PMC

Programmable Motion Controller

(including MPL version 3.0)

Installation & Operation Manual
PMC001d

19 Linden Park

Rochester, NY 14625

PMC001d.87.02.16

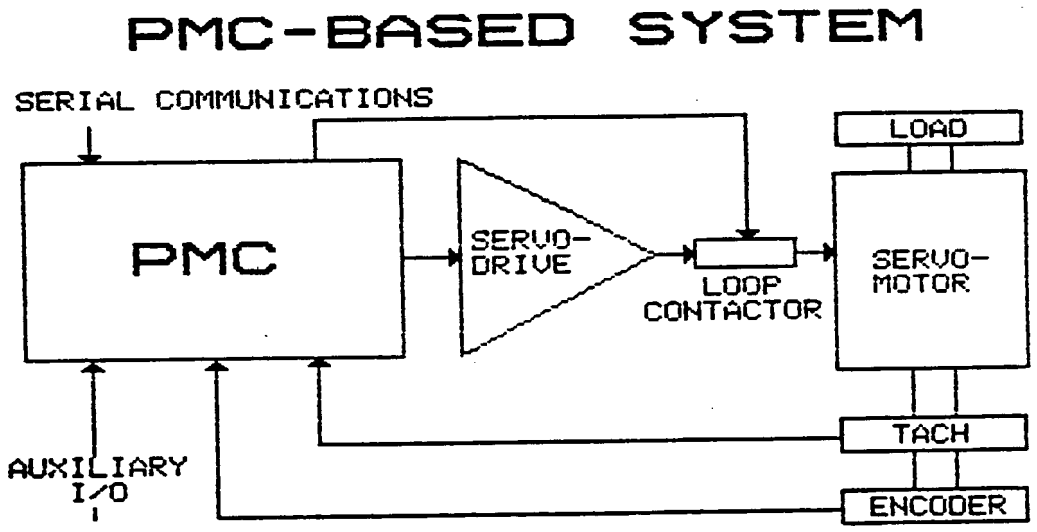
TABLE OF CONTENTS

GENERAL DESCRIPTION

1.1 INTRODUCTION

The ORMEC **Programmable Motion Controller (PMC)** is a single board microcomputer based product which facilitates the design of high performance motion control applications. A typical servo positioning system utilizing a PMC is diagrammed in Appendix 8.1.

In combination with a servodrive, a servomotor, a DC tachometer and an incremental position encoder, the PMC is used to create a closed loop digital position system. This position control system translates straightforward single character commands into high performance motion. Acceleration, velocity and distance are specified using ORMEC's **Motion Programming Language (MPL)**, and controlled by the PMC to a high degree of precision.



Straightforward, single character, MPL commands may be executed directly from the **Serial Communications Interface (SCI)**, or from non-volatile program memory to create motion control routines. This interactive aspect of MPL greatly reduces the application effort required to implement sophisticated high performance motion control systems compared with alternative approaches.

PMCs also include 16 discrete digital I/O points called the **Machine I/O (MIO)**, which can be used as an interface directly to the operator, the machine under control, or for a straightforward interface to Programmable Controllers or other computers or control electronics with discrete I/O capability.

The PMC is available as a standard product in two versions. Information in this manual applies to both versions except when otherwise noted.

PMC-903 -Standard MPL firmware with programming and 2K bytes of non-volatile RAM memory for stand-alone or computer hosted operation with fast programming even while the servo is in motion

PMC-904 -Standard MPL firmware with programming and 8K bytes of non-volatile RAM memory for stand-alone or computer hosted operation with fast programming even while the servo is in motion

Both the PMC-903 and PMC-904 can be operated in three configurations:

Computer Hosted	- interfaced through the SCI
Programmable Controller Hosted	- interfaced through the Machine I/O (MIO)
Stand Alone	- interfaced through the MIO

1.2 STAND ALONE OPERATION

Since the PMC has non-volatile MPL program memory, the user can define a "powerup routine" to coordinate with the PMC's 16 discrete Machine I/O (MIO) points and operate a stand alone motion control system, controlling a machine or machine module.

Any standard ASCII terminal can be used to "program" the desired machine operation in MPL, configuring the program to start automatically each time power is applied. This machine operation is easily programmed to mix servomotor controlled motion with the operation of the discrete I/O points.

Included in the 16 discrete parallel I/O signals of the MIO are:
- eight general purpose inputs (for use with switches or machines sensors)
- four general purpose outputs (for use with indicators, solenoids, etc.)

The MIO is flat cable compatible with industry standard Opto 22 type I/O modules allowing machine interface via AC or DC drivers and receivers with a wide range of voltage and current ratings. A detailed description of the Machine I/O Interface is in Section 4.5.

1.3 PROGRAMMABLE CONTROLLER HOSTED OPERATION

A PMC based system can be interfaced to programmable controllers through the MIO by either programming it for stand-alone operation with handshaking to a programmable controller or by using the approach described below:

- The user programs up to 32 individual motion control functions using any ASCII serial terminal, giving them single character names.

- The programmable controller selects the appropriate function by placing the motion function address in parallel at the Machine I/O (MIO) Interface and asserting the "execute" input.

- The programmable controller can tell when the function is complete and the PMC is ready for a new command by observing the "ready" signal from the MIO.

- The function can be interrupted, and motion halted, at any time by asserting the "stop" input at the MIO.

- The general purpose MIO signals may be used for "program to program" communications between the PMC and the programmable controller.

1.4 COMPUTER HOSTED OPERATION

A PMC based positioning system can be also used as an intelligent motion control peripheral serving a host computer by accepting commands through the *Serial Communications Interface (SCI).*

The host computer interface to a PMC is similar to communicating with a serial terminal. The host computer sends a byte (character) of information to the PMC, and waits for that byte to be accepted by the PMC. The acceptance of each character is indicated by the hardware handshake line, and is usually less than a millisecond.

When an MPL command is completed, the PMC executes it. Upon completion of the command, the PMC sends a right curved bracket ")" ($7D_H$ or FD_H) to the host to indicate that it is "READY" for a new command.

Therefore, simply by sending ASCII characters to the PMC, the host computer can execute PMC positioning commands directly, including commands which cause the PMC to perform complex user defined motion control functions. Only high level commands and status requests are required from the host computer, since the PMC isolates it from real time servicing requirements of the servo positioning system, allowing a single microcomputer host to coordinate the operation of several high performance servomotor systems.

Relative or absolute positioning commands of more than nine digits can be performed. In addition, system position, velocity and acceleration information can be requested asynchronously. This relieves the host from tasks such as keeping track of the current absolute position of the system or storing system parameters required by the application.

1.5 SIGNAL NAMING CONVENTIONS

Throughout this manual, references to inverted logical signals will use the convention of following the signal name with an apostrophe ('). e.g. INPUT' Some drawings or charts in this manual may use the "overbar" notation as follows:

e.g. $\overline{\text{INPUT}}$

Signals without apostrophes will be considered logically "true" or "asserted" when they are "high" or "set" i.e. at the level of the power supply (either +5 VDC or +12 VDC). If they are at 0 VDC ("low" or "cleared"), they are considered logically "false". e.g. A signal named RESET would be expected to perform the "RESET" function when it is "true" or "asserted", which is when it is "high" (at a +5 VDC level) or "set" (to a logical 1).

Conversely, signals with "overbars" or apostrophes following them are considered logically "true", or "asserted" when they are "low". e.g. A signal named RESET' should be "low", or at 0 VDC, in order to perform the "RESET" function. The term "logical complement" may be used in reference to signals meaning that they will be "low" when the function is asserted, and "high" when the function is not asserted.

1.6 COMPARISON OF SERVOS AND STEPPING MOTORS

In addition to being easier to use than a stepping motor/translator system, a PMC based motion control system has important advantages when compared to stepping motor/translator systems.

1. Closed loop operation results in load independent positioning accuracy and predictability; in addition, system position information is available for the user.

2. Torque, speed, and acceleration capabilities with existing servomotor/servodrive technology are much greater than with stepping motor technology, which leads to much higher performance of the resulting equipment.

3. Angular resolution and accuracy are tailored to the application needs, with existing technology providing resolutions in excess of 500,000 pulses per revolution where required. In contrast typical stepping motor resolutions are 100 to 200 pulses per revolution. To provide higher positioning accuracy, rotary encoders can also be mounted directly on the load or direct linear feedback can be provided using linear position encoding devices.

4. The maximum stepping rate of a PMC based servo system is 384 KHz, compared to typical stepping motor-translator stepping rates of 2 to 5 KHz.

5. An internal position error counter has 12 bit capacity allowing errors of up to ±2048 counts, compared with the intolerance of stepping motors to following errors. Should the position error exceed the capacity of this internal counter, the PMC senses it as a fault and reduces the servo loop gains to zero. In addition a buffered TTL level output is provided to drop out the loop contactor and/or power down the servodrive. This information is also available to the user.

1.7 SYSTEM INTERFACE

A typical digital position servo using a PMC is diagrammed in Appendix 8.1. Easy to use interfaces are provided for convenient connection to the other system elements, with the PMC combining all the other system components together and creating a working digital position servo system.

The PMC does not depend on the servodrive to close the velocity loop or provide feedback control system compensation, but utilizes it only as a power amplifier. Closed loop system performance is easily optimized with five internal software selectable loop gains and compensation circuits. If velocity loop compensation circuitry is available on the servodrive of choice, the sophisticated user can utilize it by wiring the DC tachometer signal to the servodrive directly *instead* of to the PMC. This approach is not usually recommended by ORMEC, however, because the following advantages of centralizing servo loop gain adjustment and compensation at the PMC are lost:

- Software selectable loop gain in the velocity loop
- Software selectable velocity loop compensation break frequency
- Interlocks which disable integral + proportional compensators, allowing smooth powerup

In order to make the PMC easy to use in a wide variety of applications, special consideration has been given to the digital interface circuitry used for the encoder and the machine sensor. These digital inputs have high input impedances (20K ohm minimum), and can be connected in single-ended or differential configurations. The differential configuration gives the maximum noise immunity and should be used for encoder cables longer than ten feet.

The inputs are shipped configured for single-ended operation with a 2 volt switching threshold and can be converted to balanced differential by the removal of a socketed SIP resistor. See the INSTALLATION Section 4.4 for details. For greater noise immunity, these digital inputs are unaffected by input pulse widths of less than a specified minimum. See Section 3.4 for details.

THEORY OF OPERATION

2.1 INTRODUCTION

A PMC based positioning system translates high level commands from the Serial Communications Interface (SCI) or MPL program memory into high performance motion. The PMC System Architecture is described in Appendix 8.3 and the PMC Analog Architecture is described in Appendix 8.4.

2.2 SYSTEM ARCHITECTURE OVERVIEW

2.2.1 Incremental Position Encoder

The PMC is configured to use an incremental position encoder. This encoder is used to measure changes in load displacement and provide position feedback. Most PMC based digital positioning systems use optical incremental encoders, however any type of position encoder that produces **TTL compatible phase quadrature** outputs is acceptable.

An encoder with phase quadrature outputs has two digital square wave outputs, which are displaced in phase by 90 degrees, that switch between 0 VDC and +5 VDC. These two outputs are commonly called "Encoder Channel A" (ENCA) and "Encoder Channel B" (ENCB), and the direction of motion determines the phase relationship between them. i.e. If for "forward" motion, Encoder Channel A leads Encoder Channel B by 90 degrees, then for "reverse" motion, Encoder Channel A will lag Encoder Channel B by 90 degrees. See Appendix 8.8 for a diagram showing the relationship between ENCA and ENCB for forward and reverse motion.

For more information on incremental position encoders, consult literature from a position encoder manufacturer. An excellent source of information on the subject is the book *Techniques for Digitizing Rotary and Linear Motion* by Dynamics Research Corporation of Wilmington, MA.

The resolution of the position encoding is determined by the number of lines per unit distance that cause output pulses. **Line count** is a term commonly used by manufacturers of incremental position encoders, and it refers to the number of line pairs of the physical encoding device per unit distance. e.g. lines per revolution or lines per inch.

2.2.2 Encoder Interface

Refer to discussion of the PMC System Architecture in Appendix 8.3, the PMC Motor Loop Interface in Appendix 8.6 and the Motion Signal Overview in Appendix 8.8 for the following discussion.

Integral to the PMC is a quadrature decoder circuit which interprets the phase quadrature encoder input channels to determine the load direction and distance traveled. The PMC's quadrature decoder circuit utilizes "4-times multiplication" circuitry, which means that every transition of the two encoder channels is utilized to create a forward or reverse motion feedback pulse, as appropriate. These signals are illustrated in Appendix 8.8.

Because four **encoder distance units** are derived from each encoder cycle, the effective resolution of a digital position control system utilizing a PMC is "4 times" the stated "linecount" of the incremental position encoder. e.g. Using an encoder with a linecount of 1000 lines per revolution results in a digital position control system with a resolution of 4000 counts per revolution.

The encoder pulses from the quadrature decoder are subtracted from the position reference informa

tion in the digital "position summing junction", which in turn has an output proportional to the "position error" of the control system.

2.2.3 Summing and Compensation Circuitry

In order to cause motion of the digital positioning control system, the on-board microprocessor outputs serial forward or reverse command pulses, which are "summed" with the decoded encoder feedback pulses in the 12 bit digital position summing junction (up/down counter). The microprocessor simultaneously provides an analog "FeedForward Velocity Reference" output (FFVELREF), which is applied directly to the velocity loop. The presence of this signal greatly reduces the amount of "position following error" required to operate the system at high speed.

The "Position Error" signal (POSERR) is processed by the position compensation amplifier (UA2C). Here the position loop gain (PLGAIN) is adjusted over a range of 1 to 255 (48 db) and an integral + proportional compensator can be enabled and adjusted by setting PLCOMP.

In general, an analog DC tachometer is used to sense motor velocity and provide velocity feedback for maximum performance. In some systems, this signal is derived by other means such as the encoder quadrature signals. This velocity feedback signal (HVTACH or LVTACH) is summed with the feedforward velocity reference (FFVELREF) and the compensated position error signal resulting in the current command signal (ICMD). This signal is used as the input to the servodrive. Velocity loop gain (VLGAIN) is adjusted over a range of 1 to 255 (48 db) and an integral + proportional compensator can be enabled and adjusted by setting VLCOMP. Detailed instructions for tuning the system are found in Section 5.4. Information on the Tuning Command and examples are in Section 6.9.

2.2.4 Modes of Operation

There are four operating modes of the PMC:

Mode 0 - IDLE

In idle mode, the PMC disables the servodrive enable output (SDRVEN) and disables the position and velocity loop compensation circuitry.

Mode 1 - VELOCITY

In velocity mode, the PMC operates as an analog velocity control system, driving the velocity loop with an the analog FeedForward Velocity Reference signal derived by the microprocessor and the reference generation circuitry. This voltage is applied to the velocity loop through the feedforward gain adjust (FFGAIN). An analog signal proportional to speed is provided for external use, by connecting to the external speed reference output and adjusting the gain with the external gain adjust (XGAIN). This mode is useful for setup and troubleshooting of the servo system, as well as being useful for advanced motion control applications which change mode from position control to some other feedback source such as force, tension etc. Distance position reference pulses are not entered into the **Position Summing Junction (PSJ)**.

Mode 2 - POSITION

In position mode, the PMC operates as a digital positioning system, controlling speed and position as a phase lock loop position controller. Digital position reference pulses derived by the Reference Generation Circuitry are summed in the PSJ and an analog feedforward voltage is applied to the velocity loop.

The positioning error in the digital PSJ is monitored, and if it exceeds the 12 bit capacity of the PSJ (± 2048 counts), the system is returned to the IDLE mode.

Mode 4 - MASTER — In master mode, the PMC operates as a Master Axis Controller, providing common motion information to other PMCs which are controlling servomotors.

2.2.5 Operation Overview

The PMC initiates a position change by sending the desired number of pulses to the Position Summing Junction through the Position Reference (POSREF) inputs at a rate proportional to the desired velocity. Simultaneously, the PMC applies an analog velocity feed-forward voltage (FFVELREF) to the velocity summing junction to minimize the error required in the Position Summing Junction.

The compensated velocity error signal is used as a current command (ICMD), and is applied to the servodrive to cause the servomotor to generate sufficient torque to move the load with the desired acceleration and velocity for the designated distance.

The tachometer and associated velocity loop stabilize this transition and an encoder output results. The polarities of the forward/reverse command pulses and the decoded encoder pulse are such that the contents of the digital Position Summing Junction is reduced as the encoder feedback is received (negative feedback). If the input pulse train continues at a constant rate long enough for the transient to decay, the encoder feedback pulse train be forced to have the same frequency as the command pulses.

When the load moves a distance equivalent to the number of command pulses, the error value in the Position Summing Junction goes to zero, causing the system to stop. Because of the PMC's "4-times" encoder multiplication circuitry, a system using an encoder with a "line count" of 2500 lines per revolution will move 1/10,000 of a revolution for each command pulse received. Since the position loop remains active as long as the system is still in Mode 2 (position mode), a holding force related to the position loop gain will be present as required to hold proper output position.

2.3 MOTION PROGRAMMING LANGUAGE ARCHITECTURE

The PMC's Motion Programming Language is architected to provide a logical, consistent and easy to use set of commands which specify high performance motion. The result is a calculator like language which provides the application designer a great deal of freedom and power with which to solve unique application needs.

The approach taken is to utilize a set of single character commands followed by an optional argument (number) and one or more single character command terminators.

This approach allows convenient communications with virtually any commercially available ASCII terminal or computer system. Moreover, the brevity of the language provides a great deal of "power per keystroke", making it quick and easy to use. In addition, it requires minimal communications overhead when interfaced to a host computer.

Like a programmable calculator, commands can be executed interactively or combined in a "program" (or "routine") for future automatic execution. These routines are stored and edited in the "program buffer" using the Program command.

Because execution time is an important factor in most high performance motion control applications, the language is designed to operate quickly, with many commands requiring less than a millisecond and the longer commands requiring a few milliseconds.

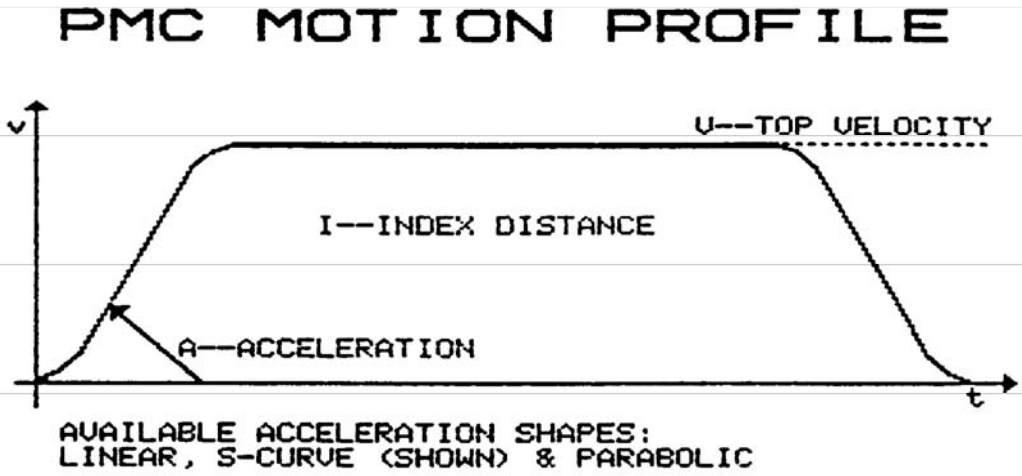
In addition, the PMC is architected to be able to service the real time requirements of commanding motion in the background, while simultaneously running this interpretive language. This feature allows the PMC to perform necessary calculations to set up future motions while simultaneously performing positioning tasks, enabling the user or host computer to talk to the PMC while motion is being controlled.

An overview of the PMC's Motion Programming Language is found in the following table:

MOTION PROGRAMMING LANGUAGE (MPL)		
Function	Description	Commands
Motion Parameter Commands	Defining Motion Parameters	Velocity Acceleration Index Jog Home
Motion Action Commands	Initiating Motion	Jog Go Index Home Contour Kill
Synchronization/ Interface Commands	Synchronizing Motion Manipulating Machine Outputs	Delay Until , & ; Terminators Output
Program Buffer Commands	Entering or Editing A Program	Program @ Labeling Motion Routines
Program Control Commands	Utilizing Subroutines & Creating Complex Motion Control Applications	Branching Looping Function Call Exit
System Setup Commands	Selecting System Options	Normalize Set/Show Tuning = Assign Axis-ID

2.3.1 MOTION Commands and Terminators

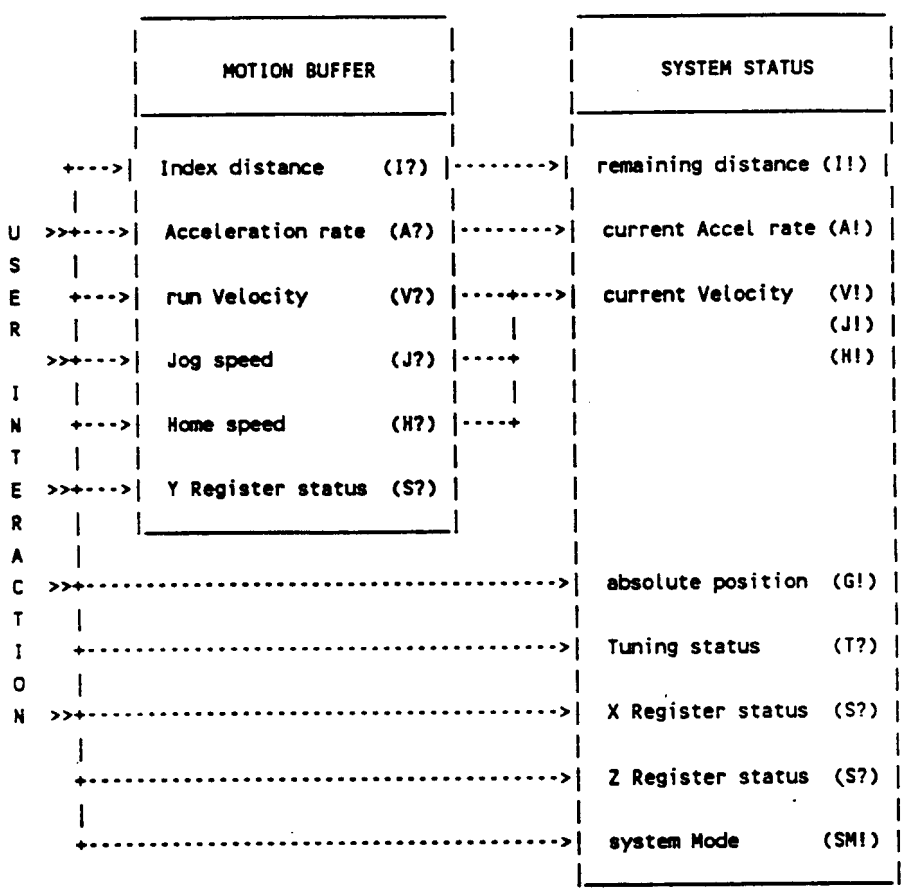
When performing indexing commands, the PMC creates motion profiles where the velocity is essentially trapezoidal as illustrated below. For the motion to be generated, the parameters of **distance**, **velocity** and **acceleration** are specified. The parameters are specified as numbers referencing distance as counts of the digital position transducer, velocity as the frequency of the pulse train from the digital position transducer (counts/sec or Hz) and acceleration as the frequency change per second (Hz per millisecond or kHz per second). For fastest possible real-time performance, each of these parameters is expressed as an integer value and has an allowable range, as well as a powerup default value. Refer to Section 6.3 for more detail regarding the parameter ranges. These parameters are stored in the **motion buffer**, from which they are be retrieved for each positioning command.



The PMC continuously keeps track of positioning system location to a range of approximately ± one billion counts, and is capable of providing that information to the host, performing software limit functions with it, and either keying program execution or moving to absolute locations within that range.

In addition to storing parameters for distance, top speed and acceleration rate, the PMC stores a home speed, a jog speed, and system configuration information. The configuration information is stored or displayed using the "S" command, and has an effect on system operation by enabling or disabling a number of product features.

Not only does the PMC allow host interaction while motion is being controlled, but the host system can also ask the PMC for up-to-date status information such as the present velocity, acceleration rate, absolute position, or distance to go until the current index is complete. This information is called system status information and interaction with the motion buffer and system status information is illustrated in the following chart.



The PMC can parametize the next motion profile while a motion is being commanded because the user interacts with the motion buffer for setting up the parameters for the next move. At the time each motion is commanded, the PMC takes information from the motion buffer and initiates the move. MOTION Commands and Terminators can be further classified as those which only set up **Parameters** for motion control, and those which also are used to initiate **Motion**. These two sets of commands are outlined in the following chart.

MOTION	
PARAMETERS	**Commands**
	Velocity - set/display maximum velocity
	Acceleration - set/display acceleration rate
	Index - set/display index distance
	Jog - set/display jog speed
	Home - set/display homing speed
	Terminators
	<cr> carriage return enters value as parameter
	? display last entered value
	! display current value
	% continuously display current value

The examples below assume that the PMC is in its powerup (default) mode. For alternate parameter ranges, see Section 6.3.

Examples
V300<cr> - set the maximum velocity to 30,000 counts/second
A3000<cr> - set the acceleration rate to 3000 counts/second per millisecond

With a 10,000 count per revolution position encoder, the above two commands set up a speed of three revs/sec (180 RPM), with an acceleration time of 10 milliseconds.

It is possible to specify a set of motion parameters for acceleration rate, top speed, and index distance that are inconsistent. i.e. The specified index distance is not long enough to reach the specified top speed using the specified acceleration rate. If this happens, the PMC will temporarily reduce the top speed enough to perform the desired position change using the specified acceleration rate. This temporary top speed value is not stored in the motion buffer but only used for the following index or series of identical indexes. The algorithm used by the PMC is very efficient for minor reductions of top speed and less efficient for very large top speed reductions. This algorithm takes between 3 msec and 50 msec of computation time dependent on the amount of the top speed reduction required, and so faster operation is possible when the specified motion parameters are consistent.

The Acceleration and Velocity commands are used only to specify motion parameters, however the Index, Jog, Home and absolute Go commands are used to start MOTION as well as specify parameters.

MOTION

M O T I O N

Commands

Index	- move relative to the current position
Go	- move to the absolute position
Jog	- move at the jog speed until told to stop
Home	- move to the encoder reference or sensor

Terminators (will enter argument)

+ (plus sign)	- start motion in positive direction
- (minus sign)	- start motion in negative direction
* (asterisk)	- stop motion

Examples

I500+	Move 500 counts positive relative to the current position; The motion actually starts when the + is received.
I-	Move negative from the current position by the distance currently stored in the motion buffer
G9306-<cr>	Go to absolute location -9306; The motion actually starts when the <cr> is received. Calculations to set up the motion are initiated when the sign of the absolute location (+ or -) is received.
I*	Stop current motion
J95+	Jog at 9500 encoder counts/sec in the positive direction
H1-	Home at 100 counts/sec in the negative direction
J-	Jog in the negative direction at the rate currently commanded jog rate

2.3.2 SYNCHRONIZATION AND INTERFACE Commands and Terminators

SYNCHRONIZATION & INTERFACE	
Commands	
Delay	- delay specified time or distance
Until	- wait until specified input condition is true
Output	- set Machine I/O output condition
Terminators	
; (semi-colon)	- wait for motion to reach steady state speed
: (colon)	- wait for motion to begin deceleration
, (comma)	- wait for motion to stop

Synchronization Characters

The fact that MPL program execution is independent of the motion it creates is a powerful feature in that it allows Machine I/O to be manipulated or successive motions to be set up while motion is taking place. Synchronization capability is provided to allow MPL program execution to synchronize with the motion being created.

<;> The <;> character can be used with the Go, Jog, Home, Index, Delay and Output commands to synchronize them with the start of constant velocity. e.g. The command I+;<cr> starts a positive index and then delays until the acceleration is complete before executing the next MPL command.

<:> The <:> character can be used with the Go, Jog, Home, Index, Delay, and Output commands to synchronize them with the start of decel of the motion in progress.

<,> The <,> character can be used with the Index, Go, Jog, Home, Delay, Output and Normalize commands to synchronize them with the completion of motion which may be in progress.

Examples

D1000	Delay 1000 milliseconds (1 sec)
D,	Delay until motion is complete
U9	Wait until inputs IN1' and IN8' are asserted
D100;	Delay until 100 msec after acceleration is complete
O1	Turn on Output OUT1'

2.3.3 PROGRAM Commands and Terminators

PROGRAM commands and terminators are used to manipulate the **Program Buffer** or **Control** the flow of program execution. The Program Buffer is memory that contains labeled groups of MPL motion control commands (called programs or routines) which will be executed sequentially at a future time. These commands are entered or edited using the Program command, with the editing functions used during the Program command detailed in Section 6.4. They are executed later by using the Branch command to tell the PMC which program label to begin.

PROGRAM

BUFFER

Commands

Program	- enter, edit, or examine motion programs
@<label>	- mark program label
<esc> (escape)	- exit program mode
<space>	- stop motion and exit program mode

Examples

P{	Enter programming mode with the cursor at the beginning of the Program Buffer
PA	Enter programming mode with the cursor at the beginning of the "A routine"
P<cr>	Enter programming mode with the cursor at the end of the Program Buffer
P?	Examine the Program Buffer from the beginning, a line at a time; Changes are not allowed.

PROGRAM

CONTROL

Commands

Branch	- conditional jump to a program label
Loop	- jump to program label a specified number of times
Function	- conditional call to a labeled subroutine
Exit	- conditional return from subroutine or program

Examples

BA	Branch unconditionally to the "A routine"
BB2	Branch to the B routine if machine input IN2' is asserted
LA100	Transfer program execution (Loop) to label A 100 times
FB	Call function B; Program execution will resume at the next MPL statement upon completion of the function.
E	Unconditionally exit (end) a function
E5	Exit a function if inputs IN4' and IN1' are asserted

SYSTEM SET-UP OR DISPLAY

Commands

Normalize	- software reset, set absolute position
Set or Show (B)	- set or display baud rate
Show (C)	- display condition of machine inputs
Set or Show (L)	- set or display software overtravel limits
Set or Show (M)	- set or display motor control mode
Show (P)	- show last program label passed
Show (S)	- display system status
Set or Show (T)	- turn on or off program Trace
Set or Show (W)	- turn on or off program Write Enable
Set (X,Y,Z)	- set or display control/communications status
= Assign Axis-ID	- assign axis identifier
Tuning	- set servo loop gain and compensation adjustments

Terminators

? (question mark)	- report parameter value
! (exclamation mark)	- report system status

Examples

N*	Normalize the PMC (restarts the firmware)
N0-	Normalize the absolute position counter to 0
N3795+	Normalize the absolute position counter to +3795
N<cr>	Normalize the serial communications (enable selected baud rate)
SB1<cr>	Set the PMC baud rate to 38.4k Baud
SX8	Set the X Register to 8_H (Bit pattern 0000 1000)
SX0B	Set the X register to $0B_H$ (Bit pattern 0000 1011) which transposes the meaning of + and - direction, enables machine inputs IN4' and IN8' to be used as overtravel limits, and selects a top velocity of 192,000 counts/sec.
SY40	Set the Y register to 40_H (Bit pattern 0100 0000) to select s-curve acceleration
TP10	Set position loop gain to 10.
T?	Display current tuning parameters.
TE?	Display the normalization error.

2.3.4 Error Codes

The PMC motion programming language checks for errors at execution time, and when one is found, an error message is generated.

ERROR CODES

Format: <bell> <space> <space> # <error code>

Error codes

A - Syntax error (invalid argument)
B - Motion error
C - Programming error (while editing or running)
D - Miscellaneous error

Note: The D2 error for input aborted (using the ESCAPE) does not have the <bell> <space> <space>. Also, in non-echo mode all errors have the format:

<error code>

See Section 6.12 for detailed information on PMC error codes.

SPECIFICATIONS

3.1 GENERAL SPECIFICATIONS

CPU	type	8085A
	speed	3.072 MHz

Motion Control Program Storage

2k bytes non-volatile RAM	PMC-903
8k bytes non-volatile RAM	PMC-904

Position loop

position error	±2047 counts max
positioning speed	384,000 counts/sec max

compensator break frequency ranges

velocity loop	1.5 to 50 Hz
position loop	0.5 to 25 Hz

3.2 SYSTEM AXIS INTERFACE SPECIFICATIONS

Power Supply Requirements

+5VDC	0.9 typical, 1.2 Amp max
±12VDC	0.15 Amp max

Digital Inputs

STOP' (TTL Input)	logical 0	Vin < 0.8 VDC Iin < -1 mA
	logical 1	Vin > 2.8 VDC Iin < 400 uA
	acceptance time	4.5 msec minimum
RESET' (NMOS Input)	logical 0	Vin < 0.8 VDC Iin < -1 mAa
	logical 1	Vin > 2.8 VDC Iin < 400 uA

Digital Input/Output

EXTREF EXTREF' (Input/Output of U28 (SN75174N) and U29 (SN74175N)

logical 0	Vin < 0.8 VDC Iin < -1 mA
logical 1	Vin > 2.8 VDC Iin < 400 uA
pulse acceptance time	1.3 us minimum
input frequency	192 kHz maximum

3.3 SERIAL COMMUNICATIONS INTERFACE SPECIFICATIONS

Standards

EIA RS-232C
RS-422

data bits	8
start bits	1
stop bits	1
parity	none

Note that the PMC can use all 8 data bits for communications, however normal ASCII terminal communications require only 7 data bits. The eighth and most significant data bit is used for binary communications with computers, and also as an informational bit in the "}" character of each "prompt" to indicate whether the servodrive is enabled or not. **For other than advanced PMC communications, terminal devices should be configured for 7 data bits.**

Baud Rates

As initially configured from the factory, the PMC will automatically set its baud rate to one of the following baud rates:

38.4K 19.2K 9600 4800 2400 1200 600 300

To allow the PMC to perform its "autobaud" function, automatically determining the baud rate of the terminal or computer communicating with it, the user must send a series of "carriage returns" <cr> which is a $0D_H$. The PMC will start with its baud rate set to 38.4K Baud, and reduce it by a factor of 2 for each character received that is not recognized as a <cr>. This series must contain enough "carriage returns" to allow the PMC to reduce its baud rate to the baud rate of the user electronics, and the "carriage returns" must not be sent at a rate faster than one every 35 milliseconds, with the complete series sent within 2.1 seconds.

Baud rate may also be set using the SB command, as detailed in Section 6.9.15. When this is done, the "autobaud" sequence of carriage returns will no longer be required in order to communicate with the PMC. Instead, the user has the responsibility to communicate at the proper baud rate.

Strapping

Standard strapping is for RS-232C Data Communications Equipment (DCE) with no flow control. i.e. Most RS-232 ASCII terminals will communicate with this interface. Other strapping options are detailed in Appendix 8.5.

3.4 MOTOR LOOP INTERFACE SPECIFICATIONS

Position Encoder

Type — Dual channel quadrature with "square wave" outputs and an optional "once per revolution" reference pulse

Resolution — Four times "linecount"

Max Data Rate — 384 kHz This rate refers to the decoded pulse train, and for this rate the "A" and "B" quadrature encoder channels are operating at 96 kHz each.

Digital Inputs

ENCA ENCB ENCREF SENSIN	acceptance time	4 usec min for Vmax = 384 kHz 8 usec min for Vmax = 192 kHz 32 usec min for Vmax = 48 kHz	
	Single-ended	max range impedance switch point	0 V to +12 V 20K ohms min 2 V (standard)
	Differential	max range impedance switch point common mode	-12 V to +12 V 40K ohms min 0 V 10 V max

Analog Inputs

Tachometer	HVTACH	impedance voltage	338K ohms 150 VDC max
	LVTACH	impedance voltage	68K ohms 60 VDC max

For tach voltages greater than 150 VDC, use a resistor external to the PMC.

Velocity Command Input

EXVCMDIN	impedance	10.0K ohms

Digital Outputs

SDRVEN SDRVEN'	I.C. type	SN7404N TTL buffer
	max sink current	-5 mA
	max source current	400 uA
	low level output	0.4 volts typical
	high level output	4.0 volts typical

Analog Outputs

ICMD XVELREF	I.C. type	LM324N Op Amp
	Range	-10 V to +10 V
	Output Current	5 mA max
	Load Resistance	2K ohms minimum

3.5 MACHINE I/O INTERFACE SPECIFICATIONS

Digital Inputs

	Logical 0	Vin < 0.8 VDC Iin < -1 mA
	Logical 1	Vin > 2.8 VDC Iin < 400 uA
	Acceptance Time	4.5 msec minimum
	EXECUTE' Acceptance Time	1.0 msec minimum

Digital Outputs

I.C. type	SN7417N TTL buffer
max sink current	-16 mA
max leakage current	-0.25 mA
low level output voltage	0.4 volts typical
high level output voltage	4.0 volts typical

3.6 MECHANICAL AND ENVIRONMENTAL SPECIFICATIONS

Dimensions	14.5"x 6.9" x 0.8" max	
Weight	one pound max	
Temperature ranges		
	Operating	0 to +50 degrees C
	Storage	-25 to +125 degrees C
Relative humidity	(w/o condensation)	0 to 90%

INSTALLATION

4.1 INTRODUCTION

The PMC can be custom mounted in an enclosure of the user's design or in a chassis supplied by ORMEC. Mounting holes are provided at the corners of the printed circuit board as shown in Appendix 8.9, and the PMC comes with Card Ejectors designed for use with Bivar ECON O GUIDE card guides.

A typical PMC configuration is diagrammed in Appendix 8.1 and 8.2. The signals terminated on each connector are described in the following interface sections.

4.2 SYSTEM AXIS INTERFACE (TM2)

The system axis interface provides for connection of the power supply voltages as well as an external position frequency reference (EXREF) used for distance based motion, a RESET' input to reset the PMC, and a STOP' input to stop motion and terminate operation of motion control programs. Pin assignments are shown below and in Appendix 8.6. A removable Phoenix Type MSTB 1.5/10ST Terminal Block or equivalent will mate with TM2, and one is included with each unit.

Pin#	Signal Name	Description
1	EXREF	EXREF is an RS-422 driver/receiver tri-statable digital output/input for use as an external position reference for distance based motion commands. (See Section 6.8 for a discussion of how to use this signal.)
2	EXREF'	EXREF' is the logical complement of EXREF.
3	COMMON	Power supply common
4	+12 VDC	+12 VDC power supply
5	-12 VDC	-12 VDC power supply
6	+5 VDC	+5 VDC power supply
7	SHIELD	EMI isolated SHIELD is an interconnection point for cable shields provided to reduce signal noise. It is recommended that the user connect the shields, at one point, to chassis ground.
8	POLARIZE	This pin is omitted from TM2 to allow polarization of the matching terminal board (TB2).
9	STOP'	Stop the current motion and break the current motion control program
10	RESET'	Reset the PMC system, including the on board microprocessor

4.3 SERIAL COMMUNICATIONS INTERFACE (JM2)

A 25 pin female "D-Subminiature" connector will mate with connector JM2. A typical installation of the serial communications interface is shown in Appendix 8.1. The complete connector pin assignments are shown in Appendix 8.5. Data, parity, start and stop bit specifications are included in the SPECIFICATIONS Section 3.3.

As can be seen in the Appendix, by strapping header connector J5, this interface is configurable to be compatible with EIA Standards RS-232, RS-422/449, and RS-423/449. Further, it can be strapped to be either a DCE or a DTE device for both RS-232 and RS-449. More detailed information about this interface is found in the SPECIFICATIONS Section 3.3.

It is shipped from the factory strapped for RS-232 DCE with no handshake to conveniently work with a standard terminal.

4.4 MOTOR LOOP INTERFACE (TM1)

The motor loop interface is provided for interconnection to the servodrive, the loop contactor, the DC tachometer, the position encoder and a machine sensor. A Phoenix Type MSTB 1.5/21ST Terminal Strip or equivalent will mate with TM1. The connector pin assignments are shown below and in Appendix 8.1.

Note that the interfaces to the encoder and the sensor input are factory configured for either single ended or differential inputs of up to 24 vdc, with a nominal switching point of 2 vdc. The switching threshhold is controlled by the value of SIP Resistor network RN3, which is socketed for possible field change. The schematic for this is located in Appendix 8.6.

Pin#	Signal Name	Description
1	ICMD	ICMD (Current Command) is an analog output, with a range of ± 10 VDC, which the PMC uses to command servomotor current (torque).
2	COMMON	Power and logic signal common
3	SDRVEN	SDRVEN (Servodrive Enable) is a 0 to 5 VDC digital output signal used to control a solid state relay which enables power to the servodrive and/or a loop contactor. This signal or its complement may also be connected to an inhibit input of some servodrives.
4	SDRVEN'	SDRVEN' is a 0 to 5 VDC digital output which is the logical complement of SDRVEN.
5	SHIELD	EMI isolated SHIELD is an interconnection point for cable shields provided to reduce signal noise. It is recommended that the user connect the shields, at one point, to chassis ground.
6	POLARIZE	This pin is omitted from TM1 to allow polarization of the matching terminal board (TB1).
7	HVTACH	This analog input is for connecting the D.C. Tachometer when the maximum anticipated tachometer voltage is between 60 and 150 VDC. This signal should be negative for "forward" motion of the servomotor.
8	LVTACH	This analog input is for connecting the D.C. Tachometer when the maximum anticipated tachometer voltage is between 0 and 60 VDC. This signal should be negative for "forward" motion of the servomotor.

9	COMMON	Power and logic signal common
10	+5 VDC	The +5 VDC power supply used for the PMC is provided at this terminal for optionally powering the incremental position encoder.
11	ENCA	ENCA (Encoder "A") is the incremental position encoder "Channel A" (one of two quadrature square wave signals). It is a digital input signal with a "low" level of 0 VDC and a "high" level of between +5 VDC and +24 VDC.
12	ENCA'	ENCA' is the logical complement of ENCA, and is present only when the incremental position encoder used has differential outputs. If single ended encoders are used this signal may be either left open or grounded.
13	ENCB	ENCB (Encoder "B") is the incremental position encoder "Channel B" (one of two quadrature square wave signals). It is a digital input signal with a "low" level of 0 VDC and a "high" level of between +5 VDC and +24 VDC.
14	ENCB'	ENCB' is the logical complement of ENCB, and is present only when the incremental position encoder used has differential outputs. If single ended encoders are used this signal may be either left open or grounded.
15	ENCR	ENCR (Encoder R) is the incremental position encoder "Channel R" (usually a "once per revolution" reference signal). It is a digital input signal with a "low" level of 0 VDC and a "high" level of between +5 VDC and +24 VDC.
16	ENCR'	ENCR' is the logical complement of ENCR, and is present only when the incremental position encoder used has differential outputs. If single ended encoders are used this signal may be either left open or grounded.
17	SHIELD	EMI isolated SHIELD is an interconnection point for cable shields provided to reduce signal noise. It is recommended that the user connect the shields, at one point, to chassis ground.
18	SENSIN	SENSIN (Sensor Input) is a machine sensor signal which may be used to synchronize motion to an external event. When a motion is primed to start, begin deceleration or stop on this signal, the signal has immediate effect (less than two encoder counts).
19	SENSIN'	SENSIN' is the logical complement of SENSIN, and is present only when the machine sensor used has differential outputs. If a single ended sensor is used this signal may be either left open or grounded.
20	EXVCMDIN	EXVCMDIN (External Velocity Command Input) is an analog input to the velocity summing junction of a PMC. Strapping for it is on Header J7 and is shown in Appendix 8.6. Default strapping routes this input directly to the velocity summing junction. Alternate strapping routes it through an adjustable gain stage (FFGAIN) shown in Appendix 8.4. Note that when it goes through the adjustable gain stage, this signal must have a positive polarity, with a maximum voltage of +2.5 VDC.
21	XVELREF	XVELREF (External Velocity Reference) is a bipolar analog output which is proportional to the commanded velocity of the PMC. The gain of this signal is independently software selectable using the TX command. Strapping for this signal on Header J7 allows either polarity of the output as shown in Appendix 8.4.

Note that for proper operation of any feedback control system, the system must be interconnected so that the feedback is negative. i.e. The feedback signal should cancel the effect of the reference input signal, not reinforce it. The direction that you choose for the motor to turn for "positive" rotation is in up to you, but after that choice is made, several signal polarities must be consistent for proper operation. There is an option available to transpose the meaning of the "+" and "-" directions by setting Bit 1 of the X Register, but the following discussion assumes that the X Register, Bit 1 is set to "0", as it is on powerup. Follow the following steps to set up the system.

1. Select the direction of motor rotation to be considered positive.

2. The D.C. tachometer signal (TM1-7 or TM1-8) must be negative for positive motor rotation.

3. The motor armature leads must be connected for negative velocity loop feedback given conditions 1 and 2 above. i.e. When power is applied with the position loop disabled, and a small velocity commanded by the PMC, the motor must proceed slowly, and not "run away" in one direction or the other. If the motor "runs away" at high speed, reverse the connections to the motor armature leads.

4. The incremental position encoder channels ENCA and ENCB must be connected for negative position feedback. For negative position feedback, the signal ENCA will "lead" ENCB by 90° for forward rotation. When the position loop is enabled, and no position reference pulse trains are present, the motor should remain at rest. If the motor does not remain at rest, reverse the ENCA and ENCB channels. If you are using differential output encoders, don't forget to also reverse ENCA' and ENCB' as well. See Appendix 8.8 for a graphic representation of these signals.

5. Refer to the GETTING STARTED Section for instructions on powering up a PMC based motion control system.

4.5 MACHINE I/O INTERFACE (JM1)

The Machine I/O Interface is provided to allow the PMC to interface with the machine or other systems using 16 discrete digital I/O points. This interface is compatible with Opto-22 type 16 position I/O racks and modules, and the polarities used were selected for logical compatibility with this type of interface. There are 11 inputs, 8 general purpose and 3 special purpose. There are 5 outputs, 4 general purpose and one special purpose.

Standard uses of the Machine I/O, as can be seen from the table below, are to handle overtravel limit switches, a hardware STOP command, and to indicate whether or not the PMC is ready to accept a new command. In addition, user programmable inputs and outputs are provided. The MOTION ROUTINE ADDRESS inputs allow interface with equipment such as Programmable Controllers, by allowing them to access previously stored MPL Routines using standard I/O points.

A Berg 34 pin mass termination flat cable connector (P/N 6690-234) or equivalent will mate with JM1. The connector pin assignments are shown in Appendix 8.7.

MACHINE I/O INTERFACE (JM1)

I/O #	Pin #	DRIVER	I/O POINT	FUNCTION
	33	+5 VDC		interface power
0	31	U48	OUT1'	four general purpose outputs
1	29	U48	OUT2'	
2	27	U48	OUT4' (FAULT')	
3	25	U48	OUT8' (IN MOTION')	
4	23	-	IN1'	four general purpose inputs
5	21	-	IN2'	
6	19	-	IN4' (-LIMIT)	
7	17	-	IN8' (+LIMIT)	
8	15	-	IN10' (ADR1')	four more general purpose inputs or motion address lines
9	13	-	IN20' (ADR2')	
10	11	-	IN40' (ADR4')	
11	9	-	IN80' (ADR8')	
12	7	-	SEL (ADR10')	motion address
13	5	-	EXECUTE'	MIO Interface handshake lines
14	3	U48	READY'	
15	-	-	STOP'	

All even pin numbers are grounded.

Signal Name — **Description**

OUT1' - OUT8' — These discrete digital outputs are controlled by the Output command. When asserted, the outputs will be at 0 volts (TTL low), causing Opto-22 compatible modules to conduct current. They may also be read from the Serial Communications interface using the O? command.

IN1' - IN80' — These discrete digital inputs can be used by the conditional Branch, Exit, Function and Until commands to control MPL program flow. They will be at 0 volts when asserted (voltage applied to the Opto-22 compatible module). They may also be read from the Serial Communications Interface using the SC! command, or polled from the Serial Communications Interface, even while an MPL program is being executed from the program buffer, using SYSTEM STATUS POLLING.

READY' — The READY' output will be asserted (TTL low) whenever the PMC is ready for a command to be entered at the SCI or for a new motion routine to be run. It will not be asserted while a command or program is executing. It is used as a "handshake" line when executing MPL programs from the Machine I/O interface.

SEL' — The SEL' input is used to select either MPL routine "0" or "1" for execution when the EXECUTE' input is asserted. If asserted, (TTL low) at the PMC input, the "1" routine will be executed; otherwise the "0" routine will be executed.

STOP' — Asserting this input will stop system motion using a pre-programmed STOP rate, and also stop any MPL program which may be executing, causing the PMC to unconditionally return to the READY state.

FAULT' — OUT4' can be programmed to automatically assert whenever a "fault" occurs by setting Bit 2 of the Z Register to 1. This FAULT' output will be cleared by the start of the next MPL command.

IN MOTION' — OUT8' can be programmed to automatically assert whenever the PMC is commanding motion by setting Bit 3 of the Z Register. It will also clear automatically when the commanded motion is complete.

-LIMIT & +LIMIT — IN4' and IN8' can be programmed to automatically operate as hardware overtravel limits by setting Bit 1 of the X Register. Hardware overtravel limit switches can be configured to operate (impede overtravel motion) when either asserted (TTL low) or not asserted (TTL high), using the **SLH** command. **SLH0** specifies that the overtravel limit is asserted when the input is a TTL low, and **SLH1** specifies that the overtravel limit is asserted when the input is a TTL high.

ADR1' - ADR10' — The IN10'-IN80' and SEL' digital inputs can be configured to allow the PMC to access up to 32 MPL routines from the Machine I/O interface. This alternative operation is accomplished by clearing Bit 4 of the Z Register. These signals should then be considered "logical address lines" ADR1'-ADR10', allowing the user to select and execute 32 different MPL routines from hardware. The motion routine labels accessible from this hardware addressing scheme are listed in the following table of *Hardware Accessible Motion Routines*. To run one of the specified routines, place the appropriate signals on these address lines as indicated in the table, and when the PMC is "ready", as indicated by the READY' output, assert the EXECUTE' input. General purpose inputs IN10'-IN80' continue to be accessible from MPL, if useful.

EXECUTE' — Strobing this discrete digital input signal to a TTL low will start execution of a pre-programmed motion routine. The routine executed is determined by either the SEL' input or the combination of the ADR1'-ADR10' address inputs. If the READY' output is not asserted by the PMC, then the EXECUTE' input will be ignored.

The following table details the hardware address requirements for selection and execution of MPL routines from the Machine I/O interface. The table documents the signals required at connector JM1 for programs labeled 0 & 1, or @ through _, including the alphabet (in capitals). Connect 1, 2, 3, 4 or 5 Motion Routine Address lines as shown below depending on the number of motion routines you want to access from hardware for your particular application.

HARDWARE ACCESSIBLE MOTION ROUTINES

Program Label	Pin Numbers at Connector JM1 7	9	11	13	15	MIS-200 Code (hex)
	$\overline{SEL}$					
0	1	-	-	-	-	0-
1	0	-	-	-	-	1-
-- 1 address line * --						
	$\overline{ADR10}$	$\overline{ADR8}$	$\overline{ADR4}$	$\overline{ADR2}$	$\overline{ADR1}$	
a	1	1	1	1	1	00
A	1	1	1	1	0	01
1 address line						
B	1	1	1	0	1	02
C	1	1	1	0	0	03
2 address lines						
D	1	1	0	1	1	04
E	1	1	0	1	0	05
F	1	1	0	0	1	06
G	1	1	0	0	0	07
3 address lines						
H	1	0	1	1	1	08
I	1	0	1	1	0	09
J	1	0	1	0	1	0A
K	1	0	1	0	0	0B
L	1	0	0	1	1	0C
M	1	0	0	1	0	0D
N	1	0	0	0	1	0E
O	1	0	0	0	0	0F
4 address lines						
P	0	1	1	1	1	10
Q	0	1	1	1	0	11
R	0	1	1	0	1	12
S	0	1	1	0	0	13
T	0	1	0	1	1	14
U	0	1	0	1	0	15
V	0	1	0	0	1	16
W	0	1	0	0	0	17
X	0	0	1	1	1	18
Y	0	0	1	1	0	19
Z	0	0	1	0	1	1A
[	0	0	1	0	0	1B
\	0	0	0	1	1	1C
]	0	0	0	1	0	1D
^	0	0	0	0	1	1E
_	0	0	0	0	0	1F
5 address lines						

"1" => high TTL level "0" => low TTL level "-" => don't care

* Only one address line is used with Z Register Bit 4 set; see Section 6.9.15e.

BCD Thumbwheel Switch Configuration

The following table can be used for selecting ten motion control routines from a standard single digit binary coded decimal thumbwheel switch. The second table outlines the configurations and wiring diagram for an implementation with a common thumbwheel switch.

Program Label	Motion Routine Address				
	ADR10	ADR8	ADR4	ADR2	ADR1
JM2-	7	9	11	13	15
P	0	1	1	1	1
Q	0	1	1	1	0
R	0	1	1	0	1
S	0	1	1	0	0
T	0	1	0	1	1
U	0	1	0	1	0
V	0	1	0	0	1
W	0	1	0	0	0
X	0	0	1	1	1
Y	0	0	1	1	0

1 => high TTL level 0 => low TTL level

PMC-903, PMC-904

	ADR1	ADR2	ADR4	ADR8	COM	ADR10
JM2-	15	13	11	9	8	7
WHEEL	1	2	4	8	C	
0					x	
1	x				x	
2		x			x	
3	x	x			x	
4			x		x	
5	x		x		x	
6		x	x		x	
7	x	x	x		x	
8				x	x	
9	x			x	x	

x - indicates signal connected to Common

C & K Components, Inc.
Thumbwheel Switch
Section Types: 21,27,31

4.6 PMC CONFIGURATION JUMPERS

The PMC has five header connectors which provide user selectable hardware options. These options are selected by the placement of "push-on" jumpers on the header connectors. All of the configuration headers are described in this section.

4.6.1 Assigning Axis-ID (Header J6)

Axis-IDs are assigned to PMCs to allow multiple units to communicate with a host computer or programming terminal through a single serial communications interface. This communications is accomplished using a "serial bus" technique, which connects all the PMCs on a "party line".

To use a serial bus, the PMCs must each have a unique Axis-ID assigned and they must have their Serial Communications Interface configured for RS-422. For systems with up to 14 PMCs on a serial communications bus, the Axis-ID is normally assigned by the placement of Jumper Straps on Header J6. For systems with 15-32 PMCs on a single serial bus, the additional Axis-IDs must be assigned in non-volatile RAM. Serial bus communications operation is described in OPERATION Section 6.7.

Default Jumper Strap Positions for Header J6

1-2 3-4 5-6 7-8

With these jumpers in place, the PMC will check at powerup to see if an Axis-ID has been previously set in non-volatile RAM. If no Axis-ID is found, the PMC will operate in its "default" serial communications mode as initially shipped from the factory and described in Section 5.1. If an Axis-ID has been previously specified, serial bus communications mode will be enabled, and the PMC will assume the "unselected" state. Refer to OPERATION Section 6.7.

If all of these straps are removed, the Serial Bus Communications Mode will be disabled, even though an Axis-ID may have been previously stored in non-volatile RAM. If some, but not all, of the straps are installed, the PMC will use the appropriate Axis-ID as detailed in the table below, enable serial bus communications, and assume the "unselected" state.

Header J6 Jumper Strap Positions				Axis-ID
7-8	5-6	3-4	1-2	
-	-	-	-	Disables Serial Bus
-	-	-	*	A
-	-	*	-	B
-	-	*	*	C
-	*	-	-	D
-	*	-	*	E
-	*	*	-	F
-	*	*	*	G
*	-	-	-	H
*	-	-	*	I
*	-	*	-	J
*	-	*	*	K
*	*	-	-	L
*	*	-	*	M
*	*	*	-	N
*	*	*	*	Assigned from non-volatile memory

- Indicates no Strap Present * Indicates Strap Present

4.6.2 Serial Communications Hardware Configuration (Header J5)

The Serial Communications Interface can be conveniently configured for RS-232, RS-422 or RS-423 using jumper straps on Header J5. The PMC can also be easily configured as either a DCE (Data Communications Equipment) or DTE (Data Terminal Equipment) device. Lastly, the PMC's hardware handshake interface lines can be disabled for communications to a "dumb" terminal lacking hardware handshaking capability. See Appendix 8.5 for the "PMC Communications Interface" schematic and a pictorial of Header J5 with various jumper strap configurations.

4.6.2.1 RS-232 DCE Communications (Default)

Default Jumper Strap Positions for Header J5

7-8 9-10 15-16 17-18 19-21 28-30 29-31

This serial communications selection is suitable for:

1) any programming terminal supplied by ORMEC
2) IBM-PCs or compatibles using ORMEC's development software (MAX)
3) most RS-232 terminals

4.6.2.2 RS-422 DCE Communications

Jumper Strap Positions for Header J5

11-12 13-14 21-22 23-24 25-26 27-28 31-32 33-34

If an you are using an ORMEC Serial Bus Interface module (SBI) to convert RS-232 to RS-422 for serial bus communications, it is possible to supply power to the SBI from your PMC. Install additional jumpers as follow:

Additional Jumpers for Supplying Power to ORMEC Serial Bus Interface Modules

1-2 3-4 5-6 19-20.

This serial communications selection is suitable for:

1) ORMEC's SBI-910-RS or SBI-911-SS (either is recommended)
2) ORMEC's SBI-900-RC or SBI-911-SC (with standard configuration)
3) other RS-422 communications devices

4.6.2.3 RS-232 DCE (No Handshake) Communications

Jumper Strap Positions for Header J5

6-8 10-12 15-16 17-18 19-21 28-30 29-31

This serial communications selection is suitable for:

1) RS-232 terminals which do not have "hardware handshaking"

4.6.2.4 Other Communications Configurations

To convert the RS-232 DCE configurations shown above to RS-232 DTE, remove the jumpers from pins 15-16 & 17-18 and replace them on pins 15-17 & 16-18.

The PMC can be configured for RS-422 DTE by placing the jumper straps as shown below:

11-13 12-14 21-23 22-24 25-27 26-28 31-33 32-34

Header J5 also allows other communications configurations, and for a detailed schematic, consult Appendix 8.5.

4.6.3 Feedforward and External Velocity Reference Selection (Header J7)

Default Jumper Strap Positions for Header J7
1-2 3-4 6-8

The jumpers from 1-2 and 3-4 may be removed, and a jumper placed from 2-4 to allow an external signal to provide the feedforward reference. The external signal should have a range of 0 to +2.5 volts maximum. This option should be chosen when the system is to accurately track an externally provided motion reference in the "motion bus slave" mode. The schematic for this header is shown in Appendix 8.6.

The jumper from 6-8 may be removed and placed from pin 7-8 to change the polarity of the external velocity reference signal. The schematic for this circuitry is shown in Appendix 8.4.

4.6.4 Daughter Board Connector (Header JM4)

Default Jumper Strap Positions for Header JM4
25-26

The jumper strap above must be in place unless there is a daughter board added to the PMC.

4.6.5 Motion Reference Bus Selection (Header JM20)

Default Jumper Strap Positions for Header JM20
1-2 4-5

The schematic for this header is documented in Appendix 8.7.

The jumper strap from pin 1-2 is strapping the motion reference bus receiver output to the internal circuitry of the PMC. If an EBC-900 daughter board is added to the PMC for applications where the PMC is referencing its motion to a remote encoder, then the jumper from pin 1-2 must be removed so that the EBC can provide the EXTREF signal for the internal circuitry.

The jumper strap from pin 4-5 is strapping the internal position reference command pulses to the motion reference bus driver IC (U29C). In some cases, usually when an EBC is used with the PMC, this jumper strap may be moved from 3-4. This change straps the internal reference pulses (provided by the on-board EBC) to the motion reference bus driver IC.

GETTING STARTED

5.1 POWERING UP AND ESTABLISHING COMMUNICATIONS

A typical PMC implementation is found in Appendix 8.1 and INSTALLATION instructions are found in Section 4. For the initial part of this section, it is not necessary to have the servodrive and servomotor attached to the PMC and, if they are attached, it is recommended that power not be applied to the servodrive until you are ready to start the tuning section. At that point, **and before the power is applied to the servodrive,** the motor should be fastened down securely. The first time tuning is attempted it is better if the mechanical load is not attached. For optimum response, it will have to be tuned again after the load is attached.

It is recommended that you interface an IBM-PC or compatible running ORMEC's development software (MAX). This software allows terminal emulation for communicating with one or more PMCs and also Upload and Download functions for storing and retrieving MPL programs on diskettes. Should an IBM-PC or compatible not be available, most ASCII serial terminals can be used.

Attach your IBM-PC or ASCII terminal to the Serial Communications Interface at connector JM2 to learn the PMC Motion Programming Language (MPL) and set up the servo gains and compensators. The PMC comes from the factory with the Serial Communications Interface configured as an RS-232C DCE, and it therefore should be compatible with most ASCII RS-232C terminals. Your terminal must be configured for one of the following baud rates: 38.4k, 19.2k, 9600, 4800, 2400, 1200, 600 or 300 baud. For more detail on the Serial Communications Interface, see the SPECIFICATIONS Section.

After attaching your terminal to connector JM2, provide power to the PMC.
then power is applied to the PMC, the on-board microprocessor first performs built-in diagnostics as indicated by the flashing two color LED indicator. After approximately 2 seconds of diagnostics, the LED should be yellow (actually flashing between red and green every four milliseconds). See the MAINTENANCE Section for details.

Upon successful completion of the diagnostics, it searches through the MPL program memory for a powerup routine. Assuming no powerup program was encountered in the user programmable non-volatile memory, the PMC is now in the IDLE Mode (0), and all parameters are at their default values.

Sending a series of "carriage returns" to the PMC allows it to automatically determine your computer or terminal's baud (communications) rate.

The PMC automatically determines the baud rate by starting with its serial interface configured for the fastest baud rate, and if a character is received which is not a valid carriage return, halving its baud rate. When the next character is received, it is examined by the PMC and, if it is not a valid carriage return, the PMC again halves its baud rate. This may be repeated up to eight times allowing baud rates from 38.4k to 300. When the proper number of carriage returns are sent so that the PMC's and host's baud rates match, communications is established. A time of 2.1 seconds is allowed for the sequence of up to 8 tries.

To initiate communications with the PMC, you must provide enough carriage returns for the PMC to determine your baud rate. They must be sent at least 35 milliseconds apart, and within a 2.1 second total time period. There is more information on the Serial Communications Interface in the SPECIFICATIONS and INSTALLATION Sections. **If your PMC does not respond to carriage returns as it should, hold the STOP' line (on TM2) to ground during powerup and try again.**

Once the PMC determines your baud rate, it will respond with the version code for its firmware,

followed by a "prompt" of =}. At this point, the PMC is operating in its interactive mode, and is ready to accept commands. For an in-depth description of OPERATION, see Section 6.

5.2 SAMPLE MPL ROUTINES

There are four sample Motion Programming Language (MPL) routines below (and in your PMC's non-volatile memory) to assist you in getting started. The routines are labeled "R" (Report), "P" (Powerup), "H" (Home) and "T" (Tuning).

The **Report** routine will be used later to report position error enabling you to tune the velocity feedforward gain, and then modified and used to provide a report of position error during a motion.

The Powerup routine in the PMC is labelled "@P_Powerup", which will **not** execute automatically at powerup. **To make it run automatically at powerup, it must be edited to replace the "@P" program label with "@@".** This particular routine starts by initializing the loop gains, compensators and feedforward gain, then selects "S-curve" acceleration and position mode (enabling the servodrive). It delays 100 milliseconds for the servodrive to power up and balances the velocity loop with the Normalization command, before calling the Home routine. When the Home routine finishes by executing its Exit statement, the Powerup routine Exits, putting the PMC in its interactive mode.

The **Home** routine included finds a unique angular position of the motor by performing a Home command, which causes the motor to run in the negative direction until the encoder reference (called encoder marker by some encoder vendors) is found, Delaying until the Homing motion stops, and then moving 125 counts in the positive direction. It then delays 50 milliseconds for the motion to settle and normalizes the absolute position counter to zero before exiting.

The fourth routine is the **Tuning** routine. It causes indexing motions in the forward and reverse directions, separated by 300 millisecond delays. It then "Loops" back to label T three times, for a total of 4 indexes in each direction. Upon completion of the 4 pairs of moves, the velocity loop gain is raised by a value of 1, and the 4 indexes are repeated. This routine will therefore gradually raise your system's velocity loop gain while performing test moves (much like a signal generator). By attaching your oscilloscope to the tachometer signal and running routine T, you can watch your system's velocity response as the PMC raises the velocity loop gain. As the gain is raised, you should observe the system becoming more and more responsive. i.e. The tach signal will rise more and more quickly in response to the move command. When the tachometer response is acceptable, as detailed later, abort the Tuning routine by pressing the SPACE bar or the ESCAPE key.

Listing of the Sample Routines

MPL	Comments
@Report	@R marks the beginning of routine R; the rest is a comment
TE!	Report the position Error
TF1+	Tune the velocity Feedforward gain up by 1 increment
D300	Delay 300 milliseconds
LR49	Loop back to label "R" 49 times
E	Unconditionally Exit the R routine
@P_Powerup	@P marks the Powerup routine; replace the P with @ to activate
TV10	Tune the Velocity loop gain to 10 (range: 0-255)
TCV9	Tune the Velocity loop Compensator to 9 (range: 0-F_H)
TP4	Tune the Position loop gain to 4 (range: 0-255)
TF42	Tune the velocity Feedforward gain to 42 (range: 0-255)
TCPA	Tune the Position loop Compensator to A (range: 0-F_H)
SY40	Enable "S-curve" acceleration by Setting the Y-register to 40_H
SM2	Set the PMC in position mode (Mode 2)
D100	Delay 100 milliseconds for the servodrive to powerup
N0-	Normalize absolute position and "balance" velocity loop offset
FH	Unconditionally call Function H
E	Unconditionally Exit the powerup routine
@Home	@H marks the beginning of the "H" routine
H-	Run in the - direction at the default Home speed (2 kHz)
D,	Delay until the encoder reference is found and the motion stops
I125+	Index 125 counts positive (to the selected "Home" position)
D50,	Delay 50 milliseconds after the motion stops
N0-	Normalize the absolute position counter to "-0"
E	Unconditionally Exit the Home routine
@Tuning	@T marks the beginning of the Tuning routine
V100	Set the top velocity to 10,000 counts/sec (100 counts/tenth of a second)
A5000	Set the acceleration rate to 5000 cts/sec per msec (2 msec accel time)
I2000	Set the index distance to 2000 counts (202 msec index time)
I+	Index in the positive direction
D300,	Delay 300 milliseconds after the motion stops
I-	Index in the negative direction
D300,	Delay 300 milliseconds after the motion stops
LT3	Loop back to label T three (3) times
D1000	Delay 1 second (1000 milliseconds)
TV1+	Tune the Velocity loop gain up by 1
T?	Report the current Tuning Parameters
BT	Unconditionally Branch to label T

5.3 EDITING THE PROGRAM BUFFER

We are now going to use the programs listed above to gain some experience editing the program buffer. In the sequences listed below, **bold** print indicates the sequence that you type, and the regular print is the information sent by the PMC. See the OPERATION Section 6.4 or the Motion Programming Language brochure for a description of the editing commands.

MPL3.0b
=)P(

@Report Enter "Program mode" with the cursor at the beginning of the program buffer; The PMC responds by displaying the first line as shown.

If you have an IBM-PC or compatible with ORMEC's motion development software (MAX), you will be able to use the cursor positioning (arrow) keys, as well as the insert and delete keys for editing. If you have a regular terminal, the cursor positioning keys listed are used.

Review the program buffer a line at a time by typing successive linefeeds, usually LINEFEED or LF on most ASCII terminals. If you can't find a linefeed key, a linefeed can be typed by typing a "control J" (depress the CTRL key and while holding it, press the J key).

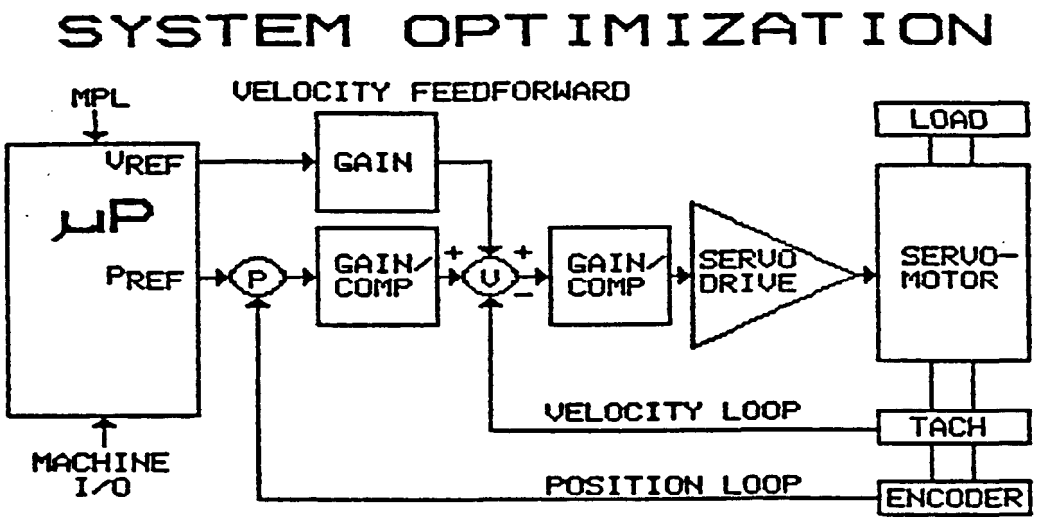
Type a linefeed **<lf>** to move the cursor down a line

TE! Type **<lf>**
.TF1+ Type **<lf>**
D300 Type **<tab>**; The cursor is now on the 3 and so you can overtype it. Overtype the 3 with a 2. You can also back up a character by typing a backspace <bs>. If you back up beyond the first character in a line the cursor will move back to the end of the previous line. The delete <del> key will move you up one line. In this way, you can move the cursor around the program buffer until you find the area you want to overtype, and then modify it. When finished editing, depress the ESCAPE **<esc>** key, to return to the interactive mode, signified by the =) "READY" prompt.

To edit the modified "Report" routine, type **PR**. Try out the rest of the editing commands found in Section 6.4 to familiarize yourself with them.

5.4 TUNING THE PMC

You are now ready to "tune" the servo loops of your PMC. To overview the analog circuitry of the PMC, see the Analog Architecture Drawing in Appendix 8.4 and the illustration below which shows the PMC gain and compensation adjustments available to the user to tune the system.



The recommended strategy for tuning the PMC is to perform the following adjustments in the order listed.

VELOCITY LOOP:	**Adjust the Velocity Loop Gain (VLGAIN)** **Adjust the Integral + Proportional Compensator (VLCOMP)**
POSITION LOOP:	**Adjust the Position Loop Gain (PLGAIN)** **Adjust the Velocity Loop Feedforward Gain (FFGAIN)** **Adjust the Integral + Proportional Compensator (PLCOMP)**

5.4.1 Adjusting the Velocity Loop Gain

This adjustment (VLGAIN) is provided to adjust the gain in the analog velocity loop closed by the DC tachometer. It is adjustable from a wide range of 1 to 255 (48 db) to be compatible with a variety of servodrives, servomotors and tachometers. The range of the DC tachometer voltage should have been determined, and the tach signal attached to either HVTACH (0 to ±150 VDC) or LVTACH (0 to ±60 VDC). See the INSTALLATION Section for details.

Your servomotor and servodrive must now be wired to the PMC, and power must be applied to the servodrive. **Before applying power to the servodrive**, the motor should be bolted down, and the first time this exercise is attempted it is better if the mechanical load is not attached. For optimum response, it will have to be tuned again after the load is attached. CAUTION: **When you apply power and enable the servodrive using the SM1 command below, your motor may start running at high speed because the velocity feedback is positive and not negative.** If this happens, refer to the INSTALLATION Section 4.4 to reverse the velocity feedback and remedy this situation.

The recommended approach to tune the velocity loop is to put the PMC in the Velocity Mode using the SM1<cr> command, adjusting the velocity loop feedforward gain with a TF100<cr> command so that the PMC can provide an analog velocity loop test signal.

For tuning the servo system, the PMC velocity parameter should be set for between one and five revolutions per second, and a commanded acceleration rate of five msec or less. Note that the Tuning routine sets the top velocity to 10,000 counts/second and the acceleration rate to 5,000 cts/sec per msec (2 msec accel time).

If your encoder has a linecount of 500, the system will have 2,000 counts per rev, and will be commanded at 5 revs/sec. If it has a linecount of 2500, the system will have 10,000 counts per rev, and will be commanded at 1 rev/sec. You should adjust the parameters in the Tuning program as appropriate for your individual system if it seems too fast or two slow. Also note that, since the feedforward gain is not yet calibrated, the arbitrary setting of TF100 may not be commanding a proper speed.. The times for the signal will be correct, however, and the feedforward gain can be adjusted as seems proper.

Attach an oscilloscope to the tachometer signal and then run the T Routine by typing BT<cr>. (A convenient filtered test point (TCH) is provided in the corner of the PMC near TM1. See Appendix 8.4 and 8.9) You can stop MPL operation and commanded motion at any time by depressing the SPACE bar. As the PMC raises the velocity loop gain, observe the tachometer signal and, when it overshoots or the system exhibits resonance by oscillating or "buzzing", terminate the operation of the motion control program by typing SPACE or ESCAPE. Note that the PMC reports its gain and compensation parameters during the tuning process, due to the program line T?. Once the routine is stopped you can type this command and receive a response as follows:

=)T? P=02 V=25 F=100 X=0 CP=00 CV=00

Since the gain is now too high, you should turn it down. This can be done by typing TV-, which turns it down by one value to V=24 or by directly setting it using the TV24<cr> command. Using a combination of these techniques, you may adjust it to the proper value. While adjusting VLGAIN, try "Indexing" or "Jogging" the system to determine the effect of your adjustment. If the "TV1+" in the Tuning routine is deleted then the routine will not adjust the gain, but only exercise the system. The velocity loop "rise time" (time to go from 10% to 90% of full value) for most servo systems should be between 1 and 15 msec, as long as the system is operating in "small signal" mode. To be operating in small signal mode, the commanded velocity must be small enough so that the motor voltage and current do not saturate (limit). Normally a speed of between one and five revs per second will not saturate the servo system.

5.4.2 Adjusting the Velocity Loop Integral + Proportional Compensator

Once VLGAIN is adjusted for no overshoot, but with a reasonably fast rise time (under 10 msec, and probably under 5 msec), VLCOMP should be adjusted. The object is to raise the value for VLCOMP to the maximum possible without adding overshoot to the tachometer response. This is done by editing the Tuning routine to change the TV1+ line to read TCV+. When this change is made, repeat the test above by typing BT<cr>. An example of what the response might look like follows:

=)T? P=02 V=25 F=20 X=0 CP=00 CV=0C

Adjustments to VLCOMP may also be made directly from the terminal, and "Jogging" or "Indexing" the system to determine the effect.

5.4.3 Adjusting the Position Loop Gain

This adjustment (PLGAIN) is provided to adjust the gain in the digital position loop closed by the incremental position encoder. It is adjustable from a wide range of 1 to 255 (48 db), to be compatible with a variety of servodrives, servomotors, tachometers and position encoder resolutions.

CAUTION: Your motor may start oscillating or running at high speed when the system is put in "position mode" using the SM2 command because the position feedback is positive and not negative. Refer to the INSTALLATION Section 4.4 to reverse the position feedback and remedy this situation.

Modify the Tuning routine line TCV+ to TP1+ to tune the position loop. Put the PMC in Position Mode by typing **SM2<cr>** and turn off the velocity loop feedforward gain by typing **TF0<cr>**.

With your oscilloscope still attached to the tachometer signal, run the Tuning routine. Keep watching the tach signal, and when it overshoots, terminate the operation of the Tuning routine by depressing the ESCAPE key. To determine the setting of PLGAIN that was achieved, type T?. The PMC will respond with the values for the gains and the compensators. An example of what the response might look like follows:

=)T? P=18 V=25 F=0 X=0 CP=00 CV=0C

Adjustments to PLGAIN may also be made directly from the terminal, and "Jogging" or "Indexing" the system to determine the effect. Once PLGAIN is adjusted to achieve a reasonably fast rise time (under 30 msec, and probably under 15 msec), FFGAIN can then be adjusted.

5.4.4 Adjusting the Velocity Feedforward Gain

Adjustments to the velocity reference feedforward gain (FFGAIN) should be made once the velocity loop gain and the position loop gain have been properly adjusted.

The adjustment procedure is to "Jog" the system in the Position mode (MODE 2) by typing **J+<cr>** and run the Report routine by typing **BR<cr>**. The Report routine as initially provided will cause the PMC to first report the position following error, and then raise its velocity feedforward gain (FFGAIN) by 1. This will be repeated every 300 milliseconds for up to 50 times. You will observe during this process that the position following error will be reduced each time FFGAIN is raised. If the following error goes negative, the system is leading the commanded position, because the feedforward gain is too high. When the following error gets near 0, the Report routine may be aborted from the keyboard by depressing the **SPACE** bar or **ESCAPE** key. Note that the **SPACE** bar both aborts the program and stops the servomotor motion. You may also stop the system at any time by typing **J*** or pushing the STOP button on your MIS-200 if you have one.

To determine the setting of FFGAIN that was achieved, type T?. The PMC will respond with the values for the gains and the compensators. An example of what the response might look like follows:

=)T? P=18 V=25 F=33 X=0 CP=00 CV=0C

5.4.5 Adjusting the Position Loop Integral + Proportional Compensator

Modify the Tuning routine line TP1+ to TCP+ to tune the position loop compensator. The object is to raise the value for PLCOMP to the maximum possible without adding overshoot to the tachometer response. With your oscilloscope still attached to the tachometer signal, run the Tuning routine. Keep watching the tach signal and, when it overshoots, terminate the operation of the Tuning routine by typing ESCAPE. To determine the setting of PLCOMP that was achieved, type T?. The PMC will respond with the values for the gains and the compensators. An example of what the response might look like follows:

=}T? P=18 V=25 F=33 X=0 CP=0A CV=0C

Again, adjust the compensator value until the response is as desired. Note that for systems where the position error is not critical other than at rest, the position loop integral + proportional compensator is optional.

The servo system is now highly tuned, and don't forget to adjust the acceleration rate to a properly chosen, and probably lower, value before attempting accelerating and decelerating to high speeds. Failure to do so will likely result in severe overshoot and may "trip out" the system due to excess position error.

Before changing your velocity acceleration and distance, use the Report routine to monitor position error during an index by trying the following experiment. First, delete the TF1+ and the D300 lines of the program in Program Mode. The routine has now been modified to report position error 50 times and, assuming that your programming terminal is running at 9600 baud, it will take approximately 5 msec per report. Now add the following program to the end of the program buffer by typing **P<cr>**.

@Index_Demo	mark the beginning of the "I" routine
I+	index in the positive direction
FR	call the "R" function
D300,	delay 300 msec after the motion command stops
BI	transfer MPL program execution to label "I"
<escape>	exit the Program mode, returning to interactive command mode
=}BI	transfer operation to the "I" program

When you run the program above, you will note that each time the servomotor moves, 50 "error reports" will be sent to the console.

5.5 PUTTING CONFIGURATION COMMANDS IN A POWERUP ROUTINE

When all your gain and compensation values are determined, you can edit the @P_Powerup routine to be labelled @@_Powerup. **The first routine in the MPL Program buffer labelled @@ will be executed automatically whenever power is applied to the system, or the software is reset using an N* command.** Don't forget to edit the lines that set the gains and compensators for the values which you have determined with your setup procedure. Note also that the Home routine included in your PMC will execute a Home function and zero the system. With a highly tuned positioning system, it is important to not saturate the power amplifier due to attempting to accelerate faster than the system can handle, otherwise the system will overshoot and appear unstable. Therefore, the Powerup routine should include a suitable Acceleration parameter. Try inserting one right before the SM2 command.

Holding the STOP' line active during powerup will cause the powerup routine to be ignored by the PMC.

The Home function included will find a unique position within one revolution of the position encoder by first finding the encoder reference signal with the H- command, and then indexing 125 counts in the positive direction. The 125 count index is obviously arbitrary and may be edited to be any number that you want, but this Home routine is arbitrary as well and you should modify it to be consistent with your application needs.

For example, if your PMC application is controlling a linear table with a ball screw, then you may want the Home routine to operate as follows:

@Home	mark the beginning of the "H" routine
SXA	Set the X Register bits 3 and 1, to select 192 kHz top speed and enable Machine I/O general purpose inputs IN4' and IN8' to operate as -LIMIT and +LIMIT respectively. SXA sets the X Register bit pattern to 00001010.
J-	Jog the table in the negative direction until the -LIMIT limit switch actuates and stops motion.
H,+	After waiting for the Jog motion to stop, move to the nearest encoder reference in the positive direction.
I125,+	After waiting for the Home motion to stop, move 125 counts in the positive direction.
D50,	Delay 50 milliseconds after the motion stops.
N-	Normalize the absolute position counter to 0.
E	Unconditionally exit the Home routine.

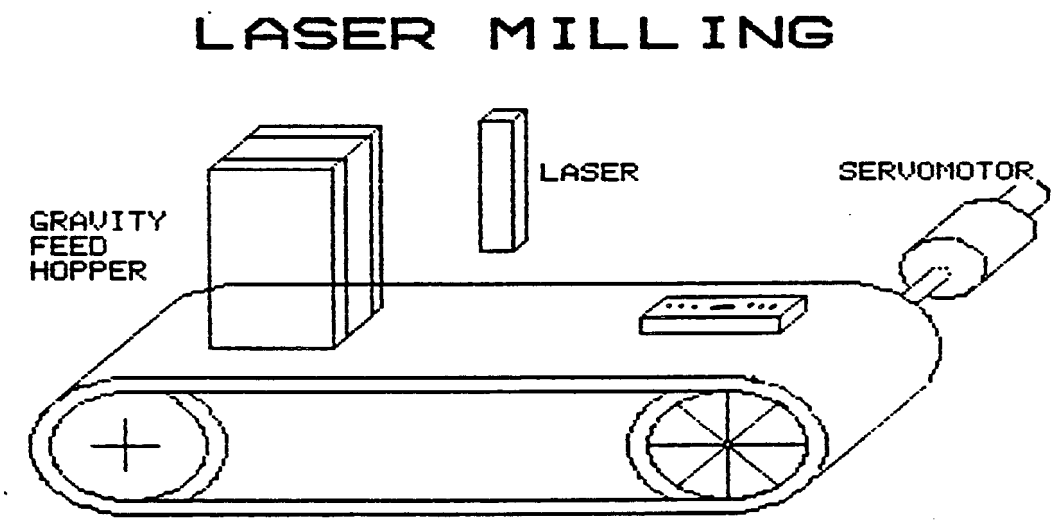
Some observations about the above Home routine follow:

* The SXA command is probably more logically included in the Powerup routine than in the Home routine, but is shown here for illustration purposes.
* This routine homes the table to the accuracy of the encoder reference, even though the limit switch is less accurate.
* The limit switch must be accurate to within one revolution of the position encoder, however, or the system could end up one revolution of the position encoder away from the true "Home position".
* The position of the limit switch should be adjusted so that its nominal trip point is far enough away from the encoder reference to prevent the problem noted above. This relationship can be interactively examined by Jogging negative until the limit is hit and the motion stops, normalizing the absolute position counter with an N- command, homing in the positive direction with the H+ command, and checking the distance between the limit switch and the encoder reference with the G! command after motion stops.
* The Home routine as listed uses the Home and Jog speeds set prior to calling it. A more robust routine would set those speeds prior to initiating motion.

5.6 LASER MILLING APPLICATION

This application description is included to give an example of programming techniques which may be useful for your application.

XYZ Company has products, A and B, which are laser milled with two distinct patterns. Pattern A includes three equally spaced holes, a 1/2" slot and three more holes, while Pattern B has both sets of three holes but omits the slot. Pattern selection should be fully programmable, allow simple changeover from Pattern A to B and operate with a minimum of operator assistance.



The program moves the bar stock 1.500" from the material hopper to the starting position, and calls Function "X" which mills the first three hole pattern. The PMC activates the laser, milling three holes precisely .250" apart. Using an input from a switch or sensor, the program selects either Function "Y" or "Z" to continue the milling operation. Function "Y" reduces the velocity and activates the laser to mill the .500" center slot. Function "Z" omits the slot and moves to the location for the final three holes.

After recalling Function "X" to repeat the process of drilling the three-hole pattern, the system returns for the next piece. This application demonstrates how MPL, with its terse, intuitive program command structure, provides the ability to easily combine command functions in a general manner to provide a wide variety of non trivial motion control solutions.

This particular program illustrates the PMC's outstanding performance. The 0.250" moves require only 32 milliseconds and the speed regulation while milling the slot is within .1%. Since this program requires only 150 bytes, the PMC's 2048 byte program space is adequate for significantly more complex applications. The power of MPL is further illustrated by the capability of changing the 3 hole pattern to 100 holes by adding only one character to program space.

MPL PROGRAMMING FOR LASER MILLING APPLICATION

MPL	COMMENT
@Laser	Label Laser milling routine "L"
V300	Set Velocity to 3.00 in/sec (30.0 kHz)
A1000	Set Accel rate to 100 in/sec^2 (1000 Hz/msec)
U1	Wait Until Run Switch (Input 1) is on
G1500+	Go to absolute position +1.500" (+1500 counts)
D,	Delay until conveyor stops
FX	Mill 3 hole pattern by calling Function "X"
FY2	Mill slot for Product A by calling Function "Y" if Input 2 is on
FZ-2	Skip slot for Product B by calling Function "Z" if Input 2 is off
FX	Mill 3 hole pattern by calling Function "X"
G0+	Return for next piece (Go to absolute position 0)
D,	Delay until conveyor stops
BL	Branch to "L" to repeat routine
@X	Label function "X"
O1	Activate laser by asserting Output 1
D100	Delay 100 milliseconds for drilling operation
O0	Deactivate laser by clearing Output 1
V150	Set Velocity to 1.50 in/sec (15.0 kHz)
I250+,	Index forward .250" (250 counts); wait until conveyor stops
LX2	Loop to function "X" twice
E	Exit "X" and continue with routine "L"
@Y	Label function "Y"
I750+,	Index forward .750" (750 counts); wait until conveyor stops
V20	Set Velocity to .2 in/sec (2.0 kHz)
O1	Activate laser by asserting Output 1
I500+,	Index forward .500" (500 counts); wait until conveyor stops
O0	Deactivate laser by clearing Output 1
V150	Set Velocity to 1.5 in/sec (15.0 kHz)
I750+,	Index forward .750" (750 counts); wait until conveyor stops
E	Exit "Y" and continue with routine "L"
@Z	Label function "Z"
V300	Set Velocity to 3.00 in/sec (30.0 kHz)
I2000+,	Index forward 2.000" (2000 counts); wait until conveyor stops
E	Exit "Z" and continue with routine "L"

OPERATION

6.1 MPL COMMAND OVERVIEW

ORMEC's Motion Programming Language (MPL) has 21 basic commands, which can be used in hundreds of variations to meet specific application needs and create robust motion control routines. Below is a brief overview of the MPL command areas and their basic functions in creating motion control applications. Syntax information for these commands are found, in short form, in Section 6.2 or, in complete detail, in Section 6.9.

Command	Name	Description/Function
@		Establish a single-letter program label in the program buffer for future reference.
A	Acceleration	Set or examine the acceleration rate.
B	Branch (GoTo)	Transfer MPL program execution to a program label with no return.
C	Contour	Generate complex motion profile
D	Delay	Delay a specified time interval (in msec) or a number of counts before executing the next command.
E	Exit (Return)	Exit an MPL subroutine and return to the MPL statement after the original subroutine call.
F	Function (GoSub)	Transfer MPL program execution to a program label. When an Exit command is executed, MPL operation resumes at the line following the original "F command". Function calls may be nested up to three levels deep.
G	Go	Move to the specified absolute position of the system. Examine the system's absolute position or commanded absolute position.
H	Home	Move at the specified Homing speed to the nearest encoder reference or machine sensor.
I	Index	Move the specified distance from the current position.
J	Jog	Move at the specified jog speed.
K	Kill	Kill any commanded motion.
L	Loop (Repeat)	Transfer MPL program execution to a program label a specified number of times, and then continue program execution with the next command in the program buffer. Loops may be nested up to three levels using a loop counter command.
N	Normalize	Define the current physical position. Reset the PMC firmware or serial communications baud rate. Establish or examine memory checksums.
O	Output	Set general purpose Machine I/O outputs.
P	Program	Enter, edit or examine a motion program.
S	Set or Show	Set or examine system mode, baud rate, registers, general purpose machine inputs, software limits, last label passed, system status, program trace or program buffer write protect.
T	Tuning	Tune servo loops or examine tuning parameters.
U	Until	Wait until the specified condition is true before executing the next MPL command.
V	Velocity	Set or examine index speed in the Motion Buffer.
=	Assign Axis-ID	Assign axis identifier to a PMC for future Serial Bus Communications.

Now that we have an understanding of the purpose of each command, there are two basic constructs in the architecture of the language which deserve special mention.

Display Characters

MPL simplifies the development of motion control applications by displaying information on the actual status of motion commands. This includes providing information on parameters in the motion buffer by typing a command and a (?), actual data on the system's speed or position by using a (!) or repeatedly displaying current speed or position by utilizing a (%).

Synchronization Characters

MPL's synchronization characters (,) (;) and (:) offer an effective method for coordinating MPL commands with motion in progress. The (,) allows MPL to wait for the previous motion be completed before executing a new command, the (;) holds program execution for the system to reach a constant speed or the motion to be completed, and the (:) causes MPL execution to wait for the end of constant speed before the next command is executed.

In the following description of MPL syntax, the following symbols are used:

< > - designates a variable
- the enclosed item (or items) may be repeated multiple times
[ ] - the enclosed item (or items) are optional

6.1.1 MPL Break Features

When MPL is executing a program or waiting for some event to finish, e.g. using the (,) (;) or (:) delay features, you may want to abort the program. This can be done by sending an ESCAPE character (ASCII 33_H) if serial communications are established. If serial communications have not been established (meaning that the baud rate has not been specified) then any character sent to the PMC at serial communications interface will abort the program execution.

When it is desirable to stop motion in addition to aborting an executing MPL program a SPACE character (ASCII 20_H) may be sent.

There is also a hardware STOP line. The signal STOP' (or BREAK') on TM2 or JM2 will always abort whatever is going on, stop any commanded motion and return the interactive command prompt.

6.2 MPL SYNTAX OVERVIEW

Acceleration	A	<rate> <cr>	A <display>
	A	[<relative> <rate>] <cr>	
	AL	<rate> <cr>	AL <display>
	AL	[<relative> <rate>] <cr>	
	AQ	<rate> <cr>	AQ <display>
	AQ	[<relative> <rate>] <cr>	
	AS	<rate> <cr>	AS <display>
	AS	[<relative> <rate>] <cr>	
Branch	B	<label> [<condition>] <cr>	
	B	<label> <op> <position> [<direction>] <cr>	
Contour	C	<timebase> <distance>	
	C	<ref-distance> <distance>	
Delay Time	D[T]	[<time>] [<sync>] <cr>	
	DM	[<distance>] [<sync>] <cr>	
	DR	[<distance>] [<sync>] <cr>	
Exit Program	E	[<condition>] <cr>	
	E	<op> <position> [<direction>] <cr>	
Function Call	F	<label> [<condition>] <cr>	
	F	<label> <op> <position> [<direction>] <cr>	
Go	G	<position> <direction> <cr>	G <display>
Home	H	[<speed>] <direction>	H <display>
	H	[<relative> <speed>] [<direction>]	
Index	I	[<distance>] #<direction># <cr>	I <display>
	I	[<relative> <distance>] #<direction># <cr>	
Jog	J	[<speed>] #<direction># <cr>	J <display>
	J	[<relative> <speed>] #<direction># <cr>	
Loop	L	[<label> <count>] <cr>	
	L	[<label> <loop cnt id> <count>] <cr>	
Normalize	N	[<position>] <direction>	N <cr>
	NC	<cr>	NC <display>
Output	O	[<sync>] <mask> [/<hex>]	O <display>
Program	P	<program> <text>	
	P	<label> <cr> <text>	
Quit	Q	<cr>	
Set or Show	S	<register> <hex> <cr>	S <display>
	SC	<display>	
	SL	<name> <position> [<sign>] <cr>	SL <display>
	SL	<name> <hex>	
	SM	<mode> <cr>	SM <display>
	SP	<display>	
	SS	<display>	
	ST	<hex>	ST <display>
	SW	<hex>	SW <display>
Tune Loops	T	#<register> [<value>] #<sign>##<cr>	T [E] <display>
Until	U	<condition> <cr>	
	U	<op> <position> [<direction>] <cr>	
Velocity	V	<speed> <cr>	V <display>
	V	[<relative> <speed>] <cr>	
Label	@	<label> <text> <cr>	@ <text><cr>
Assign ID	=	<id> <cr>	= <display>

Definition of Syntax Parameters

<condition>	The testing of specific machine inputs in the format <mask> [/<hex>]. A <cr> is always a go condition.
<count>	Number of times for operation to be repeated
<cr>	Carriage return ($0D_H$)
<display>	? display last entered value ! display current system value % [<time>] display status each <time> interval (valid for G,H,I,J,O,SC,SS,TE,V only) <sync> permitted in most cases
<distance>	Number of relative encoder counts
<direction>	+/- positive/negative <speed>, <position> or <distance> * stop system motion (Used on G,H,I,J commands) <sync> permitted in most cases
<hex>	Hexadecimal numbers (0-9, A-F)
<id>	Motion axis identifier (most displayable characters)
<label>	Displayable character used to identify a motion routine
<loop count id>	the character X, Y, or Z specifying the loop counter to be used. This allows loops to be nested up to three levels.
<mask>	a hexadecimal number that specifies which machine input bits are to be compared and which are to be ignored.
<mode>	Control mode: 0=idle; 1=velocity; 2=position; 3=position without resetting position error; 4=master axis controller
<name>	F the forward software settable overtravel limit R the reverse software settable overtravel limit H the polarity of the hardware limits
<op>	An operator specifying "greater than (>) or "less than" (<) position for commands using conditional testing
<position>	Absolute position in encoder counts
<program>	Enter, edit or display MPL program buffer: { initiate programming at beginning of program buffer <cr> initiate programming at end of program buffer ? Display program buffer from the beginning ! display entire program buffer
<rate>	Acceleration rate in kHz/sec; 100 Hz/sec; or 100's counts
<ref-distance>	A value from 0-9, A-H which specifies the number of relative distance counts of the motion reference bus for the length of a position/-position segment in the Contour command.

<register> Tuning: P=(position gain); V=(velocity gain), F=(feedforward gain), X=(external output gain), CP=(position loop compensation), CV=(velocity loop compensation) Status: X, Y, Z Program buffer write enable: W Program trace: T

<relative>
P increase the magnitude of the speed, distance, or rate by the value immediately following the <relative>
M decrease the magnitude of the speed, distance, or rate by the value immediately following the <relative>

<sign>
+/- add/subtract <value> to/from <register>
<cr> set <register> to <value>

<speed> Speed in 10 Hz, 100 Hz or .01%

<sync> Synchronization character for coordinating motion:
, wait until current motion is complete
; wait until constant speed or motion complete
: wait until end of constant speed or motion complete

<text> Motion routines, comments or editing command characters

<time> Time in milliseconds

<timebase> a value from 0-9, A-H specifying the length of a position/speed segment in the Contour command in increments of 1.33 msec.

<value> value substituted for, added to or subtracted from <register>

6.3 SETUP PARAMETER RANGES, DEFAULTS AND UNITS

Motion Parameters		Range	Default	Units
Acceleration	48kHz Mode	1-65,535	-	100 Hz/sec
Jog		2- 4,800	-	10 Hz
Velocity		2- 4,800	-	10 Hz
Home		2- 4,800	-	10 Hz
Acceleration	192kHz Mode	1-65,535	100	kHz/sec
A		1-65,535	4000	kHz/sec
AL		1-65,535	4000	kHz/sec
AQ		1-65,535	1000	kHz/sec
AS		1- 1,920	100	100 Hz
Jog		1- 1,920	400	100 Hz
Velocity		1- 1,920	20	100 Hz
Home				
Acceleration	384kHz Mode	1-65,535	-	kHz/sec
Jog		1- 3,840	-	100 Hz
Velocity		1- 3,840	-	100 Hz
Home		1- 3,840	-	100 Hz
Acceleration	Ext Mode	0-65,534	-	100 counts
Jog		2-10,000	-	.01%
Velocity		2-10,000	-	.01%
Home		2-10,000	-	.01%
Acceleration E-stop		0-65,534	1000	kHz/sec
Acceleration Limits		0-65,534	4000	kHz/sec
Index		1-2,147,483,648	500	counts
Go		0-1,073,741,824	0	counts
Normalize		0-1,073,741,824	0	counts
Delay [time]		0-65,535	0	msec
Delay [M]		0-4,294,910,759	0	motion counts
Delay [R]		0-4,294,910,759	0	reference counts
<Label>		20_H to $7E_H$	-	-
Tuning Parameters				
Position Loop Gain		0-255	2	-
Velocity Loop Gain		0-255	2	-
Feedforward Gain		0-255	0	-
External Output Gain		0-255	0	-
Velocity Loop Compensator		0-F	0	-
Position Loop Compensator		0-F	0	-
Register Parameters				
T Register		-	0	trace status
W Register		-	0	write enable
X Register		-	08	motion parameters
Y Register		-	00	special motion
Z Register		-	10	communications
Mode		0-4	0	servo-status
Baud rate		0-8	0	(autobaud)

6.4 EDITING FUNCTIONS USED DURING PROGRAM MODE

MPL commands can be combined in a motion "Program" using the Program command. To write or edit an MPL Program, type **P<cr>** or **P<label><cr>** from the keyboard when at the =) prompt.

Cursor Right — TAB (CTRL-I) or CTRL-Y moves the cursor to the right one character. Typing a TAB when the cursor is at the end of a line will move the cursor to the beginning of the next line.

Cursor Left — BACKSPACE (CTRL-H) moves the cursor to the left one character. Typing a BACKSPACE when the cursor is at the beginning of a line will move the cursor to the end of the previous line.

Cursor Down — LINEFEED (CTRL-J) moves the cursor down a line.

Cursor Up — DEL or CRTL-U moves the cursor up a line. Type a DEL when at the cursor is in the middle of a line to move it to the beginning of that line.

End of Line — CTRL-R moves the cursor to the right end of the current line.

Change Line — To change a line in a motion control program, position the cursor at the point to be changed and type the desired information. Underscores (_) may be used to mark program buffer space for future parameter changes or MPL commands.

Kill Line — CTRL-K deletes all characters from the cursor to the end of the line. It can be used to delete an entire line or only unwanted characters at the end of the line. Type CTRL-K with the cursor positioned at the end of a line to delete the "end-of-line" marker and append the next line to the current line.

Add Line — Type <cr> at the beginning of a line to insert a "blank" line before it; then type the data for that line. Since "blank" lines are not allowed, immediately typing a second <cr>, or moving the cursor, will eliminate the "blank" line just created. Type <cr> in the middle of a line to "split" the line into two lines.

Exiting — The ESCAPE key is used to exit the program command.

Program Erase — Typing a } in column 1 (immediately after a <cr>) will erase the program buffer, starting at the current location, and exit the program command. Note: executing this command will erase all information from the cursor to end of the program buffer.

6.5 STATUS REGISTERS

There are three status registers, designated X, Y, and Z. The purpose of the status registers is to allow the user to conveniently change the configuration of the PMC to meet individual motion control application needs. Each status register contains eight "bit switches", which when "set" or "cleared" will change the operation of the PMC is some way.

The status registers are examined or configured with the S command. To examine them powerup the PMC in interactive command mode and type the following sequence. **Bold** print indicates the sequence that you type and regular print is the information sent by the PMC.

MPL3.0b
=)S? X=08 Y=00 Z=00 The three parameters returned by the PMC are two digit hexadecimal values specifying the bit patterns in each of the status registers. The first digit of each pair indicates the value of bits 7-4 of the register and the second digit indicates the value of bits 3-0, respectively.

The following chart indicates the relationship between the two digit hexadecimal values and the individual "bit switches" of the Status Registers.

Status Register Bit Assignments

First Hex Digit	Bit Number 7 6 5 4	Second Hex Digit	Bit Number 3 2 1 0
0_H	0 0 0 0	0_H	0 0 0 0
1_H	0 0 0 1	1_H	0 0 0 1
2_H	0 0 1 0	2_H	0 0 1 0
3_H	0 0 1 1	3_H	0 0 1 1
4_H	0 1 0 0	4_H	0 1 0 0
5_H	0 1 0 1	5_H	0 1 0 1
6_H	0 1 1 0	6_H	0 1 1 0
7_H	0 1 1 1	7_H	0 1 1 1
8_H	1 0 0 0	8_H	1 0 0 0
9_H	1 0 0 1	9_H	1 0 0 1
A_H	1 0 1 0	A_H	1 0 1 0
B_H	1 0 1 1	B_H	1 0 1 1
C_H	1 1 0 0	C_H	1 1 0 0
D_H	1 1 0 1	D_H	1 1 0 1
E_H	1 1 1 0	E_H	1 1 1 0
F_H	1 1 1 1	F_H	1 1 1 1

1 indicates that the bit switch is set or on
0 indicates that the bit switch is cleared or off

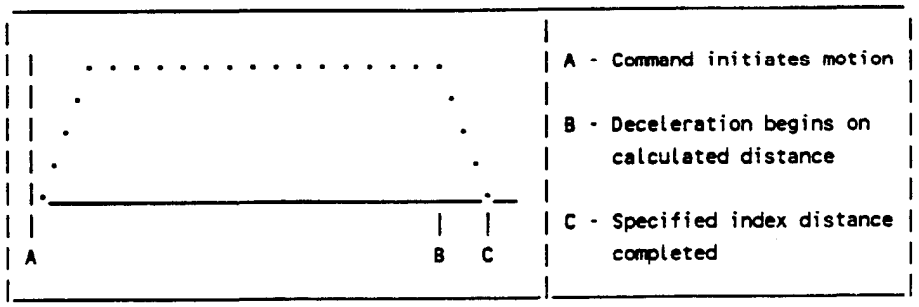
To determine the hexadecimal values for the first and second digits, find the desired bit pattern of half of the register and look up the hexadecimal value for that bit pattern in the table. e.g. the command **SX03** will set the bits in the X Register to 0000 0011.

See Section 6.9 for a detailed description of the various options that can be selected using the Status Registers.

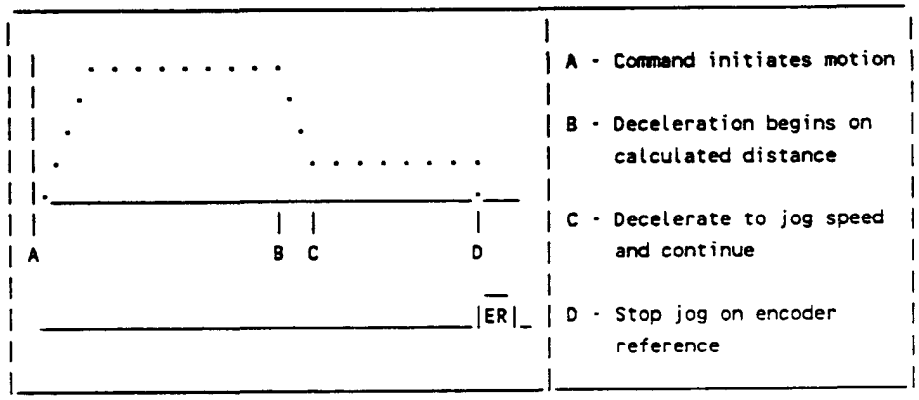
Creating Special Motions Using the Y Register

The PMC's Y Register offers users a variety of methods for starting and stopping, allowing a wide range of motion profiles. The illustrations below show some of these options and how the Y register's Bits 0-5 can be manipulated to utilize sensor inputs (SENSIN signal) and encoder reference points to create unique motion profiles. Note that all the options are not shown. Also, a linear acceleration profile is assumed throughout. Starting and stopping bits operate totally independent of each other.

STATUS COMMAND----SY00 **COMMENT**

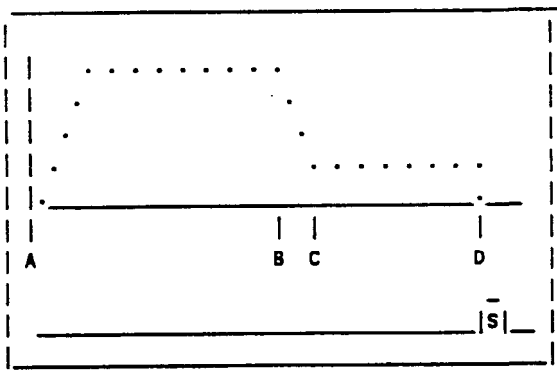


STATUS COMMAND----SY02 **COMMENT**



STATUS COMMAND----SY06

COMMENT



A - Command initiates motion

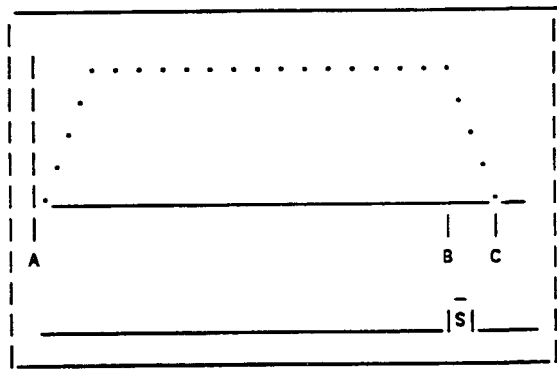
B - Deceleration begins on calculated distance

C - Decelerate to jog speed and continue

D - Stop jog on sensor

STATUS COMMAND----SY08 or SYOC

COMMENT



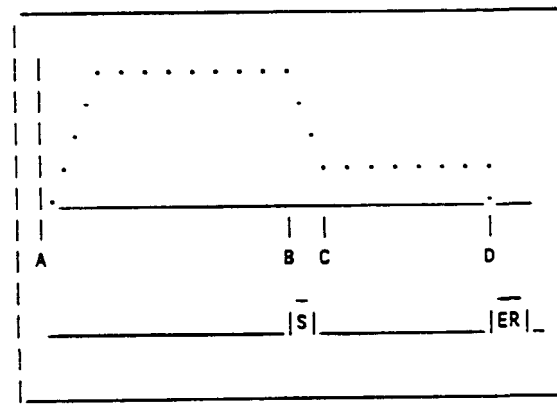
A - Command initiates motion

B - Deceleration begins on sensor

C - Motion completed

STATUS COMMAND----SY0A

COMMENT



A - Command initiates motion

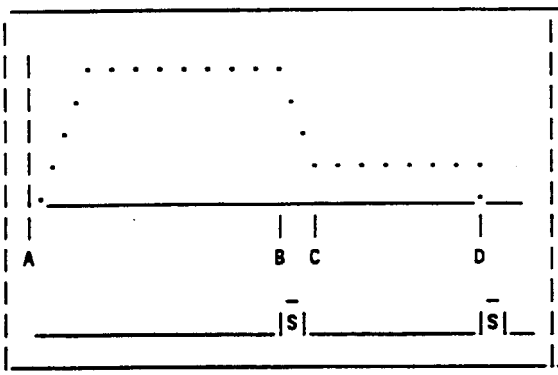
B - Deceleration begins on sensor

C - Decelerate to jog speed and continue

D - Stop jog on encoder reference

STATUS COMMAND----SY0E

COMMENT



A - Command initiates motion

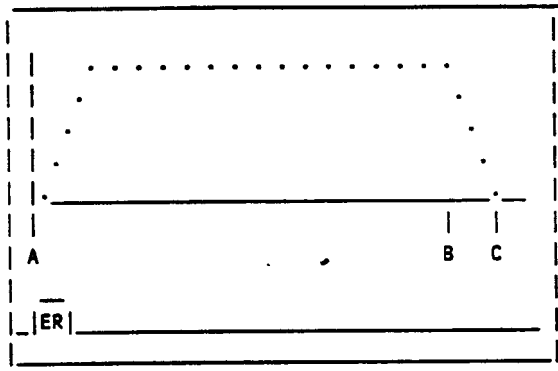
B - Deceleration begins on sensor

C - Decelerate to jog speed and continue

D - Stop jog on sensor

STATUS COMMAND----SY20

COMMENT



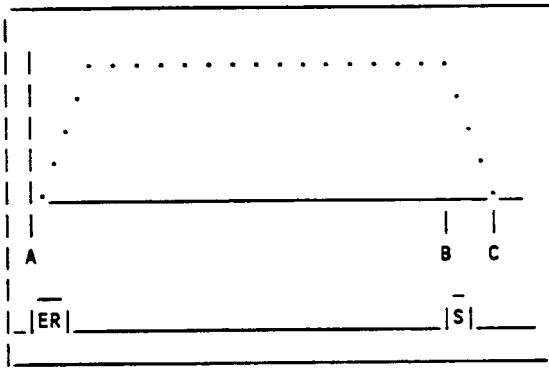
A - Motion starts on Encoder Reference

B - Deceleration begins at calculated distance

C - Specified index distance completed

STATUS COMMAND----SY28

COMMENT

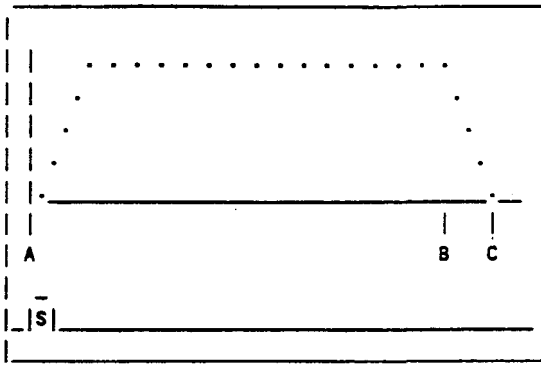


A - Motion starts on Encoder Reference

B - Deceleration begins on sensor

C - Motion completed

STATUS COMMAND----SY30



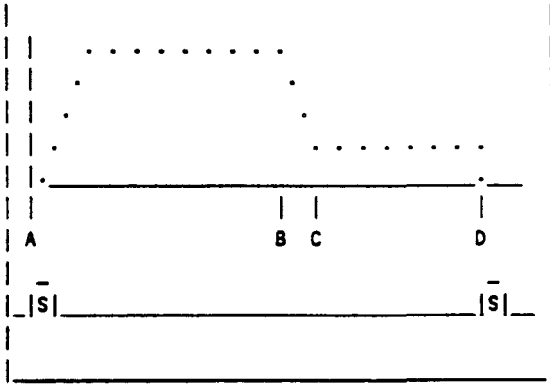
COMMENT

A - Motion starts on Sensor

B - Deceleration begins at calculated distance

C - Specified index distance completed

STATUS COMMAND----SY36



COMMENT

A - Command initiated on sensor

B - Deceleration begins at calculated distance

C - Decelerate to jog speed and continue

D - Stop jog on sensor

6.6 MACHINE I/O OPERATION

The Machine I/O Interface of ORMEC's Programmable Motion Controllers provides 16 TTL level digital I/O points. These I/O points include 11 discrete inputs and 5 discrete outputs, and are designed to be compatible with industry standard "OPTO-22 style" optically isolated I/O Modules. For logical compatibility with those modules (including the LEDs on them), the inputs and outputs are considered "asserted" or "on" when at 0 volts (TTL low) and "not asserted" or "off" when at 5 volts (TTL high).

6.6.1 General Purpose Machine Inputs

Eight of the eleven machine inputs may be read and used to control "program flow" by the Branch, Exit, Function & Until commands. These commands can specify any combination of the eight inputs to be tested using a <mask> parameter of two hexadecimal characters. A single command can test for individual selected inputs to be either "on" or "off" using the test <status> parameter. (also two hexadecimal characters)

Bits in the <mask> parameter that are set to 1 cause the corresponding machine input to be "tested" by the MPL command. Mask bits set to 0 cause the corresponding machine input to be ignored by the command. Setting a bit of the optional <status> parameter to 1 causes the command to "test" for the corresponding input to be "on", as long as it is selected for test by the <mask> parameter. Conversely, setting a <status> bit to 0 causes the command to "test" for the corresponding machine input to be "off". If the optional <status> parameter is not specified, all inputs selected by the <mask> parameter will be "tested" for the "on" condition.

6.6.2 Special Purpose Machine Inputs

Three of the discrete Machine Inputs have a prespecified affect on the operation of the PMC-903 or PMC-904. Asserting the STOP input will stop both commanded motion and MPL program execution. Asserting the EXECUTE input will cause an MPL program to start. This program to be started is selectable from the Machine I/O using either a five bit address or the select (SEL) input.

6.6.3 Machine Outputs

The Output command can be used to selectively affect any or all of the four General Purpose Machine Outputs, as defined by the <mask> parameter. In addition, a single Output command can be used to turn "on" or "off" multiple outputs by specifying their desired state with the <status> parameter.

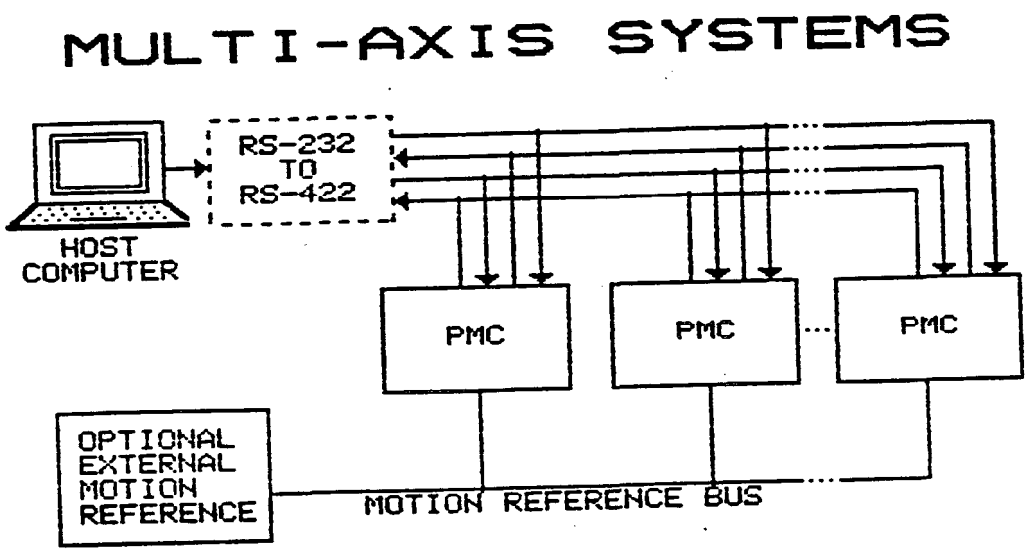
Setting a bit in the <mask> parameter to 1 causes the corresponding machine output to be affected by the command. Conversely, setting a bit in the <mask> parameter to a 0 causes the corresponding output to be unchanged by the command. Setting a bit in the optional <status> parameter to a 1 or 0 causes the Output command to turn "on" or "off" the corresponding machine output, as long as it was selected by the <mask> parameter. If this parameter is not specified, all the selected outputs will be turned on. The fifth machine output is the READY output which is asserted whenever the PMC is at the interactive level and "ready" to execute either a command or a motion program.

6.6.4 Other Discrete I/O Options

The Machine I/O Interface can also be configured for special automatic operation by setting up option parameters in the X and Z registers. The available options include hardware +/- limits, program select line, fault output and in motion output. Refer to section 6.9.17 for more detailed information.

6.7 MULTI-AXIS COMMUNICATIONS USING THE SERIAL COMMUNICATIONS BUS

PMCs allow hardware Axis-ID selection for Serial Communications Bus support. This feature supports Axis-IDs of A - N (14 devices) based on the strapping configuration of header J6 Pins 1-8. See Section 4.6 for the jumper strap to Axis-ID mapping. The Axis-ID may also be set in non-volatile memory by the user, using the "=" command. To do this, Bit 5 of the Z-Register be set before the Axis-ID can be enabled or changed from the Serial Communications Interface. This is to help prevent inadvertent use of the "=" command by the user.



6.7.1 Using Serial Bus Communications

Assuming that some, but not all, of the jumpers were in place at powerup, the PMC identified itself as an axis between A and N, and is in the inactive Serial Bus Communications mode.

The significance of enabling this communications mode is that the PMC will not send data out the Serial Communications Interface until it recognizes its Axis-ID being selected for communications. This means that once the feature is enabled, the user must send a two character "select axis" sequence before the PMC will communicate.

At powerup, a host computer or terminal must first send a sequence of "carriage returns" long enough for the PMCs to determine their baud rates (See Section 5.1), and then send the "select axis" sequence as shown below.

Syntax: <CTRL]> <id>

where <CTRL]> is ASCII ($1D_H$) and
<id> is the Axis-ID of the unit selected

Executing the above command selects the specified PMC for communications with the host. Its communications driver ICs are enabled, and it will send and receive data characters with the host computer and properly operate the flow control line. In addition, the normal "prompt" of =} will be changed to <id>}, displaying the Axis-ID of the active PMC.

All PMCs not selected automatically assume the inactive Serial Bus Communications Mode with their communications driver ICs disabled. They will continue to control motion or execute MPL programs

from the Program Buffer, but will ignore communications at the SCI interface until selected. In addition, any messages generated while in this mode will be lost.

"Binary In - Binary Out" communications mode, as selected by Status Register Z, Bit 7 is not supported when serial bus communications mode is used. The other three modes are fully supported.

6.7.2 Assigning an Axis-ID from the Serial Communications Interface

Assuming that all the straps are in place at powerup, the PMC operates in a mode where a Serial Bus Axis-ID can be set in non-volatile RAM from the Serial Communications Interface using the **= command**, as long as Bit 5 of the Z-Register has first been set. The Z-Register is manipulated using the Set/Show command. See Section 6.5 of the PMC Manual for a more complete description of the Status Registers. If the Z-Register, Bit 5 is not set (default) the Axis-ID will not be assigned. In this case, a D3 Error (Parameter Write Protected) will be issued by the PMC.

The syntax of the command used to assign an Axis-ID is as follows:

Syntax: **= <id> <cr>**

where <id> most printable ASCII characters greater than a space (20_H) except ! and ? and =.
Note: All alphabetic Axis-IDs must be upper case.

<display> ? the <id> stored in non-volatile RAM will be displayed

= the <id> will be set to = and Serial Bus Communications will be disabled

other the Axis-ID will be set to the specified printable ASCII character, the Serial Bus Communications Mode will be enabled, and the PMC's serial communications will become inactive

WARNING: If you inadvertently assign an Axis-ID and don't know what it is, you will lose communications with that PMC. The PMC becomes "inactive" on the bus (since you cannot specify a "select axis" sequence to make it the active sender & receiver).

This situation may be remedied in two ways:

1) If you have an MIS-200, hold the STOP button in and reset (or powerup) the PMC. It will powerup without Serial Bus Communications Mode enabled. The **= command** may then be used to either examine the Axis-ID present in the non-volatile RAM, disable Serial Bus Communications Mode, or reassign a new Axis-ID.
2) Power down, and remove the jumpers from Header J6, and it will have the same effect as holding the STOP button during powerup.

Once the Axis-ID has been set in non-volatile RAM, and as long as the jumpers are still in place, the PMC will automatically identify itself with the non-volatile RAM based Axis-ID and enable Serial Bus Communications Mode in the inactive state on powerup. Its operation in this mode is as described above in Section 6.7.2.

6.8 MOTION REFERENCE BUS

The Motion Reference Bus allows a PMC based motion control system to operate in synchronism with another PMC or other equipment having a digital position encoder. This allows a host computer to conveniently control multiple servomotors with "linear interpolation", in addition to several other general purpose capabilities.

This is accomplished by referencing a PMC's motion command pulses to a digital pulse train present at the EXREF' input at the System Axis Interface (TM2). The pulses present at this external motion pulse reference input are normally the motion pulses from another PMC. These pulses are driven by a tri-state RS-422 differential line driver (U29C), and received by an RS-422 differential line receiver (U28A).

For an example of how this synchronization is accomplished, consider three PMCs which have their EXREF and EXREF' signals bussed together by a twisted pair cable. In the powerup default mode, all PMCs are deriving their motion command pulses from their respective crystal controlled oscillators, and therefore working independently.

Now assume that we want PMC A to be the "motion master" and PMCs B and C to be "slaves" and go half as far as PMC A while PMC A does an index. Bit 4 of Status Register X (MOTION BUS MASTER), should be set on PMC A. This enables the tri-state driver (U29C) to drive the Motion Reference Bus. Next, Bit 6 of Status Register X, (MOTION BUS SLAVE) of the slaves should be set to select the external mode, allowing them to use position reference information received by line receiver U28A for deriving their respective position reference command information.

The parameters for the slave PMCs are now in a format that references the distance traveled by an external reference. Acceleration is expressed in units of 100's of external pulses on the Motion Bus from 0 to 65534 (0 to 6,553,400 pulses) and speed is expressed in units of hundredths of a percent of external reference frequency from 1 to 10000 (.01 to 100.00%).

The slave acceleration should now be set to zero and the slave commanded to "Jog" at a "jog velocity" of 50.00% of the Motion Bus frequency. This can be done by sending A0 and J5000+ commands to the slave. An index of the master will now also cause the slave to move half as far. One can extend this concept of externally referenced motion to implement a wide variety of coordinated motion applications such as electronic lineshafts, camshafts etc.

When the PMC is commanded to make an index in the external mode, no minimum index distance calculation is made. Since the frequency of the external reference is unknown, it is the user's responsibility to comply with the following specification:

for a=0: minimum distance = .012 * f * v

for a>0: minimum distance = (a/9.8 + f) * v/10

where: a - acceleration in 100 Hz
v - velocity in .01 %
f - external frequency in MHz

Note: Any slave axes must be set one frequency range higher than the master axis. For example, if the master axis is set in the 192kHz range, then the slave axes should be set in the 384kHz range.

6.9 MPL COMMAND DESCRIPTION

Unless specifically stated otherwise, the following command descriptions apply to the default condition of internal mode (MOTION BUS SLAVE reset) and 192 kHz velocity range (VELOCITY RANGE SELECT = 10). For a description of other non-default options, see Section 6.3. Throughout this section, the following symbols are used:

< > - designates a variable
- the enclosed item (or items) may be repeated multiple times
[ ] - the enclosed item (or items) are optional
| - designates the logical OR operator
<cr> - designates a carriage return

6.9.1 @ - Program Label Command

Purpose: **establish a single-letter program label in the program buffer**

Syntax: @ <label> <text> <cr>

<label> single byte program label; The label can be any ASCII character between an ! (21_H) and a ~ ($7E_H$). The @ character (40_H) is a unique programming label, in that this program will automatically execute on powerup or software reset of the PMC.

<text> any number of printable ASCII characters; If the label is not present it is recommended that the first character be an underscore (_) so that extra program labels are not created.

Examples:
```
@_This_is_a_comment_line
@X_Comment_after_a_program_label_'X'
@@_This_is_the_beginning_of_the_"powerup"_routine
```

6.9.2 A - Acceleration Command

Purpose: **set or examine acceleration rate for motion, limits, contour or E-stop**

Syntax:
A <rate> <cr> | A <display>
A [<relative> <rate>] <cr>
AL <rate> <cr> | AL <display>
AL [<relative> <rate>] <cr>
AQ <rate> <cr> | AQ <display>
AQ [<relative> <rate>] <cr>
AS <rate> <cr> | AS <display>
AS [<relative> <rate>] <cr>

<rate> integer (1 to 65,535) in kHz/sec specifying the acceleration rate (default: 100 kHz/sec); See Section 6.3 for other ranges.

<display> ! display current system acceleration rate (zero if at rest or top speed)
? display last entered acceleration rate

<relative> P increase the magnitude of the rate by the specified <rate>
M decrease the magnitude of the rate by the specified <rate>

Examples:

A3500<cr>	Set acceleration rate to 3500 kHz/sec or Hz/msec
AL6000<cr>	Set acceleration rate for a limit stop to 6000 kHz/sec or Hz/msec
AS400<cr>	Set acceleration rate for the stop input to 400 kHz/sec or Hz/msec
AQ100<cr>	Set acceleration rate for a contour motion stop to 100kHz/set
AP300<cr>	Increase the magnitude of the current acceleration by 300 kHz/sec or Hz/msec
A?	Display last entered acceleration rate
A!	Display current system acceleration rate
AL!	Display current limit acceleration rate

6.9.3 B - Branch (GoTo) Command

Purpose: **transfer MPL program execution to a program label**

Syntax: B <label> [<condition>] <cr> |
B <label> <op> <position> [<direction>] <cr>

Purpose: Transfer MPL program execution to a specified program label either:
- unconditionally
- based on a specified machine input condition (up to 8 inputs high or low)
- based on a specified test of absolute position

<label> MPL program label (see @ command description)

<condition> in the format <mask> [/<hex>], defining the required state of the general purpose inputs IN1'..IN8' IN10'..IN80' for the Branch command to execute. IF the specified input condition is true, THEN the command following the specified program label will be executed ELSE execution will continue with the following MPL command. IF <condition> is not specified, THEN MPL program execution is unconditionally transferred to the specified MPL program label.

<op> the operator specifying greater than or less than <position> as the condition to be tested against:

> greater than the specified absolute position. If <op> is > then the branch will be done if the current absolute position is greater than <position>. "Greater than" is defined as "more positive than the current position", or further in the plus (+) direction of travel.

< less than the specified absolute position. If <op> is < then the branch will be done if the current absolute position is less than <position>. "Less than" is defined as "more negative than the current position", or further in the minus (-) direction of travel.

<position> integer (0 to 1,073,741,823) counts specifying the absolute position to test against. Default is 0 if not specified.

<direction> + to specify <position> as a positive integer (default)
- to specify <position> as a negative integer

Examples:

BQ<cr>	Uunconditionally branch to program label 'Q'
BQ20<cr>	Branch to program label 'Q' if input A1 is asserted
BQ1/0<cr>	Branch to program label 'Q' if input I1 is not asserted
BL>200+<cr>	Branch to program label 'L' if the PMC-based system is at an absolute position greater than 200 counts in the positive direction. e.g. 201, 202, 203, etc.
BL<100-<cr>	Branch to program label 'L' if the PMC-based system is at an absolute position less than 100 counts in the negative direction. e.g. -101, -102, -103, etc.

6.9.4 C - Contour Command

Purpose: **create high performance profiled motion**

Syntax: C <timebase> <distance>
C <ref-distance> <distance>

Purpose: C <timebase> <distance>
The Contour command allows specification of a general motion-time profile in <timebase> segments over a range from 1.33 to 341.33 msec. These linear "position vs. time" segments may be commanded in real time by a host computer through the serial communications interface or "Programmed" in the MPL program buffer for later execution by the PMC.
C <ref-distance> <distance>
The Contour command also allows specification of a general motion-motion profile in <ref-distance> increments of the motion reference bus over a range from 256 to 65,536 motion reference pulses. These linear "position vs. position" segments may be commanded in real time by a host computer through the serial communications interface or "Programmed" in the MPL program buffer for later execution by the PMC. This capability allows "multi-axis contouring" capability which can be referenced to a common "master axis controller" or to an external source of motion information.

<distance> Number of relative encoder counts

<ref-distance> A value from 0-9, A-H which specifies the number of relative distance counts of the motion reference bus for the length of a position/position segment in the Contour command.

<timebase> A value from 0-9, A-H which specifies the length of a position/speed segment in the Contour command in increments of 1.33 ms.

Note: This command is designed to create high performance profiled motion which is defined and coded by a host computer. The coded Contour profile is then either downloaded into the program buffer or sent to the PMC via the serial communications interface in real time. It is not practical to define Contour data manually. **For more information refer to Section 6.11, entitled "Creating Complex Motion Profiles".**

6.9.5 D - Delay Command

Purpose: **delay a specified time interval or number of counts before executing the next command**

Syntax: D[T] [<time>] [<sync>] <cr>
DM [<distance>] [<sync>] <cr>
DR [<distance>] [<sync>] <cr>

T time, in milliseconds. If the T is not specified it is assumed that the delay command is for time.

M commanded motion, in counts. The system must be at steady state speed to use this command.

R reference clock, in counts. The system must be at steady state speed to use this command.

<time> integer (0 to 65,535) in milliseconds or counts, specifying the amount of time to be delayed before executing the next command; The resolution of the internal timer is 4 msec and due to the asynchronous nature of the delay command there is an uncertainty of 4 msec. e.g. Since <time> is "rounded up" a D1 command will delay 4 to 8 msec. A D0 command delays less than 1 msec.

<distance> integer (1 to 4,294,910,759) specifying the number of counts or reference clocks to delay

<sync> , synchronizing character which causes the PMC to wait for the system motion to stop

; synchronizing character which causes the PMC to wait for the system motion to achieve a steady state speed

: synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

Note: An ESCAPE entered during the execution of this command will end this command with a D0 error. (See Section 6.5.1)

Examples:

D16<cr>	Delay for 16 msec
DT16<cr>	Delay for 16 msec, identical to previous syntax
D24,<cr>	Delay for 24 msec beginning after motion stops
D300;<cr>	Delay 300 msec after reaching steady state speed
DM10000;<cr>	Delay for 10000 counts of commanded motion after reaching steady state speed
DR5000;<cr>	Delay for 5000 counts of the reference clock after reaching steady state speed. If in internal 192kHz mode this command would delay 5000 counts of the 192kHz clock. If in external mode this command would delay 5000 counts of the motion reference bus.

6.9.6 E - Exit Command

Purpose: **return from subroutine to statement after function call**

Syntax: E [<condition>] <cr>
E <op> <position> [<direction>] <cr>

Purpose: Exit (return from subroutine to statement after function call) either:
- unconditionally
- based on a specified machine input condition (up to 8 inputs high or low)
- based on a specified test of absolute position

<condition> in the format <mask> [/<hex>], defining the required state of the general purpose inputs IN1'..IN8' IN10'..IN80' for the Exit command to execute. IF the specified input condition is true, THEN the command following the specified program label will be executed ELSE execution will continue with the following MPL command. IF <condition> is not specified, THEN MPL program execution unconditionally returns or exits.

<op> the operator specifying greater than or less than <position> as the condition to be tested against:

> greater than the specified absolute position. If <op> is > then the exit will be executed if the current absolute position is greater than <position>. "Greater than" is defined as "more positive than the current position", or further in the plus (+) direction of travel.

< less than the specified absolute position. If <op> is < then the exit will be executed if the current absolute position is less than <position>. "Less than" is defined as "more negative than the current position", or further in the minus (-) direction of travel.

<position> integer (0 to 1,073,741,823) counts specifying the absolute position of the system to test against. Default is 0 if not specified.

<direction> + to specify <position> as a positive integer (default if not specified)
- to specify <position> as a negative integer

Examples:		
	E<cr>	Unconditional exit (return from subroutine)
	E8<cr>	Exit (return) if input IN8' is active (low)
	E3/0<cr>	Exit (return) if inputs IN1' and IN2' are inactive (high)
	E>2000+<cr>	Exit the subroutine if the PMC-based system is at an absolute position greater than 2000 counts in the positive direction.

Note: This command is valid only in program mode.

6.9.7 F - Function Command

Purpose: **call a subroutine**

Syntax: F <label> [<condition>] <cr> |
F <label> <op> <position> [<direction>] <cr>

Purpose: Call a function (subroutine) either:
- unconditionally
- based on a specified machine input condition (up to 8 inputs high or low)
- based on a specified test of absolute position

<label> MPL program label (see @ command description)

<condition> in the format <mask> [/<hex>], defining the required state of the general purpose inputs IN1'..IN8' IN10'..IN80' for the Function command to execute. IF the specified input condition is true, THEN the command following the specified program label will be executed ELSE execution will continue with the following MPL command. IF <condition> is not specified, THEN MPL program execution is unconditionally transferred to the specified MPL program label.

<op> the operator specifying greater than or less than <position> as the condition to be tested against:

> greater than the specified absolute position. If <op> is > then the branch will be done if the current absolute position is greater than <position>. "Greater than" is defined as "more positive than the current position", or further in the plus (+) direction of travel.

< less than the specified absolute position. If <op> is < then the branch will be done if the current absolute position is less than <position>. "Less than" is defined as "more negative than the current position", or further in the minus (-) direction of travel.

<position> integer (0 to 1,073,741,823) counts specifying the absolute position to test against. Default is 0 if not specified.

<direction> + to specify <position> as a positive integer (default)
- to specify <position> as a negative integer

Examples:

FE8<cr>	Call subroutine "E" if input IN8' is low
FE8/0<cr>	Call subroutine "E" if input IN8' is high
FE3/0<cr>	Call subroutine "E" if inputs IN1' and IN2' are high
FE13/0<cr>	Call subroutine "E" if input IN10', IN1' and IN2' are high
FA>200+<cr>	Call subroutine "A" if the PMC-based system is at an absolute position greater than 200 counts in the positive direction.

CAUTION: Nesting of functions is supported only three levels deep.

6.9.8 G - Go Command

Purpose: **move to the specified absolute position of the system**

Syntax: G <position><direction> [<sync>] <cr> | G <display> [<time>] <cr>

<position> integer (0 to 1,073,741,823) counts specifying the absolute position of system (default: 0)

<direction>
- \+ specify positive sign and perform motion calculations (system must be at rest before this character is entered)
- \- specify negative sign and perform motion calculations (system must be at rest before this character is entered)

<sync>
- , synchronizing character which causes the PMC to wait for the system motion to stop
- ; synchronizing character which causes the PMC to wait for the system motion to reach constant speed or complete.
- : synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

<display>
- ! display the current absolute position of the system. This is the commanded position adjusted by the current position error.
- ? display the currently commanded absolute position of the system
- * stop system motion
- % repeatedly display the current position (G!) until an SCI character is received.

<time> the rate in msec at which the % output is repeated. (default: 100)

<cr> move to the specified absolute position <position> using the motion parameters in the motion buffer. The direction of travel is determined by the current absolute position and the new commanded position.

Examples:

G!	Display the present absolute position of the system
G+<cr>	Go to the absolute zero position of the system
G200-<cr>	Move to absolute position -200
G*	Stop system motion
G%<cr>	Display current absolute position every 100 msec.

6.9.9 H - Home Command

Purpose: **move to the nearest encoder reference or sensor**

Syntax: H [<speed>] [<sync>] <direction> | H <display> [<time>] <cr>
H [<relative> <speed>] <cr>

<speed> integer (1 to 1,920) in 100 Hz units specifying the homing speed (default: 2.0 kHz); See Section 6.3 for other ranges.

<direction>
- + move at the homing rate in the positive direction until encoder reference or sensor is detected (see EXTERNAL STOP SELECT bit of Y register)
- - move at the homing rate in the negative direction until encoder reference or sensor is detected (see EXTERNAL STOP SELECT bit of Y REGISTER)
- * stop system motion

<sync>
- , synchronizing character which causes the PMC to wait for the system motion to stop
- ; synchronizing character which causes the PMC to wait for the system motion to achieve a steady state non-zero speed
- : synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

<display>
- ! display current system speed
- ? display last entered home rate
- % repeatedly display the current system speed until an SCI character is received.

<time> the rate in msec at which the % output is repeated. (default=100)

<relative>
- P increase the magnitude of the speed by the specified <speed>
- M decrease the magnitude of the speed by the specified <speed>

Examples:

H15+	Move in the positive direction to the home position at a velocity of 1.5 kHz
H,-	Wait for system to come to rest before homing in the negative direction
HM4<cr>	Decrease the magnitude of the homing speed by .4 kHz
H*	Stop system motion

6.9.10 I - Index Command

Purpose: **move the specified distance from the current position**

Syntax:
I [<distance>] #<direction># [<sync>] <cr>
I [<relative> <distance>] #<direction># [<sync>] <cr>
I <display>[<time>] <cr>

<distance> integer (1 to 2,147,483,647) specifying the relative distance to move (default: 500 counts); If this distance, designated optional above, is not specified, the system will move the distance that is currently specified in the motion buffer.

<direction>
- + move the specified relative distance in the positive direction using the acceleration and velocity parameters in the motion buffer
- - move the specified relative distance in the negative direction using the acceleration and velocity parameters in the motion buffer
- * stop system motion

<sync>
- , synchronizing character which causes the PMC to wait for the system motion to stop
- ; synchronizing character which causes the PMC to wait for the system motion to reach a constant speed or complete motion
- : synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

<display>
- ! display the distance remaining in the current or last index
- ? display the last entered index distance
- % repeatedly display the remaining distance in the current or last index (repeating I! output) until an SCI character is received.

<time> the rate in msec at which the % output is repeated.

<relative>
- P increase the magnitude of the distance by the specified <distance>
- M decrease the magnitude of the distance by the specified <distance>

Examples:

I250<cr>	Set the index distance in the motion buffer to 250 counts
I+	Index the system the previously set distance in the positive direction
I,-	Wait for last motion to end; index the previously set distance in the negative direction
I300,+	After the last motion is complete; index the system 300 counts in the positive direction
I,+,-	Wait for last motion to end; index in the positive direction; after this motion is stopped; index in the negative direction
IP300<cr>	Increase the magnitude of the index distance in the motion buffer by 300 counts
I!	Display the number of remaining counts in the current move
I?	Display the previously specified index distance
I*	Stop the current motion
I%200<cr>	Display the number of remaining counts in the current move. Update the display every 200 msec.

6.9.11 J - Jog Command

Purpose: **move at the specified jog speed**

Syntax: J [<speed>] #<direction># [<sync>] <cr> | J <display> [<time>] <cr>
J [<relative> <speed>] #<direction># [<sync>] <cr>

<speed> integer (1 to 1,920) in 100 Hz units specifying the jog rate (default: 10.0 kHz); See Section 6.3 for other ranges.

<direction>
- + jog in the positive direction at the specified speed; If the speed is not specified, the jog speed in the motion buffer will be used.
- - jog in the negative direction at the specified speed; If the speed is not specified, the jog speed in the motion buffer will be used.
- * stop system motion; System motion can be stopped by typing any character other than , or ; or <cr> if the PMC is running and still in the middle of a J command.

<sync>
- , synchronizing character which causes the PMC to wait for the system motion to stop
- ; synchronizing character which causes the PMC to wait for the system motion to reach a constant speed or be completed
- : synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

<display>
- ! display current system speed
- ? display last entered jog speed
- % repeatedly display the current system speed until an SCI character is received

<time> the rate in msec at which the % output is repeated

<relative>
- P increase the magnitude of the speed by the specified <speed>
- M decrease the magnitude of the speed by the specified <speed>

Note: The acceleration rate and jog speed can be changed while a jog motion is in progress by entering the new values and initiating another jog command.

Examples:

J!	Display the current system speed
J?	Display last entered jog speed
J36,+	Wait until any motion which may be underway is complete, and then jog in the positive direction at 3.6 kHz
J+	Jog in the positive direction at previously specified jog speed
J,-	Wait for the system to come to rest before jogging in negative direction
JM100<cr>	Decrease the magnitude of the system jog rate by 100kHz.
JP10-	Increase the magnitude of the system jog rate by 10kHz and start motion in the negative direction
J*	Stop system motion
J,-*+*-	Wait until end of last motion; jog in negative direction; stop; jog in positive direction; stop; continue jogging in negative direction

6.9.12 K - Kill Command

Purpose: **kill any system motion unconditionally**

Syntax: K [<time>] <cr>

<time> integer (0 to 65,535) in milliseconds specifying the time allowed for system deceleration at the AL rate, if the system has not stopped after the time has expired, an immediate stop will be commanded.

Examples:

K1000<cr>	**Kill any system motion after one second is allowed for deceleration**
K<cr>	**Kill system motion immediately**

6.9.13 L - Loop Command

Purpose: **transfer MPL execution to a program label a number of times**

Syntax:
L [<label> <count>] <cr>
L <label> [<loop counter ID>] <count> <cr>

<label> see @ label command description
<count> integer (0 to 65,535) number of times to loop to the specified label

Note: When the loop command is used from interactive mode, the <count> given on the command is used to replace the <count> on the first loop command encountered in the program specified by the <label>. For example, if you have the following program

```
@T
I,+
LT9
E
```

then executing the command **LT999** <cr> from interactive mode will loop program T 999 times, resulting in 1000 indexes being performed. If, however, you executed the above program using the B command, by entering **BT**, then you would get 10 indexes performed. Ten indexes are performed because the first one was performed before the Loop command was encountered, and Looping to program label T nine times therefore results in a total of 10 indexes.

An **L<cr>** will reset the loop counter to 0. This feature is only needed when you are executing a program and that program exits from a loop before the loop counter reaches 0. This can happen using the following program:

```
@G
I,+
BW1
LG2999
@W
I,-
LW19
```

The routine 'G' does 3000 indexes in the + direction as long as general purpose machine input IN1' stays high. If IN1' stays high, then when all 3000 indexes are done the 'W'

routine is executed. If IN1' goes low, then the program branches to routine 'W', having executed less then 3000 indexes. In this case, if the loop counter is not reset then the 'W' loop will not function correctly.

<loop counter ID>

the character X, Y, or Z which specifies the loop counter to be used. This allows loops to be nested up to three levels. As an example of nesting loops consider the following program:

@B	Label 'B'
I1000,+	Index 1000 counts in the positive direction after motion is complete
@C	Label 'C'
I100,-	Index 100 counts in the negative direction after motion is complete
@D	Label 'D'
I50,-	Index 50 counts in the negative direction after motion is complete
LDY9	Loop to label 'D' 9 times
LCX19	Loop to label 'C' 19 times
LB4	Loop to label 'B' 4 times. Note that loop counter id is only required on nested loops.
E	Exit unconditionally

Example: LB20<cr> Loop back to label 'B' 20 times

CAUTION: Program loops cannot be nested unless loop counter IDs are used.

6.9.14 N - Normalize Command

Purpose: **define the current physical position or reset PMC**

Syntax: N [<position>] [<sync>] <direction> | N <cr>

<position> integer (0 to 1,073,741,823) counts specifying the absolute position of system (default: 0)

<sync>
- , synchronizing character waits for the system motion to stop
- ; synchronizing character waits for the system to reach a constant speed or complete motion
- : synchronizing character which causes the PMC to wait for the system motion to complete steady state speed

<direction>
- + set absolute position counter to plus <position>
- - set absolute position counter to minus <position>
- * PMC software reset

<cr> Initiate Serial Communications Interface autobaud sequence. See Section 5.1 for information on the autobauding feature.

Note: The system must be at rest before <direction> can be entered.

Examples:

N2000,+	Wait for system motion to stop; and then set the absolute position counter to +2000
N*	Software reset PMC
N<cr><cr><cr>	Select serial 19,200 baud SCI

6.9.14a NC - Checksum on Non-volatile memory

Purpose: **calculate, display and verify checksums on non-volatile memory**

Syntax: NC <terminator>

<terminator>
- <cr> Perform a checksum verify. Generates an error flash and changes the prompt if the verify fails, otherwise it returns a prompt
- ? Displays checksums stored in memory
- ! Calculates and displays checksums. Displays what the PMC thinks checksums should be
- * Recalculates and updates checksums. Any existing errors are cleared, LED assumes normal yellow color and prompt is returned to normal

6.9.15 O - Output Command

Purpose: **set general purpose machine outputs**

Syntax: O <mask> [/<hex>] <cr> | O <display>

<mask> is a four bit hexadecimal number that specifies which machine output bits are to be manipulated or changed. Mask bits that are set to a one specify bits that are to be manipulated or changed. Mask bits that are set to zero specify the machine outputs that are to be unchanged.

<hex> is a four bit hexadecimal number that specifies the value that should be placed on the machine output. (The default if no <hex> parameter is specified is F_H which will cause all bits selected by the mask to be turned on.) A bit set to a one defines an active (low) signal level. A bit set to zero defines an inactive (high) signal level.

<display> ? display the last entered state of the general purpose machine outputs. A two digit hexadecimal value will be returned by the PMC; The first digit will be 0, and the second digit will indicate the current state of the machine outputs.

! same as ?

Note: With binary communications enabled, if the top bit of the "O" command is set, only two additional bytes will be required by the PMC. The first byte will be a number representing the mask and the second byte will be a number representing the output level for each bit.

Examples:

OC<cr>	Specify output pattern C_H; This pattern specifies that outputs OUT8' and OUT4' are active (low).
OC/0<cr>	Specify that outputs OUT8' and OUT4' are inactive (high)
O3/2	Specify that OUT2' is active and OUT1' is inactive.
O?	Display the current state of the machine outputs
O!	Display the current state of the machine outputs

6.9.16 P - Program Command

Purpose: **enter, edit or examine motion programs**

Syntax: P <program> #<text># | P <label> #<text>#

<program> { initiate programming at beginning of Program Buffer; The program buffer is the area in either or non-volatile RAM which contains MPL commands.

<cr> initiate programming at the end of the Program Buffer

? display the Program Buffer from the beginning, a line at a time; A linefeed character will display the next line in the buffer. A backspace or delete will display the previous line. The ASCII ESC($1B_H$) character will terminate the output. No changes to the Program Buffer are allowed in this mode.

! display the entire Program Buffer without any further input required. An ESC will terminate this mode.

<text> all printing ASCII characters other than a space are entered directly into the Program Buffer; If an illegal character is received, it will be ignored and the PMC will send a BELL.

<label> see @ label command description; Initiate programming at the beginning of the program with this label.

Examples: **P?** display the first command of the Program Buffer and display each additional command by entering a linefeed character until the ESC key is entered or the last command is displayed

P<cr> add program text at the end of the Program Buffer until the Program Buffer is full or the programming mode is terminated by typing an ESC key.

Note: See Section 6.4 for a description of editing functions to be used during Program Mode.

6.9.16a P - Binary Programming Command

The binary programming command is designed for compact programming of a PMC from a host computer interface. It is used by first enabling binary communications using the Z status register, bits 6 and 7 (see section 6.17).

Syntax: P <label>

<label> is required, and when it is found the PMC will output an @ character. The "{" character may be sent, in which case the programming will begin at the beginning of the program buffer. Once this mode is entered only the following characters are allowed:

<lf> move to the first character of the next line. No output.

<tab> echo current character and move to the next character.

<esc> terminate programming mode

any other character overwrites the program buffer

6.9.17 S - Set or Show Command

The group of **Set or Show** commands are widely useful for configuring a motion control system or displaying system status. The **S** command enables the user to select from the following options and also provides an ability to display system status information:

Command	Description
SB	Set baud rate for the Serial Communications Interface
SC	Show the system conditions (machine inputs)
SL	Set the software limits
SM	Select the "mode" of system operation: idle mode, velocity mode, position mode with position summing junction (PSJ) reset, position mode without position summing junction (PSJ) reset, and master axis controller mode.
SP	Show the last program label passed
SS	Show the system status: in motion, at top velocity, direction of last (or current) motion and state of drive enable line
ST	Set the program buffer trace option
SW	Set the program buffer write protect option
SX	Set or examine the X status register
SY	Set or examine the Y status register
SZ	Set or examine the Z status register

6.9.17a SB - Set Baud Rate

Purpose: Set the baud rate for the Serial Communications Interface from an MPL program. The baud rate set with the **SB** command will "take effect" at the next cycle of power or a hard reset.

Syntax: **SB <baud> <cr> | SB <display>**

<baud> Value 0 to 8 specifying the baud rate.

0 Specifies PMC autobaud sequence. For the autobaud sequence, the number of carriage returns entered from the SCI determines the selected baud rate. On powerup, entering two carriage returns will select 19,200 baud. Three carriage returns will select 9600 baud, and so on according to the table below. Refer to Section 5.1 for a description of the autobaud capability.

1	38,400
2	19,200
3	9600
4	4800
5	2400
6	1200
7	600
8	300

<cr> set baud rate as specified by <baud>

<display> ? displays the last entered baud specification (to take effect at the next cycle of power or hard reset)

! displays the current <baud> value

Note: This command requires that bit 5 of the Z Status Register be set before it can be executed.

6.9.17b SC - Show System Condition Inputs

Purpose: **display the current system inputs**

Syntax: SC <display>

<display> ! Display the state of the machine inputs. A two digit hexadecimal number will be returned by the PMC
? same as !

Example: **SC! display the current value of the system inputs**

6.9.17c SL - Set Overtravel Limit Options

Purpose: **set absolute position software limits or polarity of hardware limits**

Syntax: SL <name> <position> [<sign>] <cr> | SL <display>

<name> specifies which limit is to be set
F Set the forward travel limit. Forward is defined as the direction of motion resulting from a + direction command when the direction invert bit is 0. This forward is the same direction of travel which the + hardware limit stops.
R Set the reverse travel limit
H Set the polarity of the hardware overtravel limits. An SLH0 command specifies that the hardware overtravel limit inputs are asserted for TTL low level input signals. SLH1 specifies that the hardware overtravel limit inputs are asserted for TTL high level input signals.

Note: To use hardware overtravel limits, the X Register Bit 1 must be set with the SX command.

<position> an integer (0 to 1,073,741,823) specifying the absolute position of the limit. Default value is 0.

<sign> + set software limit at plus <position>
- set software limit at minus <position>

<display> ? display current limit values with parameter names
! display current limit values without names

Note: Bit 1 of the Z register must be set to enable this feature

Examples:

SLF100000<cr>	Set the forward limit to position +100000
SLR50000-<cr>	Set the reverse limit to position -50000
SLR<cr>	Set the reverse limit to position 0
SL!	Display the current software limits

6.9.17d SM - Set System Mode

Purpose: **select mode for PMC operation**

Syntax: SM <mode> <cr> | SM <display>

<mode> 0 enter IDLE mode; In the IDLE Mode, both the position and velocity loops are disabled and the Servodrive Enable signal (SDRVEN shown in Appendix 8.4) is disabled. This signal, or its complement SDRVEN', can be used to disable the servodrive either through an output disable input signal, provided on some servo-drives, or using a solid state relay as shown in Appendix 8.1 and 8.2.

1 enter VELOCITY mode; In the VELOCITY Mode, the velocity loop is enabled, as is the SDRVEN signal (for enabling the servodrive). Any encoder signals received will be added or subtracted to the absolute position.

2 enter POSITION mode with PSJ Reset; In the POSITION Mode, both loops are enabled, and the SDRVEN signal is asserted. When POSITION Mode is entered with a SM2 command, any error count which may be present in the PSJ is cleared before enabling the servo loops. Note: The error count that is cleared is added to the current absolute position.

3 enter POSITION Mode without PSJ Reset; In the POSITION Mode, both loops are enabled, and the SDRVEN signal is asserted. When POSITION Mode is entered with a SM3 command, any error count which may be present in the PSJ will have an immediate effect on the system when the servo loops are enabled, causing the system position to "jump" to the position where the error count will be cancelled.

4 enter MASTER AXIS mode; In the MASTER AXIS Mode, the velocity loop is enabled, as is the SDRVEN signal (for enabling the servo-drive). Any encoder signals received will be ignored. The absolute position will always be identical to the commanded position. This mode is particulary useful when no motor is hooked up.

<cr> set Mode as specified by <mode>

<display> ! display current system status information; The PMC will return two hexadecimal digits, with the first digit indicative of system motion status and the second digit the current system mode. See data below.

? display the last selected mode; The PMC will return two hexadecimal digits, with the first digit always returning zero and the second digit indicative of the last selected mode. See data below.

First Hex Digit Bit Assignments

Bit 7 reserved
Bit 6 PSJ OVERFLOW indicates whether or not the Position Summing Junction has overflowed and set the Overflow latch. 1 => PSJ OVERFLOW
Bit 5 MOTION indicates whether or not the system is in motion. 1 => IN MOTION
Bit 4 DIRECTION of the last (or current, if motion is in progress) motion; 1 => FORWARD

Second Digit Values
0 Idle Mode
1 Velocity Mode
2 Position Mode
4 Master Mode

Note: If a PSJ overflow takes place, the PMC automatically enters IDLE Mode (with the Servodrive Enable signal (SDRVEN) disabled. Clearing this fault condition with an SM2 or SM3 command will automatically clear the PSJ.

6.9.17e SP - Show Last Label Passed

Purpose: **display the last label passed**

Syntax: SP <display>

<display> ? displays the last label passed as one ascii character followed by a number indicating the number of commands executed since the label was passed.
! same as ?

Example: **SP!** display the last label passed

6.9.17f SS - Show System Status

Purpose: **display the motion profile register which provides information on the system status**

Syntax: SS <display>

<display> ? displays the motion profile register as one hex digit
! same as ?

Bit 7 DRVON indicates whether or not the servodrive is enabled 1 => DRIVE ON
Bit 6 Reserved
Bit 5 Reserved
Bit 4 Reserved
Bit 3 Reserved
Bit 2 Direction of the last (or current) motion if motion in progress. 1 => FORWARD
Bit 1 Top Velocity indicates whether or not the system is currently at top velocity. 1 => TOP VELOCITY
Bit 0 Motion indicates whether or not the system is in motion. 1 => IN MOTION

Example: SS! display the system snapshot register

6.9.17g ST - Set Program Trace Option

Purpose: **to enable or disable the program buffer trace option**

Syntax: ST <hex> | ST <display>

<hex>
- 0 disable program buffer trace option
- 1 enables program buffer trace option. Displays each MPL command at the SCI (if active) as it is executed.

<display>
- ! displays a two digit hexadecimal number indicating the current state of program buffer trace option
- ? same as !

Example:
- **ST1** enable the program trace option
- **ST!** display the current state of the trace option

6.9.17h SW - Set Program Buffer Write Enable

Purpose: **to enable or disable the write protection on the program buffer contents**

Syntax: SW <hex> | SW <display>

<hex>
- 0 protects program buffer contents from editing. Access to program buffer contents is allowed for viewing via any of the **P** commands but any attempt to change the contents of the program buffer will result in a C0 error.
- 1 enable editing of the program buffer

<display>
- ! displays a two digit hexadecimal number indicating the current state of program buffer write protect option
- ? same as !

Example: SW1 enable program buffer editing

6.9.17i SX - Set X Status Register

Purpose: **to select system parameters by specifying the condition of the X status register**

Syntax: SX <hex> <cr> | SX <display>

<hex> hexadecimal representation of the selected byte:

Bit 7 SMOOTH ACCELERATION PROFILE provides extremely smooth acceleration for all PMC motion commands. Note that only the linear acceleration profile is supported while using this mode. (1=on)

The normal PMC acceleration ramp has a minimum step size equal to 1/64 of the top velocity value, which in the case of long acceleration times and high top velocities can cause undesirable acceleration profiles. The smooth acceleration profile has a minimum step size which is independent of the top velocity. The minimum step size is approximately 11 Hz when in the 48 kHz velocity range, 47 Hz in the 192 kHz velocity range and 94 Hz when in the 384 kHz velocity range. The number of steps in acceleration is dependent on the top velocity value. The time between steps is dependent on the top velocity and acceleration values.

Bit 6 MOTION BUS SLAVE selects the Motion Reference Bus as the master reference for creating motion instead of the internal crystal controlled clock; (1=on)

Bit 5 ALTERNATE REFERENCE ENABLE causes each odd motion reference pulse to be sent directly to the Position Summing Junction and each even motion reference pulse to be used as the internal distance reference; This output is useful with the MOTION BUS SLAVE bit for establishing a nominal motor speed with respect to other moving machinery. (1=on)

Bit 4 MOTION BUS MASTER causes system to become motion bus master by supplying its motion reference pulses to the Motion Reference Bus. (1=on)

Bits 3-2 VELOCITY RANGE SELECT selects velocity range as follows:

RANGE	Bit 3	Bit 2
48k Hz	0	0
reserved	0	1
192k Hz	1	0
384k Hz	1	1

Bit 1 ENABLE LIMITS enables Machine I/O inputs IN4' and IN8' to be used as - and + limit switch inputs respectively. The AL acceleration value is used to stop motion.

Bit 0 DIRECTION INVERT transposes the meaning of + and - in motion commands.

<display> ? displays the status registers in the order XYZ; The data is displayed with labels; eg. X=00 Y=00 Z=00

! displays the status registers in the order XYZ; The data is displayed with six consecutive ASCII hex digits. e.g. 070000

Examples:

SX88<cr> Select smooth acceleration mode in 192 kHz range
SX08<cr> Terminate smooth acceleration mode in 192 kHz range
SX18<cr> Become a "motion bus master" in the 192 kHz range
SX4C<cr> Become a "motion bus slave" in the 384 kHz range
SX02<cr> Enable limits in the 48 kHz range

6.9.17j SY - Set Y Status Register

Purpose: **to select system parameters by specifying the condition of the Y status register. See Section 6.5 for how these parameters can be used to create unique motion profiles.**

Syntax: SY <hex> <cr> | SY <display>

<hex> hexadecimal representation of the selected byte:

Bit 7-6 ACCEL PROFILE SELECT specifies an acceleration profile by setting bits 6 & 7 as follows:

Bit 7	Bit 6		type
0	0	-	linear
0	1	-	s-curve (polynomial)
1	0	-	parabolic
1	1	-	reserved

Bit 5 EXTERNAL START causes a motion to start upon receiving an external signal. Bit 4 will indicate which signal will initiate motion.

Bit 4 EXTERNAL START SELECT specifies either the machine sensor input signal (SENSIN) or the encoder reference (ENCR) to start a motion. (1=machine sensor, 0=encoder reference) Bit 4 will be ignored unless Bit 5 (EXTERNAL START) is set.

Bit 3 EXTERNAL DECEL causes deceleration to occur on the machine sensor input (SENSIN) instead of a calculated distance (after full speed is attained and the distance traveled is greater than the currently specified index distance). Ordinarily, deceleration is initiated when the remaining distance is equal to the acceleration distance. (1=on)

Bit 2 EXTERNAL STOP SELECT specifies either the machine sensor input signal (SENSIN) or the encoder reference (ENCR) to stop motion during a home command or an INDEX EXTEND. (1=machine sensor, 0=encoder reference)

Bit 1 INDEX EXTEND specifies that speed should remain at the level set by the J command during deceleration rather than continuing to zero. INDEX EXTEND is used in conjunction with EXTERNAL STOP SELECT or with a machine input condition to stop the motion. (1=on)

Bit 0 SHARP JOG STOP selects a sharp (immediate) stop for jog deceleration rather than the deceleration rate specified by the A command. (1=on)

Examples: For examples of <u>some</u> of the special motions which can be created by modifying the Y Register, consult Section 6.5.

<u>Note</u>: The Y Register is in the motion buffer and therefore, altering it during a motion will only effect the next commanded motion.

6.9.17k SZ - Set Z Status Register

Purpose: **to select system parameters by specifying the condition of the Z status register**

Syntax: SZ <hex> <cr> | SZ <display>

<hex> hexadecimal representation of the selected byte:

Bit 7-6 Bits 6 & 7 of the X Register are used to select one of the available COMMUNICATIONS MODES as follow:

Bit 7	Bit 6	Communications Mode		
0	0	Decimal In	-	Decimal Out
0	1	Hexadecimal In	-	Hexadecimal Out
1	0	Binary In	-	Binary Out
1	1	Hexadecimal In	-	Binary Out

The syntax of MPL commands which may have their parameters changed in binary input mode is listed below:

A <null><byte><byte>
AL <null><byte><byte>
AS <null><byte><byte>
AQ <null><byte><byte>
D <null><byte><byte><cr>
DM <null><byte><byte><byte><byte><cr>
DR <null><byte><byte><byte><byte><cr>
G <null><byte><byte><byte><byte><cr>
H <null><byte><byte><term>
I <null><byte><byte><byte><byte><term>
J <null><byte><byte><term>
K <null><byte><byte><cr>
N <null><byte><byte><byte><byte>
S <null><byte><byte><byte>
SL <null><byte><byte><byte><byte><byte><byte><byte><byte>
T <null><byte><byte><byte><byte><byte><byte>
V <null><byte><byte>

In the descriptions above:

<null> refers to the ASCII null character (00_H).

<cr> refers to the ASCII carriage return ($0D_H$).

<byte> refers to one byte of binary data, and for the specification of numbers larger than one byte, the numbers are normally represented in unsigned binary format with the most significant byte first. An exception to this is the G command, in which the number is expressed in two's complement binary notation, with the most significant byte first.

<term> represents any of the motion terminators (+,-,*). In the SL command, the first four bytes represent the value of the upper overtravel limit, and the second four bytes represent the value of the lower overtravel limit. In the T command the six bytes represent the six gain and compensation adjustments in the order P,V,F,X,CP, & CV.

Bit 5 PARAMETER WRITE PROTECT must be set to execute MPL's = command which labels an individual motion axis or to change the baud rate. If this bit is not set (default), a new Axis-ID or baud rate cannot be assigned.

Bit 4 ADDRESS LINE SELECTION disables ADR1'-ADR8' as address lines accessible from the Machine I/O interface. When this bit is asserted (1=on), ADR1'-ADR8' are used exclusively as machine inputs and only two program labels (@0 and @1) can be addressed using SEL'.

Bit 3 IN MOTION. Enables OUT8' to indicate that a motion is being commanded. (1=on)

Bit 2 FAULT. Enables OUT4' to indicate an error has been generated by the PMC. This fault output is cleared upon entry of the next MPL command. (1=on)

Bit 1 Enables SOFTWARE LIMITS. (1=on)

Bit 0 NO ECHO prevents echo of SCI characters. (1=on)

Examples:

SZ01<cr>	Select NO ECHO mode for communications
SZ80<cr>	Select binary in/binary out communications
SZ20<cr>	Enable baud rate and axis id parameter editing
SZ02<cr>	Enable the use of software limits

6.9.18 T - Tuning Command

Purpose: **Tune the Servo Loops or examine the current tuning parameters**

Syntax: T #<register> [<value>] <sign># | T [E] <display> [<time>]

<register>
- P select position loop gain
- V select velocity loop gain
- F select feedforward gain
- X select external output gain
- CP select position loop compensation
- CV select velocity loop compensation

<value> indicates a value to use in the specified operation; This can be an absolute value to be used for a gain or compensation, or an incremental amount to add to or subtract from the currently specified value. The difference is specified by the <sign>. For gain values the <value> is entered as a decimal value (ASCII hex if hex communications is enabled). The compensation values are always entered as hex arguments.

<sign>
- \+ increment selected item by <value>
- \- decrement selected item by <value>

Note: For + and - , the <value> defaults to 1 if not specified.

<cr> set selected item to <value> and terminate command

<display>
- ? display current values with labels
- ! display current values without labels
- % repeatedly display the current error.

Note: On labelled output (?) the gain values are output in decimal unless hex communications are enabled. The compensation values are always output in ASCII hex. On unlabelled output (!) all values are always output in ASCII hex.

Compensation value assignments:

Velocity and Position Loop compensation values can be chosen from a range of 0_H to F_H. The value 0 disables the velocity loop integral + proportional compensator (selects 0 Hz. frequency for the "zero break frequency"). Values from 1 to F select higher and higher break frequencies.

Examples:

TP17<cr>	Set position gain to 17.
TV34<cr>	Set velocity gain to 34.
TP+V5-	Increment position gain by 1 and decrement velocity gain by 5.
TCP0<cr>	Set position loop for proportional gain only
TCVA<cr>	Set velocity loop for integral + proportional gain.
TE!	Display the current position error.
TE?	Display the normalization error (error cancelled when the last Normalize command was executed.
TE%	Repeatedly display the current position error.

6.9.19 U - Until Command

Purpose: **suspend MPL operation until a specified condition is true**

Syntax: U <condition> <cr>
U <op> <position> [<direction>] <cr>

Purpose: Wait before executing the next command in the program until either:
-a specified machine input condition (up to 8 inputs high or low)
-a specified test of absolute position is true

<condition> in the format <mask> [/<hex>], defining the required state of the general purpose inputs IN1'..IN8' IN10'..IN80' for the Until command to execute. IF the specified input condition is true, THEN execution will continue with the following MPL command, ELSE execution will halt on this statement until input condition is true.

<op> the operator specifying greater than or less than <position> as the condition to be tested against

> greater than the specified absolute position. If <op> is > then the wait (Until) command will be terminated when the current absolute position is greater than <position>. "Greater than" is defined as "more positive than the current position", or further in the plus (+) direction of travel.

< less than the specified absolute position. If <op> is < then the wait (Until) command will be terminated when the current absolute position is less than <position>. "Less than" is defined as "more negative than the current position", or further in the minus (-) direction of travel.

<position> integer (0 to 1,073,741,823) counts specifying the absolute position of the system to test against. Default is 0 if not specified.

<direction> + to specify <position> as a positive integer (default if not specified)
- to specify <position> as a negative integer

Examples:

U1<cr>	Wait until input IN1' is active (low) before executing the next command
U2/0<cr>	Wait until input IN2' is inactive (high) before executing the next command
U81<cr>	Wait until inputs IN80' and IN1' are active (low) before executing the next command
U>2000+<cr>	Wait until the PMC-based system moves to an absolute position greater than 2000 counts in the positive direction before executing the next command in the program.

Note: An ESCAPE character entered during the execution of this command will end this command with a D0 error (see Section 6.12), and STOP' asserted at the MIO will end this command with a D1 error.

6.9.20 V - Velocity Command

Purpose: **set or examine maximum velocity rate**

Syntax: V <speed> <cr> | V <display> [<time> <cr>]
V [<relative> <speed>] <cr>

<speed> integer (1 to 1920) in 100 Hz units specifying the maximum position encoder frequency to be commanded (default: 40.0 kHz); See Section 6.3 for other ranges.

<display>
! display current system speed
? display last entered velocity
% repeatedly display the current system speed.

<relative>
P increase the magnitude of the speed by the specified <speed> for the next index.
M decrease the magnitude of the speed by the specified <speed> for the next index.

<time> the repeat rate of the % output.

Examples:

V304<cr>	set velocity to 30.4 kHz for next motion
V?	display last entered acceleration rate
V!	display current speed
VP100<cr>	increase the magnitude of the velocity by 10 kHz for the next index
V%1000<cr>	display the current system speed and update the display every 1000 msec

6.9.21 = - Assign Motion Axis-ID

Purpose: **label motion axis**

Syntax: = <id> <cr> | = <display>

<id> Motion axis identifier; most displayable characters. Alphabetic characters must be upper case.
Caution: Using the "equals = " character as a motion Axis-ID will turn off bus communication support.

<display>
! display ID of the axis currently in use
? display ID value currently stored in program memory

Note: Before a new Axis-ID can be assigned, use the Set/Show command to set Z register, bit 5. This safety feature makes it unlikely that a new axis identifier will accidentally be assigned. See Section 6.7 for a discussion of multi-axis serial bus communications.

6.10 SYSTEM STATUS POLLING

System status polling is provided to allow access to important information during program execution. This information is acquired by sending a two character sequence to the PMC with which communications are established. These two character sequences perform the equivalent function that the indicated standard MPL command would if issued at the command level. The advantage of the system status polling commands is that the PMC does not have to be at command level for these commands to be used. Note: **The <sys poll> command must be lower case.**

Syntax: <attn> <sys poll>

<attn> Ctrl] (ASCII $1D_H$)

			(Equivalent MPL command)
<sys poll>	c	system inputs	(SC!)
	e	current following error	(TE!)
	g	current system position	(G!)
	h	hex communications checksum	
	i	distance remaining in motion	(I!)
	p	last label passed	(SP!)
	s	motion parameters	(SS!)
	v	current system velocity	(V!)
	x	current axis id	(=!)

6.11 CREATING COMPLEX MOTION PROFILES

Overview

The contour motion command provides the user with the ability to exactly control the motion of the motor including acceleration rates, velocities and decelerations. The contour motion command should be thought of as running on "ticks". The user specifies both the size of the "tick" and how far, in feedback counts, the motor should move during each "tick". The "ticks" define segments of motion.

External Mode Contour Motion

Operational Description: This scheme uses a host computer to control a motion reference source and to command each slave PMC, telling it how far to move during each "tick". The host must provide new data each "tick", until a terminate motion command is given. "Tick" information can be provided via the program buffer.

A "tick" is defined as specific number of external reference counts. The number of input counts/-"tick" is selectable by the user. This defines the tick as a distance and not a time. Since the maximum rate on the motion reference bus is 192 kHz a "tick" in this mode can be no less than 1.33333 msec.

Internal Mode Contour Motion

Operational Description: This scheme uses a host computer (or the PMCs program buffer) to command each PMC, telling it how far to move during each "tick". The host must provide new data each "tick", until a terminate motion command is given.

A "tick" is defined in time and selectable by the user. The distance that can be travelled in a segment is velocity range dependent as specified in the X status register.

Contour Motion Command

Syntax: C <tick> <count> <cr>

where: <tick> specifies the time or distance for this "tick". The units of this value are dependent on the X status register. The slave bit and velocity range determine the units. The 384kHz range is not supported and will cause an error to be generated when a C command is given and the 384 kHz range is selected.

<tick>	time/tick (192 kHz)	time/tick (48 kHz)	distance/tick (external mode)
1	1.3333 msec	5.333 msec	255 counts
2	2.6667	10.667	512
3	5.3333	21.333	1024
4	10.6667	42.666	2048
5	21.3333	85.333	4096
6	42.6667	170.667	8192
7	85.3333	341.333	16384
0	specifies this is the end of the contour motion		

The 0 must be entered as 1 hexadecimal character with no terminator or 1 binary byte with no terminator.

<count> number of counts to move in the next "tick". This is a signed 2's complement number with the sign indicating the direction. The maximum and minimum values are dependent on <tick>. Also dependent on <tick> is the number of hex digits needed to specify <count>. Both are shown in the following table:

<+ tick>	<-tick>	Max. value	# hex digits
1	9	+255	2
2	A	+512	3
3	B	+1024	3
4	C	+2048	3
5	D	+4096	4
6	E	+8192	4
7	F	+16384	4
0			0
G	H	(1.33ms at top speed)	0

If <count> > (max. value) or <count> < (min. value) is entered an A4 error message will result and motion will decelerate at the AQ deceleration rate. <count> must be entered as either 2, 3 or 4 hexadecimal characters with no terminators as shown or 2 binary bytes with no terminators. When <tick>=0 is entered no <count> can be entered.

If a <cr> is received before the proper number of hex digits are received the <cr> will be interpreted as a terminator for both the <count> value and also the C command line. An ESC character received during the <count> will be interpreted as an abort of this data entry and will output an A2 error and return to the MPL prompt level.

Binary data mode is requested by setting the binary enable bit of the Z register . Once a data mode is selected for an invocation of the C command it must be used until a <cr> is entered. (Translation: you can't mix hex and binary on the same line.)

Contour Motion Emergency Deceleration Rate

Syntax: AQ<value><cr> | AQ<display>

use: This value sets the deceleration rate to be used by the contour motion command when an error occurs as documented for the C command. This value is also used for STOP' input, Limit input and SPACE stopping.

<value> deceleration rate in units of kHz/sec in internal mode and in hundreds of counts of the motion reference bus in external mode. The range of value is 0 to 65535.

Comments:

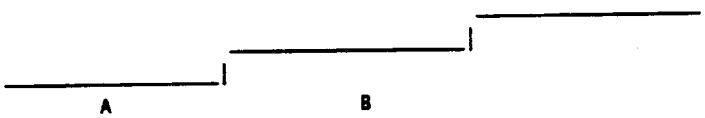
Multiple segments

The command syntax is structured to allow for multiple segments to be specified on each C command so as to reduce overhead and perform the fastest possible motion. The syntax also allows for exiting back to MPL so that all the power of MPL can be used when desired.

Host Synchronization

No segment buffering is supported. While a segment is being executed it is possible to enter the command, or data, for the next segment. To synchronize the host a prompt (}) will be output when the C command is ready to accept additional data. The C command will be ready to accept new data when the segment just commanded (or last commanded) is executing. While still on the same command line a prompt will be sent to the host between each data set. While waiting for the segment data to be used (ie. before the prompt is sent out) the PMC will ignore all characters sent to it except the **ESC** which causes an error and exits to command level and a <cr> which exits to the command level without an error. When leaving a C command and then entering it again, a prompt will be output as soon as the PMC is ready for another segment data. This will be either immediately upon entry to the command or there will be a delay until the next segment ends.

Consider the following example profile. Each step represents a new speed. Assume the Y axis is the speed (counts) and the X axis is the time, then each segment is defined with an ordered pair T_nY_n.



Starting motion is done with a CT_1Y_1 command. When the C is entered the PMC is ready for another segment so a prompt is sent out. Communications would look like (underlined characters are PMC output):

C}T1Y1}

The trailing } indicates that the PMC is ready for another segment. We would then send the next pair T2Y2. Once sent we would not get back a prompt until after the motion reaches point A, just after we go to speed Y2. Then we get the prompt back. When entered on the same command line it would look like:

C}T1Y1}T2Y2}T3Y3

No prompt will be output from the T3Y3 until point B is reached. The user can at this time enter a <cr> to return to the MPL command level. As long as he is in the command level the synchronizing prompt will not be output. Eventually the user will enter another C command. When this happens he will receive a prompt when the PMC is ready for additional data, which is after point B is reached. If the new C command is entered after point B, then the prompt will be output immediately. Otherwise, there will be a delay. If each segment is entered on a separate command line then the sequence might look like:

=}C}T1Y1}<cr> command for 1st segment.

=}C}T2Y2<cr> command for segment between points A and B. The prompt to be output after Y2 won't be output until after point A is reached. If the <cr> is entered before point A is reached the synchronizing prompt for the T2Y2 segment will not be output. The user should thus take care to not enter the data such that the Y2 data and <cr> fall on different sides of point A. (You are cutting it too close if this is a problem.)

=}C}T3Y3<cr> the 1st prompt after the C is output when the position is after point B, which may not be immediately when the C command is entered.

External Sensor

Support is provided for sensor start of the motion. The Y register is used to control sensor starts. Whenever the Y register has sensor start selected and a segment is entered from the MPL command level (C entered) then that segment will not begin until the sensor occurs.

If a sensor start is used on a segment within a profile (already in motion when sensor start requested) and the sensor does not occur exactly at the end of the current "tick" then the PMC will not change the hardware until the sensor occurs resulting in motion continuing at approximately the last entered speed.

Examples: (PMC output is underlined)

=}SY30	set sensor start on
=}C}T1Y1}T2Y2<cr>	start motion, sensor start. Command second segment, won't wait for sensor on second segment.
=}SY0	turn off sensor start
=}C}T3Y3<cr>	segment starts when previous segment ends

Contour Errors

Whenever a motion error occurs during a contour motion profile, deceleration will occur according to the AQ parameters to the best ability of the PMC. Be forewarned that because of the units assumed on the AQ command and the limited speed knowledge the deceleration rate may vary some from the desired value.

Whenever an invalid character is received while a number (hex or binary) is expected the C command will be terminated with an error message and deceleration will begin at the AQ rate.

Except in the case of a sensor start segment if a segment ever ends without new data being available, the PMC will begin a deceleration at the AQ rate. No # error message will be output when this happens.

Whenever an error occurs, if the FAULT output is enabled by the Z register then it will be turned on. This includes all the errors listed above, even those that do not output a # error message. This FAULT output could be used to stop a master in a slave environment or inform the host.

6.12 ERROR CODES AND EXCEPTION HANDLING

The PMC is designed to trap user errors and return error messages in standard formats. When the PMC is used in the host computer environment, it is the responsibility of the designer of the host computer software to handle these messages. Once an error is detected; the current command and mode are terminated {including programming mode} and if a host system or terminal has initialized the SCI, an error message is sent to it. The PMC then goes into a READY state, asserting the READY' line on the MIO, and sends a "prompt" to the SCI.

Error messages from the PMC are preceded with a number sign (#), and followed by a two character error code. A description of these error codes follows:

6.12.1 Syntax Error Codes

A0	Command not allowed in current velocity range.
A1	An invalid command has been used.
A2	An invalid terminator or designator has been used.
A3	An invalid loop counter ID has been used. Only X, Y, and Z are valid.
A4	The input value is out of the allowable range. See the "Ranges & Units" table for allowable values.
A5	Invalid address for the = (axis identifier) command has been entered. Valid addresses are printable ASCII characters other than the "space" character.
A6	An invalid HEX value, either a condition code or an output value, has been entered.
A7	The requested velocity range is not available.
A8	No valid program in the optional socket.
A9	The requested machine code is not recognized.

6.12.2 Motion Error Codes

B0	Command allowed only at rest or at top velocity.
B1	Index distance exceeded in external mode due to improper index accel-top speed specs
B2	Command not valid while the system is in motion. A motion designator or programming command was entered when the system was in motion.
B3	Motion cannot be initiated with STOP' low.
B4	Reserved.
B5	An ESCAPE character was received during a synchronization command.
B6	A motion command was entered with DRVOFF (Drive Off) asserted.
B7	Attempt to move past a software limit.
B8	Attempt to move forward with forward limit (+LIMIT) asserted.
B9	Attempt to move in reverse with reverse limit (-LIMIT) asserted.

6.12.3 Programming Error Codes

C0	Program buffer write protected. An SW0 command must be issued to disable the program buffer write protection.
C1	Program buffer overflow.
C2	Program label undefined.
C3	The program memory has a storage fault. The last entered programming character was not saved in program buffer due to hardware failure of the non-volatile RAM memory.
C4	An ESCAPE character was received during execution of an MPL program.
C5	Reserved.
C6	A Program command cannot be executed during program execution.
C7	Reserved.
C8	Attempt to modify non-volatile memory with active checksum error.
C9	Function calls nested too deep.

6.12.4 Miscellaneous Error Codes

D0	An ESCAPE character was received during a Delay command.
D1	STOP' signal at the Machine I/O Interface was asserted.
D2	Input operation aborted.
D3	Parameter Write Protected. Bit 5 of the Z register must be set to disable the write protect feature before a new Axis-ID or baud rate can be specified.
D4	Reserved.
D5	Reserved.
D6	Reserved.
D7	A SPACE character was received while in motion. Motion stopped.
D8	C segment entered during a segment stop.
D9	Position Summing Junction overflow.

MAINTENANCE

7.1 PREVENTIVE

No preventive maintenance procedures are required for the PMC family.

7.2 DEMAND

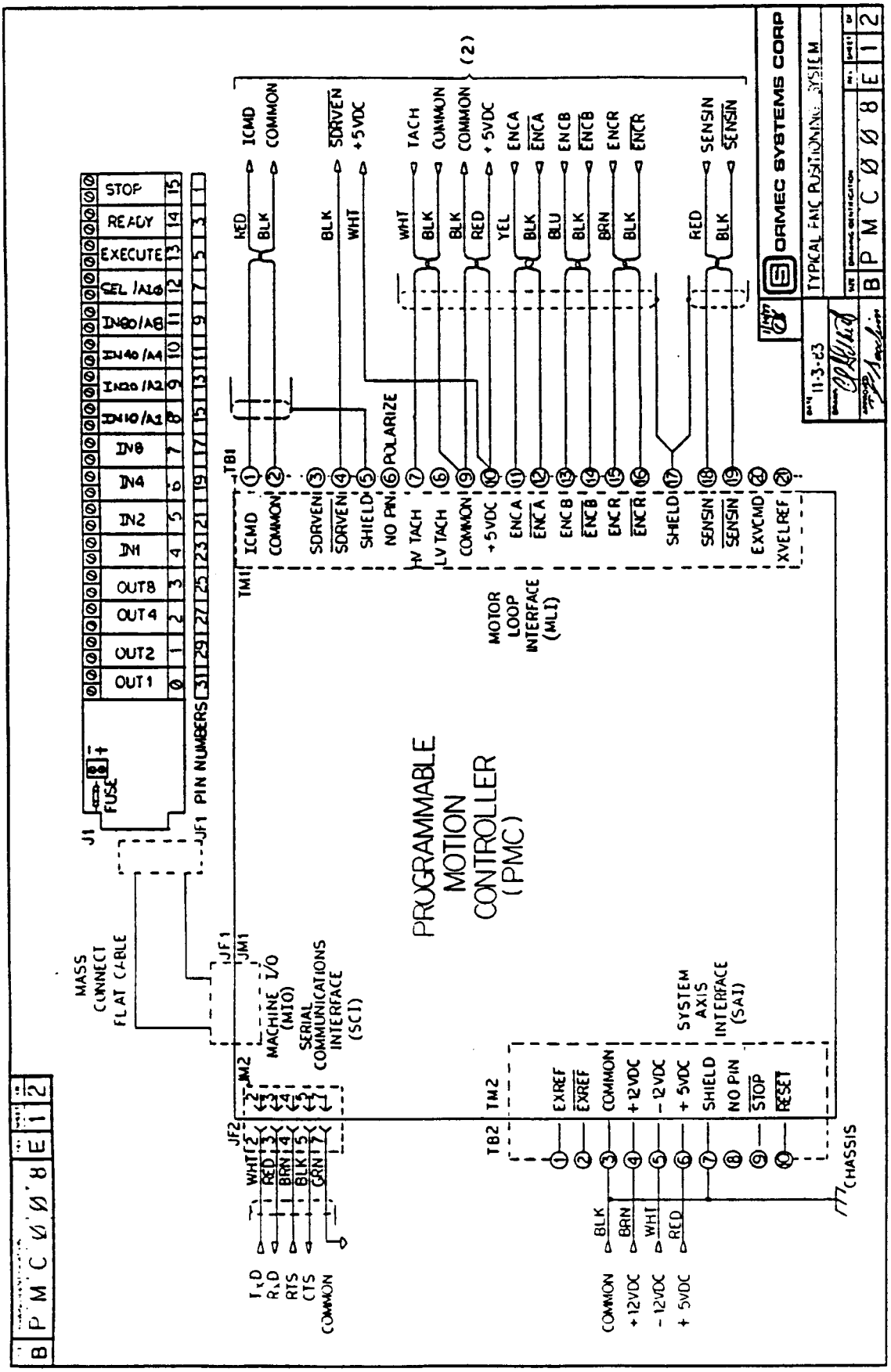
ORMEC equipment is designed modularly for simple onsite demand maintenance consisting of convenient module replacement. Most of the integrated circuits used for Input/Output are socketed for easy user replacement. If a problem occurs which is beyond the socketed I/O circuitry, the user should return the defective module for factory repair. The PMC series is designed with connector interfaces to make board replacement in the field simple and fast.

7.3 DIAGNOSTICS

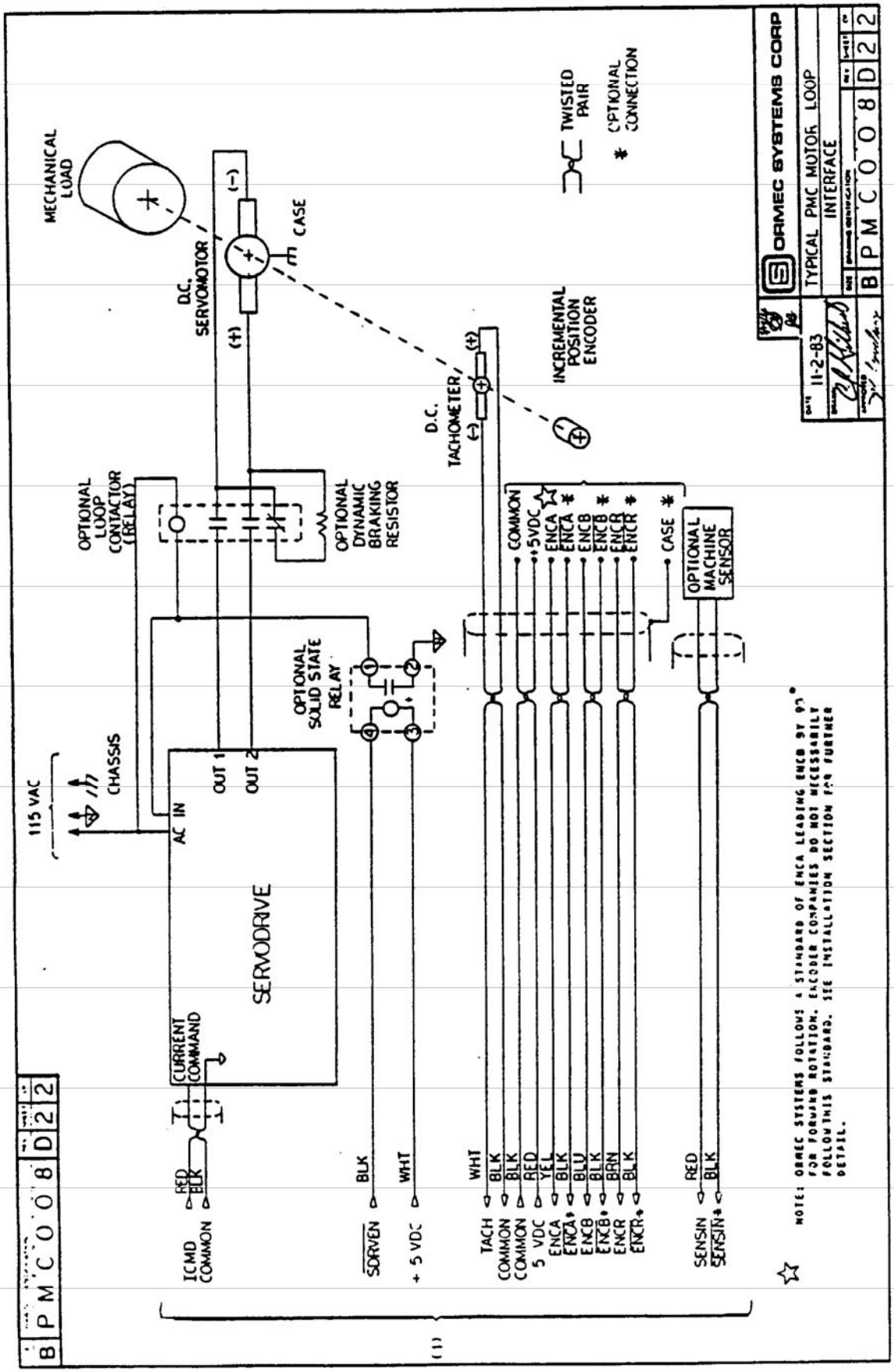
Each time the PMC powers up or is reset using the RESET' input signal, the CPU performs a set of diagnostic routines. These routines include testing both the RAM and the EPROM memory, and as these routines operate, they toggle the two color LED on the edge of the board between the Machine I/O and the Serial Communications Interface connectors. This LED has the following states:

Red	while RESET' is active
Green	while the RAM test operates (about 1 sec)
Red	while the EPROM test operates (about 1 sec)
toggling Red/Green	during normal system operation; This toggling, which looks YELLOW, is done at the 4 msec rate of the on-board real time clock, and since the LED is driven directly by the CPU, its continued toggling is dependent on the CPU's normal processing of the real time clock's interrupts.
Green 1 sec & Red 1/3 sec	RAM failure
Red 1 sec & Green 1/3 sec	EPROM failure

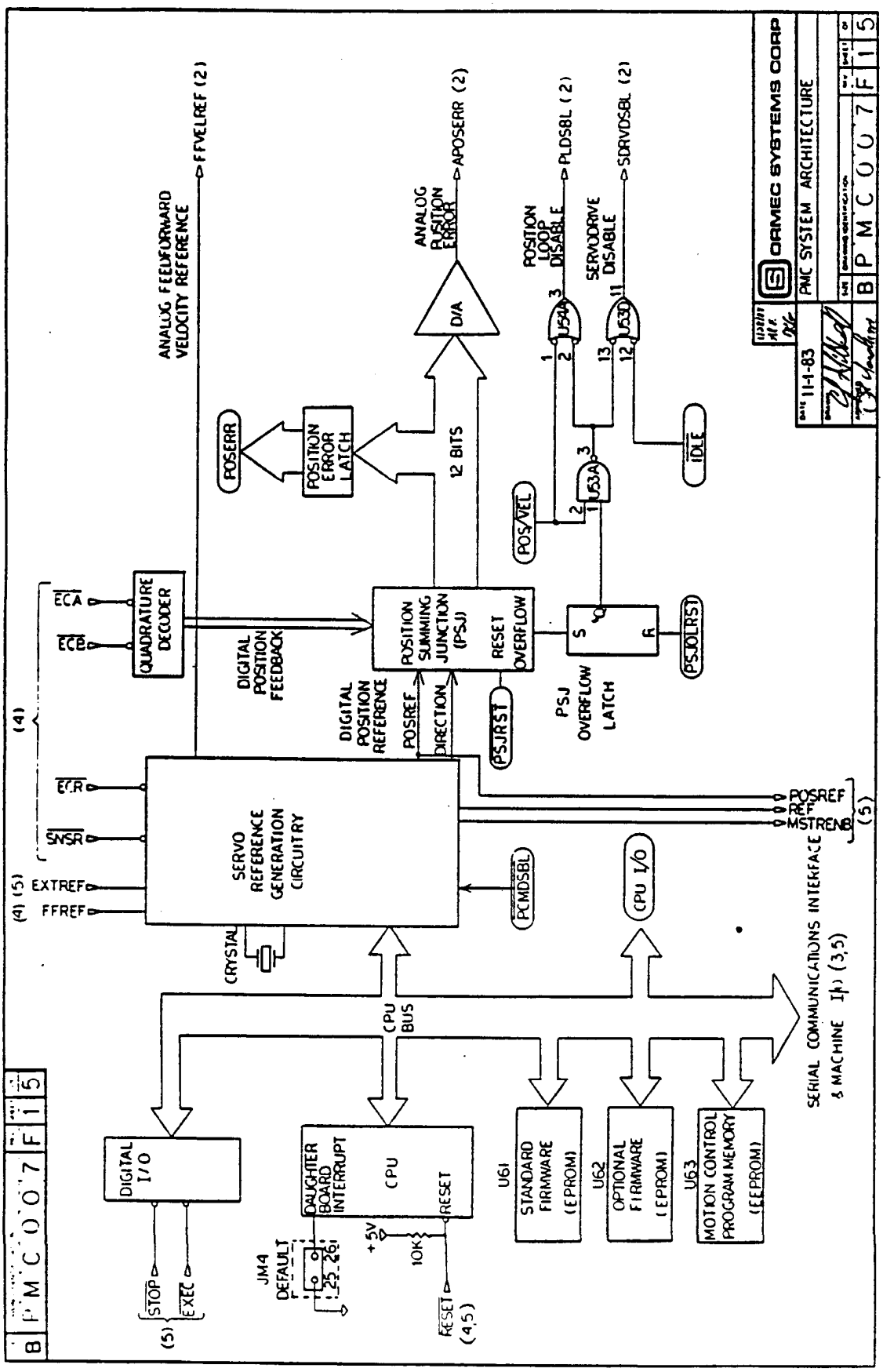
APPENDIX 8.1 TYPICAL PMC POSITIONING SYSTEM DIAGRAM



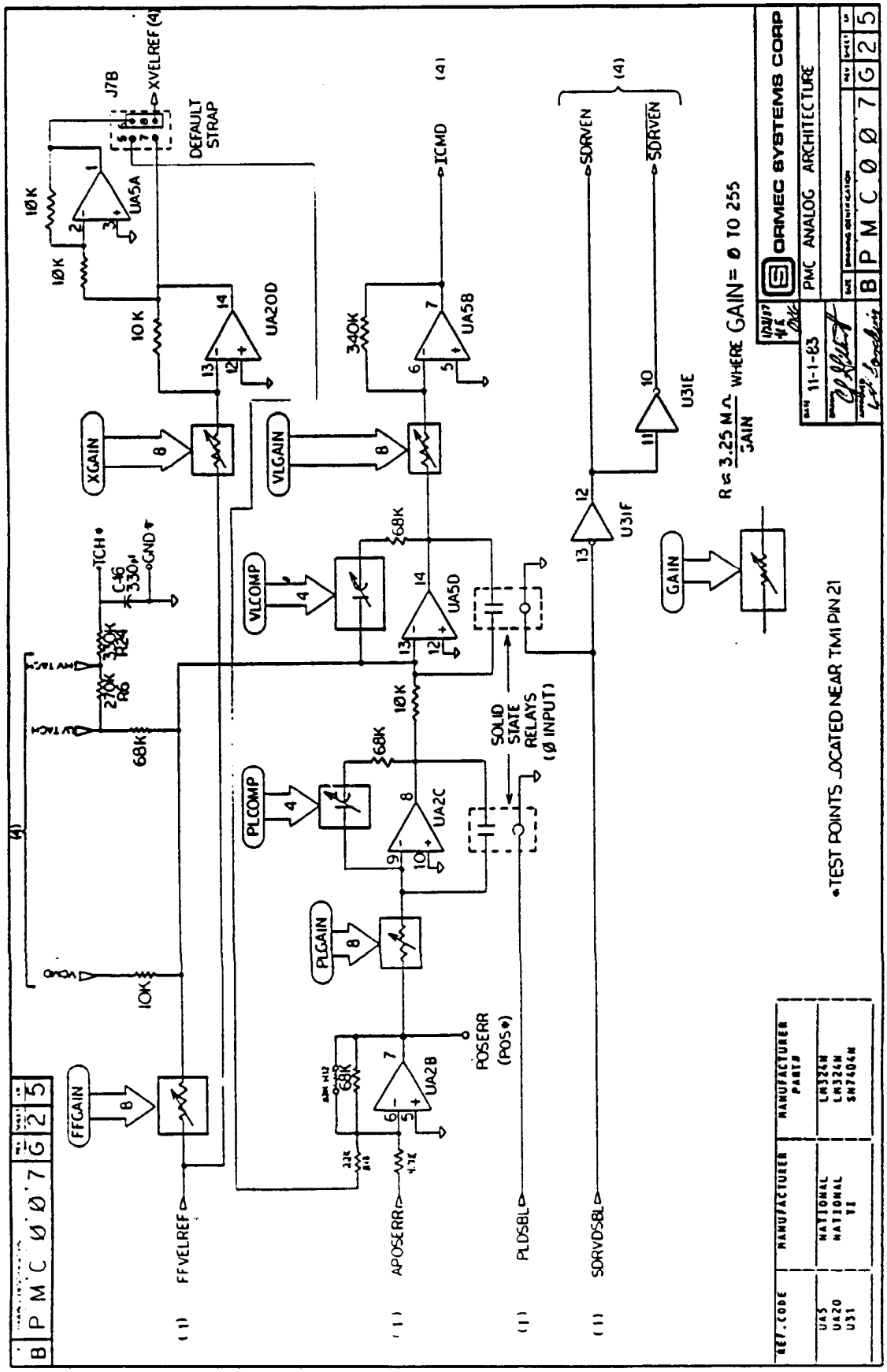
8.2 TYPICAL MOTOR LOOP INTERFACE DIAGRAM



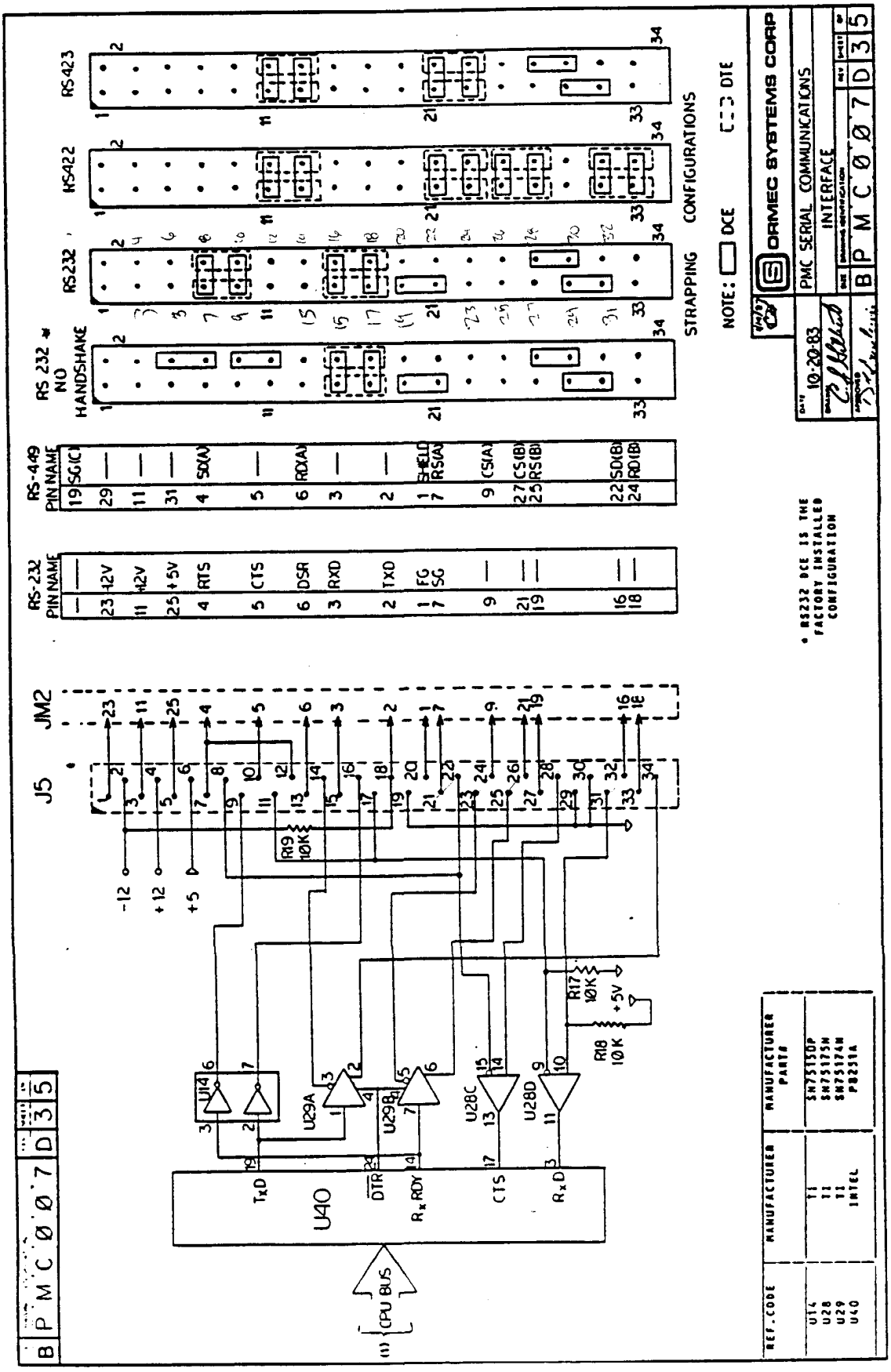
8.3 SYSTEM ARCHITECTURE DIAGRAM



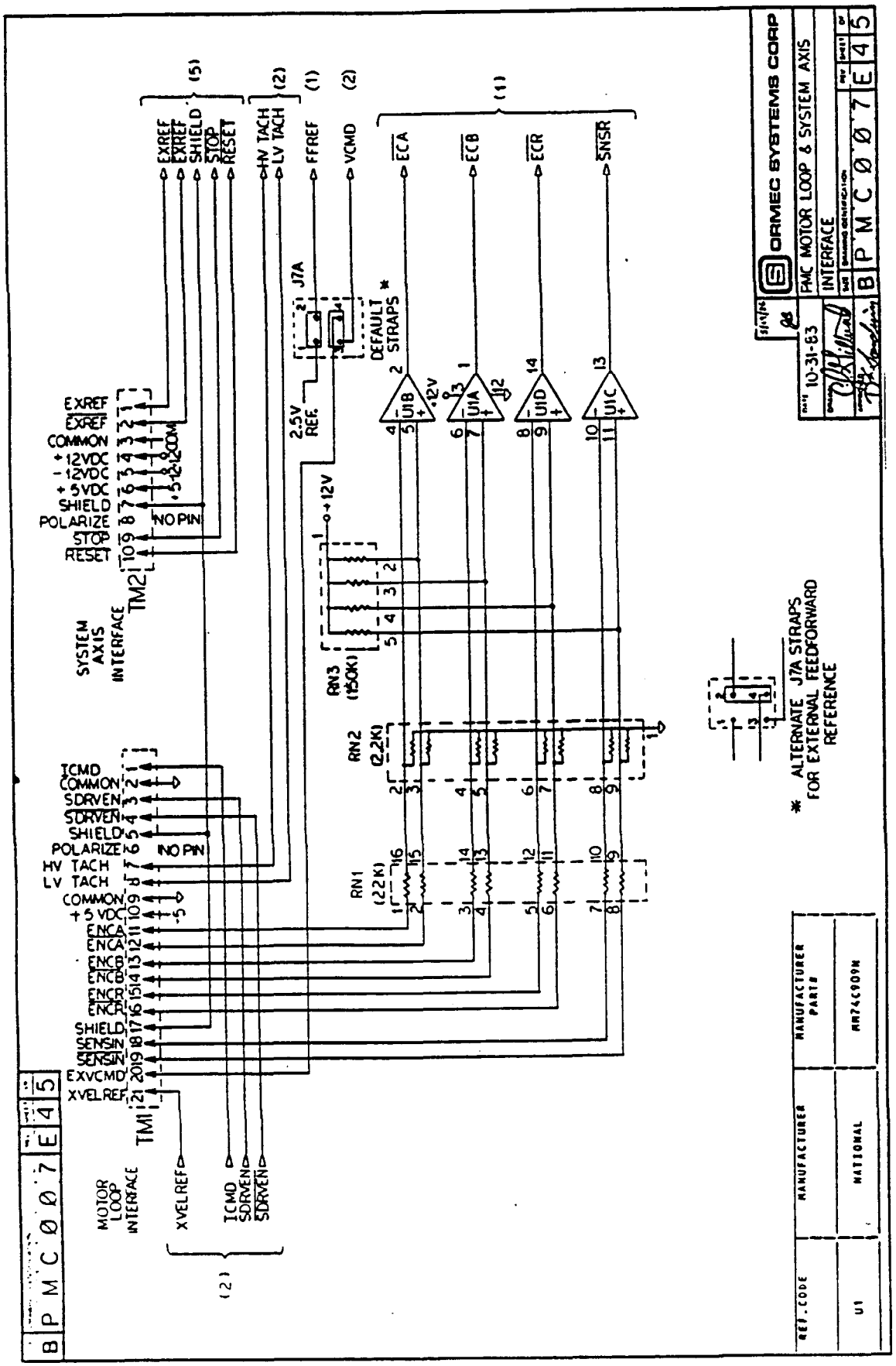
8.4 SYSTEM ANALOG ARCHITECTURE DIAGRAM



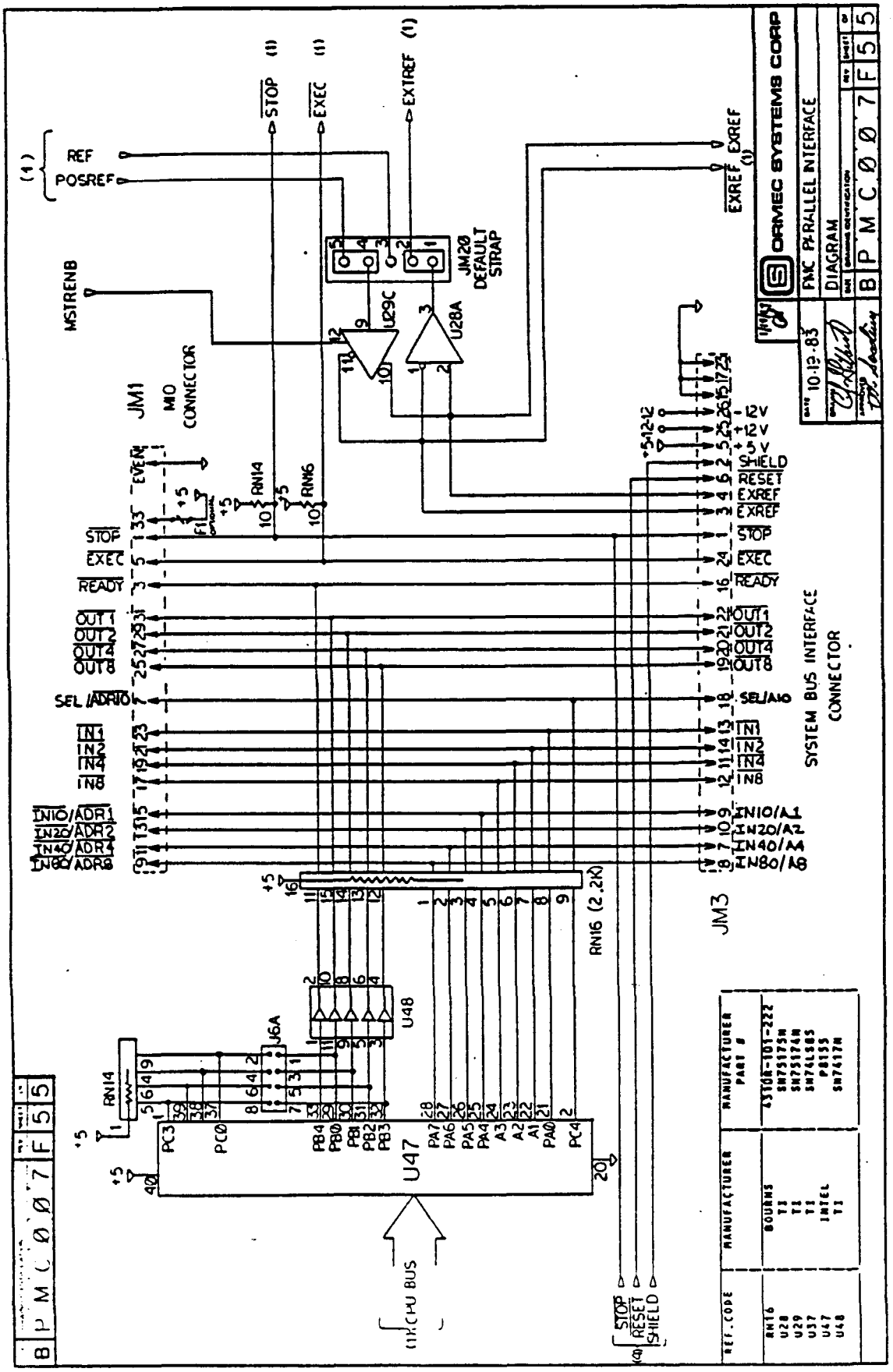
8.5 SERIAL COMMUNICATIONS INTERFACE DIAGRAM



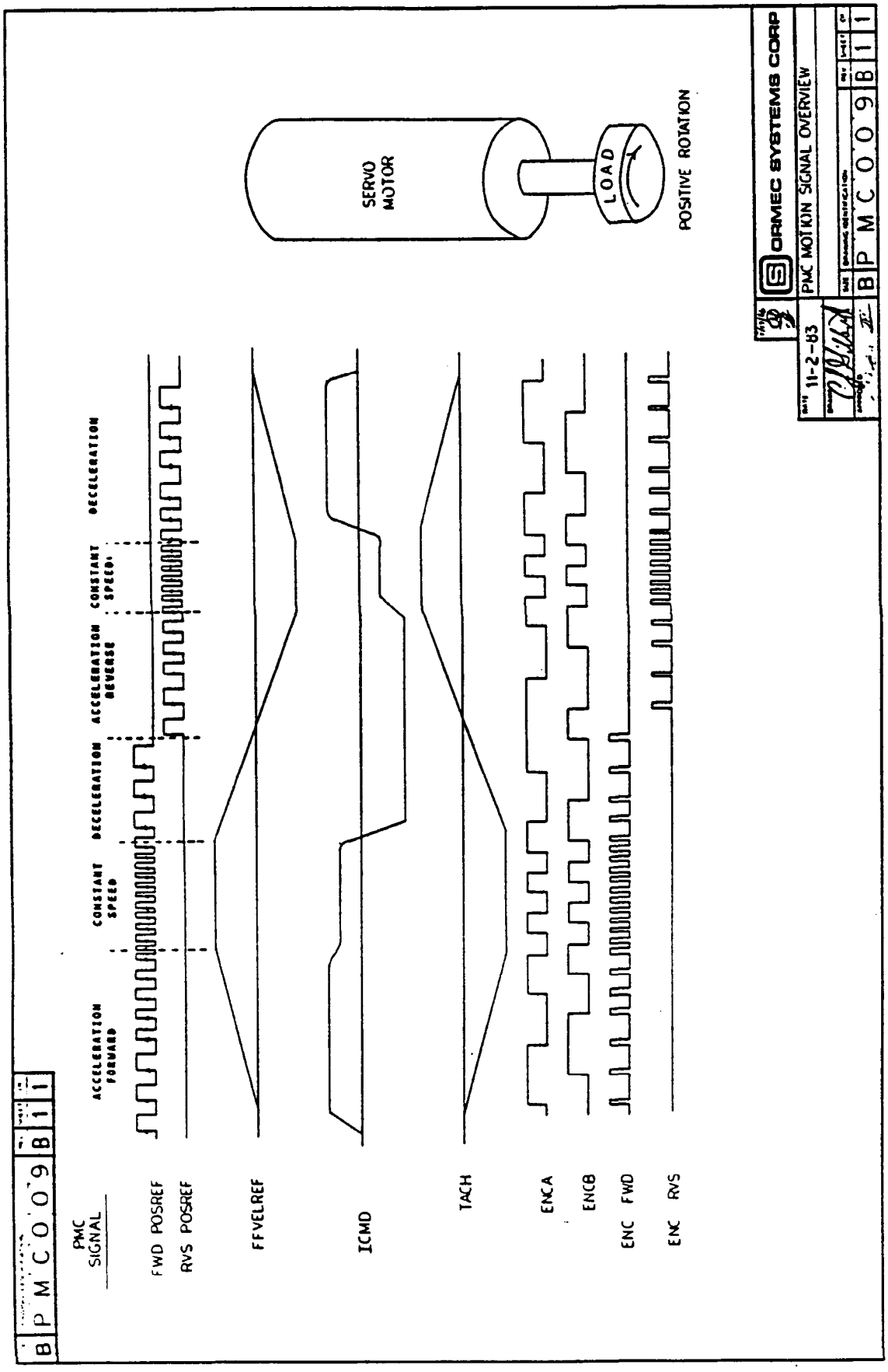
8.6 MOTOR LOOP AND SYSTEM AXIS INTERFACE DIAGRAM



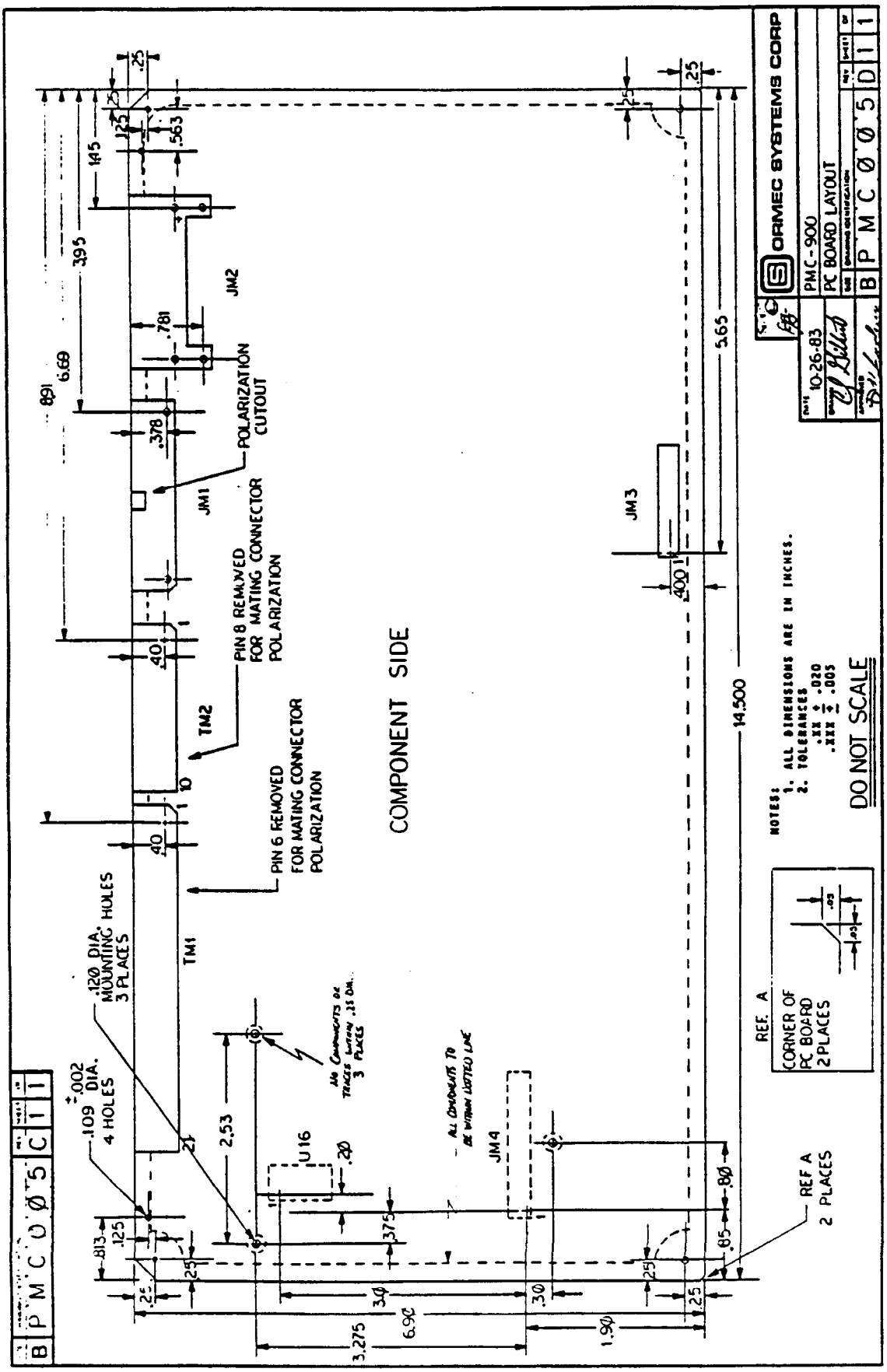
8.7 PARALLEL INTERFACE DIAGRAM



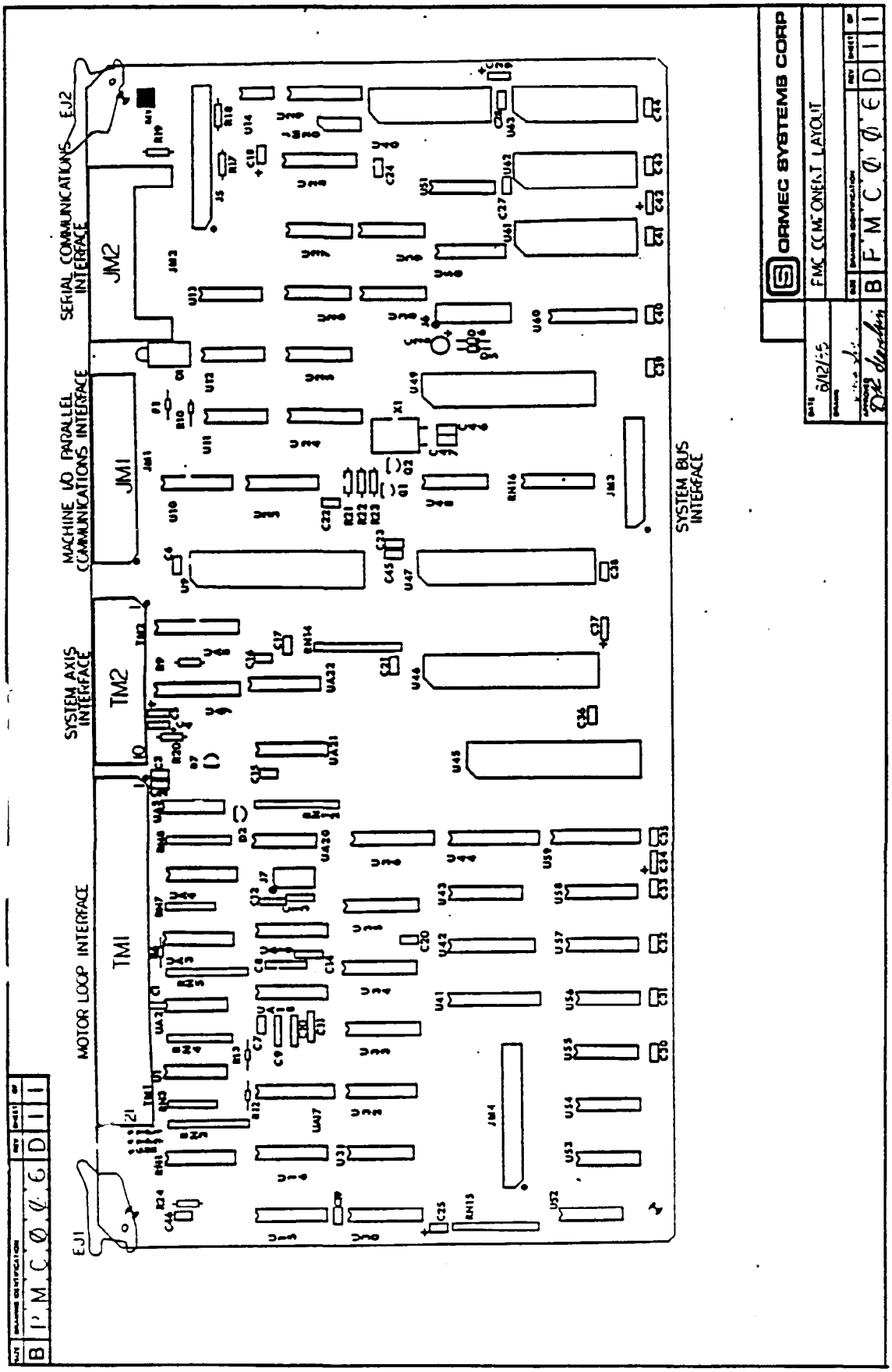
8.8 MOTION SIGNAL OVERVIEW DIAGRAM



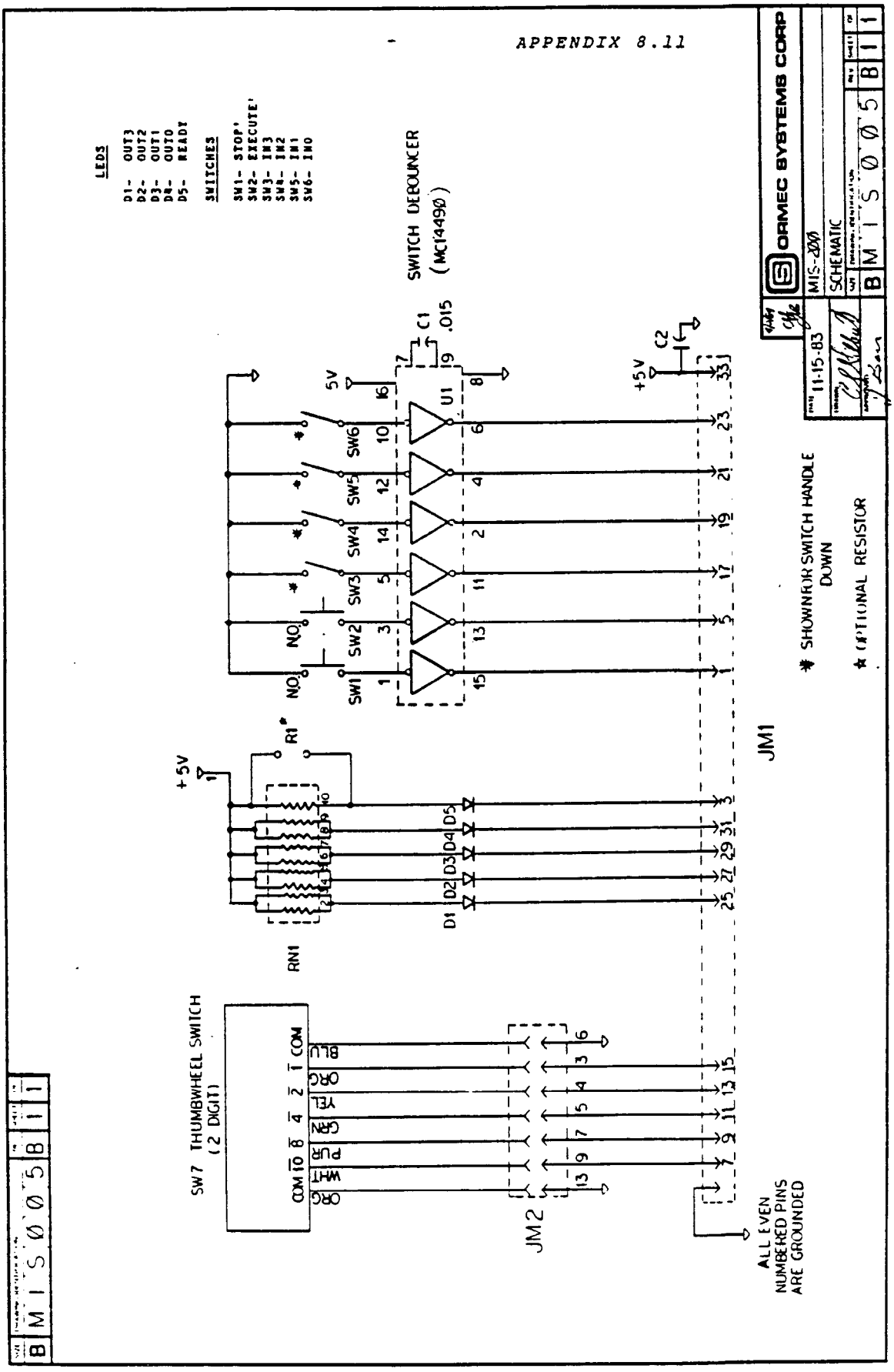
8.9 PC BOARD LAYOUT



8.10 COMPONENT LAYOUT



8.11 MIS-200 SCHEMATIC



INDEX

ORMEC
19 Linden Park • Rochester, NY 14625 • 716-385-3520

May 3, 1988

ADDITIONS & CORRECTIONS TO PMC MANUAL PMC001d

Below is a list of additions and/or corrections to the PMC manual since our last printing. We are in the process of printing a new manual to reflect the changes in MPL with Version 3.1, as compared to Version 3.0. If you have any technical questions, please call the ORMEC Service Department at (716) 385-3520.

PMC Checksum Errors

PMC Powerup Diagnostics include a "Checksum" calculation and comparison test of the contents of the non-volatile RAM. This is a method of checking the validity of the program which you entered in the Program Buffer. It examines the program currently in memory to insure that a RAM failure resulting in a changed program has not occurred.

When a Checksum failure is detected, the diagnostic LED will flash with a cycle of red for one second and green for one second, and MPL program execution will be aborted. The "MPL Prompt" will also be changed to "-}". Program operation cannot continue until the Checksum is cleared using the **NK*** command. The LED should then return to the standard yellow color. The user should then either check or download the program buffer to insure that its contents are proper. To insure that the non-volatile RAM is operating properly, powering the PMC down and back up after this update is advised. If the non-volatile RAM is used to store either an Axis ID, or a Baud Rate, then these two items should also be verified with the **SB!** or **=!** as appropriate.

The most common cause of this problem is powering down the PMC while in the process of writing an MPL program. The reason for this is that the "Checksum" is updated at the end of each Program command. If the program is changed without ending the Program command properly, then the checksum will not agree with the current contents of the Program Buffer and a false "error" will be detected the next time the PMC is

powered up. **The proper technique for terminating the Program Command is to use the escape key <Esc>, which will cause the PMC to return to the "prompt" level.**

"1x" Multiplication Circuitry

For PMC Model Nos. **PMC-903/01 and PMC-904/01** only, the quadrature decoder circuit which interprets the phase quadrature input channels to determine the load direction and distance travelled has been modified. The PMC-903/01 uses "1x" multiplication circuitry, which means that the effective resolution of a system using the PMC-903/01 is "1 times" the "linecount" of the incremental position encoder.

"2x" Multiplication Circuitry

The PMC Model Nos. **PMC-903/02 and PMC-904/02** have similar quadrature decoder circuitry but implements a "2x" multiplication scheme, so that the effective resolution of a system utilizing the PMC-903/02 is "2 times" the "linecount" of the incremental position encoder.

Special Model for High Resolution Position Feedback Devices

The PMC Model Nos. **PMC-903/03 and PMC-904/03** have a reduced position loop gain for use with high resolution position feedback devices and in selected other applications where there is too much position gain.

Upgrade of MPL-3.0 to MPL-3.1

MPL has been upgraded to version 3.1 to add some new features as well as compatibility with MPL-MATH and the extended command set of the PMC-904.

Summary of Changes in MPL-3.1

New Command to Set Deceleration

This new command allows you to set up and trigger a deceleration from top velocity. The primary use of this command is in registration systems with "blanking" capability. Suppose for example that a system is supposed to index labels rapidly and decelerate when a registration mark is detected. But it must not look for the mark until after it has reached a certain position, because there is other information on the label which is in the registration mark detector's field of view. The Until command can be used to delay until a position is reached, and the **SD** command can then be used to setup the PMC to decelerate on the sensor.

The syntax of the **SD** is:

SD <trigger> [<sync>] <cr>

Select a trigger to be used to begin the deceleration from top velocity. The trigger may be one of the following values:
D - Distance (Go and Index motions only)
E - Encoder Reference
I - Immediate trigger
O - Turn OFF trigger
S - Sensor Input

The synchronization character can be used to coordinate this command with completion of a motion, motion reaching a constant speed or reaching the end of a constant speed.

SDC <cr> Calculate the deceleration rate to be used for the next system deceleration triggered by the **SD** command. If this command is not used, the deceleration rate will be equal to the last acceleration rate. Note: when the **SDC** command is issued, the current value of the 'A' command will be used as the deceleration rate.

Examples:

SDS;<cr> Wait for top velocity, then set the deceleration trigger to be the sensor input

SDI;<cr> Wait for system to index to top velocity, then decelerate immediately

SDC Calculate the deceleration rate for the **SD** trigger.

SDD; Set up a new deceleration rate and trigger is based on the deceleration point in the current Go or Index motion.

SD? Display the deceleration trigger set by the **SD** command. This value is reset to 'O' when the system comes to rest.

SD! Same as '?'

SD% Repeatedly display the deceleration trigger.

Notes:
(1) If an index or go motion has its deceleration trigger and/or rate improperly changed by the **SD** command, the motion may not end as expected. (2) The **SD** trigger command can only be used at top velocity. (3) The **SDC** command can be issued at any time and will use the value set by the **A** command to calculate the deceleration rate.

Additions to System Status Polling

Two more parameters which can be viewed using the System Status Polling function. It is now possible to poll for the last error code (lower case l) and the current status of both general purpose machine outputs and the ready line ("o").

System status polling is provided to allow host computer access to important information during program execution. This information is acquired by sending a two character sequence to the PMC with which communications are established. These two character sequences perform the equivalent function that the indicated standard MPL command would if issued at the command level. The advantage of the system status polling commands is that the PMC does not have to be at command level for these commands to be used, i.e. system status polling can take place even while a PMC is running an MPL program.

Syntax: <attn> <sys poll>

<attn> Ctrl] (ASCII $1D_H$)

			(Equivalent MPL command)
<sys poll>	c	system inputs	(SC!)
	e	current following error	(TE!)
	g	current system position	(G!)
	h	hex communications checksum	none
	i	distance remaining in motion	(I!)
	l	**last error code**	**none**
	o	**state of outputs & ready line**	**none**
	p	last label passed	(SP!)
	r	X, Y, Z status registers	(S!)
	s	motion parameters	(SS!)
	v	current system velocity	(V!)
	x	current axis id	(=!)

Note: The <sys poll> command must be lower case.

NC Command Changed to NK

In order to enhance the non-volatile memory checksum function to work in hex communications mode, the syntax has changed from NC to NK.

The **NK<cr>** command calculates, displays and verifies checksums on the PMC's non-volatile memory. If the verify fails, the PMC generates an error flash code with its diagnostic LED, and returns an error prompt of "-}". The NK command is automatically executed at powerup.

NK?	Displays checksums stored in memory.
NK!	Calculates and displays checksums. Displays what PMC thinks checksums should be.
NK<cr>	Perform automatic non-volatile memory verify.
NK*	Recalculates and updates checksums. Errors are cleared; LED and prompt return to normal.

Note: If you have been using the **NC** command in decimal I/O mode, the command continues to work properly with Version 3.1

Normalize Position Allowed While System in Motion

The **N<position><sign>** command, which normalizes the absolute position counter in the PMC, is now permitted while the system is in motion. From the instant MPL processes the <sign> to cause N command execution, a variable time up to one millisecond may pass before the position information is actually updated.

Set Mode While System in Motion

The **SM** command, used to select the PMC's servo control mode, can now be used while the system is in motion.

Note: (1) No attempt is made to maintain proper position information when a mode change is made while the system is in motion. (2) Motion is not aborted when the mode is changed, except when switching to mode 0, which turns the servodrive off and disables the servo loops.

Syntax Change for Delay (Reference Distance) Command

The syntax of the Delay (Reference Distance) **DR** command has been changed to **DX** to allow for future compatibility with ORMEC software products. The function of the command has not changed. Below is the new syntax:

DX <distance> [<sync>] <cr>

Delay a specified number of reference clock counts before executing the next command. Note: The system must be at steady state speed to use this command.

Example:

DX5000<cr> Delay for 5000 counts of the reference clock. If in internal 192kHz mode, this command would delay 5000 counts of the 192kHz clock. If in external mode, this command would delay 5000 counts of the motion reference bus.

and } Characters No Longer Valid for MPL Programming

The pound sign "#" and right curved bracket "}" are no longer valid characters in the PMC's program buffer except during Binary Programming Mode. If you attempt to enter these characters, your terminal will sound a "beep" indicating that you attempted to enter an invalid character in the program buffer.

256 kHz Velocity Range Now Supported

MPL Version 3.1 now supports a 256 kHz velocity range by setting Bits 3 and 2 in the X Status Register to 0 and 1 respectively. This new velocity range is particularly useful for master/slave systems, where the master PMC is no longer limited to top velocity of 192 kHz. The acceleration ranges and units are identical to the 192 and 384 kHz modes. The 256 kHz velocity range has the following ranges and units:

Command	Range	Units
A	1-65,535	kHz/sec
AL	1-65,535	kHz/sec
AQ	1-65,535	kHz/sec
AS	1-65,535	kHz/sec
J	1-2,560	100 Hz
V	1-2,560	100 Hz
H	1-2,560	100 Hz

The following table states the minimum time per accel step to each of the velocity ranges (the X Register Bit Settings are also included):

Velocity Range	X Register Bits 3 - 2	Minimum time/step
48 kHz	0 0	1.3 msec
192 kHz	1 0	333 microseconds
256 kHz	0 1	500 microseconds
384 kHz	1 1	333 microseconds

Note: Switching into or out of 256 kHz range could result in loss of one encoder count of feedback.

Motion Synchronizing Characters

The effect of the ":" and ";" synchronization characters on MPL operation has been changed. The effect of the synchronizing terminators on MPL execution is as follows:

; wait for Constant Speed (including No Motion)
: wait for an Acceleration to begin
, wait for the end of commanded motion

The following table indicates whether MPL program execution will "Wait" or "Continue" as a function of the synchronization character used, and the status of motion in progress.

	No Motion	Wait for Start	Acceleration	Constant Speed	Deceleration
;	Continue	Wait	Wait	Continue	Wait
:	Wait	Wait	Continue	Wait	Continue
,	Continue	Wait	Wait	Wait	Wait

Notes:

1) "Wait for Start" refers to the time between the initiation of a motion command and the actual beginning of motion. When an externally triggered motion is commanded, this time can be significant because the actual beginning of motion is dependent on receiving the Sensor or Encoder Reference pulse (as selected by the Y-Register).

2) During a motion with "Index Extend" (Status Register Y, Bit 1) the ";" sync character formerly did not allow MPL execution to continue during the extended jog speed portion of the motion. It will allow MPL to continue now. Conversely, the ":" sync character will now wait during this period until the deceleration begins.

Delay on Commanded Motion

The DM command is now allowed during the JOG portion of an "Index Extend" motion (set by Status Register Y, Bit 1).

Additions and/or Changes to Error Codes

A7 Reserved

B1 Index distance exceeded

B4 Deceleration cannot be triggered while system is accelerating, decelerating or at rest