PCICC32 − Linux Driver Quick Installation and Usage Guide

ARW Elektronik, Germany
and
Klaus Hitschler (klaus.hitschler@gmx.de)

document history

1st version of this document	16.04.2002

Preface

Please announce changes and hints to ARW Elektronik

Installation contents

```
pcicc32
|-- driver                  all to make a driver
|   |-- Makefile            Makefile for kernel 2.4
|   |-- Makefile.2.2.13     Makefile for kernel 2.2
|   |-- askpci.c            all PCI stuff
|   |-- askpci.h
|   |-- common.h            must be the 1st include file of all driver components
|   |-- fops.c              all file operations
|   |-- fops.h
|   |-- list.c              make lists of anything
|   |-- list.h
|   |-- main.c              main part of the driver
|   |-- pcicc32.h           header file for direct use of the driver
|   |-- pcicc32_load        driver load script
|   |-- plx9050.h
|   |-- plxbug.c            to circumvent the PLX bug
|   `-- plxbug.h
|-- lib                     all to make the shared library
|   |-- Makefile            Makefile for the shared library
|   |-- libcc32.c           sources of the lib
|   |-- libcc32.h           header file to use the lib
|   `-- libcc32.so.1.0.1    the shared library
|-- pcicc32.pdf             this simple HOWTO
|-- template.c              a template file for new development
|-- Makefile                a global Makefile
|-- test                    all to make the simple test program
|   |-- Makefile            Makefile for a simple test program
|   |-- pcicc32_test
|   `-- pcicc32_test.c      source of the test program
`-- var_log_messages.txt    example of log messages
```

Compatibility

Driver and library are compiled and tested on a machine with kernel 2.2.13 (SuSE 6.1) and kernel 2.4.10 (SuSE 7.3) and RedHat 7.2. The source code is compatible to 2.2 and

2.4 kernels. For 2.2.x kernels please use the Makefile.2.2.13 to create the driver.

I think it will compile and run on future versions, but not on versions before 2.2.x.
The sources are independent of special x86 hardware features and should compile on other platforms. But this is not tested.

Features

The driver and the shared library provide functions to access the CC32 CAMAC interface, use the advanced features of the interface and makes it possible to catch LAM interrupts.

Driver and the library are capable of multi−user and multi−threading access.

Please note that some high speed access features like „autoread“ are heavily hardware supported and should not interrupted by concurrent accesses from different paths. The same is true for interrupt handling. The necessary arbitration is not done by the driver or the library.

Installation of the driver

For installation of the driver module must be "root". There is a installation (bash) script called "pcicc32_load". Please invoke it with the module number of your CC32 module as command line parameter.
For example, if your interface is configured as module #1 (Jumpers J301 .. J304) then call

```
./pcicc32_load 1
```

This installs the driver and creates a device node "/dev/cc32_1".

The module number "1" should be the factory set module number.

If you want to remove the driver do it with "rmmod pcicc32". (sometimes /sbin/rmmod ...)

Install the shared library

The shared library "libcc32.so" provides low level functions to access the CAMAC interface. You will find the prototypes of this functions in the file "libcc32.h".

Copy the library "libcc32.so.x.x.x" (now 1.0.1) to your /usr/lib directory. Then cd to /usr/lib and make 2 (soft) links:

```
ln −sf libcc32.so.x.x.x libcc32.so.1
ln −sf libcc32.so.1 libcc32.so
```

Instead of typing lots of commands you can use the build in

```
cd lib
make install
```

Please note: you must have root access rights.

Verify the installation (1)

I provided a little test program which really does nothing useful. It is named "pcicc32_test". After making lots of LEDs gloom it generates a LAM interrupt and then stops. This is normal because it tests the behavior of „no raising interrupt“. To kill the program please type Ctrl−C.

Verify the installation (2)

The "cat /proc/pcicc32" output is now more detailed. Please take a look:

```
pcicc32 information. Version 4.4 of Apr 14 2002 from Klaus
Hitschler.
 ---------------------
 Interfaces found : 1
 --- 1 ---------------
 LCR  phys/virt/size : 0xe7009000/0xe08d4000/128
 User phys/virt/size : 0xe7000000/0xe08f9000/32768
 Irq                 : 11
 CC32 is or was      : (software) connected.
 Module−Number       : 1
 FPGA−Version        : 3
 IrqCount            : 1
 Pending IrqStatus   : None
```

The output will list all found interfaces and their parameters. The output of your computer will look different.

Making driver, library or test program

Change your directory into “pcicc32”, then simply type “make”. To make each part simply type in “cd” into the appropriate directory and invoke “make”. To remove object code please call “make clean”, to install driver and lib (as root only) call “make install”.

Dynamic major number allocation

The driver uses the dynamic major number allocation. You can switch to static allocation through changing in “main.c”

```
#define MAJOR_NO 0              /* use dynamic assignment */
```

the MAJOR_NO to an appropriate number not equal 0.

Modversions

If you want to have version control check against the kernel symbols you have to

configure the switch "CONFIG_MODVERSIONS" before making your kernel. All provisions for version check are included in the driver sources. Normally the RedHat distribution configured MODVERSIONS support as default.

Debug information

To get more debug information from the driver please compile the driver with the switch DBG = __DEBUG__ (double underline), e.g. make DBG = __DEBUG__
Then additional debug information is printed into the file /var/log/messages. You can watch it with „tail −f /var/log/messages“. You must be root to do this.

Include header files

If you want to use the shared library please include the file “libcc32.h”. To access the driver ioctl() functionality without shared library you must use the include file “pcicc32.h”.

Interrupt handling

Interrupt handling is supported from driver version greater than 4.4. The user interface supports 2 ways of dealing with interrupts.
Either you use the blocking IO call cc32_wait_event() in a multi−threading environment or you use the poll/select method. Please look into the header files of the shared library. (The poll/select method is not yet tested and still not supported through the library. Please give me a notice if you use it successfully.)

Each raised interrupt disables further interruptions. Normally, when your program returns from the blocking IO−call you will handle the cause of the LAM and then re−enable the interrupts.

Generally there are 2 interrupt sources. The first interrupt is raised by the PCIADA when you try to access the CC32 module, but the module does not respond after a timeout period.
The second source feeds from the LAMs of the CC32 module. Please note, you have to unmask the LAMs to make them able to generate interrupts.

Path to modutils

Some Linux distributions provide the utilities “rmmod” and “insmod” in the standard paths. Sometimes they must be called with full path description, e.g. /sbin/rmmod pcicc32.

Historical changes

In versions greater than 4.3 the former cc32_poll_error() library function was renamed to cc32_poll_event(). This represents more the functionality and distinguishes from cc32_wait_event().

A major change was done in changing the PCI−DEVICE−ID of the PCIADA interface for use with CC32 (WIN95 forced it!). Please recompile the sources if the DEVICE−ID of your PCIADA states 0x2258. (cat /proc/pci)

Feedback

Please mail your hints, questions and remarks to klaus.hitschler@gmx.de. All feedbacks are welcome.