Vpcic32D.VXD,

PCICC32.SYS

&

PCICC32_NI

Windows 95/98 driver, Windows-NT driver for PCI-to-CAMAC Interface

User's Manual

General Remarks

The only purpose of this manual is a description of the product. It must not be interpreted a declaration of conformity for this product including the product and software.

W-Ie-Ne-R revises this product and manual without notice. Differences of the description in manual and product are possible.

W-Ie-Ne-R excludes completely any liability for loss of profits, loss of business, loss of use or data, interrupt of business, or for indirect, special incidental, or consequential damages of any kind, even if **W-Ie-Ne-R** has been advises of the possibility of such damages arising from any defect or error in this manual or product.

Any use of the product which may influence health of human beings requires the express written permission of **W-Ie-Ne-R**.

Products mentioned in this manual are mentioned for identification purposes only. Product names appearing in this manual may or may not be registered trademarks or copyrights of their respective companies.

No part of this product, including the product and the software may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means with the express written permission of **W-Ie-Ne-R**.

VPCICC32D is designed by ARW Elektronik, Germany

Table of contents

1. GENERAL DESCRIPTION

Vpcic32D.VXD provides an easy access to the CC32 CAMAC crate controller for Windows95[1] and 98 users. The same does PCICC32.SYS for Windows-NT.

A separate driver is provided for use with LINUX[2]. The properties of this driver are not explained here.

It is easy to use the drivers for your own CAMAC application. The driver is independent from the chosen programming language since WIN32 standard I/O functions are used for communication to the drivers.

There are small differences in the application interface. These differences are mentioned when appropriate (noted WIN95/98 or WIN-NT)

WIN95/98: CAMAC access is performed via an interface window of an area of virtual memory, which is defined by the driver. For user applications this window looks like normal memory. Read and write operations to the CAMAC controller and / or bus are converted into simple read and write operations whereas the destination (address) corresponds to the NAF code of the operation.

WIN-NT: CAMAC access is performed with normal file read and write operations. A "file offset" calculated from the NAF code is used to reference the access location. Since a file read/write does not know if the element to read or write should be 16-bit or 32-bit in size, this property has to be set before the read/write operation. This setup is done with a special I/O-call to the driver.

The access to the drivers are not limited to one process. Multiple processes can use the drivers. Even one driver supports multiple CAMAC interfaces.

Two different hardware interrupts (LAM and time out) are handled by the interface. The driver provides several services to operate these interrupts.

[1] Windows95 and Windows98 are trademarks of the Microsoft Corporation.

[2]LINUX is a trademark of Linus Torvalds.

2. Installation of the WIN95/98 Driver

The PCI-CAMAC system consisting of PCIADA and CC32 has been delivered with software drivers and applications on a CD-ROM. This CD-ROM includes in the PCICAMAC directory the following files:

Subdirectory \WIN95\DRIVER:

Vpcic32D.vxd	-	Win95/98 driver
pciC32.inf	-	'INF' file for installation

Subdirectory \WIN95\DRIVER\SOURCE:

Vpcic32D.h	-	driver interface header-file for applications, vpcic32d.vxd driver source

Note: The driver only works with Windows 95 / 98 in 32 bit mode including console applications. Only real 32 bit applications can use the driver. It does not work with MS-DOS[2] or WINDOWS 3.11 programs.

Note: Windows NT systems require a different driver however, the driver interface will be partially compatible.

Be sure that PCIADA card of the PCI-CAMAC system is installed in your PC. Please refer to the PCI-CAMAC user manual to insert the card.

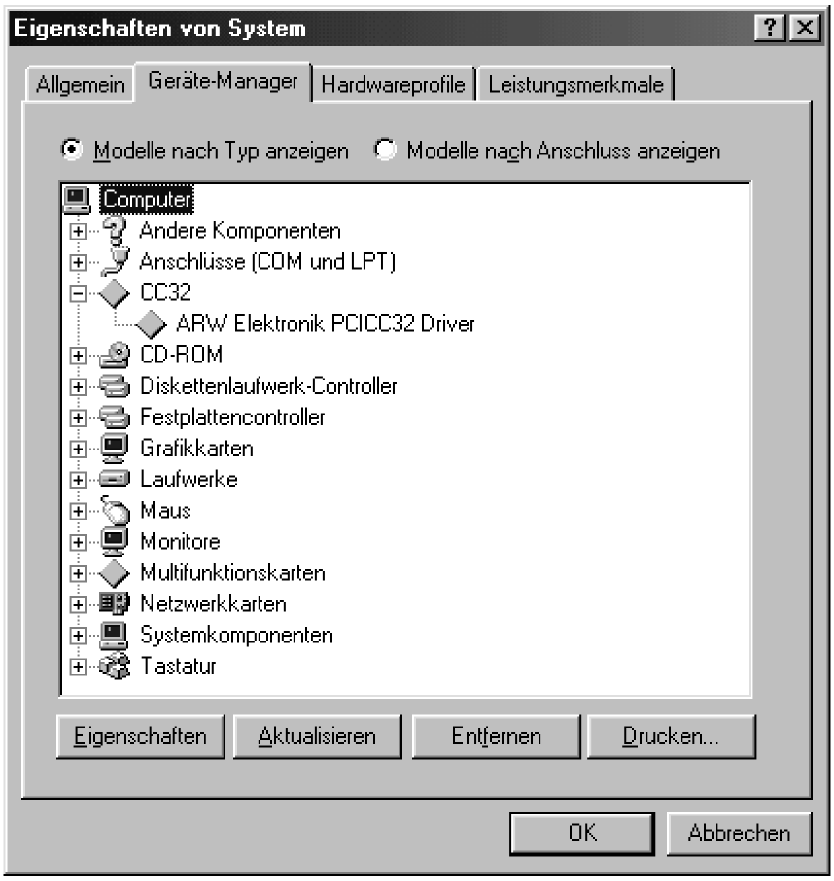
After switching on the computer the MS-Windows operating system recognizes automatically the new hardware in your system and asks for the driver and "INF" file. Insert the supplied CD into your CD-ROM drive and enter the driver's path. If your CD-ROM drive is the "D:drive" type:

D:\ PCICAMAC\WIN95\DRIVER\

In the following installation process the driver is copied to the Windows system directory "WINDOWS\SYSTEM\Vpcic32D.VXD" and the interface is entered into the Windows Registry (under HKEY_LOCAL_MACHINE\ENUM\PCI\VEN10B5&DEV2258....).

To check the driver installation and settings you will find the interface at the systems settings of the control panel (start / settings / control panel / system / device manager) where interrupt and I/O settings can be verified**.**

[2] MS-DOS and Windows 3.11 are trademarks of the Microsoft Corporation.


Eigenschaften von System

Allgemein | Geräte-Manager | Hardwareprofile | Leistungsmerkmale

(●) Modelle nach Typ anzeigen ( ) Modelle nach Anschluss anzeigen

- Computer
  - Andere Komponenten
  - Anschlüsse (COM und LPT)
  - CC32
    - ARW Elektronik PCICC32 Driver
  - CD-ROM
  - Diskettenlaufwerk-Controller
  - Festplattencontroller
  - Grafikkarten
  - Laufwerke
  - Maus
  - Monitore
  - Multifunktionskarten
  - Netzwerkkarten
  - Systemkomponenten
  - Tastatur

Picture 1. Example of settings / control panel / system / device manager

Note: The PCIADA interface card can work with both VME (VMEMM) and CAMAC (CC32) systems. Normally the PCIADA is configured for use with CC32. If you have to change the PCIADA configuration for any reason please look for the description of the program PLXeep.exe.

3. Installation of the WIN-NT Driver

The PCI-CAMAC system consisting of PCIADA and CC32 has been delivered with software drivers and applications on a CD-ROM. This CD-ROM includes in the PCICAMAC directory the following files:

Subdirectory \WINNT\DRIVER:

Pcicc32.sys	-	Windows-NT kernel driver
install.exe	-	installation program to install and start the driver

Subdirectory \WINNT\DRIVER\SOURCE:

Pcicc32.h	-	driver interface header-file for applications, pcicc32.sys driver source

Note: The driver only works with Windows-NT including console applications running on Windows-NT.

Note: Windows-95/98 systems require a different driver however, the driver interface will be partially compatible.

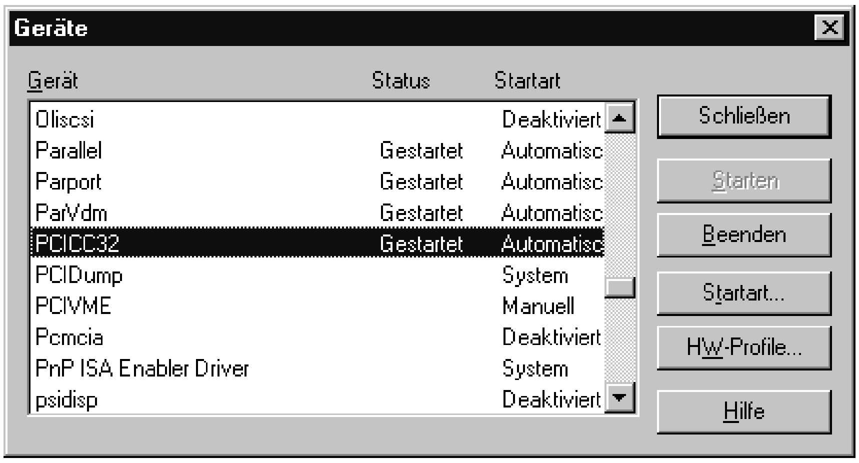
Be sure that PCIADA card of the PCI-CAMAC system is installed in your PC. Please refer to the PCI-CAMAC user manual to insert the card.

Now start your computer running Windows-NT and login as administrator. Open a dos box and change to the CDROM-path with the driver and the install program. Then call the install program with following parameters. If your CD-ROM drive is the "D:drive" type:

```
D:
cd \ PCICAMAC\WINNT\DRIVER
install –d=pcicc32.sys –v
```

In the following installation process the driver is copied to the Windows driver directory "WINNT\SYSTEM32\DRIVERS\pcicc32.sys" and some information is added to the registry. The driver is started immediately.

To check the driver installation and settings you will find the interface at the systems settings/devices with the entry "PCICC32 started automatic".



Gerät	Status	Startart
Oliscsi		Deaktiviert
Parallel	Gestartet	Automatisc
Parport	Gestartet	Automatisc
ParVdm	Gestartet	Automatisc
PCICC32	Gestartet	Automatisc
PCIDump		System
PCIVME		Manuell
Pcmcia		Deaktiviert
PnP ISA Enabler Driver		System
psidisp		Deaktiviert

Picture 2. Example of System Settings / Devices

Note: You can remove the driver with the install program too. Please invoke "install -?" for more information.

Note: The PCIADA interface card can work with both VME (VMEMM) and CAMAC (CC32) systems. Normally the PCIADA is configured for use with CC32. If you have to change the PCIADA configuration for any reason please look for the description of the program PLXeep.exe.

4. TEST OF DRIVER INSTALLATION

To test the correct driver installation as well as the hardware operation there is the program "pcicc32_test.exe" which can be found on the on the driver and application CD-ROM. To avoid interference with any other CAMAC hardware please use the program first with **only** the CC32 inserted into the CAMAC crate.

To run the program use either the "RUN" option of the Windows START menu or open the MS-DOS box and start the application with "pcic32_test.exe" after setting the path to the directory on the CD-ROM containing it. . The program can be started with different run-time parameters; the call "pcic32_test -?" prints a short help text including the parameter list.

If working correctly the pcic32_test.exe program performs for about 20 seconds multiple accesses to all CAMAC station numbers (N) with all possible sub-addresses (A) and function codes (F).

Note: The default driver path for WIN95/98 is "\\.\C:\WINDOWS\SYSTEM\Vpcic32D.vxd". If using a different path this has to be considered in the run-time parameters calling pcic32_test.exe.

Note: The default driver path for WIN-NT is "\\.\PCICC32:\CC32_1". The ".._1" pinpoints the CC32 interface configured for module number 1. For further information about the module number see the PCICC32 hardware users manual.

5. DRIVER DESCRIPTION

WIN95/98: The "Vpcic32D.vxd" is a dynamically loadable Driver, i.e. it will not be implemented during the start of the operating system. Every application using the driver has to load it first and has to release it before closing. Up to 16 PCI interface cards (PCIADA) corresponding up to 16 CAMAC controllers (CC32) can be accessed via the driver. They all are distinguished only by the different CC32 module number.

WIN-NT: The driver is loaded during the start of WINDOWS-NT. Every application using the driver has to create a path to the device first and removes it when closing. Up to 16 PCI interface cards (PCIADA) corresponding up to 16 CAMAC controllers (CC32) can be accessed via the driver. They all are distinguished only by a different CC32 path number, e.g. PCICC32:\CC32_1, PCICC32:\CC32_2, ...

The VpciC32D driver establishes a well defined software interface to the PCI-CAMAC hardware. It works in the processor ring-0 (WIN95/98) or as kernel driver (WIN-NT) and is available to all calling applications with equal rights. Thus it can be used simultaneously by multiple applications however, it does not support asynchronous calls.

Access to the driver is managed by Windows Standard I/O functions which are independent from the programming language. Header files for c++ programs are supplied with the interface. They can be easily adapted to other languages.

All driver parameters (addresses) will be taken from the Windows Registry. This simplifies the calls in the user application.

WIN95/98: For a maximum performance the PCI-to-CAMAC access is done using a 32kB memory segment for NAF coding and data transfer. User applications can directly read or write to the CAMAC dataway via this memory segment without further hardware consideration. To do this the application has to request first the window to the CAMAC address space. The driver returns a pointer to it which can be used as the base address of the segment. Within the window the relative address corresponds to the NAF code of the CAMAC call. The windows can be accessed by multiple applications simultaneously without further restrictions.

WIN-NT: For compatibility with standard file access mechanisms all reads and writes to CAMAC are done via standard file-read and file-write calls. A special IO-control call is used to setup the data access width for further reads or writes. With the standard " SetFilePointer(...)" call the next access address calculated from the NAF-code is set. No auto-increment or decrement of access addresses is supported.

5.1. Load Driver

```
        hHandle = CreateFile(PathName,0,0,NULL,0,
                              FILE_FLAG_DELETE_ON_CLOSE,NULL);
```

During this procedure number and IDs of connected CC32 CAMAC controllers is determined.

Note: The pathname differs for WIN95/98 and WIN-NT. The WIN-NT pathname carries the module number to access while the WIN95/98 path opens a more general path to all available CC32 interfaces connected to this computer. This implies for WIN-NT that the "dwInterface" variable used with the input structure of the DevciceIoControl(...) calls is only a idle placeholder. With the WIN95/98 application interface this "dwInterface" variable selects the called CC32 module number.

5.2. Driver access

The driver provides different services which can be called from a user application via the Windows function **`DeviceIoControl()`**:

```
result = DeviceIoControl(  hHandle,          handle to CC32
                           service,          service code called
                           &input,           reference to input structure
                           sizeof(input),    size of input structure
                           &output           reference to output structure
                           sizeof(output),   size of output structure
                             &DIOC_count,          reference of  real size of returned
data
                           NULL);
```

The first parameter hHandle indicates a particular CC32 controller, i.e. the driver itself considers the connection between the CC32 with given CC32 module number and the corresponding PCIADA interface card.

The second parameters describes the service to be called. This can be done by using the pre-defined service names or the corresponding numbers (see Vpcic32d.h for WIN95/98 or pcicc32.h for WIN-NT) as for instance VPCIC32_ATTACH_CC32 defined with 0x00220000. In addition each service is associated with a particular input and output structure. This requires the definition of pointers to the structures as well as to their sizes as shown in the following example.

The call of "DeviceIOControl(..)" returns a result value which indicates in case of "0" the correct operation. If an error occurred the value is different to 0. In this case the Windows function "GetLastError()" will return the error number. All error numbers are defined in the "winerror.h" header file.

WIN95/98: Example for requesting a memory window into the CC32 address space

```
VPCIC32D_DEVICE              sInterface;
VPCIC32D_WINDOW              sWindow;
DWORD                        DIOC_count;
DWORD                           dwResult;
*
*
sInterface.dwInterface = 1;          // selection of  CC32 with Module #1

// request for a memory window into CAMAC CC32  ------------------
dwResult = DeviceIoControl(hHandle, VPCIC32_ATTACH_CC32,
        &sInterface, sizeof(sInterface), &sWindow, sizeof(sWindow),
                &DIOC_count, NULL);

if (!dwResult)
  printf(„Error %d occured\n“, GetLastError());
else
  printf(„I have got a memory window @ 0x%08x\n“, sWindow.pvWindowBase);

*
*
```

The called service is “VPCIC32_ATTACH_CC32“ with a single DWORD input structure sInterface containing the CC32 number. The size of this input structure is “sizeof(sInterface)“.

The call returns the structure “VPCIC32D_WINDOW sWindow“ with size “sizeof(sWindow)“ which contains the base address for the CC32 window.

The variable „DIOC_count“ contains the real size of the returned data from the driver

5.3. Read and Write

This applies only to WIN-NT, with WIN95/98 all reads and writes are done via a direct accessible 32 kbyte memory segment.

First you have to set the file pointer to the address location calculated from the NAF-code. If the address pointer still is set nothing is to do. Then you can read or write a block of data which size is a multiple of the preset data access width (2 or 4 bytes).

```
DWORD ReadCAMAC(HANDLE hHandle,
                    unsigned long adr,
                            void *buffer,
                                    int nBytesToRead)
{
  DWORD bytesRead;

  SetFilePointer(hHandle, adr, NULL, FILE_BEGIN);
  if (!ReadFile(hHandle, buffer, nBytesToRead, &bytesRead, NULL))
          printf("Can't read (0x%08x)\n", GetLastError());

  return bytesRead;
}
```

Note: To improve performance you can open a path to each block-read or block-write NAF-codes, even if they are accessed through the same CC32 interface. Then you have to set the file pointer only once at initial setup of the path.

5.4. calculation of access address from NAF-code

According to the PCICC32 hardware manual the access address is calculated like

```
#define NAF(n, a, f)  ((unsigned long)((n << 10) + (a << 6) + ((f & 0xf) << 2)))
```

It is advisable to use this macro for all calculations.

5.5. Close driver

After using the driver it has to be closed before finishing the application:

```
CloseHandle(hHandle);
```

The driver will be automatically removed from the system when closed by the last application previously using it.

6. VPCIC32D AND PCICC32 SERVICES

Please see for reference the "vpcic32d.h" header file (WIN95/98) or the "pcicc32.h" header file (WIN-NT).

6.1. Attach Window - VPCIC32_ATTACH_CC32 [0x00220000]

This call is not applicable to WIN-NT.

VPCIC32_ATTACH_CC32 generates a memory window into the CC32 CAMAC address space.

Input structure:

```
typedef struct
{
        DWORD dwInterface;        // CC32 module number
} VPCIC32D_DEVICE;
```

Output structure:

```
typedef struct
{
        DWORD dwInterface;        // CC32 module number
        void   *pvWindowBase;          // the base address into the 32 kbyte
Window
} VPCIC32D_WINDOW;
```

6.2. Detach Window - VPCIC32_DETACH_CC32 [0x00220004]

This call is not applicable to WIN-NT.

VPCIC32_DETACH_CC32 deinitializes the PCIADA with the corresponding CC32 controller. The CAMAC windows is released.

Input structure:

```
typedef struct
{
        DWORD dwInterface;        // CC32 module number
} VPCIC32D_DEVICE;
```

Output structure: There is nothing to output. Pointer to structure is NULL, size of structure is 0.

6.3. Get Status - VPCIC32_GET_STATUS or PCICC32_GET_STATUS [0x00220008]

VPCIC32_GET_STATUS (WIN95/98) or PCICC32_GET_STATUS (WIN-NT) allows to obtain the status of the PCIADA interface card (time-out) as well as of the CC32 crate controller (LAM). This call can be used for LAM polling.

Input structure:

```
typedef struct
{
        DWORD dwInterface;      // CC32 module number, idle for WIN-NT
} VPCIC32D_DEVICE;
```

Output structure:

```
typedef struct
{
        DWORD dwInterface;      // CC32 module number, idle for WIN-NT
        WORD  bTimeout;         // PCIADA timeout
        WORD  bInterrupt;       // pending LAM interrupt of CC32
} VPCIC32D_DEVICE; // (WIN95/98), PCICC32_DEVICE (WIN-NT)
```

The returned bTimeout indicates a hardware error or disconnection of the CC32 controller (or CAMAC power off). The output bLAM shows any active pending LAM-interrupt (Look At Me).

Note: The LAM interrupt depends in addition to a LAM request from a CAMAC station on the LAM mask setting of the controller. Please see the PCI-CAMAC manual for reference.

6.4. Clear Status - VPCIC32_CLEAR_STATUS or PCICC32_CLEAR_STATUS [0x0022000C]

VPCIC32_CLEAR_STATUS (WIN95/98) or PCICC32_CLEAR_STATUS (WIN-NT) can be used to clear a pending bTimeout interrupt. The pending bLam has to be cleared at the appropriate CAMAC slave station with corresponding calls from the user.

Input structure:

```
typedef struct
{
        DWORD dwInterface;      // CC32 module number, idle for WIN-NT
} VPCIC32D_DEVICE; //(WIN95/98), PCICC32_DEVICE (WIN-NT)
```

Output structure: There is nothing to output. Pointer to structure is NULL, size of structure is 0.

6.5. SetAccessParameter - VPCIC32_SET_ACCESS_PARA or PCICC32_SET_... [0x00220010]

PCICC32_SET_ACCESS_PARA allows to set the further data access width for this path to 16-bit or 2 byte or 32-bit or 4 byte. With the wBlockTransfer parameter you can enable or disable some features for fast data readout. The UNTIL_NOT_Q enables the next read into the provided buffer until not 'Q' is signaled from the slave device or the buffer is filled. AUTOREAD enables the PCIADA to read one item ahead to increase readout speed through overlapp of PC-read and CAMAC-read.

Please note that the parameter "wAccessType" at WIN95/98 is determined with the access itself. Also the UNTIL_NOT_Q has no function at WIN95/98. Due to the direct access mechanism the access code has to manage this feature itself.

Input structure:
```
typedef struct
{
        ULONG  dwInterface;       //
        USHORT wAccessType;    // set the current access type, see "pcicc32.h" (idle @ WIN95)
        USHORT wBlockTransfer;     // see table
} PCICC32_ACCESS_COMMAND;
```

wBlockTransfer		
UNTIL_NOT_Q	read until not Q	only applicable for WINNT
AUTOREAD	PCIADA reads 1 item ahead	

Table: wBlockTransfer modes

Output structure: There is nothing to output. Pointer to structure is NULL, size of structure is 0.

6.6. ControlInterrupts – VPCIC32_CONTROL_INTERRUPTS or PCICC32_CONTROL…

VPCIC32_CONTROL_INTERRUPTS (WIN95/98) or PCICC32_CONTROL_INTERRUPTS (WIN-NT) enables or disables interrupt requests for this path. Note that interrupt requests are associated to a CC32 module and not to a special path. This implies that only one of the paths accessing a special CC32 module can provide interrupt handling for this module.

Input structure:
```
typedef struct
{
        DWORD dwInterface;         // CC32 module number, idle for WIN-NT
        WORD  wEnable;             // a 1 enables, a 0 disables interrupt requests
} VPCIC32D_IRQ_CONTROL;   // (WIN95/98), PCICC32_IRQ_CONTROL
```

Output structure: There is nothing to output. Pointer to structure is NULL, size of structure is 0.

6.7. IRQHandler – VPCIVME_INSTALL_IRQ_HANDLER [0x00220018]

This is not applicable to WIN-NT.
VPCIC32_INSTALL_IRQ_HANDLER install a callback routine to a CC32 module. This function is called whenever interrupts are enabled and the handler is installed. To prevent interrupt overflows the driver itself clears the LAM-Flip-Flop or resets the PCIADA depending on the interrupt source.

Input structure:
```
typedef struct
{
        DWORD dwInterface;         // CC32 module number, idle for WIN-NT
        DWORD  dwIrqHandler;  // void (*IrqHandler)(DWORD),User Handler
```

```
                    } VPCIC32D_IRQ_HANDLER;
```

Output structure: There is nothing to output. Pointer to structure is NULL, size of structure is 0.

When called the IRQ-handler callback function is called with the current LAM-vector as argument. For all possible LAM-vectors see 6.8.

6.8. IRQResponse – PCICC32_INSTALL_IRQ_BLOCK [0x0022001C]

This is not applicable to WIN95/98.
PCICC32_INSTALL_IRQ_BLOCK installs a blocking IO-Control call which waits for a interrupt to occur. The call returns only if a interrupt has raised and the interrupt is enabled. It returns immediately without error when a interrupt was pending. To prevent interrupt overflows you have to re-enable interrupts when the call returns without error. No input structure is needed, the output structure provides information about what interrupts ceased the blocking.

Input structure: There is nothing to input. Pointer to structure is NULL, size of structure is 0.

Output structure:

```
typedef struct
{
        DWORD dwInterface;          // CC32 module number
        DWORD dwInterruptFlags;     // flags to mark pending interrupts
} PCICC32_IRQ_RESPONSE;
```

The "dwInterruptFlags" correspond to the LAM-AND Status of the CC32 associated to this path.

Bit #	Function
0	LAM0 AND LAM_MASK0
1	LAM1 AND LAM_MASK1
*	*
*	*
23	LAM23 AND LAM_MASK23
24	0
25	0
26	0
27	bTimeout
28	LAM-BUS-OR
29	LAM-NOT-OR
30	LAM-AND-OR
31	LAM-Flip-Flop

The PCIADA-timeout interrupt flag "bTimeout" is mapped into bit #27 of "dwInterruptFlags".

6.9 Access LCR – VPCIC32_ACCESS_LCR or PCICC32_ACCESS_LCR [0x00220020]

VPCIC32_ACCESS_LCR or PCICC32_ACCESS provides a way to access the Local Configuration Registers LCR of the PLX chip hosted on PCIADA. This feature is for test and debug only.

```
typedef struct
{
        ULONG  dwInterface;        // here dummy 'cause of compatibility to WIN95
        ULONG  dwContent;          // content to write, and, or
        USHORT wRegisterAddress;   // address offset of LCR register
        UCHAR  bAccessMode;        // LCR_READ, write, or, and
        UCHAR  bBytesLane;         // the data access width
} PCICC32_LCR_ACCESS;
```

```
// data lane size constants for PCICC32_ACCESS_LCR
#define BYTE_ACCESS (UCHAR)1   // write byte wise (illegal)
#define WORD_ACCESS (UCHAR)2   //      word
#define LONG_ACCESS (UCHAR)4   //      long

// PCICC32_ACCESS_LCR access constants
#define LCR_READ        0      // read only access
#define LCR_WRITE       1      // write and read back access
#define LCR_OR          2      // read, bitwise 'or' content and read back access
#define LCR_AND         3      // read, bitwise 'and' content and read back access
#define LCR_WRITE_ONLY 4       // do not read back after write
```

With the bAccessMode constant you can define the type of access. Even a atomic OR or AND of the content with the given register is possible.
Please note that using this feature in parallel to normal use of the driver can make the driver and even the operating system unstable.

7. CAMAC LIBRARY PCICC32_NI.DLL

The "pcicc32_ni.dll" adds a user friendly CAMAC library to the PCI-CAMAC driver. This layer standardizes the CAMAC calls for the different operating systems Windows 95/98, Windows NT and LINUX allowing to port application programs between these operating systems with minimum changes.
The following paragraph describes the CC32 / PCIADA CAMAC calls based on the use of the pcicc32_ni.dll and pcicc32_ni.lib Library. Please refer to the CC32 manual and the driver description in this manual.
The PCICC32dem.C file supplied on the driver and application CD-ROM shows examples for the use of this library. Please note the different device driver path declarations for use within Windows 95/98 and Windows NT and LINUX.

7.1. Initialize and Close calls

cc32_open ***(cszPath, nModuleNumber, *handle)***;

	char *cszPath	path to driver or device, depends on OS
	int nModuleNumber	number of CC32 (default 1)
	CC32_HANDLE *handle	handle to path

cc32_close ***(handle)***;

	CC32_HANDLE handle	handle to path

7.2. CAMAC Read and Write calls

Read 16 bits with N,A,F

cc32_read_word ***(handle, N, A, F);***

	CC32_HANDLE handle	handle to path
	unsigned int N	CAMAC station N
	unsigned int A	CAMAC sub-address A
	unsigned int F	CAMAC function F
return:	unsigned short data	16 bit data (D15..D00 = R16..R1)

Read 32 bits with N,A,F and get the result Q and X

cc32_read_long ***(handle, N, A, F, Q, X);***

	CC32_HANDLE handle	handle to path
	unsigned int N	CAMAC station N

unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
char *Q	Q response
char *X	X response
return: unsigned long data	32 bit data (D23..D00 = R24..R1, D29..D24 = 0, D31,D30 Q,X)

Read 32 bits with N,A,F

cc32_read_long_all ***(handle, N, A, F);***

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
return: unsigned long data	32 bit data (D23..D00 = R24..R1, D29..D24 = 0, D31,D30 Q,X)

Write 16 bits with N,A,F

cc32_write_word ***(handle, N, A, F, uwData);***

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
unsigned short data	16 bit data (D15..D00 = W16..W1)

Write 32 bits with N,A,F

cc32_write_long ***(handle, N, A, F, ulData);***

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
unsigned long data	32 bit data (D15..D00 = W16..W1)

Read 16 bit data into a buffer with N,A,F

cc32_read_word_buffer(handle, N, A, F, pwBuffer, pdwLen)

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
unsigned long *pwBuffer	pointer to a word (16 bit) buffer

unsigned long *pdwLen	pointer to the length of the buffer in words

Read 32 bit data into a buffer with N,A,F

cc32_read_long_buffer(handle, N, A, F, pdwBuffer, pdwLen)

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
unsigned long *pdwBuffer	pointer to a long (32 bit) buffer
unsigned long *pdwLen	pointer to the length of the buffer in longs

Read 32 bit data into a buffer with N,A,F without any masking of X, Q information

cc32_read_long_all_buffer(handle, N, A, F, pdwBuffer, pdwLen)

CC32_HANDLE handle	handle to path
unsigned int N	CAMAC station N
unsigned int A	CAMAC sub-address A
unsigned int F	CAMAC function F
unsigned long *pdwBuffer	pointer to a long (32 bit) buffer
unsigned long *pdwLen	pointer to the length of the buffer in longs

Set the access features for the next transfers

cc32_access_switch (handle, unsigned short wSwitch)

CC32_HANDLE handle	handle to path
unsigned short wSwitch	Constants SW_UNTIL_NOT_Q or SW_AUTOREAD (see driver manual)

7.3. General CAMAC commands

All CC32 commands are based on NAF mapping. System commands (as CAMAC C, Z, I, …) and LAM mask / broadcast are using station numbers (N) higher than 24. Please see the CC32 hardware manual for reference.

N0*A0*Fx	=	**C** (Camac Clear)	(Write Word)
N0*A1*Fx	=	**Z** (Camac Initialize)	(Write Word)
N0*A2*Fx	=	**C** + **I**nhibit reset	(Write Word)
N0*A3*Fx	=	**Z** + **I**nhibit set	(Write Word)
N27*A0*Fx	=	**I**nhibit set	(Write Word)

N27*A1*Fx	=	**I**nhibit reset	(Write Word)

Examples:

CAMAC-C	***cc32_write_word (handle, 0, 0, 16, 0);***
CAMAC-Z	***cc32_write_word (handle, 0, 1, 16, 0);***
Set Inhibit	***cc32_write_word (handle, 27, 0, 16, 0);***
Reset Inhibit	***cc32_write_word (handle, 27, 1, 16, 0);***

7.4. LAM Operations and Calls

N28*A0*Fx = LAM-FF reset (Write / Read Word)
N28*A1*Fx = LAM-mask (Write /Read Long)
With: D23..D00 << MASK24..LMASK1
D27..D24 = 0, D28 = LAM-BUS-OR,
D29 = LAM-NOT-OR, D30 = LAM-AND-OR, D31 = LAM-FF)
N28*A2*Fx = LAM AND mask (Read Long)
With: D23..D00 << AND24…AND1
D27..D24 = 0, D28 = LAM-BUS-OR,
D29 = LAM-NOT-OR, D30 = LAM-AND-OR, D31 = LAM-FF)
N28*A3*Fx = LAM NOT mask (Read Long)
With: D23..D00 << NOT24…NOT1
D27..D24 = 0, D28 = LAM-BUS-OR,
D29 = LAM-NOT-OR, D30 = LAM-AND-OR, D31 = LAM-FF)
N28*A4*Fx = LAM (no mask) (Read Long)
With: D23..D00 << LAM24…LAM1
D27..D24 = 0, D28 = LAM-BUS-OR,
D29 = LAM-NOT-OR, D30 = LAM-AND-OR, D31 = LAM-FF)

Any pending LAM of an enabled station is shown in the LAM FF register depending on the LAM-mask.
LAM interrupt status:

cc32_poll_error ***(handle, *nTimeout, *nLam);***

CC32_HANDLE handle	handle to driver
char *nTimeout,	time out flag
char *nLam	LAM flag

Examples:

Enable all LAMs	***cc32_write_word (handle, 28, 1, 16, 0xFFFF);***
Reset LAM-FF	***cc32_write_word (handle, 28, 0, 16, 0);***
Poll LAM	***cc32_poll_error (handle, *nTimeout, *nLam);***

7.5. Software interface

The interface to the pcicc32_ni.dll is defined in the Libcc32.h header file:

```
/*--- DEFINES -----------------------------------------------------------------------*/
#define SW_UNTIL_NOT_Q   1  /* switches for cc32_access_switch( …, uSwitch);           */
#define SW_AUTOREAD      2

/* open a path to a device. E.g. "/dev/pcicc32_1" */
int   __declspec(dllexport) cc32_open(char *cszPath, int nModuleNumber, CC32_HANDLE
*handle);
/* close the opened path */
int   __declspec(dllexport) cc32_close(CC32_HANDLE handle);
/* read only a word - 16 bits - from a address made out of N,A,F */
unsigned short __declspec(dllexport) cc32_read_word(CC32_HANDLE handle, unsigned int N,
unsigned int A, unsigned int F);
/* read a long - 32 bits - from a address made out of N,A,F and get the result Q and X */
unsigned long __declspec(dllexport) cc32_read_long(CC32_HANDLE handle, unsigned int N,
unsigned int A, unsigned int F, char *Q, char *X);
/* read a long - 32 bits - without any interpretation */
unsigned long __declspec(dllexport) cc32_read_long_all(CC32_HANDLE handle, unsigned int N,
unsigned int A, unsigned int F);
/* write a word - 16 bits - to a destination made out of N,A,F */
void  __declspec(dllexport) cc32_write_word(CC32_HANDLE handle, unsigned int N, unsigned int
A, unsigned int F, unsigned short uwData);
/* write a long - 32 bits - uninterpreted to a destination made out of N,A,F */
void  __declspec(dllexport) cc32_write_long(CC32_HANDLE handle, unsigned int N, unsigned int
A, unsigned int F, unsigned long ulData);
/* poll the state of the timeout line and the LAM state. The timeout line is cleared if it was set */
int   __declspec(dllexport) cc32_poll_error(CC32_HANDLE handle, char *nTimeout, char *nLam);
/* read 'len' words or 'UNTIL_NOT_Q' from a address made out of N,A,F into a buffer*/
int __declspec(dllexport) cc32_read_word_buffer(CC32_HANDLE handle, unsigned int N, unsigned
int A, unsigned int F, unsigned short *pwBuffer, unsigned long *pdwLen);
/* read 'len' longs or 'UNTIL_NOT_Q' from a address made out of N,A,F into a buffer*/
int __declspec(dllexport) cc32_read_long_buffer(CC32_HANDLE handle, unsigned int N, unsigned
int A, unsigned int F,          unsigned long *pdwBuffer, unsigned long *pdwLen);
/* read 'len' longs or 'UNTIL_NOT_Q' from a address made out of N,A,F into a buffer, no
interpretation */
int __declspec(dllexport) cc32_read_long_all_buffer(CC32_HANDLE handle, unsigned int N,
unsigned int A, unsigned int F,  unsigned long *pdwBuffer, unsigned long *pdwLen);
/* switch UNTIL_NOT_Q or AUTOREAD on or off */
```

8. LABVIEW-VI'S

Together with the „pcicc32_ni.dll“ the following virtual instruments (VI's) for the National Instruments graphical programming software for instrumentation LabView are provided.

c32Init.vi	the VI to initialize a path to a interface
c32Close.vi	the VI to close a path opened with c32Init.vi
c32PollError.vi	the VI to get a pending LAM or an interface connection timeout
C32ReadLong.vi	the VI to read a longword (32 bit) containing X and Q
C32ReadLongS.vi	the same as above but with resolved X and Q
C32ReadWord.vi	the VI to read a word (16 bit) without reading X and Q
C32WriteLong.vi	the VI to write a longword (24 bit CAMAC)
C32WriteWord.vi	the VI to write a word (16 bit) only
c32Test1_95.vi	a simple write / read test configured for WIN95/98.
c32Test1_NT.vi	the same as above but for WIN-NT
C32Test2_95.vi	a simple LAM test.
C32Test2_NT.vi	the same as above but configured for WIN-NT.
pcicamac.vi	CAMAC example controller with loop functions and NAF/R/W display.

9. DRIVER LIMITATIONS

- Presently the driver for LINUX does not support hardware interrupt servicing however, all interrupt sources (time out and LAM) can be monitored by polling the VPCIC32D_GET_STATUS (WIN95/98) or the PCICC32_GET_STATUS (WIN-NT) function.
- Presently the Labview VI's for LINUX is not adopted to the shared library for LINUX.
- Presently some of the DLL entries does not have a Labview NI counterpart.