PILOTE pour WINDOWS
MANUEL DE L'UTILISATEUR

Gammes MCX & WAN-HDLC

www.acksys.fr
support@acksys.fr
sales@acksys.fr

Mars 2016 (révision B.6)
DT055

Gammes MCX & WAN-HDLC
Pilotes de périphérique pourWINDOWS

MARQUES DEPOSEES ®

- *ACKSYS* est une marque déposée de *ACKSYS*.
- Windows 7, Windows Vista, Windows XP, Windows NT, Windows 2000, Windows 2003 Server, Windows Seven, MS-DOS, Windows 95 sont des marques déposées de MICROSOFT.

ACKSYS
COMMUNICATIONS & SYSTEMS
ZA Val Joyeux
10, rue des Entrepreneurs
78450 VILLEPREUX
FRANCE

Telephone:	+33 (0)1 30 56 46 46
Fax:	+33 (0)1 30 56 12 95
Web:	www.acksys.fr
Hotline:	support@acksys.fr
Sales:	sales@acksys.fr

NOTICE

Le présent document n'est pas contractuel. ACKSYS ne garantit en aucune façon son contenu et dégage toute responsabilité quant à la rentabilité des produits décrits et leur conformité aux besoins de l'utilisateur. ACKSYS ne pourra en aucun cas être tenu pour responsable des erreurs éventuellement contenues dans ce document, ni des dommages, quelle qu'en soit l'importance, du fait de la fourniture, du fonctionnement ou de l'utilisation des produits.
Dans son effort pour améliorer constamment sa documentation, ACKSYS se réserve le droit de réviser périodiquement ce document, ou de changer tout ou partie de son contenu, sans aucune obligation pour ACKSYS d'en aviser qui que ce soit.

TABLES DES MATIERES

PRESENTATION

Ce pilote permet d'utiliser toutes les fonctionnalités des cartes des gammes MCX (bus ISA, bus PCI 3V3, bus PCI 5V et bus cPCI), sur les systèmes Windows.

Il existe quatre versions de ce pilote.

- La version pour processeurs 64 bits, de Windows 7.
- La version pour les processeurs 32 bits, de Windows XP à Windows 7, dont Windows 2003.
- Une version séparée pour Windows 2000. Son développement est arrêté à la version 3.2.4 et elle ne fonctionne pas sur les processeurs multicore.
- La version « non Plug & Play » est conçue pour Windows NT 4.0.

Ce pilote permet, soit de télécharger une application sur la carte, soit d'utiliser la carte en respectant l'API des communications asynchrones de Windows. Il est compatible avec la plupart des applications écrites pour ce système et ses successeurs (par exemple HyperTerminal).

Ce manuel décrit la version 3.4.6 du pilote Plug & Play, et la version 2.2.14 du pilote non Plug & Play, utilisée avec les dernières versions des EPROM des cartes. Nous nous efforçons de maintenir une compatibilité ascendante entre les versions. Si vous utilisez une carte ou un pilote ancien, certaines caractéristiques peuvent différer de ce qui est exposé dans ce manuel.

Rappel sur la gamme MCX :

Une carte est constituée d'une **carte mère** (dépendante du bus du P.C) et d'une **carte mezzanine**.

Bus	Carte mère	Carte mezzanine
PCI 3.3V et 5V	MCXUNI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
PCI 5V	MCXPCI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
cCPI 6U	MCXcPCI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
ISA	MCX	MCXBP ou MCXBPMR
ISA	MCX-Lite	Lite/U Lite/S Lite/570 Lite/104 Lite/485

Dans tout ce manuel, le nom générique MCX désigne indifféremment l'une quelconque de ces cartes.

I SPECIFICATIONS TECHNIQUES

I.1 Caractéristiques générales.

En ce qui concerne les vitesses de transmission supportées, se reporter à la description de GetCommProperties (chapitre « MODES DE COMPATIBILITE COM ») et au manuel du « firmware » utilisé (Logiciel de Base [DT002] ou Logiciel Multiprotocole [DT003]).

En ce qui concerne les signaux disponibles sur les jonctions, se reporter au manuel hardware/installation de la carte (voir paragraphe II).

Pilotes pour Windows (environnement Plug & Play)

Système d'exploitation	Windows 7 pour processeurs 64 bits Tous les Windows entre Windows 2000 et Windows 7 pour processeurs 32 bits.
Type de cartes	Toutes sauf les cartes pour bus ISA
Nombre de cartes	Limité par le nombre de slots disponibles
Logiciel de base	Version 2.7 ou postérieure
Option Multiprotocole	Version 3.1 ou postérieure
Bus supportés	PCI 5V, PCI 3V, CompactPCI 6U

Pilote pour Windows NT 4.0 (environnement non Plug & Play)

Attention, ce pilote fonctionne aussi pour les systèmes d'exploitation Windows Plug & Play mais il n'est toutefois pas conseillé de l'utiliser. Ceci simplement à cause de son intégration dans le système qui n'est pas celle d'un driver Windows standard : En effet, il apparaît dans les périphériques cachés non Plug & Play, les ports COM n'apparaissent pas dans le gestionnaire de périphériques et il laisse apparaître un conflit avec le périphérique Plug & Play non installé relatif à la carte.

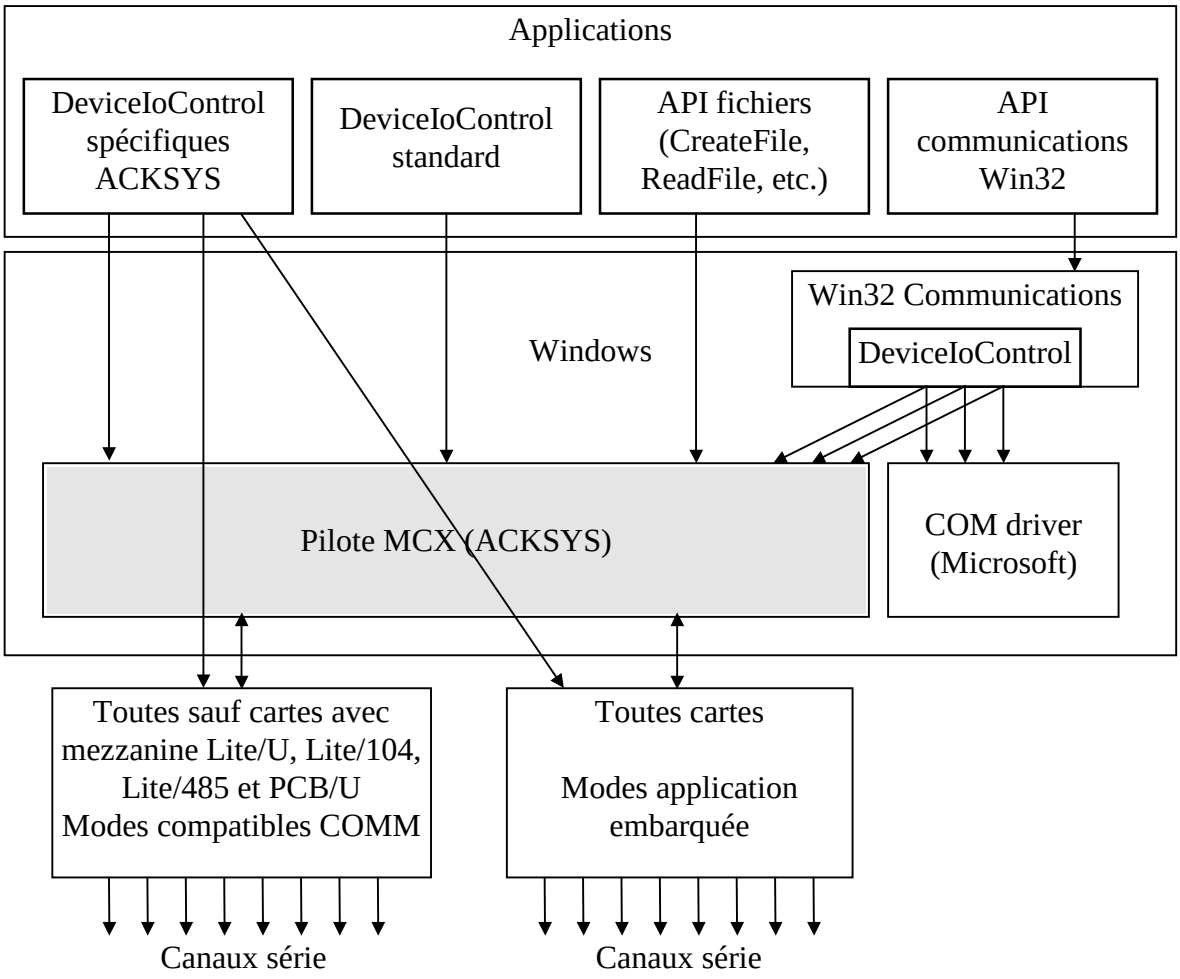
Système d'exploitation	Windows NT 4.0 SP3 au moins
Limitations	Multiprocesseurs et Hyperthreading non supportés.
Type de cartes	Toutes y compris bus ISA
Nombre de cartes PCI ou cPCI	Limité par le nombre de slots PCI disponibles
Nombre de cartes ISA	Limité par les ressources ISA disponibles (en général, 1 à 4 cartes)
Logiciel de base carte PCI/cPCI	Version 2.7 ou postérieure
Logiciel de base carte ISA	Version 2.0 ou postérieure
Logiciel de base carte MCC	Version 3.8 ou postérieure
Option Multiprotocole	Version 3.1 ou postérieure
Bus supportés	PCI 5V, PCI 3V, CompactPCI 6U, ISA

I.2 Intégration dans l'environnement Windows.

Le pilote permet :

- d'accéder directement aux ressources de la carte (mémoire double accès, FIFO, registres d'entrée-sortie) dans le cas d'applications spécifiques embarquées dans la carte ;
- d'utiliser des ports série intelligents (transmissions asynchrone, asynchrone synchronisée, synchrone orientée bit, synchrone orientée caractère), en respectant les interfaces définies dans Win32 (COMM API) pour les ports série traditionnels. Utilisé en mode asynchrone, le pilote tend à imiter le plus complètement possible le pilote du port COM1 fourni par Microsoft. Cependant, quelques différences subsistent, du fait qu'une grande partie du traitement est déportée vers la carte.

Voici comment le pilote s'insère dans l'architecture Windows :



Ce schéma montre que le pilote (« driver ») utilise les mêmes API que les ports COM. Si une application spécifique nécessite des caractéristiques offertes par la carte mais pas par l'API Win32, elles peuvent accéder directement au pilote ou même à la carte par l'intermédiaire des commandes DeviceIoControl décrites dans les chapitres MODES DE COMPATIBILITE COM et MODE MCXDOS/AUTOMCX.

Des exemples typiques de ce genre d'utilisation sont le passage en mode RS422 ou en mode synchrone.

II DOCUMENTATION COMPLEMENTAIRE

Différentes documentations techniques accompagnent les cartes de la gamme MCX, en fonction des options que vous avez choisies et donc de la façon dont vous allez utiliser votre carte. Ce manuel y fait parfois référence.

Vous trouverez toutes ces documentations sur le CDROM livré avec la carte. Celles ci sont classées de la façon suivante :

- Une documentation spécifique à la carte qui décrit le hardware :
  - ♦ caractéristiques physiques et électriques
  - ♦ brochage des connecteurs
  - ♦ signification des interrupteurs et cavaliers
  - ♦ programmation des périphériques internes.
- Une documentation sur le firmware des cartes. Il existe pour chacun des firmwares, firmware de base et firmware multiprotocole, une documentation :
  - ♦ Manuel d'utilisation du logiciel de base
  - ♦ Manuel d'utilisation du logiciel multiprotocole

  Ces manuels à l'usage du programmeur système, décrivent les paramètres et procédures utilisés avec la carte pour la manipulation des canaux de communication.

Le présent manuel qui décrit pour toute la gamme MCX le pilote de périphérique Windows.

Pour faciliter vos recherches, voici les endroits où trouver les principales informations :

Pour vous informer sur...	**Consultez d'abord...**
L'installation de la carte	• le chapitre « Installation » du présent manuel • deux chapitres du manuel hardware de votre carte : - Le chapitre consacré à la carte mère - Le chapitre décrivant à la carte mezzanine (MCXBP, etc.)
Les protocoles supportés par MCX-MULTIPROTOCOLE	« Manuel d'utilisation du logiciel Multiprotocole »
Les voyants sur la carte	• au démarrage : manuel hardware de votre carte • en cours de fonctionnement : suivant le cas, « Manuel d'utilisation du logiciel Multiprotocole » ou « Manuel d'utilisation du logiciel de Base » • voyants des extensions : chapitre concernant l'extension dans le manuel hardware de votre carte
La structure de la boîte aux lettres dans les modes Logiciel de base ou Multiprotocole	suivant le cas, « Manuel d'utilisation du logiciel Multiprotocole » ou « Manuel d'utilisation du logiciel de Base »
MCXDOS	• « Guide d'utilisation du logiciel MCXDOS » • diverses sections « Programmation » du « Manuel d'installation et caractéristiques techniques des cartes de la gamme MCX », ou bien, • section « Pour programmer dans la carte » dans le manuel hardware de votre carte
L'écriture de nouveaux drivers	• pour les communications limitées à l'asynchrone, « Manuel d'utilisation du logiciel de Base », sinon « Manuel d'utilisation du logiciel Multiprotocole » • section « Pour écrire un pilote de périphérique » dans le manuel hardware de votre carte
La programmation des interfaces électriques, des formats de transmission, des vitesses supportées	• le chapitre « Mode de compatibilité COM » de ce manuel • les commandes VINIT et RSMDE dans le « Manuel d'utilisation du logiciel de Base » • les commandes VINIT, PROTO et RSMDE dans le « Manuel d'utilisation du logiciel Multiprotocole »
Les connecteurs, les signaux disponibles sur les jonctions	section concernant l'extension utilisée, dans le manuel hardware de votre carte
Les compatibilités ou incompatibilités logicielles	le chapitre « Mode de compatibilité COM » de ce manuel

III QUELQUES PRODUITS ACKSYS CONNEXES

Matériels

Référence	Voies	Vitesses max.	Principales caractéristiques
MCX(c)UNI/S, MCX-*Lite*/S	2	1 Mbaud	Asynchrone/Synchrone, RS232/RS422
MCX(c) UNI /570-x, MCX-*Lite*/570-x	2/4	4 Mbaud	Asynchrone/Synchrone, Interfaces électriques multiples
MCX(c) UNI /BP-xx, MCX-xx...	8,16...64	1Mbaud	Asynchrone/Synchrone, RS232/RS422
MCX(c) UNI /BPMR-xx	8,16...64	1Mbaud	Asynchrone/Synchrone, RS232 isolé, RS422/485 isolé et boucle de courant
WAN-HDLC/4(C)	4	4 Mbaud	Synchrone, Interfaces électriques multiples
MCX-*Lite*/485	2	375/500 kbaud	Isolation galvanique RS232/422/BdC sous MCXDOS
MCX-*Lite*/U	4/8	115200	RS232/RS422/BdC sous MCXDOS
MCX(c)PCI/U	4/8	1,8Mbaud	RS232/RS422/BdC sous MCXDOS
MCX-*Lite*/104	1	115200	Bus PC104 sous MCXDOS

Logiciels

Référence	Caractéristiques
MCXDOS	Partage de ressources avec le PC (clavier, écran, souris, disque, réseau) Fonctionne sous MS-DOS et Windows 95 Permet le développement d'applications téléchargées sur la carte MCX
AUTOMCX	Chargeur d'applications MCXDOS sous Windows Vista/XP/2000/NT. Existe aussi pour MS-DOS et Windows 95.

Quelques options

Référence	Caractéristiques
MCXBP85230	Boîtier de connexion 8 canaux pour carte MCX-00 ou MCXPCI/BP-00 équipé d'USART 85C230, qui contiennent en particulier des FIFO étendus d'émission/réception
MCX-RAM2/8	Kit d'extension mémoire 2 ou 8 Mo pour cartes ISA
MCX-PCMCIA	Deux emplacements PCMCIA accessibles par la MCX-00 sous MCXDOS
MCX-PLQ	Plaque métallique (raidisseur) pour fixation des MCXBP sur un mur
MCXBP-RACK	Face avant 19" pour fixation des MCXBP munis de MCXBP-PLQ (environ 1U par MCXBP)
RS485, Boucle de courant	Consulter notre catalogue d'adaptateurs d'interface

INSTALLATION

I MODES D'UTILISATION DE LA CARTE.

IMPORTANT : avant toute utilisation d'une carte, vous devez choisir le MODE D'UTILISATION souhaité, en fonction des besoins particuliers de votre application.

Les cartes peuvent être utilisées dans quatre modes différents.

- ☞ Mode LOGICIEL DE BASE
  Ce mode permet une compatibilité ascendante avec des cartes plus anciennes (MCC-8, MCC-16 et MCC/II). Il permet de piloter jusqu'à 64 canaux série asynchrones.
- ☞ Mode MULTIPROTOCOLE (en option sur certaines cartes)
  Ce mode offre une compatibilité réduite avec le Logiciel de Base. Il permet de piloter de 2 à 64 canaux série asynchrones; de plus il permet de définir des protocoles synchrones avec une interface d'application proche de celle de COM1/2. Il prend en charge, outre les cartes supportées par le Logiciel de Base, les cartes avec carte mezzanine PCB/570 ou Lite/570.
- ☞ Mode MCXDOS/AUTOMCX
  Ce mode permet, après avoir développé une application s'exécutant dans la carte en environnement DOS, de télécharger DOS et l'application dans la carte à partir de Windows. Le développement proprement dit de l'application, doit être mené sur un poste de travail DOS ou compatible (Windows 9x, etc.).
- ☞ Mode BIOS
  Ce mode force la carte à exécuter ses extensions BIOS, ce qui a pour effet, soit d'exécuter une application spécifique qui doit être préalablement figée dans sa FLASH EPROM, soit de charger le système d'exploitation installé sur son disque dur embarqué.

Configuration.

Le choix du mode de fonctionnement se fait à deux niveaux : les interrupteurs et cavaliers sur la carte permettent de déterminer son comportement à la mise sous tension ou après une réinitialisation (donc avant qu'aucun pilote ne soit lancé) ; les options de l'utilitaire MCXSETUP permettent de déterminer comment la carte va interagir avec l'application.

Pour connaître la signification des interrupteurs et cavaliers des cartes, voir le manuel hardware.

Sur les cartes PCI et cPCI, la position des interrupteurs et cavaliers est aussi indiquée sur le film plastique transparent qui protège la carte mezzanine.

Mode	Bus	Carte mezzanine	Interrupteurs et cavaliers	Fonctionnalités
Logiciel de base	ISA	MCXBP MCXBPMR Lite/S	ST2/ST3 pos. 1-2	Compatibilité COM1/2 Compatibilité MCC
	PCI	MCXBP MCXBPMR PCB/S	SW1 pos. Firmware	
	cCPI	MCXBP MCXBPMR PCB/S	JP2 pos. Firmware	
Multiprotocole	ISA	MCXBP Lite/S Lite/570	ST2/ST3 pos. 1-2	Compatibilité COM1/2 Protocoles synchrones
	PCI	MCXBP MCXBPMR PCB/S PCB/570	SW1 pos. Firmware	
	cPCI	MCXBP MCXBPMR PCB/S PCB/570	JP2 pos. Firmware	
Mcxdos/automcx	ISA	Toutes	ST2/ST3 pos. 2-3	Pas de compatibilité COM1/2 Exécution de MS-DOS et d'applications spécifiques sur la carte Interface coté PC à spécifier par l'application
	PCI	Toutes	SW1 pos. Mcxdos	
	cPCI	Toutes	JP2 pos. Mcxdos	
Extension BIOS	ISA	Toutes	ST2/ST3 pos. 2-3	Pas de compatibilité COM1/2 Chargement d'un système d'exploitation quelconque sur la carte à partir de son disque embarqué Démarrage de la carte sur commande de l'application côté PC
	PCI	Toutes	SW1 pos. Standalone	
	cPCI	Toutes	JP2 pos. Standalone	

II INSTALLATION DU PILOTE PLUG & PLAY

Notez que les boîtes de dialogue données en exemple sont celles de Windows XP. Les mêmes dialogues s'afficheront dans Windows Vista, avec une présentation légèrement différente.

Pour Windows 2000 assurez-vous que vous installez la version qui lui est dédiée (les versions plus récentes ne s'installeront pas).

II.1 Choix du mode de fonctionnement

Reportez-vous à la section I. ATTENTION : le pilote Plug&Play ne supporte pas les cartes sur bus ISA.

II.2 Installation physique de la carte

ATTENTION : Sous Windows 2000, vous devez installer la carte dans l'ordinateur après avoir exécuté le programme d'installation. Sous des Windows plus récents, vous devez installer la carte dans l'ordinateur avant de lancer le programme d'installation, pour que celui-ci puisse reconnaître la présence de la carte et les ressources attribuées automatiquement.

II.3 Redémarrage de l'ordinateur

Au redémarrage, Windows détecte la présence de la carte. L'assistant d'installation s'affiche. Ignorez-le (vous pouvez soit l'annuler, soit le laisser tel quel, il disparaitra au cours de l'installation).

Placez le CD Acksys dans le lecteur, naviguez jusqu'au répertoire du pilote puis lancez le programme nommé « MCXINSTALL.EXE ».

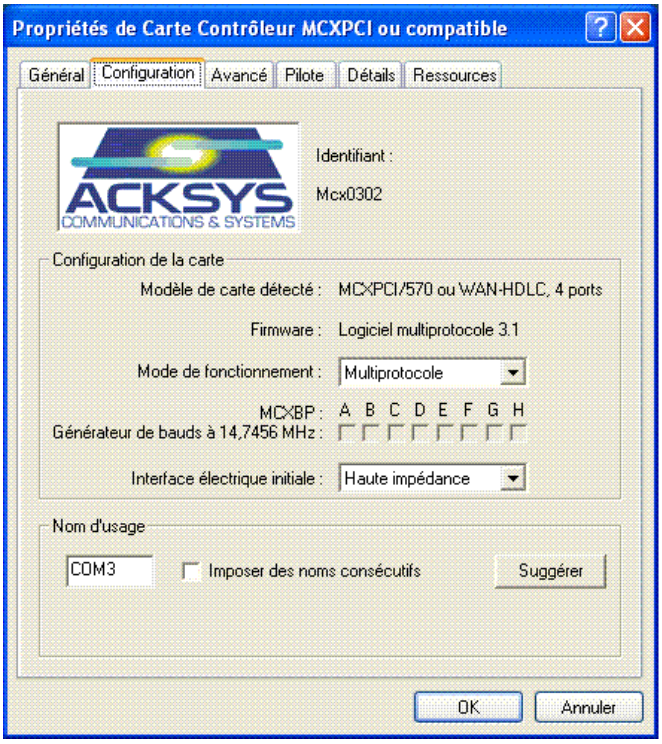
Après installation, les cartes apparaissent dans le gestionnaire de périphériques, dans le groupe « cartes série multi-ports ». Cette icône vous permet de désactiver ou de configurer la carte.

Pour les cartes dont les interrupteurs sont en position « built-in firmware » (cas le plus courant), les ports apparaissent dans le groupe « Ports (COM et LPT) ».

II.4 Configuration des propriétés des cartes installées

Les propriétés de la carte contient deux onglets particuliers : « configuration » et « avancé ».

L'onglet **Configuration** affiche quelques informations sur la carte et permet d'indiquer ses caractéristiques générales.



Propriétés générales d'une carte

- **Modèle**

  Modèle de carte, détecté par le pilote.

- **Mode de fonctionnement**

  Indique au pilote le mode de fonctionnement qui a été choisi (voir la section I). Seuls les modes de fonctionnement supportés sur le modèle choisi apparaissent.
  Normalement cette case est pré-remplie en fonction de la position d'interrupteurs détectée. Il est cependant possible de forcer le choix pour des applications particulières (par exemple, mise à jour du firmware de la carte).
  Si le mode Multiprotocole est sélectionné alors que l'option n'est pas installée dans l'EPROM de la carte, à son démarrage le pilote enregistrera une erreur dans l'Observateur d'Événements.

- **Générateur de bauds à 14,7456 MHz**

  Sur les cartes avec mezzanines MCXBP(MR) ou PCB/S, chaque bloc de canaux dispose de deux oscillateurs pour la génération des horloges de transmission. Un seul des deux oscillateurs peut être utilisé à un moment donné. L'oscillateur à utiliser doit être choisi à l'installation, en fonction de la précision souhaitée sur le calcul des horloges. Il sera alors utilisé par tous les canaux du même bloc. Cette précision peut devenir significative dans les modes synchrones, ou pour les hauts débits (supérieurs à 19200 bauds).

Les cartes avec mezzanine MCXBP ou MCXBPMR ont un à huit blocs (A à H) de 8 canaux chacun. Les cartes avec mezzanine PCB/S ont un seul bloc (A) de deux canaux. Lorsqu'une case est cochée, tous les canaux du bloc correspondant utilisent l'oscillateur à 14,7456 MHz. Sinon, ils utilisent l'oscillateur à 16 MHz.
Par souci de compatibilité, les cartes avec mezzanine PCB/570 récentes supportent ces cases à cocher bien qu'elles n'aient qu'un seul oscillateur ; elles recalculent leurs horloges comme si elles avaient deux oscillateurs.

- **Interface électrique initiale**

  Sur certaines cartes, l'interface électrique est programmable. Sur ces cartes, lorsque l'option Multiprotocole est active, il est possible d'indiquer ici quelle interface électrique le pilote doit utiliser au démarrage sur l'ensemble des canaux (avant même que les ports soient utilisés).

  ATTENTION : Par défaut l'interface électrique des mezzanines PCB/570 est en HAUTE IMPEDANCE, pour éviter les conflits électriques avec les appareils connectés. Vous devez modifier l'interface électrique avant d'utiliser les ports, soit globalement par cette liste de sélection, soit port par port avec l'API fournie, décrite plus loin.

  ATTENTION : cette option n'a aucun effet avec les extensions MCX-BPMR.

- **Nom d'usage**

  En mode MCXDOS/Automcx seulement, nom qui sera utilisé par les applications pour accéder à cette carte. En mode de compatibilité COM les noms des ports sont définis par Windows. Attention, ils ne sont pas nécessairement consécutifs.

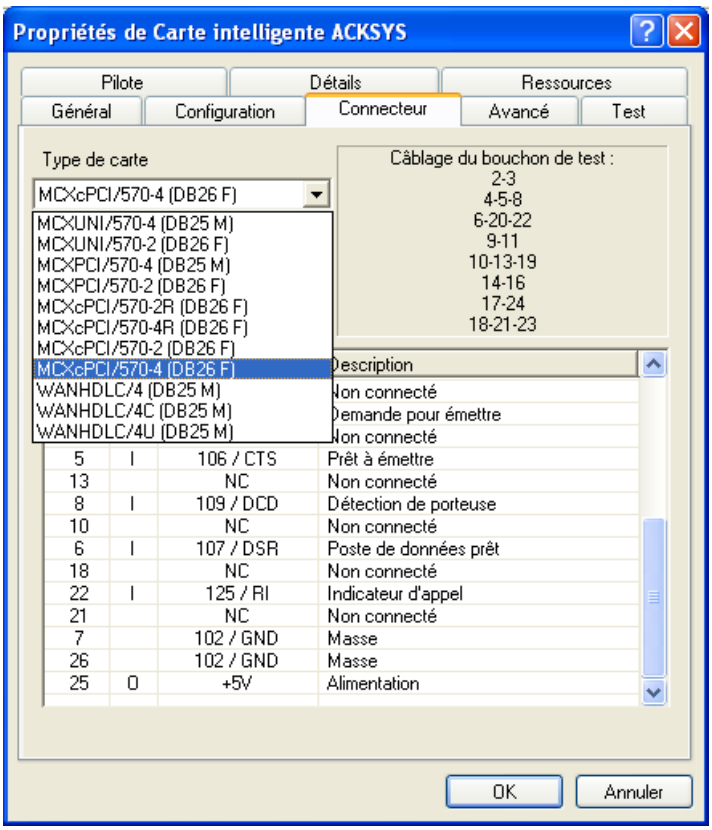
- **Imposer des noms consécutifs**

  En mode de compatibilité COM les noms de port seront redéfinis avec des numéros consécutifs à partir de celui indiqué. Un message d'erreur indique si le nom indiqué n'est pas valide ou si la plage de numéros recouvre un port existant (defini pour une autre carte).

- **Suggérer**

  En mode de compatibilité COM seulement, génère le nom du premier port COM disponible. Si la case « imposer des noms consécutifs » est cochée, le nom généré tient compte de cette contrainte.

L'onglet **Connecteur** fournit le pinout et le format des connecteurs des ports série



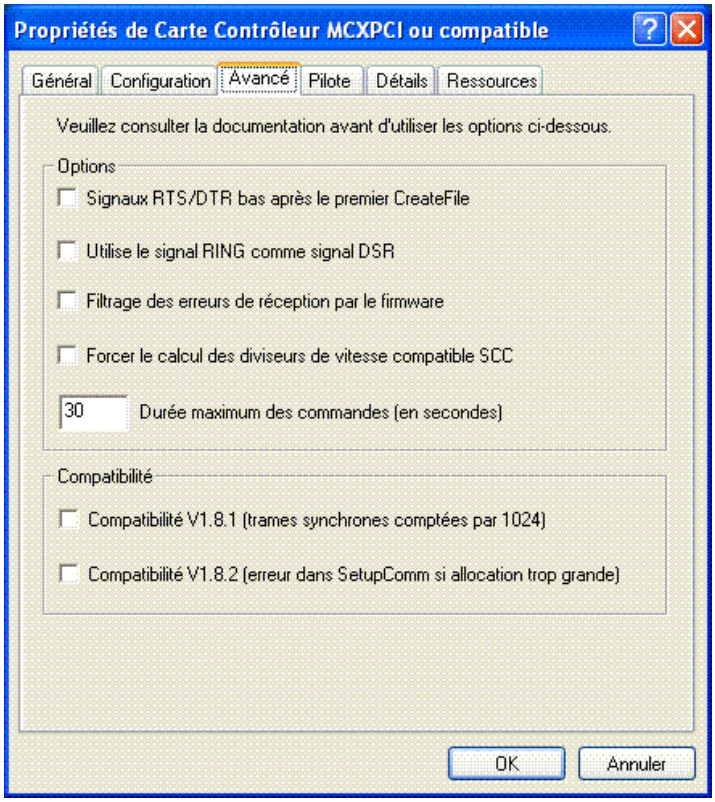
Vous disposez ainsi des informations nécessaires pour réaliser votre câble de raccordement.

L'onglet **Avancé** permet de gérer les options et la compatibilité avec les versions antérieures.

Ces options agissent sur tous les canaux de la carte.

Attention : Pour des raisons de performances, les modes de compatibilité ne seront pas supportés indéfiniment ; il est donc recommandé d'adapter les logiciels en conséquence.

Attention : A compter de la version 3.1.4, les options de compatibilité avec les versions antérieures à 1.8 ont été supprimées, ainsi que les options liées aux cartes à bus ISA.



- **Signaux RTS/DTR bas après le premier CreateFile**

  Intérêt : au démarrage, ce pilote, par compatibilité avec le pilote de ports série Microsoft™, assigne les valeurs RTS_CONTROL_ENABLE et DTR_CONTROL_ENABLE aux champs *fRtsControl* et *fDtrControl* de la structure DCB. En conséquence, la première ouverture d'un canal après le démarrage du pilote, fait automatiquement monter les signaux RTS et DTR. Ce comportement peut être gênant dans certaines applications.

  Case cochée : au démarrage, le pilote assigne aux champs *fRtsControl* et *fDtrControl* de la structure DCB les valeurs par défaut RTS_CONTROL_DISABLE et DTR_CONTROL_DISABLE. Les signaux RTS et DTR resteront inactifs jusqu'à un changement volontaire par l'application.

- **Utiliser le signal RING comme signal DSR**

  Intérêt : certaines cartes ne disposent pas de la broche DSR sur le connecteur RS232. Certaines applications utilisent ce signal.

  Case cochée : le pilote intervertit la signification des broches DSR et RING. Le signal détecté sur la broche RING est remonté à l'application comme étant un signal DSR (et réciproquement pour les cartes qui disposent de la broche DSR). Les actions de l'application censées agir sur DSR agissent sur le signal RING à la place (par exemple la modification de *fDsrSensitivity*).

  Remarque : il est aussi possible de choisir cette interversion séparément pour chaque canal (voir la section « Manuel de référence »).

- **Filtrage des erreurs de réception par le firmware**

  Intérêt : certaines applications inhibent la remontée des erreurs de réception en supprimant le bit IT5 de la commande MINTR. Cependant le pilote revalide la remontée des erreurs de réception à l'occasion de certaines opérations.

  Case cochée : le pilote inhibe la remontée des erreurs de réception en effaçant toujours le bit IT5 de la commande MINTR.

- **Forcer le calcul des diviseurs de vitesse compatible SCC**

  Intérêt : Les versions récentes du firmware calculent elles-même les paramètres de leurs générateurs de bauds, en fonction de la vitesse demandée. Ce calcul était auparavant effectué par le pilote en utilisant une commande moins sophistiquée mais moins précise du firmware. Il peut en résulter de légères différences entre les vitesses calculées sur des cartes équipées de fimwares différents.

  Case cochée : le pilote utilise systématiquement l'ancienne méthode de calcul.

- **Durée max. des commandes**

  Pour détecter les pannes éventuelles, le pilote chronomètre la durée de certains groupes de commandes envoyées à la carte. Une erreur est signalée à l'application, et un message est placé dans l'Observateur d'événements, si cette durée excède celle indiquée ici. Ce cas d'erreur est rare et la valeur par défaut (30 secondes) ne devrait normalement pas être changée. Les valeurs permises vont de 5 à 3600 secondes.

- **Compatibilité V1.8.1 (trames synchrones comptées par 1024)**

  Versions 1.8.1 et antérieures :

  sur les canaux utilisés dans un mode synchrone, les champs *cbInQue* et *cbOutQue* de la structure COMSTAT contenaient un nombre de trames multiplié par 1024.

  Versions postérieures à 1.8.1 :

  ces champs contiennent un nombre de trames.

  Intérêt : l'écriture d'application est simplifiée. D'autre part l'ancienne limite de 1024 octets par trame synchrone, qui justifiait cette bizarrerie, n'existe plus.

  Case cochée : rétablit le comportement antérieur.

- **Compatibilité V1.8.2 (erreur dans SetupComm si...)**

  Versions antérieures à 1.8.3 :

  Utiliser SetupComm() avec des tampons trop grands renvoyait une erreur à l'application.

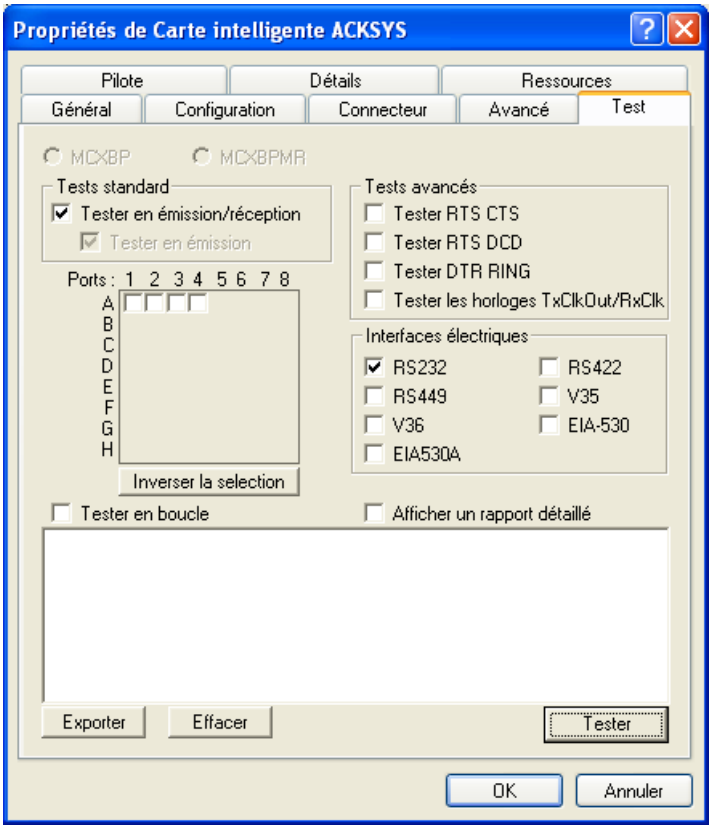
  Versions 1.8.3 et postérieures :

  Aucune erreur n'est signalée dans ce cas.

  Intérêt : La documentation Win32 précise que les paramètres de SetupComm sont indicatifs et ne renvoie pas d'erreur si ceux-ci ne peuvent pas être respectés.

  Case cochée : rétablit le comportement antérieur (avec remontée d'erreurs).

L'onglet **Test** permet de tester chaque port



III INSTALLATION DU PILOTE NON PLUG & PLAY SOUS WINDOWS NT

III.1 Contrôle de la version de Windows NT

Le pilote fourni ne fonctionne qu'à partir de **Windows NT 4 Service Pack 3**. Contrôlez que c'est bien la version installée. Sur les versions antérieures à Service Pack 3, un défaut du noyau du système empêche le démarrage du pilote.

Ce pilote ne fonctionne pas sur des unités centrales multiprocesseur, dual core ou Hyperthreading. Désactivez ces modes dans le BIOS, le cas échéant.

III.2 Choix du mode de fonctionnement.

Reportez-vous à la section I.

III.3 Installation d'une carte pour bus PCI ou cPCI

Choix des ressources attribuées à la carte.

Les ressources nécessaires aux cartes PCI sont déterminées automatiquement. Par contre vous devez contrôler que le mode de fonctionnement choisi correspond bien à la position des interrupteurs SW1. Pour cela reportez-vous au « manuel hardware » de la carte à installer.

Installation physique de la carte

Vous devez installer la carte dans l'ordinateur avant de lancer le programme d'installation, pour que celui-ci puisse reconnaître la présence de la carte et les ressources attribuées automatiquement.

Redémarrage de l'ordinateur

Windows NT 4 n'est pas plug-and-play et ne détecte donc pas de lui-même la présence de la carte. Cependant certains BIOS affichent fugitivement la liste des cartes PCI installées, avant de lancer le système d'exploitation. La carte apparaît comme une carte de communication avec un code VENDORID de 1528 et un DEVICEID de 0800.

Installation et configuration du pilote.

Reportez-vous à la section III.5. Lorsque le programme d'installation « MCXSETUP » est lancé, les cartes présentes sont automatiquement détectées et apparaissent avec un icône rouge dans la liste des cartes, pour signifier qu'elles doivent être configurées avant utilisation.

III.4 Installation d'une carte pour bus ISA

Choix des ressources attribuées à la carte.

Pour chaque carte, vous devez choisir une interruption, un port d'Entrée-Sortie et une plage d'adresses mémoire de 32 k octets. Ces ressources ne doivent bien sûr pas être utilisées par d'autres périphériques. Reportez-vous à [DT004] pour placer les interrupteurs et les cavaliers de la carte conformément à votre choix.

Remarque sur les BIOS PNP (« Plug-and-Play ») :

Les micro-ordinateurs récents sont équipés d'un BIOS dit « Plug-and-Play ». Ce type de BIOS se réserve par défaut, la totalité des ressources (Interruptions, adresses mémoire, adresses d'entrée/sortie) pour les attribuer aux cartes respectant les spécifications « Plug-and-Play ». Dans l'écran de SETUP du BIOS, contrôlez que les ressources requises par la carte sont bien attribuées au bus ISA. Contrôlez aussi les conflits avec d'autres cartes ou des périphériques intégrés, comme certains ports souris. Vous devrez donc éventuellement vous reporter aux manuels de vos cartes additionnelles et au manuel d'utilisation de votre BIOS.

Installation physique de la carte.

Il est préférable d'installer la carte dans l'ordinateur avant de lancer le programme d'installation, pour que celui-ci puisse la tester dès que les ressources auront été indiquées.

Installation et configuration du pilote.

Reportez-vous à la section III.5. Lorsque le programme d'installation « MCXSETUP » est lancé après l'installation d'une nouvelle carte ISA, celle-ci **n'apparaît pas** dans la liste des cartes, jusqu'à ce que vous l'ayez configurée avec le bouton **Ajouter**.

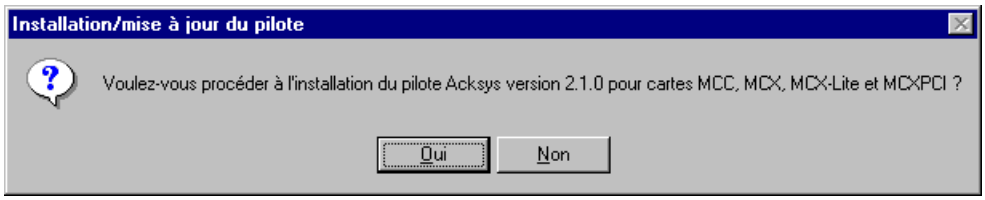
III.5 Programme d'installation « MCXSETUP »

Installation du pilote.

Après avoir placé la carte dans l'ordinateur et redémarré le système d'exploitation, placez le CD Acksys dans le lecteur, naviguez jusqu'au répertoire du pilote puis exécutez la commande :

SETUP . BAT

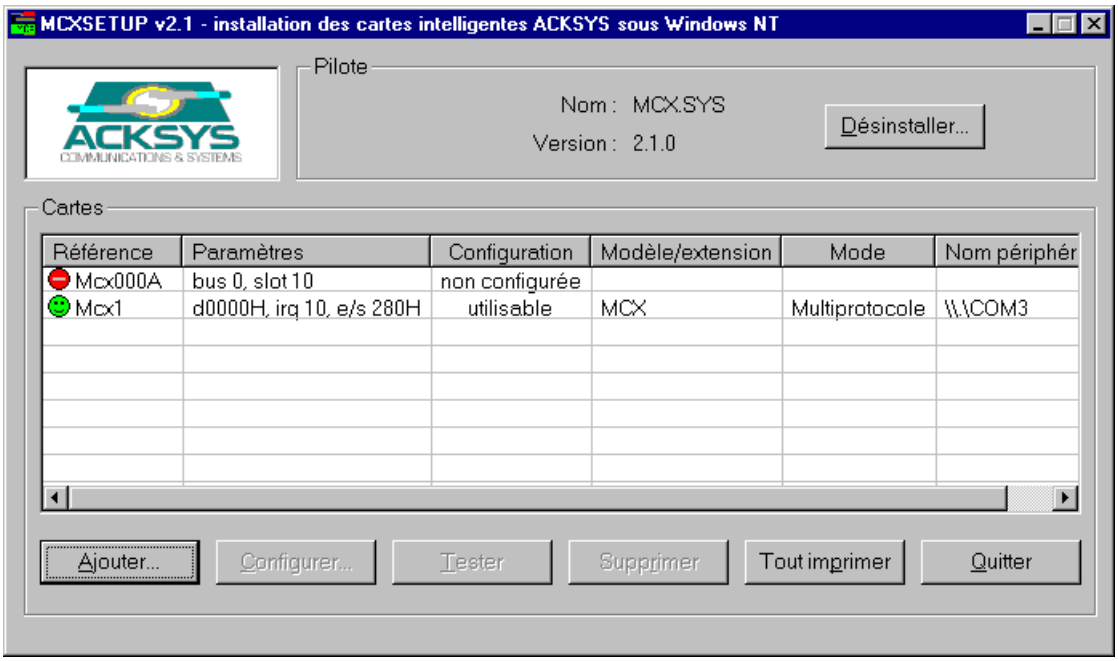
Ce script lance le programme d'installation et de configuration nommé « MCXSETUP ». Lors de sa première exécution, il détecte que le fichier pilote MCX.SYS n'est pas installé et demande une confirmation pour réaliser l'installation :



Si une version antérieure du pilote est déjà installée sur le disque, MCXSETUP contrôle que la version à installer est postérieure, et que la mise à jour est possible, avant de procéder à l'installation.

Il installe ensuite le pilote de périphériques « MCX.SYS » et quelques utilitaires, dont lui-même, sur le disque dur. Par la suite vous pourrez lancer ces utilitaires directement à partir de l'invite de commandes, ou à partir de la barre de menu (**démarrer → exécuter → mcxsetup → OK**).

La première fenêtre de MCXSETUP permet, dans la partie supérieure, de gérer l'installation du pilote, et dans la partie inférieure, de gérer l'installation et l'initialisation des cartes :



Aspect de la fenêtre principale de MCXSETUP v2.1

- **Version** indique la version du fichier pilote MCX.SYS installé sur le disque dur.
- **Désinstaller** permet de supprimer MCX.SYS, MCXSETUP et les utilitaires du disque, et supprime du Registre les indications concernant ce pilote.

Configuration des cartes installées.

La liste dans la partie inférieure de la fenêtre présente les cartes déjà détectées ou installées et leurs principales caractéristiques.

Une icône rouge signale les cartes PCI détectées mais non configurées, montrant qu'elles ne peuvent pas être utilisées. Vous devez les configurer pour pouvoir les utiliser.

Une icône jaune signale les cartes PCI ou ISA inhibées, montrant qu'elles ne peuvent pas être utilisées du fait d'une action volontaire (vous avez coché la case « inhiber » dans les propriétés de la carte). Vous pouvez réactiver cette carte pour l'utiliser.

Une icône verte signale les cartes PCI ou ISA utilisables. Les cartes PCI dans cet état peuvent être utilisées ; les cartes ISA peuvent l'être également si les ressources sont correctement indiquées.

Vous pouvez agir sur une carte en la sélectionnant puis en utilisant soit l'un des boutons en bas de la fenêtre, soit le clic droit de la souris. Le double clic ouvre la fenêtre des propriétés. (voir plus bas).

Le bouton Ajouter... permet de définir une carte ISA. Ce bouton et le bouton Configurer... donnent accès aux pages de propriétés d'une carte. Le bouton Supprimer permet de supprimer la définition d'une carte (notez que dans les pages de propriétés il est possible d'inhiber une carte sans la supprimer). Le bouton Tester permet de contrôler sommairement le bon fonctionnement d'une carte sélectionnée, et affiche un compte-rendu après quelques secondes d'attente. Le bouton Quitter vérifie, si une carte a été modifiée, si les mêmes ressources ne sont pas utilisées par plusieurs cartes, et termine après quelques recommandations d'installation.

Le bouton Tout imprimer, accessible si aucune carte n'est sélectionnée, imprime un résumé de la configuration de chaque carte. Ce bouton devient Imprimer lorsqu'une carte est sélectionnée.

La procédure générale de configuration est donc :

- **pour une carte avec bus PCI ou cPCI:**

1) Repérer la ligne concernant la carte à configurer. Si plusieurs cartes PCI sont installées, les numéros de bus et de slot indiqués permettent de les distinguer entre elles.
2) Double-cliquer sur la carte pour ouvrir la fenêtre des propriétés.
3) Sélectionner le modèle correct parmi ceux proposés.
4) Sélectionner le mode de fonctionnement conformément à la position des interrupteurs sur la carte.
5) Sélectionner les autres paramètres en fonction de vos besoins, cliquer sur OK pour refermer la fenêtre des propriétés.
6) Si le mode de fonctionnement est « Logiciel de base » ou « Multiprotocole », resélectionner la carte dans la liste et cliquer sur Tester.

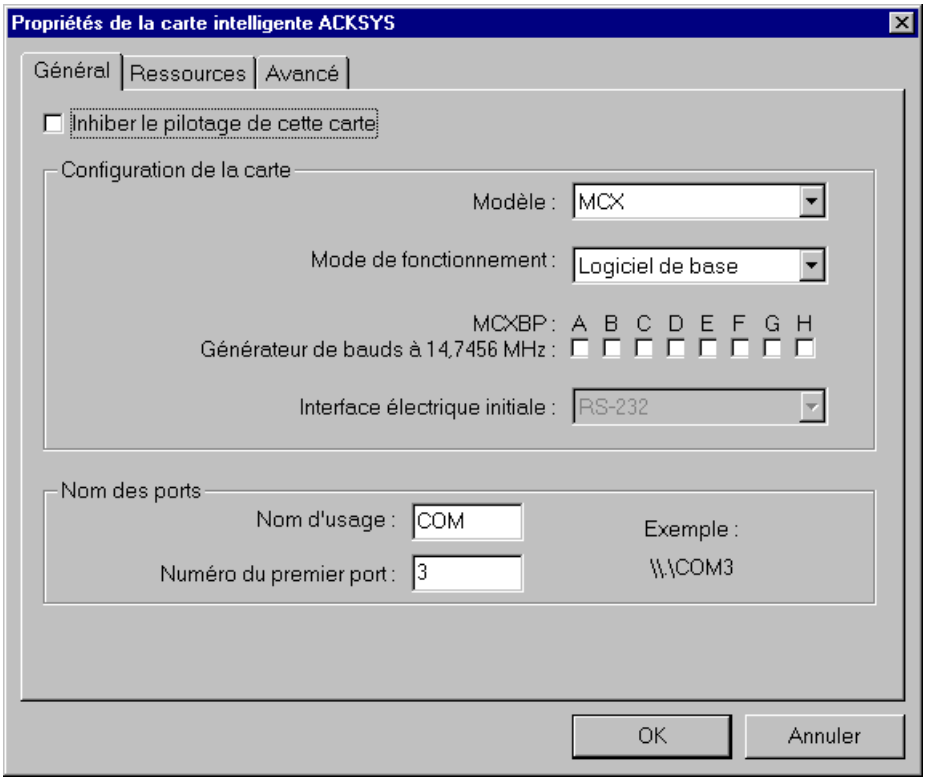
- **pour une carte pour bus ISA :**

1) Cliquer sur le bouton Ajouter...
2) Sélectionner le modèle correct parmi ceux proposés.
3) Sélectionner le mode de fonctionnement conformément à la position des cavaliers sur la carte.
4) Dans l'onglet « ressources », indiquer les ressources choisies conformément à la position des cavaliers sur la carte.
5) Sélectionner les autres paramètres en fonction de vos besoins, cliquer sur OK pour refermer la fenêtre des propriétés.
6) Si le mode de fonctionnement est « Logiciel de base » ou « Multiprotocole », resélectionner la carte dans la liste et cliquer sur Tester.
7) Cliquez sur Quitter. Lisez attentivement les messages d'information qui apparaissent alors, et suivez scrupuleusement leurs indications.

Propriétés des cartes.

Ces paramètres modifient le fonctionnement du pilote de périphérique. Ils sont pris en compte à chaque redémarrage du pilote et uniquement à ce moment. Le pilote peut être relancé soit au démarrage du système, soit manuellement dans le panneau de configuration, icône «Périphériques», soit à l'invite de commandes par les commandes « net stop mcx » puis « net start mcx », soit par MCXSETUP sur confirmation de l'usager.

L'onglet **Général** permet d'indiquer les caractéristiques de la carte à installer.



Propriétés générales d'une carte

- **Inhiber le pilotage de cette carte**

  Cette case lorsqu'elle est cochée, empêche le pilote d'accéder à la carte décrite. Elle peut être utilisée pour inhiber temporairement une carte installée, ou pour supprimer temporairement une carte de l'ordinateur sans engendrer de messages d'erreur dans l'observateur d'événements.

- **Modèle**

  Sélection du modèle de carte. Seuls les modèles correspondant au bus utilisé apparaissent dans cette liste. Vous devez indiquer précisément le modèle, car d'autres options dépendent de ce choix.

- **Mode de fonctionnement**

  Cette case permet de d'indiquer au pilote le mode de fonctionnement qui a été choisi (voir la section I). Seuls les modes de fonctionnement supportés sur le modèle choisi apparaissent. Si le mode Multiprotocole est sélectionné alors que l'option n'est pas installée dans l'EPROM de la carte, à son démarrage le pilote enregistrera une erreur dans l'Observateur d'Événements.

- **Générateur de bauds à 14,7456 MHz**

  Sur les cartes avec mezzanines MCXBP(MR) ou PCB/S ou Lite/S, chaque bloc de canaux dispose de deux oscillateurs pour la génération des horloges de transmission. Un seul des deux oscillateurs peut être utilisé à un moment donné. L'oscillateur à utiliser doit être choisi à l'installation, en fonction de la précision souhaitée sur le calcul des horloges. Il sera alors utilisé par tous les canaux du même bloc. Cette précision peut devenir significative dans les modes synchrones, ou pour les hauts débits (supérieurs à 19200 bauds).
  Les cartes avec mezzanines MCXBP ou MCXBPMR ont un à huit blocs (A à H) de 8 canaux chacun. Les cartes avec mezzanine PCB/S ou Lite/S ont un seul bloc (A) de deux canaux.
  Lorsqu'une case est cochée, tous les canaux du bloc correspondant utilisent l'oscillateur à 14,7456 MHz. Sinon, ils utilisent l'oscillateur à 16 MHz.

- **Interface électrique initiale**

  Sur certaines cartes, l'interface électrique est programmable. Sur ces cartes, lorsque l'option Multiprotocole est active, il est possible d'indiquer ici quelle interface électrique le pilote doit utiliser au démarrage sur l'ensemble des canaux.

- **Nom d'usage**

  Préfixe du nom qui sera utilisé pour désigner les canaux de cette carte. La chaîne « COM » permet d'utiliser les outils standard comme HyperTerminal. L'utilisation d'un autre nom permet d'obtenir des noms de canaux avec un format fixe (voir plus loin la description de CreateFile).

- **Numéro du premier port**

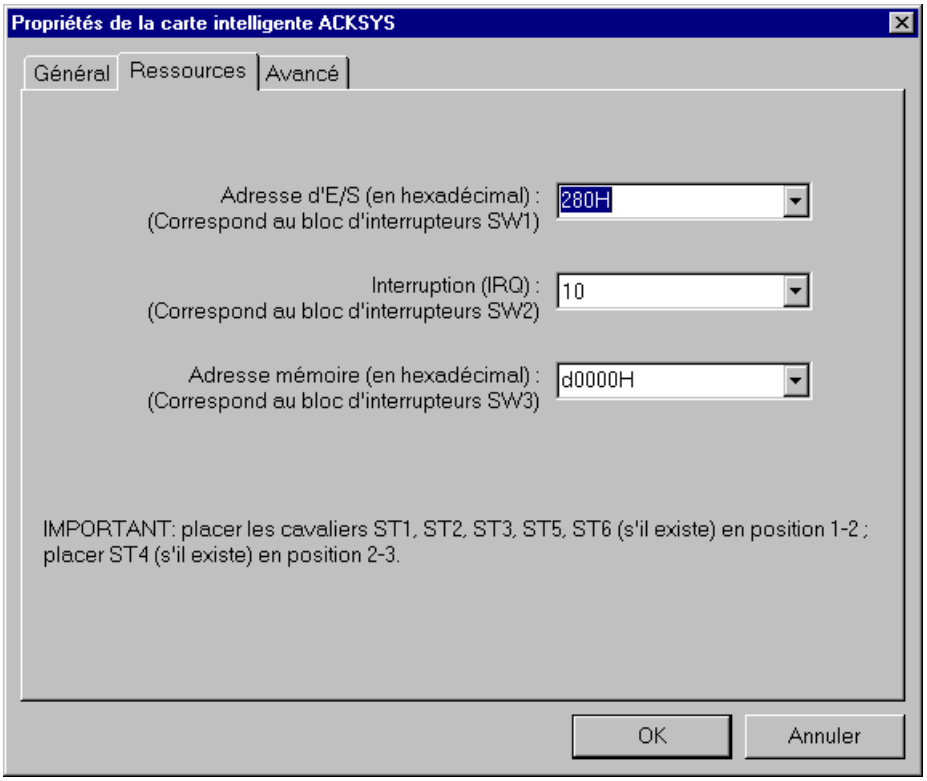
  Le nom du premier canal de la carte portera ce numéro. Les autres canaux seront numérotés séquentiellement à partir de ce numéro.

- **Exemple**

  Ici s'affiche le nom du premier canal de la carte, tel qu'il sera reconnu par le pilote. Les noms des canaux sont formés en concaténant la chaîne fixe « \\.\ », le nom d'usage et un numéro de séquence qui démarre au « numéro du premier port ».

L'onglet **Ressources** permet d'indiquer les adresses et l'interruption de la carte à installer.

Cet onglet n'est présent que pour les cartes au format du bus ISA. Les ressources des cartes PCI sont attribuées automatiquement.



Onglet ressources pour les cartes ISA

- **Adresse d'E/S**

  port d'Entrée-Sortie de la carte, nécessaire pour réserver la ressource E/S dans Windows, et pour utiliser les cartes dont la boîte aux lettres peut être désactivée.

- **Interruption**

  interruption choisie pour la carte. Cette interruption doit être affectée au bus ISA (par le BIOS PNP SETUP) et libre par ailleurs (non utilisée par un autre périphérique). Les interrupteurs SW2 doivent refléter l'interruption choisie.

- **Adresse mémoire**

  adresse choisie pour la boîte aux lettres de la carte. Une plage de 32 ko est utilisée à partir de cette adresse.

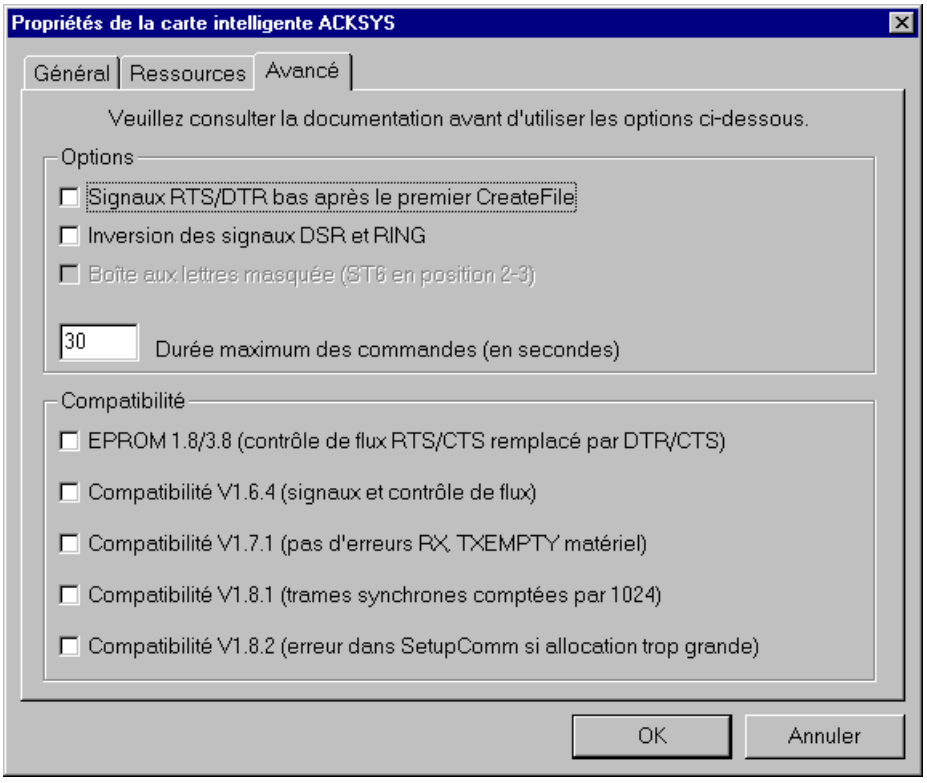
- **IMPORTANT :...**

  cette note, en bas de la page de propriétés, vous rappelle la position correcte des cavaliers de la carte pour fonctionner avec le pilote.

L'onglet **Avancé** permet de gérer les options et la compatibilité avec les versions antérieures.

Ces options agissent sur tous les canaux de la carte.

Attention : Pour des raisons de performances, les modes de compatibilité ne seront pas supportés indéfiniment ; il est donc recommandé d'adapter les logiciels en conséquence.



- **Signaux RTS/DTR bas après le premier CreateFile**

  Intérêt : au démarrage, ce pilote, par compatibilité avec le pilote de ports série Microsoft™, assigne les valeurs RTS_CONTROL_ENABLE et DTR_CONTROL_ENABLE aux champs *fRtsControl* et *fDtrControl* de la structure DCB. En conséquence, la première ouverture d'un canal après le démarrage du pilote, fait automatiquement monter les signaux RTS et DTR. Ce comportement peut être gênant dans certaines applications.

  Case cochée : au démarrage, le pilote assigne aux champs *fRtsControl* et *fDtrControl* de la structure DCB les valeurs par défaut RTS_CONTROL_DISABLE et DTR_CONTROL_DISABLE. Les signaux RTS et DTR resteront inactifs jusqu'à un changement volontaire par l'application.

- **Interversion des signaux DSR et RING**

  Intérêt : certaines cartes ne disposent pas de la broche DSR sur le connecteur RS232. Certaines applications utilisent ce signal.

  Case cochée : le pilote intervertit la signification des broches DSR et RING. Le signal détecté sur la broche RING est remonté à l'application comme étant un signal DSR (et réciproquement pour les cartes qui disposent de la broche DSR). Les actions de l'application censées agir sur DSR agissent sur le signal RING à la place (par exemple la modification de *fDsrSensitivity*).

  Remarque : il est possible de choisir cette interversion séparément pour chaque canal (voir la section « Manuel de référence »).

- **Durée max. des commandes**

  Pour détecter les pannes éventuelles, le pilote chronomètre la durée de certains groupes de commandes envoyées à la carte. Une erreur est signalée à l'application, et un message est placé dans l'Observateur d'événements, si cette durée excède celle indiquée ici. Ce cas d'erreur est rare et la valeur par défaut (30 secondes) ne devrait normalement pas être changée. Les valeurs permises vont de 5 à 3600 secondes.

- **EPROM 1.8/3.8**

  Versions 1.6.2 et antérieures :

  lorsque l'application demande le contrôle de flux par RTS, le contrôle de flux par DTR est utilisé à la place, pour supporter les anciennes révisions d'EPROM (antérieures à 1.8 (MCX) ou 3.8 (MCC)), qui ne supportent pas le contrôle de flux par RTS.

  Versions postérieures à 1.6.2 :

  le pilote permet le contrôle de flux par RTS qui est supporté par les cartes équipées d'EPROM récentes.

  Intérêt : Certaines applications pourraient tenir compte de la substitution des signaux, et espérer un contrôle de flux DTR alors qu'elles programment le RTS.

  Case cochée : la substitution du RTS par le DTR est rétablie, sous réserve que la case « Compatibilité 1.6.4 » soit grisée (sans l'option Multiprotocole) ou cochée (avec l'option Multiprotocole).

- **Compatibilité V1.6.4 (signaux et contrôle de flux)**

  Versions 1.6.4 et antérieures :

  les contrôles de flux en entrée et en sortie sont liés (voir l'annexe « Contrôle de flux »). Les signaux RTS et DTR sont activés automatiquement lorsqu'un canal est ouvert par CreateFile().

  Versions postérieures à 1.6.4 :

  les contrôles de flux définis séparément en entrée et en sortie, disponibles dans l'option Multiprotocole, sont supportés. Lorsqu'un canal est ouvert, les signaux RTS et DTR prennent l'état indiqué par les champs *fRtsControl* et *fDtrControl* de la structure DCB.

  Intérêt : conformité à la documentation Win32 Communication API.

  Case cochée : rétablit le comportement antérieur.

- **Compatibilité V1.7.1 (pas d'erreurs RX, TXEMPTY matériel)**

  Versions 1.7.2, 1.7.1 et antérieures :
  le pilote ignorait les trames synchrones reçues avec des erreurs. L'événement EV_TXEMPTY de WaitCommEvent() était remonté à l'application lors de chaque vidage du tampon d'émission de la carte, même si un WriteFile était en attente dans le pilote pour le même canal.

  Versions postérieures à 1.7.2 :
  toutes les trames reçues sont fournies à l'application, éventuellement avec une indication d'erreur. L'événement EV_TXEMPTY n'est remonté que si aucune émission n'est en attente pour le canal.

  Intérêt : compatibilité avec le pilote série Microsoft™.

  Case cochée : rétablit le comportement antérieur (sans erreurs de réception synchrone, et événement EV_TXEMPTY plus fréquent que nécessaire).

- **Compatibilité V1.8.1 (trames synchrones comptées par 1024)**

  Versions 1.8.1 et antérieures :
  sur les canaux utilisés dans un mode synchrone, les champs *cbInQue* et *cbOutQue* de la structure COMSTAT contenaient un nombre de trames multiplié par 1024.

  Versions postérieures à 1.8.1 :
  ces champs contiennent un nombre de trames.

  Intérêt : l'écriture d'application est simplifiée. D'autre part l'ancienne limite de 1024 octets par trame synchrone, qui justifiait cette bizarrerie, n'existe plus.

  Case cochée : rétablit le comportement antérieur.

- **Compatibilité V1.8.2 (erreur dans SetupComm si...)**

  Versions antérieures à 1.8.3 :
  Utiliser SetupComm() avec des tampons trop grands renvoyait une erreur à l'application.

  Versions 1.8.3 et postérieures :
  Aucune erreur n'est signalée dans ce cas.

  Intérêt : La documentation Win32 précise que les paramètres de SetupComm sont indicatifs et ne renvoie pas d'erreur si ceux-ci ne peuvent pas être respectés.

  Case cochée : rétablit le comportement antérieur (avec remontée d'erreurs).

IV CONTROLE DE L'INSTALLATION

Vous pouvez vérifier le lancement du pilote dans l'Observateur d'Événements, où deux messages d'information liés au pilote "Mcx" figurent :

a) un message indiquant le chargement et la version du pilote,

b) pour chaque carte installée, un message indiquant la version d'EPROM, le nombre de voies reconnues par la carte et quelques autres informations utiles.

Sous Windows 2000, 2003, XP et Vista

Vous pouvez contrôler le lancement du pilote par le Gestionnaire de Périphériques, accessible entre autres par un clic droit sur le poste de travail.

Les options de l'onglet « Pilote » permet d'arrêter, de redémarrer ou d'inhiber le pilote.

Sous Windows NT 4.0

Vous pouvez contrôler le lancement du pilote par le Panneau de Configuration, icône « Périphériques ».

V OUTILS DE DEVELOPPEMENT ET EXEMPLES

Dans vos programmes sources en langage C, vous pouvez communiquer avec le pilote, d'une part grâce aux fonctions standard de Win32 (par exemple CreateFile, etc.) d'autre part avec des commandes et structures propres au pilote fourni.

Voici les fichiers utiles au développement d'applications spécifiques :

Mode de fonctionnement	Fichiers
Logiciel de base Multiprotocole	SDK\MULTIPROTOCOL\INCLUDE\MCC_MCX.H, SDK\MULTIPROTOCOL\INCLUDE\MCXPROTO.H
Mcxdos/Automcx Extension BIOS	SDK\AUTOMCX\INCLUDE\AUTONT.H, SDK\AUTOMCX\LIB\AUTOMCX.LIB

Si nécessaire, recopiez ces fichiers sur le disque dur, pour qu'ils soient accessibles dans vos programmes sources en langage C.

Des exemples de programmation pour les modes « Logiciel de base » et « Multiprotocole » sont fournis dans le sous-répertoire SDK\MULTIPROTOCOL du support de distribution. En particulier le répertoire SDK\MULTIPROTOCOL\LIB contient une bibliothèque de fonctions utiles pour se familiariser avec l'utilisation du pilote ; ces fonctions utilisent le « header » SDK\MULTIPROTOCOL\INCLUDE\ACK_W32.H et sont compilées avec l'option MultiThread dans la bibliothèque SDK\MULTIPROTOCOL\LIB\MCC_MCX.LIB.

MODES DE COMPATIBILITE COM

Les sections suivantes décrivent les modes de fonctionnement « Logiciel de base » et « Multiprotocole » du pilote.

I INTERFACES DE PROGRAMMATION (API)

Chaque canal série est programmable indépendamment, que cela concerne l'interface électrique, le format des signaux, le format de trame, le protocole ou le contrôle de flux, etc. Chaque canal peut être piloté par une application différente si nécessaire.

Le nom de fichier utilisé pour accéder aux canaux est défini lors de la configuration de la carte. Le « nom d'usage » commence par « COM », suivi d'un numéro défini dans l'onglet de configuration générale de la carte.

REMARQUE : Le pilote pour Windows NT 4.0 supporte aussi des « noms d'usage » différents de « COM ». Dans ce cas les voies sont numérotées 01, 02... jusqu'au nombre de voies installées sur la carte.

Exemples

nom d'usage	numéro du premier port	Dénomination des canaux
COM	3	`\\.\COM3`, `\\.\COM4`, etc.
COM	9	`\\.\COM9`, `\\.\COM10`, etc.
Mcx1 (sous NT4 seulement)	ignoré	`\\.\Mcx101`, `\\.\Mcx102`, etc.
McxB (sous NT4 seulement)	ignoré	`\\.\McxB01`, etc.

Les API utilisables sont basés sur la documentation de Win32. Elles se répartissent en trois groupes : les **services fichiers** (CreateFile, ReadFile, etc.), les **services de communication** (SetCommState, etc.) et le service fichier **DeviceIoControl**, par lequel passent toutes les fonctions spécifiques à ce pilote, et non prévues dans Win32 (changement de protocole, d'interface électrique, etc.).

Dans ce qui suit, ces API sont regroupés suivant leur utilisation :

- ☞ communications asynchrones,
- ☞ communications synchrones,
- ☞ communications LAPB,
- ☞ services spécifiques au pilote,
- ☞ utilitaires.

I.1 Programmation des communications asynchrones.

D'une façon générale, reportez-vous à la documentation Win32 de Microsoft. Les informations ci-dessous indiquent simplement

- ☞ les cas où le pilote s'écarte des spécifications des communications série de Win32,
- ☞ les cas où la documentation Win32 est imprécise,
- ☞ les cas où la spécification n'est pas intuitive.

Services fichiers

Service	**Commentaires**
CreateFile	après **CreateFile**, les tampons d'émission et de réception sont vides.
WriteFile	le nombre de caractères à émettre doit être inférieur ou égal à 8192 octets. **WriteFile** se termine dès que les caractères à émettre sont dans le tampon de la carte, et non pas lorsqu'ils sont effectivement émis.
CloseHandle	**CloseHandle** se termine en supprimant le tampon d'émission et en ouvrant les circuits 105 et 108 (RTS et DTR). Il est recommandé d'utiliser **FlushFileBuffers** avant **CloseHandle** pour s'assurer que toutes les données du tampon ont été émises.

Services de communication série.

Service	**Commentaires**
BuildCommDCB	*sans restriction.*
BuildCommDCBAndTimeouts	Sous les réserves faites pour SetCommTimeout.
ClearCommBreak	Aucun effet, le BREAK se termine de lui-même après une seconde.
ClearCommError	Seuls les bits CE_BREAK, CE_FRAME, CE_OVERRUN, CE_RXPARITY de *lpdwErrors* sont supportés. Dans la structure COMSTAT, seuls les champs *cbInQue* et *cbOutQue* sont supportés.
EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK sont supportés. CLRBREAK : voir ClearCommBreak. SETXON, SETXOFF ne sont pas supportés.
GetCommMask	*sans restriction.*
GetCommModemStatus	voir « Annexe : limitations et différences avec les ports COM », page 61.

GetCommProperties	*dwMaxBaud :* BAUD_USER, car toute vitesse qui peut être supportée à moins de 1% près est acceptée. *dwProvSubType* : RS232, bien que ces cartes supportent également les types RS422, RS485, boucle de courant si elles sont munies des options adéquates. *dwMaxTxQueue, dwMaxRxQueue* : nombre total d'octets effectivement alloués aux tampons du canal. Les autres champs sont supportés de façon standard.
GetCommState	Renvoie les valeurs fournies à SetCommState (sauf *fParity*).
SetCommState	La valeur RTS_CONTROL_TOGGLE de *fRtsControl*, les champs *fBinary* et *EofChar*, ne sont pas supportés. Avec l'option Multiprotocole, les autres champs sont supportés normalement. Avec le logiciel de base, le pilote ignore aussi les champs ci-dessous et utilise la valeur fixe indiquée :

fDsrSensitivity	FALSE
fTXContinuesOnXoff	TRUE
fErrorChar	FALSE
fNull	FALSE
XonLim, XoffLim	128, 20
fInX, fOutX, fOutxDsrFlow, fOutxCtsFlow, fDtrControl, fRtsControl	support restreint : voir «Utilisation de contrôles de flux spéciaux.», page 59.

GetCommTimeouts	*sans restriction.*
PurgeComm	*sans restriction.*
SetCommBreak	La durée du BREAK est prédéfinie (fixée à une seconde).
SetCommMask	Pour EV_RING et EV_DSR, voir « Annexe : limitations et différences avec les ports COM », page 61. EV_RX80FULL ne donne pas les résultats attendus et EV_RXCHAR peut avoir un retard de 100 ms sur l'événement.
SetCommTimeouts	*ReadIntervalTimeout* est supporté avec une erreur de 25% + 100 ms.
SetupComm	Les tailles de tampons différentes des tailles standard sont ignorées. Une erreur peut être indiquée dans certains cas (voir la section sur la configuration des propriétés avancées, page 13). Les tailles standard sont de **8192** caractères par canal pour les **tampons d'émission**, et **512** caractères par canal pour les **tampons de réception**.
TransmitCommChar	Non supporté.
WaitCommEvent	*sans restriction.*

I.2 Programmation des communications en asynchrone synchronisé.

D'une façon générale, reportez-vous à la documentation des communications Win32 de Microsoft™. La différence essentielle avec un port asynchrone, vient du fait qu'il faut préciser le traitement des horloges avec la fonction suivante :

DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)

Permet de fixer le protocole (synchrone, asynchrone, asynchrone synchronisé) et ses options. Une description complète figure page 45.

I.3 Programmation des communications synchrones HDLC/SDLC/BISYNC.

D'une façon générale, reportez-vous à la documentation des communications Win32 de Microsoft™. Les informations ci-dessous indiquent simplement

- ☞ les cas où l'API de communication série de Win32 n'est pas appropriée,
- ☞ les cas où le pilote s'écarte des spécifications Win32,
- ☞ les cas où la documentation Win32 est imprécise.

Services fichiers

Service	**Commentaires**
CreateFile	après **CreateFile**, les tampons d'émission et de réception sont vides.
ReadFile	**ReadFile** renvoie toujours au plus une trame. Si la trame est plus longue que la taille demandée, seul le début est fourni, la fin est perdue. Si la trame est plus courte, elle est entièrement renvoyée et le tampon demandé n'est pas complètement rempli (même si une autre trame est déjà dans le tampon de la carte : elle sera fournie lors du prochain **ReadFile**). Si la trame comporte une erreur (ABORT, CRC, etc.) **ReadFile** renvoie une erreur (voir aussi les options de compatibilité dans la section décrivant l'installation).
WriteFile	chaque **WriteFile** crée une trame dont la taille est limitée à la valeur spécifiée par la commande IOCTL_SERIAL_SET_SYNC_STATE. **WriteFile** se termine dès que les caractères à émettre sont dans le tampon de la carte, et non pas lorsqu'ils sont effectivement émis.
CloseHandle	**CloseHandle** se termine en supprimant le tampon d'émission et en ouvrant les circuits 105 et 108 (RTS et DTR). Il est recommandé d'utiliser **FlushFileBuffers** avant **CloseHandle** pour s'assurer que toutes les données du tampon ont été émises.
FlushFileBuffers	**FlushFileBuffers** se termine avec l'émission du fanion de la dernière trame émise (par le **WriteFile** immédiatement antérieur), ce qui garantit que les tampons de la carte sont vides.

Services de communication série.

Service	**Commentaires**
BuildCommDCB	*sans restriction.*
BuildCommDCBAndTimeouts	Sous les réserves faites pour SetCommTimeout.
ClearCommBreak	Aucun effet, le BREAK se termine de lui-même après une seconde.
ClearCommError	Seuls les bits CE_BREAK, CE_FRAME, CE_OVERRUN, CE_RXPARITY de *lpdwErrors* sont supportés. Dans la structure COMSTAT, seuls les champs *cbInQue* et *cbOutQue* sont supportés ; ils indiquent un nombre de trames et non un nombre de caractères.

EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK sont supportés. CLRBREAK : voir ClearCommBreak(). SETXON, SETXOFF ne sont pas supportés.
GetCommMask	*sans restriction.*
GetCommModemStatus	voir « Annexe : limitations et différences avec les ports COM », page 61.
GetCommProperties	*dwMaxBaud :* BAUD_USER, car toute vitesse qui peut être supportée à moins de 1% près est acceptée. *DwProvSubType* : RS232, bien que ces cartes supportent également les types RS422, RS485, boucle de courant si elles sont munies des options adéquates. *dwMaxTxQueue, dwMaxRxQueue* : nombre total de trames effectivement allouées aux tampons du canal. Les autres champs sont supportés de façon standard.
GetCommState	Renvoie les valeurs fournies à SetCommState (sauf *fParity*).
SetCommState	Les seuls champs supportés sont *BaudRate*, *fOutxCtsFlow*, *fOutxDsrFlow*, *fDsrSensitivity*, *fDtrControl*, *fRtsControl*, *ByteSize* et *Parity* (ces deux derniers dans certains protocoles seulement).
GetCommTimeouts	*sans restriction.*
PurgeComm	*sans restriction.*
SetCommBreak	La durée du BREAK est prédéfinie (fixée à une seconde).
SetCommMask	L'événement EV_RXFLAG n'est pas supporté. Pour EV_RING et EV_DSR, voir « Annexe : limitations et différences avec les ports COM », page 61.
SetCommTimeouts	*ReadIntervalTimeout* est ignoré, sauf pour la valeur MAXDWORD. Sinon il doit être initialisé à zéro pour éviter toute incompatibilité future.
SetupComm	Aucun effet.
TransmitCommChar	Non supporté.
WaitCommEvent	*sans restriction.* EV_RXCHAR signale l'arrivée d'une trame entière.

Services spécifiques au pilote.

Service	**Commentaires**
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Permet de fixer le protocole (synchrone, asynchrone, asynchrone synchronisé) et ses options. Une description complète figure page 45.

I.4 Programmation du protocole LAPB (ou HDLC-ABM).

La programmation de ces protocoles s'appuie sur le paragraphe précédent qui décrit les liaisons synchrones d'une façon générale. Reportez-vous y pour toute information qui ne se trouve pas dans le présent paragraphe.

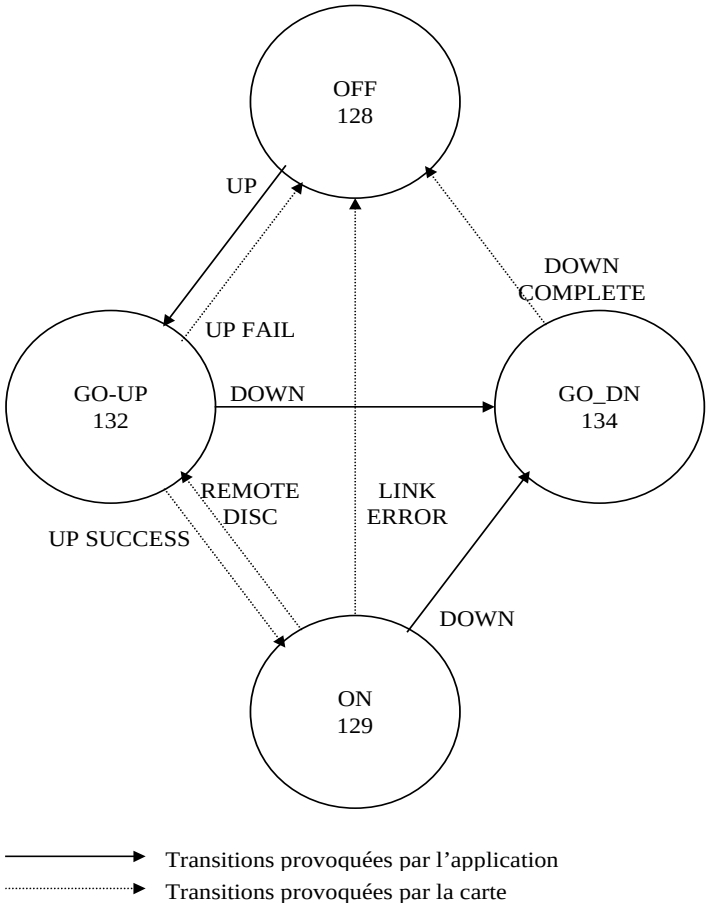
Ci-dessous figurent les informations nécessaires pour programmer une liaison LAPB. Notez que l'utilitaire **MCXMODE** possède quelques possibilités réduites de gestion de la couche liaison, utilisables en particulier en phase de test. Des exemples de programmation sont fournis dans le répertoire **sdk\multiprotocol\LIB\src**.

Primitives liées à l'utilisation de LAPB.

Service	**Commentaires**
CreateFile	Conserve le protocole courant, l'interface électrique et la vitesse de transmission. Réactive le mode de gestion des signaux. Vide le tampon de réception. **CreateFile** ne tente pas d'établir la liaison.
CloseHandle	Vide le tampon d'émission. En mode LAPB le récepteur n'est pas inhibé, ce qui permet de recevoir des trames de protocole et éventuellement d'y répondre. **CloseHandle** ne tente pas de rompre la liaison.
ReadFile	Lit une trame dans le tampon de la carte ou attend l'arrivée d'une trame d'information. Renvoie l'erreur ERROR_HANDLE_EOF si la liaison est déconnectée ou se rompt durant l'attente. **ReadFile** est soumis aux délais instaurés par **SetCommTimeouts**.
WriteFile	Place une trame dans le tampon d'émission, avec attente si le tampon est plein. Renvoie ERROR_NOT_READY si la liaison est déconnectée ou se déconnecte en cours d'attente. **WriteFile** est soumis aux délais instaurés par **SetCommTimeouts**.
SetCommMask/GetCommMask/WaitCommEvent	L'événement EV_MCXLAPB (équivalent à l'événement Win32 EV_EVENT1) peut être traité. Il est déclenché à chaque réception de l'interruption d'état LAPB (voir liste ci-dessous).
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Permet de fixer le protocole et ses options. Voir la section « Manuel de référence ».
DeviceIoControl(...,IOCTL_SERIAL_CMD_AUTO,...)	Permet d'envoyer à la carte une commande non directement supportée par l'interface Win32. En particulier la commande **PRCTL** permet de contrôler les états du protocole avec les options ABMLINKUP, ABMLINKDN, ABMSTATE. Voir les fichiers d'exemples et la référence détaillée dans la section « Fonctions CMD et CMD_AUTO ».

États du pilote

Un canal LAPB peut être dans quatre états décrits dans le schéma suivant :



Nouvel état	Déclenchement
MCX_LINK_OFF (down, déconnecté)	• DeviceIoControl(SET_SYNC_STATE) passant en protocole LAPB • DeviceIoControl(CMD_AUTO) avec commande PRCTL ABMLINKDN (si la déconnexion survient avant la fin de la commande) • Interruption État, échec d'une tentative de connexion par PRCTL ABMLINKUP • Interruption État, déconnexion suite à une erreur de transmission irrécupérable • Interruption État, réussite d'une demande de déconnexion par PRCTL ABMLINKDN
MCX_LINK_GO_UP (connexion en cours)	• DeviceIoControl(CMD_AUTO) avec commande PRCTL ABMLINKUP • Interruption État, dérangement temporaire suite à une erreur récupérable
MCX_LINK_ON (up, connecté)	• DeviceIoControl(CMD_AUTO) avec commande PRCTL ABMLINKUP (si la connexion survient avant la fin de la commande) • Interruption État, réception d'une demande ou d'un acquittement de connexion
MCX_LINK_GO_DN (rupture en cours)	• DeviceIoControl(CMD_AUTO) avec commande PRCTL ABMLINKDN

Effet des changements d'état

Aucune action n'est entreprise quand le changement d'état est provoqué par l'application. Le tableau suivant résume les actions entreprises lors des interruptions de changement d'état.

Nouvel état	Action
MCX_LINK_OFF	EV_MCXLAPB signalé si précisé par un SetCommMask antérieur Tout ReadFile renverra 0 caractères et l'erreur ERROR_HANDLE_EOF Tout WriteFile renverra l'erreur ERROR_NOT_READY.
MCX_LINK_ON	EV_MCXLAPB signalé si précisé par un SetCommMask antérieur

Obtention de l'état courant

Il y a trois manières d'obtenir une information sur l'état de la liaison.

- La première, imprécise, consiste à supposer que la commande PRCTL/ABMLINKUP passe toujours dans l'état MCX_LINK_ON, que PRCTL/ABMLINKDN passe toujours dans l'état MCX_LINK_OFF, et à détecter d'éventuelles déconnexions par les erreurs renvoyées par ReadFile et WriteFile. Cette méthode permet d'utiliser un canal LAPB comme un « filtre » avec les commandes standard du système, en établissant et en rompant la connexion avec l'utilitaire MCXMODE.
- La seconde méthode consiste à interroger le canal par la commande PRCTL/ABMSTATE.
- La troisième méthode consiste à utiliser WaitCommEvent pour détecter instantanément les changements d'état qui sont le fait de la carte ou de l'autre extrémité de la liaison. Comme une série rapide d'événements peut se manifester par un seul réveil de WaitCommEvent, il est conseillé d'interroger systématiquement l'état du canal après un réveil, et de tenir compte que des étapes dans l'enchaînement des états peuvent être sautées. Il est à noter que les événements sont mémorisés dès qu'ils sont sélectionnés par SetCommMask, et qu'ensuite, un ou plusieurs événements arrivant entre deux WaitCommEvent consécutifs ne peuvent pas être perdus (mais peuvent être combinés en un seul).

Un traitement complet devrait tenir compte de ces trois sources d'informations.

I.5 Programmation des services spécifiques.

La fonction Win32 **DeviceIoControl()** a été étendue pour dialoguer directement avec le logiciel implanté sur la carte, pour envoyer "manuellement", c'est-à-dire sans contrôle du pilote, les commandes décrites dans les manuels "Manuel d'utilisation du logiciel de base des cartes MCX" et "Manuel d'utilisation du logiciel multiprotocole des Cartes MCX". Une description complète et des exemples figurent page 49, et les sources des programmes dans « sdk\multiprotocol\examples » illustrent son emploi.

I.6 Utilitaires standard de Windows NT.

Pour exploiter simplement les outils standard de Windows NT avec les cartes MCX, il est conseillé d'utiliser le nom «COM» lors de l'installation.

Panneau de configuration

Ne pas créer les voies COM des cartes MCX par l'icône "ports" car le pilote crée ces noms automatiquement. Par contre il est possible d'utiliser ce dialogue pour changer les paramètres de la transmission (utile si une voie est utilisée avec le gestionnaire d'impression). Par ailleurs ne pas chercher à définir les adresses ni les interruptions avec ce dialogue.

HyperTerminal.exe

Cet accessoire de Windows NT 4 fonctionne normalement avec ce pilote.

Terminal.exe

Cet accessoire de Windows NT 3.x ne supporte que les ports COM1 à COM9.

Mode.exe

Cet utilitaire DOS ne supporte que les noms commençant par COM. Il ne permet de modifier que les modes de transmission asynchrones. L'utilitaire **mcxmode.exe** fourni pallie ces inconvénients (voir le paragraphe décrivant **mcxmode.exe**).

Remote Access Services (service d'accès distant)

L'interface de numérotation téléphonique de **Windows NT 4.0 ne fonctionne pas** avec ce pilote.

Imprimantes série

Un défaut dans le Gestionnaire d'Impressions de Windows NT 3.51 fait que l'ordre suivant doit être respecté pour créer une imprimante :

- Utiliser le «Panneau de configuration» icône «Ports» pour modifier, ou au moins afficher, les paramètres du port concerné. Valider impérativement par <OK>.
- «Panneau de configuration» icône «Services» : arrêter le service nommé «Spooler» et le redémarrer immédiatement. Ceci permet de prendre en compte le nouveau port et ses paramètres.
- «Panneau de configuration» icône «Imprimantes» : dans le Gestionnaire d'Impressions, utiliser le menu <Imprimante> <Créer une imprimante...>. Remplir le formulaire ; le port doit apparaître dans la liste <Imprimer vers :>. Terminer l'installation.

Command.exe (invite de commande, fenêtre DOS)

Il est possible d'utiliser les noms COM1 à COM9 directement. Au delà, utiliser les noms standard `\\.\`*nom-attribué-à-l'installation*, par exemple, pour une redirection de la sortie console :

```
C:> DIR > COM9          } Ces deux écritures
C:> DIR > \\.\COM9      } sont équivalentes
C:> DIR > \\.\COM10     Seule écriture permise si le nom installé est COM
C:> DIR > \\.\MCX101    Seule écriture permise si le nom installé est MCX1
```

I.7 Utilitaires ACKSYS supplémentaires.

MCXSTARTER, installé dans le menu Démarrage, vous donne une vue d'ensemble des possibilités de la carte, vous permet de tester la configuration appropriée pour votre système et de générer du code d'initialisation..

L'utilitaire **MCXMODE** permet de paramétrer les caractéristiques de transmission, qu'elle soit synchrone ou asynchrone. La commande MCXMODE sans paramètres, affiche une aide en ligne.

L'utilitaire **MCCIOCTL** permet d'envoyer directement à la carte la commande passée en argument. La commande MCCIOCTL sans paramètres affiche une aide en ligne.

L'utilitaire **DOSDEV** gère les liens entre les noms d'objets Windows NT et les périphériques DOS. Son effet prend fin au prochain arrêt du système. Sa syntaxe est la suivante :

`dosdev`	liste les alias.
`dosdev -l COMn \Device\McxCNN`	crée l'alias COMn pour le périphérique McxCNN.
`dosdev -r COMn`	supprime l'alias COMn.

(*n* est le numéro du port COM, *C* est le numéro de carte MCX, *NN* est le numéro de voie sur la carte)

L'utilitaire **SETMCX** permet d'afficher ou de modifier les paramètres d'une carte à partir de la ligne de commande ou d'un « batch »

`SETMCX`	aide en ligne
`SETMCX` *`n`*	affiche tous les paramètres de la carte *n*
`SETMCX` *`n param`*	affiche la valeur du paramètre *param* de la carte *n*
`SETMCX` *`n param val`*	attribue la valeur *val* au paramètre *param* de la carte *n*
`SETMCX` *`n`* `Compatibility +`*`code –code…`*	ajoute ou retire le bit *code* aux options de configuration

II MANUEL DE REFERENCE DETAILLE.

Les fonctions spécifiques au pilote sont accessibles par **DeviceIoControl**(). Elles utilisent des définitions et des structures décrites dans « **mcc_mcx.h** », ou dans « **mcxproto.h** » pour les fonctions spécifiques au mode Multiprotocole.

Ces fichiers sont fournis sur le medium d'installation dans le répertoire « SDK\MULTIPROTOCOL\INCLUDE ».

II.1 Extrait du fichier mcc_mcx.h

#include "mcc_mcx.h"

```
/* serial IOCTL codes for Windows NT */
#if defined(CTL_CODE) && defined(FILE_DEVICE_SERIAL_PORT)

#define MCX_IOCTL(code) \
             CTL_CODE(FILE_DEVICE_SERIAL_PORT,code,\
                  METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_SERIAL_GET_SYNC_STATE   MCX_IOCTL(0x901)
#define IOCTL_SERIAL_SET_SYNC_STATE   MCX_IOCTL(0x902)
#define IOCTL_SERIAL_CMD              MCX_IOCTL(0x903)
#define IOCTL_SERIAL_CMD_AUTO         MCX_IOCTL(0x904)
#define IOCTL_SERIAL_ACCESS_AREA      MCX_IOCTL(0x90A)
#define IOCTL_SERIAL_MCX_OPTIONS      MCX_IOCTL(0x90B)

#endif /* Windows NT */

------------------------------------------------------------

/* macros and structs for CMD & CMD_AUTO */
typedef struct mcc_cmd {
       unsigned char opcode;
       unsigned char status;
       unsigned char par[76];
       unsigned char ichan;
       unsigned char icond;
       unsigned char ipar1;
       unsigned char ipar2;
       unsigned char ipar3;
       unsigned char padding1;
       unsigned char *data;
       unsigned char *kdata;
       unsigned short length;
       unsigned short padding2;
}mcc_cmd;

typedef struct _MCC_CMD {    /* tampon pour ioctls CMD... */
       mcc_cmd Cb;
       unchar Data[1];
} MCC_CMD, *PMCC_CMD;

/* taille a allouer pour contenir la struct _MCC_CMD */
#define MCX_DIRECT_IO_BUFFER_SIZE(datalen)
       ((datalen)+sizeof(mcc_cmd))
```

```
typedef struct _MCX_SYNCHRONIZATION_PARAMETERS {
 unchar SynchronousMode;          /* protocole */
#define MCX_SYNC_CHAR       0    /* NOT synchronous */
#define MCX_SYNC_BISYNC     2
#define MCX_SYNC_HDLC       4
#define MCX_SYNC_LAPB       5    /* LAPB, HDLC/ABM */
 unchar Duplex;                   /* flag version+2 bits duplex */
#define MCX_WAY_VERSION     0x80 /* bit validité champ Version */
#define MCX_WAY_FULLDUPLEX  0x80 /* full duplex + version */
#define MCX_WAY_HALFDUPLEX  0x81 /* RTS bas a chaque trame */
                                 /*--------- horloges --------- */
                                 /*   ETTD      ETCD NULL-MODEM */
 unchar TransmitClockSource;     /*   TXCI      BRG     BRG     */
 unchar ReceiveClockSource;      /*   RXC       BRG     RXC     */
 unchar TxClockPinSource;        /*  TRXC_HIGH  BRG     BRG     */
#define MCX_CLOCK_RXC       0    /* modes pour TransmitClockSource... */
#define MCX_CLOCK_TXCI      1    /* ... et ReceiveClockSource */
#define MCX_CLOCK_TRXC_HIGH 0    /* TxClockPinSource = toujours haut */
#define MCX_CLOCK_TXCLOCK   1    /*     = copie de TransmitClockSource */
#define MCX_CLOCK_BRG       2    /* modes communs */
#define MCX_CLOCK_DPLL      3
 unchar MonosyncChar;
 unchar BisyncChar;
 unchar Version;                  /* valide si Duplex = MCX_WAY...*/
 McxUnshort Options;              /* défaut: 0 */
#define MCX_HDLC_USERDTR    1    /* ralentit HDLC mais permet pilotage DTR */
#define MCX_HDLC_SPECS      2    /* Utiliser la struct Protocol.Hdlc ci-dessous */
#define MCX_BISYNC_SPECS    2    /* Utiliser la struct Protocol.Bisync ci-dessous */
 McxUnshort DataLength;           /* taille max. trame (LAPB N1)  */
#define MCX_FRAMELEN_DEFAULT 0   /* défaut pour ce champ         */
union{
    struct{
         unchar RxFrames;        /* défaut: 14 */
         unchar TxFrames;        /* défaut: 4 */
         McxUnshort Spare1;      /* réservé, mettre à zéro */
         McxUnshort Spare2;      /* réservé, mettre à zéro */
         McxUnshort Spare3;      /* réservé, mettre à zéro */
         McxUnshort Spare4;      /* réservé, mettre à zéro */
         McxUnshort Spare5;      /* réservé, mettre à zéro */
    }Hdlc, Bisync;
#define MCX_HDLC_DEFAULT    0    /* défaut pour ces champs */
    struct{
         unchar Role;            /* défaut: CLIENT */
#define MCX_ROLE_CLIENT     1
#define MCX_ROLE_NETWORK    3
         unchar K;               /* défaut: 7              */
         McxUnshort N2;          /* défaut: 10 essais      */
         McxUnshort T1;          /* défaut: 2550 ms        */
         McxUnshort T2;          /* défaut: 0              */
         McxUnshort T3;          /* défaut: infini         */
         McxUnshort Spare;       /* réservé, mettre à zéro  */
#define MCX_LAPB_DEFAULT    0    /* défaut pour ces champs  */
    }Lapb;
 }Protocol;
} MCX_SYNCHRONIZATION_PARAMETERS, *PMCX_SYNCHRONIZATION_PARAMETERS;
```

```
/* structure pour IOCTL_SERIAL_ACCESS_AREA */
typedef struct _MCX_AREA_DESCRIPTOR {
    long Operation;             /* combinaison des flags suivants */
#define MCX_AREA_GET       0    /* carte vers application */
#define MCX_AREA_SET       1    /* application vers carte */
#define MCX_AREA_MEMORY    0    /* accès a la B.A.L. */
    long StartAddress;          /* adresse départ de l'accès */
                                /* relative au port de base ou au */
                                /* début de la boite aux lettres */
    long Length;                /* longueur a transférer */
    unchar Buffer[1];           /* valeurs à écrire si MCX_AREA_SET */
} MCX_AREA_DESCRIPTOR, *PMCX_AREA_DESCRIPTOR;
#define MCX_AREA_DESCRIPTOR_SIZE(dlen) \
                           ((dlen)+sizeof(MCX_AREA_DESCRIPTOR)-1)

-------------------------------------------------------------------

/* structure pour IOCTL_SERIAL_SET_OPTIONS */
typedef struct _MCX_OPTION {
    long Option;                                    /* code option */
#define MCX_OPTION_GET_CHANNEL          0x20002 /* obtenir n° canal */
#define MCX_OPTION_SET_DSR_RI_INVERSION 0x30100 /* exchange DSR/RING */
#define MCX_OPTION_GET_DSR_RI_INVERSION 0x40001 /* get DSR/RING state */
#define MCX_OPTION_GET_COMPATIBILITY    0x50004 /* get current Compatibility */
    union{
        long Long[1];
        short Short[1];
        unchar Char[1];
    }Value;                                         /* paramètres de l'option */
} MCX_OPTION, *PMCX_OPTION;
#define MCX_OPTION_SIZE(dlen) \
                           ((dlen)+sizeof(MCX_OPTION)-sizeof(long))
```

II.2 Fonctions SET/GET SYNC STATE

Deux fonctions ont été ajoutées pour paramétrer le format des trames synchrones.

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_SET_SYNC_STATE**,
frameFormatBuffer, **sizeof(MCX_SYNCHRONIZATION_PARAMETERS)**,
NULL, **0**, *lpcbBytesReturned*, *lpoOverlapped*)

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_GET_SYNC_STATE**, **NULL**, **0**,
frameFormatBuffer, **sizeof(MCX_SYNCHRONIZATION_PARAMETERS)**,
lpcbBytesReturned, *lpoOverlapped* **)**

HANDLE *hDevice*; /* Handle of the device */
PMCX_SYNCHRONIZATION_PARAMETERS *frameFormatBuffer*; /* pointer to parameters */
LPDWORD *lpcbBytesReturned*; /* size of returned params */
LPOVERLAPPED *lpoOverlapped*; /* overlapped struct. addr */

La fonction SET permet de choisir la forme des trames : HDLC, etc. L'entier pointé par *lpcbBytesReturned* vaudra toujours 0. **ATTENTION:** dans les versions de pilote antérieures à 1.8.3, cette fonction renvoie une erreur si l'état du port fixé préalablement par SetCommState, contient des options non supportées par la carte (par exemple *ByteSize*=7 en mode HDLC). Dans les versions suivantes, le port est forcé dans un état « raisonnable ».

La fonction GET permet de consulter les paramètres de trame courants. l'entier pointé par *lpcbBytesReturned* vaudra toujours sizeof(**MCX_SYNCHRONIZATION_PARAMETERS**).

La structure **MCX_SYNCHRONIZATION_PARAMETERS** est composée des éléments suivants :

UCHAR *SynchronousMode*;
Protocole : **MCX_SYNC_HDLC**, **MCX_SYNC_BISYNC**, **MCX_SYNC_LAPB** ou **MCX_SYNC_CHAR**. Le mode **MCX_SYNC_CHAR** correspond aux transmissions asynchrones, c'est le seul mode valide en mode de base. LAPB est aussi connu sous le nom HDLC/ABM.

UCHAR *Duplex*;
Simultanéité de transmission : **MCX_WAY_FULLDUPLEX** (émission et réception simultanées), ou bien **MCX_WAY_HALFDUPLEX** (émission et réception alternées ; voir dans le manuel Multiprotocole [DT003] les particularités de ce mode).

Attention : la plupart des fonctionnalités du mode half-duplex peuvent être obtenues avec un paramétrage correct du mode full-duplex. N'utiliser le mode half-duplex que s'il est incontournable (si c'est la seule façon d'empêcher l'émission lorsqu'une réception est en cours).

Le mode half-duplex force *fOutxCtsFlow* = TRUE, *fRtsControl* = RTS_CONTROL_TOGGLE, il ignore les données reçues en l'absence de DCD, il interdit d'émettre quand DCD est actif, et aussi quand DSR est actif alors que *fDsrSensitivity* = TRUE.

UCHAR *TransmitClockSource*; Origine de l'horloge d'émission (voir ci-dessous).

UCHAR *ReceiveClockSource*; Origine de l'horloge de réception (voir ci-dessous).

Constante	origine de l'horloge
MCX_CLOCK_RXC	broche 17 (RxClock)
MCX_CLOCK_TXCI	broche 15 (TxClock)
MCX_CLOCK_BRG	générateur de bauds interne
MCX_CLOCK_DPLL	décodée dans les données (seulement en codage FM ou Manchester)

UCHAR *TxClockPinSource*; Origine de l'horloge disponible broche 24 (broche 15 sur MCXBP rev. A). Attention, cette broche est désactivée si **MCX_CLOCK_TXCI** est utilisée *(*par *TransmitClockSource* ou *ReceiveClockSource)*.

Constante	origine de l'horloge
MCX_CLOCK_TRXC_HIGH	aucune (broche à l'état MARK)
MCX_CLOCK_TXCLOCK	comme *TransmitClockSource*
MCX_CLOCK_BRG	générateur de bauds interne
MCX_CLOCK_DPLL	décodée dans les données (seulement en codage FM ou Manchester)

UCHAR *MonosyncChar*; Premier caractère de synchronisation en mode BISYNC. Caractère de synchronisation unique en mode MONOSYNC. Ignoré dans les autres modes.

UCHAR *BisyncChar*; Second caractère de synchronisation en mode BISYNC. Ignoré dans les autres modes.

UCHAR *Version*; Version de la structure. Doit toujours valoir 1.

USHORT *Options*; Options du protocole. Chaque option est un bit qui doit être ajouté si l'option est souhaitée.

- **MCX_HDLC_USERDTR** autorise l'utilisation du circuit CCITT 108 (DTR) dans les modes HDLC, LAPB, X25 sur les canaux 1, 2, 3 des cartes avec extension MCXBP(MR) ou Lite/S ou PCB/S, au détriment des performances qui sont alors limitées[1].
- **MCX_HDLC_SPECS** force l'utilisation des éléments de la structure *Protocol.Hdlc*, sinon, ils sont ignorés.
- **MCX_BISYNC_SPECS** force l'utilisation des éléments de la structure *Protocol.Bisync*, sinon, ils sont ignorés.

USHORT *DataLength*; Taille maximum d'une trame. La valeur zéro appelle la taille par défaut (voir la documentation Multiprotocole [DT003], commande PROTO).

union {...} *Protocol*; Les sous-structures de cette union permettent de préciser les paramètres pour un protocole particulier.

[1] Voir la documentation de PROTO dans le « Manuel d'utilisation du logiciel Multiprotocole » [DT003]

Les éléments suivants ne doivent être fournis que pour LAPB :

UCHAR *Protocol.Lapb.Role*; **MCX_ROLE_CLIENT** si l'application joue le rôle d'un client (ETTD), **MCX_ROLE_NETWORK** si l'application joue le rôle du fournisseur de réseau.

UCHAR *Protocol.Lapb.K,N2,T1,T2,T3*
Paramètres normalisés de LAPB. **MCX_LAPB_DEFAULT** appelle la valeur par défaut[2].

USHORT *Protocol.Lapb.Spare*; Zone réservée pour LAPB.

Les éléments suivants ne doivent être fournis que pour HDLC :

UCHAR *Protocol.Hdlc.RxFrames*;
Nombre de trames pouvant être reçues sans être perdues si le PC ne les lit pas immédiatement de la carte (nombres de tampons de réception de trames sur la carte). La valeur 0 installe la valeur par défaut[2].

UCHAR *Protocol.Hdlc.TxFrames*;
Nombre de trames pouvant être mémorisées en attente d'émission par la carte (nombres de tampons d'émission de trames sur la carte). La fonction **WriteFile**() n'est jamais bloquante si un tampon de trame est disponible sur la carte au moment de son appel. La valeur 0 installe la valeur par défaut[2].

USHORT *Protocol.Hdlc.Spare1*; à *Spare5;*
Zones réservées pour HDLC.

Les éléments suivants ne doivent être fournis que pour BISYNC :

UCHAR *Protocol.Bisync.RxFrames*;
Nombre de trames pouvant être reçues sans être perdues si le PC ne les lit pas immédiatement de la carte (nombres de tampons de réception de trames sur la carte). La valeur 0 installe la valeur par défaut[2].

UCHAR *Protocol.Bisync.TxFrames*;
Nombre de trames pouvant être mémorisées en attente d'émission par la carte (nombres de tampons d'émission de trames sur la carte). La fonction **WriteFile**() n'est jamais bloquante si un tampon de trame est disponible sur la carte au moment de son appel. La valeur 0 installe la valeur par défaut[2].

USHORT *Protocol.Bisync.Spare1* à *Spare5 ;*
Zones réservées pour BISYNC.

[2] Voir la documentation de PROTO dans le « Manuel d'utilisation du logiciel Multiprotocole » [DT003]

II.3 Exemple pour SET_SYNC_STATE

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

Proto(HANDLE chan, int fonc)
{
     MCX_SYNCHRONIZATION_PARAMETERS sp;
     DWORD count;
     DWORD speed;

     //
     // passer en mode HDLC avec les horloges appropriées
     // pour un câble NULL-MODEM (horloges croisées)
     // avec 1024 octets max. par trame
     //
     sp.SynchronousMode = MCX_SYNC_HDLC;
     sp.Version = 1;
     sp.Duplex = MCX_WAY_FULLDUPLEX;
     sp.Options = 0;
     sp.DataLength = 0;  /* 1024 octets par défaut */
     sp.TransmitClockSource = MCX_CLOCK_BRG;
     sp.ReceiveClockSource = MCX_CLOCK_RXC;
     sp.TxClockPinSource = MCX_CLOCK_BRG;

     if(!DeviceIoControl(
               chan,IOCTL_SERIAL_SET_SYNC_STATE,
               &sp,sizeof(sp),NULL,0,&count,NULL)) {
          printf("SET_SYNC_STATE Ioctl: error %d\n",
     GetLastError());
          exit(1);
     }
}
```

II.4 Fonctions CMD et CMD_AUTO

Deux fonctions de communication ont été ajoutées pour dialoguer manuellement avec l'interpréteur implanté sur la carte. La description des commandes, de leurs paramètres et de la zone de données figure dans le manuel du « firmware » utilisé (logiciel de base [DT002] ou logiciel multiprotocole [DT003], suivant le cas).

```
#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl( hDevice, IOCTL_SERIAL_CMD,
        paramsFromAppToBoard, paramsToSize,
        paramsFromBoardToApp, paramsFromSize, lpcbBytesReturned,
        lpoOverlapped
        )

DeviceIoControl( hDevice, IOCTL_SERIAL_CMD_AUTO,
        paramsFromAppToBoard, paramsAppSize,
        paramsFromBoardToApp, paramsBoardSize, lpcbBytesReturned,
        lpoOverlapped
        )

HANDLE hDevice;                 /* Handle of the device */
PMCC_CMD paramsFromAppToBoard;  /* pointer to sent parameters */
DWORD paramsAppSize,            /* size of sent params */
PMCC_CMD paramsFromBoardToApp;  /* pointer to returned parameters */
DWORD paramsBoardSize,          /* size of space for returned params */
LPDWORD lpcbBytesReturned;      /* size of returned params */
LPOVERLAPPED lpoOverlapped;     /* overlapped struct. addr */
```

La fonction **IOCTL_SERIAL_CMD** permet de faire exécuter à la carte une commande quelconque.

La fonction **IOCTL_SERIAL_CMD_AUTO** permet de faire exécuter à la carte une commande quelconque, dont le paramètre 1 sera initialisé par le pilote avec le numéro de canal correspondant à *hDevice*.
Le type **MCC_CMD** est une structure qui reflète celle de la boîte aux lettres de la carte, permettant à l'application d'émettre et de recevoir des paramètres et des données.

Lorsqu'une commande est envoyée à la carte par ces fonctions, le pilote effectue les actions suivantes :

1) Si elle existe, la table *paramsFromAppToBoard→Data* est recopiée dans la zone DONNEES de la boîte aux lettres de la carte,
2) la table *paramsFromAppToBoard→Cb.par*[] est recopiée dans la zone PARAMETRES de la boîte aux lettres,
3) *paramsFromAppToBoard→Cb.opcode* est recopié dans la zone OPCODE de la boîte aux lettres,
4) la valeur binaire 0000 0001 est écrite dans l'octet de VALIDATION de la boîte aux lettres, ce qui provoque l'exécution de la commande par la carte ; puis la carte émet une interruption de fin de commande qui permet au pilote de poursuivre le traitement,

5) la zone STATUS de la boîte aux lettres est copiée dans *paramsFromBoardToApp→ Cb.status*
6) la zone PARAMETRES de la boîte aux lettres est copiée dans *paramsFromBoardToApp→ Cb.par*[],
7) si la table *paramsFromAppToBoard→Data* existe, la zone DONNEES de la boîte aux lettres est recopiée dans *paramsFromBoardToApp→Data*,
8) l'application est réveillée ou avertie, selon la valeur de *lpoOverlapped*.

Le champ *Data* de **MCC_CMD** est de taille variable. Par exemple, on peut créer cette structure par une allocation dynamique de (sizeof(**MCC_CMD**)+taille_des_données) octets ; la macro **MCX_DIRECT_IO_BUFFER_SIZE(***datalen***)** permet de calculer le nombre d'octets nécessaire. Le pilote reconnaît l'existence du champ *Data* si le champ *Cb.length* n'est pas nul. Sinon, il suppose que la commande à exécuter n'utilise pas la zone DONNEES de la boîte aux lettres.

paramsAppSize doit être égal à la somme de sizeof(**MCC_CMD**) et de la longueur de la zone de données.

Au retour, l'entier pointé par *lpcbBytesReturned* sera toujours égal à *paramsBoardSize* si la commande a été exécutée normalement.

La structure **_MCC_CMD** est composée des éléments suivants :

unsigned char *Cb.opcode*;	Code de la commande à exécuter. **mcc_mcx.h** définit les noms symboliques de ces codes.
unsigned char *Cb.status*;	En retour, résultat fourni par la carte dans la zone STATUS.
unsigned char *Cb.par[76]*;	Paramètres de la commande à exécuter ; paramètres en retour pour certaines commandes.
unsigned char *Cb.ichan*;	}
unsigned char *Cb.icond*;	} En retour, copie de la zone INTERRUPTION de la boîte
unsigned char *Cb.ipar1*;	} aux lettres. Ces cinq éléments ne sont en principe pas
unsigned char *Cb.ipar2*;	} utilisés[3].
unsigned char *Cb.ipar3*;	}
unsigned char * *Cb.data*;	Inutilisé sous Windows NT[4].
unsigned char * *Cb.kdata*;	Champ utilisé temporairement par le pilote durant l'exécution de la commande.
unsigned short *Cb.length*;	longueur utile de la zone de données, en octets.
unsigned char *Data*[0...];	Zone de données qui sera échangée avec la boîte aux lettres de la carte ; cette zone doit suivre immédiatement la structure *Cb*, d'où l'utilisation de la structure de taille variable **MCC_CMD**[5].

[3] sauf pour la commande LDIAL de la carte MCC.
[4] Sous UNIX, pointeur vers la zone de données qui sera échangée avec la boîte aux lettres de la carte.
[5] Inversement cette zone n'est pas utilisée dans le pilote UNIX.

Notes

- ☞ Il est conseillé, pour éviter toute erreur, de passer le même pointeur dans *paramsFromAppToBoard* et *paramsFromBoardToApp*, et la même longueur dans *paramsAppSize* et *paramsBoardSize*.
- ☞ Pour la compatibilité avec le pilote UNIX, il est suggéré d'initialiser *paramsFromAppToBoard→Cb.data = paramsFromAppToBoard→Data*.
- ☞ Pour éviter la confusion entre *Cb.par*[] qui commence à *Cb.par*[0], et la description des commandes qui commence au PARAMETRE 1, il est suggéré d'écrire : *paramsFromAppToBoard→Cb.par*[**N-1**] pour citer le PARAMETRE **N**.

Interactions entre ces fonctions et le fonctionnement normal du pilote

La description ci-dessous est à jour pour la version 1.8.3 du pilote. Ces interactions sont susceptibles de modifications dans les versions futures.

ALLOC :

Le pilote détecte cette commande et ajuste ses informations concernant la taille des tampons si nécessaire. Elle peut donc remplacer SetupComm() qui est ignoré.

CHDEF :

Un CHDEF est exécuté par le pilote pendant SetCommState, dans le cas ou l'on cherche à modifier *DCB.EvtChar*. Seul *EvtChar* est alors placé dans la commande CHDEF.

MINTR :

- Cette commande est exécutée durant CreateFile, CloseHandle, DeviceIoControl (IOCTL_SERIAL_SET_SYNC_STATE), SetCommMask (si on active ou on désactive l'événement EV_RXFLAG) et dans certains cas durant ReadFile.
- Les sources activées sont: IT1, IT2 (bit Mde=1), IT3 (si EV_RXFLAG est actif), IT5, IT6, IT7.

PROTO :

- Le pilote détecte cette commande et ajuste ses propres informations concernant le protocole. Elle peut donc remplacer le DeviceIoControl SET_SYNC_STATE lorsqu'on souhaite utiliser des paramètres non standardisés ; cela ne perturbe pas le fonctionnement du pilote.
- Il est ensuite nécessaire d'exécuter aussi les commandes VINIT, RXENB, MINTR et éventuellement VMODE, EscapeCommFunction() pour positionner RTS et DTR, et PurgeComm() pour vider les tampons.

RSMDE :

- Aucune interaction indésirable.

II.5 Exemples pour CMD et CMD_AUTO

Ces exemples peuvent aussi être adaptés pour envoyer d'autres commandes documentées dans les manuels des firmwares.

SetElectricalInterface() passe un canal série MCX en mode RS232, RS422, EIA530... en envoyant la commande RSMDE au « firmware » de la carte.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>
...
//
// Pour d'autres codes d'interfaces électriques disponibles,
// voir dans la documentation du firmware Multiprotocole, la
// section consacrée à la commande RSMDE.
//
#define RSMDE_RS232     0
#define RSMDE_RS422     1
#define RSMDE_EIA530    6

BOOLEAN SetElectricalInterface(HANDLE hfd)
{
     MCC_CMD cmd;
     long return_bytes;
     int lasterror;

     //
     // init opcode et paramètres, voir doc logiciel de base
     // ou multiprotocole, commande RSMDE
     //
     cmd.Cb.opcode = RSMDE;
     cmd.Cb.par[1-1] = 0; // Sera remplacé par le n° de canal
     cmd.Cb.par[2-1] = RSMDE_RS422; // Passer en mode RS422
     // Remplacer RSMDE_RS422 par la valeur adéquate pour
     // votre application
     cmd.Cb.length = 0;   // Pas de zone de données nécessaire
     // S'inspirer de RELRP pour utiliser une zone de données

     if (!DeviceIoControl( hfd, IOCTL_SERIAL_CMD_AUTO,
               &cmd,sizeof(cmd),
               &cmd,sizeof(cmd),
               &return_bytes,NULL)) {
          printf("Set422: error, code %d\n",GetLastError());
          return FALSE;
     }
     if (command->Cb.status != 0) {
          printf("RSMDE: failed, status %d\n",
               command->Cb.status);
          return FALSE;
     }
     return TRUE;
}
```

Relrp() lance sur la carte la commande RELRP, qui n'est liée à aucun canal particulier. Cette commande renvoie des informations sur le type de carte et son équipement.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

BOOL Relrp()
{
   PMCC_CMD command;/* espace pour RELRP et ses params */
   DWORD retLen;    /* longueur renvoyee par DeviceIoControl */
   int cmdLen;      /* longueur de la structure de commande */
   int dataLen;     /* Length of the RELRP Data area */
   HANDLE hDevice;

   // on doit passer par l'un des canaux (peu importe lequel)
   hDevice = CreateFile("\\\\.\\COM3",
               GENERIC_WRITE|GENERIC_READ, 0, NULL,
               OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

   //
   // init opcode et données, voir doc logiciel de base
   // ou multiprotocole, commande RELRP
   //
   dataLen = 18;
   cmdLen = MCX_DIRECT_IO_BUFFER_SIZE(dataLen);
   command = malloc(cmdLen);
   command->Cb.opcode = RELRP;
   command->Cb.length = dataLen;
   // l'affectation qui suit permet de distinguer les cartes
   // MCC, lesquelles ne modifient pas la donnée 18.
   command->Data[18-1] = MCX_TYPE_MCC;

   if (!DeviceIoControl(hDevice,IOCTL_SERIAL_CMD,
        command, cmdLen, command, cmdLen, &retLen, NULL)) {
      printf("RELRP: Win32 error %d\n", GetLastError());
      return FALSE;
   }
   if (command->Cb.status != 0) {
      printf("RELRP: failed, status %d\n", command->Cb.status);
      return FALSE;
   }
   printf("RELRP: carte type %d à %d MHz, %d canaux\n",
        command->Data[18-1], command->Data[11-1],
        command->Data[9-1]);

   free(command);
   CloseHandle(hDevice);
   return TRUE;
}
```

II.6 Fonction ACCESS_AREA

Cette fonction permet de lire et d'écrire directement dans la boîte aux lettres de la carte.

```
#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"
```

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_ACCESS_AREA**,
pAccessDescriptor, accessDescriptorLength,
pGetBuffer, getBufferLength, returnedLength,
lpoOverlapped)

```
HANDLE hDevice;                          /* Handle of the device */
PMCX_AREA_DESCRIPTOR pAccessDescriptor;  /* pointer to operation and its
                                            parameters */
DWORD accessDescriptorLength;            /* size of operation and its parameters */
PVOID pGetBuffer;                        /* pointer to returned data, NULL if unused */
DWORD getBufferLength;                   /* size of buffer for return data, 0 if unused */
LPDWORD returnedLength;                  /* pointer to actual size of returned data */
LPOVERLAPPED lpoOverlapped;              /* overlapped structure address */
```

Cette fonction permet de lire ou de modifier le contenu de la boîte aux lettres de la carte. La structure pointée par *pAccessDescriptor*, de longueur *accessDescriptorLength*, est transmise au pilote. Le pilote traite l'opération. Il renvoie éventuellement des données dans la structure pointée par *pGetBuffer* de longueur *getBufferLength*, et il indique la longueur des informations renvoyées dans le mot pointé par *returnedLength.* Les informations renvoyées sont tronquées à la plus courte des tailles indiquées par *getBufferLength* et **returnedLength*.

La structure **_MCX_AREA_DESCRIPTOR** est composée des éléments suivants :

long *Operation*;	Code de l'opération (voir plus bas).
long *StartAddress* ;	Adresse, relative au début de la boîte aux lettres de la carte que pilote *hDevice*, ou se fera l'écriture ou la lecture.
long *Length* ;	nombre d'octets à transférer.
unsigned char *Buffer*[...];	Données à écrire dans la boîte aux lettres (utilisé par l'opération MCX_AREA_SET).

Buffer[] ne contient qu'un seul octet. Pour l'écriture de plusieurs caractères, on peut allouer dynamiquement une structure de la taille requise grâce à la macro :

MCX_AREA_DESCRIPTOR_SIZE(datalen).

Les opérations reconnues sont :

MCX_AREA_GET+MCX_AREA_MEMORY Le contenu de la boîte aux lettres à l'adresse *StartAddress* est recopié sur une longueur *Length* à l'emplacement pointé par *pGetBuffer*, sur une longueur maximum de *getBufferLength*. **returnedLength* est chargé avec la longueur effectivement copiée.

MCX_AREA_SET+MCX_AREA_MEMORY Le contenu de *Buffer* est recopié à l'adresse *StartAddress* de la boîte aux lettres, sur une longueur *Length*.

II.7 Fonction MCX_OPTIONS

Une fonction a été ajoutée pour accéder à divers paramètres ou activer des comportements spécifiques aux cartes MCC et MCX.

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_MCX_OPTIONS**,
pOptionDescriptor, *optionDescriptorLength*, *pResultsBuffer*,
resultsBufferLength, *returnedResultsLength*, *lpoOverlapped*)

HANDLE *hDevice*;	/* Handle of the device */
PMCX_OPTION *pOptionDescriptor*;	/* pointer to option code and parameters */
DWORD *optionDescriptorLength*;	/* size of option code and parameters */
PMCX_OPTION *pResultsBuffer*;	/* pointer to returned data, NULL if unused */
DWORD *resultsBufferLength*;	/* size of buffer for return data, 0 if unused */
LPDWORD *returnedResultsLength*;	/* pointer to actual size of returned data */
LPOVERLAPPED *lpoOverlapped*;	/* overlapped structure address */

Cette fonction permet de modifier certains paramètres spécifiques au pilote et à la carte. La structure pointée par *pOptionDescriptor*, de longueur *optionDescriptorLength*, est transmise au pilote. Le pilote traite l'option. Il renvoie éventuellement des données dans la structure pointée par *pResultsBuffer* de longueur *resultsBufferLength*, et il indique la longueur des informations renvoyées dans le mot pointé par *returnedResultsLength*. Les informations renvoyées sont éventuellement tronquées à la taille indiquée par *resultsBufferLength*.

La structure **_MCX_OPTIONS** est composée des éléments suivants :

long *Option*;	Code de l'option ; utile seulement dans la structure transmise.
union{long *Long***; ...}** *Value*;	Paramètres/données retournées de l'option. Leur type (**long**, **short** ou **unsigned char**) dépend de l'option considérée. Les différents éléments (*Long*, *Short*, *Char*) de l'union seront utilisés suivant le cas.

Si une option requiert plusieurs paramètres, la macro **MCX_OPTION_SIZE**(paramslen) permet d'allouer dynamiquement une structure de la taille requise.

Les options reconnues dans cette version sont :

MCX_OPTION_GET_CHANNEL

Demande le numéro du canal correspondant à *hDevice*. La structure renvoyée contient un **short** (*pOptionDescriptor→ Value.Short[0])* qui indique le numéro de canal, relatif à la carte sur laquelle il est installé. Par exemple, si deux cartes MCX seize voies sont installées avec les noms COM3...COM34, le numéro renvoyé pour le handle de COM3 et COM19 sera 1, le numéro renvoyé pour COM18 et COM34 sera 16.

Des informations supplémentaires pourront être renvoyées dans une version future. **returnedResultsLength* pourra donc être supérieur à *resultsBufferLength*.

MCX_OPTION_GET_RSMODE

L'élément *pOptionDescriptor→Value.Char[0]* de la structure renvoyée contient le code de l'interface électrique actuellement programmée sur ce port. Les valeurs possibles sont documentées dans le manuel du « firmware » utilisé.

MCX_OPTION_SET_DSR_RI_INVERSION

L'élément *pOptionDescriptor→Value.Char[0]* de la structure fournie doit contenir TRUE ou FALSE. S'il contient TRUE, Les traitements des signaux DSR et RING sont intervertis sur ce canal.

MCX_OPTION_GET_DSR_RI_INVERSION

L'élément *pOptionDescriptor→Value.Char[0]* de la structure renvoyée contient TRUE si les traitements des signaux DSR et RING sont intervertis sur ce canal, et FALSE sinon.

MCX_OPTION_GET_COMPATIBILITY

L'élément *pOptionDescriptor→Value.Long[0]* de la structure renvoyée contient la valeur de la variable Compatibility telle qu'elle est utilisée sur ce canal. Cette variable reflète les options de configuration du pilote, éventuellement modifiées par la fonction MCX_OPTIONS. Les bits de la valeur renvoyée peuvent être analysés avec les définitions suivantes disponibles dans "mcc_mcx.h" :

```
MCX_COMPAT_LOWSIGS -- RTS/DTR bas au 1er CreateFile
MCX_COMPAT_DSR_RI  -- inversion DSR & RING
MCX_COMPAT_DTRFLOW-- compat.1.6.2(EPROM 1.8/3.8)
MCX_COMPAT_V164    -- compatibilité V1.6.4
MCX_COMPAT_V171    -- compatibilité V1.7.1
MCX_COMPAT_V181    -- compatibilité V1.8.1
MCX_COMPAT_QSIZE   -- compatibilité V1.8.2
```

Consulter la section d'installation pour plus de détails.

MCX_OPTION_GET_PHYSICAL_PARMS

Demande les paramètres physiques de la carte. La structure renvoyée contient MCX_OPTION_PHYSICAL_PARMS, qui est une sous structure alignée avec *pOptionDescriptor→ Value.Long[0]*. Les éléments définis dépendent du type de carte et de bus. Pour une carte MCXPCI, les éléments de ressources physiques suivants sont renvoyés :

StructVersion	0
BusType;	5 (macro « PCIBus » du DDK)
BusNumber;	Numéro du bus PCI où est la carte
SlotNumber;	Emplacement de la carte sur le bus
MemoryAddressLow;	Adresse de la boîte aux lettres
MemoryAddressHigh;	Adresse, poids forts (bus 64 bits)
MemorySize;	32768
IoAddressLow;	Adresse des ports d'E/S de la carte
IoAddressHigh;	Adresse, poids forts (bus 64 bits)
IoSize;	8
InterruptVector;	Interruption
DmaChannel;	-1
SpecificInformation;	0

Un exemple d'utilisation de cette structure est fourni sur le médium d'installation sous le nom
« \sdk\multiprotocol\examples\common\special\getphys.c ».

II.8 Exemples pour ACCESS_AREA et MCX_OPTIONS

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

DemoOptions(HANDLE hDevice)
{
     DWORD retLen;  /* longueur renvoyée dans les appels */
     int canal;     /* canal physique, résulte de GET_CHANNEL*/


     { /* obtenir le numéro de canal */

          PMCX_OPTION pOptionDesc; /* exemple avec malloc */
          MCX_OPTION optionResult; /* exemple sans malloc */

          pOptionDesc = malloc(MCX_OPTION_SIZE(0));
          pOptionDesc->Option = MCX_OPTION_GET_CHANNEL;
          DeviceIoControl(hDevice,IOCTL_SERIAL_MCX_OPTIONS,
                         pOptionDesc, MCX_OPTION_SIZE(0),
                         &optionResult, sizeof(optionResult),
                         &retLen, NULL);
          free(pOptionDesc);
          canal = optionResult.Value.Short[0];
     }

     /* la ligne suivante charge le tampon de la carte */
     WriteFile(hDevice, “ABC”, 3, &retLen, NULL);
     /* Il n’y a pas de struct Overlap, donc BTRAN est */
     /* terminé quand WriteFile se termine */

     { /* récupérer le compteur d’émission avec ACCESS_AREA */
       /* remarque : un résultat équivalent est obtenu plus */
       /* simplement par la fonction ClearCommError(Win32)  */

          MCX_AREA_DESCRIPTOR areaDesc;
          USHORT txCount;

          areaDesc.Operation = MCX_AREA_GET+MCX_AREA_MEMORY;
          areaDesc.StartAddress = 0x7f80 + 2*(canal-1);
          areaDesc.Length = sizeof(txCount);
          DeviceIoControl(hDevice,IOCTL_SERIAL_ACCESS_AREA,
                         &areaDesc, sizeof(areaDesc),
                         &txCount, sizeof(txCount),
                         &retLen, NULL);
          printf("Place libre dans le tampon d’émission : ");
          printf("%u\n",txCount);
     }
}
```

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

DemoPhysical(HANDLE hDevice)
{
      DWORD retLen;         /* longueur renvoyée dans les appels */
      PMCX_OPTION pOptionDesc;
      PMCX_OPTION_PHYSICAL_PARMS pOptionResult;
      int optionLen;

      optionLen = MCX_OPTION_SIZE(sizeof(MCX_OPTION_PHYSICAL_PARMS));
      pOptionDesc = malloc(optionLen);

      pOptionDesc->Option = MCX_OPTION_GET_PHYSICAL_PARMS;
      DeviceIoControl(hDevice,IOCTL_SERIAL_MCX_OPTIONS,
            pOptionDesc, optionLen,    // paramètres fournis
            pOptionDesc, optionLen,    // informations renvoyées
            &retLen, NULL);

      pOptionResult = (PMCX_OPTION_PHYSICAL_PARMS)
                           pOptionDesc->Value.Long ;
      printf("Adresse physique de la carte : %x%x / %x%x / %d\n",
            pOptionResult->MemoryAddressHigh,
            pOptionResult->MemoryAddressLow,
            pOptionResult->IoAddressHigh,
            pOptionResult->IoAddressLow,
            pOptionResult->InterruptVector);

      free(pOptionDesc);
}
```

III UTILISATION DE CONTROLES DE FLUX SPECIAUX.

Les cartes de la gamme MCX, équipées de l'option MCX-MULTIPROTOCOLE, et utilisées avec une version du pilote postérieure à 1.7.0, **supportent tous les contrôles de flux offerts dans l'API Win32**, et même des contrôles supplémentaires (accessibles en exécutant directement la commande VMODE).

Les cartes équipées du logiciel de base, ou de l'option Multiprotocole utilisée avec un pilote antérieur à la version 1.7.1, ne peuvent pas supporter tous les contrôles de flux proposés dans l'API de Win32. Les limites de fonctionnement du contrôle de flux sont décrites ci-dessous :

Méthodes supportées

Les contrôles de flux supportés sont les suivants :

- aucun
- XON/XOFF paramétrable
- DTR/CTS
- RTS/CTS à partir des «firmwares» rév. 2.0 (MCX) et 3.8 (MCC)

Dans tous les cas, le contrôle est fait dans les deux sens de transmission. Il est impossible de paramétrer indépendamment les deux sens.

Paramétrage de la méthode désirée

Le service *SetCommState* et l'interprétation des champs de la structure *DCB* (voir page 33) ont été adaptés pour tenir compte de ces contraintes. L'utilisation de l'option Multiprotocole et des options de configuration de la carte (c'est-à-dire des valeurs *Synchronous* et *Compatibility* du Registre) influent aussi sur le fonctionnement. Le contrôle de flux est donc supporté de la façon suivante :

a) si l'indicateur *Synchronous* vaut 1 (avec un ancien pilote et/ou un ancien « firmware »), il n'y a pas de contrôle de flux, car alors la carte ne supporte pas la commande VMODE,
b) sinon, si *fInX* ou *fOutX* est TRUE, alors le contrôle de flux est XON/XOFF avec les caractères *XonChar* et *XoffChar*,
c) sinon, si *fRtsControl* vaut RTS_CONTROL_HANDSHAKE, alors le contrôle de flux est matériel avec les signaux RTS et CTS,
d) sinon, si *fOutxCtsFlow* est TRUE ou *fOutxDsrFlow* est TRUE ou *fDtrControl* vaut DTR_CONTROL_HANDSHAKE, alors le contrôle de flux est matériel avec les signaux DTR et CTS,
e) sinon, il n'y a pas de contrôle de flux.

Compatibilité avec les versions antérieures

Les versions anciennes des cartes et des logiciels ne géraient pas le contrôle de flux par RTS/CTS. Dans les cas suivants :

- Si la version du pilote est inférieure ou égale à 1.6.2,
- ou si l'option de configuration « EPROM 1.8/3.8 » est validée,
- ou si la version du «firmware» de la carte est antérieure à 1.8,

Alors les règles c) et d) doivent être combinées en une règle unique :

c+d) sinon, si *fOutxCtsFlow* est TRUE ou *fOutxDsrFlow* est TRUE ou *fDtrControl* vaut DTR_CONTROL_HANDSHAKE ou *fRtsControl* vaut RTS_CONTROL_HANDSHAKE, alors le contrôle de flux est matériel avec les signaux DTR et CTS,

Remarque

Pour assurer la compatibilité avec les versions futures, utiliser la combinaison correspondant au câble réellement utilisé (par exemple si le signal de contrôle arrive sur CTS, ne pas le gérer avec *fOutxDsrFlow* mais avec *fOutxCtsFlow*).

IV ANNEXE : CODES ERREUR COURANTS.

Les codes erreur renvoyés par l'API Win32 sont conformes à la documentation de cet API. Insistons particulièrement sur quelques codes erreur dont l'interprétation n'est pas immédiatement évidente :

2 **ERROR_FILE_NOT_FOUND**

Le pilote n'est pas démarré. Consulter l'observateur d'événements et le gestionnaire de périphériques.

5 **ERROR_ACCESS_DENIED**

Le canal est déjà ouvert par un autre processus.

21 **ERROR_NOT_READY**

Dans **WriteFile**, l'émission est impossible car la couche liaison (LAPB) n'est pas connectée.

23 **ERROR_CRC**

Dans un **ReadFile**, indique que la trame reçue contient une erreur (tout type d'erreur engendre ce code erreur, pas seulement les erreurs de CRC).

38 **ERROR_HANDLE_EOF**

En protocole LAPB, indique que la couche liaison est déconnectée avant ou pendant l'exécution d'un **ReadFile**, et qu'aucune trame ne subsiste dans le tampon de réception.

57 **ERROR_ADAP_HDW_ERR**

Erreur inattendue sur la commande MINTR. Anomalie probable de la carte.

87 **ERROR_INVALID_PARAMETER**

1) Dans **DeviceIoControl**, **WaitCommEvent**, **ReadFile** et **WriteFile**, cette erreur peut, en particulier, indiquer que le paramètre *lpoOverlapped* n'est pas conforme aux options demandées dans le **CreateFile** ; ou bien que l'élément *hEvent* de la structure OVERLAP est incorrect.
2) Dans **DeviceIoControl**, soit l'un des paramètres, soit l'un des éléments de la structure passée en troisième paramètre, est incorrect.

122 **ERROR_INSUFFICIENT_BUFFER**

1) Dans **DeviceIoControl**, la longueur passée est incorrecte.
2) Dans **DeviceIoControl**, la valeur de l'élément *Cb.length* de la fonction IOCTL_SERIAL_CMD ou CMD_AUTO est trop petite.

995 **ERROR_OPERATION_ABORTED**

1) La carte n'a pas répondu à une commande dans le temps imparti. Les causes probables sont : l'interrupteur d'IRQ (bus ISA) est mal enfoncé ou n'est pas le bon ; la carte est très chargée, il faut augmenter le paramètre de démarrage *CommandTimeout* ; des parasites arrivent en permanence sur le canal.
2) En protocole LAPB, un **WriteFile** a été tenté alors que la liaison n'était pas, ou plus, établie.

997 **ERROR_IO_PENDING**

L'opération n'est pas terminée. Voir **GetOverlappedResult**().

1450 **ERROR_NO_SYSTEM_RESOURCES**

Un **DeviceIoControl** envoyé au pilote n'est pas reconnu. Les causes les plus probables sont : l'application envoie un **DeviceIoControl** dont le code est erroné ; la version de Windows NT n'est pas supportée.

1784 **ERROR_INVALID_USER_BUFFER**

Dans **WriteFile**, le paramètre '*count*' est trop grand par rapport à la taille du tampon ou des trames, ou il est supérieur à la limite de 31 k-octets.

V ANNEXE : LIMITATIONS ET DIFFERENCES AVEC LES PORTS COM.

Plusieurs limitations proviennent des interactions entre les possibilités respectives de l'API Win32, du pilote, du « firmware », et de la carte elle-même.

Le logiciel de base de la carte restreint les types de contrôle de flux disponibles (voir l'annexe « contrôle de flux »). L'option Multiprotocole, quoique moins performante en mode asynchrone, n'a pas cette limitation.

Vitesse de transmission 134,5 bauds : le DCB ne permet pas d'indiquer cette vitesse car la variable *BaudRate* est un LONG. Cependant le pilote supporte cette vitesse par le biais de la valeur (ULONG)(-134).

Le signal DSR (circuit 107) n'existe pas sur toutes les cartes (voir la documentation des connecteurs de la carte). Une option de configuration et une fonction de l'API permettent d'intervertir le traitement de ce signal avec RING, ce qui permet d'utiliser un pseudo-DSR moyennant un câblage adéquat.

En mode synchrone, le signal RING (circuit 125) n'est disponible qu'à partir de la révision E des boîtiers de connexion MCXBP.

DTR en mode synchrone : voir le manuel du logiciel Multiprotocole [DT003] et l'indicateur **MCX_HDLC_USERDTR** page 46.

Vitesses supportées : voir le manuel du « firmware » utilisé [DT002], [DT003].

VI QUESTIONS FREQUENTES RELATIVES AUX PORTS COM

Q : Mon programme exécute un WriteFile qui ne renvoie pas d'erreur, mais les données ne sont pas complètement émises sur le port.

R : Vous pouvez utiliser les fonctions suivantes pour attendre la fin d'émission : PurgeComm(), FlushFileBuffers(), CloseHandle() ou la terminaison du programme.

Q : Mon programme exécute un WriteFile qui ne renvoie pas d'erreur, mais aucune donnée n'est émise sur le port.

R : voir la question précédente. Vérifiez aussi que l'horloge de transmission est présente et correctement programmée (par la commande PROTO ou le DeviceIoControl `IOCTL_SERIAL_SET_SYNC_STATE`).

Q : Je ne détecte pas les variations de DSR.

R : DSR n'est pas supporté par les firmwares « Logiciel de Base » ni « Logiciel Multiprotocole ».
Vous pouvez utiliser l'onglet de configuration avancée de la carte pour contourner ce problème.

MODE MCXDOS/AUTOMCX

Les sections suivantes décrivent les modes de fonctionnement « Mcxdos/Automcx » et « Extension BIOS » du pilote.

Le mode Mcxdos/Automcx permet de charger une application sur la carte, dans un environnement DOS.

I DEVELOPPEMENT DE L'APPLICATION A TELECHARGER

Les application destinées à tourner sur la carte dans ce mode, doivent être réalisées et testées préalablement sur un ordinateur de développement fonctionnant sous MS-DOS, Windows 9× ou compatibles. Elles peuvent être réalisées par tout outil de développement natif ou croisé pourvu qu'il génère un code exécutable compatible DOS. Ainsi, les environnements MSC 6 et 7, MSVC 1.52, Borland C, Pharlap ont été utilisés avec succès.

La phase de développement nécessite l'outil MCXDOS qui permet le partage des périphériques du PC avec la carte MCX, puis la création d'un « BOOTFILE » qui est une image de disquette contenue dans un fichier disque de taille correspondante. Ce « BOOTFILE » contient le système DOS et l'application à télécharger.

II UTILITAIRES FOURNIS

Quatre utilitaires figurent sur la disquette dans le répertoire AUTOMCX\EXE :

AUTOMCX	programme de chargement du « BOOTFILE » vers la mémoire de la carte.
RESETMCX	programme de redémarrage à chaud d'une carte MCXPCI.
MCXIO	programme permettant d'exécuter des entrées-sorties sur la mémoire et les ports de la carte.
MCXDEBUG	programme permettant d'afficher et de modifier le contenu de la zone de mémoire double accès de la carte

III CHARGEMENT D'UNE CARTE

Lancer en ligne de commande l'utilitaire AUTOMCX avec deux paramètres : le « nom d'usage » de la carte saisi pendant l'installation, et le nom du fichier BOOTFILE. Par exemple :

```
automcx  mcx0403  MCXBOOT
```

IV INTERFACE DE PROGRAMMATION DE BAS NIVEAU

Les fonctions de Win32 « CreateFile », « DeviceIoControl » « CloseHandle » permettent l'accès à une carte configurée en mode automcx de la manière suivante :

```
#include <winioctl.h>
#include <mcc_mcx.h>

void *Mailbox;  /* Remplacer 'void' par un type de structure
                   appropriée pour l'application */
char *nom_carte[] = « \\\\.\\mcx_exemple »;
DWORD cbReturned;
MCX_AREA_DESCRIPTOR Area;  /* paramètres de transfert pour ports E/S
*/

/* ouvrir et fermer l'accès à la carte */
HANDLE Handle = CreateFile(nom_carte,
                    GENERIC_READ | GENERIC_WRITE,
                    0, NULL, OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL, NULL);
CloseHandle(Handle) ;

/* obtenir un pointeur sur la fenetre de mémoire partagée de la
carte */
BOOL ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_MAP_MAILBOX, NULL, 0,
                &Mailbox, sizeof(PVOID), &cbReturned, 0);

/* dévalider le pointeur ci-dessus pour libérer les ressources
utilisées */
BOOL ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_UNMAP_MAILBOX, NULL, 0,
                NULL, 0, &cbReturned, 0);

/* écrire sur un port d'E/S de la carte */
Area.Operation = MCX_AREA_SET|MCX_AREA_IOPORT;
Area.StartAddress = 0 à 7;      /* numéro relatif du port d'E/S */
Area.Length = 1;
Area.Buffer[0] = (UCHAR)cData;  /* caractère à écrire */
Ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_WRITE_PORT, &Area, sizeof(Area),
                NULL, 0, & cbReturned, 0);

/* lire un port d'E/S de la carte */
Area.Operation = MCX_AREA_GET|MCX_AREA_IOPORT;
Area.StartAddress = 0 à 7;      /* numéro relatif du port d'E/S */
Area.Length = 1;
ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_READ_PORT, &Area, sizeof(Area),
                &Area, sizeof(Area), &cbReturned, 0);
/* le résultat est disponible dans Area.Buffer[0] */

/* connaitre les ressources physiques de la carte */
```

Il est possible d'utiliser l'option **MCX_OPTION_GET_PHYSICAL_PARMS** de la fonction MCX_OPTIONS documentée à la section II.7. Se reporter à cette section pour plus de détails.

V INTERFACE SIMPLIFIEE EN C (FOURNIE A TITRE D'EXEMPLE)

Cette interface utilise l'interface de bas niveau pour offrir des fonctions plus conviviales. Vous pouvez soit inclure le source dans votre application, soit ajouter la librairie fournie « lib\automcx.lib » à votre projet.

Attention, cette interface est fournie à titre d'exemple : elle ne prétend pas fournir un outil évolué, ni traiter exhaustivement les cas d'erreur.

```
#include <autont.h>

/* ouvrir et fermer l'accès à la carte */
HANDLE Handle = OpenAutomcx(nom_de_carte) ;                /* le préfixe \\.\ est facultatif */
CloseAutomcx(Handle) ;

/* obtenir un pointeur sur la fenêtre de mémoire partagée de la
carte */
void *Mailbox = GetAutomcxPtr(Handle) ;

/* écrire sur un port d'E/S de la carte */
McxOutPort(Handle, port, valeur) ;

/* lire un port d'E/S de la carte */
/* renvoie –1 en cas d'erreur, sinon (int)(unsigned char)valeur */
int valeur = McxInPort(Handle, port) ;

/* renvoie le 1er octet du fifo, ou '\xff' en cas d'erreur */
unsigned char McxReadFifo(Handle) ;
```

VI DEFAUTS CONNUS

La synchronisation de la date n'est pas implémentée dans l'utilitaire AUTOMCX.EXE.

VII UTILISATION DE MCXDEBUG.EXE

MCXDEBUG est un programme similaire à l'outil debug de DOS ou Windows, à ceci près qu'il est limité aux 32 Ko de la boîte aux lettres des cartes MCX pour bus PCI et cPCI.

Les fonctions disponibles sont les suivantes :

- ? : help
- d : dump
- i : input
- o : output
- q : quit
- w : write

DUMP : lister 128 octets de la boîte aux lettres

Deux syntaxes différentes sont possibles :

- d
  Si vous tapez 'd' pour la première fois, le programme liste les 128 premiers octets de la boîte aux lettres. Si vous retapez 'd', il listera les 128 suivants et ainsi de suite jusqu'à la fin de la boîte aux lettres.

- d *offset*
  Permet d'atteindre directement l'*offset* désiré.

Remarques :

- La taille de l'offset est au maximum de 2 octets
  Si vous entrez 'd 555555', le programme interprétera la commande comme si vous aviez entré 'd 5555' (voir Figure 1).

- Si vous essayez d'accéder à un offset en dehors de la boîte aux lettres, vous visualiserez les 128 derniers octets de celle-ci (voir Figure 2).

- Si vous entrez par exemple 'd 7f50', vous listerez les 128 octets de 0x7f50 à 0x7fcf. Ensuite, si vous tapez 'd', vous visualiserez les 48 octets restant de 0x7fd0 à 0x7fff, comme le montre la Figure 3.

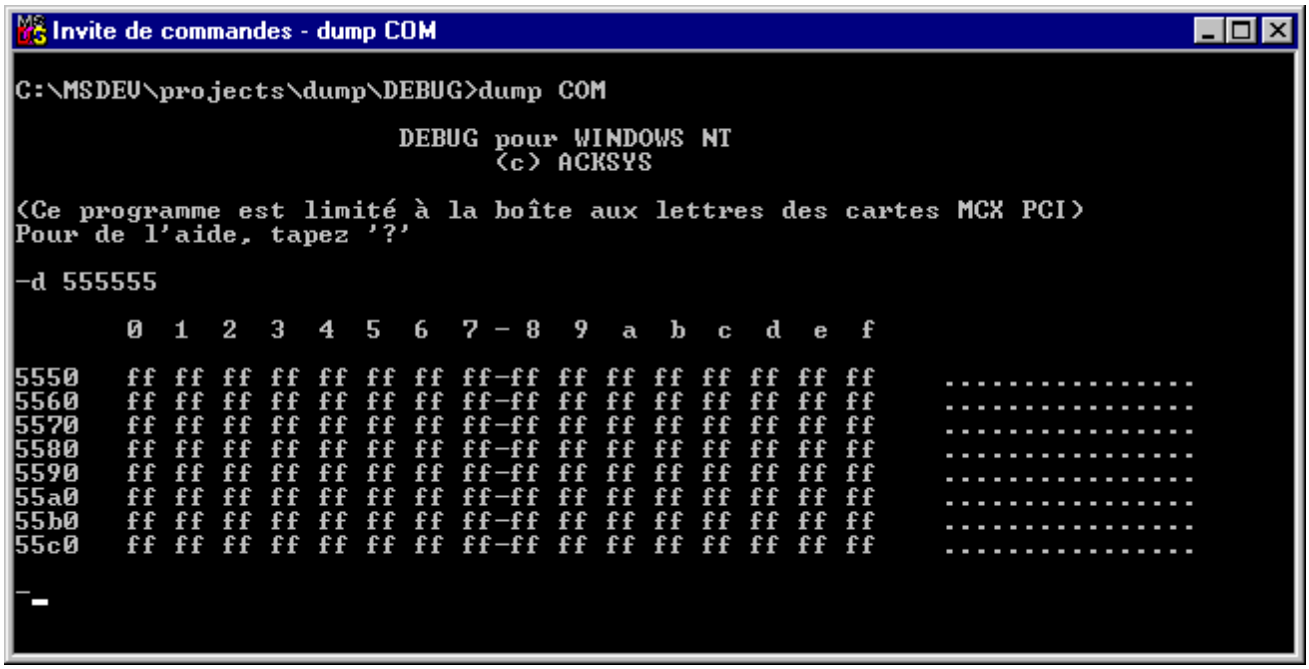


Figure 1 : un dump "fantaisiste"

Figure 2 : un dump trop loin

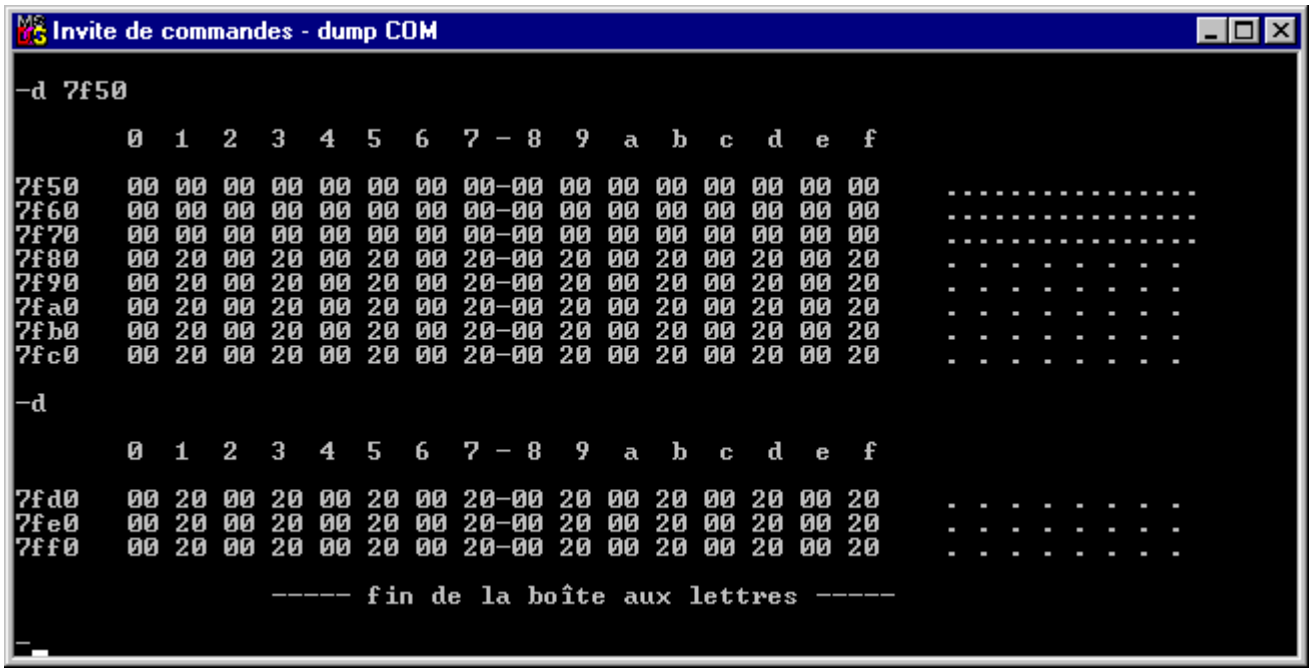


Figure 3 : un dump limite

INPUT : acquisition d'un octet depuis le port d'E/S

Ex : -i [add io]

OUTPUT : envoi d'une valeur sur le port d'E/S

Ex : -o [add io] [valeur]

WRITE : écriture dans la boîte aux lettres de la carte

Il y a deux manières d'écrire dans la boîte aux lettres :

- w + offset + texte entre guillemets (Figure 4)

- w + offset + valeurs en hexadécimal (Figure 5)

Figure 4 : écriture d'un texte

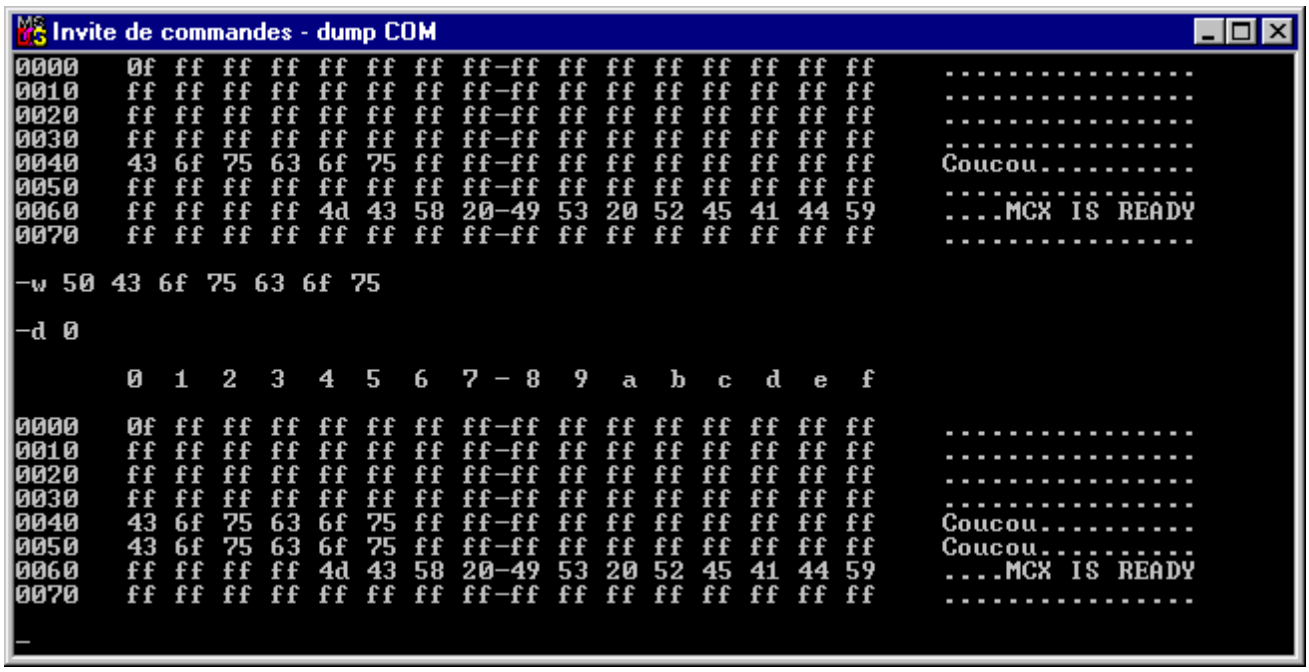


Figure 5 : écriture directe de valeurs en hexadécimal

GLOSSAIRE

API

Application Programming Interface. L'ensemble des spécifications qui permettent à un programme d'application d'utiliser un sous-système, dans notre cas, MCXDOS ou un pilote de périphérique.

Built-in firmware

Logiciel installé dans une EPROM sur la carte, il peut être de trois types : « Logiciel de base », « logiciel multiprotocole » ou une application spécifique adaptée à un besoin particulier.

Canal, voie, port

Ensemble des éléments permettant le transfert d'informations sur un des connecteurs d'une carte multivoies.

Driver, pilote de périphérique, programme de commande

logiciel fourni par ACKSYS, intégré au système d'exploitation, qui permet de programmer et d'utiliser les cartes multivoies, indépendamment des détails matériels (adresses physiques, algorithme de pilotage des cartes...).

Logiciel de Base :

installé en standard dans une EPROM sur la carte, il permet d'utiliser les canaux des cartes avec mezzanines Lite/S, PCB/S, MCXBP et MCXBPMR uniquement, en mode de transmission asynchrone uniquement. Il est compatible avec le Logiciel de Base des cartes MCC.

Logiciel Multiprotocole :

installé en surplus du Logiciel de Base si vous avez commandé l'option MCX-MULTIPROTOCOLE, et fourni en standard sur les carte avec mezzanines PCB/570 ou Lite/570, il permet d'utiliser les canaux dans toute une gamme de formats de transmission synchrones et asynchrones. Il est partiellement compatible avec le Logiciel de Base.

MCC

l'ancêtre de la carte MCX, supportant 8 ou 16 voies asynchrones. Pour assurer la pérennité des applications développées par ses clients, ACKSYS fournit sur les cartes de la gamme MCX un protocole de dialogue compatible avec celui des cartes MCC : le Logiciel de Base.

UART

(Universal Asynchronous Receiver/Transmitter) Composant d'émission/réception série asynchrone. Les plus connus sont les composants 8250, 16550...

USART

(Universal Synchronous/Asynchronous Receiver/Transmitter) Composant d'émission/ réception série asynchrone ou synchrone. Les cartes MCX utilisent des composants SCC 85C30, des HD64570.