# PMC-960

# PROGRAMMABLE MOTION CONTROLLER

INSTALLATION AND OPERATION MANUAL
PMC960f



19 Linden Park

Rochester, NY 14625

88.02.19

## TABLE OF CONTENTS

# GENERAL DESCRIPTION

## 1.1 INTRODUCTION

The **PROGRAMMABLE MOTION CONTROLLER (PMC-960)** is a microcomputer based product which facilitates the design of high performance motion control applications. In combination with a servodrive, a servomotor, and an external resolver interface, the PMC-960 is used to create a closed loop digital position system. The resulting position control system translates easy to use commands and motion parameters into high performance motion. Acceleration, velocity and distance are specified using ORMEC's **Motion Programming Language (MPL)**, and controlled by the PMC-960 to a high degree of precision.

MPL commands may be executed directly from the **Serial Communications** Interface (SCI), or combined in non-volatile program memory (ZeroPower RAM) 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.

A PMC-960 based positioning system is designed to operate in a stand-alone mode with either a terminal or a computer communicating to it through the Serial Communications Interface (SCI). Communications to the PMC-960 are similar to communications to a serial terminal. The PMC-960 receives motion commands by processing individual bytes (characters) of information sent to it by a terminal or computer. The acceptance of each character is indicated by the hardware handshake line of the serial communications interface, and is usually less than a millisecond. When a command is completely received, the PMC-960 executes it. When execution of the command is finished, the PMC-960 sends a right brace character "}" (which is ASCII Code $7D_H$) to the host to indicate that it is "READY" for a new command.

These commands may also be programmed in the non-volatile MPL Program memory for future execution. In this case, a sequence of MPL commands are entered using the Program command. When programming is complete, the program may be executed from the Serial Communications Interface using the Branch command. In addition, up to 32 individual programs can be accessed from the Machine I/O interface. See the Machine I/O interface installation section for details.

## 1.2 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' The drawings in the APPENDIXES and some charts or tables may use the "overbar" notation.

For example: $\overline{\text{INPUT}}$

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.

# THEORY OF OPERATION

## 2.1 INTRODUCTION

The PMC operates as an intelligent slave motion control system, receiving high level commands via the Serial Communications Interface (SCI) and in turn providing position and velocity reference information to the servo system. The PMC-960 Block Diagram is described in APPENDIX 8.1 and the PMC-960 Core Design is described in APPENDIX 8.2.

## 2.2 SYSTEM ARCHITECTURE OVERVIEW

### 2.2.1 Position Feedback Transducer

The PMC-960 is configured to utilize resolver information generated off-board. This information is normally generated by a resolver to digital converter such as the Analog Devices 1S24 part. When a resolver is used, the signals needed by the PMC-960 include both digital (up to 16 bits of position, RDCBSY, etc.) and analog (TACH, InterLSB, etc.) information. This interface is fully described in a following section. The position loop is closed using the full word of position data from the R to D converter for maximum noise immunity.

As an alternative, the PMC-960 can use a digital position encoder that generates phase quadrature signals. In this case the user must also provide an analog velocity feedback signal. e.g. a d.c. tachometer

### 2.2.2 Summing and Compensation Circuitry

In order to cause motion of the digital position control system, the microprocessor generates both analog velocity and digital position reference data. The analog velocity information (velocity feedforward) is summed in the velocity summing junction by an operational amplifier as a reference (command) input to the analog velocity loop. The digital reference information is compared with the feedback information from either the R to D converter or the digital position encoder in a digital Position Summing Junction (PSJ). The contents of the digital PSJ is a count that is equivalent to the digital position error present in the system. The least significant 12 bits of this digital position error are converted to a bipolar analog voltage by the 12 bit D/A CONVERTER.

The resultant position error signal (POSERR) is processed by the position compensation amplifier. 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.

Analog velocity information, generated by either an analog tachometer or the R to D converter, is used to provide velocity feedback for maximum performance. This velocity feedback signal is summed with the feedforward velocity reference (FFVREF) 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.

### 2.2.3 Modes of Operation

There are three operating modes of the PMC-960:

1. IDLE In this mode, the PMC-960 disables the "drive enable" output (DRVENB') and disables the position and velocity loop compensation circuitry.

2. VELOCITY In this mode, the PMC-960 operates as an analog velocity control system, driving the velocity loop with an analog voltage derived by the Reference Generation Circuitry. This voltage is applied to the velocity loop through the feedforward gain adjust (FFGAIN). It may also be used externally, 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. In velocity mode, distance position data is not summed in the Position Summing Junction (PSJ).

3. POSITION In this mode, the PMC-960 operates as a digital positioning system, controlling speed and position as a phase lock position controller. Digital position reference data derived by the Reference Generation Circuitry is applied to the PSJ and an analog feedforward voltage is applied to the velocity loop.

   In this mode, the following error in the digital PSJ is monitored, and if it exceeds the software following error limit (up to 32,767 counts), the system is returned to the IDLE mode.

### 2.2.4 Operation Overview

The PMC-960 initiates a position displacement by changing the digital position reference data applied to the PSJ at a rate proportional to the desired velocity. Simultaneously, the PMC-960 applies an analog velocity feedforward voltage (FFVELREF) to the velocity summing junction to minimize the error required in the Position Summing Junction.

The compensated current command signal (ICMD) is applied to the servodrive and causes the servomotor to move the load the designated distance with the desired velocity and acceleration.

The polarities of the forward/reverse command data and the feedback data are such that the contents of the digital error summer are reduced as the position feedback data is received (negative feedback). If the input digital position data continues at a constant rate long enough for the transient to decay, the digital feedback data will change at the same rate as the command data.

When the load moves a distance equivalent to the commanded change in position, the value in the position summing junction will go to zero, causing the system to stop. A system using a 12 bit resolver converter will move 1/4,096 of a revolution, or one resolver distance unit, for each unit of command data.

Since the position loop is active, a holding force related to the position loop gain will be present as required to hold the commanded output position.

## 2.3 MOTION PROGRAMMING LANGUAGE ARCHITECTURE

The PMC-960'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 intuitive commands and yet is concise enough for host computer operation.

Like a programmable calculator, or a computer running interactive BASIC, commands may be executed directly in the interactive mode or combined in a "program" or "routine" for future automatic execution. These routines are stored or edited in the "program buffer" using the Program command.

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

In addition, the PMC-960 is architected to be able to service the real time requirements of commanding a motion while simultaneously running this interpretive language. This feature allows the PMC-960 to perform necessary calculations to set up future motions while simultaneously performing positioning tasks, enabling the user to talk to the PMC-960 while it is controlling motion.

An overview of the PMC-960's Motion Programming Language is found in the following table:

| MOTION PROGRAMMING LANGUAGE (MPL) | | |
|---|---|---|
| Function | Description | Commands |
| MOTION PARAMETER COMMANDS | Defining Motion Parameters | Velocity<br>Acceleration<br>Index<br>Home |
| MOTION ACTION COMMANDS | Creating Motion | Go<br>Index<br>Home<br>Jog<br>Cam<br>contouR |
| SYNCHRONIZATION/ INTERFACE COMMANDS | Synchronizing Motion<br><br>Manipulating Machine Outputs | Dwell<br>Until<br>, & ; Synch<br>Output |
| PROGRAM BUFFER COMMANDS | Entering or Editing A Program | Program<br>@ Labeling<br>Motion Routine |
| PROGRAM CONTROL COMMANDS | Utilizing Subroutines & Creating Complex Motion Control Applications | Branching<br>Looping<br>Function Call<br>Exit |
| SYSTEM SETUP COMMANDS | Selecting System Options | Normalize<br>System<br>Table |

# SPECIFICATIONS

## 3.1 GENERAL SPECIFICATIONS

CPU

| | |
|---|---|
| type | 80C85A |
| speed | 3.072 MHz |

Motion Control Program Storage

| | |
|---|---|
| Mostek ZeroPower RAM | 2k Bytes |

Position loop

| | |
|---|---|
| max following error | 32,767 counts |
| max leading error | 2,048 counts |
| max positioning speed | 384.0 K counts/sec |

## 3.2 MECHANICAL AND ENVIRONMENTAL SPECIFICATIONS

Mechanical and Environmental

| | |
|---|---|
| Max dimensions | 11.5"x 8.5" x 0.8" |
| Max weight | one pound |
| | |
| Temperature ranges: | |
| Operating | 0 to +50 degrees C |
| Storage | -25 to +125 degrees C |
| Relative humidity | (w/o condensation) 0 to 90% |

## 3.3 ELECTRICAL SPECIFICATIONS

### 3.3.1 Power Supplies

| | |
|---|---|
| +5 VDC | 1.2A max |
| ±12 VDC | 0.15A max |

### 3.3.2 Digital Inputs

| | | |
|---|---|---|
| Logical 0 | Vin < 0.8 VDC | Iin < -1 mA |
| Logical 1 | Vin > 2.8 VDC | Iin < 400 uA |

| | Minimum Acceptance Time | Max frequency |
|---|---|---|
| ESTOP' | 4.5 msec | |
| BREAK' | 4.5 msec | |
| FAULT' | 4.5 msec | |
| EXTREF/EXTREF' | 1.3 us | 192 kHz |
| CHA/CHA' | 8 usec for 192 kHz range | |
| CHB/CHB' | 8 usec for 192 kHz range | |
| SENSOR/SENSOR' | 8 usec for 192 kHz range | |

### 3.3.3 Digital Outputs

| | |
|---|---|
| I.C. type | SN7417N TTL buffer |
| max sink current | -16 mA |
| max source current | 400 uA |

### 3.3.4 Analog Inputs

| | |
|---|---|
| TACH | Impedance > 680K ohm<br>Max voltage = ±10 VDC (unless R12 installed) |
| ALOCK | Impedance > 1M ohm<br>Max voltage = ±10 VDC |

### 3.3.5 Analog Outputs

| | |
|---|---|
| I.C. type | LM324N or LF412 Op Amp |
| Range | ±10 VDC |
| Max current | 5 mA |
| Minimum Load | 2K ohms |

### 3.3.6 Serial Inputs/Outputs

| | |
|---|---|
| Standards | EIA RS-232 and EIA RS-422 |
| Protocol | 8 data bits, 1 stop bit, no parity |
| RS-422 Driver | SN75174 |
| RS-232 Driver | SN75150 |

# INSTALLATION

## 4.1 HARDWARE INTERFACE SPECIFICATIONS

Many of the interface signals for the PMC-960 are "TTL Compatible". These I/O points may be implemented in hardware with LSTTL, TTL, NMOS, or Opto-Isolator components. The specifications for TTL Compatible inputs and outputs are:

| | | | |
|---|---|---|---|
| logical 0 | Vin < 0.8 VDC<br>Iin < -1 mA | logical 1 | Vin > 2.8 VDC<br>Iin < 400 uA |

There are 5 signal connectors and a power connector on the PMC-960 board. The following describes each connector and all of the signals associated with it.

## 4.2 SERIAL COMMUNICATIONS INTERFACE (JM6)

This connector (25 pin male D-sub) is used as a communication path between a terminal or computer and the PMC-960. It also provides low current power source.

### 4.2.1 Connections on JM6 to Provide Power

| Pin# | Signal Name | Description |
|---|---|---|
| 1 | DGND | Digital ground |
| 7 | DGND | Digital ground |
| 11 | +12 VDC | +12 VDC power supply |
| 13 | SHIELD | EMI isolated shield ground |
| 23 | -12VDC | -12 VDC power supply |
| 25 | +5 VDC | +5 VDC power supply |

### 4.2.2 Connections on JM6 to Implement RS-232 Communications

| Pin# | Signal Name | Description |
|---|---|---|
| 1 | DGND | Digital ground |
| 2 | TXD | Transmit data from host |
| 3 | RXD | Receive data from PMC |
| 4 | RTS | Request to send from host |
| 5 | CTS | Clear to send from PMC |
| 7 | DGND | Digital ground |

### 4.2.3 Connections on JM6 to Implement RS-422 Communications

| Pin# | Signal Name | Description |
|---|---|---|
| 9 | RDB | Receive data from PMC |
| 21 | RDA | Receive data from PMC (complement of RDB) |
| 14 | SDB | Send data from host |
| 2 | SDA | Send data from host (complement of SDB) |
| 18 | CSB | Clear to send from PMC |
| 19 | CSA | Clear to send from PMC (complement of CSB) |
| 17 | RSB | Request to send from host |
| 4 | RSA | Request to send from host (complement of RSB) |

## 4.3 SYSTEM BUS INTERFACE (JM1)

This connector (12 pin .1" Molex Block) provides for the bussing of the Motion Reference Bus as well as bringing some test points to a convenient spot on the board edge. The mating connector is a Molex P/N 22-01-2127 or equivalent.

| Pin# | Signal Name | Description |
|---|---|---|
| 1 | ALOCKTP | ALOCKTP (Analog Lock Test Point) is a test point for gaining easy access to the output of the analog lock gain stage. |
| 2 | FFCMDTP | FFCMDTP (Feedforward Command Test Point) is a test point for gaining easy access to the output of the feedforward gain stage. |
| 3 | ICMDTP | ICMDTP (Current Command Test Point) is a test point for gaining easy access to ICMD. |
| 4 | AGND | Analog ground |
| 5 | EXVREF | EXVREF (External Velocity Reference) is a analog output which is proportional to the commanded velocity of the PMC. The gain of this signal is independently software selectable. |
| 6 | XFFREF | XFFREF (External Feedforward Reference) is an analog input which is required to be proportional to the external reference frequency of the motion reference bus. This becomes the reference voltage for the feedforward circuitry. |
| 7 | ESTOP' | Digital input that stops the current motion and breaks the current motion control program. Upon detecting the ESTOP' input an emergency stop will occur. When that deceleration is complete the PMC will go into mode 0, |

| | | |
|---|---|---|
| | | disabling the motor. If there is no motion when the ESTOP' input is asserted then the PMC will immediately go into mode 0.<br><br>When the FAULT' or ESTOP' inputs are asserted an emergency stop is begun. Any attempt to command another motion before that stop is complete will result in a #B2 error.<br><br>Minimum acceptance time: 4.5 msec<br>**Note that this input is also present on connector JM3.** |
| 8 | DGND | Digital ground |
| 9 | EXTREF | EXTREF is an RS-422 driver/receiver tri-statable digital output/input for use as an external position reference for distance-based motion commands. |
| 10 | EXTREF' | EXTREF' is the logical complement of EXTREF |
| 11 | SHIELD | EMI isolated shield ground |
| 12 | AGND | Analog ground |

## 4.4 SERVODRIVE INTERFACE JM2

This connector (34 pin male header) will interface a Brushless DC Servodrive to the PMC-960. There are two main functions of this cable: returning to the PMC-960 feedback information on speed and position of the motor and providing controls to the drive based on the system operation. Digital and analog grounds are interspersed around sensitive signals such as the low-level analog and lower 2 bits of POS to provide a degree of shielding from EMI. Mating connector is a Berg P/N 66900-234 or equivalent.

| Pin# | Signal Name | Description |
|---|---|---|
| 1 | ICMD | ICMD (Current Command) is the analog velocity error output of the velocity loop which is connected to the input of the servodrive (analog output). |
| 2 | AGND | Analog ground |
| 3 | TACH | This analog input is for connecting the DC tachometer when the maximum anticipated tachometer voltage is between 0 and 10 VDC. This signal should be negative for "forward" motion of the servomotor. If a higher voltage range is needed, location R12 of the PMC-960, may be populated with a resistor. |
| 4 | AGND | Analog ground |
| 5 | ALOCK | ALOCK (Analog Lock) 10 volt peak analog input which provides a zero-crossing signal representing the motion of the motor within one count. |
| 6 | AGND | Analog ground |

| | | |
|---|---|---|
| 7 | DRVENB' | DRVENB' (Servodrive Enable) is an open collector digital output signal used to control a solid state relay which enables power to the servodrive and/or a loop contactor. The equipment connected must provide a pull-up resistor on this signal. |
| 8 | DGND | Digital ground |
| 9 | DRVRDY' | DRVRDY' (Servodrive Ready) is a digital input used to indicate when the servodrive is powered-on and ready. |
| 10 | DGND | Digital ground |
| 11 | BUSY | BUSY is a digital input which is high when the resolver position inputs are changing. |
| 12 | DGND | Digital ground |
| 13 | N/C | Reserved |
| 14 | DGND | Digital ground |
| 15 | POS00 | A digital input which is the least significant bit of the resolver position |
| 16 | DGND | Digital ground |
| 17 | POS01 | A digital input which represents one bit of the resolver position |
| 18 | DGND | Digital ground |
| 19 | POS02 | One input bit of resolver position |
| 20 | POS03 | One input bit of resolver position |
| 21 | POS04 | One input bit of resolver position |
| 22 | POS05 | One input bit of resolver position |
| 23 | POS06 | One input bit of resolver position |
| 24 | POS07 | One input bit of resolver position |
| 25 | POS08 | One input bit of resolver position |
| 26 | POS09 | One input bit of resolver position |
| 27 | POS10 or CHA' | One input bit of resolver position. This input is also used as CHA'--Encoder Channel "A" (one of two quadrature square wave signals and the logical complement of CHA) 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. |

| | | |
|---|---|---|
| 28 | POS11 or CHA | One input bit of resolver position. This input is also used as CHA--Encoder Channel "A" (one of two quadrature square wave signals) when the system is configured to utilize encoder feedback. It is a digital input signal with a "low" level of 0 VDC and and a "high" level of between +5 VDC and +12 VDC. |
| 29 | POS12 or CHB' | One input bit of resolver position. This input is also used as CHB'--Encoder Channel "B" (one of two quadrature square wave signals and the logical complement of CHB) 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. |
| 30 | POS13 or CHB | One input bit of resolver position. This input is also used as CHB--Encoder Channel "B" (one of two quadrature square wave signals) when the system is configured to utilize encoder feedback. It is a digital input signal with a "low" level of 0 VDC and and a "high" level of between +5 VDC and +12 VDC. |
| 31 | POS14 or CHZ' | One input bit of resolver position. This input is also used as CHZ' (Encoder Channel "Z" when the system is configured to utilize encoder feedback and is the logical complement of CHZ') 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. |
| 32 | POS15 or CHZ | A digital input which is the most significant bit of the resolver position. This input is also used as CHZ--Encoder Channel "Z" when the system is configured to utilize encoder feedback. CHZ is 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 +12 VDC. |
| 33 | +5VDC | +5 VDC power supply |
| 34 | SHIELD | EMI isolated shield ground |

## 4.5 MACHINE I/O INTERFACE (JM3 & JM4)

This connector (2-50 pin headers--polarize) is divided into the following logical groups:

Controller Interface -- This interface is usually tied to a computer or Programmable Controller which controls multiple units in a production line. This interface is used to synchronize the execution of the machine profile with other equipment.

Operator Panel -- This interface contains signals that could be used on an operator panel for a machine. These signals are read by the microprocessor and require a minimum of 4.5 msec assertion time. The ESTOP switch can be wired up to break the control current and open a loop contactor which switches the drive power. In addition the PMC-960 can also be wired up to sense when the ESTOP' signal has been asserted at which time it would do an emergency deceleration (defined max accel rate) and disable of the drive.

Machine I/O -- This interface has all of the signals that are associated with the physical machine (i.e. limit switches, spindle, sensors, actuators, etc.) that affect or are controlled by the profile program. Most signals will first be processed by an I/O board. The General Purpose I/O signals may be muxed for the sake of minimizing interconnection between the PMC-960 and the I/O board. All of the digital signals on JM3 & JM4 are Opto-22 compatible. The one analog signal (AOUT) can be made to act as an Opto-22 compatible output if 0 VDC and +5 VDC are used as the output levels.

The mating connector for JM3 is Amp P/N 1-499506-2 and the mating connector for JM4 is Amp P/N 1-499506-2.

### 4.5.1 Pin Assignment for JM3 Connector

| Pin# | Signal Name | In/Out | Description |
|---|---|---|---|
| 1 | ESTOP' | in | Asserting this input will stop the system motion and cause the PMC-960 to return to the ready state. i.e. abort any other activity taking place and return to the interactive command level Upon detecting the ESTOP' input an emergency stop will occur. When that deceleration is complete the PMC will go into mode 0, disabling the motor. If there is no motion when the ESTOP' input is asserted then the PMC will immediately go into mode 0.<br><br>When the FAULT' or ESTOP' inputs are asserted an emergency stop is begun. Any attempt to command another motion before that stop is complete will result in a #B2 error.<br><br>Digital input, 4.5 milliseconds minimum **Note that this input is also present on connector JM3.** |
| 3 | EXEC' | in | A high to low transition of this digital input signal will start execution of the pre-programmed routine selected by the **TLV** command. |
| 5 | FAULT' | in | Asserting this input will stop the system motion and cause the PMC-960 to return to the ready state. i.e. abort any other activity taking place and return to the interactive command level<br><br>When the FAULT' or ESTOP' inputs are asserted an emergency stop is begun. Any attempt to command another motion before that stop is complete will result in a #B2 error.<br><br>Digital input, 4.5 milliseconds minimum |

| | | | |
|---|---|---|---|
| 7 | ADDR4' | in | These digital input signals (ADDR4'-ADDR0') allow the user to specify a five bit motion routine address for hardware access of up to 32 user programmable motion control routines.<br><br>To run one of the specified routines, place the appropriate signals on these address lines and when the PMC-960 is "ready", as indicated by assertion of the SREADY' output, assert the START' input. |
| 9 | ADDR3' | in | Digital input signal for specifying motion routine address |
| 11 | ADDR2' | in | Digital input signal for specifying motion routine address |
| 13 | ADDR1' | in | Digital input signal for specifying motion routine address |
| 15 | ADDR0' | in | Digital input signal for specifying motion routine address |
| 17 | START' | in | Asserting this input starts execution of the MPL program at the specified motion routine address (ADDR4'-ADDR0'). |
| 19 | BREAK' | in | Asserting this input terminates execution of any MPL program in progress and causes the PMC-960 to return to the SREADY' state |
| 21 | SREADY' | out | The SREADY' output is asserted whenever the PMC-960 is ready for a command |
| 23 | SFAULT' | out | The SFAULT' output is asserted whenever an error occurs and is turned off when the next command is initiated. |
| 25 | OUT03' | out | The four general purpose digital outputs on this connector (OUT03'-OUT00') and 12 outputs on JM4 (OUT15'-OUT4') may be changed using the **OB** command. |
| 27 | OUT02' | out | General purpose output |
| 29 | OUT01' | out | General purpose output |
| 31 | OUT00' | out | General purpose output |
| 33 | IN07' | in | The eight general purpose inputs on this connector (IN07'-IN00') and the eight inputs on JM4 (IN15'-IN08') may be read using the **SI** command which reports the status of the general purpose inputs. These general purpose inputs may also be used by the conditional Branch, Exit, Function and Until commands. |

| | | | |
|---|---|---|---|
| | | | General purpose inputs IN07' & IN06' are used for + and - limits if they are enabled by setting the select limit input bit utilizing the **THL** command |
| 35 | IN06' | in | General purpose input |
| 37 | IN05' | in | General purpose input |
| 39 | IN04' | in | General purpose input |
| 41 | IN03' | in | General purpose input |
| 43 | IN02' | in | General purpose input |
| 45 | IN01' | in | General purpose input |
| 47 | IN00' | in | General purpose input |
| 49 | +5V | | +5 VDC low current supply for Opto-22 rack use |
| EVEN | DGND | | Digital ground |

### 4.5.2 Pin Assignment for JM4 Connector

| Pin# | Signal Name | In/Out | Description |
|---|---|---|---|
| 1 | AGND | --- | Analog ground |
| 3 | AOUT | out | General purpose 12 bit analog output settable by executing an **OA** command. |
| 5 | SENSOR' | in | SENSOR' is the logical complement of SENSOR, 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. |
| 7 | SENSOR | in | SENSOR (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). |
| 9 | OUT15' | out | The twelve general purpose digital outputs on this connector (OUT15'-OUT04') may be set by the **OB** command. |
| 11 | OUT14' | out | General purpose output |
| 13 | OUT13' | out | General purpose output |
| 15 | OUT12' | out | General purpose output |
| 17 | OUT11' | out | General purpose output |

| | | | |
|---|---|---|---|
| 19 | OUT10' | out | General purpose output |
| 21 | OUT09' | out | General purpose output |
| 23 | OUT08' | out | General purpose output |
| 25 | OUT07' | out | General purpose output |
| 27 | OUT06' | out | General purpose output |
| 29 | OUT05' | out | General purpose output |
| 31 | OUT04' | out | General purpose output |
| 33 | IN15' | in | The eight general purpose inputs on this connector (IN15'-IN08') may be read using the **SI** command which reports the status of the general purpose inputs. These general purpose inputs may also be used by the conditional Branch, Exit, Function and Until commands |
| 35 | IN14' | in | General purpose input |
| 37 | IN13' | in | General purpose input |
| 39 | IN12' | in | General purpose input |
| 41 | IN11' | in | General purpose input |
| 43 | IN10' | in | General purpose input |
| 45 | IN09' | in | General purpose input |
| 47 | IN08' | in | General purpose input |
| 49 | +5V | --- | +5 VDC low current supply for Opto-22 rack |
| EVEN | DGND | --- | Digital ground |

## 4.6 SYSTEM POWER INTERFACE (JM5)

This connector (.156 Molex) provides the PMC-960 with its power. The PMC-960 itself uses three regulated voltages (+5/+12/-12). The mating connector is Molex P/N 09-50-3081 or equivalent.

**Pin Assignment for JM5 Connector**

| Pin# | Signal Name | Description |
|---|---|---|
| 1 | +12 VDC | +12 VDC power supply |
| 2 | AGND | Power supply analog ground |
| 3 | -12 VDC | -12 VDC power supply |

| | | |
|---|---|---|
| 4 | POLARIZE | This pin is removed from JM5 to allow polarization of the matching connector. |
| 5 | +5 VDC | +5 VDC power supply |
| 6 | +5 VDC | +5 VDC power supply |
| 7 | SHIELD | EMI isolated shield ground |
| 8 | DGND | Power supply digital ground |
| 9 | DGND | Power supply digital ground |

## 4.7 CONFIGURATION AREAS

### 4.7.1 Analog Configuration Jumpers (J7)

The default configuration for the analog section provides an internal reference voltage to the feedforward circuitry. In addition, it provides a bipolar velocity reference signal on JM1, Pin 5. The default jumpers are placed on J7, Pins 1-3 and Pins 2-4. See Appendix 9.6 for detailed schematic.

To use the PMC-960 in the slave mode with feedforward enabled, the "master" PMC needs to provide a unipolar velocity reference which can be implemented by changing the jumpers on J7 from Pins 2-4 to Pins 4-6. The "slave" PMC should get its feedforward reference from JM1, Pin 6 and the jumpers must be changed from J7, Pins 1-3 to J7, Pins 3-5. It is also necessary to hook the master PMC's JM1, Pin 5 to the slave PMC's JM1, Pin 6.

### 4.7.2 Tach Input Scaling Resistor (R12)

By default, the PMC-960 uses a tachometer input with a maximum range of $\pm$10 VDC. By providing a resistor at location R12, the user may scale the tach input such that: (R12/(R12+55K))*(Tach Range) < $\pm$10 VDC. Example: If you want to use a tachometer that has a full-scale range of 35 volts, then an acceptable value for R12 is 22K ohms.

## 4.8 TUNING THE PMC

Once installation is complete, and serial communications is established, you are now ready to "tune" the servo loops of your PMC. The recommended strategy for tuning the PMC is to perform the following adjustments in the order listed.

| | |
|---|---|
| VELOCITY LOOP: | Adjust the Velocity Loop Gain (TGAIN & 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) |
| | Adjust the Analog Lock Gain (ALGAIN) |

### 4.8.1 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 scaled (using the TGAIN parameter) such that the output of UA5A is 10 volts for maximum speed.

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, reverse the velocity feedback to remedy the 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 (FFGAIN) with a **TGF100<cr>** command so that the PMC can provide an analog velocity loop test command 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.

For example, if your position feedback transducer has a resolution of 4,096 counts per rev, commanding it at one rev per second will take a velocity parameter of 4.1 kHz. To accelerate to this rate in five msec, the acceleration parameter should be set for 800 Hz/msec. Performing an index of 4,096 counts will then cause a one revolution motion, which has 5 msec acceleration time and takes a total of approximately one second. This test signal should be quite appropriate for tuning most systems.

However, 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 TGF100 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.

Tune the system by repetitively executing the test motion while raising the velocity loop gain. Observe the tachometer signal and, when it overshoots or the system exhibits resonance by oscillating or "buzzing". Note that the PMC can report its gain and compensation parameters during the tuning process, using the command TG?.

The velocity loop "rise time" (time to go from 10% to 90% of full value) for most servo systems should be between 3 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.

### 4.8.2 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 possibly under 5 msec), VLCOMP should be adjusted. VLCOMP has a range of 0 to 255, and the velocity loop break frequency (in Hz) is equal to VLCOMP/20. The object is to raise the compensator break frequency (and therefore the value for VLCOMP) to the maximum possible without adding objectionable overshoot to the tachometer response. This is done by using the TGB command from the terminal, and "Jogging" or "Indexing" the system to determine the effect.

### 4.8.3 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.** Reverse the position feedback to remedy this situation.

Put the PMC in Position Mode by typing **SM2<cr>** and turn off the velocity loop feedforward gain by typing **TGF0<cr>**.

With your oscilloscope still attached to the tachometer signal, watch the tach signal and cause test commands to occur, increasing the position loop gain using the TGP command. When the tach test signal overshoots, determine the setting of PLGAIN that was achieved by typing **TG?**. The PMC will respond with the values for the gains and the compensators.

Once PLGAIN is adjusted to achieve a reasonably fast rise time (under 30 msec, and probably under 15 msec), FFGAIN can then be adjusted.

### 4.8.4 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 then raise the FFGAIN using the TGF command. 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 feedforward gain is properly adjusted.

To determine the setting of FFGAIN that was achieved, type **TG?**. The PMC will respond with the values for the gains and the compensators.

### 4.8.5 Adjusting the Position Loop Integral + Proportional Compensator

PLCOMP has a range of 0 to 255, and the position loop break frequency (in Hz) is equal to VLCOMP/67. The object is to raise the compensator break frequency (and therefore the value for PLCOMP) to the maximum possible without adding objectionable overshoot to the tachometer response. With your oscilloscope still attached to the tachometer test signal, adjust the compensator value using the TGE command 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 unnecessary.

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 or drive fault.

### 4.8.6 Adjusting the Analog Lock Gain

If position system dither on the least significant bit of the position feedback is a problem, and you are using a resolver with an inter-bit analog output, the ALGAIN adjustment can be used. Dither can be observed by using the SF% command. To eliminate the dither, progressively raise the analog lock gain using the TGL command until the dither stops. Should you continue to raise the gain, and the setting gets too high, the dither will resume.

# OPERATION

## 5.1 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.

| Command | Name | Description/Function |
|---|---|---|
| @ | (label) | 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 | CAM | Perform distance based velocity changes. |
| D | DWELL | Delay a specified time interval (in msec) 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". |
| 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 resolver zero reference or machine sensor. |
| I | INDEX | Move a specified distance relative to the current position. |
| J | JOG | Move at the specified jog speed. |
| L | LOOP (REPEAT) | Transfer program execution to a program label a specified number of times, and then continue program execution with the next command in the program buffer. |
| N | NORMALIZE | Define the current physical position. Reset the PMC-960 firmware. Initiate Serial Communications Interface Autobaud Sequence. |
| O | OUTPUT | Set general purpose Machine outputs. |
| OA | analog | - set analog output |

| | | |
|---|---|---|
| OB | binary | - set binary outputs |
| P | **PROGRAM** | Enter, edit or examine a motion program. |
| Q | QUIT | Terminate execution of MPL from the program buffer and return to the interactive (READY) mode. |
| R | **CONTOUR** | Cause the PMC to execute a complex contoured motion, where the motion is defined in position-time segments. |
| S | **SYSTEM** | Select System Parameters. |
| SB | last label | - display last label and command count |
| SD | adaptive depth | - set adaptive depth |
| SE | error codes | - display last error code |
| SF | following error | - display/set following error |
| SI | inputs | - set general purpose machine inputs |
| SL | software limits | - set software end-of-travel limits |
| SM | mode | - system mode (idle, velocity or position modes) |
| SN | normalize error | - display normalization error |
| SP | profile | - motion profile parameters |
| SS | snapshot | - display snapshot of motion profile |
| ST | torque | - set torque level at positive stop |
| SV | feedrate | - system feedrate override (velocity) |
| SW | write enable | - enable writing to program buffer & table parameters |
| T | **TABLE** | Select machine configuration in Parameter Table. |
| TF | feedforward | - display table of feedforward gains |
| TG | **TABLE GAINS** | - select gain and compensation values |
| TGP | position loop | - select position loop gain |
| TGV | velocity loop | - select velocity loop gain |
| TGT | tach | - select tach gain |
| TGF | feedforward | - select feedforward gain |
| TGE | pos compensator | - select position loop compensator |
| TGB | vel compensator | - select velocity loop compensator |
| TGX | external vref | - select external velocity reference gain |
| TGL | analog lock | - select analog lock gain |
| TU | **TABLE UNITS** | - select units conversion & maximum A,V,H parameters |
| TUA | acceleration | - acceleration units conversion factor |
| TUP | position | - position units conversion factor |
| TUS | velocity | - velocity units conversion factor |
| TUB | max accel | - maximum acceleration parameter |
| TUV | max velocity | - maximum velocity parameter |
| TUH | home speed | - home command speed parameter |
| TH | **TABLE HARDWARE** | - specify system hardware configuration |
| THA | absolute counts | - specify absolute counts per revolution |
| THB | baud rate | - specify baud rate |
| THC | communications | - specify communications options |
| THI | hardware ID | - assign hardware ID |
| THL | limit switches | - enable limit switches (on/off) |
| THM | address outputs | - enable muxed address outputs (on/off) |
| THR | motion reference | - select motion reference configuration |
| THS | software limits | - enable software limit control |
| TL | **TABLE LABELS** | - store default program <labels> to parameter table |

| | | |
|---|---|---|
| **TLB** | MPL branch | - select label to execute from MPL branch command |
| **TLP** | power-up | - select label to execute on power-up or **N*** command |
| **TLV** | machine I/O | - select label to execute from Machine I/O (EXEC') |
| **U** | **UNTIL** | Wait until the specified machine input condition is true before executing the next MPL command. |
| **V** | **VELOCITY** | Set or examine index speed in the Motion Buffer. |

## 5.2 MPL SYNTAX OVERVIEW

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

```
<     > - designates a required variable
#     # - the enclosed item (or items) may be repeated multiple times
[     ] - the enclosed item (or items) are optional
   |    - designates the OR operator
```

```
Acceleration    A  <rate> <cr>                                    |  A <display>
Branch          B  <label> [<hex>[:<mask>]] [<sync>] <cr>
Cam             C  [<sign>] <position> [:<speed>] ^
                C  [<sign>] <position> [:<speed>] <cr>
                C^
Dwell           D  [<time>] [<sync>] <cr>
Exit Program    E  [<hex>[:<mask>]] [<sync>] <cr>
Function Call   F  <label> [<hex>[:<mask>]] [<sync>] <cr>
Go              G  <position> <direction> <cr>                    |  G <display>
Home
 -resolver      H  [<speed>] <direction> <cr>                     |  H <display>
 -encoder       H  <direction>                                    |  H <display>
Index           I  [<distance>] #<direction># <cr>                |  I <display>
Jog             J  [<speed>] #<direction># <cr>                   |  J <display>
Loop            L  <label> [<loop counter ID>] <count> <cr>
Normalize       N  [<position>] <direction>     |   N*            |  N<cr>
Output          OA <hex> [:<mask>] <cr>                           |  OA <display>
                OB <hex> [:<mask>] <cr>                           |  OB <display>
Program         P  <program> <text>                               |  P <label> <text>
Quit            Q  [<hex>[:<mask>]] [<sync>] <cr>
System          SB <display>
                SD <depth> <cr>                                   |  SD <display>
                SE <display>
                SF <error> <cr>                                   |  SF <display>
                SI [<hex>] <display>
                SM <mode> <cr>                                    |  SM <display>
                SN <display>
                SP <hex>                                          |  SP <display>
                SS <display>
                ST <torque> <cr>                                  |  ST <display>
                SV <value> <cr>                                   |  SV <display>
                SW <value> <cr>                                   |  SW <display>
Table           T <display>
                TF <display>
                TG #<register> [<value>] #<sign>##<cr>            |  TG <display>
                TH <parameter> <value> <cr>                       |  TH <display>
                TL <vector> <label> <cr>                          |  TL <display>
                TU <parameter> <value> <cr>                       |  TU <display>
```

| | | |
|---|---|---|
| Until | U [<hex>[:<mask>]] [<sync>] <cr> | |
| Velocity | V <speed><cr> | \| V<display> |
| (Label) | @ <label> <text> <cr> | \| @ <text> <cr> |

**Explanation of Syntax Variables**

| | | |
|---|---|---|
| <count> | Number of times for operation to be repeated | |
| <cr> | Carriage return (0DH) | |
| <depth> | distance in counts which gives the stopping distance when using adaptive depth control | |
| <direction> | +/- | positive/negative <speed>, <position> or <distance> |
| | * | stop system motion |
| | <sync> | permitted in most cases |
| <display> | ? | display last entered value |
| | ! | display current system value |
| | % [<time>] | display value each <time> interval |
| | & | display value at time of last error |
| <distance> | integer in user defined units which gives the number of relative units | |
| <error> | integer specifying following error | |
| <hex> | Hexadecimal characters (0-9, A-F) | |
| <label> | Displayable character used to identify a motion routine | |
| <mode> | Control mode: 0=idle; 1=velocity; 2=position; 3=position without resetting position summing junction | |
| <position> | integer in user defined units of absolute position | |
| <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> | integer in user defined units of acceleration | |
| <register> | Table (Gain): | |
| | P=(position gain) | |
| | V=(velocity gain) | |
| | T=(tach gain) | |
| | F=(feedforward gain) | |
| | E=(position loop compensation) | |
| | B=(velocity loop compensation) | |
| | X=(external velocity reference gain) | |
| | L=(Analog Lock Gain) | |
| <sign> | +/- | add/subtract <value> to/from <register> |
| | <cr> | set <register> to <value> |

<speed> integer in user defined units of speed

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

<text> Motion routines, comments or editing command characters

<time> Time in milliseconds

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

## 5.3 MPL CONVENTIONS

Now that we have an understanding of the purpose of each command, there are basic constructs in the architecture of MPL which deserve special mention since they are widely used and to make it easy to understand and use.

### 5.3.1 Display Characters

MPL simplifies monitoring a motion control application by displaying information on the actual status of the system. This includes providing information on parameters in the motion buffer by typing a command and a (?), data on the system's current values by using a (!), repeatedly displaying the system's current values by using a (%), or the system's values as they were on the last error by using a (&).

### 5.3.2 Synchronization Characters

MPL's synchronization characters (, and ;) offer an effective method for coordinating MPL commands with motion in progress. The comma (,) waits for the previous motion to be completed before executing a new command, and the semi-colon (;) waits for the system to reach a constant speed or the motion to be completed before executing a new command.

### 5.3.3 System Status Polling

System status polling is a feature which allows the user to monitor PMC operation. A simple two-character sequence entered at any time during PMC operation will return a variety of status information.

The following table gives the recognized characters and what status values will be returned. The value will be returned in the same format as if a ! terminator had been used in an MPL command. System status polling for each of these functions is executed by holding down the 'Control' and pressing the ']' key followed by the single-letter command below.

| Command | Parameter | Equivalent MPL Command |
|---|---|---|
| b | Last @ label passed | --- |
| f | Current Following Error | SF! |
| g | Current System Position | G! |
| h | Communications Checksum | --- |
| i | Distance to go | I! |
| m | Motion parameters | SS! |
| s | System Inputs | SI! |
| v | Current System Velocity | V! |

Note: These single letter commands must be entered as lower case letters for System Status Polling to operate properly.

### 5.3.4 Error Status Buffer

The PMC-960 has a feature where several system values are stored away when an error occurs. An error can be caused by an external fault, an impossible command request, etc. It is, many times, useful to know what the state of the PMC-960 was when an error happens. To retrieve this information the ampersand (&) terminator is used for the various commands as you would use the ! or ? terminator. The parameters available are: system speed (V&), system position(G&), system snapshot (SS&), general purpose inputs (SI&), last label passed (SB&), general purpose outputs (OB&), system mode (SM&) and last error message (SE&).

### 5.3.5 MPL Break Features

When MPL is executing a program or waiting for some event to finish (e.g. , ; dwell command) it is sometimes desirable to abort the program. This can be done by sending an ESCAPE character (ASCII 33H) if the SCI is established. If the SCI has not been established (meaning that the baud rate has not been specified) then any character sent to the PMC-960 on the SCI will abort the program execution.

There are four signals which can be used to break MPL program execution, stop the motor using a fast deceleration and/or disable the motor. The following chart details the specific function of each of these signals:

| Signal | Break MPL Program | Emergency Stop | Disable Motor |
|---|---|---|---|
| BREAK' (JM3) | yes | no | no |
| ESTOP' (JM3 & JM1) | yes | yes | yes |
| FAULT' (JM3) | yes | yes | no |
| DRVRDY' (JM2) | yes | no | yes |

### 5.3.6 MPL Communications Checksum

The PMC maintains a checksum for all serial communications. This is a 16 bit sum of all characters received since the last communications checksum system status poll request. Note that system status poll requests are not included in the checksum total.

When a communications checksum system status poll is received, the PMC will return the current checksum total and also reset the checksum to 0. The checksum display request can be sent at any time.

## 5.4 SYSTEM PARAMETER REGISTERS

There are several read-only system status registers that provide feedback to the user on the status of system operation. They are: last error code (SE), general purpose inputs (SI), normalize value (SN), and a motion snapshot register (SS). The system status registers are examined with the S command.

The read-write SP system register offers the user a variety of methods for starting and stopping motion, providing a wide range of motion profiles. The SP register is organized as 8 bit-switches which turn on and off certain features of the Index and Go command motion profiles. These features include: starting and stopping on external sensor, starting and stopping on the zero reference input, extending an Index motion until a sensor or zero reference signal, etc.

## 5.5 PARAMETER TABLE

The PMC-960 has a system parameter table which defines the default operating mode. This parameter table is stored in non-volatile RAM and can only be changed when a protection bit has been set to allow non-volatile modification. Once this bit is set, the user can use the T command to modify parameters. The T command is split up into several subcommands each with a sub-command character identifying it.

Examples of the subcommand are: **Gain Adjustments (TG)** containing values associated with the velocity and position loops, **Units (TU)** which consists of some limits on numeric values and the engineering constant, **Hardware (TH)** which has parameters for SCI communications, reference configurations, limit input modes, etc., and finally **Labels (TL)** which contains definitions of the powerup label and vectored execute label.

## 5.6 USE OF PARAMETER TABLE ON POWERUP

When power is applied to the PMC-960, parameters from the parameter table area are used to determine serial communications configuration, MPL routine to execute, gain values to set, etc. In some cases, it is possible that the user could set a parameter which would effectively lock himself out of communicating with the PMC-960 (i.e. Baud Rate value or Axis ID). To help the user in this kind of situation, the PMC-960 has a method of being powered up in a default mode other than that specified by the parameter table.

To power up the PMC-960 in the generic mode, the ESTOP' line must be asserted (grounded) before powering up or the RESET' line is pulsed. When that is done, many of the parameters are ignored and a value of zero is assumed and no powerup MPL program will be executed. Since the baud rate parameter is assumed zero, the normal autobaud sequence is executed and the communications format is interactive decimal for ease of debugging.

Additionally, if any of ESTOP', FAULT' or BREAK' are asserted (grounded) during powerup then the powerup MPL program will not be executed. Note that the test of these signals is done after the powerup diagnostics execute which results in the signals being tested about 2 seconds after power-up.

## 5.7 SETUP PARAMETER RANGES AND UNITS

| | | Range | Factory Settings | Units |
|---|---|---|---|---|
| **Motion Parameters** | | | | |
| 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 | 4000 | kHz/sec |
| Jog | | 1- 1,920 | 100 | 100 Hz |
| Velocity | | 1- 1,920 | 400 | 100 Hz |
| Home | | 1- 1,920 | 20 | 100 Hz |
| 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% |
| Index | | 1-2,147,483,648 | - | counts |
| Go | | 0-1,073,741,824 | 0 | counts |
| Normalize | | 0-1,073,741,824 | 0 | counts |
| Dwell | | 0- 65,535 | 0 | msec |
| <Label> | | $22_H$ to $7A_H$ | - | - |
| **Tuning Parameters** | | | | |
| Position Loop Gain | | 0-255 | 16 | - |
| Velocity Loop Gain | | 0-255 | 16 | - |
| Tachometer Gain | | 0-255 | 160 | - |
| Feedforward Gain | | 0-255 | 16 | - |
| Position Loop Compensator | | 0-255 | 0 | - |
| Velocity Loop Compensator | | 0-255 | 0 | - |
| External Vel Ref Gain | | 0-255 | 0 | - |
| Analog Lock Gain | | 0-255 | 0 | - |

## 5.8 EDITING FUNCTIONS USED DURING PROGRAM MODE

**Cursor Right** — TAB (CTRL-I) or CTRL-Y moves the cursor to the right one character at a time. Moving the cursor to the end of the line and continuing to tab will move the cursor to the beginning of the next line.

**Cursor Left** — BACKSPACE (CTRL-H) or DELETE moves the cursor to the left one character at a time. Moving the cursor to the beginning of the line and continuing to backspace will move the cursor to the beginning of the previous line.

**Cursor Down** — LINEFEED moves the cursor down a line at a time.

**Changes** — To change a motion control program, put the cursor at the point to be changed and overtype the desired information. Periods (.) may be used to overtype additional undesired characters or to reserve program buffer space for additional future commands or parameter changes.

**Insert Line** — Typing CTRL-V allows text to be inserted in the program buffer at the point of the cursor. After a CTRL-V, all characters typed are put into a 40 character RAM buffer until a second CTRL-V is typed or the RAM buffer is full. At that time, space is made in the program buffer and the characters are written to program buffer memory. If the insert operation is a result of the 40 character buffer being full, the insert operation is continued after the buffer is emptied. An ESCAPE can be used to exit from insert mode without inserting any characters.

**Kill Line** — CTRL-K deletes unwanted characters in the program buffer. When CTRL-K is typed, all characters from the cursor to the end of the line (next carriage return) will be deleted.

**Exiting** — The ESCAPE key is used for exiting 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 that executing this command will erase all information from the cursor to the end of the program buffer.**

## 5.9 HARDWARE ACCESSIBLE MOTION ROUTINES

HARDWARE ACCESSIBLE MOTION ROUTINES

| Program Label | (Signals at connector JM3) JM3- | $\overline{ADR4}$ 7 | $\overline{ADR3}$ 9 | $\overline{ADR2}$ 11 | $\overline{ADR1}$ 13 | $\overline{ADR0}$ 15 |
|---|---|---|---|---|---|---|
| @ ($40_H$) | | 1 | 1 | 1 | 1 | 1 |
| A ($41_H$) | | 1 | 1 | 1 | 1 | 0 |
| B ($42_H$) | | 1 | 1 | 1 | 0 | 1 |
| C ($43_H$) | | 1 | 1 | 1 | 0 | 0 |
| | 2 address lines | | | | | |
| D ($44_H$) | | 1 | 1 | 0 | 1 | 1 |
| E ($45_H$) | | 1 | 1 | 0 | 1 | 0 |
| F ($46_H$) | | 1 | 1 | 0 | 0 | 1 |
| G ($47_H$) | | 1 | 1 | 0 | 0 | 0 |
| | 3 address lines | | | | | |
| H ($48_H$) | | 1 | 0 | 1 | 1 | 1 |
| I ($49_H$) | | 1 | 0 | 1 | 1 | 0 |
| J ($4A_H$) | | 1 | 0 | 1 | 0 | 1 |
| K ($4B_H$) | | 1 | 0 | 1 | 0 | 0 |
| L ($4C_H$) | | 1 | 0 | 0 | 1 | 1 |
| M ($4D_H$) | | 1 | 0 | 0 | 1 | 0 |
| N ($4E_H$) | | 1 | 0 | 0 | 0 | 1 |
| O ($4F_H$) | | 1 | 0 | 0 | 0 | 0 |
| | 4 address lines | | | | | |
| P ($50_H$) | | 0 | 1 | 1 | 1 | 1 |
| Q ($51_H$) | | 0 | 1 | 1 | 1 | 0 |
| R ($52_H$) | | 0 | 1 | 1 | 0 | 1 |
| S ($53_H$) | | 0 | 1 | 1 | 0 | 0 |
| T ($54_H$) | | 0 | 1 | 0 | 1 | 1 |
| U ($55_H$) | | 0 | 1 | 0 | 1 | 0 |
| V ($56_H$) | | 0 | 1 | 0 | 0 | 1 |
| W ($57_H$) | | 0 | 1 | 0 | 0 | 0 |
| X ($58_H$) | | 0 | 0 | 1 | 1 | 1 |
| Y ($59_H$) | | 0 | 0 | 1 | 1 | 0 |
| Z ($5A_H$) | | 0 | 0 | 1 | 0 | 1 |
| [ ($5B_H$) | | 0 | 0 | 1 | 0 | 0 |
| \ ($5C_H$) | | 0 | 0 | 0 | 1 | 1 |
| ] ($5D_H$) | | 0 | 0 | 0 | 1 | 0 |
| ^ ($5E_H$) | | 0 | 0 | 0 | 0 | 1 |
| _ ($5F_H$) | | 0 | 0 | 0 | 0 | 0 |
| | 5 address lines | | | | | |

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

## 5.10 EPROM PROGRAM BUFFER INITIALIZATION

The PMC supports an EPROM based program buffer initialization. This initialization will take place on a power-up or n* software reset, and result in copying the contents of the EPROM to the Program Buffer (non-volatile RAM).

A 2764 may be placed in the U46 chip site. If the PMC finds that the 1st 8 bytes of the EPROM are the seven-bit ASCII Codes for "@PROGRAM" (not including the quotes) then the contents of the EPROM will be copied to the program buffer. This copying will start at the beginning, including the @PROGRAM label, and end when either a $0ff_H$ is found in the EPROM, or 1940 bytes have been copied.

Note that any data in the program buffer before the transfer will be overwritten. Also note that the string @PROGRAM must be in all capital letters and must begin at address 0 of the EPROM, with the first eight bytes of data as follows:

| | |
|---|---|
| Address | 00 01 02 03 04 05 06 07 |
| EPROM Contents (hex) | 40 50 52 4F 47 52 41 4D |

When the copying process is complete the automatic powerup routine will execute, if present.

# ADVANCED PMC OPERATION

## 6.1 MACHINE I/O OPERATION

There are a total of 16 general purpose inputs IN15' through IN00'. The top 8 bits (IN15'-IN08') are used only by the **SI-System Inputs** command. Machine inputs IN07'-IN00' can be accessed by System Input command as well as used to direct program flow conditionally using the conditional Branch (Go To), Function (Subroutine) Call, and Exit (Return) statements. In addition, program execution can be caused to wait for machine inputs using the Until command.

An example of the use of input conditions is the **U6:7** command, which will cause the MPL program to disregard inputs IN07' through IN03', but wait "until" IN02' and IN01' are both asserted and IN00' is unasserted.

Sixteen general purpose Machine Outputs are provided which may be controlled by the Output command in MPL.

## 6.2 USE OF GENERAL PURPOSE I/O SIGNALS

The PMC-960's sixteen digital inputs and sixteen digital outputs can be used by the MPL program and/or a host computer controlling the PMC-960. The default configuration of these I/O points is: all sixteen outputs are controlled by the Output command, all sixteen inputs can be read by the SI command, and the lower 8 input bits (IN07' to IN00') can be used in the F,B,U,E commands.

A bit in the parameter table allows the user to redefine two of the inputs, IN07' and IN06', as travel limit switches in the + and - direction respectively. When this configuration is chosen, travel in the + direction

will be stopped (fast deceleration) when the IN07' signal is asserted. Likewise in the other direction.

Another common use of the general purpose I/O is to define some of the output bits to be "bank select bits" which external hardware uses to chose a bank of information to be sent to the input signals. This is useful to expand the amount of I/O points but still maintain a reasonable interconnection count. Also expansion can be application dependent without the use of special firmware to handle additional I/O points.

A typical configuration might be to assign OUT13' and OUT12' as signals to select one of 4 banks of eight signals which would be applied to the IN15' through IN08' lines. The reason to use the upper byte of input bits is that the information coming through the banking scheme is usually informational only and are generally not used in the MPL program control (B, U, etc.) The bank select method of expanding the number of input signals can also be applied to output signals if the external hardware consists of latches that would be strobed by another output bit.

# MPL OPERATION

## 7.0 MPL COMMAND DESCRIPTION

Throughout this section, the following symbols are used:

```
<     > - designates a required variable
#     # - the enclosed item (or items) may be repeated multiple times
[     ] - the enclosed item (or items) are optional
   |    - designates the OR operator
```

Whenever motion parameters are referred to in this section, they will be shown in the default units. Velocity will be shown in RPM; acceleration will be shown in RPM/sec; position distances will be shown in position transducer increments. The default assumption for position resolution is 4096 counts per revolution. The unit system may be changed by the user and is designed to be user-friendly, but causes additional processor overhead. This overhead can be reduced by choosing integer units consistent with internal workings of the PMC.

## 7.1 @ - LABEL MOTION CONTROL PROGRAMS

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

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

<label> a single byte program label. (See Section 7.1.1)

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

Example:
```
@.This_is_a_comment_line
@X_Comment_after_a_program_label_'X'
```

### 7.1.1 MPL Program Label Support

The standard MPL program command supports only the printable ASCII characters (" $22_H$ - lower case z $7A_H$) as program labels. However, using the binary programming mode it is possible to use some additional non-printing labels. The following list contains all the labels which can be placed in the program buffer using the binary program mode and which will be supported by the other MPL commands.

All 256 8 bit ASCII codes with the following exceptions:

$00_H$ (NUL)
$09_H$ (TAB)
$0a_H$ (LF)
$0d_H$ (CR)
$1b_H$ (ESC)
$1d_H$ (^])
$ff_H$ (DEL)

Care should be taken when using non-printing characters. Display problems may result on some terminals. Also, care should be exercised when using the ? and ! and { and } labels. They can be input using the binary program mode but can not be reached using the Program command and may be misleading to some users.

## 7.2 A - ACCELERATION COMMAND

Purpose: Set or examine acceleration rate

Syntax: **A <rate> <cr> | A <display>**

<rate> integer specifying the acceleration rate in user defined units

<display>
- ! display current system acceleration rate (zero if at rest or top speed)
- ? display last entered acceleration rate
- % [<time>]<cr> repeatedly display the current acceleration rate (A! output) until an SCI character is received. <time> is the rate in msec at which the % output is repeated. (default: 100)

Examples:

| | |
|---|---|
| A3500<cr> | set acceleration rate to 3500 RPM/sec |
| A? | display last entered acceleration rate |
| A! | display current system acceleration rate |

## 7.3 B - BRANCH (GO TO) COMMAND

Purpose: transfer MPL program execution to a program label

Syntax: **B <label> [<hex>[:<mask>]] <cr>**

<label> a single byte program label; (See Section 7.1.1) If a vertical bar ($7C_H$) is used, the Branch command will cause MPL to continue execution at the label specified by the branch vector label parameter (TLB) in the parameter table.

<hex> a hexadecimal number that is matched with the values of corresponding machine input bits; the hex number defines the input condition necessary for a branch to occur. A bit set to 1 defines an active signal level; a bit set to 0 defines an inactive signal level.

<mask> a hexadecimal number that specifies which machine input bits are to be compared and which are to be ignored. (Default is FFH) Mask bits that are set to 1 cause corresponding machine input bits to be compared to the <hex> parameter; Mask bits that are set to 0 cause the corresponding machine input bits to be ignored.

Example:

| | |
|---|---|
| BQ<cr> | unconditional branch to label Q |
| BQ1:1<cr> | branch to program label Q if input IN00' is low, no other bits are checked |
| BQ10:F0<cr> | branch to label Q if IN04' is low and IN07'-IN05' is high. IN03'-IN00' are ignored. |

## 7.4 C - CAM COMMAND

Purpose: perform distance-based changes of speed

Syntax:
```
C [<sign>] <position> [:<speed>] <cr>
C [<sign>] <position> [:<speed>] ^
C^
```

<sign>
- \+ specify positive absolute <position>.
- \- specify negative absolute <position>.

<position> an integer in user defined units specifying a point in motion relative to the absolute origin. The system attempts to reach the new <speed> by the time this <position> is attained.

<speed> an integer in user defined units specifying new camming speed to be reached by specified <position>

<cr> causes this Cam profile segment to be executed

^ synchronization character which waits for the end of a cam segment before executing the next command

**CAUTION:** To reverse direction using the Cam command, a zero speed position must be specified.

Examples:

| | |
|---|---|
| C1000:100 | Cam to position 1000 counts in the forward direction reaching a velocity of 100 RPM by the end of the cam segment |
| C2000^ | Cam to position 2000 counts in the forward direction using the previously selected camming speed; the ^ waits for the last cam segment to be completed before executing this command |

## 7.5 D - DWELL COMMAND

Purpose: delay a specified time interval before executing the next command

Syntax: **D [<time>] [<sync>] <cr>**

<time> integer (0 to 65,535) in milliseconds 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 dwell 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.

<sync> , synchronizing character which causes the PMC-960 to wait for the system motion to stop

; synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

Note: An ESCAPE entered during the execution of this command will end this command with a #D0 error.

Example:

| | |
|---|---|
| D16<cr> | delay for 16 msec |
| D24,<cr> | delay for 24 msec beginning after motion stops |
| D300;<cr> | delay 300 msec after reaching steady state speed |

## 7.6 E - EXIT (RETURN) COMMAND

Purpose: return from subroutine to statement after function call

Syntax: **E [<hex>[:<mask>]] <cr>**

<hex> a hexadecimal number that is matched with the values of corresponding machine input bits; the hex number defines the input condition necessary for an exit (return) to occur. A bit set to 1 defines an active signal level; a bit set to 0 defines an inactive signal level.

<mask> a hexadecimal number that specifies which machine input bits are to be compared and which are to be ignored. (Default is FFH) Mask bits that are set to 1 cause corresponding machine input bits to be compared to the <hex> parameter; Mask bits that are set to 0 cause the corresponding machine input bits to be ignored.

Note: This command is valid only in program mode.

Example:

| | |
|---|---|
| E<cr> | unconditional (return) exit |
| E8:8<cr> | exit (return) if input IN03' is low |

## 7.7 F - FUNCTION COMMAND

Purpose: conditional call to a subroutine

Syntax: **F <label> [<hex>[:<mask>]] <cr>**

<label> a single byte program label; (See Section 7.1.1)

<hex> a hexadecimal number that is matched with the values of corresponding machine input bits; the hex number defines the input condition necessary for a function call to occur. A bit set to 1 defines an active signal level; a bit set to 0 defines an inactive signal level.

<mask> a hexadecimal number that specifies which machine input bits are to be compared and which are to be ignored. (Default is FFH) Mask bits that are set to 1 cause corresponding machine input bits to be compared to the <hex> parameter; Mask bits that are set to 0 cause the corresponding machine input bits to be ignored.

Example: FA<cr> unconditional call of routine 'A'

## 7.8 G - GO COMMAND

Purpose: move to the specified absolute position of the system

Syntax: **G <position> [<sync>] <sign> <cr> | G <display>**

<position> an integer in user defined units specifying the absolute position of the system (default: 0)

<sign>
- + 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-960 to wait for the system motion to stop
- ; synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

<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
- % [<time>]<cr> repeatedly display the current position (G! output) until an SCI character is received. <time> is the rate in msec at which the % output is repeated. (default: 100)
- & display absolute position when last error occurred

<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.

Example:

| | |
|---|---|
| G! | display the present absolute position of the system |
| G+<cr> | go to the absolute zero position of the system |
| G%<cr> | display current absolute position every 100 msec. |

## 7.9 H - HOME COMMAND

### 7.9.1 Home Command Using Resolver Feedback

Purpose: move to the specified resolver position or sensor

Syntax: **H [<location>] [<sync>] [<direction>] <cr> | H <display>**

<location> number in resolver counts which is the destination of the Home command around the revolution of the motor. This number is limited by the counts per revolution specified by the **THA** parameter.

<direction>
- `+` move at the homing speed in the positive direction until the <location> is reached or sensor is detected (see EXTERNAL STOP SELECT bit in the **SP** command)
- `-` move at the homing speed in the negative direction until the <location> is reached or sensor is detected (see EXTERNAL STOP SELECT bit in the **SP** command)
- `*` stop system motion

<sync>
- `,` synchronizing character which causes the PMC-960 to wait for the system motion to stop
- `;` synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

<display>
- `!` display current resolver value
- `?` display last entered <location>
- `%` [<time>]<cr> repeatedly display the current resolver value until an SCI character is received. <time> is the rate in msec at which the % output is repeated. (default=100)

Note: The home speed parameter is specified by using the **TUH** command.

Examples:

| | |
|---|---|
| H15+<cr> | move in the positive direction to resolver position 15 counts |
| H,-<cr> | wait for system to come to rest before homing in the negative direction |
| H* | stop system motion |

### 7.9.2 Home Command Using Encoder Feedback

Purpose: move to the encoder reference or sensor

Syntax: **H [<sync>] <direction> | H <display>**

<direction>
- \+ move at the homing speed in the positive direction until the encoder reference or sensor is detected (see EXTERNAL STOP SELECT bit in the **SP** command)
- \- move at the homing speed in the negative direction until the encoder reference or sensor is detected (see EXTERNAL STOP SELECT bit in the **SP** command)
- * stop system motion

<sync>
- , synchronizing character which causes the PMC-960 to wait for the system motion to stop
- ; synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

<display>
- ! display current system speed
- % [<time>]<cr> repeatedly display the current system speed until an SCI character is received. <time> is the rate in msec at which the % output is repeated. (default=100)

Note: The home speed parameter is specified by using the **TUH** command.

Examples:
- H+ move in the positive direction to the encoder reference or sensor
- H,- wait for system to come to rest before homing in the negative direction
- H* stop system motion

## 7.10 I - INDEX COMMAND

Purpose: move a specified distance relative to the current position

Syntax: **I [<distance>] #<direction># [<sync>] <cr> | I <display>**

<distance> an integer in user defined units specifying the relative distance to move. 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-960 to wait for the system motion to stop
- `;` synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

<display>
- `!` display the distance remaining in the current or last index
- `?` display the last entered index distance
- `%[<time>]<cr>` repeatedly display the remaining distance in the current or last index (repeating I! output) until an SCI character is received. <time> is the rate in msec at which the % output is repeated.

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 |
| I,+,- | wait for last motion to end; index in the positive direction; after this motion is stopped; index in the negative direction |
| I! | display the number of remaining counts in the current move |
| I? | display the previously specified relative distance <distance> |
| I* | stop the current index |
| I%200<cr> | display the number of remaining counts in the current move. Update the display every 200 msec. |

## 7.11 J - JOG COMMAND

Purpose: move at the specified jog speed

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

<speed> an integer in user-defined units specifying the jog rate

<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-960 is running and still in the middle of a J command.

<sync>
- `,` synchronizing character which causes the PMC-960 to wait for the system motion to stop
- `;` synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

<display>
- `!` display current system speed
- `?` display last entered jog speed
- `%[<time>]<cr>` repeatedly display the current system speed until an SCI character is received. <time> is the rate in msec at which the % output is repeated

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.

Example:

| | |
|---|---|
| J! | display the current system speed |
| J? | display last entered jog speed |
| 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 |
| 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 |

## 7.12 L - LOOP COMMAND

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

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

<label> a single byte program label; (See Section 7.1.1)

<loop counter ID> character X, Y or Z which specifies the loop counter to be used. This allows loops to be nested up to 3 levels.

<count> a number between 0 and 65,535.

<cr> causes the loop command to be executed

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

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

then executing the command LT999 <cr> from interactive mode will execute 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 IN01' stays high. If IN01' stays high, then when all 3000 indexes are done the 'W' routine is executed. If IN01' 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.

Example:

```
LB20<cr>      loop to label 'B' 20 times
LBX20<cr>     loop to label 'B' 20 times using loop counter X
```

## 7.13 N - NORMALIZE COMMAND

Purpose: define current physical position or reset PMC-960

Syntax: **N [<position>] [<sync>] <sign> | N <cr>**

<position> an integer in user-defined units specifying the absolute position of the system (default: 0)

<sync>
- , synchronizing character waits for the system motion to stop
- ; synchronizing character which causes the PMC-960 to wait for the system motion to reach a constant speed or for motion to stop

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

<cr> initiate Serial Communications Interface Autobaud sequence and then output MPL firmware identifier when the autobaud process is complete

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

Example:

| | |
|---|---|
| N2000,+ | wait for system motion to stop; set absolute position counter to +2000 |
| N* | software reset |
| N<cr> | display firmware ID |

## 7.14 O - OUTPUT COMMAND

### 7.14.1 OA - Set Analog Outputs

Purpose: set general purpose analog machine outputs

Syntax: **OA <value> <sign> <cr> | OA <display>**

<value> a decimal value that defines the desired voltage level of the general purpose analog output. This value can range from 0 to 2047 representing voltage outputs between 0 and 10 VDC. The polarity of the voltage is specified by <sign>.

<sign> the sign of the analog voltage output

<display>
- ? display the current setting of the general purpose analog output
- ! same as ?

### 7.14.2 OB - Set Binary Outputs

Purpose: set general purpose binary machine outputs

Syntax: **OB <hex> [:<mask>] <cr> | OB <display>**

<hex> a 16 bit hexadecimal number that is matched with the values of corresponding machine output bits; the hex number defines the output states for each bit. A bit set to 1 defines an active (low) signal level; a bit set to 0 defines an inactive (high) signal level.

<mask> a 16 bit hexadecimal number that specifies which machine outputs bits are to be set. (Default is FFFF) Mask bits that are set to 1 cause corresponding machine output bits to be set according to the <hex> parameter; Mask bits that are set to 0 cause the corresponding machine output bits to remain unchanged.

<display>
- ? display the last entered state of the general purpose machine outputs. A four digit hexadecimal value will be returned.
- ! same as ?
- & display state of machine outputs when last error occurred

Example:

| | |
|---|---|
| OBC | specify output pattern 000CH; This pattern specifies that outputs OUT03' and OUT02' are low, and that all other outputs are high |
| OBC:C | all output bits remain the same except OUT03' and OUT02' which are set low |
| OB? | display the current state of the binary outputs |

## 7.15 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 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(1BH) 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. No changes to the Program Buffer are allowed in 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 a BELL will sound.

<label> a single byte program label; (See Section 7.1.1) Initiate programming at the beginning of the program with this label.

Example:

| | |
|---|---|
| P? | display the first line 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. |

### 7.15.1 Binary Programming Command

The binary programming command is designed for compact programming of a PMC-960 from a host computer. It is used by enabling host computer communications by setting upper bit of the Communications Format byte. (See THC command.)

Purpose: enter, edit or examine motion programs using binary mode

Syntax: **P <program> | P <label>**

<program> ? 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(1BH) character will terminate the output. No changes to the Program Buffer are allowed in this mode.

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

<label> specifies a specific motion control program. When it is found the PMC-960 outputs an @. If a left squirrelly bracket ({) is entered, programming starts at 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

**Note: Any other character overwrites the program buffer byte at the current position.**

## 7.16 R - CONTOUR COMMAND

Purpose: **create high performance profiled motion**

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

Purpose: R <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.

R <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 certain multiples 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.

## 7.17 S - SYSTEM PARAMETER COMMANDS

The group of **S - System Parameter Commands** are stored in volatile memory and aid in specifying motion parameters. Default values for these parameters are generally stored in a power-up program which puts the PMC-960 in a known state on power-up.

### 7.17.1 SB - Show last label and command count

Purpose: To display the last label passed (on **@** command) and the number of commands executed since the last **@** command.

Syntax: **SB <display>**

<display>
- ? output the last label passed as a character, followed by a space (except during binary output mode), followed by a numeric value which is the count of the number of commands entered since the last @ command was executed.
- ! same as ?
- & output last label and command count at the time of the last error.

### 7.17.2 SD - Specify Adaptive Depth Stopping Distance

Purpose: Specify the stopping distance to stop motion referenced to an external sensor. See SP command to enable this feature.

Syntax: **SD <depth> <cr>** | **SD <display>**

<depth> the distance to stop the adaptive depth motion after the sensor. This value must include the deceleration distance. The allowable range for this parameter is 3 to 65535.

<display>
- ? output the last entered <depth> parameter
- ! same as ?

Use of Adaptive Depth Control

Adaptive depth can be used on an Index motion, Cam motion or Go motion. To use it, the adaptive dpeth select bit in the motion profile register (see SP command) must be set. Then execute one of the motions: index some distance; go to an absolute point; or enter a cam command to a 0 speed at any position (must be running a cam profile before doing the cam command to a 0 speed). After the motion reaches top velocity, it will look for a sensor input. The motion will stop <depth> counts after sensor.

Note:
(1) The sensor must not occur before the motion reaches top velocity.
(2) <depth> > (deceleration distance + following error) is required.

### 7.17.3 SE - System Error (displays last error code)

Purpose: displays last error code

Syntax: **SE <display>**

| <display> | | |
|---|---|---|
| | ? | displays error code register - a single byte in HEX notation which reports the last error code |
| | ! | same as ? |
| | & | same as ? |

### 7.17.4 SF - System Following Error

Purpose: To display system following error; to allow the user to specify a variable following error trip point that will cause motion to stop or a Feed to Positive Stop motion to start

Syntax: **SF <value> <cr>** | **SF <display>**

<value> the set point, 1 to 32767, which determines the position summing junction (PSJ) overflow point. The set point is the nearest power of 2 value that is greater than or equal to <value>. Powerup/reset default is 2048.

| <display> | | |
|---|---|---|
| | ? | **display the last entered <value>** |
| | ! | displays position error register - a 16 bit value reported according to the Communications Format (see THC for more information) which shows system following error |
| | %[<time>]<cr> | repeatedly display the position error until an SCI character is received. <time> is the rate in msec at which the % output is repeated |

### 7.17.5 SI - System Inputs

Purpose: set or examine general purpose inputs

Syntax: **SI [<hex>] <display>**

<hex> A single hex digit which specifies which of the muxed input banks to display along with the non-banked inputs. This parameter is only allowed when the **THM** has defined muxed input support. If muxed input support is specified and <hex> is not given, then last specified bank will be displayed.

<display>
- ? displays the machine inputs in hex notation.
- ! same as ?
- %[<time>]<cr> repeatedly display the machine inputs until an SCI character is received. <time> is the rate in msec at which the % output is repeated.
- & display state of machine inputs when last error occurred

### 7.17.6 SL - Software Limits

Purpose: Specify two positional limits of travel for any motion.

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

<position> an integer in user-defined units specifying the absolute position of the limit

<sign>
- + set specified limit to plus <position>
- - set specified limit to minus <position>

<display>
- ? display current limit values
- ! same as ?

Example:

```
sl f10000       set forward travel limit to 10000
sl r5000-       set reverse limit to position -5000
```

Use of Software Limits

The software limits work the same way as the hardware limits. There is an enable/disable switch to turn their use on and off. When enabled, MPL will stop any motion when a limit is reached and will not allow motion into a limit

to start if the limit is already true. Like hardware limits, if the current position is on one limit it is possible to command motion off of that limit.

When a software limit is encountered during motion MPL will initiate a deceleration at the system maximum deceleration rate (TUB value). If the motion was caused by a **JOG** command no error message will be output. Motion caused by any other command will get an error message output.

There are two software limits, one for each direction of travel. Like the hardware limits each of the software limits is associated with a specific 'hardware' direction of travel. Consider the example;

```
.....|.....0.....|.....
    -5000          10000
  (SLR value)    (SLF value)
```

In this example the motion direction + moves to the right and motion direction - moves to the left. The software limits will not allow travel to positions greater than +10000 or less than position -5000.

Now let's set the direction invert bit (bit 4 of the THR register). This will mean that commanded motion direction + now travels to the left and direction- travels to the right. The software limits will treat the SLF value as limiting the travel to the right. This means that the furthest you can travel in the- direction is to absolute position -10000. Similarly the SLR value limits travel to the left, thus limiting + direction travel to absolute coordinate +5000. This can be understood by thinking of the SLF software limit as defining the distance from the absolute origin which the motor can move to the right, independent of whether a + or - motion command was used to cause the motion. Likewise, the SLR software limit defines the maximum position to the left of the absolute origin.

### 7.17.7 SM - System Mode

Purpose: set or examine the system mode

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 servodrives, or using a solid state relay.

1 enter VELOCITY Mode; In the VELOCITY Mode, the velocity loop is enabled, as is the SDRVEN signal (for enabling the servodrive).

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.

When an **SM2** command is entered, and the system is at rest, then the current absolute position of the system will be adjusted by the current position error in the Position Summing Junction. This adjustment will not be done if the system is in motion or a PSJ overflow condition exists when the **SM2** command is entered.

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.
**WARNING: This jump can be many revs at high speed!**

<cr> set Mode as specified by <mode>

<display> ! display current system mode information;
? same as !
& display system mode when last error occurred

Note: If a PSJ overflow takes place, the PMC-960 automatically enters IDLE Mode (with the Servodrive Enable signal (SDRVEN) disabled. Clearing this fault condition with an SM2 command will automatically clear the PSJ.

### 7.17.8 SN - System Normalization Error

Purpose: display normalization error cancelled on last normalize command

Syntax: **SN <display>**

<display> ? Displays normalization error register to report error cancelled when the last Normalize command was executed.
! Same as ?

### 7.17.9 SP - System Profile

Purpose: set or examine motion profile parameters

Syntax: **SP <hex> | SP <display>**

<hex> is a ASCII hex byte with the following bit definitions:

| | |
|---|---|
| Bit 7 | FEED TO POSITIVE STOP enables use of feed to positive stop support (1=on) which allows torque control against a positive stop. See **ST** command for additional information. |
| Bit 6 | ADAPTIVE DEPTH CONTROL enables user to specify stopping distance relative to external sensor on Cam, Index and Go commands. (1=on) See SD command for additional information. |
| 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 (SENSOR) or the zero reference to start a motion. (1=machine sensor, 0=zero 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 (SENSOR) instead of a calculated distance (after full speed is attained). 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 (SENSOR) or the zero reference to stop motion during a home command or an INDEX EXTEND. (1=machine sensor, 0=zero 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) |

Note: The P Register is in the motion buffer and therefore, altering it during a motion will only effect the next commanded motion.

### 7.17.10 SS - System Snapshot (displays Motion Profile register)

Purpose: displays motion profile register which provides information on system status

Syntax: **SS <display>**

Motion profile snapshot register - an 8 bit byte defined as:

| | |
|---|---|
| Bit 7 | DRVON indicates whether or not the servodrive is enabled. (1= drive ON) |
| Bit 6 | FPS (Feed to Positive Stop) indicates that the PMC is holding at a positive stop with torque specified by the **ST** command. |
| Bit 5 | Reserved |
| Bit 4 | CAM HOLD indicates that a cam segment has not yet been completed. (1=currently executing segment) |
| Bit 3 | MOTION indicates whether or not the system is in motion. (1= in motion) |
| Bit 2 | TOP VELOCITY indicates whether or not the system is currently at top velocity. 1= at top velocity |
| Bit 1 | DIRECTION of the last (or current) motion, if motion is in progress. 1= forward |
| Bit 0 | Reserved |

<display>

| | |
|---|---|
| ? | displays the selected status register. |
| ! | same as ? |
| %[<time>]<cr> | repeatedly display the selected status register until an SCI character is received. <time> is the rate in msec at which the % output is repeated |
| & | display snap shot register when last error occurred |

### 7.17.11 ST - Set Torque Level at a Positive Stop

Purpose: sets the percentage of torque to be applied when a positive stop is reached on a feed to positive stop motion

Syntax: **ST <torque> <cr>** | **ST <display>**

<torque> the percentage, in 0.1%, of torque to apply upon reaching a positive stop. Range is 0 to 1000.

<display> ? displays the last specified torque
! same as ?

Using Feed to Positive Stop

Feed to Positive Stop (FPS) is a state in which the PMC-960 is applying a constant torque rather than using position or velocity control. The positive stop is assumed to be a state such that there is only a small amount of motion at a slow speed.

To use the FPS feature the following sequence must take place:

1) The FPS control bit in the motion profile register (see **SP** command) must be set before the execution of the motion command which will cause motion into the positive stop.

2) The **SF** command must be set. FPS uses the software following error overflow to determine when the system has reached a positive stop. During FPS, the software following error trip point signals the change to a torque mode rather than stopping motion with an error.

3) The **ST** command must be set. This determines the percentage of torque to be applied at a positive stop.

4) A motion command must be entered to command motion into the positive stop. Any motion command that will reach the positive stop with sufficient commanded overtravel will work. (See next item.)

5) The PMC determines that it is at the positive stop by sensing that the following error (displayed by the **SF!** command) exceeds the value set in the **SF** command. When the positive stop is reached -
   - the commanded motion will be stopped
   - the position loop and velocity loop compensators will be disabled
   - the 'torque at positive stop' will be applied.
   - the **SS!** command shows no motion but the FPS status bit will be set

   Note that in order for the positive stop to be recognized the motion command driving into it must be commanded enough past the stop so that the following error will rise above the threshold set in the SF command.

6) To end the FPS state, any stop motion command should be entered (eg. i*, j*, g*, h*, c*). The PMC will then reduce the torque to 0, adjust the current absolute position to correctly reflect the current position, reset the PSJ, return the position loop and velocity loop compensators to the values currently in the non-volatile table and

close the position loop. Any command can then be issued to move away from the stop or into it again.

7) While on a positive stop it is assumed that there is little motion of the motor. Any motion will be kept track of so long as the PSJ does not overflow. If a limit becomes active, or if the system develops a leading error of 2048 counts, the FPS motion will be terminated.

### 7.17.12 SV - System Feedrate Override (velocity)

Purpose: set or examine the feedrate override

Syntax: **SV <value> <cr> | SV <display>**

<value> a decimal number between 1 and 100 which is the percentage of programmed speed the system will actually run.

<display>
- ? displays the selected status register.
- ! same as ?

### 7.17.13 SW - System Write Enable

Purpose: turn on/off write enable for program buffer and table values

Syntax: **SW <value> <cr> | SW <display>**

<value>
- 0 Do not allow changes to non-volatile RAM (P and T commands). This is the default value on powerup.
- 1 Allow the T and P commands to change non-volatile RAM. Changing the Program Buffer also depends on the state of the Hardware Write Protect Enable (THP Command) and, if enabled, the state of input IN5'.

<display>
- ? show the current setting of the W register.
- ! same as ?

## 7.18 T - TABLE MACHINE CONFIGURATION PARAMETERS

The **T** commands are intended to provide flexibility in selecting machine configuration parameters for a motion control application. These values are stored in non-volatile RAM and therefore don't need to be set each time the motion control system is powered-up.

Note that the display characters (? and !) function differently with the T commands than with other MPL commands. A ? is used to output the values in the Parameter Table. A ! is used to output the values currently being used by the PMC. These may be different from the Table Parameter values if ESTOP' was asserted at power-up.

When in decimal output mode, the values output by executing a ? or ! with one of the **T** commands will be preceeded by a <cr> <lf> and will be labelled and output in decimal unless a particular item is more usefully displayed in hexadecimal. When in hex output mode, all values will be output in hex with no labels and no separators or terminators between items. There will not be a leading <cr><lf> and no extra terminator following the list, just the usual prompt. When in binary output mode, all values will be output in binary with no labels and no separators or terminators between items. There will not be a leading <cr><lf> and no extra terminator following the list, just the prompt.

### 7.18.1 TF - Display Automatic Feedforward Gain Settings

Purpose: display the feedforward gain values which will be used when the velocity range has been changed

Syntax: **TF <display>**

<display> ? display feedforward gains for all velocity ranges as calculated when the **TGF** was executed
! same as ?

Output format will be:

48=<gain48> 192=<gain192> 384=<gain384> Ext=<gainExt>

<gain48> feedforward gain value used in 48 kHz velocity range
<gain192> feedforward gain value used in 192 kHz velocity range
<gain384> feedforward gain value used in 384 kHz velocity range
<gainExt> feedforward gain value used in Slave (External) mode

### 7.18.2 TG - Table Gain Values

Purpose: specify current machine gain and compensation configuration by setting values in the parameter table

Syntax: **TG #<register> [<value>] #<sign>##<cr> | TG <display>**

<register>

| | |
|---|---|
| B | select velocity loop (tach) compensation |
| E | select position loop (error) compensation |
| F | select feedforward gain |
| L | select analog lock gain |
| P | select position loop gain |
| T | select tach gain |
| V | select velocity loop gain |
| X | select external velocity reference gain |

<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>. The compensation values are entered as hex arguments unless in binary input communications format.

<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 values set by user in Parameter Table which defines current gains and compensators |
| ! | same as ? |

Note: When the feedforward gain is set in one velocity range, MPL will automatically calculate the gain for the other velocity ranges. When the user changes velocity ranges, the feedforward gain will be automatically changed. The TF command can be used to display the automatically calculated values.

Examples:

| | |
|---|---|
| TGP17 <cr> | Set position gain to 17. |
| TGV34 <cr> | Set velocity gain to 34. |
| TGP++V5-- <cr> | Increment position gain by 2 and decrement velocity gain by 10. |
| TGE0 <cr> | Set position loop for proportional gain only. |
| TGB4 <cr> | Set velocity loop for integral + proportional gain. |

### 7.18.3 TH - Table Hardware Configuration

The TH commands documented in this section provide a convenient method for selecting a machine hardware configuration for a specific motion control application. The Parameter Table which contains the default machine configuration is stored in non-volatile RAM and can only be changed when the protection bit **(SW)** has been set to allow non-volatile modification.

The general syntax for the **TH** command is as follows:

Syntax: **TH <parameter> <value> <cr> | TH <display>**

<parameter>

| | |
|---|---|
| A | select counts/revolution of motor |
| B | select baud rate |
| C | select communications mode |
| I | select axis ID |
| L | select enable/disable of machine limits |
| M | select enable/disable of mux'ed Input support |
| P | select enable/disable of program buffer and table values protection. When enabled, IN5' must be true to edit the program buffer or change a table value. |
| R | select motion reference configuration |
| S | Enable Software Limit Control |

More detailed syntax is provided for each of these commands below.

### 7.18.3a THA - Specify Absolute Counts/Revolution

Purpose: specify number of absolute position counts per revolution in machine parameter table. This number is used to determine the resolver resolution and to limit the value entered on the H command. This number should normally be a power of 2.

Syntax: **THA <value> <cr> | TH <display>**

<value> a number between 4096 and 65535; specifying a 0 value indicates that an encoder is being used rather than a resolver

<display>

| | |
|---|---|
| ? | display value set by user |
| ! | display value currently used |

### 7.18.3b THB - Specify Baud Rate

Purpose: specify baud rate in machine parameter table

Syntax: **THB <value> <cr> | TH <display>**

<value> a number 0 thru 8:

| | |
|---|---|
| 0 | auto baud on powerup |
| 1 | 38400 baud |
| 2 | 19200 baud |
| 3 | 9600 baud |
| 4 | 4800 baud |
| 5 | 2400 baud |
| 6 | 1200 baud |
| 7 | 600 baud |
| 8 | 300 baud |

<display>

| | |
|---|---|
| ? | display values set by user |
| ! | display values currently used (ESTOP' Powerup changes it) |

Note: The PMC's autobaud sequence automatically determines your terminal or computer's baud rate. When a character is received, it is examined by the PMC and, if it is not a valid carriage return, the PMC 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 baud rate matches the host's, communications is established. A time of 2.1 seconds is allowed for the sequence of up to 8 tries.

### 7.18.3c THC - Specify Communications Format/Enable User Units

Purpose: configure PMC communications & engineering units

Syntax: **THC <value> <cr> | TH <display>**

<value> an ASCII hex byte:

Bit 7-6 COMMUNICATIONS FORMAT, defines the representation of numeric parameters.

| Bit 7 | Bit 6 | |
|---|---|---|
| 1 | 1 | -- Binary In, Binary Out |
| 1 | 0 | -- Hex In, Binary Out |
| 0 | 1 | -- Hex In, Hex Out |
| 0 | 0 | -- Decimal In, Decimal Out |

Bit 5 Enable Fractional Distance Support (1=ON)

Bit 4 Enable Engineering Units Support (1=ON)
Engineering Units Support causes MPL to use the acceleration, velocity and position conversion factors.

Bit 3 Reserved

Bit 2 Enable Program Single Step (1=ON)
Program Single Step causes MPL to wait for the @ character to be input each time a label is reached during program execution.

Bit 1 Enable Program Trace Mode (1=ON)
Program Trace causes MPL commands to be displayed at the SCI (if active) as the program is executed.

Bit 0 Disable Echo (1=disabled)
Disable echo prevents MPL from echoing commands sent to the SCI. It does not prevent any requested display values from being sent.

<display> ? display values set by user
! display values currently used (ESTOP' Powerup changes it)

**Binary Communications Mode**

The PMC-960 supports a Binary Input mode for computer host operation. This mode is designed to reduce the amount of data transmission needed to input numeric data.

When configured for the Binary Input mode (see **THC** command) the following requirements exists:

1. All numeric values for MPL commands must be entered in binary except those for the **THC**, **TUA**, **TUP**, **TUS** commands.

2. The syntax for numeric values in binary input mode is:

<00><byte 1><byte 2> ...

A leading 0 byte, (ASCII NULL), is used to distinguish a number from a display terminator. The required number of bytes of data follow the NULL.

The amount of data is dependent on the MPL command being executed. The data will be 8, 16 or 32 bits, requiring 1, 2, or 4 bytes of data respectively. No terminator is accepted.

3. In order to send the data $1D_H$ (ASCII ctrl ]) it is necessary to send the data twice. This character is the serial bus and system status polling attention character. The 1st occurrence of the character will be taken as an attention character. The 2nd occurrence will indicate that this is valid data and not an attention character.

Example:

To set the PMC velocity to $888_D$ in a Binary Input mode you would need to send the sequence:

```
v<00><03><78>        (a total of 4 bytes of transmission)
or in ASCII characters
v<NULL><ctrl-C><x>
```

To set the index distance to 7500 use the sequence:

```
i<00><00><00><1d><1d><4c>  (total of 7 bytes sent)
   |       |      |    |
   |       |      |    +--- 00001D4C_H = 7500 decimal
   |       |      +-------- the <1d> must be repeated so
   |       |                   that it is taken as data
   |       +--------------- the 2 <00>'s are because the I
   |                           command requires 32 bits of
   |                           data
   +----------------------- leading NULL indicating binary
                               data follows
```

**Binary Output Mode**

When the PMC-960 is in a Binary Output mode most numeric data will be output in a binary format.

When in this mode if a data byte is a $23_H$ (ASCII # character) it will be output twice. This is to allow a host computer to distinguish between an error message and a data byte of $23_H$. If a single # is received then an error occurred. If a double # was received then a single byte of $23_H$ was sent.

### 7.18.3d THI - Specify PMC Axis Identifiers

Purpose: assign axis hardware identifiers

Syntax: **THI <ID> <cr> | TH <display>**

<ID> an ASCII character greater than a SPACE (20H) and less than left square bracket '['. The E Configure machine for alternate motion reference operation

Syntax: **THR <value> <cr> | TH <display>**

<value> a ASCII hex byte with the following bit definitions:

Bit 7 MASTER ENABLE causes system to become motion bus master by supplying its motion reference pulses to the Motion Reference Bus. (1=on)

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

Bit 5 Reserved

Bit 4 DIRECTION INVERT transposes the meaning of + and - in motion commands.

Bit 3-2 ACCELERATION PROFILE allows acceleration profile types to be selected by setting bits 2&3 as follows:

| Bit 3 | Bit 2 | | type |
|---|---|---|---|
| 0 | 0 | - | linear |
| 0 | 1 | - | s-curve (polynomial) |
| 1 | 0 | - | parabolic |
| 1 | 1 | - | reserved |

Bit 1-0 Reserved

<display> ? display values set by user
! display values currently used (ESTOP' Powerup changesvalue> <cr> | TH <display>

<value> an ASCII hex value:
0 disable Hardware Program Buffer Protection
1 enable Hardware Program Buffer Protection: When Hardware Program Buffer Protection is enabled, then IN5' must be true to allow the Program Buffer to be edited. The System Write Enable must also be set using the SW1 command.

<display> ? display value set by user
! display value currently used

**7.18.3h THR - Specify Motion Referencvalue> <cr> | TH <display>**

<value> an ASCII hex value:
0 disable Hardware Program Buffer Protection
1 enable Hardware Program Buffer Protection: When Hardware Program Buffer Protection is enabled, then IN5' must be true to allow the Program Buffer to be edited. The System Write Enable must also be set using the SW1 command.

<display> ? display value set by user
! display value currently used

**7.18.3h THR - Specify Motion Reference Configuration**

Purpose: Configure machine for alternate motion reference operation

Syntax: **THR <value> <cr> | TH <display>**

<value> a ASCII hex byte with the following bit definitions:

Bit 7 MASTER ENABLE causes system to become motion bus master by supplying its motion reference pulses to the Motion Reference Bus. (1=on)

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

Bit 5 Reserved

Bit 4 DIRECTION INVERT transposes the meaning of + and - in motion commands.

Bit 3-2 ACCELERATION PROFILE allows acceleration profile types to be selected by setting bits 2&3 as follows:

| Bit 3 | Bit 2 | | type |
|---|---|---|---|
| 0 | 0 | - | linear |
| 0 | 1 | - | s-curve (polynomial) |
| 1 | 0 | - | parabolic |
| 1 | 1 | - | reserved |

Bit 1-0 Reserved

<display> ? display values set by user
! display values currently used (ESTOP' Powerup changes it)

### 7.18.3g THS - Enable Software Limit Control

Purpose: enable/disable the use of the software limits defined in the SL command.

Syntax: THS <value> <cr> | TH <display>

<value> an ASCII hex value
- 0 disable - the software limit values set in the SL command are not used to effect motion
- 1 enable - the software limit values set in the SL command are used to effect motion

<display>
- ? display values set by the user
- ! display values currently used

### 7.18.4 TL - Table Label Values

Purpose: Allows user to store in the parameter table specific program <labels> to be executed on power-up, when executing a motion control routine from the machine I/O interface or from an MPL branch statement.

Syntax: **TL <vector> <label> <cr> | TL <display>**

<vector>
- B selects a specific <label> to be executed by the MPL Branch command. A vertical bar (7CH) is used with the Branch statement to specify that MPL is to select the <label> specified in the parameter table.
- P selects a specific <label> to be executed on powerup. The PMC-960 searches the program buffer for this <label> on power-up.
- V selects a specific <label> to be executed when accessing a motion control program by asserting the EXEC' input line. A high to low transition of the EXEC' digital input signal will start execution of the <label> selected using this command.

<label> a printable ASCII character representing the program buffer label to branch to when powerup or vector execute is processed.

<display>
- ? display values set by user
- ! same as ?

### 7.18.5 TU - Table Units Configuration

The **TU** command set allows the system designer to define engineering units for convenient programming of motion control applications. When Engineering Units is enabled by setting Bit 4 of the **THC** command, MPL performs its motion calculations using the specified unit conversion factors and parameters which allows the system programmer to enter input values using units of their choice. The general syntax for the **TU** command is as follows:

Syntax: **TU <parameter> <value> <cr> | TU <display>**

<parameter>
- A select acceleration units conversion factor
- B select maximum acceleration parameter
- H select home speed parameter
- P select position units conversion factor
- S select velocity (speed) units conversion factor
- V select maximum velocity parameter

More detailed syntax is provided for each of these commands below.

### 7.18.5a TUA - Specify Acceleration Units Conversion Factor

Purpose: to specify an acceleration units conversion factor which allows MPL to convert acceleration commands specified in the user's units to the PMC's internally used units

Syntax: **TUA <value> <cr> | TU <display>**

<value> a floating point value used to convert acceleration from Hz/sec to a user defined unit. The acceleration constant (A) is calculated by using the following formula:

Hz/sec / <user defined unit> = (A) Acceleration Constant

<value> is expressed as an integer greater than 1 and less than 65536 with a colon inserted to specify the location of the "decimal point" if needed. (i.e.- [5:7777] specifies the acceleration constant 5.777) The integer entered, disregarding the placement of the colon "decimal point", must be less than 65536.

<display>
- ? display value set by user
- ! display values currently used

### 7.18.5b TUB - Specify Maximum Acceleration Parameter

Purpose: to specify a maximum machine acceleration parameter (expressed in engineering units selected by using the TUA command) in the Parameter Table.

Syntax: **TUB <value> <cr> | TU <display>**

<value> an integer which specifies the maximum acceleration rate which can be set

<display> ? display values set by user
! display values currently used

### 7.18.5c TUH - Specify Home Speed Parameter

Purpose: to specify a machine home speed parameter (expressed in engineering units selected by using the **TUV** command) in the Parameter Table.

Syntax: **TUH <value> <cr> | TU <display>**

<value> an integer which specifies the velocity to use during a home command

<display> ? display values set by user
! display values currently used

### 7.18.5d TUP - Specify Position Units Conversion Factor

Purpose: to specify a position units conversion factor which allows MPL to convert position commands (in counts) to units defined by the user

Syntax: **TUP <value> <cr> | TU <display>**

<value> a floating point integer used to convert position units from counts to a user defined unit. The position constant (P) is calculated by using the following formula:

counts / <user defined unit> = (P) Position Constant

<value> is expressed as an integer greater than 1 and less than 65536 with a colon inserted to specify the location of the "decimal point" if needed. (i.e.- [5:7777] specifies the acceleration constant 5.777) The integer entered, disregarding the placement of the colon "decimal point", must be less than 65536.

<display> ? display values set by user
! display values currently used

### 7.18.5e TUS - Specify Velocity Units (Speed) Conversion Factor

Purpose: to specify a velocity units conversion factor which allows MPL to convert velocity commands (in Hz) to units defined by the user

Syntax: **TUS <value> <cr> | TU <display>**

<value> a floating point integer used to convert velocity units from Hz to a user defined unit. The velocity constant (V) is calculated by using the following formula:

Hz / <user defined unit> = (V) Velocity Constant

<value> is expressed as an integer greater than 1 and less than 65536 with a colon inserted to specify the location of the "decimal point" if needed. (i.e.- [5:7777] specifies the velocity constant 5.777) The integer entered, disregarding the placement of the colon "decimal point", must be less than 65536.

<display> ? display values set by user
! display values currently used

### 7.18.5f TUV - Specify Maximum Velocity Parameter

Purpose: to specify a maximum machine velocity parameter (expressed in engineering units selected by using the **TUV** command) in the Parameter Table.

Syntax: **TUV <value> <cr> | TU <display>**

<value> an integer which specifies the maximum velocity which can be set for any motion

<display> ? display values set by user
! display values currently used

## 7.19 U - UNTIL COMMAND

Purpose: wait until specified condition is true

Syntax: **U [<hex>[:<mask>]] <cr>**

<hex> a hexadecimal number that is matched with the values of corresponding machine input bits; the hex number defines the input condition necessary before MPL program execution will continue. A bit set to 1 defines an active signal level; a bit set to 0 defines an inactive signal level.

<mask> a hexadecimal number that specifies which machine input bits are to be compared and which are to be ignored. (Default is FFH) Mask bits that are set to 1 cause corresponding machine input bits to be compared to the <hex> parameter; Mask bits that are set to 0 cause the corresponding machine input bits to be ignored.

Note: An ESCAPE character entered during the execution of this command will end this command with a DO error.

## 7.20 V - VELOCITY COMMAND

Purpose: set or examine velocity rate

Syntax: **V <speed> <cr> | V <display>**

<speed> integer in user-defined units specifying the Index and Go velocity.

<display>

| | |
|---|---|
| ! | display current system speed |
| ? | display last entered velocity |
| %[<time>]<cr> | repeatedly display the current system speed until an SCI character is received. <time> is the rate in msec at which the % output is repeated |
| & | display velocity when last error occurred |

## 7.21 CREATING COMPLEX MOTION PROFILES WITH THE CONTOUR COMMAND

### 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 TUS and TUV commands.

### Contour Motion Command

Syntax: R <tick> <count> <cr>

where: <tick> specifies the time or distance and the direction of motion for this "tick". The units of this value are dependent on the current velocity range. The slave bit and velocity range determine the units.

| <tick> | time/tick (384 kHz) | time/tick (192 kHz) | time/tick (48 kHz) | distance/tick (external mode) |
|---|---|---|---|---|
| 1 | n/a | 1.3333 msec | 5.333 msec | 256 counts |
| 2 | 1.3333 | 2.6667 | 10.667 | 512 |
| 3 | 2.6667 | 5.3333 | 21.333 | 1024 |
| 4 | 5.3333 | 10.6667 | 42.666 | 2048 |
| 5 | 10.6667 | 21.3333 | 85.333 | 4096 |
| 6 | 21.3333 | 42.6667 | 170.667 | 8192 |
| 7 | 42.6667 | 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". The maximum 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) is entered an A4 error message will result and motion will decelerate at the **TUB** 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. In binary mode no leading NULL character is needed before the number. 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 **R** 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 using the **THC** command. Once a data mode is selected for an invocation of the **R** command it must be used until a <cr> is entered. (Translation: you can't mix hex and binary on the same line.)

Comments:

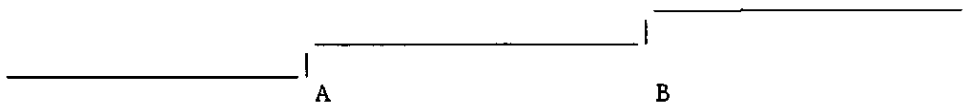
**Multiple segments**

The command syntax is structured to allow for multiple segments to be specified on each **R** 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 **R** command is ready to accept additional data. The R 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 **R** 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 $RT_1Y_1$ command. When the R is entered the PMC is ready for another segment so a prompt is sent out. Communications would look like (underlined characters are PMC output):

R}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:

R}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 **R** 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 R 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:

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

=]R]T3Y3<cr> the 1st prompt after the R is output when the position is after point B, which may not be immediately when the R 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 (R 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 |
| =]R]T1Y1]T2Y2<cr> | start motion, sensor start. Command second segment, won't wait for sensor on second segment. |
| =]SY0 | turn off sensor start |
| =]R]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 TUB parameters to the best ability of the PMC. Be forewarned that because of the units assumed on the TUB 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 R command will be terminated with an error message and deceleration will begin at the TUB 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 TUB rate. No # error message will be output when this happens.

Contour Limitations

Execution of Contour Motion requires that command data for the n+1 segment be received before the end of the $n^{th}$ segment. This data must then be processed in real time for the interrupt routines. Because of this overhead the Contour <tick> size is limited. When in a HEX input mode (requiring more processing overhead than

binary input mode) the shortest <tick> size which can be processed without a shortage of data error is a <tick> of 5.3333 msec.

Because there is less communications overhead and no conversion needed for binary input mode it is believed that a binary input mode will support faster <tick>'s, however binary input to a contour command is not possible from the Program Buffer.

## 7.22 EXCEPTION HANDLING AND ERROR CODES

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 SREADY' line, and sends a "prompt" to the SCI.

Error codes E0-EF are generated asynchronously relative to communications to the PMC-960. (Asynchronous means that no output is currently expected, usually because the PMC is at the command prompt level awaiting a new command.) When this happens it can be a problem if a computer host is communicating with the PMC-960. To relieve the host of monitoring for these errors some asynchronous error handling has been added. This additional handling is in effect when the PMC-960 is in a computer host mode. This mode is defined as having 'BINARY OUTPUT' enabled in the **THC** register.

When in this mode output of an asynchronous error code will be delayed until a prompt character, ')', is output. The delayed error code will precede the prompt. The error buffer will save the state of the PMC-960 at the time of the error, not at the time of the output of the error code. If any additional errors occur while an asynchronous error is waiting for output they will be ignored. Only the first error will be output.

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:

### 7.22.1 Cam Command Specific Errors

80 The Cam command must be at top velocity to change speed.

81 A Cam change is allowed only while Cam is running.

82 When using the Cam command, system must be at rest before direction of the system can be changed.

83 The ramp for changing velocity during the Cam command is too long for the PMC's internal calculations. Reduce the change in the velocity; or increase the acceleration value; or make the current velocity smaller.

84 A new camming <position> was requested while the system was already in motion.

85 The Cam command was given too close to the desired endpoint. The PMC does not have time to complete the command. Either enter the command earlier, increase the acceleration value, make the speed change smaller or make the endpoint later.

86 The internally calculated acceleration rate generated during the Cam command exceeds the maximum acceleration parameter. Make a

smaller change in the velocity rate or do it over a larger distance.

87 A BREAK character was received while the Cam command was waiting for the current motion to stop.

88 A BREAK character was received while the Cam command was waiting for the top velocity to be reached.

89 A BREAK character was received while the Cam command was waiting for the synchronization character (^) to be received.

8A A new cam segment is waiting to be executed. Another cam segment cannot be entered until the acceleration period begins on the current cam segment.

8B A non-zero start speed must be specified for the Cam command to operate properly.

### 7.22.2 Input Range Error Codes

90 Value entered is out of the allowable range.

91 Index distance entered is out of the allowable range.

92 Acceleration entered is out of the allowable range.

93 The absolute distance entered is out of the allowable range. Absolute distance cannot be greater than 31 bits.

94 Velocity rate entered when using C, H, J or V command is out of the allowable range. This error can occur when motion is commanded after changing velocity ranges if the stored velocity is not valid in the new velocity range.

95 Value entered is not a boolean value. A 0 or 1 is required.

96 An invalid input bank has been requested.

97 The floating point value entered is out of the allowable range.

98 The specified velocity is too small to cause motion. After engineering units conversion the velocity is below the minimum possible for the PMC and no motion would occur. Change your conversion factor or command a higher speed.

99 An overflow occurred converting position engineering units. The selected position or distance is too large.

9A Specifying an input bank is illegal when MUXED input support is disabled. (See the **THM** command.) Either enable MUXED input support and try again or try again without the bank specified.

9B Requested home location is greater than the specified resolver size using the **THA** command.

9C overflow during feedforward gain entry. Automatic calculation of gain in another velocity range has resulted in an overflow.

### 7.22.3 Syntax Error Codes

A1 An invalid command has been used.

A2 An invalid terminator or designator has been used.

A3 Reserved

A4 Reserved

A5 Invalid address entered while executing **THI** command. Valid addresses are printable ASCII characters other than the "space" character.

A6 Reserved

A7 Reserved

A8 No valid program in the optional socket.

A9 The requested machine code is not recognized.
AA Only one floating point designator (:) allowed per number.

### 7.22.4 Motion Error Codes

B1 Adaptive depth value not large enough for given motion. The SD value must be greater than the deceleration distance plus the following error. Reducing the speed or increasing either the acceleration rate or SD values will help.
B2 Command not valid while system is in motion or in an FPS (Feed to Positive Stop) state. A motion designator or programming command was entered when the system was in motion.
B3 Motion cannot be initiated with ESTOP' asserted. Unassert ESTOP' and try again.
B4 Adaptive depth control and Deceleration Stop not allowed on the same motion.
B5 An ESCAPE character was received during a synchronization command.
B6 Servodrive not enabled by the user. You must be in mode 1 or mode 2 (see SM command) before commanding motion.
B7 Reserved
B8 Attempt to move forward with forward limit (+LIMIT') asserted.
B9 Attempt to move in reverse with reverse limit (-LIMIT') asserted.

### 7.22.5 Programming Error Codes

C1 Program buffer overflow. The program buffer is full.
C2 Program label undefined. Could not find the requested label during a search (B, L, F or P commands.)
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 ZERO-POWER RAM.
C4 An ESCAPE character was received during program execution.
C5 Invalid label entered on command. A valid label can be any ASCII character between ASCII " (22H) and a lower case z (7AH) inclusive.
C6 A Program command cannot be executed during program execution.
C7 The hardware BREAK input was detected during program execution.
C8 An invalid loop counter ID was entered. Only X, Y & Z are valid.

### 7.22.6 Miscellaneous Error Codes

D0 An ESCAPE character was received during a Dwell command.
D1 Reserved
D2 Input operation aborted due to ESCAPE entered.
D3 The Parameter Table and/or program buffer is write protected, so changes cannot be made. See SW command.
D4 The velocity range requested is too large. Value entered for TUS multiplied by the value entered for TUV must be less than 384000.
D5 Break character received during Single Step wait during @ command.
D6 An ESCAPE character was received during an Until command.
D7 ESC received during contour command sync.
D8 Attempt to enter a Contour segment during a contour stop or a the contour command did not receive new data before a segment ended.

### 7.22.7 System Status Errors

| | |
|---|---|
| E0 | ESTOP' line asserted while system in motion. |
| E1 | PSJ (Position Summing Junction) overflow exists. Use an SM0 then **SM1** or **SM2** to clear the overflow condition. |
| E2 | The servodrive was not ready (DRVRDY' not true) when enabled by the user (**SM1** or **SM2** issued). |
| E3 | A FAULT input has been detected. |

# MAINTENANCE

## 8.1 DIAGNOSTICS

Each time the PMC-960 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 |

## 8.2 PREVENTIVE

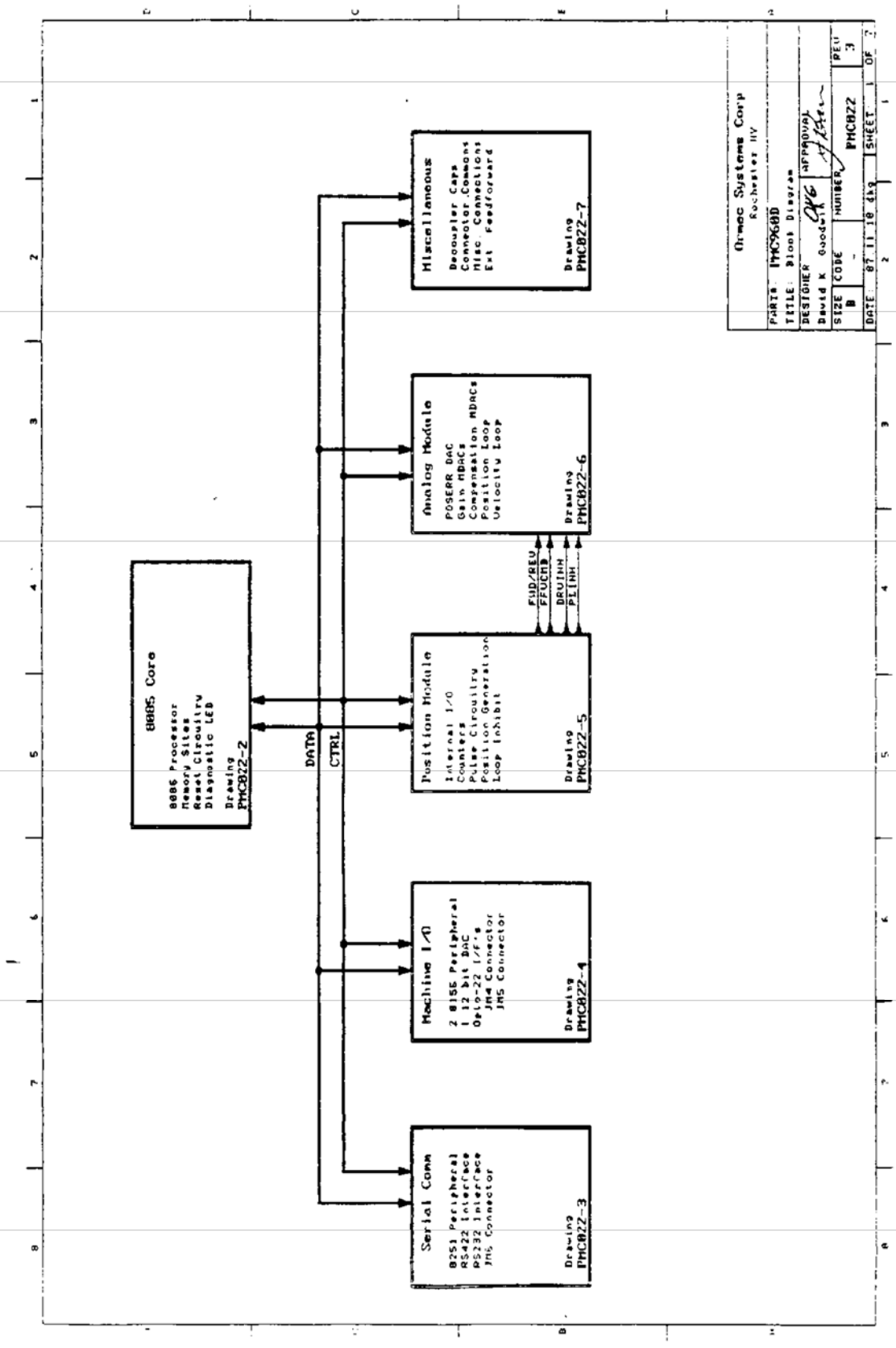
No preventive maintenance procedures are required for the PMC-960.

## 8.3 DEMAND

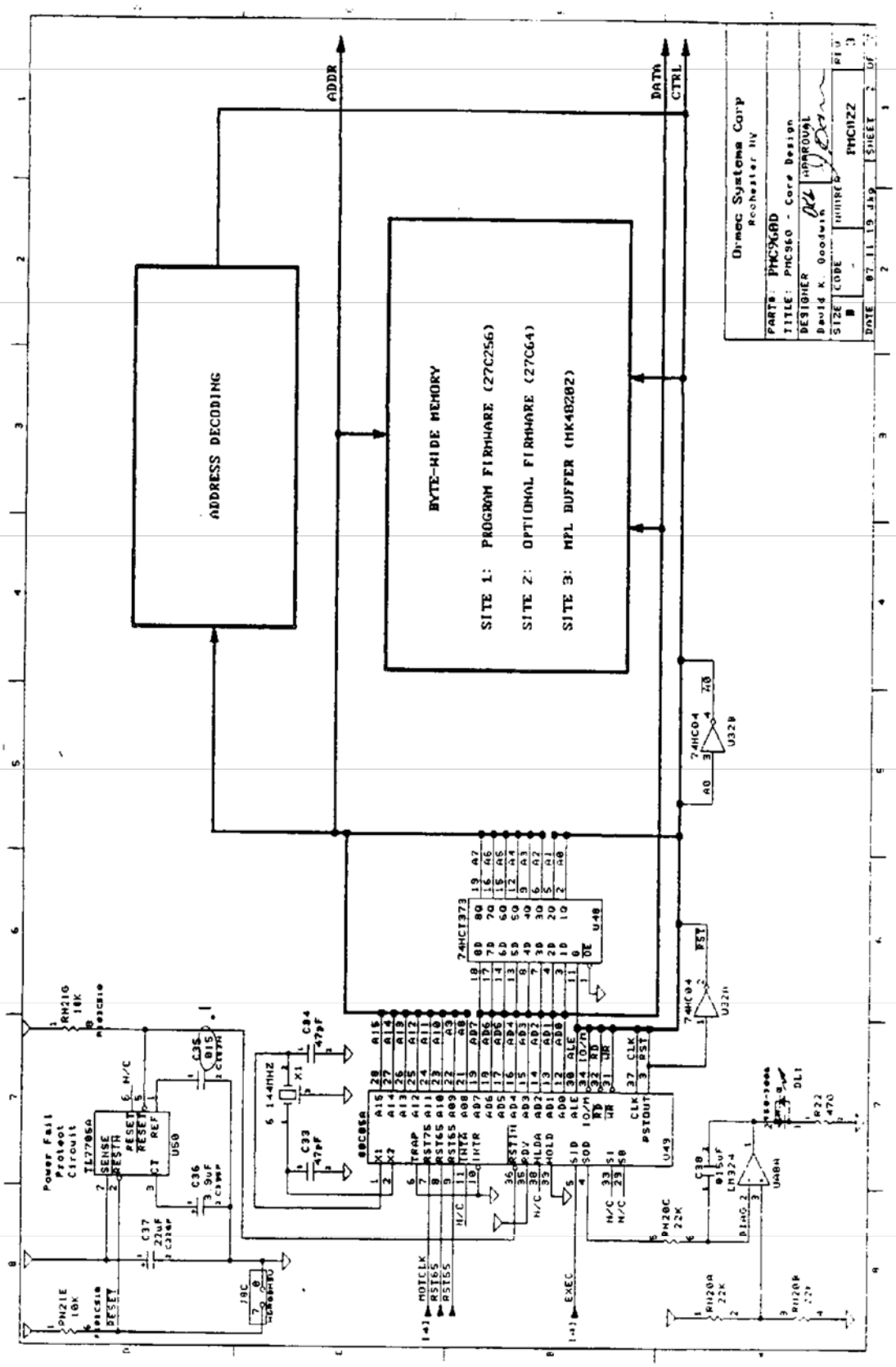
The PMC-960 is designed modularly for simple onsite demand maintenance consisting of convenient module replacement. Most of the integrated circuits used for Input/Output are socketed for easy user replacement. If a problem occurs which is beyond the socketed I/O circuitry, the user should return the defective module for factory repair. The PMC-960 is designed with connector interfaces to make board replacement in the field simple and fast.

**APPENDIX**

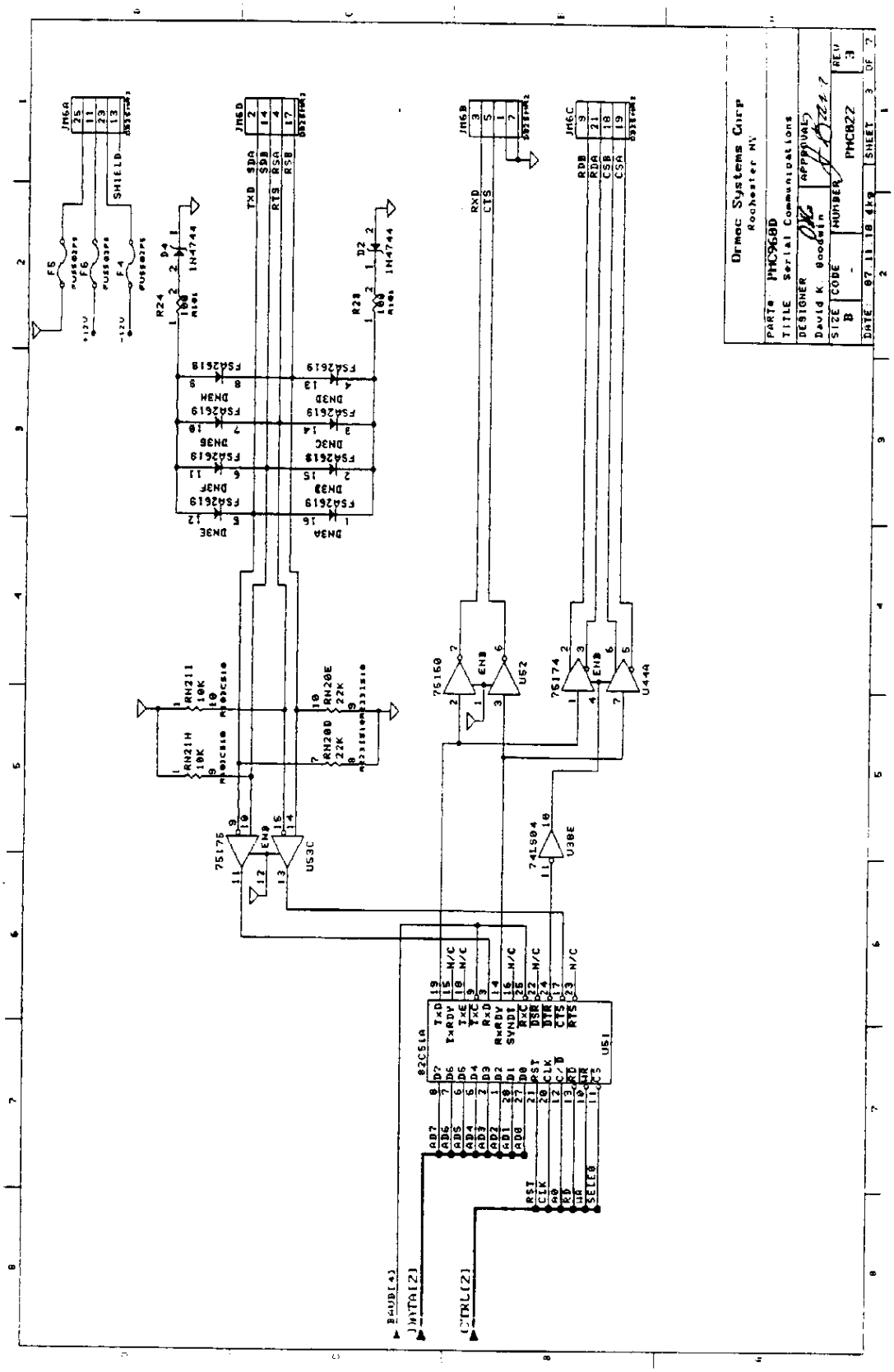
## 9.1 BLOCK DIAGRAM



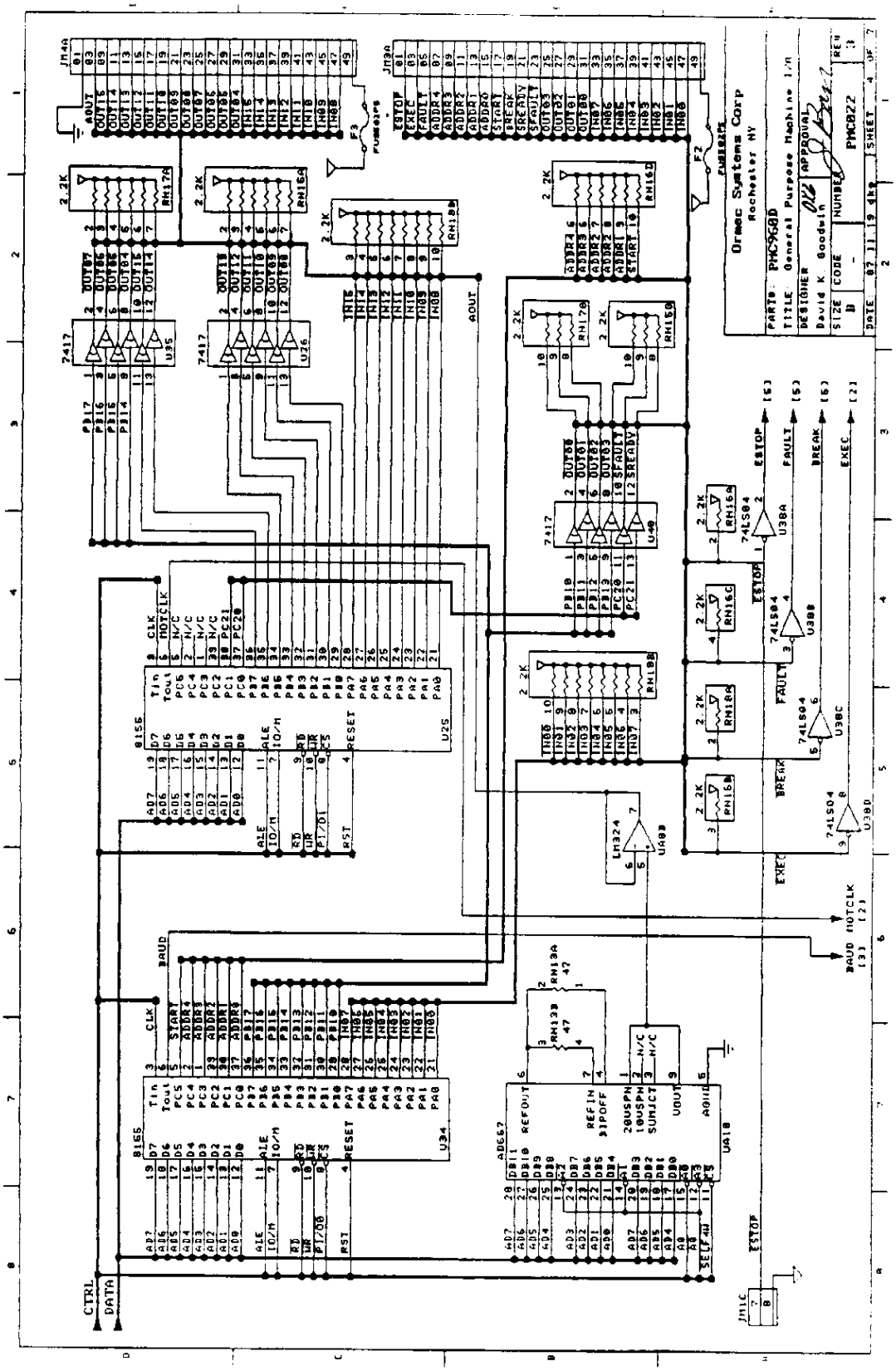
## 9.2 PMC-960 CORE DESIGN



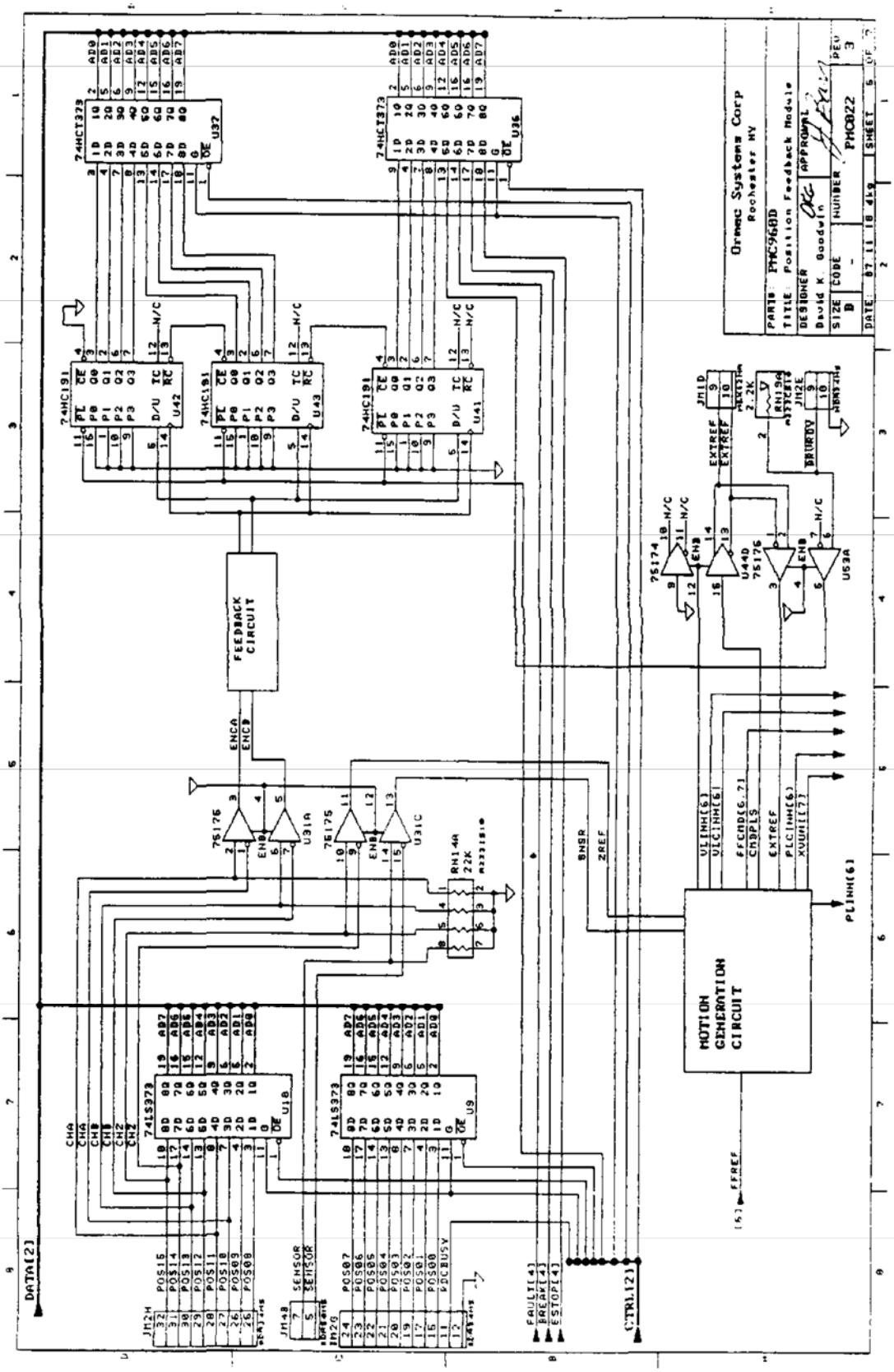
## 9.3 SERIAL COMMUNICATIONS



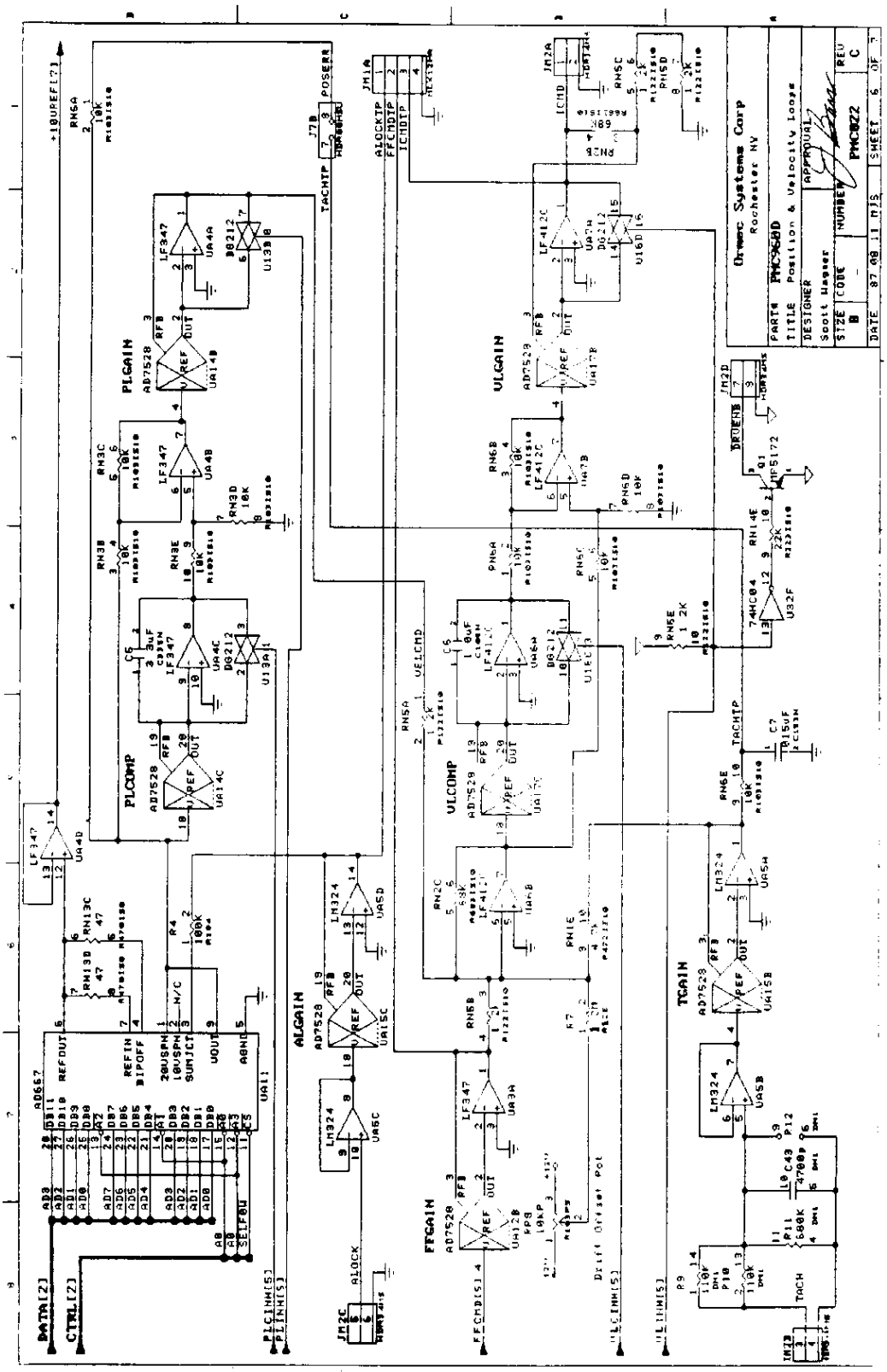
## 9.4 GENERAL PURPOSE MACHINE I/O



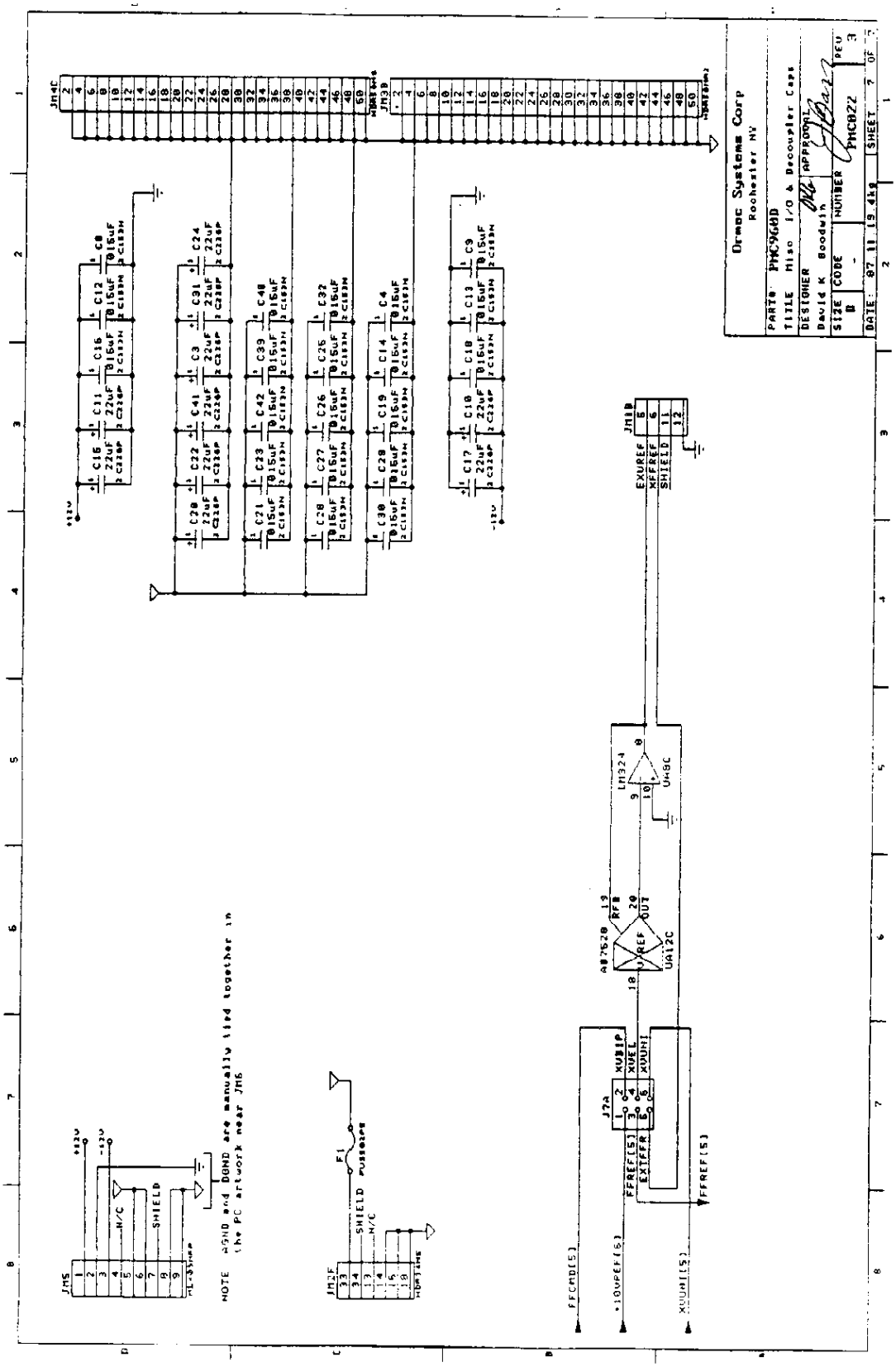
## 9.5 PULSE GENERATION / INTERNAL I/O



## 9.6 PMC-960 - POSITION & VELOCITY LOOPS



## 9.7 MISC. I/O & DECOUPLER CAPS



TECHNICAL NOTES

## 10.1 PMC SERIAL COMMUNICATIONS PROTOCOLS

PMC Communications Protocols

April 13, 1987

J. Barr & D. Goodwin

**Abstract**

This technical note describes the standards used by ORMEC's PMC product line to communicate to other devices (ordinarily host-computers) through the Serial Communications Interface (SCI). There are three levels of communications protocol discussed:

1) **Hardware Flow Control** -- Hardware signals on the SCI to control the transmission of characters through the communications channel,

2) **Software Handshake Character** -- A character (right curved bracket, }) that is used to synchronize the operation of the PMC and host device, and

3) **Attention Character** -- A character sequence starting with the <attn> character (Ctrl-], $1D_H$) to support Multiaxis operation and System Status Polling.

## Hardware Flow Control

The hardware flow control protocol used by the PMC is a modification of the RS232 standard commonly used in the industry to allow two computers to communicate at the highest data rate with bidirectional flow control. The RS232 standard made a poor assumption that only one direction of data flow, from Data Terminal Equipment (DTE) to Data Communications Equipment (DCE), needed to have flow control.

In the RS232 standard, two signals were defined to support the flow control in one direction: RTS (Request-To-Send, pin 4) is asserted TRUE by the DTE device when it has a character to send the DCE device. The DTE device must then wait for the CTS (Clear-To-Send, pin 5) signal to be asserted TRUE by the DCE equipment before it could actually send that character. The character is then sent on the TxD (Transmit Data, pin 2) signal. The RS232 standard had no provisions for controlling the speed of transmission in the reverse direction (DCE to DTE) along the RxD (Receive Data, pin 3) signal.

Because of the lack of a standard to control the speed of data on the RxD signal, the industry has had to fend for itself in developing a way to support bidirectional flow control. A variety of methods exist in the communications world which is one of the biggest reasons why it frequently takes an engineer to simply hook up a printer to a computer.

ORMEC implemented a commonly used modification to the RS232 standard[1] which redefined the use of the RTS signal but keeps the definition of CTS the same. The difference between this and the old RS232 standard is minimized to just the RTS signal and all other signals are used as interpreted in the standard. RTS becomes to the DCE device as CTS is to the DTE device and simply stated:

**No sending device should send a character as long as its respective input flow control signal is asserted FALSE.**

Some UARTs, such as Intel 8251A, implement in hardware, the flow control of the transmitted data. The implementation usually consists of holding up the transfer of data from the Transmit Buffer to the Transmit Shift Register if the CTS signal is asserted FALSE, and automatically moving the data when the CTS signal goes TRUE. Other UARTs, such as National INS8250, expect the software drivers themselves to check the incoming "Modem Status" signal(s) before sending a character to the UART. This character is unconditionally transferred from the Transmit Buffer to the Transmit Shift Register by the UART as soon as the previous character transmission is complete. Both types of UARTs are compatible with the flow control protocol used by ORMEC.

To properly manage the flow control signal used by the sending equipment, each device must implement a method to assert a FALSE condition when it is not ready to receive a character, and conversely, assert a TRUE condition when it is ready to receive a character. If the receiving device uses interrupts and a "type-ahead" ring buffer, it is simply a software task to change the level of the flow control signal based on the number of characters in the buffer. In the case of the PMC, the RxRDY signal generated by the Intel 8251A directly

---

[1] Gofton, Peter W. Mastering Serial Communications. Berkeley: Sybex, Inc., 1986.

manages the flow control signal. For more information on the timing of this hardware signal, refer to the Intel Microprocessor Peripherals Catalog.

Another way to properly manage the flow control signal using non-interrupt driven software is to assert it TRUE only when looking for a character. That is, if the application program is checking character input status and the UART does not already have a character in its receive buffer, then the application program should momentarily "strobe" the flow control signal TRUE to allow another character to be sent. On its next input status check, the application program goes through this process again, finding a character if one has been fully received, or allowing the sending equipment to send one character. This is one method used by the ORMEC ASL library on the IBM PC to communicate with PMCs.

## Software Handshake Character

The software handshake character used by PMCs is the right curved bracket (), $7D_H$) and is used as a synchronizing indicator for an operator or computer interface. The two uses for the handshake character are: 1) to indicate when the PMC is at the command level (ready to begin a new command) and 2) to indicate when a motion synchronizing request (signified by the characters ',' ';' etc.) has been satisfied.

In general, two methods are used by a host computer to achieve proper communications with PMCs:

1) both the host computer and the PMCs must prevent character overrun by using hardware flow control (see above) and,
2) whenever the host computer sends something which will generate a handshake character response from a PMC, it must wait until the handshake character has been received before sending any additional characters.

### Using Command Entry Handshakes

Most MPL commands are terminated by entering either a carriage return <cr> or a display character. (See the Operation Manual for detailed command syntax.) Once a command is terminated, the handshake character will not be sent until the execution is completed and the PMC is ready to begin a new command.

The amount of time between the termination of the command and the sending of the handshake character is command dependent and can range from less than 1 msec to many milliseconds. During this time, if another character is sent to the PMC it will remain in the hardware serial port input buffer. The hardware flow control will change to a FALSE condition indicating to the host not to send any more data. If the hardware flow control is ignored and yet another character is sent to the PMC, it will overrun the previous character in the serial port and that previous character will be lost forever. Since the lost character was probably a PMC command letter, the most likely result is a syntax error resulting from the character that is finally read in after the handshake character is sent.

The best way to avoid these character overrun problems is to implement hardware flow control and wait for the handshake character before sending another character.

Please note that the handshake is a single character (the curved bracket - }). In echo mode (THC register bit 0=0), the handshake character is preceded by 3 characters, a CR, LF, and the axis ID character. In no echo mode, only the handshake character is sent.

**Using Synchronization Character Handshake**

Whenever a synchronization character is sent by the host (assuming proper syntax) a handshake character resonse should be expected. The handshake character will be sent by the PMC when the requested condition is met. For example, the semicolon sync character (;) is a request to wait for the commanded motion to reach a constant speed. A handshake character will be returned either when the motion reaches top (or constant) velocity, or immediately, assuming that the motion is already at a constant velocity.

Consider the case of sending the sequence **"J100+;"** to a PMC. Upon receipt of the '+', the PMC will perform the necessary calculations and begin accelerating to the speed of 100. The PMC will then process the ';' causing it to wait until the velocity command reaches constant speed. When the commanded motion finally reaches the speed of 100, a handshake character (}) will be transmitted. At this point, the host computer knows that the PMC is commanding the desired speed and that it is ready to receive additional instructions from the host computer.

While the PMC is processing a synchronization character it will continue to accept serial communications. It is monitoring the input looking for an Escape ($1B_H$) character to abort the wait or an Attention (<attn>, $1D_H$) character for system status polling. All characters which are not an ESC or <attn> character will be accepted but ignored. Thus, if another command is sent to the PMC during the wait it will be lost.

Attention Character

The PMC supports an ATTENTION CHARACTER sequence to provide two services: 1) system status polling and 2) multi-axis bussed serial communication support.
System status polling syntax: <attn> <poll>
Multi-axis communications syntax: <attn> <id>

where: <attn> = ctrl-] ($1D_H$)
<poll> =lower case alphabetic character for system status polling.
<id> = upper case alphabetic character for axis ID

When requesting a system status poll, the request will be executed immediately after receiving the **<poll>**. This will delay whatever MPL processing was occurring on the PMC-960. Existing motion will continue to be serviced on an interrupt basis.

When an **"<attn><id>"** is received, each PMC will make an immediate decision to determine if the **<id>** matches it's Axis-ID as defined in the THI command. If there is a match, then this PMC will become a talker. This means that the RS-422 serial drivers are turned on and the PMC begins driving the data and flow control lines. If there is NO match, then this PMC becomes (or remains) a listener only and the serial drivers are turned (or remain) off. In listener

mode the PMC will monitor serial input but will only act on an "<attn><id>" sequence.

**Attention Character Protocol Details**

1) The serial input port is run on a polled rather than an interrupt basis and is only polled when the PMC is looking for another input character. This poll occurs between commands and during commands when additional characters or numbers are needed to define the command.

In a computer host mode, hardware flow control is necessary because the serial port may not be polled for many milliseconds. If hardware flow control is not used, characters will be missed by the PMC while it is processing a command and not polling the serial port.

Consider this example: the command **i1000+** could be followed by the system status poll request **"<attn>v"** to display current velocity. Once the **'+'** is received on the index command, the PMC firmware will begin calculating for the motion requested. During this calculation time (which can be on the order of many milliseconds), the serial port is not being polled by the PMC. Because no polling takes place during the time **<attn>** and **'v'** are sent, the **'v'** will overrun the **<attn>** character. The PMC firmware will only read a **'v'** when it gets back to poll the serial port due to the type of USART chip used on the board. In this case, a syntax error would result (**'v'** is not legal on an index command) instead of the desired system status poll.

For the previous example, the hardware flow control 1i -

| | | | |
|---|---|---|---|
| Analog Lock Gain | 0-255 | 0 | - |

## 5.8 EDITING FUNCTIONS USED DURING PROGRAM MODE

**Cursor Right** — TAB (CTRL-I) or CTRL-Y moves the cursor to the right one character at a time. Moving the cursor to the end of the line and continuing to tab will move the cursor to the beginning of the next line.

**Cursor Left** — BACKSPACE (CTRL-H) or DELETE moves the cursor to the left one character at a time. Moving the cursor to the beginning of the line and continuing to backspace will move the cursor to the beginning of the previous line.

**Cursor Down** — LINEFEED moves the cursor down a line at a time.

**Changes** — To change a motion control program, put the cursor at the point to be changed and overtype the desired information. Periods (.) may be used to overtype additional undesired characters or to reserve program buffer space for additional future commands or parameter changes.

**Insert Line** — Typing CTRL-V allows text to be inserted in the program buffer at the point of the cursor. After a CTRL-V, all characters typed are put into a 40 character RAM buffer until a second CTRL-V is typed or the RAM buffer is full. At that time, space is made in the program buffer and the characters are written to program buffer memory. If the insert operation is a result of the 40 character buffer being full, the insert operation is continued after the buffer is emptied. An ESCAPE can be used to exit from insert mode without inserting any characters.

**Kill Line** — CTRL-K del