ANNEXE D4

Le Bus d'instrumentation IEEE 488 (GPIB)

1. Caractéristiques générales

Le bus IEE488 ou GPIB est un bus parallèle permettant de relier des appareils de mesure à un calculateur et possédant les caractéristiques principales suivantes :

- 15 appareils au maximum connectés ;
- Câblage en étoile ou en bus sur une longueur maximale de 20 m ;
- vitesse maximale de 1 Mo/s.

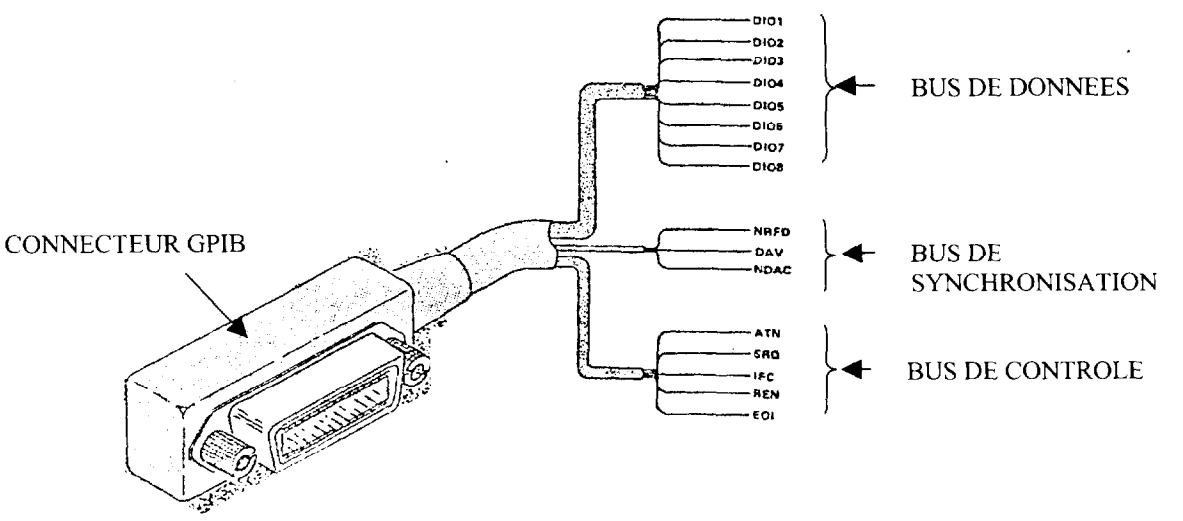
Trois fonctions sont définies sur le bus :

- La fonction **contrôleur**, en général assurée par un calculateur ou un micro-ordinateur, se charge de la gestion des transferts d'information sur le bus.
- La fonction **parleur** (*talker*) est affectée à un appareil à la fois par le contrôleur à un instant donné. Celui-ci envoie ses informations aux écouteurs.
- La fonction **écouteur** (*listener*) est attribuée à un ou plusieurs appareils par le contrôleur.

Les échanges peuvent être effectués en mode **commande** (le contrôleur commande les appareils et désigne le parleur et les écouteurs) ou en mode **transfert d'informations** (le parleur envoie ses informations vers les écouteurs)

2. Description physique

Le bus est physiquement constitué par l'ensemble des câbles de liaison qui transportent les informations d'un appareil à l'autre dans n'importe quel sens. Il s'agit d'un câblage passif de seize lignes, une masse logique, cinq blindages partiels et un blindage général.



Cet ensemble de lignes est composé de trois groupes fonctionnels :

a) ***Le bus de données*** est constitué des lignes DIO1 à DIO8 (DIO = Data In Out) qui servent à transporter les informations proprement dites. Celles-ci peuvent, suivant les circonstances être :

- Des données numériques, alphanumériques ou binaires.
- Des adresses de périphériques.
- Des commandes normalisées (multi-lignes).
- Des mots d'état (*status byte*).

b) ***Le bus de synchronisation (handshake)*** est composé de trois lignes qui transportent les signaux de contrôle. Ils garantissent la fiabilité du transfert des données circulant sur le bus suivant un protocole de type « poignée de main » *(handshake)*.
Ce protocole possède les particularités suivantes :

- Le transfert des données s'effectue de manière asynchrone. Le taux d'échange s'ajuste automatiquement à la vitesse de l'émetteur et du ou des récepteurs, plus exactement à la vitesse de l'équipement le plus lent.
- Tous les octets sont transmis sur le bus de données suivant ce protocole.
- Un ou plusieurs appareils peuvent accepter les données.
- Lorsque des commandes transitent sur le bus, l'appareil le plus lent déterminent le taux de transfert de ces ordres. Tous les équipements participent à ce protocole.
- La vitesse de transfert peut diminuer, le temps qu'un appareil prenne une lecture et la retourne ensuite pour qu'elle soit entièrement assimilée par le contrôleur.

Les trois lignes du bus sont :

NOM	DESCRIPTION
DAV	DAta Valid (donnée correcte) : cette ligne à l'état bas confirme que les informations présentes sur le bus de données sont valides et peuvent être acceptées en toute sécurité par les équipements concernés. Le contrôleur, comme tout émetteur connecté sur le bus, pilote cette ligne lorsqu'il envoie des octets.
NRFD	Not Ready For Data (pas prêt pour les données) : active à l'état bas, cette ligne permet de signaler qu'un appareil peut ou ne peut pas accepter des informations. Tous les équipements qui reçoivent des commandes, pilotent cette ligne comme les récepteurs auxquels on adresse des données.
NDAC	Not Data ACCepted (données non acceptées) : l'appareil signale, grâce à cette ligne, s'il accepte ou refuse les informations reçues. Une fois encore, tous les équipements sollicités par des commandes contrôlent ce signal, comme les récepteurs destinataires de données. La ligne NDAC ne remonte pas au niveau haut, tant que le dernier et le plus lent des équipements récepteurs n'a pas approuvé les données.

c) ***Le bus de contrôle et commande (contrôle)*** : composé de 5 lignes, il permet à un calculateur spécialisé de gérer les appareils interconnectés. Ces lignes veillent à la

circulation ordonnée des informations sur le bus. Il s'agit de commandes dites *uniligne*, puisqu'elles n'utilisent qu'un seul fil.

Ces cinq lignes sont :

NOM	DESCRIPTION
ATN	ATtentioN : lors de la validation, tous les instruments deviennent récepteur et participent à la communication. Ils doivent répondre dans un délai de 200 µs. Ce signal signifie aux équipements la présence d'un message de commande ou de donnée sur le bus. Au niveau bas, ATN prévient tous les appareils qu'une commande IEEE 488 se trouve sur le bus (un ordre ou bien une adresse). Seul le contrôleur active cette ligne.
IFC	InterFace Clear (Remise à zéro de l'interface) : cette ligne, uniquement pilotée par le contrôleur en charge, permet à ce dernier d'arrêter à tout instant (de manière asynchrone) l'opération en cours sur le bus. Tous les équipements doivent en permanence tester la ligne IFC et répondre, en cas de sollicitation, en 100 µs. Ce signal correspond au *reset* général du bus IEEE 488.
REN	Remote ENable (pilotage par le bus validé) : son niveau, exclusivement géré par le contrôleur en charge, force chaque appareil concerné à être piloté par le bus
SRQ	Service ReQuest (demande de service) : un appareil utilise cette ligne pour demander la parole et, éventuellement, interrompre l'activité du bus (mise en œuvre d'interruption possible). L'utilisation typique de ce signal, consiste à signaler la disponibilité d'une donnée ou bien avertir le contrôleur d'un problème quelconque sur un instrument. Pour déterminer l'équipement qui réclame son attention, le contrôleur effectue ce que l'on appelle un *polling* (sondage) : il interroge individuellement chaque appareil (*serial poll*) ou bien tous les récepteurs à la fois (*parallel poll*).
EOI	End Or Identify : lorsque la ligne ATN se trouve à l'état haut (FALSE), le contrôleur ou un instrument du bus peut activer la ligne EOI et la passer à l'état haut afin d'indiquer une fin de transmission. • EOI signal la fin des données. • EOI témoigne de la conduction d'un *polling* parallèle. • EOI est activée par l'instrument qui émet.

1. Adressage des appareils

Chaque appareil possède au minimum une adresse d'écouteur (*Listener*) ou de parleur (*Talker*), celle-ci est configurée matériellement (commutateurs) ou logiciellement.
L'adressage des appareils peut être assuré sur deux niveaux, primaire et secondaire. A l'adresse primaire correspond une « fonction simple » mais si celle-ci est « étendue », elle bénéficiera d'un second niveau d'adressage (secondaire). Le système comporte ainsi 31 adresses primaires et autant de secondaires. C'est au contrôleur actif qu'il revient d'adresser les appareils.

Pour cela, il émet en mode commande un octet comportant :

- un bit de parité P (poids fort) ;
- un code sur 2 bits définissant le mode ;
- 5 bits d'adresse.

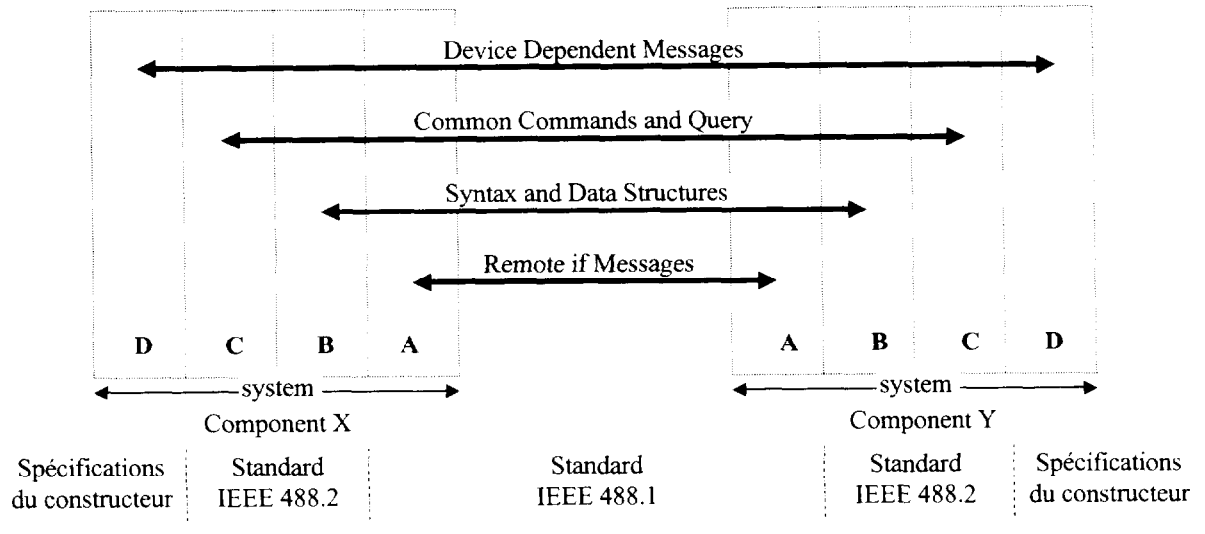
Le code sur deux bits est le suivant :

- 01 indique qu'il s'agit d'une adresse primaire d'écouteur ;
- 10 indique que l'adresse primaire est celle d'un parleur ;
- 11 est formé pour une commande secondaire ;
- 00 pour une commande auxiliaire.

Par exemple, pour un écouteur configuré à l'adresse 9, le code 41 (29 en hexadécimal et 00101001 en binaire) correspondant au caractère ASCII «) » est envoyé par le contrôleur.

Il est évident que si des appareils se voient attribuer une adresse commune, ils seront activés en même temps.
Dans le cas de commandes auxiliaires, un format sur un octet tel que P001XXXX (P étant le bit de parité) est une commande auxiliaire adressée.

4. Normalisation en couches de l'interface IEE 488



- **La couche A (*Remote if Messages*)** est la couche la plus basse de l'interface GPIB, elle décrit la partie physique du contrôleur (connecteurs, câblages, signaux électriques...) ainsi qu'un ensemble de commandes de base.
- **Les couche B et C (*Syntax and Data Structures* et *Common Commands and Query*)** représentent les fonctions de communication bas-niveau définis définies par l'IEEE 488.2.
- **La couche D (*Device Dependent Messages*)** définie la syntaxe mise en œuvre par le constructeur pour piloter son équipement, le langage SCPI (*Standard Commands for Programmable Instruments*) en est un exemple.

Les constructeurs fournissent généralement des bibliothèques de fonctions logicielles permettant de transmettre les commandes IEE 488.2 ainsi que les commandes SCPI. La librairie SICL (Standard Instrument Control Library) fournie par HP en est un exemple.

5. *Les commandes IEEE 488.1 :*

Quatre types de commande peuvent êtres transmises lorsque le bus fonctionne en mode commande (ATN = 1). A chacune de ces commandes correspond un code sur 8 bits.

- Les adresses d'écouteur (LAG : *Listen Address Group*) et de parleur (TAG : *Talken Address Group*) sont décrites précédemment.
- Les commandes multilignes universelles (UCG : *Universel Command Group*)sont reçues par tous les appareils, elles permettent de déclencher une action particulière remise à zéro d'un appareil par exemple).
- Les commandes adressées (ACG : *Addressed Command Group*) concernent les appareils préalablement initialisés comme écouteurs. Elles permettent au contrôleur d'envoyer une commande simultanée, de synchronisation par exemple.
- Les deux commandes non adressées qui peuvent être considérées comme des extensions des adresses d'écouteur et de parleur.

6. *Les commandes IEEE 488.2 :*

Au niveau supérieur, la norme IEEE 488.2 met en place un certain nombres de commandes acceptées par tous les appareils respectant le standard. Ces commandes sont envoyées sur le bus en mode données (ATN = 0) et correspondent à des chaînes de caractères ASCII (Mnemonic). Certaines commandes sont obligatoires, d'autres optionnelles.

***IDN ?**

IDeNtification query : demande à l'instrument de s'identifier par l'envoi d'un message standard, constitué de quatre champs séparés par des virgules :

- Champ 1 : fabricant Obligatoire,
- Champ 2 : modèle Obligatoire,
- Champ 3 : numéro de série ASCII "0" si non disponible (48 décimal),
- Champ 4 : révision du programme ASCII "0" si non disponible (48 décimal),

Par exemple, HEWLETT-PACKARD, 347A, 2221A01113, A1. La longueur maximale ne peut excéder 72 caractères.

***RST**

ReSeT : stoppe toutes les opérations en cours et réinitialise l'appareil dans un état prédéterminé.

***TST ?**

Self Test Query : cette commande ordonne à l'instrument de lancer une procédure d'autotest. Le résultat peut varier de -32 767 à +12 767. Un zéro indique la bonne marche des opérations. La documentation renseignera l'utilisateur sur les causes d'une réponse non nulle.

***OPC**

OPeration complete : demande à l'instrument d'armer le bit 0 dans le registre SESR à la fin du travail en cours. Ainsi, par une programmation adéquate du masque *Service Request Enable Register*, l'équipement déclenchera une demande de service lorsque la commande en cours d'exécution sera achevée.

***OPC ?**

OPeration Complete query : cette instruction impose à l'appareil de placer la valeur ASCII "1" (49 en décimal) dans son tampon de sortie, lorsqu'il a terminé tout travail en cours.

***WAI**
WAIt tu continue : cet ordre force l'appareil à attendre la fin des commandes qu'il a entreprises. Par exemple, lancer une opération de calibration, patienter jusqu'à sa fin par *WAI, puis lancer une mesure.

***CLS**
CLear Status : cet ordre remet à zéro le mot d'état ainsi que toutes les structures de données qui lui sont associées, comme l'Event status Register par exemple. Il initialise également tous les tampons, à l'exception de celui de sortie(Output Queue).

***ESE Standard**
Event status Enable : la commande *ESE permet de sélectionner la participation des événements au bit final ESB. Par exemple, *ESE 36 valide les bits 5 (*command Error*) et 2 *(Query Error*).

***ESE ?**
Event status Enable query : cet ordre lit et retourne le contenu du registre SESR. La valeur évolue entre 0 et 255.

***ESR ?**
Event status Register query : cette commande expédie à l'utilisateur la valeur associée au registre d'événement. Son décodage ultérieur renseignera sur les actions entreprises au sein de l'instrument. Sa lecture initialise à zéro ce registre.

***SRE**
Service Request Enable : cet ordre, suivi d'un masque, déterminera le ou les bits du mot d'état qui déclencheront une demande de service. Par exemple, pour permettre au bit 4 (*Message Available* MAV) de valider un SRQ, on expédiera *SRE 16.

***SRE?**
Service Request Enable query : retourne la valeur du masque de programmation expédié par la commande *SRE. La gamme s'étend de 0 à 63 ou de 128 à 191. En effet, le bit 6 (RQS) ne peut être programmé.

***STB ?** STatus Byte query : cette commande lit le mot d'état, avec le bit 6 *Master Summary Status* en lieu et place de RQS. La réponse est un entier compris entre 0 et 255.

ANNEXE D5

DOCUMENTATION DES FONCTIONS GPIB

NOM gpib - fonctions de la bibliothèque GPIB

PROTOTYPES

```
int   gpib_set_timeout(int fd, int val);
int   gpib_print(int fd, int addr, char *data);
int   gpib_input(int fd, int addr, char *buffer);
int   gpib_getbuffer(int fd, int addr, char *buffer, int count);
int   gpib_sendbuffer(int fd, int addr, char *data, int count);
int   gpib_defeoi(int fd, int c);
int   gpib_sendcmd(int fd, int cmd);
```

DESCRIPTION

Ces fonctions permettent de piloter le module GPIB et d'accéder aux données des périphériques IEEE connectés.
Avant de pouvoir être utilisées, ces fonctions doivent disposer d'un descripteur de fichier (fd) qui ne pourra être obtenu qu'après initialisation du driver (voir mgpibDrv()) et création du périphérique correspondant à la carte ou au module GPIB (voir mgpibDevCreate())…

---

int gpib_set_timeout(int fd, int val)

FONCTION: gpib_set_timeout - set the GPIB timeout counter

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)val	- timeout value in ms

RETURN:

0	- without any error
-1	- function failed, errno is set

---

int gpib_print(int fd, int addr, char *data)

FONCTION: gpib_print - send string to the addressed device

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)addr	- device address
(char *)data	- buffer address

RETURN:

0	- without any error
-1	- function failed, errno is set

---

int gpib_input(int fd, int addr, char *buffer)

FONCTION: gpib_input - read string from the addressed device

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)addr	- device address
(char *)buffer	- buffer address

RETURN:

>= 0	- # of bytes
-1	- function failed, errno is set

int gpib_getbuffer(int fd, int addr, char *buffer, int count)

FONCTION: gpib_getbuffer - read data from the addressed device

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)addr	- device address
(char *)buffer	- buffer address
(int)count	- # of bytes to read

RETURN:

0	- without any error
-1	- function failed, errno is set

---

int gpib_sendbuffer(int fd, int addr, char *data, int count)

FONCTION: gpib_sendbuffer - send buffer to the addressed device

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)addr	- device address
(char *)data	- buffer to send to the device
(int)count	- # of bytes to write

RETURN:

0	- without any error
-1	- function failed, errno is set

---

int gpib defeoi(int fd, int c)

FONCTION: gpib_defeoi - define new EOI character

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)c	- EOI character

RETURN:

0	- without any error
-1	- function failed, errno is set

---

int gpib_sendcmd(int fd, int cmd)

FONCTION: send a command, user has to include device address if necessary.

PARAMETER:

(int)fd	- file descriptor obtained from an open
(int)cmd	- command (IEC_UNLISTEN, IEC_LISTEN, IEC_SDC ...)

RETURN:

0	- without any error
-1	- function failed, errno is set
