USER MANUAL

PMAC DUAL-PORTED RAM

8K x 16 bit Dual-Ported RAM

3Ax-00PMAC-xUxx

June 8, 2004

DELTA TAU
Data Systems, Inc.

NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.
This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

DUAL-PORTED RAM COMMUNICATIONS....1
Uses of DPRAM....1
Setting up the PC Bus Dual-Ported RAM (Option 2)....1
Software Setup....2
System Memory Map (Most Compatible)....3
Starting Address....3
Setting up the VME Dual-Ported RAM (Option 2V)....4
Starting Address....4
Using the Dual Ported RAM (PC and VME)....6

DUAL-PORTED RAM AUTOMATIC FUNCTIONS....11
DPRAM Data Format....11
Control Panel Function....12
General Description....12
Register Addresses....13
Control Panel Request Words....13
Bit Format of Request Words....13
Servo Fixed Data Buffer....14
General Description....14
Register Map....15
Background Fixed Data Buffer....17
General Description....17
Register Map....17
Background Variable Data Read Buffer....21
General Description....22
Enabling....22
Single User Mode Procedure....22
Multi-User Mode Procedure....22
Data Format....23
Disabling....23
Register Map....23
Background Variable Data Write Buffer....24
General Description....24
Enabling....24
Procedure....24
Data Format....25
Disabling....25
Register Map....25
DPRAM Data Gathering Buffer....25
DPRAM ASCII Communications....27
General Description....27
Read/Write Procedure....27
Interrupts....28
Register Map....29
Binary Rotary Program Buffers....29
General Description....30
Register Map....31
Binary Command Structure....31
Internal PMAC 48-Bit Command Format....32

DUAL-PORTED RAM COMMUNICATIONS

PMAC's Option 2 provides an 8K x 16 bit dual-ported RAM that allows PMAC and its host to share an area of fast memory. For the PMAC PC and the PMAC Lite, Option 2 is a separate board that sits on the PC bus and cables to PMAC. For the PMAC VME, Option 2(V) consists of ICs added to the main board itself. Option 2 is not available for the PMAC STD. For all versions of the PMAC2, Option 2 consists of ICs added to the main board itself. The dual-ported RAM can be used for extremely fast communication of data and commands to and from PMAC.

Uses of DPRAM

The typical use in writing to PMAC is for a very fast repetitive downloading of position data and/or rotary program information in real time. The typical use in reading from PMAC is getting very fast status information repetitively. PMAC supports both automatic functions for the DPRAM communications, and user-designated functions.

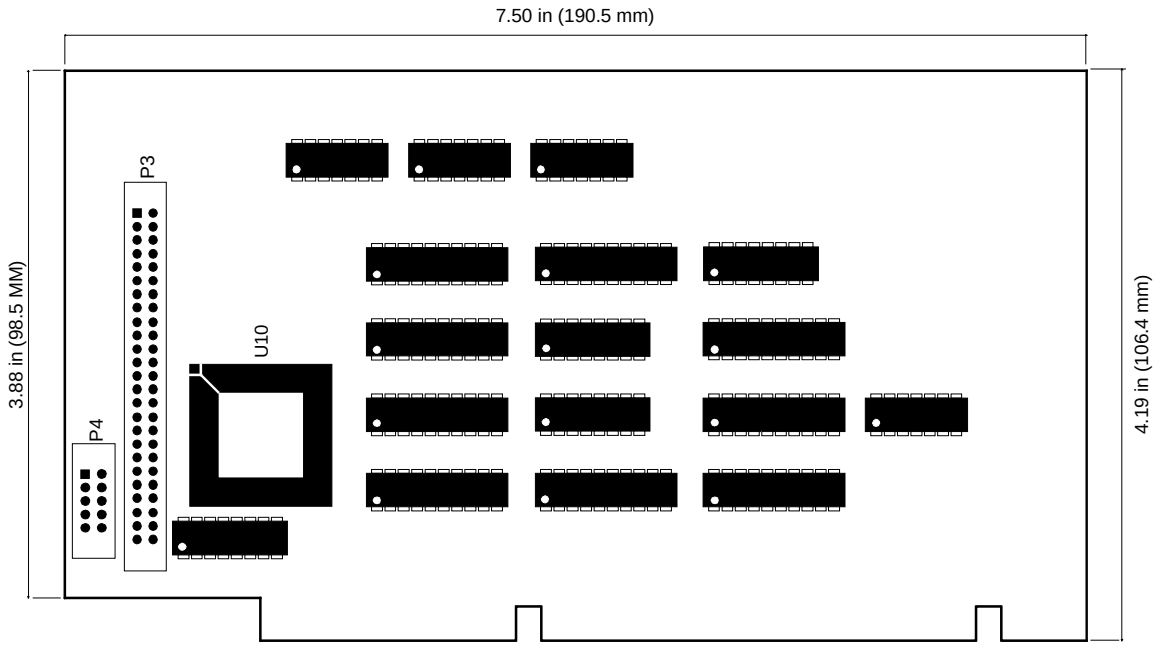
Data such as motor status, position, velocity, following error, etc. can be continuously updated and written to DPRAM by PLC programs or automatically by PMAC. Without using DPRAM, this data must be accessed by sending PMAC on-line commands, such as **?**, **P**, **V**, and **F**, through the VME mailbox registers or over the PC bus. This same data may be obtained much faster via the DPRAM without the time required to send the command through the communications port and wait for the response.

For non-automatic uses, PMAC usually accesses the dual-ported RAM through M-variables that are addressed to these locations. This can work for reading or writing. The M-variable formats likely to be used are X:, Y: (for 1 to 16 bits), DP: (for 32-bit fixed point), and F: (32-bit floating point). For sending data back to the host, PMAC's data gathering function can also be used, directed to the dual-ported RAM rather than the regular RAM (I45 controls). See the Analysis Features section of the PMAC manual for details.

Setting up the PC Bus Dual-Ported RAM (Option 2)

There is no hardware setup or connections for the Option 2 DPRAM on the PMAC2-PC or PMAC2-Lite. It is factory installed on the PMAC2 itself. For the PMAC PC bus cards, the DPRAM card plugs into an available PC-slot to the right side of your PMAC PC or Lite card. The PMAC and DPRAM card together occupy 2 PC slots. The two short cables provided connect the DPRAM to the CPU on the PMAC. The 50-pin cable connects P3 of DPRAM to J2 of PMAC (piggy-back board) or J9 of PMAC Lite. The 10-pin cable connects P4 of DPRAM to J4 of PMAC (piggy-back board) or J10 of PMAC Lite.

PMAC-PC OPTION 2
DUAL-PORTED RAM INTERFACE BOARD



To install the board, connect the cables as previously stated with the boards removed from your computer. Next, install both boards, plugging in PMAC first and then the DPRAM board in the slot next to it. Turn on your computer and establish communication with PMAC using the PMAC Executive software. At the time of this manual revision, the easiest method of configuring the Dual-Ported RAM is using the Executive Program.

There are only two hardware configurations (set with jumper E1) for the DPRAM card. Jumper E1 controls whether the DPRAM uses 8 or 16-bit access. E1 should be removed for 8-bit access in a 16-bit PC bus (AT) slot. Install E1 for 16-bit operation (faster data transfers). All other DPRAM configurations are done entirely through software programming via PMAC.

Software Setup

The software setup is simply a matter of setting a few register values in PMAC through the normal bus communications port, and saving these values to non-volatile memory. This should allow the host computer to have direct access to this memory in an open area of its memory space. Once the setup has been made successfully, the DPRAM registers can be accessed from the PC side with pointer variables.

The configuration of the DPRAM is done entirely through software. Configuring consists of selecting an 16K block (byte addressing) of unused memory space in your PC. This is not necessarily a trivial task. Depending on the computer and other accessory cards you may have (such as network cards, graphic cards, etc. that use conventional memory space and not I/O memory space), the available memory space location may vary. Think of it as adding a standard memory card to the computer where an address location must be selected in the memory map that no other device in the computer uses.

If you are unsure about the locations available for DPRAM to use, there is menu option Configure|PC Dual-Ported RAM under the Configure menu in the PMAC Executive Program which can help find an available block of memory space. Refer to the PMAC Executive Program Manual for details on how to do this. In most cases, the available memory space will be found between locations $B0000 and $EC000. Usually, a system memory map of the PC which is standard for most PCs and compatible is found below.

System Memory Map (Most Compatible)

Decimal	Hex	Function
0	0000	
16K	04000	
32K	08000	
48K	0C000	Standard 640K RAM
64K	10000	
.	.	For DOS, Programs, etc.
.	.	
.	.	
624K	9C000	
640K	A0000	
.	.	Usually VGA/EGA Space
688K	AC000	
704K	B0000	
.	.	Recommended memory space
.	.	Monochrome may start at B0000
944K	EC000	
960K	F0000	
.	.	System BIOS & ROM
1008K	FC000	

Starting Address

Once you have chosen a used block of memory space, you must now configure the DPRAM to start at that address. To do this, you must write the segment value (eight highest order bits) of your chosen start address into locations $786 and $787 of PMAC's X memory (see I/O Memory Map section of PMAC's Users Manual). These are special memory locations in PMAC which tell PMAC where in PC memory to locate the DPRAM. In addition, you also specify using the 20 or 24-bit DPRAM addressing mode. The 24-bit addressing mode locates the DPRAM above the 1 megabyte range (locations above $FFFFF). (If using an 8086 or an 8088 microprocessor (IBM PC/XT), only use 20-bit addressing and it is limited to configuring the DPRAM for locations $FC000 and under). In most cases, no matter which microprocessor being used (AT, XT, or whatever), locating the DPRAM between $B0000 and $EC000 will be sufficient.

Note:

Most will simply use the Configure|PC Dual-Ported Ram window in the PMAC Executive program to set the address for DPRAM. The following section explains how to perform this setup without the Executive program.

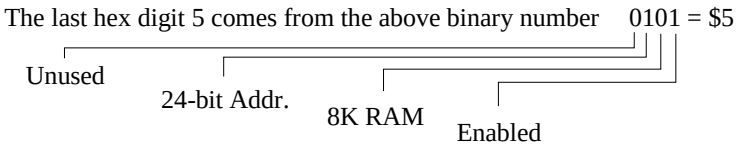
The PMAC Executive program Configure|PC Dual-Ported RAM screen specifies PC memory addresses in terms of memory segments. Each memory segment starts 16 addresses from the pervious segment, so the segment address is like the memory address without the hex digit (i.e. memory address $D4000 is segment address $D400).

Bits 0-3 (first four least significant bits) of PMAC location X:$787 as follows:

Bit 0: DPRAM enable/disable (1=enable, 0=disable)
Bit 1: 8K RAM (0=8K)
Bit 2: 24-bit/20 bit addressing enable (1=24 bit, 0=20 bit)

Example:
To configure the DPRAM to start at $EC000 in PC memory. There is 8K of DPRAM and 24-bit addressing is needed. To configure and enable the DPRAM, either use the PMAC Executive program's dual-port RAM configuration feature or send the following commands directly to PMAC.

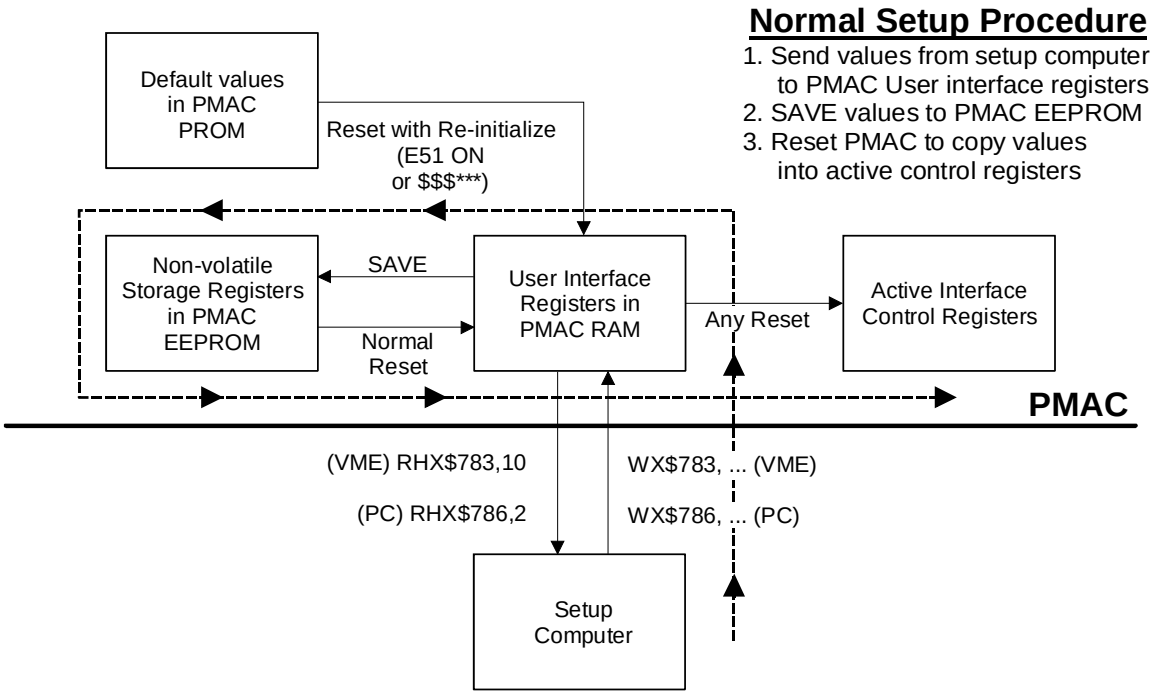
```
WX:$786,$0000E,$000C5      ;configure DPRAM for $EC000
save                       ;save configuration to EAROM
$$$                        ;re-initialize the card
```



To disable the DPRAM, you could use the PMAC Executive program's dual-port RAM configuration feature or send the following commands directly to PMAC.

```
WX:$786,$0000E,$000C4      ;disable DPRAM
save                       ;save configuration to EAROM
$$$                        ;re-initialize the card
```

VME and/or DPRAM Address Setup



Setting up the VME Dual-Ported RAM (Option 2V)

There is no hardware setup or connections for the Option 2V DPRAM on the PMAC VME or PMAC2-VME. It is factory installed on the PMAC board itself.

Starting Address

Before you can communicate with PMAC VME through the DPRAM, you must setup its starting address. Initial communications to the VME for setup of the address of the VME mailbox registers is done over the serial port. The setup of the DPRAM address can be done either with or after the mailbox setup (see Writing a Host Communications Program in the PMAC User Manual).

Before choosing the DPRAM starting address, determine what memory space is available in the VME system (so that PMAC's DPRAM does not interfere with existing RAM or other devices on the VME bus). Just like setting up the base address of PMAC VME, the starting address of DPRAM is done through software, but in a somewhat different manner. The best way to describe how to set up DPRAM is to give an example.

Note:

Most users will simply use the Configure|VME Communications window in the PMAC Executive program to set the address for DPRAM. The following section explains how to perform this setup without the Executive program.

Example:
Suppose a starting address of $1FC000 was selected for the DPRAM. Just as we did for the base address of PMAC, it is best to rewrite this address in binary and label the address bits, starting with A0 as the rightmost bit:

Address Bit	A31	A24	A23	A16	A15	A8	A7	A0
Binary	0000	0000	0001	1111	1100	0000	0000	0000
Hex	00		1F		C0		00	

Clearly, we have a value of $00 for address bits A31 - A24, $1F for address bits A23 - A16, $C0 for bits A15 - A8 and $00 for bits A7 - A0. To tell PMAC where we want DPRAM to begin, we need to break up this starting address into two parts:

1. The first part will represent the value address bits A23 through A20. This value will only be written to PMAC once during the setup process and saved.
2. The second part will represent the value address bits A19 through A14. This value will need to be written to PMAC each time it is powered up or reset.

If using 32-bit addressing, address bits A31 through A24 for the dual-ported RAM are determined by PMAC's memory location X:$0785, which is also used for the same address bits of the base address of the mailbox registers.

First we have to write the value of address bits A23 through A20 into bits 7 through 4 (high order nibble) of PMAC's memory location X:$078A, i.e. the left hex digit (most significant bits) will be the value of address bits A23 - A20, and the other digit (right digit & least significant bits) will always be $0. In this example, the value for address bits A23 - A20 would be $1, therefore we would write a value of *$10* into location X:$078A.

Now we have to determine the value of address bits A19 through A14. Every time PMAC is powered up or reset (either with the hardware reset line or use of the **$$$** command) we will need to write this value into PMAC's base address + $121 (remember, from our previous examples, PMAC's base address is $7FA000).

In this example, constructing a 6-bit hex number from bits A19 - A14 gives us a value of $3F:

Address Bits	**A19**	**A18**	**A17**	**A16**	**A15**	**A14**
Binary						
Hex						

Therefore, we write $3F from the VME host computer (master) into VME bus location $7FA121 after PMAC is powered up or reset.

Note:

This dynamic addressing scheme provides the capability for addressing up to 1M byte of DPRAM in 16K byte blocks, by changing the value of base + $121 on the fly. However, PMAC VME currently utilizes only a single 16 Kbyte block (8K x 16), so the base + $121 register only has to be written to once every time PMAC is powered up or reset.

At this point, the starting address of DPRAM is fully specified. However, we need to check two more register locations in PMAC's memory for having appropriate values. Location X:$078B must have a value of $E0 to enable the DPRAM chip installed on the PMAC VME and we must modify the value in location X:$078C (the address width register) by adding $80 to the existing value. For this example, our PMAC register values would be:

PMAC Address	Value
X:$0783	$39
X:$0784	$04
X:$0785	$00
X:$0786	$7F
X:$0787	$A0
X:$0788	$02
X:$0789	$A1
X:$078A	$10
X:$078B	$E0
X:$078C	$90

The shaded registers above contain the values we changed (from example 2.0) to enable DPRAM. A simple write command followed by a **SAVE** command to PMAC will put these values into their appropriate registers and make them permanent:

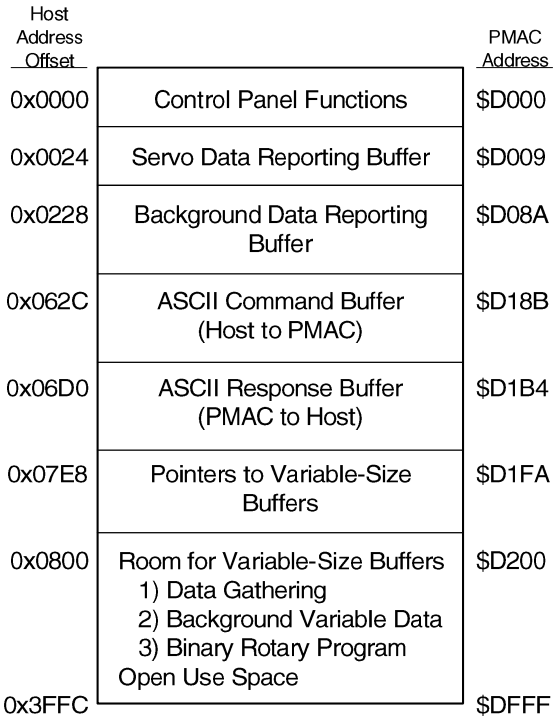
```
WX$0783,$39,$4,$0,$7F,$A0,$02,$A1,$10,$E0,$90
SAVE
```

Remember that these values must be saved with the **SAVE** command and then the card reset (with the **$$$** command, the INIT/ input line pulled low, or power cycled) before these new values will take effect. After this, writing $3F to $7FA121 (base + $121) will allow us to start using dual ported RAM.

Using the Dual Ported RAM (PC and VME)

The mapping of memory addresses between the host computer on one side, and PMAC's address space on the other side, is quite simple. Using this memory is a matter of matching the addresses on both sides. To PMAC, DPRAM simply appears as extra memory in the range $D000 to $DFFF, which can be thought of as 4K of double (48-bit) words or 8K of single (24-bit) words, in both X and Y memory (remember, X and Y memory in PMAC are 24-bit locations.).

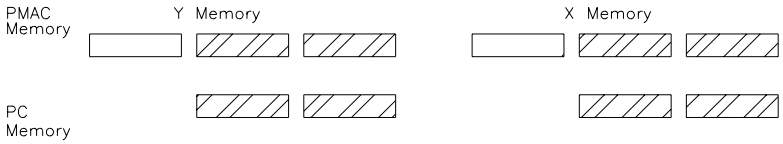
PMAC Dual-Ported RAM Memory Map



Note:

DPRAM only occupies the low 16 bits (0-15) of each 24-bit word in PMAC.

PMAC memory locations $D000 to $D1FF are reserved for fixed uses. The range of $D200 to $DFFF is open for general-purpose use. From the PMAC side, locations in DPRAM are typically accessed using M-variables. An M-variable is defined to each word to be used (such as **`M120->X:$D200,0,16,S`**), then these M-variables are used in programs. They can either be written to (e.g. **`M120=P1+P5`**) or read from (e.g., **`X(M120)`** or **`P1=M120`**).



To the host, DPRAM appears as 8K x 16-bit words of memory. Since most computers address by byte, this requires 16K of address space, or 14 bits (2^{14} = 16K) on the host bus. Consecutive 16-bit locations of DPRAM are located at *even* addresses. That is, address bit A0 (the least significant bit of your DPRAM address) should always be 0. Typically, these DPRAM locations are accessed through host word read and word write commands.

When a location in DPRAM is accessed, either reading or writing, address bit A1 selects whether it is reading or writing to PMAC's Y or X memory. If address bit A1 = 0, you are selecting the equivalent of PMAC's Y memory. Conversely, if address bit A1 = 1, selecting PMAC's X memory. The following table shows the DPRAM addresses and the corresponding host address from our previous example:

PMAC Address	Host Address
Y:$D000	$1FC000
X:$D000	$1FC002
Y:$D001	$1FC004
X:$D001	$1FC006
Y:$D200	$1FC800
X:$D200	$1FC802
....	....
Y:$DFFF	$1FFFFC
X:$DFFF	$1FFFFE

The following two equations can be helpful to help calculate both PMAC and host addresses for DPRAM:

Host address = *Host start address* + 4 x (*PMAC address* - $D000) + *offset*

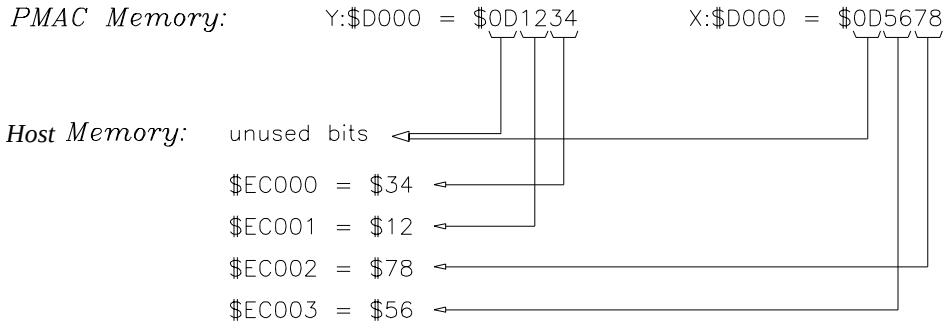
where: *offset* = 0 for accessing Y memory, and *offset* = 2 for accessing X memory. In our example, *Host start address* = $1FC000. In converting the other way, we have:

PMAC address = 0.25 x (*Host address* - *Host start address*) + $D000

Host DPRAM address bits A2 through A13 select which PMAC word is to be accessed. The value of these bits equals the PMAC equivalent address minus $D000.

Read/write Example:
Suppose the DPRAM was configured to begin at host location $EC000 and you write to PMAC

```
WY$D000, $1234
WX$D000, $5678
```



Writing to the DPRAM on the host side and reading it through PMAC is just as easy. The alignment of the numbers work the same way as illustrated above.

The host application program and the PMAC motion control and/or PLC programs may be written to allow a wide variety of control and data transfer capabilities. While certain DPRAM addresses are reserved as mentioned above, the host may set certain addresses to trigger an operation in the PMAC, the PMAC may set certain addresses to trigger an operation in the host, etc.

Programming Examples:
Example #1: Suppose a host program is written in C to read motor #1's actual position. Use the DPRAM automatic features to place this data in DPRAM or use a PMAC PLC program. To use a PLC program, define two M-variables and a one line PLC program which constantly updates a location in DPRAM. The C-program can then read this value, which is a 32-bit integer. In PMAC, enter the following M-variable definitions and PLC program.

```
M130->D:$28                    ;motor #1 actual position
M131->DP:$D000                 ;point to a location in DPRAM
OPEN PLC 1 CLEAR
M131=M130/(I108*32)            ;update motor position (in counts) in DPRAM
CLOSE
I5=2                           ; allow PLC 1 to be enabled
ENABLE PLC 1                   ; "turn on" PLC 1
```

Now, read the location corresponding to M131 ($D000 in PMAC's memory) in PC memory and extract this position information so that the C program can use it. The corresponding address in PC memory can easily be determined by talking the address location in PMAC's memory minus the starting address (**always $D000**), multiply by 4 (because the X and Y words for each location in PMAC's DPRAM space is equivalent to four bytes or locations of PC memory) and add it to the PC memory starting address selected for the DPRAM.

PC Location = ((PMAC Mem. Location (hex)- $D000) × 4) + (PC Mem Starting Address selected for DPRAM)

Assume we configured DPRAM to start at PC address $D4000. We want the equivalent PC address location for PMAC location $D200. This would be:

Hex: ($D200 -$D000) × 4 + $D4000 = $D4800

Decimal: (53760 - 53248)× 4 + 868352 = 870400

Our C code now only has to PEEK location $D4800 and $D4802 (since we need to get four bytes to get a 32 bit word) to obtain motor #1's current position. The C code subroutine may look like this.

```
long get_motor_position (){
      int         hi_word;
      unsigned    lo_word;
      lo_word = peek (0xD400, 0x0800);
      hi_word = peek (0xD400, 0x0802);
      return (hi_word *65536 +lo_word);
```

Example #2: To calculate positions for three axes in a custom C program and have PMAC move to these positions. Assign three M-variables to point to three DPRAM locations, write position values to these locations with the C program, and have a PMAC motion program use these variables in its move statements. In the sample code below, a flag bit has been used to signal the C program when the data has been taken by PMAC so the next move locations can be loaded.

```
M101->DP:$D201                 ; used for motor x's commanded position
M102->DP:$D202                 ; used for motor x's commanded position
M103->DP:$D100                 ; used for motor x's commanded position
M100->DP:$D200,0               ; go flag to stop motion program
M104->DP:$D200,1               ; flag to stop C program when positions have
                               ; been received
OPEN PROG 1 CLEAR
TA10
WHILE(M100=1)                  ;keep looking while go flag is 1
   X(M101) Y(M102) Z(M103)     ;move x,y,z to new positions
ENDWHILE
CLOSE
```

The C program calculates the positions, and would send them to PMAC via a subroutine like this:

```
void update_positions (longx, long y, long z, int stop_program)
{
    static long far *x, *y, *z;
    x = MK_FP (0xD400, 0x0804);
    y = MK_FP (0xD400, 0x0808);
    z = MK_FP (0xD400, 0x080C);
```

```
    while (peek (0xD400, 0x0801) == 1); /* wait until position received */
    *x = X;
    *y = Y;
    *z = Z;
    poke (0xD400, 0x0801,1);              /*send position flag
}
```

DUAL-PORTED RAM AUTOMATIC FUNCTIONS

PMAC provides many facilities for using the dual-ported RAM (DPRAM) to pass information back and forth between the host computer and PMAC. These facilities are comprised of the following functions:

- DPRAM Control Panel Function (Host to PMAC)
- DPRAM Servo Data Reporting Function (PMAC to Host)
- DPRAM Background Fixed Data Reporting Function (PMAC to Host)
- DPRAM Background Variable Data Reporting Function (PMAC to Host)
- DPRAM ASCII Communications Buffer (Bi-directional)
- DPRAM Binary Rotary Program Buffer (Host to PMAC)
- DPRAM Data Gathering Buffer (PMAC to Host -- already existing)
- DPRAM **<CONTROL-W>** ASCII Command Function (Host to PMAC -- superseded by DPRAM ASCII Communications)

In addition to these automatic functions, the user is free to access otherwise unused registers in the DPRAM through the use of M-variables on the PMAC side, and through pointer variables on the host side, for sending data either way between the host and PMAC.

Each of the automatic functions is described in detail below.

Note:

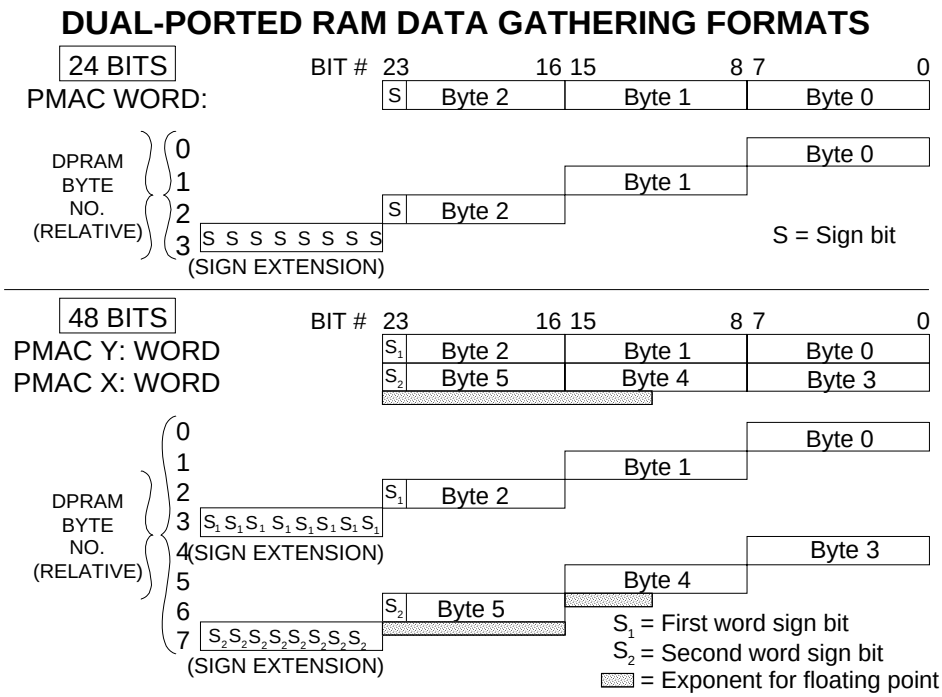
In listing memory locations, both the host memory address and the PMAC memory address will be given. The host address is given as an offset from the base address of the DPRAM in the host memory space; it is expressed in standard C-language hexadecimal notation (e.g. 0x1F80). The PMAC address is given in parentheses as an absolute location in PMAC memory; it is expressed in the standard PMAC hexadecimal notation (e.g. X:$D008)

DPRAM Data Format

Long-format data is stored in the DPRAM in 32-bit sign-extended form. That is, each short (24-bit) from PMAC is sign-extended and stored in 32 bits of DPRAM. The most significant byte is all ones or all zeros, matching bit 23. Each long (48-bit) word is treated as 2 24-bit words, with each short word sign-extended to 32 bits. The host computer must re-assemble these words into a single value. The data appears in the DPRAM in Intel format: the less significant bytes and words appear in the lower-numbered addresses.

To reassemble a long fixed-point word in the host, take the less significant 32-bit word, and mask out the sign extension (top eight bits). In C, this operation could be done with a bit-by- bit AND: (LSW and 16777215). Treat this result as an unsigned integer. Next, take the more significant word and multiply it by 16,777,216. Finally, add the two intermediate results together.

To reassemble a long floating-point word in the host, treat the less significant word the same as for the fixed-point case above. Take the bottom 12 bits of the more significant word (MSW and 4095), multiply by 16,777,216 and add to the masked less significant word. This forms the mantissa of the floating-point value. Now take the next 12 bits (MSW and 16773120) of the more significant word. This is the exponent to the power of two, which can be combined with the mantissa to form the complete value.



Control Panel Function

PMAC provides the capability to create the software equivalent of a hardware control panel in the DPRAM. By setting and clearing individual bits in the DPRAM, the host computer can duplicate all of the functions available from external switches through the JPAN connector. Also, by writing a 16-bit value, the host can control the feedrate override (% value) just as a real potentiometer would.

The DPRAM control panel functions are enabled if I2 is set to 3. They are disabled if I3 is set to other values.

General Description

The host loads command request bits for motors and coordinate systems into registers 0x0004 (Y:$D001), 0x0008 (Y:$D002), etc., then triggers the action by setting the appropriate enable bits of register 0x0000 (Y:$D000). When PMAC processes the request, it clears all of the bits of 0x0000 (Y:$D000). The DPRAM is now ready for the next set of requests.

The host loads the desired "Feedrate Override" (time-base) values for the coordinate systems it wishes to change into registers 0x0006 (X:$D001), 0x000A (X:$D002), etc., then triggers the action by setting the appropriate enable bits of register 0x0002 (X:$D000). The register holds values from 0 to 32,767, in units of 1/32,768 msec. The value represents the "elapsed time" PMAC uses in its trajectory update equations each servo cycle. If it matches the true time, the trajectories will go at the programmed speeds. If it is greater than the true time, trajectories will go faster; if it is less, they will go slower. This value corresponds to values of 0 to 8,388,352 in units of I10 (1/8,388,608 msec). At the default I10 value of 3,713,707, this corresponds to override (%) values from 0 to 225.87; for real-time execution (%100) a value of 14,507 should be used.

PMAC copies these values, multiplied by 256, into the command override register for the appropriate coordinate system (e.g. X:$0806 for C.S.1). If Ix93 -- Time Base Source Address -- for the coordinate system contains the address of this register, then the value is used as the override for the coordinate system. PMAC does not clear the enable bits in register 0x0002 (X:$D000); as long as the enable bit for a coordinate system stays set, PMAC will use the override values provided by the host.

Register Addresses

Address	Description
0x0000 (Y:$D000)	Control Panel Motor/Coordinate System Enable Mask (Set bit to enable; PMAC clears on taking action)
	Bit 0: Motor/Coordinate System # 1
	Bit 1: Motor/Coordinate System # 2
	Bit 2: Motor/Coordinate System # 3
	Bit 3: Motor/Coordinate System # 4
	Bit 4: Motor/Coordinate System # 5
	Bit 5: Motor/Coordinate System # 6
	Bit 6: Motor/Coordinate System # 7
	Bit 7: Motor/Coordinate System # 8
	Bits 8-15: Not Used
0x0002 (X:$D000)	Coordinate System Feed Pot Override Enable Mask (Set bit to enable, clear bit to disable)
	Bit 0: Coordinate System # 1
	Bit 1: Coordinate System # 2
	Bit 2: Coordinate System # 3
	Bit 3: Coordinate System # 4
	Bit 4: Coordinate System # 5
	Bit 5: Coordinate System # 6
	Bit 6: Coordinate System # 7
	Bit 7: Coordinate System # 8
	Bits 8-15: Not Used

Control Panel Request Words

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0004	0x0008	0x000C	0x0010	0x0014	0x0018	0x001C	0x0020
PMAC Address	Y:$D001	Y:$D002	Y:$D003	Y:$D004	Y:$D005	Y:$D006	Y:$D007	Y:$D008

Bit Format of Request Words

Bit	Request (1 = action requested; 0 = no action requested)
0-7	(Reserved for Delta Tau Future Use)
8	Jog-Minus (Motor Only) *
9	Jog-Plus (Motor Only) *
10	Pre-Jog (Motor Only)
11	Start (RUN) (Coord. Sys. Only)
12	Step (STEP/QUIT) (Coord. Sys. Only)
13	Stop (ABORT) (Coord. Sys. Only)
14	Home (Motor Only)
15	Feed Hold (HOLD) (Coord. Sys. Only)
* When both Jog-Minus and Jog-Plus are set, motor will stop	

Coord. Sys. #	1	2	3	4	5	6	7	8
Host Address	0x0006	0x000A	0x000E	0x0012	0x0016	0x001A	0x001E	0x0022
PMAC Address	X:$D001	X:$D002	X:$D003	X:$D004	X:$D005	X:$D006	X:$D007	X:$D008

Servo Fixed Data Buffer

PMAC can provide key motor servo data to the DPRAM where it can be accessed easily and quickly by the host computer. PMAC copies data from key motor registers into fixed registers in the DPRAM. The copying is attempted every I19 servo cycles, and is done for motors 1 through n, where n is determined by variable I59. Setting I48 to 1 and issuing the **GATHER** command enables this feature.

General Description

In operation, every I19 servo cycles, PMAC tests to see if the Host-Busy flag at 0x0024 (Y:$D009) Bit 0 is set (=1). If it is, PMAC skips this update of the buffer. If the flag is clear (=0), PMAC first sets the PMAC-Busy flag at 0x0026 (X:$D009) Bit 15 to 1, then it updates the buffer with the information from the specified motors (#1 to #{I59}), then updates the servo time at 0x0026 (X:$D009) Bits 0-14, and finally clears the PMAC-Busy flag by setting 0x0026 (X:$D009) Bit 15 to 0.

When the host wishes to read the buffer it must make sure that the PMAC-Busy flag is clear. First set the Host-Busy flag at 0x0024 (Y:$D009) Bit 0 to 1 (So PMAC stops updating and doesn't write as the host reads). Then poll the PMAC-Busy flag. If the host finds that this flag is set, it may test the flag some more times – the number of tests should always be limited – or it can decide to skip this cycle immediately. The host then reads the information it desires, and finally clears the Host-Busy flag to permit PMAC to start the next update.

The data format in the DPRAM is the same as for the Data Gathering to the Dual Port Format. 24-bit values are sign-extended to 32-bits. 48-bit values are treated as 2 24-bit values: each half is sign-extended to 32 bits, for a total of 64 bits. The data is provided in Intel format, with the low address containing the least significant word.

I19 determines the rate at which this buffer is updated by the PMAC in units of servo cycles. I59 determines the highest motor number that will be updated. For example, I59 = 1 means only Motor 1 information will be updated; I59 = 5 means information will be updated for Motors 1, 2, 3, 4, and 5. I59 = 0 means there will be no update of buffers.

On-line commands **GATHER** and **ENDGATHER** enable and disable the Servo data reporting buffer, but do not affect the Background data reporting buffer. If the **GATHER** command is issued with I48=0, the data gathering function will start instead of this servo data reporting function. This data gathering function could be storing data either in PMAC's main memory, or the DPRAM, depending on the setting of I45.

It is possible to execute both the servo data reporting function and the data gathering function simultaneously. After setting I48 to 1, the first **GATHER** command activates the servo data reporting function described here. The second **GATHER** command also activates the data gathering function that uses I20 to I44 to determine what data is to be gathered. The **ENDGATHER** command stops the function started by the most recent **GATHER** command. If both functions are running, two **ENDGATHER** commands must be issued to stop them both.

To enable this function:
1. Set I19 = update period (in servo cycles).
2. Set I59 = highest motor number (1 - 8).
3. Set I48 = 1.
4. Issue **GATHER** command.

To disable this function:
1. Issue **ENDGATHER** command, or
2. Set I19 = 0, or
3. Set I48 = 0, or
4. Set I59 = 0 (this will also disable the Background Fixed Data Buffer).

Register Map

Global Registers for Servo Fixed Data Reporting Buffer

Address	Description
(0x0024) Y:$D009	Host Status to PMAC: Bit 0 = 1 is Host-Busy, reading buffer; = 0 is not busy Bits 1-15 (reserved for future use)
(0x0026) X:$D009	PMAC Status to Host: Bits 0-14 Servo Timer Bit 15 = 1 is PMAC-Busy updating buffer; = 0 is not busy
(0x0028,A) $D00A	Global Status Bits (from Y:$0003) Low 24 Bits (First word returned on? command)
(0x002C,E) $D00B	Global Status Bits (from X:$0003) Low 24 bits (Second word returned on? command)
(0x0030-46) $D00C-11	Spare Global Var.

Motor-Specific Registers for Servo Fixed Data Reporting Buffer:

Motor Commanded Position (64 bits; 1/(Ix08*32) counts)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0048- 0x004E	0x0084- 0x008A	0x00C0- 0x00C6	0x00FC- 0x0102	0x0138- 0x013E	0x0174- 0x017A	0x01B0- 0x01B6	0x01EC- 0x01F2
PMAC Address	$D012- $D013	$D021- $D022	$D030- $D031	$D03F- $D040	$D04E- $D04F	$D05D- $D05E	$D06C- $D06D	$D07B- $D07C
Source Address	$0028	$0064	$00A0	$00DC	$0118	$0154	$0190	$01CC

Motor Actual Position (64 bits; 1/(Ix08*32) counts)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0050- 0x0056	0x008C- 0x0092	0x00C8- 0x00CE	0x0104- 0x010A	0x0140- 0x0146	0x017C- 0x0182	0x01B8- 0x01BE	0x01F4- 0x01FA
PMAC Address	$D014- $D015	$D023- $D024	$D032- $D033	$D041- $D042	$D050- $D051	$D05F- $D060	$D06E- $D06F	$D07D- $D07E
Source Address	$002B	$0067	$00A3	$00DF	$011B	$0157	$0193	$01CF

Motor Master Position (64 bits; 1/(Ix07*32) counts of the master; 1/(Ix08*32) motor counts)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0058- 0x005E	0x0094- 0x009A	0x00D0- 0x00D6	0x010C- 0x0112	0x0148- 0x014E	0x0184- 0x018A	0x01C0- 0x01C6	0x01FC- 0x0202
PMAC Address	$D016- $D017	$D025- $D026	$D034- $D035	$D043- $D044	$D052- $D053	$D061- $D062	$D070- $D071	$D07F- $D080
Source Address.	$002D	$0069	$00A5	$00E1	$011D	$0159	$0195	$01D1

Motor Compensation Position (64 bits; 1/(Ix08*32) counts)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0060- 0x0066	0x009C- 0x00A2	0x00D8- 0x00DE	0x0114- 0x011A	0x0150- 0x0156	0x018C- 0x0192	0x01C8- 0x01CE	0x0204- 0x020A
PMAC Address	$D018- $D019	$D027- $D028	$D036- $D037	$D045- $D046	$D054- $D055	$D063- $D064	$D072- $D073	$D081- $D082
Source Address.	$0046	$0082	$00BE	$00FA	$0136	$0172	$01AE	$01EA

Motor Previous DAC (32 bits; 1/256 DAC bits)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0068-0x006A	0x00A4-0x00A6	0x00E0-0x00E2	0x011C-0x011E	0x0158-0x015A	0x0194-0x0196	0x01D0-0x01D2	0x020C-0x020E
PMAC Address	$D01A	$D029	$D038	$D047	$D056	$D065	$D074	$D083
Source Address	X:$003A	X:$0076	X:$00B2	X:$00EE	X:$012A	X:$0166	X:$01A2	X:$01DE

Motor Servo Status (32 bits; low 24 bits used) 1st word returned on ? command								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x006C-0x006E	0x00A8-0x00AA	0x00E4-0x00E6	0x0120-0x0122	0x015C-0x015E	0x0198-0x019A	0x01D4-0x01D6	0x0210-0x0212
PMAC Address	$D01B	$D02A	$D039	$D048	$D057	$D066	$D075	$D084
Source Address	X:$003D	X:$0079	X:$00B5	X:$00F1	X:$012D	X:$0169	X:$01A5	X:$01E1

Motor Actual Velocity (1/(Ix09*32) counts per servo cycle)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0070 0x0072	0x00AC-0x00AE	0x00E8-0x00EA	0x0124-0x0126	0x0160-0x0162	0x019C-0x019E	0x01D8-0x01DA	0x0214-0x0216
PMAC Address	$D01C	$D02B	$D03A	$D049	$D058	$D067	$D076	$D085
Source Address	X:$0033	X:$006F	X:$00AB	X:$00E7	X:$0123	X:$015F	X:$019B	X:$01D7

Time Left in Move Segment (2*msec)								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0074-0x0076	0x00B0-0x00B2	0x00EC-0x00EE	0x0128-0x012A	0x0164-0x0166	0x01A0-0x01A2	0x01DC 0x01DE	0x0218-0x021A
PMAC Address	$D01D	$D02C	$D03B	$D04A	$D059	$D068	$D077	$D086
Source Address	X:$0020	X:$005C	X:$0098	X:$00D4	X:$0110	X:$014C	X:$0188	X:$01C4

Handwheel Pointer								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0078-0x007A	0x00B4-0x00B6	0x00F0-0x00F2	0x012C-0x012E	0x0168-0x016A	0x01A4-0x01A6	0x01E0 0x01E2	0x021C-0x021E
PMAC Address	$D01E	$D02D	$D03C	$D04B	$D05A	$D069	$D078	$D087
Source Address	X:$0029	X:$0065	X:$00A1	X:$00DD	X:$0119	X:$0155	X:$0191	X:$01CD

Spare Registers								
Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x007C-0x0082	0x00B8-0x00BE	0x00F4-0x00FA	0x0130-0x0136	0x016C-0x0172	0x01A8-0x01AE	0x01E4-0x01EA	0x0220-0x0226
PMAC Address	$D01F-$D020	$D02E-$D02F	$D03D-$D03E	$D04C-$D04D	$D05B-$D05C	$D06A-$D06B	$D079-$D07A	$D088-$D089

Background Fixed Data Buffer

PMAC can provide information of interest of the host computer through the DPRAM as a background task. This feature is enabled if I49=1. If this feature is enabled, each time PMAC executes its housekeeping tasks in background, which it does between each scan of each PLC program, it will update the buffer in DPRAM if the host has read the previous contents of the buffer. The information in this buffer is data in PMAC that is updated less often than once per servo cycle.

General Description

When this feature is enabled, each time PMAC executes it housekeeping routines, it tests to see if the Data-Ready flag at 0x0228 (Y:$D08A) Bit 0 is set (= 1). If it is, PMAC assumes the host has not finished reading the last update, so it skips this update of the buffer. If the flag is clear (=0), meaning that the host has finished reading the previous update, it will copy new data into the buffer. The information copied is for motors and coordinate systems numbered from 1 to the value of I59. After updating the buffer it sets the Data-Ready flag to 1, and sets register 0x022A (X:$D08A) equal to the servo timer to mark the time of the report.

The host computer should check the Data-Ready flag before attempting to read the information in the buffer. If the flag is not set, the host can either check several more times waiting for it to be set by PMAC, or immediately go on to other tasks. If the flag is set, the host can read the information it desires from the buffer. It must clear the Data-Ready flag after reading the information in order to permit PMAC to write another set of data into the buffer.

To Enable:

1. Set I59 = highest motor/C.S. (1 - 8), and
2. Set I49 = 1

To Disable:

1. Set I49 = 0, or
2. Set I59 = 0 (This will also disable the Servo Data Buffer)

Register Map

Global Registers for Background Fixed Data Buffer

Address	Description
0x0228 (Y:$D08A)	Buffer Status to Host and PMAC Bit 0: Data-Ready Flag =1 means PMAC done updating buffer =0 means host ready for another update from PMAC Bits 1-15: (Reserved for future use)
0x022A (X:$D08A)	PMAC Servo Timer: Updated at Data Ready Time (from X:$0000)
0x022C,E ($D08B)	Control Panel Hardware Port (from Y:$FFC0)
0x0230,2 ($D08C)	Thumbwheel Hardware Port (from Y:$FFC1)
0x0234,6 ($D08D)	Machine I/O (OPTO) Hardware Port (from Y:$FFC2)
0x0238-4A ($D08E-92)	Spare Global Variable.

Motor/Coordinate System Specific Registers for Background Fixed Data Buffer

Motor Target Position (64 bits; 1/(Ix08*32) counts)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x024C-0x0252	0x02C8-0x02CE	0x0344-0x034A	0x03C0-0x03C6	0x043C-0x0442	0x04B8-0x04BE	0x0534-0x053A	0x05B0-0x05B6
PMAC Address	$D093-$D094	$D0B2-$D0B3	$D0D1-$D0D2	$D0F0-$D0F1	$D10F-$D110	$D12E-$D12F	$D14D-$D14E	$D16C-$D16D
Source Address	$080B	$08CB	$098B	$0A4B	$0B0B	$0BCB	$0C8B	$0D4B

Motor Position Bias (64 bits; 1/(Ix08*32) counts)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0254-0x025A	0x02D0-0x02D6	0x034C-0x0352	0x03C8-0x03CE	0x0444-0x044A	0x04C0-0x04C6	0x053C-0x0542	0x05B8-0x05BE
PMAC Address	$D095-$D096	$D0B4-$D0B5	$D0D3-$D0D4	$D0F2-$D0F3	$D111-$D112	$D130-$D131	$D14F-$D150	$D16E-$D16F
Source Address	$0813	$08D3	$0993	$0A53	$0B13	$0BD3	$0C93	$0D53

Motor Status Word (32 bits; low 24 bits used) 2nd word returned on ? command								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x025C-0x025E	0x02D8-0x02DA	0x0354-0x0356	0x03D0-0x03D2	0x044C-0x044E	0x04C8-0x04CA	0x0544-0x0546	0x05C0 0x05C2
PMAC Address	$D097	$D0B6	$D0D5	$D0F4	$D113	$D132	$D151	$D170
Source Address	Y:$0814	Y:$08D4	Y:$0994	Y:$0A54	Y:$0B14	Y:$0BD4	Y:$0C94	Y:$0D54

Coordinate System Status/Definition Word (low 32 bits contains Motor Definition Word; high 32 bits contain first word returned on ?? command)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0260-0x0266	0x02DC-0x02E2	0x0358-0x035E	0x03D4-0x03DA	0x0450-0x0456	0x04CC-0x04D2	0x0548-0x054E	0x05C4-0x05CA
PMAC Address	$D098-$D099	$D0B7-$D0B8	$D0D6-$D0D7	$D0F5-$D0F6	$D114-$D115	$D133-$D134	$D152-$D153	$D171-$D172
Source Address	$0818	$08D8	$0998	$0A58	$0B18	$0BD8	$0C98	$0D58

Coordinate System A-Axis Target Position (User Units) (* Note 1)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0268-0x026E	0x02E4-0x02EA	0x0360-0x0366	0x03DC-0x03E2	0x0458-0x045E	0x04D4-0x04DA	0x0550-0x0556	0x05CC-0x05D2
PMAC Address	$D09A-$D09B	$D0B9-$D0BA	$D0D8-$D0D9	$D0F7-$D0F8	$D116-$D117	$D135-$D136	$D154-$D155	$D173-$D174
Source A	$0876	$0936	$09F6	$0AB6	$0B76	$0C36	$0CF6	$0DB6
Source B	$0896	$0956	$0A16	$0AD6	$0B96	$0C56	$0D16	$0DD6
Source C	$0819	$08D9	$0999	$0A59	$0B19	$0BD9	$0C99	$0D59

Coordinate System B-Axis Target Position (User Units) (* Note 1)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0270-0x0276	0x02EC-0x02F2	0x0368-0x036E	0x03E4-0x03EA	0x0460-0x0466	0x04DC-0x04E2	0x0558-0x055E	0x05D4-0x05DA
PMAC Address	$D09C-$D09D	$D0BB-$D0BC	$D0DA-$D0DB	$D0F9-$D0FA	$D118-$D119	$D137-$D138	$D156-$D157	$D175-$D176
Source A	$0877	$0937	$09F7	$0AB7	$0B77	$0C37	$0CF7	$0DB7
Source B	$0897	$0957	$0A17	$0AD7	$0B97	$0C57	$0D17	$0DD7
Source C	$081A	$08DA	$099A	$0A5A	$0B1A	$0BDA	$0C9A	$0D5A

Coordinate System C-Axis Target Position (User Units)* Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0278-0x027E	0x02F4-0x02FA	0x0370-0x0376	0x03EC-0x03F2	0x0468-0x046E	0x04E4-0x04EA	0x0560-0x0566	0x05DC-0x05E2
PMAC Address	$D09E-$D09F	$D0BD-$D0BE	$D0DC-$D0DD	$D0FB-$D0FC	$D11A-$D11B	$D139-$D13A	$D158-$D159	$D177-$D178
Source A	$0878	$0938	$09F8	$0AB8	$0B78	$0C38	$0CF8	$0DB8
Source B	$0898	$0958	$0A18	$0AD8	$0B98	$0C58	$0D18	$0DD8
Source C	$081B	$08DB	$099B	$0A5B	$0B1B	$0BDB	$0C9B	$0D5B

Coordinate System U-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0280-0x0286	0x02FC-0x0302	0x0378-0x037E	0x03F4-0x03FA	0x0470-0x0476	0x04EC-0x04F2	0x0568-0x056E	0x05E4-0x05EA
PMAC Address	$D0A0-$D0A1	$D0BF-$D0C0	$D0DE-$D0DF	$D0FD-$D0FE	$D11C-$D11D	$D13B-$D13C	$D15A-$D15B	$D179-$D17A
Source A	$0879	$0939	$09F9	$0AB9	$0B79	$0C39	$0CF9	$0DB9
Source B	$0899	$0959	$0A19	$0AD9	$0B99	$0C59	$0D19	$0DD9
Source C	$081C	$08DC	$099C	$0A5C	$0B1C	$0BDC	$0C9C	$0D5C

Coordinate System V-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0288-0x028E	0x0304-0x030A	0x0380-0x0386	0x03FC-0x0402	0x0478-0x047E	0x04F4-0x04FA	0x0570-0x0576	0x05EC-0x05F2
PMAC Address	$D0A2-$D0A3	$D0C1-$D0C2	$D0E0-$D0E1	$D0FF-$D100	$D11E-$D11F	$D13D-$D13E	$D15C-$D15D	$D17B-$D17C
Source A	$087A	$093A	$09FA	$0ABA	$0B7A	$0C3A	$0CFA	$0DBA
Source B	$089A	$095A	$0A1A	$0ADA	$0B9A	$0C5A	$0D1A	$0DDA
Source C	$081D	$08DD	$099D	$0A5D	$0B1D	$0BDD	$0C9D	$0D5D

Coordinate System W-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0290-0x0296	0x030C-0x0312	0x0388-0x038E	0x0404-0x040A	0x0480-0x0486	0x04FC-0x0502	0x0578-0x057E	0x05F4-0x05FA
PMAC Address	$D0A4-$D0A5	$D0C3-$D0C4	$D0E2-$D0E3	$D101-$D102	$D120-$D121	$D13F-$D140	$D15E-$D15F	$D17D-$D17E
Source A	$087B	$093B	$09FB	$0ABB	$0B7B	$0C3B	$0CFB	$0DBB
Source B	$089B	$095B	$0A1B	$0ADB	$0B9B	$0C5B	$0D1B	$0DDB
Source C	$081E	$08DE	$099E	$0A5E	$0B1E	$0BDE	$0C9E	$0D5E

Coordinate System X-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0298-0x029E	0x0314-0x031A	0x0390-0x0396	0x040C-0x0412	0x0488-0x048E	0x0504-0x050A	0x0580-0x0586	0x05FC-0x0602
PMAC Address	$D0A6-$D0A7	$D0C5-$D0C6	$D0E4-$D0E5	$D103-$D104	$D122-$D123	$D141-$D142	$D160-$D161	$D17F-$D180
Source A	$087C	$093C	$09FC	$0ABC	$0B7C	$0C3C	$0CFC	$0DBC
Source B	$089C	$095C	$0A1C	$0ADC	$0B9C	$0C5C	$0D1C	$0DDC
Source C	$081F	$08DF	$099F	$0A5F	$0B1F	$0BDF	$0C9F	$0D5F

Coordinate System Y-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02A0-0x02A6	0x031C-0x0322	0x0398-0x039E	0x0414-0x041A	0x0490-0x0496	0x050C-0x0512	0x0588-0x058E	0x0604-0x060A
PMAC Address	$D0A8-$D0A9	$D0C7-$D0C8	$D0E6-$D0E7	$D105-$D106	$D124-$D125	$D143-$D144	$D162-$D163	$D181-$D182
Source A	$087D	$093D	$09FD	$0ABD	$0B7D	$0C3D	$0CFD	$0DBD
Source B	$089D	$095D	$0A1D	$0ADD	$0B9D	$0C5D	$0D1D	$0DDD
Source C	$0820	$08E0	$09A0	$0A60	$0B20	$0BE0	$0CA0	$0D60

Coordinate System Z-Axis Target Position (User Units) * Note 1								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02A8-0x02AE	0x0324-0x032A	0x03A0-0x03A6	0x041C-0x0422	0x0498-0x049E	0x0514-0x051A	0x0590-0x0596	0x060C-0x0612
PMAC Address	$D0AA-$D0AB	$D0C9-$D0CA	$D0E8-$D0E9	$D107-$D108	$D126-$D127	$D145-$D146	$D164-$D165	$D183-$D184
Source A	$087E	$093E	$09FE	$0ABE	$0B7E	$0C3E	$0CFE	$0DBE
Source B	$089E	$095E	$0A1E	$0ADE	$0B9E	$0C5E	$0D1E	$0DDE
Source C	$0821	$08E1	$09A1	$0A61	$0B21	$0BE1	$0CA1	$0D61

Coordinate System Program Execution Status (32 bits; low 24 bits used) (Second word returned on ?? command)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B0-0x02B2	0x032C-0x032E	0x03A8-0x03AA	0x0424-0x0426	0x04A0-0x04A2	0x051C-0x051E	0x0598-0x059A	0x0614-0x0616
PMAC Address	$D0AC	$D0CB	$D0EA	$D109	$D128	$D147	$D166	$D185
Source Address	Y:$0817	Y:$08D7	Y:$0997	Y:$0A57	Y:$0B17	Y:$0BD7	Y:$0C97	Y:$0D57

Coordinate System Program Lines Remaining (32 bits) (Same value as PR command returns)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B4-0x02B6	0x0330-0x0332	0x03AC-0x03AE	0x0428-0x042A	0x04A4-0x04A6	0x0520-0x0522	0x059C-0x059E	0x0618-0x061A
PMAC Address	$D0AD	$D0CC	$D0EB	$D10A	$D129	$D148	$D167	$D186
Source Address	Y:$08AE	Y:$096E	Y:$0A2E	Y$0AEE	Y$0BAE	Y:$0C6E	Y:$0D2E	Y$0DEE

Coordinate System Time Remaining in move when I13 > 0 (2*msec)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B8-0x02BA	0x0334-0x0336	0x03B0-0x03B2	0x042C-0x042E	0x04A8-0x04AA	0x0524-0x0526	0x05A0-0x05A2	0x061C-0x061E
PMAC Address	$D0AE	$D0CD	$D0EC	$D10B	$D12A	$D149	$D168	$D187
Source Address	X:$0020	X:$005C	X:$0098	X:$00D4	X:$0110	X:$014C	X:$0188	X:$01C4

Coordinate System Time Remaining in accel/decel when I13 > 0 (2*msec)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02BC-0x02BE	0x0338-0x033A	0x03B4-0x03B6	0x0430-0x0432	0x04AC-0x04AE	0x0528-0x052A	0x05A4-0x05A6	0x0620-0x0622
PMAC Address	$D0AF	$D0CE	$D0ED	$D10C	$D12B	$D14A	$D169	$D188

Coordinate System Program Execution Address Offset (Same value as PE command returns)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02C0-0x02C2	0x033C-0x033E	0x03B8-0x03BA	0x0434-0x0436	0x04B0-0x04B2	0x052C-0x052E	0x05A8-0x05AA	0x0624-0x0626
PMAC Address	$D0B0	$D0CF	$D0EE	$D10D	$D12C	$D14B	$D16A	$D189

Motor Averaged Actual Velocity (1/[Ix09*32] counts per servo cycle)								
Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02C4-0x02C6	0x0340 0x0342	0x03BC-0x03BE	0x0438-0x043A	0x04B4-0x04B6	0x0530-0x0532	0x05AC-0x05AE	0x0628-0x062A
PMAC Address	$D0B1	$D0D0	$D0EF	$D10E	$D12D	$D14C	$D16B	$D18A
Source Address	Y:$082A	Y:$08EA	Y$09AA	Y$0A6A	Y$0B2A	Y$0BEA	Y$0CAA	Y$0D6A

*** Note 1:** The following is the logic used in the PMAC to determine which variable will be put in this slot. It is controlled by bits of the coordinate system program execution status word (PSTATUS):

```
If (PSTATUS.7 == 1 && PSTATUS.5 == 0)
     Use Source A
Else
      If (PSTATUS.9 == 1)
         Use Source B
      Else
         Use Source C
      Endif
Endif
```

PSTATUS.7 is the Segmented move flag (I13 != 0)
PSTATUS.5 is the Segmented move stop flag
PSTATUS.9 is the Tool Compensation flag

Background Variable Data Read Buffer

The Background Variable Data Read Buffer allows the user to have up to 128 user-specified PMAC registers copied into DPRAM during the background cycle. This function is controlled by I55. The buffer has two modes of operation, single user and multi-user. The default mode is the single user mode. It is active when bit 8 of the control word (Y:$D1FA) is set to zero. Multi-user mode is active when bit 8 of the control word (Y:$D1FA) is set to one.

General Description

The buffer has three parts. The first part is the header: four 16-bit words (eight host addresses) containing handshake information and defining the location and size of the rest of the table. This is at a fixed location in DPRAM (see table at end of section). The second part contains the address specifications of the PMAC registers to be copied into DPRAM. It occupies two 16-bit words (four host addresses) for each PMAC location to be copied, starting at the location specified in the header. The third part, starting immediately after the end of the second part, contains the copied information from the PMAC registers. It contains two 16-bit words (four host addresses) for each short (X or Y) PMAC location copied, and four 16-bit words (eight host addresses) for each long PMAC location copied. The data format is the same as for data gathering to dual-ported RAM.

Enabling

To start operation of this buffer:

1. Write the starting location of the second part of the buffer into register 0x07EE (X:$D1FB). This location is expressed as a PMAC address, and it must be between $D200 and $DFFF.
2. Starting at the DPRAM location specified in the above step, write the PMAC addresses of the registers to be copied, and the register types. The first 16-bit word is the PMAC address of the first register to be copied; the second 16-bit word takes a value of 0, 1, 2, or 4 to specify Y, Long, X, or Special respectively, for the first register. The third and fourth word specifies the address and type of the second register to be copied, and so on.
3. Write a number representing the size of the buffer into register 0x07EC. (Y:$D1FB). This value must be between 1 and 128. When PMAC sees that this value is greater than zero and the individual data ready bit is zero, it is ready to start copying the registers you have specified into DPRAM.
4. To enable the single user mode write a zero into the control word at 0x07E8 (Y:$D1FA). To enable the multi-user mode write a 256 (set bit 8 and clear bit 0) into the control word at 0x07E8 (Y:$D1FA) and set bit 15 = 0 of each variable's data type register (X memory register). This will tell PMAC that the host is ready to receive data and what the mode is for the data.
5. Set I55 to 1. This enables both the background variable data reporting function and the background variable data writing function.

Single User Mode Procedure

In operation, PMAC will try to copy data into the buffer each background cycle – between each scan of each PLC program. If bit 0 of the control word 0x07E8 is set to 1, it will assume that the host has not finished reading the data from the last cycle, so it will skip this cycle. If bit 0 is 0, it will copy all of the specified registers.

When PMAC is done copying the specified registers, it copies 16 bits of the servo timer register (X:$0000) into the DPRAM at 0x07EA (X:$D1FA). Then it sets Bit 0 of the control word 0x07E8 (Y:$D1FA) to let the host know that it has completed a cycle.

When the host wants to read this data, it should check to see that Bit 0 of the control word 0x07E8 (the Data Ready bit) has been set. If it has, the host can begin reading and processing the data in the DPRAM. When it is done, it should clear the Data Ready bit to let PMAC know that it can perform another cycle.

Multi-User Mode Procedure

The operation of this mode is very similar to the Single User Mode described above. The main difference is that the control word is no longer used as a global handshaking bit for updating the buffer. It enables or disables the multi-user mode only. In multi-user mode the control word is never modified by PMAC. Handshaking is now on an individual variable basis and is controlled by bit 15 of the variable's data type specifier.

Each background cycle, between each scan of each PLC program, PMAC will try to copy data into each variable in the buffer. Bit 15 of each variable's data type specifier controls whether or not PMAC is allowed to update that particular variable's value. PMAC will skip updating any variable that has bit of its data type specifier set to 1. Any variable that has bit 15 set to 0 will be updated.

When PMAC is done servicing the buffer, it copies 16 bits of the servo timer register (X:$0000) into the DPRAM at 0x07EA (X:$D1FA). This is not dependent upon updating any variables in the buffer.

When the host wants to read a register, it should check to see that Bit 15 of the data type specifier (the Data Ready bit) has been set. If it has, the host can begin reading and processing the data from that register. When it is done, it should clear the Data Ready bit to let PMAC know that it can update that register the next cycle.

Data Format

Each 24-bit (X or Y) register is sign-extended to 32 bits. For a 48-bit (Long) register, each 24-bit half is sign-extended to 32 bits, for a total of 64 bits in the DPRAM. This data starts immediately after the last address specification register.

Disabling

To disable this function, you can set the size register 0x07EC (Y:$D1FB) to 0, or simply leave the individual Data Ready bits set.

Register Map

Background Variable Data Buffer -- PMAC to Host Transfer	
Address	**Description**
0x07E8 (Y:$D1FA)	PMAC to HOST (Bit 0 = 1 for single user mode) Data Ready. PMAC done updating buffer - Host must clear for more data.
0x07EA X:$D1FA	Servo Timer (Updated at Data Ready Time)
0x07EC (Y:$D1FB)	Size of Address Buffer (measured in long integers of 32 bits each)
0x07EE (X:$D1FB)	Start of Address Buffer (Ex. $D400; must be $D200 to $DFFD)

Variable Address Buffer Format (2x16-bit words)			
X:Mem Bits 15: Data Ready (multi-user mode)	**X:Mem Bits 0 - 2: Variable type to read**	**Y:Mem Variable address**	**Dual Port Data Length**
1 = PMAC data ready 0 = Host request data	0 = PMAC Var. Y:Mem.	PMAC Address of Variable	32 bits
1 = PMAC data ready 0 = Host request data	1 = PMAC Var. Long	PMAC Address of Variable	64 bits
1 = PMAC data ready 0 = Host request data	2 = PMAC Var. X:Mem.	PMAC Address of Variable	32 bits
1 = PMAC data ready 0 = Host request data	4 = Special (Firmware 1.16) PLCC Function Block	PLCC Function Block Number. Y:$9FFF has the base address of the function blocks.	64 bits

Background Variable Data Write Buffer

The Background Variable Data Write Buffer is essentially the opposite of the Background Variable Data Read Buffer described above. It allows the user to write to up to 32 user-specified registers or particular bits in registers to PMAC without using a communications port (PC Bus, serial, or DPRAM ASCII I/O). This allows the user to set any PMAC variable without using an ASCII command such as M1=1 and without worrying about an open Rotary Buffer.

This function is controlled by I55. It is available starting with firmware version 1.15G. PLCC function blocks are available starting with firmware version 1.16.

General Description

The buffer has two parts. The first part is the header: two 16-bit words (four host addresses) containing handshake information and defining the location and size of the rest of the table. This is at a fixed location in DPRAM (PMAC address $D1F5 as shown in the table at the end of this section). The second part contains the address specifications of the PMAC registers to be copied into PMAC. It occupies six 16-bit words (twelve host addresses) for each PMAC location to be written to, starting at the location specified in the header.

Enabling

To start operation of this buffer:

1. Write the starting location of the second part of the buffer into register 0x07E8 (X:$D1F5). This location is expressed as a PMAC address, and it must be between $D200 and $DFFD.
2. Starting at the DPRAM location specified in the above step, write the PMAC addresses of the registers to be copied, and the register types. The first 16-bit word is the PMAC address of the first register to be copied; the second 16-bit word takes a value of 0 to 32768 to specify the type, width, and offset for writing to the PMAC register. The third, fourth, fifth, and sixth words specify the data to be written.

> ***Note:***
>
> If address 0 is specified, it will be writing into PMAC's servo clock register and will cause PMAC's watchdog timer to trip.

3. Write a number representing the size of the buffer into register 0x07E6. (Y:$D1F5). This value must be between 1 and 32. When PMAC sees that this value is greater than zero, it is ready to start copying the registers you have specified into PMAC. When it is finished it will change the value in this register to a 0.
4. Set I55 to 1. This enables both the background variable data read function and the background variable data write function.

Procedure

In operation, PMAC will copy the data from the buffer into PMAC during the background cycle whenever Y:$D1F5 is a not zero. If this register is 0 it will assume that the host has not finished placing the data in the buffer and will not write to PMAC. Once this register is set to a number from 1 to 32 it will copy that many registers, starting at the start of the header start address information, from the DPRAM to PMAC.

When PMAC is done copying the specified registers, it sets register Y:$D1F5 to zero to let the host know that it has completed a cycle.

When the host wants to update this buffer, it should check to see that Y:$D1F5 is zero. When it is done, it should setup the address/data structure. Then set Y:$D1F5 to the number of registers to copy to PMAC to let PMAC know that it can perform another cycle.

Data Format

PMAC X, Y, and Special registers will use the long 32-bit data 1word. The 32-bit Data 2 word is not used in this case. For PLCC function blocks, only X (bits 0-2 = 6) or Y (bits 0-2 = 4) memory writes are allowed.

For a 48-bit PMAC integer or float point value, The L (Long) format should be used. L will have the lower 32 bits of the total 48 bits in the long 32-bit data 1 word and the upper 16 bits in the lower 32-bit data 2 word. PCOMM.LIB supplies a function to convert this IEEE 64-bit number to a PMAC 48-bit floating-point number. This data starts immediately after the last address specification register.

Disabling

To disable this function, simply leave Y:$D1F5 set to zero.

Register Map

Background Variable Data Write Buffer -- Host to PMAC Transfer	
Address	**Description**
0x07E6 (Y:$D1F5)	HOST to PMAC Data Transferred. PMAC is updated when cleared. Host must set for another update.
0x07E8 X:$D1F5	Starting address of data structure ($D200 - $DFFD).

Variable Address Buffer Format for each Data Structure (6x16-bit)			
DPRAM Address	**X:Mem**	**Y:Mem**	**X:Mem Bit Definitions**
$D200	Bits 13 - 15: Special type	PMAC address	0 = PLCC Function Block 1-7 = Reserved for future use
to $DFFD	Bits 8 - 12: Offset	PMAC address	Offset = 0..23. -- This is the starting offset for the read.
	Bits 3 - 7: Width	PMAC address	Width = 0, 1, 4, 8, 12, 16, 20 -- 0 is a 24-bit width.
	Bits 0 - 2: Variable type to write	PLCC Function Block Number	0 = Y register 1 = L register 2 = X register 4 = Special Y register 6 = Special X register
	Upper 16-bits of data 1	Lower 16-bits of data 1	Data to send to PMAC
	Upper 16-bits of data 2	Lower 16-bits of data 2	Data to send to PMAC

DPRAM 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 function was implemented before V1.14, and is described in the User's Manual for PMAC.

The data-gathering buffer in DPRAM must start at address 0x0800 ($D200). Its length is determined by the **`DEFINE GATHER {size}`** command. If also using a DPRAM Background Variable Data Buffer, and/or a DPRAM Binary Rotary Program Buffer, it is important to set the starting points and sizes of those buffers so there is no overlap.

Note:

If the **{size}** requested in the **DEFINE GATHER {size}** command is smaller than required to hold an even multiple of the requested data, the actual data storage will go beyond the requested **{size}** to the next higher multiple of memory words required to hold the data. For example, if gathering one 24-bit value and one 48-bit value, three words of memory is needed. If the **{size}** specified is 4000 words (not a multiple of 3), the actual storage size will be 4002 words (the next higher multiple of 3).

PMAC to Host Transfer (memory locations set by PMAC)	
Address	**Description**
0x07FC (Y:$D1FF)	Data Gather Buffer Size.
0x07FC (X:$D1FF)	PMAC Data Gather Buffer Storage Address. If I45 = 2 and the buffer's end has been reached (this index is greater than or equal to the size), the **DEFINE GATHER** command must be issued again to allow gathering to restart.
0x0800 ($D200)	Start of Data Gather Buffer (not changeable).

The data format for the Data Gathering to the Dual Port Format is that 24-bit values are sign-extended to 32-bits. 48-bit values are treated as 2 24-bit values: each half is sign-extended to 32 bits, for a total of 64 bits. The data is provided in Intel format, with the low address containing the least significant word.

The variables that control the Data Gathering are as follows: I19 determines the rate at which this buffer is updated by the PMAC in units of servo cycles. I20 to I44 to determine what data (PMAC addresses) is to be gathered. I45 determines if the data will be stored in PMAC's main memory or the DPRAM.

On-line commands **GATHER** and **ENDGATHER** enable and disable the Data Gathering. These commands do not affect the Background data-reporting buffer but could affect the Servo fixed data reporting function. If the **GATHER** command is issued with I48=0, the data gathering function will start. If the **GATHER** command is issued with I48=1, the servo fixed data reporting function will start instead of the data gathering function.

It is possible to execute both the servo fixed data reporting function and the data gathering function simultaneously. After setting I48 to 1, the first **GATHER** command activates the servo fixed data reporting function described earlier. The next **GATHER** command will activate the data gathering function. The **ENDGATHER** command stops the function started by the most recent **GATHER** command. If both functions are running, two **ENDGATHER** commands must be issued to stop them both.

To enable this function:

1. Set I19 = update period (in servo cycles).
2. Set I45 = data storage location and mode (2 or 3).
3. Issue a **DEFINE GATHER {size}** (Determine buffer size).
4. Issue **GATHER** command (if I48=1 issue **GATHER GATHER** command).

To disable this function:

1. Issue **ENDGATHER** command, or
2. Set I19 = 0.

DPRAM ASCII Communications

PMAC can perform ASCII communications through the dual-ported RAM, as well as through the normal bus communications port and the serial port. The DPRAM ASCII communications is enabled by setting I58 to 1. This permits the host to send an ASCII command to PMAC by placing the command string characters in consecutive registers in the DPRAM and setting a flag to notify PMAC of the command. PMAC will respond by placing its response string characters in consecutive registers in the DPRAM, then setting a flag, and optionally triggering an interrupt to notify the host of the response.

General Description

Setting I58 to 1 enables the ASCII I/O feature. When this mode is enabled, the following I-variables are automatically set to the following values:

```
I3 = 2 or 0                ;Handshake control (PROM 1.15x: 3 changed to 2; 1
                           ;changed to 0, ;PROM 1.14x:  Always set to 2)
I4 = 0                     ;Checksum control
I6 = 1                     ;Error reporting control
```

These variables should be changed subsequent to setting I58 to 1.

Once the DPRAM ASCII communication is enabled, PMAC is ready to receive ASCII commands either through the normal bus port or through the DPRAM port. The active response port is whichever port through which PMAC has received the most recent command. Therefore, PMAC will respond to a host command through the port where it received the command. Communications resulting from internal **SEND** or **CMD** statements in PMAC programs will be sent to the active response port.

When sending and receiving ASCII strings through the DPRAM, the handshake control characters are not part of the strings, as they are on the other ports. Instead, they are placed in fixed control word registers.

To make the serial port the active response port, it is necessary to send the **<CTRL-Z>** command through the serial port. This will disable the DPRAM ASCII communications by setting I58 to 0.

Setting I58=0 will disable the ASCII communications -- PMAC will not accept any commands through the DPRAM. If you have been communicating through the DPRAM, it is a good idea also to send **<CTRL-X>** command to clear any pending responses.

Note:

With I58=1, sending a **<CTRL-X>** command will empty the command and response queues; it will also clear the control words 0x06D0 (Y:$D1B4) and 0x062C (Y:$D18B).

Read/Write Procedure

To initialize the buffer:

1. Clear registers 0x062C (Y:$D1B4) and 0x06D0 (Y:$D18B) Bit 0.
2. Set I58 = 1

To send a command line:

1. Put ASCII characters in Host-to-PMAC buffer 0x0630-0x06CE ($D18C - $D1B3) with a NULL character to terminate the string
2. Set Host-Output Control Flag 0x062C (Y:$D18B) Bit 0 to 1 (Host-Output Complete)

Note:

When sending ASCII command strings through the DPRAM, it is not necessary to use a carriage-return character. PMAC looks for the NULL character (ASCII value 0) to mark the end of the string, and looks at the Host-Output Control Flag to know when it has been given a command.

The buffer in DPRAM is limited to 159 characters; however, a command line of up to 200 characters can be sent by using the DPRAM buffer twice. The first 159 characters are placed in DPRAM without a terminating NULL character and the host-output-complete flag is set; then the remaining up to 41 characters are sent with a NULL character to terminate, and the host-output-complete flag is set again

To read a response:

1. Wait for Host-Input Control Word 0x06D0 (Y:$D1B4) > 0 (Response Ready)
2. Interpret the value in this register to determine what type of response is present. If the register does not show an error, continue with the following steps.
3. Read 0x06D2 (X:$D1B4) to find the number of characters in the response.
4. Read the Host-Input Buffer starting at 0x06D4 ($D1B5) until the specified number of characters has been read.
5. Clear the Host-Input Control Word 0x06D0 (Y:$D1B4).
6. Repeat steps 1 to 5 until the Host-Input Control Word contains an <ACK> to mark the end of transmission.

PROM version 1.15 provides a register in the DPRAM, 0x062E (X:$D18B) to be used specifically for sending PMAC a control character. The read/write procedure is exactly as described above except now instead of writing a string to the Host-to PMAC buffer, write the ASCII value of the control character to the dedicated register 0x062E (X:$D18B).

Note:

PMAC's ASCII response strings in DPRAM do not end with a carriage-return character. The host computer has two ways of knowing where the end of the string is. First, the register immediately preceding the string is given the number of characters in the string – convenient for Pascal programmers. Second, the string is terminated with the NULL character (ASCII value 0) – convenient for C programmers.

Interrupts

I56 = 1 enables the Dual-Ported RAM ASCII interrupt feature. With this enabled, the PMAC will interrupt the Host when a PMAC-to-Host buffer is ready to be read by the Host.

VME users will get the normal VME bus communications interrupt (IRQ line specified by the value in X:$788) with an interrupt vector (default $A2) one greater than the value specified in X:$789 when the Dual Port RAM ASCII PMAC to HOST data buffer is ready. The non-DPRAM (mailbox) VME interrupt vectors remain as before ($A0 and $A1 default).

PC users will get IR7 interrupt from the PMAC PC or PMAC Lite if jumper E55 is installed. The EQU4 signal is used to generate this interrupt, so it is unavailable for position-compare use when I56 = 1. Due to the fact that the 8259 does not latch the interrupt on a transition interrupt, the PMAC will hold the IR7 line true until the host services the Dual Port ASCII PMAC-to-Host buffer and sets 0x06D0 (Y:$D1B4) to 0 (saying it has the data).

Because of this the host may see an IR7 interrupt still active when it gets another interrupt. If so, the PMAC has not overwritten the PMAC-to-Host ASCII buffer; it just has not satisfied the above described condition. Also if 0x06D0 (Y:$D1B4) = 0, the IR7 was from the previous exchange and there is no data to be received by the Host. The last transmission should be an <ACK> regardless of whether you have I56 equal to one or zero.

Register Map

ASCII Host-to-PMAC Transfer	
Address	**Description**
0x062C (Y:$D18B)	ASCII Host-Output Control Word Bit 0: Host-Output-Control Flag = 0: Host Output Enable. Set to 0 by PMAC when it has processed a command string. = 1: Host Output Complete. Set to 1 by the Host when it has loaded a full command string. Bits 1-15: (Reserved for future use)
0x062E (X:$D18B)	Bits 0-7: (Host-Output) Control character to be sent to PMAC
0x0630-0x06CE ($D18C-$D1B3)	Host-to-PMAC (Host-Output) Transfer Buffer -- Up to 159 characters with a NULL character (ASCII value 0) terminating the string.

ASCII PMAC-to-Host Transfer	
Address	**Description**
0x06D0 (Y:$D1B4)	ASCII Host-Input Control Word (PMAC termination character) **Upper/Lower 8 of 16 bits(shown as hex digits):** 00/00: PMAC Output Enable: Set to this value by HOST, to note that it has processed the previous response data. PMAC will not update the Host Input buffer unless this control word is zero. 00/0D: (<CR>) Set to this value by PMAC after it has filled the Host-Input buffer, to note that there is more data to follow in transmission 01/0D: (<CR>) Set to this value by PMAC after it has filled the Host-Input buffer, to note a PMAC program "CMD" command response - End of transmission; no ACK is sent 02/0D: (<CR>) Set to this value by PMAC after it has filled the Host-Input buffer, to note a PMAC program "SEND" command message - End of transmission; No ACK is sent 8d/dd: Set to this value by PMAC to note an error in the command; where "ddd" = 12-bit BCD-coded Error Number (See I6 description for error listing)
0x06D2 (X:$D1B4)	(# of ASCII characters + 1) in the Host-Input buffer; set by PMAC after it has written to the buffer
0x06D4-0x07D2 ($D1B5-D1F4)	PMAC-to-Host (Host-Input) Transfer Buffer --Up to 255 characters with a NULL character (ASCII value 0) terminating the string.

There are two ASCII character bytes per 16-bit register in DPRAM. The first character of the pair is in the LSB. This format should be convenient for host computers with Intel processors, but probably will require a byte swap for host computers with Motorola processors.

Binary Rotary Program Buffers

The Binary Rotary Program Buffers in PMAC's DPRAM allows the host computer to send program commands to PMAC in binary format for the fastest possible transmission of these commands. A standard rotary program buffer must be established in PMAC's internal memory with the **DEFINE ROTARY** command, as well as a buffer in DPRAM for the binary transmission from the host. Only the first of two binary rotary buffers that PMAC supports in DPRAM is described here. Call or fax Delta Tau if the details (memory register address/description) for the second one are required.

When PMAC receives a binary-format program command in DPRAM from the host, it simply copies it into the rotary program buffer in internal memory. The end result is the same as if an ASCII program command had been sent to PMAC through any of the ports, but the transmission is quicker because PMAC gets the command all at once, and it does not have to parse the command from ASCII format.

Note:

Delta Tau has PC subroutines written in C to interface with the DPRAM binary rotary program buffer. Contact the factory for details.

General Description

The host defines the start and size of the Binary Rotary Buffer in 0x07F8 and 0x07FA (Y: and X:$D1FE), sets the Host and PMAC indexes at 0x07F4 and 0x07F6 (Y: and X:$D1FD) to zero the beginning of the buffer, fills the rotary buffer with data, sets the host index to the end of the buffer then enable the Binary Rotary buffer transfer by setting I57 = 1. The host can use the PMAC Indexes to determine when to update the Dual Port buffer or the BREQ interrupt will be active on the internal Rotary buffer also.

Rules:

1. The data in the buffer must contain a complete line. Therefore the line must be in the buffer before the Host Binary Rotary Buffer Index is updated.
2. The buffer size parameter is subject to the following restrictions:
   a. Minimum Size = six words (24 bytes) which means it could have only one command per line.
   b. The size must be an even number (6,8,12 etc.) so that no 64-bit instruction command will wrap in the buffer.
3. Host Buffer Full when PMAC_Index = (Host_Index+4)% buffer_size
4. Host Buffer Empty when PMAC_Index = Host_Index
5. In the DPRAM rotary buffer at the Host_Index a special end of buffer command needs to be stored after each transfer. It is the following:

   DPRotBuf(Host_Index) = $800
   DPRotBuf(Host_Index) + 1 = $FFFF

   Note:

   These values are Intel format.

6. Not all motion program commands can be sent through the binary rotary program buffer. First, any command that cannot be sent to a rotary buffer through an ASCII command string cannot be sent in binary form (e.g. **IF**, **WHILE**, **GOTO**, **GOSUB**). Second, the binary command syntax does not support mathematical expressions. Any place that the ASCII command syntax description uses the form **{data}** or **{expression}**, the binary command must use the form **{constant}** instead.

Register Map

HOST to PMAC Transfer	
Address	**Description**
0x07F0 (Y:$D1FC)	PMAC to HOST Binary Rotary Buffer Status Word Bit 15 = 1 :Error (Stops processing commands) Bit14 = 1 :Internal Rotary buffer full (Busy flag) PMAC Index stops updating. Bits 7-0 = Code Error ------ ---------- 1 Internal Rotary Buffer size = 0 or DPRAM Rotary Buffer Size = 0 These flags are set and reset by the PMAC. The Busy flag is set when the PMAC internal rotary buffer is full. This however does not mean the DPRAM Binary Rotary buffer is full (See Rules). The Busy flag is reset when the PMAC internal rotary buffer is not full or the DPR binary rotary buffer is empty.
0x07F2 (X:$D1FC)	Spare
0x07F4 (Y:$D1FD)	Host Binary Rotary Buffer Index (for 32 bits)
0x07F6 (X:$D1FD)	PMAC Binary Rotary Buffer Index (for 32 bits)
0x07F8 (Y:$D1FE)	Size of Binary Rotary Buffer (in long integers - 32 bits)
0x07FA (X:$D1FE)	Starting Binary Rotary Buffer PMAC Address (Ex. $D600, must be >= $D200)
???	Binary Rotary Buffer (Host to ensure that does not conflict with Data Gather Buffer)

Binary Command Structure

The first, second, and third 16-bit words contain the 48 bits of data for the internal 48-bit PMAC command format. The fourth word of the 64-bit dual port rotary buffer format is not used in the transfer and is reserved for future use.

Allowed Rotary Buffer Commands	
CMD Type	**Description**
0	Single letter, CALL, DWELL with floating point data
1	Commands with integer or mask type data
2	In=, Mn=, Pn=, Qn=, Mn|=, Mn^=, Mn&=, Mn==

Command Type = 0		
Code 1	**Code 2**	**PMAC Command**
1 - 26	-	A through Z letter commands with floating point data; 14 & 15 = N & O are not used here
27	-	CALL
28	0	TA
28	1	TS
28	2	PVT
28	67	TM
28	68	DWELL
28	69	DELAY
28	70	:
28	71	^
28	72	CCR

Command Type = 1		
Code 1	**Code 2**	**PMAC Command**
30	0/1	ENABLE / DISABLE PLC
30	0/1/2	ABS / INC / FRAX
30	3	HOME
30	4	Not Available
30	5	Not Available
30	6	HOMEZ
30	0	CIRn
30	1	LINEAR
30	2	NORM
30	3	PSET
30	4	SPLINEn
30	5	STOP
30	6	BSTART
30	7	BSTOP
30	8	WAIT (Not Implemented)
30	9	RAPID
30	10	CCn
30	11	TSELn
30	12	ADISn
30	13	AROTn
30	14	IDISn
30	15	IROTn
30	16	TINIT

Command Type = 2		
Code 1	**Code 2**	**PMAC Command**
14	0	In={constant} (n = 0 to 1023)
14	1	Mn={constant} (n = 0 to 1023)
14	2	Pn={constant} (n = 0 to 1023)
14	3	Qn={constant} (n = 0 to 1023)
15	0	Mn|={constant} (n = 0 to 1023)
15	1	Mn&={constant} (n = 0 to 1023)
15	2	Mn^={constant} (n = 0 to 1023)
15	3	Mn=={constant} (n = 0 to 1023)

Internal PMAC 48-Bit Command Format

This section details the internal structure of the permitted binary commands. It is for reference for those designing their own subroutines to create these commands. Contact the Delta Tau factory for pre-written subroutines that create these commands.

Command Type = 0

Sub-type A: A thru Z and CALL

(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
FLOAT MANTISSA ( 36 BITS )       | 1| CODE1 = 1-27 | EXPONENT(6 BIT) |
^   1,3 ... = A, B ... & 27 = CALL
|          * Note 1
|--- Start of line bit
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
 S| FRACTIONAL PART OF FLOAT MANTISSA ( 36 BITS )
 ^-- SIGN
```

Sub-type B: TA, TS, PVT, TM, DWELL, DELAY, :,^ CCR

(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
     CODE2          | Not Used     | 1|  CODE1 = 28  | EXPONENT(6 BIT) |
    0,1,2 = TA,TS,PVT                ^
    67,68,69 = TM,DWELL,DELAY        |---Start of line bit
    70,71,72 = :, ^, CCR
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
 S| FRACTIONAL PART OF FLOAT MANTISSA (24 BITS)
 ^-- SIGN
```

*** Note 1: Code1 14 and 15 normally letters N and O are not available for use and are used for 14 = 'In=','Mn=','Pn=','Qn=' commands and 15 = 'Mn|=','Mn^=','Mn&=','Mn==' commands.

Command Type = 1

Sub-type A: ENABLE/DISABLE PLC

(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
7  6  5  4  3  2  1  0 |  N. U  |^ |1 |  CODE1 = 30  | 1 = ENA/DIS
      PLC #              |  ^
                         |  |--- Start of line bit
                         |-- CODE2 - 0/1 = ENA/DIS PLC
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8
          PLC #
```

Sub-type B: ABS/INC
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
Not Used         |  ^  |   N. U.    | 1|  CODE1 = 30  | 2 = FRAX/ABS/INC
                   0 = ABS            ^
            CODE2 - 1 = INC           |---Start of line bit
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
Z     Y     X     W     V     U     C     B     A  R
          ABS/INC AXIS ( Set bit = 1 )
```

Sub-type C: FRAX
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
Not Used         |  ^  |  N. U.     | 1|  CODE1 = 30  | 2 = FRAX/ABS/INC
             CODE2  2 = FRAX          ^
                                      |---Start of line bit
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
Z     Y     X     W     V     U     C     B     A
          FRAX AXIS ( Set bit = 1 )
```

Sub-type D: HOME & HOMEZ
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
 Not Used                         | 1| CODE1 = 30   | CODE2 3,6 =
                                                      HOME,HOMEZ
                                    ^                      4,5 = Not Used
                                    |---Start of line bit
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
                                                 8  7  6  5  4  3  2  1
                                                  MOTOR # (Set bit = 1)
```

Sub-type E: CIRn, SPLINEn, CCn, TSELn, ADISn, AROTn, IDISn, IROTn
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  CODE2               |     N. U. | 1| CODE1 = 30   | 0
     ^                              ^
     |                              |---Start of line bit
     |
   CODE2 0,4,10,11,12,13,14,15 = CIRn,SPLINEn,CCn,TSELn,ADISn,AROTn
                                 IDISn,IROTn
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
S | "n" = Integer number value  = 1 - 1023 & value of 1024 =  0
^-- SIGN (Two's complement)
```

Sub-type F: LINEAR, NORM, PSET, STOP, BSTART, BSTOP, WAIT, RAPID, TINIT
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  CODE2               |  N. U.    | 1| CODE1 = 30   | 0
     ^                              ^
     |                              |---Start of line bit
     |
   CODE2 1,2,3,5,6, = LINEAR, NORM, PSET, STOP, BSTART,
         7,8,9,16   = BSTOP, WAIT, RAPID, TINIT
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
0 |   0 = No DATA
```

Command Type = 2

Sub-type A: In=, Mn=, Pn=, Qn=
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  Integer "n"= 0 - 1023     |CODE2| 1| CODE1 = 14   | EXPONENT(6 BIT) |
                               ^    ^
                               |    |---Start of line bit
                               |
                              CODE2   0 = In, 1 = Mn, 2 = Pn, 3 = Qn
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
 S| FRACTIONAL PART OF FLOAT MANTISSA (24 BITS)
 ^-- SIGN
```

Sub-type B: Mn|=, Mn^=, Mn&=, Mn==
(24-Bit Word 1)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  Integer "n"= 0 - 1023     |CODE2| 1| CODE1 = 15   | EXPONENT(6 BIT) |
                               ^    ^
                               |    |---Start of line bit
                               |
                              CODE2 0 = Mn|=, 1 = Mn^=, 2 = Mn&=, 3 = Mn==
```

(24-Bit Word 2)

```
------------------------------------------------------------------------
23|22|21|20|19|18|17|16|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
 S| FRACTIONAL PART OF FLOAT MANTISSA (24 BITS)
 ^-- SIGN
```