



Delta Tau PMAC2 PCI Ultralite Driver EPICS Device Support 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 EPICS device support 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
2.2	Device Support Code Installation	4
3	Using The Device Support Code	5
3.1	ASCII Driver Support.....	5
3.2	DPRAM Driver Support	6
3.3	DPRAM Status Record.....	7
4	Debugging And EPICS Support	8
4.1	Running Driver In Debug Mode.....	8

1 Introduction

This section details the hardware and software requirements for using the Delta Tau PMAC2 PCI Ultralite 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(s) must have the DPRAM option installed.

1.2 Software Requirements

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

- Redhat Enterprise Linux Version 4. (Linux kernel 2.6.9)
- EPICS version 3.14.8.2
- Asyn Driver version 4.5

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

1.3 References

- [1] *PMAC2 PCI Ultralite Driver User Manual*. Alan Greer, Observatory Sciences Ltd.
- [2] *asynDriver: Asynchronous Driver Support Release 4.5*. Marty Kraimer, Eric Norum and Mark Rivers, Argonne National Laboratory
- [3] *Turbo PMAC/PMAC2 Software Reference Manual*, Delta Tau Systems, Inc.

2 Installation

2.1 Driver Installation

Device driver installation is covered in the Driver user manual [1] and so will not be discussed here.

2.2 Device Support Code Installation

This section will cover installation of the basic device support code for the PMAC. This includes building the necessary device support libraries as well as the status record. Before performing the following installation, ensure the EPICS asynDriver version 4.5 (or later) has been installed and built.

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

- **configure**. Contains configuration files for building the EPICS libraries.

- **pmacApp.** Contains all source code for the EPICS device support software. The EPICS DPRAM status record code is included here as it is required for the build of the DPRAM driver device support code.

Edit the `configure/RELEASE` file, ensuring that the `EPICS_BASE` and `ASYN` variables point to the correct locations. Save the file and `cd` back to the top directory. Type `gmake` and all of the software should build correctly.

3 Using the Device Support Code

The device support code uses the `asynDriver` library. To initialise a PMAC PCI card at start up the following command needs to be issued:

```
drvAsynPmacInit("C0", "/dev/pmacpci0", 0, 0)
```

This command should be added to the `st.cmd` (or other) startup file. The string “C0” specifies the name of the port that the records will require in their DTYP fields. C0 corresponds to @C0. See below. The device “/dev/pmacpci0” should point to the PMAC PCI device that is required, and so could be different to that given in the example above if there is more than one card present in the system. The final two values are the `autoConnect` and `processEos` (see `asynDriver` documentation) and should be set to zero for the PMAC device support code.

3.1 ASCII Driver Support

PMAC ASCII driver device support code is provided for the following records:

- `stringin`
- `stringout`
- `ai`
- `ao`
- `longin`
- `longout`
- `bi`
- `bo`
- `mbbi`
- `mbbo`

When using one of these records the DTYP field should be set to “PMAC-PCI ASCII”. The input records are used for monitoring variables; the output records are used for issuing commands. The exception to this rule is the `stringout` record. Whenever a value is entered into the `stringout` record it is issued as a command to the PMAC card. The value of the `stringout` is then replaced with the answer from the PMAC card (if there is an answer).

For the in records (`stringin`, `ai`, `longin`, `bi`, `mbbi`) the INP field must be specified as follows:

```
@#Cn Sn _____
```

where n is the card (port) number after the C, n represents the motor number after the S, and the underlined part is replaced with the command to which the answer is required.

For example,

```
@#C0 S0 I1
```

would look for card (port) 0, motor 0 and get the value of I1. There are many situations where the motor number is not important, as in the case above but for consistency it is always supplied. If the motor is not important then this number can be set to 0.

For the out records (except stringout) the OUT field must be specified as follows:

```
@#Cn Sn _____
```

where n is the card (port) number after the C, n represents the motor number after the S, and the underlined part is replaced with the command to issue the PMAC with. For example, setting a longout record with an OUT field of

```
@#C0 S0 I2=
```

will append the value supplied to the command I2= and issue that command to the PMAC card. As before if the motor number is unimportant then leave it set to zero.

The special case for ASCII support is the stringout record. This record can just have the OUT field set to the card number and motor number. For example

```
@#C0 S0
```

To issue a command enter the value into the record and process it. The value in the record will then be replaced by the response from the PMAC if there is one available. This record is useful for configuring and testing the PMAC card.

The test database included with the distribution contains various examples of different records set up to perform different actions, and the GUI screens provided allow interaction with these records.

3.2 DPRAM Driver Support

PMAC DPRAM driver device support is provided for the following records:

- ai
- ao
- bi
- bo
- login
- longout
- mbbi
- mbbo
- status

When using one of these records the DTYP field should be set to “PMAC-PCI DPRAM”. The input records are used for monitoring DPRAM locations and the output records allow setting values into the DPRAM locations. The format for setting up the location is best explained through an example:

```
@#C0 S0 DP:$060450
```

The first two parts are the same as before, C0 represents the port C0 specified from the initialisation call, and the S0 is unused for DPRAM. The next part specifies the address that this record is pointing to and the type of data contained in that particular address. The types of data that can be read or written to are:

- X. 1 to 24 bits fixed-point in X-memory
- Y. 1 to 24 bits fixed-point in Y-memory
- D. 48 bits fixed-point across both X and Y-memory
- L. 48 bits floating-point across both X and Y-memory
- DP. 32 bits fixed-point (low 16 bits of X and Y)
- F. 32 bits floating-point (low 16 bits of X and Y)

The address specified is the Turbo PMAC address as defined in the *Turbo PMAC/PMAC2 Software Reference* manual [3].

DPRAM function	Address
Control Panel Functions	\$060000
Motor Data Reporting Buffer	\$06001A
Background Data Reporting Buffer	\$06019D
DPRAM ASCII Command Buffer	\$0603A7
DPRAM ASCII Response Buffer	\$0603D0
Background Variable Read Buffer Control	\$060411
Background Variable Write Buffer Control	\$060413
Binary Rotary Program Buffer Control	\$060414
DPRAM Data Gathering Buffer Control	\$06044F
Variable-Sized Buffers & Open-Use Space	\$060450
End of Small (8k x 16) DPRAM	\$060FFF
End of Large (32k x 16) DPRAM	\$063FFF

It is important to note that any record can be used to read any data type. For example a longin could read from a location F:\$060450. This location actually contains a floating-point value. The device support code would decode the floating point type correctly and then cast it into the long value required for the record.

The test database included with the distribution contains various examples of different records set up to writing into and reading from PMAC DPRAM locations, and the GUI screens provided allow interaction with these records.

3.3 DPRAM Status Record

The DPRAM status record is designed to read up to 32 bits from a DPRAM location and make those 32 values available as binary true/false outputs. This can be used to report various status bits (hence the name) available from the PMAC card depending on how the DPRAM memory has been set up. The record support code is contained

in the directory `pmacApp/src/statusRecord`. The status record uses the same device support code that is used for other records that access DPRAM locations and so the format for the “DTYP” field is also the same. For example

```
@#C0 S0 DP:$060450
```

There are no special instructions relating to the status record or its use, it can be used like any other EPICS record.

4 Debugging and EPICS Support

4.1 Running Driver In Debug Mode

The PMAC2 PCI Ultralite Driver EPICS device support code has been written using the `asynDriver`: debugging using the `asynDriver` is covered fully in the documentation supplied with the `asynDriver`. However, a summary of commands is shown below that can be entered in the shell to turn on the debugging provided by the `asynDriver` and device support code.

The command `asynReport(n)` will provide diagnostic details for the current ports. The value supplied for `n` dictates the level of the report. For each PMAC board there will be two ports, one for port communications and one for DPRAM access. This is necessary because the port communications can block whereas the DPRAM access calls do not block.

To turn on tracing two calls need to be made for each port. They are

```
asynSetTraceFile("PMAC_PORT_C0", 0, "stdout")
asynSetTraceMask("PMAC_PORT_C0", 0, 31)
asynSetTraceFile("PMAC_DPR_C0", 0, "stdout")
asynSetTraceMask("PMAC_DPR_C0", 0, 31)
```

This will provide debugging output from the `asynDriver`.