# PRG

# POSITION REFERENCE GENERATOR

INSTALLATION AND OPERATION MANUAL
PRG001e

# TABLE OF CONTENTS


| Section | Description | Page |
|---|---|---|
| 1.0 | **GENERAL DESCRIPTION** | 3 |
| .1 | Introduction | |
| .2 | Computer Hosted Operation | |
| .3 | Programmable Controller Hosted Operation | |
| .4 | Stand Alone Operation | |
| .5 | Signal Naming Conventions | |
| 2.0 | **THEORY OF OPERATION** | 6 |
| .1 | Introduction | |
| .2 | Positioning Command Language Architecture | |
| 3.0 | **SPECIFICATIONS** | 13 |
| .1 | Electrical Specifications | |
| .2 | Mechanical and Environmental | |
| 4.0 | **INSTALLATION** | 14 |
| .1 | Mounting | |
| .2 | Electrical Installation | |
| .3 | Parallel Motion Control Channel (JM1 OR JM6) | |
| .4 | Serial Motion Control Channel (JM4) | |
| .5 | Digital Position Link Interface (JM3) | |
| .6 | Machine I/O Interface (JM2) | |
| .7 | High Speed Sensor I/O Interface (JM5) | |
| .8 | Mapping of PC Edge To D-Series Connector | |
| 5.0 | **GETTING STARTED** | 26 |
| 6.0 | **OPERATION** | 31 |
| .1 | Overview | |
| .2 | Setup Parameter Ranges | |
| .3 | Frequency Referenced Motion | |
| .4 | Distance Referenced Motion | |
| .5 | PRF1x1c Indexer Firmware Command Description | |
| .6 | I/O Condition Table | |
| .7 | Exception Handling and Error Codes | |
| .8 | Status Registers | |
| 7.0 | **MAINTENANCE** | 50 |
| .1 | Preventive | |
| .2 | Demand | |
| 8.0 | **APPENDIX** | |
| .1 | PRG System Interface Diagram | |
| .2 | PRG PC Board Layout | |
| .3 | PRG Component Layout | |
| .4 | PRG Serial MCC Interface | |
| .5 | PRG Machine I/O Interface | |
| .6 | PRG Parallel MCC Interface | |
| .7 | PRG HSS and DPL Interface | |
| .8 | Sales/Service Policy | |

# GENERAL DESCRIPTION

## 1.1 INTRODUCTION

The ORMEC **Position Reference Generator (PRG)** is a microcomputer based product which facilitates the design of high performance position control applications. A typical servo positioning system utilizing a PRG is diagrammed in Appendix 8.1. The PRG operates in conjunction with the ORMEC **Motor Loop Controller (MLC).**

In combination with a servodrive, a servomotor, a DC tachometer and an incremental position encoder, the MLC is used to create a closed loop digital position servo. The resulting position control system works much like a stepping motor/translator system in that it converts TTL level command pulses into incremental movement. For a more detailed discussion of the MLC consult the ORMEC Motor Loop Controller Manual (MLC001).

Since the speed and accuracy of an MLC based positioning system are one or two orders of magnitude greater than a typical stepping motor application, creating positioning information to drive it at its full capability is a significant real time problem. The PRG solves that problem by translating straightforward ASCII commands into positioning pulse information with frequencies of up to 192 kHz. This servo "position reference" information is transmitted to the MLC through a differential interface called the **Digital Position Link (DPL).**

The PRG is available as a standard product in three versions. Information in this manual applies to all three versions except when otherwise noted. Note particularly that all information relating to programming does not apply to the PRG-900, since it does not have the programming option.

PRG-900 - standard indexing firmware and no programming

PRG-901 - standard indexing firmware with programming and 2k bytes EEPROM non-volatile memory

PRG-902 - standard indexing firmware with programming and 2k bytes RAM memory

The PRG-901 was designed to be operated in three configurations:

**Computer Hosted** - interfaced through either the serial or the parallel **Motion Control Channel (MCC)**

**Programmable Controller Hosted** - interfaced through the **Machine I/O (MIO)**

**Stand Alone** - interfaced through the MIO

## 1.2 COMPUTER HOSTED OPERATION

A PRG based positioning system can be used as a slave serving a host computer through either a parallel or a serial interface called the **Motion Control Channel (MCC).** The PRG plugs directly into a MULTIBUS (registered trademark of Intel Corporation) card cage but also has connectors for use with other systems through a parallel I/O interface or a serial interface. When plugged into a MULTIBUS system, the PRG is mapped to two adjacent locations in I/O space. For details see the OPERATION Section.

Communications between a host computer and the PRG are essentially identical to communications with a USART serial device. The host computer sends a byte (character) of information to the PRG, and then must wait for that byte to be processed by the PRG. The completion of processing of each character is indicated by a bit in the status register for parallel mode, or by the hardware handshake line in serial mode.

When a command is completed by sending a "command terminator", the PRG will then execute that particular command. Upon finishing the command, the PRG will send a "greater than" (>) signal to the host to indicate that it is "READY" for a new command.

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

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

## 1.3 PROGRAMMABLE CONTROLLER HOSTED OPERATION

A PRG based system can perform multiple complex motion control functions under the direction of a Programmable Controller using a simple interface. This is accomplished using a PRG-901 as follows:

- The user programs motion control functions using any ASCII terminal device, giving them single character ASCII names
- The programmable controller selects the appropriate function by placing the ASCII address in parallel at the Machine I/O (MIO) Interface and asserting the EXECUTE' input.
- The programmable controller can tell when the function is complete by observing the READY signal from the MIO.
- The function can be interrupted at any time by asserting the STOP' input at the MIO.

## 1.4 STAND ALONE OPERATION

Since the PRG-901 is equipped with its own Machine I/O and non-volatile positioning command language program memory, it can operate in a stand alone mode, controlling a small machine or machine module directly from its positioning command language program. Included in the 24 I/O signals of the MIO are four general purpose inputs and four general purpose outputs, which can be used for switches or machine sensors and outputs such as indicators, or solenoids. The MIO is flat cable compatible with industry standard Opto 22 type input/output modules allowing flexible interface to high voltage AC or DC drivers and receivers.

A detailed description of the Machine I/O Interface is in SECTION 4.6.

## 1.5 SIGNAL NAMING CONVENTIONS

Throughout this manual, references to inverted logical signals will use the convention of following the signal name with an apostrophe ('). e.g. INPUT' The drawings in the APPENDIXES will use the "overbar" notation. Moreover, 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. The signal RESET is expected to reset the MLC when it is "true" or "asserted" (at a +5 VDC level).

Conversely, signals with apostrophes following them are considered logically "true", or "asserted" when they are "low", and logically "false" when they are high. e.g. The signal RESET' should be at 0 VDC when it is desired to reset the MLC.

# THEORY OF OPERATION

## 2.1 INTRODUCTION

The PRG operates as an intelligent slave motion control system, receiving high level commands via the Motion Control Channel (MCC) and in turn providing position reference information to the servo system (MLC) through the Digital Position Link (DPL) at connector JM3.

The MCC physically corresponds to connector JM1 or JM6 for parallel operation or connector JM4 for serial operation. When running a program, the PRG receives its commands from the program buffer instead of from the MCC. The host may interrupt a program at any time, however, by transmitting a character to the PRG.

The PRG positioning command language is architected to provide a logical, consistent and easy to use set of commands which can be used to specify high performance motion. The result is a calculator like language allowing the application designer a great deal of freedom and power with which to solve unique application needs.

The approach taken is to utilize a set of single character ASCII commands followed by an optional argument (number) and one or more single character ASCII terminators. This syntax is as follows:

<command> [<argument>] #<terminator>#

The square brackets ([]) around the argument indicate that the argument is optional. The pound signs (#) around the terminator indicate that multiple terminators are allowed on some of the commands.

This approach allows convenient communications with virtually any commercially available ASCII terminal or computer system. Moreover, the brevity of the language makes it both easy to use and requires a minimum of communications overhead when interfaced to a host computer.

Like a programmable calculator, or a computer running interactive BASIC, commands may be executed in the interactive mode or used in a "program".

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 only a few milliseconds maximum.

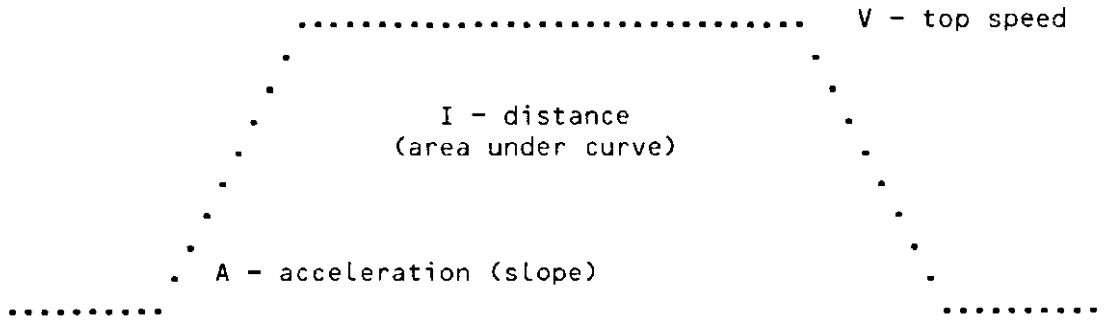
In addition, the PRG is architected to be able to service the real time requirements of commanding a motion while simultaneously running this interpretive language. This feature allows the PRG to perform necessary calculations to set up future motions while simultaneously performing positioning tasks.

## 2.2 POSITIONING COMMAND LANGUAGE ARCHITECTURE

The architecture of the PRG positioning command language includes three major groups of commands and terminators. These groups deal with MOTION, INTERFACE and PROGRAMMING.

### 2.2.1 MOTION Commands and Terminators

The PRG creates motion profiles where the velocity is essentially trapezoidal as illustrated below. For the motion to be generated, the parameters of **distance, velocity** and **acceleration** must be specified. Since the PRG deals with digital positioning servo systems, the parameters will be specified as integer numbers referencing distance as counts, velocity as a frequency (Hz) and acceleration as a frequency change per second (Hz per millisecond or kHz per second). Each of these parameters is expressed as an integer value and has an allowable range, as well as a default value. Refer to SECTIONS 6.2 and 6.3 for more detail regarding the parameter ranges. These parameters are stored in the **motion buffer**, where they will be retrieved each time a positioning command occurs.

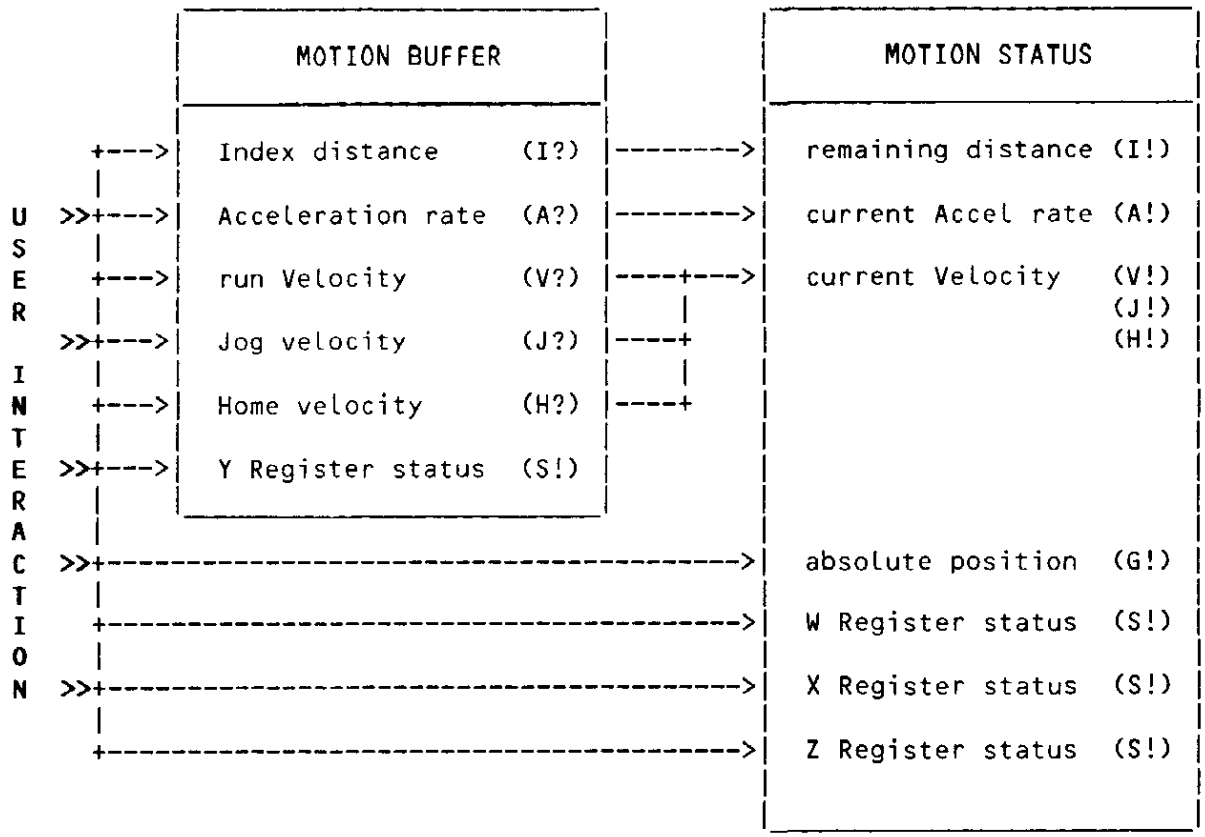



**Typical PRG Motion Profile**

The PRG also keeps track of positioning system location to a range of approximately ± one billion counts, and is capable of returning that information to the host or moving to absolute locations within that range.

In addition to storing parameters for distance, top speed and acceleration rate, the PRG stores a home speed, a jog speed, and configuration information which is contained in four 8 bit registers, identified as Status Registers W,X,Y and Z. The status information is stored in the form of bit switches that select various motion and communications options. The X Register also contains information as to whether the servodrive is on or off, and therefore can be used by the host system to tell if a fault has occurred.

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




The PRG can parameterize the next motion profile while a motion is being commanded because the user interacts with the MOTION BUFFER for setting up the parameters for the next move. At the time each motion is commanded, the PRG takes the pertinent information from the MOTION BUFFER for the move.

MOTION Commands and Terminators can be further classified as those which only set up **Parameters** for motion control, and those which also deal with causing **Action.** These two sets of commands are outlined below.

**MOTION**

PARAMETERS

Commands

| | |
|---|---|
| Velocity | - set or examine maximum velocity rate |
| Acceleration | - set or examine acceleration rate |
| Status (W,Y) | - set or examine motion status |

Terminators

<cr> (carriage return) - enter new argument

The following examples assume that the PRG is in its powerup (default) mode.

Examples

| | |
|---|---|
| V300<cr> | - set the maximum velocity to 30.0 kHz |
| A3000<cr> | - set the acceleration rate to 3000 Hz/millisecond |
| SW03 | - set the W register to $03_H$ (hexadecimal) which transposes the meaning of + and - direction and also reduces the top velocity to 48.00 kHz, increasing the resolution of the acceleration and velocity specifications |
| SY40 | - set the Y register to $40_H$ which selects the s-curve acceleration type |

With acceleration and velocity specified by the above parameter commands, the distance may be specified by the relative Index or the absolute Go command. The Jog and Home commands have indeterminate distances which will be determined by future asynchronous events. i.e. Home will be terminated by an encoder reference signal or a high speed sensor signal, and Jog will be ended by a future command to stop. These commands may be used to start ACTION as well as specify parameters.

**MOTION**

ACTION

Commands

| | |
|---|---|
| Index | - move relative to the current position |
| Go | - move to the absolute position |
| Jog | - move at the jog rate |
| Home | - move until encoder reference or sensor |

Terminators

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

Examples

| | |
|---|---|
| I500+ | - move 500 counts positive relative to the current position The motion actually starts when the + is received. |
| I- | - move negative from the current position by the distance currently stored in the motion buffer |
| G9306-<cr> | - go to absolute location -9306 |
| I* | - stop current motion |
| J95+ | - jog at 9.5 kHz in the positive direction |
| H1- | - home at 0.1 kHz in the negative direction |
| J- | - jog in the negative direction at the rate specified in the motion buffer |

### 2.2.2 INTERFACE Commands and Terminators

INTERFACE commands and terminators can be further classified as dealing with **Communications** or **Synchronization** with real time events.

**INTERFACE**

**COMMUNICATIONS**

Commands

| | |
|---|---|
| **N**ormalize | - select comm mode, reset, set position |
| **O**utput | - set Machine I/O output condition |
| **S**tatus (X,Z) | - set or examine control/communications status |

Terminators

| | |
|---|---|
| **?** (question mark) | - report motion buffer value |
| **!** (exclamation mark) | - report motion status value |

Examples

| | |
|---|---|
| N* | - Normalize the PRG (restarts the firmware) |
| N0- | - Normalize the absolute position counter to -0 |
| N3795+ | - Normalize the absolute position counter to +3795 |
| N<cr> | - Normalize the MCC communications |
| OA | - set the general purpose machine outputs to (HHHL) |
| SX8 | - set the lower three bits of the X Register to 000 |

**INTERFACE**

**SYNCH**

Commands

| | |
|---|---|
| **D**elay | - delay specified time interval |
| **U**ntil | - wait until specified condition is true |

Terminators

| | |
|---|---|
| , (comma) | - wait for motion to stop |
| ; (semi-colon) | - wait for motion to reach steady state non-zero speed |

Examples

| | |
|---|---|
| D1000 | - delay 1000 milliseconds (1 sec) |
| D, | - delay until motion is complete |
| UA | - wait until condition A is true |
| D; | - delay until steady accel is complete |

### 2.2.3 PROGRAM Commands and Terminators

PROGRAM commands and terminators can be further classified as dealing with manipulating the program **Buffer** or used for **Control.**

| PROGRAM | | |
|---|---|---|
| BUFFER | Commands | |
| | Program | - enter, edit, or examine motion programs |
| | @<label> | - mark program label |
| | <esc> (escape) | - exit program mode |
| | Quit | - terminate program space & exit program mode |

Examples

P<cr> - enter programming mode with the cursor at the end of program space

PA - enter programming mode with the cursor at the beginning of the "A routine"

P! - enter programming mode with the cursor at the beginning of program space

P? - examine program space (from the beginning, a line at a time) changes are not allowed in this mode

@@ Startup - mark the beginning of the "@ routine" The @ routine is a special user writeable routine which executes whenever the power is applied, the system is RESET, or the N* command is executed. The word "Startup" is a user definable comment following the program marker label @.

| PROGRAM | | |
|---|---|---|
| | Commands | |
| CONTROL | Branch | - conditional jump to program label |
| | Loop | - repeat program segment specified number of times |
| | Function | - conditional call to a subroutine |
| | Exit | - conditional return from subroutine or program |
| | Terminators | |
| | <cond> | - branch condition (@ through 0) |

Examples

| | |
|---|---|
| BA | - branch unconditionally to the "A routine" |
| BBA | - branch to the B routine on condition A |
| LA100 | - loop back to the A label 100 times |
| FB | - call function B; Program control returns to the point where the function was called at the completion of the function. |
| E | - mark the end of a function |
| ED | - end the function on condition D |

The , (comma) and ; (semi-colon) terminators, covered in the INTERFACE group are extremely important for use in synchronizing motion control programs with high performance motion. They allow the user to set up new moves which synchronize with moves which are underway, as well as coordinate Machine Outputs with the end of motion, or the beginning of a new constant speed.

When the PRG encounters either the , in a motion control program, it causes the program to wait for the motion to end. If it finds a ;, the program will wait for a new constant speed to begin.

When you start writing PRG positioning command language programs, it is important to remember that these programs are designed to operate in "real time". The PRG, when it powers up, defaults to the ECHO communications mode (controlled by Status Register Z) where it echos all characters of a PRG program to the MCC, if it is active. This feature, while useful for debugging PRG programs, slows down the operation of the PRG depending on the baud rate of the programming terminal (or the speed that the host computer receives characters from the MCC). To operate a program from the MCC, while not slowing down PRG operation, use the **SZ1** command. For details, see SECTION 6.5.

### 2.2.4 Error Codes

The PRG positioning command language checks for errors at execution time, and when one is found, an error message is generated.

**ERROR CODES**

Format: # <error code> <bell>

Error codes

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

# SPECIFICATIONS

## 3.1 ELECTRICAL SPECIFICATIONS

| | | |
|---|---|---|
| Power requirements | | +5VDC, 1.6A max<br>±12VDC, 0.1A (for RS-232 drivers) |
| CPU | type | 8085A |
| | speed | 3.072 MHz |
| Program storage | | up to 16k bytes RAM or 2k bytes EEPROM (optional) |

All signals found at interfaces are assumed to be TTL level digital signals unless otherwise noted. Schematics of the interfaces are found in the APPENDIXES.

Parallel interface — conforms to Intel MULTIBUS specification (Intel part #9800683) for an intelligent slave; all signals TTL compatible

MULTIBUS access — I/O mapped to two adjacent I/O locations for 8 bit operation (16 bit CPU I/O compatibility optional) The IORC' and IOWC' control signals must be low for less than 25 usec.

Serial interface — conforms to either EIA RS-232 or RS-422/449 with autobaud for several standard rates between 19.2k and 300 Baud; standard strapping is for Data Communications Equipment (DCE) i.e. a standard RS-232 terminal will communicate with this interface

DPL interface — uses RS-422 differential line drivers and receivers and is compatible with all the ORMEC Motor Loop Controllers

High Speed Sensor Interface

all HSS signals must have durations of more than 2.4 usec

optocoupler specifications

| | | |
|---|---|---|
| input impedance | - | 220 ohm typical |
| recommended input voltage | - | 0 - 5 volts |
| min assertion voltage | - | 2.8 volts |
| | - | 2.4 volts typical |
| min assertion current | - | 8.0 mA |
| | - | 5.5 mA typical |
| maximum input current | - | 25.0 mA |

## 3.2 MECHANICAL AND ENVIRONMENTAL SPECIFICATIONS

| | | |
|---|---|---|
| Max dimensions | | 12" x 6.75" x 0.8" |
| Max weight | | one pound |
| Temperature range | Operating | 0 to +70 degrees C |
| | Storage | -25 to +125 degrees C |
| Relative humidity | (w/o condensation) | 0 to 90% |

# INSTALLATION

## 4.1 MOUNTING

The PRG can be custom mounted in an enclosure of the user's design or plugged into a MULTIBUS card cage. Mounting holes are provided at the corners of the printed circuit board as shown in APPENDIX 8.2.

## 4.2 ELECTRICAL INSTALLATION

A typical PRG configuration is diagrammed in APPENDIX 8.3. The signals terminated on each connector are described in the following interface sections.

The +5 VDC power is supplied through the JM1 connector. For pin assignments see SECTION 4.3. The ±12 VDC power is supplied through either the JM1 connector or the JM4 connector. For pin assignments see SECTION 4.3 (JM1) or SECTION 4.4 (JM4).

## 4.3 PARALLEL MOTION CONTROL CHANNEL (JM1 OR JM6)

The parallel motion control channel follows the information transfer protocol of the MULTIBUS. This protocol is asynchronous and is implemented using the following MULTIBUS signals.

```
ADDRESS LINES  0 through 7   (select two adjacent locations)
DATA LINES     0 through 7
INTERRUPTS     0 through 7   (select one)
IORC
IOWC
XACK
```

This protocol is suitable for parallel communications with other computer systems as well by using a parallel I/O interface. For detailed information on this subject refer to the Intel publication "Intel MULTIBUS Specification" (Intel P/N 9800683).

For user convenience, these signals are also brought out to JM6, which is a 26 pin PC edge connector suitable for mass cable termination. The pinout is designed to be compatible with the MOSTEK STD Bus Parallel I/O (PIO) interface board, to which ORMEC has implemented a hardware and software interface.

Signals and pin assignments for both JM1 and JM6, as well as a 25 pin D-Series which mates with JM6 are listed in the following tables. A schematic for this interface is in APPENDIX 8.6.

## PARALLEL MOTION CONTROL CHANNEL

| MULTIBUS MNEMONIC | JM1 | JM6 | D SERIES CONNECTOR for JM6 |
|---|---|---|---|
| INIT' | 14 | NC | NC |
| IORC' | 21 | 13,21 | 7,11 |
| IOWC' | 22 | 3,14 | 2,20 |
| XACK' | 23 | 25 | 13 |
| ADR0' | 57 | 20 | 23 |
| ADR1' | 58 | 4 | 15 |
| ADR2' | 55 | 15 | 8 |
| ADR3' | 56 | 16 | 21 |
| ADR4' | 53 | 17 | 9 |
| ADR5' | 54 | 18 | 22 |
| ADR6' | 51 | 19 | 10 |
| ADR7' | 52 | 22 | 24 |
| ADR8' | 49 | NC | NC |
| ADR9' | 50 | NC | NC |
| ADRA' | 47 | NC | NC |
| ADRB' | 48 | NC | NC |
| DAT0' | 73 | 12 | 19 |
| DAT1' | 74 | 11 | 6 |
| DAT2' | 71 | 10 | 18 |
| DAT3' | 72 | 9 | 5 |
| DAT4' | 69 | 8 | 17 |
| DAT5' | 70 | 7 | 4 |
| DAT6' | 67 | 6 | 16 |
| DAT7' | 68 | 5 | 3 |
| GND | 1,2,11,12,75, 76,85,86 | 2,24 | 14,25 |
| +5VDC | 3,4,5,6,81, 82,83,84 | NC | NC |
| +12VDC | 7,8 | NC | NC |
| -12VDC | 79,80 | NC | NC |

| MULTIBUS MNEMONIC | JM1 | JM6 | J8 STRAPPING | D SERIES CONN for JM6 |
|---|---|---|---|---|
| INT0' | 41 | 26 | 49-50 | NC |
| INT1' | 42 | 26 | 51-52 | NC |
| INT2' | 39 | 26 | 53-54 | NC |
| INT3' | 40 | 26 | 55-56 | NC |
| INT4' | 37 | 26 | 57-58 | NC |
| INT5' | 38 | 26 | 59-60 | NC |
| INT6' | 35 | 26 | 61-62 | NC |
| INT7' | 36 | 26 | 63-64 | NC |

Since a single PRG only requires two address locations, multiple units can be connected together on the same parallel channel. Each PRG must have a unique address which can be selected from the following table:

**PARALLEL MCC ADDRESSING TABLE**

| MULTIBUS DATA | MULTIBUS STATUS | J8 STRAPPING | | MULTIBUS DATA | MULTIBUS STATUS | J8 STRAPPING | |
|---|---|---|---|---|---|---|---|
| $02_H$ | $03_H$ | 47-48 | 31-32 | $82_H$ | $83_H$ | 47-48 | 23-24 |
| $06_H$ | $07_H$ | 45-46 | 31-32 | $86_H$ | $87_H$ | 45-46 | 23-24 |
| $0A_H$ | $0B_H$ | 43-44 | 31-32 | $8A_H$ | $8B_H$ | 43-44 | 23-24 |
| $0E_H$ | $0F_H$ | 41-42 | 31-32 | $8E_H$ | $8F_H$ | 41-42 | 23-24 |
| $12_H$ | $13_H$ | 39-40 | 31-32 | $92_H$ | $93_H$ | 39-40 | 23-24 |
| $16_H$ | $17_H$ | 37-38 | 31-32 | $96_H$ | $97_H$ | 37-38 | 23-24 |
| $1A_H$ | $1B_H$ | 35-36 | 31-32 | $9A_H$ | $9B_H$ | 35-36 | 23-24 |
| $1E_H$ | $1F_H$ | 33-34 | 31-32 | $9E_H$ | $9F_H$ | 33-34 | 23-24 |
| $22_H$ | $23_H$ | 47-48 | 29-30 | $A2_H$ | $A3_H$ | 47-48 | 21-22 |
| $26_H$ | $27_H$ | 45-46 | 29-30 | $A6_H$ | $A7_H$ | 45-46 | 21-22 |
| $2A_H$ | $2B_H$ | 43-44 | 29-30 | $AA_H$ | $AB_H$ | 43-44 | 21-22 |
| $2E_H$ | $2F_H$ | 41-42 | 29-30 | $AE_H$ | $AF_H$ | 41-42 | 21-22 |
| $32_H$ | $33_H$ | 39-40 | 29-30 | $B2_H$ | $B3_H$ | 39-40 | 21-22 |
| $36_H$ | $37_H$ | 37-38 | 29-30 | $B6_H$ | $B7_H$ | 37-38 | 21-22 |
| $3A_H$ | $3B_H$ | 35-36 | 29-30 | $BA_H$ | $BB_H$ | 35-36 | 21-22 |
| $3E_H$ | $3F_H$ | 33-34 | 29-30 | $BE_H$ | $BF_H$ | 33-34 | 21-22 |
| $42_H$ | $43_H$ | 47-48 | 27-28 | $C2_H$ | $C3_H$ | 47-48 | 19-20 |
| $46_H$ | $47_H$ | 45-46 | 27-28 | $C6_H$ | $C7_H$ | 45-46 | 19-20 |
| $4A_H$ | $4B_H$ | 43-44 | 27-28 | $CA_H$ | $CB_H$ | 43-44 | 19-20 |
| $4E_H$ | $4F_H$ | 41-42 | 27-28 | $CE_H$ | $CF_H$ | 41-42 | 19-20 |
| $52_H$ | $53_H$ | 39-40 | 27-28 | $D2_H$ | $D3_H$ | 39-40 | 19-20 |
| $56_H$ | $57_H$ | 37-38 | 27-28 | $D6_H$ | $D7_H$ | 37-38 | 19-20 |
| $5A_H$ | $5B_H$ | 35-36 | 27-28 | $DA_H$ | $DB_H$ | 35-36 | 19-20 |
| $5E_H$ | $5F_H$ | 33-34 | 27-28 | $DE_H$ | $DF_H$ | 33-34 | 19-20 |
| $62_H$ | $63_H$ | 47-48 | 25-26 | $E2_H$ | $E3_H$ | 47-48 | 17-18 |
| $66_H$ | $67_H$ | 45-46 | 25-26 | $E6_H$ | $E7_H$ | 45-46 | 17-18 |
| $6A_H$ | $6B_H$ | 43-44 | 25-26 | $EA_H$ | $EB_H$ | 43-44 | 17-18 |
| $6E_H$ | $6F_H$ | 41-42 | 25-26 | $EE_H$ | $EF_H$ | 41-42 | 17-18 |
| $72_H$ | $73_H$ | 39-40 | 25-26 | $F2_H$ | $F3_H$ | 39-40 | 17-18 |
| $76_H$ | $77_H$ | 37-38 | 25-26 | $F6_H$ | $F7_H$ | 37-38 | 17-18 |
| $7A_H$ | $7B_H$ | 35-36 | 25-26 | $FA_H$ | $FB_H$ | 35-36 | 17-18 |
| $7E_H$ | $7F_H$ | 33-34 | 25-26 | $FE_H$ | $FF_H$ | 33-34 | 17-18 |

Note: For eight bit addressing, (factory standard) strap J8 (15-17) otherwise U47 (INTEL D3205) must be inserted and the upper four bit addresses (ADRB - ADR8) are selected as follows:

| High Order Address (ADRB'-ADR8') | J8 STRAPPING |
|---|---|
| don't care | 15-17 |
| $0_H$ | 15-16 |
| $1_H$ | 13-14 |
| $2_H$ | 11-12 |
| $3_H$ | 9-10 |
| $4_H$ | 7- 8 |
| $5_H$ | 5- 6 |
| $6_H$ | 3- 4 |
| $7_H$ | 1- 2 |

The DATA address is used to write or read a data byte to/from the PRG (MCC channel). Writing to the STATUS address does a hardware reset of the PRG board. Reading the STATUS address will return a byte with the lower four bits as follows:

bit 0 - PRG (MCC Channel) input buffer full'
bit 1 - PRG (MCC Channel) output buffer empty'
bit 2 - MOTION' (same as JM2-13)
bit 3 - READY' (same as JM2-11)

The PRG (MCC channel) output buffer full signal can be strapped to trigger an interrupt as shown above. Therefore, the PRG can interrupt the host computer when it has an output character, rather than the host continually polling the PRG status.

## 4.4 SERIAL MOTION CONTROL CHANNEL (JM4)

A 3M or equivalent 26 pin female mass termination PC edge connector with .1 inch finger spacing will mate with connector JM4. The connector strapping configurations and pin assignments are shown in APPENDIX 8.4. As can be seen in the APPENDIX, this interface is configurable with EIA Standards RS-232, RS-422/449, and RS-423/449. Further it can be strapped to be either a DCE or a DTE device for both RS-232 and RS-449. As it is shipped from the factory, it is strapped for RS-232 DCE with no handshake, to work with a standard terminal.

Note that pin 26 is unused since this connector is compatible with 25 pin "D-Series" connectors. Also note that the pin numbers shown below correspond to the numbering system used for connector JM4 and will be different for D-Series connectors (RS-232). SECTION 4.8 has a mapping of 26 pin edge connector to 25 pin D-Series connector numbering.

A Serial Motion Control Channel wiring diagram and strapping configuration for header connector J7 is found in APPENDIX 8.4.

## 4.5 DIGITAL POSITION LINK INTERFACE (JM3)

The DIGITAL POSITION LINK INTERFACE is the interface with the ORMEC Motor Loop Controller, and it is designed to utilize a mass termination connector. A 3M or equivalent 26 pin female mass termination PC edge connector with .1 inch finger spacing will mate with connector JM3. All descriptions of the DPL use the 25 pin "D-Series" connector pin assignments, and therefore the following table references these pin numbers rather than the PC edge connector numbers. Refer to Section 4.8 for a mapping of 26 pin edge connector to 25 pin "D-Series" connector numbering. Note that PC edge connector pin 26 is unused to maintain compatibility with a 25 pin D-Series connector. A schematic for this interface is in APPENDIX 8.7.

| Signal Name | "D-Series" Pin # | Description |
|---|---|---|
| COMMON | 2,15 | power and logic signal common |
| FORWARD COMMAND<br>FORWARD COMMAND' | 24<br>11 | Each high transition of FWDCMD causes the load to move forward one encoder distance unit. (digital output) |
| REVERSE COMMAND<br>REVERSE COMMAND' | 25<br>12 | Each high transition of RVSCMD causes the load to move backward one encoder distance unit. (digital output) |
| RESET<br>RESET' | 23<br>10 | A high state of RESET zeroes the error fault detection circuit and holds the digital error summer contents at zero. (digital output) |
| POSITION LOOP DISABLE<br>POSITION LOOP DISABLE' | 17<br>4 | A high state of PLDIS disables the position loop while allowing the system to operate in a velocity controlled mode. (digital output) |
| DRIVE DISABLE<br>DRIVE DISABLE' | 18<br>5 | A high state of DRVDIS will disable the buffered DRVON output which is used to operate the "loop contactor" and or the power to the servodrive. In addition, DRVDIS reduces the loop gains in both the position and the velocity loops to zero by shorting the outputs of the respective summing operational amplifiers to their respective summing junctions. It also resets the integrators in the respective integral + proportional compensators if used. (digital output) |
| DRIVE OFF<br>DRIVE OFF' | 9<br>22 | A high state of DRVOFF indicates that a fault has been detected or DRVDIS is high. (digital input) |
| ENCODER REFERENCE<br>ENCODER REFERENCE' | 6<br>19 | ENCREF is the unprocessed output from the digital position encoder reference output. (if implemented) |

## 4.6 MACHINE I/O INTERFACE (JM2)

The MACHINE I/O INTERFACE is provided to allow the PRG to interface directly with the outside world. Standard uses of this interface, as can be seen from the table below, are to handle limit switches, a hardware "STOP/BREAK" command, and to provide user information as to whether the servomotor is in motion and whether or not the PRG is ready to accept a new command. In addition, user programmable inputs and outputs are provided. The MOTION ROUTINE ADDRESS inputs allow interface with equipment such as Programmable Controllers, by allowing them to access previously stored Motion Control Programs using standard I/O points.

A 3M or equivalent 50 pin female mass termination PC edge connector with .1 inch finger spacing will mate with JM2. It provides standard user inputs and outputs which are OPTO/22 compatible. A schematic for this interface is in APPENDIX 8.5.

### MACHINE I/O INTERFACE (JM2)

| I/O # | JM2 Pin# | | DRIVER | STANDARD | PROGRAM Option | |
|---|---|---|---|---|---|---|
| | 49 | | | +5 VDC | +5 VDC | |
| 0 | 47 | | | reserved | IN0' | input nibble |
| 1 | 45 | | | reserved | IN1' | |
| 2 | 43 | I | - | reserved | IN2' | |
| 3 | 41 | N | | reserved | IN3' | |
| | | P | | | | |
| 4 | 39 | U | | reserved | EXECUTE' | |
| 5 | 37 | T | - | -LIMIT' | -LIMIT' | |
| 6 | 35 | S | | +LIMIT' | +LIMIT' | |
| 7 | 33 | | | STOP' | STOP' | |
| 8 | 31 | | | FORWARD' | FORWARD' | |
| 9 | 29 | 0 | U3 | ADECEL' | ADECEL' | |
| 10 | 27 | U | | MOTION' | MOTION' | |
| 11 | 25 | T | | READY' | READY' | |
| | | P | | | | |
| 12 | 23 | U | | OUT0' | OUT0' | output nibble |
| 13 | 21 | T | U4 | OUT1' nibble | OUT1' | |
| 14 | 19 | S | | OUT2' output | OUT2' | |
| 15 | 17 | | | OUT3' | OUT3' | |
| 16 | 15 | | | reserved | ADR0' | motion routine address |
| 17 | 13 | | U5 | reserved | ADR1' | |
| 18 | 11 | I | | reserved | ADR2' | |
| 19 | 9 | N | | reserved | ADR3' | |
| | | P | | | | |
| 20 | 7 | U | | reserved | ADR4' | |
| 21 | 5 | T | | reserved | ADR5' | |
| 22 | 3 | S | U6 | reserved | ADR6' | |
| 23 | 1 | | | reserved | ADR7' | |

All even pin numbers are grounded.

## MACHINE I/O INTERFACE

| Signal Name | Description |
|---|---|
| bits OUT3' - OUT0' | output nibble set by the Out command; refer to I/O Condition Table in SECTION 6.6 (digital output) |
| READY' | a low state of READY' indicates that the PRG is ready for the next command (digital output) |
| MOTION' | a low state of MOTION' indicates the system is in motion (digital output) |
| ADECEL' | a low state of ADECEL' indicates that the system is accelerating or decelerating; a high state indicates that the system is at top velocity or at rest (digital output) |
| FORWARD' | a low state of FORWARD' indicates that the system is moving in the forward direction; a high state indicates that the system is moving in the reverse direction (digital output) |
| STOP' | a low active input will stop the system motion and cause the PRG to return to the READY state i.e. abort any other activity taking place and return to the interactive command level (digital input, 4 milliseconds minimum) |
| +LIMIT'<br>-LIMIT' | low active inputs that will stop the system motion; only a motion command in the opposite direction will cause system motion when either '+' or '-' LIMIT' is asserted (provided for limit safety switches on X-Y tables) (digital inputs, 4 milliseconds minimum) |
| EXECUTE' | {programming option} a high to low transition will start execution of the pre-programmed routines addressed by the Motion Routine Address (bits ADR7' - ADR0') (digital input, 1 millisecond minimum) |
| bits ADR7' - ADR0' | {programming option} Motion Routine Address bits which address the pre-programmed subroutine in the program buffer upon an asserted EXECUTE' signal; these eight address bits correspond to those of the program marker byte (digital inputs) |
| bits IN3' - IN0' | {programming option} Input Condition bits used by the conditional Branch, Exit, and Function commands ; refer to the I/O Condition Table in SECTION 6.6 (digital inputs) |

The output conditions shown in the I/O CONDITION TABLE (SECTION 6.6) require that an inverting driver is inserted in socket U4. A 74LS00 driver is supplied with the board although the following chart indicates other pin compatible drivers:

| TTL DRIVER | OUTPUT BUF/INV | PULLUP TYPE | CURRENT Sink/Source (mA) Stnd | LS | S | O/C VOLTAGE |
|---|---|---|---|---|---|---|
| 74x00 | INV | Active | 16/0.4 | 8/0.4 | 20/1.0 | - |
| 74x03 | INV | O/C | 16/ - | 8/ - | 20/ - | 5V |
| 74x08 | BUF | Active | 16/0.8 | 8/0.4 | 20/1.0 | - |
| 74x09 | BUF | O/C | 16/ - | 8/ - | 20/ - | 5V |
| | | | | | | |
| 74x26 | INV | O/C | 16/ - | 8/ - | | 15V |
| 74x32 | BUF | Active | 16/0.8 | 8/0.8 | 20/1.0 | - |
| 74x37 | INV | Active | 48/1.2 | 24/1.2 | 60/3.0 | - |
| 74x38 | INV | O/C | 48/ - | 24/ - | 60/ - | 5V |

INV - Inverter BUF - Buffer O/C - Open Collector

## Hardware Motion Address Configurations

The following two tables can be used to select 32 or fewer motion control routines from hardware signals on the MIO. The first table demonstrates the signals required at connector JM2 for programs labeled @ through _ including the alphabet (in capitals). The second set of tables demonstrate a method for selecting programs labeled 0-9 with a standard thumbwheel switch.

## Sample Hardware Motion Routine Addresses

If 4, 8, 16 or 32 motion control routines are sufficient for the application, then 2, 3, 4 or 5 Motion Routine Address lines respectively are connected as shown below. The unused lines on the left side of the table are left unconnected if designated High (pullup resistors assert a high level) or grounded if designated Low.

| Program Label | Motion Routine Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{ADR7}$ | $\overline{ADR6}$ | $\overline{ADR5}$ | $\overline{ADR4}$ | $\overline{ADR3}$ | $\overline{ADR2}$ | $\overline{ADR1}$ | $\overline{ADR0}$ |
| JM2- | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 |
| @ ($40_H$) | H | L | H | H | H | H | H | H |
| A ($41_H$) | H | L | H | H | H | H | H | L |
| B ($42_H$) | H | L | H | H | H | H | L | H |
| C ($43_H$) | H | L | H | H | H | H | L | L |
| D ($44_H$) | H | L | H | H | H | L | H | H |
| E ($45_H$) | H | L | H | H | H | L | H | L |
| F ($46_H$) | H | L | H | H | H | L | L | H |
| G ($47_H$) | H | L | H | H | H | L | L | L |
| H ($48_H$) | H | L | H | H | L | H | H | H |
| I ($49_H$) | H | L | H | H | L | H | H | L |
| J ($4A_H$) | H | L | H | H | L | H | L | H |
| K ($4B_H$) | H | L | H | H | L | H | L | L |
| L ($4C_H$) | H | L | H | H | L | L | H | H |
| M ($4D_H$) | H | L | H | H | L | L | H | L |
| N ($4E_H$) | H | L | H | H | L | L | L | H |
| O ($4F_H$) | H | L | H | H | L | L | L | L |
| P ($50_H$) | H | L | H | L | H | H | H | H |
| Q ($51_H$) | H | L | H | L | H | H | H | L |
| R ($52_H$) | H | L | H | L | H | H | L | H |
| S ($53_H$) | H | L | H | L | H | H | L | L |
| T ($54_H$) | H | L | H | L | H | L | H | H |
| U ($55_H$) | H | L | H | L | H | L | H | L |
| V ($56_H$) | H | L | H | L | H | L | L | H |
| W ($57_H$) | H | L | H | L | H | L | L | L |
| X ($58_H$) | H | L | H | L | L | H | H | H |
| Y ($59_H$) | H | L | H | L | L | H | H | L |
| Z ($5A_H$) | H | L | H | L | L | H | L | H |
| [ ($5B_H$) | H | L | H | L | L | H | L | L |
| \ ($5C_H$) | H | L | H | L | L | L | H | H |
| ] ($5D_H$) | H | L | H | L | L | L | H | L |
| ^ ($5E_H$) | H | L | H | L | L | L | L | H |
| _ ($5F_H$) | H | L | H | L | L | L | L | L |

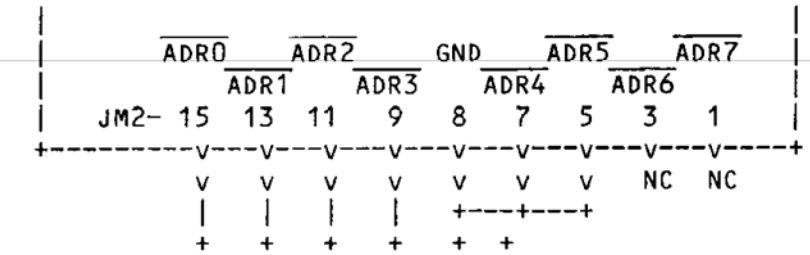
(H)igh TTL level (L)ow TTL level

## BCD Thumbwheel Switch Configuration

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

| Program Label | Motion Routine Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{ADR7}$ | $\overline{ADR6}$ | $\overline{ADR5}$ | $\overline{ADR4}$ | $\overline{ADR3}$ | $\overline{ADR2}$ | $\overline{ADR1}$ | $\overline{ADR0}$ |
| JM2- | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 |
| 0 ($30_H$) | H | H | L | L | H | H | H | H |
| 1 ($31_H$) | H | H | L | L | H | H | H | L |
| 2 ($32_H$) | H | H | L | L | H | H | L | H |
| 3 ($33_H$) | H | H | L | L | H | H | L | L |
| 4 ($34_H$) | H | H | L | L | H | L | H | H |
| 5 ($35_H$) | H | H | L | L | H | L | H | L |
| 6 ($36_H$) | H | H | L | L | H | L | L | H |
| 7 ($37_H$) | H | H | L | L | H | L | L | L |
| 8 ($38_H$) | H | H | L | L | L | H | H | H |
| 9 ($39_H$) | H | H | L | L | L | H | H | L |

(H)igh TTL level (L)ow TTL level




| WHEEL | 1 | 2 | 4 | 8 | C |
|---|---|---|---|---|---|
| 0 | | | | | x |
| 1 | x | | | | x |
| 2 | | x | | | x |
| 3 | x | x | | | x |
| 4 | | | x | | x |
| 5 | x | | x | | x |
| 6 | | x | x | | x |
| 7 | x | x | x | | x |
| 8 | | | | x | x |
| 9 | x | | | x | x |

x - indicates signal connected to Common

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

## 4.7 HIGH SPEED SENSOR I/O INTERFACE (JM5)

The high speed sensor I/O interface allows optical isolation or use of RS-422 differential line drivers and receivers for high speed sensors used for the coordination of motion with respect to external events, as well as coordination of multiple PRG's in a single application. The circuitry and strapping for this interface is found in APPENDIX 8.7.

The two optically coupled isolators have high bandwidth to allow much faster input than could be achieved through OPTO 22 compatible modules. In addition they are tied directly into the pulse generating hardware of the PRG, rather than going through the overhead of the CPU, for accurate control of external event driven functions such as "SENSOR Accelerate", "SENSOR Decelerate" or "SENSOR Stop". See the Y-Register bits of the Status Command in the OPERATION Section.

In addition to the SENSOR' input, there is also an External Reference input (EXTREF'), allowing the PRG to have motion that is referenced to the motion of another system, rather than to the internal crystal controlled clock. Another advanced capability associated with the HIGH SPEED SENSOR INTERFACE is the ability for a PRG to place its output pulses (independent of system direction) on a Distance Referenced pulse bus (RS-422 compatible) which is on pins 7 and 8. For a description of the X-Register bits of the Status Command, see the OPERATION Section, and for a discussion of distance referenced motion, see SECTION 6.4.

RS-422 compatible differential line receivers (U13) are also available to be used to interface to the SENSOR' and/or the EXTREF' inputs, if desired.

## 4.8 MAPPING OF PC EDGE CONNECTOR (26 pin) TO D-SERIES (25 pin)

25 pin D-Series connectors are compatible with 26 pin PC Edge Card connectors with the exception that their pin numbering systems are different. To help eliminate confusion on this issue, these mappings between the two connector types are included in this manual.

| PC Edge Conn | 25 pin D-series |
|---|---|
| 1 | 1 |
| 2 | 14 |
| 3 | 2 |
| 4 | 15 |
| 5 | 3 |
| 6 | 16 |
| 7 | 4 |
| 8 | 17 |
| 9 | 5 |
| 10 | 18 |
| 11 | 6 |
| 12 | 19 |
| 13 | 7 |
| 14 | 20 |
| 15 | 8 |
| 16 | 21 |
| 17 | 9 |
| 18 | 22 |
| 19 | 10 |
| 20 | 23 |
| 21 | 11 |
| 22 | 24 |
| 23 | 12 |
| 24 | 25 |
| 25 | 13 |
| 26 | NC |

| 25 pin D-series | PC Edge Conn |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 5 |
| 4 | 7 |
| 5 | 9 |
| 6 | 11 |
| 7 | 13 |
| 8 | 15 |
| 9 | 17 |
| 10 | 19 |
| 11 | 21 |
| 12 | 23 |
| 13 | 25 |
| 14 | 2 |
| 15 | 4 |
| 16 | 6 |
| 17 | 8 |
| 18 | 10 |
| 19 | 12 |
| 20 | 14 |
| 21 | 16 |
| 22 | 18 |
| 23 | 20 |
| 24 | 22 |
| 25 | 24 |
| NC | 26 |

## GETTING STARTED

No adjustment and setup are required with the Position Reference Generator other than the strapping options described in the INSTALLATION Section.

Whatever your planned interface to the PRG, it is recommended that for getting started, you interface an ASCII terminal to connector JM4 to learn the PRG Positioning Command Language. The PRG comes from the factory with this connector configured as an RS-232C DCE with no handshake, and it therefore should be compatible with most ASCII RS-232C terminals. See APPENDIX 8.4 for alternate strapping configurations of header connector J7.

After your MLC is installed according to the procedure in its manual, you should attach the DPL cable between the MLC and the PRG connector JM3. Also attach your ASCII terminal to connector JM4, and provide power to the system. The terminal should be configured for one of the following baud rates: 19.2k, 9600, 4800, 2400, 1200, 600 or 300.

When power is applied to the PRG, the on-board microprocessor searches through the program space to attempt to find a startup routine. The startup routine is a user defined positioning command language program using the program label @ (ASCII $40_H$). In addition, the PRG continually checks back and forth between the parallel and the serial interfaces to see if either interface is attempting to get control. If a character is received at either interface, the PRG examines this character to determine if it is a carriage return (ASCII $0D_H$). From the parallel interface, control is gained by simply sending the PRG a carriage return.

Gaining control from the serial interface is more complicated in that there are several acceptable terminal baud rates. The PRG will automatically determine the baud rate by starting with its serial interface configured for 19.2 k baud, and if a character is received which is not a valid carriage return, the PRG will halve its baud rate to 9600. When the next character is received, it is examined by the PRG and if it is not a valid carriage return, the PRG will again halve its baud rate. This is repeated up to seven times allowing baud rates from 19.2k to 300. If a valid carriage return is received, control is given to the serial interface. Normally, the PRG is in a fast scan loop alternately checking the parallel interface and the serial interface for inputs, however once any character is received at the serial interface, 1.7 seconds are allowed for the sequence of up to 7 tries before the PRG again checks the parallel interface.

Therefore, to initiate communications with the PRG, you must provide information for the PRG to determine the terminal baud rate. This is done by repeatedly pressing the carriage return <cr> enough times for the PRG to recognize it within a 1.7 second time period.

When control is gained, the PRG will respond with the code for its firmware, followed by a "prompt" of =>. The rest of this section assumes that the PRG is using the standard programming firmware (PRF121c or PRF131c). At this point, the PRG is operating in its interactive mode, and is ready to accept commands. For a review of the command architecture, consult SECTION 2.2, and for a detailed and precise description of the commands, consult SECTION 6.5.

Assuming no startup program was encountered in the user programmable non-volatile memory, setup parameters in the motion buffer are now at their default values. For a list of the default values, see SECTION 6.2. To check the values of these parameters try the sequences found below. In the sequences listed, **bold** print indicates the sequence that you type, and the regular print is the information sent by the PRG.

PRF131c

=>**I?** 0500 - index distance is set to 500 counts

=>**V?** 0400 - top velocity is set to 40.0 kHz

=>**A?** 4000 - acceleration rate is set to 4000 Hz/millisecond This information tells you that if you execute an I+ command, the PRG will send a pulse train of 500 counts to the MLC which ramps up to 40.0 kHz at a rate of 4000 Hz/millisecond, and then ramps down from 40.0 kHz at 4000 Hz/millisecond. Altogether this index lasts for about 23 milliseconds.

=>**I1<cr>** 0440 #B4 - attempting to set an index distance of 1 count results in a B4 error because the move distance is too short for the specified top speed and acceleration rate

=>**I?** 0500 - the index distance in the buffer is still set to 500 counts because of the error on the attempt to change it

=>**I+** - the motor should index 500 counts in the positive direction; You will notice that the PRG did not send another prompt, because you are still in the I command. Typing another + will result in another positive move of 500 counts, and typing - will result in a negative move of 500 counts. The I command can be terminated with a carriage return <cr>.

Let's reset everything back to its default setting and examine the rest of the default parameters. To do this, terminate the I command with a <cr>, and then type **N***.

=>**N*** - normalizes the PRG

=>**H?** 20 - Home speed is set to 2.0 kHz

=>**J?** 0100 - Jog speed is set to 10.0 kHz

=>**G?** 00 - the absolute position counter is set to 0

Let's try the jog command:

=>**J-** - the system should now be "jogging" at a speed of 10.0 kHz; Note that the PRG did not send another prompt because you are still in the J command. Type any character at this point to stop the servomotor. You are still in the J command, but now the only characters that can be typed without soliciting an error from the PRG are the "+" "-" "," and <cr>. The + will cause motion to start again in the positive direction,

at which time any character will again stop the motion. The - will cause motion to start again in the negative direction, at which time any character will again stop the motion. The , will cause a ">" character to be returned from the PRG because the system is not in motion at present, and the <cr> will terminate the jog command.

Try the sequence below and observe the results.

**=>J--++,>----<cr>**

Try some other jog sequences using the above terminators.

You may note that the effect of the , terminator is to return you to the beginning of the J command (as though you had just typed J) and typing <cr> immediately thereafter will result in a #A2 error. The , and the ; terminators can be used by a host computer to determine when motion is complete or a new constant speed is reached. They are also useful in a motion control program for synchronizing the program with motion which is underway. For more detail on this, see SECTION 2.2.

Some other interesting things can be done with the J command, by returning to the command mode while the servomotor is still "jogging". The definition of jogging is just "running until told to stop" and it can be done at any speed that the system is capable of running.

Let's try this sequence:

| | |
|---|---|
| **=>N*** | - normalize the system |
| **=>J+<cr>** | - start the system jogging and return to command mode |
| **=>J5+<cr>** | - change the jog speed to .5 kHz and jog at that speed |
| **=>J50<cr>** | - change the motion buffer jog speed to 5.0 kHz |
| **=>J?** 50 | - examine the motion buffer jog speed |
| **=>J!** 05 | - the actual jog speed is still .5 kHz at this point |
| **=>J-** | - jog at 5.0 kHz in the negative direction |
| **=>A1** | - change the acceleration rate to 1 kHz/sec |
| **=>J100-** | - jog at 10.0 kHz in the negative direction; It will take about 5 seconds to reach this speed because of the very low acceleration rate. |
| **=>J*** | - stop the motion |
| **=>J+** | - jog at the last speed in the positive direction |
| **=>N*** | - normalize the system (stops fast since A=4000) |

Let's examine the state of the Status Registers.

**=>S!** 000F0000 - note that to examine the Status Registers, the ! was required This is because the status registers, other than the Y Register, are not in the MOTION BUFFER but are an inquiry into the current status of the PRG. The first two characters indicate the status of the W Register (in ASCII hexadecimal), the next two the X Register, followed by the Y and Z Registers.

For a detailed discussion of the Status Registers, see SECTION 6.8.
Try some other Status Register commands.

| | |
|---|---|
| **=>SXE** | disable the position loop; At this point you can turn the shaft of the servomotor a few degrees, since the position loop is not active. |
| **=>SXF** | enable the position loop; The shaft should spring back! |
| **=>SY40** | enable the s-curve acceleration; If you use an oscilloscope to observe the tach signal, you will see that the acceleration is no longer linear. |
| **=>N-** | normalize the absolute position counter to -0 |
| **=>I2000-<cr>** | index 2000 counts in the - direction |
| **=>I----<cr>** | index -2000 counts four more times; Don't try to initiate a new index while the system is still moving, or you'll get an error message! |
| **=>G!**-010000 | your system is now at -10,000 counts; You did five -2000 count indexes, remember? |
| **=>G65+** | go to absolute location +65 |

## PROGRAMMING

Now lets try writing a motion control program.

| | |
|---|---|
| **=>P!**<br>Q | go into program mode and put the cursor at the beginning of programming space; If your program buffer is clear, your PRG should have typed a Q as shown. If there is something other than a Q there, then someone has been programming your EEPROM! You can review what they put there by typing successive linefeeds (usually LINEFEED or LF on most ASCII terminals). Typing linefeeds if there is a Q there will have no effect, since that Q marks the end of program space. |

Assuming that a Q was there, let's proceed.

| | |
|---|---|
| **@A TEST PROGRAM<cr>**<br>**A4000.**<br>**V100.**<br>**I500..-**<br>**D300,**<br>**I+**<br>**D300,**<br>**BA**<br>**Q** | you overtype the Q with @A, which labels routine A; The rest of the line until the <cr> is a comment. |
| **=>BA<cr>** | branch unconditionally to routine A; Your system should now be indexing back and forth 500 counts. |

Note also the 300 millisecond delay in between indexes.

It is good programming style to set the acceleration rate and top velocity so that no matter what they might be when this routine starts, it will still execute predictably. It is possible to set them in another routine first however, and then call this routine from the other.

The periods are used to reserve space to increase the size of those parameters at a future date by editing the routine.

Let's edit the program to increase the delay to 600 msec.

**=>PA**

| | |
|---|---|
| @A TEST PROGRAM | type linefeed <lf> and the cursor will move down a line |
| A4000. | type <lf> |
| V100. | type <lf> |
| I500..- | type <lf> |
| D300, | type <tab>; The cursor is now on the 3 and so you can overtype it with a 6. You can also back up a character by typing a backspace <bs> or a delete <del>. If you back up beyond the first character in a line the cursor will move back to the first character in the previous line, and the PRG will type that line. In this way, you can move the cursor around program space until you find the area you want to overtype, and then modify it. When fininshed editing, type the escape <esc> key, and you will return to the interactive mode, signified by the => "READY" signal. |

To run the modified program, type **BA<cr>** again.

You now have an introduction to the PRG's capabilities, and to learn more, study the positioning command language architecture in SECTION 2.2 or the commands in the OPERATION Section.

ORMEC has a design aid available, called the **Machine Interface Simulator (MIS)**, which will facilitate learning about the possible uses of the Machine I/O.

# OPERATION

## 6.1 OVERVIEW

The PRG is controlled or programmed by sending commands through the Motion Control Channel (MCC). These commands specify control parameters, request display of information, or effect system motion. Each command is started with a single ASCII character indicating the command type. The command then contains an optional argument, and one or more terminators. The command language architecture is described in SECTION 2.2 and a full command description is found in SECTION 6.5.

The PRG will send an ASCII "greater-than" (>) character whenever it is READY for another command. The status of DRVOFF' is indicated by the top bit of the READY character. Therefore the READY character is $03E_H$ if the drive is off and $0BE_H$ otherwise (normal operation). This provides a convenient way for a host computer to check for a fault at the Motor Loop Controller (MLC).

A "number sign" (#) character ($023_H$) followed by a two character error code will be sent to the MCC whenever an error is detected. Refer to SECTION 6.7 for definitions of these error codes.

The following symbols are used in the remainder of this section:

| | |
|---|---|
| **<token>** | = variable item |
| [ ] | = optional enclosed item |
| # # | = repeatable enclosed item |
| { } | = programming option enclosed item |
| \| | = OR operator |

The general format of a command is:

<command> [<argument>] #<terminator>#

where: <command> - single alphabetic character
<argument> - alphanumeric
<terminator> - non alphanumeric

This **command** character can be upper case e.g. $A(41_H)$ or lower case e.g. $a(61_H)$. The top bit is set to indicate that a binary argument is to follow. However, binary arguments can only be used if they have been enabled by the Z Register, bit 2 (BINMCC). For more detail on binary MCC communications, read the Status command description in SECTION 6.5. The **argument** is often optional and necessary only when required by the syntax of the command. Multiple **terminators** are possible to modify the current command where appropriate.

**Description of terminators/designators:**

| | | | |
|---|---|---|---|
| **<cr>** | carriage return | ($0D_H$) | end command argument(s) |
| * | asterisk | ($2A_H$) | stop current motion |
| ? | question mark | ($3F_H$) | display selected motion buffer parameter |
| ! | exclamation mark | ($21_H$) | display selected motion status parameter |
| + | plus sign | ($2B_H$) | start motion in positive direction |
| - | minus sign | ($2D_H$) | start motion in negative direction |
| , | comma | ($2C_H$) | wait for the system motion to stop before accepting the next character; The prompt character (>) is displayed indicating that the system is at rest. Any MCC character will terminate the wait for the system to stop with a B5 error. |
| . | period | ($2E_H$) | null character which is ignored on input; This character is normally used to reserve space in the positioning command language programming space for changing the value of a parameter over its range. |

## 6.2 SETUP PARAMETER RANGES

| Parameter | Token | Units | Range | Default |
|---|---|---|---|---|
| **Acceleration rate** | <ramp> | | | |
| Internal mode | | | | |
| ALTVEL = 0 | | kHz/sec | 1 to 65,535 | 4000 |
| ALTVEL = 1 | | 100 Hz/sec | 1 to 65,535 | - |
| External mode | | 100 counts | 0 to 65,534 | - |
| **Run, Jog, Home speed** | <rate> | | | |
| Internal mode | | | | |
| ALTVEL = 0 | | 100 Hz | 1 to 1920 | 400 |
| ALTVEL = 1 | | 10 Hz | 1 to 4800 | - |
| External mode | | .1 % | 1 to 1000 | - |
| **Relative distance** | <rpos> | counts | ± 2,147,483,647 | 500 |
| **Absolute distance** | <apos> | counts | ± 1,073,741,823 | 0 |
| **Delay time** | <time> | msec | 1 to 65,535 | - |
| **I/O Condition** | <cond> | - | @($40_H$) to O(4FH) | - |
| **{Label}** | <label> | - | $20_H$ to FFH | - |

## 6.3 FREQUENCY REFERENCED MOTION

Upon powerup the PRG defaults to frequency referenced motion commands which are referenced to the frequency of the crystal controlled system clock. These commands are accessed with parameters that are in standard frequency referenced formats. Acceleration is expressed in units of kHz/sec (Hz/millisecond) from 1 to 65,535 and speed is expressed in tenths of a kHz from 1 to 1920 (.1 to 192.0 kHz) Bit 1 of the W Register (ALTVEL) is set to 0.

With the alternate velocity selected (ALTVEL=1) the acceleration is expressed in units of 100 Hz/sec (.1 Hz/millisecond) from 1 to 65,535 (.1 Hz/millisecond to 6,535.5 Hz/millisecond; and the speed is expressed in units of 10 Hz from 1 to 4800 (10 Hz to 48.00 kHz). The PRF1x1c Indexer firmware described in SECTION 6.5 with internal distance reference (internal mode) selected (EXT/INT' - bit 6 of X status byte reset) operates in this mode. A minimum distance calculation based on the specified acceleration and velocity computes the minimum distance for an index.

## 6.4 DISTANCE REFERENCED MOTION

The distance referenced motion mode allows a PRG based motion control system to operate in synchronism with another PRG or other equipment having a digital position encoder. This allows a host computer to conveniently control multiple axes with "linear interpolation", in addition to several other general purpose capabilities.

This is accomplished by referencing a PRG's motion command pulses to a digital pulse train present at the EXTREF' input on header connector J9A near the High Speed Sensor Interface. For a schematic of this interface see APPENDIX 8.7. The pulses present at this external motion pulse reference input are normally the motion pulses from another PRG (REFXPL) present also on header J9A. These pulses are interfaced through a tri-state RS-422 differential line driver (U12) to pins 7 and 8 of JM5. Note that these outputs are received by RS-422 differential line receiver U13 which is factory strapped to EXTREF'.

For an example of how this synchronization is accomplished, consider three PRGs which have their JM5-7 and JM5-8 pins bussed together by a twisted pair cable. In the powerup default mode, both PRGs are deriving their motion command pulses from their respective crystal controlled oscillators, and therefore working independently.

Now assume that we want PRG **a** to be the "master" and PRGs **b** and **c** to be "slaves" and go half as far as PRG **a** at the same time that PRG **a** does an index. Status Register X, bit 4 (XPLENB) should be set on the master PRG with an **SX90**. This enables the tri-state driver U12 to drive the **"high speed motion reference bus"**. Next, Status Register X, bit 6 (EXT/INT') of the slaves should be set to select the external mode, with an **SXC0** command.

The parameters for the slave PRG are now in a format that references the distance traveled by an external reference. Acceleration is expressed in units of 100's of external counts from 0 to 65,534 (0 to 6,553,400) and speed is expressed in units of tenths of a percent of external reference frequency from 1 to 1000 (.1 to 100.0 %). ALTVEL (bit 1 of W status byte) has no effect in this mode.

The slave acceleration should now be set to zero and the slave commanded to "jog" at 50.0% of the EXTREF' input frequency. This is done with an A0 and J500+ command to the slave. An index of the master will now also cause the slave to move half as far. One can extend this concept of externally referenced motion to implement a wide variety of coordinated motion applications.

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

for a=0: minimum distance = .12 * f * v

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

where: a - acceleration in 100 Hz
v - velocity in .1 %
f - external frequency in MHz

## 6.5 PRF1x1c INDEXER FIRMWARE COMMAND DESCRIPTION

The following command descriptions apply to the default condition of internal mode (EXT/INT' reset) and ALTVEL = 0 (ALTVEL reset). For a description of other non-default options, see SECTION 6.2.

### @ - comment {program marker label}

Syntax : @ [<comment>] <cr>
{Syntax}: @ <label> [<comment>] <cr>

where : <label> single byte program marker with all eight bits significant; The @ sign character ($40_H$) is a unique programming label, in that this program will automatically execute on powerup or software reset of the PRG.
<comment> any number of characters exclusive of the <cr>; If the label is not present it is recommended that the first character be a space so that extra program labels are not created.

Example : @ This is a comment line
@X Comment after a program label 'X'
@@ This is the beginning of the "powerup" routine

**A = Acceleration - set or examine acceleration rate**

Syntax : A [<ramp>] <cr> | A <term>

where : <ramp> integer (1 to 65,535) in kHz/sec specifying the acceleration rate (default: 4000 kHz/sec); See SECTION 6.2 for other ranges.

<term> ! = display current system acceleration rate (zero if at rest or top speed)

? = display last entered acceleration rate

Example : A3500<cr> @ set acceleration rate to 3500 kHz/sec
A? @ display last entered acceleration rate
A! @ display current system acceleration rate

**B = {Branch} - conditional jump to program label**

Syntax : B <label> [<cond>] | B <label> <cr>

where : <label> see @ command label description

<cond> single character (A to 0) representing one of the conditions in the I/O Condition Table; IF the specified input condition is true THEN the command following the specified program label will be executed ELSE execution will continue with the following command.

Example : BQ<cr> @ unconditional branch to routine Q
BQM @ conditional 'M' branch to routine Q

**D = Delay - delay specified time interval**

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

where : <time> integer (0 to 65,535) in milliseconds specifying the amount of time delay before executing the next command; The resolution of the internal timer is 4 msec and due to the asynchronous nature of the delay command there is an uncertainty of 4 msec. Therefore since <time> is "rounded up" the minimum delay is a delay of 1 will delay 4 to 8 msec.

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

; = synchronizing character which causes the PRG to wait for the system motion to achieve a steady state non-zero speed

Note : A character entered during the execution of this command will end this command with a D0 error

Example : D16<cr> @ delay for 16 msec

**E = {Exit} - conditional return from subroutine or program**

Syntax : E <cond> | E <cr>

where : <cond> single character (A to O) representing one of the conditions in the I/O Condition Table; IF specified input condition is true THEN execution will return to the command following the last Function call or to interactive mode if a function is not active ELSE next command is executed

Note : this command is valid only in program mode
Example : E<cr> @ unconditional (return) exit
EG @ conditional 'G' return

**F = {Function} - conditional call to a subroutine**

Syntax : F <label> <cond> | F <label> <cr>

where : <label> see @ command label description
<cond> single character (A to O) representing one of the conditions in the I/O Condition Table; IF specified input condition is true THEN routine specified by the label is executed ELSE next command is executed

Example : FA<cr> @ unconditional call of routine 'A'
FAM @ call routine 'A' on 'condition M'

**CAUTION** : Nesting of functions is not supported, therefore only one function can be active at a time

**G = Go - move or examine absolute position**

Syntax : G [<apos>] [<sync>] <sign> <cr> | G <term>

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

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

<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)

<cr> move to the absolute position <sign> <apos> at rates specified by A and V

<term> ! = display the current absolute position of the system
? = equivalent to G!
* = stop system motion

```
Example : G!        @ display the present absolute position of the system
          G+<cr>    @ go to the absolute zero position of the system
          G200-<cr> @ move to absolute position -200
          G*        @ stop system motion
```

**H = Home - move until encoder reference / sensor**

```
Syntax  : H [<rate>] [<sync>] <term>
```

where :

| | | |
|---|---|---|
| <rate> | | integer (1 to 1,920) in 100 Hz units specifying the homing speed (default: 2.0 kHz) see SECTION 6.2 for other ranges |
| <sync> | , | = synchronizing character which causes the PRG to wait for the system motion to stop |
| | ; | = synchronizing character which causes the PRG to wait for the system motion to achieve a steady state non-zero speed |
| <term> | + | = move at the homing rate in the positive direction until encoder reference or sensor is detected (see bit 3 of Y status byte) |
| | - | = move at the homing rate in the negative direction until encoder reference or sensor is detected (see bit 3 of Y status byte) |
| | * | = stop system motion |
| | ! | = display current system speed |
| | ? | = equivalent to H! |

```
Example : H15+     @ move to the homing position in the positive direc-
                     tion with a velocity of 1.5 kHz
          H,-      @ wait for system to come to rest before homing in
                     the negative direction
          H*       @ stop system motion
```

**I = Index - move to the relative position**

```
Syntax  : I [<rpos>] #<mdes># <cr> | I <term>
```

where :

| | | |
|---|---|---|
| <rpos> | | integer (1 to 2,147,483,647) specifying the relative position to move (default: 500 counts) |
| <mdes> | + | = move <rpos> relative distance in the positive direction with the rates specified by the A and V commands |
| | - | = move <rpos> relative distance in the negative direction with the rates specified by the A and V commands |
| | , | = synchronizing character which causes the PRG to wait for the system motion to stop |
| | ; | = synchronizing character which causes the PRG to wait for the system motion to achieve a steady state non-zero speed |

<term> * = stop system motion
! = display the distance remaining in the current or last index
? = display the last entered index distance

Example : I250<cr> @ set the index distance to 250 counts
I+ @ index the system to the previously set distance in the positive direction
I,- @ wait for last motion to end; index to the previously set distance in the negative direction
I300,+ @ after the last motion is complete; index the system 300 counts in the positive direction
I,+,- @ wait for last motion to end; index in the positive direction; after this motion is stopped; index in the negative direction
I! @ display the number of remaining counts in the current move
I? @ display the previously specified relative distance <rpos>
I* @ stop the current index

**J = Jog - move at the jog rate**

Syntax : J [<rate>] #<mdes># <cr> | J <term>

where : <rate> integer (1 to 1,920) in 100 Hz units specifying the jog rate (default: 10.0 kHz); See SECTION 6.2 for other ranges.
<mdes> + = jog in the positive direction at the specified rate
- = jog in the negative direction at the specified rate
, = synchronizing character which causes the PRG to wait for the system motion to stop
; = synchronizing character which causes the PRG to wait for the system motion to achieve a steady state non-zero speed
* = stop system motion; System motion can be stopped by typing any character other than , or ; if the PRG is running and still in the middle of a J command.
<term> ! = display current system speed
? = display last entered jog rate

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 speed
J? @ display last entered jog rate
J36,+ @ wait until last motion is done; jog in positive direction at 3.6 kHz
J+ @ jog in the positive direction at previously specified 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
```

## L = {Loop} - repeat program segment specified number of times

Syntax : L <label> <count> <cr>

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

Note : the Loop command is valid only in program mode

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

**CAUTION** : Program loops cannot be nested. Interaction between multiple loops will produce unexpected results.

## N = Normalize - set absolute position or reset PRG

Syntax : N [<apos>] [,] <term>

where :
- <apos> integer (0 to 1,073,741,823) counts specifying the absolute position of the system (default: 0)
- , = synchronizing character which causes the PRG to wait for the system motion to stop
- <term> + = set absolute position counter to plus <apos>
- − = set absolute position counter to minus <apos>
- <cr> = start serial/parallel MCC selection (autobaud if serial)
- * = PRG software reset

Note : The system must be at rest before the above terminators can be executed.

```
Example : N2000,+   @ wait for system motion to stop; set absolute posi-
                      tion counter to +2000
          N*        @ software reset PRG
          N<cr><cr> @ select serial 19,200 baud MCC
```

## O = Out - set Machine I/O Interface state

Syntax : O <cond> | O !

where :
- <cond> single character (@ to O) representing the table conditions specified by the Machine I/O Interface Table
- ! = display the current <cond> of the system; I/O Condition Table letter

Example : OC @ output condition 'C'
O! @ display the current condition letter

**P = {Program} - enter or examine motion program**

Syntax : P <des> #<char># | P <label> #<char>#

where : <des> ! = initiate programming at beginning of program buffer; The program buffer is a free space in either EEPROM (PRG-901) or RAM (PRG-902), which starts at the beginning and ends with the characters <cr>Q.

? = display program buffer from the beginning, a command at a time; The ASCII ESC($1B_H$) character will terminate the output while any other character will cause the next command to be displayed

<cr> = initiate programming at the end of the program buffer

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

<char> characters other than those described below are entered directly into the program buffer

ESC = escape($1B_H$), exit program mode
**CAUTION:** Don't use this command if the end of program buffer is overwritten.

TAB | CTL-Y = horizontal tabulation($09_H$), move program buffer pointer forward one byte and display character or entire line (if advanced to the next line) NOTE: the PRG-004 Programming Terminal has an ENTER key which produces a CTL-Y

BS | DEL = backspace($08_H$), IF buffer pointer is on a command character THEN move buffer pointer to the previous command character; output: <cr><lf><entire line><cr> ELSE move buffer pointer back one byte; echo BS

LF = advance buffer pointer to the next line and display it

Q = end the program buffer at the current location and exit program mode The Q must be immediately preceeded by a <cr>.

Note : A non-printing character in the program buffer is displayed as a ~ ($7E_H$).

Example : P? @ display the first command of the program buffer and display each additional command by entering any 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 a 'Q' is entered
**CAUTION:** Never use an ESC to exit this mode.

**Q = {Quit} - terminate program space and exit program mode**

Syntax : Q

**S = Status - set or examine control status registers**

Syntax : S <id> <data> <cr> | S!

where : <id> single character identifier (W | X | Y | Z) referencing the desired status register (see below)
<data> hexadecimal representation of the selected byte
! = display status registers in the order WXYZ

**W Register**

bit 0 - DIRINV (direction invert) transposes the meaning of + and -
bit 1 - ALTVEL (alternate velocity) selects 48.00 kHz as the maximum velocity and increases the resolution of the acceleration and velocity parameters. See SECTION 6.2.
bit 2 - reserved
bit 3 - reserved
bit 4 - reserved
bit 5 - reserved
bit 6 - reserved
bit 7 - reserved

Note : W bits cannot be changed when the system is in motion

**X Register**

bit 0 - PLDIS' (Position Loop Disable) DPL output; This output disables the position loop in the MLC, allowing the MLC based servomotor system to perform non-positioning operations.
bit 1 - DRVDIS' (Drive Disable) DPL output; This output asserts the drive inhibit (DRVINH) output of the MLC as well as disabling the position and velocity compensators in the MLC. The MLC is designed with buffered line driver outputs for driving a solid state relay to power down the servodrive and/or disable a LOOP CONTACTOR (dynamic breaking relay).
bit 2 - MLCRST' (MLC reset') DPL output; Resets the MLC.
bit 3 - DRVON (drive on) DPL input for determining if there has been a position error counter overflow fault at the MLC. This bit is used as a lower nibble set mask on output (See SECTION 6.8 for details).

bit 4 - XPLENB (external pulse enable) enables this PRG's motion reference pulses as the source on the Motion Reference Bus
bit 5 - ARENB (alternate reference enable) selects mode whereby each odd reference pulse is passed through to the FWDCMD on the DPL and each even source reference pulse is used as the internal distance reference; This output is probably useful only with external mode for establishing a nominal motor speed with respect to other moving machinery.

bit 6 - EXT/INT' (external/internal') selects external distance referenced motion mode rather than use the internal clock as a time base; This bit can only be changed if the system is not in motion.

bit 7 - FWD/RVS' (forward/reverse') indicates direction of current or last motion; This bit is used as a upper nibble set mask on output (See SECTION 6.8 for details).

Note : To set X bits 0 thru 2 the mask bit 3 must be set and likewise to set X bits 4 thru 6 the mask bit 7 must be set. Therefore the low bits can be set independent of the upper bits.

**CAUTION** : Altering any of the W or X bits will cause an immediate effect.

**Y Register**

bit 0 - SHPJOG (sharp jog) selects a sharp (immediate) stop upon jog deceleration rather than the controlled deceleration rate specified by the **A** command

bit 1 - SENDCL (sensor decelerate) selects the sensor input to initiate motion deceleration after full speed is attained; Ordinarily deceleration is initiated when the remaining distance is equal to the acceleration distance.

bit 2 - JOGSTP (jog stop) specifies that speed should remain at the level set by the **J** command during deceleration rather than continuing to zero; JOGSTP is usually used in conjunction with a sensor stop.

bit 3 - SENSTP (sensor stop) selects the sensor input signal (SENSOR') rather than the encoder reference to stop the motion during a home command; This bit is also used to select either the encoder reference or the sensor input signal for a motion using the JOGSTP option (bit 2). To wire the SENSOR' input, see APPENDIX 8.7.

bit 4 - SENACL (sensor accelerate) causes a motion to begin upon receiving either the sensor input signal (SENSOR') or the encoder reference depending on the state of bit 5. To wire the SENSOR' input, see APPENDIX 8.7.

bit 5 - SENSRT (sensor start) selects the sensor input signal (SENSOR') rather than the encoder reference to start a motion; To wire the SENSOR' input, see APPENDIX 8.7.

bit 6 - ACLTYP (acceleration/deceleration type) various acceleration
bit 7 profiles are selected by setting these two bits (bits 6 & 7 ) as follows:

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

Note : Y bits are only examined when a motion is started; therefore altering them during a motion will only effect the next commanded motion.

**CAUTION** : The minimum distance calculation does not currently account for the additional acceleration distance of the parabolic ramp nor is it used in external mode.

**Z Register**

bit 0 - NONECH (non-echo) prevents the echo of all MCC characters as well as preventing the echo of characters from the program buffer while the PRG is running a program

**CAUTION:** The auto distance calculation is not performed when in the non-echo mode.

bit 1 - HEXMCC (hexadecimal MCC) specifies that all input data be interpreted and output data be displayed in ASCII hexadecimal; Default input/output is in ASCII decimal.

bit 2 - BINMCC (binary MCC) enables binary input/output; Binary input to the PRG for any given data transfer is selected by setting the high order bit of the ASCII command character. That character must then be followed by the appropriate number of two's complement binary interpreted bytes (most significant byte first). These bytes are not echoed regardless of the setting of NONECH (bit 0).

| Command | # bits | # bytes | terminator |
|---|---|---|---|
| A | 16 | 2 | None |
| D | 16 | 2 | None |
| G | 30 | 4 | Yes |
| H | 11 | 2 | Yes |
| I | 31 | 4 | Yes |
| J | 11 | 2 | Yes |
| N | 30 | 4 | None |
| S | 32 | 4 | None |
| V | 11 | 2 | None |

The high order bit is ignored for all other commands not in the above table. The <cr> terminator is not required with the commands indicated above and the "greater than" prompt is returned after the specified number of bytes have been sent. Note: The (G)o and (N)ormalize commands expect their arguments in two's complement form and the <cr> initiates action. The <sign> character is not recognized in binary mode.

Binary output from the PRG (in two's complement form) is requested by setting the high order bit of the ASCII '?' or '!' terminator; The number of binary output bytes can be determined from the above table.

Note The high order bit of the ASCII characters is ignored if BINMCC is not set.

bits 3-7 reserved

**U = Until - wait until specified condition is true**

Syntax : U <cond>

where : <cond> single character (A to O) representing one of the conditions in the I/O Condition Table; WHEN specified input condition is true execution will continue with the next command

Note : A character entered during the execution of this command will end this command with a D0 error, and STOP' asserted at the MIO will end this command with a D1 error.

Example : UC @ wait until the 'C' condition is true

**V = Velocity - set or examine maximum velocity rate**

Syntax : V <rate> <cr> | V <term>

where : <rate> = integer (1 to 1,920) in 100 Hz units specifying the velocity rate (default: 40.0 kHz) see SECTION 6.2 for other ranges

<term> ! = display current system speed

? = display last entered velocity rate

Example : V304<cr> @ set velocity to 30.4 kHz for next motion
V? @ display last entered acceleration rate
V! @ display current speed

## 6.6 I/O CONDITION TABLE

| Condition Character | Input condition IN3 JM2- 41 | IN2 43 | IN1 45 | IN0 47 | Output condition OUT3 JM2- 17 | OUT2 19 | OUT1 21 | OUT0 23 |
|---|---|---|---|---|---|---|---|---|
| @($40_H$) | - | - | - | - | H | H | H | H |
| A($41_H$) | - | - | - | L | H | H | H | L |
| B($42_H$) | - | - | L | - | H | H | L | H |
| C($43_H$) | - | - | L | L | H | H | L | L |
| D($44_H$) | - | L | - | - | H | L | H | H |
| E($45_H$) | - | L | - | L | H | L | H | L |
| F($46_H$) | - | L | L | - | H | L | L | H |
| G($47_H$) | - | L | L | L | H | L | L | L |
| H($48_H$) | L | - | - | - | L | H | H | H |
| I($49_H$) | L | - | - | L | L | H | H | L |
| J($4A_H$) | L | - | L | - | L | H | L | H |
| K($4B_H$) | L | - | L | L | L | H | L | L |
| L($4C_H$) | L | L | - | - | L | L | H | H |
| M($4D_H$) | L | L | - | L | L | L | H | L |
| N($4E_H$) | L | L | L | - | L | L | L | H |
| O($4F_H$) | L | L | L | L | L | L | L | L |

(H)igh TTL level (L)ow TTL level (-) don't care

## 6.7 EXCEPTION HANDLING AND ERROR CODES

The PRG is designed to trap user errors and return error messages in standard formats. It is the responsibility of the designer of the host computer software to handle these messages. Once an error is detected; the current mode is terminated {including programming mode}, an error message is sent to the MCC, and the PRG will accept a new command following the output of the prompt.

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

**Syntax error codes:**

A1 - invalid command; After the PRG sent a prompt character, a command character other than one described in SECTION 6.5 was entered.

A2 - invalid terminator or designator; A character other than a terminator or designator expected by the command was entered.

A3 - input argument > 32 bits; A number greater than the maximum size of 4,294,967,296 was entered.

A4 - out of index range (1 to 2,147,483,647); A number out of the allowable range of <rpos> was entered.

A5 - out of absolute range (1 to 1,073,741,823); A number out of the allowable range of <apos> was entered.

A6 - out of acceleration range (default range is 1 to 65,535); A number out of the allowable range of <ramp> was entered.

A7 - out of velocity range (default range is 1 to 1920); A number out of the allowable range of <rate> was entered.

**Motion error codes:**

B1 - currently unused

B2 - command not legal while system in motion; A motion designator or programming command was entered when the system was in motion.

B3 - attempt to move system with +LIMIT', or -LIMIT' low; Forward motion was attempted with +LIMIT' low or reverse motion was attempted with -LIMIT' low. These signals are found at the Machine I/O Interface.

B4 - move distance too short; Calculated trapezoidal profile minimum distance is greater than actual move distance. The PRG will output the calculated minimum distance preceeding the #B4.

B5 - MCC character received during wait for motion complete or wait for steady state velocity; Any character entered during a comma designator wait for the current motion to complete terminates the wait with this error.

B6 - attempt to move system with drive off; A motion designator was entered with DRVOFF asserted.

B7 - invalid use of the Jog command; Speed changes can only be made while a jog motion is in progress.

**{Programming error codes}:**

C1 - program buffer overflow; There was an attempt to enter more characters than the program buffer can contain.

C2 - undefined program label; There was a branch, function call, or loop to a nonexistent program label.

C3 - program memory storage fault; The last entered character programming character not saved in program buffer due to hardware failure of the RAM or EEPROM memory.

C4 - MCC character received during command execution

C5 - currently unused

C6 - command not legal during program execution; There was an attempt to execute the Program command during program execution.

**Miscellaneous error codes:**

D0 - MCC character received during Delay or Until command

D1 - STOP' signal at the Machine I/O Interface asserted

## 6.8 STATUS REGISTERS

There are four status registers, designated W,X,Y, and Z. The purpose of the status registers is to allow the user to conveniently change the configuration of the PRG to meet individual motion control application needs. Each status register contains eight "bit switches", which when "set" or "cleared" will change the operation of the PRG is some way, with the exception of the X Register in which two of the bits are informational bits and cannot be set or cleared.

The status registers are examined or configured with the S command. To examine the status registers, powerup the PRG in interactive command mode and type the following sequence. See the GETTING STARTED (SECTION 5) for more detail. In the following sequences, **bold** print indicates the sequence that you type and regular print is the information sent by the PRG.

PRF131c
=>**S!**000F0000 - To examine the status registers, you typed S!, and the PRG in turn sent eight characters, which are hexadecimal digits indicating the bit patterns in each status register. The first two digits indicate the bit pattern in the W Register, the next two digits indicate the bit pattern in the X Register, the next two digits indicate the bit pattern in the Y Register and the last two digits indicate the bit pattern in the Z Register. The first digit of each pair indicates the value of the upper "nibble" (bits 7-4) and the second digit indicates the value of the lower nibble (bits 3-0).

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

**Status Register Bit Assignments**

| Upper Nibble Value | Bit Number 7 6 5 4 | Lower Nibble Value | Bit Number 3 2 1 0 |
|---|---|---|---|
| $0_H$ | 0 0 0 0 | 0H | 0 0 0 0 |
| $1_H$ | 0 0 0 1 | 1H | 0 0 0 1 |
| $2_H$ | 0 0 1 0 | 2H | 0 0 1 0 |
| $3_H$ | 0 0 1 1 | 3H | 0 0 1 1 |
| $4_H$ | 0 1 0 0 | 4H | 0 1 0 0 |
| $5_H$ | 0 1 0 1 | 5H | 0 1 0 1 |
| $6_H$ | 0 1 1 0 | 6H | 0 1 1 0 |
| $7_H$ | 0 1 1 1 | 7H | 0 1 1 1 |
| $8_H$ | 1 0 0 0 | 8H | 1 0 0 0 |
| $9_H$ | 1 0 0 1 | 9H | 1 0 0 1 |
| $A_H$ | 1 0 1 0 | AH | 1 0 1 0 |
| $B_H$ | 1 0 1 1 | BH | 1 0 1 1 |
| $C_H$ | 1 1 0 0 | CH | 1 1 0 0 |
| $D_H$ | 1 1 0 1 | DH | 1 1 0 1 |
| $E_H$ | 1 1 1 0 | EH | 1 1 1 0 |
| $F_H$ | 1 1 1 1 | FH | 1 1 1 1 |

1 indicates that the bit switch is set or high
0 indicates that the bit switch is cleared or low

To determine the hexadecimal values for the upper and lower nibbles, find the desired bit pattern of each nibble and look up the hexadecimal value for that bit pattern in the table. e.g. the command **SW03** will set the bits in the W Register to 0000 0011.

The general orientation of each of the status registers is listed below. For a detailed description of the meaning of each bit switch in each status register, refer to the S command in SECTION 6.5.

**W Register** - modify the meaning of motion commands

Example:

=>**SW2** - sets the W Register bit pattern to 0000 0010, which reduces the maximum velocity of the PRG from 192.0 kHz to 48.00 kHz as well as increasing the resolution of the velocity and acceleration specifications

=>**S!**020F0000 - displays the current status register values

The X Register is unique in that it has two bits (bits 3 and 7) which are informational bits that can be examined, but not directly manipulated, by the status command. Bit 3 indicates whether or not the MLC currently is asserting the "drive on" output (to enable the servodrive power) and bit 7 indicates the direction of the most recent (or current, if the system is moving) motion.

For the MLC to assert the "drive on" output and enable servodrive power, two conditions must be met. First, the PRG must have the X Register, bit 1 output (DRVDIS') high. Second, the MLC fault detection circuitry must be reset. Therefore, if the DRVDIS' is high and the X Register, bit 3 (DRVON) is low, then the MLC fault detection circuitry must be reset by momentarily clearing the X Register, bit 2 (MLCRST') and then reasserting it to remove the RESET condition. This may be accomplished with an **N*** command, or by executing an **SXB** status command, followed by a few millisecond delay (**D3**) and executing an **SXF** command.

Because bits 3 and 7 can't be manipulated directly, and since the lower 4 bits of the X Register have a different orientation than the upper 4 bits, bits 3 and 7 are used on output as "lower and upper nibble set masks". i.e. In order to change the bit pattern of bits 0-2, bit 3 must be high on output. Conversely, if bit 3 is not high on output, the bit pattern of bits 0-3 will remain unchanged. Also, in order to change the bit pattern of bits 4-6, bit 7 must be high on output. Conversely, if bit 7 is not high on output, the bit pattern of bits 4-6 will remain unchanged.

Remember that asserting these bits for an output command will not set them for input, but only allow the user to "mask" the manipulation of the lower three bits from the manipulation of the upper three bits. The following sequences will illustrate this principle.

=>**S!**000F0000 - displays the current status register values; Displayed are the default values.

=>**SX08** - clears bits 0-2 and leaves bits 4-6 unchanged; Bits 4-6 are unchanged because bit 7 was low. Consult the chart above for status register bit assignments.

=>**SX8** - does the same thing as SX08 because the PRG does not require leading zeros to be entered

=>**SXF** - sets bits 0-2 and leaves bits 4-6 unchanged

=>**SXE0** - sets bits 5 & 6, clears bit 4 and leaves bits 0-2 unchanged since bit 3 was low on output

=>S!006F0000 - displays current status register values

=>SXD - sets bits 0 and 2, clears bit 1 and leaves bits 4-6 unchanged

=>S!00650000 - displays current status register values

**X Register** (lower 4 bits) - communications with the Motor Loop Controller

**X Register** (upper 4 bits) - control communications with the "Motion Reference Bus"; This part of the X Register is used to coordinate the motions of multiple PRGs. For more information about coordinating the movement of PRGs, refer to SECTION 6.4.

**Y Register** - alter the type of motion performed by motion commands which follow; This register is used to control the shape of motion profiles and accurately synchronize motion with external sensors interfaced with the High Speed Sensor (HSS) Interface.

**Z Register** - control Motion Control Communications with Host; This register is used to turn on or off echo of PRG activity to the MCC, as well as enable binary or ASCII hexadecimal communications.

# MAINTENANCE

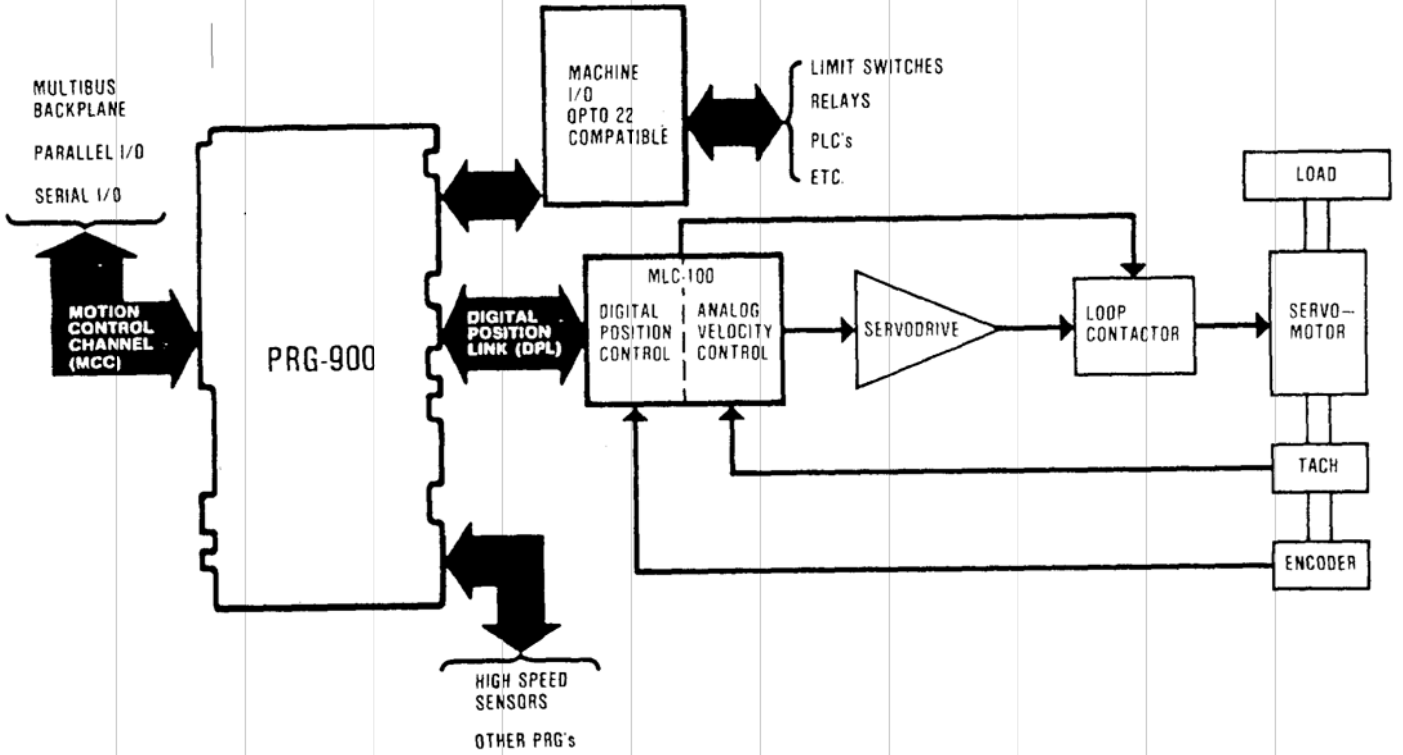
## 7.1 PREVENTIVE

No preventive maintenance procedures are required for the PRG-900 family.

## 7.2 DEMAND

ORMEC equipment is designed modularly for simple onsite demand maintenance consisting of convenient module replacement. Most of the integrated circuits used for Input/Output are socketed for easy user replacement. If a problem occurs which is beyond the socketed I/O circuitry, the user should return the defective module for factory repair. The PRG-900 family is designed with connector interfaces to make board replacement in the field simple and fast.

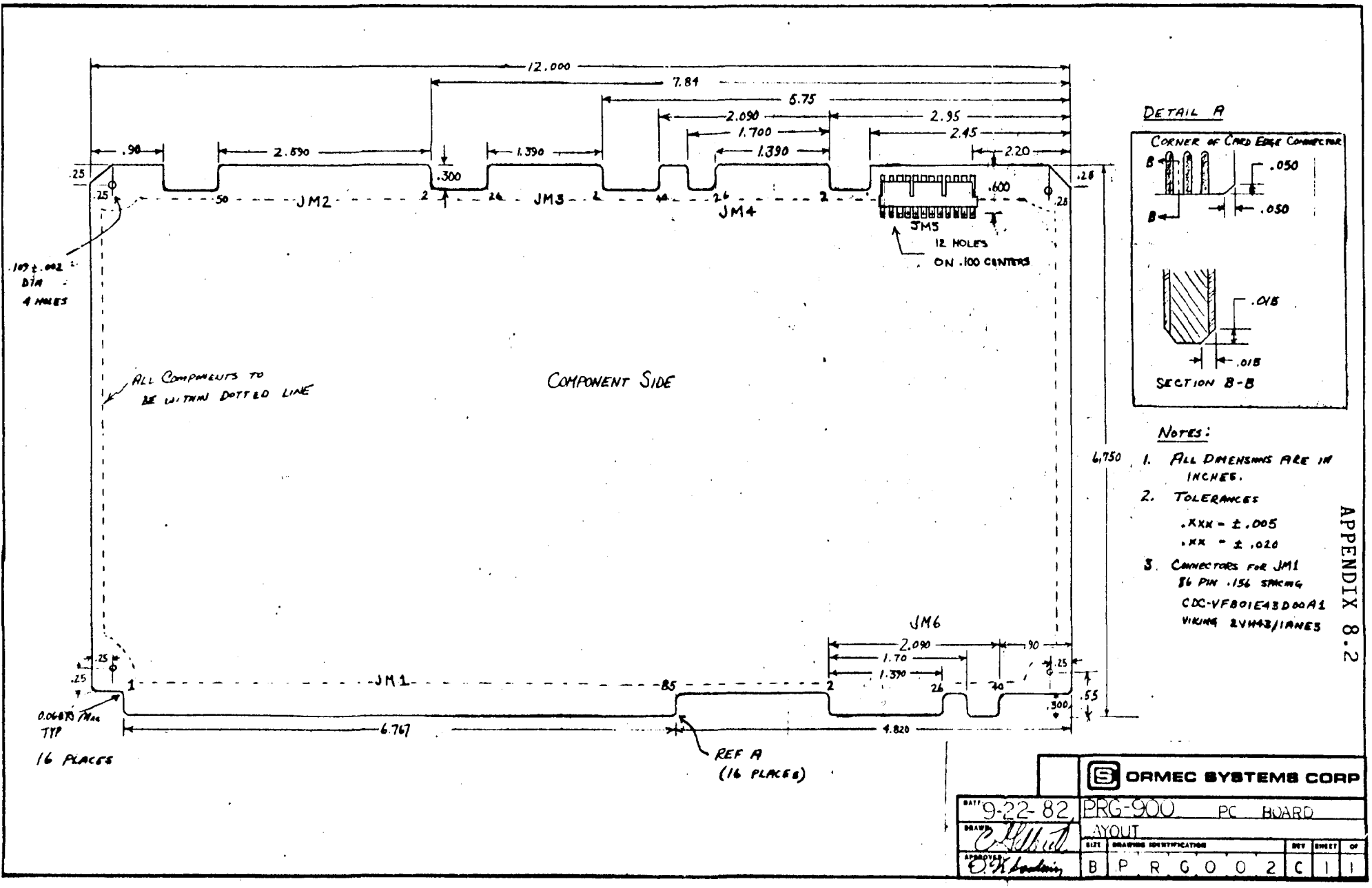
| SIZE | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | P R G 0 0 4 | R | 1 | 1 |




APPENDIX 8.1

ORMEC SYSTEMS CORP

| | |
|---|---|
| DATE 10/26/82 | PRG System Interface Diagram |
| DRAWN Mary Bryja | |
| APPROVED M. Minio | SIZE B · DRAWING IDENTIFICATION P R G 0 0 4 · REV R · SHEET 1 · OF 1 |



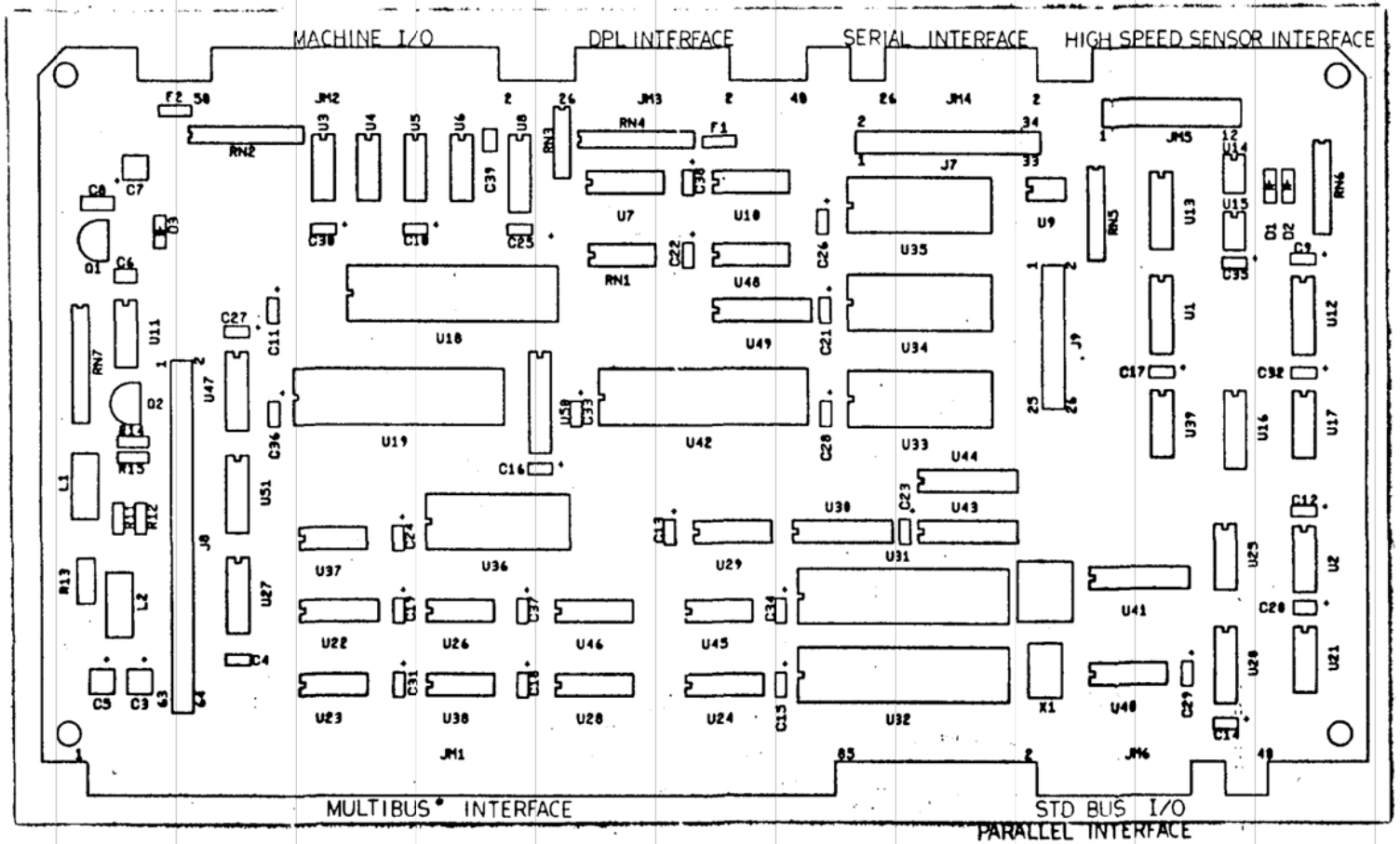

NOTES:

1. ALL DIMENSIONS ARE IN INCHES.
2. TOLERANCES
   .XXX = ± .005
   .XX = ± .020
3. CONNECTORS FOR JM1
   86 PIN .156 SPACING
   CDC-VFB01E43D00A1
   VIKING 2VH43/1ANE5

APPENDIX 8.2

| | | | | | |
|---|---|---|---|---|---|
| ORMEC SYSTEMS CORP | | | | | |
| DATE 9-22-82 | PRG-900 PC BOARD LAYOUT | | | | |
| DRAWN | | | | | |
| APPROVED | SIZE B | DRAWING IDENTIFICATION PRG002 | REV C | SHEET 1 | OF 1 |

| SZ | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | P R G 0 0 3 | B | 1 | 1 |




APPENDIX 8.3

ORMEC SYSTEMS CORP

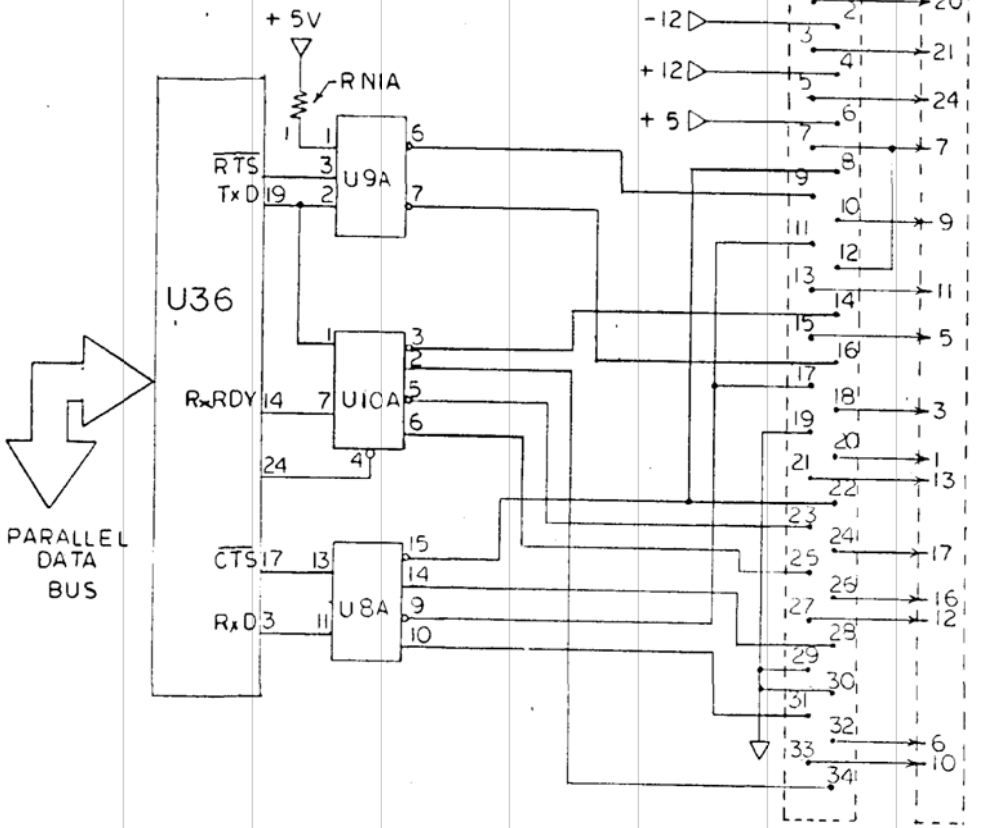
| | |
|---|---|
| DATE | 8/4/82 |
| DRAWN | ME Miner |
| APPROVED | DA Sanders |

POSITION REFERENCE GENERATOR
COMPONENT LAYOUT

| SZ | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | P R G 0 0 3 | B | 1 | 1 |

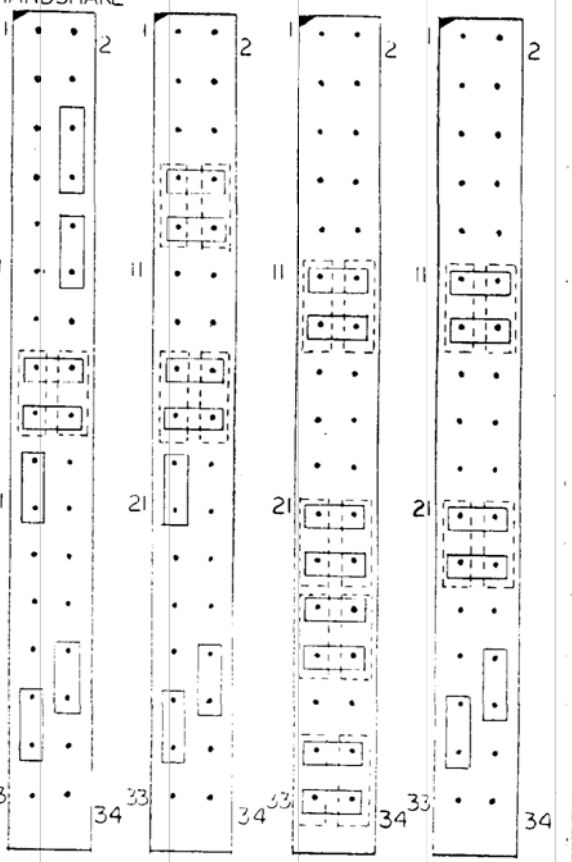
CONTACT SUPPLY

| SIZE | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | PRG005 | C | 1 | 4 |




| RS-232 PIN | NAME | RS-449 PIN | NAME |
|---|---|---|---|
| — | — | 19 | SG(C) |
| 23 | 12V | 29 | — |
| 11 | 12V | 11 | — |
| 25 | 5V | 31 | — |
| 4 | RTS | 4 | SD(A) |
| 5 | CTS | 5 | — |
| 6 | DSR | 6 | RD(A) |
| 3 | RXD | 3 | — |
| 2 | TXD | 2 | SI(A) |
| 1 | FG | 1 | SHIELD |
| 7 | SG | 7 | RS(A) |
| 9 | — | 9 | CS(A) |
| 21 | — | 27 | CS(B) |
| 19 | — | 25 | RS(B) |
| 16 | — | 22 | SD(B) |
| 18 | — | 24 | RD(B) |

PIN ASSIGNMENTS ON D-SUB CONNECTOR




STRAPPING CONFIGURATIONS

NOTE: ▭ DCE ⬚ DTE

| REF. DES. | ORMEC PART NO. | MANUFACTURER PART NO. | DESCRIPTION |
|---|---|---|---|
| U8 | UDT003A | SN75175N | T.I. QUAD DIFFERENTIAL RECEIVER |
| U9 | UDT001A | 75150P | T.I. DUAL LINE DRIVER (RS232) |
| U10 | UDT002A | SN75174N | T.I. QUAD DIFFERENTIAL DRIVER |
| U36 | UDN004A | P8251A | INTEL-NMOS USART |

* RS 232 DCE IS THE FACTORY INSTALLED CONFIGURATION

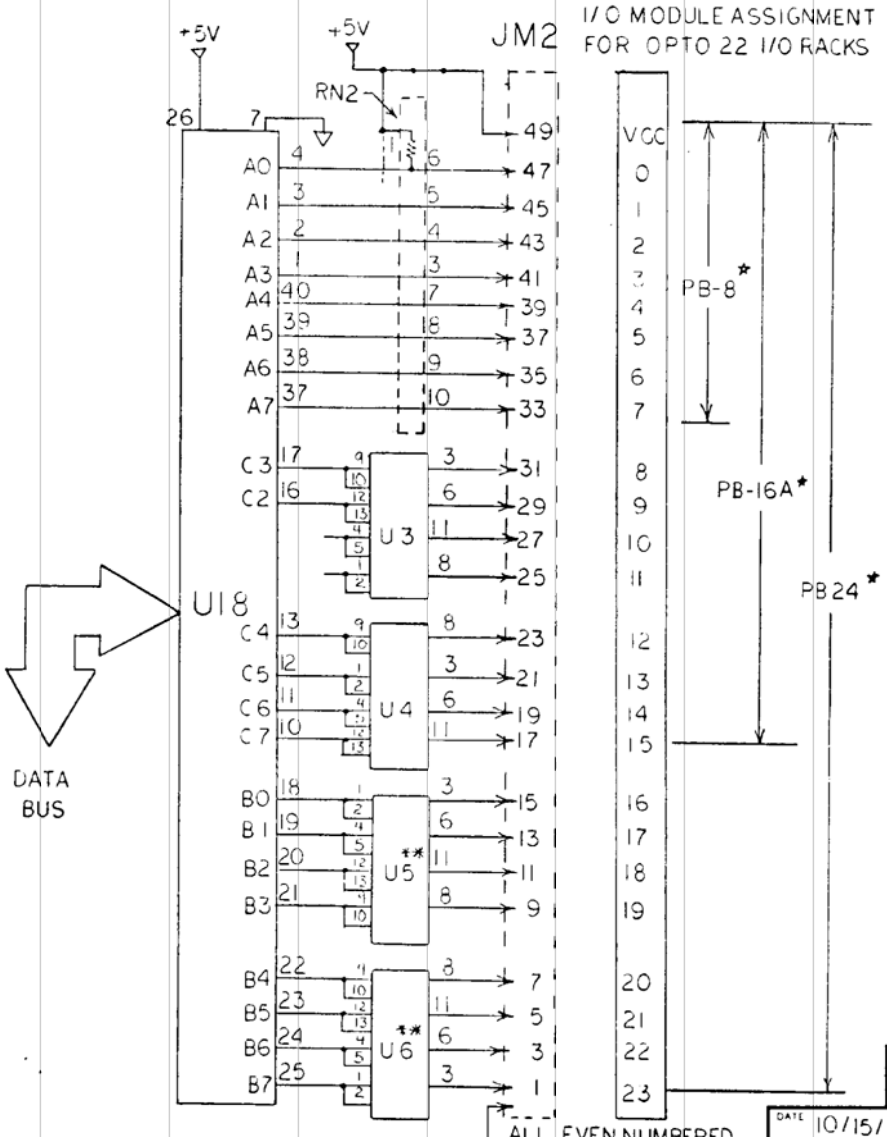
ORMEC SYSTEMS CORP

| | |
|---|---|
| DATE | 10-4-82 |
| DRAWN | |
| APPROVED | |

PRG SERIAL MCC INTERFACE

| SIZE | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | PRG005 | C | 1 | 4 |

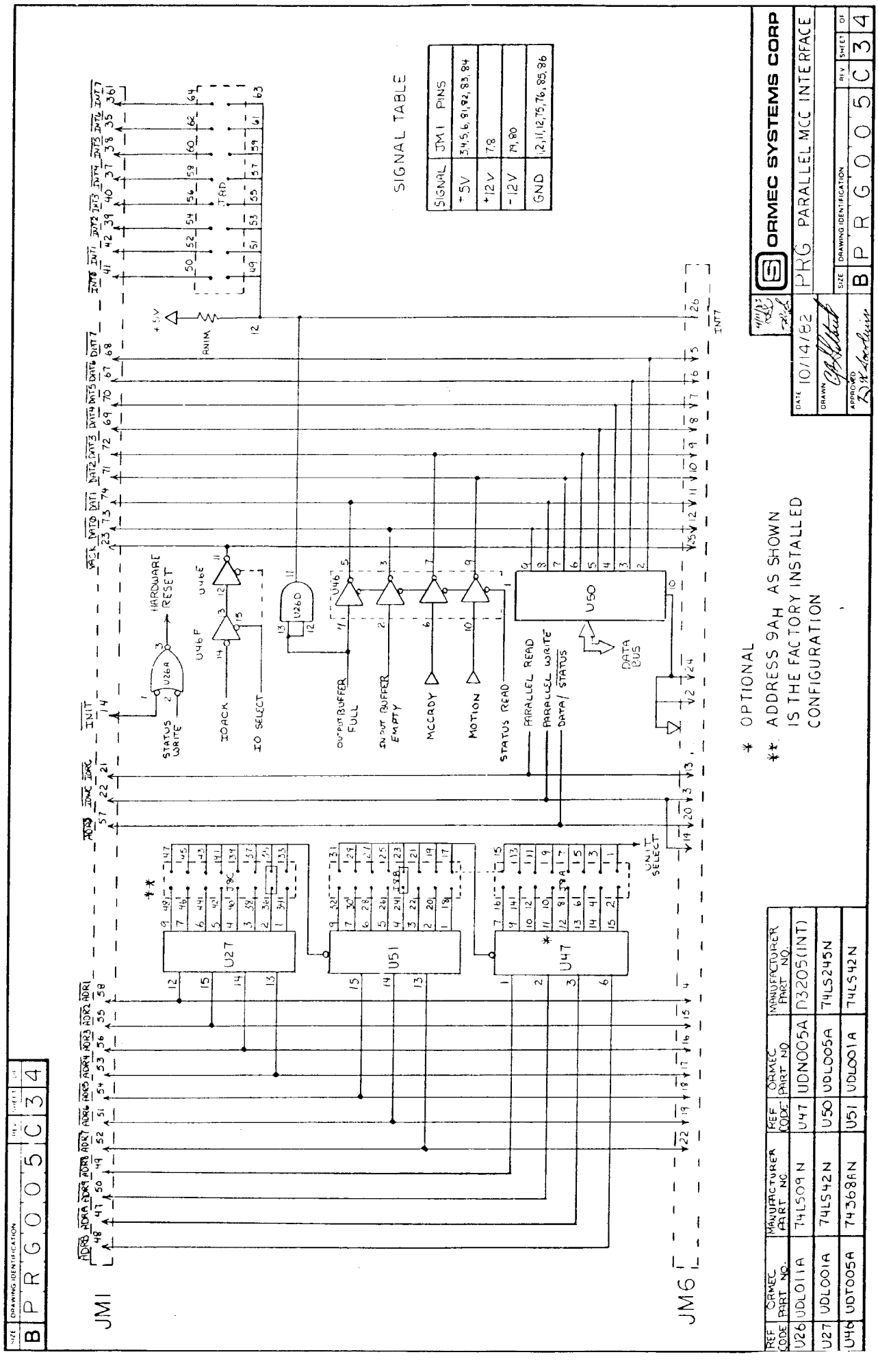
| SIZE | DRAWING IDENTIFICATION | REV | SHEET | OF |
|---|---|---|---|---|
| B | PRG005 | B | 2 | 4 |




| REF CODE | ORMEC PART NO. | MANUFACTURER PART NO | MNFR |
|---|---|---|---|
| RN2 | RENO11A | 4310R-101-222 | BOURNS |
| U3 | UDL007A | 74LS00N | TI |
| U4 | UDL007A | 74LS00N | TI |
| U5 | RENO16A | iSBC902 | INTEL |
| U6 | RENO16A | iSBC902 | INTEL |
| U18 | UDN003A | P8255A | INTEL |

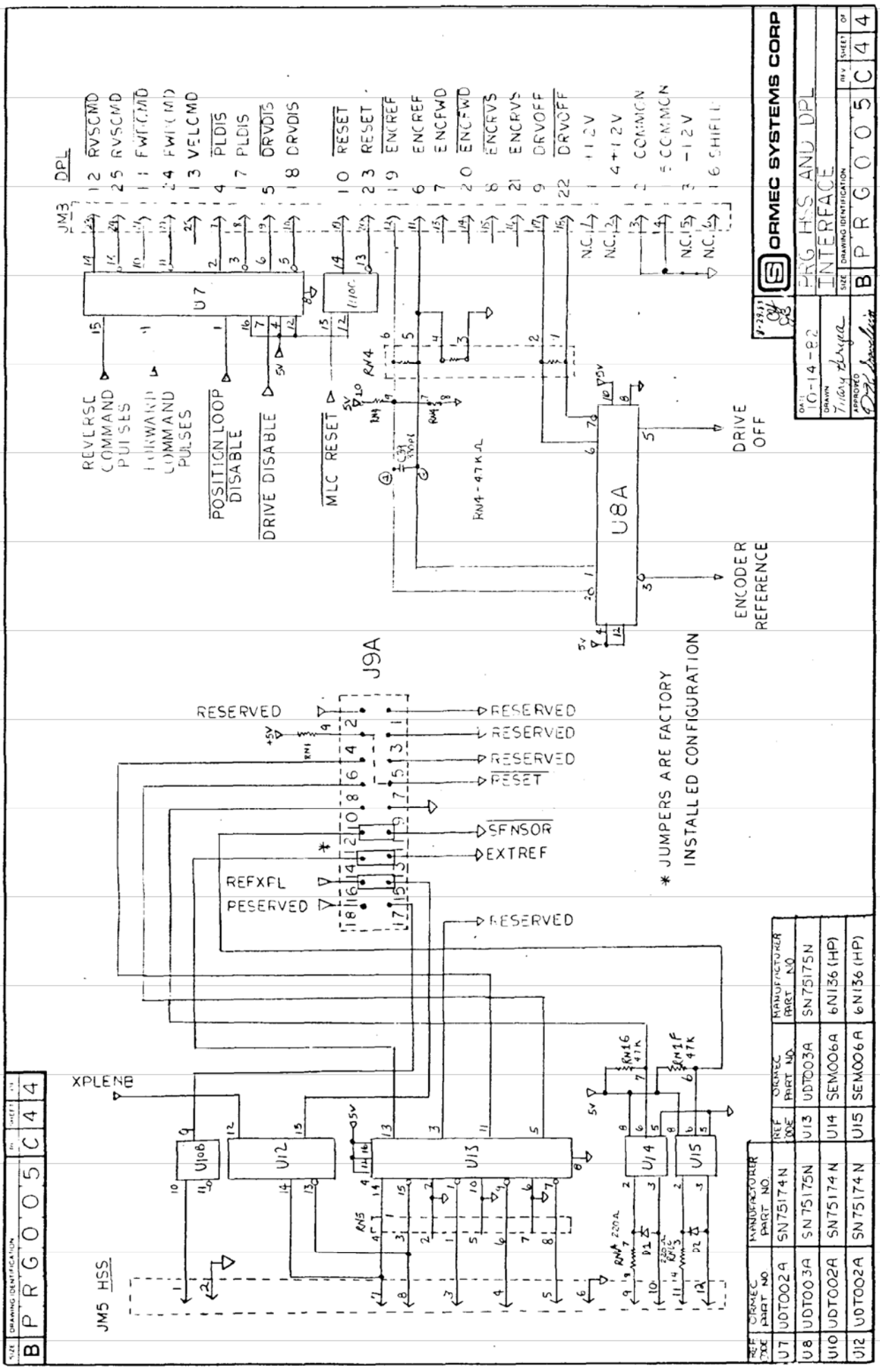
ORMEC SYSTEMS CORP

| DATE | 10/15/82 | PRG MACHINE I/O INTERFACE | | | |
|---|---|---|---|---|---|
| DRAWN | C.P. Hilliard | | | | |
| APPROVED | D.P. Spadaro | SIZE B | DRAWING IDENTIFICATION PRG005 | REV B | SHEET 2 | OF 4 |




| REF CODE | ORMEC PART NO. | MANUFACTURER PART NO. |
|---|---|---|
| U26 | UDL011A | 74LS09 N |
| U27 | UDL001A | 74LS42N |
| U46 | UDT005A | 74368AN |

| REF CODE | ORMEC PART NO | MANUFACTURER PART NO. |
|---|---|---|
| U47 | UDN005A | D3205(INT) |
| U50 | UDL005A | 74LS245N |
| U51 | UDL001A | 74LS42N |


ORMEC SYSTEMS CORP
PRG HSS AND DPL INTERFACE
SIZE B
DRAWING IDENTIFICATION PRG005
REV C
SHEET 4 OF 4
DATE 10-14-82
DRAWN Tiffany Shaya
APPROVED
DPL
JM3
RVSCMD
RVSCMD
FWDCMD
FWDCMD
VELCMD
PLDIS
PLDIS
DRVDIS
DRVDIS
RESET
RESET
ENCREF
ENCREF
ENCFWD
ENCFWD
ENCRVS
ENCRVS
DRVOFF
DRVOFF
+12V
+12V
COMMON
COMMON
-12V
SHIELD
N.C.
U7
U8A
REVERSE COMMAND PULSES
FORWARD COMMAND PULSES
POSITION LOOP DISABLE
DRIVE DISABLE
MLC RESET
DRIVE OFF
ENCODER REFERENCE
RN4 - 47K Ω
J9A
RESERVED
RESERVED
RESERVED
RESERVED
SENSOR
EXTREF
REFXFL
RESERVED
RESERVED
* JUMPERS ARE FACTORY INSTALLED CONFIGURATION
JM5 HSS
XPLENB
U10B
U12
U13
U14
U15
RN16 47K
RN1F 47K
RN5
RN4 220Ω
D1
D2
REF DES | ORMEC PART NO | MANUFACTURER PART NO
U7 | UDT002A | SN75174N
U8 | UDT003A | SN75175N
U10 | UDT002A | SN75174N
U12 | UDT002A | SN75174N
REF DES | ORMEC PART NO | MANUFACTURER PART NO
U13 | UDT003A | SN75175N
U14 | SEM006A | 6N136 (HP)
U15 | SEM006A | 6N136 (HP)