grace à l'utilisation de la commande de format de données, l'utilisateur a le contrôle sur les aspects suivants de la chaine de données.

Les commandes de données qui affectent le fonctionnement des modèles 220 et 230 sont :

3.2.4.1 La commande G0

Permet de transmettre l'emplacement de la mémoire d'affichage avec les préfixes.

3.2.4.2 La commande G1

Transmet l'emplacement de mémoire d'affichage avec les préfixes.

3.2.4.3 La commande G2

Transmet l'emplacement de l'adresse du tampon GPIB avec des préfixes.

3.2.4.4 La commande G3

Transmet l'emplacement de l'adresse du tampon GPIB sans préfixes.

3.2.4.5 La commande G4

Transmet toute la mémoire du programme avec des des préfixes.

3.2.4.6 La commande G5

Transmet toute la mémoire du programme sans préfixes.

3.2.5 Les commandes de fonction

Elles contrôlent la sortie réelle de la source de l'instrument. Elles exécutent les mêmes opérations que le bouton OPERATE du panneau avant. Nous avons utilisé les commandes F0 et F1 pour notre cas d'application.

3.2.5.1 La commande F0 (standby)

La sortie de la source sera mise à zéro : en mode veille (limite de tension du modèle 220 < 32v, limite de courant du modèle 230 = 2 mA).

```
char s4[] ="F0";

if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
              {
                     printf("Writing to device %u was successful\n",
listener);
       }
       else
              {
                     printf("Writing to device %u failed\n", listener);
              }
```

Figure 72 : syntaxe de la commande F0. [14]

Figure 73 : résultat de l'exécution de la commande F0. [13]

3.2.5.2 La commande F1 (*Operate*)

La sortie est programmée sur la valeur source dans l'emplacement actuel de la mémoire d'affichage (mode fonctionnel)

```
if ( QAPIExtWriteString(handle, listener, (char*)&s4, strlen(s4),0))
                {
                        printf("Writing to device %u was successful\n",
listener);
        }
else
                {
                        printf("Writing to device %u failed\n", listener);
                }
```

Figure 74 : Syntaxe de la commande F1. [14]

Figure 75 : résultat de l'exécution de la commande F1. [13]

3.2.6 La commande P1T2

Cette commande place l'instrument dans le mode programme continu et le mode de déclenchement de démarrage-GTE. Le témoin de programme CONTINU sur le panneau avant de l'instrument s'allume.

```
92. char s1[] ="P1T2X";
93.
94.    if ( QAPIExtWriteString(handle, listener, (char*)&s1, strlen(s1),0))
95.          {
96.                printf("Writing to device %u was successful\n", listener);
97.    }
98.
99.    else
100.               {
101.                     printf("Writing to device %u failed\n", listener);
102.               }
```

Figure 76 : syntaxe de la commande P1T2. [14]

Figure 77 : résultat de l'exécution de la commande P1T2. [13]

Conclusion

Lors du travail décrit dans cette partie nous avons pu établir une communication entre un instrument de laboratoire « Keythley 220 » et un PC via le bus de communication GPIB, dans les deux modes locaux et à distance. Dans ce contexte, nous avons développé un programme, en utilisant l'environnement de développement intégré (IDE) du Visual C^{++} et nous sommes parvenus à exécuter l'essentiel des commandes de la carte GPIB, des commandes de Bus et des commandes dépendantes de l'instrument. L'ensemble des commandes déployées nous a permis de contrôler l'instrument parfaitement l'instrument en cours d'utilisation.

Conclusion générale

Conclusion générale

Le travail présenté dans ce mémoire a porté sur l'établissement d'une communication entre un ordinateur et un instrument de laboratoire via le bus GPIB et ce en vue de le contrôler en mode à distance.

Après une présentation des outils nécessaires à la réalisation de ce travail et de l'architecture de ce bus de communication à utiliser, nous avons abordé la phase de connexion et de contrôle de l'instrument « *Keithley/modèle 220* » avec un PC. Comme outils (Hardware/Software) de travail, il nous a été possible d'utiliser les éléments suivants :

- Un ordinateur bureautique muni d'un bus d'extension de type PCI et du logiciel de programmation « *Visual C*$^{++}$ »,
- Un instrument de table « *Keithley/modèle 220* » à contrôler ; il s'agit d'un générateur de courant programmable de haute précision.
- Une carte d'interface « *GPIB-PCI Quancom* » ; c'est l'élément clef pour permettre l'établissement de la communication entre le PC et l'instrument à contrôler.
- Un câble GPIB pour relier la carte et l'instrument et permettre le transfert de données et de commandes entre les deux entités communicantes.
- Le logiciel de programmation « *Visual C*$^{++}$ » à environnement de développement intégré. C'est l'outil qui nous a permet l'élaboration des différents programmes et sub-routines pour la gestion de toutes les fonctionnalités de l'instrument.

A l'issue de ce travail, nous avons démontré la capacité d'exécuter, à distance, toutes les fonctions de l'instrument qui n'étaient avant accessible qu'en mode local via les divers boutons de la face avant de l'instrument. Un exemple non exhaustif des fonctions testées peut inclure : La mise en marche/arrêt de l'instrument, blocage du mode local, fixation des valeurs de courant ou de tension à utiliser, affichage des valeurs générées…, etc.

On signale ici que les programmes développés revêtent l'aspect d'être universels et peuvent être facilement adaptés pour le contrôle d'un autre instrument de table à condition bien sûre qu'il soit doté d'une interface GPIB. Ainsi le système actuel, peut être étendu pour inclure d'autres instruments qui peuvent être nécessaires pour une application ou une expérience donnée.

Enfin, nous n'oublierons pas d'affirmer qu'au cours de la réalisation de ce travail, nous avons eu l'occasion de découvrir et d'apprendre à propos de nouveaux éléments matériels et de

Conclusion générale

techniques dans le domaine de l'informatique et de l'électronique. Aussi ayant effectué notre stage dans un laboratoire de recherche, nous avons également pu être en contact avec des experts dans notre domaine, travailler dans un environnement favorable au progrès et dans un esprit de persévérance.

Bibliographie

- **Ouvrages :**

[1]Jacques Guizol, les bus d'un ordinateurcours de aix marseille université 4/01/05.

[2] Magazin Ybet, cours 'le cours hardware 1',les bus utilisés par les ordinateurs.

[3]Dictionnaire technique de materiel-informatique.be: les définitions classées par thématique: les principaux composants des ordinateurs : **'vlb – vesa local bus'**consulté le : 01/07/2018 : http://www.materiel-informatique.be/vlb.php

[4]Dictionnaire technique de materiel-informatique.be: les définitions classées parthématique:les principaux composants des ordinateurs : '**le bus pci interne'** consulté le 02/07/2018 : http://www.materiel-informatique.be/pci.php

[5] Electrofriends 'Introduction to PCI protocol' **consulté le**: 10/07/2018 :http://electrofriends.com/articles/computer-science/protocol/introduction-to-pci-protocol/2/

[6] Edition National Instruments Corporation372012B-01, cours : Specification, NAT7210 IEEE 488.2 Controller Chip - National Instruments**.**

[7]Quancom information system: 'pcigpib-1- quancom pci gpib carte avec uPD7210 Chip compatible.' **Consulté le** 15/07/2018 :

http://www.quancom.de/quancom/quancom01.nsf/home_prod_fra.htm?OpenFrameSet&Frame=unten&Src=http://www.quancom.de/qprod01/fra/pb/pcigpib_1.htm

[8] Cours: 'LE BUS IEEE-488', publié le 19 decembre. 1992

[10] Cours : 'GPIB system concept-492P Programmer's.' 'gpib functions and message-492P Programmer's**.**

[11] Jacques Tichon, Christian couwenbergh, rudi giot, salvador garcia acevedo, « Techniques de l'ingénieur : communication avec les périphériques »

[13] Photo prise durant le stage de formation du 27 mai jusqu'au 31 juillet 2018 au centre de recherches nucléaire de Birine (CRNB).

[14] Texte saisi (extraits du programme exécuté pour chaque fonction).

Bibliographie

- **Manuels :**

[9] Manual English gpib1.pdf (le manuel de la carte QUANCOM GPIB).

[12] Manual programming model 220, 230, contains IEEE Programming Information keithley.

LISTE DES FIGURES

LISTE DES FIGURES

LISTE DES FIGURES