Gemini Mount Control System Report

Notes on the Design and Use of the EPICS PMAC Device Driver

Andy Foster

Version 1

17th July1996

1. Introduction

These notes have been written as a result of a two day meeting with Tom Coleman, the author of the EPICS PMAC device driver.

In order to use the driver, it is necessary to have at least version 1.15 of the PMAC firmware.

The driver is written in the standard EPICS way. The record support layer calls the device support layer which in turn calls the driver. In the device support layer, there are two separate modules, these are the ASCII module and the Dual-Ported Ram (DPRAM) module. Figure 1 shows the architecture of the driver code.

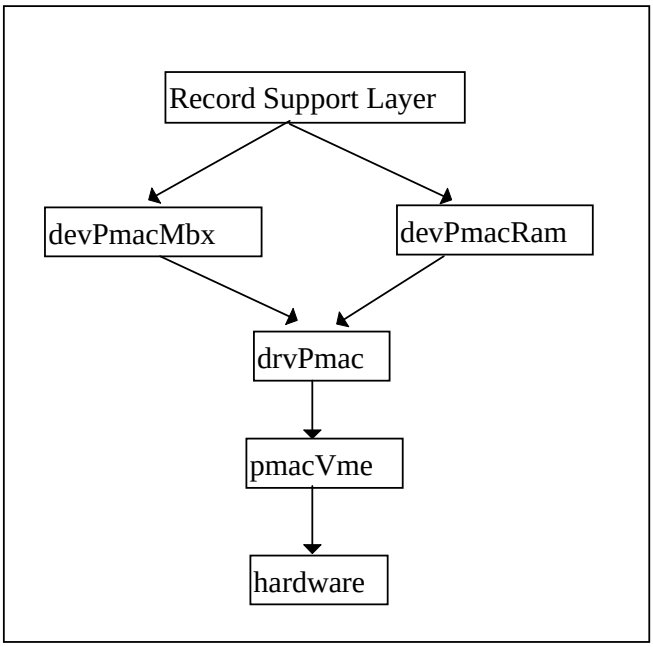

Figure 1: The architecture of the EPICS PMAC Device Driver

In figure 1, 'devPmacMbx' is the device support code for ASCII mailbox communication. 'devPmacRam' is the device support code for communication with the DPRAM. 'drvPmac' is the driver support code for PMAC. 'pmacVme' is a set of routines which perform data conversions between the various types which PMAC supports. The code which handles the low-level communication with the mailbox registers is also contained in this module.

2. PMAC Variables

PMAC defines four types of variable. These are specified by a single letter, I, P, Q , M followed by a number 0 to 1023. We briefly describe each type of variable in the following sections.

2.1 I-Variables

Initialisation or setup variables determine the personality of the card for a given application. They are at fixed locations in memory and have predefined meanings. Most are integer values. In particular, the I-variables are used for:

- General card setup
- Motor(s) setup
- Coordinate System(s) setup
- Encoder(s) setup

2.2 P-Variables

These are general purpose user variables. They are 48-bit floating point variables at fixed locations in PMAC's memory, but with no predefined use. P-Variables can be used in programs for any purpose, i.e. they can hold positions, distances, velocities, times, angles etc. A given P-Variable is accessible from all coordinate systems and therefore provides a way of passing information between them.

2.3 Q-Variables

Like P-Variables, these are general purpose 48-bit floating point variables at fixed locations in memory. However, the value held in a given Q-Variable is specific to the coordinate system which is referencing it. This means that Q1 can hold different values in two different coordinate systems.

2.4 M-Variables

These are provided to allow the user easy access to PMAC's memory and I/O space. M-Variables point to a location in the memory-I/O space of PMAC's processor. The user defines an M-Variable by assigning it to a memory location, and defining the size and format of the value in this location.

3. ASCII Module

The ASCII module supports communication with the PMAC mailbox registers. This module can be called from the following records:

ANALOGUE IN
ANALOGUE OUT
LONG IN
LONG OUT
MBB IN
MBB OUT
STRING OUT

3.1 Processing of the 'IN' Records: ANALOGUE, LONG, MBB

In Capfast terminology, the 'IN' record must be connected to a 'hardware-in' symbol. The DTYP property of the 'IN' record must be set to 'PMAC-VME ASCII'. An example of how the 'hardware-in' VAL property should be set is given below:
VAL: #C0 S0 @<param>

When the 'IN' record is processed, the expression <param> would be sent to the mailbox registers on PMAC card 0. The signal part (S0) of the VAL field is not used, but must be specified in order for the database to initialise correctly. The processing of the 'IN' record is asynchronous because it takes between 400ms and 1s (depending on the complexity of the command) to get an answer back. When PMAC fills the mailbox with an answer, the record processing continues and puts the returned value into the 'IN' record VAL field. Notice the 'STRING IN' record is not supported. Therefore, there is no way to get a string back into the EPICS database from the mailbox registers. Because of the asynchronous processing of these records and the delays introduced, one should keep the number of these records to a minimum during database design.

In a typical example, a binary-in record would be connected up to a hardware-in record of which the <param> property might be set to I100. This I-Variable controls the activation and deactivation of motor number 1. The ‘IN’ records do not support the ‘I/O Intr’ scan type. If the scan period were set to 10 seconds, then every 10 seconds the command ‘I100’ would be sent to the mailbox registers. This would cause the value of this I-Variable to be put into the ‘IN’ VAL field every 10 seconds.

3.2 Processing of the 'OUT' Records: ANALOGUE, LONG, MBB, STRING

In Capfast terminology, the 'OUT' record must be connected to a 'hardware-out' symbol. The DTYP property of the 'OUT' record must be set to 'PMAC-VME ASCII'. An example of how the 'hardware-out' VAL property should be set is given below:
VAL: #C0 S0 @<param>

Consider the following example of an MBBO record. The record has its various bits defined as follows:

Bit No.	Name	Value
0	MOVE	10
1	FAST	11
2	POSIT	12

Now assume that the 'hardware-out' VAL property is: #C0 S0 @&1B. In PMAC terminology, ‘&1B’ means ‘Make Coordinate System number 1 point to the Beginning...’.

When a value is written to the VAL field of the MBBO record, it will be processed. If VAL=0, the record will concatenate the value 10 to the end of '&1B' and send '&1B10' to the mailbox registers on PMAC card 0. This command says ‘Make Coordinate System number 1 point to the Beginning of Motion Program number 10’. Similarly, if VAL=2, the value sent would be '&1B12', this command says ‘Make Coordinate System number 1 point to the Beginning of Motion Program number 12’. Therefore, the MBBO record provides a convenient way of setting up several motion programs in an EPICS database. To actually run the motion program, once it has been setup, requires the command ‘R’ to be sent through the mailbox registers. The OUT records do not get any data back from the mailbox registers even if data is placed there by PMAC as a result of the command sent. A consequence of this is that it would not be possible to detect that an error had occurred because of the command sent. The signal part (S0) of the 'hardware-out' VAL field is not used but it must be specified in order for the database to build correctly.

The LONG and ANALOGUE records behave in the same way with respect to concatenating the 'hardware-out' <param> and record VAL fields together before sending the resultant to the PMAC mailbox registers. The STRING OUT record behaves differently. It does not concatenate these fields together. Thus, to send the 'close the loop' command with the STRING OUT record, one would do the following:

STRING OUT VAL = '#1J/'
hardware-out VAL: #C0 S0

4. DPRAM Module

4.1 DPRAM Memory Mapping

The memory mapping of the PMAC DPRAM in the EPICS driver is illustrated in figure 2 below:

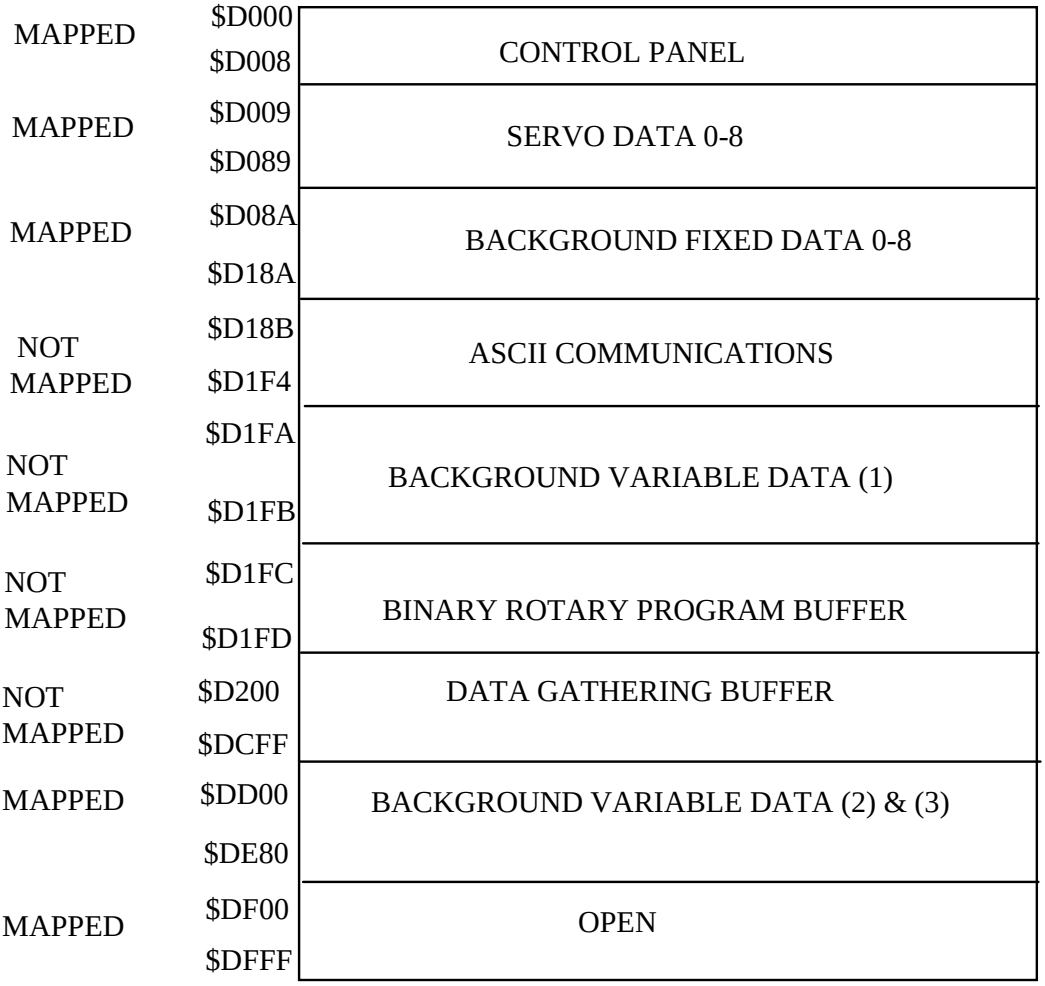

Figure 2: Memory Mapping of DPRAM in EPICS PMAC Driver

CONTROL PANEL: This region of memory allows the user to create the software equivalent of a hardware control panel. By setting and clearing individual bits in this region, the host computer can duplicate all of the functions available from external switches through the JPAN connector. Use of this region is supported by the driver.

SERVO DATA 0-8: This region of memory contains data such as the current position and error for a motor. The data is updated every I19 servo cycles. The number of motors copied is specified by I59. Setting I48=1 and issuing the GATHER command starts the copying. The data which are copied to this region are fixed by PMAC and this is not configurable. Reading from this area of DPRAM is supported by the driver.

BACKGROUND FIXED DATA 0-8: This region of memory is used by PMAC to transfer data of interest about each motor and coordinate system to the host computer. The 'data of interest' are fixed by PMAC and this is not configurable. The data in this region are updated in background as part of PMAC's "housekeeping" tasks. PMAC will only update these data if the previous data have been read (see section "Design of DPRAM Module"). These data are updated less than once per servo cycle. Copying of data is enabled if I49=1. Reading from this region is supported by the driver.

ASCII COMMUNICATIONS: This region of DPRAM can be used instead of the mailbox registers to communicate between the host computer and the PMAC. These communications are enabled by setting I58=1. Two separate buffers are used in the communication. A buffer of length 159 characters (including a NULL character to terminate the string) is used for Host-to-PMAC transfer. A buffer of length 255 characters (including a NULL character) is used for the returned PMAC-to-Host string. This method of communication is NOT supported by the driver.

BACKGROUND VARIABLE DATA (1): This is the header information which sets up the copying of up to 128 user-specified PMAC registers during the background cycle. The header information includes how to handshake and the location and size of the rest of the table. This region of memory is not mapped in the driver but background variable data copying is supported. This is because the driver writes appropriate header information setting the size of the table as 384 addresses and positions the table at $DD00. These numbers are hardcoded into the driver. The rest of the table is the region BACKGROUND VARIABLE DATA (2) & (3). Note this header information is at a fixed location in DPRAM.

BINARY ROTARY PROGRAM BUFFER: This buffer allows the host computer to send program commands to PMAC in binary format for the fastest possible transmission of these commands. The end address of this region, shown in figure 2 as $D1FD, is user configurable. The rotary buffer and its features are NOT supported by the driver.

DATA GATHERING BUFFER: PMAC's data gathering function can create a rotary buffer in DPRAM, so that the host computer can pick up the data as it is being gathered. This buffer must start at $D200. Its length is determined by the DEFINE GATHER {size} command. The driver does not have a data gathering buffer defined and does NOT support data gathering.

BACKGROUND VARIABLE DATA (2) & (3): The second and third parts of the background variable data buffer. The second part contains the address specifications of the PMAC registers to be copied into DPRAM. It occupies 2 16-bit words for each PMAC location to be copied. The third part contains the copied information from the PMAC registers. It contains 2 16-bit words for each X or Y PMAC location copied and 4 16-bit words for each long PMAC location copied. The data format is the same as for data gathering to DPRAM. Copying of data to this region is enabled when I55 is set to 1.

OPEN: The remaining region of memory. It is important that some DPRAM is left open. The user is free to assign M-variables to any addresses in this region.

4.2 Record Support for DPRAM

The DPRAM module can be called from the following records:

ANALOGUE IN
ANALOGUE OUT
BINARY IN
BINARY OUT
EVENT
LONG IN
LONG OUT
MBB IN
MBB OUT
STATUS

4.2.1 Processing of the 'IN' Records: ANALOGUE, BINARY, LONG, MBB

In Capfast terminology, the 'IN' record must be connected to a 'hardware-in' symbol. The DTYP property of the 'IN' record must be set to 'PMAC-VME DPRAM'. An example of how the 'hardware-in' VAL property should be set is given below:
VAL: #C0 S<offset> @<param>

Here, C0 refers to PMAC card number 0. <param> refers to the base memory address in DPRAM and <offset> the decimal number of bytes from that base address at which to read the value. An example would be: <offset> = 15, <param> = D:$D012. This would cause the record to read the value at: $D021 when processed. The leading D: refers to the type of the value being read, in this case a signed 48-bit integer (see table 1 for a complete list of types). The address space in DPRAM is 48 bits wide. This is broken down into two 24-bit words, prefixed with X: and Y: as shown in figure 3. Of the 24-bits, only the lower 16 are actually used for data. The upper 8 bits are filled with zeros.

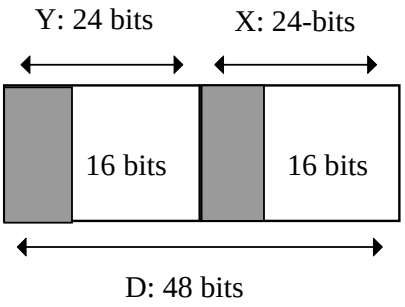

Figure 3 Bit Pattern of PMAC DPRAM

4.2.2 Design of the DPRAM Module

The 'IN' records, described above, are not truly interrupt driven. The driver contains 3 scan loops each running at 10Hz. If the scan mechanism chosen for the record is 'I/O Intr', one of these loops will put the value at the address given, into the record VAL field every 0.1s. Thus, it is not possible to see value changes that are occurring faster than this. The reason why the records are not interrupt driven is that, physically, there is no hardware signal which is raised when a new value is written into DPRAM. Therefore, the best that can be done is to scan the memory at some fixed rate. The rate of 10Hz is hardcoded into the driver, but this could be changed and the code recompiled if necessary. The 3 scan loops in the driver read different regions of DPRAM. These regions are: SERVO DATA, BACKGROUND FIXED DATA and BACKGROUND VARIABLE DATA.

In the case of SERVO DATA, the driver scan loop sets a controlling bit in memory which temporarily stops PMAC from updating this region. It then reads the data in the region and clears the control bit to allow PMAC to continue updating.

In the case of BACKGROUND FIXED DATA and BACKGROUND VARIABLE DATA, new data is only written to these regions by PMAC after a control bit has been cleared. When PMAC writes data to these regions, it sets the control bit. No new data will be written while that bit is set. The driver scan loop clears this bit. Therefore, PMAC will not update these regions faster than the driver scan loop is running.

To monitor the value of an I-variable, in the same way that the PMAC Executive 'watch' window performs, you would need to do the following:

- Set-up the address of the I-variable in the region BACKGROUND VARIABLE DATA (2).
- The value of the I-variable will be written to BACKGROUND VARIABLE DATA (3). This region is scanned at 10Hz by a driver loop as described above. An 'IN' record set to 'I/O Intr' will pick up these changes at that frequency. It is worth remembering that 1 motor typically uses 100 I-variables and the BACKGROUND VARIABLE DATA region is limited to a maximum of 128 PMAC registers. Therefore, one could only just monitor all the I-variables associated with 1 motor!

4.2.3 Processing of the 'OUT' Records: ANALOGUE, BINARY, LONG, MBB

In Capfast terminology, the 'OUT' record must be connected to a 'hardware-out' symbol. The DTYP property of the 'OUT' record must be set to 'PMAC-VME DPRAM'. An example of how the 'hardware-out' VAL property should be set is given below:
VAL: #C0 S<offset> @<param>

Here, C0 refers to PMAC card number 0. <param> refers to the base memory address in DPRAM and <offset> the decimal number of bytes from that base address at which to write the value. An example would be: <offset> = 15, <param> = DP:$DF00. This would cause the record to write data at: $DF0F when processed. The leading DP: refers to the type of the value being written, in this case a signed 32-bit integer. If a file had previously been downloaded which contained the mapping, M900->DP:$DF0F, processing the record would have the effect of updating M900. Notice that the address used for assignment of the M-Variable resides in the OPEN region of DPRAM.

4.2.4 PMAC Driver Data Types

The PMAC DPRAM is complicated by the fact that it supports many different data types. A given memory location in DPRAM will have a particular data type associated with it. To support the raw PMAC data types and to make the handling of the data easier, several further data types have been added within the PMAC driver code. Conversion routines between the different data types are also to be found within the driver. Table 1 lists the supported data types.

DATA PREFIX	DATA TYPE	REGION OF DPRAM	ORIGIN
X	Unsigned 24-bit Integer	Servo Data	Raw PMAC
Y	Unsigned 24-bit Integer	Servo Data, Back Fixed	Raw PMAC
SX	Signed 24-bit Integer	Servo Data	Device Driver
SY	Signed 24-bit Integer	Back Fixed	Device Driver
HX	Unsigned 16-bit Integer	Control Panel, Servo Data, Back Fixed	Device Driver
HY	Unsigned 16-bit Integer	Control Panel, Servo Data, Back Fixed	Device Driver
D	Signed 48-bit Integer	Servo Data, Back Fixed	Raw PMAC
L	48-bit Floating Point	Back Fixed	Raw PMAC
DP	Signed 32-bit Integer	Open	Raw PMAC
F	32-bit Floating Point	Open	Raw PMAC

Table 1: Data Types associated with different regions of PMAC DPRAM

The region BACKGROUND VARIABLE DATA is not shown in table 1. This is because the type of the data written to this area depends on the memory location that it comes from. Since the user can choose to record data from any location in this region, any of the above data types could apply.

5. PMAC Configuration

The PMAC Executive program and the EPICS device driver are mutually exclusive. They cannot be used at the same time. The reason for this is that they require different settings for I3 and I9. If the values of these variables are changed while the driver is running, it will hang. The suggested procedure before loading a database which uses the PMAC device driver is to first configure the card using the PMAC Executive program. The PMAC Executive window should be **closed** before using EPICS.

5.1 Card Configuration

The PMAC Executive program should be used to map the DPRAM in VME memory space. It should also be used to map the mailbox registers and to set-up the interrupts which drive communication through these registers. The interrupt vector for the PMAC card **must** start on an even address. It occupies three addresses.

$$$*** is the PMAC global reset command. One of the effects of sending this command is to reset all the I-variables. It does not reset M-variables.

The following I-variables settings are required for the device driver to operate correctly. These should be set with the Executive program:
I1 = 0
I3 = 2
I4 = 0
I6 = 1
I9 = 0

6. Data Gathering

As can be seen from figure 2, the DATA GATHERING BUFFER is not mapped into the device driver memory. This means that the driver does not support the data gathering function available through PMAC. Even though it will be possible to send commands to PMAC to put it in data collection mode, these data cannot be read through the device driver as it currently stands. Since data can be logged on

every servo cycle, which could running at 2kHz, there is a question as to whether EPICS record processing would be fast enough to support reading these data back. A future version of the driver may support the use of Waveform records to read these data from DPRAM.

Data gathering on the PMAC allows a maximum of 24 addresses to be logged. Each address is fixed by PMAC at either 24 bits or 48 bits. Therefore, choosing different memory locations to log will automatically mean choosing different size data types. The layout of the recorded data will vary according to user selection of what is being logged. This makes the support of data gathering difficult to implement. Figure 4 shows how 24-bit data and 48-bit data will be stored in DPRAM.

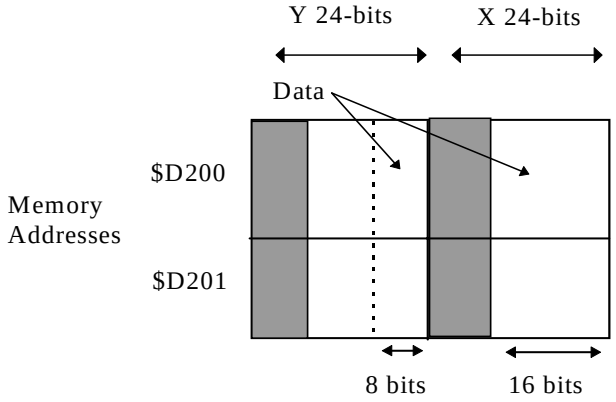

Figure 4 Storage of 24-bit and 48-bit Data in DPRAM

In figure 4, a 24-bit value is stored at memory location $D200. The lower 16 bits of the value are stored in the X region of the address and the remaining 8 bits of the value are stored in the Y region. If the data value had been 48-bits instead, the bottom 24-bits would be stored at $D201 while the upper 24-bits would be stored at $D200. At each address, the bit pattern would be as for a single 24-bit value. This means that bits 8-15 of the Y region are not used. They are filled with zeros. Note that bits 16-23 of X and Y are never used for data storage.

An alternative to reading the gathered data through EPICS records would be to use a VxWorks task. This would involve reading the raw memory addresses at some fixed rate. This rate would depend on the servo cycle and the size of the memory cache where these data would be held. At some appropriate time, the cache would be flushed to a hard disk file.

7. Example Databases which use the PMAC Driver

7.1 Use of the 'Status' record

Several example databases are supplied with the PMAC device driver. They make use of the 'status' record. This record allows easy monitoring of status bits associated with each motor being controlled. The 'status' record allows 32 bits to be monitored. There are 32 associated output links. When a bit changes state, the associated output link is triggered.

7.2 The 'startup' script

The startup script performs several functions:

- It loads a library which contains all the compiled source code related to the driver. The library is called 'pmacLib.o'. It contains the driver support, the device support and the record support code.
- It loads the PMAC databases, using 'dbLoadRecords'. This function reads '*.db' files and is able to perform macro substitution when the database is loaded.
- It runs the program 'pmacVmeConfig(1, 2, 3, 4, 5)'. Here argument 1 is the card number, 2 is the base address of the PMAC card, 3 is the DPRAM address, 4 is the middle address of the interrupt vector and 5 is the interrupt level. If the card has no DPRAM, argument 3 should be zero.
- It runs the program 'pmacDrvConfig(1, 2, 3, 4, 5)'. Here argument 1 is the card number. Arguments 2, 3 and 4 are meant to be the scan rates of the driver loops which update the SERVO DATA, BACKGROUND FIXED and BACKGROUND VARIABLE regions of DPRAM.

However, these arguments are not currently supported. The frequencies of the scan loops can only be changed through a recompilation of the code. Argument 5 is a flag which enables and disables the use of the mailbox registers. The default setting is enabled.

7.3 Typical Design of PMAC Databases

The design of an EPICS application which uses PMAC can be broken down neatly into a series of separate databases. In a typical motor control application, there will be separate databases communicating with the different regions of PMAC memory. For example, one database would define all the I-Variables of interest. This talks through the mailbox registers. A second database would communicate with the SERVO DATA region of DPRAM. In this database, the status of each motor could be kept. A third database would talk to the BACKGROUND FIXED region of DPRAM. Again, this could be used to flag status bits from amplifiers and coordinate systems.

7.4 Mailbox Registers

Consider a ‘binary-in’ record with DTYP set to PMAC-VME ASCII and processing every 10 seconds. If this were connected to a ‘hardware-in’ symbol with VAL: #C0 S0 @I100, then every 10 seconds, the command 'I100' would be sent to PMAC, causing it to report the value of this variable back to the mailbox. The value returned would then be put into the ‘binary-in’ VAL field. I100 reports whether motor 1 is 'activated' or 'de-activated'.

7.5 Servo Data Region of DPRAM

An example database designed to communicate with this area of DPRAM might contain a ‘status’ record linked to several ‘binary-in’ records. The ‘status’ record DTYP field would be set to PMAC-VME DPRAM. If the ‘status’ input is from a ‘hardware-in’ symbol with VAL: #C0 S0 @X:$D00B and the SCAN field is set to ‘I/O Intr’, the driver will read that address every 0.1 seconds and cause the ‘status’ record to process. This address corresponds to the place in the SERVO DATA region where global status bits are copied to. They are copied from elsewhere in PMAC’s memory by the PMAC processor. These bits convey information such as:
Data Gathering Function On
Data Gather to Start on Servo
Data Gather to Start on Trigger
Stimulus Table Entered
In the database, separate ‘binary-ins’ would be connected to the output links of the ‘status’ record. Each ‘binary-in’ would correspond to one of the above pieces of information and this could be displayed very easily by the EPICS display manager.
The database might also contain 4 ‘analogue-in’ records. These could be set to process on ‘I/O Intr’. Table 2 shows the setting of the 'hardware-in' VAL property and the data which would be retrieved every 0.1 seconds, if this were the case.

Hardware-In VAL	Data Retrieved
#C0 S15 @D:$D012	Motor 2 Commanded Position
#C0 S30 @D:$D014	Motor 3 Actual Position
#C0 S45 @D:$D016	Motor 4 Master Position
#C0 S60 @SX:$D01C	Motor 5 Actual Velocity

Table 2: Hardware-In VAL Settings and Data Retrieved from DPRAM

7.6 Background Fixed Data Region of DPRAM

A similarly designed database, using 'Analogue-In' records and 'status' records could be used to read data from this region of DPRAM. In particular, an 'Analogue-In' connected to a 'hardware-in' which has its VAL property set as: #C0 S31 @D:$D093, would retrieve the 'Target Position for Motor 2'. A 'status' record connected to a 'hardware-in' whose VAL property was set as: #C0 S62 @Y:$D097 would read the status word for motor 3. The bits in this word correspond to the following information:
In-position true
Warning following error limit exceeded
Fatal following error limit exceeded
Amplifier fault error
Stopped on position limit

Amplifier enabled