



Delta Tau PMAC2 PCI Ultralite Linux Driver User Manual



Alan Greer
Observatory Sciences Ltd

25th May 2007

*Observatory Sciences Ltd.
William James House
Cowley Road
CAMBRIDGE
UNITED KINGDOM.
CB4 0WX*

Phone: (+44)-(0)1223-508257

Revision Summary:

1. Date: 22nd June 2006
Revision: V1.0
Changes: Initial version of the PMAC2 PCI Driver manual.
2. Date: 25th May 2007
Revision: V1.1
Changes: Modified for general release
3. Date:
Revision:
Changes:
4. Date:
Revision:
Changes:
5. Date:
Revision:
Changes:
6. Date:
Revision:
Changes:

Contents

1	Introduction.....	4
1.1	Hardware Requirements.....	4
1.2	Software Requirements.....	4
1.3	References.....	4
2	Installation.....	4
2.1	Driver Installation	4
3	Using the Driver.....	5
3.1	Open.....	5
3.2	Close	5
3.3	Write	5
3.4	Read	6
3.5	Ioctl	6
3.6	Mmap	7
4	Linux PMAC Test Program.....	9
4.1	Talking to the PMAC PCI Card.....	9
4.2	PLC Program Downloading.....	10
5	Debugging And Support	12
5.1	Running Driver In Debug Mode	12

1 Introduction

This section details the hardware and software requirements for using the PMAC2 PCI Ultralite Linux 2.6 driver and associated device support code.

1.1 Hardware Requirements

The following hardware is required to run the driver:

- PC with x86 CPU and at least 1 PCI slot
- At least 1 Delta Tau PMAC2 PCI Ultralite card installed. A maximum of 8 cards are supported.
- If Dual Ported RAM is to be used then the card must have the DPRAM option installed.

1.2 Software Requirements

The following software was used to build and test the driver support code:

- Redhat Enterprise Linux Version 4. (Linux kernel 2.6.9)

There is no guarantee that the software will build or run correctly with versions of the Linux kernel different to that listed above.

1.3 References

- [1] *Turbo PMAC/PMAC2 Software Reference Manual, Delta Tau Systems, Inc.*

2 Installation

2.1 Driver Installation

Having uncompressed and installed the files contained in the supplied tar file, the top-level directory will be seen to contain the following files:

- pmacpci.c
- pmacpci.h
- pmacpcid.h
- pmac.init
- Makefile
- pmacpci.conf

Edit the file pmacpci.conf. Change the <owner name>, <group name> and <mode name> to the required values. The <insmod options> can be used to load the driver in debug mode if required. To load the driver in debug mode add the following line to the pmacpci.conf file:

```
options      pmac_debug=1
```

Save the file once it has been edited and the correct options set up. Once complete as a normal user type gmake. This will compile the code ready for installation. There

should be no warnings or errors generated by the compilation. After this has completed become a super user and run the pmac.init script. Type

```
./pmac.init start
```

and hit return. This will load the driver using the configuration script that you previously edited. Installation of the driver is now complete.

3 Using the Driver

This section identifies how the driver can be used for any application. No knowledge of EPICS is required for this and the device support code is not needed. Any application can access the PMAC driver by using the system calls explained below. More information on the system calls themselves can be found in the Linux man pages.

3.1 Open

```
int open(const char *pathname, int flags)
```

To use the driver in an application it must first be opened. The open system call is used to convert a pathname into a file descriptor. When successful the file descriptor returned should be stored and used for all subsequent calls to this device. To use the PMAC driver the flags should be set to O_RDWR (read or write). If there is an error opening then this function will return -1 and the value of errno will be set appropriately. Below is an example of how to use this function:

```
int fd = 0;
char pmac_device[14] = "/dev/pmacpci0\0";

fd = open(pmac_device, O_RDWR);
if (fd < 0){
    perror("pmac error: ");
}
```

3.2 Close

```
int close(int fd)
```

Once finished with the driver it is good practise to close it. The close system call takes a file descriptor and closes the device.

3.3 Write

The write system call allows ASCII communication through the PCI bus communications port with the PMAC PCI card. ASCII strings can be sent to the PMAC card using the function prototyped below.

```
int write(int fd, const void *buffer, int nbytes)
```

The file descriptor (fd) used is the one obtained from the open call. For the PMAC the buffer expected is a null terminated string. This string will be issued to the PMAC

PCI card. The final parameter required is the number of bytes that are being written. The return value is the number of bytes written on success, and -1 on error with `errno` getting set appropriately. See the example below. It assumes `fd` has already been obtained from opening the device.

```
int result = 0;
char cmd[5] = "type\0";
result = write(fd, cmd, strlen(cmd));
if (result < 0){
    perror("pmac error: ");
}
```

3.4 Read

The read system call allows ASCII communication through the PCI bus communications port with the PMAC PCI card. ASCII strings can be read from the PMAC card using the function prototyped below.

```
ssize_t read(int fd, void *buffer, size_t count)
```

The file descriptor (`fd`) used is the one obtained from the open call. The returned buffer value will be a null terminated string. The return value of the function will be the number of bytes read, or a -1 if there is an error. As usual with an error the `errno` will be set with the error type. The count is the maximum number of bytes that will be read. See the example below. It assumes `fd` has already been obtained from opening the device.

```
int result = 0;
char buf[256];
result = read(fd, buf, 256);
if (result < 0){
    perror("pmac error: ");
}
```

3.5 ioctl

The `ioctl` system call can be used to access the DPRAM on board memory banks, although it is more efficient to memory map the location using the `mmap` call (explained below). As well as providing access to the DPRAM the `ioctl` call can be used to perform maintenance on the driver by flushing the buffer, resetting the card or even setting the debug level.

Using the `ioctl` call can be slightly more complicated than the other system calls, as some of the `ioctl` calls require a structure containing information. Any programs that wish to use the `ioctl` calls will therefore need to include "pmacpci.h". This header file defines the structure that is used to pass the information.

The structure contains the following fields:

- Buffer. This is a character buffer used to pass strings to the driver or to receive strings from the driver.
- Err_code. This may contain an error code if appropriate. (Not currently used).
- Dpr_offset. For `ioctl` calls that utilise DPRAM this specifies the offset from the beginning of DPRAM memory.
- Dpr_count. For `ioctl` calls that manipulate DPRAM memory locations this field specifies the size of memory that is used.

- Status. This field contains the status from ioctl calls; the values are defined in the header files.

The ioctl function prototype is shown below.

```
int ioctl(int fd, int request, ...);
```

The file descriptor (fd) used is the one obtained from the open call. The request parameter is used to specify which ioctl call is to be made. The various ioctl calls are described in detail below. Finally, if the specified ioctl call requires a structure (of type `_Device_Ext` as described above) then the third parameter must specify the address of the structure.

The following ioctl calls can be made.

IOCTL_GET_MEM. This copies an area of DPRAM into a buffer supplied. The fields required from the structure are `dpr_offset` and `dpr_count`. The copied memory will be placed into buffer.

IOCTL_SET_MEM. This copies the buffer into an area of DPRAM memory. The fields required from the structure are `dpr_offset` and `spr_count`.

IOCTL_FLUSH. This performs a flush of the IO port of the PMAC card. No parameters are required.

IOCTL_DPRAM_CMD. Use this ioctl call to issue a command to the PMAC card using DPRAM ASCII communications. The command should be written into the buffer field of the structure. After completion, the buffer field will contain any response received from the card.

IOCTL_RESET. Use this command to issue a complete reset of the PMAC card. This will issue a “\$\$\$***” command to the PMAC card. No parameters are required.

IOCTL_DEBUG. This ioctl call can be used to set the debug level of the driver while it is in operation. In this case the third parameter passed to the ioctl system call is not the address of the structure but the actual debug level required (0-5). Level 0 indicates no debug messages while level 5 indicates maximum debug messages. Typing “dmesg” will display a kernel log that contains any debug messages from the PMAC driver. See section 4 for a full explanation.

3.6 Mmap

The mmap system call is the fastest and recommended way of accessing the on board DPRAM. The DPRAM can be memory mapped so that it appears as a block of memory in the user application. It can then be manipulated just like any other memory location. The mmap function prototype is shown below.

```
void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);
```

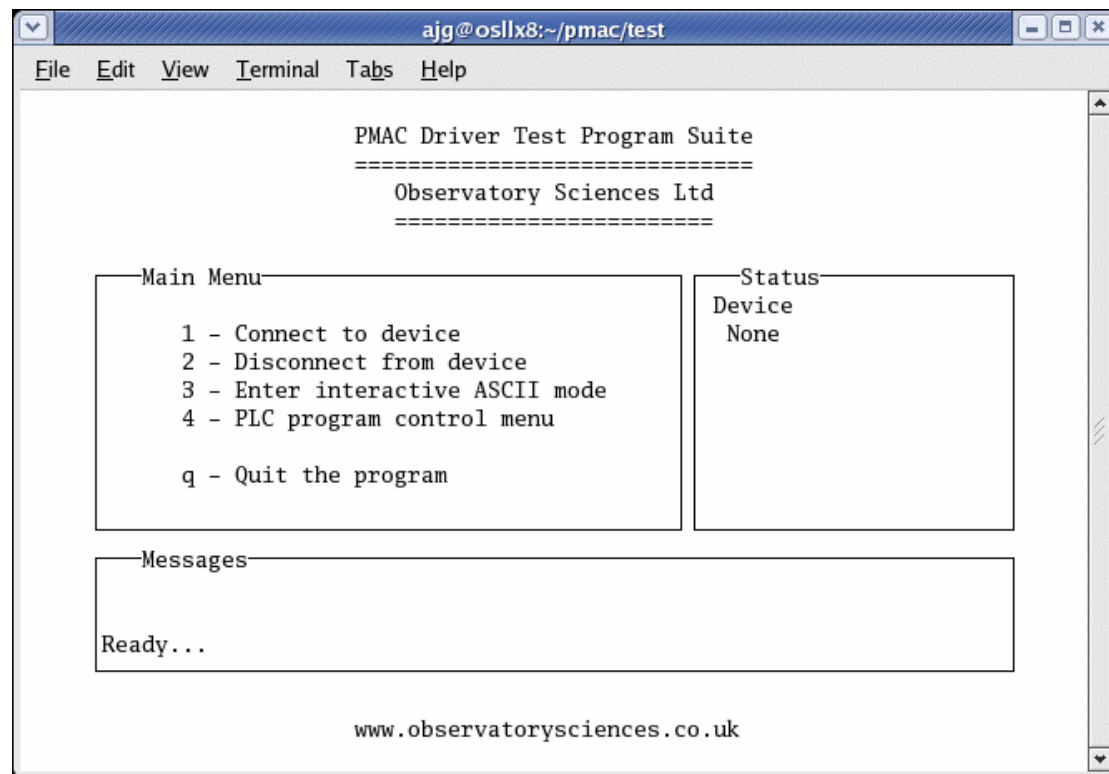
For use with the PMAC2 PCI Ultralite card it is recommended to call the function with the following parameters. Set start to 0. Assuming DPRAM of 32k x 16 then the variable length should be set to 65535. The prot value should be the OR value of PROT_READ and PROT_WRITE. The flags can be set to MAP_SHARED. The file descriptor (fd) used is the one obtained from the open call, and finally offset should be set to zero. The return value will be a pointer to the start of the mapped DPRAM and should be cast as a pointer to a character for easier byte manipulation. See the example below.

```
char *pDpram;  
pDpram = (char *)mmap(0, 65535, PROT_READ | PROT_WRITE,  
MAP_SHARED, fd, 0);  
If (pDpram == MAP_FAILED){  
    printf("Pmac: Memory map failed.\n");  
}
```

Once this pointer has been obtained DPRAM memory can be written to or read from directly.

4 Linux PMAC Test Program

Included with the driver is a test program that allows access to most of the PMAC functionality. The program runs in a terminal and is designed to connect to any PMAC card through the driver. When the program is started you will be presented with the following display shown below.

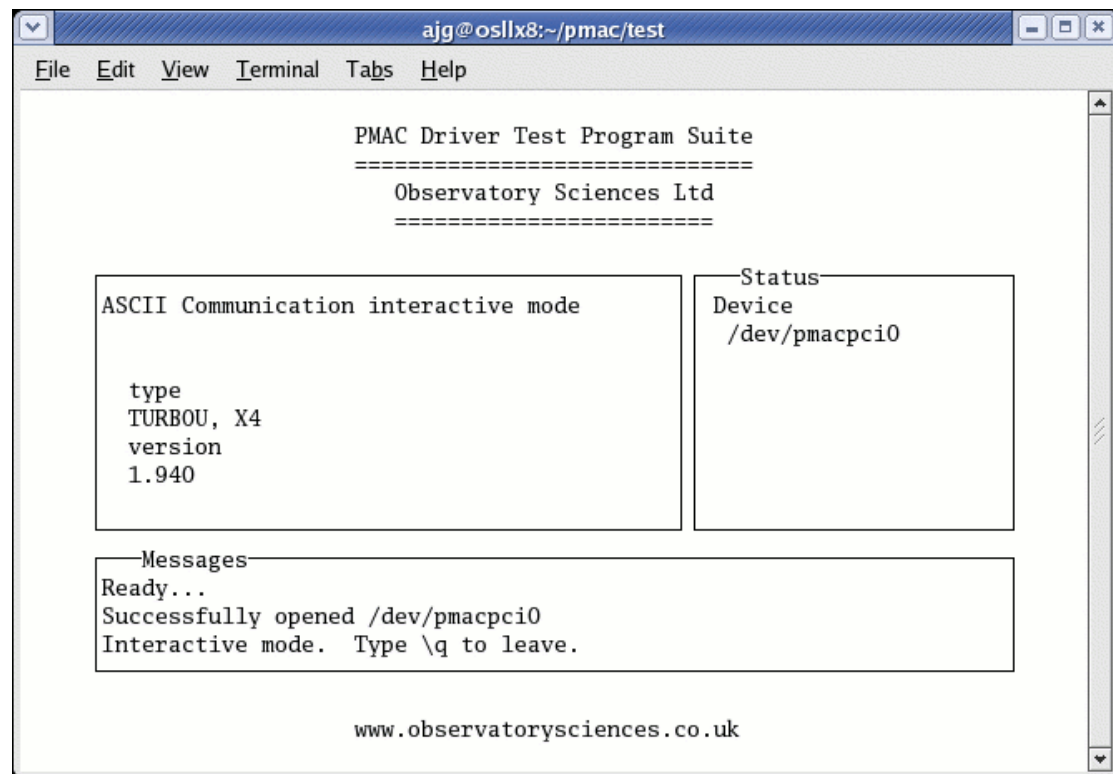


The screen is split into three windows, the main window, status window and messages window. The status and message windows will never change, they will just update the information contained within. The main window however, will display various menus and selections depending on which options have been selected. Most menu options are selected simply by selecting the corresponding number; there is no need to press return.

4.1 Talking to the PMAC PCI Card

From the main menu you need to connect to the device. Press 1 and you will be prompted to enter the path to the device. If there is one PMAC PCI card present in the system enter `"/dev/pmacpci0"` and hit return. If there is more than one card present then enter the appropriate path and file and hit return. You will be returned to the main menu above, but underneath the Device heading of the status window the text will display the currently connected device. A message will also be displayed in the messages window informing you of a successful connection. In the event that connecting was unsuccessful a message will be displayed and the currently connected device status will remain at "None".

Once connected you can talk to the card by entering the interactive ASCII mode (option 3) from the main menu. In the interactive mode typing in a command and hitting return will send the command to the PMAC card. If the command has a response then this will be displayed underneath. See below.



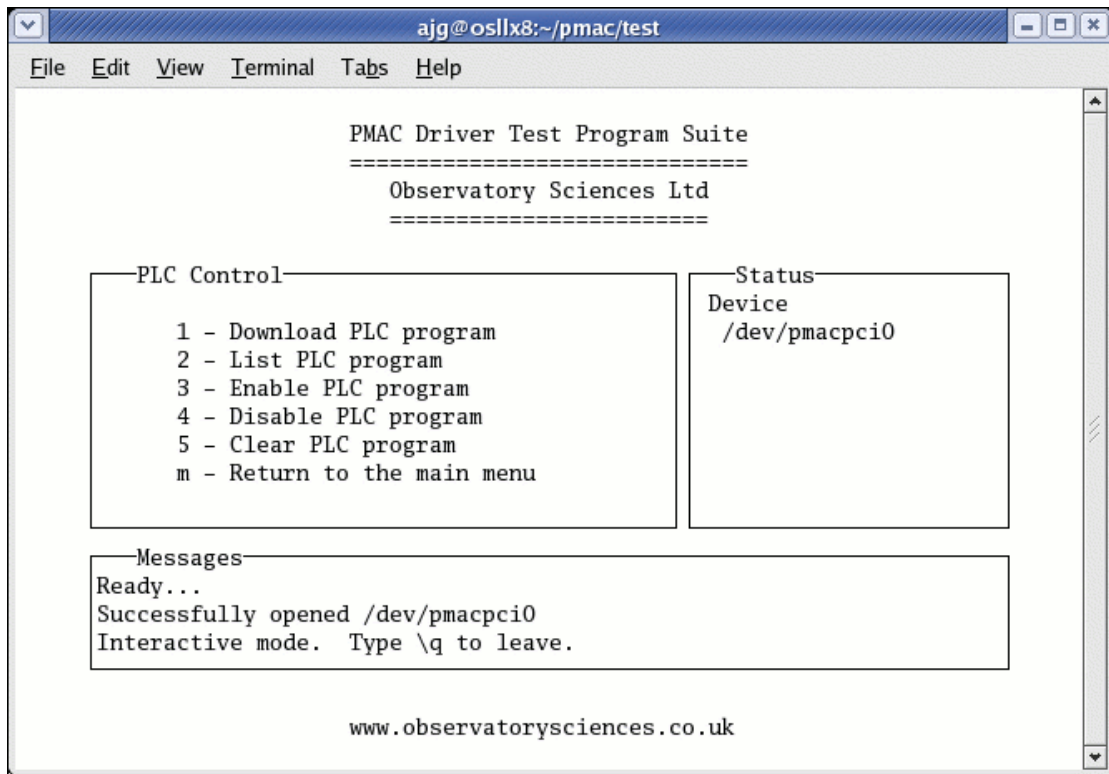
To exit the interactive mode type “\q” and hit return. All interactive ASCII communications are executed through the communications port, not using DPRAM.

4.2 PLC Program Downloading

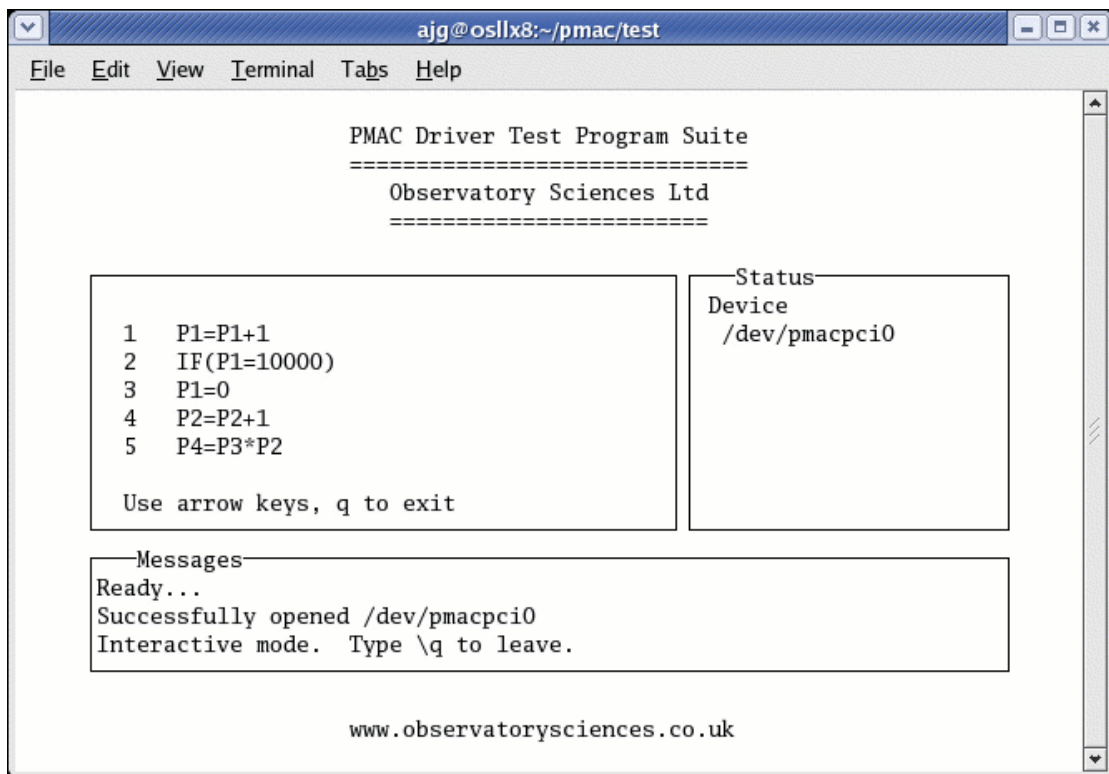
The program also offers some assistance for downloading PLC programs to the PMAC board. From the main menu select option 4. This will open the PLC Control menu offering the following options:

- 1) Download PLC program
- 2) List PLC program
- 3) Enable PLC program
- 4) Disable PLC program
- 5) Clear PLC program

Downloading a program allows you to specify the file that contains the program and the slot that you would like to download it to (0-31). Only uncompiled PLC programs can be downloaded using the test program.



When selecting to list a PLC program you will be asked to enter the slot number of the PLC program that you wish to list (0-31). Once entered the program will be displayed.



To scroll through the program that is currently displayed use the arrow keys. Pressing q will quit out of the display screen and return to the PLC menu.

Enabling and disabling PLC programs is fairly self-explanatory. As usual you will be asked for the slot number and then the corresponding program will be started or stopped.

Clearing a PLC program will remove permanently it so be sure to have a copy of the program if it might be required in the future.

5 Debugging And Support

5.1 Running Driver In Debug Mode

As explained in sections 2.1 and 3.5 the driver can be loaded in debug mode and the debug mode can be changed after the driver has been loaded by making the ioctl system call. To view any messages that have been generated simply type “dmesg” in a terminal. All debug messages that have originated from the PMAC PCI driver will be prefixed by “pmac:”

Some extra information about the cards present in the system can be viewed by typing “more /proc/driver/pmac/pmacpci” in a terminal. If the driver has not been loaded then this file will not be found.