REFERENCE GUIDE

PMAC Quick Reference

Reference Guide for PMAC Products

3A0-PMACQR-xPRx

December 3, 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

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.

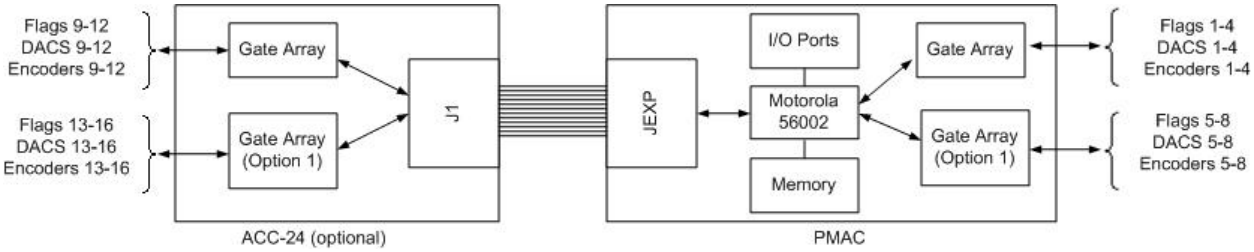
In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them 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 directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION

Description of PMAC

PMAC, pronounced *Pe'-MAC,* stands for Programmable Multi-Axis Controller. It is a family of high-performance servo motion controllers capable of commanding up to eight axes of motion simultaneously with a high level of sophistication.



There are five hardware versions of PMAC: the PMAC PC, the PMAC Lite, the PMAC VME, the PMAC STD and the PMAC Mini. These cards differ from each other in their form factor, the nature of the bus interface, and in the availability of certain I/O ports.

- Motorola's Digital Signal Processor (DSP) DSP56k is the CPU for PMAC and it handles all the calculations for all eight axes.
- The registers in PMAC's DSPGATE Gate-Array ICs are mapped into the memory space of PMAC's processor. Each DSPGATE contains four consecutively numbered channels; there may be up to four DSPGATEs in a PMAC system, for up to 16 channels.
- There are two types of servo DSPGATE Gate-Array ICs: The PMAC type that allows only the control of analog amplifiers with ±10V command signals and the PMAC2 type that is capable also of digital direct PWM or stepper command signals.
- Each PMAC channel provided by a PMAC DSPGATE has one DAC output, one encoder input and four dedicated flag inputs: two end-of-travel limits, one home input and one amplifier fault input.
- Any PMAC can control up to eight motors or axis as long as enough channels are provided. Every PMAC contains one DSPGATE, which has channels 1 through 4 (PMAC Mini has only two channels). If Option 1 is ordered (not available on PMAC Lite or PMAC Mini), a second DSPGATE is provided, which has channels 5 through 8. If Acc-24 is ordered (not available on PMAC STD), a third DSPGATE is provided which has channels 9 through 12. If Acc-24 Option 1 is ordered as well (not available on PMAC STD), a fourth DSPGATE is provided, which has channels 13 through 16.
- PMAC has its own memory and microprocessor. Therefore, any version of PMAC may run as a standalone controller or a host computer may command it either over a serial port or a bus port.

Types of PMAC

PMAC PC or PMAC VME Features

Standard Features	
Motorola DSP 56k digital signal processor	Linear and circular interpolation
Four output digital-to-analog (DAC) converters	256 motion programs capacity
Four full encoder channels	Asynchronous PLC program capability
16 general purpose I/O, OPTO-22 compatible	Rotating buffer for large programs
Multiplexer port for expanded I/O	36-bit position range (+/- 64 billion counts)
Overtravel limit, home, amplifier fault/enable flags	16-bit DAC output resolution
Display port for LCD and VFD displays	S-curve acceleration and deceleration
Bus and/or RS-422 control	Cubic trajectory calculations, splines
Stand-alone operation	Electronic gearing
G-code command processing for CNC	Advanced PID servo motion algorithms

Optional Features	
Up to 16 digital-to-analog (DAC) converters outputs	Yaskawa absolute encoders inputs
Up to 16 full encoder channels	Analog feedback inputs
8Kx16 dual-ported RAM	MLDTs feedback inputs
Flash memory (no battery)	Parallel binary feedback
40, 60 or 80 MHz CPU	Optically isolated encoder inputs
Extended (pole-placement) servo algorithm	RS-232 or RS-422 serial communication converters
Super-high accuracy clock crystal (<10 ppm)	Analog-to-digital converted inputs
Voltage-to-frequency (V/F) converters	On-board voltage to frequency converter
12-bit resolver-to-digital converter inputs	Up to a total of 2048 multiplexed I/O points
Sinusoidal encoder feedback inputs	Up to 100 meters remote I/O operation

PMAC PC

Recommended for applications with more than four channel requirements in either a PC based or stand alone environment. More than four channels can be used for more than four motors operation, dual-feedback axis (two encoder input each) or commutated motors (two DACs each). For three or four channels applications, the PMAC Lite board is suggested instead.

PMAC Lite

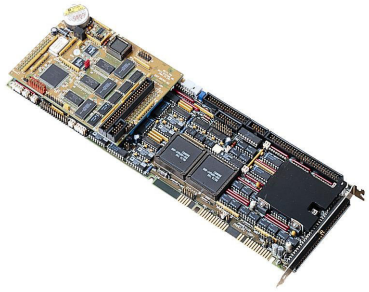
The PMAC Lite is recommended for applications with three or four channel requirements in either a PC based or stand alone environment. The term Lite stands for the limitation of only one DSPGATE Gate-Array IC on board. The number of channels can always be expanded from 4 to 12 through the use of an Acc-24P. The PMAC Lite board is provided also in a stand-alone box, the PMAC Pack, complete with power supplies and connectors. For one or two channels applications, the PMAC Mini board is suggested instead.

PMAC VME

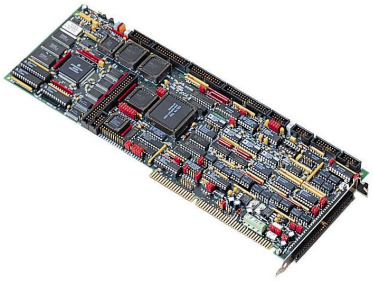
With the same features as the PMAC PC, the PMAC VME is the only option for VME based applications. The PMAC VME can be ordered with either four or maybe axes (Option 1). The dual-ported RAM option in a PMAC VME is on-board.

PMAC STD

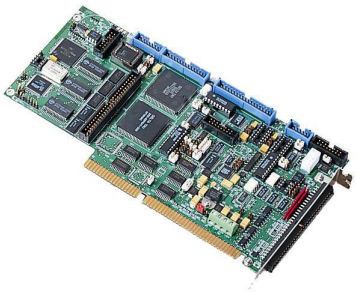
With the same features as the PMAC PC, the PMAC STD is the only option for STD based applications. The dual-ported RAM option is not available for the PMAC STD and it is limited to eight channels, no Acc-24 is available for it.



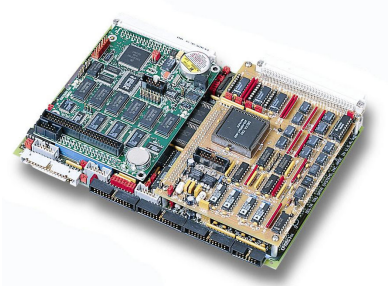
PMAC PC



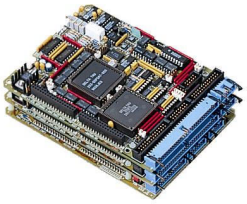
PMAC Lite



PMAC Mini

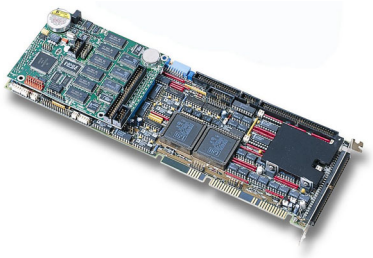


PMAC VME

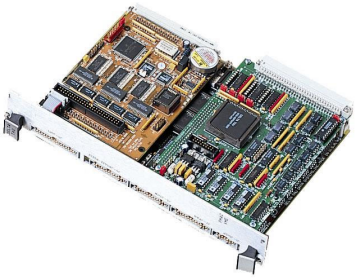


PMAC STD

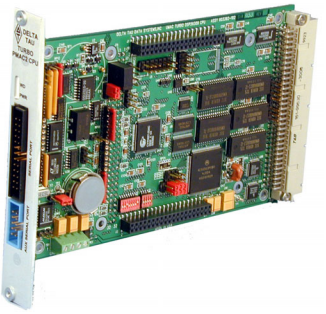
PMAC Pack



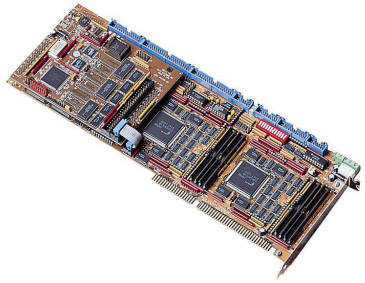
Turbo PMAC PC



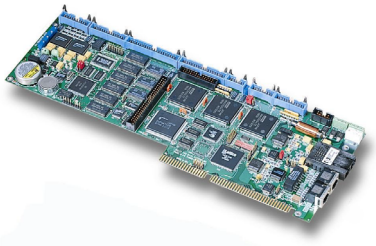
Turbo PMAC VME



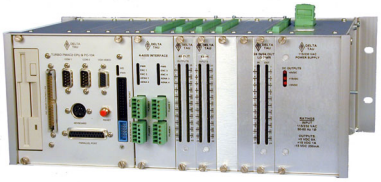
Turbo PMAC2 3U



Turbo PMAC2 PC



Turbo PMAC2 PC Ultralite



UMAC Turbo System

PMAC Mini

The PMAC Mini is recommended for applications with one or two channel requirements in either a PC based or stand alone environment.

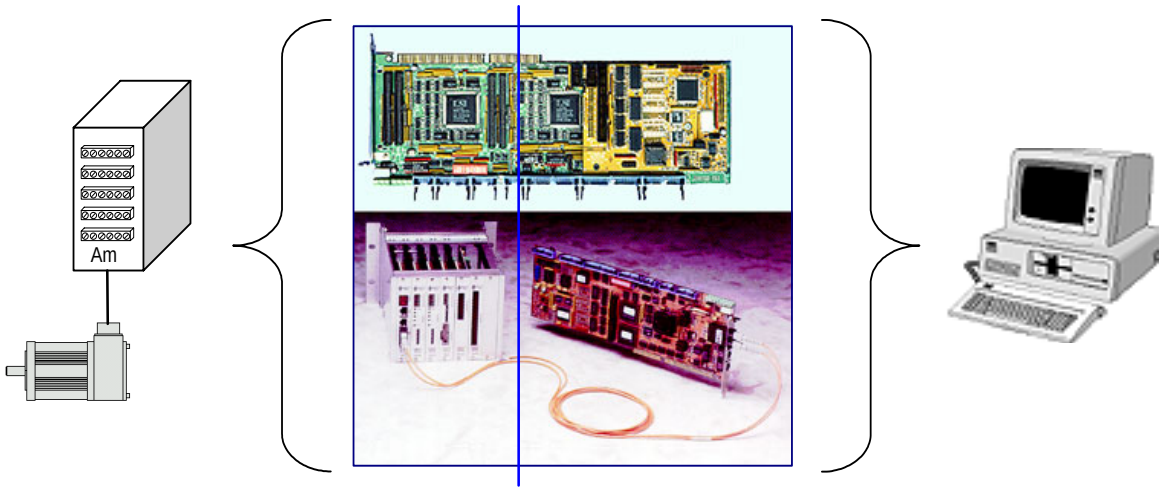
The dual-ported RAM option in a PMAC Mini is on-board. Two extra full encoder channels (for a total of four on-board) can be used for dual feedback applications or with the two optional voltage-to-frequency (V/F) converters, for stepper drivers or hybrid amplifiers control. There is no control panel port or bus interrupt in the PMAC Mini board. The PMAC Mini board is provided also in a stand-alone box, the Mini Pack, complete with power supplies and connectors.

PMAC2

PMAC2 is available in either PC, PCI, or VME formats. It is suggested for applications that require a digital amplifier control (direct PWM signals) or applications with a combination of analog and digital axis. PMAC2 is recommended also for the use of its built-in features that are optional in PMAC (1): pulse and direction outputs, MLDT inputs, optional 12-bits analog to digital inputs, two extra encoder inputs, improved position compare/capture feature and one channel of parallel feedback.

The PMAC2 is available with four or eight axes, with only four axes as the PMAC2 Lite and with only two axes as the PMAC2 Mini.

PMAC2 Ultralite



The term Ultralite stands for no DSPGATE Gate-Array ICs on board of this kind of PMAC2. The ASICs are located in a different set of boards, usually remotely located from PMAC2, referred as MACRO stations. In fact, the PMAC2 Ultralite in combination with the MACRO station can be seen as a PMAC2 divided in two halves: the central processing portion that contains the DSP processor and the distributed circuitry that connects to motors, amplifiers and different I/O points.

The PMAC2 Ultralite and the MACRO (Motion And Control Ring Optical) stations are linked with a fiber optic or twisted pair connection. This clever distribution of components brings many benefits: drastic reduction of wiring complexity, elimination of interference by electromagnetic noise and long distance connections (3000 m, ~2 miles with glass fiber).

Turbo PMAC Family

The Turbo PMAC is based in the 56300 Motorola DSP processor. Its power and speed allows handling up to 32 axes in up to 16 different coordinate systems. Compared with other PMACs, the Turbo PMAC has a highly improved lookahead feature that allows tighter control of acceleration and more accurate cornering profiles.

Motion programs and PLCs developed for other versions of PMAC are compatible with Turbo PMAC. The main difference in the setup is the increased number of variables necessary to control up to 32 axes. The main Turbo PMAC board has the necessary hardware to connect up to eight channels. The number of channels could be expanded from 8 to 40 by means of either the Acc-24P or Acc-24P2 for PMAC style or PMAC2 respectively. The Turbo PMAC2 is also provided in a 3U format and it is the main component of the UMAC (Universal Motion and Automation Controller) products.

PMAC Connectors and Indicators

Display Port Outputs (JDISP Port)

The JDISP connector (J1) connects the PMAC to the Acc-12 or Acc-12A liquid crystal displays, or of the Acc-12C vacuum fluorescent display. Both text and variable values may be shown on these displays through the use of the **DISPLAY** command, executing in either motion or PLC programs.

Control-Panel Port I/O (JPAN Port)

The JPAN connector (J2 on PMAC PC, Lite, VME, and top board of PMAC STD) is a 26-pin connector with dedicated control inputs, dedicated indicator outputs, a quadrature encoder input, and an analog input. The control inputs are low true with internal pull-up resistors. They have predefined functions unless the Control Panel Disable I-Variable (I2) has been set to 1. If this is the case, they may be used as general-purpose inputs by assigning an M-Variable to their corresponding memory-map locations (bits of Y address $FFC0).

Thumbwheel Multiplexer Port I/O (JTHW Port)

The Thumbwheel Multiplexer Port, or Multiplexer Port, on the JTHW (J3) connector has eight input lines and eight output lines. The output lines can be used to multiplex large numbers of inputs and outputs on the port, and Delta Tau provides accessory boards and software structures (special M-Variable definitions) to capitalize on this feature. Up to 32 of the multiplexed I/O boards may be daisy-chained on the port, in any combination.

Serial Port Connection

For serial communications, use a serial cable to connect the PC's COM port to PMAC's serial port connector (J4 on PMAC PC, Lite, and VME; J1 on PMAC STD's bottom board). Delta Tau provides cables for this purpose: Acc-3D connects PMAC PC or VME to a DB-25 connector; Acc-3L connects PMAC Lite to a DB-9 connector; and Acc-3S connects PMAC STD to a DB-25 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for a particular setup.

General-Purpose Digital Inputs and Outputs (JOPTO Port)

PMAC's JOPTO connector (J5 on PMAC PC, Lite, and VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Delta Tau's Acc-21F is a six-foot cable for this purpose. The PMAC STD has a different form of this connector from the other versions of PMAC. Its JOPT connector (J4 on the base board) has 24 I/O, individually selectable in software as inputs or outputs.

Machine Connectors

The primary machine interface connector is JMACH1 (J8 on PMAC PC, J11 on PMAC Lite, P2 on PMAC VME, J4 on PMAC STD top board). It contains the pins for four channels of machine I/O: analog outputs, incremental encoder inputs, and associated input and output flags, plus power-supply connections. The next machine interface connector is JMACH2 (J7 on PMAC PC, P2A on PMAC VME, J4 on the middle board of an 8-channel PMAC STD, not available on a PMAC Lite). Essentially it is identical to the JMACH1 connector for one to four more axes. It is present only if the PMAC card has been fully populated to handle eight axes (Option 1), because it interfaces the optional extra components.

LED Indicators

PMACs with the Option CPU have three LED indicators: red, yellow, and green. The red and green LEDs have the same meaning as with the standard CPU: when the green LED is lit, this indicates that power is applied to the +5V input; when the red LED is lit, this indicates that the watchdog timer has tripped and shut down the PMAC.

The new yellow LED located beside the red and green LEDs, when lit, indicates that the phase-locked loop that multiplies the CPU clock frequency from the crystal frequency on the Option CPU is operational and stable. This indicator is for diagnostic purposes only; it may not be present on all boards.

Working with PMAC

When used for the first time, the card must be configured for a specific application, using both hardware and software features, in order to run that application properly. PMAC is shipped from the factory with defaults set in hardware and software set up to be satisfactory for the most common application types. Working with PMAC is very simple and its ease of use and power is based in the following features:

- A clever interrupt-driven scheme allows every task, each motion program and PLC, to run independently of each other.
- Pointer M-Variables allow monitoring virtually any register in PMAC's memory from different sources: motion programs, PLCs or the host computer.
- Communications are activated continuously. At any moment, any variable or status command could be interrogated.
- Up to eight axes could be either synchronized together, controlled individually or in any combination in between.
- Data gathering and reporting functions allows saving data such as motion trajectories, velocity profiles or any set of variables for later analysis and plot.

Hardware Setup

On the PMAC, there are many jumpers (pairs of metal prongs), called E-points (on the bottom board of the PMAC STD they are called W-points). Some have been shorted together; others have been left open. These jumpers customize the hardware features of the board for a given application. Check each jumper configuration using the appropriate hardware reference for the particular PMAC being set. Further instructions for the jumper setup can be found in the PMAC User manual. After all the jumpers have been properly set, PMAC can be installed either inside the host computer or linked with a serial cable to it.

Software Setup

PMAC has a large set of Initialization parameters (I-Variables) that determine the personality of the card for a specific application. Many of these are used to configure a motor properly. Once setup, these variables may be stored in non-volatile EAROM memory (using the **SAVE** command) so the card is always configured properly (PMAC loads the EAROM I-Variable values into RAM on power-up).

The easiest way to program, setup and troubleshoot PMAC is by using the PMAC Executive Program PEWIN and its related add-on packages P1Setup and PMACPlot. PEWIN has the following main tools and features:

- The terminal window is the main channel of communication between the user and PMAC
- Watch window for real-time system information and debugging
- Position window for displaying the position, velocity and following error of all motors on the system
- Several ways to tune PMAC systems
- Interface for data gathering and plotting

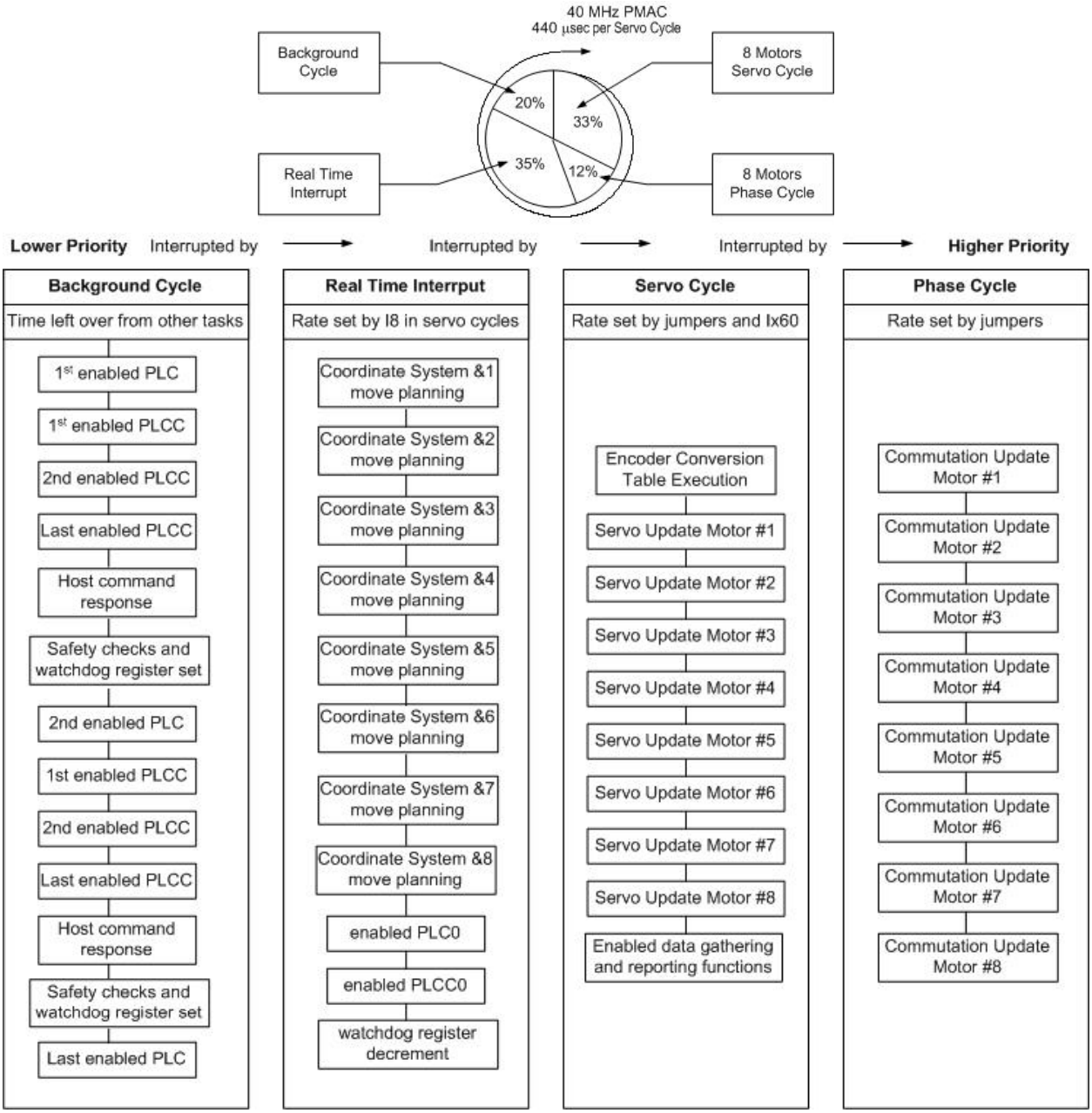
In PEWIN, the value of an I-Variable may be queried simply by typing in the name of the I-Variable. For instance, typing **I900<CR>** causes the value of the I900 to be returned. Change the value by typing in the name, an equals sign, and the new value (e.g. **I900=3<CR>**). Remember that if any I-Variables are changed during this setup, use the **SAVE** command before powering down or reset the card, or the changes that have been made will be lost.

Programming PMAC

Motion or PLCs programs are entered in any text file and then downloaded with PEWIN to PMAC. PEWIN provides a built-in text editor for this purpose but any other text editor could be used conveniently. Most PMAC commands can be issued from any terminal window communicating with PMAC. Online commands allow, for example, to jog motors, change variables, report variables values, start and stop programs, query for status information and even write short programs and PLCs. In fact, the downloading process is just a sequence of valid PMAC commands sent line by line by PEWIN from a particular text file.

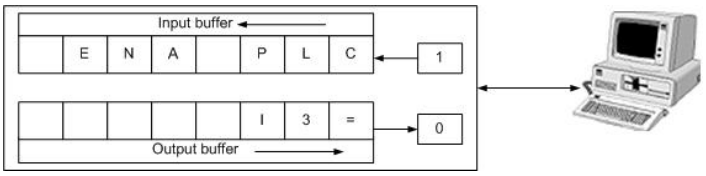
PMAC Tasks

As an example, a 40 MHz PMAC could perform the following tasks with the estimated percentage of the total computational power as indicated:



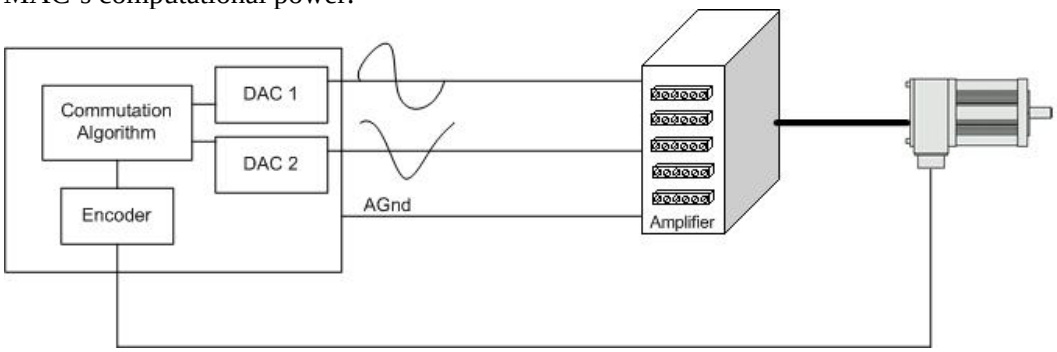
Single Character I/O

Bringing in a single character from, or sending out a single character to, the serial port or host port (PC or STD) is the highest priority in PMAC. This task takes only 200 nsec per character, but having it at this high priority ensures that the host cannot outrun PMAC on a character-by-character basis. This task is never a significant portion of PMAC's total calculation time. Note that this task does not include processing a full command; that happens at a lower priority (see the Background Tasks section).



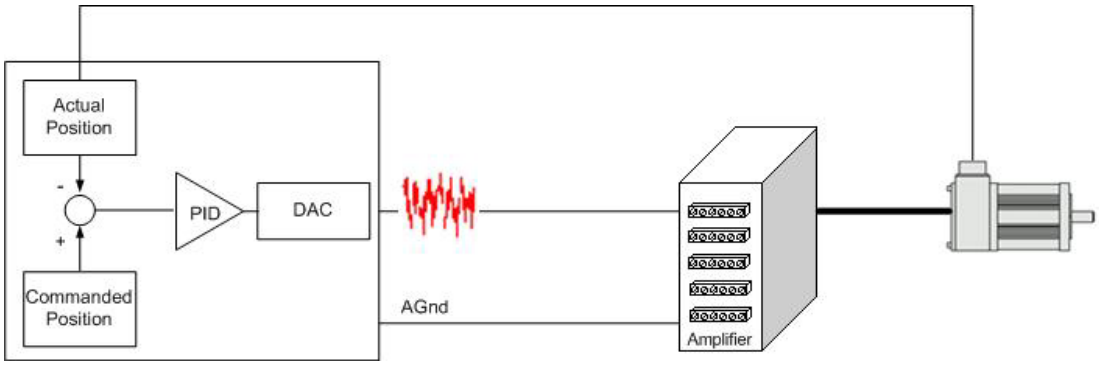
Commutation Update

The commutation (phasing) update is the second highest priority on PMAC. In a 20 MHz PMAC, this task takes 3 µsec per update cycle for each motor commutated by PMAC (Ix01=1). The master clock frequency and jumpers E98, E29-E33, determines the frequency of this task. The default update frequency is 9 kHz (110 µsec cycle). At the default, the commutation of each motor takes approximately 3% of PMAC's computational power.



Servo Update

The servo update – computing the new commanded position, reading the new actual position, and computing a command output based on the difference between the two – is the third highest priority on PMAC. In a 20 MHz PMAC, this task takes 30 µsec per update cycle for each activated motor (Ix00=1) plus about 30 µsec for general servo tasks such as the encoder conversion table. The master clock frequency and jumpers E98, E29-E33, E3-E6 determine the frequency of this task. The default update frequency is 2.26 kHz (442 µsec cycle). At the default, the servo update of each motor takes approximately 7% of PMAC's computational power.

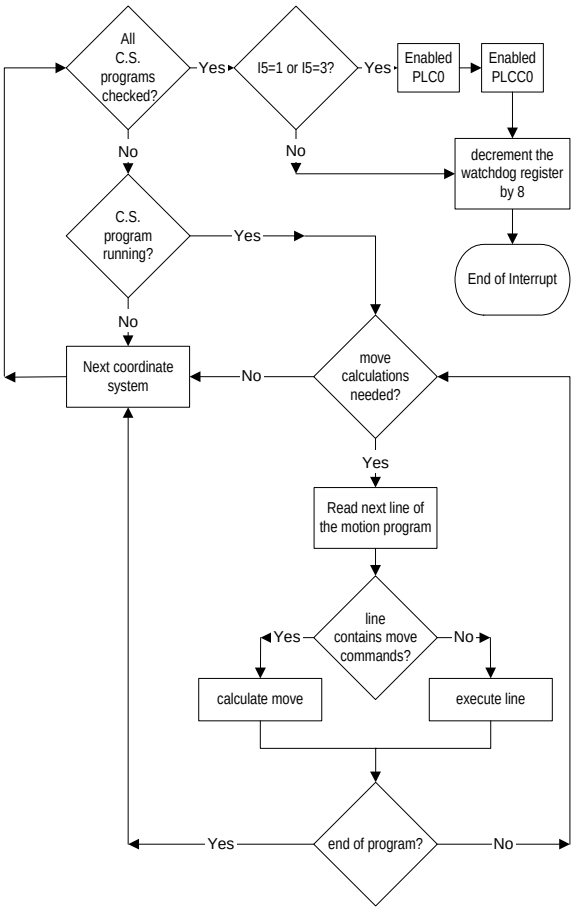


VME Mailbox Processing

Reading or writing a block of up to sixteen characters through the VME mailbox registers is the fourth highest priority in PMAC. The host controls the rate at which this happens. This never takes a significant portion of PMAC's computational power.

Real-Time Interrupt Tasks

The real-time interrupt (RTI) tasks are the fifth highest priority on PMAC. They occur immediate after the servo update tasks at a rate controlled by parameter I8 (every I8+1 servo update cycles). There are two significant tasks occurring at this priority level: PLC 0 / PLCC0 and motion program move planning.



PMAC will scan the lines of each program running in the different coordinate systems and will calculate the necessary number of move commands.

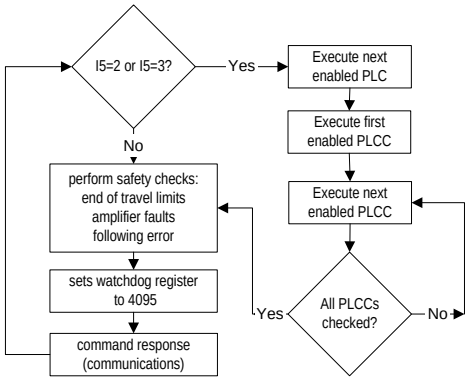
The number of move commands of pre-calculation can either be zero, one or two and depending on the type of motion commands and the mode in which the program is being executed.

Non-move commands are executed immediately as they are found. The scan of any given motion program will stop as the necessary number of moves is calculated. It resumes when previous move commands are completed and more move-planning calculations are required.

In the execution of a motion program, if PMAC finds two jumps backward (toward the top) in the program while looking for the next move command, PMAC will pause execution of the program and not try to blend the moves together. It will go on to other tasks and resume execution of the motion program on a later scan. Two statements can cause such a jump back: **`ENDWHILE`** and **`GOTO`** (**`RETURN`** does not count).

Background Tasks

In the time not taken by any of the higher-priority tasks, PMAC will be executing background tasks. There are three basic background tasks: command processing, PLC programs 1-31, and housekeeping. The frequency of these background tasks is controlled by the computational load on PMAC: the more high-priority tasks are executed, the slower the background tasks will cycle through; and the more background tasks there are, the slower they will cycle through.



Each PLC program executes one scan (to the end or to an **`ENDWHILE`** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping, and respond to a host command, if any.

All enabled PLCC programs execute one scan (to the end or to an **ENDWHILE** statement) starting from lowest numbered to highest uninterrupted by any other background task (although it can be interrupted by higher priority tasks). At power-on\reset, PLCC programs run after the first PLC program runs.

The receipt of a control character from any port is a signal to PMAC that it must respond to a command. The most common control character is the carriage return (<**CR**>), which tells PMAC to treat all the preceding alphanumeric characters as a command line. Other control characters have their own meanings, independent of any alphanumeric characters received. Here PMAC will take the appropriate action to the command, or if it is an illegal command, it will report an error to the host.

Between each scan through each background PLC program, PMAC performs its housekeeping duties to keep itself properly updated. The most important of these are the safety limit checks (following error, overtravel limit, fault, watchdog, etc.) Although this happens at a low priority, a minimum frequency is ensured because the watchdog timer will trip, shutting down the card, if this frequency gets too low.

Observations

PMAC has an on-board watchdog timer circuit whose job it is to detect a number of conditions that could result in dangerous malfunction. At the default settings, if the RTI frequency were to drop below about 50 Hz, or the background cycle is not performed at least every 512 RTI cycles the timer would trip. The purpose of this two-part control of the timer is to make sure all aspects of the PMAC software are being executed, both in foreground (interrupt-driven) and background. If anything keeps either type of routine from executing, the watchdog will fail quickly.

PLC0 or PLCC0 are meant to be used for only a very few tasks (usually a single task) that must be done at a higher frequency than the other PLC tasks. The PLC 0 will execute every real-time interrupt as long as the tasks from the previous RTI have been completed. PLC 0 is potentially the most dangerous task on PMAC as far as disturbing the scheduling of tasks is concerned. If it is too long, it will starve the background tasks for time. The first thing to notice is that communications and background PLC tasks will become sluggish. In the worst case, the watchdog timer will trip, shutting down the card, because the housekeeping task in background did not have the time to keep it updated.

Although it is very rare for a motion program to cause a watchdog failure, this does happen on occasion. If there is an empty (no-motion) loop, the motion program acts much like a PLC 0 during this period. These empty loops, which are used usually to wait for a certain condition, provide fast response to the change in condition, but their fast repetition occupies a lot of CPU time, and can starve the background tasks for time. Particularly if several coordinate systems are executing empty loops at the same time, serious background time limitations can be created which can be severe enough to trip the watchdog timer.

If there are a huge number of lines of intensive calculations (e.g. 100) before any move or dwell is encountered, there can be such a long time before background calculations are resumed (more than 512 RTI cycles) it is possible to trip the watchdog timer. If this problem occurs, the calculations should be split apart with short DWELL commands to give other tasks time to execute.

It is possible to use compiled PLCC programs for faster execution. The faster execution of the compiled PLCs comes from two factors: first, from the elimination of interpretation time, and second, from the capability of the compiled PLC programs to execute integer arithmetic. The space dedicated to store up to 32 compiled PLC programs, however, is limited to 15K (15,360) 24-bit words of PMAC memory; or 14K (14,336) words if there is a user-written servo as well.

In between each scan of each individual background interpreted PLC program, PMAC will execute one scan of all active background compiled PLCs. This means that the background compiled PLCs execute at a higher scan rate than the background interpreted PLCs. For example, if there are seven active background interpreted PLCs, each background compiled PLC will execute seven scans for each scan of a background interpreted PLC.

Most of the housekeeping functions are safety checks such as following error limits and overtravel limits. Since compiled PLCCs are executed at the same rate as the housekeeping functions, code to complement or replace these functions could be placed in a compiled PLCC. If, for example, an extra input flag is wanted for position capturing purposes either the end-of-travel limit inputs or the amplifier fault input could be used. The automatic check of the input flag could be disabled by an appropriate setting of the corresponding Ix25 variable and replaced by a PLCC code that will check a general purpose input where the amplifier fault or end-of-travel limit would be connected instead.

On power-up\reset, PLC programs are executed sequentially from 1 to 31. This makes PLC1, the first code executed, the ideal place to perform initialization commands like other PLCs disabling, motors phasing and motion programs start. After its execution, PLC1 could disable itself with the command DIS PLC1, running only once on power-up\reset.

Bits of the first word returned from the global status bits request command, **???**:

Bit 22 Real-Time Interrupt Re-entry: This bit is 1 if a real-time interrupt task has taken long enough so that it was still executing when the next real-time interrupt came (I8+1 servo cycles later). It stays at 1 until the card is reset, or until this bit is changed manually to 0. If motion program calculations cause this, it is not a serious problem. If PLC 0 causes this (no motion programs running) it could be serious.

Bit 20 Servo Error: This bit is 1 if PMAC could not complete its servo routines properly. This is a serious error condition. It is 0 if the servo operations have been completed properly.

Priority Level Optimization

Usually, PMAC will have enough speed and calculation power to perform all of the tasks asked of it without worry. Some applications will put a large demand on a certain priority level and to make PMAC run more efficiently. When PMAC begins to run out of time, problems such as sluggish communications, slow PLC/PLCC scan rates, run-time errors, and even tripping the watchdog timer can occur.

The active part of the Encoder Conversion Table is ended by the first Y word that is equal to all zeros. For an application with less than eight encoders (the default table converts the eight incremental encoder registers on the base PMAC), a last entry with all zeros in the Y word could be defined as necessary.

Check to see if everything performed in the Real Time Interrupt (RTI) is necessary or if some of it could be moved to a lower priority or slowed down. PLC0 could be done as PLCC1, or the RTI could be done every fourth or fifth servo cycle setting I8=3 or higher.

Large PLC programs can be split into a few shorter PLC programs. This increases the frequency of housekeeping and communications by giving more breaks in PLC scans.

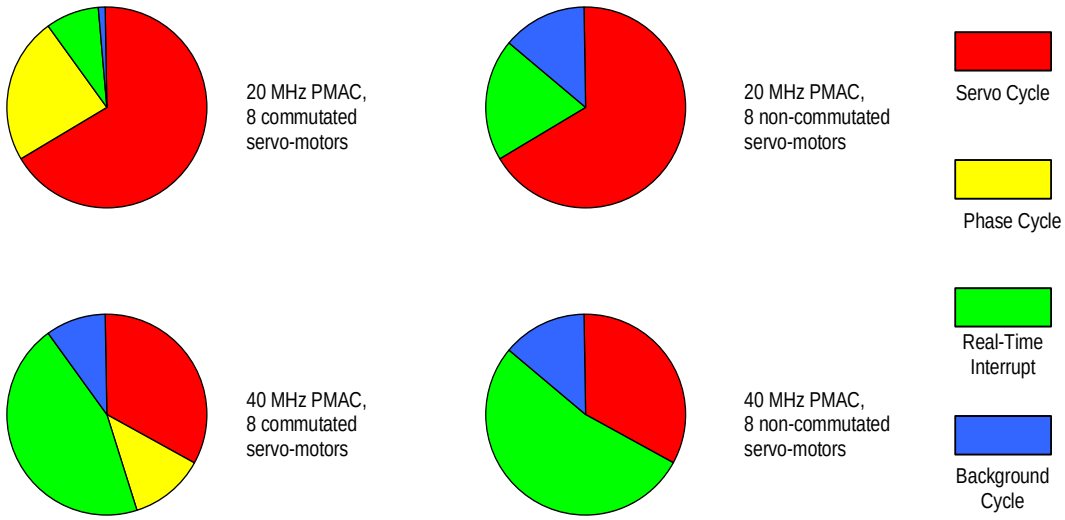
Motion program **WHILE (condition)WAIT** statements can be done as follows:

```
WHILE (condition)
      DWELL20
ENDWHILE
```

This will give more time to other RTI jobs such as Move Planning and PLC/PLCC0.

If routines of lower priority than the servo loop are not executing fast enough, consider slowing down the servo update rate (increasing the update time). The PMAC may be updating faster than is required for the dynamic performance needed. If so, processor time is being wasted on needless extra updates. For example, doubling the servo update time from 442 μsec to 885 μsec, virtually doubles the time available for motion and PLC program execution, allowing much faster motion block rates and PLC scan rates. This frequency change could be executed either by jumpers or individually per motor by means of the Ix60 variable.

A faster than 20 MHz PMAC will perform calculations faster, in proportion to the corresponding clock rate increase. In general, a clock rate increase is used to increase the real time interrupt (RTI) share of the total computational time available. These cases include applications where large move calculations are involved (small-moves contouring), maintaining the same servo-loop rate and therefore the same control performance.

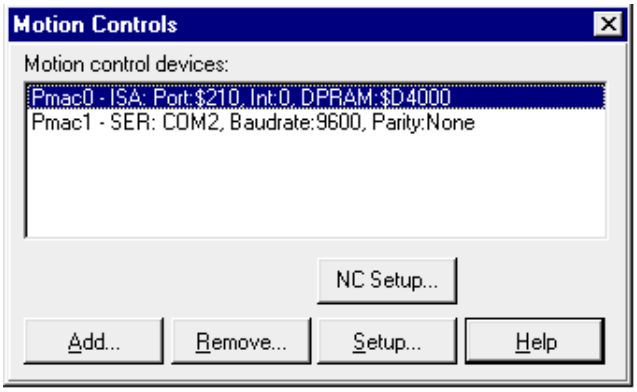


PMAC EXECUTIVE PROGRAM, PEWIN

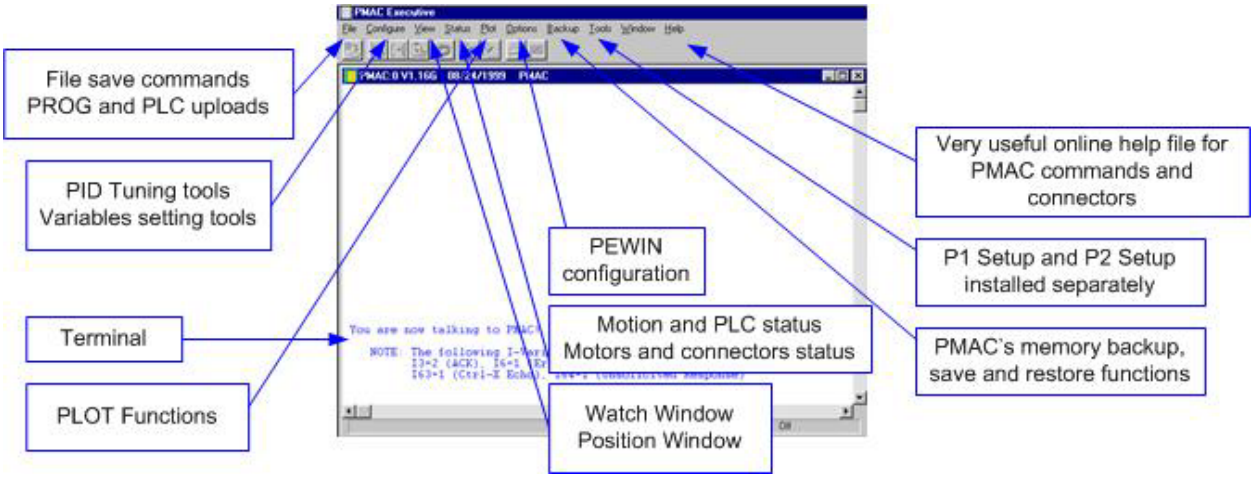
With PEWIN, PMAC can be configured and controlled. PEWIN is designed as a development tool for creating and managing PMAC implementations. It provides a terminal interface to the PMAC and a text editor for writing and editing PMAC motion programs and PLC programs. Additionally, PEWIN contains a suite of tools for configuring and working with PMAC and its accessories including interfaces for jogging motors, extensive system utilities, screens for viewing various PMAC variables and status registers.

Configuring PEWIN

1. Define a new device using the MOTIONEXE.EXE application provided.



2. Open PEWIN and select the Open Terminal menu. Select the device created in the previous step.

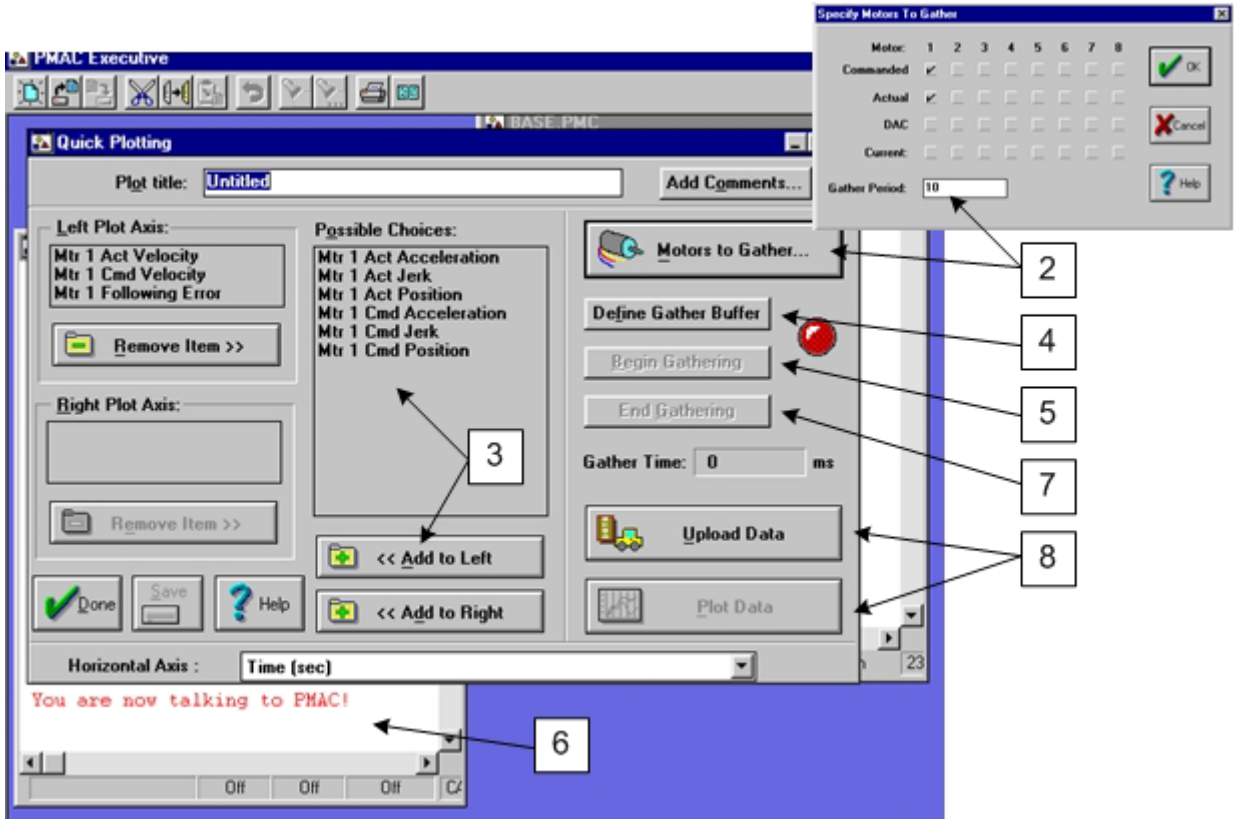


3. The colors and different options can be set through the **Preferences** command present in the Options menu. Disable the automatic status-reporting feature by un-checking the **Enable Terminal Status Bar** from the Terminal preferences.

Quick Plot Feature

To run the quick plot feature:

1. Press **ALT+P** and press **Enter**.
2. Select the motors and the feature to gather.
3. Select what to plot from the possible choices and then press **Add to left** or **Add to right**.
4. Press the **Define Gather Buffer** button.
5. Press the **Begin Gathering** button.
6. Click on the terminal part of the screen and run the motion program or **Jog** command.
7. Press the **End Gathering** button when the motion is completed.
8. First press the **Upload Data** button and then the **Plot Data** button.



The Plot feature relies on the PMAC gathering functions. It is useful for analyzing motion profiles and trajectories. Simulating an X-Y plot graphically can be an important aid in understanding the set of parameters involved in a circular interpolation move.

Saving and Retrieving PMAC Parameters

It is important to save the complete set of PMAC parameters in the host computer periodically. In case of a failure or replacement, a single file created this way will allow restoring all the variables and programs necessary for the particular application. To activate this function click on the terminal window, press **CTRL+B** for the Backup menu, select **Save Configuration** and **Global Configuration**. Select a name to be saved as. Usually, the date is included as part of the file name for later identification. For example, PMAC0112 has four digits for the application identifier and four digits for the date.

After the file is saved, verify it with the feature part of the same pull-down menu. This will make sure PMAC's memory matches the recently saved file and therefore that it is a valid restoring file.

To restore a configuration simply select **Restore** from the same Backup menu. Verify PMAC's memory after the restore function as well.

The Watch and Position Windows

The position window is accessed through the **`POSITION`** command of the View menu, or **ALT+V** and **P** from the terminal window. It is a convenient way to check PMAC parameters continuously, such as position velocity and following error. Right clicking on this window allows the items selections as well as its format and update period.

The Watch window of the same View menu performs a similar function. Instead of the motion-related parameters, any variable value in PMAC can be displayed constantly. Right clicking on this window allows selecting the display format from hexadecimal, decimal and binary reporting values.

Uploading and Downloading Files

These functions are accessible through the File menu. The uploading function is of great importance. With these functions, it is possible to open a text editor with the contents of the requested PLC, Motion Program, M-Variables definitions or values, I-Variables values, etc. With this function, what commands or values PMAC has in memory can be checked and IF conditions and WHILE loops are indented, making the program flow better. The File menu also activates a more interactive and complete editor utility, providing a way (also by the communication functions) to compile PLCs and download files including MACRO names.

Using MACRO Names and Include Files

PEWIN allows using custom names in place of the common names for variables and functions that PMAC expects (P, Q, M, I):

Example:

File downloaded

```
#define PUMP P1

OPEN PLC1 CLEAR
      PUMP=1
      DISABLE PLC1
CLOSE
```

Uploaded translated PMAC code

```
OPEN PLC 1 CLEAR
      P1=1
      DISPLC1
CLOSE
```

Make sure the **Support MACROs/PLCCs** option is checked before downloading. The MACRO must be defined before it can be used. In general, MACRO definitions are at the beginning of the text file. MACROs must be up to 255 valid ASCII characters and cannot have spaces in between (the underscore "_" is suggested in place of a space).

The MACRO definitions or any PMAC code can be placed in a separate file and be included with a single line in the text file. The file name must include a full path in order for PEWIN to find it.

Example: `#include "c:\deltatau\files\any.pmc"`

Downloading Compiled PLCCs

PLCCs are compiled by PEWIN in the downloading process. Only the compiled code gets downloaded to PMAC. Therefore, save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. Compiled PLCs are firmware dependent and must be recompiled when the firmware is changed in PMAC.

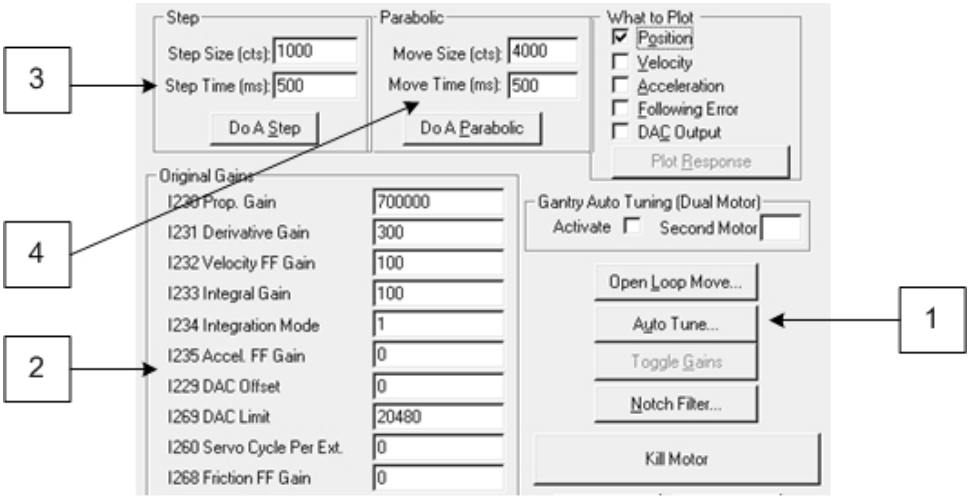
If more than one PLCC is programmed, all the PLCC code must belong to the same ASCII text file. PEWIN will compile all the PLCC code present on the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, all the other PLCCs that might have been present in memory will be erased, remaining only the last compiled code.

The multiple-file download feature of the PEWIN File menu allows the PLCC codes to be in different files. They will be combined by PEWIN in the downloading process.

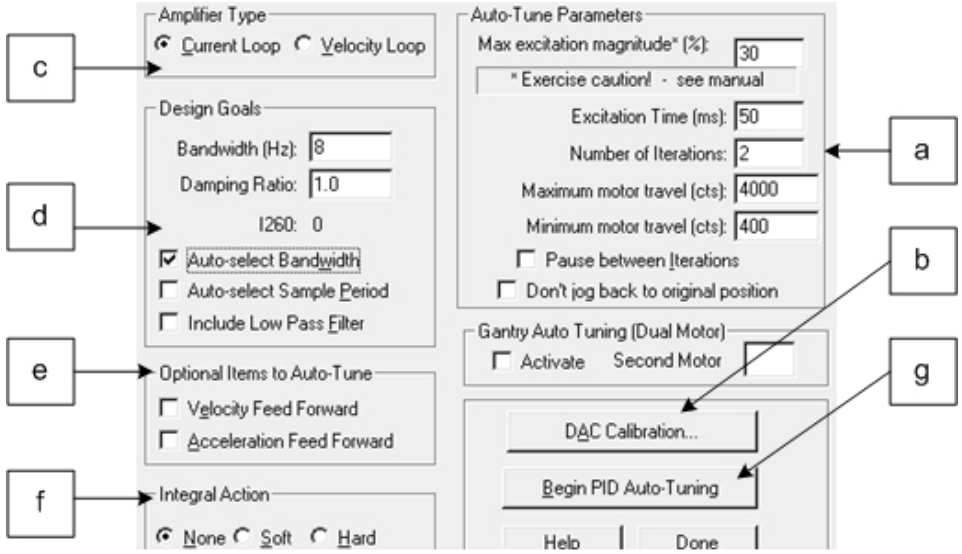
PID Tuning Utility

This function is accessible from the terminal window by pressing **ALT+C** from the Configure menu and T for Tuning. The Autotuning feature finds the PID parameters with virtually no effort. In most cases, the parameters are very close to optimal, and in some cases require further fine-tuning.

In this screen, press the **Page-Up** or **Page-Down** keys on the keyboard to select the motor number.

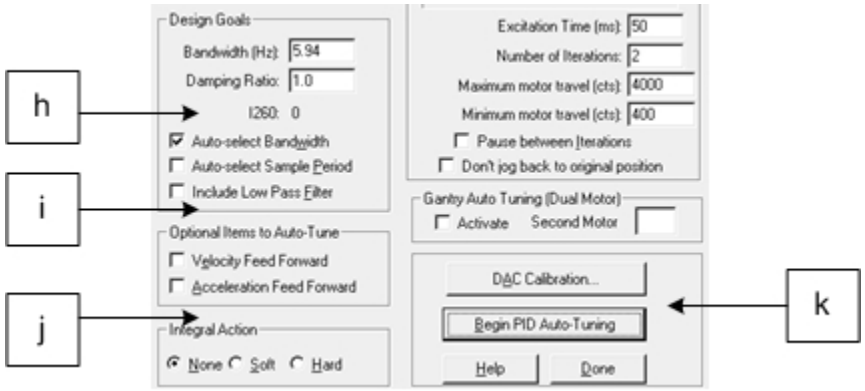


1. Select the **Auto Tune** feature. This is the first interaction to find a starting bandwidth parameter.



a. Make sure to read the PEWIN manual section related to the safety issues of this procedure.
b. Perform a DAC calibration if necessary.
c. Select the type of amplifier being tuned.
d. Let the Auto Tune select the bandwidth by checking Auto Select bandwidth.
e. Do not activate any feed forward parameters in this first pass.
f. Do not activate the integral action component in this first pass.
g. Start the first Auto Tuning interaction. Most likely the motor will move after **Begin** is clicked.

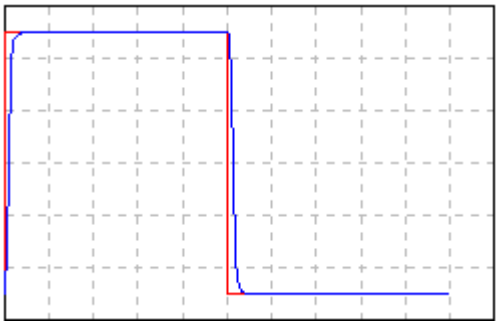
Second Interaction



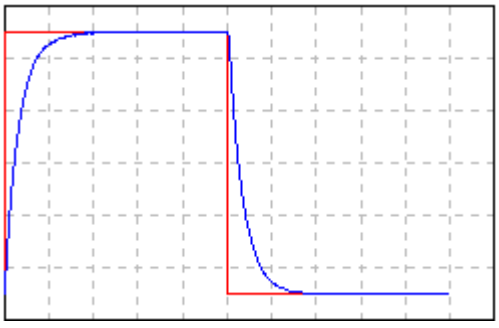
h. The calculated bandwidth can be increased up to three times. Uncheck the **Auto Select Bandwidth** this time.
i. Add the feed forward parameters as necessary.
j. Add the integral actions function as necessary.
k. Perform the second pass of the Auto Tuning. After it is completed, select **Implement Now** to activate the selected parameters.

2. After the Auto Tuning is completed, the PID parameters can be changed for a final fine-tuning if necessary.

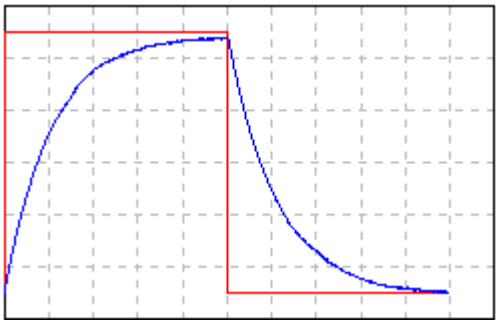
3. Perform a step response and use the following guidelines for the selection of the appropriate I-Variables:



Ideal Case
The motor closely follows the commanded position

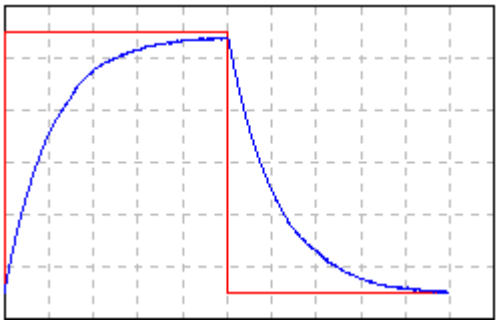


Position Offset
Cause: friction or constant force / system limitation
Fix: Increase K_I (Ix33) and maybe use more K_P(Ix30)



Sluggish Response

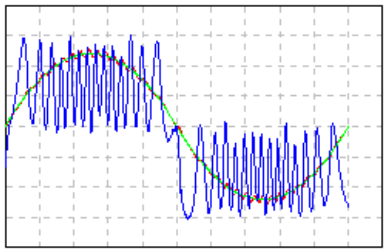
Cause: Too much damping or too little proportional gain
Fix: Increase K_P (Ix30) or decrease K_D (Ix31)



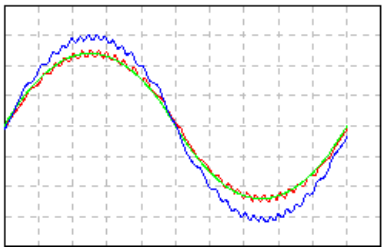
Overshoot and Oscillation

Cause: Too little damping or too much proportional gain
Fix: Decrease K_P (Ix30) or increase K_D (Ix31)

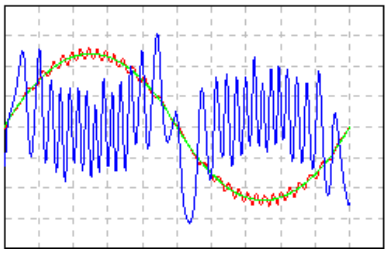
4. Perform a parabolic move and use the following guidelines for the selection of the appropriate I-Variables:



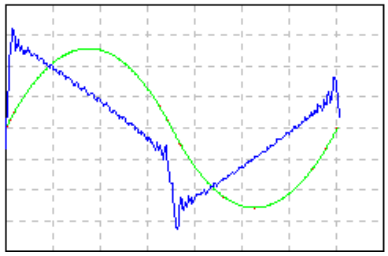
Ideal Case
The following error is reduced at minimum and is concentrated in the center, evenly along the move



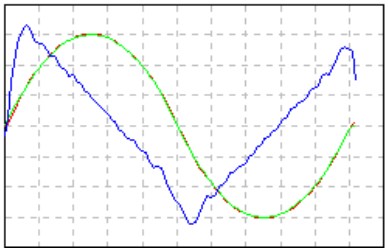
High vel \ FE correlation
Cause: damping
Fix: Increase K_{vel} (Ix32)



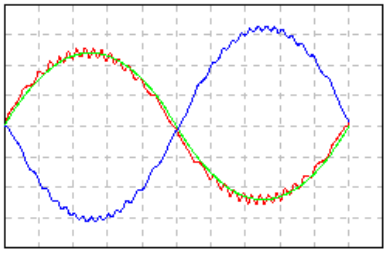
High vel \ FE correlation
Cause: friction
Fix: Increase Integral gain (Ix33) or Friction Feedforward (Ix68)



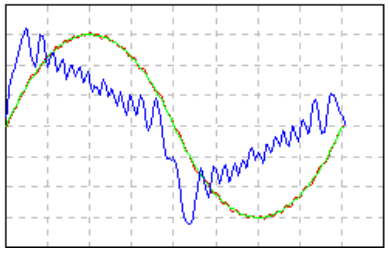
High acc \ FE correlation
Cause: Integral lag
Fix: Increase K_{aff} (Ix35)



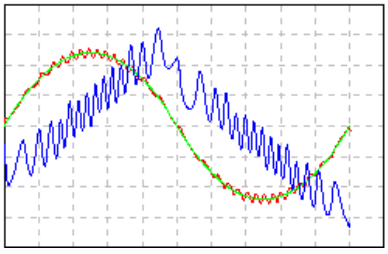
High acc \ FE correlation
Cause: Physical system limitations
Fix: Use less sudden acceleration



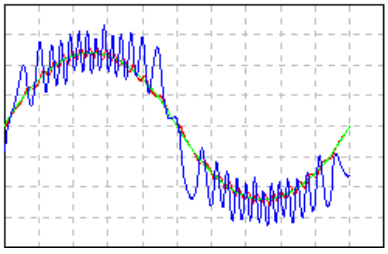
Negative vel \ FE correlation
Cause: Too much velocity FF
Fix: Decrease K_{vel} (Ix32)



High vel \ FE correlation
Cause: damping and friction
Fix: Increase K_{vel} (Ix32)



High acc \ FE correlation
Cause: Too much acc FF
Fix: Decrease K_{aff} (Ix35)



High vel\FE and acc\FEcorrelation
Cause: Integral lag and friction
Fix: Increase K_{aff} (Ix35)

Other Features

- Setup of the PMAC encoder conversion table
- Setup of the Notch and Low Pass Filter parameters
- Coordinate systems configurations
- Access to P1Setup and P2Setup (packages provided separately). These setup utilities provide a user-friendly approach for setting up and tuning PMAC (1), with P1Setup, or PMAC2 using P2Setup
- Online PMAC Software and Hardware help files
- Jog Ribbon and connector status
- Screens to display, organize or change I, P, Q and M variables
- Firmware downloading (through MOTIONEXE) for PMACs with flash memory.

INSTALLING AND CONFIGURING PMAC

Jumpers Setup

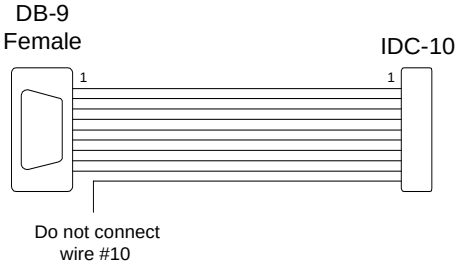
On the PMAC, there are many jumpers (pairs of metal prongs), called E-points (on the bottom board of the PMAC STD they are called W-points). Some have been shorted together; others have been left open. These jumpers customize the hardware features of the board for a given application. Each jumper configuration should be checked using the appropriate hardware reference for the particular PMAC being set. Further instructions for the jumper setup can be found in the PMAC User manual. After all the jumpers have been properly set, PMAC can be installed either inside the host computer or linked with a serial cable to it.

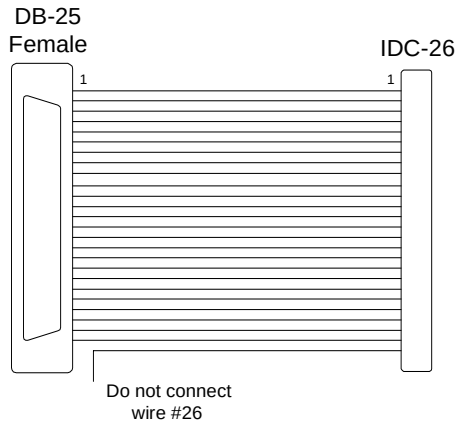
Serial Connections

For serial communications, use a serial cable to connect the PC's COM port to the PMAC's serial port connector (J4 on PMAC PC, Lite, and VME; J1 on PMAC STD's bottom board). Delta Tau provides cables for this purpose: Acc-3D connects PMAC PC or VME to a DB-25 connector; Acc-3L connects PMAC Lite to a DB-9 connector; and Acc-3S connects PMAC STD to a DB-25 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for a particular setup.

If using the Acc-26 Serial Communications converter, connect from the PC COM port to Acc-26 with a standard DB-9 or DB-25 cable and from Acc-26 to PMAC using the cable provided with Acc-26. Since the serial ports on PMAC PC and PMAC VME are RS-422, this accessory can be useful to provide the level conversion between RS-232 and RS-422 (communications is possible without this conversion, but at reduced noise margin). Because the conversion is optically isolated, the accessory also helps prevent noise and ground loop problems.

If a cable must be made, the easiest approach is to use a flat cable prepared with flat-cable type connectors as indicated in the following diagrams:





Establishing Host Communications

Either the Executive or Setup program can be used to establish initial communications with the card. Both programs have menus that tell the PC where to expect to find the PMAC and how to communicate with it at that location. If telling it to look for PMAC on the bus, also tell it PMAC's base address on the bus (this was set up with jumpers on PMAC). If telling it to look for PMAC on a COM port, tell it the baud rate (this was set up with jumpers or switches on the PMAC). Once the program knows where and how to communicate with PMAC, it will attempt to find PMAC at that address by sending a query command and waiting for the response. If it gets the expected type of response, it will report that it has found PMAC. If it does not get the expected type of response after several attempts, it will report that it has not found PMAC.

Terminal Mode Communications

Once the program reports that it has found PMAC, the program should be in terminal emulation mode, so that the PC is acting as a dumb terminal to PMAC. Check to see if a response is received by typing **I10<CR>.** (**<CR>** means carriage return — the **Enter** or **Return** key). PMAC should respond with a six or seven digit number. If the expected results are not received, check the following:

1. Make sure the green LED (power indicator) on PMAC's CPU board is ON. If it is not, find out why PMAC is not getting a +5V voltage supply.
2. Make sure the red LED (watchdog timer indicator) on PMAC's CPU board isOFF. If it is ON, make sure PMAC is getting very close to 5V supply – at less than 4.75V, or the watchdog timer will trip, shutting down the card. The voltage can be probed at pins 1 and 3 of the J8 connector (A1 and A2 on the PMAC VME). If the voltage is satisfactory, follow these steps:
   - Turn off PMAC or the Host computer where it is plugged into.
   - Place the Jumper E51 (the hardware re-initialization jumper) and turn PMAC back on.
   - If PMAC is in bootstrap mode, send a **<CONTROL-R>** character to PMAC to bypass the firmware download.
   - If communications are successful type **$$$***** and **SAVE** in the terminal window.
   - Turn off PMAC, remove the jumper E51 and try communications again.

Bus Communications

3. Make sure that the bus address jumpers (E91-E92, E66-E71) set the same address as the bus address on the Executive program.
4. If there is something else on the bus at the same address, try changing the bus address to see if communications can be established at a new address. Usually, address 768 (300 hex) is open.

Serial Communications

5. Verify that the proper port on the PC is being used. Make sure that the Executive program is addressing the COM1 port, which is cabled out of the COM1 connector.
6. The baud rate specified in the Executive program should match the baud rate setting of the E44-E47 jumpers on PMAC.
7. With a breakout box or oscilloscope, make sure there is action on the transmit lines from the PC as while typing into the Executive program. If not, there is a problem on the PC end.
8. Probe the return communication line while giving PMAC a command that requires a response (e.g. **<CONTROL-F>**). If there is no action, change jumpers E9-E16 on PMAC to exchange the send and receive lines. If there is action, but the host program does not receive characters, RS-232 might be receiving circuitry that does not respond at all to PMAC's RS-422 levels. If there is another model of PC available, try using it as a test (most models accept RS-422 levels quite well). If the computer still will not accept the signals, try a level-conversion device, such as Acc-26.

Resetting PMAC for First Time Use

Once communications have been established, type the following commands in the terminal window:

```
$$$***                      ;Global Reset
P0..1023=0                  ;Reset P-variables values
Q0..1023=0                  ;Reset Q-variables values
M0..1023->* M0..1023=0      ;Reset M-variables definitions and values
UNDEFINE ALL                ;Undefine Coordinate Systems
SAVE                        ;Save this initial, "clean"configuration
```

Connections

Typically, the user connections are made to a terminal block that is attached to the JMACH connector by a flat cable (Acc-8D or 8P). The pinout numbers on the terminal block are the same as those on the JMACH connector for PMAC PC. While the numbering scheme for the pins on machine connectors on PMAC VME is different from that for PMAC PC, the physical arrangement is the same, and PMAC VME users can use the same terminal numbers on the terminal block board in following the instructions given below.

Note:

Make sure PMAC is not powered while the connections are being made. Leave any loads disconnected from the motor at this point.

Power Supplies

Digital Power Supply

```
1.5A @ +5V (+/-5%) (7.5W)
(Eight-channel configuration, with a typical load of encoders)
```

The host computer provides the 5V power supply if the PMAC is installed in its internal bus.

With the board plugged into the bus, it will pull +5V power from the bus automatically and it cannot be disconnected. In this case, there must be no external +5V supply, or the two supplies will fight each other, possibly causing damage. This voltage could be measured between pins 1 and 3 of the terminal block.

In a stand-alone configuration, when PMAC is not plugged in a computer bus, it will need an external 5V supply to power its digital circuits. The +5V line from the supply should be connected to pin 1 or 2 of the JMACH connector (usually through the terminal block), and the digital ground to pin 3 or 4.

Analog Power Supply

```
0.3A @ +12 to +15V (4.5W)
0.25A @ -12 to -15V (3.8W)
(Eight-channel configuration)
```

The analog output circuitry on PMAC is optically isolated from the digital computation circuitry, and so requires a separate power supply. This is brought in on the JMACH connector. The positive supply – +12 to +15V – should be brought in on the A+15V line on pin 59. The negative supply – -12 to -15V – should be brought in on the A-15V line on pin 60. The analog common should be brought in on AGND line on pin 58.

Typically, this supply can come from the servo amplifier; many commercial amplifiers provide such a supply. If this is not the case, an external supply may be used. Even with an external supply, the AGND line should be tied to the amplifier common. It is possible to get the power for the analog circuits from the bus, but doing so defeats optical isolation. In this case, no new connections need to be made. However, you should be sure jumpers E85, E87, E88, E89, and E90 are set up for this circumstance. (The card is not shipped from the factory in this configuration.)

Flags Power Supply (Optional)

Each channel of PMAC has four dedicated digital inputs on the machine connector: +LIMn, -LIMn (overtravel limits), HMFLn (home flag), and FAULTn (amplifier fault). In most PMACs, these inputs can be kept isolated from other circuits. A power supply from 12 to 24V can be used to power the corresponding opto-isolators related to these inputs. This feature is not available in PMAC PC without Option 1, PMAC VME or the PMAC STD board.

Overtravel Limits and Home Switches

When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive overtravel limits that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction. The direction sense of +LIMn and -LIMn is as follows: +LIMn should be placed at the negative end of travel, and -LIMn should be placed at the positive end of travel.

Disabling the Overtravel Limits Flags

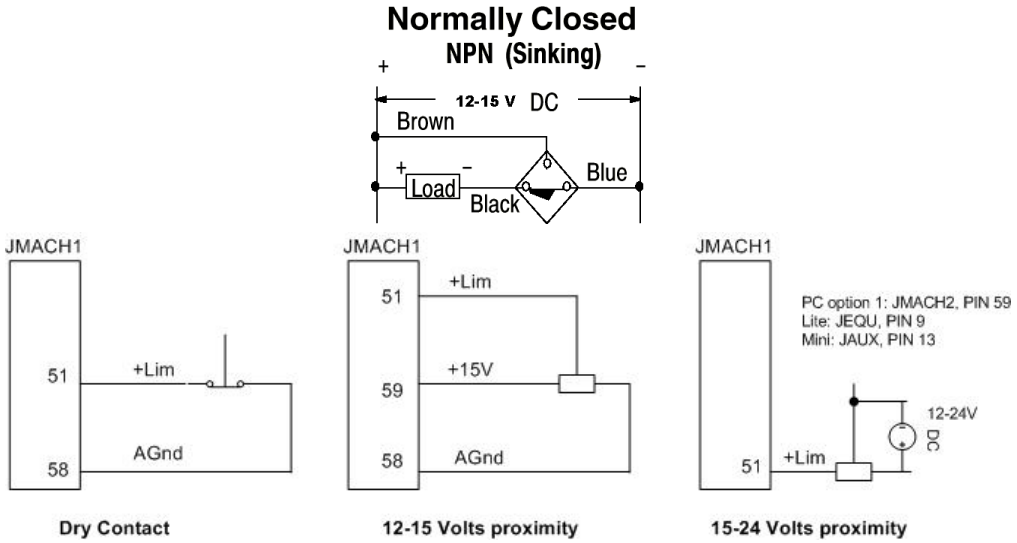
If no overtravel limits are used, they must be disabled through a change to variable Ix25. On the terminal window, the following commands will disable the limits functions for all eight motors. Select the motor numbers as appropriate.

The OR (|) bit-by-bit function used here is accessible by pressing shift + "\" in the computer's keyboard.

```
I125=I125|$20000              ;Motor #1
I225=I225|$20000              ;Motor #2
I325=I325|$20000              ;Motor #3
I425=I425|$20000              ;Motor #4
I525=I525|$20000              ;Motor #5
I625=I625|$20000              ;Motor #6
I725=I725|$20000              ;Motor #7
I825=I825|$20000              ;Motor #8
```

Types of Overtravel Limits

PMAC expects a closed-to-ground connection for the limits to not be considered on fault. This arrangement provides a failsafe condition and therefore it cannot be reconfigured differently in PMAC. Usually, a passive normally closed switch is used. If a proximity switch is needed instead, the following type is recommended:



Related PMAC jumpers must be configured appropriately, following the corresponding PMAC Hardware Reference manual.

Home Switches

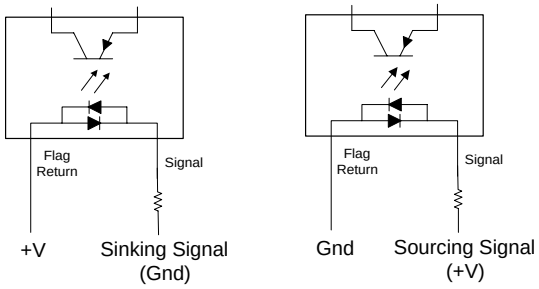
While normally closed-to-ground switches are required for the overtravel limits inputs, the home switches could be either normally closed or normally open types. The polarity is determined by the home sequence setup, through the I-Variables

I902, I907, ... I977. However, for the following reasons, the same type of switches used for overtravel limits are recommended:

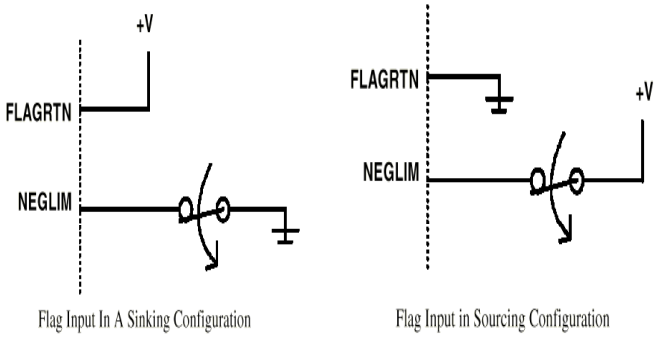
- Normally closed switches are proven to have greater electrical noise rejection than normally open types.
- Using the same type of switches for every input flag simplifies maintenance stock and replacements.

PMACPack and PMAC2 Flag Inputs

The PMAC Pack and PMAC2 interface accessories include a bipolar opto-isolating circuitry (chip PS-2705-4NEC) for flag and amplifier fault connections:



Examples:



Flag Input In A Sinking Configuration

Flag Input in Sourcing Configuration

Checking the Flag Inputs

In the PEWIN terminal window, define the following M-Variables for the flags of the motors under consideration:

Flag Type	Motor #1	Motor #2	Motor #3	Motor #4
HMFL input status	M120->X:$C000,20,1	M220->X:$C004,20,1	M320->X:$C008,20,1	M420->X:$C00C,20,1
-LIM input status	M121->X:$C000,21,1	M221->X:$C004,21,1	M321->X:$C008,21,1	M421->X:$C00C,21,1
+LIM input status	M122->X:$C000,22,1	M222->X:$C004,22,1	M322->X:$C008,22,1	M422->X:$C00C,22,1
Flag Type	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
HMFL input status	M520->X:$C010,20,1	M620->X:$C014,20,1	M720->X:$C018,20,1	M820->X:$C01C,20,1
-LIM input status	M521->X:$C010,21,1	M621->X:$C014,21,1	M721->X:$C018,21,1	M821->X:$C01C,21,1
+LIM input status	M522->X:$C010,22,1	M622->X:$C014,22,1	M722->X:$C018,22,1	M822->X:$C01C,22,1

Open a Watch Window and press Insert to enter the M-Variable number to watch. Interacting with the switch or sensor, monitor the change in the corresponding M-Variable. A value of zero indicates that the flag is closed to ground and therefore the limit is not in fault, the motor will be able to run in that direction (See Ix25). If the value is 1, the flag is open instead.

Motor Signals Connections

Incremental Encoder Connection

Each JMACH connector provides two +5V outputs and two logic grounds for powering encoders and other devices. The +5V outputs are on pins 1 and 2; the grounds are on pins 3 and 4. The encoder signal pins are grouped by number: all those numbered 1 (CHA1, CHA1/, CHB1, CHC1, etc.) belong to encoder #1. The encoder number does not have to match the motor number, but usually does. If the PMAC is not plugged into a bus and drawing its +5V and GND from the bus, use these pins to bring in +5V and GND from the power supply.

Connect the A and B (quadrature) encoder channels to the appropriate terminal block pins. For encoder 1, the CHA1 is pin 25, CHB1 is pin 21. If using a single-ended signal, leave the complementary signal pins floating -- do not ground them.

However, if single-ended encoders are used, check the settings of the jumpers E18 to E21 and E24 to E27.

For a differential encoder, connect the complementary signal lines -- CHA1/ is pin 27, and CHB1/ is pin 23. The third channel (index pulse) is optional; for encoder 1, CHC1 is pin 17, and CHC1/ is pin 19.

Checking the Encoder Inputs

Once the encoders have been properly wired, it is important to check its functionality and its polarity.

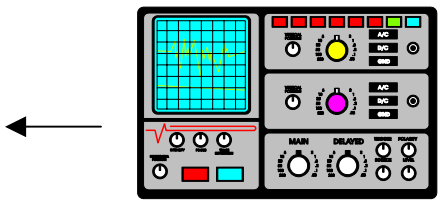
Note:

Make sure the motor is not powered while performing this test.

In the PEWIN, open a Position window by pressing **Alt+V** and **P** from the terminal window. Rotate the encoder to monitor the corresponding position value of the motor in the Position window. Make sure that a rotation in the positive direction increments the position values. Also, make sure that the number of counts per revolution of the encoder matches the number read by PMAC when a complete revolution of the motor has been rotated. If necessary, for troubleshooting purposes, place an oscilloscope in the encoder inputs to check the appropriate signals provided by the encoder:

Example for Encoder #1:

- Channel A in pin 25 of JMACH1 (Acc-8D or Acc-8P)
- Channel B in pin 21 of JMACH1 (Acc-8D or Acc-8P)
- Ground in pin 3 or 4 of JMACH1 (Acc-8D or Acc-8P)



Checking the DAC Outputs

Before connecting the DAC outputs to the amplifier, it is opportune to check the DAC outputs operation.

Note:

Make sure the amplifier is not connected while performing this test.

In the PEWIN terminal window, define the following M-Variables for the DACs of the motors under consideration:

	Motor #1	**Motor #2**	**Motor #3**	**Motor #4**
DAC output	M102->Y:$C003,8,16,S	M202->Y:$C002,8,16,S	M302->Y:$C00B,8,16,S	M402->Y:$C00A,8,16,S
	Motor #5	**Motor #6**	**Motor #7**	**Motor #8**
DAC output	M502->Y:$C013,8,16,S	M602->Y:$C012,8,16,S	M702->Y:$C01B,8,16,S	M802->Y:$C01A,8,16,S

Example for DAC #1:
Type the following in the terminal window:

```
M102->Y:$C003,8,16,S
I100=0
M102=16383
```
<measure 5V between pins 43 and 58 of JMACH1, (Acc-8D or Acc-8P)>
```
M102=-16383
```
<measure -5V between pins 43 and 58 of JMACH1, (Acc-8D or Acc-8P)>
```
I100=1
```

DAC Output Signals

If PMAC is not performing the commutation for the motor, only one analog output channel is required to command the motor. This output channel can be either single-ended or differential, depending on what the amplifier is expecting.

For a single-ended command using PMAC channel 1, connect DAC1 (pin 43) to the command input on the amplifier. Connect the amplifier's command signal return line to PMAC's AGND line (pin 58). In this setup, leave the DAC1/ pin floating; do not ground it.

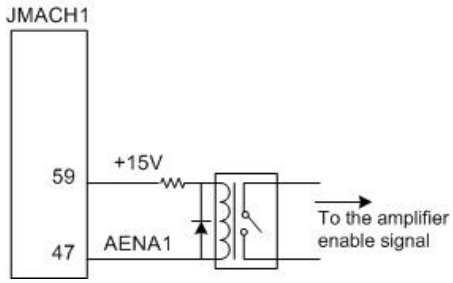
For a differential command using PMAC channel 1, connect DAC1 (pin 43) to the Plus Command input on the amplifier. Connect DAC1/ (pin 45) to the minus-command input on the amplifier. PMAC's AGND should be still connected to the amplifier common.

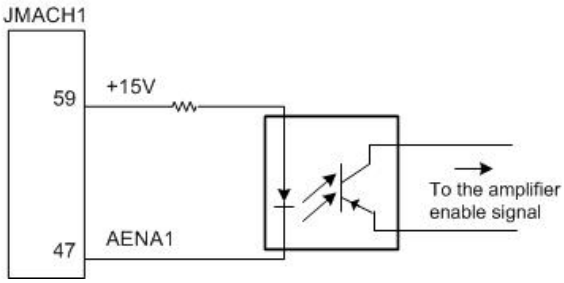
If the amplifier is expecting separate sign and magnitude signals, connect DAC1 (pin 43) to the magnitude input. Connect AENA1/DIR1 (pin 47) to the sign (direction input). Amplifier signal returns should be connected to AGND (pin 58). This format requires some parameter changes on PMAC; (See Ix02 and Ix25.). Jumper E17 controls the polarity of the direction output; this may have to be changed during the polarity test. This magnitude-and-direction mode is suited for driving servo amplifiers that expect this type of input, and for driving voltage-to-frequency (V/F) converters, such as PMAC's Acc-8D Option 2 board, for running stepper motor drivers.

If using PMAC to commutate the motor, use two analog output channels for the motor. Each output may be single-ended or differential, just as for the DC motor. The two channels must be numbered consecutively, with the lower-numbered channel having an odd number (e.g. use DAC1 and DAC2 for a motor, or DAC3 and DAC4, but not DAC2 and DAC3, or DAC2 and DAC4). For motor #1 example, connect DAC1 (pin 43) and DAC2 (pin 45) to the analog inputs of the amplifier. If using the complements as well, connect DAC1/ (pin 45) and DAC2/ (pin 46) the minus-command inputs; otherwise leave the complementary signal outputs floating. To limit the range of each signal to +/- 5V, use parameter I169.

Amplifier Enable Signal (AENAx/DIRn)

Most amplifiers have an enable/disable input that permits complete shutdown of the amplifier regardless of the voltage of the command signal. PMAC's AENA line is meant for this purpose. If not using a direction and magnitude amplifier or voltage-to-frequency converter, use this pin to enable and disable the amplifier (wired to the enable line). AENA1/DIR1 is pin 47. This signal is an open-collector output and requires a pull up resistor to A+15V. For early tests, this amplifier signal should be under manual control. Jumper E17 controls the polarity of the signal. The default is low-true (conducting) enable. For any other kind of amplifier enable signal, a dry contact of a relay or a solid-state relay can be used:





In addition, the amplifier enable signal can be controlled manually by setting Ix00=0 and using the properly defined Mx14 variable.

Amplifier Fault Signal (FAULTn)

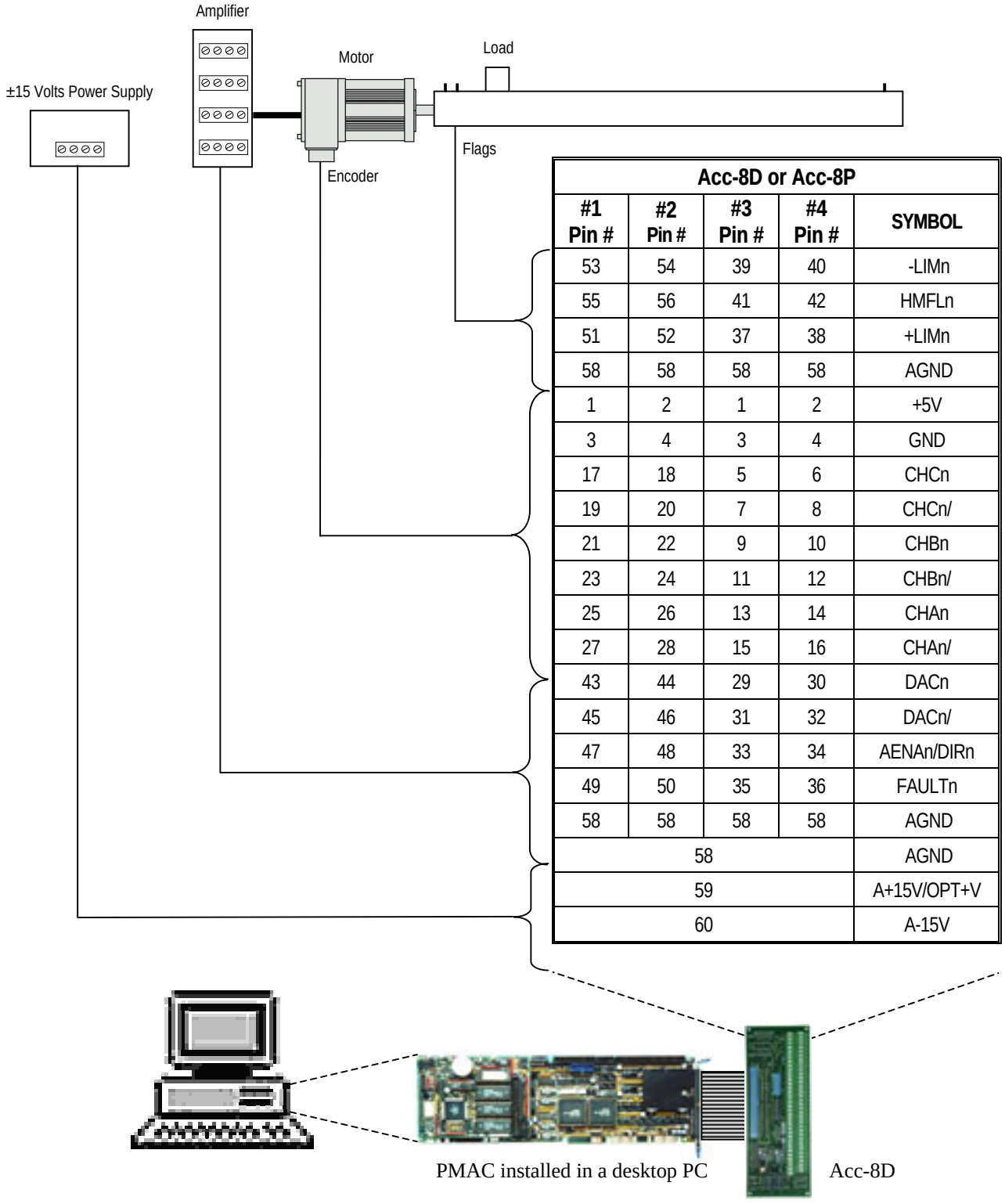
This input can take a signal from the amplifier so PMAC knows when the amplifier is having problems, and can shut down action. The polarity is programmable with I-Variable Ix25 (I125 for motor #1) and the return signal is analog ground (AGND). FAULT1 is pin 49. With the default setup, this signal must be actively pulled low for a fault condition. In this setup, if nothing is wired into this input, PMAC will consider the motor not to be in a fault condition. The amplifier fault signal can be monitored using the properly defined Mx23 variable.

General-Purpose Digital Inputs and Outputs (JOPTO Port)

PMAC's JOPTO connector (J5 on PMAC PC, Lite, and VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Acc-21F is a six-foot cable for this purpose. Typically, these inputs and outputs are accessed in software through the use of M-Variables. In the suggested set of M-Variable definitions, variables M1 through M8 are used to access outputs 1 through 8, respectively, and M11 through M18 to access inputs 1 through 8, respectively. This port maps into PMAC's memory space at Y address $FFC2.

- The Acc-21S is an I/O simulator for the PMAC JOPTO port; it provides eight switch inputs and eight LED outputs.
- The Acc-21S is a good tool for I/O simulation and troubleshooting of the JOPTO port in PMAC.

Machine Connections Example



Acc-8D or Acc-8P				
#1 Pin #	#2 Pin #	#3 Pin #	#4 Pin #	SYMBOL
53	54	39	40	-LIMn
55	56	41	42	HMFLn
51	52	37	38	+LIMn
58	58	58	58	AGND
1	2	1	2	+5V
3	4	3	4	GND
17	18	5	6	CHCn
19	20	7	8	CHCn/
21	22	9	10	CHBn
23	24	11	12	CHBn/
25	26	13	14	CHAn
27	28	15	16	CHAn/
43	44	29	30	DACn
45	46	31	32	DACn/
47	48	33	34	AENAn/DIRn
49	50	35	36	FAULTn
58	58	58	58	AGND
58				AGND
59				A+15V/OPT+V
60				A-15V

This diagram is just an example of one of the many variations of the machine connections. PMAC jumpers must be set appropriately following both the appropriate PMAC Hardware Reference and the PMAC User manuals.

Software Setup

PMAC has a large set of initialization parameters (I-Variables) that determine the personality of the card for a specific application. Many of these are used to configure a motor properly. Using PEWIN, follow these steps for Software Setup:

1. Fully reset PMAC to ensure a clean memory configuration before start:

```
$$$***                          ;Global Reset
P0..1023=0  Q0..1023=0          ;Reset P-variables and Q-variables values
M0..1023->* M0..1023=0          ;Reset M-variables definitions and values
UNDEFINE ALL                    ;Undefine Coordinate Systems
SAVE                            ;Save this initial, clean configuration
```

2. Define the safety I-Variables appropriately (x stands for the motor number, 1 through 8):

	Motor Safety I-Variables	**Range**	**Default**	**Units**
Ix00	Motor x Activate	0 .. 1	0 (1 for Motor 1)	none
Ix11	Motor x Fatal Following Error Limit	0 .. 8,388,607	32000	1/16 Count
Ix12	Motor x Warning Following Error Limit	0 .. 8,388,607	16000	1/16 Count
Ix13	Motor x + Software Position Limit	+/- 2^{47}	0 (Disabled)	Encoder Counts
Ix14	Motor x - Software Position Limit	+/- 2^{47}	0 (Disabled)	Encoder Counts
Ix15	Motor x Abort/Lim Decel Rate	Positive floating point	0.25	Counts/msec^2
Ix16	Motor x Maximum Velocity	Positive floating point	32	Counts/msec
Ix17	Motor x Maximum Acceleration	Positive floating point	0.015625	Counts/msec^2
Ix19	Motor x Maximum Jog Acceleration	Positive floating point	0.015625	Counts/msec^2
Ix25	Motor x Flag Address	PMAC X addresses	see Ix25 table	Extended legal PMAC X addresses

For dual feedback systems:

$$Ix08 \cdot \frac{Number\ of\ counts\ of\ the\ position\ encoder}{Units\ of\ Distance\ of\ the\ position\ encoder} = Ix09 \cdot \frac{Number\ of\ counts\ of\ the\ velocity\ encoder}{Units\ of\ Distance\ of\ the\ velocity\ encoder}$$

3. Leave any loads disconnected from the motor at this point.

   Test the polarity and functioning of the motor by means of open loop commands. For the open loop command to work the overtravel limits must be either disabled (See Ix25) or properly connected. Type the following in the terminal:

```
#1O10     ; “Pound one, ‘O’ ten” will output 10% of the DAC on motor #1. It
          ; is about 0.6V on default settings
          ; <Observe the motor turning in the positive direction; the position
          ; window should indicate motor #1 counting up>
 #1O-10   ; “Pound one, ‘O’ negative ten” will output a negative 10% of the
          ; DAC on motor #1, about -0.6V
          <Observe the motor turning in the negative direction; the position
          ; window should indicate motor #1 decreasing>
```

   If no motion is observed, slowly increase the percentage of the output command issued. If after 50% no reaction of the motor occurred, check the DAC outputs following the guidelines in the previous sections.

4. Perform a tuning procedure as described in the PEWIN chapter.

5. After the tuning process has been completed satisfactory, check it by means of the following online commands:

```
SAVE            ;Save this setup
#1J+            ;Jog Motor #1 continuously in the positive direction
#1J-            ;Jog Motor #1 continuously in the negative direction
#1J=2000        ;Jog Motor #1 to a known location
```

6. Create a PMAC memory backup file as described in the PEWIN chapter.

PROGRAMMING PMAC

Programming PMAC is very simple; the ease of use and power is based in the following features:

- A clever interrupt-driven scheme allows every task, each motion program and PLC, to run independently of each other.
- Pointer M-Variables allow monitoring virtually any register in PMAC's memory from different sources: motion programs, PLCs or the host computer.
- Communications are activated continuously. At any moment, any variable or status command could be interrogated.
- Up to eight Axes can be either synchronized together, controlled individually or in any combination in between.
- Data gathering and reporting functions allows saving data such as motion trajectories, velocity profiles or any set of variables for later analysis and plot.

PMAC is fundamentally a command-driven device. PMAC performs by issuing it ASCII command text strings and generally, PMAC provides information to the host in ASCII text strings.

When PMAC receives an alphanumeric text character over one of its ports, it does nothing but place the character in its command queue. It requires a control character (ASCII value 1 to 31) to cause it to take action. The most common control character used is the carriage return (**<CR>**; ASCII value 13), which tells PMAC to interpret the preceding set of alphanumeric characters as a command and to take the appropriate action.

Online Commands

Many of the commands given to PMAC are on-line commands; that is, they are executed immediately by PMAC to cause some action, change some variable, or report some information back to the host.

Some commands, such as **P1=1**, are executed immediately if there is no open program buffer, but are stored in the buffer if one is open. Other commands, such as **X1000 Y1000**, cannot be on-line commands; there must be an open buffer – even if it is a special buffer for immediate execution. These commands will be rejected by PMAC (reporting an ERR005 if I6 is set to 1 or 3) if there is no buffer open. Still other commands, such as **J+**, are on-line commands only and cannot be entered into a program buffer (unless in the form of **CMD"J+"**, for instance).

There are three basic classes of on-line commands:

1. Motor-specific commands, which affect only the motor that is currently addressed by the host
2. Coordinate-system-specific commands, which affect only the coordinate system that is currently addressed by the host
3. Global commands, which affect the card regardless of any addressing modes.

A motor is addressed by a **#n** command, where **n** is the number of the motor, with a range of 1 to 8, inclusive. This motor is the one addressed until another **#n** is received by the card. For instance, the command line **#1J+#2J-** tells Motor 1 to jog in the positive direction, and Motor 2 to jog in the negative direction. There are only a few types of motor-specific commands. These include the jogging commands, a homing command, an open loop command, and requests for motor position, velocity, following error, and status.

A coordinate system is addressed by a **&n** command, where **n** is the number of the coordinate system, with a range of 1 to 8, inclusive. This coordinate system stays the one addressed until another **&n** command is received by the card. For instance, the command line **&1B6R&2B8R** tells Coordinate System 1 to run Motion Program 6 and Coordinate System 2 to run Motion Program 8. There are a variety of types of coordinate-system-specific commands. Axis definition statements act on the addressed coordinate system, because motors are matched to an axis in a particular coordinate system. Since it is a coordinate system that runs a motion control program, all program control commands act on the addressed coordinate system. Q-Variable assignment and query commands are also coordinate system commands, because the Q-Variables themselves belong to a coordinate system.

Some on-line commands do not depend on which motor or coordinate system is addressed. For instance, the command **P1=1** sets the value of P1 to 1 regardless of what is addressed. Among these global on-line commands are the buffer management commands. PMAC has multiple buffers, one of which can be open at a time. When a buffer is open, commands can be entered into the buffer for later execution.

Control character commands (those with ASCII values 0 - 31D) are always global commands. Those that do not require a data response act on all cards on a serial daisychain. These characters include carriage return **<CR>**, backspace **<BS>**, and several special-purpose characters. This allows, for instance, commands to be given to several locations on the card in a single line, and have them take effect simultaneously at the **<CR>** at the end of the line (**&1R&2R<CR>** causes both Coordinate Systems 1 and 2 to run).

Buffered (Program) Commands

As their name implies, buffered commands are not acted on immediately, but held for later execution. PMAC has many program buffers – 256 regular motion program buffers, eight rotary motion program buffers (1 for each coordinate system), and 32 PLC program buffers. Before commands can be entered into a buffer, that buffer must be opened (e.g. **OPEN PROG 3**, **OPEN PLC 7**). Each program command is added onto the end of the list of commands in the open buffer; to replace the existing buffer, use the **CLEAR** command immediately after opening to erase the existing contents before entering the new ones. After finishing entering the program statements, use the **CLOSE** command to close the opened buffer.

Computational Features

I-Variables

I-Variables (initialization, or setup variables) determines the personality of the card for a given application. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 1024 I-Variables, from I0 to I1023, and they are organized as follows:

```
I0 -- I79:        General card setup
I80 -- I99:       Geared Resolver setup
I100 -- I184:     Motor #1 setup
I185 -- I199:     Coordinate System 1 setup
I200 -- I284:     Motor #2 setup
I285 -- I299:     Coordinate System 2 setup
I800 -- I884:     Motor #8 setup
I885 -- I899:     Coordinate System 8 setup
I900 -- I979:     Encoder 1 - 16 setup
I980 -- I1023:    Reserved for future use
```

Values assigned to an I-Variable may be either a constant or an expression. The commands to do this are on-line (immediate) if no buffer is open when sent, or buffered program commands if a buffer is open.

Examples:

```
I120 = 45
I120 = (I120+P25*3)
```

For I-Variables with limited range, an attempt to assign an out-of-range value does not cause an error. The value is rolled over automatically to within the range by modulo arithmetic (truncation). For example, I3 has a range of 0 to 3 (4 possible values). The command **I3=5** would actually assign a value of 5 modulo 4 = 1 to the variable.

On PMACs with battery-backed RAM, most of the I-Variable values can be stored in a 2K x 8 EEPROM IC with the **SAVE** command. These values are safe here even in the event of a battery-backed RAM failure, so the basic setup of the board is not lost. After a new value is given to one of these I-Variables, the **SAVE** command must be issued in order for this value to survive a power-down or reset.

The I-Variables that are not saved to EEPROM are held in battery-backed RAM. These variables do not require a **SAVE** command to be held through a power-down or reset, and the previous value is not retained anywhere. These variables are: I19-I44, Ix13, Ix14.

On PMACs with flash memory backup (those with Option 4A, 5A, or 5B), all of the I-Variable values can be stored in the flash memory with the **SAVE** command. If there is an EEPROM IC on the board, it is not used. After a new value is given to any I-Variable, the **SAVE** command must be issued in order for this value to survive a power-down or reset.

Default values for all I-Variables are contained in the manufacturer-supplied firmware. They can be used individually with the **I{constant}=*** command, or in a range with the **I{constant}..{constant}=*** command. Upon board re-initialization by the **$$$***** command or by a reset with E51 in the non-default setting, all default settings are copied from the firmware into active memory. The last saved values are not lost; they are just not used.

P-Variables

P-Variables are general-purpose user variables. They are 48-bit floating-point variables at fixed locations in PMAC's memory, but with no pre-defined use. There are 1024 P-Variables, from P0 to P1023. A given P-Variable means the same thing from any context within the card; all coordinate systems have access to all P-Variables (contrast Q-Variables, which are coupled to a given coordinate system). This allows for useful information passing between different coordinate systems. P-Variables can be used in programs for any purpose desired: positions, distances, velocities, times, modes, angles, intermediate calculations, etc.

If a command consisting simply of a constant value is sent to PMAC, PMAC assigns that value to variable P0. For example, if the command **342<CR>** is sent to PMAC, it will interpret it as **P0=342<CR>**. This capability is intended to facilitate simple operator terminal interfaces. It does mean, however, that it is not a good idea to use P0 for other purposes, because it is easy to change this accidentally.

Q-Variables

Q-Variables, like P-Variables, are general-purpose user variables: 48-bit floating-point variables at fixed locations in memory, with no pre-defined use. However, the meaning of a given Q-Variable (and hence the value contained in it) is dependent on which coordinate system is utilizing it. This allows several coordinate systems to use the same program (for instance, containing the line X(Q1+25) Y(Q2), but to do have different values in their own Q-Variables (which in this case, means different destination points).

Several Q-variables have special uses. The **ATAN2** (two-argument arctangent) function uses Q0 automatically as its second argument (the cosine argument). The **READ** command places the values it reads following letters A through Z in Q101 to Q126, respectively, and a mask word denoting which variables have been read in Q100. The S (spindle) statement in a motion program places the value following it into Q127.

Based on that and since a total of 1024 Q-Variables are shared between potentially eight Coordinate Systems (128 variables each), the practical range of the Q-Variables to be used safely in motion programs is therefore Q1 to Q99.

The set of Q-Variables working within a command depends on the type of command. When accessing a Q-Variable from an on-line (immediate) command from the host, it is the Q-variable for the currently host-addressed coordinate system (with the **&n** command). When accessing a Q-Variable from a motion program statement, it is the Q-Variable belonging to the coordinate system running the program. If a different coordinate system runs the same motion program, it will use different Q-variables.

When accessing a Q-Variable from a PLC program statement, it is the Q-Variable for the coordinate system that has been addressed by that PLC program with the **ADDRESS** command. Each PLC program can address a particular coordinate system independent of other PLC programs and independent of the host addressing. If no **ADDRESS** command is used in the PLC program, the program uses the Q-Variables for Coordinate System 1.

M-Variables

To permit easy access to PMAC's memory and I/O space, M-Variables are provided. Generally, a definition must be made only once with an on-line command. On PMACs with battery backup, the definition is held automatically. On PMACs with flash backup, the **SAVE** command must be used to retain the definition through a power-down or reset. The user defines an M-variable by assigning it to a location and defining the size and format of the value in this location. An M-variable can be a bit, a nibble (4 bits), a byte (8 bits), 1-1/2 bytes (12 bits), a double-byte (16 bits), 2-1/2 bytes (20 bits), a 24-bit word, a 48-bit fixed-point double word, a 48-bit floating-point double word, or special formats for dual-ported RAM and for the thumbwheel multiplexer port.

There are 1,024 M-Variables (M0 to M1023), and as with other variable types, the number of the M-variable may be specified with either a constant or an expression: M576 or M(P1+20) when read from; the number must be specified by a constant when written to.

The definition of an M-Variable is done using the defines arrow (**->**) composed of the minus sign and greater than symbols. An M-Variable may take one of the following types, as specified by the address prefix in the definition:

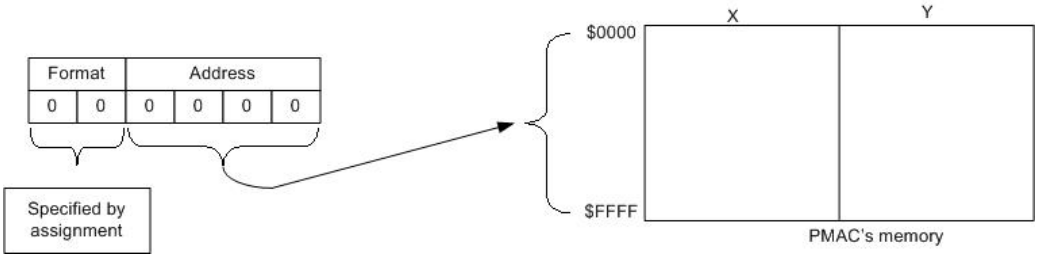
```
X:    1 to 24 bits fixed-point in X-memory
Y:    1 to 24 bits fixed-point in Y-memory
D:    48 bits fixed-point across both X- and Y-memory
L:    48 bits floating-point across both X- and Y-memory
DP:   32 bits fixed-point (low 16 bits of X and Y) (for use in dual-ported RAM)
F:    32 bits floating-point (low 16 bits of X and Y) (for use in dual-ported RAM)
TWD:  Multiplexed BCD decoding from Thumbwheel port
TWB:  Multiplexed binary decoding from Thumbwheel port
TWS:  Multiplexed serial I/O decoding from Thumbwheel port
TWR:  Multiplexed serial resolver decoding from Thumbwheel port
*:    No address definition; uses part of the definition word as general-
      purpose variable
```

If an X or Y type of M-Variable is defined, the starting bit to use, the number of bits, and the format (decoding method) must be defined also.

Typical M-Variable definition statements are:

```
M1->Y:$FFC2,8,1
M102->Y:49155,8,16,S
M103->X:$C003,0,24,S
M161->D:$002B
M191->L:$0822
M50->DP:$D201
M51->F:$D7FF
M100->TWD:4,0.8.3,U
```

The M-Variable definitions are stored as 24-bit codes at PMAC addresses Y:$BC00 (for M0) to Y:$BFFF (for M1023). For all but the thumbwheel multiplexer port M-Variables, the low 16 bits of this code contains the address of the register pointed to by the M-Variable (the high 8-bits tell what part of the address is used and how it is interpreted).



If another M-Variable points to this part of the definition, it can be used to change the subject register. The main use of this technique is to create arrays of P- and Q-Variables or arrays in dual-ported RAM or in user buffers (see on-line command **`DEFINE UBUFFER`**).

Many M-Variables have a more limited range than PMAC's full computational range. If a value outside of the range of an M-Variable is placed to that M-Variable, PMAC rolls over the value automatically to within that range and does not report any errors. For example, with a single bit M-Variable, any odd number written to the variable ends up as 1, any even number ends up as 0. If a non-integer value is placed in an integer M-Variable, PMAC rounds to the nearest integer automatically.

Once defined, an M-Variable may be used in programs just as any other variable – through expressions. When the expression is evaluated, PMAC reads the defined memory location, calculates a value based on the defined size and format, and utilizes it in the expression.

Care should be exercised in using M-Variables in expressions. If an M-Variable is something that can be changed by a servo routine (such as instantaneous commanded position), which operates at a higher priority the background expression evaluation, there is no guarantee that the value will not change in the middle of the evaluation. For instance, if in the expression (M16- M17)*(M16+M17) the M-Variables are instantaneous servo variables, the user cannot be sure that M16 or M17 will have the same value both places in the expression, or that the values for M16 and M17 will come from the same servo cycle. The first problem can be overcome by setting P1=M16 and P2=M17 right above this, but there is no general solution to the second problem.

Array Capabilities

It is possible to use a set of P-Variables as an array. To read or assign values from the array, simply replace the constant specifying the variable number with an expression in parentheses.

Example:

```
P1=10                          ; Array index variable
P3=P(P1)                       ; Same as P3=P10
```

To write to the array, M-Variables must be used. An M-Variable defined to the corresponding P-Variable address will allow changing any P-Variable and therefore the contents of the array.

Example: Values 31 to 40 will be assigned to variables P1 through P10

```
M34->L:$1001                   ; Address location of P1
M35->Y:$BC22,0,16              ; Definition word of M34
OPEN PLC 15 CLEAR
P100=31
WHILE (P100!>40)               ; From 31 to 40
    M34=P100                   ; Value is written to the array
    P100=P100+1                ; Next value
    M35=M35+1                  ; Next Array position (next P-variable)
ENDWHILE
DISABLEPLC15                   ; This PLC runs only once
```

```
CLOSE
ena PLC15                       ; Enable the PLC (I5 must be 2 or 3)
P1..10                          ; List the values of P1 to P10
```

The same concept applies for Q-Variables and M-Variables arrays, although the address range for them is different.

Operators

PMAC operators work like those in any computer language: they combine values to produce new values.

PMAC uses the four standard arithmetic operators: **+**, **-**, *****, and **/**. The standard algebraic precedence rules are used: multiply and divide are executed before add and subtract, operations of equal precedence are executed left to right, and operations inside parentheses are executed first.

PMAC also has the **%** modulo operator, which produces the resulting remainder when the value in front of the operator is divided by the value after the operator. Values may be integer or floating point. This operator is useful particularly for dealing with counters and timers that roll over.

When the modulo operation is done by a positive value X, the results can range from 0 to X (not including X itself). When the modulo operation is done by a negative value -X, the results can range from -X to X (not including X itself). This negative modulo operation is useful when a register can roll over in either direction.

PMAC has three logical operators that do bit-by-bit operations: **&** (bit-by-bit **AND**), **|** (bit-by-bit **OR**), and **^** (bit-by- bit **EXCLUSIVE OR**). If floating-point numbers are used, the operation works on the fractional as well as the integer bits. **&** has the same precedence as ***** and **/**; **|** and **^** have the same precedence as **+** and **-**. Use of parentheses can override the default precedence.

Functions

These perform mathematical operations on constants or expressions to yield new values. The general format is:

{function name} ({expression})

The available functions are **SIN**, **COS**, **TAN**, **ASIN**, **ACOS**, **ATAN**, **ATAN2**, **SQRT**, **LN**, **EXP**, **ABS**, and **INT**.

The global I-Variable I15 controls whether the units for the trigonometric functions are degrees or radians.

SIN	This is the standard trigonometric sine function.
COS	This is the standard trigonometric cosine function.
TAN	This is the standard trigonometric tangent function.
ASIN	This is the inverse sine (arc-sine) function with its range reduced to +/-90 degrees.
ACOS	This is the inverse cosine (arc-cosine) function with its range reduced to 0 -- 180 degrees.
ATAN	This is the standard inverse tangent (arc-tangent) function.
ATAN2	This is an expanded arctangent function, which returns the angle whose sine is the expression in parentheses and whose cosine is the value of Q0 for that coordinate system. If doing the calculation in a PLC program, make sure that the proper coordinate system has been addressed in that PLC program. (Actually, it is only the ratio of the magnitudes of the two values, and their signs, that matter in this function). It is distinguished from the standard ATAN function by the use of two arguments. The advantage of this function is that it has a full 360-degree range, rather than the 180-degree range of the single-argument ATAN function.
LN	This is the natural logarithm function (log base e).
EXP	This is the exponentiation function (e^x). **Note:** To implement the y^x function, use $e^{x \ln(y)}$ instead. A sample PMAC expression would be EXP(P2*LN(P1)) to implement the function $P1^{P2}$.
SQRT	This is the square root function.
ABS	This is the absolute value function.
INT	This is a truncation function, which returns the greatest integer less than or equal to the argument (INT(2.5)=2, INT(-2.5)=-3).

Functions and operators can be used either in Motion Programs, PLCs, or as online commands. For example, the following commands can be typed in a terminal window:

```
P1=SIN (45) P1          ; Reports the sine value of a 45° angle
I130=I130/2             ; Lower the proportional gain of Motor #1 by half
I125=I125|$20000        ; Disable the end-of-travel limits of Motor #1
```

Comparators

A comparator evaluates the relationship between two values (constants or expressions). It is used to determine the truth of a condition in a motion or PLC program. The valid comparators for PMAC are:

```
=    (equal to)
!=   (not equal to)
>    (greater than)
!>   (not greater than; less than or equal to)
<    (less than)
!<   (not less than; greater than or equal to)
~    (approximately equal to -- within one)
!~   (not approximately equal to -- at least one apart)
```

Note that **<=** and **>=** are not valid PMAC comparators. The comparators **!>** and **!<**, respectively, should be used in their place.

User-Written Phase and User-Written Servo Algorithms

For the sophisticated user with unusual and/or difficult commutation needs, PMAC provides the hooks for custom user-written commutation (phasing) or servo algorithms. These routines must be written in Motorola 56000 assembly language code, usually on a PC or compatible and cross assembled for the 56000.

Memory Map

PMAC's processor is the Motorola 56001 DSP. The 56001 has dual data buses, each 24-bits wide, so that both operands in a calculation may be brought in simultaneously. Each bus has access to a 16-bit address space (0000hex to FFFFhex), which provides 65,536 24-bit words. One bus and address space is called X, and the other is called Y. Therefore, when specifying a single-word memory location, one must use X: or Y: with the 16-bit address. PMAC's input and output is mapped into the same address space with the memory.

PMAC uses double-word memory for both extended fixed-point values and for floating-point values (single words are always fixed point). The fixed-point double word locations are specified by a D: (double), and the floating-point double word locations are specified by an L: (long). This matches the syntax of M-Variable declarations for these registers.

PMAC addresses may be specified with either decimal or hexadecimal values; the hex values must be preceded by a `$` to be interpreted as hex. For example, `Y:$FFC0` is the hexadecimal specification, and `Y:65472` is the decimal specification of the same word address.

M-Variables are defined by providing the word address, the offset, the width, and the format (irrelevant for bits). Several M-Variables were defined at the factory to match to inputs and outputs. For instance, M11 thru M18 were assigned to Machine Inputs 1 thru 8 (MI1-MI8), and M1 to M8 were assigned to Machine Outputs 1 thru 8 (MO1-MO8).

The PMAC architecture is very open, allowing the user to examine and use many internal registers. Usually this is done through the use of M-Variables, which point to locations in the memory-I/O space of the PMAC processor. Once defined to point to the proper location, an M-Variable can be treated as any other variable for reading and writing.

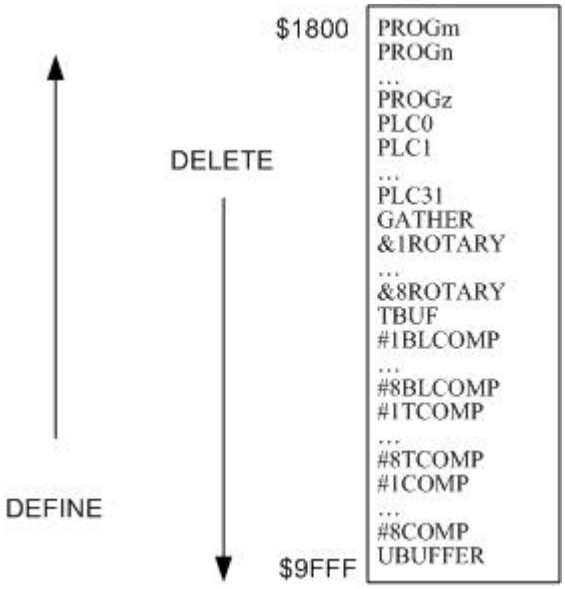
Warning:

Certain registers that are under PMAC's automatic control, particularly those used in the servo calculations, can cause problems if written to them directly.

Range	X-Memory	Y-Memory	Type
$0000 - $00FF	Fixed-Use calculation Registers	Fixed-Use calculation Registers	Internal DSP Memory
$0100 - $17FF	Fixed-Use calculation Registers	Fixed-Use calculation Registers	External Static RAM (Battery Backed)
$1800 - $BBFF	User Buffer Storage Space	User Buffer Storage Space	External Static RAM (Battery Backed)
$BC00 - $BFFF	User-Written Servo Storage	M-Variable Definitions	External Static RAM (Battery Backed)
$C000 - $C03F			DSP-Gate Registers
$D000 - $DFFF	Bits 0 to 15	Bits 0 to 15	Dual-Ported RAM
$E000 - $F000	VME Setup Registers (bits 0 to 7)	Mailbox Registers (bits 0 to 7)	VME bus registers
$F000 - $FFFF	N / A		I / O Registers

User Buffer Storage Space

- 256 Motion Programs can be held. All programs must be stopped before any can be opened.
- All programs must be stopped before any can run.
- A PLC program can be opened while others are running.
- Buffers must be defined from end of memory toward beginning. Buffers must be deleted from beginning of memory to end.

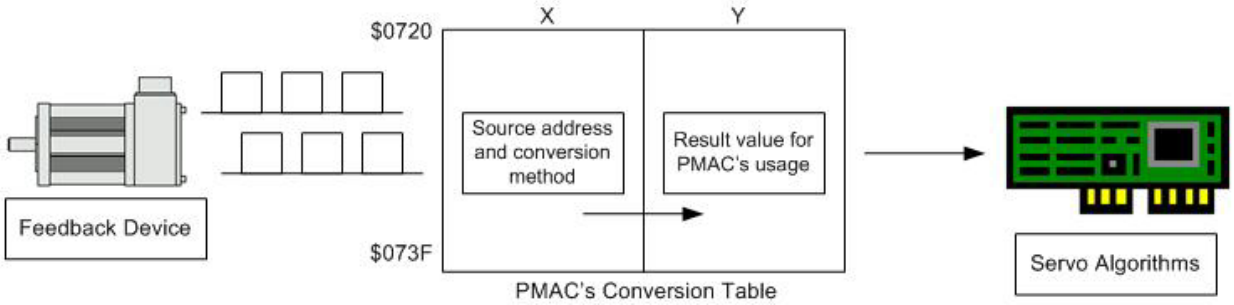


Encoder Conversion Table

PMAC uses a multiple-step process to work with its feedback and master position information, and with external time-base sources, to provide maximum power and flexibility. For most PMAC users with quadrature encoders, this process can be virtually transparent, with no need to worry about the details. However, some users will need to understand this conversion process in some detail to make the changes necessary to use other types of feedback, to optimize their system, or to perform special functions. The PMAC Executive Program for PC-compatible computers has a special editing screen for the conversion table that makes viewing it and changing it very easy.

Conversion Table Structure

The Encoder Conversion Table has two columns, one in the X memory space of the processor, and one in the Y memory space. The X-column holds the converted data, while the Y-column holds the addresses of the source registers, and the conversion methods used on the data in each of those source registers. Basically, the table is set up by writing to the Y-column, and PMAC uses the Y-column data to fill up the X-column each servo cycle.



The encoder conversion table starts at address $720 (1824 decimal) in PMAC's memory. It can continue through address $73F (1855 decimal). The active part of the table is ended by the first Y word that is all zeros. The encoder table as shipped from the factory converts the eight incremental encoder registers on the base PMAC board in locations $720 through $727 (1824 to 1831). Locations $728 and $729 create time base information from the converted Encoder 4 register ($723). Y:$72A is zero, ending the active part of the table.

Some conversion types need more than one entry; the other Y-words are further setup parameters for the conversion. The conversion result is placed in the last (highest address) X-word, and the other X-words hold intermediate data.

Example:

$728 (1832)	$400723	Time-base from converted Enc. 4
$729 (1833)	$000295	Time-base scale factor for above

The result of this time base value based on encoder #4 is placed in register X:$0729, the second and last entry for this conversion.

Further Position Processing

Once the position feedback signals have been processed by the Encoder Conversion Table (which happens at the beginning of each servo cycle), the data is ready for use by the servo loop. For each activated motor, PMAC takes the position information in the 24-bit register pointed to by Ix03 and extends it in software to a 48-bit register that holds the actual motor position. Several other features are available for conditioning the feedback signal as needed:

- Axis Position Scaling: in the coordinate system axis definition a scale factor determines the relationship between encoder counts and user units to be used in motion programs.
- Leadscrew Compensation: a compensation table containing corrective values for errors due to the leadscrew imperfections can be created for each motor.
- Backlash Compensation: On reversal of the direction of the commanded velocity, a pre-programmed backlash distance is added to or subtracted from the commanded position.
- Torque Compensation Tables: The table belonging to a motor provides a torque correction to that motor as a function of that motor's position.

PMAC Position Registers

The PMAC Executive position window or the online command **P** reports the value of the actual position register plus the position bias register plus the compensation correction register and if bit 16 of Ix05 is 1 (handwheel offset mode) minus the master position register:

```
M175->X:$002A,16,1      ; Bit 16 of I105
M162->D:$002B           ; #1 Actual position (1/[Ix08*32] cts)
M164->D:$0813           ; #1 Position bias (1/[Ix08*32] cts)
M167->D:$002D           ; #1 Present master ((handwheel) pos (1/[Ix07*32] cts
                        ; of master or (1/[Ix08*32] cts of slaved motor)
M169->D:$0046           ; #1 Compensation correction
```

$$P100 = \frac{(M162 + M164 + M169 - M175 * M167)}{I108 * 32}$$

P100 will report the same value as the online command **P** or the position window in the PMAC Executive program.

The addresses given are for Motor #1. For the registers for another motor x add (x-1)*$3C – (x-1)*60 – to the appropriate motor #1 address.)

```
M161->D:$0028                   ; #1 Commanded position (1/[Ix08*32] cts)
```

The motor commanded position registers contain the value in counts where the motor is commanded to move. It is set through **JOG** online commands or axis move commands (**X10**) inside motion programs.

To read this register in counts: `P161 = M161 / (I108*32)`

```
M162->D:$002B                   ; #1 Actual position (1/[Ix08*32] cts)
```

The actual position register contains the information read from the feedback sensor after it has been converted properly through the encoder conversion table and extended from a 24-bits register to a 48-bits register.

To read this register in counts: `P162 = M162 / (I108*32)`

```
M163->D:$080B                   ; #1 Target (end) position (1/[Ix08*32] cts)
```

This register contains the most recent programmed position and it is called the target position register. If I13>0, PMAC is in segmentation mode and the value of M163 corresponds to the last interpolated point calculated.

To read this register in counts: `P163 = M163 / (I108*32)`

```
M164->D:$0813                   ; #1 Position bias (1/[Ix08*32] cts)
```

- This register contains the offset specified in the axis definition command `#1->X + <offset>`.
- The online command **{axis}={constant}**or the motion program command **PSET** adds the specified offset to the existing M164 offset: `M164 = M164 + <new_offset>`.

To read this register in counts: `P164 = M164 / (I108*32)`

```
M165->L:$081F                   ; &1 X-axis target position (engineering units)
```

M165 contains the programmed axis position through a motion program, **X10** for example, in engineering units. It also gets updated by the online command "**{axis}={constant}**" or the motion program command **PSET.**

```
M166->X:$0033,0,24,S            ; #1 Actual velocity (1/[Ix09*32] cts/cyc)
```

M166 is the actual velocity register. For display purposes use the Motor filtered actual velocity, M174.

To read this register in cts/msec: `P166 = M166 * 8388608 / (I109 * 32 * I10 * (I160+1))`

```
M167->D:$002D                   ; #1 Present master ((handwheel) pos (1/[Ix07*32] cts
                                ; of master or (1/[Ix08*32] cts of slaved motor)
```

M167 is related to the master/slave relationship set through Ix05 and Ix06. It contains the present number of counts the master. To read this register in counts: `P167 = M167 / (I108*32)`

or `P167 = M167 / (I107*32)`

```
M169->D:$0046                ; #1 Compensation correction
```

Calculated leadscrew compensation correction according to actual position (M162) and the leadscrew compensation table set through the **`define comp`** command.

To read this register in counts: `P169 = M169 / (I108*32)`

```
M172->L:$082B                ; #1 Variable jog position/distance (counts)
```

Contains the distance for the **`J=*`** command.

```
Example:     M172=2000   J=*   ;Jog to position 2000 encoder counts
M173->Y:$0815,0,24,S         ; #1 Encoder home capture offset (counts)
```

Contains the home offset from the reset/power-on position. This is important for the capture/compare features.

Example:

```
If (M117=1)
  P103=M103-M173              ; Captured position minus offset
endif
M174->Y:$082A,24              ; #1 filtered actual velocity (1/[Ix09*32]
                              ; cts/servo cycle)
```

These registers contain the actual velocities averaged over the previous 80 real-time interrupt periods (80*[I8+1] servo cycles); this is useful for display purposes.

To read this register in cts/msec: `P174 = M174 * 8388608 / (I109 * 32 * I10 * (I160+1))`

```
M175->D:$0840                 ; #1 following error (1/[Ix08*32] cts)
```

Following error is the difference between motor desired and measured position at any instant. When the motor is open loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

$$P176 = \frac{M161 - M162 + M164 + M169 - M175 * M167}{I108 * 32}$$

To read this register in counts: `P176 = M175 / (I108*32)`

Homing Search Moves

If PMAC is not using an absolute feedback sensor that will keep a point of reference on the machine, the axis should be homed before running a motion program or **`JOG`** commands. If a home search procedure is not performed after power-up\reset, PMAC will consider the power-up\reset position as the zero point reference.

I-Variable	Description	I-Variable	Description
Ix03	Motor x Position Address	Ix26	Motor x Home Offset
Ix20	Motor x Jog/Home Acceleration Time	I902, I907,..	Encoder 0 Capture Control (PMAC 1 only)
Ix21	Motor x Jog/Home S-Curve Time	I903, I908,..	Encoder 0 Flag Select (PMAC 1 only)
Ix23	Motor x Homing Speed & Direction	Ix25	Motor x Flag Address

The flag channel used by Ix25 must match the position feedback channel used by Ix03 (indirectly from the conversion table).

Description	M-Variable	Description	M-Variable
ENC capture/compare position register	Mx03	Fault input status	Mx23
ENC 3rd channel input status	Mx19	Desired-velocity-zero bit	Mx33
HMFL input status	Mx20	In-position bit	Mx40
-LIM input status	Mx21	Home-complete bit	Mx45
+LIM input status	Mx22	Encoder home capture offset (counts)	Mx73

Home commands can be issued on the terminal window, a Motion Program or a PLC Program:

```
HOME1..8          ;Home axis 1 to 8 in a Motion Program. Program is halted
                  ;until home is completed.
#1HM              ;Online command for homing motor #1 from the terminal window.
CMD"#1HM"         ;Online command for homing motor #1 from a PLC program.
      while (…)   ;If a command statement is used in a PLC, the lines after
                  ;must have a while
      endwhile    ;loop waiting for the home procedure to complete (see main
                  ;PMAC manual for details).
```

HOMEZ is similar to these **HOME** commands but no motion will result in this kind of home search. PMAC will determine the zero reference home position in the place where the axes are found when **HOMEZ** is issued.

Command and Send Statements

Using the **COMMAND** or **CMD** statement, online commands could be issued from a PLC or Motion program having the same result as if they were issued from a host computer or a terminal window. Certain online commands might not be valid when issued from a running program. For example, a **JOG** command to a motor part of a coordinate system running a motion program will be invalid. It is a good idea to have I6 not set to 2 in early development so it will be known when PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hang up from a full response queue or from disturbing the normal host communications protocol.

Messages to a host computer or terminal window can be issued using the **SEND** command.

If there is no host on the port to which the message is sent, or the host is not ready to read the message, the message is left in the queue. If several messages back up in the queue this way, the program issuing the messages will halt execution until the messages are read. This is a common mistake when the **SEND** command is used outside of an Edge-Triggered condition in a PLC program. On the serial port, it is possible to send messages to a non-existent host by disabling the port handshaking with I1=1.

If a program, particularly a PLC program, sends messages immediately on power-up/reset, it can confuse a host-computer program (such as the PMAC Executive Program) that is trying to find PMAC by querying it and looking for a particular response.

It is possible, particularly in PLC programs, to order the sending of messages or command statements faster than the port can handle them. Usually, this will happen if the same **SEND** or **CMD** command is executed every scan through the PLC. For this reason, it is good practice to have at least one of the conditions that causes the **SEND** or **CMD** command to execute to be set false immediately to prevent execution of this **SEND** or **CMD** command on subsequent scans of the PLC.

MOTION PROGRAMS

PMAC can hold up to 256 motion programs at one time. Any coordinate system can run any of these programs at any time, even if another coordinate system is already executing the same program. PMAC can run as many motion programs simultaneously as there are coordinate systems defined on the card (up to eight). A motion program can call any other motion program as a subprogram, with or without arguments.

PMAC's motion program language is perhaps best described as a cross between a high-level computer language like BASIC or Pascal, and G-Code (RS-274) machine tool language. In fact, it can accept straight G-Code programs directly (provided it has been set up properly). It has the calculational and logical constructs of a computer language and move specification constructs similar to machine tool languages. Numerical values in the program can be specified as constants or expressions.

Motion or PLCs programs are entered in any text file to be downloaded afterwards to PMAC. PEWIN provides a built-in text editor for this purpose but any other text editor could be used conveniently. Once the code has been written, it can be downloaded to PMAC using PEWIN.

All PMAC commands can be issued from any terminal window communicating with PMAC. Online commands allow, for example, to jog motors, change variables, report variables values, start and stop programs, query for status information and even write short programs and PLCs. In fact, the downloading process is just a sequence of valid PMAC commands sent line by line from a particular text file.

How PMAC Executes a Motion Program

Basically, a PMAC program exists to pass data to the trajectory generator routines that compute the series of commanded positions for the motors every servo cycle. The motion program must be working ahead of the actual commanded move to keep the trajectory generators fed with data.

PMAC processes program lines either in zero, one, or two moves (including **DWELL**s and **DELAY**s) ahead. Calculating one move ahead is necessary in order to be able to blend moves together; calculating a second move ahead is necessary if proper acceleration and velocity limiting is to be done, or a three-point spline is to be calculated (**SPLINE** mode). For linear blended moves with I13 (move segmentation time) equal to zero (disabled), PMAC calculates two moves ahead, because the velocity and acceleration limits are enabled here. In all other cases, PMAC is calculating one move ahead.

No Moves Ahead	Two Moves Ahead	One Move Ahead
Rapid	Linear with I13=0	Linear with I13>0
Home	Spline 1	Circle
Dwell		PVT
`b1s` (step through the program)		
Ix92=1 (blending disabled)		

When a **RUN** command is given and every time the actual execution of programmed moves progresses into a new move, a flag is set saying it is time to do more calculations in the motion program for that coordinate system. At the next RTI, if this flag is set, PMAC will start working through the motion program processing each command encountered. This can include multiple modal statements, calculation statements, and logical control statements. Program calculations will continue (which means no background tasks will be executed) until one of the following conditions occurs:

1. The next move, a **DWELL** command or a **PSET** statement is found and calculated.
2. End of, or halt to the program (e.g. **STOP**) is encountered.
3. Two jumps backward in the program (from **ENDWHILE** or **GOTO**) are performed.
4. A **WAIT** statement is encountered (usually in a **WHILE** loop).

If calculations stop on condition 1 or 2, the calculation flag is cleared and will not be set again until actual motion progresses into the next move (1) or a new **RUN** command is given (2). If calculations stop on conditions 3 or 4, the flag remains set, so calculations will resume at the next RTI. In these cases there is an empty (no-motion) loop, the motion program acts much like a PLC 0 during this period.

If PMAC cannot finish calculating the trajectory for a move by the time execution of that move should begin, PMAC will abort the program, showing a run-time error in its status word.

Coordinate Systems

A coordinate system in PMAC is a grouping of one or more motors for the purpose of synchronizing movements. A coordinate system (even with only one motor) can run a motion program; a motor cannot. PMAC can have up to eight coordinate systems, addressed as **&1** to **&8**, in a flexible fashion (e.g. eight coordinate systems of one motor each, one coordinate system of eight motors, four coordinate systems of two motors each, etc.).

In general, if certain motors should move in a coordinated fashion, put them in the same coordinate system. To move them independently of each other, put them in separate coordinate systems. Different coordinate systems can run separate programs at different times (including overlapping times), or even run the same program at different (or overlapping) times.

A coordinate system must be established first by assigning axes to motors in axis definition statements. A coordinate system must have at least one motor assigned to an axis within that system, or it cannot run a motion program, even non-motion parts of it. When a program is written for a coordinate system, if simultaneous motions are wanted of multiple motors, their move commands are simply put on the same line and the moves will be coordinated.

Axis Definitions

An axis is an element of a coordinate system. It is similar to a motor, but not the same thing. An axis is referred to by letter. There can be up to eight axes in a coordinate system, selected from X, Y, Z, A, B, C, U, V, and W. An axis is defined by assigning it to a motor with a scaling factor and an offset (X, Y, and Z may be defined as linear combinations of three motors, as may U, V, and W). The variables associated with an axis are scaled floating-point values.

In the vast majority of cases, there will be a one-to-one correspondence between motors and axes. That is, a single motor is assigned to a single axis in a coordinate system. Even when this is the case, however, the matching motor and axis are not completely synonymous. The axis is scaled into engineering units, and deals only with commanded positions. Except for the **PMATCH** function, calculations go only from axis commanded positions to motor commanded positions, not the other way around.

More than one motor may be assigned to the same axis in a coordinate system. This is common in gantry systems, where motors on opposite ends of the crosspiece are always trying to do the same movement. By assigning multiple motors to the same axis, a single programmed axis move in a program causes identical commanded moves in multiple motors. Commonly, this is done with two motors, but up to eight motors can be used in this manner with PMAC. Remember that the motors still have independent servo loops, and that the actual motor positions will not necessarily be exactly the same.

An axis in a coordinate system can have no motors attached to it (a phantom axis), in which case programmed moves for that axis cause no movement, although the fact that a move was programmed for that axis can affect the moves of other axes and motors. For instance, if sinusoidal profiles are desired on a single axis, the easiest way to do this is to have a second, phantom axis and program circularly interpolated moves.

Axis Definition Statements

A coordinate system is established by using axis definition statements. An axis is defined by matching a motor (which is numbered) to one or more axes (which are specified by letter).

The simplest axis definition statement is something like **`#1->X`**. This simply assigns motor #1 to the X axis of the currently addressed coordinate system. When an X axis move is executed in this coordinate system, motor #1 will make the move. In addition, the axis definition statement defines the scaling of the axis' user units. For instance, **`#1->10000X`** also matches motor #1 to the X axis, but this statement sets 10,000 encoder counts to one X-axis user unit (e.g. inches or centimeters). Usually, this scaling feature is universally used. Once the scaling has been defined in this statement, the axis can be programed in engineering units without ever needing to deal with the scaling again.

Permitted Axis Names: **X,Y,Z,U,V,W,A,B,C**

X,Y,Z: Traditionally Main Linear Axes

- Matrix Axis Definition
- Matrix Axis Transformation
- Circular Interpolation
- Cutter Radius Compensation

A,B,C: Traditionally Rotary Axes

(A rotates about X, B about Y, C about Z)

- Position Rollover (Ix27)

U,V,W: Traditionally Secondary Linear Axes

- Matrix Axis Definition

Writing a Motion Program

1. Open a program buffer with **`OPEN PROG {constant}`** where **`{constant}`** is an integer from 1 to 32767 representing the motion program to be opened.
2. Motion Programs 1000, 1001, 1002 and 1003 can contain G-codes, M-codes, T-codes and D-codes for machine tool G-codes or RS-274 programming method. These buffers can be used for general PMAC code programming as long as G-codes programming is not needed in PMAC.
3. PMAC can hold up to 256 motion programs at one time. For continuous execution of programs larger than PMAC's memory space, a special PROG0, the rotary motion program buffers, allow for the downloading of program lines during the execution of the program and for the overwriting of already executed program lines.
4. The **`CLEAR`** command empties the currently opened program, PLC, rotary, etc. buffer.
5. Many of the statements in PMAC motion programs are modal in nature. These include move modes, which specify what type of trajectory a move command will generate; this category includes **`LINEAR`**, **`RAPID`**, **`CIRCLE`**, **`PVT`**, and **`SPLINE`**.
6. Moves can be specified either incrementally (distance) or absolutely (location) – individually selectable by axis – with the **`INC`** and **`ABS`** commands. Move times (**`TA`**, **`TS`**, and **`TM`**) and/or speeds (**`F`**), are implemented in modal commands. Modal commands can precede the move commands they are to affect, or they can be on the same line as the first of these move commands.
7. The move commands themselves consist of a one-letter axis-specifier followed by one or two values (constant or expression). All axes specified on the same line will move simultaneously in a coordinated fashion on execution of the line; consecutive lines execute sequentially (with or without stops in between, as determined by the mode). Depending on the modes in effect, the specified values can mean destination, distance, and/or velocity.
8. If the move times (**`TA`**, **`TS`**, and **`TM`**) and/or speeds (**`F`**) are not declared specifically in the motion program the default parameters from the I-variables Ix87, Ix88 and Ix89 will be used instead.

Note:

Do not rely on these parameters to declare the move times in the program. This will keep the move parameters with the move commands, lessening the chances of future errors, and making debugging easier.

9. In a motion program, PMAC has **WHILE** loops and **IF..ELSE** branches that control program flow. These constructs can be nested indefinitely. In addition, there are **GOTO** statements, with either constant or variable arguments (the variable **GOTO** can perform the same function as a **CASE** statement). **GOSUB** statements (constant or variable destination) allow subroutines to be executed within a program. **CALL** statements permit other programs to be entered as subprograms. Entry to the subprogram does not have to be at the beginning -- the statement **CALL 20.15000** causes entry into Program 20 at line **N15000**. **GOSUBs** and **CALLs** can be nested only 15 deep.
10. The **CLOSE** statement closes the currently opened buffer. This should be used immediately after the entry of a motion, PLC, rotary, etc. buffer. If the buffer is left open, subsequent statements that are intended as on-line commands (e.g. **P1=0**) will get entered into the buffer instead. It is good practice to have close at the beginning and end of any file to be downloaded to PMAC. When PMAC receives a **CLOSE** command, it appends a **RETURN** statement to the end of the open program buffer automatically. If any program or PLC in PMAC is structured improperly (e.g. no **ENDIF** or **ENDWHILE** to match an **IF** or **WHILE**), PMAC will report an ERR003 at the **CLOSE** command for any buffer until the problem is fixed.

Example:

```
close                   ; Close any buffer opened
delete gather           ; Erase unwanted gathered data
undefine all            ; Erase coordinate definitions in all coordinate systems
#1->2000X               ; Motor #1 is defined as axes X
OPEN PROG 1 CLEAR       ; Open buffer to be written
LINEAR                  ; Linear interpolation
INC                     ; Incremental mode
TA100                   ; Acceleration time is 100 msec
TS0                     ; No S-curve acceleration component
F50                     ; Feedrate is 50 Units per Ix90 msec
X1                      ; One unit of distance, 2000 encoder counts
CLOSE                   ; Close written buffer, program one
```

Running a Motion Program

1. Select the coordinate system where the motion program will be running. This is done by issuing the **&** command followed by the coordinate system number, e.g., **&1** for the coordinate system one.
2. Select the program that to run with the **B{constant}** command, where the **{constant}** represents the number of the motion program buffer. Use the **B** command to change motion programs, and after any motion program buffer has been opened. It is not necessary to use it if running the same motion program repeatedly without modification; when PMAC finishes executing a motion program, the program counter for the coordinate system is set automatically to point to the beginning of that program, ready to run it again.
3. Once it is pointing to the motion program to run, issue the command to start execution of the program. For continuous execution of the program, use the **R** command (**<CTRL-R>** for all coordinate systems simultaneously). The program will execute all the way through unless stopped by command or an error condition.
4. To execute just one move or a small section of the program, use the **S** command (**<CTRL-S>** for all coordinate systems simultaneously). The program will execute to the first move **DWELL**, **DELAY**, or if it first encounters a **BLOCKSTART** command, it will execute to the **BLOCKSTOP** command.

5. When a **RUN** or **STEP** command is issued, PMAC checks the coordinate system to make sure it is in proper working order. If it finds anything in the coordinate system is not set up properly, it will reject the command, sending a **<BELL>** command back to the host. If I6 is set to 1 or 3, it will report an error number as well telling the reason the command was rejected. PMAC will reject a **RUN** or **STEP** command for any of the following reasons:
   - A motor in the coordinate system has both overtravel limits tripped (ERR010)
   - A motor in the coordinate system is currently executing a move (ERR011)
   - A motor in the coordinate system is not in closed-loop control (ERR012)
   - A motor in the coordinate system in not activated {Ix00=0} (ERR013)
   - There are no motors assigned to the coordinate system (ERR014)
   - A fixed (non-rotary) motion program buffer is open (ERR015)
   - No motion program has been pointed to (ERR016)
   - After a **/** or **** stop command, a motor in the coordinate system is not at the stop point (ERR017)
6. Before starting the program, issue a **CTRL+A** command to PMAC to ensure that all the motors will be potentially in closed loop and that all previous motions are aborted. Also, if in doubt, the functioning of each motor can be checked individually prior to running a program by means of **JOG** commands. For example, **#1J^2000** will make motor #1 move 2000 encoder counts and that would confirm if the motors are able to run motion programs or not.
7. All motors in the addressed coordinate system have to be ready to run a motion program. Depending on Ix25, even if one motor defined in the coordinate system is not closing the loop, all motors in the coordinate system could be brought down to impede the running of any motion program.
8. Sometimes the feedrate override for the current addressed coordinate system is set at zero and no motion will occur as a result of this. Check the feedrate override parameter by issuing a **&1%** command on the terminal window (replace 1 for the appropriate coordinate system number). If it is zero or too low, set it to an appropriate value. The **&1%100** command will set it to 100 %.
9. For troubleshooting purposes, change the feedrate override to lower than 100% value. If the program is run for the first time, a preceding **%10** command could be issued to run the motion program in slow motion. Running the program slowly will allow observing the programmed path more clearly, it will demand less calculation time from PMAC and it will prevent damages due to potentially wrong acceleration and/or feedrate parameters.
10. A motion program can be stopped by sending **&1a** or, for simplicity, a **CTRL+A** command which will stop any motion.
11. If the motion of any axis becomes uncontrollable and it should be stopped, issue a **CTRL+K** command to kill all the motors in PMAC (disabling the amplifier enable line if connected). However, the motor might come to a stop in an uncontrollable way and proceed to move due to its own inertia.
12. A motion program can be stopped also by issuing a **CTRL+Q** command. The last programmed moves in the buffer will be completed before the program quits execution. It can be resumed by issuing an **R** command alone, without first pointing to the beginning of the buffer by the **B** command.

Subroutines and Subprograms

It is possible to create subroutines and subprograms in PMAC motion programs to create well-structured modular programs with re-usable subroutines. The **GOSUBx** command in a motion program causes a jump to line label Nx of the same motion program. Program execution will jump back to the command immediately following the **GOSUB** when a **RETURN** command is encountered. This creates a subroutine.

The **CALLx** command in a motion program causes a jump to PROG x, with a jump back to the command immediately following the **CALL** when a **RETURN** command is encountered. If x is an integer, the jump is to the beginning of PROG x; if there is a fractional component to x, the jump is to line label **N(y*100,000)**, where y is the fractional part of x. This structure permits the creation of special subprograms, either as a single subroutine, or as a collection of subroutines, that can be called from other motion programs.

The **PRELUDE** command allows creating an automatic subprogram call before each move command or other letter-number command in a motion program.

Passing Arguments to Subroutines

These subprogram calls are made more powerful by use of the **READ** statement. The **READ** statement in the subprogram can go back up to the calling line and pick off values (associated with other letters) to be used as arguments in the subprogram. The value after an A would be placed in variable Q101 for the coordinate system executing the program, the value after a B would be placed in Q102, and so on (Z value goes in Q126). Letters N or O cannot be passed.

This structure is useful particularly for creating machine tool style programs in which the syntax must consist solely of letter number combinations in the parts program. Since PMAC treats the G, M, T, and D codes as special subroutine calls, the **READ** statement can be used to let the subroutine access values on the part-program line after the code.

The **READ** statement also provides the capability of seeing what arguments have actually been passed. The bits of Q100 for the coordinate system are used to note whether arguments have been passed successfully; bit 0 is 1 if an A argument has been passed, bit 1 is 1 if a B argument has been passed, and so on, with bit 25 set to 1 if a Z argument has been passed. The corresponding bit for any argument not passed in the latest subroutine or subprogram call is set to 0.

Example:

```
close delete gather undefine all
#1->2000X
open prog1 clear
LINEAR INC TA100 TS0 F50   ;Mode and timing parameters
gosub 100 H10              ;Subroutine call passing parameter H with value 10
return                     ;End of the main program section (execution ends)
n100                       ;Subroutines section. First subroutine labeled 100
read(h)                    ;Read the H parameter value passed
IF (Q100 & $80 > 0)        ;If the H parameter has been passed …
      X(Q108)              ;Use the H parameter value contained in Q108
endif
return                     ;End of the subroutine labeled 100
close                      ;End of the motion program code
```

G, M, T, and D-Codes (Machine-Tool Style Programs)

PMAC permits the execution of machine tool style RS-274 (G-Code) programs by treating G, M, T, and D codes as subroutine calls. This permits the machine tool manufacturer to customize the codes for their own machine, but it requires the manufacturer to do the actual implementation of the subroutines that will execute the desired actions.

When PMAC encounters the letter G with a value in a motion program, it treats the command as a call to motion program 10n0, where n is the hundreds' digit of the value. The value without the hundreds' digit (modulo 100 in mathematical terms) controls the line label within program 10n0 to which operation will jump – this value is multiplied by 1000 to specify the number of the line label. When a return statement is encountered, it will jump back to the calling program.

For example: G17 will cause a jump to N17000 of PROG 1000; G117 will cause a jump to N17000 of PROG 1010; G973.1 will cause a jump to N73100 of PROG 1090.

M-codes are the same, except they use PROG 10n1; T-codes use PROG 10n2; D-codes use PROG 10n3.

Most of the time, these codes have numbers within the range 0 to 99, so only PROGs 1000, 1001, 1002, and 1003 are required to execute them. For those who want to extend code numbers past 100, PROGs 1010, 1011, etc. will be required to execute them.

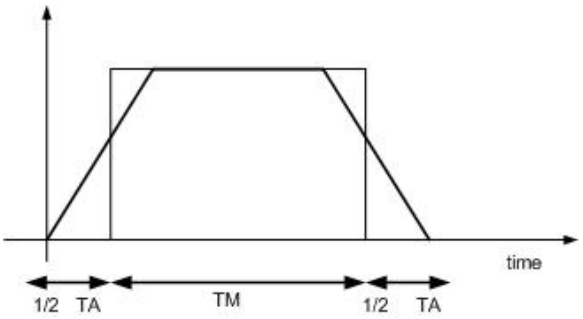
Linear Blended Moves

The move time is set directly by TM or indirectly based on the the distances and feedrate (F) parameters set:

```
TM100              or      FRAX(X,Y)
X3 Y4
                           X3 Y4 F50     ; TM = I190 · √(3² + 4²) / 50 = 5000 / 50 = 100 msec
```

- If the move time above calculated is less than the TA time set, the move time used will be the TA time instead. In this case, the programmed TA (or `2*TS` if `TA<2*TS`) results in the minimum move time of a linearly interpolated move.
- If the TA programmed results to be less than twice the TS programmed, `TA<2*TS`, the TA time used will be `2*TS` instead.
- The acceleration time TA of a blended move cannot be longer than two times the previous TM minus the previous TA, otherwise the value 2*(TM- ½ TA) will be used as the current TA instead.
- The safety variables Ix16 and Ix17 will override these parameters if they are found to violate the programmed limits.
  - If TM < TA, TM = TA
  - If TA < 2*TS, TA = 2*TS
  - If TA_{i+1} > 2*(TM_i- ½ TA_i), TA_{i+1} = 2*(TM_i - ½ TA_i)

Example:



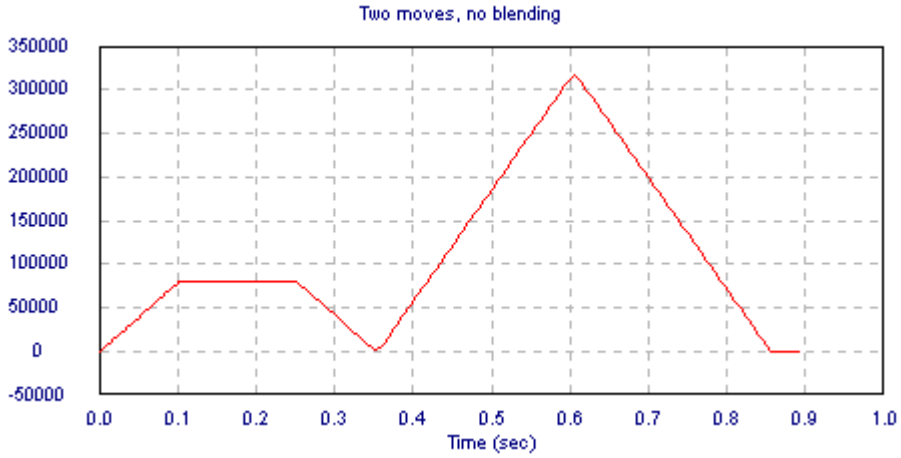
To illustrate how PMAC blends linear moves, a series of velocity Vs time profiles will be shown. The moves are defined with zero S-curve components. The concepts described here could be used for non-zero S-curve linear moves.

1. Consider the following motion program code:

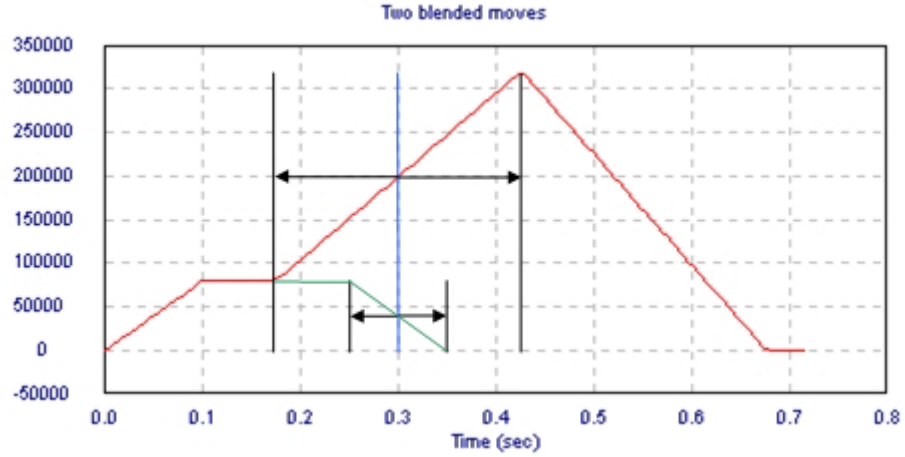
```
close
delete gather
undefine all
&1
#1->2000x
OPEN PROG 1 CLEAR
      LINEAR          ; Linear mode
      INC             ; Incremental mode
      TA100           ; The acceleration time is 100 msec, TA1
```

```
        TS0             ; No S-curve component
        TM250           ; Move time is 250 msec, TM1
        X10             ; Move distance is 10 units, 20000 counts
        TA250           ; Acceleration \ deceleration of the blended move is
250 msec , TA2
        X40             ; Move distance is 40 units, 80000 counts
CLOSE
```

2. The two move commands are plotted without blending, placing a **DWELL0** command in between the two moves:

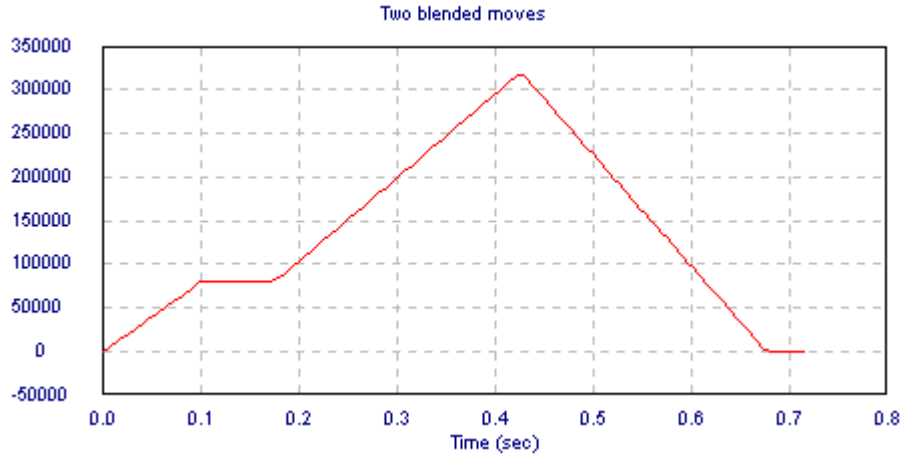


3. The two moves are now plotted with the blending mode activated. To find out the blending point, trace straight lines through the middle point of each acceleration lines of both velocity profiles:



Observations

1. The total move time is given by: $\frac{TA_1}{2} + TM_1 + TM_2 + \frac{TA_2}{2} = 50 + 250 + 250 + 125 = 675\ msec$
2. The acceleration of the second blended move can be extended only up to a certain limit, 2*(TM- ½ TA):

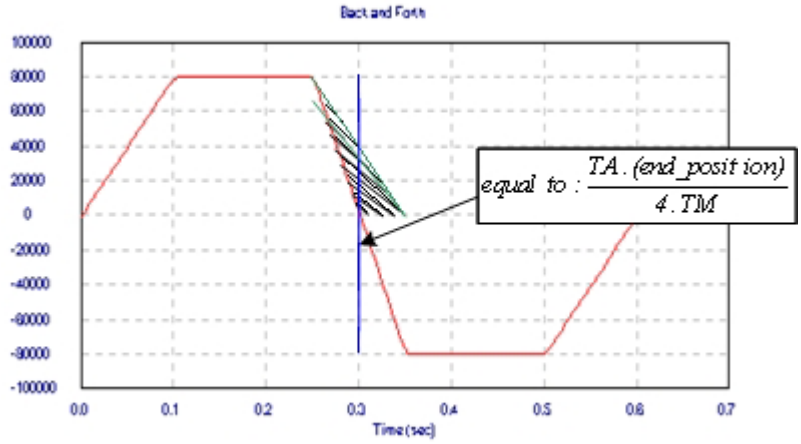


PMAC looks two moves ahead of actual move execution to perform its acceleration limit and can recalculate these two moves to keep the accelerations under the Ix17 limit. However, there are cases where more than two moves, some much more than two, would have to be recalculated in order to keep the accelerations under the limit. In these cases, PMAC will limit the accelerations as much as it can, but because the earlier moves have been executed already, they cannot be undone and therefore, the acceleration limit will be exceeded.

3. When performing a blended move that involves a change of direction, the end point might not be reached.

Example:

```
TA100
TM250

      X10       ; This would reach only to position =   10 − 100.10 / 4.250 = 9

      X-10
```



In order to reach the desired position, since the move involves a change in direction and stop, simply place a **DWELL0** command between moves. This command will disable blending for that particular move:

```
TA100
TM250
X10
DWELL0
X-10
```

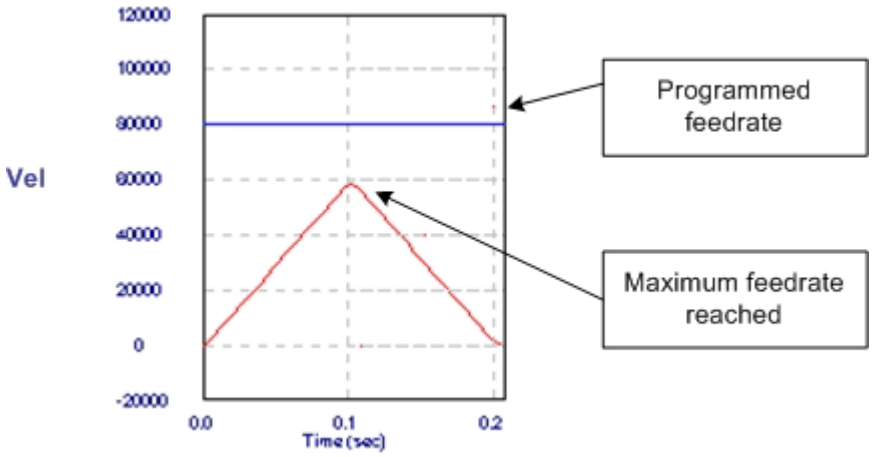
4. Since the value of TA determines the minimum time in which a programmed move can be executed, it could limit the maximum move velocity and therefore the programmed feedrate might not be reached. This is seen in triangular velocity profile moves types, especially when a sequence of short distance moves is programmed.

Example:

```
close
delete gather
undefine all
&1
#1->2000X
I190=1000
OPEN PROG 1 CLEAR
      LINEAR          ; Linear mode
      INC             ; Incremental mode
      TA100           ; Acceleration time is 100 msec, TA1
      TS0             ; No S-curve component
      F40             ; Feedrate is 40 length_units / second
      X3              ; TM = 3.I190/40 = 3000/40 = 75 msec
CLOSE
```

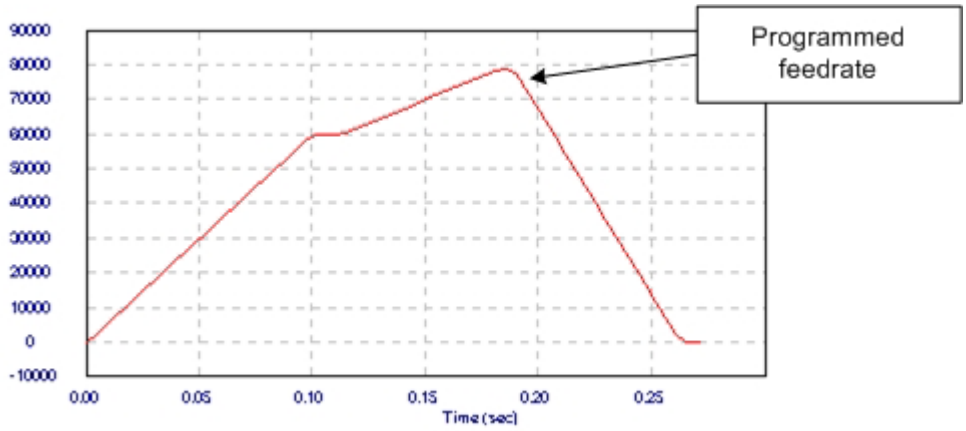
Since the calculated TM for the given feedrate is 75 msec and the programmed TA for this move is 100 msec, the TM used will be 100 msec instead. This yields the following feedrate value instead of the programmed one:

$$F = \frac{3.I190}{100} = \frac{3000}{100} = 30 \frac{units\ of\ distance}{second}$$

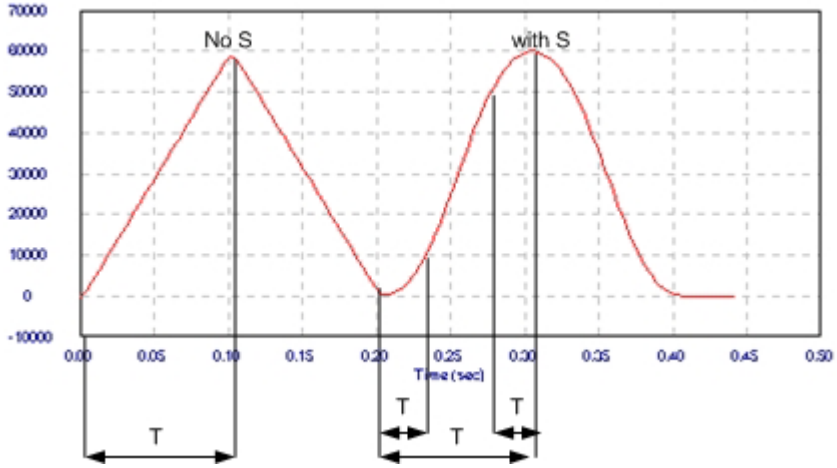


To be able to reach the desired velocity, a longer move can be performed split into two sections. The first move will be executed using a suitable TA to get the motor to move from rest. The second move will have a lower acceleration time TA in order to decrease the move time TM and so reach the programmed feedrate.

```
OPEN PROG 1
CLEAR
       LINEAR
       INC
       TS0
       F40
       TA100
       X3
       TA75
       X3
       CLOSE
```

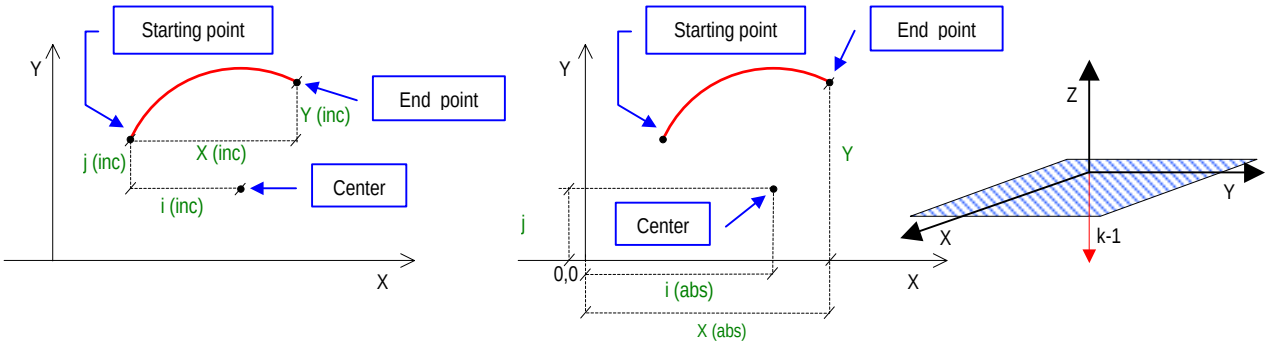


5. All the previous analysis was performed assuming a zero S curve component. A move executed with an S curve component will be similar in shape but with rounded sections at the beginning and end of the acceleration lines.



Circular Interpolation

PMAC allows circular interpolation on the X, Y, and Z-axes in a coordinate system. As with linear blended moves, TA and TS control the acceleration to and from a stop, and between moves. Circular blended moves can be feedrate-specified (**F**) or time-specified (**TM**), just as with linear moves. It is possible to change back and forth between linear and circular moves without stopping. When linear interpolation is needed, enter the **LINEAR** command and Circle1 or Circle2 for circular interpolation.



1. PMAC performs arc moves by segmenting the arc and performing the best cubic fit on each segment. I-Variable I13 determines the time for each segment. I13 must be set greater than zero to put PMAC into this segmentation mode in order for arc moves to be done. If I13 is set to zero, circular arc moves will be done in linear fashion.

   The practical range of I13 for the circular interpolation mode is 5-10 msec. A value of 10 msec is recommended for most applications, a lower than 10 msec I13 value will improve the accuracy of the interpolation (calculating points of the curve more often) but will also consume more of PMAC's total computational power.

2. When PMAC is segmenting moves (I13 > 0) automatically, which is required for Circular Interpolation. The Ix17 accelerations limits and the Ix16 velocity limits are not observed.

3. Any axes used in the circular interpolation are automatically feedrate axes for circular moves, even if they were not so specified in an **FRAX** command. Other axes may or may not be feedrate axes. Any non-feedrate axes commanded to move in the same move command will be linearly interpolated so as to finish in the same time. This permits easy helical interpolation.

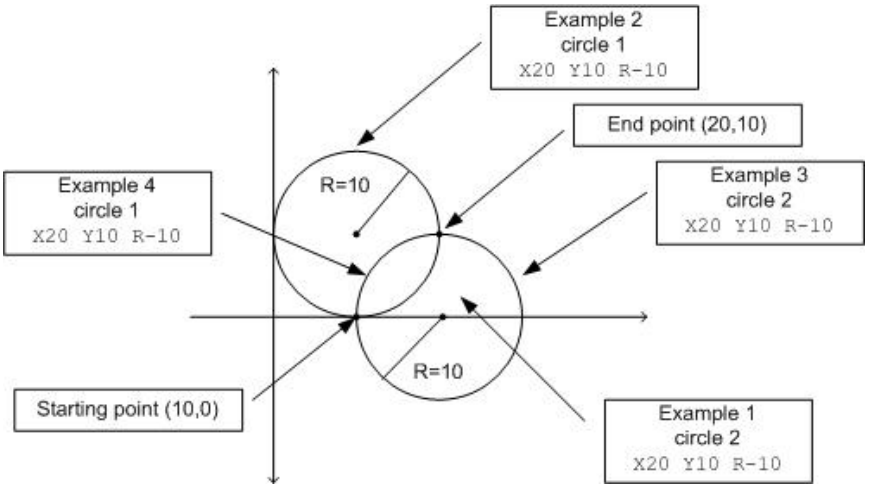
4. The plane for the circular arc must have been defined by the **NORMAL** command (the default -- **NORMAL K-1** -- defines the XY plane). This command can define only planes in XYZ-space, which means that only the X, Y, and Z axes can be used for circular interpolation. Other axes specified in the same move command will be interpolated linearly to finish in the same time. The most commonly used planes are:

   ```
   NORMAL K-1      ; XY plane -- equivalent to G17
   NORMAL J-1      ; ZX plane -- equivalent to G18
   NORMAL I-1      ; YZ plane -- equivalent to G19
   ```

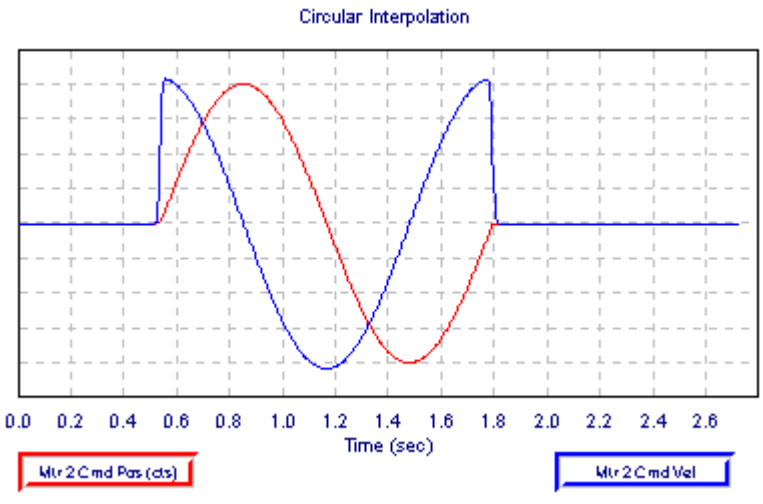
5. To put the program in circular mode, use the **CIRCLE1** program command for clockwise arcs (G02 equivalent) or **CIRCLE2** for counterclockwise arcs (G03 equivalent). **LINEAR** will restore PMAC to linear blended moves. Once in circular mode, a circular move is specified with a move command specifying the move endpoint and either the vector to the arc center or the distance (radius) to the center. The endpoint may be specified either as a position or as a distance from the starting point, depending on whether the axes are in absolute (**ABS**) or incremental (**INC**) mode (individually specifiable).

   ```
   X{Data} Y{Data} R{Data}           ;Radius of the circle is given
   X{Data} Y{Data} I{Data} J{Data}   ;Center coordinates of the circle are given
   ```

6. If the vector method of locating the arc center is used, the vector is specified by its I, J, and K components (I specifies the component parallel to the X axis, J to the Y axis, and K to the Z axis). This vector can be specified as a distance from the starting point (i.e. incrementally), or from the XYZ origin (i.e. absolutely). The choice is made by specifying R in an ABS or INC statement (e.g. **ABS (R)** or **INC (R)**). This affects I, J, and K specifiers together. (**ABS** and **INC** without arguments affect all axes, but leave the vectors unchanged). The default is for incremental vector specification.
7. PMAC's convention is to take the short arc path if the R value is positive and the long arc path if R is negative:
   - If the value of R is positive, the arc to the move endpoint is the short route (<=180 degrees)
   - If the value of R is negative, the arc to the move endpoint is the long route (>=180 degrees)



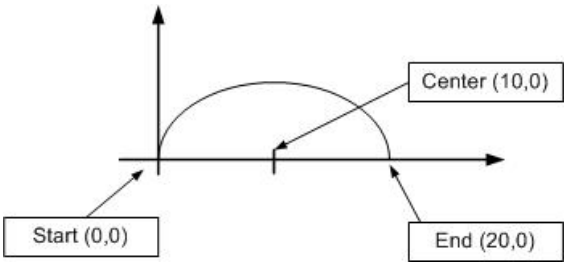
8. When performing a circular interpolation, the individual axes describe a position Vs time profile close to a sine and cosine shape. This is true also for their velocity and acceleration profiles. Therefore, circular interpolation makes an ideal feature to described trigonometric profiles. Further, the period (and so frequency) of the sine or cosine waves can be set by the total move time given by TA+TM.




```
close
delete gather
undefine all
&1
#2->2000Y    ;X is phantom
open prog1 clear
inc
inc (r)
ta300
ts0
tm1000       ;TA+TM is period
i13=10
normal k-1   ;X-Y plane
circle1      ;clockwise
x0 y0 i10    ;complete circle
close
&1b1r
```

Example:

```
I13=10          ;Move Segmentation Time
NORMAL K-1      ;XY plane
INC             ;Incremental End Point definition
INC (R)         ;Incremental Center Vector
definition
CIRCLE 1        ;Clockwise circle
X20 Y0 I10 J0   ;Arc move
```



Splined Moves

PMAC can perform cubic splines (cubic in terms of the position vs. time equations) to blend together a series of points on an axis. Splining is suited particularly to odd (non-Cartesian) geometries, such as radial tables and rotary-axis robots, where there are odd axis profile shapes even for regular tip movements.

In **SPLINE1** mode, a long move is split into equal-time segments, each of TA time. Each axis is given a destination position in the motion program for each segment with a normal move command line like **X1000Y2000**. Looking at the move command before this and the move command after this, PMAC creates a cubic position-vs.-time curve for each axis so that there is no sudden change of either velocity or acceleration at the segment boundaries. The commanded position at the segment boundary may be relaxed slightly to meet the velocity and acceleration constraints.

PMAC can work only with integer (millisecond) values for the TA segment times. If a non-integer value is specified for the TA time, PMAC will round it to the nearest integer automatically. It will not report an error. This rounding will change the speeds and times for the trajectory.

At the beginning and end of a series of splined moves, PMAC adds a zero-distance segment of TA time for each axis automatically, and performs the spline between this segment and the adjacent one. This results in a S-curve acceleration to and from a stop.

PMAC's **SPLINE2** mode is very similar to the **SPLINE1** mode, except that the requirement that the TA spline segment time remain constant is removed.

PVT-Mode Moves

For the user who desires more direct control over the trajectory profile, PMAC offers Position-Velocity-Time (PVT) mode moves. In these moves, the axis states are specified directly at the transitions between moves (unlike in blended moves). This requires more calculation by the host, but allows tighter control of the profile shape. For each piece of a move, the end position or distance, the end velocity, and the piece time are specified.

PMAC is put in this mode with the program statement **PVT{data}**, where **{data}** is a constant, variable, or expression, representing the piece time in milliseconds. This value should be an integer; if it is not, PMAC will round it to the nearest integer. The piece time may be changed between pieces, either with another **PVT{data}** statement, or with a **TA{data}** statement. The program is taken out of this mode with another move mode statement (e.g. **LINEAR**, **RAPID**, **CIRCLE**, **SPLINE**).

A PVT mode move is specified for each axis to be moved with a statement of the form **{axis}{data}:{data}**, where **{axis}** is a letter specifying the axis, the first **{data}** is a value specifying the end position or the piece distance, depending on whether the axis is in absolute or incremental mode, respectively, and the second **{data}** is a value representing the ending velocity.

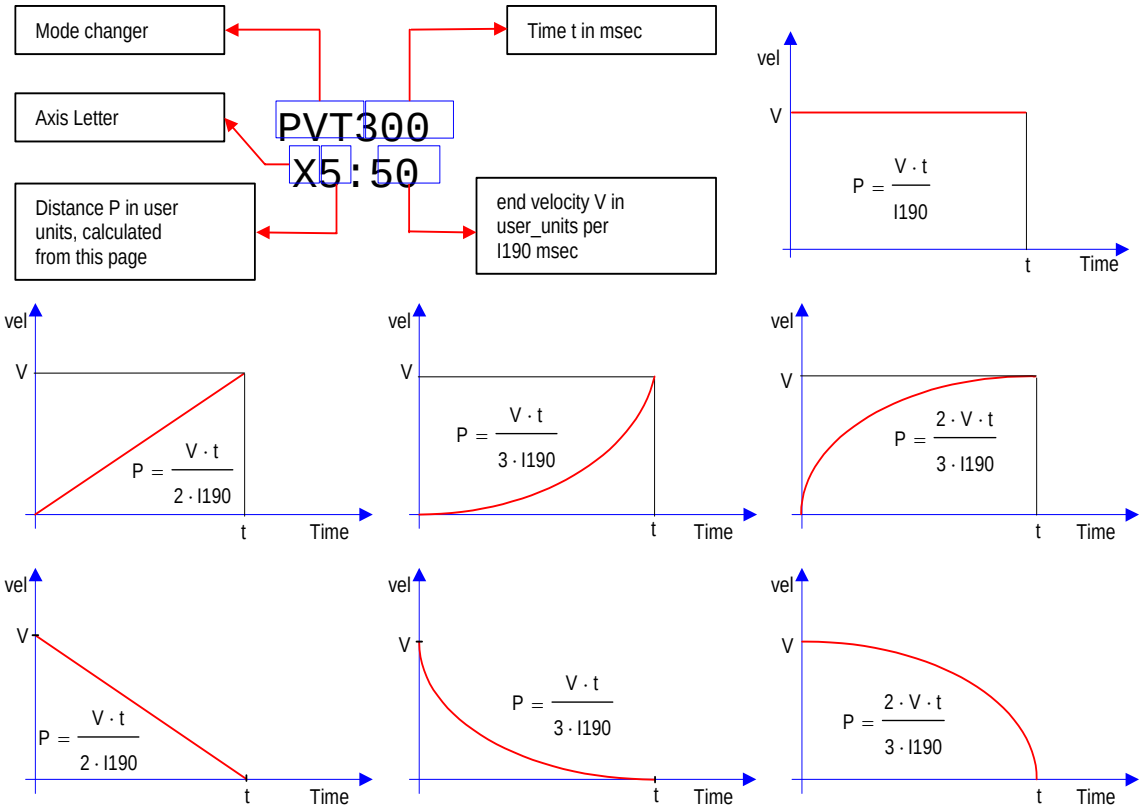
The units for position or distance are the user length or angle units for the axis, as set in the Axis Definition statement. The units for velocity are defined as length units divided by time units, where the length units are the same as those for position or distance, and the time units are defined by variable Ix90 for the coordinate system (feedrate time units). The velocity specified for an axis is a signed quantity.

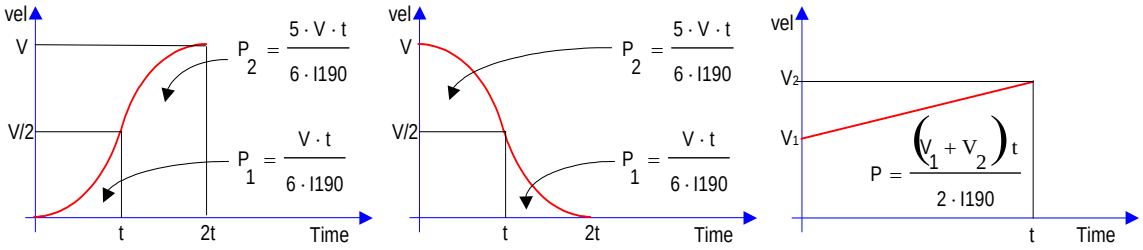
From the specified parameters for the move piece, and the beginning position and velocity (from the end of the previous piece), PMAC computes the only third-order position trajectory path to meet the constraints. This results in linearly changing acceleration, a parabolic velocity profile, and a cubic position profile for the piece.

Since a non-zero end velocity for the move can be specified (directly or indirectly), it is not a good idea to step through a program of transition-point moves, and great care must be exercised in downloading these moves in real time. With the use of the **BLOCKSTART** and **BLOCKSTOP** statements surrounding a series of PVT moves, the last of which has a zero end velocity, it is possible to use a **STEP** command to execute only part of a program.

The PVT mode is the most useful for creating arbitrary trajectory profiles. It provides a building block approach to putting together parabolic velocity segments to create whatever overall profile is desired. The following diagram shows common velocity segment profiles. PVT mode can create any profile that any other move mode can.

PVT mode provides excellent contouring capability, because it takes the interpolated commanded path exactly through the programmed points. It creates a path known as a Hermite Spline. **LINEAR** and **SPLINE** modes are second and third order B-splines, respectively, which pass to the inside of programmed points. Compared to PMAC's SPLINE mode, PVT produces a more accurate profile.

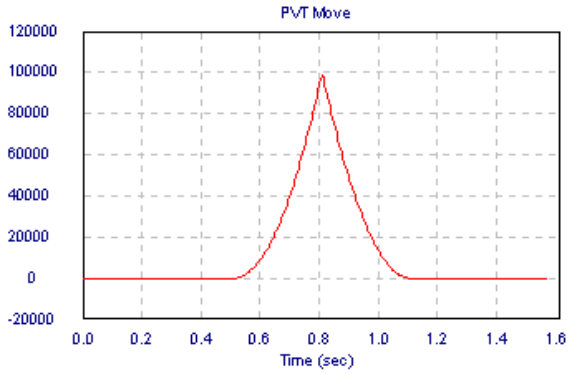




Replace I190 for the appropriate Ix90 variable according to coordinate system x.

Example:

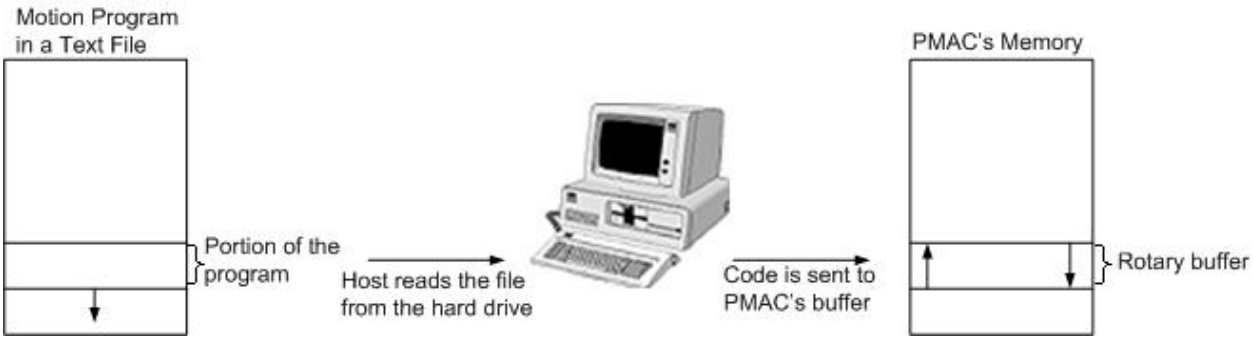
```
close delete gather undefine all
&1 #1->2000X

OPEN PROG 1 CLEAR
INC
PVT300    ;Time is 300 msec per section

X5:50     ; P = 50 user_units / I190 msec . 300 msec / 3 = 15000 / 3000 = 5 user_units

X5:0      ; P = 50 user_units / I190 msec . 300 msec / 3 = 15000 / 3000 = 5 user_units

CLOSE
```



Other Programming Features

Rotary Motion Program Buffers

PMAC has a limited memory space shared for motion programs, PLCs, compensation tables and gathering buffers. The rotary motion program buffers allow running motion programs larger than the available space in PMAC's memory.



Communication routines provided by Delta Tau have the necessary code to implement this feature in a host computer.

Internal Time Base, the Feedrate Override

Each coordinate system has its own time base that helps control the speed of interpolated moves in that coordinate system.

If Ix93 is set at default, this parameter could be changed by different means:

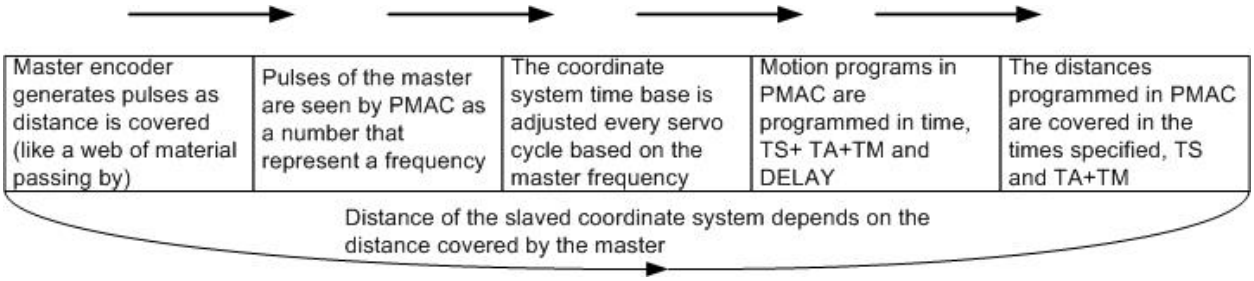
- %*n*, where $0 < n < 100$ — Online or CMD command that runs all motion commands in slow motion.
- %*n*, where $100 < n \leq 225$ — Online or CMD command that runs all motion commands proportionally faster.
- %0 — Online or CMD command that freezes all motions and timing in that C.S.

- %100 — Online or CMD command that restores the real-time reference (1 msec = 1 msec).
- M197 = I10 — Suggested M-Variable for time base change. Equal to I10 is 100%, equal to 0 is 0%.

The variable Ix94 controls the rate at which the time base changes: $Ix94 = \frac{I10^2}{t \cdot 2^{23}}$, where t is the slew rate time in msec.

External Time Base Control (Electronic Cams)

The time reference of each coordinate system can be changed from the default internal reference, controlled by the **%** command and variables Mx97, to an external source (usually a frequency reference from a master encoder). A simple change of the variable Ix93 allows switching between the internal time base and an external source. In this fashion, motion programs can be developed and tested running in real-time (internal time base) and synchronized later to a master frequency when proven to be functional and completed.



The only setup part of this feature is an entry in the conversion table that will also indicate a scale factor for the maximum frequency that the master can possibly input to PMAC. This maximum frequency will represent 100% or real time.

Position Following (Electronic Gearing)

PMAC has several methods of coordinating the axes under its control to axes not under its control. The simplest method is basic position following. This is a motor-by-motor function, not a coordinate system function as time-base following. An encoder signal from the master axis (which is not under PMAC's control) is fed into one of PMAC's encoder inputs. Typically, this master signal is either from an open-loop drive or a handwheel knob. Ix05 and Ix06 control this function.

Cutter Radius Compensation

PMAC provides the capability for performing cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z-axes, which should be physically perpendicular to each other. The compensation offsets the described path of motion perpendicular to the path by a programmed amount. Cutter radius compensation is valid only in **LINEAR** and **CIRCLE** move modes. The moves must be specified by **F** (feedrate), not **TM** (move time). PMAC must be in move segmentation mode (I13 > 0) to do this compensation. (I13 > 0 is required for **CIRCLE** mode anyway.) Program commands **CC0**, **CC1**, **CC2**, **CCR** and **NORMAL** control this feature.

Synchronous M-Variable Assignment

The scan of a motion program and execution of the commands in it are governed by the lookahead feature. PMAC will calculate move commands ahead of time for a proper blending and will execute every instruction in between immediately. This ahead-of-time situation would make an M-Variable assignment asynchronous to the motion profiles unless a double equal sign is used instead. M1==1, for example, will indicate to PMAC that the assignment has to take place at the blending point between the previous move encountered and the next. In LINEAR and CIRCLE mode moves, this blending occurs V*TA/2 distance ahead of the specified intermediate point, where V is the commanded velocity of the axis, and TA is the acceleration (blending) time. This is available only for M-Variables and are not in the form TWB, TWD, TWR, TWS.

Synchronizing PMAC to Other PMACs

When multiple PMACs are used together, intercard synchronization is maintained by passing the servo clock signal from the first card to the others. With careful writing of programs, this permits complete coordination of axes on different cards.

Axis Transformation Matrices

PMAC provides the capability to perform matrix transformation operations on the X, Y, and Z-axes of a coordinate system. These operations have the same mathematical functionality as the matrix forms of the axis definition statements, but these can be changed on the fly in the middle of programs; the axis definition statements should be fixed for a particular application. The matrix transformations permit translation, rotation, scaling, mirroring, and skewing of the X, Y, and Z-axes. They can be useful for English/metric conversion, floating origins, making duplicate mirror images, repeating operations with angle offsets, and more. The matrices are implemented by the use of Q-Variables and **DEFINE TBUF**, **TSEL**, **TINIT**, **ADIS**, **IDIS**, **AROT** and **IROT** commands.

Position-Capture and Position-Compare Functions

The position-capture function latches the current encoder position at the time of an external event into a special register. It is executed totally in hardware, without the need for software intervention (although it is set up, and later serviced, in software). This means that the only delays in the capture are the hardware gate delays (negligible in any mechanical system), so this provides an incredibly accurate capture function. The move-until-trigger functions (either jog or motion program) conveniently use the position capture feature for continuous motions until a trigger condition is reached.

Essentially, the position-compare feature is the opposite of the position-capture function. Instead of storing the position of the counter when an external signal changes, it changes an external signal when the counter reaches a certain position.

Learning a Motion Program

It is possible to have PMAC learn lines of a motion program using the on-line **LEARN** command. In this operation, the axes are moved to the desired position and the command is given to PMAC. PMAC then adds a command line to the open motion program buffer that represents this position. This process can be repeated to learn a series of points.

The motors can be open loop or closed loop as they are moved around.

PLC PROGRAMS

PMAC will stop the scanning of the motion program lines when enough move commands have been calculated ahead of time. This feature is called look-ahead and it is necessary to properly blend the moves together and to observe the motion safety parameters. In the following example, PMAC calculates up to the third move and will stop the program scanning until the first move is completed; that is, when more move planning is required:

Example:

```
OPEN PROG 1 CLEAR              ; Open program buffer
I13=0                          ; Two moves ahead of calculation
LINEAR INC TA100 TS0 F50       ; Mode commands
X1                             ; First Move
X1                             ; Second Move
X1                             ; Third Move
M1=1                           ; This line will be executed only after the
                               ; first move is completed
CLOSE                          ; Close written buffer, program one
```

In contrast, enabled PLCs are continuously executed from beginning to end regardless of what any other PLC or Motion program is doing. PLCs are called asynchronous because they are designed for actions that are asynchronous to the motion.

Also, they are called PLC programs because they perform many of the same functions as hardware programmable logic controllers. PLC programs are numbered 0 through 31 for both the compiled and uncompiled PLCs. This means that there can be both a compiled PLC n and an uncompiled PLC n stored in PMAC. The faster execution of the compiled PLCs comes from two factors: first, from the elimination of interpretation time, and second, from the capability of the compiled PLC programs to execute integer arithmetic. However, the space dedicated to store up to 32 compiled PLC programs is limited to 15K (15,360) 24-bit words of PMAC memory; or 14K (14,336) words if there is a user-written servo as well. PLC programs 1-31 are executed in background. Each PLC program executes one scan (to the end or to an **ENDWHILE** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping, and respond to a host command, if any. In between each scan of each individual background interpreted PLC program, PMAC will execute one scan of all active background compiled PLCs. This means that the background compiled PLCs execute at a higher scan rate than the background interpreted PLCs. For example, if there are seven active background interpreted PLCs, each background compiled PLC will execute seven scans for each scan of a background interpreted PLC. At power-on\reset PLCC programs run after the first PLC program runs. These are the suggested uses of all the available PLC buffers:

- **PLC0:** PLC0 is a special fast program that operates at the end of the servo interrupt cycle with a frequency specified by variable I8 (every I8+1 servo cycles). This program is meant for a few time-critical tasks and it should be kept small, because its rapid repetition can steal time from other tasks. A PLC 0 that is too large can cause unpredictable behavior and can even trip PMAC's watchdog timer by starving background tasks of time to execute.
- **PLCC0:** The compiled PLCC0 should be used in the same instances as PLC0, taking advantage of the faster execution rate that a compiled PLC provides. Both PLC0 and PLCC0 can be defined at the same time.
- **PLC1:** This is the first code that PMAC will run on power-up, assuming that I5 was saved with a value of 2 or 3. This makes PLC1 the appropriate PLC to initialize parameters, perform commutated motors phase search and run motion programs. PLC1 can also disable other PLCs before they start running and can disable itself at the end of its execution.

- **PLC2:** Since PLC1 is suggested as an initialization PLC (and can run potentially only once on power-up), PLC2 is the first PLC in the remaining sequence from 2 to 31. This makes PLC2 the ideal place to copy digital input information from I\O expansion boards like the Acc-34 into its image variables. This way, PLCs 3 to 30 could use the input information, writing the necessary output changes to the outputs image variables.
- **PLC3 to PLC30:** PLC programs are useful particularly for monitoring analog and digital inputs, setting outputs, sending messages, monitoring motion parameters, issuing commands as if from a host, changing gains, and starting and stopping moves. By their complete access to PMAC variables and I/O and their asynchronous nature, they become very powerful adjuncts to the motion control programs.
- **PLCC3 to PLCC30:** Compiled PLCs are convenient for its faster execution compared to regular PLCs. Since the execution rate of compiled PLCs is the same as some of the safety checks (following error limits, hardware overtravel limits, software overtravel limits, and amplifier faults), PLCCs are ideal to replace or complement them. However, due to its limited allocated memory space, PLCCs should be reserved only for faster execution critical tasks.
- **PLC31:** This is the last executed PLC in the sequence from 1 to 31. PLC31 is recommended for copying the output image variable (changed in lower number PLCs executed previously) into the actual outputs of an I\O expansion board like, for example, the Acc-34A.

Entering a PLC Program

PLCs are programmed in the same way as motion programs are in a text editor window for later downloading to PMAC.

Before starting to write the PLC, make sure that memory has not been tied up in data gathering or program trace buffers, by issuing **DELETE GATHER** and **DELETE TRACE** commands.

1. Open the buffer for entry with the **OPEN PLC n** statement, where **n** is the buffer number. Next, if there is anything currently in the buffer that should not be kept, it should be emptied with the **CLEAR** statement (PLC buffers may not be edited on the PMAC itself; they must be cleared and re-entered). If the buffer is not cleared, new statements will be added onto the end of the buffer.
2. When finished, close the buffer with the **CLOSE** command. Opening a PLC program buffer automatically disables that program. After it is closed, it remains disabled, but it can be re-enabled again with the **ENABLE PLC n** command, where n is the buffer number (0--31). I5 must also be set properly for a PLC program to operate.
3. At closing, PMAC checks to make sure all **IF** branches and **WHILE** loops have been terminated properly. If not, it reports an error, and the buffer is inoperable. Then correct the PLC program in the host and re-enter it (clearing the erroneous block in the process, of course). This process is repeated for all of the PLC buffers to be used.

Because all PLC programs in PMAC's memory are enabled at power-on/reset, it is good practice to have I5 saved as 0 in PMAC's memory when developing PLC programs. This will allow PMAC to be reset and have no PLCs running (an enabled PLC only runs if I5 is set properly) and recover more easily from a PLC programming error.

Structure Example:

```
CLOSE
DELETE GATHER
DELETE TRACE
OPEN PLC n CLEAR
      {PLC statements}
CLOSE
ENABLE PLC n
```

To erase an uncompiled PLC program, open the buffer, clear the contents, then close the buffer again. This can be done with three commands on one line, as in:

```
OPEN PLC 5 CLEAR CLOSE
```

PLC Program Structure

The important thing to remember in writing a PLC program is that each PLC program is effectively in an infinite loop; it will execute over and over again until told to stop. (These are called PLC because of the similarity in how they operate to hardware Programmable Logic Controllers – the repeated scanning through a sequence of operations and potential operations.)

Calculation Statements

Much of the action taken by a PLC is done through variable value assignment statements: **{variable}={expression}**. The variables can be I, P, Q, or M-types and the action thus taken can affect many things inside and outside the card. Perhaps the simplest PLC program consists of one line:

```
P1=P1+1
```

Every time the PLC executes, usually hundreds of times per second, P1 will increment by one.

Of course, these statements can get a lot more involved. The statement:

```
P2=M162/(I108*32*10000)*COS (M262/(I208*32*100))
```

could be converting radial (M162) and angular (M262) positions into horizontal position data, scaling at the same time. Because it updates this frequently, whoever needs access to this information (e.g. host computer, operator, motion program) can be assured of having current data.

Conditional Statements

Most action in a PLC program is conditional, dependent on the state of PMAC variables, such as inputs, outputs, positions, counters, etc. Action can be level-triggered or edge-triggered; both can be done, but the techniques are different.

Level-Triggered Conditions

A branch controlled by a level- triggered condition is easier to implement. Taking our incrementing variable example and making the counting dependent on an input assigned to variable M11, we have:

```
IF (M11=1)
      P1=P1+1
ENDIF
```

As long as the input is true, P1 will increment several hundred times per second. When the input goes false, P1 will stop incrementing.

Edge-Triggered Conditions

To increment P1 once for each time M11 goes true (triggering on the rising edge of M11 sometimes called a one-shot or latched), a compound condition to trigger the action is needed. Then as part of the action, set one of the conditions false, so the action will not occur on the next PLC scan. The easiest way to do this is through the use of a shadow variable which will follow the input variable value. Action is taken only when the shadow variable does not match the input variable. Our code would become:

```
IF (M11=1)
      IF (P11=0)
            P1=P1+1
            P11=1
      ENDIF
ELSE
      P11=0
ENDIF
```

Make sure that P11 can follow M11 both up and down. Set P11 to 0 in a level-triggered mode.

WHILE Loops

Normally a PLC program executes all the way from beginning to end within a single scan. The exception to this rule occurs if the program encounters a true **WHILE** condition. In this case, the program will execute down to the **ENDWHILE** statement and exit this PLC. After cycling through all of the other PLCs, it will re-enter this PLC at the **WHILE** condition statement, not at the beginning. This process will repeat as long as the condition is true. When the **WHILE** condition goes false, the PLC program will skip past the **ENDWHILE** statement and proceed to execute the rest of the PLC program.

To increment the counter as long as the input is true and prevent execution of the rest of the PLC program, program:

```
WHILE (M11=1)
      P1=P1+1
ENDWHILE
```

This structure makes it easier to hold up PLC operation in one section of the program, so other branches in the same program do not have to have extra conditions so they do not execute when this condition is true. Contrast this to using an IF condition (see above).

COMMAND and SEND Statements

One of the most common uses of PLCs is to start motion programs and Jog motors by means of command statements.

Some **COMMAND** action statements should be followed by a **WHILE** condition to ensure they have taken effect before proceeding with the rest of the PLC program. This is true if a second **COMMAND** action statement that requires the first **COMMAND** action statement to finish will follow. (Remember, **COMMAND** action statements are processed only during the communications section of the background cycle.) For example, to stop any motion in a Coordinate System and start motion program 10, the following PLC can be used:

```
M187->Y:$0817,17,1            ; &1 In-position bit (AND of motors)
      OPEN PLC3 CLEAR
IF (M11=1)                    ; input is ON
      IF (P11=0)              ; input was not ON last time
            P11=1             ; set latch
            COMMAND"&1A"      ; ABORT all motion
            WHILE (M187=0)    ; wait for motion to stop.
            ENDW
            COMMAND"&1B10R"   ; start program 10
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
CLOSE
```

Any **SEND**, **COMMAND**, or **DISPLAY** action statement should be done only on an edge-triggered condition, because the PLC can cycle faster than these operations can process their information, and the communications channels can get overwhelmed if these statements get executed on consecutive scans through the PLC.

```
IF (M11=1)                    ; input is ON
      IF (P11=0)              ; input was not ON last time
            COMMAND"#1J+"     ; JOG motor
            P11=1             ; set latch
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
```

Timers

Timing commands like **DWELL** or **DELAY** are reserved only to motion programs and cannot be used for timing purposes on PLCs. Instead, PMAC has four 24-bit timers to write to and count down once per servo cycle. These timers are at registers X:$0700, Y:$0700, X:$0701, and Y:$0701. Usually a signed M-Variable is assigned to the timer; a value is written to it representing the desired time in servo cycles (multiply milliseconds by 8,388,608/I10); then the PLC waits until the M-Variable is less than 0.

Example:

```
M90->X:$0700,0,24,S           ; Timer register 1 (8388608/I10 msec)
M91->Y:$0700,0,24,S           ; Timer register 2 (8388608/I10 msec)
M92->X:$0701,0,24,S           ; Timer register 3 (8388608/I10 msec)
M93->Y:$0701,0,24,S           ; Timer register 4 (8388608/I10 msec)
OPEN PLC3 CLEAR
M1=0                          ; Reset Output1 before start
M90=1000*8388608/I10          ; Set timer to 1000 msec, 1 second
WHILE (M90>0)                 ; Loop until counts to zero
ENDWHILE
M1=1                          ; Set Output 1 after time elapsed
DIS PLC3                      ; disables PLC3 execution (needed in this example)
CLOSE
```

If more timers are needed, the best technique to use is in memory address X:0. This 24-bit register counts up once per servo cycle. Store a starting value for this, then with each scan subtract the starting value from the current value and compare the difference to the amount of time to wait.

Example:

```
M0->X:$0,24                   ; Servo counter register
M85->X:$07F0,24               ; Free 24-bit register
M86->X:$07F1,24               ; Free 24-bit register
OPEN PLC 3 CLEAR
M1=0                          ; Reset Output1 before start
M85=M0                        ; Initialize timer
M86=0
WHILE(M86<1000)               ; Time elapsed less than specified time?
    M86=M0-M85
    M86=M86*I10/8388608       ; Time elapsed so far in milliseconds
ENDWHILE
M1=1                          ; Set Output 1 after time elapsed
DISABLEPLC3                   ; disables PLC3 execution (needed in this example)
CLOSE
```

Even if the servo cycle counter rolls over (starts from zero again after the counter is saturated), by subtracting into another 24-bit register rollover is handled gracefully.

Rollover Example:

```
M0    =     1000
M85   =     16777000
M86   =     1216
```

Bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0
M85	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	0	0
M86	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

← Carry-out bit

Compiled PLC Programs

PLCCs are compiled by PEWIN in the downloading process. Only the compiled code gets downloaded to PMAC. Therefore, save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. Compiled PLCs are firmware dependent and so they must be recompiled when the firmware is changed in PMAC.

If more than one PLCC is programmed, all the PLCCs code must belong to the same ASCII text file. PEWIN will compile all the PLCC code present on the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, all the rest of the PLCCs that might have been present in memory will be erased, leaving only the last compiled code.

The multiple-file download feature of the PEWIN File menu allows the PLCC codes to be in different files. They are combined by PEWIN in the downloading process.

The use of L-Variables in a PLC program statement tells the compiler that the statement is to be executed using integer operations instead of floating-point operations.

To implement integer arithmetic in a compiled PLC, define any L-Variables to be used and substitute them in the programs for the variables that were used in the interpreted form (usually M-Variables). The compiler will interpret statements containing only L-Variables (properly defined) and integer constants as operations to be executed using integer arithmetic in compiled PLCs. Preparation of compiled PLCs is a multi-step process. The basic steps are as follows:

1. Write and debug the PLC programs in interpreted form (simple PLCs programs).
2. Change all references to PLCs to be compiled from PLC to PLCC.
3. For integer arithmetic, define L-Variables and substitute these for the old variable names in the programs.
4. Combine all of the PLC programs to be compiled into one file on the PC.
5. Make sure the Support MACROs/PLCCs option is checked before downloading.
6. Activate the compiled PLCs. If operation is not correct, return to step 1 or 2.
7. PLCCs can be deleted using the **DELETE PLCCn** command (replace n by the appropriate number).

TROUBLESHOOTING

PMAC is a highly reliable device and has several safety mechanisms to prevent continuous damage and malfunctions. When PMAC shuts down or an erratic behavior is observed, the following reset procedure should be used.

Resetting PMAC to Factory Defaults

1. If PMAC is communicating with the host computer, skip steps 2-7 on this list.
2. Turn off PMAC or the host computer where PMAC is installed.
3. Remove all cables connected to PMAC leaving connected only the serial port and power cables if present.
4. Using the appropriate hardware reference for the particular PMAC in question, check that all its jumpers are at the default configuration or changed properly to accommodate the particular setup for the machine. Make sure that jumper E50 is properly installed; otherwise any **SAVE** command issued to PMAC will not have any effect.
5. Place the jumper E51 in PMAC (1) or jumper E3 on PMAC2. This is a hardware re-initialization jumper.
6. After power-up, try establishing communications again with a reliable software package like the PEWIN program provided by Delta Tau.
7. On power-up, with the re-initialization jumper installed, some PMACs with the flash memory option will be in bootstrap mode. This means that PMAC will accept a binary file downloaded to change its internal firmware. If this is the case, follow the instructions on the PEWIN screen to disable the downloading process (usually pressing **CTRL+R**).
8. Try communications with PEWIN and type the following commands when the terminal is opened successfully (follow the communications troubleshooting section below in case communications are still not established):

```
$$$***                      ;Global Reset
P0..1023=0                  ;Reset P-variables values
Q0..1023=0                  ;Reset Q-variables values
M0..1023->* M0..1023=0      ;Reset M-variables definitions and values
UNDEFINE ALL                ;Undefine Coordinate Systems
SAVE                        ;Save this initial, clean configuration
```

9. If the re-initialization jumper was installed, remove it at this time. Restore PMAC in the computer and power it up.
10. Try communications again and configure PMAC for the application. Make sure there is a backup file saved in the host computer with all the parameters and programs that PMAC needs to run the application. Furthermore, since the host computer could also fail and be replaced, save the configuration file both in the host computer and in a floppy disk stored in a safe place. This file must be downloaded and a **SAVE** command must be issued to PMAC.

The Watchdog Timer (Red LED)

The PMAC motion control board has an on-board watchdog timer (sometimes called a dead-man timer or a get-lost timer) circuit whose job it is to detect a number of conditions that could result in dangerous malfunction, and shut down the card to prevent a malfunction. The philosophy behind the use of this circuit is that it is safer to have the system not operate at all than to have it operate improperly.

Because the watchdog timer wants to fail and many components of the board, both hardware and software, must be working properly to keep it from failing, it may not be immediately obvious what the cause of a watchdog timer failure is.

The hardware circuit for the watchdog timer requires that two basic conditions be met to keep it from tripping. First, it must see a DC voltage greater than approximately 4.75V. If the supply voltage is below this value, the circuit's relay will trip. This prevents corruption of registers due to insufficient voltage. The second necessary condition is that the timer must see a square wave input (provided by the PMAC software) of a frequency greater than approximately 25 Hz. If the card, for whatever reason, due either to hardware or software problems, cannot set and clear this bit repeatedly at this frequency or higher, the circuit's relay will trip.

Every RTI, PMAC reads the 12-bit watchdog timer register (Y register $1F) and decrements the value by 8 – this toggles bit 3. If the resulting value is not less than zero, it copies the result into a register that forces the bit 3 value onto the watchdog timer. Repeated, this process provides a square-wave input to the watchdog timer.

In the background, PMAC executes one scan through an individual PLC program, then checks to see if there are any complete commands, responding if there are, then executes the housekeeping functions. This cycle is repeated endlessly.

Most of the housekeeping functions are safety checks such as following error limits and overtravel limits. When it is done with these checks, PMAC sets the 12-bit watchdog timer register back to its maximum value. As long as this occurs regularly at least every 512 RTI cycles, the watchdog timer will not trip.

The purpose of this two-part control of the timer is to make sure all aspects of the PMAC software are being executed, both in foreground (interrupt-driven) and background. If anything keeps either type of routine from executing, the watchdog will fail quickly. The only recover for this failure, assuming the 5V power supply is satisfactory, is to hardware reset PMAC.

Establishing Communications

Either the Executive or Setup program can be used to establish initial communications with the card. Both programs have menus that tell the PC where to expect to find the PMAC and how to communicate with it at that location. If told to look for PMAC on the bus, also tell it PMAC's base address on the bus (this was set up with jumpers on PMAC). If told to look for PMAC on a COM port, tell it the baud rate (this was set up with jumpers or switches on the PMAC).

Once the program knows where and how to communicate with PMAC, it will attempt to find PMAC at that address by sending a query command and waiting for the response. If it gets the expected type of response, it will report that it has found PMAC and can proceed.

If it does not get the expected type of response after several attempts, it will report that it has not found PMAC. Check the following:

General

1. Is the green LED (power indicator) on PMAC's CPU board ON, as it should be? If it is not, find out why PMAC is not getting a +5V voltage supply.
2. Is the red LED (watchdog timer indicator) on PMAC's CPU board OFF, as it should be? If it is ON, make sure PMAC is getting very close to 5V supply – at less than 4.75V, the watchdog timer will trip, shutting down the card. The voltage can be probed at pins 1 and 3 of the J8 connector (A1 and A2 on the PMAC VME). If the voltage is satisfactory, inspect PMAC to see that all inter-board connections and all socketed ICs are well seated. If the card still will not run with the red LED off, contact the factory.

Bus Communications

1. Do the bus address jumpers (E91-E92, E66-E71) set an address that matches the bus address that the Executive program is trying to communicate with?
2. Is there something else on the bus at the same address? Try changing the bus address to see if communications can be established at a new address. Usually, Address 768 (300 hex) is open.

Serial Communications

1. Is the proper port on the PC being used? Make sure that if the Executive program is addressing the COM1 port, the COM1 connector has been cabled out.
2. Does the baud rate specified in the Executive program match the baud rate setting of the E44-E47 jumpers on PMAC?
3. With a breakout box or oscilloscope, make sure there is action on the transmit lines from the PC while typing into the Executive program. If not, there is a problem on the PC end.
4. Probe the return communication line while PMAC receives a command that requires a response (e.g. **<CONTROL-F>**). If there is no action, change jumpers E9-E16 on PMAC to exchange the send and receive lines. If there is action, but the host program does not receive characters, RS-232 could be receiving circuitry that does not respond at all to PMAC's RS-422 levels. If there is another model of PC, try using it as a test (most models accept RS-422 levels quite well). If the computer will not accept the signals, a level-conversion device, such as Acc-26 may be needed.

Motor Parameters

1. No movement at all. Check the following:
   a. Are both limits held low to AGND and sourcing current out of the pins?
   b. Is there proper supply to A+15V, A-15V, and AGND?
   c. Is the proportional gain (Ix30) greater than zero?
   d. Can any output be measured at the DAC pin when an O command has been given?
   e. Is the following error limit being tripped? Increase the fatal following error limit (Ix11) by setting it to a more appropriate value, and try to move again.
2. Movement, but sluggish. Check the following:
   a. Is proportional gain (Ix30) too low? Try increasing it (as long as stability is kept).
   b. Is the big step limit (Ix67) too low? Try increasing it to 8,000,000 -- near the maximum -- to eliminate any effect.
   c. Is the output limit (Ix69) too low? Try increasing it to 32,767 (the maximum) to make sure PMAC can output adequate voltage.
   d. Can an integrator help? Try increasing integral gain (Ix33) to 10,000 or more and the integration limit (Ix63) to 8,000,000.
3. Runaway condition. Check the following:
   a. Is there feedback? Check that the position changes can be read in both directions.
   b. Does the feedback polarity match output polarity? Recheck the polarity match as explained above.
4. Brief movement, and then stop. Check the following:
   a. Is the following error limit being tripped? Increase the fatal following error limit (Ix11) by setting it to a more appropriate value, and try to move again.

If holding position well, but cannot move the motor, probably the hardware limits are not being held low. Check which limits I125 is addressed to (usually +/-LIM1), then make sure those points are held low (to AGND), and sourcing current (unscrew the wire from the terminal block and put the ammeter in series with this circuit to confirm this). Refer to the section Installing and Configuring PMAC for details on checking the limit inputs.

If the motor dies after it has been given a **JOG** command, the fatal following error limit has been exceeded. If this has happened, it is either because a move has been requested that is more than the system can physically do (if so, reduce I122), or because it is very badly tuned (if this is the case, increase proportional gain I130). To restore closed-loop control, issue the **J/** command.

Motion Programs

If the program does not run at all, there are several possibilities:

1. Can the program be listed? In terminal mode, type **LIST PROG 1** (or whichever program), and see if it is there. If not, try to download it to the card again.
2. Is the program buffer closed? Type **A** just in case the program is running; type **CLOSE** to close any open buffer; type **B1** (or the program #) to point to the top of the program; and type **R** to try to run it again.
3. Can each motor in the coordinate system be jogged in both directions? If not, review that motor's setup.
4. Have any motors been assigned to the coordinate system that is not really set up yet? Every motor in the coordinate system must have its limits held low, even if there is no real motor attached.

Try the following steps for any other motion program problem:

1. Type **&1%100** in the terminal window.
2. Check that only one of the motors can be jogged that is to be used in the motion program.
3. Type the following commands in a text editor to be downloaded to PMAC:

```
close              ; Close any buffer opened
delete gather      ; Erase unwanted gathered data
undefine all       ; Erase coordinate definitions in all coordinate systems
#1->2000X          ; Replace #1 for the motor being used and 2000 by the
                   ; appropriate scale factor for the number of counts
                   ; per user units
OPEN PROG 1 CLEAR  ; Prepare buffer to be written
LINEAR             ; Linear interpolation
INC                ; Incremental mode
TA500              ; Acceleration time is 500 msec
TS0                ; No S-curve acceleration component
TM2000             ; Total move time is 500 + 2000=2500 msec
X1                 ; One unit of distance, 2000 encoder counts
CLOSE              ; Close written buffer, program one
```

4. To run it, press **CTRL+A** and then type **B1R** in the terminal window.
5. Repeat steps 2 through 4 for all the motors intended to run in the actual motion program.

A good method to test motion programs is to run them at lower than one hundred percent override rate. Any value for n from 1 to 100 in the **%n** online command will run the motion programs slower, increasing the chances of success of execution. For example, in the terminal window type: **&1 %75 B1R**

If a program runs successfully at lower feedrate override values, there can be mainly two reasons why it fails at 100%: either there is insufficient calculation time for the programmed moves or the acceleration and\or velocity parameters involved are unsuitable for the machine into consideration. Look for further details in the PMAC Tasks section.

PLC Programs

PLCs and PLCCs are one of the most common sources for communication or watchdog timer failures.

Any **SEND**, **COMMAND**, or **DISPLAY** action statement should be done only on an edge-triggered condition, because the PLC can cycle faster than these operations can process their information, and the communications channels can get overwhelmed if these statements are executed on consecutive scans through the PLC.

```
IF (M11=1)                    ; input is ON
      IF (P11=0)              ; input was not ON last time
            COMMAND"#1J+"     ; JOG motor
            P11=1             ; set latch
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
```

PLC0 or PLCC0 should be used for only a few tasks (usually a single task) that must be done at a higher frequency than the other PLC tasks. The PLC 0 will execute every real-time interrupt as long as the tasks from the previous RTI have been completed. PLC 0 is potentially the most dangerous task on PMAC as far as disturbing the scheduling of tasks is concerned. If it is too long, it will starve the background tasks for time. The first thing noticed is that communications and background PLC tasks will become sluggish. In the worst case, the watchdog timer will trip, shutting down the card, because the housekeeping task in background did not have the time to keep it updated.

Because all PLC programs in PMAC's memory are enabled at power-on/reset, it is good practice to have I5 saved as 0 in PMAC's memory when developing PLC programs. This will allow PMAC to be reset, no PLCs running (an enabled PLC only runs if I5 is set properly) and recover more easily from a PLC programming error.

As an example, type these commands in the terminal window. After that, open a watch window and monitor for P1 to be counting up:

```
OPEN PLC1 CLEAR         ; Prepare buffer to be written
P1=P1+1                 ; P1 continuously incrementing
CLOSE                   ; Close written buffer, PLC1
I5=2
Press <CTRL+D> and type ENA PLC1
```

APPENDIX A – PMAC ERROR CODE SUMMARY

I6, Error Reporting Mode:

This parameter controls how PMAC reports errors in command lines. When I6 is set to 0 or 2, PMAC reports an error with a **<BELL>** character only. When I6 is 0, the **<BELL>** character is given for invalid commands issued both from the host and from PMAC programs (using **CMD"{command}"**). When I6 is 2, the **<BELL>** character is given only for invalid commands from the host; there is no response to invalid commands issued from PMAC programs. In no mode is there a response to valid commands issued from PMAC programs.

When I6 is set to 1 or 3, an error number message can be reported along with the **<BELL>** character. The message comes in the form of **ERRnnn<CR>**, where **nnn** represents the three-digit error number. If I3 is set to 1 or 3, there is a **<LF>** character in front of the message.

When I6 is set to 1, the form of the error message is **<BELL>{error message}**. This setting is the best for interfacing with host-computer driver routines. When I6 is set to 3, the form of the error message is **<BELL><CR>{error message}**. This setting is appropriate for use with the PMAC Executive Program in terminal mode.

Currently, the following error messages can be reported:

Error	Problem	Solution
ERR001	Command not allowed during program execution	(should halt program execution before issuing command)
ERR002	Password error	(should enter the proper password)
ERR003	Data error or unrecognized command	(should correct syntax of command)
ERR004	Illegal character: bad value (>127 ASCII) or serial parity/framing error	(should correct the character and or check for noise on the serial cable)
ERR005	Command not allowed unless buffer is open	(should open a buffer first)
ERR006	No room in buffer for command	(should allow more room for buffer -- **DELETE** or **CLEAR** other buffers)
ERR007	Buffer already in use	(should **CLOSE** currently open buffer first)
ERR008	MACRO Link error	Register X:$0798 holds the error value
ERR009	Program structural error (e.g. **ENDIF** without **IF**)	(should correct structure of program)
ERR010	Both overtravel limits set for a motor in the C.S.	(should correct or disable limits)
ERR011	Previous move not completed	(should **Abort** it or allow it to complete)
ERR012	A motor in the coordinate system is open-loop	(should close the loop on the motor)
ERR013	A motor in the coordinate system is not activated	(should set Ix00 to 1 or remove motor from C.S.)
ERR014	No motors in the coordinate system	(should define at least one motor in C.S.)
ERR015	Not pointing to valid program buffer	(should use **B** command first, or clear out scrambled buffers)
ERR016	Running improperly structured program (e.g. missing **ENDWHILE**)	(should correct structure of program)
ERR017	Trying to resume after **/** or **** with motors out of stopped position	(should use **J=** to return motor[s] to stopped position)

APPENDIX B – PMAC I-VARIABLES SUMMARY

	Global I-Variables	**Range**	**Default**	**Units**
I1	Serial Handshake Line Disable	0 .. 3	0	None
I2	Control Panel Disable	0 .. 3	1	None
I3	I/O Handshake Mode	0 .. 3	1	None
I4	Communications Checksum Enable	0 .. 3	0	None
I5	PLC Programs On/Off	0 .. 3	0	None
I6	Error Reporting Mode	0 .. 3	3	None
I7	In-Position # of Consecutive Cycles	0 .. 255	0	Background computation cycles (minus one)
I8	Real Time Interrupt Period	0 .. 255	2	Servo Interrupt Cycles
I9	Full/Abbrev. Listing Form	0 .. 3	2	None
I10	Servo Interrupt Time	0 .. 8,388,607	3713707	1 / 8,388,608 msec
I11	Program Move Calc. Time	0 .. 8,388,607	0	Msec
I12	Jog-to-Pos. Calc. Time	1 .. 8,388,607	10	Msec
I13	Programmed Move Segmentation Time	0 .. 8,388,607	0	Msec
I14	Auto Position Match On Run Enable	0 .. 1	1	None
I15	Deg/Radians for User Trig	0 .. 1	0 (degrees)	None
I16	Rotary Buffer Request On Point	0 .. 8,388,607	5	Command lines.
I17	Rotary Buffer Request Off Point	0 .. 8,388,607	10	Program lines
I18	Fixed Buffer Full Warning Point	0 .. 8,388,607	10	Long Memory Words

	Data Gathering I-Variables	**Range**	**Default**	**Units**
I19	Data Gathering Period (In Servo Cycles)	0 .. 8,388,607	1	Servo Interrupt Cycles
I20	Data Gathering Selection Mask	$000000 .. $FFFFFF	$0	None
I21	Data Gathering Source 1 Address	$000000 .. $FFFFFF	$0	Modified PMAC addresses
I22-I44	Data Gathering Source 2 thru 24 Addresses	$000000 .. $FFFFFF	$0	Modified PMAC addresses
I45	Data Gathering Buffer Location And Mode	0 .. 3	0	None

	Other Global I-Variables	**Range**	**Default**	**Units**
I47	Address Of Pointer For Control-W Command	$0000 .. $FFFF (0 .. 65,535)	$0	Legal PMAC Y addresses
I48	DPRAM Servo Data Enable	0 .. 1	0	None
I49	DPRAM Background Data Enable	0 .. 1	0	None
I50	RAPID Mode Control	0 .. 1	1	None
I51	Leadscrew Compensation Enable	0 .. 1	0	None
I52	Feed Hold Slew Rate	0 .. 8,388,607	37137	I10 units / segmentation period
I53	Program Step Mode Control	0 .. 1	0	None
I55	DPR Background Data buffer enable	0 .. 1	0	None
I56	DPRAM Communications Interrupt Enable	0 .. 1	0	None
I57	DPRAM Binary Rotary Buffer Enable	0 .. 1	0	None
I58	DPRAM ASCII Communications Enable	0 .. 1	0	None
I59	DPRAM Buffer Max Motor/CS Number	0 .. 8	0	None
I60	Auto-Converted ADC Register Address	0, $FFD0 .. $FFFE	0	PMAC Y addresses
I61	Number of Auto-Converted ADC pair Registers	0 .. 7	0	Number of registers minus 1
I62	Internal Message Carriage Return Control	0 .. 1	0	None
I63	Control-X Echo Enable	0 .. 1	0	None
I64	Internal Response Tag Enable	0 .. 1	0	None
I8x	Motor x 3rd Resolver Gear Ratio	0 .. 4095	0	Second-resolver turns per third-resolver turn
I89	Cutter Comp Outside Corner Break Point	-1.0 -- 1.0	0.99848 (cos 1°)	cos Dq
I90	Minimum Arc Angle	Non-negative floating point	0 (sets 2-20)	Semi-circles (p radians; 180 degrees)
I9x	Motor x 2nd Resolver Gear Ratio	0 .. 4095	0	Primary-resolver turns per second-resolver turns
I99	Backlash Hysteresis	0 .. 8,388,607	64 (= 4 counts)	1/16 Count

	Motor Definition I-Variables	**Range**	**Default**	**Units**
Ix00	Motor x Activate	0 .. 1	1 (for motor #1)	None
Ix01	Motor x PMAC-Commutate Enable	0 .. 1	0	None
Ix02	Motor x DAC Address	PMAC addresses	See Ix02 table	Extended legal PMAC X and Y addresses
Ix03	Motor x Position Address	PMAC X addresses	See encoder table	Extended legal PMAC X addresses
Ix04	Motor x 'Velocity' Address	PMAC X addresses	Same as Ix03	Legal PMAC X addresses
Ix05	Motor x Master Position Address	PMAC X addresses	$073F	Legal PMAC X addresses
Ix06	Motor x Master Follow Enable	0 .. 1	0	None
Ix07	Motor x Master Scale Factor	-8,388,608 .. 8,388,607	96	None
Ix08	Motor x Position Scale Factor	0 .. 8,388,607	96	None
Ix09	Motor x Velocity Scale Factor	0 .. 8,388,607	96	None
Ix10	Motor x Power-on Servo Position Address	PMAC addresses	$0	Extended PMAC or multiplexer-port addresses

	Motor Safety I-Variables	**Range**	**Default**	**Units**
Ix11	Motor x Fatal Following Error Limit	0 .. 8,388,607	32000	1/16 Count
Ix12	Motor x Warning Following Error Limit	0 .. 8,388,607	16000	1/16 Count
Ix13	Motor x + Software Position Limit	± 247	0 (Disabled)	Encoder Counts
Ix14	Motor x - Software Position Limit	± 247	0 (Disabled)	Encoder Counts
Ix15	Motor x Abort/Lim Decel Rate	Positive floating point	0.25	Counts/msec2
Ix16	Motor x Maximum Velocity	Positive floating point	32	Counts/msec
Ix17	Motor x Maximum Acceleration	Positive floating point	0.015625	Counts/msec2
Ix19	Motor x Maximum Jog Acceleration	Positive floating point	0.015625	Counts/msec2
	Motor Movement I-Variables	**Range**	**Default**	**Units**
Ix20	Motor x Jog/Home Acceleration Time	0 .. 8,388,607	0 (so Ix21 controls)	Msec
Ix21	Motor x Jog/Home S-Curve Time	0 .. 8,388,607	50	Msec
Ix22	Motor x Jog Speed	Positive floating point	32	Counts / msec
Ix23	Motor x Homing Speed & Direction	Floating point	32	Counts / msec
Ix25	Motor x Flag Address	PMAC X addresses	See Ix25 table	Extended legal PMAC X addresses
Ix26	Motor x Home Offset	-8,388,608 .. 8,388,607	0	1/16 Count
Ix27	Motor x Position Rollover Range	0 .. 8,388,607	0	Counts
Ix28	Motor x In-Position Band	0 .. 8,388,607	160 (=10 counts)	1/16 Count
Ix29	Motor x DAC/1st Phase Bias	-32,768 .. 32,767	0	DAC Bits

	Servo Control I-Variables	**Range**	**Default**	**Units**
Ix30	Motor x Proportional Gain	-8,388,608 .. 8,388,607	2000	(Ix08/219) DAC bits/Encoder count
Ix31	Motor x Derivative Gain	-8,388,608 .. 8,388,607	1280	(Ix30*Ix09)/226 DAC bits/(Counts/cycle)
Ix32	Motor x Velocity Feed Forward Gain	0 .. 8,388,607	1280	(Ix30*Ix08)/226 DAC bits/(Counts/cycle)
Ix33	Motor x Integral Gain	0 .. 8,388,607	0	(Ix30*Ix08)/242 DAC bits/(counts*cycles)
Ix34	Motor x Integration Mode	0 .. 1	1	none
Ix35	Motor x Acceleration Feed Forward Gain	0 .. 8,388,607	0	(Ix30*Ix08)/226 DAC bits/(counts/cycle2)
Ix36	Motor x PID Notch Filter Coefficient N1	-2.0 .. +2.0	0	none (actual z-transform coefficient)
Ix37	Motor x PID Notch Filter Coefficient N2	-2.0 .. +2.0	0	none (actual z-transform coefficient)
Ix38	Motor x PID Notch Filter Coefficient D1	-2.0 .. +2.0	0	none (actual z-transform coefficient)
Ix39	Motor x PID Notch Filter Coefficient D2	-2.0 .. +2.0	0	none (actual z-transform coefficient)
Ix40-Ix56	Motor x Extended Servo Loop I-Variable			

	Motor Servo Loop Modifiers	**Range**	**Default**	**Units**
Ix57	Motor x Continuous Current Limit	0 .. 32,767	0	Bits of a 16-bit DAC
Ix58	Motor x Integrated Current Limit	0 .. 8,388,607	0	230 (DAC bits)2 * servo cycles
Ix59	Motor x User Written Servo Enable	0 .. 3	0	None
Ix60	Motor x Servo Cycle Period Extension	0 .. 255	0	Servo Interrupt Periods
Ix63	Motor x Integration Limit	-8,388,608 .. 8,388,607	4194304	1/16 count
Ix64	Motor x 'Deadband Gain'	-32,768 .. 32,767	0 (no deadband)	None
Ix65	Motor x Deadband Size	0 .. 32,767	16 (=1 count)	1/16 count
Ix67	Motor x Position Error Limit	0 .. 8,388,607	4,194,304	1/16 count
Ix68	Motor x Friction Feedforward	-32,768 .. 32,767	0	DAC bits
Ix69	Motor x DAC Limit	0 .. 32,767	20,480 (~6.25V)	DAC bits

	Commutation I-Variables	**Range**	**Default**	**Units**
Ix70	Motor x Number of Commutation Cycles	0 .. 255	1	Commutation cycles
Ix71	Motor x Counts/N Commutation Cycles	0 .. 8,388,607	1000	Counts
Ix72	Motor x Commutation Phase Angle	0 .. 255	85 (=120° e)	360/256 elec. deg. (1/256 commutation cycle)
Ix73	Motor x Phase Finding Value	0 .. 32,767	0	bits of 16-bit DAC
Ix74	Motor x Phase Finding Time	0 .. 255	0	Servo Interrupt Cycles (for Ix80 = 0 or 1) Servo Interrupt Cycles*256 (for Ix80 = 2 or 3)
Ix75	Motor x Power-On Phase Position Offset	-8,388,608 -- 8,388,607	0	Encoder counts * Ix70
Ix76	Motor x Velocity Phase Advance Gain	0 .. 8,388,607	0	Angle/Vel
Ix77	Motor x Magnetization Current	-32,768 .. 32,767	0	DAC bits
Ix78	Motor x Slip Gain	0 .. 8,388,607	0	238 (electrical cycles/update)/DAC bit
Ix79	Motor x 2nd Phase DAC Bias	-32,768 .. 32,767	0	DAC bits
Ix80	Motor x Power On Mode	0 .. 3	0	none
Ix81	Motor x Power-On Phase Position Address	PMAC addresses	$0	Extended PMAC or multiplexer-port addresses
Ix83	Motor x Ongoing Position Address	PMAC addresses	see Ix83 table	Legal PMAC 'X' and 'Y' addresses

	Further Motor I-Variables	**Range**	**Default**	**Units**
Ix85	Motor x Backlash Takeup Rate	0 .. 8,388,607	0	(1/16 Counts) / Background Cycle
Ix86	Motor x Backlash Size	0 .. 8,388,607	0	1/16 Count

	Coordinate System I-Variables	**Range**	**Default**	**Units**
Ix87	C.S. x Default Acceleration Time	0 .. 8,388,607	0 (so Ix88 controls)	Msec
Ix88	C.S. x Default S-Curve Time	0 .. 8,388,607	50	Msec
Ix89	C.S. x Default Feedrate	Positive floating point	1000	(User position units)/(feedrate time units)
Ix90	C.S. x Feedrate Time Units	Positive floating point	1000.0	Msec
Ix91	C.S. x Default Working Program Number	0 .. 32,767	0	Motion Program Numbers
Ix92	C.S. x Move Blend Disable	0 .. 1	0	None
Ix93	C.S. x Time Base Address	PMAC "X" addresses	See Ix93 table	Legal PMAC addresses
Ix94	C.S. x Time Base Slew Rate	0 .. 8,388,607	1644	2-23msec/ servo cycle
Ix95	C.S. x FeedHold Decel Rate	0 .. 8,388,607	1644	2-23msec/servo cycle
Ix96	C.S. x Circle Error Limit	Positive floating point	0 (function disabled)	User length units
Ix98	Coordinate System x Maximum Feedrate	Non-negative floating-point	0	None

	Encoder/Flag Setup I-Variables	**Range**	**Default**	**Units**
I900, I905,..	Encoder 0 Decode Control	0 .. 15	7	None
I901, I906,..	Encoder 0 Delay Filter Disable	0 .. 1	0	None
I902, I907,..	Encoder 0 Capture Control	0 .. 15	1	None
I903, I908,..	Encoder 0 Flag Select	0 .. 3	0	None

	MACRO Support I-Variables	**Range**	**Default**	**Units**
I1000	MACRO Node Auxiliary Register Enable	0 .. $FFFF (0 .. 65,535)	$0	None
I1001	MACRO Ring Check Period	0 .. 255	0	Servo cycles
I1003	MACRO Type 1 Master/Slave Comm. Timeout	0 .. 255	0	Servo cycles
I1004	MACRO Ring Error Shutdown Count		0	MACRO ring errors
I1005	MACRO Ring Sync Packet Shutdown Count	0 .. 65,535		MACRO sync packets

APPENDIX C – PMAC ON-LINE (IMMEDIATE) COMMANDS

On-Line Command	Function	Syntax	Syntax
<CONTROL-A>	Abort all programs and moves	ASCII Value 1D	$01
<CONTROL-B>	Report status word for all motors	ASCII Value 2D	$02
<CONTROL-C>	Report all coordinate system status words	ASCII Value 3D	$03
<CONTROL-D>	Disable all PLC programs	ASCII Value 4D	$04
<CONTROL-E>	Report configured address contents in binary (one-shot gathering)	ASCII Value 5D	$05
<CONTROL-F>	Report following errors for all motors	ASCII Value 6D	$06
<CONTROL-G>	Report global status word	ASCII Value 7D	$07
<CONTROL-H>	Erase last character	ASCII Value 8D	$08 (<BACKSPACE>)
<CONTROL-I>	Repeat last command line	ASCII Value 9D	$09 (<TAB>)
<CONTROL-K>	Kill all motors	ASCII Value 11D	$0B
<CONTROL-L>	Close open rotary buffer	ASCII Value 12D	$0C
<CONTROL-M>	Enter command line	ASCII Value 13D	$0D (<CR>)
<CONTROL-N>	Report command line checksum	ASCII Value 14D	$0E
<CONTROL-O>	Feed hold on all coordinate systems	ASCII Value 15D	$0F
<CONTROL-P>	Report positions of all motors	ASCII Value 16D	$10
<CONTROL-Q>	Quit all executing motion programs	ASCII Value 17D	$11
<CONTROL-R>	Begin execution of motion programs in all coordinate systems	ASCII Value 18D	$12
<CONTROL-S>	Step working motion programs in all coordinate systems	ASCII Value 19D	$13
<CONTROL-T>	Toggle serial port half/full duplex mode	ASCII Value 20D	$14
<CONTROL-U>	Open rotary program buffer(s)	ASCII Value 21D	$15
<CONTROL-V>	Report velocity of all motors	ASCII Value 22D	$16
<CONTROL-W>	Take command line from dual-ported RAM	ASCII Value 23D	$17
<CONTROL-X>	Cancel in-process communications	ASCII Value 24D	$18
<CONTROL-Y>	Report last command line	ASCII Value 25D	$19
<CONTROL-Z>	Set PMAC in serial port communications mode	ASCII Value 26D	$1A
#	Report currently addressed motor	#	
#{constant}	Address a motor	#{constant}	
#{constant}->	Report the specified motor's coordinate system axis definition	#{constant}->	
#{constant}->0	Clear axis definition for specified motor	#{constant}->0	
#{constant}->{axis definition}	Assign an axis definition for the specified motor	#{constant}->{axis definition}	
$	Reset motor	$	
$$$	Full card reset	$$$	
$$$***	Global card reset and re-initialization	$$$***	
%	Report the addressed coordinate system's feedrate override value	%	
%{constant}	Set the addressed coordinate system's feedrate override value	%{constant}	
&{constant}	Address a coordinate system	&{constant}	
&	Report currently addressed coordinate system	&	

/	Halt program execution at end of currently executing move	/	
?	Report motor status	?	
??	Report the status words of the addressed coordinate system	??	
???	Report global status words	???	
@	Report currently addressed card on serial daisychain	@	
@{card}	Address a card on the serial daisychain	@{card}	
\	Do a program hold (permitting jogging while in hold mode)	\	
A	Abort all programs and moves in the currently addressed coordinate system	A	
ABS	Select absolute position mode for axes in addressed coordinate system	ABS	ABS({axis}[,{axis}...])
{axis}={constant}	Re-define the specified axis position	{axis}={constant}	
B{constant}	Point the addressed coordinate system to a motion program	B{constant}	
CLEAR	Erase currently opened buffer	CLEAR	CLR
CLOSE	Close the currently opened buffer	CLOSE	CLS
{constant}	Assign value to variable P0 or to table entry	{constant}	
DATE	Report PROM firmware revision date	DATE	DAT
DEFINE BLCOMP	Define backlash compensation table	DEFINE BLCOMP {entries},{count length}	DEF BLCOMP {entries},{count length}
DEFINE COMP (one-dimensional)	Define leadscrew compensation table	DEFINE COMP {entries},[#{source},[#{target},]]{count length}	
DEFINE COMP (two-dimensional)	Define two-dimensional leadscrew compensation table	DEFINE COMP {entr1}.{entr2},#{src1},[#{src2},[#{trgt}]],{lgt1},{lgt2}	DEF COMP …
DEFINE GATHER	Create a data gathering buffer	DEFINE GATHER [{constant}]	DEF GAT [{constant}]
DEFINE ROTARY	Define a rotary motion program buffer	DEFINE ROTARY{constant}	DEF ROT{constant}
DEFINE TBUF	Create a buffer for axis transformation matrices	DEFINE TBUF {constant}	DEF TBUF {constant}
DEFINE TCOMP	Define torque compensation table	DEFINE TCOMP {entries},{count length}	DEF TCOMP {entries},{count length}
DEFINE UBUFFER	Create a buffer for user variable use	DEFINE UBUFFER {constant}	DEF UBUF {constant}
DELETE BLCOMP	Erase backlash compensation table	DELETE BLCOMP	DEL BLCOMP
DELETE COMP	Erase leadscrew compensation table	DELETE COMP	DEL COMP
DELETE GATHER	Erase the data gather buffer	DELETE GATHER	DEL GAT
DELETE PLCC	Erase specified compiled PLC program	DELETE PLCC {constant}	DEL PLCC {constant}
DELETE ROTARY	Delete rotary motion program buffer of addressed coordinate system	DELETE ROTARY	DEL ROT
DELETE TBUF	Delete buffer for axis transformation matrices	DELETE TBUF	DEL TBUF

DELETE TCOMP	Erase torque compensation table	DELETE TCOMP	DEL TCOMP
DELETE TRACE	Formerly: Erase the motion program trace buffer	DELETE TRACE	DEL TRAC
DISABLE PLC	Disable specified PLC program(s)	DISABLE PLC {constant}[,{constant}] DISABLE PLC {constant}..{constant}	DIS PLC {constant}[,{constant}] DIS PLC {constant}..{constant}
DISABLE PLCC	Disable compiled PLCC program(s)	DISABLE PLCC {constant}[,{constant}] DISABLE PLCC {constant}..{constant}	DIS PLCC {constant}[,{constant}] DIS PLCC {constant}..{constant}
ENABLE PLC	Enable specified PLC program(s)	ENABLE PLC {constant}[,{constant}] ENABLE PLC {constant}..{constant}	ENA PLC {constant}[,{constant}] ENA PLC {constant}..{constant}
ENABLE PLCC	Enable specified PLCC program(s)	ENABLE PLCC {constant}[,{constant}] ENABLE PLCC {constant}..{constant}	ENA PLCC {constant}[,{constant}] ENA PLCC {constant}..{constant}
ENDGATHER	Stop data gathering	ENDGATHER	ENDG
F	Report motor following error	F	
FRAX	Specify the coordinate system's feedrate axes	FRAX FRAX({axis}[,{axis}...])	
GATHER	Begin data gathering	GATHER [TRIGGER]	GAT [TRIG]
H	Perform a feedhold	H	
HOME	Start Homing Search Move	HOME	HM
HOMEZ	Do a Zero-Move Homing	HOMEZ	HMZ
I{constant}	Report the current I-variable value(s)	I{constant}[..{constant}]	
I{constant}={expression}	Assign a value to an I-variable	I{constant}[..{constant}]={expression}	
I{constant}=*	Assign factory default value to an I-variable	I{constant}[..{constant}]=*	
INC	Specify incremental move mode	INC INC({axis}[,{axis}...])	
J!	Adjust motor commanded position to nearest integer count	J!	
J+	Jog positive	J+	
J-	Jog negative	J-	
J/	Jog stop	J/	
J:{constant}	Jog relative to commanded position	J:{constant}	
J:*	Jog to specified variable distance from present commanded position	J:*	
J=	Jog to pre-jog position	J=	

J={constant}	Jog to specified position	J={constant}	
J=*	Jog to specified variable position	J=*	
J=={constant}	Jog to specified motor position and make that position the pre-jog position	J=={constant}	
J^{constant}	Jog relative to actual position	J^{constant}	
J^*	Jog to specified variable distance from present actual position	J^*	

{jog command}^{constant}	Jog until trigger	J=^{constant} J={constant}^{constant} J:{constant}^{constant} J^{constant}^{constant} J=*^{constant} J:*^{constant} J^*^{constant}	
K	Kill motor output	K	
LEARN	Learn present commanded position	LEARN[({axis}[,{axis}...]]	LRN[({axis}[,{axis}...]]
LIST	List the contents of the currently opened buffer	LIST	
LIST COMP	List contents of addressed motor's compensation table	LIST COMP	
LIST COMP DEF	List definition of addressed motor's compensation table	LIST COMP DEF	
LIST GATHER	Report contents of the data gathering buffer	LIST GATHER [{start}] [,{length}]	LIS GAT [{start}] [,{length}]
LIST LDS	List linking addresses of ladder functions	LIST LDS	
LIST LINK	List linking addresses of internal PMAC routines	LIST LINK	
LIST PC	List program at program counter	LIST PC[,[{constant}]]	
LIST PE	List program at program execution	LIST PE[,[{constant}]]	
LIST PLC	List the contents of the specified PLC program	LIST PLC {constant}	
LIST PROGRAM	List the contents of the specified motion program	LIST PROGRAM {constant} [{start}] [,{length}]	LIST PROG{constant} [{start}] [,{length}]
M{constant}	Report the current M-Variable value(s)	M{constant}[..{constant}]	
M{constant}={expression}	Assign value to M-Variable(s)	M{constant}[..{constant}]={expression}	
M{constant}->	Report current M-Variable definition(s)	M{constant}[..{constant}]->	
M{constant}->*	Self-referenced M-Variable definition	M{constant}[..{constant}]->*	
M{constant}->D:{address}	Long fixed-point M-Variable definition	M{constant}[..{constant}]->D[:]{address}	
M{constant}->DP:{address}	Dual-ported RAM fixed-point M-Variable definition	M{constant}[..{constant}]->DP[:]{address}	
M{constant}->F:{address}	Dual-Ported RAM Floating-Point M-Variable definition	M{constant}[..{constant}]->F[:]{address}	
M{constant}->L:{address}	Long word floating-point M-Variable definition	M{constant}[..{constant}]->L[:]{address}	
M{constant}->TWB:{address}	Binary thumbwheel-multiplexer definition	M{constant}[..{constant}]->TWB[:]{muxaddr},{offset},{size},{format}	

M{constant}->TWD:{address}	BCD thumbwheel-multiplexer M-Variable definition	M{constant}[..{constant}]->TWD[:]{muxaddr},{offset},{size}[.{dp}],{format}	
M{constant}->TWR:{addr},{off.}	Resolver thumbwheel-multiplexer M-Variable definition	M{constant}[..{constant}]->TWR[:]{muxaddr},{offset}	
M{constant}->TWS:{address}	Serial thumbwheel-multiplexer M-Variable definition	M{constant}[..{constant}]->TWS[:]{muxaddr}	
M{constant}->X/Y:{address}	Short word M-Variable definition	M{constant}[..{constant}]->X[:]{address},{offset}[,{width}[,{format}]] M{constant}[..{constant}]->Y[:]{address},{offset}[,{width}[,{format}]]	
MACROAUX	Report or write MACRO auxiliary parameter value	MACROAUX {NodeNum}{ParamNum}[={constant}]	MX{NodeNum}{ParamNum}[={constant}]
MACROAUXREAD	Read MACRO auxiliary parameter value	MACROAUXREAD{NodeNum}{ParamNum}{Variable}	MXR{NodeNum}{ParamNum}{Variable}
MACROAUXWRITE	Write MACRO auxiliary parameter value	MACROAUXWRITE {NodeNum}{ParamNum}{Variable}	MXW{NodeNum}{ParamNum}{Variable}
MACROSLV{command} {node#}	Send command to Type 1 MACRO slave	MACROSLAVE {command}{node #}	MS{command}{node #}
MACROSLV{node#},{slave variable}	Report Type 1 MACRO auxiliary parameter value	MACROSLAVE {node #},{slave variable}	MS{node #},{slave variable}
MACROSLV{node#},{slave var}={const}	Set Type 1 MACRO auxiliary parameter value	MACROSLAVE {node #},{slave variable}={constant}	MS{node #},{slave variable}={constant}
MACROSLVREAD	Read (copy) Type 1 MACRO auxiliary parameter value	MACROSLVREAD {node #},{slave variable},{PMAC variable}	MSR{node #},{slave variable},{PMAC variable}
MACROSLVWRITE	Write (copy) Type 1 MACRO auxiliary parameter value	MACROSLVWRITE {node #},{slave variable},{PMAC variable}	MSW{node #},{slave variable},{PMAC variable}
MFLUSH	Clear pending synchronous M-Variable assignments	MFLUSH	
O{constant}	Open loop output	O{constant}	
OPEN PLC	Open a PLC program buffer for entry	OPEN PLC {constant}	
OPEN PROGRAM	Open a fixed motion program buffer for entry	OPEN PROGRAM {constant}	OPEN PROG {constant}
OPEN ROTARY	Open all existing rotary motion program buffers for entry	OPEN ROTARY	OPEN ROT
P	Report motor position	P	
P{constant}	Report the current P-Variable value(s)	P{constant}[..{constant}]	
P{constant}={expression}	Assign a value to a P-Variable	P{constant}[..{constant}]={expression}	
PAUSE PLC	Pause specified PLC program(s)	PAUSE PLC {constant}[,{constant}...]	PAU PLC {constant}[,{constant}...]

PASSWORD={string}	Enter/set program password	PASSWORD={string}	
PC	Report program counter	PC	
PE	Report program execution pointer	PE	
PMATCH	Re-match axis positions to motor positions	PMATCH	
PR	Report rotary program remaining	PR	
Q	Quit program at end of move	Q	
Q{constant}	Report Q-Variable value	Q{constant}[..{constant}]	
Q{constant}={expression}	Q-Variable value assignment	Q{constant}[..{constant}] ={expression}	
R	Run motion program	R	
R[H]{address}	Report the contents of a specified memory addresses	R[H]{address} [,{constant}]	
RESUME PLC	Resume execution of specified PLC programs	RESUME PLC {constant}[,{constant}...]	RES PLC {constant}[,{constant}...]
S	Execute one move step of motion program	S	
SAVE	Copy setup parameters to non-volatile memory	SAVE	
SIZE	Report the amount of unused buffer memory in PMAC	SIZE	
TYPE	Report type of PMAC	TYPE	
UNDEFINE	Erase coordinate system definition	UNDEFINE	UNDEF
UNDEFINE ALL	Erase coordinate definitions in all coordinate systems	UNDEFINE ALL	UNDEF ALL
V	Report motor velocity	V	
VERSION	Report PROM firmware version number	VERSION	VER
W{address}	Write value(s) to a specified addresses	W{address},{value} [,{value}...]	
Z	Make commanded axis positions zero	Z	

APPENDIX D – PMAC PROGRAM COMMAND SPECIFICATIONS

Function	Syntax	Syntax	Type
Position-Only Move Specification	{axis}{data}[{axis}{data}...]		PROG / ROT
Position and Velocity Move Specification	{axis}{data}:{data} [{axis}{data}:{data}...]		PROG / ROT
Move Until Trigger	{axis}{data}^{data}[{axis}{ data}^{data}...]		Motion Program
Circular Arc Move Specification	{axis}{data} [{axis}{data}...] {vector}{data}		PROG / ROT
A-Axis Move	A{data}		PROG / ROT
Absolute Move Mode	ABS [({axis}[,{axis}...])]		PROG / ROT
Motor/Coordinate System Modal Addressing	ADDRESS [#{constant}][&{constant}]	ADR [#{constant}] [&{constant}]	PLC 1 to 31 only
Absolute displacement of X, Y, and Z axes	ADIS{constant}		PROG / ROT
Conditional AND	AND ({condition})		PLC program only
Absolute rotation/scaling of X, Y, and Z axes	AROT{constant}		PROG / ROT
B-Axis Move	B{data}		PROG / ROT
Mark Start of Stepping Block	BLOCKSTART	BSTART	PROG / ROT
Mark End of Stepping Block	BLOCKSTOP	BSTOP	PROG / ROT
C-Axis Move	C{data}		PROG / ROT
Jump to Subprogram With Return	CALL{data} [{letter}{data}...]		PROG / ROT
Turn Off Cutter Radius Compensation	CC0		PROG / ROT
Turn On Cutter Radius Compensation Left	CC1		PROG / ROT
Turn On Cutter Radius Compensation Right	CC2		PROG / ROT
Set Cutter Compensation Radius	CCR{data}		PROG / ROT
Set Blended Clockwise Circular Move Mode	CIRCLE1	CIR1	PROG / ROT
Set Blended Counterclockwise Circular Move Mode	CIRCLE2	CIR2	PROG / ROT
Program Command Issuance	COMMAND "{command}"	CMD "{command}"	PROG / ROT / PLC
Program Control-Character Command Issuance	COMMAND^{letter}	CMD^{letter}	PROG / ROT / PLC
Tool Data (D-Code)	D{data}		PROG / ROT
Delay for Specified Time	DELAY{data}	DLY{data}	PROG / ROT
Disable PLC Program(s)	DISABLE PLC {constant}[,{constant}] DISABLE PLC {constant}..{constant}	DIS PLC {constant}[,{constant}] DIS PLC {constant}..{constant}	PROG / ROT / PLC

Disable Compiled PLC Program(s)	DISABLE PLCC {constant}[,{constant}] DISABLE PLCC {constant}..{constant}	DIS PLCC {constant}[,{constant}] DIS PLCC {constant}..{constant}	PROG / ROT / PLC EXCEPT PLC0, PLCC0
Display Text to Display Port	DISPLAY [{constant}] "{message}"	DISP [{constant}] "{message}"	PROG / ROT / PLC
Formatted Display of Variable Value	DISPLAY {constant}, {constant}.{constant}, {variable}	DISP {constant}, {constant}.{constant}, {variable}	PROG / ROT / PLC
Dwell for Specified Time	DWELL{data}	DWE{data}	PROG / ROT
Start False Condition Branch	ELSE		Motion or PLC
Start False Condition Branch	ELSE {action}		Motion Program
Enable PLC Buffer(s)	ENABLE PLC {constant}[,{constant}] ENABLE PLC {constant}..{constant}	ENA PLC {constant}[,{constant}] ENA PLC {constant}..{constant}	PROG / ROT / PLC
Enable Compiled PLC Program(s)	ENABLE PLCC {constant}[,{constant}] ENABLE PLCC {constant}..{constant}	ENA PLCC {constant}[,{constant}] ENA PLCC {constant}..{constant}	PROG / ROT / PLC
Mark End of Conditional Block	ENDIF	ENDI	Motion or PLC
Mark End of Conditional Loop	ENDWHILE	ENDW	Motion or PLC
Set Move Feedrate (Velocity)	F{data}		PROG / ROT
Specify Feedrate Axes	FRAX [({axis}[,{axis}...])]		PROG / ROT
Preparatory Code (G-Code)	G{data}		PROG / ROT
Unconditional Jump With Return	GOSUB{data}		Motion Program
Unconditional Jump Without Return	GOTO{data}		Motion Program
Programmed Homing	HOME {constant}[,{constant}...] HOME {constant}..{constant}[,{constant}..{constant}...]	HM {constant}[,{constant}...] HM {constant}..{constant}[,{constant}..{constant}...]	PROG / ROT
Programmed Zero-Move Homing	HOMEZ {constant}[,{constant}...] HOMEZ {constant}..{constant}[,{constant}..{constant}...]	HMZ {constant}[,{constant}...] HMZ {constant}..{constant}[,{constant}..{constant}...]	PROG / ROT
I-Vector Specification for Circular Moves or Normal Vectors	I{data}		PROG / ROT
Set I-Variable Value	I{constant}={expression}		PROG / ROT / PLC
Incremental displacement of X, Y, and Z axes	IDIS{constant}		PROG / ROT
Conditional branch	IF ({condition})		Motion or PLC
Conditional branch	IF ({condition}) {action}[{action}...]		PROG / ROT

Incremental Move Mode	INC [({axis}[,{axis}...])]		PROG / ROT
Incremental rotation/scaling of X, Y, and Z axes	IROT{constant}		PROG / ROT
J-Vector Specification for Circular Moves	J{data}		PROG / ROT
K-Vector Specification for Circular Moves	K{data}		PROG / ROT
Blended Linear Interpolation Move Mode	LINEAR	LIN	PROG / ROT
Set M-Variable Value	M{constant}={expression}		PROG / ROT
Synchronous M-Variable Value Assignment	M{constant}=={expression}		Motion Program
M-Variable 'And-Equals' Assignment	M{constant}&={expression}		PROG / ROT
M Variable 'Or-Equals' Assignment	M{constant}|={expression}		PROG / ROT
M-Variable 'XOR-Equals' Assignment	M{data}^={expression}		PROG / ROT
Machine Code (M-Code)	M{data}		PROG / ROT
Read MACRO auxiliary parameter value	MACROAUXREAD{NodeNum}{ParamNum}{Variable}	MXR{NodeNum}{ParamNum}{Variable}	background PLC only
Write MACRO auxiliary parameter value	MACROAUXWRITE{NodeNum}{ParamNum}{Variable}	MXW{NodeNum}{ParamNum}{Variable}	background PLC only
Read (copy) Type 1 MACRO auxiliary parameter value	MACROSLVREAD{node #},{slave variable},{PMAC variable}	MSR{node #},{slave variable},{PMAC variable}	PLC 1 to 31 only
Write (copy) Type 1 MACRO auxiliary parameter value	MACROSLVWRITE{node #},{slave variable},{PMAC variable}	MSW{node #},{slave variable},{PMAC variable}	PLC 1 to 31 only
Program Line Label	N{constant}		PROG / ROT
Define Normal Vector to Plane of Circular Interpolation and Cutter Radius Compensation	NORMAL {vector}{data} [{vector}{data}...]	NRM {vector}{data} [{vector}{data}...]	PROG / ROT
Alternate Line Label	O{constant}		PROG / ROT
Conditional OR	OR ({condition})		PLC program only
Set P-Variable Value	P{constant}={expression}		PROG / ROT
Pause execution of PLC program(s)	PAUSE PLC {constant}[,{constant}...] PAUSE PLC {constant}[..{constant}]	PAU PLC {constant} [,{constant}...] PAU PLC {constant} [..{constant}]	PROG / ROT / PLC
Specify automatic subroutine call function	PRELUDE1{command} PRELUDE0		Motion Program
Redefine current axis positions (Position SET)	PSET{axis}{data} [{axis}{data}...]		Motion Program
Set Position-Velocity-Time mode	PVT{data}		PROG / ROT
Set Q-Variable Value	Q{constant}={expression}		PROG / ROT / PLC
Set Circle Radius	R{data}		PROG / ROT
Set Rapid Traverse Mode	RAPID	RPD	PROG / ROT
Read Arguments for Subroutine	READ({letter},[{letter}...])		Motion Program
Resume execution of PLC programs(s)	RESUME PLC {constant}[,{constant}...] RESUME PLC{constant}[..{constant}]	RES PLC {constant} [,{constant}...] RES PLC {constant} [..{constant}]	PROG / ROT / PLC

Return From Subroutine Jump/End Main Program	RETURN	RET	Motion Program
Spindle data command	S{data}		PROG / ROT
Cause PMAC to Send Message	SEND"{message}" SENDS"{message}" SENDP"{message}"		PROG / ROT / PLC
Cause PMAC to Send Control Character	SEND^{letter} SENDS^{letter} SENDP^{letter}		PROG / ROT / PLC
Put program in uniform cubic spline motion mode	SPLINE1		PROG / ROT
Put program in non-uniform cubic spline motion mode	SPLINE2		PROG / ROT
Stop program execution	STOP		Motion Program
Tool Select Code (T-Code)	T{data}		PROG / ROT
Set Acceleration Time	TA{data}		PROG / ROT
Initialize selected transformation matrix	TINIT		PROG / ROT
Set Move Time	TM{data}		PROG / ROT
Set S-Curve Acceleration Time	TS{data}		PROG / ROT
Select active transformation matrix for X, Y, and Z axes	TSELECT{constant}		PROG / ROT
U-Axis Move	U{data}		PROG / ROT
V-Axis Move	V{data}		PROG / ROT
W-Axis Move	W{data}		PROG / ROT
Suspend program execution	WAIT		PROG / ROT
Conditional looping	WHILE ({condition})		Motion or PLC
Conditional looping	WHILE ({condition}) {action}		PROG / ROT
X-Axis Move	X{data}		PROG / ROT
Y-Axis Move	Y{data}		PROG / ROT
Z-Axis Move	Z{data}		PROG / ROT

APPENDIX E – MOTOR SUGGESTED M-VARIABLE DEFINITIONS

Registers Associated with Encoder/DAC	Motor #1	Motor #2	Motor #3	Motor #4	Motor #5	Motor #6	Motor #7	Motor #8
ENC 24-bit counter position	M101->X:$C001,0,24,S	M201->X:$C005,0,24,S	M301->X:$C009,0,24,S	M401->X:$C00D,0,24,S	M501->X:$C011,0,24,S	M601->X:$C015,0,24,S	M701->X:$C019,0,24,S	M801->X:$C01D,0,24,S
DAC 16-bit analog output	M102->Y:$C003,8,16,S	M202->Y:$C002,8,16,S	M302->Y:$C00B,8,16,S	M402->Y:$C00A,8,16,S	M502->Y:$C013,8,16,S	M602->Y:$C012,8,16,S	M702->Y:$C01B,8,16,S	M802->Y:$C01A,8,16,S
ENC capture/compare position register	M103->X:$C003,0,24,S	M203->X:$C007,0,24,S	M303->X:$C00B,0,24,S	M403->X:$C00F,0,24,S	M503->X:$C013,0,24,S	M603->X:$C017,0,24,S	M703->X:$C01B,0,24,S	M803->X:$C01F,0,24,S
ENC interpolated position (1/32 ct)	M104->X:$0720,0,24,S	M204->X:$0721,0,24,S	M304->X:$0722,0,24,S	M404->X:$0723,0,24,S	M504->X:$0724,0,24,S	M604->X:$0725,0,24,S	M704->X:$0726,0,24,S	M804->X:$0727,0,24,S
ADC 16-bit analog input	M105->Y:$C006,8,16,S	M205->Y:$C007,8,16,S	M305->Y:$C00E,8,16,S	M405->Y:$C00F,8,16,S	M505->Y:$C016,8,16,S	M605->Y:$C017,8,16,S	M705->Y:$C01E,8,16,S	M805->Y:$C01F,8,16,S
EQU compare flag latch control	M111->X:$C000,11,1	M211->X:$C004,11,1	M311->X:$C008,11,1	M411->X:$C00C,11,1	M511->X:$C010,11,1	M611->X:$C014,11,1	M711->X:$C018,11,1	M811->X:$C01C,11,1
EQU compare output enable	M112->X:$C000,12,1	M212->X:$C004,12,1	M312->X:$C008,12,1	M412->X:$C00C,12,1	M512->X:$C010,12,1	M612->X:$C014,12,1	M712->X:$C018,12,1	M812->X:$C01C,12,1
EQU compare invert enable	M113->X:$C000,13,1	M213->X:$C004,13,1	M313->X:$C008,13,1	M413->X:$C00C,13,1	M513->X:$C010,13,1	M613->X:$C014,13,1	M713->X:$C018,13,1	M813->X:$C01C,13,1
AENA/DIR Output	M114->X:$C000,14,1	M214->X:$C004,14,1	M314->X:$C008,14,1	M414->X:$C00C,14,1	M514->X:$C010,14,1	M614->X:$C014,14,1	M714->X:$C018,14,1	M814->X:$C01C,14,1
EQU compare flag	M116->X:$C000,16,1	M216->X:$C004,16,1	M316->X:$C008,16,1	M416->X:$C00C,16,1	M516->X:$C010,16,1	M616->X:$C014,16,1	M716->X:$C018,16,1	M816->X:$C01C,16,1
ENC position-captured flag	M117->X:$C000,17,1	M217->X:$C004,17,1	M317->X:$C008,17,1	M417->X:$C00C,17,1	M517->X:$C010,17,1	M617->X:$C014,17,1	M717->X:$C018,17,1	M817->X:$C01C,17,1
ENC Count-error flag	M118->X:$C000,18,1	M218->X:$C004,18,1	M318->X:$C008,18,1	M418->X:$C00C,18,1	M518->X:$C010,18,1	M618->X:$C014,18,1	M718->X:$C018,18,1	M818->X:$C01C,18,1
ENC 3rd channel input status	M119->X:$C000,19,1	M219->X:$C004,19,1	M319->X:$C008,19,1	M419->X:$C00C,19,1	M519->X:$C010,19,1	M619->X:$C014,19,1	M719->X:$C018,19,1	M819->X:$C01C,19,1
HMFL input status	M120->X:$C000,20,1	M220->X:$C004,20,1	M320->X:$C008,20,1	M420->X:$C00C,20,1	M520->X:$C010,20,1	M620->X:$C014,20,1	M720->X:$C018,20,1	M820->X:$C01C,20,1
-LIM input status	M121->X:$C000,21,1	M221->X:$C004,21,1	M321->X:$C008,21,1	M421->X:$C00C,21,1	M521->X:$C010,21,1	M621->X:$C014,21,1	M721->X:$C018,21,1	M821->X:$C01C,21,1
+LIM input status	M122->X:$C000,22,1	M222->X:$C004,22,1	M322->X:$C008,22,1	M422->X:$C00C,22,1	M522->X:$C010,22,1	M622->X:$C014,22,1	M722->X:$C018,22,1	M822->X:$C01C,22,1
FAULT input status	M123->X:$C000,23,1	M223->X:$C004,23,1	M323->X:$C008,23,1	M423->X:$C00C,23,1	M523->X:$C010,23,1	M623->X:$C014,23,1	M723->X:$C018,23,1	M823->X:$C01C,23,1

Motor Status Bits	Motor #1	Motor #2	Motor #3	Motor #4	Motor #5	Motor #6	Motor #7	Motor #8
Stopped-on-position-limit bit	M130->Y:$0814,11,1	M230->Y:$08D4,11,1	M330->Y:$0994,11,1	M430->Y:$0A54,11,1	M530->Y:$0B14,11,1	M630->Y:$0BD4,11,1	M730->Y:$0C94,11,1	M830->Y:$0D54,11,1
Positive-end-limit-set bit	M131->X:$003D,21,1	M231->X:$0079,21,1	M331->X:$00B5,21,1	M431->X:$00F1,21,1	M531->X:$012D,21,1	M631->X:$0169,21,1	M731->X:$01A5,21,1	M831->X:$01E1,21,1
Negative-end-limit-set bit	M132->X:$003D,22,1	M232->X:$0079,22,1	M332->X:$00B5,22,1	M432->X:$00F1,22,1	M532->X:$012D,22,1	M632->X:$0169,22,1	M732->X:$01A5,22,1	M832->X:$01E1,22,1
Desired-velocity-zero bit	M133->X:$003D,13,1	M233->X:$0079,13,1	M333->X:$00B5,13,1	M433->X:$00F1,13,1	M533->X:$012D,13,1	M633->X:$0169,13,1	M733->X:$01A5,13,1	M833->X:$01E1,13,1
Dwell-in-progress bit	M135->X:$003D,15,1	M235->X:$0079,15,1	M335->X:$00B5,15,1	M435->X:$00F1,15,1	M535->X:$012D,15,1	M635->X:$0169,15,1	M735->X:$01A5,15,1	M835->X:$01E1,15,1
Running-program bit	M137->X:$003D,17,1	M237->X:$0079,17,1	M337->X:$00B5,17,1	M437->X:$00F1,17,1	M537->X:$012D,17,1	M637->X:$0169,17,1	M737->X:$01A5,17,1	M837->X:$01E1,17,1
Open-loop-mode bit	M138->X:$003D,18,1	M238->X:$0079,18,1	M338->X:$00B5,18,1	M438->X:$00F1,18,1	M538->X:$012D,18,1	M638->X:$0169,18,1	M738->X:$01A5,18,1	M838->X:$01E1,18,1
Amplifier-enabled status bit	M139->Y:$0814,14,1	M239->Y:$08D4,14,1	M339->Y:$0994,14,1	M439->Y:$0A54,14,1	M539->Y:$0B14,14,1	M639->Y:$0BD4,14,1	M739->Y:$0C94,14,1	M839->Y:$0D54,14,1
In-position bit	M140->Y:$0814,0,1	M240->Y:$08D4,0,1	M340->Y:$0994,0,1	M440->Y:$0A54,0,1	M540->Y:$0B14,0,1	M640->Y:$0BD4,0,1	M740->Y:$0C94,0,1	M840->Y:$0D54,0,1
Warning-following error bit	M141->Y:$0814,1,1	M241->Y:$08D4,1,1	M341->Y:$0994,1,1	M441->Y:$0A54,1,1	M541->Y:$0B14,1,1	M641->Y:$0BD4,1,1	M741->Y:$0C94,1,1	M841->Y:$0D54,1,1
Fatal-following-error bit	M142->Y:$0814,2,1	M242->Y:$08D4,2,1	M342->Y:$0994,2,1	M442->Y:$0A54,2,1	M542->Y:$0B14,2,1	M642->Y:$0BD4,2,1	M742->Y:$0C94,2,1	M842->Y:$0D54,2,1
Amplifier-fault-error bit	M143->Y:$0814,3,1	M243->Y:$08D4,3,1	M343->Y:$0994,3,1	M443->Y:$0A54,3,1	M543->Y:$0B14,3,1	M643->Y:$0BD4,3,1	M743->Y:$0C94,3,1	M843->Y:$0D54,3,1
Home-complete bit	M145->Y:$0814,10,1	M245->Y:$08D4,10,1	M345->Y:$0994,10,1	M445->Y:$0A54,10,1	M545->Y:$0B14,10,1	M645->Y:$0BD4,10,1	M745->Y:$0C94,10,1	M845->Y:$0D54,10,1

Motor Move Registers	**Motor #1**	**Motor #2**	**Motor #3**	**Motor #4**	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
Commanded position (1/[Ix08*32] cts	M161->D:$0028	M261->D:$0064	M361->D:$00A0	M461->D:$00DC	M561->D:$0118	M661->D:$0154	M761->D:$0190	M861->D:$01CC
Actual position (1/[Ix08*32] cts)	M162->D:$002B	M262->D:$0067	M362->D:$00A3	M462->D:$00DF	M562->D:$011B	M662->D:$0157	M762->D:$0193	M862->D:$01CF
Target (end) position (1/[Ix08*32]	M163->D:$080B	M263->D:$08CB	M363->D:$098B	M463->D:$0A4B	M563->D:$0B0B	M663->D:$0BCB	M763->D:$0C8B	M863->D:$0D4B
Position bias (1/[Ix08*32] cts)	M164->D:$0813	M264->D:$08D3	M364->D:$0993	M464->D:$0A53	M564->D:$0B13	M664->D:$0BD3	M764->D:$0C93	M864->D:$0D53
X-axis target position (engineering units)	M165->L:$081F	M265->L:$0820	M365->L:$0821	M465->L:$0819	M565->L:$081A	M665->L:$081B	M765->L:$081C	M865->L:$081D
Actual velocity (1/[Ix09*32] cts/cyc)	M166->X:$0033,0,24,S	M266->X:$006F,0,24,S	M366->X:$00AB,0,24,S	M466->X:$00E7,0,24,S	M566->X:$0123,0,24,S	M666->X:$015F,0,24,S	M766->X:$019B,0,24,S	M866->X:$01D7,0,24,S
Present master (handwheel) pos (1/[Ix07*32] cts)	M167->D:$002D	M267->D:$0069	M367->D:$00A5	M467->D:$00E1	M567->D:$011D	M667->D:$0159	M767->D:$0195	M867->D:$01D1
Filter Output (DAC bits)	M168->X:$0045,8,16,S	M268->X:$0081,8,16,S	M368->X:$00BD,8,16,S	M468->X:$00F9,8,16,S	M568->X:$0135,8,16,S	M668->X:$0171,8,16,S	M768->X:$01AD,8,16,S	M868->X:$01E9,8,16,S
Compensation correction	M169->D:$0046	M269->D:$0082	M369->D:$00BE	M469->D:$00FA	M569->D:$0136	M669->D:$0172	M769->D:$01AE	M869->D:$01EA
Present phase pos. includes fraction in Y-register	M170->D:$0041	M270->D:$007D	M370->D:$00B9	M470->D:$00F5	M570->D:$0131	M670->D:$016D	M770->D:$01A9	M870->D:$01E5
Present phase position (counts*Ix70)	M171->X:$0041,0,24,S	M271->X:$007D,0,24,S	M371->X:$00B9,0,24,S	M471->X:$00F5,0,24,S	M571->X:$0131,0,24,S	M671->X:$016D,0,24,S	M771->X:$01A9,0,24,S	M871->X:$01E5,0,24,S
Variable jog position/distance (counts)	M172->L:$082B	M272->L:$08EB	M372->L:$09AB	M472->L:$0A6B	M572->L:$0B2B	M672->L:$0BEB	M772->L:$0CAB	M872->L:$0D6B
Encoder home capture offset (counts)	M173->Y:$0815,0,24,S	M273->Y:$08D5,0,24,S	M373->Y:$0995,0,24,S	M473->Y:$0A55,0,24,S	M573->Y:$0B15,0,24,S	M673->Y:$0BD5,0,24,S	M773->Y:$0C95,0,24,S	M873->Y:$0D55,0,24,S
filtered actual vel. (1/[Ix09*32] cts/servo cycle)	M174->Y:$082A,0,24,S	M274->Y:$08EA,0,24,S	M374->Y:$09AA,0,24,S	M474->Y:$0A6A,0,24,S	M574->Y:$0B2A,0,24,S	M674->Y:$0BEA,0,24,S	M774->Y:$0CAA,0,24,S	M874->Y:$0D6A,0,24,S
Motor #1 following error (1/[Ix08*32] cts)	M175->D:$0840	M275->D:$0900	M375->D:$09C0	M475->D:$0A80	M575->D:$0B40	M675->D:$0C00	M775->D:$0CC0	M875->D:$0D80

Coordinate System Status Bits	Coordinate System 1	Coordinate System 2	Coordinate System 3	Coordinate System 4	Coordinate System 5	Coordinate System 6	Coordinate System 7	Coordinate System 8
Program-running bit	M180->X:$0818,0,1	M280->X:$08D8,0,1	M380->X:$0998,0,1	M480->X:$0A58,0,1	M580->X:$0B18,0,1	M680->X:$0BD8,0,1	M780->X:$0C98,0,1	M880->X:$0D58,0,1
Circle-radius-error bit	M181->Y:$0817,21,1	M281->Y:$08D7,21,1	M381->Y:$0997,21,1	M481->Y:$0A57,21,1	M581->Y:$0B17,21,1	M681->Y:$0BD7,21,1	M781->Y:$0C97,21,1	M881->Y:$0D57,21,1
Run-time-error bit	M182->Y:$0817,22,1	M282->Y:$08D7,22,1	M382->Y:$0997,22,1	M482->Y:$0A57,22,1	M582->Y:$0B17,22,1	M682->Y:$0BD7,22,1	M782->Y:$0C97,22,1	M882->Y:$0D57,22,1
Continuous motion request	M184->X:$0818,4,1	M284->X:$08D8,4,1	M384->X:$0998,4,1	M484->X:$0A58,4,1	M584->X:$0B18,4,1	M684->X:$0BD8,4,1	M784->X:$0C98,4,1	M884->X:$0D58,4,1
In-position bit (AND of motors)	M187->Y:$0817,17,1	M287->Y:$08D7,17,1	M387->Y:$0997,17,1	M487->Y:$0A57,17,1	M587->Y:$0B17,17,1	M687->Y:$0BD7,17,1	M787->Y:$0C97,17,1	M887->Y:$0D57,17,1
Warning-following-error bit (OR)	M188->Y:$0817,18,1	M288->Y:$08D7,18,1	M388->Y:$0997,18,1	M488->Y:$0A57,18,1	M588->Y:$0B17,18,1	M688->Y:$0BD7,18,1	M788->Y:$0C97,18,1	M888->Y:$0D57,18,1
Fatal-following-error bit (OR)	M189->Y:$0817,19,1	M289->Y:$08D7,19,1	M389->Y:$0997,19,1	M489->Y:$0A57,19,1	M589->Y:$0B17,19,1	M689->Y:$0BD7,19,1	M789->Y:$0C97,19,1	M889->Y:$0D57,19,1
Amp-fault-error bit (OR of motors)	M190->Y:$0817,20,1	M290->Y:$08D7,20,1	M390->Y:$0997,20,1	M490->Y:$0A57,20,1	M590->Y:$0B17,20,1	M690->Y:$0BD7,20,1	M790->Y:$0C97,20,1	M890->Y:$0D57,20,1

Motor Axis Definition Registers	Motor #1	Motor #2	Motor #3	Motor #4	Motor #5	Motor #6	Motor #7	Motor #8
X/U/A/B/C-Axis scale factor (cts/unit)	M191->L:$0822	M291->L:$08E2	M391->L:$09A2	M491->L:$0A62	M591->L:$0B22	M691->L:$0BE2	M791->L:$0CA2	M891->L:$0D62
Y/V-Axis scale factor (cts/unit)	M192->L:$0823	M292->L:$08E3	M392->L:$09A3	M492->L:$0A63	M592->L:$0B23	M692->L:$0BE3	M792->L:$0CA3	M892->L:$0D63
Z/W-Axis scale factor (cts/unit)	M193->L:$0824	M293->L:$08E4	M393->L:$09A4	M493->L:$0A64	M593->L:$0B24	M693->L:$0BE4	M793->L:$0CA4	M893->L:$0D64
Axis offset (cts)	M194->L:$0825	M294->L:$08E5	M394->L:$09A5	M494->L:$0A65	M594->L:$0B25	M694->L:$0BE5	M794->L:$0CA5	M894->L:$0D65

Coordinate System Variables	Coordinate System 1	Coordinate System 2	Coordinate System 3	Coordinate System 4	Coordinate System 5	Coordinate System 6	Coordinate System 7	Coordinate System 8
Host commanded time base (I10 units)	M197->X:$0806,0,24,S	M297->X:$08C6,0,24,S	M397->X:$0986,0,24,S	M497->X:$0A46,0,24,S	M597->X:$0B06,0,24,S	M697->X:$0BC6,0,24,S	M797->X:$0C86,0,24,S	M897->X:$0D46,0,24,S
Present time base (I10 units)	M198->X:$0808,0,24,S	M298->X:$08C8,0,24,S	M398->X:$0988,0,24,S	M498->X:$0A48,0,24,S	M598->X:$0B08,0,24,S	M698->X:$0BC8,0,24,S	M798->X:$0C88,0,24,S	M898->X:$0D48,0,24,S

APPENDIX F – I/O SUGGESTED M-VARIABLE DEFINITIONS

I/O M-Variables	Definition
MI/O0	M900->Y:$FFD0,0,1
MI/O1	M901->Y:$FFD0,1,1
MI/O2	M902->Y:$FFD0,2,1
MI/O3	M903->Y:$FFD0,3,1
MI/O4	M904->Y:$FFD0,4,1
MI/O5	M905->Y:$FFD0,5,1
MI/O6	M906->Y:$FFD0,6,1
MI/O7	M907->Y:$FFD0,7,1
MI/O8	M908->Y:$FFD0,8,1
MI/O9	M909->Y:$FFD0,9,1
MI/O10	M910->Y:$FFD0,10,1
MI/O11	M911->Y:$FFD0,11,1
MI/O12	M912->Y:$FFD0,12,1
MI/O13	M913->Y:$FFD0,13,1
MI/O14	M914->Y:$FFD0,14,1
MI/O15	M915->Y:$FFD0,15,1
MI/O16	M916->Y:$FFD0,16,1
MI/O17	M917->Y:$FFD0,17,1
MI/O18	M918->Y:$FFD0,18,1
MI/O19	M919->Y:$FFD0,19,1
MI/O20	M920->Y:$FFD0,20,1
MI/O21	M921->Y:$FFD0,21,1
MI/O22	M922->Y:$FFD0,22,1
MI/O23	M923->Y:$FFD0,23,1
MI/O24	M924->Y:$FFD1,0,1
MI/O25	M925->Y:$FFD1,1,1
MI/O26	M926->Y:$FFD1,2,1
MI/O27	M927->Y:$FFD1,3,1
MI/O28	M928->Y:$FFD1,4,1
MI/O29	M929->Y:$FFD1,5,1
MI/O30	M930->Y:$FFD1,6,1
MI/O31	M931->Y:$FFD1,7,1
MI/O32	M932->Y:$FFD1,8,1
MI/O33	M933->Y:$FFD1,9,1
MI/O34	M934->Y:$FFD1,10,1
MI/O35	M935->Y:$FFD1,11,1
MI/O36	M936->Y:$FFD1,12,1
MI/O37	M937->Y:$FFD1,13,1
MI/O38	M938->Y:$FFD1,14,1
MI/O39	M939->Y:$FFD1,15,1
MI/O40	M940->Y:$FFD1,16,1
MI/O41	M941->Y:$FFD1,17,1
MI/O42	M942->Y:$FFD1,18,1
MI/O43	M943->Y:$FFD1,19,1
MI/O44	M944->Y:$FFD1,20,1
MI/O45	M945->Y:$FFD1,21,1
MI/O46	M946->Y:$FFD1,22,1
MI/O47	M947->Y:$FFD1,23,1

Control-Panel Port Input Bits	Definition
Jog Minus Input	M20->Y:$FFC0,8,1
Jog Plus Input	M21->Y:$FFC0,9,1
Prejog Input	M22->Y:$FFC0,10,1
Start (Run) Input	M23->Y:$FFC0,11,1
Step/Quit Input	M24->Y:$FFC0,12,1
Stop (Abort) Input	M25->Y:$FFC0,13,1
Home Command Input	M26->Y:$FFC0,14,1
Feed Hold Input	M27->Y:$FFC0,15,1
Motor/C.S. Select Input Bit 0	M28->Y:$FFC0,16,1
Motor/C.S. Select Input Bit 1	M29->Y:$FFC0,17,1
Motor/C.S. Select Input Bit 2	M30->Y:$FFC0,18,1
Motor/C.S. Select Input Bit 3	M31->Y:$FFC0,19,1
Selected Motor/C.S. Number	M32->Y:$FFC0,16,4,C

Thumbwheel Port Bits	Definition
SEL0 Output	M40->Y:$FFC1,8,1
SEL1 Output	M41->Y:$FFC1,9,1
SEL2 Output	M42->Y:$FFC1,10,1
SEL3 Output	M43->Y:$FFC1,11,1
SEL4 Output	M44->Y:$FFC1,12,1
SEL5 Output	M45->Y:$FFC1,13,1
SEL6 Output	M46->Y:$FFC1,14,1
SEL7 Output	M47->Y:$FFC1,15,1
SEL0-7 Outputs byte	M48->Y:$FFC1,8,8,U
DAT0 Input	M50->Y:$FFC1,0,1
DAT1 Input	M51->Y:$FFC1,1,1
DAT2 Input	M52->Y:$FFC1,2,1
DAT3 Input	M53->Y:$FFC1,3,1
DAT4 Input	M54->Y:$FFC1,4,1
DAT5 Input	M55->Y:$FFC1,5,1
DAT6 Input	M56->Y:$FFC1,6,1
DAT7 Input	M57->Y:$FFC1,7,1
DAT0-7 Inputs byte	M58->Y:$FFC1,0,8,U

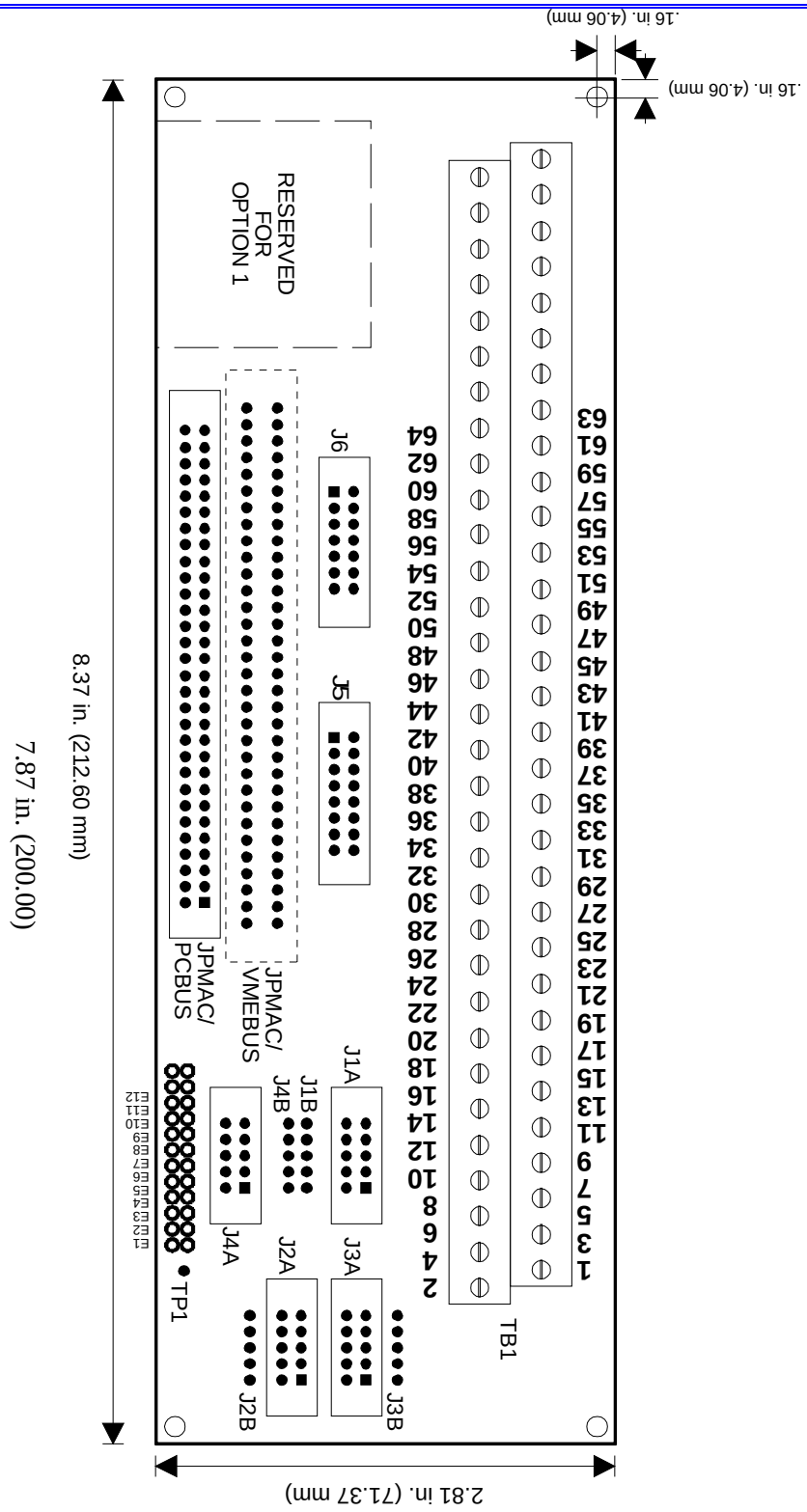
To clear all existing definitions	M0..1023->*
Servo cycle counter	M0->X:$0,0,24,U

General Purpose Inputs and Outputs	Definition
Machine Output 1	M1->Y:$FFC2,8,1
Machine Output 2	M2->Y:$FFC2,9,1
Machine Output 3	M3->Y:$FFC2,10,1
Machine Output 4	M4->Y:$FFC2,11,1
Machine Output 5	M5->Y:$FFC2,12,1
Machine Output 6	M6->Y:$FFC2,13,1
Machine Output 7	M7->Y:$FFC2,14,1
Machine Output 8	M8->Y:$FFC2,15,1
Machine Outputs 1-8 treated as byte	M9->Y:$FFC2,8,8,U
Machine Input 1	M11->Y:$FFC2,0,1
Machine Input 2	M12->Y:$FFC2,1,1
Machine Input 3	M13->Y:$FFC2,2,1
Machine Input 4	M14->Y:$FFC2,3,1
Machine Input 5	M15->Y:$FFC2,4,1
Machine Input 6	M16->Y:$FFC2,5,1
Machine Input 7	M17->Y:$FFC2,6,1
Machine Input 8	M18->Y:$FFC2,7,1
Machine Inputs 1-8 treated as byte	M19->Y:$FFC2,0,8,U

PMAC Built-in timers	Definition
Timer register 1 (8388608/I10 msec)	M90->X:$0700,0,24,S
Timer register 2 (8388608/I10 msec)	M91->Y:$0700,0,24,S
Timer register 3 (8388608/I10 msec)	M92->X:$0701,0,24,S
Timer register 4 (8388608/I10 msec)	M93->Y:$0701,0,24,S

Open memory; cleared to 0 on power-on/reset	$0770 - $077F
Open registers (stored in battery-backed RAM)	$07F0 - $07FF

APPENDIX G – ACC-8D/8P PINOUT DESCRIPTIONS



	Pin #	Symbol	Function		Pin #	Symbol	Function
Digital Power	1	+5V	OUTPUT	Analog Power	58	AGND	INPUT
	2	+5V	OUTPUT		59	A+15V/OPT+V	INPUT
	3	GND	COMMON		60	A-15V	INPUT
	4	GND	COMMON	**Refer to the appropriate PMAC Hardware Reference manual for connections and jumper descriptions.**			
	57	FEFCO/	OUTPUT				
Encoder Inputs 1, 5, 9, 13	25	CHA	INPUT	Encoder Inputs 3, 7, 11, 15	13	CHA	INPUT
	27	CHA/	INPUT		15	CHA/	INPUT
	21	CHB	INPUT		9	CHB	INPUT
	23	CHB/	INPUT		11	CHB/	INPUT
	17	CHC	INPUT		5	CHC	INPUT
	19	CHC/	INPUT		7	CHC/	INPUT
	1	+5V	OUTPUT		1	+5V	OUTPUT
	3	GND	COMMON		3	GND	COMMON
Amplifier 1, 5, 9, 13	43	DAC	OUTPUT	Amplifier 3, 7, 11, 15	29	DAC	OUTPUT
	45	DAC/	OUTPUT		31	DAC/	OUTPUT
	47	AENA/DIR	OUTPUT		33	AENA/DIR	OUTPUT
	49	FAULT	INPUT		35	FAULT	INPUT
	58	AGND	INPUT		58	AGND	INPUT
Flags 1, 5, 9, 13	51	+LIM	INPUT	Flags 3, 7, 11, 15	37	+LIM	INPUT
	53	-LIM	INPUT		39	-LIM	INPUT
	55	HMFL	INPUT		41	HMFL	INPUT
	58	AGND	INPUT		58	AGND	INPUT
Encoder Inputs 2, 6, 10, 14	26	CHA	INPUT	Encoder Inputs 4, 8, 12, 16	14	CHA	INPUT
	28	CHA/	INPUT		16	CHA/	INPUT
	22	CHB	INPUT		10	CHB	INPUT
	24	CHB/	INPUT		12	CHB/	INPUT
	18	CHC	INPUT		6	CHC	INPUT
	20	CHC/	INPUT		8	CHC/	INPUT
	1	+5V	OUTPUT		1	+5V	OUTPUT
	3	GND	COMMON		3	GND	COMMON
Amplifier 2, 6, 10, 14	44	DAC	OUTPUT	Amplifier 4, 8, 12, 16	30	DAC	OUTPUT
	46	DAC/	OUTPUT		32	DAC/	OUTPUT
	48	AENA/DIR	OUTPUT		34	AENA/DIR	OUTPUT
	50	FAULT	INPUT		36	FAULT	INPUT
	58	AGND	INPUT		58	AGND	INPUT
Flags 2, 6, 10, 14	52	+LIM	INPUT	Flags 4, 8, 12, 16	38	+LIM	INPUT
	54	-LIM	INPUT		40	-LIM	INPUT
	56	HMFL	INPUT		42	HMFL	INPUT
	58	AGND	INPUT		58	AGND	INPUT