



Licence pour le logiciel ADAS Electronique :

- Ce logiciel est employé pour configurer et utiliser les ressources de la carte conformément à la documentation technique du produit.
- Grâce à ce produit, nous gérons tous les particularismes inhérents aux systèmes électroniques et les pièges à éviter, ainsi vous pourrez vous consacrer directement à votre projet (application).
- La copie de ce logiciel est interdite.
- Désassembler, décompiler ou décoder ce logiciel est interdit.
- L'utilisateur doit acquérir une licence logicielle pour l'utilisation du driver de la carte.
- Ce logiciel fonctionne sous Windows 2000 et Windows XP et est multi carte. Il permet de gérer jusqu'à 10 cartes dans un même système.

NOTES:

DRIVER-2K-XP-PCI 160

SOMMAIRE

Chapitre A	Installation du matériel	4
Chapitre B	Installation du logiciel.....	5
Chapitre C	Guide de survie.....	11
Chapitre D	Fonctions logicielles	12
	IOCTL_PCI 160_MAP_MEMORY_RAM_CFG_PCI	14
	IOCTL_PCI 160_MAP_MEMORY_RAM_CFG_160	15
	IOCTL_PCI 160_MAP_MEMORY_RAM_MESURE.....	16
	IOCTL_PCI 160_UNMAP_MEMORY.....	17
	IOCTL_PCI 160_MAP_MEMORY_MASTER	18
	IOCTL_PCI 160_READ_MASTER_MEMORY.....	19
	IOCTL_PCI 160_MASTER_NEW_BLOCK	21
	IOCTL_PCI 160_START	22
	IOCTL_PCI 160_STOP	23
	IOCTL_PCI 160_RESET_AMCC_INIT.....	24
	IOCTL_PCI 160_INTERRUPTON.....	25
	IOCTL_PCI 160_NB_DEVICE.....	26
	IOCTL_PCI 160_NUM_VERSION.....	27
TESTS.EXE	28
Annexe	29

Chapitre A Installation du matériel



- ✚ L'ordinateur doit être hors tension.
- ✚ Déchargez-vous de votre potentiel électrostatique éventuel en touchant une surface métallique.
- ✚ Installez les cartes en prenant soin de les configurer suivant les spécifications techniques et vos propres besoins.

Rappelons que nos cartes **PCI** sont « Plug and Play » et vous dispensent de configurer l'interface avec l'ordinateur. (Voir documentation technique de chaque carte et le chapitre annexe).

- ✚ Mettez l'ordinateur sous tension.

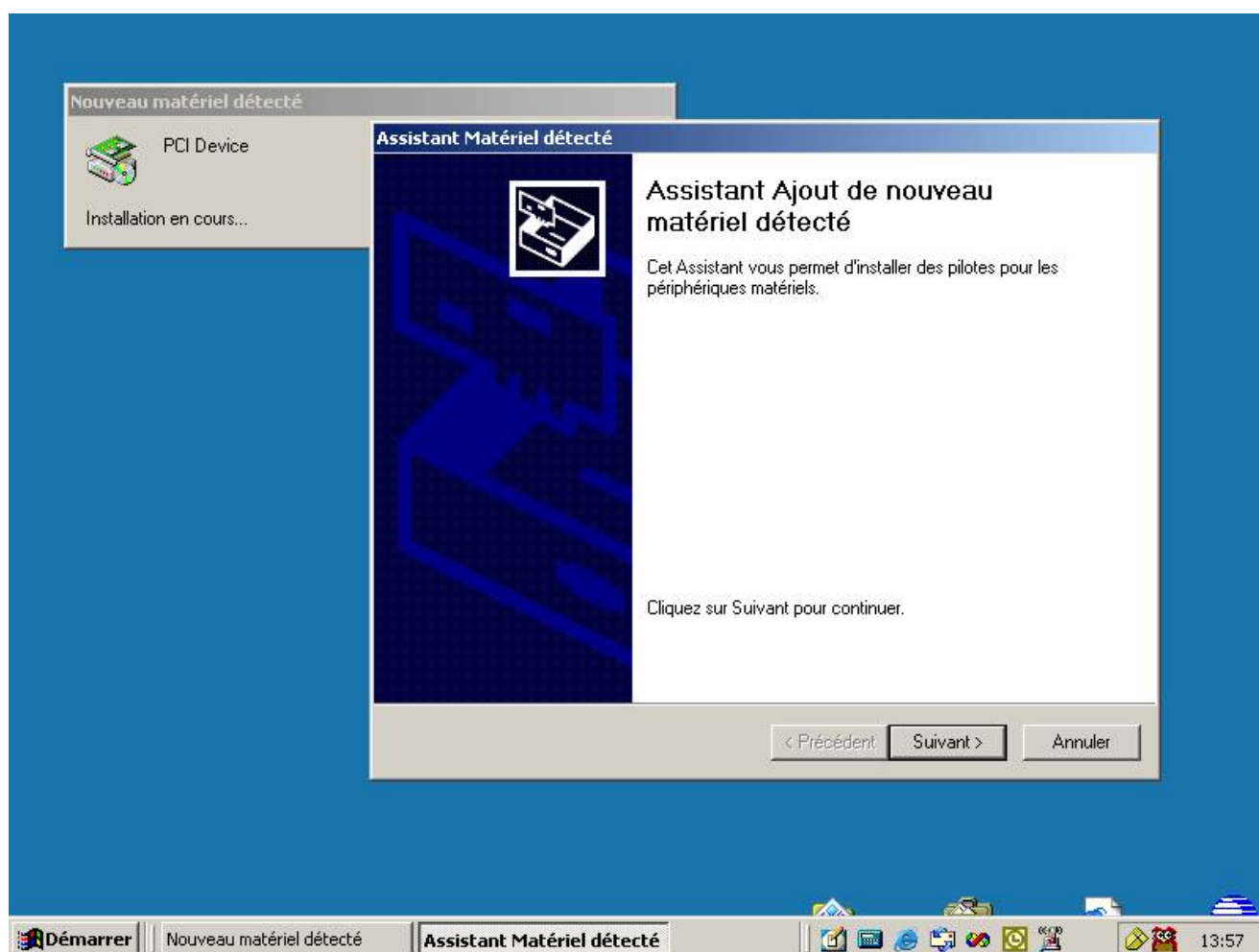


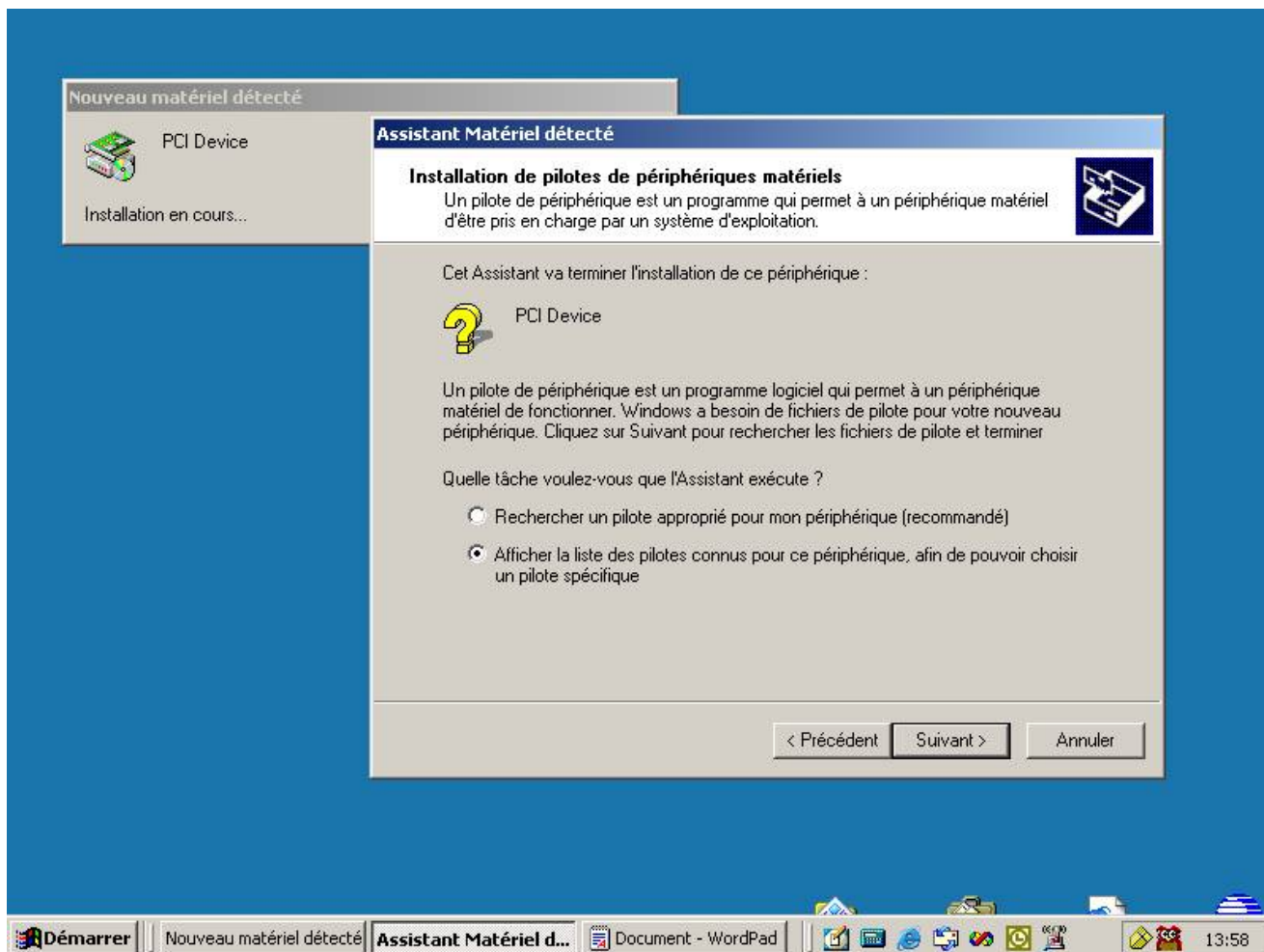
>>>Nota :

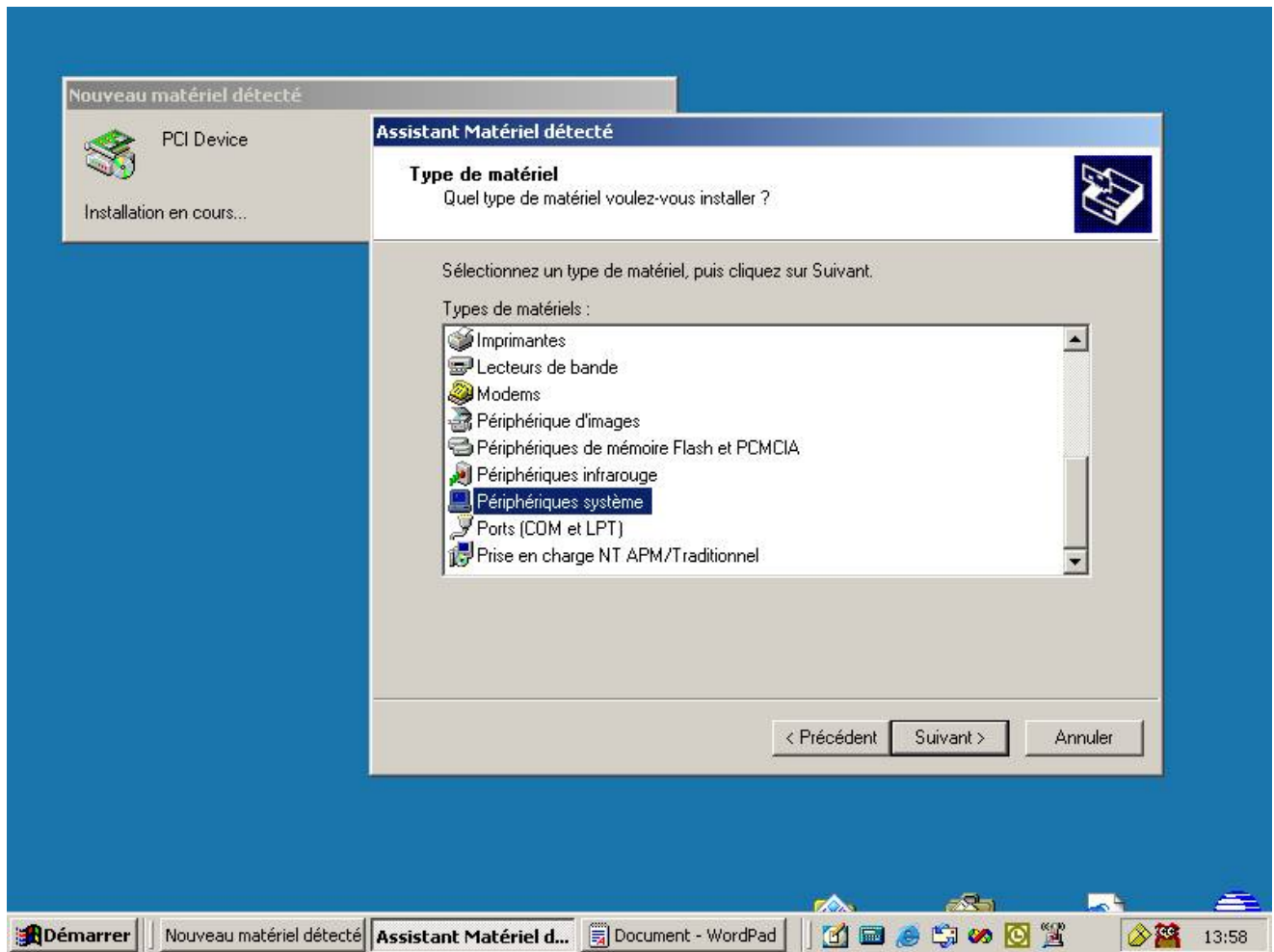
Par souci de simplification, et de bon déroulement de la procédure d'installation, nous vous conseillons de ne pas changer les autres cartes d'emplacement lors de l'ajout de la carte **PCI 160**.

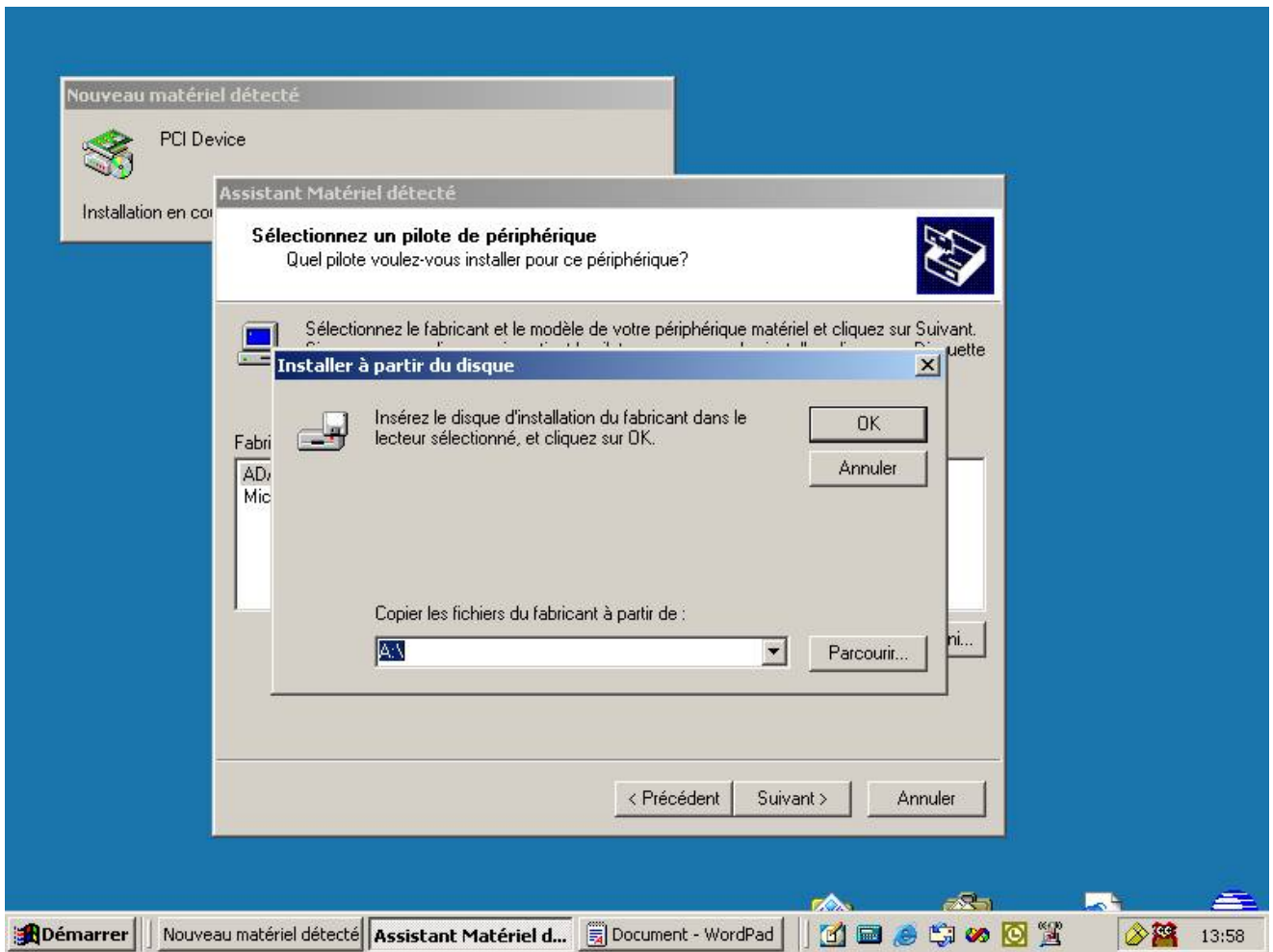
Chapitre B Installation du logiciel

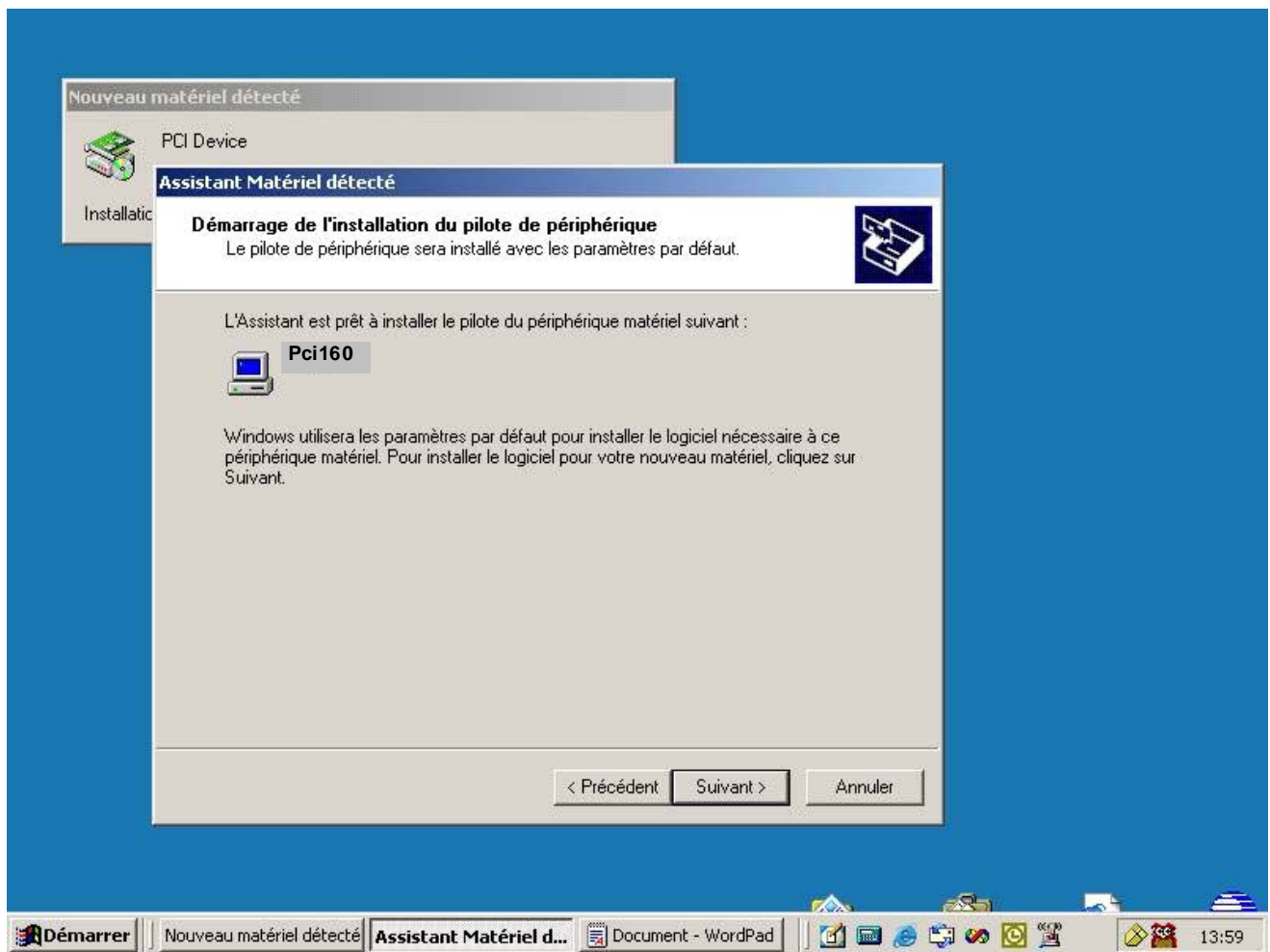
- ✚ Une fois que l'OS est booté son utilitaire va détecter le nouveau Device.
- ✚ A ce moment là, suivre la procédure indiquée à l'écran pour ajouter le driver, c'est-à-dire :

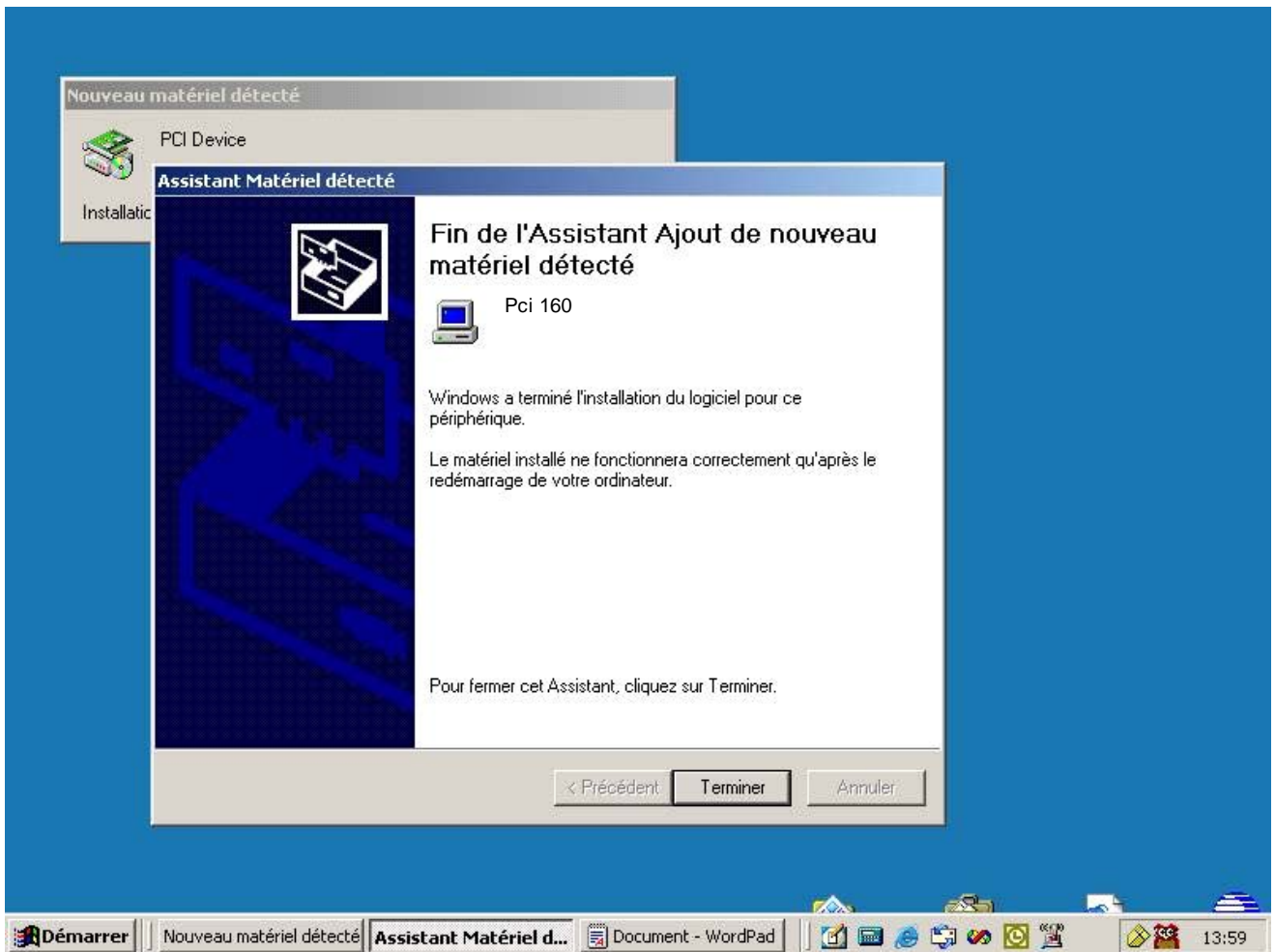












⇒ Redémarrez l'ordinateur :

Maintenant l'ordinateur a identifié le nouveau matériel et il est prêt à l'emploi.

Le driver tourne au niveau Kernel.

Chaque ressource de la carte peut être sollicitée au niveau « user » en faisant appel aux différents IOControl que nous allons expliquer dans le chapitre suivant.

Nous vous indiquons que les anciens logiciels applicatifs (mode user) sont compatibles avec ce nouveau driver.

Une simple re-compilation est nécessaire.

Avec une carte **PCI**, et les OS Windows 2000 ou Windows XP vous ne devez pas rencontrer de problèmes.

Le « Plug and Play » est total aussi bien Hardware que Software.

>>>Rappels :

Windows 2000 ou Windows XP exécute les applications dans le mode user.

Uniquement les drivers mode Kernel peuvent tourner en mode Privilégié.

C'est le mode qui permet d'accéder aux périphériques.

Dans ce contexte, pour gérer au mieux les périphériques, le programmeur utilise les IOCTLs pour communiquer directement avec le driver mode Kernel.

Directement dans une application ou par l'intermédiaire d'une DLL.

Nous vous conseillons de suivre l'exemple d'application fourni.

Le programme exécutable est nommé tests.exe

Ce programme reprend principalement les IOCTLs du driver.

IOCTLs

La fonction **DeviceIoControl** envoie directement au driver un code de contrôle. Nous l'utiliserons pour envoyer tous les codes de contrôle supportés par le driver de la carte.

BOOL DeviceIoControl(

HANDLE *hDevice*, // handle to device of interest

DWORD *dwIoControlCode*, // control code of operation to perform

LPVOID *lpInBuffer*, // pointer to buffer to supply input data

DWORD *nInBufferSize*, // size of input buffer

LPVOID *lpOutBuffer*, // pointer to buffer to receive output data

DWORD *nOutBufferSize*, // size of output buffer

LPDWORD *lpBytesReturned*, // pointer to variable to receive output byte count

LPOVERLAPPED *lpOverlapped* // pointer to overlapped structure for asynchronous operation

);

IOCTL	Description	Elements envoyés	Elements Recus
IOCTL_PCI160_MAP_MEMORY_RAM_CFG_PCI	map la mémoire de configuration PCI	NULL	Pointeur sur la mémoire CFG PCI
IOCTL_PCI 160_MAP_MEMORY_RAM_CFG_160	map la mémoire de configuration 160	NULL	Pointeur sur la mémoire CFG 160
IOCTL_PCI 160_MAP_MEMORY_RAM_MESURE	map la mémoire mesure	NULL	Pointeur sur la mémoire mesure
IOCTL_PCI 160_UNMAP_MEMORY	Unmap de la mémoire réservée	Pointeur sur la mémoire à libérer	NULL
IOCTL_PCI 160_MAP_MEMORY_MASTER	Map la mémoire que le PCI Master peut utiliser	NULL	Pointeur sur la mémoire Master
IOCTL_PCI 160_READ_MASTER_MEMORY	Lecture de x Dword de la mémoire Master	- les index par rapport à la base - x Dword	- les valeurs des index - le nombre d'index effectifs
IOCTL_PCI 160_MASTER_NEW_BLOCK	Requête pour un nouveau bloc	- Master write transfer count	NULL
IOCTL_PCI 160_START	Commande START de la carte	NULL	NULL
IOCTL_PCI 160_STOP	Commande STOP de la carte	NULL	NULL
IOCTL_PCI160_RESET_AMCC_INIT	Reset de la carte et repositionnement du pointeur Master	NULL	NULL
IOCTL_PCI 160_INTERRUPT	Rapport des interruptions	NULL	1 valeurs 32 bits : - InterruptCount;
IOCTL_PCI 160_NB_DEVICE	Retourne le nombre de PCI 160 détectées	NULL	Pointeur sur le nombre de carte
IOCTL_PCI 160_NUM_VERSION	Retourne le numéro de version du driver	NULL	Pointeur sur la version

IOCTL_PCI_160_MAP_MEMORY_RAM_CFG_PCI

map l'espace des registres opérationnel **PCI** dans l'espace user.

Cette opération est utilisée à l'initialisation de l'application.

Ce code permet d'accéder à cette mémoire grâce au pointeur retourné.

```
dwIoControlCode = IOCTL_PCI_160_MAP_MEMORY_RAM_CFG_PCI;
    // operation code
lpInBuffer = NULL;    // address of input buffer; not used; must be NULL
nInBufferSize = 0;    // size of input buffer; not used; must be zero
lpOutBuffer = & pMemCfgPCI;    // address of output buffer;
nOutBufferSize = sizeof(PVOID); // size of output buffer;
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to .n input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by lpInBuffer.

Not used with this operation. Set to zero.

lpOutBuffer

Points to PCI 160 Operation Register.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by lpOutBuffer.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into lpOutBuffer.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

If the operation fails, **DeviceIoControl** returns FALSE. (May be insufficient resource)

IOCTL_PCI_160_MAP_MEMORY_RAM_CFG_160

map l'espace utilisateur de la carte dans l'espace user.

Cette opération est utilisée à l'initialisation de l'application.

Ce code permet d'accéder à cette mémoire grâce au pointeur retourné.

```
dwIoControlCode = IOCTL_PCI_160_MAP_MEMORY_RAM_CFG_160;
    // operation code
lpInBuffer = NULL;    // address of input buffer; not used; must be NULL
nInBufferSize = 0;    // size of input buffer; not used; must be zero
lpOutBuffer = & pMemIO160;    // address of output buffer;
nOutBufferSize = sizeof(PVOID); // size of output buffer;
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to PCI 160 User Register.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data

stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

If the operation fails, **DeviceIoControl** returns FALSE. (may be insufficient resource)

IOCTL_PCI 160_MAP_MEMORY_RAM_MESURE

map l'espace mémoire de la carte dans l'espace user (fenêtre de 4 MO).
Cette opération est utilisée à l'initialisation de l'application.

Ce code permet d'accéder à cette mémoire grâce au pointeur retourné.

```
dwIoControlCode = IOCTL_PCI 160_MAP_MEMORY_RAM_MESURE;  
// operation code  
lpInBuffer = NULL; // address of input buffer; not used; must be NULL  
nInBufferSize = 0; // size of input buffer; not used; must be zero  
lpOutBuffer = & pMemRamMesure; // address of output buffer;  
nOutBufferSize = sizeof(PVOID); // size of output buffer;  
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to PCI 160 Measure Memory.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE. (may be insufficient resource)

IOCTL_PCI 160_UNMAP_MEMORY

Libère la mémoire mappée, cette opération est utilisée généralement à la sortie de l'application.

Cette fonction est la fonction inverse des fonctions suivantes :

```
IOCTL_PCI 160_MAP_MEMORY_RAM_CFG_PCI  
IOCTL_PCI 160_MAP_MEMORY_RAM_CFG_160  
IOCTL_PCI 160_MAP_MEMORY_RAM_MESURE
```

```
dwIoControlCode = IOCTL_PCI 160_UNMAP_MEMORY;  
    // operation code  
lpInBuffer = &pMemRam; // address of input buffer;  
nInBufferSize = sizeof(PVOID); // size of input buffer;  
lpOutBuffer = NULL; // address of output buffer; not used; must be NULL  
nOutBufferSize = 0; // size of output buffer; not used; must be zero  
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. (&pMemCfgPCI, &pMemCfg160, &pMemRamMeasure)

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.
Not used with this operation. Set to zero.

lpOutBuffer

Points to the **PCI 160** Memory of Measure buffer .

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a DWORD that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE.

IOCTL_PCI 160_MAP_MEMORY_MASTER

map la mémoire pointée par les accès Bus Master de la carte **PCI 160**, cette opération est utilisée à l'initialisation de l'application,

Elle permet d'accéder à cette mémoire grâce au pointeur retourné.

```
dwIoControlCode = IOCTL_PCI 160_MAP_MEMORY_MASTER;
// operation code
lpInBuffer = NULL; // address of input buffer; not used; must be NULL
nInBufferSize = 0; // size of input buffer; not used; must be zero
lpOutBuffer = &pMaster; // address of output buffer;
nOutBufferSize = sizeof(PVOID); // size of output buffer;
lpBytesReturned = &cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to the **PCI 160** MemoryBus Master buffer .

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a DWORD that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE.

IOCTL_PCI_160_READ_MASTER_MEMORY

Permet de lire le contenu de la mémoire pointée par les accès Bus Master de la carte **PCI 160**.

```
dwIoControlCode = IOCTL_PCI_160_READ_MASTER_MEMORY;  
// operation code  
lpInBuffer = BufferR; // address of input buffer;  
nInBufferSize = 4 * nbByteTransfer; // size of input buffer; 512 Koctets max  
lpOutBuffer = BufferW; // address of output buffer;  
nOutBufferSize = 4 * 131072 ; // size of output buffer; 512 Koctets max  
lpBytesReturned = &NumberOfBytesRead; // address of number of  
bytes read
```

Parameters

lpInBuffer

Points to an input buffer. You send an index for each value return.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Points to an output buffer.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE.

>>>Examples

You want to read address 0 2 4 7 of the RAM Master you can do :

...

```
BufferRW[0] = 0x0 ;
```

```
BufferRW[1] = 0x2 ;
```

```
BufferRW[2] = 0x4 ;
```

```
BufferRW[3] = 0x7 ;
```

```
DeviceIoControl
```

```
(  
    hDevice,  
    IOCTL_PCI_160_READ_MASTER_MEMORY,  
    BufferRW,  
    4*4,                // number of byte transfer  
    BufferRW,  
    4*131072,           //=512 Koctets max  
    &NumberOfBytesRead, // address of number of bytes read  
    NULL                // address of structure for data  
);
```

....

IOCTL_PCI 160_MASTER_NEW_BLOCK

Permet de relancer un nouveau transfert de bloc, de la carte **PCI 160** vers la mémoire UC.

On paramètre le nombre d'octets à transférer.

Cette opération s'effectue uniquement en mode Master.

```
dwIoControlCode = IOCTL_PCI 160_MASTER_NEW_BLOCK;
    // operation code
lpInBuffer = BufferR; // address of input buffer;
Represent the Master Write Transfer Count.
nInBufferSize = 4 * 1; // Size of the Transfer Count.
lpOutBuffer = NULL; // address of output buffer; Not used with this
operation. Set to NULL.
nOutBufferSize = 0 ; // size of output buffer; Not used with this operation.
Set to zero.
lpBytesReturned = &cbReturned; // address of number of bytes read
```

Parameters

lpInBuffer

Points to an input buffer. Represent the Master write Transfer Count.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer. Not used with this operation. Set to NULL.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

Not used with this operation. Set to zero.

lpBytesReturned

Points to a DWORD that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE.

IOCTL_PCI 160_START

Lance la commande START de la carte.

```
dwIoControlCode = IOCTL_PCI 160_START;  
    // operation code  
lpInBuffer = NULL;    // Not used with this operation. Set to NULL.  
nInBufferSize = 0;    // Not used with this operation. Set to zero.  
lpOutBuffer = NULL; // Not used with this operation. Set to NULL.  
nOutBufferSize = 0 ; // Not used with this operation. Set to zero.  
lpBytesReturned = &cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer. Not used with this operation. Set to NULL.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

Not used with this operation. Set to zero.

lpBytesReturned

Points to a DWORD that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

If the operation fails, **DeviceIoControl** returns FALSE.

IOCTL_PCI 160_STOP

Lance la commande STOP de la carte.

```
dwIoControlCode = IOCTL_PCI 160_STOP;  
    // operation code  
lpInBuffer = NULL;    // Not used with this operation. Set to NULL.  
nInBufferSize = 0;    // Not used with this operation. Set to zero.  
lpOutBuffer = NULL; // Not used with this operation. Set to NULL.  
nOutBufferSize = 0 ; // Not used with this operation. Set to zero.  
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer. Not used with this operation. Set to NULL.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

Not used with this operation. Set to zero.

lpBytesReturned

Points to a DWORD that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

If the operation fails, **DeviceIoControl** returns FALSE.

IOCTL_PCI 160_RESET_AMCC_INIT

Cette commande réinitialise la carte **PCI 160** par un reset de l'AMCC.

```
dwIoControlCode = IOCTL_PCI 160_RESET_AMCC_INIT;  
// operation code  
lpInBuffer = NULL; // address of input buffer; not used; must be NULL  
nInBufferSize = 0; // size of input buffer; not used; must be zero  
lpOutBuffer = NULL; // address of output buffer; not used; must be  
NULL  
nOutBufferSize = 0; // size of output buffer; not used; must be zero  
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero

lpOutBuffer

Points to an output buffer. Not used with this operation. Set to NULL.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

Not used with this operation. Set to zero.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

If the operation fails, **DeviceloControl** returns FALSE.

IOCTL_PCI 160_INTERRUPTION

Permet d'obtenir un rapport sur les interruptions.

Ce rapport est remis à zéro une fois l'opération terminée.

Les valeurs de retour sont :

BufferRW [0] =InterruptCount; nombre total d'interruptions reçues depuis le dernier rapport

```
dwIoControlCode = IOCTL_PCI 160_INTERRUPTION;
```

```
// operation code
```

```
lpInBuffer = NULL; // address of input buffer; Not used with this operation. Set to NULL.
```

```
nInBufferSize = 0; // size of input buffer; Not used with this operation. Set to zero.
```

```
lpOutBuffer = &BufferRW; // address of output buffer
```

```
nOutBufferSize = 4*1 ; // size of output buffer; 1 DWORDs return
```

```
lpBytesReturned = &cbReturned; // address of number of bytes read
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

The driver return 1 DWORDs

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

If the operation succeeds and the device **PCI 160** is accessible, **DeviceloControl** returns TRUE.

The **DeviceloControl** couldn't return FALSE.

IOCTL_PCI 160_NB_DEVICE

Cette commande retourne le nombre de cartes **PCI 160** détectées par le Driver.

Ce numéro comporte 4 octets.

Le driver supporte au maximum 10 cartes numérotées de 0 à 9.

```
dwIoControlCode = IOCTL_PCI 160_NB_DEVICE ;  
// operation code  
lpInBuffer = NULL; // Not used with this operation. Set to NULL.  
nInBufferSize = 0; // Not used with this operation. Set to zero.  
lpOutBuffer = &BufferRW; // address of output buffer  
nOutBufferSize = 4*1 ; // size of output buffer; 1 DWORDs return  
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

The operation always succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

IOCTL_PCI 160_NUM_VERSION

Cette commande retourne le numéro de version du driver de la carte.

Ce numéro comporte 4 octets.

Il est codé en ASCII suivant l'exemple :

BufferRW= 0x56313130; soit V 1 1 0

```
dwIoControlCode = IOCTL_PCI 160_NUM_VERSION;
// operation code
lpInBuffer = NULL; // Not used with this operation. Set to NULL.
nInBufferSize = 0; // Not used with this operation. Set to zero.
lpOutBuffer = &BufferRW; // address of output buffer
nOutBufferSize = 4*1 ; // size of output buffer; 1 DWORDs return
lpBytesReturned =&cbReturned; // address of actual bytes of output
```

Parameters

lpInBuffer

Points to an input buffer. Not used with this operation. Set to NULL.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

Not used with this operation. Set to zero.

lpOutBuffer

Points to an output buffer.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Points to a **DWORD** that receives the actual size, in bytes, of the data stored into *lpOutBuffer*.

Return Values

The operation always succeeds and the device **PCI 160** is accessible, **DeviceIoControl** returns TRUE.

TESTS.EXE

Ce programme est fourni sur la disquette.

Il reprend les codes IOCTL dans des exemples simples, et vous permettra avec son code source de démarrer votre projet rapidement.

Le programme source peut être directement compilé et modifié avec Microsoft

Visual C++ 6.0. ou version ultérieure.

Annexe

PC INTERRUPT LEVELS

IRQ0	Timer Tick
IRQ1	Keyboard Controller
IRQ2	Cascade interrupt
IRQ3	COM2, COM4
IRQ4	COM1, COM3
IRQ5	Audio
IRQ6	Diskette
IRQ7	LPT1, LPT3
IRQ8	Real time clock
IRQ9	(PCI device) default video
IRQ10	(PCI device) default SCSI
IRQ11	(PCI device) default audio
IRQ12	Mouse interrupt
IRQ13	Math co-processor
IRQ14	IDE primary
IRQ15	IDE secondary



Ressource éventuellement exploitable par une carte PCI